TPC Benchmark™ C Full Disclosure Report Using Sun Microsystems Ultra Enterprise 2 Model 2200, and IBM DB2 for Solaris Version 2.1.1 RDBMS

Sun Microsystems Computer Company
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

Revision 1, April 1996
Submitted for review
Compliant with Revision 3.0 of the TPC-C specification

lease
cycle

Abstract

Overview

This report documents the methodology and results of the TPC Benchmark C™ test conducted on the Ultra Enterprise 2 Model 2200 system, running IBM DB2 for Solaris V2.1.1 RDBMS and Independence Technologies Inc. Tuxedo 4.2.1.

TPC Benchmark C Metrics

The standard TPC Benchmark™C metrics, tpmC (transactions per minute), price per tpmC (five year capital cost per measured tpmC), and the availability date are reported as required by the benchmark specification.

Executive Summary Statements

Pages v-vii contain the executive summary of the benchmark result for the Sun Microsystems Ultra Enterprise 2 Model 2200.

First Printing

Sun Microsystems Computer Company believes that the information in this document is accurate as of its publication date. The information in this document is subject to change without notice. Sun Microsystems Computer Company assumes no responsibility for any errors that may appear in this document.

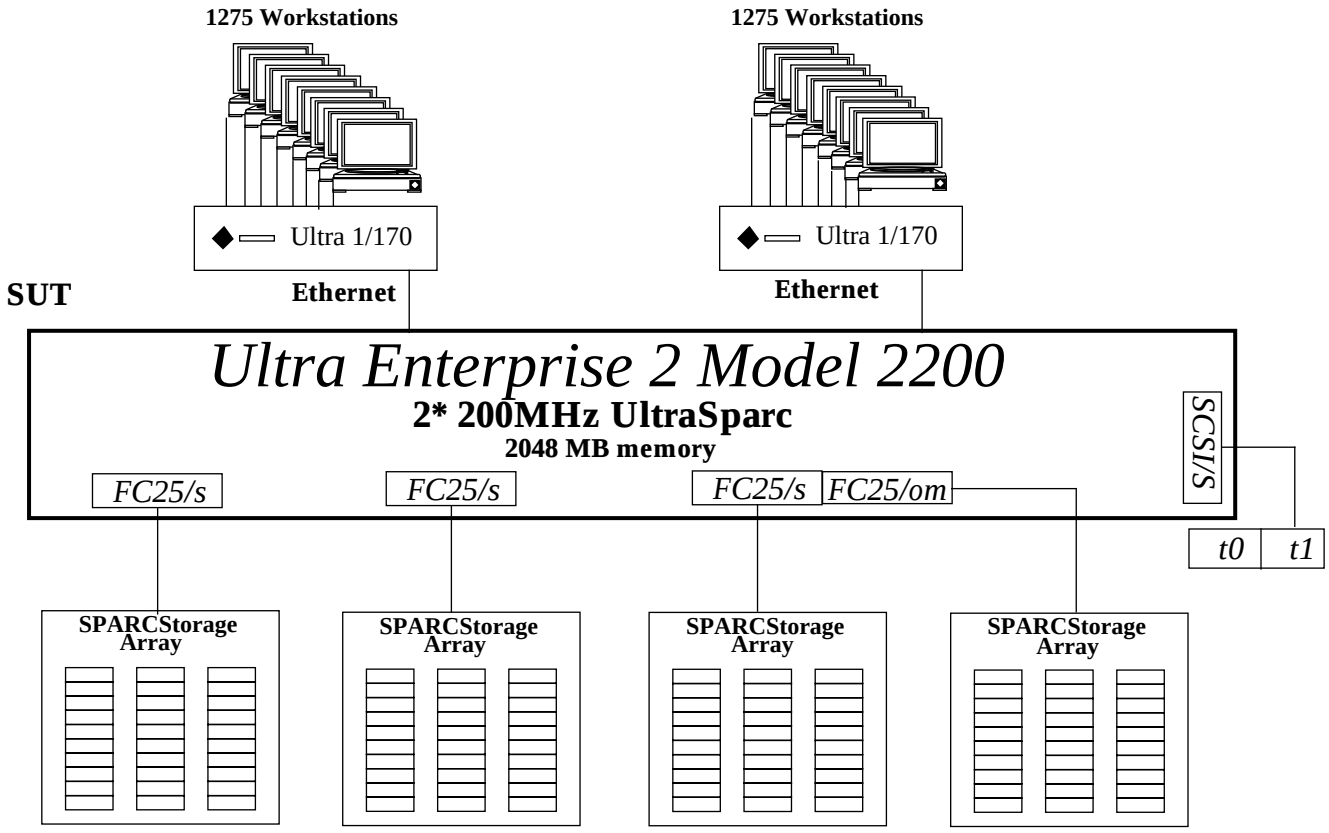
The pricing information in this document is believed to accurately reflect prices in effect on April 16, 1996. However, Sun Microsystems Computer Company provides no warranty on the pricing information in this document.

The performance information in this document is for guidance only. System performance is highly dependent on many factors including system hardware, system and user software, and user application characteristics. Customer applications must be carefully evaluated before estimating performance. Sun Microsystems Computer Company does not warrant or represent that a user can or will achieve a similar performance expressed in tpmC or normalized price/performance ($/tpmC). No warranty on system performance or price/performance is expressed or implied in this document.

Sun Microsystems	Ultra Enterprise 2	TPC-C Rev 3.2
IBM Corporation	C/S w/2 Front-Ends	Report Date:11/18/96

Total System Cost	TPC-C Throughput	Price/Performance	Availability Date
$360,488.05	3107.17 tpmC	$116 per tpmC	Aug 31, 1996

Processors	Database Manager	Operating System	Other Software	Number of Users
2 UltraSPARC I 200 MHz	DB2 for Solaris V2.1.1 RDBMS	Solaris 2.5.1	Tuxedo Ver. 4.2.1	2550



System Components	Server System	Front End Systems
Database Nodes:	1 Ultra Enterprise 2 Model 2200	2 Ultra Enterprise 1 Model 170
Processors	2 x 200 MHz UltraSPARC RISC	1 x 167 MHz UltraSPARC each
Cache memory	16KB D, 16KB I, 1MB External	32KB (D+I), 512KB External each
Main memory	2048 MB	512 MB each
Disk controllers	4 FC25/S + 1 SCSI/S	1 x SCSI-2 each
Disk Drives	122 x 2.1 GB SCSI	1 x 2.1GB SCSI-2 in each
Total Disk Storage	256 GB	2 GB in each
Terminals	1 Console	2 Consoles
10 BaseT Hubs	None	118 x 24-Port

Sun Microsystems

Ultra Enterprise 2 C/S w/2 Front-Ends

TPC-C

Report Date: 11/18/96

Order Number	Description	Quantity	Unit Price	Extended Price	Maint. rate/unit ¤	5 yr total Maint.
Server Hardware						
Pricing from CAT Technology, Inc.						
A14-UCB2-9S-256EB	Ultra Enterprise 2 Model 2200 (2*200MHz proc, 1MB Ecache, 256-MB RAM+ 2*2.1GB SCSI disk)	1	24371.25	24371.25	225	12150
X7004A	256MB SIMM expansion	8	5325.00	42600.00		
792A	SSA Model 112 w/ 30 2.1 GB Disks	4	35100.00	140400	120	25920
1057A	FibreChannel S/Bus Host Adapter	3	1200.00	3600		
595A	FibreChannel Optical Module	1	450.00	450		
X6256A	4-8GB 4mm DDS2 Internal Drive	1	1125.00	1125		
	Ultra Enterprise 2 Subtotal:			**212,546.25**		**38,070.00**
Client Hardwaer						
Pricing from CAT Technology, Inc.						
A11-UBA1-9S-064CB	Ultra Enterprise 1 Model 170 (64MB RAM + 1 x 2.1GB SCSI Disk)	2	9746.25	19493	76	8208
X1058A	SBus Quad Ethernet Controller	2	746.00	1492		
X7003A	128 MB SIMM expansion	8	1395.00	11160		
	Ultra Enterprise 1 Subtotal:			**32,144.50**		**8,208.00**
User Connectivity						
Terminal Pricing from SunExpress, Hubs from MicroWarehouse, Inc.						
SX-WY35-G-G35	Wyse 35 terminal for console	3	295.00	885		
Data Com/Bocca 24 plus	24 Port Hub (10% spare)	118	349.00	41182		
	Terminals and Connectivity Subtotal:			**42,067.00**		
	ALL HARDWARE SUBTOTALS:			**286,757.75**		**46,278.00**
Server and Client Software						
Sun Softwrore Pricing from Sun						
SOLS-C	Solaris Software on CD-ROM	1	100	100		
CC-3.0.1-S	SPARCompiler Single User License	1	695	695	20	1,140.00
	Sun Software Subtotal:			**795.00**		**1,140.00**
DB2 Pricing from IBM DB2						
DB2 v2 for common server	DB2 for common server (38 users)	1	6209	6209		0.00
	DB2 Software Subtotal:			**6,209.00**		
Client Software						
Tuxedo Pricing from BEA Systems, Inc.						
	Tuxedo 4.2.1 License	2	7500	15000	1125	11,250.00
	Tuxedo Software Subtotal:			**15,000.00**		**11,250.00**
	ALL SOFTWARE SUBTOTALS:			**22,004.00**		**12,390.00**

Sun maintenance rate/unit is monthly rate and Tuxedo maintenance rate/unit is yearly rate
H/W Maintenance 1 YR warranty (maintenance at 50%) & 4 YR Maintenance

Discount			
Sun Prepaid Maintenance Discount (Hardware)	15%		-6,941.70
	TOTAL HARDWARE AND SOFTWARE COST (5 YR)		**360,488.05**
	Price Per TpmC		116.02

116.02

Numerical Quantities Summary for Ultra Enterprise 2 Model 2200

MQTH, Computed Maximum Qualified Throughput **3107.17 tpmC**
% throughput difference, reported & reproducibility runs < 0.5%

Response Times (in secs)	90th Percentile	Average	Maximum
Menu	0.25	0.02	0.70
New-Order	1.40	0.79	11.67
Payment	1.40	0.63	4.88
Order-Status	1.20	0.56	4.47
Delivery(interactive)	0.04	0.02	0.33
Delivery(deferred)	2.00	0.73	5.00
Stock-level	2.00	0.90	6.00

Transaction Mix, in percent of total transactions

New-Order	44.80%
Payment	43.12%
Order-Status	4.03%
Delivery	4.03%
Stock-level	4.02%

Keying/Think Times (in secs)	Average.	Min.	Maximum
New-Order	18.02/12.19	18.01/0	18.09/122.0
Payment	3.01/12.28	3.01/0	3.09/122.0
Order-Status	2.02/10.22	2.01/0	2.05/90.5
Delivery	2.02/5.19	2.01/0	2.11/52.0
Stock-level	2.02/5.09	2.01/0	2.05/52.0

Test Duration

Ramp-up time	15 minutes
Measurement Interval	30 minutes
Number of transactions (all types) completed in measurement interval	208050

Contents

Figures

Tables

Preface

This report documents the compliance of the Sun Microsystems TPC Benchmark ™C testing on the Ultra Enterprise 2 Model 2200 running IBM DB2 for Solaris V2.1.1 RDBMS with the *TPC Benchmark ™C Standard Revision 3.0.*

These tests were run using the IBM DB2 for Solaris V2.1.1 RDBMS running with Solaris 2.5.1 and iTi Tuxedo 4.2.1 on the Ultra Enterprise 2.

Document Structure

The *TPC Benchmark ™C Full Disclosure Report* is organized as follows:

- The main body of the document lists each item in Clause 8 of the TPC Benchmark ™C Standard and explains how each specification is satisfied.
- Appendix A contains the application source code that implements the transactions and forms modules.
- Appendix B contains the code used to create and load the database.
- Appendix C contains the configuration information for the operating system, the RDBMS and Tuxedo.
- Appendix D contains the 180-day space calculations.
- Appendix E contains the code used to generate transactions and measure response times.
- Appendix F contains the screen layouts of all the forms.
- Appendix G contains the hardware and software price quotes.

Additional Copies

To request additional copies of this report, write to the following address:

TPC Benchmark Reports
Sun Microsystems. Inc.
2550 Garcia Ave.
Mountain View, CA, 94043

Attn.: Rod Tansimore
Mailstop: UPAL1-324

Or, copies may be ordered from the administrator of the TPC:

Shanley P.R.
777 N First Street, Suite 600
San Jose, CA 95112-6311
(408) 295-8894
FAX (408) 295-2613

TPC Benchmark ™C Full Disclosure

Introduction

The *TPC Benchmark ™C Standard Specification* requires test sponsors to publish, and make available to the public, a full disclosure report for the results to be considered compliant with the Standard. The required contents of the full disclosure report are specified in Clause 8.

This report is intended to satisfy the Standard's requirement for full disclosure. It documents the compliance of the benchmark tests reported in the *TPC Benchmark ™C* results for the Sun Microsystems Ultra Enterprise 2 Model 2200 running the IBM DB2 for Solaris V2.1.1 RDBMS.

In the *Standard Specification*, the main headings in Clause 8 are keyed to the other clauses. The headings in this report use the same sequence, so that they correspond to the titles or subjects referred to in Clause 8.

Each section in this report begins with the text of the corresponding item from Clause 8 of the *Standard Specification*, printed in italic type. The plain type text that follows explains how the tests comply with the TPC Benchmark ™C requirement. In sections where Clause 8 requires extensive listings, the section refers to the appropriate appendix at the end of this report.

1- General Items

1.1 Application Code and Definition Statements

The application program (as defined in Clause 2.1.7) must be disclosed. This includes, but is not limited to, the code implementing the five transactions and the terminal input and output functions.

Appendix A contains the application source code that implements the transactions and forms modules.

1.2 Sponsors

A statement identifying the benchmark sponsor(s) and other participating companies must be provided.

This benchmark test was sponsored by Sun Microsystems Computer Company and IBM Corporation.

1.3 Parameter Settings

Settings must be provided for all customer-tunable parameters and options which have been changed from the defaults found in actual products, including but not limited to:

- *Database tuning options*
- *Recovery/commit options*
- *Consistency/locking options*
- *Operating system and application configuration parameters*
- *Compilation and linkage options and run-time optimizations used to create/install applications, OS, and/or databases.*

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

Appendix C contains all the required parameter settings.

1.4 Configuration Diagrams

Diagrams of both measured and priced configurations must be provided, accompanied by a description of the differences.

Figure 1 is a diagram of the benchmark configuration. Figure 2 is a configuration diagram of the priced system.

1.4.1 Configuration Items for the Ultra Enterprise 2 Model 2200

For the priced configuration, the server machine was a Ultra Enterprise 2 Model 2200 which consisted of the following:

- Two UltraSPARC I 200 MHz Processors.
- 2048 MB of main memory.
- Two Internal 2.1GB disks.
- Four SPARCstorage Arrays Model 112 (30 x 2.1GB SCSI disks per Array).
- Three Fiber Channel Host Adaptors.
- One Fiber Channel Optical Module.
- One Quad Ethernet Controller.
- One Internal CD-ROM
- 4-8 GB Backup Tape Device.

The benchmark configuration used the same configuration as above except that three SPARCstorage Arrays were used rather than four.

For both the priced and benchmark configurations, the client machines were two Ultra Enterprise 1 Model 170s. Each contained:

- One UltraSPARC 167 MHz Processor.
- 512 MB of Main Memory.
- One Internal 2.1 GB SCSI disk.
- One Quad Ethernet Controller.
- One Internal CD-ROM.

The benchmark configuration used a Remote Terminal Emulator (RTE) to emulate TPC-C user sessions. The driver systems were directly connected through ethernet to the two Ultra Enterprise 1 Model 170s which emulated the database client sessions.

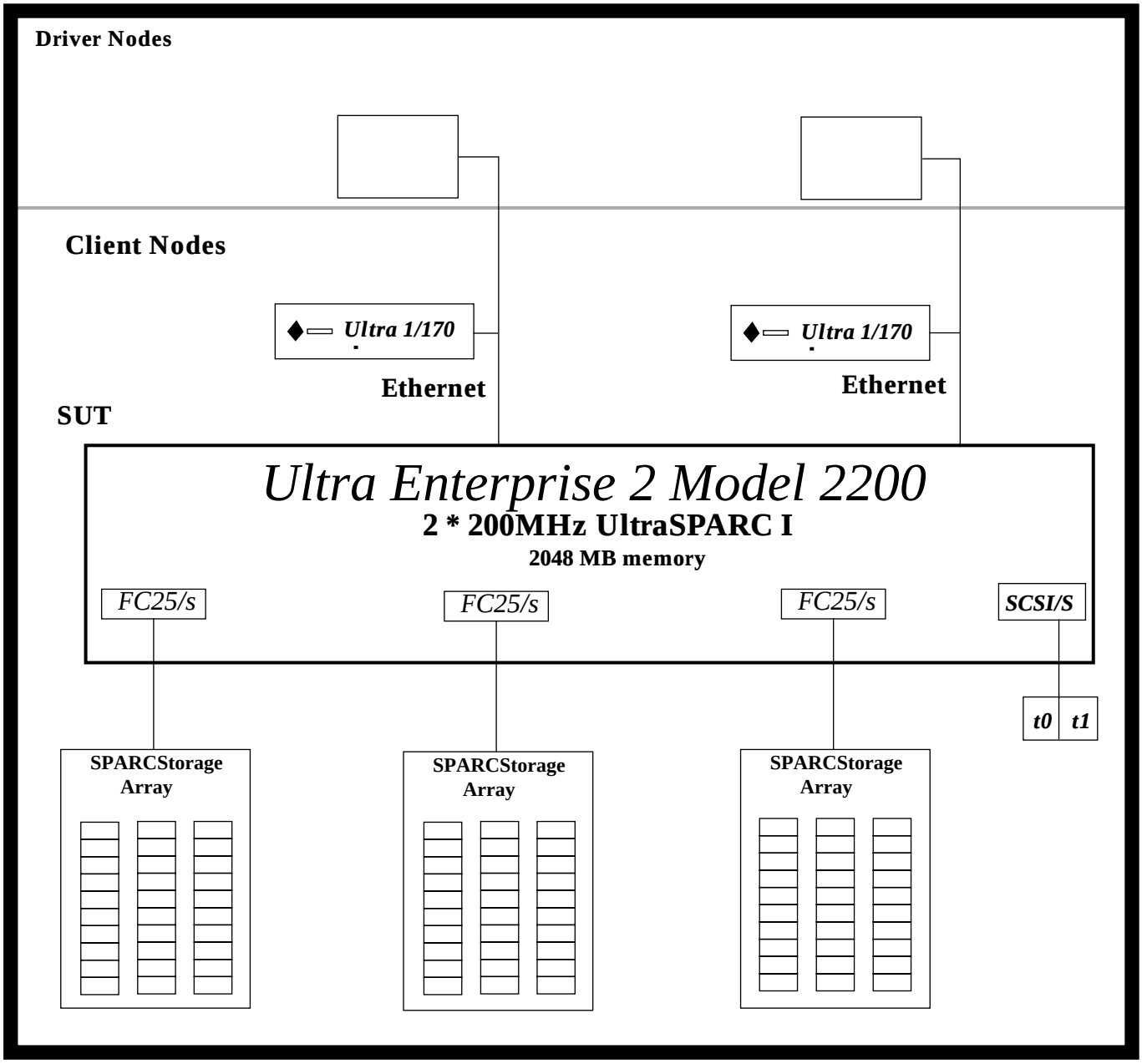


Figure 1: The Sun Ultra Enterprise 2 Model 2200 Benchmark Configuration

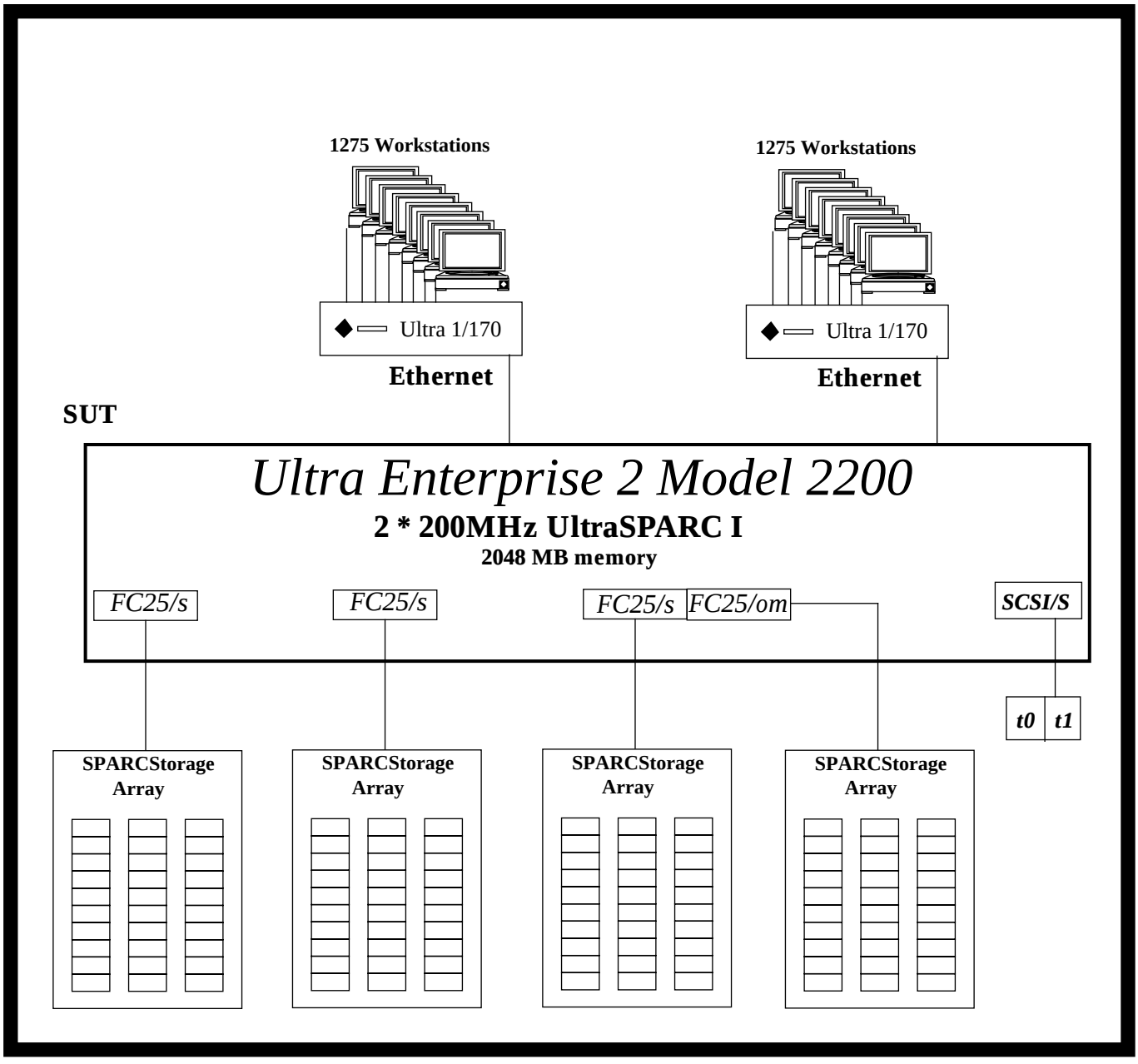


Figure 2: The Sun Ultra Enterprise 2 Model 2200 Priced Configuration.

2 - Clause 1 Related Items

2.1 Table Definitions

Listing must be provided for all table definition statements and all other statements used to set up the database.

Appendix B describes the programs that define, create, and populate an IBM DB2 for Solaris V2.1.1 database for TPC-C testing.

2.2 Physical Organization of Database

The physical organization of tables and indices, within the database, must be disclosed.

Space was allocated to IBM DB2 for Solaris V2.1.1 on the server according to the data in section 5.2. The size of the database devices on each disk drive was calculated to provide even distribution of load across the disk drives.

2.3 Insert and Delete Operations

It must be ascertained that insert and/or delete operations to any of the tables can occur concurrently with the TPC-C transaction mix. Furthermore, any restrictions in the SUT database implementation that precludes inserts beyond the limits defined in Clause 1.4.11 must be disclosed. This includes the maximum number of rows that can be inserted and the maximum key value for these new rows.

There were no restrictions on insert and delete operations to any tables beyond the limits defined in Clause 1.4.11.

2.4 Partitioning

While there are a few restrictions placed upon horizontal or vertical partitioning of tables and rows in the TPC-C benchmark (see Clause 1.6), any such partitioning must be disclosed.

Partitioning was not used in this implementation.

2.5 Table Replication

Replication of tables, if used, must be disclosed (see Clause 1.4.6).

No tables were replicated in this implementation.

2.6 Table Attributes

Additional and/or duplicated attributes in any table must be disclosed along with a statement on the impact on performance (see Clause 1.4.7).

No additional or duplicate attributes were added to any of the tables.

3 - Clause 2 Related Items

3.1 Random Number Generation

The method of verification for the random number generation must be described.

The Random Number Generator used was the one that appeared in the article titled “Random Number Generators: Good Ones Are Hard To Find” in the communications of the ACM - October 1988, Volume 31, Number 10. The properties of this random number generator are well-known and are documented in the article as producing a uniformly distributed pseudo-random sequence. To generate a random number, the driver programs first use a seed based on the host address, current time and the process-id of the respective session. This guarantees that each emulated user on all the RTE machines is mathematically independent of others.

3.2 Input/Output Screen Layouts

The actual layout of the terminal input/output screens must be disclosed.

The screen layouts are shown in Appendix F.

3.3 Terminal Feature Verification

The method used to verify that the emulated terminals provide all the features described in Clause 2.2.2.4 must be explained.

The terminal features were verified by manually exercising each specification on a representative Ultra Enterprise 1 running Solaris 2.5.1.

3.4 Presentation Manager or Intelligent Terminal

Any usage of presentation managers or intelligent terminals must be explained.

The TPC-C forms module was implemented using the standard System V curses libraries.

3.5 Transaction Statistics

Table 1 lists the numerical quantities that Clauses 8.1.3.5 to 8.1.3.11 requires.

Table 1: Transaction Statistics

Transaction Type	Statistics	Percentage
New Order	Home warehouse	99.02
	Remote warehouse	0.98
	Rolled back transactions	1.03
	Average items per order	9.99
Payment	Home warehouse	85.15
	Remote warehouse	14.85
	Non-primary key access	59.92
Order Status	Non-primary key access	60.35
Delivery	Skipped transactions	0.00
Transaction Mix	New order	44.80
	Payment	43.12
	Order status	4.03
	Delivery	4.03
	Stock level	4.02

3.6 Queueing Mechanism

The queueing mechanism used to defer the execution of the Delivery transaction must be disclosed.

Delivery transactions were submitted to servers using the same Tuxedo call mechanism that other transactions used. The only difference was that the call was asynchronous - i.e., control returned to the client process immediately and the deferred delivery completed asynchronously.

4 - Clause 3 Related Items

4.1 Transaction System Properties (ACID)

The results of the ACID tests must be disclosed along with a description of how the ACID requirements were met. This includes disclosing which case was followed for the execution of Isolation Test 7.

The TPC Benchmark C Standard Specification defines a set of transaction processing system properties that a system under test (SUT) must support during the execution of the benchmark. Those properties are Atomicity, Consistency, Isolation, and Durability (ACID). This section defines each of these properties, describes the steps taken to ensure that they were present during the test and describes a series of tests done to demonstrate compliance with the standard.

4.2 Atomicity

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

Tests waived by the auditor since they had been previously performed.

4.3 Consistency

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

The TPC Benchmark C standard requires System Under Test to meet the 12 consistency conditions listed in Clause 3.3.2.

The TPC Benchmark C Standard Specification requires explicit demonstration that the conditions are satisfied for the first four conditions only.

In order to demonstrate the consistency of the application, the following steps were taken:

1. Prior to the start of the benchmark run, the consistency of the database was verified by applying the consistency conditions 1-4 described above.

2. A fully-scaled run with a 30 minute steady state period was executed.
3. Upon the completion of the benchmark, the consistency of the database was determined by applying the same consistency conditions used in step 1.

4.4 Isolation

Isolation can be defined in terms of phenomena that can occur during the execution of concurrent transactions. These phenomena are P0 ("Dirty Write"), P1 ("Dirty Read"), P2 ("Non-repeatable Read") and P3 ("Phantom"). The table in Clause 3.4.1 of the TPC-C specifications defines the isolation requirements which must be met by the TPC-C transactions. Sufficient conditions must be enabled at either the system or application level to ensure the required isolation is maintained.

Tests waived by the auditor since they had been previously performed.

4.4.1 Stock-Level Test

This test verifies that the Stock-Level transaction reads only committed rows no older than the last 20 orders (Clause 9.2.2.6).

Tests waived by the auditor since they had been previously performed.

4.5 Durability

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

List of single failures:

Permanent irrecoverable failure of any single durable medium containing TPC-C database tables or recovery log data.

Instantaneous interruption (system crash/system hang) in processing which requires system reboot to recover.

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

Sun Microsystems executed one durability test to satisfy the durability requirements for this implementation of TPC Benchmark C. The combined test for loss of memory and instantaneous interruption was performed with a fully

scaled database under the full load of terminals. The test for loss of data and the test for loss of log were waived by the auditor since they had been previously performed.

4.5.1 Permanent Irrecoverable Failure

The loss of data disk and loss of log disk tests were waived by the auditor since they had been previously performed.

4.5.2 Instantaneous Interruption and Loss of Memory

Instantaneous interruption and loss of memory tests were combined because the loss of power erases the contents of memory. This failure was induced by removing the SUT's primary power while the benchmark was running.

1. The D_NEXT_O_ID fields for all rows in district table were summed up to determine the initial count of the total number of orders (count1).

2. A fully-scaled test was executed. On the driver system, the committed and rolled back New-Order transactions were recorded.

3. After 5 minutes into the measurement period, the SUT's primary power was removed.

4. The test was aborted on the driver.

5. Power was restored to the SUT and a normal system recovery was done. A recovery was automatically performed by IBM DB2 for Solaris V2.1.1 when the database was restarted and brought on-line. The recovery restored the database to the consistent point just after the last committed transaction had occurred before the induced failure.

6. The count of committed and rolled back New-Order transactions from the driver and the number of orders in the ORDERS table were compared. The number of transactions missed "in flight" was zero.

7. Step 1 was repeated to determine the total number of orders (count2). Count2-count1 was compared with the number of committed records.

5 - Clause 4 Related Items

5.1 Initial Cardinality of Tables

The Cardinality (e.g. number of rows) of each table, as it existed at the start of the benchmark run (see Clause 4.2), must be disclosed. If the database was over-scaled and inactive rows of the WAREHOUSE table were deleted (see Clause 4.2.2) the cardinality of the WAREHOUSE table as initially configured and the number of rows deleted must be disclosed.

The TPC-C database for this test was configured with 255 warehouses.

Table 2: Cardinality of Tables

Table	Occurrences
Warehouse	255
District	2550
Customer	7650000
History	7650000
Orders	7650000
New order	2295000
Order line	76491850
Stock	25500000
Item	100000

5.2 Database Layout

The distribution of tables and logs across all media must be explicitly depicted for the tested and priced systems.

Figure 3 depicts the distribution of the database for the tested system. Figure 4 depicts the distribution of the database for the priced system.

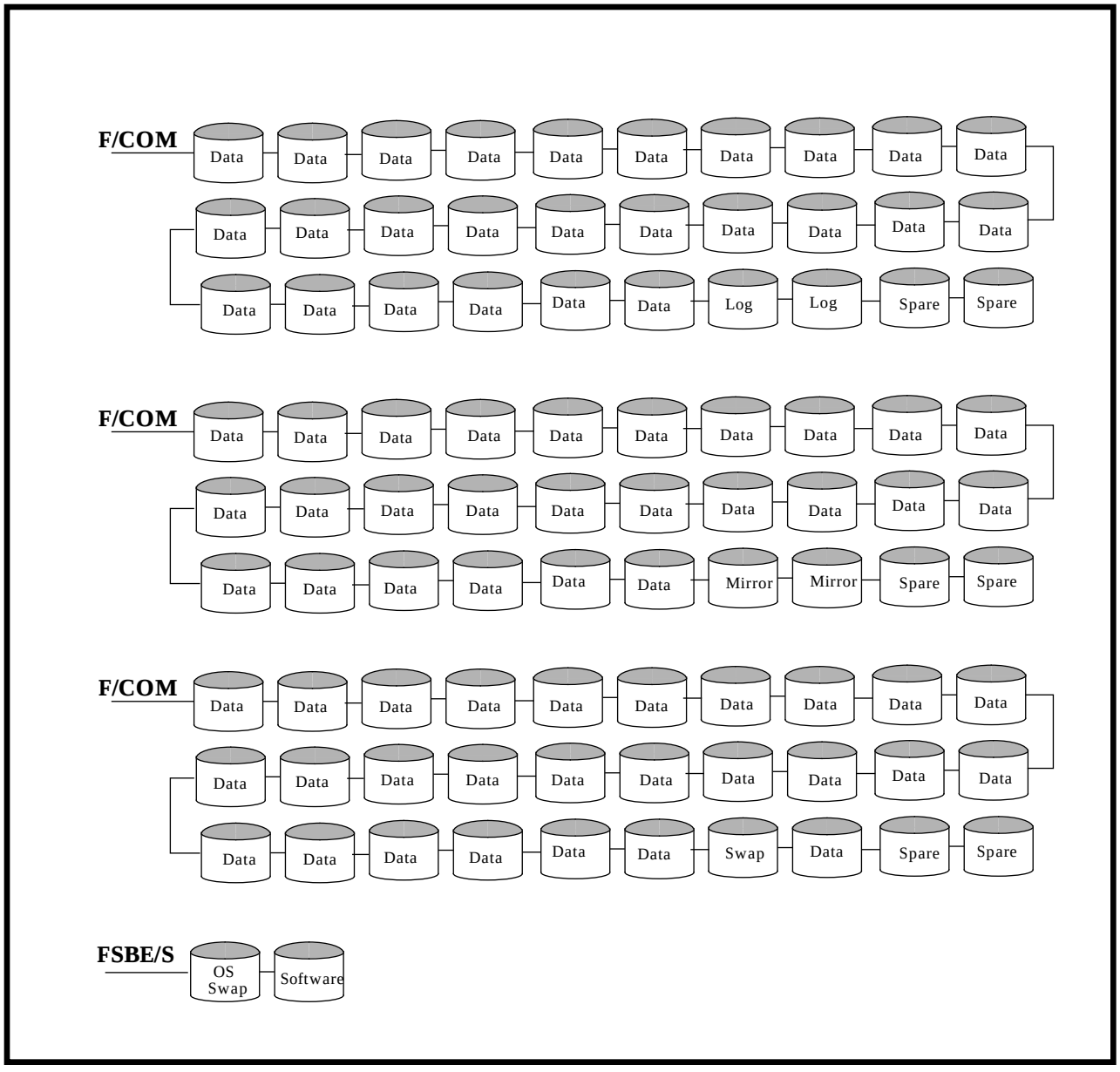


Figure 3: Database Layout of the Tested System

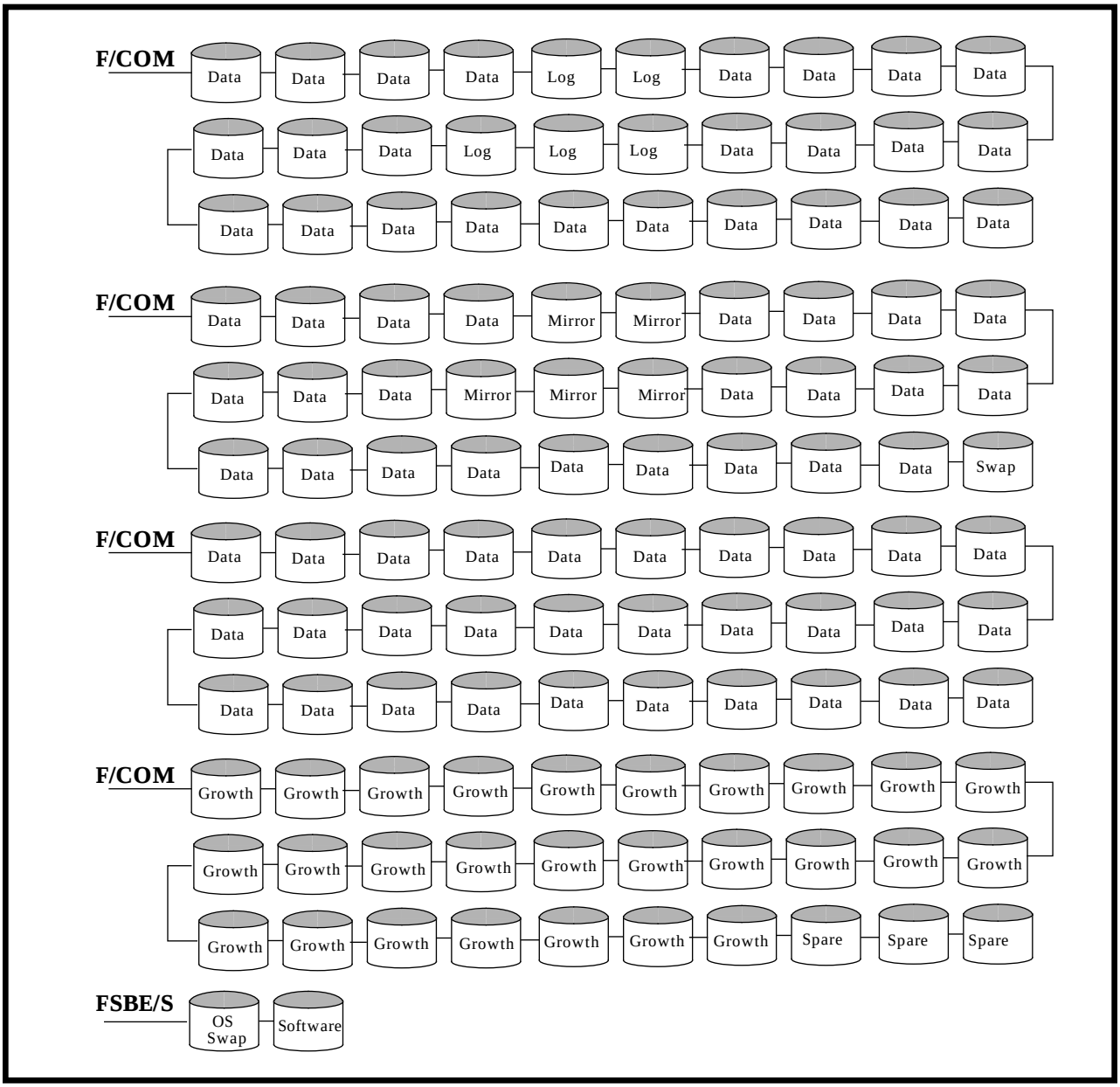


Figure 4: Database Layout of the Priced System

5.3 Type of Database

A statement must be provided that describes:

1. The data model implemented by the DBMS used (e.g., relational, network hierarchical).

2. The database interface (e.g., embedded, call level) and access language (e.g., SQL, DL/1, COBOL read/write) used to implement the TPC-C transactions. If more than one interface/access language is used to implement TPC-C, each interface/access language must be described and a list of which interface/access language is used with which transaction type must be disclosed.

IBM DB2 for Solaris V2.1.1 is a relational database management system.The interface used was Embedded C and stored procedures.

5.4 Mapping of Database

The mapping of database partitions/replications must be explicitly described.

No table partitioning or replication was used.

5.5 180 Day Space Computation

Details of the 180 day space computations along with proof that the database is configured to sustain 8 hours of growth for the dynamic tables (Order, Order-Line, and History) must be disclosed (see Clause 4.2.3).

The 180 day space computation is shown in Appendix D.

6 - Clause 5 Related Items

6.1 Measured tpmC

Measured tpmC must be reported.

The measured tpmC was *3107.17*

6.2 Response Times

Ninetieth percentile, maximum and average response times must reported for all transaction types as well as for the menu response time.

Table 3: Response Times

Type	Average	Maximum	90% percentile
New-Order	0.79	11.67	1.40
Payment	0.63	4.88	1.40
Order-Status	0.56	4.47	1.20
Interactive Delivery	0.02	0.33	0.04
Deferred Delivery	0.73	5.00	2.00
Stock-Level	0.90	6.00	2.00
Menu	0.02	0.70	0.25

6.3 Keying and Think Times

The minimum, the average, and the maximum keying and think times must be reported for all transaction types.

Table 4: Keying Times

Type	Average	Maximum	Minimum
New-Order	18.02	18.09	18.01
Payment	3.01	3.09	3.01
Order-Status	2.02	2.05	2.01
Interactive Delivery	2.02	2.11	2.01
Stock-Level	2.02	2.05	2.01

Table 5: Think Times

Type	Average	Maximum	Minimum
New-Order	12.19	122.00	0.00
Payment	12.28	122.00	0.00
Order-Status	10.22	90.53	0.00
Interactive Delivery	5.19	52.00	0.00
Stock-Level	5.09	52.00	0.00

6.4 Response Time Frequency Distribution Curves

Response Time frequency distribution curves (see Clause 5.6.1) must be reported for each transaction type.

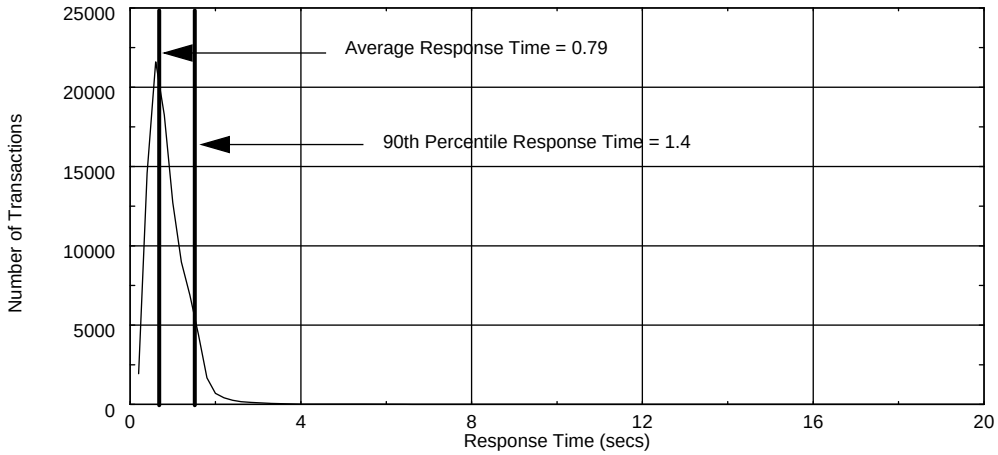


Figure 5: New Order Response Time Distribution

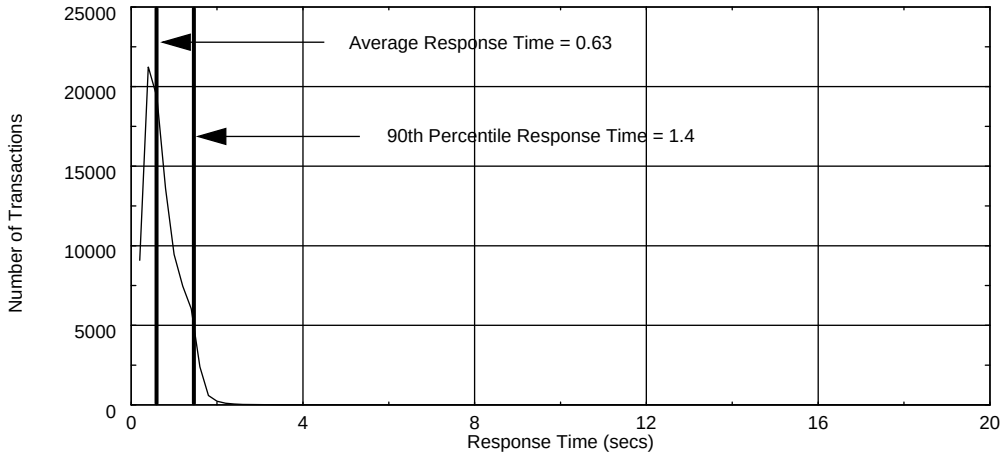


Figure 6: Payment Response Time Distribution

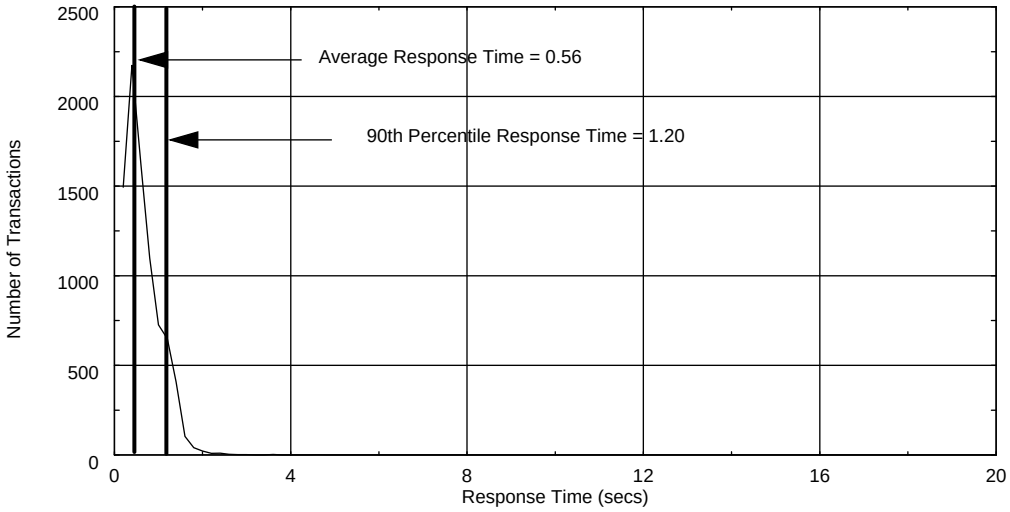


Figure 7: Order Status Response Time Distribution

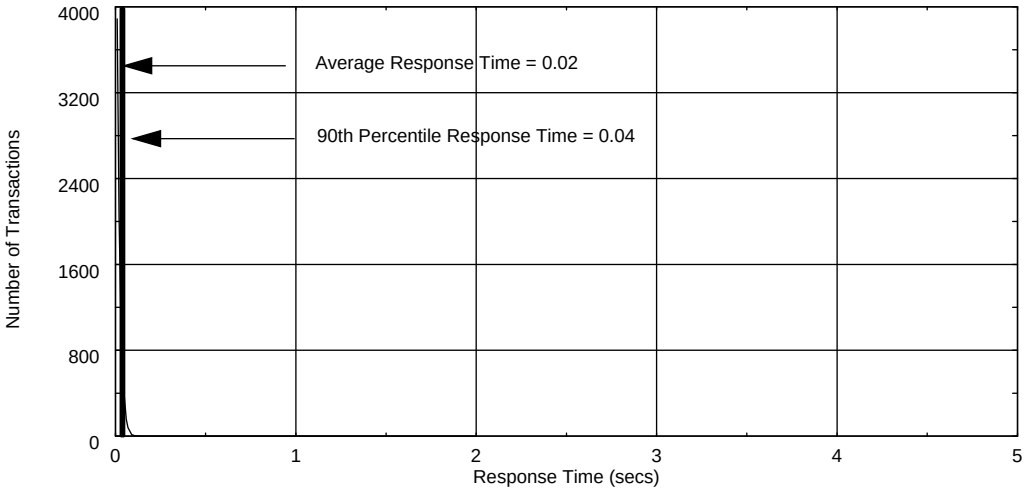


Figure 8: Delivery Response Time Distribution

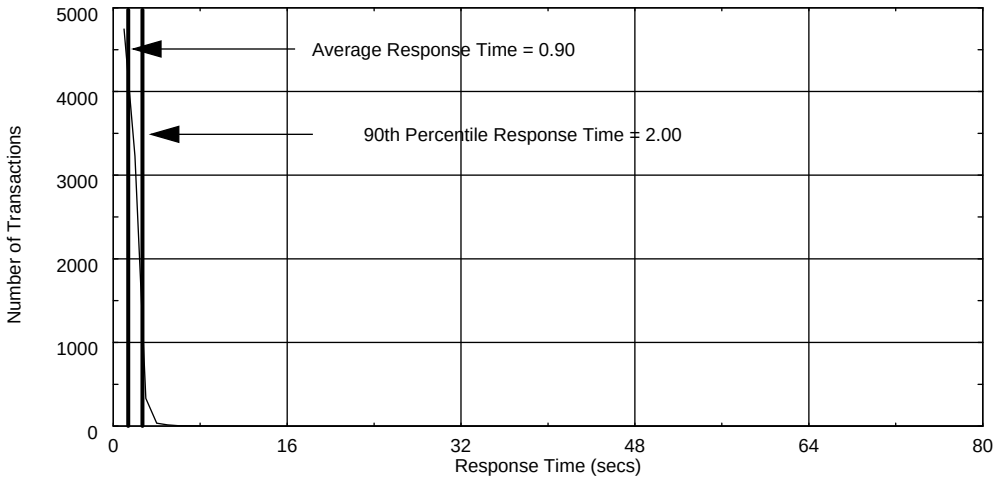


Figure 9: Stock Level Response Time Distribution

6.5 Response time versus throughput

The performance curve for response times versus throughput (see Clause 5.6.2) must be reported for the New Order transaction.

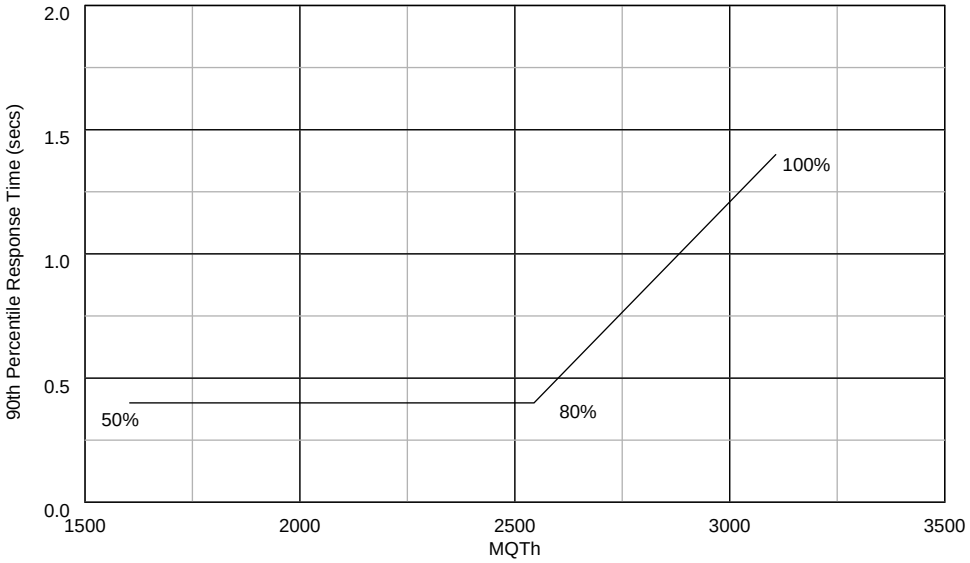


Figure 10: Response Time versus Throughput

6.6 Think Time distribution curves

Think Time frequency distribution curves (see Clause 5.6.3) must be reported for each transaction type.

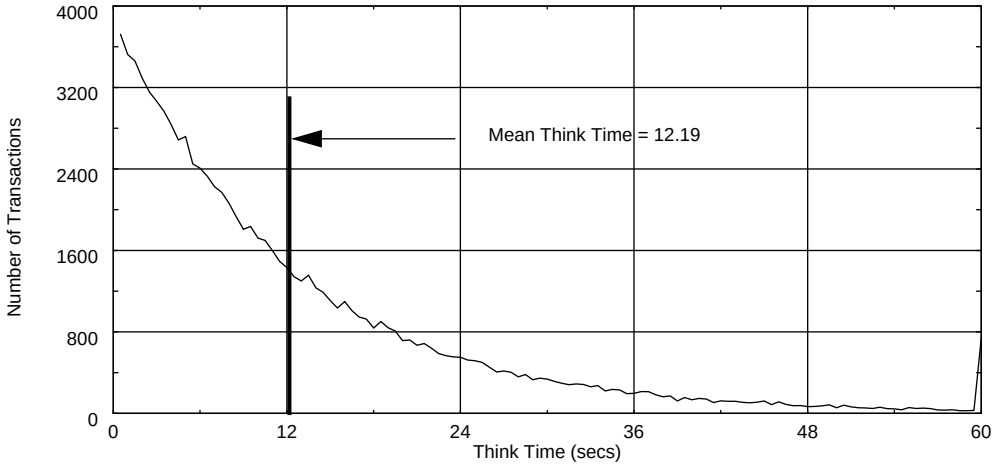


Figure 11: New Order Think Time Distribution

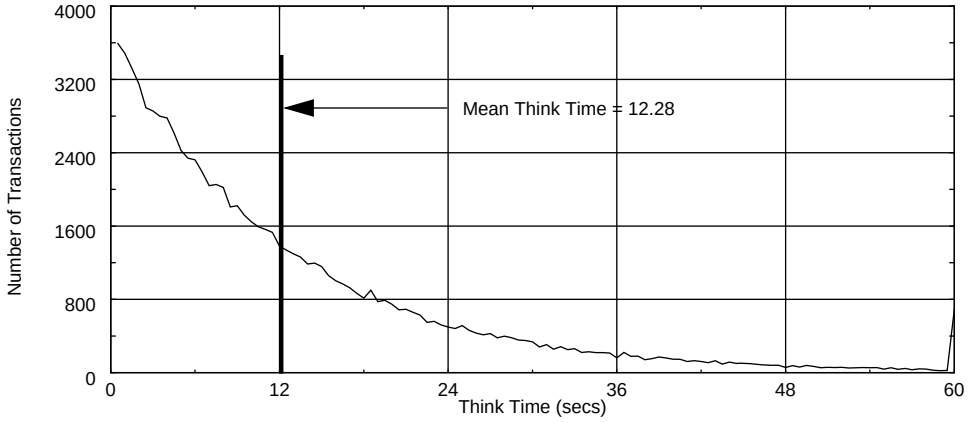


Figure 12: Payment Think Time Distribution

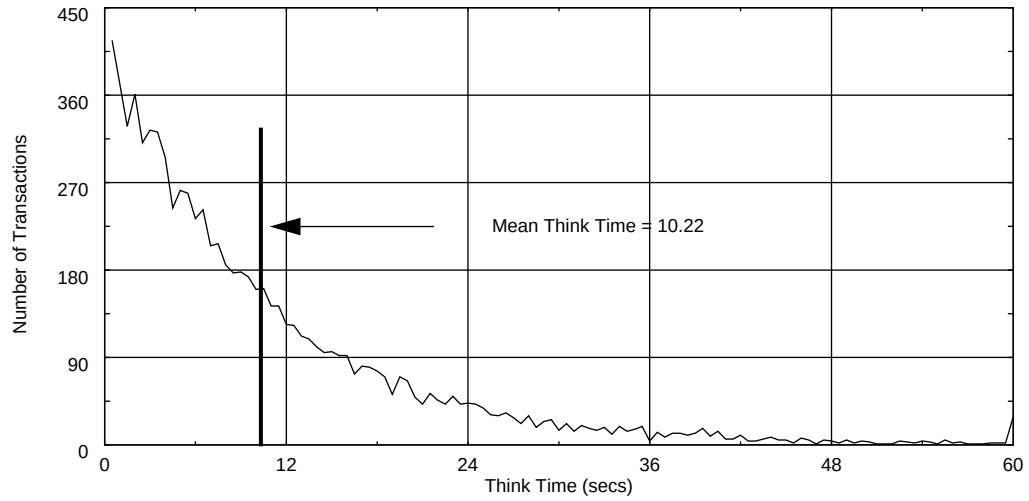


Figure 13: Order Status Think Time Distribution

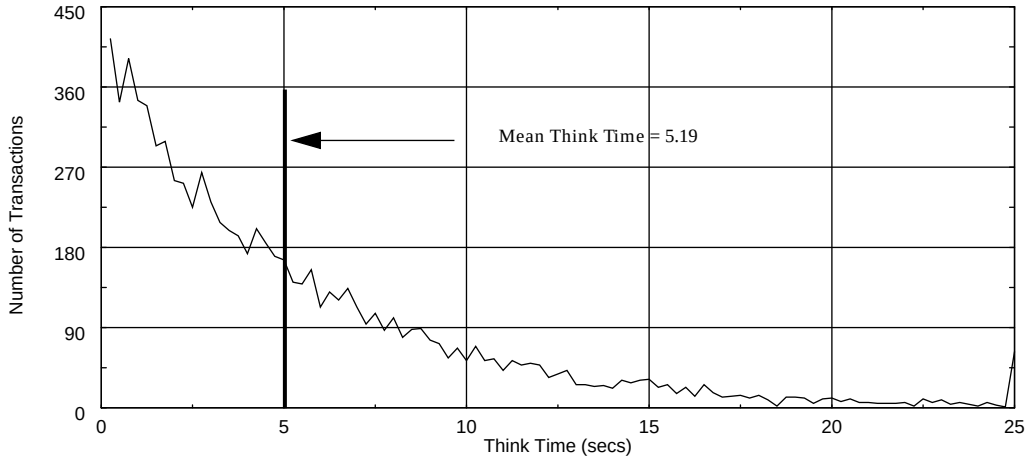


Figure 14: Delivery Think Time Distribution

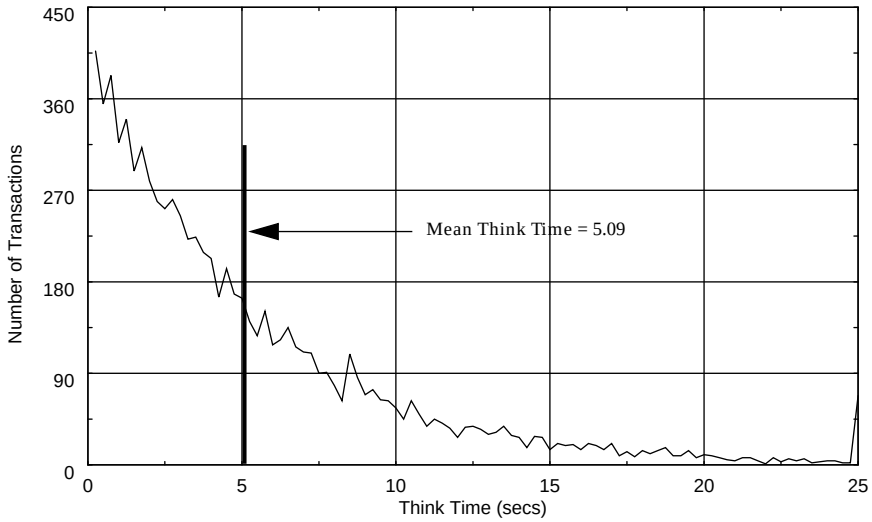


Figure 15: Stock Level Think Time Distribution

6.7 Keying Time Frequency Distribution Curves

Keying Time frequency distribution curves (see Clause 5.6.4) must be reported for each transaction type.

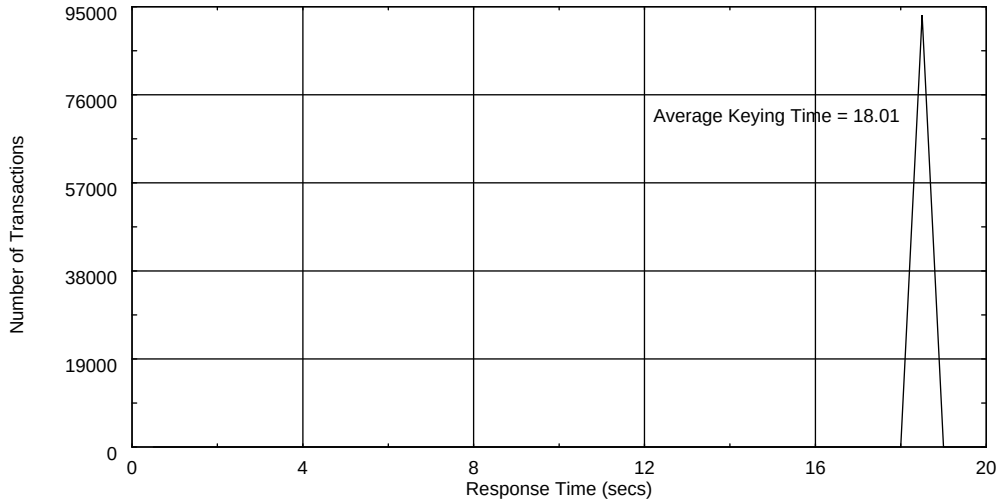


Figure 16: New Order Keying Time Distribution

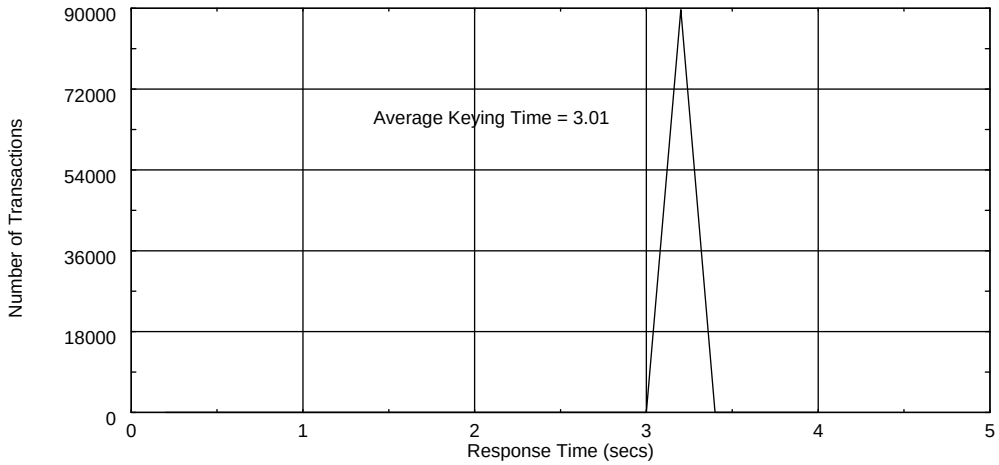


Figure 17: Payment Keying Time Distribution

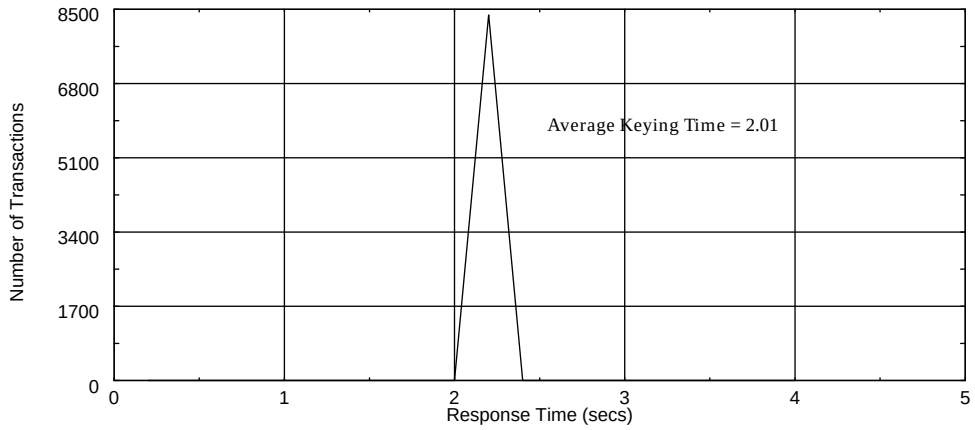


Figure 18: Order Status Keying Time Distribution

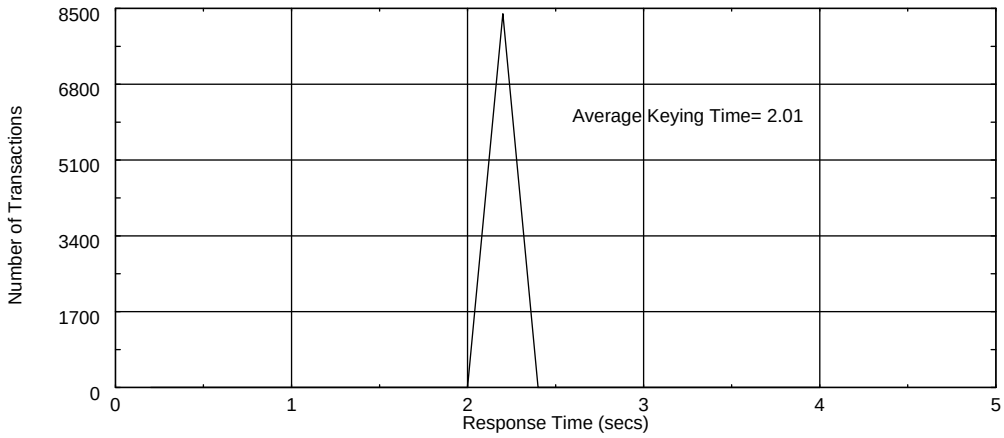


Figure 19: Delivery Keying Time Distribution

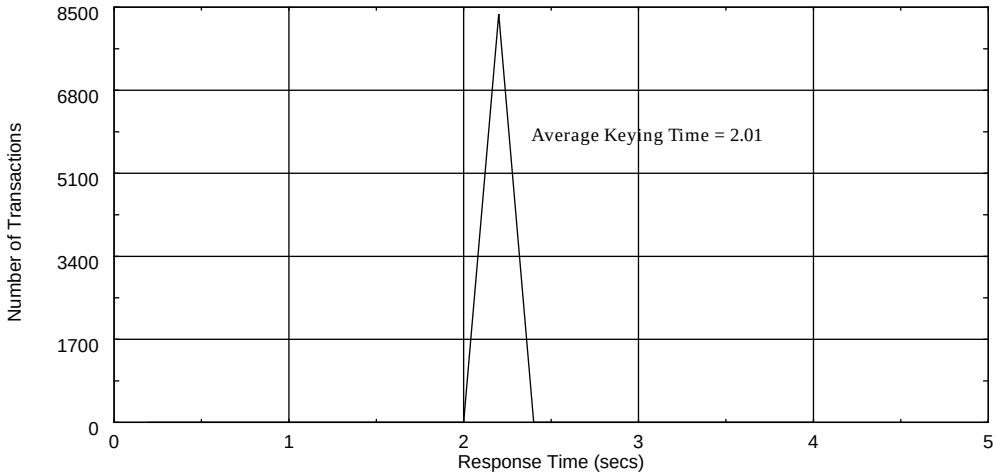


Figure 20: Stock Level Keying Time Distribution

6.8 Throughput versus Elapsed Time

A graph of throughput versus elapsed time (see Clause 6.6.5) must be reported for the New-Order transaction.

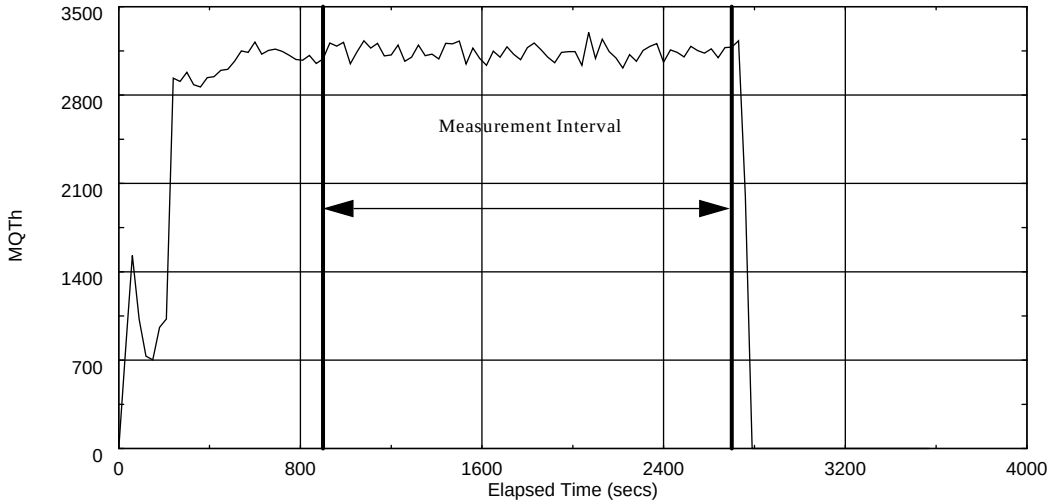


Figure 21: Throughput versus Time

6.9 Steady State Determination

The method used to determine that the SUT had reached a steady state prior to commencing the measurement interval (see Clause 5.5) must be described.

The transaction throughput rate (tpmC) and response times were relatively constant after the initial 'ramp up' period. The throughput and response time were verified by examining the throughput (tpmC) graph reported at 30 second intervals for the duration of the benchmark. Ramp up, steady state, and ramp down are clearly discernible in the graph, Figure 21.

6.10 Work Performed During Steady State

A description of how the work normally performed during a sustained test (for example checkpointing, writing redo/undo log records, etc.), actually occurred during the measurement interval must be reported.

6.10.1 Checkpoint

DB2 for Solaris uses a write-ahead-logging protocol to guarantee recovery. This protocol uses 'Soft' checkpoint to write least-recently-used database pages to disk independent of transaction commit. However, enough log information to redo / undo the change to a database page is committed to disk before the database page itself is written. This protocol therefore renders checkpoint unnecessary for DB2 for Solaris. For a more detailed description of the general principles of the write-ahead-logging protocol, please refer to the IBM research paper "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging" by C.Mohan, Database Technology Institute, IBM Almaden Research Center.

6.11 Reproducibility

A description of the method used to determine the reproducibility of the measurement results must be reported.

In a repeat run, a throughput of 3104.63 tpmC was achieved.

6.12 Measurement Period Duration

A statement of the duration of the measurement interval for the reported Maximum Qualified Throughput (tpmC) must be included.

The measurement interval was 30 minutes.

6.13 Transaction Mix Regulation

The method of regulation of the transaction mix (e.g., card decks or weighted random distribution) must be described. If weighted distribution is used and the RTE adjusts the weights associated with each transaction type, the maximum adjustments to the weight from the initial value must be disclosed.

The weighted distribution algorithm as described in Clause 5.2.4.1 of the TPC-C specification was used to regulate the transaction mix. Weights for the various transactions were statically assigned.

6.14 Numerical Results

The percentage of the total mix for each transaction type must be disclosed.

See Table 1 for results.

6.15 New-Orders Rolled-Back

The percentage of New-Order transactions rolled back as a result of invalid item number must be disclosed.

See Table 1 for results.

6.16 Order-Line Average

The average number of order-lines entered per New-Order transaction must be disclosed.

See Table 1 for results.

6.17 Remote Order-Lines

The percentage of remote order-lines entered per New-Order transaction must be disclosed.

See Table 1 for results.

6.18 Remote Payments

The percentage of remote payment transactions must be disclosed.

See Table 1 for results.

6.19 Customer Lastname

The percentage of customer selections by customer last name in the Payment and Order-Status transactions must be disclosed.

See Table 1 for results.

6.20 Deliverys Skipped

The percentage of Delivery transactions skipped due to there being fewer than necessary orders in the New-Order table must be disclosed.

See Table 1 for results.

6.21 Checkpoints

The number of checkpoints in the Measurement Interval, the time in seconds from the start of the Measurement Interval to the first checkpoint and the Checkpoint Interval must be disclosed.

DB2 for Solaris uses a write-ahead-logging protocol to guarantee recovery. This protocol uses ‘Soft’ checkpoint to write least-recently-used database pages to disk independent of transaction commit. However, enough log information to redo / undo the change to a database page is committed to disk before the database page itself is written. This protocol therefore renders checkpoint unnecessary for DB2 for Solaris. For a more detailed description of the general principles of the write-ahead-logging protocol, please refer to the IBM research

paper “ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging” by C.Mohan, Database Technology Institute, IBM Almaden Research Center.

7 - Clause 6 Related Items

7.1 RTE Description

If the RTE is commercially available, then its inputs must be specified. Otherwise, a description must be supplied of what inputs (e.g. scripts) to the RTE had been used.

The RTE used was developed by Sun Microsystems Computer Company and is proprietary. It consists of a *master_rte* program which forks off the individual RTE processes and controls the run. After the run completes, a separate report generator program collects all the log files and generates the final statistics of a run.

Inputs to the RTE include the names of the RTE machines to run on, client machines to attach to, the database scale, the ramp-up, measurement and ramp-down times. The script used to set these values is shown below:

```
setenv driver   tpcc_master.com        #Name of driver script
setenv ramp_up          900           # ramp_up interval (secs)
setenv ramp_down        60            # ramp_down interval (secs)
setenv trigger_time    700            # Trigger time for synchronization
setenv scale           255             # Number of warehouses
setenv stdy_state      1800            # length in seconds of individual runs


# Enter the following parameters related to tpcc_master.com

set users = ( 1275 1275 )   # Number of slaves on each machine
set rte_machines = ( oakland munson ) # Names of rte machines
set clnt_machines = ( claremont jenner ) # Names of client machines (same # as #rtes)
```

The code used to generate the transactions and record response times is shown in Appendix E.

7.2 Emulated Components

It must be demonstrated that the functionality and performance of the components being emulated in the Driver System are equivalent to that of the priced system. The results of the test described in Clause 6.6.3.4 must be disclosed.

In the priced configuration, workstations are connected to the clients via telnet in the same way as the emulated system. In the tested configuration, the driver system emulates both the terminal and terminal server by making a direct connection to the SUT for each terminal.

7.3 Configuration Diagrams

A complete funcitional diagram of both the benchmark configuration and the configuration of the proposed (target) system must be disclosed. A detailed list of all software and hardware functionality being performed on the Driver System, and its interface to the SUT must be disclosed (see Clause 6.6.3.6).

Figure 1 is a diagram of the benchmarked configuration and Figure 2 shows the configuration of the priced configuration. Section 1.4 of this Full Disclosure Report gives details on both configurations.

7.4 Network Configuration

The network configurations of both the tested services and the proposed (target) services which are being represented and a thorough explanation of exactly which parts of the proposed configuration are being replaced with the Driver System must be disclosed (see Clause 6.6.4).

The tested configuration used one LAN for each driver system, connecting the driver system to the corresponding client system and the server system.

The target system is priced with one LAN between each client machine and the server. There are 3 LAN's between each client and the corresponding workstation "terminals". There were 425 workstation "terminals" on each LAN, a total of 1275 workstation "terminals" per client. This approach was used in a prior published TPC-C version 3.0 FDR and limits the number of workstation "terminals" to fewer than 500 per LAN as approved by the TPC TAB.

7.5 WAN/LAN Bandwidth

The bandwidth of the network(s) used in the tested/priced configuration must be disclosed.

Local area networks (LANs) with a bandwidth of ten (10) megabits per second were used in the tested / priced system.

7.6 Operator Intervention

If the configuration requires operator intervention, the mechanism and the frequency of this intervention must be disclosed.

The Ultra Enterprise 2 Model 2200 configuration reported does not require any operator intervention to sustain the reported throughput.

8 - Clause 7 Related Items

8.1 System Pricing

A detailed list of hardware and software used in the priced system must be reported. Each separately orderable item must have vendor part number, description, release/revision level, and either general availability status or committed delivery date. If package-pricing is used, vendor part number of the package and a description uniquely identifying each of the components of the package must be disclosed. Pricing source(s) and effective date(s) of price(s) must also be reported.

The Executive Summary on page vi lists pricing information for all components. The hardware pricing is from CAT Technology except for the Wyse terminals which are from SunExpress and the hubs which are from DataComm / Warehouse.

8.2 Support Pricing

The total 5-year price of the entire configuration must be reported including: hardware, software, and maintenance charges. Separate component pricing is recommended. The basis of all discounts used must be disclosed.

8.2.1 Sun Hardware and Software Support

The Silver Program of the SunService Support Program was used in all Sun pricing calculations. This program provides complete service with both on-site and telephone assistance. Features of this program include telephone assistance from 8:00 am to 5:00 pm, Monday - Friday; and on-site service assistance from 8:00 am to 5:00 pm, Monday - Friday; and Solaris maintenance releases. This service provides live telephone transfer of software fixes and 4 hour on-site response for urgent problems.

All Sun hardware has a one year warranty. During the warranty period, the monthly price for the Silver Program is 50% of the usual monthly price.

8.2.2 IBM DB2 for Solaris Standard Technical Support

IBM Standard Technical Support includes:

- Product maintenance updates and fixing of defects.
- Telephone access for product maintenance and fixing of defects, available from 9:00 a.m. to 5:00 p.m., Monday through Friday.

8.3 Discounts

The following generally available discounts to any buyer with like conditions were applied to the priced configurations:

- a 15% Sun support prepayment discount.

8.4 Availability

The Committed delivery date for general availability (availability date) of products used in the price calculations must be reported. When the priced system includes products with different availability dates, the reported availability date for the priced system must be the date at which all components are committed to be available.

Tuxedo 4.2.1 is generally available now. DB2 for Solaris 2.1.1 is generally available now from IBM Coproration; the benchmark made use of performance patches which will be generally available on May 15th 1996. All Sun Microsystems Computer Company products will be generally available by August 31st 1996.

8.5 TpmC, Price/TpmC

A statement of the measured tpmC, as well as the respective calculations for 5-year pricing, price/performance (price/tpmC), and the availability date must be included.

The Maximum Qualified Throughput for the Ultra Enterprise 2 Model 2200 was 3107.17 tpmC at $140.52 per tpmC. The availability date for the entire configuration is August 31st, 1996.

9 - Clause 8 Related Items

9.1 Auditor's Report

The auditor's name, address, phone number, and a copy of the auditor's attestation letter indicating compliance must be included in the Full Disclosure Report.

See attached attestation letter for the Sun report as well as the Auditor's name and address.

Appendix A: Application Code

This Appendix contains the application source code that implements the transactions and Forms modules.

```
/*
 * (c) Copyright Sun Microsystems Inc. 1994
 * Written: 15/10/94
 *
 */
#include <syslog.h>
#include <errno.h>
#include <stdio.h>
#include "tpcc_client.h"

int w_id = 1, d_id = 1, queue_no = 1;/* Global w-id, d_id */

/* Field structures */
struct no_struct neworder;
struct pay_struct payment;
struct del_struct delivery;
struct stock_struct stocklevel;
struct ord_struct ordstat;
char newo_service[5] = "NEWO", paym_service[5] = "PAYM";
char ords_service[5] = "ORDS", del_service[4] = "DEL";
char stock_service[6] = "STOCK";

main()
{

        do_init();

        while (1) {
        switch(put_menu()) {
        case NEWORDER:
                do_neworder();
                break;
        case PAYMENT:
                do_payment();
                break;
        case DELIVERY:
                do_delivery();
                break;
        case STOCKLEVEL:
                do_stocklevel();
                break;
        case ORDSTAT:
                do_ordstat();
                break;
        case EXIT:
                cleanup(0);
        }
        }
}


do_init()
{
        openlog("Client", LOG_PID|LOG_CONS, 0);
        init_scr();             /* Init screen */
        init_neworder();
        init_payment();
        init_ordstat();
        init_delivery();
        init_stocklevel();
}


init_neworder()
{
        neworder.w_id = w_id;/* Post wid of this client */
        init_newo_tx();
}

init_payment()
{
        payment.w_id = w_id;
        init_paym_tx();
}
```

```
init_ordstat()
{
        ordstat.w_id = w_id;
        init_ords_tx();
}

init_delivery()
{
        delivery.w_id = w_id;
        init_del_tx();
}

init_stocklevel()
{
        stocklevel.w_id = w_id;
        stocklevel.d_id = d_id;
        init_stock_tx();
}

do_neworder()
{
        int no_of_items;

        no_of_items = get_neworder();
        if (no_of_items == -1)
                return;
        if (neworder_tx(no_of_items))
                cleanup(TXERRCODE);
        else
{
                put_neworder(no_of_items);
}
}

do_payment()
{
        if (get_payment() == -1)
                return;
        if (payment_tx())
                cleanup(TXERRCODE);
        else
{
                put_payment();
}
}

do_delivery()
{
        if (get_delivery() == -1)
                return;
        if (queue_delivery())
                cleanup(TXERRCODE);
        else
                put_delivery();
}

do_stocklevel()
{
        if (get_stocklevel() == -1)
                return;
        if (stocklevel_tx())
                cleanup(TXERRCODE);
        else
                put_stocklevel();
}


do_ordstat()
{
        int no_of_items;

        if (get_ordstat() == -1)
                return;
        no_of_items = ordstat_tx();
        if (no_of_items == TXERRCODE)
                cleanup(TXERRCODE);
        else
                put_ordstat(no_of_items);
}


/*
 * (c) Copyright Sun Microsystems Inc. 1994
 * Written: 15/10/94
 *
 */


#include <syslog.h>
#include <errno.h>
#include <stdio.h>
#include "tpcc_term.h"
#include "tpcc_client.h"



char display_buffer[3000];
static char errmsg[80];         /* For error messages */
char blank_mesg[70]= "                                                                      ";
static int      print_dollar;
static int      err_flag;               /* Flag to indicate if errmsg printed */


extern int current_max_field;



/* Form field structures */
struct fld neworder_fld[16] = {
INT, 2, 28, 2, 1, (char *)&neworder.d_id, /* d-id */
INT, 4, 11, 3, 1, (char *)&neworder.c_id, /* c-id */
INT, 4, 11, 2, 0, (char *)&neworder.w_id, /* w-id */
STRING, 20, 60, 2, 0, neworder.o_entry_d, /* entry date */
STRING, 17, 24, 3, 0, neworder.c_last,
STRING, 3, 51, 3, 0, neworder.c_credit, /* customer credit */
AMOUNT, 5, 63, 3, 0, (char *)&neworder.c_discount, /* c-discount */
INT, 8, 14, 4, 0, (char *)&neworder.o_id, /* order-id */
INT, 2, 41, 4, 0, (char *)&neworder.o_ol_cnt, /* order-count */
AMOUNT, 5, 58, 4, 0, (char *)&neworder.w_tax, /* w_tax */
AMOUNT, 5, 73, 4, 0, (char *)&neworder.d_tax, /* d_tax */
STRING, 25, 18, 22, 0, neworder.status, /* Execution status */
AMOUNT, 9, 69, 22, 0, (char *)&neworder.total, /* Total amount */
-1, -1, -1, -1, -1, (char *)NULL
};
```

```
struct fld payment_fld[] = {
INT, 2, 51, 4, 1, (char *)&payment.d_id, /* d-id */
INT, 4, 10, 9, 1, (char *)&payment.c_id, /* c-id */
INT, 4, 32, 9, 1, (char *)&payment.c_w_id,
INT, 2, 53, 9, 1, (char *)&payment.c_d_id,
STRING, 17, 28, 10, 1, payment.c_last, /* customer name */
AMOUNT, 8, 22, 15, 1, (char *)&payment.h_amount,
STRING, 20, 0, 2, 0, payment.h_date,
INT, 4, 11, 4, 0, (char *)&payment.w_id, /* w-id */
STRING, 21, 0, 5, 0, payment.w_street_1,
STRING, 21, 0, 6, 0, payment.w_street_2,
STRING, 21, 0, 7, 0, payment.w_city,
STRING, 3, 21, 7, 0, payment.w_state,
STRING, 11, 24, 7, 0, payment.w_zip,
STRING, 21, 41, 5, 0, payment.d_street_1,
STRING, 21, 41, 6, 0, payment.d_street_2,
STRING, 21, 41, 7, 0, payment.d_city,
STRING, 3, 62, 7, 0, payment.d_state,
STRING, 11, 65, 7, 0, payment.d_zip,
STRING, 17, 8, 10, 0, payment.c_first,
STRING, 3, 25, 10, 0, payment.c_middle,
STRING, 21, 8, 11, 0, payment.c_street_1,
STRING, 21, 8, 12, 0, payment.c_street_2,
STRING, 21, 8, 13, 0, payment.c_city,
STRING, 3, 29, 13, 0, payment.c_state,
STRING, 11, 32, 13, 0, payment.c_zip,
STRING, 11, 57, 10, 0, payment.c_since,
STRING, 3, 57, 11, 0, payment.c_credit,
AMOUNT, 15, 54, 15, 0, (char *)&payment.c_balance,
AMOUNT, 14, 16, 16, 0, (char *)&payment.c_credit_lim,
AMOUNT, 5, 57, 12, 0, (char *)&payment.c_discount,
STRING, 51, 11, 18, 0, payment.c_data_1,
STRING, 51, 11, 19, 0, payment.c_data_2,
STRING, 51, 11, 20, 0, payment.c_data_3,
STRING, 51, 11, 21, 0, payment.c_data_4,
STRING, 20, 57, 13, 0, payment.c_phone,
-1, -1, -1, -1, -1, (char *)NULL
};


struct fld delivery_fld[] = {
INT, 2, 16, 4, 1, (char *)&delivery.o_carrier_id,
INT, 4, 11, 2, 0, (char *)&delivery.w_id,
STRING, 26, 18, 6, 0, delivery.status,
-1, -1, -1, -1, -1, (char *)NULL
};


struct fld stocklevel_fld[] = {
INT, 2, 23, 4, 1, (char *)&stocklevel.threshold,
INT, 4, 11, 2, 0, (char *)&stocklevel.w_id,
INT, 2, 28, 2, 0, (char *)&stocklevel.d_id,
INT, 3, 11, 6, 0, (char *)&stocklevel.low_stock,
-1, -1, -1, -1, -1, (char *)NULL
};


struct fld ordstat_fld[] = {
INT, 2, 28, 2, 1, (char *)&ordstat.d_id,
INT, 4, 10, 3, 1, (char *)&ordstat.c_id,
STRING, 17, 43, 3, 1, ordstat.c_last,
INT, 4, 11, 2, 0, (char *)&ordstat.w_id,
STRING, 17, 23, 3, 0, ordstat.c_first, /* customer name */
STRING, 3, 40, 3, 0, ordstat.c_middle,
AMOUNT, 10, 14, 4, 0, (char *)&ordstat.c_balance,
INT, 8, 14, 6, 0, (char *)&ordstat.o_id, /* order-id */
STRING, 20, 37, 6, 0, ordstat.o_entry_d, /* entry date */
INT, 2, 75, 6, 0, (char *)&ordstat.o_carrier_id,
-1, -1, -1, -1, -1, (char *)NULL
};


/* Sent on termination to reset the terminal to a clean state */
unsigned char cleanup_template[] = {
27, 42, 27, 104, 27, 46, 50, 27, 45, 48, 55, 27, 67, 27, 68, 70, 27, 71, 48,
0 };

/* The menu screen.  Presents a single character input field */

unsigned char menu_screen[] = {
27,'*',     /* Clear to nulls */
27,',',     /* Clear to protected */
27,0x22,    /* Set input type to numeric */
27,')','N','e','w','-','O','r','d','e','r','(','n',')',
' ',' ','P','a','y','m','e','n','t','(','p',')',
' ',' ','O','r','d','e','r','-','S','t','a','t','u','s','(','o',')',
' ',' ','D','e','l','i','v','e','r','y','(','d',')',
' ',' ','S','t','o','c','k','-','L','e','v','e','l','(','s',')',
' ',' ','E','x','i','t','(','e',')',
27,'(',' ', 27,')',
27,'&',
27,'N',
27,'(',
27,'0','1','S',
27,'B', 0 };


/* The output screens  These are basically a set of sprintf strings for the */
/* various put functions.  The comment beside each line indicates the record
*/
/* entry loaded into each field                                              */

unsigned char payment_output_screen[] = {
27,';',27,'+',0x1a,27,0x27,27,'O',27,'G','0',
27,'-','0','!',' ',27,')','%','s',              /* payment.h_date */
27,'-','0','#','Q',27,')','%','0','2','d',      /* payment.d_id */
0x1f,27,')','%','s',                            /* payment.w_street_1 */
27,']','H',27,')','%','s',                      /* payment.d_street_1 */
0x1f,27,')','%','s',                            /* payment.w_street_2 */
27,']','H',27,')','%','s',                      /* payment.d_street_2 */
0x1f,27,')','%','s',                            /* payment.w_city */
27,']','4',27,')','%','s',                      /* payment.w_state */
27,']','7',27,')','%','s',                      /* payment.w_zip */
27,']','H',27,')','%','s',                      /* payment.d_city */
27,']',']',27,')','%','s',                      /* payment.d_state */
27,']','`',27,')','%','s',                      /* payment.d_zip */
0x1f,0x1f,27,']','+',27,')','%','0','4','d',    /* payment.c_id */
27,']','A',27,')','%','0','4','d',              /* payment.c_w_id */
27,']','U',27,')','%','0','2','d',              /* payment.c_d_id */
0x1f,27,']','&',27,')','%','1','7','s',         /* payment.c_first */
27,']','8',27,')','%','3','s',                  /* payment.c_middle */
27,']',';',27,')','%','1','7','s',              /* payment.c_last */
27,']','X',27,')','%','s',                      /* payment.c_since */
0x1f,27,']',0x27,27,')','%','s',                /* payment.c_street_1 */
```

```
27,’]’,’X’,27,’)’,’%’,’3’,’s’,                /* payment.c_credit */
0x1f,27,’]’,0x27,27,’)’,’%’,’s’,               /* payment.c_street_2 */
27,’]’,’X’,27,’)’,’%’,’0’,’5’,’.’,’2’,’f’,    /* payment.c_discount */
0x1f,27,’]’,0x27,27,’)’,’%’,’s’,               /* payment.c_city */
27,’]’,’<‘,27,’)’,’%’,’s’,                     /* payment.c_state */
27,’]’,’?’,27,’)’,’%’,’s’,                     /* payment.c_zip */
27,’]’,’X’,27,’)’,’%’,’s’,                     /* payment.c_phone */
0x1f,0x1f,27,’]’,’5’,27,’)’,’$’,’%’,’0’,’7’,’.’,’2’,’f’,  /* payment.h_amount */
27,’]’,’U’,27,’)’,’$’,’%’,’0’,’1’,’4’,’.’,’2’,’f’,       /* payment.c_balance */
0x1f,27,’]’,’ ’,27,’)’,’$’,’%’,’0’,’1’,’3’,’.’,’2’,’f’,/* payment.c_credit_lim*/
0x1f,0x1f,27,’]’,’*’,27,’)’,’%’,’s’,           /* payment.c_data_1 */
0x1f,27,’]’,’*’,27,’)’,’%’,’s’,                /* payment.c_data_2 */
0x1f,27,’]’,’*’,27,’)’,’%’,’s’,                /* payment.c_data_3 */
0x1f,27,’]’,’*’,27,’)’,’%’,’s’,                /* payment.c_data_4 */
0 };


/* This sequence is output at the end of each screen so the slave can
detect */
/* end of output                                                    */

unsigned char term_sequence[] = {
27,’-’,’0’,’6’,’j’,27,’)’,’*’,’*’,’(‘,’(‘,0};      /* TERM_SEQ  */


unsigned char order_line_output_screen[] = {
0x1f,27,’]’,’!’,’%’,’0’,’4’,’d’,               /* ol_supply_w_id */
27,’]’,’(‘,’%’,’0’,’6’,’d’,                    /* ol_i_id */
27,’]’,’1’,’%’,’0’,’2’,’d’,                    /* ol_quantity */
27,’]’,’6’,’%’,’0’,’9’,’.’,’0’,’2’,’f’,        /* ol_amount */
27,’]’,’D’,’%’,’s’,                            /* ol_delivery_d */
0};


unsigned char ordstat_header_output_screen[] = {
27,’;’,27,’+’,0x1a,27,0x27,27,’O’,27,’G’,’0’,
27,’-’,’0’,’!’,’;’,27,’)’,’%’,’0’,’2’,’d’,          /* ordstat.d_id */
0x1f,27,’]’,’*’,27,’)’,’%’,’0’,’4’,’d’,             /* ordstat.c_id */
27,’]’,’6’,27,’)’,’%’,’1’,’7’,’s’,                  /* ordstat.c_first */
27,’]’,’G’,27,’)’,’%’,’3’,’s’,                      /* ordstat.c_middle */
27,’]’,’J’,27,’)’,’%’,’1’,’7’,’s’,                  /* ordstat.c_last */
0x1f,27,’]’,’-’,27,’)’,’$’,’%’,’0’,’9’,’.’,’0’,’2’,’f’,  /* ordstat.c_balance */
0x1f,0x1f,27,’]’,’-’,27,’)’,’%’,’0’,’8’,’d’,        /* ordstat.o_id */
27,’]’,’E’,27,’)’,’%’,’s’,                          /* ordstat.o_entry_d */
27,’]’,’j’,27,’)’,’%’,’0’,’2’,’d’,                  /* ordstat.o_carrier_id */
27,’-’,’0’,’&’,’&’,0x1f,       /* one before lines because cr at start of ol */
0 };


unsigned char stock_output_screen[] = {
27,’;’,27,’+’,0x1a,27,0x27,27,’O’,27,’G’,’0’,
27,’-’,’0’,’#’,’7’,27,’)’,’%’,’0’,’2’,’d’,          /* stocklevel.threshold */
0x1f,0x1f,27,’]’,’*’,27,’)’,’%’,’0’,’3’,’d’,        /* stocklevel.low_stock */
0 };


unsigned char delivery_output_screen[] = {
27,’;’,27,’+’,0x1a,27,0x27,27,’O’,27,’G’,’0’,
27,’-’,’0’,’#’,’0’,27,’)’,’%’,’0’,’2’,’d’,          /* delivery.o_carrier_id */
0x1f,0x1f,27,’]’,’2’,27,’)’,’%’,’s’,                /* delivery.status */
0 };

unsigned char neworder_header_output_screen[] = {
27,’;’,27,’+’,0x1a,27,0x27,27,’O’,27,’G’,’0’,
27,’-’,’0’,’!’,’<‘,27,’)’,’%’,’0’,’2’,’d’,          /* neworder.d_id */
27,’]’,’[‘,27,’)’,’%’,’s’,                          /* neworder.o_entry_d */
27,’-’,’0’,0x22,’*’,27,’)’,’%’,’0’,’4’,’d’,         /* neworder.c_id */
27,’]’,’7’,27,’)’,’%’,’s’,                          /* neworder.c_last */
27,’]’,’R’,27,’)’,’%’,’s’,                          /* neworder.c_credit */
27,’]’,’^’,27,’)’,’%’,’0’,’5’,’.’,’0’,’2’,’f’,     /* neworder.c_discount */
27,’-’,’0’,’#’,’-’,27,’)’,’%’,’0’,’8’,’d’,         /* neworder.o_id */
27,’]’,’H’,27,’)’,’%’,’0’,’2’,’d’,                  /* neworder.o_ol_cnt */
27,’]’,’Y’,27,’)’,’%’,’0’,’5’,’.’,’0’,’2’,’f’,     /* neworder.w_tax */
27,’]’,’h’,27,’)’,’%’,’0’,’5’,’.’,’0’,’2’,’f’,     /* neworder.d_tax */
27,’-’,’0’,’5’,’ ‘,’E’,’x’,’e’,’c’,’u’,’t’,’i’,’o’,’n’,’
‘,’S’,’t’,’a’,’t’,’u’,’s’,’:’,27,’)’,’%’,’s’,                /* neworder.status */
27,’]’,’d’,27,’)’,’$’,’%’,’0’,’8’,’.’,’0’,’2’,’f’,  /* neworder.total */
27,’-’,’0’,’%’,’%’,0x1f,       /* one before lines because cr at start of ol */
0 };


/* For each unused line of neworder output send this sequence */
unsigned char neworder_blank_output_screen[] = { 0x1f,27,’T’,27,’)’,0};


unsigned char neworder_line_output_screen[] = {
0x1f,27,’]’,’!’,’%’,’0’,’4’,’d’,                    /* ol_supply_w_id */
27,’]’,’(‘,’%’,’0’,’6’,’d’,                         /* ol_i_id */
27,’]’,’1’,’%’,’s’,                                 /* i_name */
27,’]’,’K’,’%’,’0’,’2’,’d’,                         /* ol_quantity */
27,’]’,’Q’,’%’,’0’,’3’,’d’,                         /* s_quantity */
27,’]’,’X’,’%’,’s’,                                 /* brand */
27,’]’,’\ \’,’$’,’%’,’0’,’6’,’.’,’0’,’2’,’f’,      /* i_price */
27,’]’,’e’,’$’,’%’,’0’,’7’,’.’,’0’,’2’,’f’,        /* ol_amount */
0};


/* The input screens.  The input fields are delimited by 27,( sp .. sp 27,) */
/* This is set unprotect/set protect.  The user will only be able to enter  */
/* data on the spaces.  The rest is just the text output                    */

unsigned char payment_input_screen[] = {
27,’*’,     /* Clear to nulls */
27,’,’,     /* Clear to protected */
27,0x22,    /* Input to numeric */
27,’)’,27,’G’,’8’,      /* Set underline mode */
27,’-’,’0’,’ ‘,’B’,’P’,’a’,’y’,’m’,’e’,’n’,’t’,
0x1f,’D’,’a’,’t’,’e’,’:’,
0x1f,0x1f,’W’,’a’,’r’,’e’,’h’,’o’,’u’,’s’,’e’,’:’,’ ‘,’%’,’0’,’4’,’d’,
27,’]’,’G’,’D’,’i’,’s’,’t’,’r’,’i’,’c’,’t’,’:’,’ ‘,27,’(‘,’ ‘,’ ‘,27,’)’,
27,’-’,’0’,’(‘,’ ‘,’C’,’u’,’s’,’t’,’o’,’m’,’e’,’r’,’:’,’ ‘,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,
‘C’,’u’,’s’,’t’,’-’,’W’,’a’,’r’,’e’,’h’,’o’,’u’,’s’,’e’,’:’,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,
‘C’,’u’,’s’,’t’,’-’,’D’,’i’,’s’,’t’,’r’,’i’,’c’,’t’,’:’,’ ‘,27,’(‘,’ ‘,’ ‘,27,’)’),
0x1f,’N’,’a’,’m’,’e’,’:’,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’),
27,’-’,’0’,’)’,’P’,’S’,’i’,’n’,’c’,’e’,’:’,’ ‘,
27,’-’,’0’,’*’,’P’,’C’,’r’,’e’,’d’,’i’,’t’,’:’,’ ‘,
27,’-’,’0’,’+’,’P’,’%’,’D’,’i’,’s’,’c’,’:’,’ ‘,
27,’-’,’0’,0x2c,’P’,’P’,’h’,’o’,’n’,’e’,’:’,
0x1f,0x1f,’A’,’m’,’o’,’u’,’n’,’t’,’ ‘,’P’,’a’,’i’,’d’,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’
‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’),
27,’-’,’0’,0x2e,’C’,’N’,’e’,’w’,’ ‘,’C’,’u’,’s’,’t’,’-’,’B’,’a’,’l’,’a’,’n’,’c’,’e’,’:’,’ ‘,
0x1f,’C’,’r’,’e’,’d’,’i’,’t’,’ ‘,’L’,’i’,’m’,’i’,’t’,’:’,’ ‘,
```

```
0x1f,0x1f,’C’,’u’,’s’,’t’,’-’,’D’,’a’,’t’,’a’,’:’,’ ‘,
27,’&’,              /* Protect mode */
27,’N’,              /* Page edit mode */
27,’(‘,             /* unprotect */
27,’0’,’1’,’S’,      /* Program send key */
27,’B’, 0 };         /* Block Mode */


unsigned char orderstat_input_screen[] = {
27,’*’,     /* Clear to nulls */
27,’,’,     /* Clear to protected */
27,0x22,    /* Input to numeric */
27,’)’,27,’G’,’8’,      /* Set underline mode */
27,’-’,’0’,’ ‘,’B’,’O’,’r’,’d’,’e’,’r’,’-’,’S’,’t’,’a’,’t’,’u’,’s’,
0x1f,’W’,’a’,’r’,’e’,’h’,’o’,’u’,’s’,’e’,’:’,’ ‘,’%’,’0’,’4’,’d’,’ ‘,’ ‘,
‘D’,’i’,’s’,’t’,’r’,’i’,’c’,’t’,’:’,’ ‘,27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,’C’,’u’,’s’,’t’,’o’,’m’,’e’,’r’,’:’,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,
‘ ‘,’ ‘, ‘N’,’a’,’m’,’e’,’:’,’ ‘,
27,’-’,’0’,0x22,’J’,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,
0x1f,’C’,’u’,’s’,’t’,’-’,’B’,’a’,’l’,’a’,’n’,’c’,’e’,’:’,’ ‘,
0x1f,0x1f,’O’,’r’,’d’,’e’,’r’,’-’,’N’,’u’,’m’,’b’,’e’,’r’,’:’,’ ‘,
27,’-’,’0’,0x25,0x25,’8’,’E’,’n’,’t’,’r’,’y’,’-’,’D’,’a’,’t’,’e’,’:’,
27,’-’,’0’,0x25,0x25,’Z’,’C’,’a’,’r’,’r’,’i’,’e’,’r’,’-’,’N’,’u’,’m’,’b’,’e’,’r’,’:’,’ ‘,
0x1f,’S’,’u’,’p’,’p’,’_’,’W’,’ ‘,’ ‘,’I’,’t’,’e’,’m’,’_’,’I’,’d’,’ ‘,’ ‘,
‘Q’,’t’,’y’,’ ‘,’ ‘,’A’,’m’,’o’,’u’,’n’,’t’,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’D’,’e’,’l’,’i’,’v’,’e’,’r’,’y’,’-
’,’D’,’a’,’t’,’e’,
27,’&’,              /* Protect mode */
27,’N’,              /* Page edit mode */
27,’(‘,             /* unprotect */
27,’0’,’1’,’S’,      /* Program send key */
27,’B’, 0 };         /* Block Mode */


unsigned char delivery_input_screen[] = {
27,’*’,     /* Clear to nulls */
27,’,’,     /* Clear to protected */
27,0x22,    /* Input to numeric */
27,’)’,27,’G’,’8’,      /* Set underline mode */
27,’-’,’0’,’ ‘,’B’,’D’,’e’,’l’,’i’,’v’,’e’,’r’,’y’,
0x1f,’W’,’a’,’r’,’e’,’h’,’o’,’u’,’s’,’e’,’:’,’ ‘,’%’,’0’,’4’,’d’,’ ‘,’ ‘,’ ‘,
0x1f,0x1f,’C’,’a’,’r’,’r’,’i’,’e’,’r’,’ ‘,’N’,’u’,’m’,’b’,’e’,’r’,’:’,’ ‘,27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,0x1f,’E’,’x’,’e’,’c’,’u’,’t’,’i’,’o’,’n’,’ ‘,’S’,’t’,’a’,’t’,’u’,’s’,’:’,’ ‘,
27,’&’,              /* Protect mode */
27,’N’,              /* Page edit mode */
27,’(‘,             /* unprotect */
27,’0’,’1’,’S’,      /* Program send key */
27,’B’, 0 };         /* Block Mode */


unsigned char stock_input_screen[] = {
27,’*’,     /* Clear to nulls */
27,’,’,     /* Clear to protected */
27,0x22,    /* Input to numeric */
27,’)’,27,’G’,’8’,      /* Set underline mode */
27,’-’,’0’,’ ‘,’B’,’S’,’t’,’o’,’c’,’k’,’-’,’l’,’e’,’v’,’e’,’l’,
0x1f,’W’,’a’,’r’,’e’,’h’,’o’,’u’,’s’,’e’,’:’,’ ‘,’%’,’0’,’4’,’d’,’ ‘,’ ‘,’ ‘,
‘ ‘,’ ‘,’ ‘,’D’,’i’,’s’,’t’,’r’,’i’,’c’,’t’,’:’,’ ‘,’%’,’0’,’2’,’d’,’ ‘,’ ‘,’ ‘,
0x1f,0x1f,’S’,’t’,’o’,’c’,’k’,’ ‘,’L’,’e’,’v’,’e’,’l’,
‘ ‘,’T’,’h’,’r’,’e’,’s’,’h’,’o’,’l’,’d’,’:’,’ ‘,
27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,0x1f,’L’,’o’,’w’,’ ‘,’S’,’t’,’o’,’c’,’k’,’:’,’ ‘,
27,’&’,              /* Protect mode */
27,’N’,              /* Page edit mode */
27,’(‘,             /* unprotect */
27,’0’,’1’,’S’,      /* Program send key */
27,’B’, 0 };         /* Block Mode */


unsigned char neworder_input_screen[] = {
27,’*’,     /* Clear to nulls */
27,’,’,     /* Clear to protected */
27,0x22,    /* Input to numeric */
27,’)’,27,’G’,’8’,      /* Set underline mode */
27,’-’,’0’,’ ‘,’B’,’N’,’e’,’w’,’ ‘,’O’,’r’,’d’,’e’,’r’,
0x1f,’W’,’a’,’r’,’e’,’h’,’o’,’u’,’s’,’e’,’:’,’ ‘,’%’,’0’,’4’,’d’,’ ‘,’ ‘,’ ‘,
‘D’,’i’,’s’,’t’,’r’,’i’,’c’,’t’,’:’,’ ‘,27,’(‘,’ ‘,’ ‘,27,’)’,
27,’-’,’0’,’!’,’U’,’D’,’a’,’t’,’e’,’:’,
0x1f,’C’,’u’,’s’,’t’,’o’,’m’,’e’,’r’,’:’,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,
‘ ‘,’ ‘,’ ‘,’N’,’a’,’m’,’e’,’:’,’ ‘,
27,’-’,’0’,0x22,’J’,’C’,’r’,’e’,’d’,’i’,’t’,’:’,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,
‘%’,’D’,’i’,’s’,’c’,’:’,’ ‘,
0x1f,’O’,’r’,’d’,’e’,’r’,’ ‘,’N’,’u’,’m’,’b’,’e’,’r’,’:’,
‘ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,
‘N’,’u’,’m’,’b’,’e’,’r’,’ ‘,’o’,’f’,’ ‘,’L’,’i’,’n’,’e’,’s’,’:’,
‘ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,
‘W’,’_’,’t’,’a’,’x’,’:’,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,
‘D’,’_’,’t’,’a’,’x’,’:’,
0x1f,0x1f,’S’,’u’,’p’,’p’,’_’,’W’,
‘ ‘,’ ‘,’I’,’t’,’e’,’m’,’_’,’I’,’d’,’ ‘,’ ‘,
‘I’,’t’,’e’,’m’,’ ‘,’N’,’a’,’m’,’e’,
27,’]’,’K’,’Q’,’t’,’y’,’ ‘,’ ‘,’S’,’t’,’o’,’c’,’k’,’ ‘,’ ‘,’B’,’/’,’G’,
‘ ‘,’ ‘,’P’,’r’,’i’,’c’,’e’,’ ‘,’ ‘,’ ‘,’ ‘,’A’,’m’,’o’,’u’,’n’,’t’,
0x1f,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,
27,’]’,’K’,27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,
27,’]’,’K’,27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,
27,’]’,’K’,27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,
27,’]’,’K’,27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,
27,’]’,’K’,27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,
27,’]’,’K’,27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,
27,’]’,’K’,27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,
27,’]’,’K’,27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,
27,’]’,’K’,27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,
27,’]’,’K’,27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,
27,’]’,’K’,27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,
27,’]’,’K’,27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,
27,’]’,’K’,27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,
27,’]’,’K’,27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,’ ‘,27,’(‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,’ ‘,27,’)’,’ ‘,’ ‘,
27,’]’,’K’,27,’(‘,’ ‘,’ ‘,27,’)’,
0x1f,’E’,’x’,’e’,’c’,’u’,’t’,’i’,’o’,’n’,’ ‘,’S’,’t’,’a’,’t’,’u’,’s’,’:’,
```

```
27,'-','0','5','\\','T','o','t','a','l',':',
27,'&',             /* Protect mode */
27,'N',             /* Page edit mode */
27,'(',            /* unprotect */
27,'0','1','S',     /* Program send key */
27,'B', 0 };        /* Block Mode */

unsigned char error_string[] = {
27,0x27,27,')',27,'-','0','6',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',27,'-','0','6',' '
' ','%','s',27,'p','@',
27,'&',             /* Protect mode */
27,'N',             /* Page edit mode */
27,'(',             /* unprotect */
27,'0','1','S',     /* Program send key */
27,'B',  0 };       /* Block Mode */


unsigned char clear_exec_string[] = {
27,0x27,27,')',27,'-','0','5',
' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
27,'p','@',
27,'&',             /* Protect mode */
27,'N',             /* Page edit mode */
27,'(',             /* unprotect */
27,'0','1','S',     /* Program send key */
27,'B',0};

unsigned char clear_error_string[] = {
27,0x27,27,')',27,'-','0','6',
' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',
27,'p','@',
27,'&',             /* Protect mode */
27,'N',             /* Page edit mode */
27,'(',             /* unprotect */
27,'0','1','S',     /* Program send key */
27,'B',0};



/*
 * Function: put_menu
 * Displays the menu and gets user's selection
 */
put_menu()
{
        char choice;
        int  result = 0;
        int ret_code;

        ret_code = write(1,menu_screen,strlen(menu_screen));
        while( ! result) {
                get_input(0);
                get_char(&choice ,0);/* Get a single character response */
                switch(choice) {
                case 'n':
                case 'N':
                        result = (NEWORDER);
                        break;
                case 'p':
                case 'P':
                        result = (PAYMENT);
                        break;
                case 'o':
                case 'O':
                        result = (ORDSTAT);
                        break;
                case 'd':
                case 'D':
                        result = (DELIVERY);
                        break;
                case 's':
                case 'S':
                        result = (STOCKLEVEL);
                        break;
                case 'e':
                case 'E':
                        result = (EXIT);
                        break;
                default:
                        print_error_on_field();
                }
        }
        return(result);
}


/*
 * Function: clr_fields
 * This function clears the field values of all the
 * fields in the given fld structure
 */
clr_fields(fldp)
struct fld *fldp;
{
        struct fld *nextp = fldp;

        while ( nextp->type != -1 ) {
                if (nextp->input_allowed) {
                switch (nextp->type) {
                case INT:
                        *((int *)(nextp->value)) = 0;
                        break;
                case AMOUNT:
                        *((double *)(nextp->value)) = 0.0;
                        break;
                case STRING:
                        *(nextp->value) = 0;
                        break;
                }
                }
                nextp++;
        }
}


/*
 * Function: get_neworder
 * Display neworder form and get user input
 * Return # of items entered
 */
```

```
get_neworder()
{
        int num_lines, cur_y, i = 0, term = 0;/* Terminating character */
        struct fld *nextp;
        struct no_itm_struct *beginp, *curp;
        int allnulls;
        int ret_code;
        int cur_field,status;

        cur_field = 0;

        sprintf(display_buffer,(char *)neworder_input_screen,w_id);
        ret_code = write(1,display_buffer,strlen(display_buffer));
        /* Init. all fields to NULL */
        clr_fields(neworder_fld);
        for (curp = &neworder.n_items[0];
                        curp < &neworder.n_items[15]; curp++) {
                curp->ol_supply_w_id = curp->ol_i_id = curp->ol_quantity = 0;
        }

error_input:
        get_input(cur_field);
        cur_field = 0;
        if(get_integer(&neworder.d_id, 2, cur_field) != 0)
                goto error_input;
        cur_field++;
        if(get_integer(&neworder.c_id, 4, cur_field) != 0)
                goto error_input;
        cur_field++;
        beginp = &neworder.n_items[0];
        curp = beginp;
        for (i=0 ; i < 15; i++) {
                if(get_integer(&curp->ol_supply_w_id, 4, cur_field) != 0)
                        goto error_input;
                cur_field++;
                if(get_integer(&curp->ol_i_id, 6, cur_field) != 0)
                        goto error_input;
                cur_field++;
                if(get_integer(&curp->ol_quantity, 2, cur_field) != 0)
                        goto error_input;
                cur_field++;
                curp++;
        }


        if (neworder.d_id == 0 || neworder.c_id == 0) {
                        strcpy(errmsg, "All required fields have not been
entered");
                        put_err(errmsg);
                        goto error_input;
        }
        /* Check if all fields entered for each line item */
        allnulls = 0;
        num_lines = 15;
        for (i = 0; i < 15; i++) {
                if (neworder.n_items[i].ol_supply_w_id == 0) {
                        if (neworder.n_items[i].ol_i_id == 0 &&
                                neworder.n_items[i].ol_quantity == 0) {
                                if ( !allnulls) {/* First null line */
                                        num_lines = i;/* Number of line items */
                                        allnulls = 1;
                                }
                                continue;
                        }
                }
                else if (allnulls == 1)
                        goto error;
                else if (neworder.n_items[i].ol_i_id != 0 &&
                                neworder.n_items[i].ol_quantity != 0)
                        continue;
error:
        strcpy(errmsg, "Invalid item-line entered                ");
        put_err(errmsg);
        goto error_input;
        }

        if (err_flag) {
                err_flag = 0;
                clear_err();
        }
        return(num_lines);
}

/*
 * Function: get_payment
 * Display payment form and get user input
 */
get_payment()
{
        int i, term = 0;/* Terminating character */
        struct fld *nextp;
        int ret_code;
        int cur_field,status;

        cur_field = 0;

        sprintf(display_buffer,(char *)payment_input_screen,w_id);
        ret_code = write(1,display_buffer,strlen(display_buffer));

        /* Init all fields to NULL */
        clr_fields(payment_fld);
error_input:
        get_input(cur_field);
        cur_field = 0;

        if(get_integer(&payment.d_id, 2, cur_field) != 0)
                goto error_input;
        cur_field++;
        if(get_integer(&payment.c_id, 4, cur_field) != 0)
                goto error_input;
        cur_field++;
        if(get_integer(&payment.c_w_id, 4, cur_field) != 0)
                goto error_input;
        cur_field++;
        if(get_integer(&payment.c_d_id, 2, cur_field) != 0)
                goto error_input;
        cur_field++;
        if(get_string(payment.c_last, 17, cur_field) != 0)
                goto error_input;
        cur_field++;
        if(get_amount(&payment.h_amount, 8, cur_field) != 0)
                goto error_input;
        cur_field++;
                        /* Check if reqd. fields have been entered */
        if (payment.d_id == 0 || payment.c_d_id == 0 ||
                payment.c_w_id == 0 || payment.h_amount == 0.0)
```

```
                strcpy(errmsg, "All required fields have not been entered
");
        else if (payment.c_id == 0 && payment.c_last[0] == '\0')
{
                strcpy(errmsg, "You must enter either the Customer-id or
Name");
}
        else if (payment.h_amount > 9999.99)
                strcpy(errmsg, "Invalid amount entered                  ");
        else {
                if (err_flag) {
                        err_flag = 0;
                clear_err();
                }
                return(0);
        }
        put_err(errmsg);
        goto error_input;
}


/*
 * Function: get_ordstat
 * Display ordstat form and get user input
 */


get_ordstat()
{
        int i, term = 0;/* Terminating character */
        struct fld *nextp;
        int ret_code;
        int cur_field,status;

        cur_field = 0;

        sprintf(display_buffer,(char *)orderstat_input_screen,w_id);
        ret_code = write(1,display_buffer,strlen(display_buffer));

        /* Init all fields to NULL */
        clr_fields(ordstat_fld);
error_input:
        get_input(cur_field);
        cur_field = 0;

        if(get_integer(&ordstat.d_id, 2, cur_field) != 0)
                goto error_input;
        cur_field++;
        if(get_integer(&ordstat.c_id, 4, cur_field) != 0)
                goto error_input;
        cur_field++;
        if(get_string(ordstat.c_last, 17, cur_field) != 0)
                goto error_input;
        cur_field++;

        /* Check if reqd. fields have been entered */
        if (ordstat.d_id == 0 ) {
                strcpy(errmsg, "All required fields have not been entered
");
        }
        else if (ordstat.c_id == 0 && ordstat.c_last[0] == '\0') {
                strcpy(errmsg, "You must enter either the Customer-id or
Name");
        }
        else {
                if (err_flag) {
                        err_flag = 0;
                        clear_err();
                }
                return(0);
        }
        put_err(errmsg);
        goto error_input;
}



/*
 * Function: get_stocklevel
 * Display stocklevel form and read user input
 */
get_stocklevel()
{
        struct fld *cp = &stocklevel_fld[0];/* threshold */
        int term;
        int ret_code;
        int cur_field,status;

        cur_field = 0;

        sprintf(display_buffer,(char *)stock_input_screen,w_id,d_id);
        ret_code = write(1,display_buffer,strlen(display_buffer));

        /* Init all fields to NULL */
        clr_fields(stocklevel_fld);
error_input:
        get_input(cur_field);
        cur_field = 0;


        /* Get threshold */
        if(get_integer(&stocklevel.threshold, 2, cur_field) !=0)
                goto error_input;

        if (*((int *)(cp->value)) == 0) {
        print_error_on_field();
                goto error_input;
        }
        return(0);
}


/*
 * Function: get_delivery
 * Display delivery form and read user input
 */
get_delivery()
{
        struct fld *cp = &delivery_fld[0];
        int term;
        int ret_code;
        int cur_field,status;

        cur_field = 0;

        sprintf(display_buffer,(char *)delivery_input_screen,w_id);
```

```
        ret_code = write(1,display_buffer,strlen(display_buffer));

        /* Init all fields to NULL */
        clr_fields(delivery_fld);
error_input:
        get_input(cur_field);
        cur_field = 0;

        if(get_integer(cp->value, cp->len, cur_field) !=0)
                goto error_input;
        if (*((int *)(cp->value)) == 0) {
                print_error_on_field();
                goto error_input;
        }
        return(0);
}


/* DISPLAY FUNCTIONS */
/* Display error message on last line of screen */
put_err(errmsg)
char *errmsg;
{
        sprintf(display_buffer,(char *)error_string,errmsg);
        write(1,display_buffer,strlen(display_buffer));
        err_flag = 1;
}

/* DISPLAY FUNCTIONS */
/* Display error message on last line of screen */
clear_err()
{
        sprintf(display_buffer,(char *)clear_error_string);
        write(1,display_buffer,strlen(display_buffer));
}



put_neworder(count)
int count;/* Count of items */
{
        struct fld *nextp = neworder_fld;
        struct no_itm_struct *valp = neworder.n_items;
        int i, cur_y;
        int ret_code;


        sprintf(display_buffer,(char *)clear_exec_string);
        write(1,display_buffer,strlen(display_buffer));
        sprintf(display_buffer,(char *)neworder_header_output_screen,
                        neworder.d_id,          /* d-id */
                        neworder.o_entry_d,     /* entry date */
                        neworder.c_id,          /* c-id */
                        neworder.c_last,
                        neworder.c_credit,      /* customer credit */
                        neworder.c_discount,    /* c-discount */
                        neworder.o_id,          /* order-id */
                        neworder.o_ol_cnt,      /* order-count */
                        neworder.w_tax,         /* w_tax */
                        neworder.d_tax,         /* d_tax */
                        neworder.status,        /* Execution status */
                        neworder.total);        /* Total amount */
        ret_code = write(1,display_buffer,strlen(display_buffer));

        for (i = 0; i < count; i++, valp++) {
        sprintf(display_buffer,(char *)neworder_line_output_screen,
                valp->ol_supply_w_id,
                valp->ol_i_id,
                valp->i_name,
                valp->ol_quantity,
                valp->s_quantity,
                valp->brand,
                valp->i_price,
                valp->ol_amount);
        ret_code = write(1,display_buffer,strlen(display_buffer));
        }
        for (i = count; i < 15; i++) {
        sprintf(display_buffer,(char *)neworder_blank_output_screen);
        ret_code = write(1,display_buffer,strlen(display_buffer));
        }
        sprintf(display_buffer,(char *)term_sequence);
        ret_code = write(1,display_buffer,strlen(display_buffer));
        wait_for_send();


}


put_payment()
{
int ret_code;

        sprintf(display_buffer,(char *)payment_output_screen,
                payment.h_date,
                payment.d_id,           /* d-id */
                payment.w_street_1,
                payment.d_street_1,
                payment.w_street_2,
                payment.d_street_2,
                payment.w_city,
                payment.w_state,
                payment.w_zip,
                payment.d_city,
                payment.d_state,
                payment.d_zip,
                payment.c_id, /* c-id */
                payment.c_w_id,
                payment.c_d_id,
                payment.c_first,
                payment.c_middle,
                payment.c_last, /* customer name */
                payment.c_since,
                payment.c_street_1,
                payment.c_credit,
                payment.c_street_2,
                payment.c_discount,
                payment.c_city,
                payment.c_state,
                payment.c_zip,
                payment.c_phone,
                payment.h_amount,
                payment.c_balance,
                payment.c_credit_lim,
                payment.c_data_1,
                payment.c_data_2,
                payment.c_data_3,
```

```
                payment.c_data_4);
        ret_code = write(1,display_buffer,strlen(display_buffer));
        sprintf(display_buffer,(char *)term_sequence);
        ret_code = write(1,display_buffer,strlen(display_buffer));
        wait_for_send();
}


put_ordstat(count)
int count;
{
        int ret_code;
        struct fld *nextp = ordstat_fld;
        struct ord_itm_struct *valp = ordstat.o_items;
        int i, cur_y;


        sprintf(display_buffer,(char *)ordstat_header_output_screen,
                ordstat.d_id,
                ordstat.c_id,
                ordstat.c_first,  /* customer name */
                ordstat.c_middle,
                ordstat.c_last,
                ordstat.c_balance,
                ordstat.o_id,       /* order-id */
                ordstat.o_entry_d,      /* entry date */
                ordstat.o_carrier_id);
        ret_code = write(1,display_buffer,strlen(display_buffer));

        for (i = 0; i < count; i++, valp++) {
        sprintf(display_buffer,(char *)order_line_output_screen,
                valp->ol_supply_w_id,
                valp->ol_i_id,
                valp->ol_quantity,
                valp->ol_amount,
                valp->ol_delivery_d);
        ret_code = write(1,display_buffer,strlen(display_buffer));
        }
        sprintf(display_buffer,(char *)term_sequence);
        ret_code = write(1,display_buffer,strlen(display_buffer));
        wait_for_send();
}


put_stocklevel()
{
        int ret_code;

        sprintf(display_buffer,(char *)stock_output_screen,
                stocklevel.threshold,
                stocklevel.low_stock);
        ret_code = write(1,display_buffer,strlen(display_buffer));
        sprintf(display_buffer,(char *)term_sequence);
        ret_code = write(1,display_buffer,strlen(display_buffer));
        wait_for_send();
}

put_delivery()
{
        int ret_code;


        sprintf(display_buffer,(char *)delivery_output_screen,
                delivery.o_carrier_id,
                delivery.status);
        ret_code = write(1,display_buffer,strlen(display_buffer));
        sprintf(display_buffer,(char *)term_sequence);
        ret_code = write(1,display_buffer,strlen(display_buffer));
        wait_for_send();
}



cleanup(code)
{
        int ret_code;

        ret_code = write(1,cleanup_template,strlen(cleanup_template));
        if (code == TXERRCODE) {
                put_err(TXERRMSG);
                wait_for_send();
                return;
        }
        restore_tx();

        exit(code);
}


/*
 * File: del.ec
 * Delivery client Tuxedo code
 * Author : Shanti S
 * Date   : 8/04/93
 *
 */

#include <stdio.h>
#include <syslog.h>

#include “tpcc_client.h”

/* Tuxedo */
#include “atmi.h”
#include “Uunix.h”

struct req_struct *delp;
extern struct del_struct delivery;
/*
 * Function: init deltat
 * Init. Tuxedo
 */
init_del_tx()
{
        if ((delp = (struct req_struct *)tpalloc(“CARRAY”, NULL, sizeof(struct
req_struct))) == NULL) {
                syslog(LOG_ERR, “del tpalloc failed. tperrno = %d”, tperrno);
                tpterm();
                cleanup(1);
        }
}


int
queue_delivery()
{
```

```
	/* Copy structure to Tuxedo buffer */
	delp->w_id = delivery.w_id;
	delp->o_carrier_id = delivery.o_carrier_id;
	time(&delp->qtime);


	if (tpacall(del_service, (char *)delp, sizeof(struct req_struct),
		TPSIGRSTRT|TPNOREPLY) == -1) {
		syslog(LOG_ERR, "del tpacall failed. tperrno = %d", tperrno);
		if (tperrno == TPEOS)
			syslog(LOG_ERR, "Uunixerr = %d", Uunixerr);
		return(TXERRCODE);
	}
	strcpy(delivery.status, "Delivery has been queued");
	return(0);
}

/*
 * Neworder Tuxedo client
 * Author : Shanti S
 * Date   : 8/03/93
 *
 */
#include <stdio.h>
#include <syslog.h>

#include "tpcc_client.h"

/* Tuxedo includes */
#include "atmi.h"
#include "Uunix.h"

struct no_struct *newop;


/*
 * Function: init neworder
 * This is the first function called by do_init, hence we initialize
 * Tuxedo here using 'tpinit'
 * Init. Tuxedo
 */
init_newo_tx()
{
	if (tpinit(NULL) == -1) {
		syslog(LOG_ERR, "tpinit failed. tperrno = %d", tperrno);
		if (tperrno == TPEOS)
			syslog(LOG_ERR, "Unix system call that failed = %d",
Uunixerr);
		cleanup(1);
	}
	if ((newop = (struct no_struct *)tpalloc("CARRAY", NULL,
sizeof(struct no_struct))) == NULL) {
		syslog(LOG_ERR, "newo tpalloc failed. tperrno = %d",
tperrno);
		tpterm();
		cleanup(1);
	}
}


/*
 * This function is called by cleanup to exit from Tuxedo
 */
restore_tx()
{
	tpterm();
}


neworder_tx(linecnt)
int linecnt;/* Number of lines on order */
{
	long olen;

	*newop = neworder;/* Copy structure to Tuxedo buffer */
	newop->o_ol_cnt = linecnt;

	if (tpcall(newo_service, (char *)newop, sizeof(struct no_struct),
		(char **)&newop, &olen, TPSIGRSTRT|TPNOTIME) == -1) {
		if (tperrno == TPEOS) {
		/* Don't log errors always, as outofspace errors can pile up */
		syslog(LOG_ERR, "newo tpcall failed.tperrno = %d, Uunixerr =
%d",
		tperrno, Uunixerr);
		}
		return(TXERRCODE);
	}
	neworder = *newop;/* Copy results back */
	return(0);
}

/*
 * Order-status client Tuxedo code
 * Author : Shanti S
 * Date   : 8/02/93
 *
 */

#include <stdio.h>
#include <syslog.h>

#include "tpcc_client.h"

/* Tuxedo */
#include "atmi.h"
#include "Uunix.h"

struct ord_struct *ordsp;
/*
 * Function: init ordstat
 * Init. Tuxedo
 */
init_ords_tx()
{
	if ((ordsp = (struct ord_struct *)tpalloc("CARRAY", NULL, sizeof(struct
ord_struct))) == NULL) {
		syslog(LOG_ERR, "ords tpalloc failed. tperrno = %d", tperrno);
		tpterm();
		cleanup(1);
	}
}


int
```

```
ordstat_tx()
{
        long olen;

        *ordsp = ordstat;/* Copy structure to Tuxedo buffer */

        if (tpcall(ords_service, (char *)ordsp, sizeof(struct ord_struct),
                (char **)&ordsp, &olen, TPSIGRSTRT|TPNOTIME) == -1) {
                syslog(LOG_ERR, "ords tpcall failed. tperrno = %d",
tperrno);
                if (tperrno == TPEOS)
                        syslog(LOG_ERR, "Uunixerr = %d", Uunixerr);
                return(TXERRCODE);
        }
        ordstat = *ordsp;
        return(ordsp->item_cnt);
}

/*
 * Payment Tuxedo client
 * Author : Shanti S
 * Date   : 8/03/93
 *
 */
#include <stdio.h>
#include <syslog.h>

#include "tpcc_client.h"

/* Tuxedo includes */
#include "atmi.h"
#include "Uunix.h"

struct pay_struct *paymp;


/*
 * Function: init payment
 * Init. Tuxedo
 */
init_paym_tx()
{
        if ((paymp = (struct pay_struct *)tpalloc("CARRAY", NULL,
sizeof(struct pay_struct))) == NULL) {
                syslog(LOG_ERR, "paym tpalloc failed. tperrno = %d",
tperrno);
                tpterm();
                cleanup(1);
        }
}


payment_tx()
{
        long olen;

        *paymp = payment;/* Copy structure to Tuxedo buffer */

        if (tpcall(paym_service, (char *)paymp, sizeof(struct pay_struct),
                (char **)&paymp, &olen, TPSIGRSTRT|TPNOTIME) == -1)
{
                syslog(LOG_ERR, "paym tpcall failed. tperrno = %d",
tperrno);
                if (tperrno == TPEOS)
                        syslog(LOG_ERR, "Uunixerr = %d", Uunixerr);
                return(TXERRCODE);
        }
        payment = *paymp;/* Copy results back */
        return(0);
}

/*
 * Stocklevel client Tuxedo code
 * Author : Shanti S
 * Date   : 8/04/93
 *
 */

#include <stdio.h>
#include <syslog.h>

#include "tpcc_client.h"

/* Tuxedo */
#include "atmi.h"
#include "Uunix.h"

struct stock_struct *stockp;
/*
 * Function: init stocktat
 * Init. Tuxedo
 */
init_stock_tx()
{
        if ((stockp = (struct stock_struct *)tpalloc("CARRAY", NULL, sizeof(struct
stock_struct))) == NULL) {
                syslog(LOG_ERR, "stock tpalloc failed. tperrno = %d", tperrno);
                tpterm();
                cleanup(1);
        }
}


int
stocklevel_tx()
{
        long olen;

        *stockp = stocklevel;/* Copy structure to Tuxedo buffer */

        if (tpcall(stock_service, (char *)stockp, sizeof(struct stock_struct),
                (char **)&stockp, &olen, TPSIGRSTRT|TPNOTIME) == -1) {
                syslog(LOG_ERR, "stock tpcall failed. tperrno = %d", tperrno);
                if (tperrno == TPEOS)
                        syslog(LOG_ERR, "Uunixerr = %d", Uunixerr);
                return(TXERRCODE);
        }
        stocklevel.low_stock = stockp->low_stock;
        return(0);
}

/*
 * (c) Copyright Sun Microsystems Inc. 1994
 * Written: 15/10/94
 *
 */
```

```
#include <stdio.h>
#include <syslog.h>
#include <errno.h>
#include "tpcc_term.h"

#define INVALID_INTEGER 1

extern double atof();
extern int w_id, d_id;

char *field_array[60];
int current_max_field;
int current_field;
char screen_buffer[3000];

unsigned char init_terminal[] = {
27,'x','0','\t',0x80,
27,'x','1',0x80,0x80,
27,'x','4','\r',0x80,0 };

unsigned char term_out[] = { 27,'V',0};


char send_beep [] = {7,0};
char disable_screen_buffer [] = {27,'&',27,')',0};

unsigned char first_screen[] = {
27,'*',     /* Clear to nulls */
27,',',     /* Clear to protected */
27,0x22,
27,')','W','a','r','e','h','o','u','s','e',' ',27,'(',' ',' ',' ',' ',
27,')', 'D','i','s','t','r','i','c','t',' ',27,'(',' ',' ',
27,')',
27,'&',
27,'N',
27,'(',
27,'0','1','S',
27,'B', 0 };



init_scr()
{
      int ret_code;
      int terminate = 0;


      ret_code = write(1,init_terminal,strlen(init_terminal));
      ret_code = write(1,first_screen,strlen(first_screen));


      current_field = 0;
      /* Display INIT screen and get user input for w_id, d_id */
      while (!terminate) {
            get_input(current_field);
            if (get_integer(&w_id, 4,current_field ) == 0) {
                  current_field++;
                  if (get_integer(&d_id, 2,current_field) == 0)
                        terminate = 1;
            }
      }
}

restore_scr()
{
}

/* Disable the screen and wait for the user to hit the send key */
wait_for_send()
{

      write(1,disable_screen_buffer,strlen(disable_screen_buffer));
      get_input(0);


}



/*
 * INPUT ROUTINES
 */

int
get_input(field_no)
int field_no;
{
      /* Get Input string */

      int i = 0, ch;
      int terminate=0,was_in_field = 0 ;
      char *input_buff_ptr;
      char tab_buffer[256],*tab_buffer_ptr;
      int number_char_returned,residue;

      input_buff_ptr = screen_buffer;
      residue = sizeof(screen_buffer);
      tab_buffer_ptr = tab_buffer;
      for (i=0;i<field_no+1;i++)
            *tab_buffer_ptr++ = '\t';
      *tab_buffer_ptr = 0;
      write(1,tab_buffer,strlen(tab_buffer));
      while (!terminate)
      {
            number_char_returned = read(0,input_buff_ptr,residue);

            input_buff_ptr += number_char_returned;
            residue -=number_char_returned;
            if ((*(input_buff_ptr - 1) == '\r') ||
                  (*(input_buff_ptr - 2) == '\r'))
                  terminate = 1;
      }
      input_buff_ptr = screen_buffer;
      current_max_field = 0;
      was_in_field = 0;
      while (*input_buff_ptr != '\r')
      {
            if (*input_buff_ptr == '\t')
            {
                  *input_buff_ptr = 0;
                  if (was_in_field)
                        current_max_field++;
                  was_in_field = 0;
            }
            else
```

```
                {
                        if (!was_in_field)
                                field_array[current_max_field] = input_buff_ptr;
                        was_in_field = 1;
                }
                input_buff_ptr++;
        }
}

print_error_on_field()
{
        int ret_code;

        ret_code = write(1,send_beep,strlen(send_beep));
}


/*
 * Function: get_char
 * Get a single char
 */
int
get_char(buf, field_no)
char *buf;
int field_no;
{
        int term, i, llen = 0, remlen, result, end, oldlen = 0;
        char chbuf[20], outbuf[20], *p;


        *buf = *field_array[field_no];

        return(0);
}
/*
 * Function: get_integer
 * Get an integer variable
 */
int
get_integer(buf, len,field_no)
int *buf;
int len, field_no;
{
        int term, i, llen = 0, remlen, result, end, oldlen = 0;
        char chbuf[20], outbuf[20], *p;

        oldlen = llen;
        llen = len; result = 0; end = 0;

        p = field_array[field_no];


        /* Now validate buf to see if it is an integer */
        if (oldlen > llen)/* If leftover input from previous edit, use it */
                llen = oldlen;
        for (i=0; i < llen; i++, p++) {
                if ( *p >= '0' && *p <= '9' && !end )
                        result = result * 10 + ( *p - '0');
                else if ( *p == ' ' )
                        end = 1;     /* Allow blanks only at end of input */
                else {
                        print_error_on_field(field_no);
                        return(INVALID_INTEGER);
                }
        }
        if (llen) {     /* If user entered any input */
                *buf = result;          /* Result integer */
        }
        return(0);
}

/*
 * Function: get_amount
 * Get an amount variable
 */
int
get_amount(buf, len ,field_no)
double *buf;
int len, field_no;
{
        int term, i, remlen, llen = 0, oldlen = 0, dotfound, num_afterdots;
        char chbuf[20], *p;
        char money[20], *moneyptr;
        char outbuf[20];
        int end = 0;

        oldlen = llen;/* Length from previous edit */
        llen = len; end = 0;
        moneyptr = money;
        p = field_array[field_no];

        /* Now validate buf to see if it is an amount */
        dotfound = 0;
        num_afterdots = 0;
        if (oldlen > llen)/* Use leftover from previous edit */
                llen = oldlen;
        for (i=0; i < llen; i++, p++) {
                if ( *p >= '0' && *p <= '9'  && !end ) {
                        if (dotfound) {/* fractional part */
                                if (num_afterdots >= 2) {
                                print_error_on_field(field_no);
                                return(INVALID_INTEGER);
                                }
                                num_afterdots++;
                        }
                        *moneyptr++ = *p;
                }
                else if ( *p == '.'  && !end ) {
                        if ( ! dotfound )
                                dotfound = 1;
                        else {
                           print_error_on_field(field_no);
                           return(INVALID_INTEGER);
                        }
                        *moneyptr++ = *p;
                }
                else if ( *p == ' ' )
                        end = 1;     /* Allow blanks only at end */
                else {
                    print_error_on_field(field_no);
                    return(INVALID_INTEGER);
                }
        }

        /* If length is less than requested, redisplay input
         * right-justified */
```

```
        if (llen) {/* If user entered any input */
                *moneyptr = '\0';
                *buf = atof(money);/* Result amount */
        }
        return(0);
}

/*
 * Function: get_string
 * Get a string variable
 * Returns: Terminating character
 */
int
get_string(buf, len, field_no)
char *buf;
int len, field_no;
{
        int remlen, term, i, llen = len;
        char *p1,*p2;

        p1 = (char *)field_array[field_no];
        for (i=0;i<len;i++)
        {
        buf[i] = *p1;
                p1++;
        }
        buf[len] = '\0';/* Result string */
        return(0);
}

/*
 * Copyright (c) 1995 by Sun Microsystems, Inc.
 */

#pragma ident "@(#)tpcc_srv_newordpay.sqc      1.13    95/09/14        SMI"

/*
 * File: newo.ec
 * Neworder transaction code for Informix using Tuxedo
 * Author : Shanti S
 * Date   : 8/03/93
 * Ported to DB2 by allanp 10-5-95
 * - Incorporates neworder/payment/order status transactions
 *
 */

#include "tpcc_db2.h"
#include <stdlib.h>
#include <sys/errno.h>
#include "tpcc_client.h"

void n_debugit (struct neword_struct *, char *);
void p_debugit(struct payment_struct *, char *);
void o_debugit(struct ordstat_struct *, char *);

#include "atmi.h"               /* TUXEDO header file */
#include "userlog.h"
#define TRAN_OK                 0
#define FATAL_SQLERROR          -1
#define INVALID_ITEM            100

/* Macros for structure operations */

#define MOVETO(struct, element1, element2) \
                        struct->element2 = struct##_in->element1
#define MOVEBACK(struct, element1, element2) \
                        struct##_in->element2 = struct->element1
#define MOVETOS(struct, element1, element2) \
                        struct->element2 = (short)struct##_in->element1
#define MOVEBACKI(struct, element1, element2) \
                        struct##_in->element2 = (int)struct->element1
#define MOVETOL(struct, element1, element2) \
                        struct->element2 = (long)struct##_in->element1
#define MOVEBACKL(struct, element1, element2) \
                        struct##_in->element2 = (long)struct->element1
#define MOVETOF(struct, element1, element2) \
                        struct->element2 = (float)struct##_in->element1
#define MOVEBACKD(struct, element1, element2) \
                        struct##_in->element2 = (double)struct->element1

#define MOVECTO(struct, element1, element2, cnt) { \
        int     i; \
        strncpy(struct->element2, struct##_in->element1, cnt);    \
        struct->element2[cnt] = '\0';    \
        for(i=0; i<=cnt; i++) \
                if(isspace(struct->element2[i])) \
                { \
                        struct->element2[i] = '\0'; \
                        break; \
                } \
        }
#define MOVECBACK(struct, element1, element2, cnt) { \
                strncpy(struct##_in->element2, struct->element1, cnt); \
                struct##_in->element2[cnt] = '\0'; \
        }

int duplicate_items = 0;
EXEC SQL include sqlca;

icmp (const void *a, const void *b)
{
  struct items_struct *one = (struct items_struct *)a;
  struct items_struct *two = (struct items_struct *)b;
  if (one->s_OL_I_ID == two->s_OL_I_ID) duplicate_items = 1;
  return (one->s_OL_I_ID - two->s_OL_I_ID);
}

EXEC SQL BEGIN DECLARE SECTION;
char dbname[8] = "dbbench";
EXEC SQL END DECLARE SECTION;

/*-------------------------------------------------------------------*/
/* Initialize environment: open database, set transaction       */
/* characteristics, etc...                                      */
/* (This is executed first when TUXEDO boots up.)               */
/*-------------------------------------------------------------------*/
int init_all_tx()
{
  char *dbname_ptr;
  if ((dbname_ptr = getenv("TPCC_DBNAME")) != NULL) {
    strcpy (dbname, dbname_ptr);
  }

  EXEC SQL CONNECT TO :dbname IN SHARE MODE;
  if (sqlca.sqlcode != 0) {
```

```
    fprintf(stderr, "NEWORD: CONNECT TO %s: SQLCODE = %d\n",
dbname, SQLCODE);
     exit(-1);
  }

}

/*--------------------------------------------------------------*/
/*            NEWORD Service                                    */
/*--------------------------------------------------------------*/
neworder_tx(clientmsg)

    TPSVCINFO *clientmsg;      /* TUXEDO message buffer structure */
{

  struct no_struct *neword_in;
  struct neword_struct neword1, *neword = &neword1;

  struct sqlda in_sqlda;

  int rc = 0;
  char buf[512] = "";
  int itemno;
  int all_local;

  neword_in = (struct no_struct *) clientmsg->data;

  all_local = 1;
  MOVETOS(neword, w_id, s_W_ID);
  MOVETOS(neword, d_id, s_D_ID);
  MOVETOS(neword, c_id, s_C_ID);
  MOVETOS(neword, o_id, s_O_ID);
  MOVETOS(neword, o_ol_cnt, s_O_OL_CNT);
  neword->s_O_ENTRY_D[0] = (char)NULL;
  neword->s_C_CREDIT[0] = (char)NULL;
  neword->s_C_LAST[0] = (char)NULL;
  neword->s_status_line[0] = (char)NULL;
  for (itemno = 0; itemno < neword_in->o_ol_cnt; itemno++) {
    if (neword_in->n_items[itemno].ol_supply_w_id != neword_in-
>w_id)
      all_local = 0;
    MOVETOS(neword, n_items[itemno].ol_supply_w_id,
item[itemno].s_OL_SUPPLY_W_ID);
    MOVETOL(neword, n_items[itemno].ol_i_id,
item[itemno].s_OL_I_ID);
    neword->item[itemno].s_I_NAME[0] = (char)NULL;
    MOVETOS(neword, n_items[itemno].ol_quantity,
item[itemno].s_OL_QUANTITY);
    MOVETO(neword, n_items[itemno].brand[0],
item[itemno].s_brand_generic);
  }
  neword->s_all_local = all_local;

  duplicate_items = 0;
  qsort (neword->item, neword->s_O_OL_CNT, sizeof (neword-
>item[0]), icmp);
  neword->duplicate_items = duplicate_items;

  in_sqlda.sqln = 1;
  in_sqlda.sqld = 1;
  in_sqlda.sqlvar[0].sqltype = SQL_TYP_CHAR;
  in_sqlda.sqlvar[0].sqldata = (char *)neword;
  in_sqlda.sqlvar[0].sqllen = sizeof( struct neword_struct );
  in_sqlda.sqlvar[0].sqlind = NULL;
  neword->deadlocks = 0;
  neword->s_transtatus = FATAL_SQLERROR;

  EXEC SQL CALL "newords" USING DESCRIPTOR :in_sqlda;

  if (sqlca.sqlcode) {
    rc = sqlaintp(buf, 512, 78, &sqlca);
    fprintf(stderr,"\nNEWORD dari failed with sqlcode =
%ld\n%s\n",sqlca.sqlcode, buf);
    n_debugit(neword, "sqlcode != 0");
    return(-1);
  }

  if (neword->s_transtatus <= FATAL_SQLERROR) {
    fprintf(stderr,"NEWORD transaction failed, transtatus = %d.\n",
        neword->s_transtatus);
    n_debugit(neword, "s_transtatus <= FATAL_SQLERROR");
    return(-1);
  }

  MOVEBACKI(neword, s_W_ID, w_id);
  MOVEBACKI(neword, s_D_ID, d_id);
  MOVEBACKI(neword, s_C_ID, c_id);
  MOVEBACKI(neword, s_O_ID, o_id);
  MOVEBACKI(neword, s_O_OL_CNT, o_ol_cnt);
  MOVEBACK(neword, s_C_DISCOUNT, c_discount);
  MOVEBACKD(neword, s_W_TAX, w_tax);
  MOVEBACKD(neword, s_D_TAX, d_tax);
  MOVECBACK(neword, s_O_ENTRY_D, o_entry_d, 19);
  MOVECBACK(neword, s_C_CREDIT, c_credit, 2);
  MOVECBACK(neword, s_C_LAST, c_last, LAST_LEN - 1);
  MOVECBACK(neword, s_status_line, status, 24);
  for (itemno = 0; itemno < neword_in->o_ol_cnt; itemno++) {
    neword->s_total_amount += neword->item[itemno].s_OL_AMOUNT;
    MOVEBACKI(neword, item[itemno].s_OL_SUPPLY_W_ID,
n_items[itemno].ol_supply_w_id);
    MOVEBACKI(neword, item[itemno].s_OL_I_ID, n_items[itemno].ol_i_id);
    MOVECBACK(neword, item[itemno].s_I_NAME, n_items[itemno].i_name,
24);
    MOVEBACKI(neword, item[itemno].s_OL_QUANTITY,
n_items[itemno].ol_quantity);
    MOVEBACKI(neword, item[itemno].s_S_QUANTITY,
n_items[itemno].s_quantity);
    MOVEBACK(neword, item[itemno].s_brand_generic,
n_items[itemno].brand[0]);
    MOVEBACK(neword, item[itemno].s_I_PRICE, n_items[itemno].i_price);
    MOVEBACK(neword, item[itemno].s_OL_AMOUNT,
n_items[itemno].ol_amount);
  }
  neword_in->total = neword->s_total_amount *
      (1 - neword->s_C_DISCOUNT) * (1 + neword->s_W_TAX + neword-
>s_D_TAX);

  return (rc);
}

/*--------------------------------------------------------------*/
/*            PAYMENT Service                                   */
/*--------------------------------------------------------------*/
payment_tx(clientmsg)
    TPSVCINFO *clientmsg;    /* TUXEDO message buffer structure */
{
```

```
    struct payment_struct payment1, *payment = &payment1;
    struct pay_struct *payment_in;

    struct sqlda in_sqlda;

    int rc = 0;
    char buf[512] = "";
    char tmpstr[202];

    payment_in = (struct pay_struct *) clientmsg->data;

    MOVETOS(payment, w_id, s_W_ID);
    MOVETOS(payment, d_id, s_D_ID);
    MOVETOL(payment, c_id, s_C_ID);
    MOVETOS(payment, c_w_id, s_C_W_ID);
    MOVETOS(payment, c_d_id, s_C_D_ID);
    MOVETO(payment, h_amount, s_H_AMOUNT);
    payment->s_H_DATE[0] = (char)NULL;
    payment->s_W_STREET_1[0] = (char)NULL;
    payment->s_W_STREET_2[0] = (char)NULL;
    payment->s_W_CITY[0] = (char)NULL;
    payment->s_W_STATE[0] = (char)NULL;
    payment->s_W_ZIP[0] = (char)NULL;
    payment->s_D_STREET_1[0] = (char)NULL;
    payment->s_D_STREET_2[0] = (char)NULL;
    payment->s_D_CITY[0] = (char)NULL;
    payment->s_D_STATE[0] = (char)NULL;
    payment->s_D_ZIP[0] = (char)NULL;
    payment->s_C_FIRST[0] = (char)NULL;
    payment->s_C_MIDDLE[0] = (char)NULL;
    MOVECTO(payment, c_last, s_C_LAST, 16);
    payment->s_C_STREET_1[0] = (char)NULL;
    payment->s_C_STREET_2[0] = (char)NULL;
    payment->s_C_CITY[0] = (char)NULL;
    payment->s_C_STATE[0] = (char)NULL;
    payment->s_C_ZIP[0] = (char)NULL;
    payment->s_C_PHONE[0] = (char)NULL;
    payment->s_C_SINCE[0] = (char)NULL;
    payment->s_C_CREDIT[0] = (char)NULL;
    payment->s_C_DATA[0] = (char)NULL;

    payment->deadlocks = 0;
    payment->s_transtatus = FATAL_SQLERROR;
    /* Input SQLDA */
    in_sqlda.sqln = 1;
    in_sqlda.sqld = 1;
    in_sqlda.sqlvar[0].sqltype = SQL_TYP_CHAR;
    in_sqlda.sqlvar[0].sqldata = (char *)payment;
    in_sqlda.sqlvar[0].sqllen = sizeof( struct payment_struct );
    in_sqlda.sqlvar[0].sqlind = NULL;

    EXEC SQL CALL "pays" USING DESCRIPTOR :in_sqlda;

    if (sqlca.sqlcode) {
       rc = sqlaintp(buf, 512, 78, &sqlca);
       fprintf(stderr,"\nPAYMENT dari failed with sqlcode =
%ld\n%s\n",sqlca.sqlcode, buf);
       p_debugit(payment, "sqlcode != 0");
       return(-1);
    }

    if (payment->s_transtatus <= FATAL_SQLERROR) {
       fprintf(stderr,"PAYMENT transaction failed, transtatus = %d.\n",
            payment->s_transtatus);
       p_debugit(payment, "s_transtatus <= FATAL_SQLERROR");
       return(-1);
    }

    MOVEBACKI(payment, s_W_ID, w_id);
    MOVEBACKI(payment, s_D_ID, d_id);
    MOVEBACKI(payment, s_C_ID, c_id);
    MOVEBACKI(payment, s_C_W_ID, c_w_id);
    MOVEBACKI(payment, s_C_D_ID, c_d_id);
    MOVEBACK(payment, s_H_AMOUNT, h_amount);
    MOVEBACK(payment, s_C_CREDIT_LIM, c_credit_lim);
    MOVEBACK(payment, s_C_BALANCE, c_balance);
    MOVEBACK(payment, s_C_DISCOUNT, c_discount);
    MOVECBACK(payment, s_H_DATE, h_date, 19);
    MOVECBACK(payment, s_W_STREET_1, w_street_1, 20);
    MOVECBACK(payment, s_W_STREET_2, w_street_2, 20);
    MOVECBACK(payment, s_W_CITY, w_city, 20);
    MOVECBACK(payment, s_W_STATE, w_state, 20);
    MOVECBACK(payment, s_W_ZIP, w_zip, 9);
    MOVECBACK(payment, s_D_STREET_1, d_street_1, 20);
    MOVECBACK(payment, s_D_STREET_2, d_street_2, 20);
    MOVECBACK(payment, s_D_CITY, d_city, 20);
    MOVECBACK(payment, s_D_STATE, d_state, 20);
    MOVECBACK(payment, s_D_ZIP, d_zip, 9);
    MOVECBACK(payment, s_C_FIRST, c_first, 16);
    MOVECBACK(payment, s_C_MIDDLE, c_middle, 2);
    MOVECBACK(payment, s_C_LAST, c_last, 16);
    MOVECBACK(payment, s_C_STREET_1, c_street_1, 20);
    MOVECBACK(payment, s_C_STREET_2, c_street_2, 20);
    MOVECBACK(payment, s_C_CITY, c_city, 20);
    MOVECBACK(payment, s_C_STATE, c_state, 20);
    MOVECBACK(payment, s_C_ZIP, c_zip, 9);
    MOVECBACK(payment, s_C_PHONE, c_phone, 16);
    MOVECBACK(payment, s_C_SINCE, c_since, 10);
    MOVECBACK(payment, s_C_CREDIT, c_credit, 2);
    if (!strncmp(payment->s_C_CREDIT, "BC", 2)) {
       if (strlen(payment->s_C_DATA) > 150) {
          strncpy(payment_in->c_data_1, payment->s_C_DATA, 50);
          payment_in->c_data_1[50] = '\0';
          strncpy(payment_in->c_data_2, &payment->s_C_DATA[50], 50);
          payment_in->c_data_2[50] = '\0';
          strncpy(payment_in->c_data_3, &payment->s_C_DATA[100], 50);
          payment_in->c_data_3[50] = '\0';
          strcpy(payment_in->c_data_4, &payment->s_C_DATA[150]);
       }
       else if (strlen(payment->s_C_DATA) > 100) {
          strncpy(payment_in->c_data_1, payment->s_C_DATA, 50);
          payment_in->c_data_1[50] = '\0';
          strncpy(payment_in->c_data_2, &payment->s_C_DATA[50], 50);
          payment_in->c_data_2[50] = '\0';
          strcpy(payment_in->c_data_3, &payment->s_C_DATA[100]);
          payment_in->c_data_4[0] = '\0';
       }
       else if (strlen(payment->s_C_DATA) > 50) {
          strncpy(payment_in->c_data_1, payment->s_C_DATA, 50);
          payment_in->c_data_1[50] = '\0';
          strcpy(payment_in->c_data_2, &payment->s_C_DATA[50]);
          payment_in->c_data_3[0] = '\0';
          payment_in->c_data_4[0] = '\0';
       }
       else {
          strcpy(payment_in->c_data_1, payment->s_C_DATA);
```

```
        payment_in->c_data_2[0] = '\0';
        payment_in->c_data_3[0] = '\0';
        payment_in->c_data_4[0] = '\0';
      }
  }

  return (rc);
}

/*---------------------------------------------------------------*/
/*            ORDSTAT Standard Service                           */
/*---------------------------------------------------------------*/
ordstat_tx(clientmsg)
    TPSVCINFO *clientmsg;    /* TUXEDO message buffer structure */
{
  struct ord_struct *ordstat_in;
  struct ordstat_struct ordstat1, *ordstat = &ordstat1;

  struct sqlda in_sqlda;

  int rc = 0;
  char buf[512] = "";
  int itemno;

  ordstat_in = (struct ord_struct *) clientmsg->data;

  ordstat->s_ol_cnt = 0;
  MOVETOS(ordstat, w_id, s_W_ID);
  MOVETOS(ordstat, d_id, s_D_ID);
  MOVETOL(ordstat, c_id, s_C_ID);
  ordstat->s_C_FIRST[0] = (char)NULL;
  ordstat->s_C_MIDDLE[0] = (char)NULL;
  MOVECTO(ordstat, c_last, s_C_LAST, 16);
  ordstat->s_O_ENTRY_D[0] = (char)NULL;

  ordstat->deadlocks = 0;
  ordstat->s_transtatus = FATAL_SQLERROR;
  /* Input SQLDA */
  in_sqlda.sqln = 1;
  in_sqlda.sqld = 1;
  in_sqlda.sqlvar[0].sqltype = SQL_TYP_CHAR;
  in_sqlda.sqlvar[0].sqldata = (char *)ordstat;
  in_sqlda.sqlvar[0].sqllen = sizeof( struct ordstat_struct );
  in_sqlda.sqlvar[0].sqlind = NULL;

  EXEC SQL CALL "ords" USING DESCRIPTOR :in_sqlda;

  if (sqlca.sqlcode) {
    rc = sqlaintp(buf, 512, 78, &sqlca);
    fprintf(stderr,"\nORDSTAT dari failed with sqlcode =
%ld\n%s\n",sqlca.sqlcode, buf);
    o_debugit(ordstat, "sqlcode != 0");
    return(-1);
  }

  if (ordstat->s_transtatus <= FATAL_SQLERROR) {
    fprintf(stderr,"ORDSTAT transaction failed, transtatus = %d.\n",
        ordstat->s_transtatus);
    o_debugit(ordstat, "s_transtatus <= FATAL_SQLERROR");
    return(-1);
  }

  MOVEBACKI(ordstat, s_ol_cnt, item_cnt);
  MOVEBACKI(ordstat, s_W_ID, w_id);
  MOVEBACKI(ordstat, s_D_ID, d_id);
  MOVEBACKI(ordstat, s_C_ID, c_id);
  MOVEBACKI(ordstat, s_O_ID, o_id);
  MOVEBACKI(ordstat, s_O_CARRIER_ID, o_carrier_id);
  MOVEBACK(ordstat, s_C_BALANCE, c_balance);
  MOVECBACK(ordstat, s_C_FIRST, c_first, 16);
  MOVECBACK(ordstat, s_C_MIDDLE, c_middle, 2);
  MOVECBACK(ordstat, s_C_LAST, c_last, 16);
  MOVECBACK(ordstat, s_O_ENTRY_D, o_entry_d, 19);
  for (itemno = 0; itemno < ordstat->s_ol_cnt; itemno++) {
    MOVEBACKI(ordstat, item[itemno].s_OL_SUPPLY_W_ID,
o_items[itemno].ol_supply_w_id);
    MOVEBACKI(ordstat, item[itemno].s_OL_I_ID, o_items[itemno].ol_i_id);
    MOVEBACKI(ordstat, item[itemno].s_OL_QUANTITY,
o_items[itemno].ol_quantity);
    MOVEBACK(ordstat, item[itemno].s_OL_AMOUNT,
o_items[itemno].ol_amount);
    MOVECBACK(ordstat, item[itemno].s_OL_DELIVERY_D,
o_items[itemno].ol_delivery_d, 10);
  }

  return (rc);
}

/*---------------------------------------------------------------*/
/*       n_debugit                                               */
/*---------------------------------------------------------------*/
void n_debugit (struct neword_struct *neword_ptr, char *msg)
{
  FILE *debug_fp;
  char  debug_fn[25] = "/tmp/neword.debug.out";
  int   j, items;
  if ((debug_fp = fopen(debug_fn, "a+")) == NULL)
  {
    return;
  }
  fprintf(debug_fp,"n_debugit information follows (%s)\n", msg);

fprintf(debug_fp,"================================================
===\n");
  fprintf(debug_fp,"neword_struct {\n");
  fprintf(debug_fp,"\ts_W_ID        = %hd (%hX)\n",
        neword_ptr->s_W_ID,neword_ptr->s_W_ID);
  fprintf(debug_fp,"\ts_D_ID        = %hd (%hX)\n",
        neword_ptr->s_D_ID,neword_ptr->s_D_ID);
  fprintf(debug_fp,"\ts_C_ID        = %ld (%lX)\n",
        neword_ptr->s_C_ID,neword_ptr->s_C_ID);
  fprintf(debug_fp,"\ts_C_LAST      = %s\n",
        neword_ptr->s_C_LAST);
  fprintf(debug_fp,"\ts_C_CREDIT    = %s\n",
        neword_ptr->s_C_CREDIT);
  fprintf(debug_fp,"\ts_C_DISCOUNT  = %0.4f\n",
        neword_ptr->s_C_DISCOUNT);
  fprintf(debug_fp,"\ts_O_OL_CNT    = %hd (%hX)\n",
        neword_ptr->s_O_OL_CNT,neword_ptr->s_O_OL_CNT);
  fprintf(debug_fp,"\ts_O_ID        = %ld (%lX)\n",
        neword_ptr->s_O_ID,neword_ptr->s_O_ID);
  fprintf(debug_fp,"\ts_O_ENTRY_D   = %s\n",
        neword_ptr->s_O_ENTRY_D);
  fprintf(debug_fp,"\ts_status_line  = %s\n",
        neword_ptr->s_status_line);
  fprintf(debug_fp,"\ts_total_amount = %0.2f\n",
```

```
        neword_ptr->s_total_amount);
  fprintf(debug_fp,"\ts_all_local    = %hd (%hX)\n",
        neword_ptr->s_all_local,neword_ptr->s_all_local);
  fprintf(debug_fp,"\ts_transtatus   = %hd (%hX)\n",
        neword_ptr->s_transtatus,neword_ptr->s_transtatus);
  fprintf(debug_fp,"\ts_W_TAX       = %0.4f\n",
        neword_ptr->s_W_TAX);
  fprintf(debug_fp,"\ts_D_TAX       = %0.4f\n",
        neword_ptr->s_D_TAX);
  fprintf(debug_fp,"\tduplicate_items= %hd (%hX)\n",
        neword_ptr->duplicate_items,neword_ptr->duplicate_items);
  fprintf(debug_fp,"\tdeadlocks     = %hd (%hX)\n",
        neword_ptr->deadlocks,neword_ptr->deadlocks);
  fprintf(debug_fp,"\titems {\n");
  items = neword_ptr->s_O_OL_CNT;
  for (j=0; j<items; j++) {
    fprintf(debug_fp,"\t\ts_OL_SUPPLY_W_ID[%d] = %hd (%hX)\n",
          j,neword_ptr->item[j].s_OL_SUPPLY_W_ID,neword_ptr-
>item[j].s_OL_SUPPLY_W_ID);
    fprintf(debug_fp,"\t\ts_OL_I_ID[%d]       = %ld (%lX)\n",
        j,neword_ptr->item[j].s_OL_I_ID,neword_ptr->item[j].s_OL_I_ID);
    fprintf(debug_fp,"\t\ts_I_NAME[%d]       = %s\n",
          j,neword_ptr->item[j].s_I_NAME);
    fprintf(debug_fp,"\t\ts_OL_QUANTITY[%d]   = %hd (%hX)\n",
          j,neword_ptr->item[j].s_OL_QUANTITY,neword_ptr-
>item[j].s_OL_QUANTITY);
    fprintf(debug_fp,"\t\ts_S_QUANTITY[%d]    = %hd (%hX)\n",
          j,neword_ptr->item[j].s_S_QUANTITY,neword_ptr-
>item[j].s_S_QUANTITY);
    fprintf(debug_fp,"\t\ts_brand_generic[%d] = %c\n",
          j,neword_ptr->item[j].s_brand_generic);
    fprintf(debug_fp,"\t\ts_I_PRICE[%d]      = %0.2f\n",
          j,neword_ptr->item[j].s_I_PRICE);
    fprintf(debug_fp,"\t\ts_OL_AMOUNT[%d]    = %0.2f\n\n",
          j,neword_ptr->item[j].s_OL_AMOUNT);
  }
  fprintf(debug_fp,"\t}\n}\n\n");
  fclose(debug_fp);
}

/*---------------------------------------------------------------*/
/*      p_debugit                                                */
/*---------------------------------------------------------------*/
void p_debugit (struct payment_struct *payment_ptr, char *msg)
{
  FILE *debug_fp;
  char  debug_fn[25] = "/tmp/payment.debug.out";
  if ((debug_fp = fopen(debug_fn, "a+")) == NULL)
  {
    return;
  }
  fprintf(debug_fp,"p_debugit information follows (%s)\n", msg);

fprintf(debug_fp,"=========================================
========\n");
  fprintf(debug_fp,"payment_struct {\n");
  fprintf(debug_fp,"\ts_W_ID        = %hd (%hX)\n",
        payment_ptr->s_W_ID,payment_ptr->s_W_ID);
  fprintf(debug_fp,"\ts_D_ID        = %hd (%hX)\n",
        payment_ptr->s_D_ID,payment_ptr->s_D_ID);
  fprintf(debug_fp,"\ts_C_ID        = %ld (%lX)\n",
        payment_ptr->s_C_ID,payment_ptr->s_C_ID);
  fprintf(debug_fp,"\ts_C_D_ID      = %hd (%hX)\n",
        payment_ptr->s_C_D_ID,payment_ptr->s_C_D_ID);
fprintf(debug_fp,"\ts_C_W_ID      = %hd (%hX)\n",
        payment_ptr->s_C_W_ID,payment_ptr->s_C_W_ID);
fprintf(debug_fp,"\ts_transtatus   = %hd (%hX)\n",
        payment_ptr->s_transtatus,payment_ptr->s_transtatus);
fprintf(debug_fp,"\ts_H_AMOUNT    = %0.2f\n",
        payment_ptr->s_H_AMOUNT);
fprintf(debug_fp,"\ts_H_DATE      = %s\n",
        payment_ptr->s_H_DATE);
fprintf(debug_fp,"\ts_W_STREET_1   = %s\n",
        payment_ptr->s_W_STREET_1);
fprintf(debug_fp,"\ts_W_STREET_2   = %s\n",
        payment_ptr->s_W_STREET_2);
fprintf(debug_fp,"\ts_W_CITY      = %s\n",
        payment_ptr->s_W_CITY);
fprintf(debug_fp,"\ts_W_STATE     = %s\n",
        payment_ptr->s_W_STATE);
fprintf(debug_fp,"\ts_W_ZIP       = %s\n",
        payment_ptr->s_W_ZIP);
fprintf(debug_fp,"\ts_D_STREET_1   = %s\n",
        payment_ptr->s_D_STREET_1);
fprintf(debug_fp,"\ts_D_STREET_2   = %s\n",
        payment_ptr->s_D_STREET_2);
fprintf(debug_fp,"\ts_D_CITY      = %s\n",
        payment_ptr->s_D_CITY);
fprintf(debug_fp,"\ts_D_STATE     = %s\n",
        payment_ptr->s_D_STATE);
fprintf(debug_fp,"\ts_D_ZIP       = %s\n",
        payment_ptr->s_D_ZIP);
fprintf(debug_fp,"\ts_C_FIRST     = %s\n",
        payment_ptr->s_C_FIRST);
fprintf(debug_fp,"\ts_C_MIDDLE    = %s\n",
        payment_ptr->s_C_MIDDLE);
fprintf(debug_fp,"\ts_C_LAST      = %s\n",
        payment_ptr->s_C_LAST);
fprintf(debug_fp,"\ts_C_STREET_1   = %s\n",
        payment_ptr->s_C_STREET_1);
fprintf(debug_fp,"\ts_C_STREET_2   = %s\n",
        payment_ptr->s_C_STREET_2);
fprintf(debug_fp,"\ts_C_CITY      = %s\n",
        payment_ptr->s_C_CITY);
fprintf(debug_fp,"\ts_C_STATE     = %s\n",
        payment_ptr->s_C_STATE);
fprintf(debug_fp,"\ts_C_ZIP       = %s\n",
        payment_ptr->s_C_ZIP);
fprintf(debug_fp,"\ts_C_PHONE     = %s\n",
        payment_ptr->s_C_PHONE);
fprintf(debug_fp,"\ts_C_SINCE     = %s\n",
        payment_ptr->s_C_SINCE);
fprintf(debug_fp,"\ts_C_CREDIT    = %s\n",
        payment_ptr->s_C_CREDIT);
fprintf(debug_fp,"\ts_C_CREDIT_LIM = %0.2f\n",
        payment_ptr->s_C_CREDIT_LIM);
fprintf(debug_fp,"\ts_C_DISCOUNT   = %0.2f\n",
        payment_ptr->s_C_DISCOUNT);
fprintf(debug_fp,"\ts_C_BALANCE    = %0.2f\n",
        payment_ptr->s_C_BALANCE);
fprintf(debug_fp,"\tdeadlocks     = %hd (%hX)\n",
        payment_ptr->deadlocks,payment_ptr->deadlocks);
fprintf(debug_fp,"\ts_C_DATA      = %s\n",
        payment_ptr->s_C_DATA);
fprintf(debug_fp,"\n}\n\n");
fclose(debug_fp);
```

```
}

/*---------------------------------------------------------------*/
/*      o_debugit                                                */
/*---------------------------------------------------------------*/
void o_debugit (struct ordstat_struct *ordstat_ptr, char *msg)
{
  FILE *debug_fp;
  char  debug_fn[25] = "/tmp/ordstat.debug.out";
  int   j, items;
  if ((debug_fp = fopen(debug_fn, "a+")) == NULL)
  {
    return;
  }
  fprintf(debug_fp,"o_debugit information follows (%s)\n", msg);

fprintf(debug_fp,"=======================================
==========\n");
  fprintf(debug_fp,"ordstat_struct {\n");
  fprintf(debug_fp,"\ts_W_ID         = %hd (%hX)\n",
        ordstat_ptr->s_W_ID,ordstat_ptr->s_W_ID);
  fprintf(debug_fp,"\ts_D_ID         = %hd (%hX)\n",
        ordstat_ptr->s_D_ID,ordstat_ptr->s_D_ID);
  fprintf(debug_fp,"\ts_C_ID         = %ld (%lX)\n",
        ordstat_ptr->s_C_ID,ordstat_ptr->s_C_ID);
  fprintf(debug_fp,"\ts_C_FIRST      = %s\n",
        ordstat_ptr->s_C_FIRST);
  fprintf(debug_fp,"\ts_C_MIDDLE     = %s\n",
        ordstat_ptr->s_C_MIDDLE);
  fprintf(debug_fp,"\ts_C_LAST       = %s\n",
        ordstat_ptr->s_C_LAST);
  fprintf(debug_fp,"\ts_C_BALANCE    = %0.2f\n",
        ordstat_ptr->s_C_BALANCE);
  fprintf(debug_fp,"\ts_O_ID         = %ld (%lX)\n",
        ordstat_ptr->s_O_ID,ordstat_ptr->s_O_ID);
  fprintf(debug_fp,"\ts_O_ENTRY_D    = %s\n",
        ordstat_ptr->s_O_ENTRY_D);
  fprintf(debug_fp,"\ts_O_CARRIER_ID = %hd (%hX)\n",
        ordstat_ptr->s_O_CARRIER_ID,ordstat_ptr-
>s_O_CARRIER_ID);
  fprintf(debug_fp,"\ts_ol_cnt       = %hd (%hX)\n",
        ordstat_ptr->s_ol_cnt,ordstat_ptr->s_ol_cnt);
  fprintf(debug_fp,"\ts_transtatus   = %hd (%hX)\n",
        ordstat_ptr->s_transtatus,ordstat_ptr->s_transtatus);
  fprintf(debug_fp,"\tdeadlocks      = %hd (%hX)\n",
        ordstat_ptr->deadlocks,ordstat_ptr->deadlocks);
  fprintf(debug_fp,"\titems {\n");
  items = ordstat_ptr->s_ol_cnt;
  for (j=0; j<items; j++) {
    fprintf(debug_fp,"\t\ts_OL_SUPPLY_W_ID[%d] = %hd (%hX)\n",
          j,ordstat_ptr->item[j].s_OL_SUPPLY_W_ID,ordstat_ptr-
>item[j].s_OL_SUPPLY_W_ID);
    fprintf(debug_fp,"\t\ts_OL_I_ID[%d]       = %ld (%lX)\n",
          j,ordstat_ptr->item[j].s_OL_I_ID,ordstat_ptr-
>item[j].s_OL_I_ID);
    fprintf(debug_fp,"\t\ts_OL_QUANTITY[%d]   = %hd (%hX)\n",
          j,ordstat_ptr->item[j].s_OL_QUANTITY,ordstat_ptr-
>item[j].s_OL_QUANTITY);
    fprintf(debug_fp,"\t\ts_OL_AMOUNT[%d]     = %0.2f\n",
          j,ordstat_ptr->item[j].s_OL_AMOUNT);
    fprintf(debug_fp,"\t\ts_OL_DELIVERY_D[%d] = %s\n\n",
          j,ordstat_ptr->item[j].s_OL_DELIVERY_D);
  }
  fprintf(debug_fp,"\t}\n}\n\n");
  fclose(debug_fp);
}

/* Start of Tuxedo code */
int
tpsvrinit (argc, argv)
char **argv;
{
      return(init_all_tx());          /* Prepare transaction */
}

void
tpsvrdone()
{
}

NEWO(rqst)
TPSVCINFO *rqst;
{
      if (neworder_tx(rqst))
            tpreturn(TPFAIL, 0, rqst->data, sizeof(struct no_struct), 0);
      else
            tpreturn(TPSUCCESS, 0, rqst->data, sizeof(struct no_struct), 0);
}

PAYM(rqst)
TPSVCINFO *rqst;
{
      if (payment_tx(rqst) )
            tpreturn(TPFAIL, 0, rqst->data, sizeof(struct pay_struct), 0);
      else
            tpreturn(TPSUCCESS, 0, rqst->data, sizeof(struct pay_struct), 0);
}

ORDS(rqst)
TPSVCINFO *rqst;
{
      if (ordstat_tx(rqst) )
            tpreturn(TPFAIL, 0, rqst->data, sizeof(struct ord_struct), 0);
      else
            tpreturn(TPSUCCESS, 0, rqst->data, sizeof(struct ord_struct), 0);
}

/*
 * Copyright (c) 1994 by Sun Microsystems, Inc.
 */

#pragma ident "@(#)tpcc_srv_stockdel.sqc      1.5     94/12/07        SMI"

/*
 * File: tpcc_srv_stockdel.sqc
 *
 * Author : Shanti S
 * Date   : 8/04/93
 * Ported to DB2 by allanp
 * 11/29/95
 */

#include "tpcc_db2.h"
#include <time.h>
#include "tpcc_client.h"
#include <sys/utsname.h>
```

```
void static d_debugit(struct delivery_struct *, char *);
void s_debugit (struct stocklev_struct *, char *);


#include "atmi.h"               /* TUXEDO header file */
#define TRAN_OK                 0
#define FATAL_SQLERROR          -1

/* Macros for structure operations */

#define MOVETO(struct, element1, element2) \
                struct->element2 = struct##_in->element1
#define MOVEBACK(struct, element1, element2) \
                struct##_in->element2 = struct->element1
#define MOVETOS(struct, element1, element2) \
                struct->element2 = (short)struct##_in->element1
#define MOVEBACKI(struct, element1, element2) \
                struct##_in->element2 = (int)struct->element1
#define MOVETOL(struct, element1, element2) \
                struct->element2 = (long)struct##_in->element1
#define MOVEBACKL(struct, element1, element2) \
                struct##_in->element2 = (long)struct->element1
#define MOVETOF(struct, element1, element2) \
                struct->element2 = (float)struct##_in->element1
#define MOVEBACKD(struct, element1, element2) \
                struct##_in->element2 = (double)struct->element1

#define MOVECTO(struct, element1, element2, cnt) { \
        int     i; \
        strncpy(struct->element2, struct##_in->element1, cnt);    \
        struct->element2[cnt] = '\0';    \
        for(i=0; i<=cnt; i++) \
                if(isspace(struct->element2[i])) \
                { \
                        struct->element2[i] = '\0'; \
                        break; \
                } \
        }
#define MOVECBACK(struct, element1, element2, cnt) { \
                strncpy(struct##_in->element2, struct->element1, cnt); \
                struct##_in->element2[cnt] = '\0'; \
        }

EXEC SQL include sqlca;

EXEC SQL BEGIN DECLARE SECTION;
char dbname[8] = "dbbench";
EXEC SQL END DECLARE SECTION;

static char outbuf[1024];       /* Buffer for results file */
static int tx_count = 0;        /* Transaction counter */
static FILE *delfile;

/*---------------------------------------------------------------*/
/* Initialize environment: open database, set transaction       */
/* characteristics, etc...                                      */
/* (This is executed first when TUXEDO boots up.)               */
/*---------------------------------------------------------------*/
int init_all_tx ()

{
  char *dbname_ptr;

  if ((dbname_ptr = getenv("TPCC_DBNAME")) != NULL) {
    strcpy (dbname, dbname_ptr);
  }

  EXEC SQL CONNECT TO :dbname IN SHARE MODE;
  if (sqlca.sqlcode != 0) {
    fprintf(stderr, "DELSTOCK: CONNECT TO %s: SQLCODE = %d\n",
dbname, SQLCODE);
    exit(-1);
  }

}

/* void static sqlerror(); */

/*---------------------------------------------------------------*/
/*              DELIVERY Service                                 */
/*---------------------------------------------------------------*/
delivery_tx (clientmsg)
    TPSVCINFO *clientmsg;    /* TUXEDO message buffer structure */
{
  struct req_struct *delivery_in;
  struct delivery_struct delivery1, *delivery = &delivery1;

  struct sqlda in_sqlda;

  int rc = 0;
  char buf[512] = "";
  int i;

  delivery_in = (struct req_struct *) clientmsg->data;

  MOVETOS(delivery, w_id, s_W_ID);
  MOVETOS(delivery, o_carrier_id, s_O_CARRIER_ID);
  for (i=0; i<10; i++)
      delivery->s_O_ID[i] = 0;

  tx_count++;
  sprintf(outbuf, "Starting transaction %d queued at %d\n",
        tx_count, delivery_in->qtime);

  in_sqlda.sqln = 1;
  in_sqlda.sqld = 1;
  in_sqlda.sqlvar[0].sqltype = SQL_TYP_CHAR;
  in_sqlda.sqlvar[0].sqldata = (char *)delivery;
  in_sqlda.sqlvar[0].sqllen = sizeof( struct delivery_struct );
  in_sqlda.sqlvar[0].sqlind = NULL;
  delivery->deadlocks = 0;
  delivery->s_transtatus = FATAL_SQLERROR;

  EXEC SQL CALL "deliverys" USING DESCRIPTOR :in_sqlda;

  if (sqlca.sqlcode) {
    rc = sqlaintp(buf, 512, 78, &sqlca);
    fprintf(stderr,"\nDELIVERY dari failed with sqlcode =
%ld\n%s\n",sqlca.sqlcode, buf);
    d_debugit(delivery, "sqlcode != 0");
    return(-1);
  }

  if (delivery->s_transtatus <= FATAL_SQLERROR) {
    fprintf(stderr,"DELIVERY transaction failed, transtatus = %d.\n",
          delivery->s_transtatus);
```

```
      d_debugit(delivery, "s_transtatus <= FATAL_SQLERROR");
      return(-1);
   }

   MOVEBACKI(delivery, s_W_ID, w_id);
   MOVEBACKI(delivery, s_O_CARRIER_ID, o_carrier_id);

   for (i = 0; i < 10; i++) {
      if (delivery->s_O_ID[i] == 0) {
            /* No order found for this district */
                  sprintf(outbuf+strlen(outbuf),
                  "Delivery for District %d skipped\n", i+1);
      }
      else {
         sprintf(outbuf+strlen(outbuf),
         "Delivered order %d for district %d, warehouse %d, carrier
%d\n",
               delivery->s_O_ID[i], i+1, delivery->s_W_ID,
               delivery->s_O_CARRIER_ID);
      }
   }

   sprintf(outbuf+strlen(outbuf), "Transaction completed at %d\n",
time(0));
   fwrite(outbuf, strlen(outbuf), 1, delfile);
   fflush(delfile);

   return (rc);
}

/*---------------------------------------------------------------*/
/*            STOCKLEV Service                                   */
/*---------------------------------------------------------------*/
stocklevel_tx(clientmsg)
    TPSVCINFO *clientmsg;    /* TUXEDO message buffer structure */
{
   struct stocklev_struct stocklev1, *stocklev = &stocklev1;
   struct stock_struct *stocklev_in;

   struct sqlda in_sqlda;

   int rc = 0;
   char buf[512] = "";

   EXEC SQL BEGIN DECLARE SECTION;
   long    d_next_o_id, o_id;
   long    w_id, d_id, low_stock;
   short   threshold;
   EXEC SQL END DECLARE SECTION;

   stocklev_in = (struct stock_struct *) clientmsg->data;

   MOVETOS(stocklev, w_id, s_W_ID);
   MOVETOS(stocklev, d_id, s_D_ID);
   MOVETOS(stocklev, threshold, s_threshold);
   MOVETOL(stocklev, low_stock, s_low_stock);

   stocklev->deadlocks = 0;
   stocklev->s_transtatus = FATAL_SQLERROR;
   /* Input SQLDA */
   in_sqlda.sqln = 1;
   in_sqlda.sqld = 1;
   in_sqlda.sqlvar[0].sqltype = SQL_TYP_CHAR;
   in_sqlda.sqlvar[0].sqldata = (char *)stocklev;
   in_sqlda.sqlvar[0].sqllen = sizeof( struct stocklev_struct );
   in_sqlda.sqlvar[0].sqlind = NULL;

   EXEC SQL CALL "stklvs" USING DESCRIPTOR :in_sqlda;

   if (sqlca.sqlcode) {
      rc = sqlaintp(buf, 512, 78, &sqlca);
      fprintf(stderr,"\nSTOCKLEV dari failed with sqlcode =
%ld\n%s\n",sqlca.sqlcode, buf);
      s_debugit(stocklev, "sqlcode != 0");
      return(-1);
   }

   if (stocklev->s_transtatus <= FATAL_SQLERROR) {
      fprintf(stderr,"STOCKLEV transaction failed, transtatus = %d.\n",
            stocklev->s_transtatus);
      s_debugit(stocklev, "s_transtatus <= FATAL_SQLERROR");
      return(-1);
   }

   MOVEBACKI(stocklev, s_W_ID, w_id);
   MOVEBACKI(stocklev, s_D_ID, d_id);
   MOVEBACKI(stocklev, s_threshold, threshold);
   MOVEBACKI(stocklev, s_low_stock, low_stock);

   return (0);
}

/*---------------------------------------------------------------*/
/*       d_debugit                                               */
/*---------------------------------------------------------------*/
void d_debugit (struct delivery_struct *delivery_ptr, char *msg)
{
   FILE *debug_fp;
   char  debug_fn[25] = "/tmp/delivery.debug.out";
   int  i;
   if ((debug_fp = fopen(debug_fn, "a+")) == NULL)
   {
      return;
   }
   fprintf(debug_fp,"debugit information follows (%s)\n", msg);

fprintf(debug_fp,"================================================
===\n");
   fprintf(debug_fp,"delivery_struct {\n");
   fprintf(debug_fp,"\ts_W_ID         = %hd (%hX)\n",
         delivery_ptr->s_W_ID,delivery_ptr->s_W_ID);
   fprintf(debug_fp,"\ts_O_CARRIER_ID = %hd (%hX)\n",
         delivery_ptr->s_O_CARRIER_ID,delivery_ptr->s_O_CARRIER_ID);
   for (i=0;i<10;i++)
          fprintf(debug_fp,"\ts_O_ID[%d] = %hd (%hX)\n",
                i, delivery_ptr->s_O_ID[i],delivery_ptr->s_O_ID[i]);
   fprintf(debug_fp,"\ts_transtatus   = %hd (%hX)\n",
         delivery_ptr->s_transtatus,delivery_ptr->s_transtatus);
   fprintf(debug_fp,"\tdeadlocks      = %hd (%hX)\n",
         delivery_ptr->deadlocks,delivery_ptr->deadlocks);
   fprintf(debug_fp,"}\n\n");
   fclose(debug_fp);
}

/*---------------------------------------------------------------*/
/*       s_debugit                                               */
```

```
/*---------------------------------------------------------------*/
void s_debugit (struct stocklev_struct *stocklev_ptr, char *msg)
{
  FILE *debug_fp;
  char  debug_fn[35] = "/tmp/stklv.debug.out";
  if ((debug_fp = fopen(debug_fn, "a+")) == NULL)
  {
    return;
  }
  fprintf(debug_fp,"debugit information follows (%s)\n", msg);

fprintf(debug_fp,"=========================================
========\n");
  fprintf(debug_fp,"stocklev_struct {\n");
  fprintf(debug_fp,"\ts_W_ID         = %hd (%hX)\n",
        stocklev_ptr->s_W_ID,stocklev_ptr->s_W_ID);
  fprintf(debug_fp,"\ts_D_ID         = %hd (%hX)\n",
        stocklev_ptr->s_D_ID,stocklev_ptr->s_D_ID);
  fprintf(debug_fp,"\ts_threshold    = %hd (%hX)\n",
        stocklev_ptr->s_threshold,stocklev_ptr->s_threshold);
  fprintf(debug_fp,"\ts_low_stock    = %ld (%lX)\n",
        stocklev_ptr->s_low_stock,stocklev_ptr->s_low_stock);
  fprintf(debug_fp,"\ts_transtatus   = %hd (%hX)\n",
        stocklev_ptr->s_transtatus,stocklev_ptr->s_transtatus);
  fprintf(debug_fp,"\tdeadlocks      = %hd (%hX)\n",
        stocklev_ptr->deadlocks,stocklev_ptr->deadlocks);
  fprintf(debug_fp,"}\n\n");
  fclose(debug_fp);
}

/* Tuxedo */
tpsvrinit(argc, argv)
char **argv;
{
      char *p;
      char filename[200];
      int proc_no, count;
      struct utsname name;

      if ((p = getenv("TMPDIR")) == (char *)NULL) {
            userlog("TMPDIR environment variable not set\n");
            exit(1);
      }

      proc_no = (int)getpid();

      /* Get hostname of our machine  and create results file */
      uname( &name);
      strcpy(filename, p);
      sprintf(filename+strlen(filename), "/%s.del%d", name.nodename,
proc_no);
      delfile = fopen(filename, "w");
      if (delfile == NULL) {
            userlog("Cannot create file %s\n", filename);
      }

      return(init_all_tx());          /* Prepare transaction */
}

void
tpsvrdone()
{
      fclose(delfile);        /* Close results file */
}

DEL(rqst)
TPSVCINFO *rqst;
{
      if (delivery_tx(rqst) )
            tpreturn(TPFAIL, 0, rqst->data, sizeof(struct req_struct), 0);
      else
            tpreturn(TPSUCCESS, 0, rqst->data, sizeof(struct req_struct), 0)
;

}

STOCK(rqst)
TPSVCINFO *rqst;
{
      if (stocklevel_tx(rqst) )
            tpreturn(TPFAIL, 0, rqst->data, sizeof(struct stock_struct), 0);
      else
            tpreturn(TPSUCCESS, 0, rqst->data, sizeof(struct stock_struct), 0)
;

}

/*****************************************************************/
/*      File: delivery(s).sqc (Server code)                      */
/*****************************************************************/

#include "../include/tpcc.h"

void static sqlerror(char *msg, int ptat, struct sqlca *psqlca);
void static debugit(struct delivery_struct *, char *msg);

/*---------------------------------------------------------------*/
/*            DELIVERY Service                                   */
/*---------------------------------------------------------------*/
int deliverys (void *reserved1,
            void *reserved2,
            struct sqlda *pin_sqlda,
            struct sqlca *pca)
{

  static int tran_cntr=0;
  struct delivery_struct *delivery;

  EXEC SQL include sqlca;

  EXEC SQL BEGIN DECLARE SECTION;

  short  no_d_id,            w_id;
  short  o_carrier_id,       c_delivery_cnt;

  short  no_o_id_i;

  long    o_c_id;
  long   no_o_id;

  double c_balance,          total_amount;

  char ol_delivery_d[27],     curr_tmstmp[27];

  long   c_id, oltotal_amount;
  EXEC SQL END DECLARE SECTION;
```

```
    int i, rc;
    time_t   tlong;
    int      retry_count = -1;

    /* Retrieve the 'delivery' structure from the SQLDA */
    delivery =(struct delivery_struct *)pin_sqlda->sqlvar[0].sqldata;
    current_tmstmp(&curr_tmstmp[0]);
    w_id = delivery->s_W_ID;
    o_carrier_id = delivery->s_O_CARRIER_ID;

    ++tran_cntr;

  retry_tran:
        retry_count++;

    for (i=1; i <= DISTRICTS_PER_WAREHOUSE; i++) {

      /*-----------------*/
      /*  Read NEW_ORDER */
      /*-----------------*/

      no_d_id = i;        no_o_id_i = 0;

      EXEC SQL SELECT MIN(no_o_id)
        INTO :no_o_id:no_o_id_i
        FROM new_order
        WHERE no_w_id = :w_id
        AND no_d_id = :no_d_id;   /* return NULL for empty set */

      if (sqlca.sqlcode != 0)
      {
        if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
        sqlerror("SELECT new_order", 115, &sqlca);
        goto ferror;
      }

      if (no_o_id_i < 0)        /* Refer to clause 2.7.4.2, bullet 3 in spec.*/
      {                         /* Need to report if more than 1 or 1% of    */
                                /* transactions have no new_order rows       */
        delivery->s_O_ID[i-1] = 0;
        continue;
      }

      delivery->s_O_ID[i-1] = no_o_id;

      /*------------------------*/
      /*  Read & Update ORDER  */
      /*------------------------*/

      EXEC SQL BEGIN COMPOUND ATOMIC STATIC

        DELETE FROM new_order
        WHERE no_w_id = :w_id
        AND no_d_id = :no_d_id
        AND no_o_id = :no_o_id;

      UPDATE orders SET o_carrier_id = :o_carrier_id
        WHERE o_id = :no_o_id AND o_w_id = :w_id AND o_d_id =
:no_d_id;

      SELECT o_c_id INTO :o_c_id FROM orders
        WHERE o_ID = :no_o_id AND o_w_id = :w_id AND o_d_id = :no_d_id;

      SELECT SUM(ol_amount) INTO :oltotal_amount FROM order_line
        WHERE ol_w_id = :w_id AND ol_d_id = :no_d_id AND ol_o_id = :no_o_id;

      UPDATE ORDER_LINE SET ol_delivery_d = :curr_tmstmp
        WHERE ol_w_id = :w_id AND ol_d_id = :no_d_id AND ol_o_id = :no_o_id;

      END COMPOUND;

      if (sqlca.sqlcode < 0) {
        switch (sqlca.sqlerrd[LASTCODE]) {
         case 0:
          if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
          sqlerror("delete from NEW_ORDER",200, &sqlca);
          goto ferror;
         case 1:
          if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
          sqlerror("update ORDERS",208, &sqlca);
          goto ferror;
         case 2:
          if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
          sqlerror("SELECT FROM orders",216, &sqlca);
          goto ferror;
         case 3:
          if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
          sqlerror("select from ORDER_LINE",224, &sqlca);
          goto ferror;
         case 4:
          if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
          sqlerror("update ORDER_LINE",232, &sqlca);
          goto ferror;
         default:
          if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
          sqlerror("default case of compound SQL", 240, &sqlca);
          goto ferror;
        }
      }  /* end of processing for Compound SQL statement */
      total_amount = (double)oltotal_amount / (double)100.0;

      c_id = o_c_id;
      EXEC SQL UPDATE customer
        SET c_balance = c_balance + :total_amount,
        c_delivery_cnt = c_delivery_cnt + 1
          WHERE c_id = :c_id
          AND c_w_id = :w_id
          AND c_d_id = :no_d_id;
      if (sqlca.sqlcode != 0)
      {
        if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
        sqlerror("UPDATE customer",269, &sqlca);
        goto ferror;
      }

    } /* End of ... for (i=1; i <= DISTRICTS_PER_WAREHOUSE; i++)  */

    EXEC SQL commit;
    if (sqlca.sqlcode != 0) {
      if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
      sqlerror("COMMIT",317, &sqlca);
      goto ferror;
    }
```

```
    delivery->s_transtatus = TRAN_OK;
    rc = 0;
    goto mexit;

  ferror:
    delivery->s_transtatus = FATAL_SQLERROR;
    rc = -1;
    EXEC SQL rollback work;
    if (sqlca.sqlcode != 0)
      sqlerror("ROLLBACK FAILED", 666, &sqlca);
    debugit(delivery, "ferror code");

  mexit:
    delivery->deadlocks = retry_count;
    memcpy( pca, &sqlca, sizeof( struct sqlca ) );
    if ( sqlca.sqlcode < 0 )
      rc = SQLZ_DISCONNECT_PROC;
    else
      rc = SQLZ_HOLD_PROC;

    return(rc);
}

/*---------------------------------------------------------------*/
/*      sqlerror                                                 */
/*---------------------------------------------------------------*/
void static sqlerror(char *msg, int ptat, struct sqlca *psqlca)
{
    FILE *err_fp;
    char  err_fn[25] = "/tmp/delivery.err.out";
    int   err_ok;
    int j,k;

    err_fp = fopen(err_fn, "a+");
    fprintf(err_fp,"DELIVERY: %2d sqlcode: %4d %s\n", ptat, psqlca-
>sqlcode, msg);
    if (psqlca->sqlerrmc[0] != ' ' || psqlca->sqlerrmc[1] != ' ') {
      fprintf(err_fp,"DELIVERYS:slerrmc:  ");
      for(j = 0; j < 5; j++)
      {
        for(k = 0; k < 14; k++) fprintf(err_fp,"%x ", psqlca->sqlerrmc[j*10+k]);
         fprintf(err_fp,"    ");
        for(k = 0; k < 14; k++) fprintf(err_fp,"%c",  psqlca->sqlerrmc[j*10+k]);
         fprintf(err_fp,"\n");
         if (j < 4) fprintf(err_fp,"          ");
      }
    }

    fprintf(err_fp,"DELIVERYS:sqlerrp:  ");
    for(j = 0; j < 8; j++)    fprintf(err_fp,"%c", psqlca->sqlerrp[j]);
    fprintf(err_fp,"\n");

    fprintf(err_fp,"DELIVERYS:sqlerrd: ");
    for(j = 0; j < 6; j++)    fprintf(err_fp," %d", psqlca->sqlerrd[j]);
    fprintf(err_fp,"\n");

    if (psqlca->sqlwarn[0] != ' ') {
      fprintf(err_fp,"DELIVERYS:sqlwarn: ");
      for(j = 0; j < 8; j++)    fprintf(err_fp,"%c ", psqlca->sqlwarn[j]);
      fprintf(err_fp,"\n");
    }

    fprintf(err_fp,"\n");
    fclose(err_fp);
}

/*---------------------------------------------------------------*/
/*      debugit                                                  */
/*---------------------------------------------------------------*/
void debugit (struct delivery_struct *delivery_ptr, char *msg)
{
    FILE *debug_fp;
    char  debug_fn[25] = "/tmp/delivery.debug.out";
    int  i;
    char  timeStamp[27];
    current_tmstmp(&timeStamp[0]);
    timeStamp[19] = (char)NULL;
    if ((debug_fp = fopen(debug_fn, "a+")) == NULL)
    {
      return;
    }
    fprintf(debug_fp,"debugit information follows %s (%s)\n", timeStamp,
msg);

fprintf(debug_fp,"=============================================
=======\n");
    fprintf(debug_fp,"delivery_struct {\n");
    fprintf(debug_fp,"\ts_W_ID         = %hd (%hX)\n",
          delivery_ptr->s_W_ID,delivery_ptr->s_W_ID);
    fprintf(debug_fp,"\ts_O_CARRIER_ID = %hd (%hX)\n",
          delivery_ptr->s_O_CARRIER_ID,delivery_ptr->s_O_CARRIER_ID);
    for (i=0;i<10;i++)
          fprintf(debug_fp,"\ts_O_ID[%d] = %hd (%hX)\n",
                i, delivery_ptr->s_O_ID[i],delivery_ptr->s_O_ID[i]);
    fprintf(debug_fp,"\ts_transtatus   = %hd (%hX)\n",
          delivery_ptr->s_transtatus,delivery_ptr->s_transtatus);
    fprintf(debug_fp,"\tdeadlocks      = %hd (%hX)\n",
          delivery_ptr->deadlocks,delivery_ptr->deadlocks);
    fprintf(debug_fp,"}\n\n");
    fclose(debug_fp);
}

/*****************************************************************/
/*      File: neword(s).sqc (Server code)                        */
/*****************************************************************/

#include "../include/tpcc.h"
#include <stdlib.h>

void static sqlerror(char *msg, int ptat, struct sqlca *psqlca);
void static nsqlerror(char *msg, int ptat);
void static debugit(struct neword_struct *, char *);

int newords (void *reserved1,
             void *reserved2,
             struct sqlda *pin_sqlda,
             struct sqlca *pca)
{

    struct neword_struct *neword;
    int duplicate_items = 0;

    EXEC SQL include sqlca;

    int     retry_count = -1;
```

```
EXEC SQL BEGIN DECLARE SECTION;
short   w_id,           d_id;
short   supply_w_id,    ol_quantity,        s_quantity;
short   ol_ln,          OLITEMS,            olcount;
short   s_remote_cnt,       s_order_cnt;

short   i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12, i13, i14;

char    c_last[17],     c_credit[3];

double  c_discount;
double  ware_tax,       dist_tax;

long    o_id,           next_o_id;
long    s_id,           ol_i_id,            s_ytd;

long    id0, id1, id2,  id3,  id4,  id5,  id6,  id7;
long    id8, id9, id10, id11, id12, id13, id14;

char    dist[25],       dist1[25],          dist2[25] ;
char    dist3[25],      dist4[25],          dist5[25] ;
char    dist6[25],      dist7[25],          dist8[25] ;
char    dist9[25],      dist10[25],         item_name[25] ;
char    i_data[51],     s_data[51] ;
char    curr_tmstmp[27];
short   allLocal;

long    o_c_id;
long    c_id;
long    iol_amount;
long    item_price;
EXEC SQL END DECLARE SECTION;

double  ol_amount;
double  i_price;
char    bad_item = 'n';
int     i, j, item_cnt, item_idx, isRemote, rc;
int     item_processed[15]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

/*---------------------------------------*/
/*  Declare CURSORS                      */
/*---------------------------------------*/

EXEC SQL DECLARE Item_Stock_cursor CURSOR FOR
  WITH listtable ( list ) as
  ( VALUES  :id0, :id1:i1, :id2:i2, :id3:i3, :id4:i4,
    :id5:i5, :id6:i6, :id7:i7, :id8:i8, :id9:i9,
    :id10:i10, :id11:i11, :id12:i12, :id13:i13, :id14:i14 )
  SELECT i_id, i_price, i_name, i_data, s_quantity,
  s_data, s_order_cnt, s_ytd,
  s_dist_01, s_dist_02, s_dist_03, s_dist_04, s_dist_05,
  s_dist_06, s_dist_07, s_dist_08, s_dist_09, s_dist_10
    FROM item, stock, listtable
    WHERE i_id = listtable.list
    AND s_i_id = i_id AND s_w_id = :w_id FOR FETCH ONLY;

EXEC SQL DECLARE Item_cursor CURSOR FOR
  SELECT i_id, i_price, i_name, i_data
  FROM item
  WHERE i_id in (:id0, :id1:i1, :id2:i2, :id3:i3, :id4:i4,
                 :id5:i5, :id6:i6, :id7:i7, :id8:i8, :id9:i9,
                 :id10:i10, :id11:i11, :id12:i12, :id13:i13, :id14:i14) FOR
FETCH ONLY;

EXEC SQL DECLARE Stock_cursor CURSOR FOR
  SELECT s_quantity, s_data, s_order_cnt,
  s_remote_cnt, s_ytd,
  s_dist_01, s_dist_02, s_dist_03, s_dist_04, s_dist_05,
  s_dist_06, s_dist_07, s_dist_08, s_dist_09, s_dist_10
    FROM stock
    WHERE s_w_id = :supply_w_id
    AND s_i_id = :ol_i_id
    FOR UPDATE OF s_quantity, s_order_cnt, s_remote_cnt, s_ytd;

EXEC SQL
  DECLARE dist_cur CURSOR FOR
  SELECT d_tax, d_next_o_id FROM district
  WHERE d_id = :d_id AND d_w_id = :w_id
  FOR UPDATE OF d_next_o_id;

/* Retrieve the 'neword' structure from the SQLDA */
neword =(struct neword_struct *)pin_sqlda->sqlvar[0].sqldata;
duplicate_items = neword->duplicate_items;

item_idx = -1;
current_tmstmp(&curr_tmstmp[0]);
curr_tmstmp[19] = (char)NULL;
allLocal = neword->s_all_local;
OLITEMS = neword->s_O_OL_CNT;
d_id = neword->s_D_ID;
w_id = neword->s_W_ID;
c_id = neword->s_C_ID;
o_c_id = c_id;

retry_tran:
retry_count++;
if (duplicate_items && retry_count > 0)
  for (i=0; i<15; i++) item_processed[i]=0;
/*---------------------------------------------------------------*/
/* Use CURSOR to SELECT DISTRICT information and and then to     */
/* UPDATE DISTRICT.                                              */
/*---------------------------------------------------------------*/

EXEC SQL OPEN dist_cur;

if (sqlca.sqlcode != 0) {
  if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
  sqlerror("OPEN dis_cur",185, &sqlca);
  goto ferror;
}

EXEC SQL
  FETCH dist_cur INTO
  :dist_tax, :next_o_id;

if (sqlca.sqlcode != 0) {
  if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
  sqlerror("FETCH dist_cur",206, &sqlca);
  goto ferror;
}

o_id = next_o_id;
neword->s_O_ID = o_id;
next_o_id++;

EXEC SQL
```

```
    UPDATE district
    SET d_next_o_id = :next_o_id
    WHERE CURRENT OF dist_cur;

  if (sqlca.sqlcode != 0) {
    if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
    sqlerror("UPDATE dist_cur",234, &sqlca);
    goto ferror;
  }

  i1 = i2 = i3 = i4 = -1;
  i5 = i6 = i7 = i8 = i9 = -1;
  i10 = i11 = i12 = i13 = i14 = -1;
  switch (OLITEMS) {
   case 15:
     id14 = neword->item[14].s_OL_I_ID;
     i14 = 0;
   case 14:
     id13 = neword->item[13].s_OL_I_ID;
     i13 = 0;
   case 13:
     id12 = neword->item[12].s_OL_I_ID;
     i12 = 0;
   case 12:
     id11 = neword->item[11].s_OL_I_ID;
     i11 = 0;
   case 11:
     id10 = neword->item[10].s_OL_I_ID;
     i10 = 0;
   case 10:
     id9 = neword->item[9].s_OL_I_ID;
     i9 = 0;
   case  9:
     id8 = neword->item[8].s_OL_I_ID;
     i8 = 0;
   case  8:
     id7 = neword->item[7].s_OL_I_ID;
     i7 = 0;
   case  7:
     id6 = neword->item[6].s_OL_I_ID;
     i6 = 0;
   case  6:
     id5 = neword->item[5].s_OL_I_ID;
     i5 = 0;
   case  5:
     id4 = neword->item[4].s_OL_I_ID;
     i4 = 0;
   case  4:
     id3 = neword->item[3].s_OL_I_ID;
     i3 = 0;
   case  3:
     id2 = neword->item[2].s_OL_I_ID;
     i2 = 0;
   case  2:
     id1 = neword->item[1].s_OL_I_ID;
     i1 = 0;
   case  1:
     id0 = neword->item[0].s_OL_I_ID;
     break;
   default:
     nsqlerror("default switch on OLITEMS for local",293);
     goto ferror;
  } /* switch */

  if (allLocal)
  {
    EXEC SQL OPEN Item_Stock_cursor;
    if (sqlca.sqlcode != 0) {
      if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
      sqlerror("open Item_Stock_cursor",305, &sqlca);
      goto ferror;
    }
  }
  else /* REMOTE (not allLocal) */
  {
    EXEC SQL OPEN Item_cursor;
    if (sqlca.sqlcode != 0) {
      if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
      sqlerror("open Item_cursor",319, &sqlca);
      goto ferror;
    }
  } /* end if (allLocal) ... else ... */

  /*-------------------------------------------------------*/
  /*              Process lines loop                       */
  /*-------------------------------------------------------*/

  for (item_cnt = 0; item_cnt < OLITEMS;)
  {
    if (allLocal)
    {
      EXEC SQL FETCH Item_Stock_cursor INTO
        :ol_i_id, :item_price, :item_name, :i_data,
        :s_quantity, :s_data, :s_order_cnt, :s_ytd,
        :dist1, :dist2, :dist3, :dist4, :dist5,
        :dist6, :dist7, :dist8, :dist9, :dist10;

      if (sqlca.sqlcode <  0) {
        if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
        sqlerror("FETCH Item_Stock_cursor",362, &sqlca);
        goto ferror;
      } else if (sqlca.sqlcode > 0) {
        bad_item = 'y';   break;
      }
    }
    else  /* REMOTE (not allLocal) */
    {
      if (duplicate_items) {
        j = item_idx + 1;
        if (j < OLITEMS && j > 0 && neword->item[j].s_OL_I_ID == ol_i_id)
goto rdup;
      }
      EXEC SQL FETCH Item_cursor INTO
        :ol_i_id, :item_price, :item_name, :i_data;
      if (sqlca.sqlcode <  0) {
        if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
        sqlerror("FETCH Item_cursor",385, &sqlca);
        goto ferror;
      } else if (sqlca.sqlcode > 0) {
        bad_item = 'y';   break;
      }
    } /* end if (allLocal) ... else ... */
    i_price = (double) item_price / (double) 100.0;

   rdup:
    /*-------------------------------------------------------*/
```

```
    /*  Match fetched stock with order line and move data  */
    /*--------------------------------------------------------*/
    if (neword->item[item_cnt].s_OL_I_ID == ol_i_id) {
      item_idx = item_cnt;
    } else {
      for (i = OLITEMS-1; i > 0 ; i--) {
        if (neword->item[i].s_OL_I_ID == ol_i_id &&
            ! (duplicate_items && item_processed[i]))
        {
          item_idx = i;
          item_processed[i]=1;
          break;
        }
      }
    }

    ++item_cnt;
    supply_w_id = neword->item[item_idx].s_OL_SUPPLY_W_ID;
    if (!allLocal)
    {
      EXEC SQL OPEN Stock_cursor;
      if (sqlca.sqlcode != 0) {
        if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
        sqlerror("OPEN Stock_cursor",409, &sqlca);
        goto ferror;
      }
      EXEC SQL FETCH Stock_cursor INTO
        :s_quantity, :s_data, :s_order_cnt,
        :s_remote_cnt, :s_ytd,
        :dist1, :dist2, :dist3, :dist4, :dist5,
        :dist6, :dist7, :dist8, :dist9, :dist10;
      if (sqlca.sqlcode != 0) {
        if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
        sqlerror("FETCH Stock_cursor",422, &sqlca);
        goto ferror;
      }
    } /* end if (!allLocal) */
    s_ytd += neword->item[item_idx].s_OL_QUANTITY;
    s_quantity -= neword->item[item_idx].s_OL_QUANTITY;
    switch (d_id) {
     case 1:  strncpy(dist, dist1, 24);   break;
     case 2:  strncpy(dist, dist2, 24);   break;
     case 3:  strncpy(dist, dist3, 24);   break;
     case 4:  strncpy(dist, dist4, 24);   break;
     case 5:  strncpy(dist, dist5, 24);   break;
     case 6:  strncpy(dist, dist6, 24);   break;
     case 7:  strncpy(dist, dist7, 24);   break;
     case 8:  strncpy(dist, dist8, 24);   break;
     case 9:  strncpy(dist, dist9, 24);   break;
     case 10: strncpy(dist, dist10, 24);  break;
     default:
      nsqlerror("default switch on d_id",444);
      goto ferror;
    }
    dist[24] = 0;
    if (s_quantity < 10) s_quantity += 91;
    strcpy(neword->item[item_idx].s_I_NAME, item_name);
    neword->item[item_idx].s_I_PRICE = i_price;

    neword->item[item_idx].s_OL_AMOUNT = neword-
>item[item_idx].s_I_PRICE *
      neword->item[item_idx].s_OL_QUANTITY;

    if (strstr(i_data, "ORIGINAL") != NULL &&
       strstr(s_data, "ORIGINAL") != NULL)
      neword->item[item_idx].s_brand_generic = 'B';
    else
      neword->item[item_idx].s_brand_generic = 'G';

    neword->item[item_idx].s_S_QUANTITY = s_quantity;
    ol_ln = item_idx + 1;
    ol_quantity = neword->item[item_idx].s_OL_QUANTITY;
    ol_amount = neword->item[item_idx].s_OL_AMOUNT;
    iol_amount = ol_amount * 100.0;

    s_order_cnt++;
    if (allLocal) {
      EXEC SQL BEGIN COMPOUND ATOMIC STATIC

        UPDATE stock
        SET s_quantity = :s_quantity, s_order_cnt = :s_order_cnt,
        s_ytd = :s_ytd
          WHERE s_w_id =  :w_id AND s_i_id = :ol_i_id;

      INSERT INTO order_line VALUES
        (:o_id, :d_id, :w_id, :ol_ln, :ol_i_id,
         :supply_w_id, '0001-01-01-00.00.01.000000',
         :ol_quantity, :iol_amount, :dist);

      END COMPOUND;

      if (sqlca.sqlcode != 0) {
        switch (sqlca.sqlerrd[LASTCODE]) {
         case 0:
          if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
          sqlerror("UPDATE stock",490, &sqlca);
          goto ferror;
         case 1:
          if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
          sqlerror("INSERT INTO order_line",498, &sqlca);
          goto ferror;
         default:
          if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
          sqlerror("default case of compound sql",506, &sqlca);
          goto ferror;
        }
      }
    } else { /* REMOTE (not allLocal) */
      isRemote = (supply_w_id == w_id) ? 0 : 1;
      s_remote_cnt += isRemote;
      EXEC SQL
        UPDATE STOCK
        SET s_quantity = :s_quantity, s_order_cnt= :s_order_cnt,
        s_remote_cnt = :s_remote_cnt, s_ytd = :s_ytd
          WHERE CURRENT OF Stock_cursor;
      if (sqlca.sqlcode != 0) {
        if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
        sqlerror("UPDATE stock",524, &sqlca);
        goto ferror;
      }

      EXEC SQL CLOSE Stock_cursor;
      if (sqlca.sqlcode != 0) {
        if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
        sqlerror("CLOSE Stock_cursor",534, &sqlca);
        goto ferror;
```

```
      }
      EXEC SQL
        INSERT INTO order_line VALUES
        (:o_id, :d_id, :w_id, :ol_ln, :ol_i_id,
         :supply_w_id, '0001-01-01-00.00.01.000000',
         :ol_quantity, :iol_amount, :dist);
      if (sqlca.sqlcode != 0) {
        if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
        sqlerror("INSERT INTO order_line",549, &sqlca);
        goto ferror;
      }
    } /* end if (allLocal) ... else ... */
}   /* end for (item_cnt = 0; item_cnt < OLITEMS;) */

if (bad_item == 'y') {
  neword->s_transtatus = INVALID_ITEM;
  rc = -1;

  EXEC SQL BEGIN COMPOUND ATOMIC STATIC

    INSERT INTO orders VALUES
    (:o_id, :o_c_id, :d_id, :w_id, :curr_tmstmp, 0, :OLITEMS, :allLocal);

  INSERT INTO new_order VALUES (:o_id, :d_id, :w_id);

  SELECT w_tax, c_discount, c_last, c_credit
    INTO :ware_tax, :c_discount, :c_last, :c_credit
    FROM warehouse, customer
    WHERE w_id = :w_id
    AND c_id = :c_id
    AND c_w_id = :w_id
    AND c_d_id = :d_id;

  END COMPOUND;
  if (sqlca.sqlcode != 0) {
    if(sqlca.sqlcode == DEADLOCK) goto mexit;
    switch (sqlca.sqlerrd[LASTCODE]) {
     case 0:
       sqlerror("insert into ORDERS",593, &sqlca);
       goto ferror;
     case 1:
       sqlerror("insert into NEW_ORDER",601, &sqlca);
       goto ferror;
     case 2:
       sqlerror("SELECT WAREHOUSE & CUSTOMER",609, &sqlca);
       goto ferror;
     default:
       sqlerror("default case of compound sql",617, &sqlca);
       goto ferror;
    }
  }

  EXEC SQL rollback work ;
  if (sqlca.sqlcode != 0) {
    sqlerror("rollback",636, &sqlca);
    goto ferror;
  }

  goto mexit;
} else {

  /*---------------------------------------*/
  /*  Read WAREHOUSE and CUSTOMER          */
  /*---------------------------------------*/

  EXEC SQL BEGIN COMPOUND ATOMIC STATIC

    INSERT INTO orders VALUES
    (:o_id, :o_c_id, :d_id, :w_id, :curr_tmstmp, 0, :OLITEMS, :allLocal);

  INSERT INTO new_order VALUES (:o_id, :d_id, :w_id);

  SELECT w_tax, c_discount, c_last, c_credit
    INTO :ware_tax, :c_discount, :c_last, :c_credit
    FROM warehouse, customer
    WHERE w_id = :w_id
    AND c_id = :c_id
    AND c_w_id = :w_id
    AND c_d_id = :d_id;

  commit;

  END COMPOUND;

  if (sqlca.sqlcode != 0) {
    switch (sqlca.sqlerrd[LASTCODE]) {
     case 0:
       if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
       sqlerror("insert into ORDERS",789, &sqlca);
       goto ferror;
     case 1:
       if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
       sqlerror("insert into NEW_ORDER",797, &sqlca);
       goto ferror;
     case 2:
       if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
       sqlerror("SELECT WAREHOUSE & CUSTOMER",805, &sqlca);
       goto ferror;
     case 3:
       if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
       sqlerror("COMMIT",813, &sqlca);
       goto ferror;
     default:
       if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
       sqlerror("default case of compound sql",821, &sqlca);
       goto ferror;
    }
  }

  neword->s_C_DISCOUNT = c_discount;
  neword->s_W_TAX = ware_tax;
  neword->s_D_TAX = dist_tax;
  strcpy(neword->s_C_LAST, c_last);
  strcpy(neword->s_C_CREDIT, c_credit);
  memcpy(neword->s_O_ENTRY_D, curr_tmstmp, 19);
  neword->s_O_ENTRY_D[19] = (char)NULL;

}

/*-----------------------------------------------*/
/*  Return to client                             */
/*-----------------------------------------------*/
neword->s_transtatus = TRAN_OK;
rc = 0;
goto mexit;
```

```
 ferror:
   neword->s_transtatus = FATAL_SQLERROR;
   rc = -1;
   EXEC SQL rollback work;
   if (sqlca.sqlcode != 0)
      sqlerror("ROLLBACK FAILED", 666, &sqlca);
   debugit(neword, "ferror code");

 mexit:
   neword->deadlocks = retry_count;
   memcpy( pca, &sqlca, sizeof( struct sqlca ) );
   if ( sqlca.sqlcode < 0 )
      rc = SQLZ_DISCONNECT_PROC;
   else
      rc = SQLZ_HOLD_PROC;

   return(rc);
}

/*---------------------------------------------------------------*/
/*       sqlerror                                                */
/*---------------------------------------------------------------*/
void static sqlerror(char *msg, int ptat, struct sqlca *psqlca)
{
   FILE *err_fp;
   char  err_fn[25] = "/tmp/neword.err.out";
   int j,k;

   err_fp = fopen(err_fn, "a+");
   fprintf(err_fp,"NEWORD: %2d sqlcode: %4d %s\n", ptat, psqlca-
>sqlcode, msg);
   if (psqlca->sqlerrmc[0] != ' ' || psqlca->sqlerrmc[1] != ' ') {
      fprintf(err_fp,"NEWORDS:slerrmc:  ");
      for(j = 0; j < 5; j++)
      {
         for(k = 0; k < 14; k++) fprintf(err_fp,"%x ", psqlca-
>sqlerrmc[j*10+k]);
         fprintf(err_fp,"    ");
         for(k = 0; k < 14; k++) fprintf(err_fp,"%c",  psqlca-
>sqlerrmc[j*10+k]);
         fprintf(err_fp,"\n");
         if (j < 4) fprintf(err_fp,"          ");
      }
   }

   fprintf(err_fp,"NEWORDS:sqlerrp:  ");
   for(j = 0; j < 8; j++)    fprintf(err_fp,"%c", psqlca->sqlerrp[j]);
   fprintf(err_fp,"\n");

   fprintf(err_fp,"NEWORDS:sqlerrd: ");
   for(j = 0; j < 6; j++)    fprintf(err_fp," %d", psqlca->sqlerrd[j]);
   fprintf(err_fp,"\n");

   if (psqlca->sqlwarn[0] != ' ') {
      fprintf(err_fp,"NEWORDS:sqlwarn: ");
      for(j = 0; j < 8; j++)    fprintf(err_fp,"%c ", psqlca->sqlwarn[j]);
      fprintf(err_fp,"\n");
   }

   fprintf(err_fp,"\n");
   fclose(err_fp);
}

/*---------------------------------------------------------------*/
/*       non-sql error                                           */
/*---------------------------------------------------------------*/
void static nsqlerror(char *msg, int ptat)
{
   FILE *err_fp;
   char  err_fn[25] = "/tmp/neword.err.out";

   err_fp = fopen(err_fn, "a+");
   fprintf(err_fp,"NEWORD: %2d %s\n", ptat,msg);
   fclose(err_fp);
}

/*---------------------------------------------------------------*/
/*       debugit                                                 */
/*---------------------------------------------------------------*/
void debugit (struct neword_struct *neword_ptr, char *msg)
{
   FILE *debug_fp;
   char  debug_fn[25] = "/tmp/neword.debug.out";
   char  timeStamp[27];
   int   j, items;
   current_tmstmp(&timeStamp[0]);
   timeStamp[19] = (char)NULL;
   if ((debug_fp = fopen(debug_fn, "a+")) == NULL)
   {
      return;
   }
   fprintf(debug_fp,"debugit information follows %s (%s)\n", timeStamp, msg);

fprintf(debug_fp,"==============================================
===\n");
   fprintf(debug_fp,"neword_struct {\n");
   fprintf(debug_fp,"\ts_W_ID         = %hd (%hX)\n",
         neword_ptr->s_W_ID,neword_ptr->s_W_ID);
   fprintf(debug_fp,"\ts_D_ID         = %hd (%hX)\n",
         neword_ptr->s_D_ID,neword_ptr->s_D_ID);
   fprintf(debug_fp,"\ts_C_ID         = %ld (%lX)\n",
         neword_ptr->s_C_ID,neword_ptr->s_C_ID);
   fprintf(debug_fp,"\ts_C_LAST       = %s\n",
         neword_ptr->s_C_LAST);
   fprintf(debug_fp,"\ts_C_CREDIT     = %s\n",
         neword_ptr->s_C_CREDIT);
   fprintf(debug_fp,"\ts_C_DISCOUNT   = %0.4f\n",
         neword_ptr->s_C_DISCOUNT);
   fprintf(debug_fp,"\ts_O_OL_CNT     = %hd (%hX)\n",
         neword_ptr->s_O_OL_CNT,neword_ptr->s_O_OL_CNT);
   fprintf(debug_fp,"\ts_O_ID         = %ld (%lX)\n",
         neword_ptr->s_O_ID,neword_ptr->s_O_ID);
   fprintf(debug_fp,"\ts_O_ENTRY_D    = %s\n",
         neword_ptr->s_O_ENTRY_D);
   fprintf(debug_fp,"\ts_status_line  = %s\n",
         neword_ptr->s_status_line);
   fprintf(debug_fp,"\ts_total_amount = %0.2f\n",
         neword_ptr->s_total_amount);
   fprintf(debug_fp,"\ts_all_local    = %hd (%hX)\n",
         neword_ptr->s_all_local,neword_ptr->s_all_local);
   fprintf(debug_fp,"\ts_transtatus   = %hd (%hX)\n",
         neword_ptr->s_transtatus,neword_ptr->s_transtatus);
   fprintf(debug_fp,"\ts_W_TAX        = %0.4f\n",
         neword_ptr->s_W_TAX);
   fprintf(debug_fp,"\ts_D_TAX        = %0.4f\n",
```

```
        neword_ptr->s_D_TAX);
  fprintf(debug_fp,"\tduplicate_items= %hd (%hX)\n",
        neword_ptr->duplicate_items,neword_ptr->duplicate_items);
  fprintf(debug_fp,"\tdeadlocks      = %hd (%hX)\n",
        neword_ptr->deadlocks,neword_ptr->deadlocks);
  fprintf(debug_fp,"\titems {\n");
  items = neword_ptr->s_O_OL_CNT;
  for (j=0; j<items; j++) {
    fprintf(debug_fp,"\t\ts_OL_SUPPLY_W_ID[%d] = %hd (%hX)\n",
          j,neword_ptr->item[j].s_OL_SUPPLY_W_ID,neword_ptr-
>item[j].s_OL_SUPPLY_W_ID);
    fprintf(debug_fp,"\t\ts_OL_I_ID[%d]       = %ld (%lX)\n",
          j,neword_ptr->item[j].s_OL_I_ID,neword_ptr->item[j].s_OL_I_ID);
    fprintf(debug_fp,"\t\ts_I_NAME[%d]        = %s\n",
          j,neword_ptr->item[j].s_I_NAME);
    fprintf(debug_fp,"\t\ts_OL_QUANTITY[%d]   = %hd (%hX)\n",
          j,neword_ptr->item[j].s_OL_QUANTITY,neword_ptr-
>item[j].s_OL_QUANTITY);
    fprintf(debug_fp,"\t\ts_S_QUANTITY[%d]    = %hd (%hX)\n",
          j,neword_ptr->item[j].s_S_QUANTITY,neword_ptr-
>item[j].s_S_QUANTITY);
    fprintf(debug_fp,"\t\ts_brand_generic[%d] = %c\n",
          j,neword_ptr->item[j].s_brand_generic);
    fprintf(debug_fp,"\t\ts_I_PRICE[%d]       = %0.2f\n",
          j,neword_ptr->item[j].s_I_PRICE);
    fprintf(debug_fp,"\t\ts_OL_AMOUNT[%d]     = %0.2f\n\n",
          j,neword_ptr->item[j].s_OL_AMOUNT);
  }
  fprintf(debug_fp,"\t}\n}\n\n");
  fclose(debug_fp);
}

/*************************************************************************/
/*      File: ord(s).sqc (Server code)                                   */
/*************************************************************************/

#include "../include/tpcc.h"

void static sqlerror(char *msg, int ptat, struct sqlca *psqlca);
void static debugit(struct ordstat_struct *, char *);

/*-----------------------------------------------------------------*/
/*            ORDSTAT Standard Service                             */
/*-----------------------------------------------------------------*/
int ords (void *reserved1,
        void *reserved2,
        struct sqlda *pin_sqlda,
        struct sqlca *pca)
{
#define  MAXCLAST   3000

  struct ordstat_struct *ordstat;
  EXEC SQL include sqlca;

  EXEC SQL BEGIN DECLARE SECTION;
  long   o_c_id;
  long   tmp_c_id;
  short  w_id,              d_id;
  short  o_carrier_id,      ol_supply_w_id,      ol_quantity;
  char   c_first[17],       c_middle[3],         c_last[17];
  char   o_entry_d[27],     ol_delivery_d[27];
  long   iol_amount;
  double c_balance;
  long   ol_i_id,           o_id;

  long    clast_count;
  EXEC SQL END DECLARE SECTION;

  double  ol_amount;
  int     i, rc;
  int     retry_count = -1;
  int clast_idx;

  EXEC SQL DECLARE read_clast_cur CURSOR FOR
    SELECT  c_id, c_first
    FROM customer
    WHERE c_last = :c_last
    AND   c_w_id = :w_id
    AND   c_d_id = :d_id
    ORDER BY c_first
    FOR FETCH ONLY;

  EXEC SQL DECLARE read_order_cur CURSOR FOR
    SELECT o_id, o_entry_d, o_carrier_id
    FROM orders
    WHERE o_w_id = :w_id AND o_d_id = :d_id AND o_c_id = :o_c_id
    ORDER BY o_id DESC FOR FETCH ONLY;

  EXEC SQL DECLARE read_orderline_cur CURSOR FOR
    SELECT ol_i_id, ol_supply_w_id, ol_quantity, ol_amount, ol_delivery_d
    FROM order_line
    WHERE ol_w_id = :w_id AND ol_d_id = :d_id AND ol_o_id = :o_id FOR
FETCH ONLY;

  ordstat =(struct ordstat_struct *)pin_sqlda->sqlvar[0].sqldata;

  w_id = ordstat->s_W_ID;        d_id = ordstat->s_D_ID;

 retry_tran:
      retry_count++;

  /*-----------------------------------------------*/
  /*  Read CUSTOMER using C_LAST                   */
  /*-----------------------------------------------*/
  if (ordstat->s_C_ID == 0) {
    strcpy(c_last, ordstat->s_C_LAST);

    EXEC SQL OPEN read_clast_cur;
    if (sqlca.sqlcode != 0) {
      if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
      sqlerror("OPEN CURSOR read_clast_cur",96, &sqlca);
      goto ferror;
    }

    i = 0;
    do {
      EXEC SQL FETCH read_clast_cur
        INTO :tmp_c_id, :c_first;
      if (sqlca.sqlcode != 0) {
        if (sqlca.sqlcode < 0 || (sqlca.sqlcode == 100 && i == 0)) {
          if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
          sqlerror("FETCH CURSOR read_clast_cur",107, &sqlca);
          goto ferror;
        }
      }
      else {
```

```
            c_id[i] = tmp_c_id;
            i++;
          }
        } while (sqlca.sqlcode == 0);

        /*-------------------------*/
        /* select round-up n/2 row */
        /*-------------------------*/
        i = (int)((i+1)/2) - 1;
        ordstat->s_C_ID = c_id[i];
      }  /* end if (ordstat->s_C_ID == 0)   */

      /*-------------------------------------------------*/
      /*  Read CUSTOMER using C_ID                       */
      /*-------------------------------------------------*/

      tmp_c_id = ordstat->s_C_ID;

      EXEC SQL SELECT c_first, c_middle, c_last, c_balance
        INTO :c_first, :c_middle, :c_last, :c_balance
        FROM customer
        WHERE c_id = :tmp_c_id
        AND c_w_id = :w_id
        AND c_d_id = :d_id;

      if (sqlca.sqlcode != 0) {
        if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
        sqlerror("SELECT customer via c_id'",140, &sqlca);
        goto ferror;
      }

    FCUST:
      strcpy( ordstat->s_C_FIRST, c_first );
      strcpy( ordstat->s_C_LAST, c_last);
      strcpy( ordstat->s_C_MIDDLE, c_middle );
      ordstat->s_C_BALANCE = c_balance;

      /*-------------------------------------------------------*/
      /*  Read newest ORDER                                    */
      /*-------------------------------------------------------*/

      o_c_id = ordstat->s_C_ID;
      EXEC SQL OPEN read_order_cur;
      if (sqlca.sqlcode != 0) {
        if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
        sqlerror("OPEN CURSOR read_order_cur",156, &sqlca);
        goto ferror;
      }

      EXEC SQL FETCH read_order_cur
        INTO :o_id, :o_entry_d, :o_carrier_id;
      if (sqlca.sqlcode != 0) {
        if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
        sqlerror("FETCH CURSOR read_order_cur",171, &sqlca);
        goto ferror;
      }

      ordstat->s_W_ID = w_id;
      ordstat->s_D_ID = d_id;
      ordstat->s_O_ID = o_id;
      strcpy(ordstat->s_O_ENTRY_D, o_entry_d);
      ordstat->s_O_CARRIER_ID = o_carrier_id;

      /*-------------------------------------------------*/
      /*  Read ORDER_LINEs                               */
      /*-------------------------------------------------*/

      EXEC SQL OPEN read_orderline_cur;
      if (sqlca.sqlcode != 0) {
        if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
        sqlerror("OPEN CURSOR read_orderline_cur",194, &sqlca);
        goto ferror;
      }

      i=0;
      do {

        EXEC SQL FETCH read_orderline_cur
          INTO :ol_i_id, :ol_supply_w_id, :ol_quantity, :iol_amount,
          :ol_delivery_d;
        if (sqlca.sqlcode != 0 ) {
          if (sqlca.sqlcode < 0 ) {
            if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
            sqlerror("FETCH CURSOR read_orderline_cur",209, &sqlca);
            goto ferror;
          }
        } else {
          ol_amount = (double)iol_amount / (double)100.0;
          ordstat->item[i].s_OL_I_ID = ol_i_id;
          ordstat->item[i].s_OL_SUPPLY_W_ID = ol_supply_w_id;
          ordstat->item[i].s_OL_QUANTITY = ol_quantity;
          ordstat->item[i].s_OL_AMOUNT = ol_amount;
          strcpy(ordstat->item[i].s_OL_DELIVERY_D, ol_delivery_d);
          i++;
        }
      } while (sqlca.sqlcode == 0);

      ordstat->s_ol_cnt = i;

      /*---------------------------------------*/
      /* COMMIT transaction */
      /*---------------------------------------*/

      EXEC SQL COMMIT;
      if (sqlca.sqlcode != 0) {
        if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
        sqlerror("COMMIT",231, &sqlca);
        goto ferror;
      }

      ordstat->s_transtatus = TRAN_OK;
      rc = 0;
      goto mexit;

    ferror:
      rc = -1;
      ordstat->s_transtatus = FATAL_SQLERROR;
      EXEC SQL rollback work;
      if (sqlca.sqlcode != 0)
        sqlerror("ROLLBACK FAILED", 666, &sqlca);
      debugit(ordstat, "ferror code");

    mexit:
      ordstat->deadlocks = retry_count;
      memcpy( pca, &sqlca, sizeof( struct sqlca ) );
      if ( sqlca.sqlcode < 0 )
```

```
    rc = SQLZ_DISCONNECT_PROC;
  else
    rc = SQLZ_HOLD_PROC;

  return(rc);
}

/*---------------------------------------------------------------*/
/*       sqlerror                                                */
/*---------------------------------------------------------------*/
void static sqlerror(char *msg, int ptat, struct sqlca *psqlca)
{
  FILE *err_fp;
  char  err_fn[25] = "/tmp/ord.err.out";
  int j,k;

  err_fp = fopen(err_fn, "a+");
  fprintf(err_fp,"ORDS: %2d sqlcode: %4d %s\n", ptat, psqlca->sqlcode,
msg);
  if (psqlca->sqlerrmc[0] != ' ' || psqlca->sqlerrmc[1] != ' ') {
    fprintf(err_fp,"NEWORDS:slerrmc:  ");
    for(j = 0; j < 5; j++)
    {
      for(k = 0; k < 14; k++) fprintf(err_fp,"%x ", psqlca->sqlerrmc[j*10+k]);
       fprintf(err_fp,"    ");
      for(k = 0; k < 14; k++) fprintf(err_fp,"%c",  psqlca->sqlerrmc[j*10+k]);
       fprintf(err_fp,"\n");
       if (j < 4) fprintf(err_fp,"          ");
    }
  }

  fprintf(err_fp,"ORDS:sqlerrp:  ");
  for(j = 0; j < 8; j++)    fprintf(err_fp,"%c", psqlca->sqlerrp[j]);
  fprintf(err_fp,"\n");

  fprintf(err_fp,"ORDS:sqlerrd: ");
  for(j = 0; j < 6; j++)    fprintf(err_fp," %d", psqlca->sqlerrd[j]);
  fprintf(err_fp,"\n");

  if (psqlca->sqlwarn[0] != ' ') {
    fprintf(err_fp,"ORDS:sqlwarn: ");
    for(j = 0; j < 8; j++)    fprintf(err_fp,"%c ", psqlca->sqlwarn[j]);
    fprintf(err_fp,"\n");
  }

  fprintf(err_fp,"\n");
  fclose(err_fp);
}

/*---------------------------------------------------------------*/
/*       debugit                                                 */
/*---------------------------------------------------------------*/
void debugit (struct ordstat_struct *ordstat_ptr, char *msg)
{
  FILE *debug_fp;
  char  debug_fn[25] = "/tmp/ordstat.debug.out";
  char  timeStamp[27];
  int   j, items;
/*  strcpy(timeStamp, current_tmstmp()); ** JDM ** */
  current_tmstmp(&timeStamp[0]);
  timeStamp[19] = (char)NULL;
  if ((debug_fp = fopen(debug_fn, "a+")) == NULL)
  {
    return;
  }
  fprintf(debug_fp,"debugit information follows %s (%s)\n", timeStamp,
msg);

fprintf(debug_fp,"=========================================
======\n");
  fprintf(debug_fp,"ordstat_struct {\n");
  fprintf(debug_fp,"\ts_W_ID         = %hd (%hX)\n",
        ordstat_ptr->s_W_ID,ordstat_ptr->s_W_ID);
  fprintf(debug_fp,"\ts_D_ID         = %hd (%hX)\n",
        ordstat_ptr->s_D_ID,ordstat_ptr->s_D_ID);
  fprintf(debug_fp,"\ts_C_ID         = %ld (%lX)\n",
        ordstat_ptr->s_C_ID,ordstat_ptr->s_C_ID);
  fprintf(debug_fp,"\ts_C_FIRST      = %s\n",
        ordstat_ptr->s_C_FIRST);
  fprintf(debug_fp,"\ts_C_MIDDLE     = %s\n",
        ordstat_ptr->s_C_MIDDLE);
  fprintf(debug_fp,"\ts_C_LAST       = %s\n",
        ordstat_ptr->s_C_LAST);
  fprintf(debug_fp,"\ts_C_BALANCE    = %0.2f\n",
        ordstat_ptr->s_C_BALANCE);
  fprintf(debug_fp,"\ts_O_ID         = %ld (%lX)\n",
        ordstat_ptr->s_O_ID,ordstat_ptr->s_O_ID);
  fprintf(debug_fp,"\ts_O_ENTRY_D    = %s\n",
        ordstat_ptr->s_O_ENTRY_D);
  fprintf(debug_fp,"\ts_O_CARRIER_ID = %hd (%hX)\n",
        ordstat_ptr->s_O_CARRIER_ID,ordstat_ptr->s_O_CARRIER_ID);
  fprintf(debug_fp,"\ts_ol_cnt       = %hd (%hX)\n",
        ordstat_ptr->s_ol_cnt,ordstat_ptr->s_ol_cnt);
  fprintf(debug_fp,"\ts_transtatus   = %hd (%hX)\n",
        ordstat_ptr->s_transtatus,ordstat_ptr->s_transtatus);
  fprintf(debug_fp,"\tdeadlocks      = %hd (%hX)\n",
        ordstat_ptr->deadlocks,ordstat_ptr->deadlocks);
  fprintf(debug_fp,"\titems {\n");
  items = ordstat_ptr->s_ol_cnt;
  for (j=0; j<items; j++) {
    fprintf(debug_fp,"\t\ts_OL_SUPPLY_W_ID[%d] = %hd (%hX)\n",
          j,ordstat_ptr->item[j].s_OL_SUPPLY_W_ID,ordstat_ptr-
>item[j].s_OL_SUPPLY_W_ID);
    fprintf(debug_fp,"\t\ts_OL_I_ID[%d]       = %ld (%lX)\n",
          j,ordstat_ptr->item[j].s_OL_I_ID,ordstat_ptr->item[j].s_OL_I_ID);
    fprintf(debug_fp,"\t\ts_OL_QUANTITY[%d]    = %hd (%hX)\n",
          j,ordstat_ptr->item[j].s_OL_QUANTITY,ordstat_ptr-
>item[j].s_OL_QUANTITY);
    fprintf(debug_fp,"\t\ts_OL_AMOUNT[%d]      = %0.2f\n",
          j,ordstat_ptr->item[j].s_OL_AMOUNT);
    fprintf(debug_fp,"\t\ts_OL_DELIVERY_D[%d]  = %s\n\n",
          j,ordstat_ptr->item[j].s_OL_DELIVERY_D);
  }
  fprintf(debug_fp,"\t}\n}\n\n");
  fclose(debug_fp);
}

/*****************************************************************/
/*     File: pay(s).sqc (Server code)                            */
/*****************************************************************/

#include "../include/tpcc.h"

void static sqlerror(char *msg, int ptat, struct sqlca *psqlca);
void static debugit(struct payment_struct *, char *);
```

```
/*-------------------------------------------------------------*/
/*            PAYMENT Service                                  */
/*-------------------------------------------------------------*/
int pays (void *reserved1,
        void *reserved2,
        struct sqlda *pin_sqlda,
        struct sqlca *pca)
{

#define  MAXCLAST   3000

  int     retry_count = -1;

  struct payment_struct *payment;

  EXEC SQL include sqlca;

  EXEC SQL BEGIN DECLARE SECTION;
  short   w_id,              d_id;
  short   c_d_id,            c_w_id;
  short   c_payment_cnt;

  double  c_credit_lim,      c_discount,          h_amount;
  double  c_balance;
  double  d_ytd,             w_ytd,               c_ytd_payment;

  char    w_street_1[21],    w_street_2[21];
  char    w_city[21],        w_state[3],          w_zip[10];
  char    d_street_1[21],    d_street_2[21],      d_city[21];
  char    d_state[3],        d_zip[10],           c_first[17];
  char    c_middle[3],       c_last[17],          c_street_1[21];
  char    c_street_2[21],    c_city[21],          c_state[3];
  char    c_zip[10],         c_phone[17],         c_credit[3];
  char    c_data1[251],      h_data[25],          d_name[11];
  char    c_data2[251];
  char    w_name[11],        c_new_data[501],     curr_tmstmp[27];
  char    h_date[27],        c_since[27];

  long    c_id, tmp_c_id, ih_amount;
  long    h_c_id;

  long    clast_count;
  EXEC SQL END DECLARE SECTION;

  long array_c_id[MAXCLAST];

  int len_cdata1, len_cdata2, len_newcdata, tot_len, i;
  int CUROPEN=0, rc;

  EXEC SQL DECLARE read_clast_cur CURSOR FOR
    SELECT c_first, c_id
    FROM customer
    WHERE c_last = :c_last
    AND   c_w_id = :c_w_id
    AND   c_d_id = :c_d_id
    ORDER BY c_first FOR FETCH ONLY;

  EXEC SQL DECLARE Cust_Cursor1 CURSOR FOR
    SELECT c_first, c_middle, c_last, c_street_1, c_street_2, c_city,
    c_state, c_zip, c_phone, c_since, c_credit, c_credit_lim,
    c_discount, c_balance, c_ytd_payment, c_payment_cnt
      FROM customer
      WHERE c_id   = :c_id
      AND   c_w_id = :c_w_id
      AND   c_d_id = :c_d_id
      FOR UPDATE OF c_balance, c_ytd_payment, c_payment_cnt;

  EXEC SQL DECLARE Cust_Cursor1b CURSOR FOR
    SELECT c_id, c_first, c_middle, c_last, c_street_1, c_street_2, c_city,
    c_state, c_zip, c_phone, c_since, c_credit, c_credit_lim,
    c_discount, c_balance, c_ytd_payment, c_payment_cnt
      FROM customer
      WHERE c_last   = :c_last
      AND   c_w_id = :c_w_id
      AND   c_d_id = :c_d_id
      FOR UPDATE OF c_balance, c_ytd_payment, c_payment_cnt;

  EXEC SQL DECLARE Cust_Cursor2 CURSOR FOR
    SELECT c_data1, c_data2
    FROM customer
    WHERE c_id   = :c_id
    AND c_w_id = :c_w_id
    AND c_d_id = :c_d_id
    FOR UPDATE OF  c_balance, c_ytd_payment, c_payment_cnt, c_data1,
c_data2 ;

   /* Retrieve the 'payment' structure from the SQLDA */
   payment =(struct payment_struct *)pin_sqlda->sqlvar[0].sqldata;

   current_tmstmp(&curr_tmstmp[0]); /* **JDM ** */
  curr_tmstmp[19] = (char)NULL;
   memcpy(payment->s_H_DATE, &curr_tmstmp[0], 19);
   payment->s_H_DATE[19] =  (char)NULL;
   h_amount = payment->s_H_AMOUNT;
   ih_amount = h_amount * 100.0;
   w_id = payment->s_W_ID;
   d_id = payment->s_D_ID;
   c_id = payment->s_C_ID;
   c_w_id = payment->s_C_W_ID;
   c_d_id = payment->s_C_D_ID;

 retry_tran:
       retry_count++;

   if (c_id == 0) {
   strcpy(c_last, payment->s_C_LAST);

     EXEC SQL OPEN read_clast_cur;
     if (sqlca.sqlcode != 0) {
       if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
       sqlerror("OPEN read_clast_cur",138, &sqlca);
       goto ferror;
     }

     i = 0;
     do {
       EXEC SQL FETCH read_clast_cur
         INTO :c_first, :tmp_c_id;
       if (sqlca.sqlcode != 0) {
         if (sqlca.sqlcode < 0 || (sqlca.sqlcode == 100 && i == 0)) {
           if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
           sqlerror("FETCH read_clast_cur",156, &sqlca);
           goto ferror;
         }
       }
       else {array_c_id[i++] = tmp_c_id;}
```

```
    } while (sqlca.sqlcode == 0);

    i = (int)((i+1)/2) - 1;
    c_id = array_c_id[i];
  }

  EXEC SQL OPEN Cust_Cursor1;
  if (sqlca.sqlcode != 0) {
    if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
    sqlerror ("OPEN Cust_Cursor1",42, &sqlca);
    goto ferror;
  }
  EXEC SQL
    FETCH Cust_Cursor1
    INTO :c_first, :c_middle, :c_last, :c_street_1, :c_street_2, :c_city,
    :c_state, :c_zip, :c_phone, :c_since, :c_credit, :c_credit_lim,
    :c_discount, :c_balance, :c_ytd_payment, :c_payment_cnt;
  if (sqlca.sqlcode != 0) {
    if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
    sqlerror ("FETCH Cust_Cursor1",43, &sqlca);
    goto ferror;
  }

FCUST:

  strcpy( payment->s_C_FIRST,c_first);
  strcpy( payment->s_C_MIDDLE,c_middle);
  strcpy( payment->s_C_LAST,c_last);
  strcpy( payment->s_C_STREET_1,c_street_1);
  strcpy( payment->s_C_STREET_2,c_street_2);
  strcpy( payment->s_C_CITY,c_city);
  strcpy( payment->s_C_STATE,c_state);
  strcpy( payment->s_C_ZIP,c_zip);
  strcpy( payment->s_C_PHONE,c_phone);
  memcpy( payment->s_C_SINCE,c_since,10);
  payment->s_C_SINCE[10] = (char)NULL;
  strcpy( payment->s_C_CREDIT,c_credit);
  payment->s_C_CREDIT_LIM = c_credit_lim;
  payment->s_C_DISCOUNT = c_discount;

  c_balance -= h_amount;

  payment->s_C_BALANCE = c_balance;

  c_ytd_payment = c_ytd_payment + h_amount;
  c_payment_cnt = c_payment_cnt + 1;

  if (c_credit[0] == 'B' && c_credit[1] == 'C') {   /* 10% of customers */

    EXEC SQL OPEN Cust_Cursor2;
    if (sqlca.sqlcode != 0) {
      if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
      sqlerror("OPEN Cust_Cursor2",270, &sqlca);
      goto ferror;
    }
    EXEC SQL FETCH Cust_Cursor2 INTO :c_data1, :c_data2;
    if (sqlca.sqlcode != 0) {
      if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
      sqlerror("SELECT c_data FROM customer",280, &sqlca);
      goto ferror;
    }

    sprintf(c_new_data, "%d,%d,%d,%d,%d,%1.2f",
          c_id, c_d_id, c_w_id, d_id, w_id, h_amount);
    len_newcdata = strlen(c_new_data);
    len_cdata1 = strlen(c_data1);
    len_cdata2 = strlen(c_data2);
    tot_len = len_cdata1 + len_cdata2 + len_newcdata;
    if (tot_len > 500) c_data2[len_cdata2-(tot_len-500)] = (char)NULL;
    strcat(c_new_data, c_data1);
    strcat(c_new_data, c_data2);
    memcpy(payment->s_C_DATA, c_new_data, 200);
    memcpy(c_data1, c_new_data, 250);
    strncpy(c_data2, c_new_data+250, 250);

    /*---------------------------------------------*/
    /* Update BAD CUSTOMER                         */
    /*---------------------------------------------*/

    EXEC SQL UPDATE customer
      SET c_balance = :c_balance,
      c_ytd_payment = :c_ytd_payment,
      c_payment_cnt = :c_payment_cnt,
      c_data1 = :c_data1,
      c_data2 = :c_data2
        WHERE CURRENT OF Cust_Cursor2;
    if (sqlca.sqlcode != 0) {
      if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
      sqlerror("UPDATE customer",314, &sqlca);
      goto ferror;
    }
  }  /* end if bad customer */
  else {

    /*---------------------------------------------*/
    /* Update GOOD CUSTOMER                        */
    /*---------------------------------------------*/

    EXEC SQL UPDATE customer
      SET c_balance = :c_balance,
      c_ytd_payment = :c_ytd_payment,
      c_payment_cnt = :c_payment_cnt
        WHERE CURRENT OF Cust_Cursor1;
    if (sqlca.sqlcode != 0) {
      if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
      sqlerror("UPDATE customer",335, &sqlca);
      goto ferror;
    }
  }

  /*--------------------------------------*/
  /* Start Compound SQL                   */
  /*--------------------------------------*/

  EXEC SQL BEGIN COMPOUND ATOMIC STATIC

    UPDATE district SET d_ytd = d_ytd + :h_amount
    WHERE d_id = :d_id AND d_w_id = :w_id;

  SELECT d_street_1, d_street_2, d_city, d_state, d_zip, d_name
    INTO :d_street_1, :d_street_2, :d_city, :d_state, :d_zip, :d_name
    FROM district WHERE d_id = :d_id AND d_w_id = :w_id;

  UPDATE warehouse SET w_ytd = w_ytd + :h_amount WHERE w_id =
:w_id;
```

```
SELECT w_street_1, w_street_2, w_city, w_state, w_zip, w_name
  INTO :w_street_1, :w_street_2, :w_city, :w_state, :w_zip, :w_name
  FROM warehouse WHERE w_id = :w_id;

END COMPOUND;

if (sqlca.sqlcode != 0) {
  switch (sqlca.sqlerrd[LASTCODE]) {
   case 0:
     if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
     sqlerror("UPDATE district",369, &sqlca);
     goto ferror;
   case 1:
     if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
     sqlerror("SELECT FROM district",376, &sqlca);
     goto ferror;
   case 2:
     if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
     sqlerror("UPDATE warehouse",383, &sqlca);
     goto ferror;
   case 3:
     if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
     sqlerror("SELECT FROM warehouse",390, &sqlca);
     goto ferror;
   default:
     if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
     sqlerror("default case of compound sql",347, &sqlca);
     goto ferror;
  }
}

strcpy( payment->s_W_STREET_1,w_street_1);
strcpy( payment->s_W_STREET_2,w_street_2);
strcpy( payment->s_W_CITY,w_city);
strcpy( payment->s_W_STATE,w_state);
strcpy( payment->s_W_ZIP,w_zip);
strcpy( payment->s_D_STREET_1,d_street_1);
strcpy( payment->s_D_STREET_2,d_street_2);
strcpy( payment->s_D_CITY,d_city);
strcpy( payment->s_D_STATE,d_state);
strcpy( payment->s_D_ZIP,d_zip);

sprintf(h_data, "%s    %s", w_name, d_name);

/*---------------------------------------*/
/* Start Compound SQL                    */
/*---------------------------------------*/

h_c_id = c_id;
EXEC SQL BEGIN COMPOUND ATOMIC STATIC

  INSERT INTO history
  VALUES (:h_c_id, :c_d_id, :c_w_id, :d_id, :w_id, :curr_tmstmp,
        :ih_amount, :h_data);

COMMIT;

END COMPOUND;

if (sqlca.sqlcode != 0) {
  switch (sqlca.sqlerrd[LASTCODE]) {
   case 0:
     if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
     sqlerror("INSERT INTO history",457, &sqlca);
     goto ferror;
   case 1:
     if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
     sqlerror("COMMIT",464, &sqlca);
     goto ferror;
   default:
     if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
     sqlerror("default case of compound sql",405, &sqlca);
     goto ferror;
  }
}

/*---------------------------------------------*/
/*  Return to client                           */
/*---------------------------------------------*/
payment->s_transtatus = TRAN_OK;
rc = 0;
goto mexit;

ferror:
  payment->s_transtatus = FATAL_SQLERROR;
  rc = -1;
  EXEC SQL rollback work;
  if (sqlca.sqlcode != 0)
    sqlerror("ROLLBACK FAILED", 666, &sqlca);
  debugit(payment, "ferror code");

mexit:
      payment->deadlocks = retry_count;
  memcpy( pca, &sqlca, sizeof( struct sqlca ) );
  if ( sqlca.sqlcode < 0 )
    rc = SQLZ_DISCONNECT_PROC;
  else
    rc = SQLZ_HOLD_PROC;

  return(rc);
}

/*-------------------------------------------------------------*/
/*       sqlerror                                              */
/*-------------------------------------------------------------*/
void static sqlerror(char *msg, int ptat, struct sqlca *psqlca)
{
  FILE *err_fp;
  char  err_fn[25] = "/tmp/pay.err.out";
  int j,k;

  err_fp = fopen(err_fn, "a+");
  fprintf(err_fp,"PAY: %2d sqlcode: %4d %s\n", ptat, psqlca->sqlcode, msg);
  if (psqlca->sqlerrmc[0] != ' ' || psqlca->sqlerrmc[1] != ' ') {
    fprintf(err_fp,"PAYS:slerrmc:  ");
    for(j = 0; j < 5; j++)
    {
      for(k = 0; k < 14; k++) fprintf(err_fp,"%x ", psqlca->sqlerrmc[j*10+k]);
      fprintf(err_fp,"    ");
      for(k = 0; k < 14; k++) fprintf(err_fp,"%c",  psqlca->sqlerrmc[j*10+k]);
      fprintf(err_fp,"\n");
      if (j < 4) fprintf(err_fp,"          ");
    }
  }

  fprintf(err_fp,"PAYS:sqlerrp:  ");
```

```
    for(j = 0; j < 8; j++)    fprintf(err_fp,"%c", psqlca->sqlerrp[j]);
    fprintf(err_fp,"\n");

    fprintf(err_fp,"PAYS:sqlerrd: ");
    for(j = 0; j < 6; j++)    fprintf(err_fp," %d", psqlca->sqlerrd[j]);
    fprintf(err_fp,"\n");

    if (psqlca->sqlwarn[0] != ' ') {
      fprintf(err_fp,"PAYS:sqlwarn: ");
      for(j = 0; j < 8; j++)    fprintf(err_fp,"%c ", psqlca->sqlwarn[j]);
      fprintf(err_fp,"\n");
    }

    fprintf(err_fp,"\n");
    fclose(err_fp);
}

/*---------------------------------------------------------------*/
/*       debugit                                                 */
/*---------------------------------------------------------------*/
void debugit (struct payment_struct *payment_ptr, char *msg)
{
    FILE *debug_fp;
    char  debug_fn[39] = "/tmp/payment.debug.out";
    char  timeStamp[27];
    current_tmstmp(&timeStamp[0]);
    timeStamp[19] = (char)NULL;
    if ((debug_fp = fopen(debug_fn, "a+")) == NULL)
    {
      return;
    }
    fprintf(debug_fp,"debugit information follows %s (%s)\n", timeStamp,
msg);

fprintf(debug_fp,"=========================================
========\n");
    fprintf(debug_fp,"payment_struct {\n");
    fprintf(debug_fp,"\ts_W_ID         = %hd (%hX)\n",
          payment_ptr->s_W_ID,payment_ptr->s_W_ID);
    fprintf(debug_fp,"\ts_D_ID         = %hd (%hX)\n",
          payment_ptr->s_D_ID,payment_ptr->s_D_ID);
    fprintf(debug_fp,"\ts_C_ID         = %ld (%lX)\n",
          payment_ptr->s_C_ID,payment_ptr->s_C_ID);
    fprintf(debug_fp,"\ts_C_D_ID       = %hd (%hX)\n",
          payment_ptr->s_C_D_ID,payment_ptr->s_C_D_ID);
    fprintf(debug_fp,"\ts_C_W_ID       = %hd (%hX)\n",
          payment_ptr->s_C_W_ID,payment_ptr->s_C_W_ID);
    fprintf(debug_fp,"\ts_transtatus   = %hd (%hX)\n",
          payment_ptr->s_transtatus,payment_ptr->s_transtatus);
    fprintf(debug_fp,"\ts_H_AMOUNT     = %0.2f\n",
          payment_ptr->s_H_AMOUNT);
    fprintf(debug_fp,"\ts_H_DATE       = %s\n",
          payment_ptr->s_H_DATE);
    fprintf(debug_fp,"\ts_W_STREET_1   = %s\n",
          payment_ptr->s_W_STREET_1);
    fprintf(debug_fp,"\ts_W_STREET_2   = %s\n",
          payment_ptr->s_W_STREET_2);
    fprintf(debug_fp,"\ts_W_CITY       = %s\n",
          payment_ptr->s_W_CITY);
    fprintf(debug_fp,"\ts_W_STATE      = %s\n",
          payment_ptr->s_W_STATE);
    fprintf(debug_fp,"\ts_W_ZIP        = %s\n",
          payment_ptr->s_W_ZIP);
    fprintf(debug_fp,"\ts_D_STREET_1   = %s\n",
          payment_ptr->s_D_STREET_1);
    fprintf(debug_fp,"\ts_D_STREET_2   = %s\n",
          payment_ptr->s_D_STREET_2);
    fprintf(debug_fp,"\ts_D_CITY       = %s\n",
          payment_ptr->s_D_CITY);
    fprintf(debug_fp,"\ts_D_STATE      = %s\n",
          payment_ptr->s_D_STATE);
    fprintf(debug_fp,"\ts_D_ZIP        = %s\n",
          payment_ptr->s_D_ZIP);
    fprintf(debug_fp,"\ts_C_FIRST      = %s\n",
          payment_ptr->s_C_FIRST);
    fprintf(debug_fp,"\ts_C_MIDDLE     = %s\n",
          payment_ptr->s_C_MIDDLE);
    fprintf(debug_fp,"\ts_C_LAST       = %s\n",
          payment_ptr->s_C_LAST);
    fprintf(debug_fp,"\ts_C_STREET_1   = %s\n",
          payment_ptr->s_C_STREET_1);
    fprintf(debug_fp,"\ts_C_STREET_2   = %s\n",
          payment_ptr->s_C_STREET_2);
    fprintf(debug_fp,"\ts_C_CITY       = %s\n",
          payment_ptr->s_C_CITY);
    fprintf(debug_fp,"\ts_C_STATE      = %s\n",
          payment_ptr->s_C_STATE);
    fprintf(debug_fp,"\ts_C_ZIP        = %s\n",
          payment_ptr->s_C_ZIP);
    fprintf(debug_fp,"\ts_C_PHONE      = %s\n",
          payment_ptr->s_C_PHONE);
    fprintf(debug_fp,"\ts_C_SINCE      = %s\n",
          payment_ptr->s_C_SINCE);
    fprintf(debug_fp,"\ts_C_CREDIT     = %s\n",
          payment_ptr->s_C_CREDIT);
    fprintf(debug_fp,"\ts_C_CREDIT_LIM = %0.2f\n",
          payment_ptr->s_C_CREDIT_LIM);
    fprintf(debug_fp,"\ts_C_DISCOUNT   = %0.2f\n",
          payment_ptr->s_C_DISCOUNT);
    fprintf(debug_fp,"\ts_C_BALANCE    = %0.2f\n",
          payment_ptr->s_C_BALANCE);
    fprintf(debug_fp,"\tdeadlocks     = %hd (%hX)\n",
          payment_ptr->deadlocks,payment_ptr->deadlocks);
    fprintf(debug_fp,"\ts_C_DATA       = %s\n",
          payment_ptr->s_C_DATA);
    fprintf(debug_fp,"\n}\n\n");
    fclose(debug_fp);
}

/*****************************************************************************/
/*      File: stklv(s).sqc (Server code)                                     */
/*****************************************************************************/

#include "../include/tpcc.h"

void static sqlerror(char *, int , struct sqlca *);
void debugit (struct stocklev_struct *, char *);

/*---------------------------------------------------------------*/
/*            STOCKLEV Service                                   */
/*---------------------------------------------------------------*/
int stklvs (void *reserved1,
          void *reserved2,
          struct sqlda *pin_sqlda,
          struct sqlca *pca)
{
```

```
struct stocklev_struct *stocklev;

EXEC SQL include sqlca;

EXEC SQL BEGIN DECLARE SECTION;
long    d_next_o_id, o_id, low_stock;
short   w_id, d_id, threshold;
EXEC SQL END DECLARE SECTION;

int rc = 0;
int retry_count = -1;

/* Retrieve the 'stocklev' structure from the SQLDA */
stocklev =(struct stocklev_struct *)pin_sqlda->sqlvar[0].sqldata;

/*----------------------------------------*/
/*  Read DISTRICT                         */
/*----------------------------------------*/
w_id = stocklev->s_W_ID;
d_id = stocklev->s_D_ID;

retry_tran:
     retry_count++;

EXEC SQL SELECT d_next_o_id
  INTO :d_next_o_id
  FROM district
  WHERE d_w_id = :w_id
  AND d_id = :d_id;

if (sqlca.sqlcode != 0) {
  if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
  sqlerror("SELECT read_district_cur",49, &sqlca);
  goto ferror;
}

/*----------------------------------------*/
/*  Read ORDER_LINE & STOCK               */
/*----------------------------------------*/
threshold = stocklev->s_threshold;   o_id = d_next_o_id - 21;
EXEC SQL BEGIN COMPOUND ATOMIC STATIC

  SELECT count(*)
  into :low_stock
  from ( SELECT distinct S_I_ID
       FROM ORDER_LINE, STOCK
       where OL_W_ID = :w_id
       and OL_D_ID = :d_id
       and OL_O_ID < :d_next_o_id and OL_O_ID > :o_id
       and S_I_ID = OL_I_ID
       and S_W_ID = OL_W_ID
       and S_QUANTITY < :threshold ) foo;

COMMIT;

END COMPOUND;

stocklev->s_low_stock = low_stock;

if (sqlca.sqlcode != 0) {
  switch (sqlca.sqlerrd[LASTCODE]) {
   case 0:
     if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
     sqlerror("select count(distinct)",83, &sqlca);
     goto ferror;
   case 1:
     if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
     sqlerror("commit",87, &sqlca);
     goto ferror;
   default:
     if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
     sqlerror("default case of compound sql",91, &sqlca);
     goto ferror;
  }
}

stocklev->s_transtatus = TRAN_OK;
rc = 0;
goto mexit;

ferror:
  rc = -1;
  stocklev->s_transtatus = FATAL_SQLERROR;
  EXEC SQL rollback work;
  if (sqlca.sqlcode != 0)
    sqlerror("ROLLBACK FAILED", 666, &sqlca);
  debugit(stocklev, "ferror code");

mexit:
  stocklev->deadlocks = retry_count;
  memcpy( pca, &sqlca, sizeof( struct sqlca ) );
  if ( sqlca.sqlcode < 0 )
    rc = SQLZ_DISCONNECT_PROC;
  else
    rc = SQLZ_HOLD_PROC;

  return(rc);
}

/*----------------------------------------------------------------*/
/*       sqlerror                                                 */
/*----------------------------------------------------------------*/
void static sqlerror(char *msg, int ptat, struct sqlca *psqlca)
{
  FILE *err_fp;
  char  err_fn[25] = "/tmp/stklv.err.out";
  int j,k;

  err_fp = fopen(err_fn, "a+");
  fprintf(err_fp,"STKLV: %2d sqlcode: %4d %s\n", ptat, psqlca->sqlcode, msg);
  if (psqlca->sqlerrmc[0] != ' ' || psqlca->sqlerrmc[1] != ' ') {
    fprintf(err_fp,"STKLV:slerrmc:  ");
    for(j = 0; j < 5; j++)
    {
      for(k = 0; k < 14; k++) fprintf(err_fp,"%x ", psqlca->sqlerrmc[j*10+k]);
      fprintf(err_fp,"    ");
      for(k = 0; k < 14; k++) fprintf(err_fp,"%c",  psqlca->sqlerrmc[j*10+k]);
      fprintf(err_fp,"\n");
      if (j < 4) fprintf(err_fp,"          ");
    }
  }

  fprintf(err_fp,"STKLV:sqlerrp:  ");
  for(j = 0; j < 8; j++)    fprintf(err_fp,"%c", psqlca->sqlerrp[j]);
  fprintf(err_fp,"\n");
```

```
   fprintf(err_fp,"STKLV:sqlerrd: ");
   for(j = 0; j < 6; j++)    fprintf(err_fp," %d", psqlca->sqlerrd[j]);
   fprintf(err_fp,"\n");

   if (psqlca->sqlwarn[0] != ' ') {
      fprintf(err_fp,"STKLV:sqlwarn: ");
      for(j = 0; j < 8; j++)    fprintf(err_fp,"%c ", psqlca->sqlwarn[j]);
      fprintf(err_fp,"\n");
   }

   fprintf(err_fp,"\n");
   fclose(err_fp);
}

/*---------------------------------------------------------------*/
/*       debugit                                                 */
/*---------------------------------------------------------------*/
void debugit (struct stocklev_struct *stocklev_ptr, char *msg)
{
   FILE *debug_fp;
   char  debug_fn[25] = "/tmp/stklv.debug.out";
   char  timeStamp[27];
   current_tmstmp(&timeStamp[0]);
   timeStamp[19] = (char)NULL;
   if ((debug_fp = fopen(debug_fn, "a+")) == NULL)
   {
      return;
   }
   fprintf(debug_fp,"debugit information follows %s (%s)\n", timeStamp,
msg);

fprintf(debug_fp,"=========================================
========\n");
   fprintf(debug_fp,"stocklev_struct {\n");
   fprintf(debug_fp,"\ts_W_ID         = %hd (%hX)\n",
         stocklev_ptr->s_W_ID,stocklev_ptr->s_W_ID);
   fprintf(debug_fp,"\ts_D_ID         = %hd (%hX)\n",
         stocklev_ptr->s_D_ID,stocklev_ptr->s_D_ID);
   fprintf(debug_fp,"\ts_threshold    = %hd (%hX)\n",
         stocklev_ptr->s_threshold,stocklev_ptr->s_threshold);
   fprintf(debug_fp,"\ts_low_stock    = %ld (%lX)\n",
         stocklev_ptr->s_low_stock,stocklev_ptr->s_low_stock);
   fprintf(debug_fp,"\ts_transtatus   = %hd (%hX)\n",
         stocklev_ptr->s_transtatus,stocklev_ptr->s_transtatus);
   fprintf(debug_fp,"\tdeadlocks      = %hd (%hX)\n",
         stocklev_ptr->deadlocks,stocklev_ptr->deadlocks);
   fprintf(debug_fp,"}\n\n");
   fclose(debug_fp);
}
```

Appendix B: Database Design

B

This Appendix contains the scripts used to create the database and the load program used to load the database initially.

```
/**************************************************************************/
/*      file:  crtb.sqc                                                   */
/*                                                                        */
/*      TPC-C benchmark for DB2 for Solaris. This program creates
*/
/*      the required tables.                                              */
/*                                                                        */
/**************************************************************************/

#include “ctest.h”
#include “../include/lval.h”
#include <stdlib.h>

EXEC SQL include sqlca ;
EXEC SQL include sqlda ;

EXEC SQL BEGIN DECLARE SECTION;                 /* Declare Host
Variables  */
char    errmsg[201];
long    ware_num ;
long    dist_num ;
long    cust_num ;
long    ord_num ;
long    oline_num ;
long    count ;
long    ware_count ;
char dbname[8] = “tpcc”;
EXEC SQL END DECLARE SECTION;

int i, j;

int main ( void );
void create_item_tbl( void );
void create_stock_tbl( void );
void create_ware_tbl( void );
void create_dist_tbl( void );
void create_cust_tbl( void );
void create_hist_tbl( void );
void create_nu_ord_tbl( void );
void create_ordr_tbl( void );
void create_oline_tbl( void );

/*----------------------------------------------------------------------*/
/*      main                                                            */
/*----------------------------------------------------------------------*/


int main ( void )
{
  int option;
  char str[80];
  char msg[30];

  char *dbname_ptr;
  if ((dbname_ptr = getenv(“TPCC_DBNAME”)) != NULL) {
    strcpy (dbname, dbname_ptr);
  }

  EXEC SQL CONNECT TO :dbname IN EXCLUSIVE MODE;
  sprintf(msg, “CONNECT TO %s”, dbname);
  SQLERR(&sqlca, msg);

  fprintf(stdout,”Pick table to be created: \n”);
  fprintf(stdout,”1. all tables \n”);
  fprintf(stdout,”2. warehouse \n”);
  fprintf(stdout,”3. district \n”);
  fprintf(stdout,”4. item \n”);
  fprintf(stdout,”5. stock \n”);
  fprintf(stdout,”6. customer \n”);
  fprintf(stdout,”7. history \n”);
  fprintf(stdout,”8. orders \n”);
  fprintf(stdout,”9. new-order \n”);
  fprintf(stdout,”10. order-line \n”);
```

```
  fprintf(stdout,"\nEnter option: ");
  scanf("%d", &option);

  switch (option) {
   case 1: /* ALL */
     create_ware_tbl();
     create_dist_tbl();
     create_item_tbl();
     create_stock_tbl();
     create_cust_tbl();
     create_hist_tbl();
     create_ordr_tbl();
     create_nu_ord_tbl();
     create_oline_tbl();
     break;
   case 2: /* WAREHOUSE */
     create_ware_tbl();
     break;
   case 3: /* DISTRICT  */
     create_dist_tbl();
     break;
   case 4: /* ITEM      */
     create_item_tbl();
     break;
   case 5: /* STOCK     */
     create_stock_tbl();
     break;
   case 6: /* CUSTOMER  */
     create_cust_tbl();
     break;
   case 7: /* HISTORY   */
     create_hist_tbl();
     break;
   case 8: /* ORDERS    */
     create_ordr_tbl();
     break;
   case 9: /* NEW_ORDER    */
     create_nu_ord_tbl();
     break;
   case 10: /* ORDER_LINE    */
     create_oline_tbl();
     break;
   default:
     fprintf(stdout, "error: invalid option. \n");
     break;
  }

  EXEC SQL COMMIT WORK;          SQLERR(&sqlca, "COMMIT");

  EXEC SQL CONNECT RESET;        SQLERR(&sqlca, "CONNECT
RESET");

  return 0;
}

/*----------------------------------------------------------------*/
/*      create item table                                         */
/*----------------------------------------------------------------*/
void create_item_tbl( void )
{
  fprintf(stdout, "building item table ...\n");

  EXEC SQL DROP TABLE ITEM;
  EXEC SQL COMMIT WORK; SQLERR(&sqlca, "COMMIT");

  EXEC SQL create table ITEM
    (
    I_ID           INTEGER     NOT NULL,
    I_IM_ID        INTEGER     NOT NULL,
    I_NAME         CHAR(24)    NOT NULL,
    I_PRICE        INTEGER     NOT NULL,
    I_DATA         CHAR(50)    NOT NULL
    );
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "create table ITEM");
}

/*----------------------------------------------------------------*/
/*      create stock table                                        */
/*----------------------------------------------------------------*/

void create_stock_tbl( void )
{
  fprintf(stdout, "building stock table ...\n");

  EXEC SQL DROP TABLE STOCK;
  EXEC SQL COMMIT WORK; SQLERR(&sqlca, "COMMIT");

  EXEC SQL create table STOCK
    (
    S_I_ID         INTEGER     NOT NULL,
    S_W_ID         SMALLINT    NOT NULL,
    S_REMOTE_CNT   SMALLINT    NOT NULL,
    S_QUANTITY     SMALLINT    NOT NULL,
    S_ORDER_CNT    SMALLINT    NOT NULL,
    S_YTD          INTEGER     NOT NULL,
    S_DIST_01      CHAR(24)    NOT NULL,
    S_DIST_02      CHAR(24)    NOT NULL,
    S_DIST_03      CHAR(24)    NOT NULL,
    S_DIST_04      CHAR(24)    NOT NULL,
    S_DIST_05      CHAR(24)    NOT NULL,
    S_DIST_06      CHAR(24)    NOT NULL,
    S_DIST_07      CHAR(24)    NOT NULL,
    S_DIST_08      CHAR(24)    NOT NULL,
    S_DIST_09      CHAR(24)    NOT NULL,
    S_DIST_10      CHAR(24)    NOT NULL,
    S_DATA         CHAR(50)    NOT NULL
    );
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "create table STOCK");
}

/*----------------------------------------------------------------*/
/*      create warehouse table                                    */
/*----------------------------------------------------------------*/
void create_ware_tbl( void )
{
  fprintf(stdout, "building warehouse table...\n");

  EXEC SQL DROP TABLE WAREHOUSE;
  EXEC SQL COMMIT WORK; SQLERR(&sqlca, "COMMIT");

  EXEC SQL create table WAREHOUSE
  (
    W_ID        SMALLINT       NOT NULL,
    W_NAME      CHAR(10)       NOT NULL,
    W_STREET_1  CHAR(20)       NOT NULL,
    W_STREET_2  CHAR(20)       NOT NULL,
```

```
    W_CITY      CHAR(20)        NOT NULL,
    W_STATE     CHAR(2)         NOT NULL,
    W_ZIP       CHAR(9)         NOT NULL,
    W_TAX       FLOAT           NOT NULL,
    W_YTD       DECIMAL(12,2)   NOT NULL
  );
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "12");
}

/*-------------------------------------------------------------------*/
/*      create dist table                                            */
/*-------------------------------------------------------------------*/
void create_dist_tbl( void )
{
  fprintf(stdout, "building district table ...\n");

  EXEC SQL DROP TABLE DISTRICT;
  EXEC SQL COMMIT WORK; SQLERR(&sqlca, "COMMIT");

  EXEC SQL create table DISTRICT
  (
    D_ID        SMALLINT        NOT NULL,
    D_W_ID      SMALLINT        NOT NULL,
    D_NAME      CHAR(10)        NOT NULL,
    D_STREET_1  CHAR(20)        NOT NULL,
    D_STREET_2  CHAR(20)        NOT NULL,
    D_CITY      CHAR(20)        NOT NULL,
    D_STATE     CHAR(2)         NOT NULL,
    D_ZIP       CHAR(9)         NOT NULL,
    D_TAX       FLOAT           NOT NULL,
    D_YTD       DECIMAL(12,2)   NOT NULL,
    D_NEXT_O_ID INTEGER         NOT NULL
  );
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "");
}

/*-------------------------------------------------------------------*/
/*      create customer table                                        */
/*-------------------------------------------------------------------*/
void create_cust_tbl( void )
{
  fprintf(stdout, "building customer table ...\n");
  EXEC SQL DROP TABLE CUSTOMER;
  EXEC SQL COMMIT WORK; SQLERR(&sqlca, "COMMIT");

  EXEC SQL create table CUSTOMER
  (
    C_ID            INTEGER       NOT NULL,
    C_D_ID          SMALLINT      NOT NULL,
    C_W_ID          SMALLINT      NOT NULL,
    C_FIRST         CHAR(16)      NOT NULL,
    C_MIDDLE        CHAR(2)       NOT NULL,
    C_LAST          CHAR(16)      NOT NULL,
    C_STREET_1      CHAR(20)      NOT NULL,
    C_STREET_2      CHAR(20)      NOT NULL,
    C_CITY          CHAR(20)      NOT NULL,
    C_STATE         CHAR(2)       NOT NULL,
    C_ZIP           CHAR(9)       NOT NULL,
    C_PHONE         CHAR(16)      NOT NULL,
    C_SINCE         TIMESTAMP     NOT NULL,
    C_CREDIT        CHAR(2)       NOT NULL,
    C_CREDIT_LIM    DECIMAL(12,2) NOT NULL,
    C_DISCOUNT      FLOAT         NOT NULL,
    C_DELIVERY_CNT  SMALLINT      NOT NULL,
    C_BALANCE       DECIMAL(12,2) NOT NULL,
    C_YTD_PAYMENT   DECIMAL(12,2) NOT NULL,
    C_PAYMENT_CNT   SMALLINT      NOT NULL,
    C_DATA1         CHAR(250)     NOT NULL,
    C_DATA2         CHAR(250)     NOT NULL
  );
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "");
}

/*-------------------------------------------------------------------*/
/*      create hist table                                            */
/*-------------------------------------------------------------------*/
void create_hist_tbl( void )
{
  fprintf(stdout, "building hist table ...\n");

  EXEC SQL DROP TABLE HISTORY;
  EXEC SQL COMMIT WORK; SQLERR(&sqlca, "COMMIT");

  EXEC SQL create table HISTORY
  (
    H_C_ID          INTEGER     NOT NULL,
    H_C_D_ID        SMALLINT    NOT NULL,
    H_C_W_ID        SMALLINT    NOT NULL,
    H_D_ID          SMALLINT    NOT NULL,
    H_W_ID          SMALLINT    NOT NULL,
    H_DATE          TIMESTAMP   NOT NULL,
    H_AMOUNT        INTEGER     NOT NULL,
    H_DATA          CHAR(24)    NOT NULL  /* data in the original row is
                                  average 18 bytes; rows added
                                  in payment transaction have
                                  average 20 bytes (24 if you
                                  count W_NAME and D_NAME as 10) */
  );
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "");
}


/*-------------------------------------------------------------------*/
/*      create nu_ord table                                          */
/*-------------------------------------------------------------------*/
void create_nu_ord_tbl( void )
{
  fprintf(stdout, "building nu_ord table ...\n");

  EXEC SQL DROP TABLE NEW_ORDER;
  EXEC SQL COMMIT WORK; SQLERR(&sqlca, "COMMIT");

  EXEC SQL create table NEW_ORDER
  (
    NO_O_ID         INTEGER     NOT NULL,
    NO_D_ID         SMALLINT    NOT NULL,
    NO_W_ID         SMALLINT    NOT NULL
  );
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "3");
}


/*-------------------------------------------------------------------*/
/*      create orders table                                          */
/*-------------------------------------------------------------------*/
```

```
void create_ordr_tbl( void )
{
  fprintf(stdout, "building orders table ...\n");
  EXEC SQL DROP TABLE ORDERS;
  EXEC SQL COMMIT WORK; SQLERR(&sqlca, "COMMIT");

  EXEC SQL create table ORDERS
  (
    O_ID            INTEGER     NOT NULL,
    O_C_ID          INTEGER     NOT NULL,
    O_D_ID          SMALLINT    NOT NULL,
    O_W_ID          SMALLINT    NOT NULL,
    O_ENTRY_D       TIMESTAMP   NOT NULL,
    O_CARRIER_ID    SMALLINT    NOT NULL,
    O_OL_CNT        SMALLINT    NOT NULL,
    O_ALL_LOCAL     SMALLINT    NOT NULL
  );
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "2");
}

/*----------------------------------------------------------------------*/
/*      create order_line table                                         */
/*----------------------------------------------------------------------*/

void create_oline_tbl( void )
{
  fprintf(stdout, "building order_line table ...\n");
  EXEC SQL DROP TABLE ORDER_LINE;
  EXEC SQL COMMIT WORK; SQLERR(&sqlca, "COMMIT");

  EXEC SQL create table ORDER_LINE
  (
    OL_O_ID         INTEGER     NOT NULL,
    OL_D_ID         SMALLINT    NOT NULL,
    OL_W_ID         SMALLINT    NOT NULL,
    OL_NUMBER       SMALLINT    NOT NULL,
    OL_I_ID         INTEGER     NOT NULL,
    OL_SUPPLY_W_ID  SMALLINT    NOT NULL,
    OL_DELIVERY_D   TIMESTAMP   NOT NULL,
    OL_QUANTITY     SMALLINT    NOT NULL,
    OL_AMOUNT       INTEGER     NOT NULL,
    OL_DIST_INFO    CHAR(24)    NOT NULL
  );
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "6");
}

/************************************************************************/
/*  cridx.sqc  -  Create indexes                                        */
/*                                                                      */
/*     TPC-C benchmark for DB2 for Solaris. This program creates
*/
/*     the required tables.                                             */
/*                                                                      */
/************************************************************************/

#include "ctest.h"
#include "../include/lval.h"

EXEC SQL include sqlca ;
EXEC SQL include sqlda ;
                                /*  PROTOTYPES.  */
void crix_dist_tbl( void );
void crix_cust_tbl( void );
void crix_hist_tbl( void );
void crix_nu_ord_tbl( void );
void crix_ordr_tbl( void );
void crix_oline_tbl( void );
void crix_item_tbl( void );
void crix_stock_tbl( void );
void crix_ware_tbl( void );

EXEC SQL BEGIN DECLARE SECTION;              /* Declare Host Variables  */
char   errmsg[201];
long   ware_num ;
long   dist_num ;
long   cust_num ;
long   ord_num ;
long   oline_num ;
long   count ;
long   ware_count ;
char dbname[8] = "tpcc";
EXEC SQL END DECLARE SECTION;

int i, j;
double timestamp1,        timestamp2,        elapse;


/*----------------------------------------------------------------------*/
/*      main                                                            */
/*----------------------------------------------------------------------*/

main ()
{
  int option;
  char str[80];
  char msg[30];

  char *dbname_ptr;
  if ((dbname_ptr = getenv("TPCC_DBNAME")) != NULL) {
    strcpy (dbname, dbname_ptr);
  }

  EXEC SQL CONNECT TO :dbname IN SHARE MODE;
  sprintf(msg, "CONNECT TO %s", dbname);
  SQLERR(&sqlca, msg);

  fprintf(stdout,"Pick table to create index on: \n");
  fprintf(stdout,"1. all tables \n");
  fprintf(stdout,"2. warehouse \n");
  fprintf(stdout,"3. district \n");
  fprintf(stdout,"4. item \n");
  fprintf(stdout,"5. stock \n");
  fprintf(stdout,"6. customer \n");
  fprintf(stdout,"7. history \n");
  fprintf(stdout,"8. orders \n");
  fprintf(stdout,"9. new-order \n");
  fprintf(stdout,"10. order-line \n");
  fprintf(stdout,"\nEnter option: ");
  scanf("%d", &option);

  switch (option) {
  case 1: /* ALL */
    crix_ware_tbl();
    crix_dist_tbl();
    crix_item_tbl();
    crix_stock_tbl();
```

```
    crix_cust_tbl();
    crix_hist_tbl();
    crix_ordr_tbl();
    crix_nu_ord_tbl();
    crix_oline_tbl();
    break;
  case 2: /* WAREHOUSE */
    crix_ware_tbl();
    break;
  case 3: /* DISTRICT  */
    crix_dist_tbl();
    break;
  case 4: /* ITEM      */
    crix_item_tbl();
    break;
  case 5: /* STOCK     */
    crix_stock_tbl();
    break;
  case 6: /* CUSTOMER  */
    crix_cust_tbl();
    break;
  case 7: /* HISTORY   */
    crix_hist_tbl();
    break;
  case 8: /* ORDERS    */
    crix_ordr_tbl();
    break;
  case 9: /* NEW_ORDER    */
    crix_nu_ord_tbl();
    break;
  case 10: /* ORDER_LINE   */
    crix_oline_tbl();
    break;
  default:
    fprintf(stdout, "error: invalid option. \n");       break;
  }
  EXEC SQL COMMIT WORK;         SQLERR(&sqlca, "COMMIT");
  EXEC SQL CONNECT RESET;       SQLERR(&sqlca, "CONNECT
RESET");
}

/*----------------------------------------------------------------------*/
/*      create index for item table                                     */
/*----------------------------------------------------------------------*/
void crix_item_tbl( void )
{
  fprintf(stderr, "Create index for ITEM table.\n");
  timestamp1 = current_time();
  EXEC SQL DROP INDEX item_idx1;
  EXEC SQL CREATE UNIQUE INDEX item_idx1 on ITEM(i_id) ;
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "Create index for ITEM");
  EXEC SQL COMMIT WORK;
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "Commit");
  timestamp2 = current_time();
  elapse = timestamp2 - timestamp1;
  printf("Item table index created in %8.2f seconds.\n",elapse);
}

/*----------------------------------------------------------------------*/
/*      create index for stock table                                    */
/*----------------------------------------------------------------------*/

void crix_stock_tbl( void )
{
  /* WWW 94/6/6: Reversed order of columns in index for 100% clustering */
  fprintf(stderr, "Create index for STOCK table.\n");
  timestamp1 = current_time();
  EXEC SQL DROP INDEX stock_idx1;
  EXEC SQL CREATE UNIQUE INDEX stock_idx1 on STOCK(s_w_id, s_i_id) ;
  /*EXEC SQL CREATE UNIQUE INDEX stock_idx1 on STOCK(s_i_id, s_w_id) ;*/
  if (sqlca.sqlcode != 0)     SQLERR(&sqlca, "Create index for STOCK");
  EXEC SQL COMMIT WORK;
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "Commit");
  timestamp2 = current_time();
  elapse = timestamp2 - timestamp1;
  printf("Stock table index created in %8.2f seconds.\n",elapse);
}

/*----------------------------------------------------------------------*/
/*      create index for warehouse table                                */
/*----------------------------------------------------------------------*/
void crix_ware_tbl( void )
{
  fprintf(stderr, "Create index for WAREHOUSE table.\n");
  timestamp1 = current_time();
  EXEC SQL drop index ware_idx1;
  EXEC SQL create unique index ware_idx1 on WAREHOUSE(W_ID) ;
  if (sqlca.sqlcode != 0)     SQLERR(&sqlca, "Create index for WAREHOUSE");
  EXEC SQL COMMIT WORK;
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "Commit");
  timestamp2 = current_time();
  elapse = timestamp2 - timestamp1;
  printf("Warehouse index created in %8.2f seconds.\n",elapse);
}

/*----------------------------------------------------------------------*/
/*      create index for dist table                                     */
/*----------------------------------------------------------------------*/
void crix_dist_tbl( void )
{
  fprintf(stderr, "Create index for DISTRICT table.\n");
  timestamp1 = current_time();
  EXEC SQL drop index dist_idx1;
  EXEC SQL create unique index dist_idx1 on DISTRICT(d_id, d_w_id) ;
  if (sqlca.sqlcode != 0)     SQLERR(&sqlca, "Create index for DISTRICT");
  EXEC SQL COMMIT WORK;
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "Commit");
  timestamp2 = current_time();
  elapse = timestamp2 - timestamp1;
  printf("District table index created in %8.2f seconds.\n",elapse);
}

/*----------------------------------------------------------------------*/
/*      create index for customer table                                 */
/*----------------------------------------------------------------------*/
void crix_cust_tbl( void )
{
  fprintf(stderr, "Create index for CUSTOMER table.\n");

  timestamp1 = current_time();
  EXEC SQL drop index cust_idx1;
  EXEC SQL create unique index cust_idx1 on CUSTOMER(c_w_id, c_d_id, c_id) ;
  if (sqlca.sqlcode != 0)     SQLERR(&sqlca, "Create index1 for CUSTOMER");
  EXEC SQL drop index cust_idx2;
  EXEC SQL create index cust_idx2 on CUSTOMER(c_w_id, c_d_id, c_last, c_first,
c_id) ;
```

```
  if (sqlca.sqlcode != 0)     SQLERR(&sqlca, "Create index2 for
CUSTOMER");
  EXEC SQL COMMIT WORK;
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "Commit");
  timestamp2 = current_time();
  elapse = timestamp2 - timestamp1;
  printf("Customer table index created in %8.2f seconds.\n",elapse);

}

/*---------------------------------------------------------------------*/
/*      create index for hist table                                    */
/*---------------------------------------------------------------------*/
void crix_hist_tbl( void )
{
  fprintf(stderr, "No index is to be built for HISTORY table.\n");
}


/*---------------------------------------------------------------------*/
/*      create index for nu_ord table                                  */
/*---------------------------------------------------------------------*/
void crix_nu_ord_tbl( void )
{
  fprintf(stderr, "Create index for NEW_ORDER table.\n");
  timestamp1 = current_time();
  EXEC SQL drop index nu_ord_idx1;
  EXEC SQL create unique index nu_ord_idx1 on NEW_ORDER
    (no_w_id, no_d_id, no_o_id);
  if (sqlca.sqlcode != 0)     SQLERR(&sqlca, "Create index for
NEW_ORDER");
  EXEC SQL COMMIT WORK;
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "Commit");
  timestamp2 = current_time();
  elapse = timestamp2 - timestamp1;
  printf("New_order table index created in %8.2f seconds.\n",elapse);
}


/*---------------------------------------------------------------------*/
/*      create index for order table                                   */
/*---------------------------------------------------------------------*/

void crix_ordr_tbl( void )
{
  fprintf(stderr, "Create index for ORDERS table.\n");
  timestamp1 = current_time();
  EXEC SQL drop index ordr_idx1;
  EXEC SQL create unique index ordr_idx1
         on ORDERS(o_w_id, o_d_id, o_id) ;
  if (sqlca.sqlcode != 0)     SQLERR(&sqlca, "Create index1 for
ORDERS");
  EXEC SQL drop index ordr_idx2;
  EXEC SQL create unique index ordr_idx2
         on ORDERS(o_w_id, o_d_id, o_c_id, o_id desc);
  if (sqlca.sqlcode != 0)     SQLERR(&sqlca, "Create index2 for
ORDERS");
  EXEC SQL COMMIT WORK;
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "Commit");
  timestamp2 = current_time();
  elapse = timestamp2 - timestamp1;
  printf("Orders table index created in %8.2f seconds.\n",elapse);
}

/*---------------------------------------------------------------------*/
/*      create index for order-line table                              */
/*---------------------------------------------------------------------*/

void crix_oline_tbl( void )
{
  fprintf(stderr, "Create index for ORDER_LINE table.\n");
  timestamp1 = current_time();
  EXEC SQL drop index oline_idx1;
  EXEC SQL create unique index oline_idx1
         on ORDER_LINE(ol_w_id, ol_d_id, ol_o_id, ol_number) ;
  if (sqlca.sqlcode != 0)     SQLERR(&sqlca, "Create index for ORDERS_LINE");
  EXEC SQL COMMIT WORK;
  if (sqlca.sqlcode != 0)    SQLERR(&sqlca, "Commit");
  timestamp2 = current_time();
  elapse = timestamp2 - timestamp1;
  printf("Order_line table index created in %8.2f seconds.\n",elapse);
}

/*********************************************************************/
/*      file:  gendata.sqc                                           */
/*                                                                   */
/*      TPC-C benchmark of DB2 for Solaris. This program creates the   */
/*      randomly generated data and loads the data into the tables.    */
/*                                                                   */
/*********************************************************************/

#include "ctest.h"
#include "../include/lval.h"
#include <stdlib.h>

#define COMMIT_CNT      1000
#define C_OL_I_ID       1723
#define A_OL_I_ID       8191

EXEC SQL include sqlca ;
EXEC SQL include sqlda ;

/*  PROTOTYPES.  */
void load_dist_tbl( void );
void load_cust_tbl( void );
void load_hist_tbl( void );
void load_nu_ord_tbl( void );
void load_ordr_tbl( void );
void load_item_tbl( void );
void load_stock_tbl( void );
void load_ware_tbl( void );

EXEC SQL BEGIN DECLARE SECTION;              /* Declare Host Variables  */
char   errmsg[201];
long   ware_num ;
long   dist_num ;
long   cust_num ;
long   ord_num ;
long   oline_num ;
long   count ;
long   ware_count ;
double currtime;
char   dbname[8] = "dbbench";
char   currtmstmp[27];
EXEC SQL END DECLARE SECTION;
```

```
int i, j, hit ;
int using_db2 = 1;
double timestamp1,        timestamp2,        elapse;

int load_order_table=1;
int load_order_line_table=1;


/*---------------------------------------------------------------------*/
/*      main                                                           */
/*---------------------------------------------------------------------*/


int main (int argc, char *argv[])
{
  int option;
  char str[80];
  char msg[30];
  char *dbname_ptr;

  if (argc == 2)
  {
    using_db2 = 0;
    option = atoi(argv[1]);
  }
  else
  {
    fprintf(stdout,"Pick table to be loaded: \n");
    fprintf(stdout,"------------------------\n");
    fprintf(stdout,"1. all tables \n\n");
    fprintf(stdout,"2. warehouse table\n");
    fprintf(stdout,"3. district table\n");
    fprintf(stdout,"4. item table\n");
    fprintf(stdout,"5. stock table\n");
    fprintf(stdout,"6. customer table\n");
    fprintf(stdout,"7. history table\n");
    fprintf(stdout,"8. order and order-line tables\n");
    fprintf(stdout,"9. new-order table\n\n");
    fprintf(stdout,"88. order table\n");
    fprintf(stdout,"89. order-line table\n");
    fprintf(stdout,"------------------\n");
    fprintf(stdout,"\nEnter option: ");
    scanf("%d", &option);
  }

  if (using_db2) {
    if ((dbname_ptr = getenv("TPCC_DBNAME")) != NULL) {
      strcpy (dbname, dbname_ptr);
    }
    EXEC SQL CONNECT TO :dbname IN SHARE MODE;
    fprintf(stderr, "\nStart using database %s\n\n", dbname);
    if(sqlca.sqlcode != 0)
    {
      SQLERR(&sqlca, "CONNECT");
    }
  }

  ware_count = WAREHOUSES;

  switch (option) {
   case 1: /* ALL */
    load_ware_tbl();
    load_dist_tbl();
    load_item_tbl();
    load_stock_tbl();
    load_cust_tbl();
    load_hist_tbl();
    load_ordr_tbl();
    load_nu_ord_tbl();
    break;
   case 2: /* WAREHOUSE */
    load_ware_tbl();
    break;
   case 3: /* DISTRICT  */
    load_dist_tbl();
    break;
   case 4: /* ITEM      */
    load_item_tbl();
    break;
   case 5: /* STOCK     */
    load_stock_tbl();
    break;
   case 6: /* CUSTOMER  */
    load_cust_tbl();
    break;
   case 7: /* HISTORY   */
    load_hist_tbl();
    break;
   case 8: /* ORDERS and ORDER_LINE   */
    load_ordr_tbl();
    break;
   case 9: /* NEW_ORDER */
    load_nu_ord_tbl();
    break;
   case 88: /* ORDERS   */
    load_order_line_table=0;
    load_ordr_tbl();
    break;
   case 89: /* ORDER_LINE   */
    load_order_table=0;
    load_ordr_tbl();
    break;
   default:
    fprintf(stderr, "Error: invalid option = %d \n",(option));
    break;
  }

  if (using_db2) {
    EXEC SQL COMMIT WORK;
    if(sqlca.sqlcode != 0)
    {
      SQLERR(&sqlca, "Commit");
    }
    EXEC SQL CONNECT RESET;
    if(sqlca.sqlcode != 0)
    {
      SQLERR(&sqlca, "Stop using database");
    }
  }

  fprintf(stderr, "Finished. \n");
  return 0;
}

/*---------------------------------------------------------------------*/
```

```
/*      load item table                                         */
/*--------------------------------------------------------------------*/
void load_item_tbl( void )
{
  EXEC SQL BEGIN DECLARE SECTION;/* Declare Host Variables
*/
  long  item_num ;
  long  item_im_id ;
  char  item_name[25] ;
  long  item_price ;
  char  item_data[51] ;
  EXEC SQL END DECLARE SECTION;

  long ins_cnt = COMMIT_CNT;
  sqlca.sqlcode = 0;
  initialize_random(13,42);

  timestamp1 = current_time();
  fprintf(stderr, "Now loading ITEMs");
  for (item_num = 1; item_num <= ITEMS; item_num++)
  {
    /* create image id field */
    item_im_id = rand_integer( 1, 10000 ) ;
    /* create name field */
    create_random_a_string( item_name, 14, 24);
    /* create price field */
    item_price = rand_integer( 100, 10000 ) ;
    /* create ORIGINAL field */
    create_a_string_with_original( item_data, 26, 50, 10.5, &hit ) ;

    if (!using_db2)
    {
      printf ("%d|%d|%s|%d|%s\n", item_num, item_im_id,
item_name, item_price, item_data);
    }
    else
    {
      EXEC SQL
        insert into ITEM
        values(:item_num, :item_im_id, :item_name, :item_price,
:item_data) ;
      if(sqlca.sqlcode != 0)
      {
        SQLERR(&sqlca, "INSERT ITEM");
      }
      else
      {
        --ins_cnt;
        if (ins_cnt == 0)
        {
          ins_cnt = COMMIT_CNT;
          fprintf(stderr, ".");
          EXEC SQL commit work ;
          if (sqlca.sqlcode != 0) SQLERR(&sqlca, "COMMIT work");
        }
      }
    }
  }
  fprintf(stderr, "..loaded\n");

  if (using_db2)
  {
    EXEC SQL commit work ;
    if(sqlca.sqlcode != 0) SQLERR(&sqlca, "COMMIT work");
  }
  timestamp2 = current_time();
  elapse = timestamp2 - timestamp1;
  fprintf(stderr,"Item table loaded in %8.2f seconds.\n\n",elapse);
  return;
}

/*--------------------------------------------------------------------*/
/*      load stock table                                        */
/*--------------------------------------------------------------------*/

void load_stock_tbl( void )
{
  EXEC SQL BEGIN DECLARE SECTION;          /* Declare Host Variables */
  long  stock_num  ;
  long  stock_quant ;
  longs_ytd;
  longs_order_cnt, s_remote_cnt;
  char  stock_dist_01[25] ;
  char  stock_dist_02[25] ;
  char  stock_dist_03[25] ;
  char  stock_dist_04[25] ;
  char  stock_dist_05[25] ;
  char  stock_dist_06[25] ;
  char  stock_dist_07[25] ;
  char  stock_dist_08[25] ;
  char  stock_dist_09[25] ;
  char  stock_dist_10[25] ;
  char  stock_data[51] ;
  EXEC SQL END DECLARE SECTION;

  long ins_cnt = COMMIT_CNT;
  sqlca.sqlcode = 0;
  initialize_random(7,11);

  timestamp1 = current_time();
  for (stock_num = 1; stock_num <= STOCK_PER_WAREHOUSE; stock_num++)
  {
    if (stock_num%500 == 0) fprintf(stderr, "Now loading Stock for item
#%d",stock_num);
    for (ware_num = 1; ware_num <= ware_count; ware_num++)
    {
      stock_quant = rand_integer( 10, 100 ) ;
      create_random_a_string( stock_dist_01, 24, 24) ;  /* create text field */
      create_random_a_string( stock_dist_02, 24, 24) ;  /* create text field */
      create_random_a_string( stock_dist_03, 24, 24) ;  /* create text field */
      create_random_a_string( stock_dist_04, 24, 24) ;  /* create text field */
      create_random_a_string( stock_dist_05, 24, 24) ;  /* create text field */
      create_random_a_string( stock_dist_06, 24, 24) ;  /* create text field */
      create_random_a_string( stock_dist_07, 24, 24) ;  /* create text field */
      create_random_a_string( stock_dist_08, 24, 24) ;  /* create text field */
      create_random_a_string( stock_dist_09, 24, 24) ;  /* create text field */
      create_random_a_string( stock_dist_10, 24, 24) ;  /* create text field */

      /* create ORIGINAL field */
      create_a_string_with_original( stock_data, 26, 50, 10.5, &hit ) ;
      s_ytd = s_order_cnt = s_remote_cnt = 0;

      if (!using_db2)
      {
```

```
            printf
("%d|%hd|%hd|%hd|%hd|%d|%s|%s|%s|%s|%s|%s|%s|%s|%
s|%s|%s|\n", stock_num, ware_num, s_remote_cnt, stock_quant,
s_order_cnt, s_ytd, stock_dist_01, stock_dist_02, stock_dist_03,
stock_dist_04, stock_dist_05, stock_dist_06, stock_dist_07,
stock_dist_08, stock_dist_09, stock_dist_10, stock_data);
          }
          else
          {
            EXEC SQL
              insert into STOCK
              values(:stock_num, :ware_num, :s_remote_cnt, :stock_quant,
                     :s_order_cnt, :s_ytd,
                     :stock_dist_01, :stock_dist_02,
                     :stock_dist_03, :stock_dist_04,
                     :stock_dist_05, :stock_dist_06,
                     :stock_dist_07, :stock_dist_08,
                     :stock_dist_09, :stock_dist_10,
                     :stock_data ) ;
            if(sqlca.sqlcode != 0)
            {
              SQLERR(&sqlca, "INSERT STOCK");
            }
            else
            {
              --ins_cnt;
              if (ins_cnt == 0)
              {
                ins_cnt = COMMIT_CNT;
                fprintf(stderr, ".");
                EXEC SQL commit work ;
               if (sqlca.sqlcode != 0) SQLERR(&sqlca, "COMMIT work");
              }
            }
          }
        }
          fprintf(stderr, "..loaded\n");
    }

    if (using_db2)
    {
      EXEC SQL commit work ;
      if(sqlca.sqlcode != 0) SQLERR(&sqlca, "commit work");
    }
    timestamp2 = current_time();
    elapse = timestamp2 - timestamp1;
    fprintf(stderr,"Stock table loaded in %8.2f seconds.\n\n",elapse);
    return;
}

/*---------------------------------------------------------------------*/
/*      load warehouse table                                           */
/*---------------------------------------------------------------------*/
void load_ware_tbl( void )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char ware_name[11] ;
    char ware_street_1[21] ;
    char ware_street_2[21] ;
    char ware_city[21] ;
    char ware_state[3]  ;
    char ware_zip[10]  ;
    double ware_tax ;
    EXEC SQL END DECLARE SECTION;

    long ins_cnt = COMMIT_CNT;
    sqlca.sqlcode = 0;
    initialize_random(23,111);

    timestamp1 = current_time();
    fprintf(stderr, "Now loading Warehouse\n");
    for (ware_num = 1; ware_num <= ware_count; ware_num++)
    {
      fprintf(stderr, "\tWarehouse #%d",ware_num);
      create_random_a_string( ware_name,      6,10) ; /* create name */
      create_random_a_string( ware_street_1, 10,20) ; /* create street 1 */
      create_random_a_string( ware_street_2, 10,20) ; /* create street 2 */
      create_random_a_string( ware_city,     10,20) ; /* create city */
      create_random_a_string( ware_state,     2,2) ;  /* create state */
      create_random_n_string( ware_zip,       4,4) ;  /* create zip */
      strcat(ware_zip, "11111");
      /*ware_tax = create_random_float_val_return( 0.0, 0.2000 ) ;*/
      ware_tax = ((double) rand_integer(0, 2000)) / (double)10000.0;
      if (!using_db2)
      {
        printf ("%hd|%s|%s|%s|%s|%s|%s|%f|%f|\n", ware_num, ware_name,
ware_street_1, ware_street_2, ware_city, ware_state, ware_zip, ware_tax, 300000.0);
      }
      else
      {
        EXEC SQL
          insert into WAREHOUSE
          values(:ware_num, :ware_name, :ware_street_1, :ware_street_2,
                 :ware_city, :ware_state, :ware_zip, :ware_tax, 300000.0 ) ;
        if (sqlca.sqlcode != 0)
        {
          SQLERR(&sqlca, "INSERT WAREHOUSE");
        }
        else
        {
          --ins_cnt;
          if (ins_cnt == 0)
          {
            ins_cnt = COMMIT_CNT;
            EXEC SQL commit work ;
            if (sqlca.sqlcode != 0) SQLERR(&sqlca, "COMMIT work");
          }
        }
      }
      fprintf(stderr, "..loaded\n");
    }

    if (using_db2)
    {
      EXEC SQL commit work ;
      if(sqlca.sqlcode != 0) SQLERR(&sqlca, "commit work");
    }
    timestamp2 = current_time();
    elapse = timestamp2 - timestamp1;
    fprintf(stderr,"Warehouse table loaded in %8.2f seconds.\n\n",elapse);
    return;
}

/*---------------------------------------------------------------------*/
/*      load dist table                                                */
/*---------------------------------------------------------------------*/
```

```
void load_dist_tbl( void )
{
  EXEC SQL BEGIN DECLARE SECTION;
  char dist_name[11];
  char dist_street_1[21];
  char dist_street_2[21];
  char dist_city[21];
  char dist_state[3];
  char dist_zip[10];
  double dist_tax;
  long  next_o_id;
  EXEC SQL END DECLARE SECTION;

  long ins_cnt = COMMIT_CNT;
  next_o_id = CUSTOMERS_PER_DISTRICT + 1;
  sqlca.sqlcode = 0;
  initialize_random(44,73);

  timestamp1 = current_time();
  for (ware_num = 1; ware_num <= ware_count; ware_num++)
  {
    fprintf(stderr, "Now loading Districts for Warehouse #%d\n",
ware_num);
    for (dist_num = 1; dist_num <= DISTRICTS_PER_WAREHOUSE;
dist_num++)
    {

      create_random_a_string( dist_name,     6,10) ; /* create name */
      create_random_a_string( dist_street_1, 10,20) ; /* create street 1
*/
      create_random_a_string( dist_street_2, 10,20) ; /* create street 2
*/
       create_random_a_string( dist_city,     10,20) ; /* create city */
       create_random_a_string( dist_state,     2,2) ;  /* create state */
       create_random_n_string( dist_zip,       4,4) ;  /* create zip */
       strcat(dist_zip, "11111");
       /*dist_tax = create_random_float_val_return( 0.0, 0.2000 ) ; */
       dist_tax = ((double) rand_integer(0, 2000)) / (double)10000.0;

       if (!using_db2)
       {
         printf
("%hd|%hd|%s|%s|%s|%s|%s|%s|%f|%f|%d\n",dist_num,ware
_num,dist_name,dist_street_1,dist_street_2,dist_city,dist_state,dist_zi
p,dist_tax,30000.0,next_o_id);
       }
       else
       {
         EXEC SQL
           insert into DISTRICT
           values(:dist_num, :ware_num, :dist_name, :dist_street_1,
                 :dist_street_2, :dist_city, :dist_state, :dist_zip,
                 :dist_tax, 30000.0, :next_o_id ) ;
         if (sqlca.sqlcode != 0)
         {
           SQLERR(&sqlca, "");
         }
         else
         {
           --ins_cnt;
           if (ins_cnt == 0)
           {
             ins_cnt = COMMIT_CNT;
             EXEC SQL commit work ;
             if (sqlca.sqlcode != 0) SQLERR(&sqlca, "COMMIT work");
           }
         }
       }
       fprintf(stderr,
             "\tDistrict %ld loaded    sqlcode=%d\n",
             dist_num, sqlca.sqlcode);
     }
   }

   if (using_db2)
   {
     EXEC SQL commit work ;
     if (sqlca.sqlcode != 0) SQLERR(&sqlca, "COMMIT work");
   }
   timestamp2 = current_time();
   elapse = timestamp2 - timestamp1;
   fprintf(stderr,"District table loaded in %8.2f seconds.\n\n",elapse);
   return;
}

/*-------------------------------------------------------------------*/
/*      load customer table                                          */
/*-------------------------------------------------------------------*/
void load_cust_tbl( void )
{
  EXEC SQL BEGIN DECLARE SECTION;
  char cust_last[17];
  char cust_middle[3];
  char cust_first[17];
  char cust_street_1[21];
  char cust_street_2[21];
  char cust_city[21];
  char cust_state[3];
  char cust_zip[10];
  char cust_phone[17];
  char cust_credit[3];
  char cust_data1[251];
  char cust_data2[251];

  char tmp_str[3];

  double cust_discount;
  double luck_of_the_draw;
  EXEC SQL END DECLARE SECTION;

  int ph1, ph2, ph3, ph4;
  long ins_cnt = COMMIT_CNT;
  sqlca.sqlcode = 0;
  initialize_random(10,64);

  timestamp1 = current_time();
  sprintf(tmp_str, "OE");
  strcpy(currtmstmp, current_tmstmp());

  for (cust_num = 1; cust_num <= CUSTOMERS_PER_DISTRICT; cust_num++)
  {
    if (cust_num%50 == 0) fprintf(stderr, "Now loading Customer #%d:\n",
cust_num);
    for (ware_num = 1; ware_num <= ware_count; ware_num++)
    {
```

```
        for (dist_num = 1; dist_num <=
DISTRICTS_PER_WAREHOUSE; dist_num++)
        {
          if (cust_num <= 1000) /* create last name */
            create_random_last_name( cust_last, cust_num);
          else                  /* create last name */
            create_random_last_name( cust_last, 0);
          create_random_a_string( cust_first,     8,16) ; /* create first
name */
          create_random_a_string( cust_street_1, 10,20) ; /* create street
1 */
          create_random_a_string( cust_street_2, 10,20) ; /* create street
2 */
          create_random_a_string( cust_city,     10,20) ; /* create city */
          create_random_a_string( cust_state,     2,2) ;  /* create state */
          create_random_n_string( cust_zip,       4,4) ;  /* create zip */
          strcat(cust_zip, "11111");
          /* create phone number */
          create_random_n_string( cust_phone,    16,16) ;
          luck_of_the_draw = create_random_float_val_return( 0, 100.0
) ;
          if ( luck_of_the_draw < 10.2 )
            strcpy( cust_credit, "BC" ) ;
          else
            strcpy( cust_credit, "GC" ) ;
          /* create discount rate */
          /*cust_discount = create_random_float_val_return( 0.0, 0.5000
); */
          cust_discount = ((double) rand_integer(0, 5000)) /
(double)10000.0;
          /* create customer data */
          create_random_a_string( cust_data1, 250, 250);
          create_random_a_string( cust_data2, 50,  250);

          if (!using_db2)
          {
            printf
("%d|%hd|%hd|%s|%s|%s|%s|%s|%s|%s|%s|%s|%f|%
f|%hd|%f|%f|%hd|%s|%s\n",cust_num,dist_num,
ware_num,cust_first,tmp_str,cust_last,
cust_street_1,cust_street_2,cust_city,cust_state,cust_zip,cust_phone,cu
rrtmstmp,cust_credit,50000.00,cust_discount,0,-10.0,10.0, 1,
cust_data1,cust_data2);
          }
          else
          {
            EXEC SQL
              insert into CUSTOMER
              values(:cust_num, :dist_num, :ware_num, :cust_first,
                     :tmp_str, :cust_last, :cust_street_1,
                     :cust_street_2, :cust_city, :cust_state, :cust_zip,
                     :cust_phone, :currtmstmp, :cust_credit,
                     50000.00, :cust_discount, 0, -10.0,
                     10.0, 1, :cust_data1, :cust_data2 ) ;
            if (sqlca.sqlcode != 0)
            {
              SQLERR(&sqlca, "INSERT CUSTOMER");
            }
            else
            {
              --ins_cnt;
              if (ins_cnt == 0)
              {
                ins_cnt = COMMIT_CNT;
                fprintf(stderr, ".");
                EXEC SQL commit work ;
                if (sqlca.sqlcode != 0) SQLERR(&sqlca, "COMMIT work");
              }
            }
          }
        }
        fprintf(stderr, "..loaded\n");
      }
    }

    if (using_db2)
    {
      EXEC SQL commit work;
      if (sqlca.sqlcode != 0) SQLERR(&sqlca, "COMMIT work");
    }
    timestamp2 = current_time();
    elapse = timestamp2 - timestamp1;
    fprintf(stderr,"Customer table loaded in %8.2f seconds.\n\n",elapse);
    return;
}

/*-------------------------------------------------------------------------*/
/*      load hist table                                                    */
/*-------------------------------------------------------------------------*/
void load_hist_tbl( void )
{
    EXEC SQL BEGIN DECLARE SECTION;
    char hist_data[25] ;
    EXEC SQL END DECLARE SECTION;

    long ins_cnt = COMMIT_CNT;
    sqlca.sqlcode = 0;
    initialize_random(15,63);

    timestamp1 = current_time();
    strcpy(currtmstmp, current_tmstmp());
    for (ware_num = 1; ware_num <= ware_count; ware_num++)
    {
      fprintf(stderr, "Now loading History for Warehouse #%d:\n", ware_num);
      for (dist_num = 1; dist_num <= DISTRICTS_PER_WAREHOUSE; dist_num++)
      {
        fprintf(stderr, "\tDistrict #%d ", dist_num);
        for (cust_num = 1; cust_num <= CUSTOMERS_PER_DISTRICT;
cust_num++)
        {
          /* create history data */
          create_random_a_string( hist_data, 12,24) ;

          if (!using_db2)
          {
           printf ("%d|%hd|%hd|%hd|%hd|%s|%d|%s\n",cust_num, dist_num,
ware_num, dist_num,ware_num,currtmstmp, 1000,hist_data);
          }
          else
          {
            EXEC SQL
              insert into HISTORY
              values(:cust_num, :dist_num, :ware_num, :dist_num,
                     :ware_num, :currtmstmp, 1000,
                     :hist_data ) ;
            if (sqlca.sqlcode != 0)
```

```
            {
              SQLERR(&sqlca, "");
            }
            else
            {
              --ins_cnt;
              if (ins_cnt == 0)
              {
                ins_cnt = COMMIT_CNT;
                fprintf(stderr, ".");
                EXEC SQL commit work ;
                if (sqlca.sqlcode != 0) SQLERR(&sqlca, "COMMIT
work");
              }
            }
          }
        }
      fprintf(stderr, "..loaded\n");
      }
  }

  if (using_db2)
  {
    EXEC SQL commit work;
    if (sqlca.sqlcode != 0) SQLERR(&sqlca, "commit work");
  }
  timestamp2 = current_time();
  elapse = timestamp2 - timestamp1;
  fprintf(stderr,"History table loaded in %8.2f seconds.\n\n",elapse);
  return;
}


/*---------------------------------------------------------------------*/
/*      load nu_ord table                                              */
/*---------------------------------------------------------------------*/
void load_nu_ord_tbl( void )
{
  EXEC SQL BEGIN DECLARE SECTION;
  long nu_ord_id ;
  EXEC SQL END DECLARE SECTION;

  int nu_ord_hi ;
  long ins_cnt = COMMIT_CNT;

  timestamp1 = current_time();
  /* compute maximum and minimum
     order numbers for this
     district */
  nu_ord_hi = CUSTOMERS_PER_DISTRICT -
NU_ORDERS_PER_DISTRICT + 1;
  if (nu_ord_hi < 0) {
     nu_ord_hi = CUSTOMERS_PER_DISTRICT -
(CUSTOMERS_PER_DISTRICT / 3) + 1;
     fprintf(stderr,"\n**** WARNING ****
NU_ORDERS_PER_DISTRICT is >
CUSTOMERS_PER_DISTRICT\n");
     fprintf(stderr,"              Check the values in file lval.h\n");
     fprintf(stderr,"              Loading New-Order with 1/3 of
CUSTOMERS_PER_DISTRICT\n");
  }
  sqlca.sqlcode = 0;
  initialize_random(99,37);

  for (ware_num = 1; ware_num <= ware_count; ware_num++)
  {
    fprintf(stderr, "Now loading New-Order for Warehouse #%d:\n", ware_num);
    for (dist_num = 1; dist_num <= DISTRICTS_PER_WAREHOUSE; dist_num++)
    {
      fprintf(stderr, "\tDistrict #%d ", dist_num);
      for (nu_ord_id = nu_ord_hi;
           nu_ord_id <= CUSTOMERS_PER_DISTRICT;
           nu_ord_id++)
      {
        if (!using_db2)
        {
          printf ("%d|%hd|%hd\n", nu_ord_id, dist_num,ware_num);
        }
        else
        {
          EXEC SQL
            insert into NEW_ORDER
            values(:nu_ord_id, :dist_num, :ware_num ) ;
          if (sqlca.sqlcode != 0)
          {
            SQLERR(&sqlca, "INSERT NEW_ORDER");
          }
          else
          {
            --ins_cnt;
            if (ins_cnt == 0)
            {
              ins_cnt = COMMIT_CNT;
              fprintf(stderr, ".");
              EXEC SQL commit work ;
              if (sqlca.sqlcode != 0) SQLERR(&sqlca, "COMMIT work");
            }
          }
        }
      }
      fprintf(stderr, "..loaded\n");
    }
  }

  if (using_db2)
  {
    EXEC SQL commit work ;
    if (sqlca.sqlcode != 0) SQLERR(&sqlca, "COMMIT work");
  }
  timestamp2 = current_time();
  elapse = timestamp2 - timestamp1;
  fprintf(stderr,"New-Order table loaded in %8.2f seconds.\n\n",elapse);
  return;
}


/*---------------------------------------------------------------------*/
/*      load order and order_line tables                               */
/*---------------------------------------------------------------------*/

void load_ordr_tbl( void )
{
  FILE *orders_fp;
  EXEC SQL BEGIN DECLARE SECTION;
  long ordr_carrier_id;
  long ordr_cnt;
```

```
    long ordr_ol_cnt;
    long oline_ol_num;
    long oline_ol_cnt;
    long oline_item_num;
    long ordr_all_local;

    long oline_amount;
    char oline_dist_info[25];
    char null_tmstmp[27];
    EXEC SQL END DECLARE SECTION;

    char null_ts[] = "0001-01-01-00.00.01.000000";
    long ins_cnt = COMMIT_CNT;
    oline_dist_info[24] = '\0';

    timestamp1 = current_time();
    strcpy(currtmstmp, current_tmstmp());
    strcpy(null_tmstmp, null_ts);
    initialize_random(42,13);

        if (!using_db2 && load_order_table && load_order_line_table) {
      if ((orders_fp = fopen("ASCII_ORDERS_TABLE", "w")) == NULL)
      {
        fprintf(stderr, "Error opening ASCII_ORDERS_TABLE\n");
        return;
      }
    }

    for (ware_num = 1; ware_num <= ware_count; ware_num++)
    {
      fprintf(stderr, "Now loading ");
      if (load_order_table && load_order_line_table)
        fprintf(stderr,"Order & Order-Line ");
      else if (load_order_table)
        fprintf(stderr,"Order ");
      else
        fprintf(stderr,"Order-Line ");
      fprintf(stderr, "table(s) for Warehouse #%d:\n", ware_num);
      for (dist_num = 1; dist_num <= DISTRICTS_PER_WAREHOUSE;
dist_num++)
      {
        fprintf(stderr, "\tDistrict #%d ", dist_num);
        seed_1_3000();
        for (ord_num = 1; ord_num <= CUSTOMERS_PER_DISTRICT;
ord_num++)
        {
          if (ord_num < 2101)
            ordr_carrier_id = rand_integer( 1, 10 ) ;
          else
            ordr_carrier_id = 0;

          cust_num = random_1_3000();
          ordr_ol_cnt =
rand_integer(MIN_OL_PER_ORDER,MAX_OL_PER_ORDER);
          ordr_all_local = 1;
          if (load_order_table) {
            if (!using_db2) {
              if (load_order_line_table) {
                fprintf
(orders_fp,"%d|%d|%d|%d|%s|%d|%d|%d|\n",
                        ord_num, cust_num, dist_num, ware_num,
currtmstmp,
                        ordr_carrier_id, ordr_ol_cnt, 1 ) ;
            }
            else {
              printf ("%d|%d|%d|%d|%s|%d|%d|%d|\n",
                      ord_num, cust_num, dist_num, ware_num, currtmstmp,
                      ordr_carrier_id, ordr_ol_cnt, 1 ) ;
            }
          }
          else
          {
            EXEC SQL insert into ORDERS
              values( :ord_num, :cust_num, :dist_num, :ware_num,
                    :currtmstmp, :ordr_carrier_id,
                    :ordr_ol_cnt, 1 ) ;
            if (sqlca.sqlcode != 0) SQLERR(&sqlca, "16");
            --ins_cnt;
            if (ins_cnt == 0)
            {
              ins_cnt = COMMIT_CNT;
              fprintf(stderr, ".");
              EXEC SQL commit work;
              if (sqlca.sqlcode != 0) SQLERR(&sqlca, "commit work");
            }
          }
        }

        sqlca.sqlcode = 0;
        for ( oline_ol_num = 1; oline_ol_num <= ordr_ol_cnt; oline_ol_num++ )
        {
          oline_item_num   = rand_integer(1, ITEMS) ;
          create_random_a_string( oline_dist_info, 24, 24) ;
          oline_amount = rand_integer(001, 999999);
          if (load_order_line_table) {
            if (ord_num < 2101)
            {
              oline_amount = 0;
              if (!using_db2)
              {
                printf ("%d|%hd|%hd|%hd|%d|%hd|%s|%hd|%d|%s\n",
ord_num,
                        dist_num, ware_num, oline_ol_num, oline_item_num,
                        ware_num, currtmstmp, 5, oline_amount, oline_dist_info);
              }
              else
              {
                EXEC SQL
                  insert into ORDER_LINE
                  values( :ord_num, :dist_num, :ware_num, :oline_ol_num,
                        :oline_item_num, :ware_num,
                        :currtmstmp, 5, :oline_amount,
                        :oline_dist_info ) ;
                if (sqlca.sqlcode != 0) SQLERR(&sqlca, "77");
              }
            }
            else
            {
              if (!using_db2)
              {
                printf ("%d|%hd|%hd|%hd|%d|%hd|%s|%hd|%d|%s\n",
                      ord_num, dist_num,
                      ware_num, oline_ol_num, oline_item_num, ware_num,
                      null_tmstmp, 5,
                      oline_amount, oline_dist_info);
              }
```

```
                else
                {
                  EXEC SQL
                    insert into ORDER_LINE
                    values( :ord_num, :dist_num, :ware_num,
:oline_ol_num,
                            :oline_item_num, :ware_num,
                            :null_tmstmp, 5, :oline_amount,
                            :oline_dist_info ) ;
                  if (sqlca.sqlcode != 0) SQLERR(&sqlca, "78");
                }
              }

              if (using_db2)
              {
                --ins_cnt;
                if (ins_cnt == 0)
                {
                  ins_cnt = COMMIT_CNT;
                  fprintf(stderr, ".");
                  EXEC SQL commit work;
                  if (sqlca.sqlcode != 0) SQLERR(&sqlca, "commit
work");
                }
              }
            }
          }
          fprintf(stderr, "..loaded\n");
        }
      }

      if (using_db2)
      {
        EXEC SQL commit work ;
        if (sqlca.sqlcode != 0) SQLERR(&sqlca, "commit work");
      }
          if (!using_db2 && load_order_table && load_order_line_table) {
                fclose(orders_fp);
          }
      timestamp2 = current_time();
      elapse = timestamp2 - timestamp1;
      if (load_order_table && load_order_line_table)
        fprintf(stderr,"Order & Order-Line ");
      else if (load_order_table)
        fprintf(stderr,"Order ");
      else
        fprintf(stderr,"Order-Line ");
      fprintf(stderr,"table(s) loaded in %8.2f seconds.\n\n",elapse);
      return;
}
```

Appendix C: Tunable Parameters

This Appendix contains the configuration information for the operating system, the RDBMS and Tuxedo.

Operating System Configuration Values

The Solaris 2.5.1 kernel configuration parameters set in the file /etc/system are given below.

Solaris 2.5.1 Configuration File for Ultra Enterprise 2 Model 2200

```
set tune_t_fsflushr = 50
set autoup = 300
set maxusers = 96
set bufhwm = 8000
set spt_max = 99

set pt_cnt=1024

set msgsys:msginfo_msgmax = 65535
set msgsys:msginfo_msgmnb = 65535
set msgsys:msginfo_msgmap = 402
set msgsys:msginfo_msgmni = 200
set msgsys:msginfo_msgssz = 4
set msgsys:msginfo_msgtql = 400
set msgsys:msginfo_msgseg = 16384
*
set semsys:seminfo_semmni = 128
set semsys:seminfo_semmap = 129
```

```
set semsys:seminfo_semmns = 800
set semsys:seminfo_semmnu = 512

*Shared memory parameters
set shmsys:shminfo_shmmax=0x7FFFFFFF
set shmsys:shminfo_shmmni=320
set shmsys:shminfo_shmseg=32

set md:mddb_bootlist1="ssd:0:16 ssd:8:16 ssd:16:16"
```

Solaris 2.5.1 configuration file for the client systems:

```
set pt_cnt=1400
set maxusers=192

set msgsys:msginfo_msgmax = 65535
set msgsys:msginfo_msgmnb = 65535
set msgsys:msginfo_msgmni = 1500
set msgsys:msginfo_msgmap = 6400
set msgsys:msginfo_msgtql = 6400
set msgsys:msginfo_msgseg = 32768
set msgsys:msginfo_msgssz = 128
*
set semsys:seminfo_semmni = 320
set semsys:seminfo_semmap = 400
set semsys:seminfo_semmns = 2400
set semsys:seminfo_semmnu = 2400
set semsys:seminfo_semume = 2400
set semsys:seminfo_semmsl = 200

set shmsys:shminfo_shmmax=0x20000000
set shmsys:shminfo_shmmni=320
set shmsys:shminfo_shmseg=32
```

RDBMS Configuration values

DB2 parameters :

```
     Database Manager Configuration

   Node type = Database Server with local and remote clients

 Database manager configuration release level          = 0x0600

 CPU speed (millisec/instruction)          (CPUSPEED) = 2.973565e-06
 Max number of concurrently active databases     (NUMDB) = 1
 Transaction processor monitor name       (TP_MON_NAME) =

 Diagnostic error capture level           (DIAGLEVEL) = 3
 Diagnostic data directory path             (DIAGPATH) =
```

```
Default database monitor switches
 Buffer pool                (DFT_MON_BUFPOOL) = OFF
 Lock                       (DFT_MON_LOCK) = OFF
 Sort                       (DFT_MON_SORT) = OFF
 Statement                  (DFT_MON_STMT) = OFF
 Table                      (DFT_MON_TABLE) = OFF
 Unit of work               (DFT_MON_UOW) = OFF

Database monitor SQL statement size (bytes) (SQLSTMTSZ) = 0

SYSADM group name           (SYSADM_GROUP) = DB2NEW
SYSCTRL group name         (SYSCTRL_GROUP) =
SYSMAINT group name         (SYSMAINT_GROUP) =

Database manager authentication  (AUTHENTICATION) = CLIENT

Default database path       (DFTDBPATH) = /db2/db2.new

Database monitor heap size (4KB)  (MON_HEAP_SZ) = 48
UDF shared memory set size (4KB)  (UDF_MEM_SZ) = 256

Backup buffer default size (4KB)  (BACKBUFSZ) = 1024
Restore buffer default size (4KB) (RESTBUFSZ) = 1024

Sort heap threshold (4KB)   (SHEAPTHRES) = 4096

Directory cache support     (DIR_CACHE) = YES

Application support layer heap size (4KB)  (ASLHEAPSZ) = 15
Max requester I/O block size (bytes)  (RQRIOBLK) = 32767
Query heap size (4KB)       (QUERY_HEAP_SZ) = 1000
DRDA services heap size (4KB)  (DRDA_HEAP_SZ) = 128

Priority of agents          (AGENTPRI) = SYSTEM
Max number of existing agents  (MAXAGENTS) = 144
Max number of concurrent agents  (MAXCAGENTS) = MAXAGENTS
Maximum number of idle agents  (MAX_IDLEAGENTS) = 2

Keep DARI process           (KEEPDARI) = YES
Max number of DARI processes  (MAXDARI) = 1

Index re-creation time      (INDEXREC) = RESTART

Transaction manager database name  (TM_DATABASE) =
Transaction resync interval (sec)  (RESYNC_INTERVAL) = 180

SPM name                    (SPM_NAME) =
SPM log size               (SPM_LOG_FILE_SZ) = 256
SPM resync agent limit      (SPM_MAX_RESYNC) = 20

Service name                (SVCENAME) = db2s00
Transaction program name    (TPNAME) =
Default accounting string   (DFT_ACCOUNT_STR) =

IPX/SPX fileserver name     (FILESERVER) =
IPX/SPX DB manager object name  (OBJECTNAME) =
```

```
IPX/SPX socket number                (IPX_SOCKET) = 879E

Directory services type               (DIR_TYPE) = NONE
Directory path name                (DIR_PATH_NAME) = /.:/subsys/database/
Directory object name              (DIR_OBJ_NAME) =
Routing information object name    (ROUTE_OBJ_NAME) =
Default client comm. protocols     (DFT_CLIENT_COMM) =

ADSM node name                     (ADSM_NODENAME) =
ADSM owner                         (ADSM_OWNER) =
ADSM password                      (ADSM_PASSWORD) =


   Database Configuration for Database dbbench

Database configuration release level            = 0x0600
Database release level                          = 0x0600

Database territory                              = C
Database code page                              = 819
Database code set                               = ISO8859-1
Database country code                           = 1

Directory object name              (DIR_OBJ_NAME) =

Backup pending                                  = NO
Database is consistent                          = YES
Rollforward pending                             = NO

Log retain for recovery status                  = YES
User exit for logging status                    = NO

Number of frequent values retained  (NUM_FREQVALUES) = 10
Number of quantiles retained        (NUM_QUANTILES) = 20

Database heap (4KB)                    (DBHEAP) = 512
Catalog cache size (4KB)          (CATALOGCACHE_SZ) = 64
Log buffer size (4KB)                  (LOGBUFSZ) = 128
Utilities heap size (4KB)         (UTIL_HEAP_SZ) = 5000
Buffer pool size (4KB)                 (BUFFPAGE) = 452500
Max storage for lock lists (4KB)       (LOCKLIST) = 600

Sort list heap (4KB)                   (SORTHEAP) = 16
SQL statement heap (4KB)               (STMTHEAP) = 2048
Default application heap (4KB)         (APPLHEAPSZ) = 328
Package cache size (4KB)               (PCKCACHESZ) = 40
Statistics heap size (4KB)         (STAT_HEAP_SZ) = 4384

Interval for checking deadlock (ms)    (DLCHKTIME) = 3000
Percent. of lock lists per application (MAXLOCKS) = 20
Lock timeout (sec)                  (LOCKTIMEOUT) = -1

Changed pages threshold            (CHNGPGS_THRESH) = 40
Number of asynchronous page cleaners  (NUM_IOCLEANERS) = 2
Number of I/O servers                (NUM_IOSERVERS) = 1
Index sort flag                       (INDEXSORT) = YES
```

```
Sequential detect flag                (SEQDETECT) = NO
Default prefetch size (4KB)       (DFT_PREFETCH_SZ) = 32

Default number of containers                  = 1
Default tablespace extentsize (4KB)   (DFT_EXTENT_SZ) = 32

Max number of active applications        (MAXAPPLS) = 128
Average number of active applications     (AVG_APPLS) = 1
Max DB files open per application        (MAXFILOP) = 800

Log file size (4KB)                  (LOGFILSIZ) = 65534
Number of primary log files           (LOGPRIMARY) = 6
Number of secondary log files          (LOGSECOND) = 0
Changed path to log files             (NEWLOGPATH) =
Path to log files                       = /db2/tpcc.log/
Next active log file                    =
First active log file                   =

Group commit count                    (MINCOMMIT) = 3
Percent log file reclaimed before soft chckpt (SOFTMAX) = 25
Log retain for recovery enabled         (LOGRETAIN) = ON
User exit for logging enabled           (USEREXIT) = OFF

Auto restart enabled                (AUTORESTART) = ON
Index re-creation time                (INDEXREC) = SYSTEM (RESTART)
Default number of loadrec sessions  (DFT_LOADREC_SES) = 1
Recovery history retention (days)   (REC_HIS_RETENTN) = 366
```

Tuxedo Configuration values

```
*RESOURCES
IPCKEY          40001
UID             30
GID             5432
PERM            0666
MAXACCESSERS    1400
MAXSERVERS      100
MAXSERVICES     200
MASTER          lassen
SCANUNIT        60
MODEL           SHM
LDBAL           Y
SANITYSCAN      5
DBBLWAIT        1
BBLQUERY        60
BLOCKTIME       5

*MACHINES
hornsby   LMID=lassen
        ROOTDIR="/export/home/tuxedo"
        APPDIR="/export/home/dbbench/tuxedo"
        TUXCONFIG="/export/home/dbbench/tuxedo/tuxconfig.lassen"
        ULOGPFX="/export/home/dbbench/tuxedo/ULOG"
```

```
*GROUPS
GROUP1      GRPNO=1
        LMID=lassen
        OPENINFO=NONE
        CLOSEINFO=""

*SERVERS
DEFAULT:    SRVGRP=GROUP1
DEFAULT:    REPLYQ=N
DEFAULT:    MAXGEN=2
DEFAULT:    RESTART=Y
DEFAULT:    CLOPT="-A"

newordpay_db2 SRVID=400 MIN=15 RQADDR="nop150"
stockdel_db2 SRVID=500 MIN=4 RQADDR="std500"

*SERVICES
PAYM
NEWO
ORDS
DEL
STOCK
```

Compilation Flags

These are the compilation flags used to compile the application code:

```
- xO4 -xcg92 -Xa -Kpic -lm
```

Appendix D: Disk Storage

The calculations used to determine the storage requirements for the 8 hour logical log and the 180-day space calculations are contained in this appendix.

The calculations for the 8 hour recovery log was determined as follows :

The number of logpages used during the measurement run was determined by monitoring the timestamps on the log files after the run. This showed that the shortest period before a log file switch was 14 minutes. Consequently 65,534 * 4K log pages were consumed every 14 minutes:

(65534 * 4 * 8 * 60) / (14 * 1024 * 1024) = 8.57 GB.

An equivalent amount of disk space was configured for the log mirror.

Warehouses	255
tpmC	3107.17

Table	Rows	Initial Data	Population Index	(4K pages) 5% space	Daily Growth	Total
Customer	7650000	1530613	122892	76531	0	1730036
District	2550	107	8	5	0	120
History	7650000	114237	0	0	22272	136509
Item	100000	2440	302	122	0	2864
Neworder	2295000	10297	10246	515	0	21058
Order	7650000	72894	72595	0	14211	159700
Orderline	76500000	1275502	384382	0	248671	1908555
Stock	25500000	2126063	89474	106303	0	2321840
Warehouse	255	43	1	2	0	46
Total		5132196	679900	183478	285154	6280729

Other Storage	**4K pages**
Index overhead	5,678
storage overhead	1248

Total DB Storage	6002500

5% Growth	183478	(4K pages)
Daily Growth	285154	
Free space	5696252	(Db2 does not benefit performance from free space)
Dynamic Space	1462633	
Static Space	4539867	

Daily Spread	0	(DB2 can be configured to set to zero)		
180day Space	55867671.90	In Gb =	213.12	
Log page rate:	65534	(4K pages, 1 log file) pe	14.00	mins
8hr LogSpace	2246880.00	(4k pages) In Gb =	8.57	(GB)
mirror LogSpace			8.57	(GB)
DB2 catalog etc			0.55	(GB)
OS, file system etc			0.35	(GB)
swap			3.00	(GB)
Total space			234.16	(GB)
Price system capacity			241.56	(GB)
Unused disk capacity			7.40	(GB)

Appendix E: Driver Scripts

E

The following code sections show how the transactions are generated and how statistics are gathered. Each of the transaction functions generates the input data for that transaction, sends it to the client, reads the output form and computes keying, response and think time statistics.

This is the main loop of the RTE:

```
/* run for ramp up without capturing the stats */
        i=0;
        in_ramp = 1;
        while (1)
        {
                tx_type = do_menu(); /* Select transaction */
                switch (tx_type) {
                case NEWORDER:
                        do_neworder();
                        break;
                case PAYMENT:
                        do_payment();
                        break;
                case DELIVERY:
                        do_delivery();
                        break;
                case ORDSTAT:
                        do_ordstat();
                        break;
                case STOCKLEVEL:
                        do_stocklevel();
                        break;
                default:
                        fprintf(stderr, "%s: Slave %d: Internal error. Tx-type = %d \n",
                        hostname, slave_num, tx_type);
                        cleanup(-1);
                }
                end_time = gettime();
```

```
                if ( end_time >= control->end_rampup &&
                                end_time < control->end_stdystate )
                                in_ramp = 0;
                else
                        in_ramp = 1;
                if (end_time >= control->end_rampdown)
                        break;
        }
```

The do_menu function selects the transaction to execute based on the weighted distribution algorithm.

```
int
do_menu()
{
        int val, result, menu_start, menu_end, menu_resp;
        char ch;
        /* Read menu line from client */
        /* Choose tx. type*/
        /* Now select menu and compute menu response time */
        menu_start = gettime();
        /* Write menu selection to client */
        /* Read input form for this transaction type */
        menu_end = gettime();
        menu_resp = menu_end - menu_start;
        if ( ! in_ramp) {
                statsp->menu_resp += menu_resp;
                /* Post in histogram bucket */
                if ((menu_resp / MENU_BUCKET) < MENU_MAX)
                        statsp->menu_hist[menu_resp / MENU_BUCKET]++;
                else
                        statsp->menu_hist[MENU_MAX - 1]++;
                if (menu_resp > statsp->menu_max)
                        statsp->menu_max = menu_resp;
        }
        return(result);
}
/*
 * Function: do_neworder
 * This function executes the neworder transaction
 * It generates all the input fields, sends it to the
 * client over the keying time, measures the response
 * time, reads the results and delays for the think time.
 */
 /* The code for the other transactions is similar */
do_neworder()
{
        struct newo_fld no;
        struct items_fld *itemp = no.items;
        int ol_cnt, rbk, remote = 0, i, x;
        char *bufp = fldbuf;
        int start_time, end_time, key_time, resp_time, elapse_time, del;
        start_time = gettime();
        /* Now wait for keying time */
        poll (0, 0, NEWO_KEY);
        /* Generate all input data */
        no.d_id = random(1, 10);
        no.c_id = NURand(1023, 1, 3000, CONST_CID);
```

```
ol_cnt = random(5, 15);
rbk = random(1, 100);/* trans. to be rolledback */
sprintf(bufp, "%02d%04d", no.d_id, no.c_id);
bufp += strlen(bufp);
/* Generate all the item fields */
for (i=0; i < ol_cnt; i++, itemp++) {
        itemp->ol_i_id = NURand(8191, 1, 100000, CONST_IID);
        /* If last item and rbk, select unused item */
        if (i == ol_cnt - 1 && rbk == 1) {
                itemp->ol_i_id = 100001;
        }
        x = random(1, 100);
        if ( x > 1)
                itemp->ol_supply_w_id = W_ID;
        else {
                /* Select a warehouse other than w_id */
                do {
                        x = random(1, control->scale);
                } while (x == W_ID);
                itemp->ol_supply_w_id = x;
                remote++;
        }
        itemp->ol_quantity = random(1, 10);
        sprintf(bufp, "%04d%06d%02d",itemp->ol_supply_w_id,
        itemp->ol_i_id, itemp->ol_quantity);
        bufp += strlen(bufp);
}
strcpy(bufp, leave_key);
bufp += 2;
/* Compute keying time info */
end_time = gettime();
key_time = end_time - start_time;
start_time = end_time;

/* Now send fields to client */
/* Read output screen from client */
end_time = gettime();
/* Store elapse time info for thruput */
elapse_time = end_time - control->start_time;
/* compute the how long it took to run the tx */
resp_time = end_time - start_time + control->newo_delta;
/* Wait think time */
del = delay(control->newo_think, 5*control->newo_think);
poll(0, 0, del + control->newo_delta);
end_time = gettime();
/* Now post all stats */
if ( ! in_ramp && end_time <= control->end_stdystate) {
        statsp->newo_cnt++; /* another one bytes the dust */
        if ( rbk == 1 )
                statsp->newo_rbkcnt++;
        statsp->newo_remote += remote;
        statsp->newo_olcnt += ol_cnt;
        statsp->newo_key += key_time;
        /* Save keying time in histogram bucket */
        statsp->newo_resp += (double) resp_time;/* sum up the response time */
        /* Save response time in histogram bucket */
```

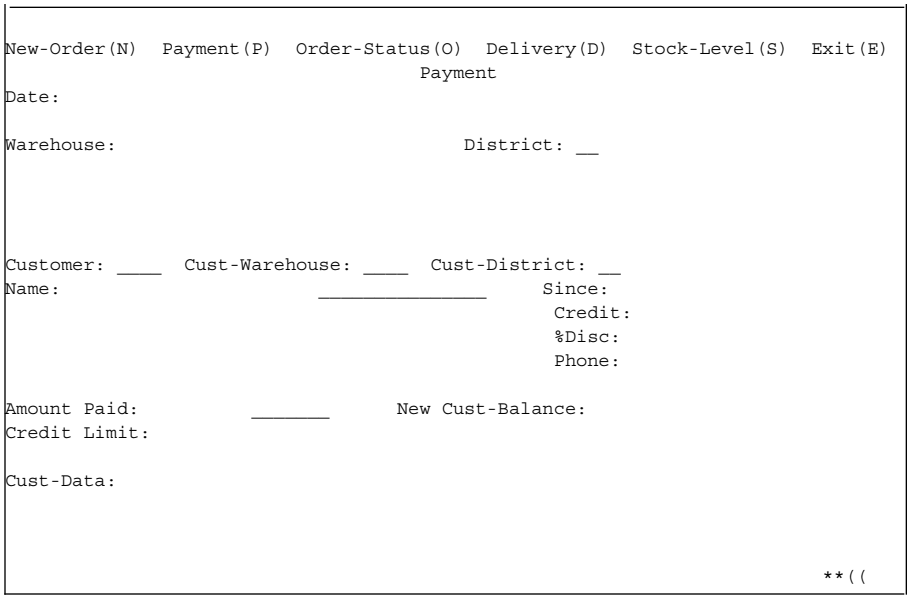
```
                statsp->newo_think += (double) del;
                /* Save think time in histogram bucket */
        }
}
```

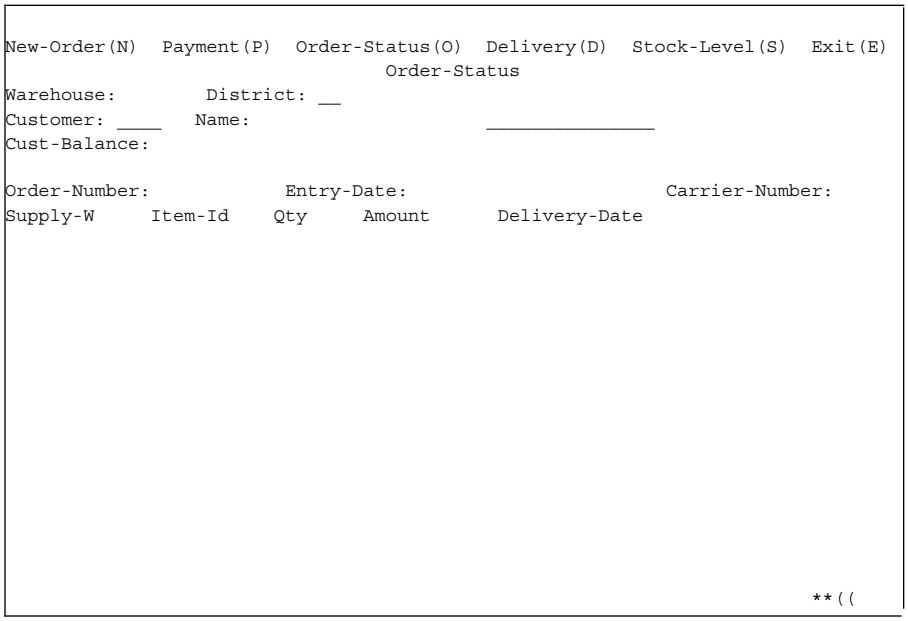
Appendix F: Screen Layouts

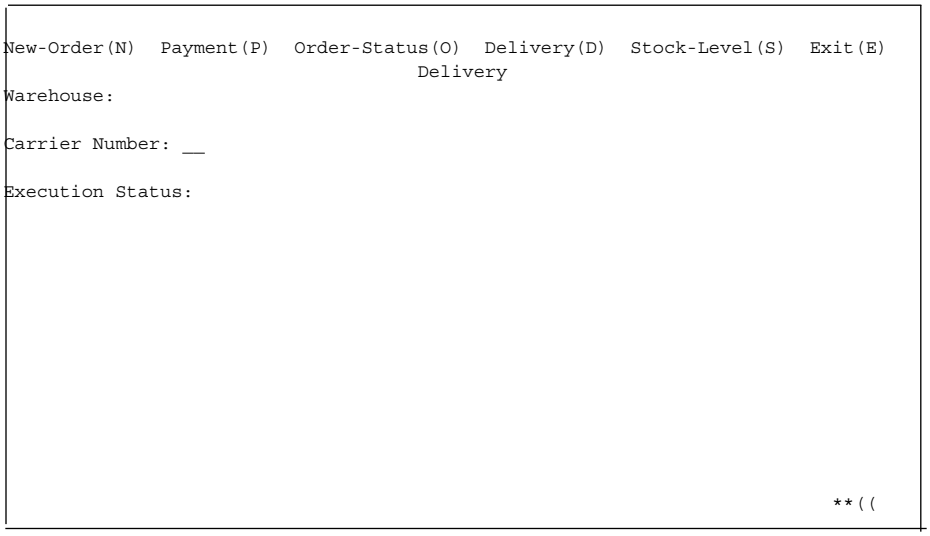
This Appendix contains the screen form layouts for the 5 transactions.

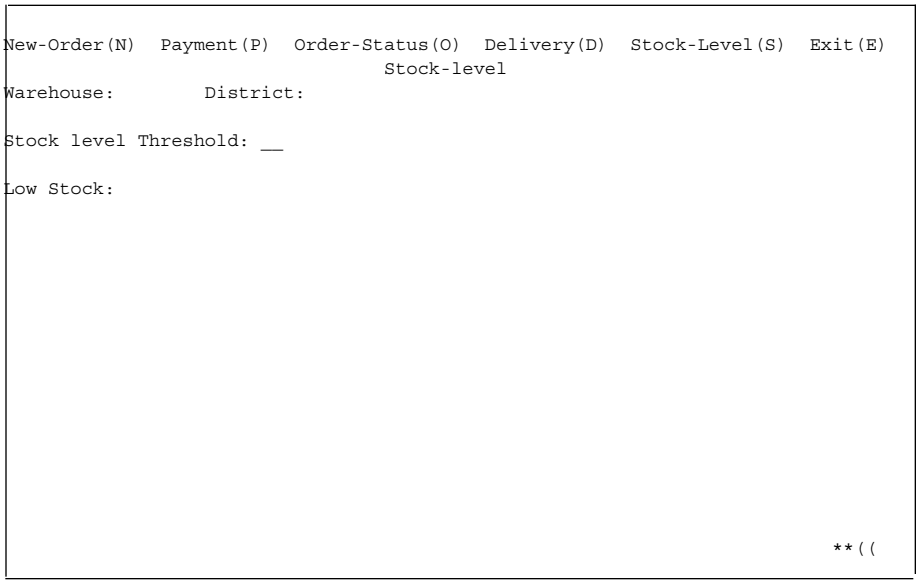
```
New-Order(N)  Payment(P)  Order-Status(O)  Delivery(D)  Stock-Level(S)  Exit(E)
                                  New Order
Warehouse:        District: __                        Date:
Customer: ____   Name:                    Credit:      %Disc:
Order Number:           Number of Lines:           W_tax:         D_tax:

 Supp_W  Item_Id  Item Name                 Qty  Stock  B/G  Price    Amount
  ____   ______                            __
  ____   ______                            __
  ____   ______                            __
  ____   ______                            __
  ____   ______                            __
  ____   ______                            __
  ____   ______                            __
  ____   ______                            __
  ____   ______                            __
  ____   ______                            __
  ____   ______                            __
  ____   ______                            __
  ____   ______                            __
  ____   ______                            __
Execution Status:                                          Total:
                                                                      **((
```


New-Order(N) Payment(P) Order-Status(O) Delivery(D) Stock-Level(S) Exit(E)
Payment
Date:
Warehouse: District: __
Customer: ____ Cust-Warehouse: ____ Cust-District: __
Name: _____ Since:
Credit:
%Disc:
Phone:
Amount Paid: _____ New Cust-Balance:
Credit Limit:
Cust-Data:
**((


New-Order(N) Payment(P) Order-Status(O) Delivery(D) Stock-Level(S) Exit(E)
Order-Status
Warehouse: District: __
Customer: ____ Name: _____
Cust-Balance:
Order-Number: Entry-Date: Carrier-Number:
Supply-W Item-Id Qty Amount Delivery-Date
**((


New-Order(N) Payment(P) Order-Status(O) Delivery(D) Stock-Level(S) Exit(E)
Delivery
Warehouse:
Carrier Number: __
Execution Status:
**((


New-Order(N) Payment(P) Order-Status(O) Delivery(D) Stock-Level(S) Exit(E)
Stock-level
Warehouse: District:
Stock level Threshold: __
Low Stock:
**((

Appendix G: Price Quotes

The following pages contain the pricing quotes for the hardware and software included in this FDR.

11-14-1996 12:01AM FROM CAT TECHNOLOGY 408 345 9191 P. 01

CATechnology Sales Quotation

DATE 11/14/96
QUOTA 961114A

TO: John J Boughenamn
Manager
Database Engineering
Sun Microsystems Computer Corporation
2550 Garcia Avenue, MS MPK12-305
Mountain View, CA 94043-1100

From: Dick Crouch
CATechnology, Inc.
3031 Tisch Way
San Jose, CA 95128
408-345-9199
FAX 345-9191

CREDIT TERMS NET 30 15%/NO PAST DUE | FOB WAREHOUSE | PAGE 1

ITEM	PART NO	DESCRIPTION	NET PRICE	QTY	EXT PRICE
		Ultra Enterprise 2 Model 2200			
1	A14-UC02-08-256EB	Ultra Enterprise 2 Model 2200	$24,371	1	$24,371
2	X7006A	256MB SIMM Expansion	$5,326	8	$42,608
3	7F2A	9GA Model 112 w/ 30 2.1 GB Drives	$35,100	4	$140,400
4	1957A	Fibre Channel SBus Host Adapter	$1,200	3	$3,600
5	9F6A	Fibre Channel Optical Module	$450	1	$450
6	X6758A	4-8GB 4mm DDS2 Internal Tape Drive	$1,125	1	$1,125
7	A11-UBA1-08-0640B	UltraServer 1 Model 170	$9,748	2	$19,496
8	X1018A	SBus Quad Ethernet Controller	$748	2	$1,492
9	X7003A	2 x 64MB Memory for UltraServer	$1,385	8	$11,080
		TOTAL			$244,897

Note: Above configurations are generally available and prices are guaranteed for 60 days