TPC Benchmark™ C

Full Disclosure Report

IBM RISC System/6000

PowerPC Server R30 c/s

PowerPC Server J30 c/s

using

IBM DB2 for AIX Version 2.1

IBM System Performance and Evaluation Center

Submitted For Review

June 19, 1995

Special Notices

The following terms used in this publication are trademarks of **International Business Machines** Corporation in the United States and/or other countries:

RISC System/6000
AIX
IBM
DB2
DATABASE 2

The following terms used in this publication are trademarks of other companies as follows:

TPC Benchmark	Trademark of the Transaction Processing Performance Council
Emulex	Trademark of Emulex Corporation
Tuxedo-System/T	Trademark of UNIX System Laboratories, Inc. Distributed by Information Management Company (IMC)

The phrase RS/6000 is an abbreviation for RISC System/6000.
The phrase DB2 for AIX is an abbreviation for IBM DATABASE 2 for AIX.

Second Edition June 19, 1995

The information contained in this document has not been submitted to any formal test and is distributed on an AS IS basis without any warranty either expressed or implied. The use of this information or the implementation of any of these techniques is a customer's responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item has been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environment do so at their own risk.

In this document, any references made to an IBM licensed program are not intended to state or imply that only IBM's licensed program may be used; any functionally equivalent program may be used.

It is possible that this material may contain references to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such products, programming, or services in your country.

All performance data contained in this publication was obtained in a controlled environment, and therefore the results which may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable date in their specific environment.

Request for additional copies of this document should be sent to the following address:

TPC Benchmark Administrator
IBM System and Performance Evaluation Center
Mail Stop 9440
11400 Burnet Road
Austin, TX 78758
FAX Number (512) 838-1852

© Copyright International Business Machines Corporation 1995. All rights reserved.

Permission is hereby granted to reproduce this document in whole or in part, provided the copyright notice printed above is set forth in full text on the title page of each item reproduced.

NOTE: US. Government Users - Documentation related to restricted rights: Use, duplication, or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Abstract

This report documents the full disclosure information required by the TPC Benchmark™ C Standard Specification Revision 3.0 dated February 15, 1995, for measurements on the IBM RISC System/6000 PowerPC Server R30 & J30.

The software used on the RS/6000-R30 & J30 systems includes AIX Version 4.1.2 operating system, DB2 for AIX data base manager, and Tuxedo System/T Version 4.2.1 transaction manager.

IBM RISC System/6000 PowerPC Server R30

Company Name	**System Name**	**Data Base Software**	**Operating System Software**
	RS/6000 PowerPC Server R30	DB2 for AIXVersion 2.1	AIX Version 4.1.2
Availability Date: DB2 for AIX available December 15, 1995 CPU card 3 & 4 available August 25, 1995			

Total System Cost	**TPC-C Throughput**	**Price/Performance**
-Hardware -Software -5 Years Maintenance	Sustained maximum throughput of system running TPC-C expressed in transactions per minute	Total system cost/tpmC ($1,106,716/3,119.16)
$1,106,716	3,119.16 **tpmC**	$355/t**pmC**

IBM RISC System/6000 PowerPC Server J30

Company Name	**System Name**	**Data Base Software**	**Operating System Software**
	RS/6000 PowerPC Server J30	DB2 for AIXVersion 2.1	AIX Version 4.1.2
Availability Date: DB2 for AIX available December 15, 1995 CPU card 3 & 4 available August 25, 1995			

Total System Cost	**TPC-C Throughput**	**Price/Performance**
-Hardware -Software -5 Years Maintenance	Sustained maximum throughput of system running TPC-C expressed in transactions per minute	Total system cost/tpmC ($1,089,830.51/3,119.16)
$1,089,831	3,119.16 **tpmC**	$349/t**pmC**

Numerical Quantities Summary for the IBM RS/6000 PowerPC Server R30 & J30

MQTH, computed Maximum Qualified Throughput: 3119.16 tpmC

Response Times (in seconds)	**90th Percentile**	**Average**
New Order	3.12	1.96
Payment	2.6	1.47
Order-Status	2.53	1.41
Delivery (interactive)	0.46	0.42
Delivery (deferred)	3	1.76
Stock-Level	5.11	3.17
Menu	0.7	0.59

Transaction Mix, in percent of total transactions	**Percent**
New Order	44.41%
Payment	43.27%
Order-Status	4.09%
Delivery	4.11%
Stock-Level	4.10%

Keying/Think Times (in seconds)	**Min.**	**Average**	**Max.**
New Order	18/0.01	18.06/12.25	18.36/113.74
Payment	3/0.01	3.02/12.27	3.12/122
Order-Status	2/0.01	2.01/10.35	2.06/97.21
Delivery	2/0.01	2.0/5.16	2.09/40.16
Stock-Level	2/0.01	2.0/5.16	2.08/52

Test Duration	
Ramp-up Time	15:00 minutes
Measurement interval	25 minutes
Transactions during measurement interval (all types)	175,586
Ramp-down time	15:00minutes

Preface

TPC Benchmark™ C Standard Specification was developed by the Transaction Processing Performance Council (TPC). It was released on August 13, 1992 and updated with revision 3.0 on February 15, 1995.

This is the full disclosure report for benchmark testing of the IBM RS/6000 systems according to the TPC Benchmark™ C Standard Specification. Measurements were done on the IBM RS/6000 PowerPC Server R30 & J30 system.

TPC Benchmark™ C exercises the system components necessary to perform tasks associated with that class of on-line transaction processing (OLTP) environments emphasizing a mixture of read-only and update intensive transactions. This is a complex OLTP application environment exercising a breadth of system components associated by such environments characterized by:

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

This benchmark defines four on-line transactions and one deferred transaction, intended to emulate functions that are common to many OLTP applications. However, this benchmark does not reflect the entire range of OLTP requirements. The extent to which a customer can achieve the results reported by a vendor is highly dependent on how closely TPC-C approximates the customer application. The relative performance of systems derived from this benchmark does not necessarily hold for other workloads or environments. Extrapolations to any other environment are not recommended.

Benchmark results are highly dependent upon workload, specific application requirements, and systems design and implementation. Relative system performance will vary as a result of these and other factors. Therefore, TPC-C should not be used as a substitute for a specific customer application benchmarking when critical capacity planning and/or product evaluation decisions are contemplated.

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

1. General Items

1.1 Application Code Disclosure

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 RS/6000 application code for the five TPC Benchmark™ C transactions. Appendix D contains the terminal functions and layouts.

1.2 Benchmark Sponsor

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

This benchmark was sponsored by **International Business Machines** 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:

- *Data Base tuning options*
- *Recovery/commit options*
- *Consistency/locking options*
- *Operating system and application configuration parameters.*

Appendix B contains the system, data base, and application parameters changed from their default values used in these TPC Benchmark™ C tests.

1.4 Configuration Diagrams

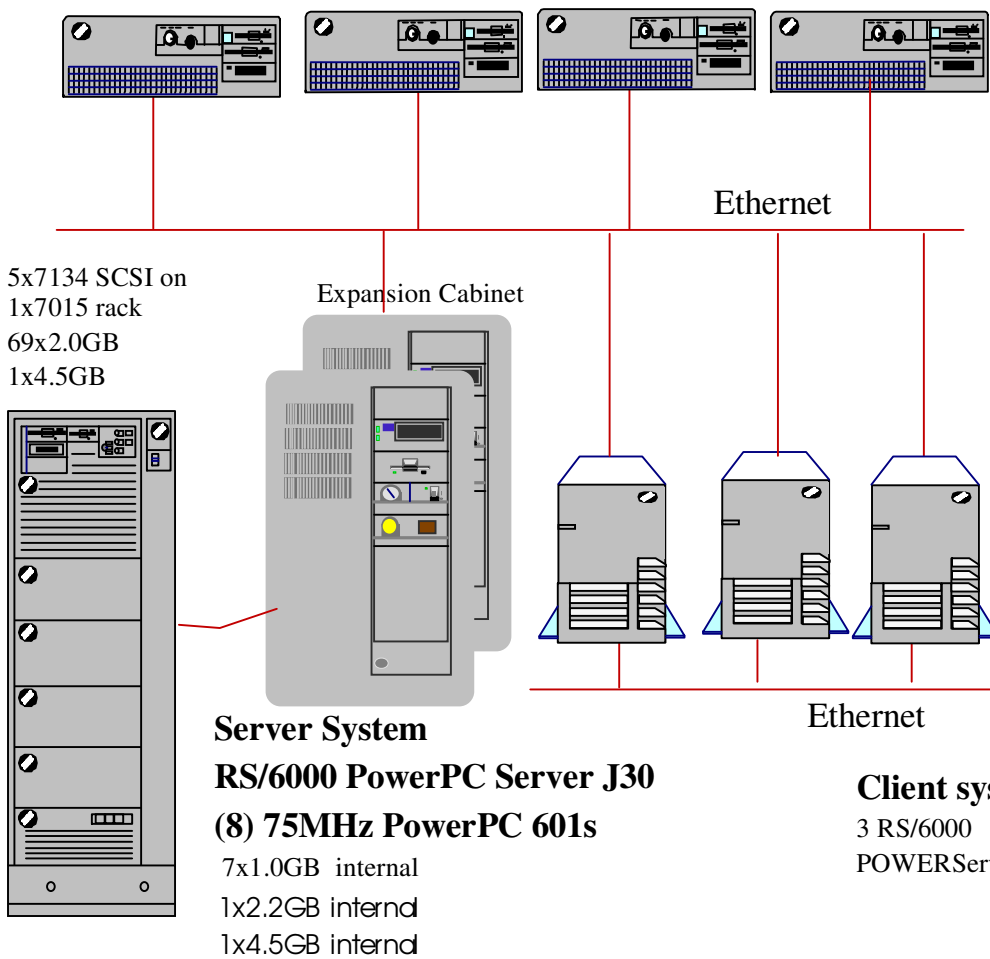
Diagrams of both measured and priced configurations must be provided, accompanied by a description of the differences. This includes, but is not limited to:

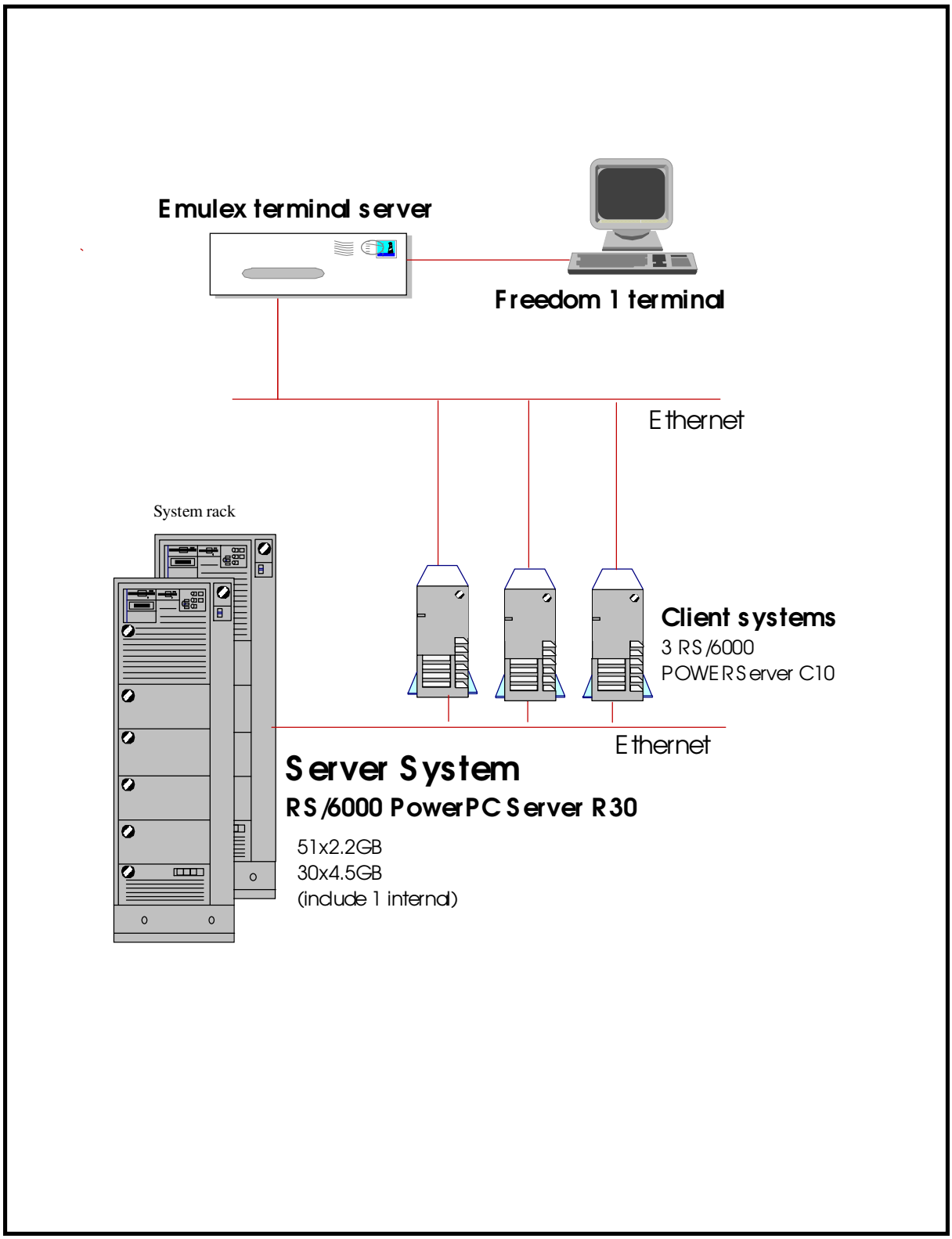
- *Number and type of processors*
- *Size of allocated memory, and any specific mapping/partitioning of memory unique to the test Number and type of disk units (and controllers, if applicable)*
- *Number of channels or bus connections to disk units, including the protocol type*
- *Number of LAN (e.g. Ethernet) connections, including routers, work stations, terminals, etc., that were physically used in the test or are incorporated into the pricing structure (see Clause 8.1.8)*
- *Type and run-time execution location of software components (e.g. DBMS, client processes, transaction monitors, software drivers, etc.)*

The Remote Terminal Emulator (RTE) used by IBM for these TPC Benchmark™ C tests is an internally developed IBM tool. The components of the configuration being emulated by the RTE are the terminals , and the Emulex Terminal Servers and Modules.

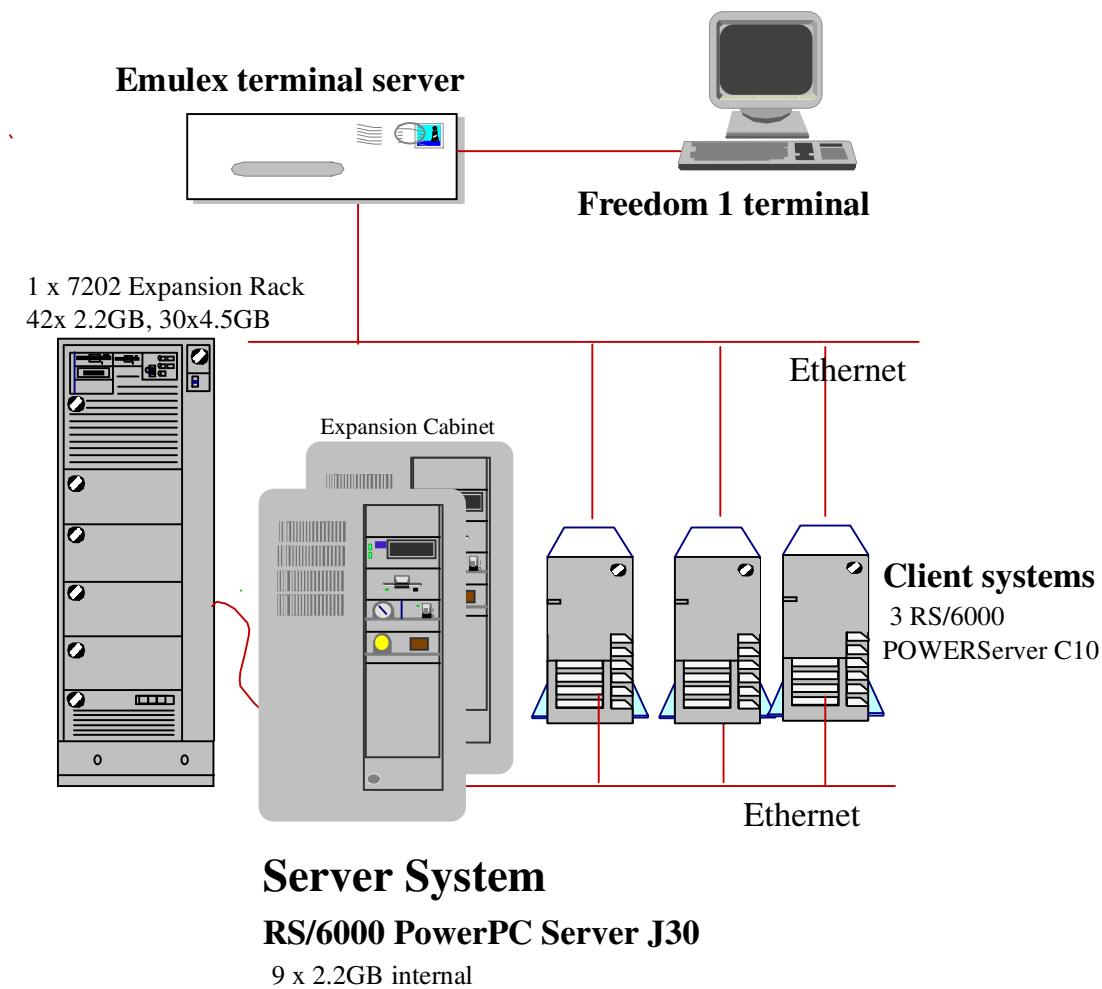
Appendix E contains a listing of the RTE scripts and inputs used in the benchmark testing.

The RS/6000 POWERServer C10 (clients) executed the terminal I/O and submitted transactions to the Tuxedo System/T transaction manager also running on the clients. DB2 for AIX is the DBMS executing on the servers.


RTE Drivers
4 RS/6000 POWERServer 380
Ethernet
5x7134 SCSI on
1x7015 rack
69x2.0GB
1x4.5GB
Expansion Cabinet
Server System
RS/6000 PowerPC Server J30
(8) 75MHz PowerPC 601s
7x1.0GB internal
1x2.2GB internal
1x4.5GB internal
Ethernet
Client systems
3 RS/6000
POWERServer C10


Emulex terminal server
Freedom 1 terminal
Ethernet
System rack
Client systems
3 RS/6000
POWERServer C10
Ethernet
Server System
RS/6000 PowerPC Server R30
51x2.2GB
30x4.5GB
(include 1 internal)

RISC System/6000 PowerPC Server J30 Priced Configuration



2. Clause 1: Logical Data Base Design Related Items

2.1 Table Definitions

Listings must be provided for all table definition statements and all other statements used to setup the data base.

Appendix C contains the table creation, index creation, and the data generation programs to load the data base.

2.2 Database Organization

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

Physical space was allocated to DB2 for AIX on the server disks according to the details provided in figure 5-1. The size of the space segments on each disk was calculated to provide even distribution of data across the disk subsystem. The indices were defined and built after the initial table load by executing the index creation script in section C.2 of Appendix C.

2.3 Insert and/or 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 restriction in the SUT data base 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/or delete operations to any of the tables. The space required for an additional five percent of the initial table cardinality was allocated to DB2 for AIX and priced as static space.

2.4 Horizontal or Vertical Partitioning

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

Partitioning was not used for any of the measurements reported in this full disclosure.

3. Clause 2: Transaction and Terminal Profiles Related Items

3.1 Verification for the Random Number Generator

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

The srandom(), getpid() and gettimeofday() functions are used to produce unique random seeds for each driver. The drivers use these seeds to seed the srand(), srandom() and srand48() functions. Random numbers are produced using wrappers around the standard system random number generators.

The negative exponential distribution uses the following function to generate the distribution. This function has the property of producing a negative exponential curve with a specified average and a maximum value 4 times the average.

```
const double RANDOM_4_Z = 0.89837799236185
const double RANDOM_4_K = 0.97249842407114

double neg_exp_4(double average {
                    return - average * (1/RANDOM_4_Z * log (1 - RANDOM_4_K * drand48()));
                    }
```

The random functions used by the driver system and the data base generation program were verified. The C_LAST column was queried to verify the random values produced by the database generation program. After a measurement, the HISTORY, ORDER, and ORDER_LINE tables were queried to verify the randomness of values generated by the driver. The rows were counted and grouped by customer and item numbers.

Here is an example of one SQL query used to verify the random number generation functions:

- create table TEMP (W_ID smallint, D_ID smallint, C_LAST char(16), CNTR smallint);
- insert into TEMP select C_W_ID, C_D_ID, C_LAST, COUNT(*) from CUSTOMER group by C_W_ID, C_D_ID, C_LAST;
- select CNTR, COUNT(*) from TEMP group by CNTR order by 1;

3.2 Input/Output Screens

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

Appendix D contains the screen layouts for all transactions corresponding to the specification in clauses 2.4.3, 2.5.3, 2.6.3, 2.7.3 and 2.8.3.

3.3 Priced Terminal Features

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

The Liberty Freedom 1 terminals and Emulex Performance 6000 88-port (P6000-88) terminal servers being emulated by the RTE support all of the requirements in clause 2.2.2.4. A Liberty Freedom 1 terminal, physically attached to an Emulex P6000-88 terminal server, was connected to the client system via a telnet session. All transactions were executed from the terminal with no exceptions occurring during the test.

3.4 Presentation Managers

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

The terminal emulated in the priced configuration is Liberty Freedom 1 terminal. Liberty Freedom 1 terminal does not use a presentation manager nor is it an intelligent display. All of the processing of the input/output screens is handled by the client. The terminal is attached to the RS/6000 via a telnet session from an Emulex P6000-88 terminal server.

3.5 Home and Remote Order-lines

The percentage of home and remote order-lines in the New-Order transactions must be disclosed.

Table 3-1 shows the percentage of home and remote transactions that occurred during the measurement period for the New-Order transactions.

3.6 New-Order Rollback Transactions

The percentage of New-Order transactions that were rolled back as a result of an illegal item number must be disclosed.

Table 3-1 shows the percentage of New-Order transactions that were rolled back due to an illegal item being entered.

3.7 Number of Items per Order

The number of items per order entered by New-Order transactions must be disclosed.

Table 3-1 shows the average number of items ordered per New-Order transaction.

3.8 Home and Remote Payment Transactions

The percentage of home and remote Payment transactions must be disclosed.

Table 3-1 shows the percentage of home and remote transactions that occurred during the measurement period for the Payment transactions.

3.9 Non-Primary Key Transactions

The percentage of Payment and Order-Status transactions that used non-primary key (C_LAST) access to the data base must be disclosed.

Table 3-1 shows the percentage of non-primary key accesses to the data base by the Payment and Order-Status transactions.

3.10 Skipped Delivery Transactions

The percentage of Delivery transactions that were skipped as a result of an insufficient number of rows in the NEW-ORDER table must be disclosed.

Table 3-1 shows the percentage of Delivery transactions missed due to a shortage of supply of rows in the NEW-ORDER table.

3.11 Mix of Transaction Types

The mix (i.e. percentages) of transaction types seen by the SUT must be disclosed.

Table 3-1 shows the mix percentage for each of the transaction types executed by the SUT.

3.12 Queuing Mechanism of Delivery

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

The Delivery transaction was submitted using an asynchronous call to a Tuxedo System/T service. Tuxedo System/T returns an immediate response to the calling program and schedules the work to be performed. This allows the Delivery transaction to be submitted, obtain an interactive response and queue the actual data base transaction for deferred execution. Please see the application code in Appendix A for details.

Table 3-1 Numerical Quantities for Transaction and Terminal Profiles (RS/6000 R30 & J30)

New Order	RS/6000
Percentage of Home order lines	99.00%
Percentage of Remote order lines	1.00%
Percentage of Rolled Back Transactions	1.00%
Number of Items per order	10.02
Payment	
Percentage of Home transactions	84.94%
Percentage of Remote transactions	15.06%
Non-Primary Key Access	
Percentage of Payment using C_LAST	59.95%
Percentage of Order-Status using C_LAST	59.60%
Delivery	
Delivery transactions skipped	0
Transaction Mix	
New-Order	44.41%
Payment	43.27%
Order-Status	4.09%
Stock-Level	4.11%
Delivery	4.10%

4. Clause 3: Transaction and System Properties

The results of the ACID test must be disclosed along with a description of how the ACID requirements were met.

All ACID tests were conducted according to specification. The Atomicity, Consistency, Isolation and all Durability tests were performed on the RS/6000 R30 & J30.

4.1 Atomicity Requirements

The system under test must guarantee that data base 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.

4.1.1 Atomicity of Completed Transaction

Perform the Payment transaction for a randomly selected warehouse, district, and customer (by customer number) and verify that the records in the CUSTOMER, DISTRICT, and WAREHOUSE tables have been changed appropriately.

The following steps were performed to verify the Atomicity of completed transactions.

1. The balance was retrieved from the CUSTOMER table for a random Customer, District and Warehouse giving BALANCE_1.
2. The Payment transaction was executed for the Customer, District and Warehouse used in step 1
3. The balance was retrieved again for the Customer used in step 1 and step 2 giving BALANCE_2. It was verified that BALANCE_1 was greater than BALANCE_2 by AMT.

4.1.2 Atomicity of Aborted Transactions

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

The following steps were performed to verify the Atomicity of the aborted Payment transaction:

1. The Payment application code was changed to execute a rollback of the transaction instead of performing the commit.
2. Using the balance, BALANCE_2, from the CUSTOMER table retrieved for the completed transaction, the Payment transaction was executed for the Customer, District, and Warehouse used in step 1 of the section 4.1.1, using a payment amount (AMT) of 410.00. The transaction rolled back due to the change in the application code from step 1.
3. The balance was retrieved again for the Customer used for step 2 giving BALANCE_3. It was verified that BALANCE_2 was equal to BALANCE_3.

4.2 Consistency Requirements

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

4.2.1 Consistency Condition 1

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

- *W_YTD = sum(D_YTD)*

for each warehouse defined by (W_ID = D_W_ID)

The following SQL query was executed before and after a measurement to show that the data base was always in a consistent state both initially and after a measurement.

```
Select D_W_ID, W_YTD, SUM(D_YTD)
        from DISTRICT, WAREHOUSE
        where D_W_ID = W_ID
        group by D_W_ID, W_YTD
        order by D_W_ID;
```

4.2.2 Consistency Condition 2

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

- *D_NEXT_O_ID - 1 = max(O_ID) = max(NO_O_ID)*

for each district defined by (D_W_ID = O_W_ID = NO_W_ID) and (D_ID = O_D_ID = NO_D_ID). This condition does not apply to the NEW-ORDER table for any districts which have no outstanding new orders.

The following SQL queries were executed before and after a measurement to show that the data base was always in a consistent state both initially and after a measurement.

```
Select D_W_ID, D_ID, D_NEXT_O_ID -1, MAX(O_ID)
        from DISTRICT, ORDERS
        where D_ID = O_D_ID and D_W_ID = O_W_ID
        group by D_W_ID, D_ID, D_NEXT_O_ID
        order by D_W_ID, D_ID;

Select D_W_ID, D_ID, D_NEXT_O_ID -1, MAX(NO_O_ID)
        from DISTRICT, NEW_ORDER
        where D_ID = NO_D_ID and D_W_ID = NO_W_ID
        group by D_W_ID, D_ID, D_NEXT_O_ID
        order by D_W_ID, D_ID;
```

4.2.3 Consistency Condition 3

Entries in the New-Order table must satisfy the relationship:

- *max(NO_O_ID) - min(NO_O_ID) + 1 = [number of rows in the New-Order table for this district]*

for each district defined by NO_W_ID and NO_D_ID. This condition does not apply to any districts which have no outstanding new orders.

The following SQL query was executed before and after a measurement to show that the data base was always in a consistent state both initially and after a measurement.

```
Select COUNT(*), MAX(NO_O_ID) - MIN(NO_O_ID) +1
        from NEW_ORDER, DISTRICT
        where NO_W_ID = D_W_ID and NO_D_ID = D_ID
        group by NO_W_ID, NO_D_ID;
```

4.2.4 Consistency Condition 4

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

- *sum(O_OL_CNT) = [number of rows in the ORDER-LINE table for this district]*

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

The following SQL queries was executed before and after a measurement to show that the data base was always in a consistent state both initially and after a measurement.

- Select O_W_ID, O_D_ID, SUM(O_OL_CNT) from ORDERS group by O_W_ID and O_D_ID.
- Select OL_W_ID, OL_D_ID, COUNT(*) from ORDER_LINE group by OL_W_ID and OL_D_ID.

4.2.5 Consistency Tests

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

The queries defined in 4.2.1 through 4.2.4 were run after initial data base build and prior to executing any transactions. All queries showed that the data base was in a consistent state.

After executing transactions at full load for approximately sixty minutes the queries defined in 4.2.1 through 4.2.4 were run again. All queries showed that the data base was still in a consistent state.

4.3 Isolation Requirements

Operations of concurrent data base transactions must yield results which are indistinguishable from the results which would be obtained by forcing each transaction to be serially executed to completion in some order.

4.3.1 Isolation Test 1

This test demonstrates isolation for read-write conflicts of Order-Status and New-Order transactions.

The following steps were performed to satisfy the test of isolation for Order-Status and New-Order transactions:

1. 1st terminal: Start a New-Order transaction with the necessary inputs. The transaction is delayed for 60 seconds just prior to the Commit.
2. 2nd terminal: Start an Order-Status transaction for the same customer as used in the New-Order transaction.
3. 2nd terminal: The Order-Status transaction attempts to read the file but is locked out by the New-Order transaction waiting to complete.
4. 1st terminal: The New-Order transaction is released and the Commit is executed releasing the record. With the CUSTOMER record now released, the Order-Status transaction can now complete.
5. 2nd terminal: Verify that the Order-Status transaction delays for approximately 60 seconds and that the results displayed for the Order-Status transaction match the input for the New-Order transaction.

4.3.2 Isolation Test 2

This test demonstrates isolation for read-write conflicts of Order-Status and New-Order transactions when the New-Order transaction is rolled back.

The following steps were performed to satisfy the test of isolation for Order-Status and a rolled back New-Order transactions:

1. 1st terminal: Perform an Order-Status transaction for a random customer. The transaction is competed.
2. 2nd terminal: Perform a New-Order transaction for the same customer in the Order-Status transaction in Step 1, include an invalid item number in the last order. The transaction is delayed for 60 seconds just prior to the Rollback.

3. 2nd terminal: Start another Order-Status transaction for the same customer used in the New-Order transaction. The Order-Status transaction attempts to read the CUSTOMER table but is locked by the New-Order transaction.
4. 2nd terminal: Roll back the New-Order transaction. With the CUSTOMER record now released, the Order-Status transaction completes.
5. Verify the results returned from the 1st and 2nd Order-Status transactions match.

4.3.3 Isolation Test 3

This test demonstrates isolation for write-write conflicts of two New-Order transactions.

The following steps were performed to verify isolation of two New-Order transactions:

1. 1st terminal: Start a New-Order transaction using the necessary inputs. The transaction is delayed for 60 seconds just prior to the Commit.
2. 2nd terminal: Start a second New-Order transaction for the same customer used by the 1st terminal. This transaction is forced to wait while the 1st terminal holds a lock on the CUSTOMER record requested by the 2nd terminal.
3. 1st terminal: The New-Order transaction is allowed to complete and Commit the transaction. With the CUSTOMER record released, the 2nd terminal New-Order transaction will complete.
4. Verify the order number from the 2nd terminal New-Order transaction is 1 greater than the order number from the 1st terminal. Verify D_NEXT_O_ID is 1 greater than the order number of the New-Order transaction from the 2nd terminal.

4.3.4 Isolation Test 4

This test demonstrates isolation for write-write conflicts of two New-Order transactions when one transaction is rolled back.

The following steps were performed to verify the isolation of two New-Order transactions after one is rolled back:

1. 1st terminal: Start a New-Order transaction using the necessary inputs to cause a roll back (invalid item number). The transaction is delayed for 60 seconds just prior to the Roll back.
2. 2nd terminal: Start a second New-Order transaction for the same customer used by the 1st terminal. This transaction is forced to wait while the 1st terminal holds a lock on the CUSTOMER record requested by the 2nd terminal.
3. 1st terminal: The New-Order transaction is allowed to complete and the transaction is rolled back due to the invalid item number.
4. 2nd terminal: With the CUSTOMER record released, the 2nd terminal New-Order transaction will complete normally.
5. Verify the order number from the 2nd terminal New-Order transaction is equal to the next order number before either New-Order transaction was started. Verify D_NEXT_O_ID is 1 greater than D_NEXT_O_ID before either transaction was started and is 1 greater than the order number for the 2nd New-Order transaction.

4.3.5 Isolation Test 5

This test demonstrates isolation for write-write conflicts of Payment and Delivery transactions.

The following steps were performed to successfully conduct this test:

1. 1st terminal: A Delivery transaction is started. The Delivery transaction batch job is started with code added to the transaction to delay the transaction just prior to the Commit. (NOTE: This delay is only used during the Isolation tests).
2. 2nd terminal: Start a Payment transaction for a customer that will have an order delivered in the transaction started in step 1. The Payment transaction is forced to wait while the Delivery transaction holds a lock on the CUSTOMER record.

3. 1st terminal: After the delay time period is expired for the Delivery transaction, the transaction is Committed.
4. 2nd terminal: With the CUSTOMER record released, the Payment transaction is now able to complete.
5. The record is viewed from the CUSTOMER table for the customer and C_BALANCE is verified to reflecting the changes from the Delivery and Payment transactions.

4.3.6 Isolation Test 6

This test demonstrates isolation for write-write conflicts of Payment and Delivery transactions when the Delivery transaction is rolled back.

The following steps were performed to successfully conduct this test:

1. 1st terminal: Start a Delivery transaction using the Delivery batch job with the delay coded in to halt the transaction just prior to the Roll back.
2. 2nd terminal: Start a Payment transaction for a customer that will have an order delivered in the transaction started in step 1. The Payment transaction is forced to wait while the Delivery transaction holds a lock on the CUSTOMER record.
3. 1st terminal: The Delivery transaction is rolled back after the time period expires.
4. 2nd terminal: With the CUSTOMER record released, the Payment transaction is now able to complete.
5. The record is viewed from the CUSTOMER table for the customer and C_BALANCE is verified to reflecting the change only from the Payment transaction.

4.3.7 Isolation Test 7

This test demonstrates repeatable reads for the New-Order transaction while an interactive transaction updates the price of an item.

1. 1st terminal: Using DB2 command line processor, view and record prices for two items (x,y) selected at random.
2. 2nd terminal: A New-Order transaction is started that contains item x twice and y once. This transaction is stopped after reading the price of item x from the item file the first time.
3. 1st terminal: Using DB2 command line processor, an update transaction is started for items x and y, increasing their price by 10%. Case A, transaction 3 stalls.
4. 2nd terminal: The New-Order transaction is allowed to complete. It is verified that the prices for items x and y are the same throughout the entire transaction and that they match the results of step 1.
5. 1st terminal: After the New-Order transaction completes, the update transaction completes and is committed.
6. 1st terminal: Step 1 is repeated, noting that the prices of items x and y now match those set in step 3.

4.4 Durability Requirements

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

4.4.1 Permanent Unrecoverable Failure of any Single Durable Medium

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

Failure of Durable Medium containing recovery log data and Instantaneous Interruption and Memory Failure.

This test was conducted on a fully scaled database. The following steps were performed successfully.

1. The current count of the total number of orders was determined by the sum of D_NEXT_O_ID of all rows in the DISTRICT table giving SUM_1.
2. A test was started and allowed to run for twelve minutes.
3. One of the disks containing the DB2 for AIX transaction log data was powered off. Since this disk was mirrored, DB2 for AIX continued to process the transactions successfully.
4. The test continued for another 1 1/2 minutes.
5. The system was immediately shut down by switching the Emergency Power Off , thereby removing system power.
6. The disk from step 3 was powered back on.
7. The system was powered back on and rebooted.
8. Step 1 is performed returning the value for SUM_2. It was verified that SUM_2 was equal to SUM_1 plus the completed New_Order transactions recorded by the RTE and that no entries existed for rolled-back transactions.
9. Consistency condition 3 was verified.

Failure of Durable Medium containing TPC-C data base tables.

The following steps were successfully performed to pass the Durability test of failure of a disk unit with data base tables:

1. The complete database is backed up to disk.
2. The current count of the total number of orders was determined by the sum of D_NEXT_O_ID of all rows in the DISTRICT table giving SUM_1.
3. A test was started and allowed to run for 10 minutes.
4. The disk containing some of the Customer, Orders, Order-line, Stock tables was powered off.
5. The run was aborted on the RTE.
6. The disk from step 4 was powered back on. The filesystems that contain the database were unmounted.
7. The DB2 for AIX database was restored from the backup copy in step 1.
8. DB2 for AIX was restarted and its transaction log was used to roll forward the transactions that had completed but weren't written to disk before the failure.
9. Step 2 was performed returning SUM_2. It was verified that SUM_2 was equal to SUM_1 plus the completed New_Order transactions recorded by the RTE and that no entries existed for rolled-back transactions.
10. Consistency condition 3 was verified.

.

5. Clause 4: Scaling and Data Base Population Related Items

5.1 Cardinality of Tables

The cardinality (e.g., the number of rows) of each table, as it existed at the start of the benchmark run, must be disclosed.

Table 5-1 portrays the TPC Benchmark™ C defined tables and the number of rows for each table as they were built initially.

Table 5-1 Initial Cardinality of Tables (RS/6000 R30 & J30)

Table Name	Rows
Warehouse	275
District	2,750
Item	100,000
History	8,250,000
Orders	8,250,000
Customer	8,250,000
New_order	2,475,000
Order_line	82,494,120
Stock	27,500,000

5.2 Distribution of Tables and Logs

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

The following diagram depict the data base configuration.

Figure 5-1 RS/6000 R30 & J30 Data Distribution Benchmark Configuration

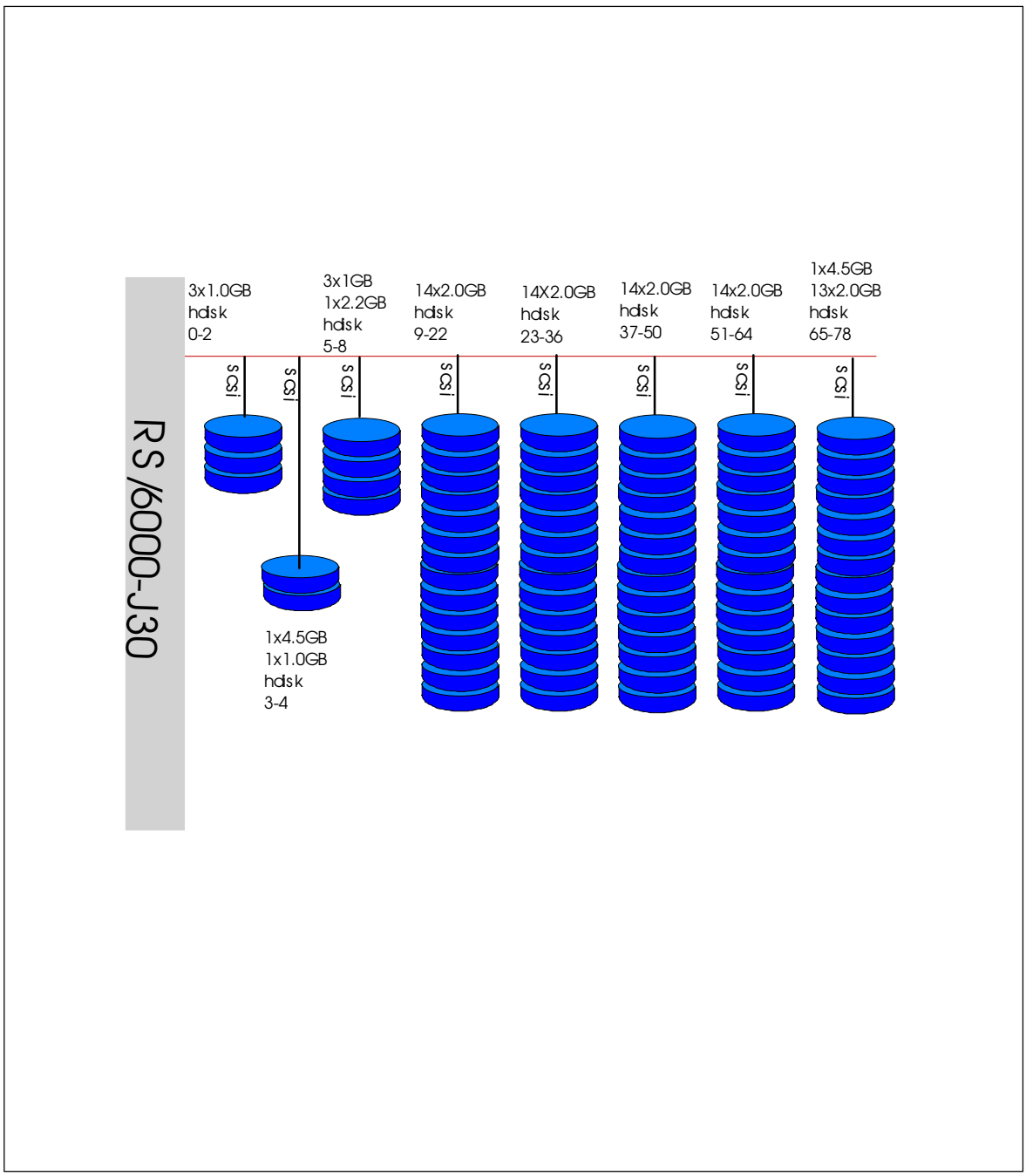
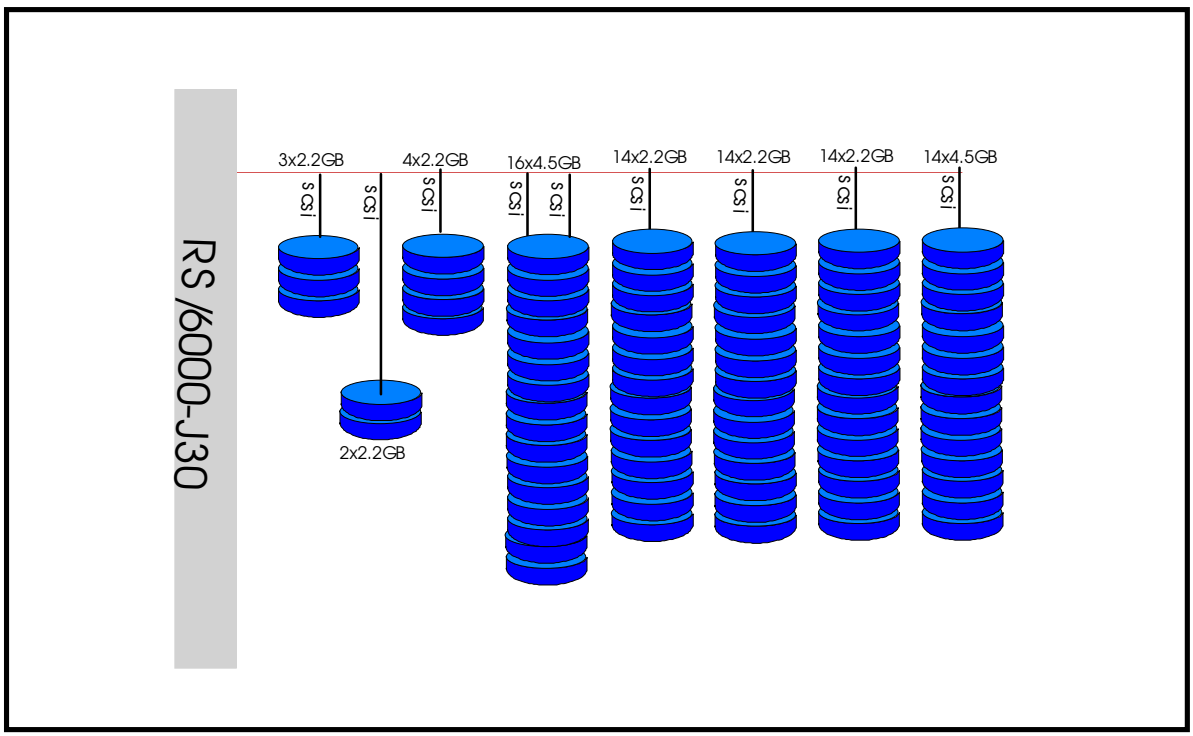
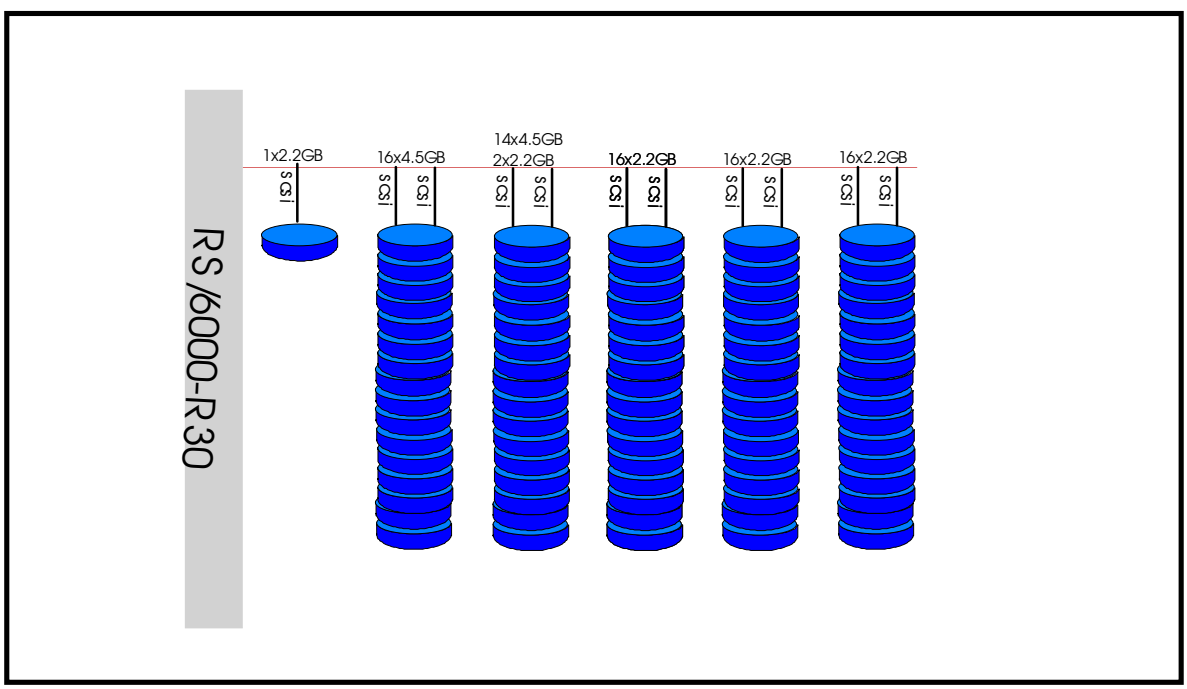


Figure 5-2 RS/6000 R30 & J30 Data Distribution Priced Configuration

The 1.0 GB and 2.0 GB drives were substituted with 2.2 GB or 4.5GB drive to accommodate the 180 day space. The 2.2 GB and 4.5 GB is a faster drive than either 1.0 GB or 2.0GB drive.





RS/6000 R30 & J30 Data and 180 Day Distribution Priced Configuration

The table below shows the distribution of the 180 day growth space in the priced configuration. Section 5.5 contains the details of the 180 day space calculations.

Disk	Capacity (GB)	Contents	Space allocns (GB)
hdisk0-80	51x2.2 30x4.5	OS	0.76
		Paging	3.12
		180 day growth	2.43
		Warehouse table	0.04MB
		District table	0.37MB
		Item table	0.01
		Stock table	9.59
		Customer table	7.18
		History table	0.54
		Order table	0.63
		New Order table	0.09
		Order line table	7.78
		Transaction Log	8.12
		Mirrored Transaction log	8.12
		180 day growth	214.36
		unused	1.48

5.3 Data Base Model Implemented

A statement must be provided that describes the data base model implemented by the DBMS used.

The relational data base used for this testing was DB2 for AIX Version 2.1 .

5.4 Partitions/Replications Mapping

The mapping of data base partitions/replications must be explicitly described.

IBM did not implement horizontal or vertical partitioning for these TPC-C tests.

5.5 180 day space calculations

RS/6000 PowerPC Server R30 & J30

Numbers are in Kbytes unless otherwise indicated.

Table	Data	Index	5% space	Daily Growth	File Space required
Warehouse	32	4	2	0	38
District	316	40	16	0	372
Item	9,760	1,340	488	0	11,588
History	478,512	0	0	86,840	565,352
Orders	297,448	313,156	0	53,980	664,584
Customer	6,602,644	593,272	330,132	0	7,526,048
New_order	44,420	44,200	2,221	0	90,841
Order_line	5,502,360	1,658,176	0	998,559	8,159,095
Stock	9,171,252	429,688	458,563	0	10,059,503

Allocation	6,491,024
Index Overhead	5,350
Total Page	14,160,000
Usable page	14,158,848
Space overhead	1,152
Static space	19,685,729
Dynamic space	6,278,320
Free space	47
Daily growth	1,139,379
Daily spread	0
180 day space(KB)	224,774,033

180 day space (GB)	214.36
Database logs (GB)	8.12
Mirror logs (GB)	8.12
Paging Space (GB)	3.12
OS, file systems etc (GB)	0.76
Required System disk capacity (GB)	231.36
Priced System capacity (GB)	232.84
Unused disk capacity (GB)	1.48

6. Clause 5: Performance Metrics and Response Time Related Items

6.1 Response Times

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

Table 6-1 lists the response times and the ninetieth percentiles for each of the transaction types for the measured systems.

6.2 Keying and Think Times

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

Table 6-1 lists the TPC-C keying and think times for the measured systems.

Table 6-1 RISC System/6000 R30 & J30 Response, Think and Keying Times

Response Times	New Order	Payment	Order Status	Delivery (int./def.)	Stock Level	Menus
90 %	3.12	2.6	2.53	0.46/3.0	5.11	0.7
Average	1.96	1.47	1.41	0.42/1.76	3.17	0.59
Maximum	7.76	6.53	4.54	4.31/7.0	10.89	15.75
			Think Times			
Minimum	0.01	0.01	0.01	0.01	0.01	N/A
Average	12.25	12.27	10.35	5.16	5.16	N/A
Maximum	113.74	122	97.21	40.16	52	N/A
			Keying Times			
Minimum	18	3	2	2	2	N/A
Average	18.06	3.02	2.01	2	2	N/A
Maximum	18.36	3.12	2.06	2.09	2.08	N/A

6.3 Response Time Frequency Distribution

Response time frequency distribution curves must be reported for each transaction type.

Figure 6-3-1. RS/6000 R30 & J30 New-Order Response Time Distribution

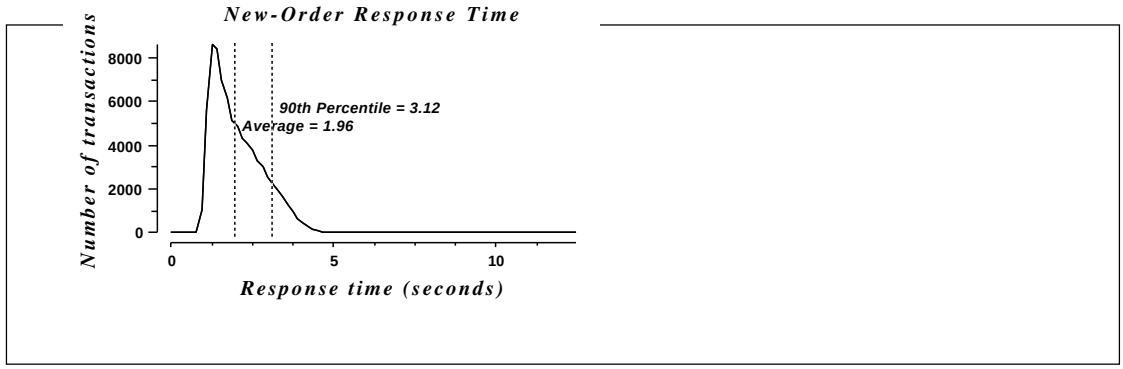


figure 6-3-2. RS/6000 R30 & J30 Payment Response Time Distribution

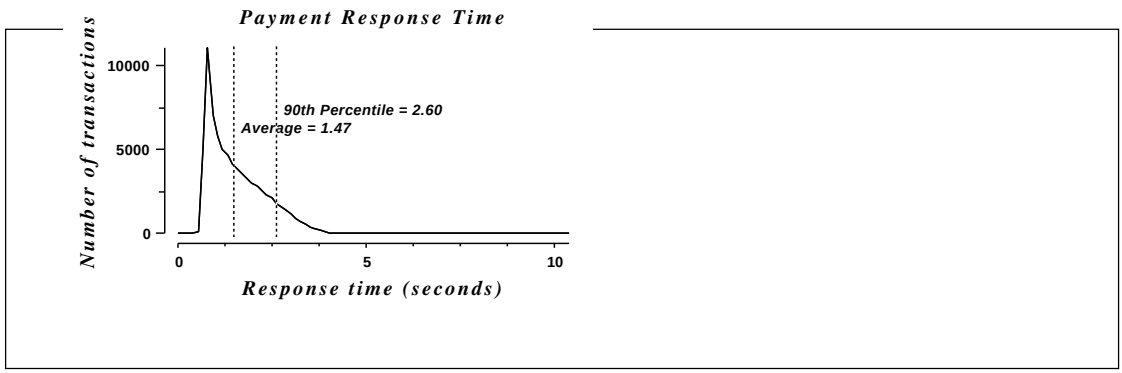


figure 6-3-3. RS/6000 R30 & J30 Order-Status Response Time Distribution

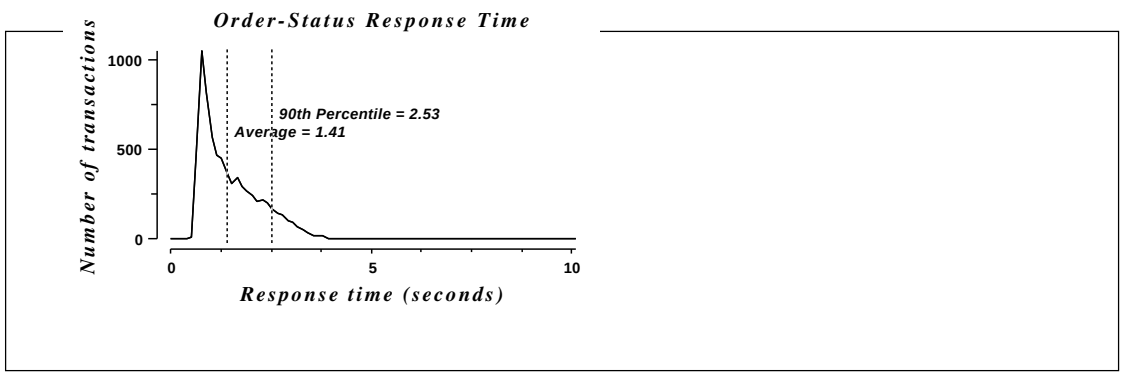


Figure 6-3-4. RS/6000 R30 & J30 Delivery (Interactive) Response Time Distribution

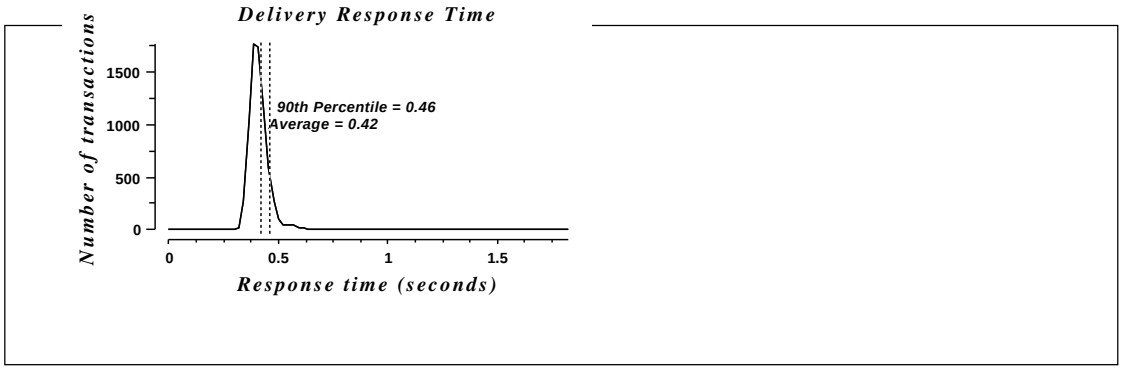


Figure 6-3-5. RS/6000 R30 & J30 Delivery (Deferred) Response Time Distribution

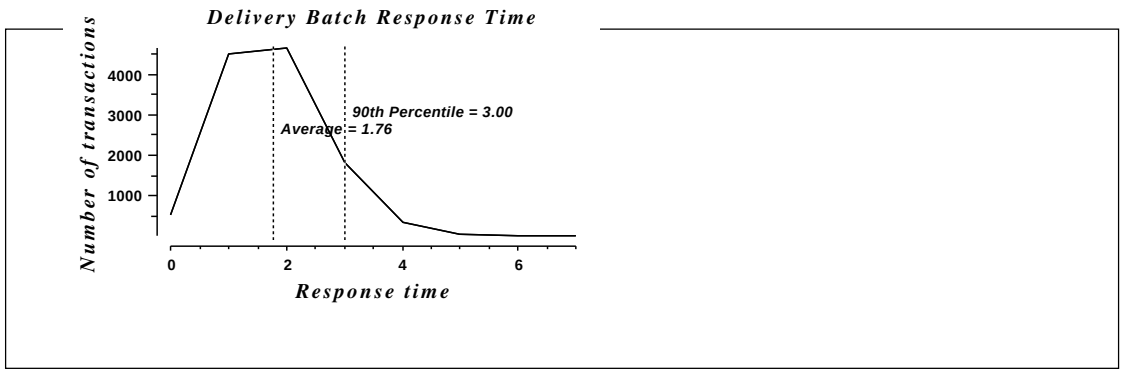
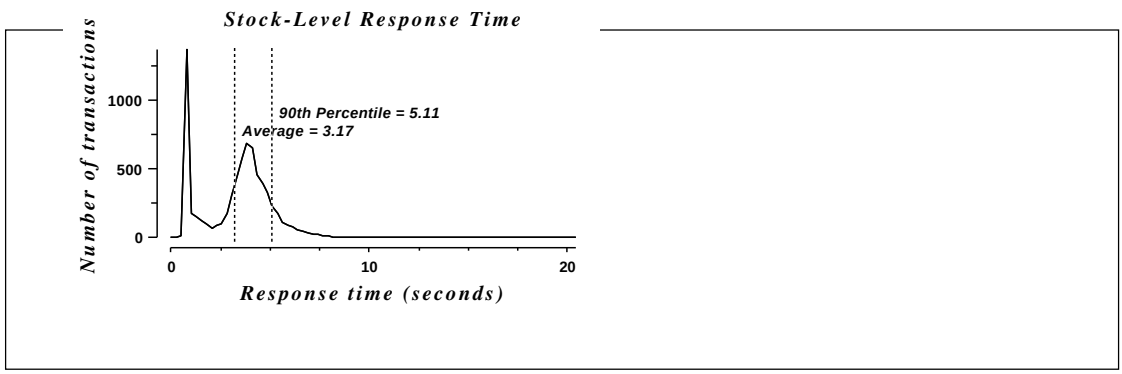


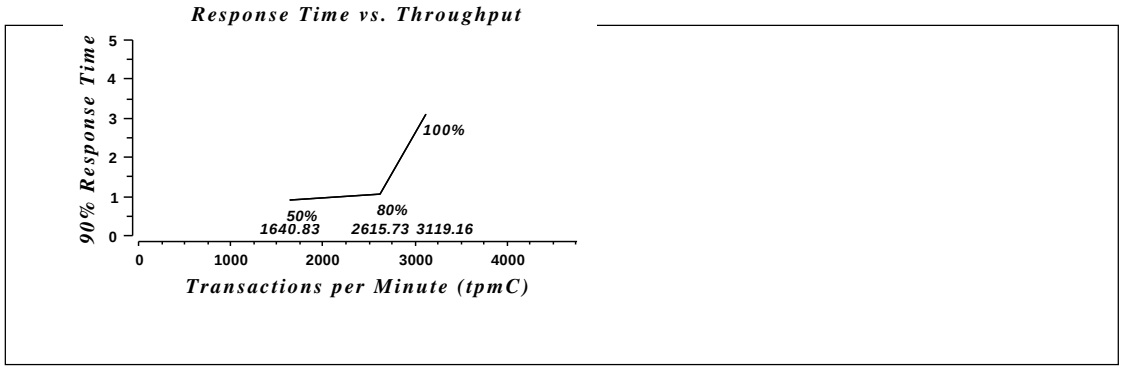
Figure 6-3-6. RS/6000 R30 & J30 Stock-Level Response Time Distribution



6.4 Performance Curve for Response Time versus Throughput

The performance curve for response times versus throughput must be reported for the New-Order transaction.

Figure 6-4-1. RISC System/6000 R30 & J30 New-Order Response Time vs. Throughput



6.5 Think Time Frequency Distribution

Think time frequency distribution curves must be reported for each transaction type.

Figure 6-5-1.RISC System/6000 R30 & J30 New-Order Think Time Distribution

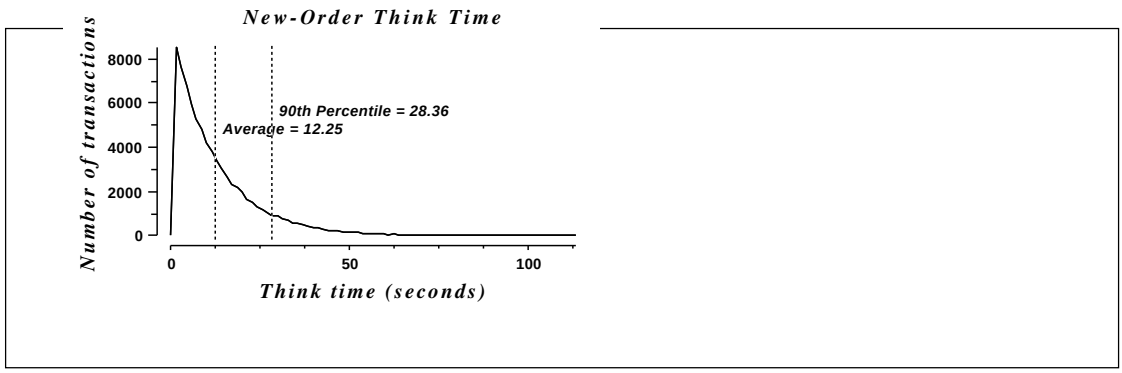


Figure 6-5-2.RISC System/6000 R30 & J30 Payment Think Time Distribution

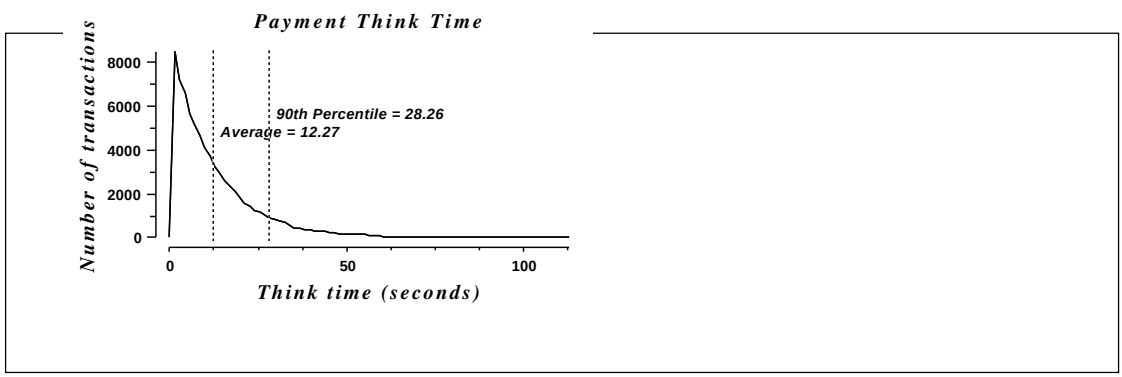


Figure 6-5-3.RISC System/6000 R30 & J30 Order-Status Think Time Distribution

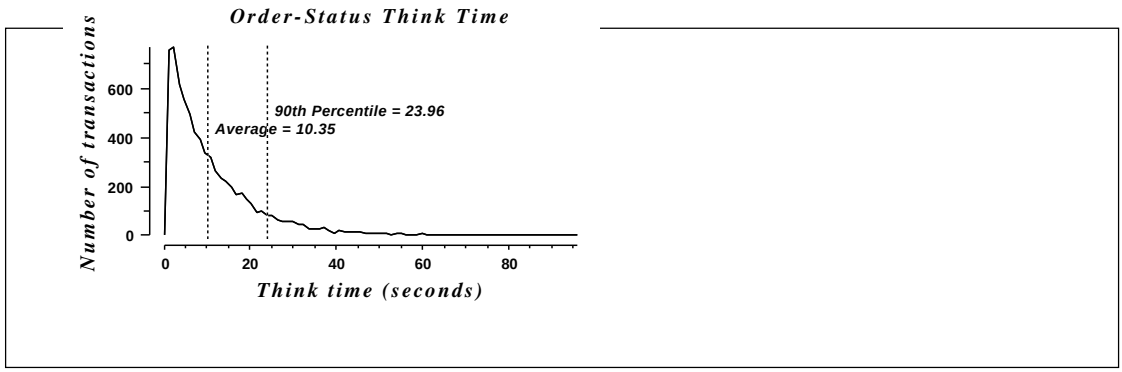


Figure 6-5-4.RISC System/6000 R30 & J30 Delivery Think Time Distribution

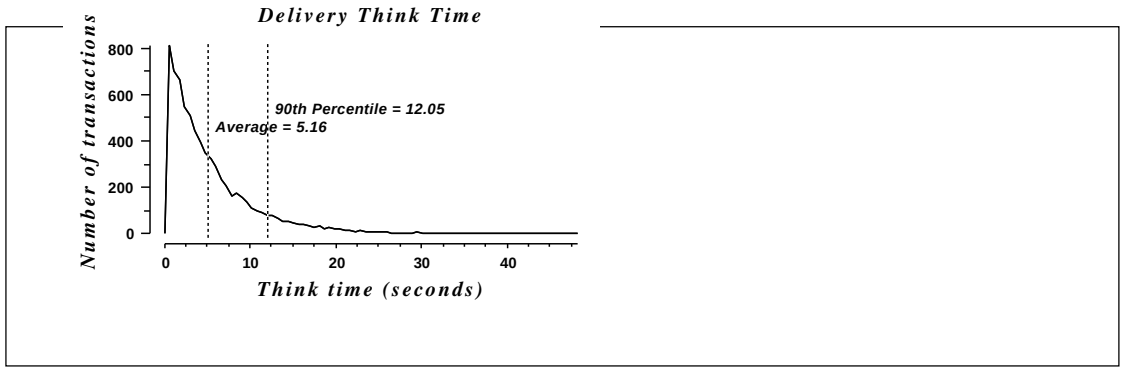
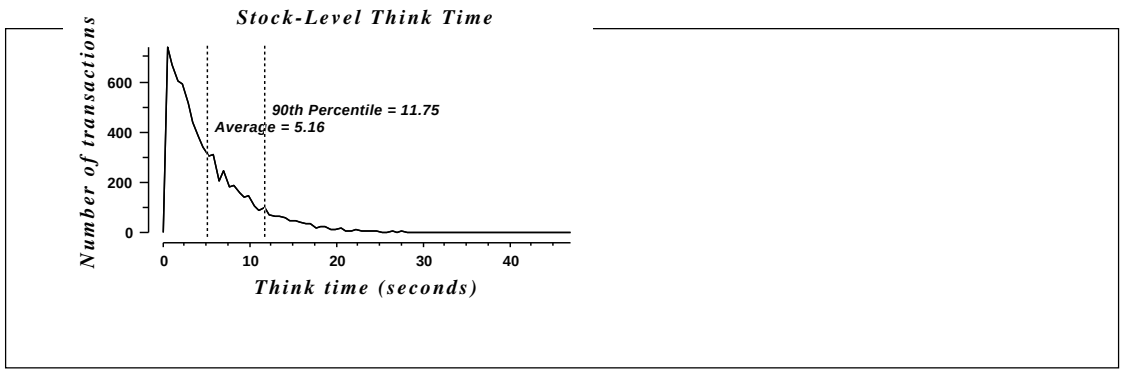


Figure 6-5-5.RISC System/6000 R30 & J30 Stock-Level Think Time Distribution



6.6 Keying Time Frequency Distribution

Keying time frequency distribution curves must be reported for each transaction type.

Figure 6-6-1. RISC System/6000 R30 & J30 New-Order Keying Time Distribution

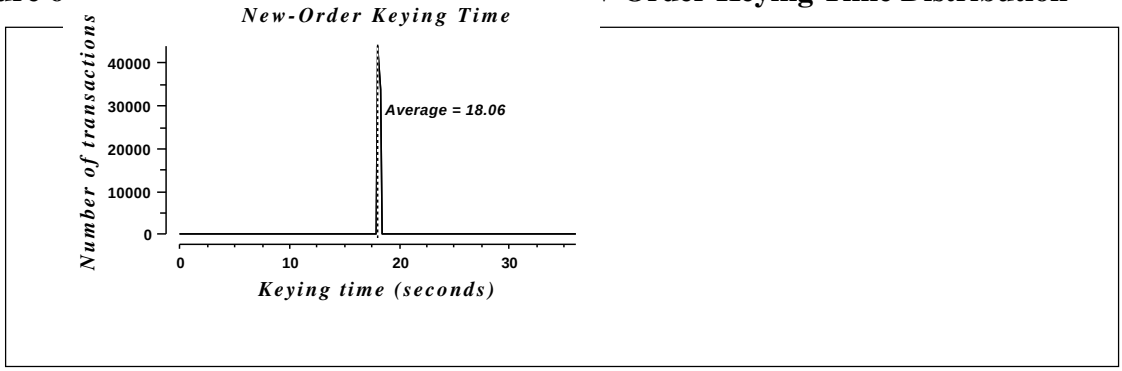


Figure 6-6-2. RISC System/6000 R30 & J30 Payment Keying Time Distribution

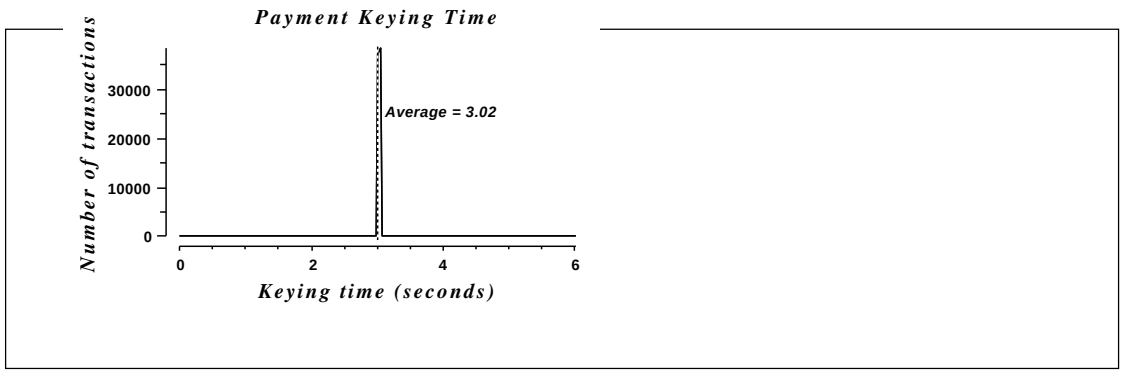


Figure 6-6-3. RISC System/6000 R30 & J30 Order-Status Keying Time Distribution

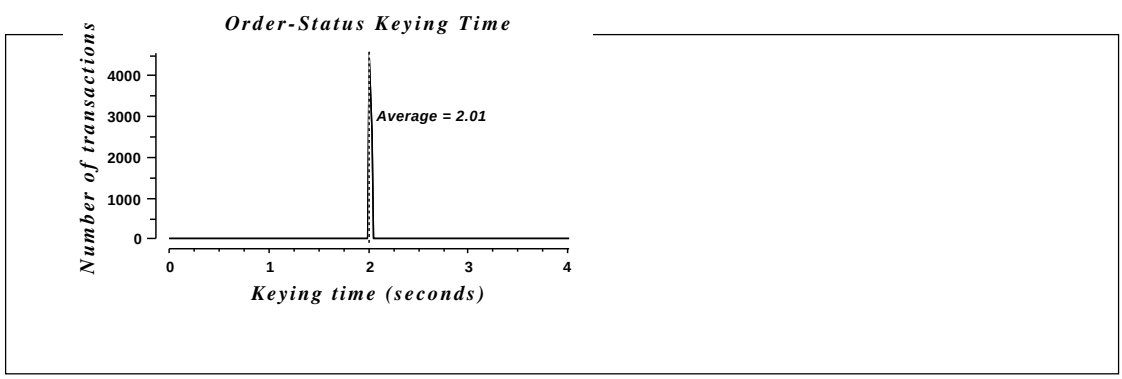


Figure 6-6-4.RISC System/6000 R30 & J30 Delivery Keying Time Distribution

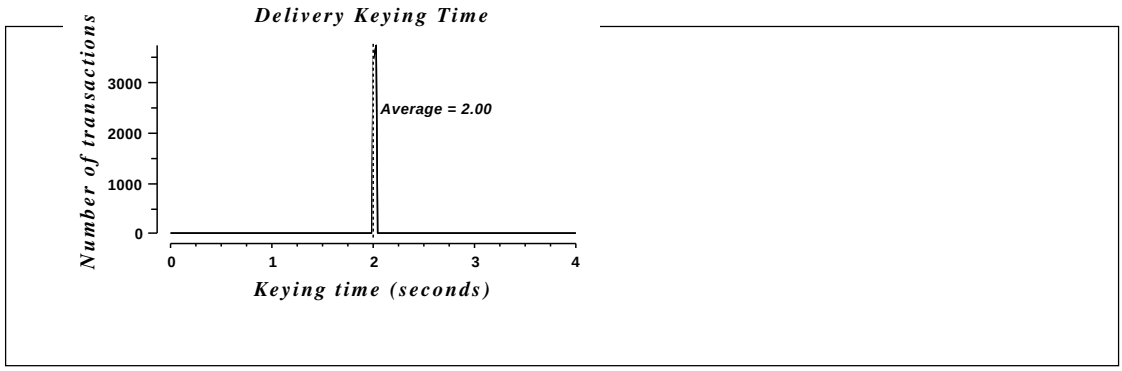
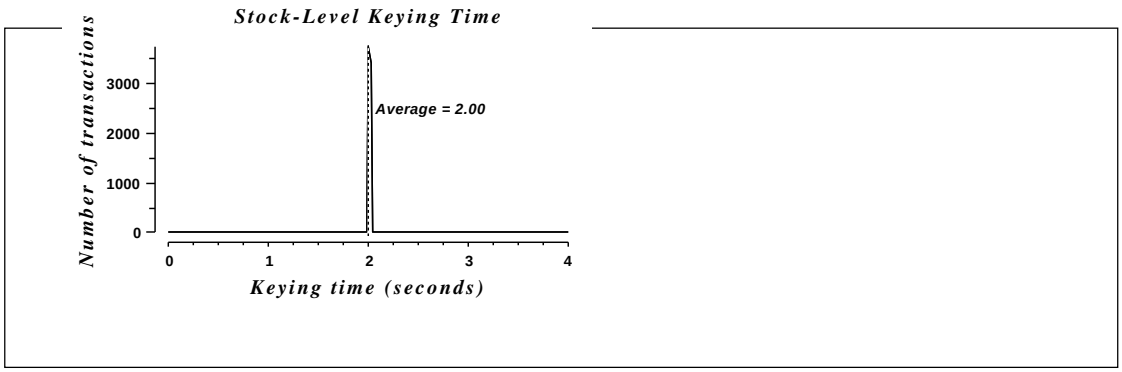


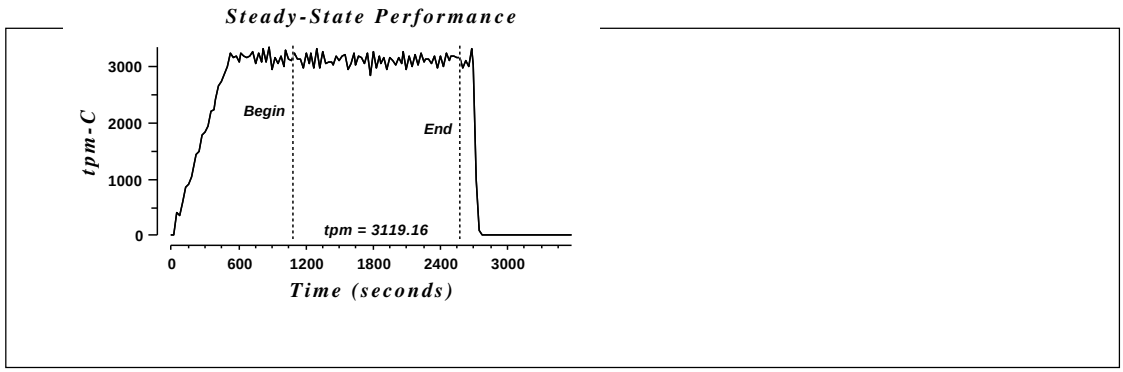
Figure 6-6-5. RISC System/6000 R30 & J30 Stock-Level Keying Time Distribution



6.7 Throughput versus Elapsed Time

A graph of throughput versus elapsed time must be reported for the New-Order transaction.

Figure 6-7-1. RISC System/6000 R30 & J30 New-Order Throughput vs. Elapsed Time



6.8 Steady State Determination

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

All the emulated users were allowed to logon and do transactions. The timestamping interval was set to start after several minutes of rampup. Refer to the Numerical Quantities Summary pages for the rampup times for the measured system. Figure 6.7.1 New-Order throughput versus Elapsed Time graph shows that the system was in steady state at the beginning of the Measurement Interval.

6.9 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.

6.9.1 Transaction Flow

For each of the TPC Benchmark™C transaction types, the following steps are executed:

Tuxedo System/T was used as a transaction manager (TM). Each transaction was divided into three programs: a front end program which handled all screen I/O, a database client program which connected to the database and served as a Tuxedo System/T server(a backend program), and a database server program which handled all database operations at the SUT. Both the front end and back end programs ran on the client system. The front end program communicates with the database client program through Tuxedo System/T messages. The database client program communicates with the Server system over Ethernet using DB2 for AIX DARI(Database Application Remote Interface) calls. Besides calling Tuxedo System/T functions for user connection and message communication, all other functions are transparent to the application code. Tuxedo System/T routes the transaction and balances the load according to the options defined in the Tuxedo System/T configuration file listed in appendix B.2. The transaction flow is described below.

- When Tuxedo System/T boots up, it creates one or more server process(es) for each transaction. Several server processes were defined in the Tuxedo System/T configuration file.
- Each TPC-C user invokes the TPC-C ***main*** (front end) program.
- The TPC-C ***main*** program connects to Tuxedo System/T before starting any transaction operation.
- The TPC-C ***main*** program displays the TPC-C transaction menu on the user terminal.
- The TPC-C user chooses the transaction type and proceeds to fill the screen fields required for that transaction.
- The TPC-C ***main*** program accepts all values entered by the user and transmits those values to one of the TPC-C back end programs. The transmission is performed through a Tuxedo System/T function call. Each TPC-C back end program has a "service-name". This service-name is specified whenever the TPC-C main program requests a Tuxedo System/T service. Tuxedo System/T routes that message according to the service-name and the information defined in the Tuxedo System/T configuration file.
- A TPC-C back end server program receives a message from its queue and proceeds to execute all database operations related to the service-name specified. All the information entered on the user terminal is contained in the Tuxedo System/T message.
- Once the transaction is committed, the TPC-C back end server program loads the message buffer with the transaction output and returns control to the Tuxedo System/T Manager.
- Tuxedo System/T manager routes the message back to the TPC-C ***main*** program.
- The TPC-C ***main*** program takes the message content and writes the transaction output on the user terminal.

6.9.2 Database Transaction

All database operations are performed by the TPC-C database client programs. The process is described below:

Using DB2 Client Application Enabler and DARI calls, the TPC-C back-end program interacts with DB2 for AIX to perform SQL data manipulations such as update, select, delete and insert, as required by the transaction. After all database operations are performed for a transaction, the transaction is committed.

DB2 for AIX proceeds to update the database as follows:

When DB2 for AIX changes a database table with an update, insert, or delete operation, the change is initially made in memory, not on disk. When there is not enough space in the memory buffer to read in additional data pages, DB2 for AIX will make space by flushing some modified pages to disk. Before a change is made to the database, it is first recorded in the transaction log. This ensures that the database can be recovered completely in the event of a failure. Using the transaction log, transactions that started but did not complete prior to a failure can be undone, and transactions recorded as complete in the transaction log but not yet written to disk can be redone.

6.9.3 Checkpoints

The DB2 for AIX 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 AIX. 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.

6.10 Reproducibility

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

A repeatability measurement was taken on the RS/6000 PowerPC Server J30 for the same length of time as the measured run. The repeatability measurement was 3084.60 tpmC.

6.11 Measurement Interval

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

A twenty five minute Measurement Interval was used for the RS/6000 R30 & J30 .

7. Clause 6: SUT, Driver, and Communication Definition Related Items

7.1 RTE Availability

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

IBM used an internally developed RTE for these tests. Appendix E contains the scripts used in the testing.

7.2 Functionality and Performance of 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.

In the priced configuration, terminals are connected to the client systems via Emulex P6000-88 Ethernet terminal concentrators. Transactions are submitted from the terminals (via terminal concentrators) to the client systems, which are front-ends of the DBMS server.

In the TPC-C SUT configuration, this terminal network (from the terminal to the front-end client) was emulated on the RS/6000-380 systems, acting as RTE drivers. This was done due to the large number of terminals, terminal servers and associated cables that would otherwise be required.

To measure the response time differences between those collected by the RTE and the actual response time when a real terminal and terminal server is used, a fully loaded EMULEX terminal server with 88 serial cables connected directly to an external driver was used to run the same TPC-C transactions as the emulated terminals, during a measurement at full system load. The measured response time differences in the table below are added to the response time reported by the RTE

Transction Type	New-Order	Payment	Order-Status	Delivery	Stock-Level
Transaction Resp Time Difference	0.42	0.19	0.39	0.19	0.25
Menu Resp Time Difference	0.38	0.23	0.21	0.15	0.18

7.3 Network Bandwidth

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

The Ethernet used in the LAN complies with the IEEE 802.3 standard and has a bandwidth of 10 Megabits per second.

7.4 Operator Intervention

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

The RS/6000 R30 & J30 configuration reported do not require any operator intervention to sustain the reported throughput during the eight hour period.

8. Clause 7: Pricing Related Items

8.1 Hardware and Programs Used

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

The detailed list of all hardware and programs for the priced configuration is listed in the pricing sheets (please refer to Section 8.2 for details) for each system reported. The prices for all products and features that are provided by IBM are available the same day as product or feature availability.

DB2 for AIX pricing is based on the number of database connections. Pricing for Tuxedo System/T was quoted by UNIX System Laboratories, Inc..

8.2 Five Year Cost of System Configuration

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.

The price sheets for the RS/6000-R30 & J30 are contained on the following pages. The basis for the discounts used are disclosed below.

- Extended Maintenance Option (EMO):
  - ► This is a discount for prepayment of maintenance costs for the system unit, disk, and the terminals. A discount of seventeen percent is available for this configuration based on payment for five years maintenance at time of purchase.
- Mid-Range Service Option (MRSO):
  - ► This discount is available for customers when agreement is reached for the customer to perform specified service duties (consult marketing representative for details). This discount is applied to the balance after the Extended Maintenance Option Discount is applied. For the TPC Benchmark™C configurations the MRSO discount is seventeen percent for the system, disk and terminals.
- Emulex Volume Discount
  - ► Emulex offers a 0% volume discount for the RS/6000 R30 & J30.

.

8.3 Availability Dates

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.

All products are currently generally available today except for the following products (general availability dates are provided):

-

Product	Availability Date
DB2 for AIX Version 2.1	December 15, 1995
CPU card 3 &4	August 25, 1995

8.4 Statement of tpmC and Price/Performance

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 disclosed.

.

System	tmpC	5-year System Cost	$/tmpC	Availability Date
RS/6000 R30	3,119.16	$1,106,716.01	$355	now except parts noted in section 8.3
RS/6000 J30	3,119.16	$1,089,830.51	$349	now except parts noted in section 8.3

IBM RISC System/6000-R30 Five Year System Price Configuration

Item Description	PART#	Unit Price	Qty	Ext Price	MAINT	TOTALS
PowerPC Server R30-8 Dual 601, 64mb Memory, 1.1gb Disk,SCSI2 Diff F/W Adptr, CDROM	7015-R30	80,900	1	80,900	38,832	119,732.00
Dual Processor card, 1mb L2	4302	15,000	3	45,000	21,600	66,600.00
Select 64mb->512mb Mem.	4146	36,450	1	36,450	0	36,450.00
Add 512mb Memory Cards	4158	48,600	3	145,800	0	145,800.00
Int. 1.1gb->2.2gb SCSI2 Sel.	3039	0	1	0	0	0.00
Ethernet Adapter	2980	722	1	722	0	722.00
SCSI-2 DE HiPerf Adapters	2416	1,370	10	13,700	0	13,700.00
System Rack Model R00	7202-900	3,110	2	6,220	2,976	9,196.00
Deep Door for MP Rack	6096	500	1	500	0	500.00
High Density SCSI Disk S/S	7134-010	13,400	5	67,000	33,600	100,600.00
4.5 GB Disk Drive Mod 14 Sel	2742	3,390	28	94,920	0	94,920.00
4.5 GB Disk Drive Mod	2714	4,500	2	9,000	0	9,000.00
2.2 GB Disk Drive Mod 14 Sel	2732	1,760	42	73,920	0	73,920.00
2.2 GB Disk Drive Module	2712	2,600	8	20,800	0	20,800.00
Internal Expansion Unit	4002	1,800	5	9,000	19,200	28,200.00
SCSI Cables	2902	321	5	1,605	0	1,605.00
POWERsrv C10 16mb/1gb	7009	10,995	3	32,985	16,560	49,545.00
External L2 Cache 1MB	4044	1,500	2	3,000	0	3,000.00
Select 4mb -> 32mb SIMM	4026	2,105	12	25,260	0	25,260.00
Add. 32mb SIMM Memory	4012	2,400	12	28,800	0	28,800.00
Select 1gb -> 2gb Disk	2583	1,300	3	3,900	0	3,900.00
Ethernet Adapter	9980	incl	3	0	0	0.00
Terminal Server Accelerator	2407	6,284	3	18,852	3,312	22,164.00
Emulex Terminal Servers	P6000	2,040	32	65,280	0	65,280.00
Emulex 8port modules	PE6008	324	344	111,456	0	111,456.00
Emulex Terminal Srv (spares)	P6000	2,040	4	8,160	0	8,160.00
Emulex 8port mod (spares)	PE6008	324	34	11,016	0	11,016.00
Freedom One Terminal	Liberty	259	4	1,036	152	1,188.00
		Hardware Subtotal		915,282	136,232	1,051,514.00
Software						
AIX 4.1.2 J30 + Support	5765-393	3,000	1	3,000	23,700	26,700.00
AIX 3.2.5 C10 + Support	5756-030	6,350	3	19,050	2,970	22,020.00
Async Term Srv Accel	5765-268	195	3	585	0	585.00
IMC Tuxedo EPT, C10 Client		3,500	3	10,500	7,875	18,375.00
DB2 for AIX V2.1 Server (60)		29,708	1	29,708	0	29,707.50
DB2/6000 Client Support		incl	1	0	0	0.00
Emulex Software - Site		149	1	149	0	149.00
		Software Subtotal		62,992	34,545	97,536.50
		System Total		978,274	170,777	1,149,050.50
Discounts	%Disc	Volume				
MidRange Service Agreement (23,133.60)	17%	136,080				
Extended Maint. Option (19,200.89)	17%	112,946				
						========
				Total Five Year Cost		1,106,716.01
				tpmC		3,119.16
				$/tpmC		354.81

Warranties:
1. All listed RISC System/6000 hardware includes One (1) Year Onsite Parts & Labor Warrant
2. Emulex P6000, PE6008 units include Five (5) Year Parts & Labor Warranty
3. Liberty Freedom One Terminals include Four (4) Years Parts & Labor Warranty

IBM RISC System/6000-J30 Five Year System Price Configuration

Item Description	PART#	Unit Price	Qty	Ext Price	MAINT	TOTALS
PowerPC Server J30-8 Dual 601, 64mb Memory, 2.2gb Disk,SCSI2 Diff F/W Adptr, CDROM	7013-J30	67,500	1	67,500	32,400	99,900.00
Dual Processor card, 1mb L2	4302	15,000	3	45,000	21,600	66,600.00
Select 64mb->512mb Mem.	4146	36,450	1	36,450	0	36,450.00
Add 512mb Memory Cards	4158	48,600	3	145,800	0	145,800.00
Int. 2.2gb SCSI2 Disk Select	9133	incl	1	0	0	0.00
Int. 2.2gb SCSI2 Disks	3053	2,500	6	15,000	0	15,000.00
Ethernet Adapter	2980	722	1	722	0	722.00
SCSI-2 DE HiPerf Adapters	2416	1,370	9	12,330	0	12,330.00
Expansion Cabinet	7013-J01	11,000	1	11,000	5,280	16,280.00
Int. 2.2gb SCSI2 Disks	3053	2,500	2	5,000	0	5,000.00
System Rack Model R00	7202-900	3,820	1	3,820	912	4,732.00
High Density SCSI Disk S/S	7134-010	13,400	5	67,000	33,600	100,600.00
4.5 GB Disk Drive Mod 14 Sel	2742	3,390	28	94,920	0	94,920.00
4.5 GB Disk Drive Module	2714	4,500	2	9,000	0	9,000.00
2.2 GB Disk Drive Mod 14 Sel	2732	1,760	42	73,920	0	73,920.00
Internal Expansion Unit	4002	1,800	5	9,000	19,200	28,200.00
SCSI Cables	2902	321	5	1,605	0	1,605.00
POWERsrv C10 16mb/1gb	7009	10,995	3	32,985	16,560	49,545.00
External L2 Cache 1MB	4044	1,500	2	3,000	0	3,000.00
Select 4mb -> 32mb SIMM	4026	2,105	12	25,260	0	25,260.00
Add. 32mb SIMM Memory	4012	2,400	12	28,800	0	28,800.00
Select 1gb -> 2gb Disk	2583	1,300	3	3,900	0	3,900.00
Ethernet Adapter	9980	incl	3	0	0	0.00
Terminal Server Accelerator	2407	6,284	3	18,852	3,312	22,164.00
Emulex Terminal Servers	P6000	2,040	32	65,280	0	65,280.00
Emulex 8port modules	PE6008	324	344	111,456	0	111,456.00
Emulex Terminal Srv (spares)	P6000	2,040	4	8,160	0	8,160.00
Emulex 8port mod (spares)	PE6008	324	34	11,016	0	11,016.00
Freedom One Terminal	Liberty	259	4	1,036	152	1,188.00
				-----	-----	-----
		Hardware Subtotal		907,812	133,016	1,040,828.00
Software						
AIX 4.1.2 J30 + Support	5765-393	3,000	1	3,000	16,500	19,500.00
AIX 3.2.5 C10 + Support	5756-030	6,350	3	19,050	2,970	22,020.00
Async Term Srv Accel	5765-268	195	3	585	0	585.00
IMC Tuxedo EPT, C10 Client		3,500	3	10,500	7,875	18,375.00
DB2 for AIX V2.1 Server (60)		29,708	1	29,708	0	29,707.50
DB2/6000 Client Support		incl	1	0	0	0.00
Emulex Software - Site		149	1	149	0	149.00
				-----	-----	-----
		Software Subtotal		62,992	27,345	90,336.50
				========	========	========
		System Total		970,804	160,361	1,131,164.50
Discounts	%Disc	Volume				
MidRange Service Agreement (22,586.88)	17%	132,864				
Extended Maint. Option (18,747.11)	17%	110,277				
						========
			Total Five Year Cost			1,089,830.51
			tpmC			3,119.16
			$/tpmC			349.40

Warranties:
1. All listed RISC System/6000 hardware includes One (1) Year Onsite Parts & Labor
2. Emulex P6000, PE6008 units include Five (5) Year Parts & Labor Warranty
3. Liberty Freedom One Terminals include Four (4) Years Parts & Labor Warranty

8. Clause 9: Audit Related Items

If the benchmark has been independently audited, then the auditor's name, address, phone number, and a brief audit summary report indicating compliance must be included in the Full Disclosure Report. A statement should be included, specifying when the complete audit report will become available and who to contact in order to obtain a copy.

The auditor's attestation letter is included at the front of this report.

Appendix A: TPC-C Application Source

A.1 Client/Terminal handler code

```
/************************************************************************/
/*                                                                      */
/*      File: main.c                                                    */
/*                                                                      */
/*      program description:                                            */
/*                                                                      */
/*      TPC-C benchmark of DB2 for AIX. this program comprises the      */
/*      TPCC benchmark transactions.                                    */
/*                                                                      */
/*                                                                      */
/************************************************************************/

/*
#define AUDIT
#define DEBUG
*/
/*--------------------------------------------------------*/
/* The makefile will include these modules                */
/*--------------------------------------------------------*/
#include <stdio.h>
#include "../include/tpcc.h"
#include "../app/tuxclient.h"
#include "./termio.h"

/*--------------------------------------------------------*/
/*              main                                      */
/*--------------------------------------------------------*/
main (argc, argv)
        int     argc;
        char    *argv[];
{
struct neword_struct     *neword;
struct payment_struct    *payment;
struct ordstat_struct    *ordstat;
struct delivery_struct   *delivery;
struct stocklev_struct   *stocklev;
char            buf[5], servname[30];
char            pbuf[15];
int             key;
int             p;

connect_to_TM();
load_scr_bufs(argc,argv);
set_up_port();
clear_menu();

for (;;) {
        read(fdi, buf, 1);
        switch (buf[0]) {
                case '1':
           neword = (struct neword_struct *)TMbuffer;
                clear_neword(neword);
                key = read_neword(neword);
         if (key != CNTRL_C_KEY) {
#ifdef AUDIT
strcpy(servname, "NEWORD_DSQL");
#else
/*
                    if (neword->s_O_OL_CNT < 5)
                      strcpy(servname, "NEWORD_DSQL");
                    else if (neword->s_all_local == 1)
                      sprintf(servname, "NEWORD_L_%d", neword->s_O_OL_CNT);
                    else
                      sprintf(servname, "NEWORD_R_%d", neword->s_O_OL_CNT);
          sprintf(servname, "NEWORD");
          sprintf(servname, "NEWORD_HP");
*/
          strcpy(servname, "neword_sql");
#endif
          key = submit_TM_tran(servname);
          neword = (struct neword_struct *)TMbuffer;
          if (key != -1)
                    write_neword(neword);
          else {
             p = 0;
             p += rowcol(2,1,&pbuf[p]);
             pbuf[p] = (char)NULL;
             write(fdo, pbuf, p);
             }
          }
                break;
               case '2':
          payment = (struct payment_struct *)TMbuffer;
                clear_payment(payment);
                key = read_payment(payment);
         if (key != CNTRL_C_KEY) {
          key = submit_TM_tran("payment_sql");
          payment = (struct payment_struct *)TMbuffer;
          if (key != -1)
                    write_payment(payment);
          else {
             p = 0;
             p += rowcol(2,1,&pbuf[p]);
             pbuf[p] = (char)NULL;
             write(fdo, pbuf, p);
             }
          }
                break;
               case '3':
         ordstat = (struct ordstat_struct *)TMbuffer;
                clear_ordstat(ordstat);
                key = read_ordstat(ordstat);
         if (key != CNTRL_C_KEY) {
          key = submit_TM_tran("ordstat_sql");
          ordstat = (struct ordstat_struct *)TMbuffer;
          if (key != -1)
                    write_ordstat(ordstat);
          else {
             p = 0;
             p += rowcol(2,1,&pbuf[p]);
             pbuf[p] = (char)NULL;
             write(fdo, pbuf, p);
             }
          }
                break;
               case '4':
         fprintf(stderr,"delivery transaction from main/n");
         delivery = (struct delivery_struct *)TMbuffer;
                clear_delivery(delivery);
                key = read_delivery(delivery);
         if (key != CNTRL_C_KEY) {
          key = submit_TM_atran("delivery_sql");
          strcpy(delivery->s_exec_status,
             "Delivery has been queued");
          if (key != -1)
                    write_delivery(delivery);
          else {
             p = 0;
             p += rowcol(2,1,&pbuf[p]);
             pbuf[p] = (char)NULL;
             write(fdo, pbuf, p);
             }
          }
                break;
               case '5':
         stocklev = (struct stocklev_struct *)TMbuffer;
                clear_stocklev(stocklev);
                key = read_stocklev(stocklev);
         if (key != CNTRL_C_KEY) {
          key = submit_TM_tran("stocklev_sql");
          stocklev = (struct stocklev_struct *)TMbuffer;
          if (key != -1)
                    write_stocklev(stocklev);
          else {
             p = 0;
             p += rowcol(2,1,&pbuf[p]);
             pbuf[p] = (char)NULL;
             write(fdo, pbuf, p);
             }
          }
                break;
#ifdef AUDIT
               case '6':
         neword = (struct neword_struct *)TMbuffer;
                clear_neword(neword);
                key = read_neword(neword);
         if (key != CNTRL_C_KEY) {
          submit_TM_tran("NEWORD_AUDIT");
          neword = (struct neword_struct *)TMbuffer;
                write_neword(neword);
          }
                break;
#endif
               case '9':
                    end_screen();
          disconnect_from_TM();
                    return(0);
               default:
                    break;
               } /* end switch statement */
   } /* end for loop */
}



/************************************************************************/
/*                                                                      */
/*      File: tpcc.h                                                    */
/*                                                                      */
/*      program description:                                            */
```

```
/*                                                                      */
/*     This module contains global variables and data definitions       */
/*     for the tpcc application.                                        */
/*                                                                      */
/************************************************************************/

/*     #define DEBUG          */

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <time.h>                       /* functions: time() */

#include "sqlenv.h"
#include "sqlenv.h"
#include "lval.h"

#define NULL_TIMESTAMP   '0001-01-01-00.00.01.000000'   /* used in new_order */
#define INVALID_ITEM            100
#define TRAN_OK                 0
#define DEADLOCK                -911
#define FATAL_SQLERROR          -1

#define LASTCODE                3
extern void prtsqlca();

extern char *current_tmstmp( void );

short ware_num;
short dist_num;
struct neword_struct {
  short   s_W_ID;
  short   s_D_ID;
  short   s_C_ID;
  char    s_C_LAST[17];
  char    s_C_CREDIT[3];
  double  s_C_DISCOUNT;
  short   s_O_OL_CNT;
  long    s_O_ID;
  char    s_O_ENTRY_D[20];
  char    s_status_line[25];
  double  s_total_amount;
  short   s_all_local;
  short   s_transtatus;
  float   s_W_TAX;   /* addition for CSTUXEDO */
  float   s_D_TAX;   /* addition for CSTUXEDO */
  struct  items_struct {
    short   s_OL_SUPPLY_W_ID;
    long    s_OL_I_ID;
    char    s_I_NAME[25];
    short   s_OL_QUANTITY;
    short   s_S_QUANTITY;
    char    s_brand_generic;
    double  s_I_PRICE;
    double  s_OL_AMOUNT;
  } item[15];
  short   driverid;
  short   duplicate_items;
};

struct payment_struct {
  short    s_W_ID;
  short    s_D_ID;
  short    s_C_ID;
  short    s_C_D_ID;
  short    s_C_W_ID;
  short    s_transtatus; /* addition for CSTUXEDO */
  double   s_H_AMOUNT;
  char     s_H_DATE[20];  /* addition for CSTUXEDO */
  char     s_W_STREET_1[21];
  char     s_W_STREET_2[21];
  char     s_W_CITY[21];
  char     s_W_STATE[3];
  char     s_W_ZIP[10];
  char     s_D_STREET_1[21];
  char     s_D_STREET_2[21];
  char     s_D_CITY[21];
  char     s_D_STATE[3];
  char     s_D_ZIP[10];
  char     s_C_FIRST[17];
  char     s_C_MIDDLE[3];
  char     s_C_LAST[17];
  char     s_C_STREET_1[21];
  char     s_C_STREET_2[21];
  char     s_C_CITY[21];
  char     s_C_STATE[3];
  char     s_C_ZIP[10];
  char     s_C_PHONE[17];
  char     s_C_SINCE[20]; /* addition for CSTUXEDO */
  char     s_C_CREDIT[3];
  double   s_C_CREDIT_LIM;
  double   s_C_DISCOUNT;
  double   s_C_BALANCE;
  char s_C_DATA[200];  /* addition for CSTUXEDO */
  short   driverid;
};

struct ordstat_struct {
  short   s_W_ID;
  short   s_D_ID;
  short   s_C_ID;
  char    s_C_FIRST[17];
  char    s_C_MIDDLE[3];
  char    s_C_LAST[17];
  double  s_C_BALANCE;
  long    s_O_ID;
  char    s_O_ENTRY_D[20];
  short   s_O_CARRIER_ID;
  short   s_ol_cnt;
  short   s_transtatus;    /* addition for CSTUXEDO */
  struct  oitems_struct {
    short   s_OL_SUPPLY_W_ID;
    long    s_OL_I_ID;
    short   s_OL_QUANTITY;
    double  s_OL_AMOUNT;
    char    s_OL_DELIVERY_D[20];
  } item[15];
  short   driverid;
};

struct delivery_struct {
  short   s_W_ID;
  short   s_O_CARRIER_ID;
  int     s_queued_time;
  short   s_transtatus;  /* addition for CSTUXEDO */
  char    s_exec_status[50];
  short   driverid;
  char    results[1024];
};

struct stocklev_struct {
  short   s_W_ID;
  short   s_D_ID;
  short   s_threshold;
  int     s_low_stock;
  short s_transtatus; /* addition for CSTUXEDO */
  short   driverid;
};



/************************************************************************/
/*                                                                      */
/*    File: tuxclient.h                                                 */
/*                                                                      */
/************************************************************************/

/*----------------------------------------------------------------*/
/*    include files                                               */
/*----------------------------------------------------------------*/
#include "atmi.h"        /* TUXEDO atmi library */

/*----------------------------------------------------------------*/
/*    global variables                                            */
/*----------------------------------------------------------------*/
int    transtatus;
char    *TMbuffer;
long    TMinbufsize, TMoutbufsize;

#define TMINBUFSIZE    1536
/* #define TMINBUFSIZE    4096 */

/*----------------------------------------------------------------*/
/*    connect to TM                                               */
/*----------------------------------------------------------------*/
connect_to_TM ()
{
TMinbufsize = TMINBUFSIZE;

if (tpinit((TPINIT *)NULL) == -1) {
    printf("rtn: tpinit(), tperrno: %d \n", tperrno);
    exit(1);
    }

if ((TMbuffer = tpalloc("CARRAY", NULL, TMinbufsize)) == NULL) {
    printf("rtn: tpalloc(), tperrno: %d \n", tperrno);
    exit(2);
    }
}

/*----------------------------------------------------------------*/
/*    disconnect from TM                                          */
/*----------------------------------------------------------------*/
disconnect_from_TM ()
```

```
{
if (tpterm() == -1) {
    printf("rtn: tpterm(), tperrno: %d \n", tperrno);
    exit(3);
    }
}

/*---------------------------------------------------------------*/
/*    submit TM tran                                             */
/*---------------------------------------------------------------*/
submit_TM_tran (servname)
    char    *servname;
{
transtatus = tpcall(servname,TMbuffer,TMinbufsize,&TMbuffer,
          &TMoutbufsize, 0);
if (transtatus == -1) {
    char    str[60];
    sprintf(str, "rtn: tpcall(), tperrno: %d, transtatus: %d \n",
        tperrno, transtatus);
    fprintf(stderr, str);
    return(-1);
    }
return(0);
}

/*---------------------------------------------------------------*/
/*    submit TM asynchronous transaction                         */
/*---------------------------------------------------------------*/
submit_TM_atran (servname)
    char    *servname;
{
transtatus = tpacall(servname,TMbuffer,TMinbufsize, TPNOREPLY);
if (transtatus == -1) {
    char    str[60];
    sprintf(str, "rtn: tpacall(), tperrno: %d, transtatus: %d \n",
        tperrno, transtatus);
    fprintf(stderr, str);
    return(-1);
    }
return(0);
}




/*************************************************************************/
/*                                                                       */
/*      File: termio.h                                                   */
/*                                                                       */
/*      program description:                                             */
/*                                                                       */
/*      This module contains all the screen io subroutines for the       */
/*      TPC-C benchmark.                                                 */
/*                                                                       */
/*************************************************************************/
/*
#define DEBUG
#define ACID_TEST
*/

/*-----------------------------------------------*/
/*    terminal types                             */
/*-----------------------------------------------*/

/*
#define HFT
*/
#define WYSE50_TERM

/*-----------------------------------------------*/
/*    include files                              */
/*-----------------------------------------------*/
#ifndef TPCCH
/* #include "../include/tpcc.h" */
#endif

#include <stdio.h>
#include <termio.h>
#include <string.h>
#include <time.h>       /* functions: time() */

#define FORM_DATE          1
#define FORM_DATETIME      2

#define ERRMSG_ROW       2

#define INVALID_ITEM       100
#define FATAL_SQLERROR      -1

#define SCR_READ_NUM 100 /* minimum number of characters to leave screen */
#define SCR_READ_TIMEOUT 2 /* minimum time to leave screen, 10=1sec */
#define CIRBUF_SIZE       1000

/*-----------------------------------------------*/
/*            Keyboard Keys                      */
/*-----------------------------------------------*/
#define ENTER_KEY    10
#define TAB_KEY       9
#define BACK_TAB_KEY   2
#define BACKSPACE_KEY    8
#define CNTRL_C_KEY    3

/*-----------------------------------------------*/
/*            Video Atributes                    */
/*-----------------------------------------------*/
#define SPACE            ' '
#define UNDERL           '_'

#ifdef HFT
#define CLEAR_SCREEN_1       "[1H[J"
#define CLEAR_SCREEN_3       "[3H[J"
#define BEEP_SOUND       ""
#endif

/* wyse, televideo, fredom one, ... */
#ifdef ASCII_TERM
#define CLEAR_SCREEN_1       "=  Y"
#define CLEAR_SCREEN_3       "=! Y"
#define BEEP_SOUND       ""
#endif

#ifdef WYSE50_TERM
#define CLEAR_SCREEN_1       "\033=  \033Y"
#define CLEAR_SCREEN_3       "\033=\" \033Y"
#define BEEP_SOUND       ""
#endif

/*-----------------------------------------------*/
/*            Terminal Buffers                   */
/*-----------------------------------------------*/
static char    clear_menu_buf[100];
static char    clear_neword_buf[900];
static char    clear_payment_buf[350];
static char    clear_ordstat_buf[300];
static char    clear_delivery_buf[150];
static char    clear_stocklev_buf[150];

/*-----------------------------------------------*/
/*            Buffers Size                       */
/*-----------------------------------------------*/
static int     clear_menu_size;
static int     clear_neword_size;
static int     clear_payment_size;
static int     clear_ordstat_size;
static int     clear_delivery_size;
static int     clear_stocklev_size;

/*-----------------------------------------------*/
/*            screen routines                    */
/*-----------------------------------------------*/
static int     clear_menu();

static int     clear_neword();
static int     read_neword();
static int     write_neword();

static int     clear_payment();
static int     read_payment();
static int     write_payment();

static int     clear_ordstat();
static int     read_ordstat();
static int     write_ordstat();

static int     clear_delivery();
static int     read_delivery();
static int     write_delivery();

static int     clear_stocklev();
static int     read_stocklev();
static int     write_stocklev();

/*-----------------------------------------------*/
/*            tty definitions                    */
/*-----------------------------------------------*/
static struct termio in_tty,out_tty, savetty;
static int fdi=0,fdo=1;

/*---------------------------------------------------------------*/
/* sets port up for fast transfer in raw mode                    */
/*---------------------------------------------------------------*/
set_up_port()
{
struct termio fastport;
```

```
ioctl(fdi, TCGETA, &in_tty);
ioctl(fdo, TCGETA, &out_tty);

memcpy(&savetty, &in_tty, sizeof(struct termio));

fastport = in_tty;
fastport.c_lflag = 0;
fastport.c_oflag = 0;
fastport.c_cc[VMIN] = SCR_READ_NUM; /* min num of chars to leave screen */
fastport.c_cc[VTIME] = SCR_READ_TIMEOUT; /* minimum time to leave screen,
10=1sec */

ioctl(fdo, TCSETAW, &fastport);
ioctl(fdi, TCSETAW, &fastport);

ioctl(fdo, TCFLSH, 2 );
ioctl(fdi, TCFLSH, 2 );
}

/*---------------------------------------------------------------*/
/*              load screen buffers                              */
/*---------------------------------------------------------------*/
load_scr_bufs (argc,argv)
        int     argc;
        char    *argv[];
{
if (argc < 3) {
        printf("*** error: \n");
        printf("        - warehouse id must be supplied.\n");
        printf("        - district id must be supplied.\n");
        exit(1);
        }

ware_num = atoi(argv[1]);
dist_num = atoi(argv[2]);

strcpy(clear_menu_buf, CLEAR_SCREEN_1);
rowcol(1,1, &clear_menu_buf[strlen(clear_menu_buf)]);
strcat(clear_menu_buf, "(1)New-Order (2)Payment (3)Order-Status (4)");
strcat(clear_menu_buf, "Delivery (5)Stock-Level (9)Exit ");

clear_menu_size = strlen(clear_menu_buf);

strcpy(clear_neword_buf, CLEAR_SCREEN_3);
rowcol(3,36, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "New Order");
rowcol(4,1, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "Warehouse: ");
sprintf(&clear_neword_buf[strlen(clear_neword_buf)], "%d", ware_num);
rowcol(4,19, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "District:");
rowcol(4,55, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "Date:");
rowcol(5,1, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "Customer: ____");
rowcol(5,19, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "Name:");
rowcol(5,44, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "Credit:");
rowcol(5,57, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "Disc:");
rowcol(6,1, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "Order Number:");
rowcol(6,25, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "Number of Lines:");
rowcol(6,52, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "W_tax:");
rowcol(6,67, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "D_tax:");
rowcol(6,52, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "W_tax:");
rowcol(6,67, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "D_tax:");
rowcol(8,2, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "Supp_W  Item_Id  Item Name");
rowcol(8,45, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "Qty  Stock  B/G  Price    Amount");
rowcol(9,3, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "____");
rowcol(9,10, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "_______");
rowcol(9,45, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "__");
rowcol(10,3, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "____");
rowcol(10,10, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "_______");
rowcol(10,45, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "__");
rowcol(11,3, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "____");
rowcol(11,10, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "_______");
rowcol(11,45, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "__");
rowcol(12,3, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "____");
rowcol(12,10, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "_______");
rowcol(12,45, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "__");
rowcol(13,3, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "____");
rowcol(13,10, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "_______");
rowcol(13,45, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "__");
rowcol(14,3, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "____");
rowcol(14,10, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "_______");
rowcol(14,45, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "__");
rowcol(15,3, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "____");
rowcol(15,10, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "_______");
rowcol(15,45, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "__");
rowcol(16,3, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "____");
rowcol(16,10, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "_______");
rowcol(16,45, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "__");
rowcol(17,3, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "____");
rowcol(17,10, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "_______");
rowcol(17,45, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "__");
rowcol(18,3, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "____");
rowcol(18,10, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "_______");
rowcol(18,45, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "__");
rowcol(19,3, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "____");
rowcol(19,10, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "_______");
rowcol(19,45, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "__");
rowcol(20,3, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "____");
rowcol(20,10, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "_______");
rowcol(20,45, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "__");
rowcol(21,3, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "____");
rowcol(21,10, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "_______");
rowcol(21,45, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "__");
rowcol(22,3, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "____");
rowcol(22,10, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "_______");
rowcol(22,45, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "__");
rowcol(23,3, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "____");
rowcol(23,10, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "_______");
rowcol(23,45, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "__");
rowcol(24,1, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "Execution Status:");
rowcol(24,62, &clear_neword_buf[strlen(clear_neword_buf)]);
strcat(clear_neword_buf, "Total:");

clear_neword_size = strlen(clear_neword_buf);

strcpy(clear_payment_buf, CLEAR_SCREEN_3);
rowcol(3,38, &clear_payment_buf[strlen(clear_payment_buf)]);
strcat(clear_payment_buf, "Payment");
rowcol(4,1, &clear_payment_buf[strlen(clear_payment_buf)]);
strcat(clear_payment_buf, "Date:");
rowcol(6,1, &clear_payment_buf[strlen(clear_payment_buf)]);
strcat(clear_payment_buf, "Warehouse: ");
sprintf(&clear_payment_buf[strlen(clear_payment_buf)], "%d", ware_num);
rowcol(6,42, &clear_payment_buf[strlen(clear_payment_buf)]);
strcat(clear_payment_buf, "District: __");
rowcol(11,1, &clear_payment_buf[strlen(clear_payment_buf)]);
```

```
strcat(clear_payment_buf, "Customer: ____");
rowcol(11,17, &clear_payment_buf[strlen(clear_payment_buf)]);
strcat(clear_payment_buf, "Cust-Warehouse: ____");
rowcol(11,39, &clear_payment_buf[strlen(clear_payment_buf)]);
strcat(clear_payment_buf, "Cust-District: __");
rowcol(12,1, &clear_payment_buf[strlen(clear_payment_buf)]);
strcat(clear_payment_buf, "Name:");
rowcol(12,29, &clear_payment_buf[strlen(clear_payment_buf)]);
strcat(clear_payment_buf, "____________________");
rowcol(12,50, &clear_payment_buf[strlen(clear_payment_buf)]);
strcat(clear_payment_buf, "Since:");
rowcol(13,50, &clear_payment_buf[strlen(clear_payment_buf)]);
strcat(clear_payment_buf, "Credit:");
rowcol(14,50, &clear_payment_buf[strlen(clear_payment_buf)]);
strcat(clear_payment_buf, "%Disc:");
rowcol(15,50, &clear_payment_buf[strlen(clear_payment_buf)]);
strcat(clear_payment_buf, "Phone:");
rowcol(17,1, &clear_payment_buf[strlen(clear_payment_buf)]);
strcat(clear_payment_buf, "Amount Paid:");
/* rowcol(17,18, &clear_payment_buf[strlen(clear_payment_buf)]); */
rowcol(17,24, &clear_payment_buf[strlen(clear_payment_buf)]);
strcat(clear_payment_buf, "________");
rowcol(17,38, &clear_payment_buf[strlen(clear_payment_buf)]);
strcat(clear_payment_buf, "New Cust-Balance");
rowcol(18,1, &clear_payment_buf[strlen(clear_payment_buf)]);
strcat(clear_payment_buf, "Credit Limit:");
rowcol(20,1, &clear_payment_buf[strlen(clear_payment_buf)]);
strcat(clear_payment_buf, "Cust-Data:");

clear_payment_size = strlen(clear_payment_buf);

strcpy(clear_ordstat_buf, CLEAR_SCREEN_3);
rowcol(3,35, &clear_ordstat_buf[strlen(clear_ordstat_buf)]);
strcat(clear_ordstat_buf, "Order-Status");
rowcol(4,1, &clear_ordstat_buf[strlen(clear_ordstat_buf)]);
strcat(clear_ordstat_buf, "Warehouse: ");
sprintf(&clear_ordstat_buf[strlen(clear_ordstat_buf)], "%d", ware_num);
rowcol(4,19, &clear_ordstat_buf[strlen(clear_ordstat_buf)]);
strcat(clear_ordstat_buf, "District:");
rowcol(5,1, &clear_ordstat_buf[strlen(clear_ordstat_buf)]);
strcat(clear_ordstat_buf, "Customer: ____   Name:");
rowcol(5,44, &clear_ordstat_buf[strlen(clear_ordstat_buf)]);
strcat(clear_ordstat_buf, "____________________");
rowcol(6,1, &clear_ordstat_buf[strlen(clear_ordstat_buf)]);
strcat(clear_ordstat_buf, "Cust-Balance:");
rowcol(8,1, &clear_ordstat_buf[strlen(clear_ordstat_buf)]);
strcat(clear_ordstat_buf, "Order-Number:");
rowcol(8,26, &clear_ordstat_buf[strlen(clear_ordstat_buf)]);
strcat(clear_ordstat_buf, "Entry-Date:");
rowcol(8,60, &clear_ordstat_buf[strlen(clear_ordstat_buf)]);
strcat(clear_ordstat_buf, "Carrier-Number:");
rowcol(9,1, &clear_ordstat_buf[strlen(clear_ordstat_buf)]);
strcat(clear_ordstat_buf, "Supply-W:");
rowcol(9,14, &clear_ordstat_buf[strlen(clear_ordstat_buf)]);
strcat(clear_ordstat_buf, "Item-Id:");
rowcol(9,25, &clear_ordstat_buf[strlen(clear_ordstat_buf)]);
strcat(clear_ordstat_buf, "Qty");
rowcol(9,33, &clear_ordstat_buf[strlen(clear_ordstat_buf)]);
strcat(clear_ordstat_buf, "Amount");
rowcol(9,45, &clear_ordstat_buf[strlen(clear_ordstat_buf)]);
strcat(clear_ordstat_buf, "Delivery-Date");

clear_ordstat_size = strlen(clear_ordstat_buf);

strcpy(clear_delivery_buf, CLEAR_SCREEN_3);
rowcol(3,38, &clear_delivery_buf[strlen(clear_delivery_buf)]);
strcat(clear_delivery_buf, "Delivery");
rowcol(4,1, &clear_delivery_buf[strlen(clear_delivery_buf)]);
strcat(clear_delivery_buf, "Warehouse: ");
sprintf(&clear_delivery_buf[strlen(clear_delivery_buf)], "%d", ware_num);
rowcol(6,1, &clear_delivery_buf[strlen(clear_delivery_buf)]);
strcat(clear_delivery_buf, "Carrier Number: __");
rowcol(8,1, &clear_delivery_buf[strlen(clear_delivery_buf)]);
strcat(clear_delivery_buf, "Execution Status:");

clear_delivery_size = strlen(clear_delivery_buf);

strcpy(clear_stocklev_buf, CLEAR_SCREEN_3);
rowcol(3,35, &clear_stocklev_buf[strlen(clear_stocklev_buf)]);
strcat(clear_stocklev_buf, "Stock-Level");
rowcol(4,1, &clear_stocklev_buf[strlen(clear_stocklev_buf)]);
strcat(clear_stocklev_buf, "Warehouse: ");
sprintf(&clear_stocklev_buf[strlen(clear_stocklev_buf)], "%d", ware_num);
rowcol(4,19, &clear_stocklev_buf[strlen(clear_stocklev_buf)]);
strcat(clear_stocklev_buf, "District: ");
sprintf(&clear_stocklev_buf[strlen(clear_stocklev_buf)], "%d", dist_num);
rowcol(6,1, &clear_stocklev_buf[strlen(clear_stocklev_buf)]);
strcat(clear_stocklev_buf, "Stock Level Threshold: __");
rowcol(8,1, &clear_stocklev_buf[strlen(clear_stocklev_buf)]);
strcat(clear_stocklev_buf, "low stock:");

clear_stocklev_size = strlen(clear_stocklev_buf);
```

```
#ifdef DEBUG
fprintf(stderr, "clear_menu_size: %d \n", clear_menu_size);
fprintf(stderr, "clear_neword_size: %d \n", clear_neword_size);
fprintf(stderr, "clear_payment_size: %d \n", clear_payment_size);
fprintf(stderr, "clear_ordstat_size: %d \n", clear_ordstat_size);
fprintf(stderr, "clear_delivery_size: %d \n", clear_delivery_size);
fprintf(stderr, "clear_stocklev_size: %d \n", clear_stocklev_size);
fprintf(stderr, "%s \n", clear_neword_buf);
#endif
}

/*----------------------------------------------------------------*/
/*              rowcol                                            */
/*----------------------------------------------------------------*/
static int rowcol (r,c,str)
    int    r,c;
    char    *str;
{
static int    size;

#ifdef HFT
size = 4;
if (r < 10) size += 1;
else size += 2;
if (c < 10) size += 1;
else size += 2;
sprintf(str, "[%d;%dH", r, c);
return (size);
#endif

#ifdef ASCII_TERM
str[0] = '';
str[1] = '=';
str[2] = ' ' + r - 1;
str[3] = ' ' + c - 1;
str[4] = '\0'
return(4);
#endif

#ifdef WYSE50_TERM
str[0] = '\033';
str[1] = '=';
str[2] = ' ' + r - 1 ;
str[3] = ' ' + c - 1;
str[4] = '\0';
return(4);
#endif
}

/*----------------------------------------------------------------*/
/*              clear menu                                        */
/*----------------------------------------------------------------*/
static int clear_menu ()
{
write(fdo, clear_menu_buf, clear_menu_size);
}

/*----------------------------------------------------------------*/
/*              clear new order screen                            */
/*----------------------------------------------------------------*/
static int clear_neword (neword)
        struct neword_struct    *neword;
{
neword->s_W_ID = ware_num;

write(fdo, clear_neword_buf, clear_neword_size);
}

/*----------------------------------------------------------------*/
/*              clear payment screen                              */
/*----------------------------------------------------------------*/
static int clear_payment (payment)
        struct payment_struct    *payment;
{
payment->s_W_ID = ware_num;

write(fdo, clear_payment_buf, clear_payment_size);
}

/*----------------------------------------------------------------*/
/*              clear ordstat screen                              */
/*----------------------------------------------------------------*/
static int clear_ordstat (ordstat)
        struct ordstat_struct    *ordstat;
{
ordstat->s_W_ID = ware_num;

write(fdo, clear_ordstat_buf, clear_ordstat_size);
}

/*----------------------------------------------------------------*/
/*              clear delivery screen                             */
/*----------------------------------------------------------------*/
```

```
static int clear_delivery (delivery)
        struct delivery_struct    *delivery;
{
delivery->s_W_ID = ware_num;

write(fdo, clear_delivery_buf, clear_delivery_size);
}


/*---------------------------------------------------------------*/
/*              clear stocklev screen                            */
/*---------------------------------------------------------------*/
static int clear_stocklev (stocklev)
        struct stocklev_struct    *stocklev;
{
stocklev->s_W_ID = ware_num;
stocklev->s_D_ID = dist_num;

write(fdo, clear_stocklev_buf, clear_stocklev_size);
}

/*---------------------------------------------------------------*/
/*              read new order screen                            */
/*---------------------------------------------------------------*/
static int read_neword (neword)
        struct neword_struct    *neword;
{
struct items_struct    temp[15];
static int        temp_cnt, i, row, key, num;

temp_cnt = 0;
neword->s_C_ID = 0;
neword->s_D_ID = 0;
neword->s_total_amount = 0;
for (i=0; i < 15; i++) {
    temp[i].s_OL_SUPPLY_W_ID = 0;
        temp[i].s_OL_I_ID = 0;
        temp[i].s_OL_QUANTITY = 0;
    neword->item[i].s_OL_SUPPLY_W_ID = 0;
        neword->item[i].s_OL_I_ID = 0;
        neword->item[i].s_OL_QUANTITY = 0;
   }

NOF1:
   key = getintfld(neword->s_D_ID, &num, 2, 4, 29); /* district number */
   neword->s_D_ID = num;
   if (key == CNTRL_C_KEY) goto read_neword_exit;
   if (key == BACK_TAB_KEY) {
    i = 14;
    row = 23;
    goto NOF5;
    }
NOF2:
   key = getintfld(neword->s_C_ID, &num, 4, 5, 12); /* customer number */
   neword->s_C_ID = num;
   if (key == CNTRL_C_KEY) goto read_neword_exit;
   if (key == BACK_TAB_KEY) goto NOF1;

   i = 0;
   row = 9;
NOF3:
        key = getintfld(temp[i].s_OL_SUPPLY_W_ID,&num,4,row,3);
        temp[i].s_OL_SUPPLY_W_ID = num;
        if (key == CNTRL_C_KEY) goto read_neword_exit;
        if (key == BACK_TAB_KEY) {
        if (i == 0) goto NOF2;
        else {
            --i;
            --row;
            goto NOF5;
            }
        }
NOF4:
        key = getintfld(temp[i].s_OL_I_ID,&num,6,row,10);
        temp[i].s_OL_I_ID = num;
        if (key == CNTRL_C_KEY) goto read_neword_exit;
        if (key == BACK_TAB_KEY) goto NOF3;
NOF5:
        key = getintfld(temp[i].s_OL_QUANTITY,&num,2,row,45);
        temp[i].s_OL_QUANTITY = num;
        if (key == BACK_TAB_KEY) goto NOF4;
    if (key == ENTER_KEY || key == CNTRL_C_KEY) goto read_neword_exit;

   i++;
   row++;
   if (i == 15) goto NOF1;
   else goto NOF3;

read_neword_exit:
    temp_cnt = i + 1;
    if (neword->s_D_ID == 0) {
        while (neword->s_D_ID == 0) {
            errmsg("D_ID must be entered.");
            key = getintfld(neword->s_D_ID, &num, 2, 4, 29);
            neword->s_D_ID = num;
          }
        }
    if (neword->s_C_ID == 0) {
        while (neword->s_C_ID == 0) {
            errmsg("C_ID must be entered.");
            key = getintfld(neword->s_C_ID, &num, 4, 5, 12);
            neword->s_C_ID = num;
          }
        }
    neword->s_all_local = 1;
    row = 9;
    neword->s_O_OL_CNT = 0;
    for (i = 0; i < temp_cnt; i++) {
      if(!((temp[i].s_OL_SUPPLY_W_ID == 0) &&
       (temp[i].s_OL_I_ID == 0) &&
       (temp[i].s_OL_QUANTITY == 0))) {
       while (temp[i].s_OL_SUPPLY_W_ID == 0) {
         errmsg("OL_SUPPLY_W_ID must be entered.");
              key=getintfld(temp[i].s_OL_SUPPLY_W_ID,&num,4,row,3);
              temp[i].s_OL_SUPPLY_W_ID = num;
         }
       while (temp[i].s_OL_I_ID == 0) {
         errmsg("OL_I_ID must be entered.");
            key=getintfld(temp[i].s_OL_I_ID,&num,6,row,10);
            temp[i].s_OL_I_ID = num;
         }
       while (temp[i].s_OL_QUANTITY == 0) {
         errmsg("OL_QUANTITY must be entered.");
            key=getintfld(temp[i].s_OL_QUANTITY,&num,2,row,45);
            temp[i].s_OL_QUANTITY = num;
         }
           if (temp[i].s_OL_SUPPLY_W_ID != neword->s_W_ID)
         neword->s_all_local = 0;
       neword->item[neword->s_O_OL_CNT].s_OL_SUPPLY_W_ID =
                temp[i].s_OL_SUPPLY_W_ID;
       neword->item[neword->s_O_OL_CNT].s_OL_I_ID =
                temp[i].s_OL_I_ID;
       neword->item[neword->s_O_OL_CNT].s_OL_QUANTITY =
                temp[i].s_OL_QUANTITY;
       neword->s_O_OL_CNT++;
       } /* end if */
       row++;
      } /* end for */
    return(key);
}

/*---------------------------------------------------------------*/
/*              read payment screen                              */
/*---------------------------------------------------------------*/
static int read_payment (payment)
        struct payment_struct    *payment;
{
static int     key, num;
static double    fnum;

payment->s_C_ID = 0;
payment->s_D_ID = 0;
payment->s_C_D_ID = 0;
payment->s_C_W_ID = 0;
payment->s_H_AMOUNT = 0;
payment->s_C_LAST[0] = '\0';

PF1:
   key = getintfld(payment->s_D_ID, &num, 2, 6, 52); /* district number */
   payment->s_D_ID = num;
   if (key == CNTRL_C_KEY) goto read_payment_exit;
   if (key == BACK_TAB_KEY) goto PF6;

PF2:
   key = getintfld(payment->s_C_ID, &num, 4, 11, 11); /* customer number */
   payment->s_C_ID = num;
   if (key == CNTRL_C_KEY) goto read_payment_exit;
   if (key == BACK_TAB_KEY) goto PF1;

PF3:
   key = getintfld(payment->s_C_W_ID, &num, 4, 11, 33); /* customer warehouse */
   payment->s_C_W_ID = num;
   if (key == CNTRL_C_KEY) goto read_payment_exit;
   if (key == BACK_TAB_KEY) goto PF2;

PF4:
   key = getintfld(payment->s_C_D_ID, &num, 2, 11, 54); /* customer district */
   payment->s_C_D_ID = num;
   if (key == CNTRL_C_KEY) goto read_payment_exit;
   if (key == BACK_TAB_KEY) goto PF3;

PF5:
   key=getstrfld(payment->s_C_LAST,payment->s_C_LAST,16,12,29); /* last name */
   if (key == CNTRL_C_KEY) goto read_payment_exit;
   if (key == BACK_TAB_KEY) goto PF4;
```

```
PF6:
   key = getmoneyfld(payment->s_H_AMOUNT, &fnum, 8, 17, 24); /* amount */
   if (fnum > 9999.99) {
    errmsg("Amount is too large.");
    goto PF6;
    }
   payment->s_H_AMOUNT = fnum;
   if (key == CNTRL_C_KEY) goto read_payment_exit;
   if (key == TAB_KEY) goto PF1;
   if (key == BACK_TAB_KEY) goto PF5;

read_payment_exit:
    while (payment->s_D_ID == 0) {
       errmsg("D_ID must be entered.");
       key = getintfld(payment->s_D_ID, &num, 2, 6, 52);
       payment->s_D_ID = num;
       }
    while ((payment->s_C_ID == 0) &&
          (payment->s_C_LAST[0] == '\0')) {
       errmsg("C_ID or C_LAST must be entered.");
       key = getintfld(payment->s_C_ID, &num, 4, 11, 11);
       payment->s_C_ID = num;
       if (payment->s_C_ID == 0)
          key=getstrfld(payment->s_C_LAST,payment->s_C_LAST,16,12,29);
       }
    while (payment->s_C_W_ID == 0) {
       errmsg("C_W_ID must be entered.");
       key = getintfld(payment->s_C_W_ID, &num, 4, 11, 33);
       payment->s_C_W_ID = num;
       }
    while (payment->s_C_D_ID == 0) {
       errmsg("C_D_ID must be entered.");
       key = getintfld(payment->s_C_D_ID, &num, 2, 11, 54);
       payment->s_C_D_ID = num;
       }
    while (payment->s_H_AMOUNT == 0.0) {
       errmsg("H_AMOUNT must be entered.");
       key = getmoneyfld(payment->s_H_AMOUNT, &fnum, 8, 17, 24);
       if (fnum > 9999.99)
          errmsg("Amount is too large.");
       else
          payment->s_H_AMOUNT = fnum;
       }
    return(key);
}

/*---------------------------------------------------------------*/
/*              read ordstat screen                              */
/*---------------------------------------------------------------*/
static int read_ordstat (ordstat)
       struct ordstat_struct    *ordstat;
{
static int     key, num;
static double    fnum;

ordstat->s_C_ID = 0;
ordstat->s_D_ID = 0;
ordstat->s_C_LAST[0] = '\0';

OF1:
   key = getintfld(ordstat->s_D_ID, &num, 2, 4, 29); /* district number */
   ordstat->s_D_ID = num;
   if (key == CNTRL_C_KEY) goto read_ordstat_exit;
   if (key == BACK_TAB_KEY) goto OF3;

OF2:
   key = getintfld(ordstat->s_C_ID, &num, 4, 5, 11); /* customer number */
   ordstat->s_C_ID = num;
   if (key == CNTRL_C_KEY) goto read_ordstat_exit;
   if (key == BACK_TAB_KEY) goto OF1;

OF3:
   key = getstrfld(ordstat->s_C_LAST,ordstat->s_C_LAST,16,5,44); /* last name */
   if (key == CNTRL_C_KEY) goto read_ordstat_exit;
   if (key == TAB_KEY) goto OF1;
   if (key == BACK_TAB_KEY) goto OF2;

read_ordstat_exit:
    while (ordstat->s_D_ID == 0) {
       errmsg("D_ID must be entered.");
       key = getintfld(ordstat->s_D_ID, &num, 2, 4, 29);
       ordstat->s_D_ID = num;
       }
    while ((ordstat->s_C_ID == 0) &&
          (ordstat->s_C_LAST[0] == '\0')) {
       errmsg("C_ID or C_LAST must be entered.");
       key = getintfld(ordstat->s_C_ID, &num, 4, 5, 11);
       ordstat->s_C_ID = num;
       if (ordstat->s_C_ID == 0)
          key=getstrfld(ordstat->s_C_LAST,ordstat->s_C_LAST,16,5,44);
       }

#ifdef DEBUG
fprintf(stderr, "ordstat->s_C_ID: %d \n", ordstat->s_C_ID);
fprintf(stderr, "ordstat->s_C_LAST: %s \n", ordstat->s_C_LAST);
#endif

    return(key);
}

/*---------------------------------------------------------------*/
/*              read delivery screen                             */
/*---------------------------------------------------------------*/
static int read_delivery (delivery)
       struct delivery_struct    *delivery;
{
static int     key, num;

delivery->s_O_CARRIER_ID = 0;

do {
   key = getintfld(delivery->s_O_CARRIER_ID, &num, 2, 6, 17); /* Carrier Id */
   delivery->s_O_CARRIER_ID = num;
   if (key == CNTRL_C_KEY) goto read_delivery_exit;
   if (num == 0) {
    errmsg("Blank field not allowed.");
    key = TAB_KEY;
    }

   } while (key != ENTER_KEY);

read_delivery_exit:
    time((time_t *)&(delivery->s_queued_time));
    return(key);
}

/*---------------------------------------------------------------*/
/*              read stocklev screen                             */
/*---------------------------------------------------------------*/
static int read_stocklev (stocklev)
       struct stocklev_struct    *stocklev;
{
static int     key, num;

stocklev->s_threshold = 0;

do {
   key = getintfld(stocklev->s_threshold, &num, 2, 6, 24); /* threshold */
   stocklev->s_threshold = num;
   if (key == CNTRL_C_KEY) goto read_stocklev_exit;
   if (num == 0) {
    errmsg("Blank field not allowed.");
    key = TAB_KEY;
    }

   } while (key != ENTER_KEY);

read_stocklev_exit:
    return(key);
}

/*---------------------------------------------------------------*/
/*              write new order screen                           */
/*---------------------------------------------------------------*/
static write_neword (neword)
       struct neword_struct    *neword;
{
static char     buf[1500];
static int      i, pos, digits;

if (neword->s_transtatus == FATAL_SQLERROR)
    return(fatal_error());

pos = 0;

if (neword->s_transtatus != INVALID_ITEM) {
   pos += rowcol(4,61,&buf[pos]);
   pos += mystrcpy(&buf[pos], neword->s_O_ENTRY_D);
   }

pos += rowcol(5,25,&buf[pos]);
pos += mystrcpy(&buf[pos], neword->s_C_LAST);

pos += rowcol(5,52,&buf[pos]);
memcpy(&buf[pos], neword->s_C_CREDIT, 2);
pos += 2;

if (neword->s_transtatus != INVALID_ITEM) {
   pos += rowcol(5,64,&buf[pos]);
   sprintf(&buf[pos], "%1.2f", neword->s_C_DISCOUNT * 100);
   pos += strlen(&buf[pos]);
   }

pos += rowcol(6,15,&buf[pos]);
digits = myitoa(neword->s_O_ID, &buf[pos]);
pos += digits;
```

```
if (neword->s_transtatus != INVALID_ITEM) {
   pos += rowcol(6,42,&buf[pos]);
   digits = myitoa(neword->s_O_OL_CNT, &buf[pos]);
   pos += digits;
   pos += rowcol(6,59,&buf[pos]);
   sprintf(&buf[pos], "%1.2f", neword->s_W_TAX * 100);
   pos += strlen(&buf[pos]);
   pos += rowcol(6,74,&buf[pos]);
   sprintf(&buf[pos], "%1.2f", neword->s_D_TAX * 100);
   pos += strlen(&buf[pos]);

   for (i=0; i < neword->s_O_OL_CNT; i++) {
      if((neword->item[i].s_OL_SUPPLY_W_ID == 0) &&
     (neword->item[i].s_OL_I_ID == 0) &&
     (neword->item[i].s_OL_QUANTITY == 0))
     continue;

      pos += rowcol(9+i,3, &buf[pos]);
      pos += mystrcpy(&buf[pos], "____");
      pos += rowcol(9+i,3, &buf[pos]);
      digits = myitoa(neword->item[i].s_OL_SUPPLY_W_ID, &buf[pos]);
      pos += digits;
      pos += rowcol(9+i,10, &buf[pos]);
      pos += mystrcpy(&buf[pos], "______");
      pos += rowcol(9+i,10, &buf[pos]);
      digits = myitoa(neword->item[i].s_OL_I_ID, &buf[pos]);
      pos += digits;

      pos += rowcol((9+i),19,&buf[pos]);
      pos += mystrcpy(&buf[pos], neword->item[i].s_I_NAME);
      pos += rowcol(9+i,45, &buf[pos]);
      pos += mystrcpy(&buf[pos], "__");
      pos += rowcol(9+i,45, &buf[pos]);
      digits = myitoa(neword->item[i].s_OL_QUANTITY, &buf[pos]);
      pos += digits;
      pos += rowcol((9+i),50,&buf[pos]);
      digits = myitoa(neword->item[i].s_S_QUANTITY, &buf[pos]);
      pos += digits;
      pos += rowcol((9+i),58,&buf[pos]);
      buf[pos] = neword->item[i].s_brand_generic;
      pos++;
      memcpy(&buf[pos], "   ", 3);
      pos += 3;
      myftomoney(neword->item[i].s_I_PRICE, &buf[pos], 7);
      pos += 7;
      pos += rowcol((9+i),71,&buf[pos]);
      myftomoney(neword->item[i].s_OL_AMOUNT, &buf[pos], 8);
      pos += 8;
      }

   pos += rowcol(24,70,&buf[pos]);
   myftomoney(neword->s_total_amount, &buf[pos], 9);
   pos += 9;
   }
else {
/*
   for (i=0; i < neword->s_O_OL_CNT; i++) {
      pos += rowcol((9+i),3,&buf[pos]);
      memset(&buf[pos], SPACE, 50);
      pos += 50;
      sprintf(&buf[pos], "\n");
      pos += strlen(&buf[pos]);
      }
*/
   pos += rowcol(24,19,&buf[pos]);
   pos += mystrcpy(&buf[pos], "Item number is not valid");
   }

pos += rowcol(2,1,&buf[pos]);

buf[pos] = '\0';
write(fdo, buf, pos);

#ifdef DEBUG
fprintf(stderr, "write_neword_size: %d \n", pos);
#endif
}

/*---------------------------------------------------------------------*/
/*              write payment screen                                   */
/*---------------------------------------------------------------------*/
static int write_payment (payment)
        struct payment_struct    *payment;
{
static char     buf[800];
static int      cd_size, i, pos, size;

if (payment->s_transtatus == FATAL_SQLERROR)
    return(fatal_error());

pos = 0;
pos += rowcol(4,7,&buf[pos]);
pos += mystrcpy(&buf[pos],payment->s_H_DATE);
pos += rowcol(7,1,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_W_STREET_1);
pos += rowcol(7,42,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_D_STREET_1);
pos += rowcol(8,1,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_W_STREET_2);
pos += rowcol(8,42,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_D_STREET_2);
pos += rowcol(9,1,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_W_CITY);
pos += rowcol(9,22,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_W_STATE);
pos += rowcol(9,25,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_W_ZIP);
pos += rowcol(9,42,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_D_CITY);
pos += rowcol(9,63,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_D_STATE);
pos += rowcol(9,66,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_D_ZIP);
/* pos += rowcol(11,1,&buf[pos]); */
pos += rowcol(11,11,&buf[pos]);
pos += formatitoa(payment->s_C_ID, &buf[pos], 4);
pos += rowcol(12,9,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_C_FIRST);
pos += rowcol(12,26,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_C_MIDDLE);
pos += rowcol(12,29,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_C_LAST);
pos += rowcol(12,58,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_C_SINCE);
pos += rowcol(13,9,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_C_STREET_1);
pos += rowcol(13,58,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_C_CREDIT);
pos += rowcol(14,9,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_C_STREET_2);
pos += rowcol(14,58,&buf[pos]);
sprintf(&buf[pos], "%1.2f", payment->s_C_DISCOUNT * 100);
/* sprintf(&buf[pos], "%1.2f", payment->s_C_DISCOUNT); */
pos += strlen(&buf[pos]);
pos += rowcol(15,9,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_C_CITY);
pos += rowcol(15,30,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_C_STATE);
pos += rowcol(15,33,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_C_ZIP);
pos += rowcol(15,58,&buf[pos]);
pos += mystrcpy(&buf[pos], payment->s_C_PHONE);
/* pos += rowcol(17,57,&buf[pos]); */
pos += rowcol(17,56,&buf[pos]);
/* myftomoney(payment->s_C_BALANCE, &buf[pos], 10); */
/* pos += 10; */
myftomoney(payment->s_C_BALANCE, &buf[pos], 13);
pos += 13;
/* pos += rowcol(18,17,&buf[pos]); */
pos += rowcol(18,18,&buf[pos]);
/* myftomoney(payment->s_C_CREDIT_LIM, &buf[pos], 9); */
/* pos += 9; */
myftomoney(payment->s_C_CREDIT_LIM, &buf[pos], 14);
pos += 14;

if (payment->s_C_CREDIT[0] == 'B' &&
    payment->s_C_CREDIT[1] == 'C') {
    pos += rowcol(20,12,&buf[pos]);
    cd_size = strlen(payment->s_C_DATA);
    for (i=0; i < 4; i++) {
      if (cd_size > 50) {
          memcpy(&buf[pos], &payment->s_C_DATA[i*50], 50);
          pos += 50;
       cd_size -= 50;
       }
       else {
       strcpy(&buf[pos], &payment->s_C_DATA[i*50]);
       pos += strlen(&payment->s_C_DATA[i*50]);
       break;
       }
      if (i < 3) pos += rowcol((21+i),12,&buf[pos]);
      }
   }

pos += rowcol(2,1,&buf[pos]);

buf[pos] = '\0';
write(fdo, buf, pos);

#ifdef DEBUG
fprintf(stderr, "write_payment_size: %d \n", pos);
#endif
}
```

```
/*-------------------------------------------------------------*/
/*              write ordstat screen                           */
/*-------------------------------------------------------------*/
static int write_ordstat (ordstat)
        struct ordstat_struct    *ordstat;
{
static char     buf[1000];
static int      i, pos;

if (ordstat->s_transtatus == FATAL_SQLERROR)
    return(fatal_error());

pos = 0;

pos += rowcol(5,11,&buf[pos]);
pos += formatitoa(ordstat->s_C_ID, &buf[pos], 4);
pos += rowcol(5,24,&buf[pos]);
pos += mystrcpy(&buf[pos], ordstat->s_C_FIRST);
pos += rowcol(5,41,&buf[pos]);
pos += mystrcpy(&buf[pos], ordstat->s_C_MIDDLE);
pos += rowcol(5,44,&buf[pos]);
pos += mystrcpy(&buf[pos], ordstat->s_C_LAST);
pos += rowcol(6,16,&buf[pos]);
/* myftomoney(ordstat->s_C_BALANCE, &buf[pos], 10); */
/* pos += 10; */
myftomoney(ordstat->s_C_BALANCE, &buf[pos], 13);
pos += 13;
pos += rowcol(8,15,&buf[pos]);
pos += myitoa(ordstat->s_O_ID, &buf[pos]);
pos += rowcol(8,38,&buf[pos]);
pos += mystrcpy(&buf[pos], ordstat->s_O_ENTRY_D);
pos += rowcol(8,76,&buf[pos]);
pos += myitoa(ordstat->s_O_CARRIER_ID, &buf[pos]);

for (i=0; i < ordstat->s_ol_cnt; i++) {
    buf[pos++] = '\n';
    pos += rowcol((i+10),3,&buf[pos]);
    pos += myitoa(ordstat->item[i].s_OL_SUPPLY_W_ID, &buf[pos]);
    pos += rowcol((i+10),15,&buf[pos]);
    pos += myitoa(ordstat->item[i].s_OL_I_ID, &buf[pos]);
    pos += rowcol((i+10),25,&buf[pos]);
    pos += myitoa(ordstat->item[i].s_OL_QUANTITY, &buf[pos]);
    pos += rowcol((i+10),33,&buf[pos]);
    myftomoney(ordstat->item[i].s_OL_AMOUNT, &buf[pos], 9);
    pos += 9;
    pos += rowcol((i+10),47,&buf[pos]);
    pos += mystrcpy(&buf[pos],ordstat->item[i].s_OL_DELIVERY_D);
    }

pos += rowcol(2,1,&buf[pos]);

buf[pos] = '\0';
write(fdo, buf, pos);

#ifdef DEBUG
fprintf(stderr, "write_ordstat_size: %d \n", pos);
#endif
}

/*-------------------------------------------------------------*/
/*              write delivery screen                          */
/*-------------------------------------------------------------*/
static int write_delivery (delivery)
        struct delivery_struct    *delivery;
{
static char     buf[100];
static int      pos;

if (delivery->s_transtatus == FATAL_SQLERROR)
    return(fatal_error());

pos = 0;

pos += rowcol(8,19,&buf[pos]);
pos += mystrcpy(&buf[pos], delivery->s_exec_status);
pos += rowcol(2,1,&buf[pos]);

buf[pos] = '\0';
write(fdo, buf, pos);

#ifdef DEBUG
fprintf(stderr, "write_delivery_size: %d \n", pos);
#endif
}

/*-------------------------------------------------------------*/
/*              write stocklev screen                          */
/*-------------------------------------------------------------*/
static write_stocklev (stocklev)
        struct stocklev_struct    *stocklev;
{
static char     buf[100];
static int      pos;

if (stocklev->s_transtatus == FATAL_SQLERROR)
    return(fatal_error());

pos = 0;

pos += rowcol(8,12,&buf[pos]);
pos += myitoa(stocklev->s_low_stock, &buf[pos]);
pos += rowcol(2,1,&buf[pos]);

buf[pos] = '\0';
write(fdo, buf, pos);

#ifdef DEBUG
fprintf(stderr, "write_stocklev_size: %d \n", pos);
#endif
}

/*-------------------------------------------------------------*/
/*              fatal error message                            */
/*-------------------------------------------------------------*/
fatal_error ()
{
char     buf[50];
char    *ptr;
int      pos;

ptr = buf;
ptr += rowcol(24,19,ptr);
ptr += mystrcpy(ptr, "SQL Fatal Error.");
/* For ACID test purposes, the following should be 10,10 */
/* else make it 2,1 */
/* Well, I don't know if this is  supposed to be here but.... CCC */
#ifdef ACID_TEST
#ifdef NO_CCCHACK
   rowcol(10,10,&buf[pos]);
   pos = strlen(buf);
   buf[pos] = '\0';
#else
   ptr += rowcol(10,10,ptr);
   *ptr = '\0';
#endif
#else
#ifdef NO_CCCHACK
   rowcol(2,1,&buf[pos]);
   pos = strlen(buf);
   buf[pos] = '\0';
#else
   ptr += rowcol(2,1,ptr);
   *ptr = '\0';
#endif
#endif

write(fdo, buf, strlen(buf));
}

/*-------------------------------------------------------------*/
/*              get integer field                              */
/*-------------------------------------------------------------*/
static  int getintfld(inum, onum, size, row, col)
    int             inum, *onum, size, row, col;
{
    static char     str[20], buf[20];
    static int      strlength, status, j, i, key, num;

    for (i = myitoa(inum, buf); i < size; i++)
    buf[i] = UNDERL;
    buf[i] = '\0';

    strlength = rowcol(row, col, str);
    strcpy(&str[strlength], buf);
    write(fdo, str, (strlength + size));

    do {
    key = get_term_value(buf, size, row, col);
    if (key == CNTRL_C_KEY)
        return (CNTRL_C_KEY);

    for (status = 0, i = 0, j = 0; i < size; i++) {
        if ((buf[i] >= '0' && buf[i] <= '9') && status == 0)
        str[j++] = buf[i];
        else if (buf[i] == SPACE || buf[i] == UNDERL) {
        if (j == 0)
            continue;
        else
            status = 1;
        } else {
        errmsg("An invalid character was detected.");
        status = -1;
        break;
        }
    }
```

```
    } while (status == -1);

    strlength = j;
    str[strlength] = '\0';
    *onum = atoi(str);

    for (i = (size - 1) - strlength; i > -1; --i)
    buf[i] = UNDERL;
    for (j = 0, i = size - strlength; i < size; i++, j++)
    buf[i] = str[j];

    strlength = rowcol(row, col, str);
    strcpy(&str[strlength], buf);
    write(fdo, str, (strlength + size));

    return (key);
}

/*-----------------------------------------------------------------*/
/*              get money field                                    */
/*-----------------------------------------------------------------*/
static
getmoneyfld(inum, onum, size, row, col)
    double          inum, *onum;
    int             size, row, col;
{
    static char     str[20], buf[20];
    static int      strlength, pointstatus, status, j, i, key, num;
    double          myatof();

    sprintf(buf, "%1.2f", inum);
    i = strlen(buf);

    for (; i < size; i++)
    buf[i] = UNDERL;
    buf[i] = '\0';

    strlength = rowcol(row, col, str);
    strcpy(&str[strlength], buf);
    write(fdo, str, (strlength + size));

    do {
    key = get_term_value(buf, size, row, col);
    if (key == CNTRL_C_KEY)
        return (CNTRL_C_KEY);

    for (pointstatus = 0, status = 0, i = 0, j = 0; i < size; i++) {
        if ((buf[i] >= '0' && buf[i] <= '9') && status == 0)
        str[j++] = buf[i];
        else if (buf[i] == UNDERL || buf[i] == SPACE) {
        if (j == 0)
            continue;
        else
            status = 1;
        } else if (buf[i] == '.' && status == 0 && pointstatus == 0) {
        str[j++] = buf[i];
        pointstatus = 1;
        } else if (buf[i] == '$' && j == 0) {
        str[j++] = buf[i];
        } else {
        errmsg("An invalid format was detected.");
        status = -1;
        break;
        }
    }

    } while (status == -1);

    strlength = j;
    str[strlength] = '\0';
    if (str[0] == '$')
    *onum = (double) myatof(&str[1]);
    else
    *onum = (double) myatof(str);

    myftomoney(*onum, buf, size);
    buf[size] = '\0';

    strlength = rowcol(row, col, str);
    strcpy(&str[strlength], buf);
    write(fdo, str, (strlength + size));

    return (key);
}

/*-----------------------------------------------------------------*/
/*              get string field                                   */
/*-----------------------------------------------------------------*/
static int getstrfld (istr, ostr, size, row, col)
    char    *istr, *ostr;
        int     size, row, col;
{
static char    str[80];
static int     i, key, strlength, istrlength;

strlength = rowcol(row,col,str);
istrlength = mystrcpy(&str[strlength],istr);
write(fdo,str, (strlength+istrlength));

strcpy(ostr, istr);
for (i=istrlength; i < size; i++) ostr[i] = UNDERL;

key = get_term_value(ostr, size, row, col);

for (i=size-1;; i--) {
    if (i == -1) {
        ostr[0] = '\0';
        break;
        }
    if (ostr[i] != UNDERL && ostr[i] != SPACE) {
        ostr[i+1] = '\0';
        break;
        }
    }

return(key);
}

/*-----------------------------------------------------------------*/
/*              get term value                                     */
/*-----------------------------------------------------------------*/
static int
get_term_value(buf, size, row, col)
    char            *buf;
    int             size, row, col;
{
    static int      last_cirbuf_pos = 0, cirbuf_pos = 0, readchars = 0;
    static int      buf_pos, curstr_pos, sizeplus;
    static char     cirbuf[CIRBUF_SIZE];
    static char     str[100], in_buf[100], curstr[100];
    static int      col_pos, strlength;
    int             no_chars_yet = 1;

    buf_pos = 0;
    col_pos = col;
    sizeplus = size + 1;

    strlength = rowcol(row, col_pos, str);
    write(fdo, str, strlength);

    for (;;) {
    if (readchars > 0) {
        for(curstr_pos = 0; readchars > 0; readchars--, last_cirbuf_pos++){
        if (cirbuf[last_cirbuf_pos] == TAB_KEY ||
            cirbuf[last_cirbuf_pos] == ENTER_KEY ||
            cirbuf[last_cirbuf_pos] == CNTRL_C_KEY ||
            cirbuf[last_cirbuf_pos] == BACK_TAB_KEY) {
/* ANGELAHACK */
/*          write(fdo, curstr, curstr_pos); */
            ++last_cirbuf_pos;
            --readchars;
            return ((int) cirbuf[last_cirbuf_pos - 1]);
        } else if (cirbuf[last_cirbuf_pos] == BACKSPACE_KEY) {
            if (col == col_pos) {
            write(fdo, BEEP_SOUND, strlen(BEEP_SOUND));
            } else {
            --col_pos;
            --buf_pos;
            buf[buf_pos] = UNDERL;
            curstr[curstr_pos] = '\0';
            strcpy(str, curstr);
            rowcol(row, col_pos, &str[strlen(str)]);
            str[strlen(str)] = UNDERL;
            rowcol(row, col_pos, &str[strlen(str)]);
            write(fdo, str, strlen(str));
            curstr_pos = 0;
            }
        } else if (((cirbuf[last_cirbuf_pos] >= 'A' &&
                    cirbuf[last_cirbuf_pos] <= 'Z') ||
                (cirbuf[last_cirbuf_pos] >= 'a' &&
                    cirbuf[last_cirbuf_pos] <= 'z') ||
                (cirbuf[last_cirbuf_pos] >= '0' &&
                    cirbuf[last_cirbuf_pos] <= '9') ||
                (cirbuf[last_cirbuf_pos] == ' ') ||
                (cirbuf[last_cirbuf_pos] == '.')) &&
              (buf_pos < size)) {
/* CCCHACK */
/*      } else if (buf_pos < size) { */
            if (no_chars_yet) {
            int i;
            no_chars_yet = 0;
            buf[0] = cirbuf[last_cirbuf_pos];
            for (i=0; i < size; i++)
                buf[i] = UNDERL;
            write(fdo, buf, size);
            strlength = rowcol(row, col_pos, str);
```

```
            write(fdo, str, strlength);
            curstr_pos = 0;
            buf_pos = 0;
            }
            curstr[curstr_pos] = cirbuf[last_cirbuf_pos];
            ++curstr_pos;
            buf[buf_pos] = cirbuf[last_cirbuf_pos];
            ++buf_pos;
            ++col_pos;
        } else
            write(fdo, BEEP_SOUND, strlen(BEEP_SOUND));
        }

        write(fdo, curstr, curstr_pos);
    }
    /*--------------------------*/
    /* read screen              */
    /*--------------------------*/
    readchars = read(fdi, in_buf, SCR_READ_NUM);

    if ((cirbuf_pos + readchars + 1) > CIRBUF_SIZE) {
        cirbuf_pos = 0;
        last_cirbuf_pos = 0;
    }
    memcpy(&cirbuf[cirbuf_pos], in_buf, readchars);
    cirbuf_pos += readchars;
    }
}

/*-------------------------------------------------------------*/
/*              myitoa - integer to ascii                      */
/*-------------------------------------------------------------*/
static int myitoa (num, buf)
        int num;
        char *buf;
{
static int i,j,sign;
static char temp[20];

if (( sign=num) < 0 ) num *= -1;

i = 0;
do      {
        temp[i++] = num % 10 + '0';
        } while ( (num /= 10 ) > 0);

if ( sign < 0 ) temp[i++]= '-';

for (--i, j=0; i >= 0; i--, j++)
        buf[j] = temp[i];

return(j);
}

/*-------------------------------------------------------------*/
/*              formatitoa - format integer to ascii           */
/*-------------------------------------------------------------*/
static formatitoa (num, buf, size)
        int num;
        char *buf;
    int size;
{
static int     i, j, len;
static char    str[20];

len = myitoa(num,buf);

if (len == size) return(size);

i = len;
j = size;
do { buf[--j] = buf[--i];
   } while(i > 0);

do { buf[--j] = UNDERL;
   } while(j > 0);

return(size);
}

/*-------------------------------------------------------------*/
/*              myatof - ascii to float                        */
/*-------------------------------------------------------------*/
static double myatof (buf)
        char *buf;
{
static double    num;
static int    i, j;

for (num=0.0, i=strlen(buf)-1, j=1; i >= 0; i--) {
    if (buf[i] == '.') {
        num /= j;
        j = 1;
        }
    else if (buf[i] == '-') {
        num *= -1.0;
        }
    else {
        num += ((buf[i] - '0') * j);
        j *= 10;
        }
    }

return(num);
}

/*-------------------------------------------------------------*/
/*              mymoneytoa - money to ascii                    */
/*-------------------------------------------------------------*/
static int mymoneytoa (num, buf)
        double num;
        char *buf;
{
static int    i, intnum, dec, strlength;
static double    fdec;

i=0;
if (num < 0.0) {
    buf[i++] = '-';
    num *= -1.0;
    }

intnum = (int)num;
fdec = num - (double)intnum;
dec = (int)((fdec*100.0)+0.5);

strlength = myitoa(intnum, &buf[i]);
if (dec > 0) {
    buf[strlength++] = '.';
    strlength += myitoa(dec, &buf[strlength]);
    }

return(strlength);
}

/*-------------------------------------------------------------*/
/*              myftomoney - float to formatted money          */
/*-------------------------------------------------------------*/
static int myftomoney (num, buf, space)
        double num;
        char *buf;
    int space;
{
static char    str[20];
static int    i;

sprintf(str, "$%1.2f", num);
i = strlen(str);

for (i--,space--; i >= 0 && space >= 0; i--, space--)
    buf[space] = str[i];
for (; space >= 0; space--)
    buf[space] = SPACE;
}

/*-------------------------------------------------------------*/
/*              mystrcpy - string copy returning str length    */
/*-------------------------------------------------------------*/
static int mystrcpy (str1, str2)
    char    *str1, *str2;
{
static int    i;

for (i=0; *str2 != '\0'; i++) {
    *str1 = *str2;
    str1++;
    str2++;
    }

return(i);
}

/*-------------------------------------------------------------*/
/*  error message                                              */
/*-------------------------------------------------------------*/
errmsg (msg)
    char    *msg;
{
static char    str[100];

rowcol(ERRMSG_ROW,1,str);
strcat(str,msg);
write(fdo,str,strlen(str));
sleep(3);
rowcol(ERRMSG_ROW,1,str);
#ifdef WYSE50_TERM
```

```
strcat(str,"\033T");
#else
strcat(str,"                                    ");
#endif
write(fdo,str,strlen(str));
}

/*-------------------------------------------------------------*/
/*  end screen                                                 */
/*-------------------------------------------------------------*/
end_screen ()
{
ioctl(fdo, TCSETAW, &savetty);
ioctl(fdi, TCSETAW, &savetty);

ioctl(fdo, TCFLSH, 2 );
ioctl(fdi, TCFLSH, 2 );
}



/*************************************************************************/
/*                                                                       */
/*      File: delivery.sqc  : Delivery Client Code                       */
/*                                                                       */
/*************************************************************************/

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

#include "../include/tpcc.h"
#ifdef ACID_TEST
#include <time.h>
#include <sys/time.h>
#define acidtime(tvsec,tvusec) tvsec*1000+tvusec/1000
#endif

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

/*-------------------------------------------------------------*/
/*              DELIVERY Service                               */
/*-------------------------------------------------------------*/
#ifndef DARIVERSION
int delivery_sql (struct delivery_struct *delivery)
#else
#ifdef USE_SQLEPROC
int deliverys (struct sqlchar *pinput_data,
               struct sqlda   *pin_sqlda,
               struct sqlda   *pout_sqlda,
               struct sqlca   *pca )
#else
int deliverys (void *reserved1,
               void *reserved2,
               struct sqlda *pin_sqlda,
               struct sqlca *pca)
#endif
#endif
{

#ifdef ACID_TEST
  FILE *out;
  struct timeval tv;
  struct timezone tz;
  int audittime;
  int wait = 1;
#endif

  static int tran_cntr=0, pos;
#ifdef DARIVERSION
  struct delivery_struct *delivery;
#endif

  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;

  short  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;

#ifdef DARIVERSION
  /* Retrieve the 'delivery' structure from the SQLDA */
  delivery =(struct delivery_struct *)pin_sqlda->sqlvar[0].sqldata;
#endif
#ifdef DEBUGIT
  debugit(delivery, "Server upon entry");
#endif
  strcpy(curr_tmstmp, current_tmstmp());
  w_id = delivery->s_W_ID;
  o_carrier_id = delivery->s_O_CARRIER_ID;

#ifdef ACID_TEST
  out=fopen("/tmp/tpcc.del.out","w");
  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  fprintf(out,"\n---------- START of DELIVERY PID: %d ----------\n\n",getpid());
  fprintf(out,"DELIVERY PID: %d, begin transaction time: %u \n",getpid(), audittime);
  fprintf(out, "W_ID: %d   CARRIER: %d\n", w_id, o_carrier_id);
#endif

  time(&tlong);
  pos = 0;
  ++tran_cntr;
  sprintf(&delivery->results[pos], "\n----Transaction %d started----\n", tran_cntr);
  pos += strlen(&delivery->results[pos]);
  sprintf(&delivery->results[pos], "queued-time: %d\n", delivery->s_queued_time);
  pos += strlen(&delivery->results[pos]);
  sprintf(&delivery->results[pos], "W:%4d C:%4d\n",
        w_id, o_carrier_id);
  /* pos += strlen(&delivery.results[pos]); */
  pos += 14;

 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       */
#ifdef ACID_TEST
        gettimeofday(&tv, &tz);
        audittime = acidtime(tv.tv_sec, tv.tv_usec);
        fprintf(out,"DELIVERY PID: %d, district %2d no orders: %u \n",  \
              getpid(), no_d_id, audittime);
        if (w_id == 13 && no_d_id == 7 && wait) {
          wait = 0;
          i--;
          gettimeofday(&tv, &tz);
          audittime = acidtime(tv.tv_sec, tv.tv_usec);
          fprintf(out, "DELIVERY PID: %d, sleeping: %u \n",
                getpid(), audittime);
          sleep(45);
          gettimeofday(&tv, &tz);
          audittime = acidtime(tv.tv_sec, tv.tv_usec);
          fprintf(out, "DELIVERY PID: %d, awake: %u \n",
                getpid(), audittime);
          continue;
        }
#endif
        sprintf(&delivery->results[pos],"D %2d no orders\n", no_d_id);
        /* pos += strlen(&delivery.results[pos]); */
        pos += 15;
        continue;
    }

#ifdef ACID_TEST
      gettimeofday(&tv, &tz);
      audittime = acidtime(tv.tv_sec, tv.tv_usec);
      fprintf(out, "DELIVERY PID: %d, District %2d   Order:%7d: %u\n",
```

```
            getpid(), no_d_id, no_o_id, audittime);
#endif
      sprintf(&delivery->results[pos], "D: %2d O:%7d\n", no_d_id, no_o_id);
      /* pos += strlen(&delivery.results[pos]); */
      pos += 16;

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

      /*EXEC SQL BEGIN COMPOUND NOT ATOMIC STATIC*/
      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;

#ifdef ACID_TEST
      if (i == 1 ) {
        gettimeofday(&tv, &tz);
        audittime = acidtime(tv.tv_sec, tv.tv_usec);
        fprintf(out,"DELIVERY PID: %d, before update customer time: %u \n",  \
              getpid(), audittime);
        fprintf(out, "W_ID: %d \t D_ID: %d \t C_ID: %d \t AMOUNT: %0.2f\n", \
              delivery->s_W_ID, no_d_id, o_c_id, total_amount);
      }
#endif

      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++)  */

#ifdef ACID_TEST
    if (delivery->s_O_CARRIER_ID != 7) {
      gettimeofday(&tv, &tz);
      audittime = acidtime(tv.tv_sec, tv.tv_usec);
      fprintf(out, "DELIVERY PID: %d, wait before commit time: %u \n", \
            getpid(), audittime);
      sleep(60);
                } else {
      gettimeofday(&tv, &tz);
      audittime = acidtime(tv.tv_sec, tv.tv_usec);
      fprintf(out, "DELIVERY PID: %d, wait before roll-back time: %u \n",
            getpid(), audittime);
      sleep(60);
      EXEC SQL rollback work;
      if (sqlca.sqlcode != 0) fprintf(out,"*** ROLLBACK FAILED ***\n");
      gettimeofday(&tv, &tz);
      audittime = acidtime(tv.tv_sec, tv.tv_usec);
      fprintf(out, "DELIVERY PID: %d, rolled-back time: %u \n",
            getpid(), audittime);
      fprintf(out,"\n---------- END of DELIVERY PID: %d ----------\n\n",getpid());
      fclose(out);
      goto ferror;
    }
#endif

    EXEC SQL commit;
    if (sqlca.sqlcode != 0) {
      if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
      sqlerror("COMMIT",317, &sqlca);
      goto ferror;
    }
#ifdef ACID_TEST
    gettimeofday(&tv, &tz);
    audittime = acidtime(tv.tv_sec, tv.tv_usec);
    if (delivery->s_O_CARRIER_ID != 7) {
      fprintf(out, "DELIVERY PID: %d, commit transaction time: %u \n", \
            getpid(), audittime);
    }
    fprintf(out,"\n---------- END of DELIVERY PID: %d ----------\n\n",getpid());
    fclose(out);
#endif

    time(&tlong);
    sprintf(&delivery->results[pos], "end-time: %d\n", tlong);

    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:
#ifdef DARIVERSION
#ifdef USE_SQLEPROC
    memcpy(pout_sqlda->sqlvar[0].sqldata, delivery,
          sizeof(struct delivery_struct));
#endif
    memcpy( pca, &sqlca, sizeof( struct sqlca ) );
    if ( sqlca.sqlcode < 0 )
      rc = SQLZ_DISCONNECT_PROC;
    else
      rc = SQLZ_HOLD_PROC;
#endif

#ifdef DEBUGIT
    debugit(delivery, "Server prior to return");
#endif
    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;

#ifndef DARIVERSION
    err_fp = stdout;
    fprintf(stderr,"DELIVERY: %2d sqlcode: %4d %s\n", ptat, psqlca->sqlcode, msg);
#else
    err_fp = fopen(err_fn, "a+");
#endif
    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");
#ifdef DARIVERSION
    fclose(err_fp);
#endif
}

/*----------------------------------------------------------------*/
/*       debugit                                                  */
/*----------------------------------------------------------------*/
void debugit (struct delivery_struct *delivery_ptr, char *msg)
{
    FILE *debug_fp;
    char  debug_fn[25] = "/tmp/delivery.debug.out";
    char  timeStamp[27];
    strcpy(timeStamp, current_tmstmp());
    timeStamp[19] = (char)NULL;
#ifndef DARIVERSION
    fprintf(stderr,"delivery debugit called: %s <====\n", msg);
#endif
    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);
    fprintf(debug_fp,"\ts_queued_time  = %ld (%lX)\n",
          delivery_ptr->s_queued_time,delivery_ptr->s_queued_time);
    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,"\ts_exec_status  = %s\n",
          delivery_ptr->s_exec_status);
    fprintf(debug_fp,"\tresults        = %s\n",
          delivery_ptr->results);
    fprintf(debug_fp,"}\n\n");
    fclose(debug_fp);
}

/*****************************************************************************/
/*      File: newordpay.sqc (Client code)                                    */
/*****************************************************************************/

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

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

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

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);
}

#ifdef CSTUXEDO
EXEC SQL BEGIN DECLARE SECTION;
char dbname[8] = "tpcc";
EXEC SQL END DECLARE SECTION;

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

{
    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                                     */
/*----------------------------------------------------------------*/
neword_sql (clientmsg)

    TPSVCINFO *clientmsg;      /* TUXEDO message buffer structure */
#else
    int neword_sql (struct neword_struct *neword)
#endif
{

#ifdef CSTUXEDO
    struct neword_struct *neword;
#endif

    struct sqlda in_sqlda;
#ifdef USE_SQLEPROC
    struct sqlda out_sqlda;
    struct neword_struct out_neword;
#endif

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

#ifdef CSTUXEDO
    neword = (struct neword_struct *) clientmsg->data;
#endif

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

    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->s_transtatus = FATAL_SQLERROR;

#ifndef USE_SQLEPROC
    EXEC SQL CALL "newords" USING DESCRIPTOR :in_sqlda;
#else
    /* Output SQLDA */
    out_sqlda.sqln = 1;
    out_sqlda.sqld = 1;
    out_sqlda.sqlvar[0].sqltype = SQL_TYP_CHAR;
    out_sqlda.sqlvar[0].sqldata = (char *)&out_neword;
    out_sqlda.sqlvar[0].sqllen = sizeof( struct neword_struct );
    out_sqlda.sqlvar[0].sqlind = NULL;
    out_neword.s_transtatus = FATAL_SQLERROR;
    rc = sqleproc("newords", NULL, &in_sqlda, &out_sqlda, &sqlca);
    memcpy(neword, &out_neword, sizeof( struct neword_struct));
#endif

    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");
#ifdef CSTUXEDO
    tpreturn(TPFAIL, 0, clientmsg->data, clientmsg->len, 0);
#endif
    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");
#ifdef CSTUXEDO
    tpreturn(TPFAIL, 0, clientmsg->data, clientmsg->len, 0);
#endif
    return(-1);
  }

#ifdef CSTUXEDO
  tpreturn(TPSUCCESS, 0, clientmsg->data, clientmsg->len, 0);
#else
  if (neword->s_transtatus == INVALID_ITEM) rc = -1;
#endif

  return (rc);
}

/*---------------------------------------------------------------*/
/*            PAYMENT Service                                    */
/*---------------------------------------------------------------*/
#ifdef CSTUXEDO
payment_sql (clientmsg)
    TPSVCINFO *clientmsg;   /* TUXEDO message buffer structure */
#else
    int payment_sql (struct payment_struct *payment)
#endif
{
#ifdef CSTUXEDO
  struct payment_struct *payment;
#endif

  struct sqlda in_sqlda;
#ifdef USE_SQLEPROC
  struct sqlda out_sqlda;
  struct payment_struct out_payment;
#endif

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

#ifdef CSTUXEDO
  payment = (struct payment_struct *) clientmsg->data;
#endif

  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;

#ifndef USE_SQLEPROC
  EXEC SQL CALL "pays" USING DESCRIPTOR :in_sqlda;
#else
  /* Output SQLDA */
  out_sqlda.sqln = 1;
  out_sqlda.sqld = 1;
  out_sqlda.sqlvar[0].sqltype = SQL_TYP_CHAR;
  out_sqlda.sqlvar[0].sqldata = (char *)&out_payment;
  out_sqlda.sqlvar[0].sqllen = sizeof( struct payment_struct );
  out_sqlda.sqlvar[0].sqlind = NULL;
  out_payment.s_transtatus = FATAL_SQLERROR;
  rc = sqleproc("pays", NULL, &in_sqlda, &out_sqlda, &sqlca);
  memcpy(payment, &out_payment, sizeof( struct payment_struct));
#endif

  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");
#ifdef CSTUXEDO
    tpreturn(TPFAIL, 0, clientmsg->data, clientmsg->len, 0);
#endif
    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");
#ifdef CSTUXEDO
    tpreturn(TPFAIL, 0, clientmsg->data, clientmsg->len, 0);
#endif
    return(-1);
  }

#ifdef CSTUXEDO
  tpreturn(TPSUCCESS, 0, clientmsg->data, clientmsg->len, 0);
#endif

  return (rc);
}

/*---------------------------------------------------------------*/
/*            ORDSTAT Standard Service                           */
/*---------------------------------------------------------------*/
#ifdef CSTUXEDO
ordstat_sql (clientmsg)
    TPSVCINFO *clientmsg;   /* TUXEDO message buffer structure */
#else
    int ordstat_sql (struct ordstat_struct *ordstat)
#endif
{
#ifdef CSTUXEDO
  struct ordstat_struct *ordstat;
#endif

  struct sqlda in_sqlda;
#ifdef USE_SQLEPROC
  struct sqlda out_sqlda;
  struct ordstat_struct out_ordstat;
#endif

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

#ifdef CSTUXEDO
  ordstat = (struct ordstat_struct *) clientmsg->data;
#endif

  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;

#ifndef USE_SQLEPROC
  EXEC SQL CALL "ords" USING DESCRIPTOR :in_sqlda;
#else
  /* Output SQLDA */
  out_sqlda.sqln = 1;
  out_sqlda.sqld = 1;
  out_sqlda.sqlvar[0].sqltype = SQL_TYP_CHAR;
  out_sqlda.sqlvar[0].sqldata = (char *)&out_ordstat;
  out_sqlda.sqlvar[0].sqllen = sizeof( struct ordstat_struct );
  out_sqlda.sqlvar[0].sqlind = NULL;
  out_ordstat.s_transtatus = FATAL_SQLERROR;
  rc = sqleproc("ords", NULL, &in_sqlda, &out_sqlda, &sqlca);
  memcpy(ordstat, &out_ordstat, sizeof( struct ordstat_struct));
#endif

  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");
#ifdef CSTUXEDO
    tpreturn(TPFAIL, 0, clientmsg->data, clientmsg->len, 0);
#endif
    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");
#ifdef CSTUXEDO
    tpreturn(TPFAIL, 0, clientmsg->data, clientmsg->len, 0);
#endif
    return(-1);
  }

#ifdef CSTUXEDO
  tpreturn(TPSUCCESS, 0, clientmsg->data, clientmsg->len, 0);
#endif
  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_I
D);
    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: stklv.sqc (Client code)                                            */
/*********************************************************************************/

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

void debugit (struct stocklev_struct *, char *);

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

EXEC SQL include sqlca;

#ifdef CSTUXEDO
EXEC SQL BEGIN DECLARE SECTION;
char dbname[8] = "tpcc";
EXEC SQL END DECLARE SECTION;
/*-------------------------------------------------------------*/
/*       tpsvrinit                                             */
/*-------------------------------------------------------------*/
/* Initialize environment: open database, set transaction      */
/* characteristics, etc...                                     */
/* (This is executed first when TUXEDO boots up.)              */
/*-------------------------------------------------------------*/
int tpsvrinit (int argc, char **argv)

{
  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, "STKLV: CONNECT TO %s: SQLCODE = %d\n", dbname,
SQLCODE);
    exit(-1);
  }
}
#endif

/*-------------------------------------------------------------*/
/*            STOCKLEV Service                                 */
/*-------------------------------------------------------------*/
#ifdef CSTUXEDO
stocklev_sql (clientmsg)
    TPSVCINFO *clientmsg;   /* TUXEDO message buffer structure */
#else
    int stocklev_sql (struct stocklev_struct *stocklev)
#endif
{
#ifdef CSTUXEDO
  struct stocklev_struct *stocklev;
#endif

  struct sqlda in_sqlda;
#ifdef USE_SQLEPROC
  struct sqlda out_sqlda;
  struct stocklev_struct out_stocklev;
#endif

  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;

#ifdef CSTUXEDO
  stocklev = (struct stocklev_struct *) clientmsg->data;
#endif

  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;

#ifdef DEBUGIT
  debugit(stocklev, "Client prior to calling dari");
#endif

#ifndef USE_SQLEPROC
  EXEC SQL CALL "stklvs" USING DESCRIPTOR :in_sqlda;
#else
  /* Output SQLDA */
  out_sqlda.sqln = 1;
  out_sqlda.sqld = 1;
  out_sqlda.sqlvar[0].sqltype = SQL_TYP_CHAR;
  out_sqlda.sqlvar[0].sqldata = (char *)&out_stocklev;
  out_sqlda.sqlvar[0].sqllen = sizeof( struct stocklev_struct );
  out_sqlda.sqlvar[0].sqlind = NULL;
  out_stocklev.s_transtatus = FATAL_SQLERROR;
  rc = sqleproc("stklvs", NULL, &in_sqlda, &out_sqlda, &sqlca);
  memcpy(stocklev, &out_stocklev, sizeof( struct stocklev_struct));
#endif

#ifdef DEBUGIT
  debugit(stocklev, "Client upon return from dari");
#endif

  if (sqlca.sqlcode) {
    rc = sqlaintp(buf, 512, 78, &sqlca);
    fprintf(stderr,"\nSTOCKLEV dari failed with sqlcode = %ld\n%s\n",sqlca.sqlcode,
buf);
    debugit(stocklev, "sqlcode != 0");
#ifdef CSTUXEDO
    tpreturn(TPFAIL, 0, clientmsg->data, clientmsg->len, 0);
#endif
    return(-1);
  }

  if (stocklev->s_transtatus <= FATAL_SQLERROR) {
    fprintf(stderr,"STOCKLEV transaction failed, transtatus = %d.\n",
          stocklev->s_transtatus);
    debugit(stocklev, "s_transtatus <= FATAL_SQLERROR");
#ifdef CSTUXEDO
    tpreturn(TPFAIL, 0, clientmsg->data, clientmsg->len, 0);
#endif
    return(-1);
  }

#ifdef CSTUXEDO
  tpreturn(TPSUCCESS, 0, clientmsg->data, clientmsg->len, 0);
#endif

  return (0);
}

/*-------------------------------------------------------------*/
/*       debugit                                               */
/*-------------------------------------------------------------*/
void debugit (struct stocklev_struct *stocklev_ptr, char *msg)
{
  FILE *debug_fp;
  char  debug_fn[25] = "/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);
}
```

A.2 Server code

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

#include "../include/tpcc.h"
#ifdef ACID_TEST
#include <time.h>
#include <sys/time.h>
#define acidtime(tvsec,tvusec) tvsec*1000+tvusec/1000
#endif

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

/*-------------------------------------------------------------------*/
/*              DELIVERY Service                                     */
/*-------------------------------------------------------------------*/
#ifndef DARIVERSION
int delivery_sql (struct delivery_struct *delivery)
#else
#ifdef USE_SQLEPROC
int deliverys (struct sqlchar *pinput_data,
               struct sqlda   *pin_sqlda,
               struct sqlda   *pout_sqlda,
               struct sqlca   *pca )
#else
int deliverys (void *reserved1,
               void *reserved2,
               struct sqlda *pin_sqlda,
               struct sqlca *pca)
#endif
#endif
{

#ifdef ACID_TEST
   FILE *out;
   struct timeval tv;
   struct timezone tz;
   int audittime;
   int wait = 1;
#endif

   static int tran_cntr=0, pos;
#ifdef DARIVERSION
   struct delivery_struct *delivery;
#endif

   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;

   short  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;

#ifdef DARIVERSION
   /* Retrieve the 'delivery' structure from the SQLDA */
   delivery =(struct delivery_struct *)pin_sqlda->sqlvar[0].sqldata;
#endif
#ifdef DEBUGIT
   debugit(delivery, "Server upon entry");
#endif
   strcpy(curr_tmstmp, current_tmstmp());
   w_id = delivery->s_W_ID;
   o_carrier_id = delivery->s_O_CARRIER_ID;

#ifdef ACID_TEST
   out=fopen("/tmp/tpcc.del.out","w");
   gettimeofday(&tv, &tz);
   audittime = acidtime(tv.tv_sec, tv.tv_usec);
   fprintf(out,"\n---------- START of DELIVERY PID: %d ----------\n\n",getpid());
   fprintf(out,"DELIVERY PID: %d, begin transaction time: %u \n",getpid(), audittime);
   fprintf(out, "W_ID: %d   CARRIER: %d\n", w_id, o_carrier_id);
#endif

   time(&tlong);
   pos = 0;
   ++tran_cntr;
   sprintf(&delivery->results[pos], "\n----Transaction %d started----\n", tran_cntr);
   pos += strlen(&delivery->results[pos]);
   sprintf(&delivery->results[pos], "queued-time: %d\n", delivery->s_queued_time);
   pos += strlen(&delivery->results[pos]);
   sprintf(&delivery->results[pos], "W:%4d C:%4d\n",
           w_id, o_carrier_id);
   /* pos += strlen(&delivery.results[pos]); */
   pos += 14;

 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       */
#ifdef ACID_TEST
        gettimeofday(&tv, &tz);
        audittime = acidtime(tv.tv_sec, tv.tv_usec);
        fprintf(out,"DELIVERY PID: %d, district %2d no orders: %u \n",  \
                getpid(), no_d_id, audittime);
        if (w_id == 13 && no_d_id == 7 && wait) {
          wait = 0;
          i--;
          gettimeofday(&tv, &tz);
          audittime = acidtime(tv.tv_sec, tv.tv_usec);
          fprintf(out, "DELIVERY PID: %d, sleeping: %u \n",
                  getpid(), audittime);
          sleep(45);
          gettimeofday(&tv, &tz);
          audittime = acidtime(tv.tv_sec, tv.tv_usec);
          fprintf(out, "DELIVERY PID: %d, awake: %u \n",
                  getpid(), audittime);
          continue;
        }
#endif
        sprintf(&delivery->results[pos],"D %2d no orders\n", no_d_id);
        /* pos += strlen(&delivery.results[pos]); */
        pos += 15;
        continue;
```

```
    }

#ifdef ACID_TEST
    gettimeofday(&tv, &tz);
    audittime = acidtime(tv.tv_sec, tv.tv_usec);
    fprintf(out, "DELIVERY PID: %d, District %2d   Order:%7d: %u\n",
          getpid(), no_d_id, no_o_id, audittime);
#endif
    sprintf(&delivery->results[pos], "D: %2d O:%7d\n", no_d_id, no_o_id);
    /* pos += strlen(&delivery.results[pos]); */
    pos += 16;

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

    /*EXEC SQL BEGIN COMPOUND NOT ATOMIC STATIC*/
    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;

#ifdef ACID_TEST
    if (i == 1 ) {
      gettimeofday(&tv, &tz);
      audittime = acidtime(tv.tv_sec, tv.tv_usec);
      fprintf(out,"DELIVERY PID: %d, before update customer time: %u \n",  \
            getpid(), audittime);
      fprintf(out, "W_ID: %d \t D_ID: %d \t C_ID: %d \t AMOUNT: %0.2f\n", \
            delivery->s_W_ID, no_d_id, o_c_id, total_amount);
    }
#endif

    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++)  */

#ifdef ACID_TEST
  if (delivery->s_O_CARRIER_ID != 7) {
    gettimeofday(&tv, &tz);
    audittime = acidtime(tv.tv_sec, tv.tv_usec);
    fprintf(out, "DELIVERY PID: %d, wait before commit time: %u \n", \
          getpid(), audittime);
    sleep(60);
                } else {
    gettimeofday(&tv, &tz);
    audittime = acidtime(tv.tv_sec, tv.tv_usec);
    fprintf(out, "DELIVERY PID: %d, wait before roll-back time: %u \n",
          getpid(), audittime);
    sleep(60);
    EXEC SQL rollback work;
    if (sqlca.sqlcode != 0) fprintf(out,"*** ROLLBACK FAILED ***\n");
    gettimeofday(&tv, &tz);
    audittime = acidtime(tv.tv_sec, tv.tv_usec);
    fprintf(out, "DELIVERY PID: %d, rolled-back time: %u \n",
          getpid(), audittime);
    fprintf(out,"\n---------- END of DELIVERY PID: %d ----------\n\n",getpid());
    fclose(out);
    goto ferror;
  }
#endif

  EXEC SQL commit;
  if (sqlca.sqlcode != 0) {
    if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
    sqlerror("COMMIT",317, &sqlca);
    goto ferror;
  }
#ifdef ACID_TEST
  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  if (delivery->s_O_CARRIER_ID != 7) {
    fprintf(out, "DELIVERY PID: %d, commit transaction time: %u \n", \
          getpid(), audittime);
  }
  fprintf(out,"\n---------- END of DELIVERY PID: %d ----------\n\n",getpid());
  fclose(out);
#endif

  time(&tlong);
  sprintf(&delivery->results[pos], "end-time: %d\n", tlong);

  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:
#ifdef DARIVERSION
#ifdef USE_SQLEPROC
  memcpy(pout_sqlda->sqlvar[0].sqldata, delivery,
        sizeof(struct delivery_struct));
#endif
  memcpy( pca, &sqlca, sizeof( struct sqlca ) );
  if ( sqlca.sqlcode < 0 )
    rc = SQLZ_DISCONNECT_PROC;
  else
    rc = SQLZ_HOLD_PROC;
#endif

#ifdef DEBUGIT
  debugit(delivery, "Server prior to return");
#endif
  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;

#ifndef DARIVERSION
  err_fp = stdout;
  fprintf(stderr,"DELIVERY: %2d sqlcode: %4d %s\n", ptat, psqlca->sqlcode, msg);
#else
  err_fp = fopen(err_fn, "a+");
```

```
#endif
  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");
#ifdef DARIVERSION
  fclose(err_fp);
#endif
}

/*---------------------------------------------------------------*/
/*       debugit                                                 */
/*---------------------------------------------------------------*/
void debugit (struct delivery_struct *delivery_ptr, char *msg)
{
  FILE *debug_fp;
  char  debug_fn[25] = "/tmp/delivery.debug.out";
  char  timeStamp[27];
  strcpy(timeStamp, current_tmstmp());
  timeStamp[19] = (char)NULL;
#ifndef DARIVERSION
  fprintf(stderr,"delivery debugit called: %s <====\n", msg);
#endif
  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);
  fprintf(debug_fp,"\ts_queued_time  = %ld (%lX)\n",
        delivery_ptr->s_queued_time,delivery_ptr->s_queued_time);
  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,"\ts_exec_status  = %s\n",
        delivery_ptr->s_exec_status);
  fprintf(debug_fp,"\tresults        = %s\n",
        delivery_ptr->results);
  fprintf(debug_fp,"}\n\n");
  fclose(debug_fp);
}

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

#include "../include/tpcc.h"
#ifdef ACID_TEST
#include <time.h>
#include <sys/time.h>
#define acidtime(tvsec,tvusec) tvsec*1000+tvusec/1000
#endif

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

/*---------------------------------------------------------------*/
/*             ORDSTAT Standard Service                          */
/*---------------------------------------------------------------*/
#ifndef DARIVERSION
int ordstat_sql (struct ordstat_struct *ordstat)
#else
#ifdef USE_SQLEPROC
int ords (struct sqlchar *pinput_data,
        struct sqlda   *pin_sqlda,
        struct sqlda   *pout_sqlda,
        struct sqlca   *pca )
#else
int ords (void *reserved1,
        void *reserved2,
        struct sqlda *pin_sqlda,
        struct sqlca *pca)
#endif
#endif
{
#define  MAXCLAST   3000

#ifdef ACID_TEST
  FILE *out;
  struct timeval tv;
  struct timezone tz;
  int audittime;
  int re_read_order_cur = 1;
#endif

#ifdef DARIVERSION
  struct ordstat_struct *ordstat;
#endif
  EXEC SQL include sqlca;

  EXEC SQL BEGIN DECLARE SECTION;
  short  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;
  EXEC SQL END DECLARE SECTION;

  double  ol_amount;
  int     i, rc;
  int     retry_count = -1;
  long    c_id[MAXCLAST];

  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;

#ifdef DARIVERSION
  ordstat =(struct ordstat_struct *)pin_sqlda->sqlvar[0].sqldata;
#endif
#ifdef DEBUGIT
  debugit(ordstat, "Server upon entry");
#endif
#ifdef ACID_TEST
  out=fopen("/tmp/tpcc.ord.out","a+");
  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  fprintf(out,"\n---------- START of ORDSTAT PID: %d ----------\n\n",getpid());
  fprintf(out, "ORDSTAT PID: %d, begin transaction time: %u \n",
        getpid(), audittime);
  fprintf(out, "W_ID: %d \t D_ID: %d \t C_ID: %d \n",
        ordstat->s_W_ID, ordstat->s_D_ID, ordstat->s_C_ID);
#endif

  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;
  }

  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
                */
  /*-------------------------------------------------------*/

#ifdef ACID_TEST
 reopen:
  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  fprintf(out, "ORDSTAT PID: %d, before order read time: %u \n",
          getpid(), audittime);
#endif

  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;
  }

#ifdef ACID_TEST
  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  fprintf(out, "ORDSTAT PID: %d, after order read o_id=%d d_id=%d w_id=%d
c_id=%d: %u \n",
          getpid(), o_id, d_id, w_id, tmp_c_id, audittime);
  if (tmp_c_id == 1313 && re_read_order_cur) {
    re_read_order_cur = 0;
    EXEC SQL CLOSE read_order_cur;
    if (sqlca.sqlcode != 0) {
      if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
      sqlerror("CLOSE CURSOR read_order_cur",230, &sqlca);
      goto ferror;
    }
    gettimeofday(&tv, &tz);
    audittime = acidtime(tv.tv_sec, tv.tv_usec);
    fprintf(out, "ORDSTAT PID: %d, sleeping: %u \n",
          getpid(), audittime);
    sleep(45);
    gettimeofday(&tv, &tz);
    audittime = acidtime(tv.tv_sec, tv.tv_usec);
    fprintf(out, "ORDSTAT PID: %d, awake: %u \n",
          getpid(), audittime);
    goto reopen;
  }
#endif

  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;
  }
#ifdef ACID_TEST
  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  fprintf(out, "ORDSTAT PID: %d, commit transaction time: %u \n",
          getpid(), audittime);
  fprintf(out, "W_ID: %d \t D_ID: %d \t O_ID: %d\n",
          ordstat->s_W_ID, ordstat->s_D_ID, ordstat->s_O_ID);
  fprintf(out,"\n---------- END of ORDSTAT PID: %d ----------\n\n",getpid());
  fclose(out);
#endif

  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:
#ifdef DARIVERSION
#ifdef USE_SQLEPROC
  memcpy(pout_sqlda->sqlvar[0].sqldata, ordstat,
        sizeof(struct ordstat_struct));
#endif
  memcpy( pca, &sqlca, sizeof( struct sqlca ) );
  if ( sqlca.sqlcode < 0 )
    rc = SQLZ_DISCONNECT_PROC;
  else
    rc = SQLZ_HOLD_PROC;
#endif

#ifdef DEBUGIT
  debugit(ordstat, "Server prior to return");
#endif
  return(rc);
}

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

#ifndef DARIVERSION
  err_fp = stdout;
  fprintf(stderr,"ORDS: %2d sqlcode: %4d %s\n", ptat, psqlca->sqlcode, msg);
#else
  err_fp = fopen(err_fn, "a+");
#endif
  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");
#ifdef DARIVERSION
  fclose(err_fp);
#endif
}

/*---------------------------------------------------------------*/
/*       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());
  timeStamp[19] = (char)NULL;
#ifndef DARIVERSION
  fprintf(stderr,"ordstat debugit called: %s <====\n", msg);
#endif
  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_I
D);
    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"
#ifdef ACID_TEST
#include <time.h>
#include <sys/time.h>
#define acidtime(tvsec,tvusec) tvsec*1000+tvusec/1000
#endif

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

/*---------------------------------------------------------------*/
/*             PAYMENT Service                                   */
/*---------------------------------------------------------------*/
#ifndef DARIVERSION
int payment_sql (struct payment_struct *payment)
#else
#ifdef USE_SQLEPROC
int pays (struct sqlchar *pinput_data,
        struct sqlda   *pin_sqlda,
        struct sqlda   *pout_sqlda,
        struct sqlca   *pca )
#else
int pays (void *reserved1,
        void *reserved2,
        struct sqlda *pin_sqlda,
        struct sqlca *pca)
#endif
#endif
{

#define  MAXCLAST   3000

#ifdef ACID_TEST
  FILE *out;
  struct timeval tv;
  struct timezone tz;
  int audittime;
#endif

  int     retry_count = -1;

#ifdef DARIVERSION
  struct payment_struct *payment;
```

```
#endif

  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;
  short   h_c_id;

  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_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 ;

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

#ifdef DEBUGIT
  debugit(payment, "Server upon entry");
#endif
  strcpy(curr_tmstmp, current_tmstmp());

  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;
  strcpy(c_last, payment->s_C_LAST);
  memcpy(payment->s_H_DATE, curr_tmstmp, 19);
  payment->s_H_DATE[19] =  (char)NULL;

#ifdef ACID_TEST
  out=fopen("/tmp/tpcc.pay.out","w");
  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  fprintf(out,"\n--------- START of PAYMENT PID: %d ----------\n\n",getpid());
  fprintf(out, "PAYMENT PID: %d, begin transaction time: %u \n",
        getpid(), audittime);
  fprintf(out, "W_ID: %d \t D_ID: %d \t C_ID: %d \n",
        payment->s_W_ID, payment->s_D_ID, payment->s_C_ID);
#endif

 retry_tran:
                retry_count++;

#ifdef ACID_TEST
  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  fprintf(out, "PAYMENT PID: %d, before read & update customer time: %u \n",
        getpid(), audittime);
#endif

  if (c_id == 0) {

    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;
  }

#ifdef ACID_TEST
  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  fprintf(out, "PAYMENT PID: %d, after read customer time: %u \n",
        getpid(), audittime);
  fprintf(out,"W_ID: %d \t D_ID: %d \t C_ID: %d \n",
        payment->s_W_ID, payment->s_D_ID, payment->s_C_ID);
  fprintf(out,"AMOUNT: %0.2f \t C_BALANCE: %8.2f \n",
        payment->s_H_AMOUNT, c_balance);
#endif

  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;
  payment->s_C_BALANCE = c_balance;

  c_balance -= h_amount;
```

```
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 NOT ATOMIC STATIC*/
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;
#ifdef ACID_TEST
   EXEC SQL

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

   if (sqlca.sqlcode != 0) {
     if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
     sqlerror("INSERT INTO history",430, &sqlca);
     goto ferror;
   }
#else
   /*EXEC SQL BEGIN COMPOUND NOT ATOMIC STATIC*/
   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;
     }
   }
#endif

#ifdef ACID_TEST
   gettimeofday(&tv, &tz);
   audittime = acidtime(tv.tv_sec, tv.tv_usec);
   fprintf(out, "PAYMENT PID: %d, after update customer time: %u \n",
         getpid(), audittime);
   fprintf(out, "W_ID: %d \t D_ID: %d \t C_ID: %d \n",
         payment->s_W_ID, payment->s_D_ID, payment->s_C_ID);
   fprintf(out,"AMOUNT: %0.2f \t C_BALANCE: %8.2f \n",
         payment->s_H_AMOUNT, c_balance);
   gettimeofday(&tv, &tz);
   audittime = acidtime(tv.tv_sec, tv.tv_usec);
   fprintf(out, "PAYMENT PID: %d, wait before commit time: %u \n",
```

```
        getpid(), audittime);
  sleep(30);

  if (payment->s_H_AMOUNT == 410.0) {
    EXEC SQL rollback work;
    if (sqlca.sqlcode != 0) fprintf(out,"PAY:*** ROLLBACK FAILED ***\n");
    gettimeofday(&tv, &tz);
    audittime = acidtime(tv.tv_sec, tv.tv_usec);
    fprintf(out, "PAYMENT PID: %d, rolled back time: %u \n",
          getpid(), audittime);
  }
  else {
    EXEC SQL COMMIT;
    if (sqlca.sqlcode != 0) {
      sqlerror("COMMIT",438, &sqlca);
      goto ferror;
    }
  }

  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  if (payment->s_H_AMOUNT != 410.0) {
    fprintf(out, "PAYMENT PID: %d, commit transaction time: %u \n",
          getpid(), audittime);
  }
  fprintf(out,"\n---------- END of PAYMENT PID: %d ----------\n\n",getpid());
  fclose(out);
#endif

  /*-----------------------------------------------*/
  /*  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:
#ifdef DARIVERSION
#ifdef USE_SQLEPROC
  memcpy(pout_sqlda->sqlvar[0].sqldata, payment,
        sizeof(struct payment_struct));
#endif
  memcpy( pca, &sqlca, sizeof( struct sqlca ) );
  if ( sqlca.sqlcode < 0 )
    rc = SQLZ_DISCONNECT_PROC;
  else
    rc = SQLZ_HOLD_PROC;
#endif

#ifdef DEBUGIT
  debugit(payment, "Server prior to return");
#endif
  return(rc);
}

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

#ifndef DARIVERSION
  err_fp = stdout;
  fprintf(stderr,"PAY: %2d sqlcode: %4d %s\n", ptat, psqlca->sqlcode, msg);
#else
  err_fp = fopen(err_fn, "a+");
#endif
  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");
#ifdef DARIVERSION
  fclose(err_fp);
#endif
}

/*-----------------------------------------------------------------*/
/*       debugit                                                   */
/*-----------------------------------------------------------------*/
void debugit (struct payment_struct *payment_ptr, char *msg)
{
  FILE *debug_fp;
  char  debug_fn[25] = "/tmp/payment.debug.out";
  char  timeStamp[27];
  strcpy(timeStamp, current_tmstmp());
  timeStamp[19] = (char)NULL;
#ifndef DARIVERSION
  fprintf(stderr,"payment debugit called: %s <====\n", msg);
#endif
  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: neword(s).sqc (Server code)                                    */
/*****************************************************************************/

#include "../include/tpcc.h"
#include <stdlib.h>
#ifdef ACID_TEST
#include <time.h>
#include <sys/time.h>
#define acidtime(tvsec,tvusec) tvsec*1000+tvusec/1000
#endif

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

#ifndef DARIVERSION
int duplicate_items = 0;

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);
}

int neword_sql (struct neword_struct *neword)
#else
#ifdef USE_SQLEPROC
int newords (struct sqlchar *pinput_data,
            struct sqlda   *pin_sqlda,
            struct sqlda   *pout_sqlda,
            struct sqlca   *pca )
#else
int newords (void *reserved1,
            void *reserved2,
            struct sqlda *pin_sqlda,
            struct sqlca *pca)
#endif
#endif
{

#ifdef ACID_TEST
  FILE *out;
  char out_fn[50];
  struct timeval tv;
  struct timezone tz;
  int audittime;
  int wait = 1;
  int mypid;
#endif

#ifdef DARIVERSION
  struct neword_struct *neword;
  int duplicate_items = 0;
#endif

  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;

  short  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;

#ifdef DARIVERSION
  /* Retrieve the 'neword' structure from the SQLDA */
  neword =(struct neword_struct *)pin_sqlda->sqlvar[0].sqldata;
  duplicate_items = neword->duplicate_items;
#else
#ifndef ACID_TEST
  duplicate_items=0;
  qsort (neword->item, neword->s_O_OL_CNT, sizeof (neword->item[0]), icmp);
#endif
#endif
#ifdef DEBUGIT
  debugit(neword, "Server upon entry");
#endif

  item_idx = -1;
  strcpy(curr_tmstmp, current_tmstmp());
  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;

#ifdef ACID_TEST
  mypid = getpid();
  sprintf(out_fn, "/tmp/tpcc.neword.out.%d",mypid);
```

```
  out=fopen(out_fn,"a");
  allLocal = 0;
  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  fprintf(out,"\n---------- START of NEWORDER PID: %d ----------\n\n",mypid);
  fprintf(out,"NEWORD PID: %d, begin transaction time: %u \n",mypid,audittime);
  fprintf(out, "W_ID: %d \t D_ID: %d \t C_ID: %d \n",
        neword->s_W_ID, neword->s_D_ID, neword->s_C_ID);
#endif

 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.                                               */
  /*----------------------------------------------------------------*/
#ifdef ACID_TEST
  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  fprintf(out, "NEWORD PID: %d, before fetch D_TAX & D_NEXT_O_ID time: %u \n",
        mypid, audittime);
#endif

  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;
  }

#ifdef ACID_TEST
  o_id = next_o_id;
  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  fprintf(out, "NEWORD PID: %d, after fetch D_TAX & D_NEXT_O_ID time: %u \n",
        mypid, audittime);
  fprintf(out, "W_ID: %d \t D_ID: %d \t D_NEXT_O_ID: %d \n",
        neword->s_W_ID, neword->s_D_ID, next_o_id);

  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  fprintf(out, "NEWORD PID : %d, before inserting into orders: %u \n",
        mypid, audittime);
  EXEC SQL
    INSERT INTO orders VALUES
    (:o_id, :o_c_id, :d_id, :w_id, :curr_tmstmp, 0, :OLITEMS, :allLocal);
  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  fprintf(out, "NEWORD PID : %d, after inserting into orders: %u \n",
        mypid, audittime);
  if (sqlca.sqlcode != 0) {
    if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
    sqlerror("insert into ORDERS",686, &sqlca);
    goto ferror;
  }

  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  fprintf(out, "NEWORD PID : %d, before inserting into new_order: %u\n",
        mypid, audittime);
  EXEC SQL
    INSERT INTO new_order VALUES (:o_id, :d_id, :w_id);
  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  fprintf(out, "NEWORD PID : %d, after inserting into new_order: %u \n",
        mypid, audittime);
  if (sqlca.sqlcode != 0) {
    if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
    sqlerror("insert into NEW_ORDER",694, &sqlca);
    goto ferror;
  }

  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  fprintf(out, "NEWORD PID : %d, before select of warehouse,customer: %u \n",
        mypid, audittime);

  EXEC SQL
    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;
  gettimeofday(&tv, &tz);
  audittime = acidtime(tv.tv_sec, tv.tv_usec);
  fprintf(out, "NEWORD PID : %d, after select of warehouse,customer: %u \n",
        mypid, audittime);
  if (sqlca.sqlcode != 0) {
    if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
    sqlerror("SELECT WAREHOUSE & CUSTOMER",702, &sqlca);
    goto ferror;
  }

#endif

  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) */
  {
#ifndef ACID_TEST
    EXEC SQL OPEN Item_cursor;
    if (sqlca.sqlcode != 0) {
      if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
      sqlerror("open Item_cursor",319, &sqlca);
      goto ferror;
    }
#endif
  } /* 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) */
    {
#ifndef ACID_TEST
      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;
#else
      EXEC SQL SELECT i_id, i_price, i_name, i_data
        INTO :ol_i_id, :item_price, :item_name, :i_data
        FROM item
        WHERE i_id = :id0;

      if (sqlca.sqlcode == 0) {
        gettimeofday(&tv, &tz);
        audittime = acidtime(tv.tv_sec, tv.tv_usec);
        i_price = (double) item_price / (double) 100.0;
        fprintf(out, "NEWORD PID: %d, after read item_price time: %u \n",
              mypid, audittime);
        fprintf(out, "I_ID: %d \t I_PRICE: %0.2f \n",
              ol_i_id, i_price);
        if  (wait) {
          sleep (15);
          wait = 0;
        }
      }
#endif
      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;
        }
      }
      /*if (item_idx != i) nsqlerror("Could not match ITEM!!!",483);*/
    }

    ++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;
    neword->s_total_amount += neword->item[item_idx].s_OL_AMOUNT;

    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 NOT ATOMIC STATIC */
      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 ... */
#ifdef ACID_TEST
    if (item_cnt<OLITEMS) id0 = neword->item[item_cnt].s_OL_I_ID;
#endif
  }   /* end for (item_cnt = 0; item_cnt < OLITEMS;) */

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

#ifndef ACID_TEST
    /*EXEC SQL BEGIN COMPOUND NOT ATOMIC STATIC*/
    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;
      }
    }

#else
    gettimeofday(&tv, &tz);
    audittime = acidtime(tv.tv_sec, tv.tv_usec);
    fprintf(out,"NEWORD PID : %d, wait before rollback time: %u \n",
          mypid, audittime);
    sleep(60);
#endif

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

#ifdef ACID_TEST
    gettimeofday(&tv, &tz);
    audittime = acidtime(tv.tv_sec, tv.tv_usec);
    fprintf(out, "NEWORD PID: %d, after rolled back time: %u \n",
          mypid, audittime);
    fprintf(out, "W_ID: %d \t D_ID: %d \t O_ID: %d\t D_NEXT_O_ID: %d \n ",
          neword->s_W_ID, neword->s_D_ID, neword->s_O_ID, next_o_id);
    fprintf(out,"\n---------- END of NEWORDER PID: %d ----------\n\n",mypid);
    fclose(out);
#endif

    goto mexit;
  } else {

#ifdef ACID_TEST

    gettimeofday(&tv, &tz);
    audittime = acidtime(tv.tv_sec, tv.tv_usec);
    fprintf(out, "NEWORD PID : %d, wait before commit time: %u \n",
          mypid, audittime);

    if (c_id != 123 )
      sleep(60);

    EXEC SQL COMMIT;
    gettimeofday(&tv, &tz);
    audittime = acidtime(tv.tv_sec, tv.tv_usec);
    fprintf(out, "NEWORD PID: %d, commit transaction time: %u \n",
          mypid, audittime);
    if (sqlca.sqlcode != 0) {
      sqlerror("COMMIT",564, &sqlca);
      goto ferror;
    }
    EXEC SQL OPEN dist_cur;

    if (sqlca.sqlcode != 0) {
      sqlerror("OPEN dis_cur",740, &sqlca);
      goto ferror;
    }
    EXEC SQL
      FETCH dist_cur INTO
      :dist_tax, :next_o_id;

    if (sqlca.sqlcode != 0) {
      sqlerror("FETCH dist_cur",752, &sqlca);
      goto ferror;
    }
    fprintf(out, "W_ID: %d \t D_ID: %d \t O_ID: %d\t D_NEXT_O_ID: %d \n ",
          neword->s_W_ID, neword->s_D_ID, neword->s_O_ID, next_o_id);
    fprintf(out,"\n---------- END of NEWORDER PID: %d ----------\n\n",getpid());
    fclose(out);
    EXEC SQL COMMIT;
    if (sqlca.sqlcode != 0) {
      sqlerror("COMMIT",564, &sqlca);
      goto ferror;
    }

#else       /* not ACID_TEST */

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

    /* EXEC SQL BEGIN COMPOUND NOT ATOMIC STATIC */
    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;
      }
    }
#endif          /* of ifdef ACID_TEST */

    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;

    neword->s_total_amount = neword->s_total_amount *
      (1 - neword->s_C_DISCOUNT) * (1 + ware_tax + dist_tax);
  }

  /*---------------------------------------------*/
  /*  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:
#ifdef DARIVERSION
#ifdef USE_SQLEPROC
  memcpy(pout_sqlda->sqlvar[0].sqldata, neword,
        sizeof( struct neword_struct ));
#endif
  memcpy( pca, &sqlca, sizeof( struct sqlca ) );
  if ( sqlca.sqlcode < 0 )
    rc = SQLZ_DISCONNECT_PROC;
  else
    rc = SQLZ_HOLD_PROC;
#endif

#ifdef DEBUGIT
  debugit(neword, "Server prior to return");
#endif
  return(rc);
}

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

#ifndef DARIVERSION
  err_fp = stdout;
  fprintf(stderr,"NEWORD: %2d sqlcode: %4d %s\n", ptat, psqlca->sqlcode, msg);
#else
  err_fp = fopen(err_fn, "a+");
#endif
  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");
#ifdef DARIVERSION
  fclose(err_fp);
#endif
}


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

#ifndef DARIVERSION
  fprintf(stdout,"NEWORD: %2d %s\n", ptat,msg);
  fprintf(stderr,"NEWORD: %2d %s\n", ptat,msg);
#else
  err_fp = fopen(err_fn, "a+");
  fprintf(err_fp,"NEWORD: %2d %s\n", ptat,msg);
  fclose(err_fp);
#endif
}

/*-------------------------------------------------------------*/
/*       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;
  strcpy(timeStamp, current_tmstmp());
  timeStamp[19] = (char)NULL;
#ifndef DARIVERSION
  fprintf(stderr,"neword debugit called: %s <====\n", msg);
#endif
  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: stklv(s).sqc (Server code)                                     */
/*****************************************************************************/

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

#ifdef ACID_TEST
#include <time.h>
#include <sys/time.h>
#define acidtime(tvsec,tvusec) tvsec*1000+tvusec/1000
#endif

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

/*---------------------------------------------------------------------*/
/*              STOCKLEV Service                                       */
/*---------------------------------------------------------------------*/
#ifndef DARIVERSION
int stocklev_sql (struct stocklev_struct *stocklev)
#else
#ifdef USE_SQLEPROC
int stklvs (struct sqlchar *pinput_data,
            struct sqlda   *pin_sqlda,
            struct sqlda   *pout_sqlda,
            struct sqlca   *pca )
#else
int stklvs (void *reserved1,
            void *reserved2,
            struct sqlda *pin_sqlda,
            struct sqlca *pca)
#endif
#endif
{

#ifdef ACID_TEST
   time_t acidTime;
   FILE *out;
   struct timeval tv;
   struct timezone tz;
   int audittime;
#endif

#ifdef DARIVERSION
   struct stocklev_struct *stocklev;
#endif

   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;

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

#ifdef DEBUGIT
   debugit(stocklev, "Server upon entry");
#endif
   /*-----------------------------------------*/
   /*  Read DISTRICT                          */
   /*-----------------------------------------*/
   w_id = stocklev->s_W_ID;
   d_id = stocklev->s_D_ID;

#ifdef ACID_TEST
   out=fopen("/tmp/tpcc.stklv.out","a+");
   gettimeofday(&tv, &tz);
   audittime = acidtime(tv.tv_sec, tv.tv_usec);
   fprintf(out,"\n---------- START of STOCKLEVEL PID: %d ----------\n\n",getpid());
   fprintf(out, "STOCKLEVEL PID: %d, begin transaction time: %u \n",
         getpid(), audittime);
   fprintf(out, "W_ID: %d \t D_ID: %d \n",
         stocklev->s_W_ID, stocklev->s_D_ID);
   gettimeofday(&tv, &tz);
   audittime = acidtime(tv.tv_sec, tv.tv_usec);
   time(&acidTime);
   fprintf(out,"STOCKLEV PID: %d, before read of district: %u %s",
         getpid(), audittime, ctime(&acidTime));
#endif

 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;
#ifdef ACID_TEST
   gettimeofday(&tv, &tz);
   audittime = acidtime(tv.tv_sec, tv.tv_usec);
   time(&acidTime);
   fprintf(out,"STOCKLEV PID: %d, after read of district d_next_o_id=%d: %u %s",
         getpid(), d_next_o_id, audittime, ctime(&acidTime));
#endif

   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;
#ifdef ACID_TEST
   gettimeofday(&tv, &tz);
   audittime = acidtime(tv.tv_sec, tv.tv_usec);
   time(&acidTime);
   fprintf(out,"STOCKLEV PID: %d, before read of order_line and stock time: %u %s",
         getpid(), audittime, ctime(&acidTime));

   EXEC SQL
     SELECT count(distinct S_I_ID)
     into :low_stock
     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;

   stocklev->s_low_stock = low_stock;

   gettimeofday(&tv, &tz);
   audittime = acidtime(tv.tv_sec, tv.tv_usec);
   time(&acidTime);
   fprintf(out,"STOCKLEV PID: %d, after read of order_line and stock time: %u %s",
         getpid(), audittime, ctime(&acidTime));
   fprintf(out, "W_ID: %d \t D_ID: %d \t O_ID: %d \n",
         stocklev->s_W_ID, stocklev->s_D_ID, o_id);
   fprintf(out, "D_NEXT_O_ID: %d \t THRESHOLD: %d \t LOW_STOCK: %d \n",
         d_next_o_id, stocklev->s_threshold, stocklev->s_low_stock);

   if (sqlca.sqlcode != 0) {
     if(sqlca.sqlcode == DEADLOCK) goto retry_tran;
     sqlerror("select count(distinct)",83, &sqlca);
     goto ferror;
   }
   EXEC SQL COMMIT;
   if (sqlca.sqlcode != 0) {
     sqlerror("COMMIT",414, &sqlca);
     goto ferror;
   }
   gettimeofday(&tv, &tz);
   audittime = acidtime(tv.tv_sec, tv.tv_usec);
   time(&acidTime);
   fprintf(out, "STOCKLEV PID: %d, commit transaction time: %u %s",
```

```
        getpid(), audittime, ctime(&acidTime));
  fprintf(out,"\n---------- END of STOCKLEV PID: %d ----------\n\n",getpid());
  fclose(out);
#else
  /*EXEC SQL BEGIN COMPOUND NOT ATOMIC STATIC*/
  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;
    }
  }
#endif

  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:
#ifdef DARIVERSION
#ifdef USE_SQLEPROC
  memcpy(pout_sqlda->sqlvar[0].sqldata, stocklev,
        sizeof(struct stocklev_struct));
#endif
  memcpy( pca, &sqlca, sizeof( struct sqlca ) );
  if ( sqlca.sqlcode < 0 )
    rc = SQLZ_DISCONNECT_PROC;
  else
    rc = SQLZ_HOLD_PROC;
#endif

#ifdef DEBUGIT
  debugit(stocklev, "Server prior to return");
#endif
  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;

#ifndef DARIVERSION
  err_fp = stdout;
  fprintf(stderr,"STKLV: %2d sqlcode: %4d %s\n", ptat, psqlca->sqlcode, msg);
#else
  err_fp = fopen(err_fn, "a+");
#endif
  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");
#ifdef DARIVERSION
  fclose(err_fp);
#endif
}

/*----------------------------------------------------------------*/
/*       debugit                                                  */
/*----------------------------------------------------------------*/
void debugit (struct stocklev_struct *stocklev_ptr, char *msg)
{
  FILE *debug_fp;
  char  debug_fn[25] = "/tmp/stklv.debug.out";
  char  timeStamp[27];
  strcpy(timeStamp, current_tmstmp());
  timeStamp[19] = (char)NULL;
#ifndef DARIVERSION
  fprintf(stderr,"stklv debugit called: %s <====\n", msg);
#endif
  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: Tunable Parameters

B.1 Database Parameters

```
               DBMS CONFIGURATION for RS/6000-R30 & J30

         Database Manager Configuration

    Node type = Database Server with local and remote clients

Database manager configuration release level            = 0x0600

CPU speed (millisec/instruction)             (CPUSPEED) = 9.021775e-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) = STAFF
SYSCTRL group name                   (SYSCTRL_GROUP) =
SYSMAINT group name                  (SYSMAINT_GROUP) =

Database manager authentication      (AUTHENTICATION) = CLIENT

Default database path                (DFTDBPATH) = /home/tpcc

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) = 59
Max number of existing agents        (MAXAGENTS) = 144
Max number of concurrent agents      (MAXCAGENTS) = MAXAGENTS
Maximum number of idle agents        (MAX_IDLEAGENTS) = 3

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

Service name                         (SVCENAME) = db2server
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) =


      Database Configuration for Database tpcc

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                  = NO
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) = 410000
Max storage for lock lists (4KB)     (LOCKLIST) = 150

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) = 12
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) = 125
Average number of active applications  (AVG_APPLS) = 1
Max DB files open per application    (MAXFILOP) = 800

Log file size (4KB)                  (LOGFILSIZ) = 4000
Number of primary log files          (LOGPRIMARY) = 50
Number of secondary log files        (LOGSECOND) = 0
Changed path to log files            (NEWLOGPATH) =
Path to log files                               = /tpcc.log/
Next active log file                            =
First active log file                           =

Group commit count                   (MINCOMMIT) = 3
Percent log file reclaimed before soft chckpt (SOFTMAX) = 95
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
```

B.2 Transaction Monitor Parameters

```
*RESOURCES
IPCKEY 44003
UID 202
GID 1
MASTER client1
PERM 0666
MODEL SHM
LDBAL Y
MAXACCESSERS 1000
MAXSERVERS 80
MAXSERVICES 80
SCANUNIT 60
SANITYSCAN 5
BLOCKTIME 3
BBLQUERY 60

*MACHINES
client2 LMID=client1
 ROOTDIR="/u/tuxedo"
 APPDIR="/u/tpcc/dari.tux/app"
 TUXCONFIG="/u/tpcc/dari.tux/install.tux/tuxconfig"
 ULOGPFX="/u/tpcc/dari.tux/install.tux/ULOG"
```

```
*GROUPS group1 LMID=client2 GRPNO=1

*SERVERS

DEFAULT:    SRVGRP=group1
DEFAULT:    REPLYQ=N
DEFAULT:    MAXGEN=2
DEFAULT:    RESTART=Y
DEFAULT:    CLOPT="-A"
#

#
newordpay SRVID=10 MIN=4 RQADDR="10"
newordpay  SRVID=20 MIN=4 RQADDR="20"
neword pay SRVID=30 MIN=4 RQADDR="30"
#
stklv SRVID=300 MIN=2 RQADDR="300"     # STOCKLEV servers     1
stklv SRVID=310 MIN=2 RQADDR="310"     # STOCKLEV servers     1
stklv SRVID=320 MIN=2 RQADDR="320"     # STOCKLEV servers     1
#
delivery SRVID=350 MIN=1 RQADDR="350"  # DELIVERY servers     1
delivery SRVID=360 MIN=1 RQADDR="360"  # DELIVERY servers     1
#
#ENVFILE="$(APPDIR)/cstuxenv"

*SERVICES
payment_sql
stocklev_sql
ordstat_sql
delivery_sql
neword_sql
```

B.3 AIX Parameters

```
keylock     normal        State of system keylock at boot time
maxbuf      20            Maximum number of pages in block I/O BUFFER CACHE
maxmbuf     2048          Maximum Kbytes of real memory allowed for MBUFS
maxuproc    2000          Maximum number of PROCESSES allowed per user
autorestart  false        Automatically REBOOT system after a crash
iostat          true      Continuously maintain DISK I/O history
realmem     2097152       Amount of usable physical memory in Kbytes
conslogin   enable        System Console Login
maxpout     0             HIGH water mark for pending write I/Os per file
minpout     0             LOW water mark for pending write I/Os per file
full core       false     enable full  CORE dump
```

Appendix C: Database Setup Code

C.1 Table Creation Code

```
/*************************************************************************/

/*      file:  crtb.sqc                                                  */
/*                                                                       */
/*      TPC-C benchmark for DB2/6000. 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          SMALLINT    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          SMALLINT    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");
}
```

C.2 Index Creation Scripts

```
#!/bin/ksh
#
# create index program for tpcc
#

: ${TPCC_DBNAME:=tpcc}

create_unique_index () {
  tname=$1
  iname=$2
  shift; shift
  keys=$*
  echo ""
  echo "Creating unique index $iname on $tname table"
  db2 "drop index $iname" > /dev/null;
  echo db2 "create unique index $iname on $tname ($keys)";
  sqlcode=`db2 -ec +o "create unique index $iname on $tname ($keys)"`;
  if [ $sqlcode != 0 ]
  then
   echo "create unique index failed with sqlcode = $sqlcode"
   exit $sqlcode;
  fi
  sqlcode=`db2 -ec +o "COMMIT WORK"`;
  if [ $sqlcode != 0 ]
  then
   echo "COMMIT failed with sqlcode = $sqlcode"
   exit $sqlcode;
  fi
}

create_index () {
  tname=$1
  iname=$2
  shift; shift
  keys=$*
  echo ""
  echo "Creating index $iname on $tname table"
  db2 "drop index $iname" > /dev/null;
  echo db2 "create index $iname on $tname ($keys)";
  sqlcode=`db2 -ec +o "create index $iname on $tname ($keys)"`;
  if [ $sqlcode != 0 ]
  then
   echo "create index failed with sqlcode = $sqlcode"
   exit $sqlcode;
  fi
  sqlcode=`db2 -ec +o "COMMIT WORK"`;
  if [ $sqlcode != 0 ]
  then
   echo "COMMIT failed with sqlcode = $sqlcode"
   exit $sqlcode;
  fi
}

if [[ $# = 1 ]]
then
  option=$1
else
  echo "Pick table to create index on: ";
  echo "1. all tables ";
  echo "2. warehouse ";
  echo "3. district ";
  echo "4. item ";
  echo "5. stock ";
  echo "6. customer ";
  echo "7. history ";
  echo "8. orders ";
  echo "9. new-order ";
  echo "10. order-line ";
  echo "\nEnter option: \c";
  read option;
fi

echo ""
echo "connecting to database $TPCC_DBNAME"
sqlcode=`db2 -ec +o "CONNECT TO $TPCC_DBNAME IN SHARE MODE"`;
if [ $sqlcode != 0 ]
then
  echo "connect to $TPCC_DBNAME failed sqlcode = $sqlcode"
  exit $sqlcode;
fi
```

```
case $option in
 # all tables
 1)  echo "creating all indexes..."
     time crtix 10
     time create_unique_index warehouse ware_idx1 w_id
     time create_unique_index district dist_idx1 d_id,d_w_id
     time create_unique_index item item_idx1 i_id
     time create_unique_index stock stock_idx1 s_w_id,s_i_id
     time create_unique_index customer cust_idx1 c_w_id,c_d_id,c_id
     time create_index customer cust_idx2 c_w_id,c_d_id,c_last,c_first,c_id
     echo "\nNo index to be created for history table"
     time create_unique_index orders ordr_idx1 o_w_id,o_d_id,o_id
     time create_unique_index orders ordr_idx2 o_w_id,o_d_id,o_c_id,o_id desc
     time create_unique_index new_order nu_ord_idx1 no_w_id,no_d_id,no_o_id ;;
 # warehouse table
 2)  time create_unique_index warehouse ware_idx1 w_id ;;
 # district table
 3)  time create_unique_index district dist_idx1 d_id,d_w_id ;;
 # item table
 4)  time create_unique_index item item_idx1 i_id ;;
 # stock table
 5)  time create_unique_index stock stock_idx1 s_w_id,s_i_id ;;
 # customer table
 6)  time create_unique_index customer cust_idx1 c_w_id,c_d_id,c_id
     time create_index customer cust_idx2 c_w_id,c_d_id,c_last,c_first,c_id ;;
 # history table
 7)  echo "\nNo index to be created for history table" ;;
 # orders table
 8)  time create_unique_index orders ordr_idx1 o_w_id,o_d_id,o_id
     time create_unique_index orders ordr_idx2 o_w_id,o_d_id,o_c_id,o_id desc ;;
 # new_order table
 9)  time create_unique_index new_order nu_ord_idx1 no_w_id,no_d_id,no_o_id ;;
 # order_line table
 10) time create_unique_index order_line oline_idx1
ol_w_id,ol_d_id,ol_o_id,ol_number ;;
 # default
 *)  echo "Invalid option $option was selected"
     exit 1
esac

sqlcode=`db2 -ec +o "CONNECT RESET"`;
if [ $sqlcode != 0 ]
then
  echo "CONNECT RESET failed with sqlcode = $sqlcode"
  exit $sqlcode;
fi

exit 0
```

C.3 Data Generation Code

```
/************************************************************************/
/*      file:  gendata.sqc                                              */
/*                                                                      */
/*      TPC-C benchmark of DB2/6000. 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] = "tpcc";
char   currtmstmp[26];
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 ;
  long          s_ytd;
  long          s_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 (ware_num = 1; ware_num <= ware_count; ware_num++)
  {
    fprintf(stderr, "Now loading Stock for Warehouse #%d",ware_num);
    for (stock_num = 1; stock_num <= STOCK_PER_WAREHOUSE; stock_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|%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
      {
        /* WWW 94/6/6: changed column ordering to match crtb.sqc */
        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|%f|%f|%d\n",dist_num,ware_num,dist_name,dist_s
treet_1,dist_street_2,dist_city,dist_state,dist_zip,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 (ware_num = 1; ware_num <= ware_count; ware_num++)
  {
    fprintf(stderr, "Now loading Customers 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++)
      {
        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|%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,currtmstmp,cu
st_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[26];
  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 D. Screen Layouts

D.1 New Order Transaction

```
(1)New-Order(2)Payment(3)Order-Status(4)Delivery(5)Stock-Level(9)Exit

                              New Order
Warehouse:9999  District: 99                  Date:DD-MM-YYYY hh:mm:ss
Customer: 9999  Name:XXXXXXXXXXXXXXXXXXXXX  Credit: XX  %Disc: 99.99
Order Number: 99999999  Number of Lines: 99  W_Tax:99.99 D_tax:99.99

Supp_W  Item_Id Item Name           Qty  Stock  B/G Price     Amount
 9999   999999  XXXXXXXXXXXXXXXXXX  99   9999   X  $999.99  $9999.99
 9999   999999  XXXXXXXXXXXXXXXXXX  99   9999   X  $999.99  $9999.99
 9999   999999  XXXXXXXXXXXXXXXXXX  99   9999   X  $999.99  $9999.99
 9999   999999  XXXXXXXXXXXXXXXXXX  99   9999   X  $999.99  $9999.99
 9999   999999  XXXXXXXXXXXXXXXXXX  99   9999   X  $999.99  $9999.99
 9999   999999  XXXXXXXXXXXXXXXXXX  99   9999   X  $999.99  $9999.99
 9999   999999  XXXXXXXXXXXXXXXXXX  99   9999   X  $999.99  $9999.99
 9999   999999  XXXXXXXXXXXXXXXXXX  99   9999   X  $999.99  $9999.99
 9999   999999  XXXXXXXXXXXXXXXXXX  99   9999   X  $999.99  $9999.99
 9999   999999  XXXXXXXXXXXXXXXXXX  99   9999   X  $999.99  $9999.99
 9999   999999  XXXXXXXXXXXXXXXXXX  99   9999   X  $999.99  $9999.99
 9999   999999  XXXXXXXXXXXXXXXXXX  99   9999   X  $999.99  $9999.99
 9999   999999  XXXXXXXXXXXXXXXXXX  99   9999   X  $999.99  $9999.99
 9999   999999  XXXXXXXXXXXXXXXXXX  99   9999   X  $999.99  $9999.99
Execution Status: XXXXXXXXXXXXXXXXXXXX
```

D.2 Payment Transaction

```
(1)New-Order(2)Payment(3)Order-Status(4)Delivery(5)Stock-Level(9)Exit

                                 Payment
Date: DD-MM-YYYY hh:mm:ss

Warehouse: 9999                      District: 99
XXXXXXXXXXXXXXXXXXX                  XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX                  XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX XX-XXXXX-XXXX    XXXXXXXXXXXXXXXXXXX XX-XXXXX-XXXX

Customer: 9999  Cust-Warehouse:9999 Cust-District: 99
Name: XXXXXXXXXXXXXXXXXXX XX XXXXXXXXXXX   Since: DD-MM-YYYY
      XXXXXXXXXXXXXXXXXXX          Credit: XX
      XXXXXXXXXXXXXXXXXXX          %Disc: 99.99
      XXXXXXXXXXXXXXXXXXX XX-XXXXX-XXXX   Phone: XXXXXXX-XXX-XXX-XXXX

Amount Paid:     $9999.99           New Cust-Balance: $-9999999999.99
Credit Limit: $9999999999.99

Cust-Data:XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
          XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
          XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

D.3 Order-Status Transaction

```
(1)New-Order(2)Payment(3)Order-Status(4)Delivery(5)Stock-Level(9)Exit

                                Order-Status

Warehouse: 9999   District: 99
Customer: 9999  Name: XXXXXXXXXXXXXXXXXXX XX XXXXXXXXXXXXXX
Cust-Balance: $-9999999.99

Order-Number:99999999  Entry-Date:DD-MM-YYYY hh:mm:ss Carrier-Number:99
Supply-W   Item-Id  Qty      Amount      Delivery Date
 9999      999999   99       $99999.99   DD-MM-YYYY
 9999      999999   99       $99999.99   DD-MM-YYYY
 9999      999999   99       $99999.99   DD-MM-YYYY
 9999      999999   99       $99999.99   DD-MM-YYYY
 9999      999999   99       $99999.99   DD-MM-YYYY
 9999      999999   99       $99999.99   DD-MM-YYYY
 9999      999999   99       $99999.99   DD-MM-YYYY
 9999      999999   99       $99999.99   DD-MM-YYYY
 9999      999999   99       $99999.99   DD-MM-YYYY
 9999      999999   99       $99999.99   DD-MM-YYYY
 9999      999999   99       $99999.99   DD-MM-YYYY
 9999      999999   99       $99999.99   DD-MM-YYYY
 9999      999999   99       $99999.99   DD-MM-YYYY
 9999      999999   99       $99999.99   DD-MM-YYYY
```

D.4 Delivery Transaction

```
(1)New-Order(2)Payment(3)Order-Status(4)Delivery(5)Stock-Level(9)Exit

                                Delivery

Warehouse: 9999

Carrier Number: 99

Execution Status: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

D.5 Stock-Level Transaction

```
(1)New-Order(2)Payment(3)Order-Status(4)Delivery(5)Stock-Level(9)Exit

                                Stock-Level

Warehouse: 9999    District: 99

Stock Level Threshold: 99

low stock: 999
```

Appendix E: RTE Scripts

E.1 RTE Parameters

```
/* define hosts that will be used here */
MASTER master
SLAVES driver1, driver2, driver3, driver4
SUT   = "ewildhorse"
CLIENT_REAL = "client1 client2 client4"

CLIENT client1t  tpcc ccpt1
CLIENT client2t   tpcc  ccpt1
CLIENT client4t  tpcc ccpt1

RLOGIN rlogin
PROGRAM typewatch "rte_typewatch %s %c 23 %m"
TELNET telnet 23
#if 0
TTY    tty slave3 client1t "/dev/tty%d" 2-89
#endif
RAMPUP=15:00
RUNTIME=30:00
RAMPDOWN=15:00

INTERVAL=25:00   /* Interval to calculate mix from */

/*NOBEGIN              = 1  */

BEGIN_WAIT=9:00
/*RAMPDOWN_WAIT=2:00*/
KEYSTROKE_PACKET_SIZE = 10
KEYSTROKE_SLEEP       = 25:00

MAX_CONCURRENT_SPAWN  = 20
SPAWN_COUNT           = 2
OUTPUTNAME                    = "../runs/analyze2"

/* User variables.  Think, Emulex Delay (Trans, Menu), %desired, %min, %max */
NEWORDER = "12.20,   0.42, 0.38"
PAYMENT  = "12.20,   0.19, 0.23,  43.2, 42, 45.3"
ORDSTAT  = "10.2,   0.39, 0.21,   4.1,  3.9,  4.2"
DELIVERY = "05.2,   0.19, 0.15,   4.1,  3.9,  4.2"
STOCKLEV = "05.2,   0.25, 0.18,   4.1,  3.9,  4.2"

START client1t   telnet 917
START client2t    telnet 917
START client4t   telnet 916

#define TES_FLAG_TRACE         0x00000010
#define TES_FLAG_KEYSTROKE_TIME 0x00000200
#define TES_FLAG_LOCAL_LOG      0x00000400
#define TES_FLAG_LOCAL_TRACE    0x00000800

#if 0
SETFLAG ALL TES_FLAG_LOCAL_TRACE
SETFLAG ALL TES_FLAG_TRACE
#endif

#if 0
/* SETFLAG client1tx telnet 1 TES_FLAG_KEYSTROKE_TIME */
/* SETFLAG ALL TES_FLAG_TRACE */
/* SETFLAG client1tx telnet 1 TES_FLAG_TRACE */
#endif
```

E.2 Master Script

```
#define _H_CUR01
#include <cur00.h>
#undef _H_CUR01
extern "C" {
#include "data/cur01.h"
int wrefresh (WINDOW *);
int wclrtoeol(WINDOW *);
int setupterm(char*,FILE*,int*);
int nodelay(int);
int keypad(int);
int wgetch(WINDOW *);
}
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include "data/rte.h"
#include "data/Stats.h"
#include "data/misc.h"
#include "user_tpcc.h"

struct header_s {
    int slave;
    int num;
    int type;
    int num_timestamps;
    int user_data_length;
    int data_type;
};


char *get_variable(char *name);
int get_variable(char *name, int *number);
int send_global_data(void);
int make_ratios (double *buffer);
extern int ramp_up_complete;
extern int interval_start_time, interval_stop_time;
extern "C" int strcasecmp(char *s1, char *s2);
extern "C" int strncasecmp(char *s1, char *s2, int n);

struct UserSpawnData {
    int Warehouse;
    int District;
};

/* user_master.C */
int user_statistics_print(void);
int user_spawn(int *length, char *buffer);
int user_finished(int length, char *buffer);

SlaveStatus slave_status[MAX_SLAVES];

extern Stats status[MAX_TRAN_TYPE][MAX_TIMES];
extern WINDOW *statistics_win;
extern UserGlobal *shmglobal;

/* Transaction mix parameters */
double ratio_desired[6], ratio_min[6], ratio_max[6], ratio_range[6];
char *ratio_names[] = { "RTE", "NEWORDER", "PAYMENT", "ORDSTAT",
"DELIVERY",
                "STOCKLEV", NULL };
char *Status_Names[] = {"Menu", "Keying", "Response", "Think"};


char *transaction_names[] = { "RTE", "New Order", "Payment", "Order Stat",
                    "Delivery", "Stock Level", NULL };

static int current_status = 2, status_needs_refresh = 1;

int user_statistics_print(void) {
    int i;
    static int count = 0;
    double ratios[6];
    if (status_needs_refresh) {
                count = 0;
                status_needs_refresh = 0;
                wmove   (statistics_win, 0, 0);
                wprintw (statistics_win, "%11s %8s %8s %8s %8s %8s %6s %6s
%6s",
                                Status_Names[current_status], "90%", "Avg", "Min",
"Max",
                                "Samples", "Ratio", "Mix", "Think");
    }
    make_ratios(ratios);


    for (i = 1; i <= 5; i++) {
                /* The reason we do this is because calculating the percentiles
                   is expensive */
                if (count % 10 == 0) {
                    wmove   (statistics_win, i, 0);
                    wprintw (statistics_win, "%11s %8.2f",
                                    transaction_names[i],
status[i][current_status].ninety()/1000.0);
                    count = 0;
                }
                wmove   (statistics_win, i, 21);
                wprintw (statistics_win, "%8.2f %8.2f %8.2f %8d %6.2f %6.2f %6.2f",
                                status[i][current_status].average()/1000.0,
                                status[i][current_status].min()/1000.0,
                                status[i][current_status].max()/1000.0,
                                status[i][current_status].samples(),
                                ratios[i], shmglobal->chances[i],
                                status[i][3].average()/1000.0);
    }
    wmove   (statistics_win, 7, 0);

    extern int runtime_counts[MAX_TRAN_TYPE];
    extern int begin_time, ramp_up, run_time;
    int start = interval_start_time;
    int stop  = interval_stop_time;
    double interval = ((double)(stop-start) / (1000*60));
```

```
    double samples = status[1][2].samples();
    if (interval <= 0 || samples <= 0) {
                wprintw (statistics_win, "TPM-C: %7s / ", "-------");
    } else {
                wprintw (statistics_win, "TPM-C: %7.2f / ", samples/interval);
    }
    samples = runtime_counts[1];
    if (samples > 0) {
                start = begin_time+((ramp_up>=0)?ramp_up:0);
                if (run_time > 0 && stop > begin_time + ramp_up + run_time) {
                    stop = begin_time + ramp_up + run_time;
                }
                interval = (double)(stop - start)/(1000.0*60.0);
                wprintw (statistics_win, "%7.2f", samples/interval);
    } else {
                wprintw (statistics_win, "-------");
    }

    count++;
    return RTE_OK;
}

const int MAX_WAREHOUSES=1000;
/* All of this 10 stuff is district size.  Should be a constant.
   Maybe fix that later */
int num_warehouses = -1;
int warehouses[MAX_WAREHOUSES*10];
int user_spawn(int number, int *length, char *buffer) {
    int i, min_index;
    UserSpawnData *ptr = (UserSpawnData *)buffer;
    *length = sizeof(*ptr);
    /* Count up number of warehouses we WANT to have */
    if (num_warehouses < 0) {
                num_warehouses = 0;
                for (i = 0; i < MAX_SLAVES; i++) {
                    num_warehouses += slave_status[i].desired;
                }
                num_warehouses = (num_warehouses-1)/10+1;
    }

    min_index = 0;
    for (i = 1; i <= (num_warehouses)*10 && i < MAX_WAREHOUSES; i++) {
                if (warehouses[i] < warehouses[min_index]) {
                    min_index = i;
                }
    }

    ptr->Warehouse = min_index / 10 + 1;
    ptr->District  = min_index % 10 + 1;
    warehouses[min_index]++;
/*    iprint (IPRINT_INFO, "Driver for Warehouse %d, District %d started.
warehouses[%d]++ = %d\n",
          ptr->Warehouse, ptr->District, min_index, warehouses[min_index]); */
    return RTE_OK;
}

int user_finished(int length, char *buffer) {
    UserSpawnData *ptr = (UserSpawnData *)buffer;
    int temp = (ptr->Warehouse-1)*10+ptr->District-1;
    warehouses[temp]--;
/*    iprint (IPRINT_INFO, "Driver for Warehouse %d, District %d died.
warehouses[%d]-- = %d\n",
          ptr->Warehouse, ptr->District, temp, warehouses[temp]); */
    return RTE_OK;
}


double limit(double min, double max, double val) {
    if (val < min)
                return min;
    if (val > max)
                return max;
    return val;
}


int make_ratios (double *buffer) {
    int neword   = status[NEWORDER][0].samples();
    int payment  = status[PAYMENT] [0].samples();
    int ordstat  = status[ORDSTAT] [0].samples();
    int delivery = status[DELIVERY][0].samples();
    int stocklev = status[STOCKLEV][0].samples();
    int total    = neword + payment + ordstat + delivery + stocklev;
    int i;

    if (total == 0) {
                buffer[NEWORDER] = 100.0;
                for (i = 2; i < 6; i++) {
                    buffer[i] = ratio_desired[i];
                    buffer[NEWORDER] -= buffer[i];
                }
                return 0;
    }

    buffer[PAYMENT]  = (double)payment  / (double)total * 100.0;
    buffer[ORDSTAT]  = (double)ordstat  / (double)total * 100.0;
    buffer[DELIVERY] = (double)delivery / (double)total * 100.0;
    buffer[STOCKLEV] = (double)stocklev / (double)total * 100.0;
    buffer[NEWORDER] = 100.0- buffer[PAYMENT] - buffer[ORDSTAT] -
                                                buffer[DELIVERY] - buffer[STOCKLEV];

    return total;
}

int user_global_update(int *length, char *buffer) {
    UserGlobal *shmglobal = (UserGlobal *)buffer;
    static double last[6];
    static int users_last=-1;
    double ratios[6];
    double current[6];
    int i, different = 0;

    *length = sizeof(*shmglobal);

    make_ratios(ratios);

    /* Calculate ratios we want for next time */
    /* Note: we just keep on with the desired values until ramp-up is complete
          this at least starts us out without any humps or spikes in the
                graph */
    if (ramp_up_complete) {
                current[NEWORDER] = 100.0;
                for (i = 2; i < 6; i++) {
                    if (ratio_desired[i] > ratios[i]) {
                                current[i] = ratio_max[i];
                    } else {
                                current[i] = 2*ratio_desired[i] - ratios[i];
                                if (current[i] < ratio_min[i])
                                    current[i] = ratio_min[i];
                    }
                    current[NEWORDER] -= current[i];
                }
    } else {
                for (i = 1; i < 6; i++) {
                    current[i] = ratio_desired[i];
                }
    }

    /* Add up all the users */
    /* This needs to be changed to be more transparent */
    shmglobal->total_users = 0;
    for (i = 0; i < MAX_SLAVES; i++) {
                shmglobal->total_users += slave_status[i].active;
    }


    for (i = 2; i < 6; i++) {
                if (current[i] != last[i])
                    different = 1;
    }

    // Don't send if it's the same as last time
    if ( !different && shmglobal->total_users == users_last ) {
                return RTE_ERROR;
    }

    users_last = shmglobal->total_users;
    for (i = 1; i < 6; i++) {
                shmglobal->chances[i] = last[i] = current[i];
    }

    send_global_data();

    return RTE_OK;
}


int parse_array(char *string, int max, int *buffer) {
    int i, rc;
    char *ptr;
    char *temp = strdup(string);
    ptr = strtok(temp, ",");
    for (i = 0; ptr && i < max; i++) {
                rc = sscanf(ptr, "%d", &buffer[i]);
                if (rc < 1) {
                    free(temp);
                    return i;
                }
                ptr = strtok(NULL, ",");
    }
    free(temp);
    return i;
}

int parse_array(char *string, int max, double *buffer) {
    int i, rc;
```

```
    char *ptr;
    char *temp = strdup(string);
    ptr = strtok(temp, ",");
    for (i = 0; ptr && i < max; i++) {
                rc = sscanf(ptr, "%lf", &buffer[i]);
                if (rc < 1) {
                    free(temp);
                    return i;
                }
                ptr = strtok(NULL, ",");
    }
    free(temp);
    return i;
}


int user_init() {
    double  dbuffer[32];
    int   rc, i;
    char *ptr;

    if (get_variable("KEYSTROKE_SLEEP", &shmglobal->keystroke_sleep) !=
RTE_OK) {
                shmglobal->keystroke_sleep = 0;
    }
    if (get_variable("LOGIN_TIMEOUT", &shmglobal->login_timeout) != RTE_OK) {
                shmglobal->login_timeout = 120; /* 2 minutes */
    }
    if (get_variable("KEYSTROKE_PACKET_SIZE",
&shmglobal->keystroke_packet_size) != RTE_OK) {
                shmglobal->keystroke_packet_size = 0;
    }
    shmglobal->login_timeout *= 1000;
    if (get_variable("WAREHOUSES", &num_warehouses) != RTE_OK) {
                num_warehouses = -1;
    }
    iprint(IPRINT_INFO, "Login Timeout = %s\n", mstoa(shmglobal->login_timeout,
0));
    iprint(IPRINT_INFO, "Keystroke Sleep = %s\n",
mstoa(shmglobal->keystroke_sleep*1000, 0));
    iprint(IPRINT_INFO, "Keystroke Packet Size= %d\n",
shmglobal->keystroke_packet_size);

    if (!(ptr = get_variable("NEWORDER"))) {
                iprint_error ("Error. NEWORDER variable not found\n");
                exit (1);
    }
    if (parse_array(ptr, 3, dbuffer)!=3) {
                iprint_error ("Error. NEWORDER should be think, emulex_menu,
emulex_response");
                exit (1);
    }
    shmglobal->think         [NEWORDER] = dbuffer[0];
    shmglobal->emulex_menu    [NEWORDER] = dbuffer[1];
    shmglobal->emulex_response[NEWORDER] = dbuffer[1];

    for (i = 2; i < 6; i++) {
                if (!(ptr = get_variable(ratio_names[i])) ||
                    (parse_array(ptr, 6, dbuffer)!=6)) {
                    iprint(__FILE__, __LINE__, IPRINT_ERROR,
                          "Error. %s should be think, emulex_delay, desired, min, max",
                                ratio_names[i]);
                    exit (1);
                }
                shmglobal->think[i]          = dbuffer[0];
                shmglobal->emulex_menu[i]     = dbuffer[1];
                shmglobal->emulex_response[i] = dbuffer[2];
                ratio_desired[i]             = dbuffer[3];
                ratio_min[i]                 = dbuffer[4];
                ratio_max[i]                 = dbuffer[5];
                ratio_range[i]               = ratio_max[i]-ratio_min[i];
    }

    return RTE_OK;
}

int user_extra_data(header_s *header) {
    int i;
    int num_timestamps;

    if (header->data_type != RTE_ITEM_KEYSTROKE_TIMES)
                return RTE_OK;
    int *times = (int *)((char *)header+sizeof(struct header_s));
    num_timestamps = header->user_data_length / 4 - 1;

    iprint (IPRINT_TRACE, "Keystroke times = ");
    for (i = 0 ; i < num_timestamps; i++) {
                iprint (IPRINT_TRACE, "%d ", times[i]);
    }
    iprint (IPRINT_TRACE, "\n", times[i]);

    return RTE_OK;
}
```

```
int user_process_command(char *command) {
    char buffer[256], *ptr;
    int i, found, len;
    strncpy (buffer, command, 256);
    ptr = strtok (buffer, " \t");
    found = 0;
    if (!strcasecmp (ptr, "display")) {
        while (ptr && (ptr = strtok(NULL, " \t"))) {
            if (*ptr == '\0')
                continue;
            for (i = 0; i < 5; i++) {
                len = min(strlen(Status_Names[i]), strlen(ptr));
                if (!strncasecmp (ptr, Status_Names[i], len)) {
                    status_needs_refresh = found = 1;
                    current_status = i;
                    return RTE_OK;
                }
            }
            printf ("Unknown type to display: %s\n", ptr);
        }
    }
    printf ("Unknown Command: %s\n", command);
    return RTE_ERROR;
}


int user_begin() {
    return RTE_OK;
}

void user_make_header(char *buffer) {
    int i;
    struct user_data_header *data = (struct user_data_header *)buffer;
}
```

E.3 User Script

```
/*****************************************************************************/
/***                TPCC FILE FOR ALL USERS                                ***/
/*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/time.h>
#include "rte_slave.h"
#include "user_tpcc.h"

extern "C" {
#define neword_struct tux_neword_struct
#define stocklev_struct tux_stocklev_struct
#define delivery_struct tux_delivery_struct
#define ordstat_struct tux_ordstat_struct
#define payment_struct tux_payment_struct
#include "tuxedo_tpcc.h"
#undef neword_struct
#undef stocklev_struct
#undef delivery_struct
#undef ordstat_struct
#undef payment_struct
#include "atmi.h"

#define TMINBUFSIZE      1536
int              transtatus;
char             *TMbuffer;
long             TMinbufsize = TMINBUFSIZE, TMoutbufsize;
}
int submit_TM_atran(char *servname);
int submit_TM_tran(char *servname);

extern SHM_Slave *shm;
extern TableEntrySlave *shmentry;
extern DriverStatus    *status;
extern echo_trace(char *);
extern echo_trace();
extern char *expect_save;

const char *SQL_TPERRNO_MESSAGE = "tperrno";
const char *SQL_RTN_MESSAGE =     "rtn:";
const char *SQL_FATAL_MESSAGE =   "SQL Fatal Error";
const char *ROLLBACK_MESSAGE =    "Item number is not valid";

int           WHSEID;                       /* warehouse number for each users */
int           DISTRICTID;                   /* warehouse number for each users */

/*****************************************************************************/
```

```
/* The "uniform()" function has range of the absolute value of the        */
/* difference between the min. and the max values upto 2147483647.      */
/*************************************************************************/
/*-------------------------------------------------------*/
/* NURand                                                */
/*-------------------------------------------------------*/
/* A: 255 for C_LAST, 1023 for C_ID, 8191 for OL_I_ID   */
/* x: 0 for C_LAST, 1 for C_ID and OL_I_ID              */
/* y: 999 for C_LAST, 3000 for C_ID, 100000 for OL_I_ID */
/*-------------------------------------------------------*/
long
NURand(int A, int x, int y, long cval)
{
    return (((uniform(0, A) | uniform(x, y)) + cval) % (y - x + 1)) + x;
}

/*-------------------------------------------------------*/
/* getname                                               */
/*-------------------------------------------------------*/
/* generates a random number from 0 to 999 inclusive    */
/* a random name is generated by associating a random   */
/* string with each digit of the generated number       */
/* three strings are concatenated to generate lastname  */
/*-------------------------------------------------------*/
char            *
getname()
{
    char          *last_name_parts[] =
    {
     "BAR",
     "OUGHT",
     "ABLE",
     "PRI",
     "PRES",
     "ESE",
     "ANTI",
     "CALLY",
     "ATION",
     "EING"
    };
    char           lastname[128];
    int            random_num;

    random_num = NURand(255, 0, 999, LASTC);
    strcpy(lastname, last_name_parts[random_num / 100]);
    random_num %= 100;
    strcat(lastname, last_name_parts[random_num / 10]);
    random_num %= 10;
    strcat(lastname, last_name_parts[random_num]);
    return (lastname);
}


/*************************************************************************/
/***                  Delivery Transaction                           ***/
/*************************************************************************/
int
Delivery()
{
    static struct delivery_struct delivery, delivery_new;
    char           carrier[32];
    int            resp, charlen;
    double         think_max, think_DEL;
    char          const *ptr;
    struct tux_delivery_struct *tmbuf;
    int tuxstat;
    char servname[30];

    delivery_new.invalid = 0;

    /* Calculate what we are going to enter next time */
    delivery_new.carrier = uniform(1, 10); /* carrier # 1 to 10 */

    set_typing_delay(0);

    tmbuf = (struct tux_delivery_struct *)TMbuffer;
    tmbuf->s_O_CARRIER_ID = delivery_new.carrier;
    tmbuf->s_W_ID         = WHSEID;
    tuxstat = submit_TM_atran("delivery_new_DSQL");

    transaction_sleep_do();
    transaction_start(DELIVERY, sizeof(delivery), &delivery);           /* logging */
    delivery = delivery_new;
    echo_trace ("Waiting for Menu (DELIVERY)");
    usleep(shmglobal->emulex_menu[DELIVERY]*1000000.0+0.9);
    transaction_mark(WHERE_NOW);    /* end of menu, beginning of keying
                                                                        * time */
    echo_trace ("Keying");
    sleep (2);
    transaction_mark(WHERE_NOW);    /* end of keying, beginning of
                                                                        * response time
*/
    echo_trace ("Wait for Response");
    usleep(shmglobal->emulex_response[DELIVERY]*1000000.0+0.9);
    transaction_mark(WHERE_NOW);    /* end of response, beginning of
                                                                        * think time */
    if (tuxstat != 0) {
                delivery.invalid = 1;
                iprint (IPRINT_ERROR, "Slave %d: Delivery tuxedo error (%d)\n",
                        shmentry->num, tuxstat);
                return RTE_ERROR;
    }
    echo_trace ("Thinking");
    transaction_sleep_set(neg_exp_4(shmglobal->think[DELIVERY])*1000.0);
    return (RTE_OK);
}

/*************************************************************************/
/***                  New Order Transaction                          ***/
/*************************************************************************/
int
NewOrder()
{
    static struct neword_struct neword, neword_new;
    int            RBtrans, maxwhses, remoteflag, i, resp, charlen;
    double          think_max, think_NO;
    char           buff[1024], district[32], customer[32], warehouse[32], item[32],
quantity[32];
    char          const *ptr;
    struct tux_neword_struct *tmbuf;
    int tuxstat;
    char servname[30];


    neword_new.rollback=0;
    neword_new.invalid = 0;

    /*** SECTION TO DETERMINE ROLLBACK TRANSACTION FOR 1% OF NEW
ORDERS ***/
    RBtrans = 0;
    if (uniform(1, 5000) <= 50)
                RBtrans = 1;

    neword_new.did = uniform(1, 10);      /* district number */
    neword_new.cid = NURand(1023, 1, 3000, CUSTC);                  /* customer # 1 to
3000 */

    if (shmglobal->total_users < 10)
                maxwhses = 1;
    else
                maxwhses = (shmglobal->total_users-1) / 10+1;       /* max number of
warehouses */

    neword_new.nloop = uniform(5, 15);  /* for 5 to 15 iterations */
    remoteflag = 0;                             /* for remote orders */
    neword_new.olremote=0;                  /* find total number of remote order-lines */

    for (i = 0; i < neword_new.nloop; i++) {
                neword_new.item[i].olswid = WHSEID;
                if (maxwhses > 1 && (uniform(0.0, 100.0) < 1.0)) {
                    /* for 1% of items (if * uniform()==0) */
                    /* Generate a uniform whse no. */
                    while (neword_new.item[i].olswid == WHSEID)
                                /* different from WHSEID * no. */
                                neword_new.item[i].olswid = uniform(1, maxwhses);
                    neword_new.olremote++;                  /* find total number of remote
order-lines */
                    remoteflag = 1;
                }

                if (neword_new.nloop == i+1 && RBtrans)                 /* If last iteration
of loop */
                    neword_new.item[i].oliid = 999999;    /* and a RollBack
                                                                                *
transaction. */
                else {
                    neword_new.item[i].oliid = NURand(8191, 1, 100000, ITEMC);    /*
item 1 to 100,000 */
                }
                neword_new.item[i].olquantity = uniform(1, 10);         /* quantity 1 to 10
*/
    }                                                           /* end of for n_loop */

    if (remoteflag == 1)        /* 1 for remote orders */
                neword_new.oremote = 1;
    else
                neword_new.oremote = 0;

    tmbuf = (struct tux_neword_struct *)TMbuffer;
    tmbuf->s_W_ID = WHSEID;
    tmbuf->s_D_ID = neword_new.did;
    tmbuf->s_C_ID = neword_new.cid;

    tmbuf->s_O_OL_CNT  =  neword_new.nloop;
    tmbuf->s_all_local = !neword_new.olremote;
```

```
    for (i=0; i < tmbuf->s_O_OL_CNT; i++) {
                tmbuf->item[i].s_OL_SUPPLY_W_ID = neword_new.item[i].olswid;
                tmbuf->item[i].s_OL_I_ID       = neword_new.item[i].oliid;
                tmbuf->item[i].s_OL_QUANTITY    = neword_new.item[i].olquantity;
    }

    transaction_sleep_do();
    transaction_start(NEWORDER, sizeof(neword), &neword);         /* logging */
    neword = neword_new;

    echo_trace ("Waiting for Menu (NEWORDER)");
    usleep(shmglobal->emulex_menu[NEWORDER]*1000000.0+0.9);
    transaction_mark(WHERE_NOW);    /* end of menu, beginning of keying
                                                                    * time */
    echo_trace ("Keying");
    transaction_mark(WHERE_NOW);    /* end of keying, beginning of
                                                                    * response time
*/

    sprintf(servname, "NEWORD_IBM");
    tuxstat = submit_TM_tran(servname);
    echo_trace ("Wait for Response");
    usleep(shmglobal->emulex_response[NEWORDER]*1000000.0+0.9);
    transaction_mark(WHERE_NOW);    /* end of response, beginning of
                                                                    * think time */
    if (tuxstat != 0) {
                neword.invalid = 1;
                iprint (IPRINT_ERROR, "Slave %d: NewOrd tuxedo error (%d)\n",
                            shmentry->num, tuxstat);
                return RTE_ERROR;
    }
    tmbuf = (struct tux_neword_struct *)TMbuffer;
    /* Grab order id from return string */
    neword.oid = tmbuf->s_O_ID;
    if (RBtrans) {
                neword.rollback=1;
    }
    echo_trace ("Thinking");
    transaction_sleep_set(neg_exp_4(shmglobal->think[NEWORDER])*1000.0);

    return (RTE_OK);
}

/*****************************************************************************/
/***                Order Status Transaction                           ***/
/*****************************************************************************/
int
OrderStatus()
{
    static struct ordstat_struct ordstat, ordstat_new;
    char            buff[1024], district[32], customer[32];
    int             resp, charlen;
    double          think_max, think_OS;
    char           const *ptr;
    struct tux_ordstat_struct *tmbuf;
    int tuxstat;
    char servname[30];

    ordstat_new.invalid = 0;

    ordstat_new.did = uniform(1, 10);       /* district number 1 to 10 */
    if (uniform(1, 100) <= 60) {/* for 60% of transactions */
                strcpy(ordstat_new.clast, getname());    /* by customer last name */
                if (ordstat_new.clast[0] < 'A' || ordstat_new.clast[0] > 'Z') {
                    iprint (IPRINT_ERROR, "ASSERTION: OrderStatus getname()
returns invalid name! '%s'\n", ordstat_new.clast);
                }
                ordstat_new.byname = 1;
                ordstat_new.cid = 0;
    } else {
                ordstat_new.cid = NURand(1023, 1, 3000, CUSTC);   /* cust. # 1 to
3000 */
                ordstat_new.byname = 0;
                ordstat_new.clast[0] = (char) NULL;
    }

    tmbuf = (struct tux_ordstat_struct *)TMbuffer;

    tmbuf->s_W_ID = WHSEID;
    tmbuf->s_D_ID = ordstat_new.did;
    if (ordstat_new.byname) {
                tmbuf->s_C_ID = 0;
                strcpy(tmbuf->s_C_LAST, ordstat_new.clast);
    } else {
                tmbuf->s_C_ID      = ordstat_new.cid;
                tmbuf->s_C_LAST[0] = (char)NULL;
    }

    set_typing_delay(0);
    transaction_sleep_do();
    transaction_start (ORDSTAT, sizeof(ordstat), &ordstat);             /* logging */
    ordstat = ordstat_new;
    echo_trace ("Waiting for Menu (ORDSTAT)");
    usleep(shmglobal->emulex_menu[ORDSTAT]*1000000.0+0.9);
    transaction_mark(WHERE_NOW);    /* end of menu, beginning of keying
                                                                    * time */
    echo_trace ("Keying");
    sleep (2);

    transaction_mark(WHERE_NOW);    /* end of keying, beginning of
                                                                    * response time
*/
    echo_trace ("Wait for Response");
    tuxstat = submit_TM_tran("ORDSTAT_DSQL");
    usleep(shmglobal->emulex_response[ORDSTAT]*1000000.0+0.9);
    transaction_mark(WHERE_NOW);    /* end of response, beginning of
                                                                    * think time */
    if (tuxstat != 0) {
                ordstat.invalid = 1;
                iprint (IPRINT_ERROR, "Slave %d: OrdStat tuxedo error (%d)\n",
                            shmentry->num, tuxstat);
                return RTE_ERROR;
    }
    echo_trace ("Thinking");
    transaction_sleep_set(neg_exp_4(shmglobal->think[ORDSTAT])*1000.0);
    return (RTE_OK);
}

/*****************************************************************************/
/***                Payment Transaction                                ***/
/*****************************************************************************/
int
Payment()
{
    static struct payment_struct payment, payment_new;
    int             maxwhses, dollars, cents;
    int             resp, charlen;
    double          think_max, think_PAY;
    char            buff[1024], district[32], customer[32], waredist[32], amount[32];
    char           const *ptr;
    struct tux_payment_struct *tmbuf;
    int tuxstat;
    char servname[30];

    payment_new.invalid = 0;
    if (shmglobal->total_users < 10)
                maxwhses = 1;
    else
                maxwhses = (shmglobal->total_users-1) / 10+1;         /* max number of
warehouses */

    payment_new.did = uniform(1, 10);    /* district number 1 to 10 */

    if (uniform(1, 100) <= 60) {/* for 60% of transactions */
                strncpy(payment_new.clast, getname(), 17);              /* by customer
last name */
                if (payment_new.clast[0] < 'A' || payment_new.clast[0] > 'Z') {
                    iprint (IPRINT_ERROR, "ASSERTION: payment_new getname()
returns invalid name! '%s'\n", payment_new.clast);
                    exit (10);
                }
                payment_new.byname = 1;
                payment_new.cid = 0;
    } else {
                payment_new.cid = NURand(1023, 1, 3000, CUSTC);/* cust. # 1 to
3000 */
                payment_new.byname = 0;
                payment_new.clast[0] = (char) NULL;
    }

    if (maxwhses < 2 || uniform(1, 100) <= 85) {/* for 85 % of transactions */
                payment_new.cwid = WHSEID;
                payment_new.cdid = payment_new.did;
                payment_new.remote = 0;
    } else {                                    /* for 15 % of transactions */
                payment_new.cwid = WHSEID;
                while (payment_new.cwid == WHSEID)
                    payment_new.cwid = uniform(1, maxwhses);         /* warehouse 1 to
max

 * whses */
                payment_new.remote = 1;
                payment_new.cdid = uniform(1, 10);      /* district 1 to 10 */
    }

    dollars = uniform(1, 5000);                 /* dollar amt = 1 to 5000 */
    if (dollars == 5000)
                cents = 0;
    else
                cents = uniform(0, 99);
    payment_new.amount = ((double) dollars) + ((double) cents) / 100.0;

    tmbuf = (struct tux_payment_struct *)TMbuffer;

    tmbuf->s_W_ID = WHSEID;
```

```
    tmbuf->s_D_ID = payment_new.did;
    tmbuf->s_C_W_ID = payment_new.cwid;
    tmbuf->s_C_D_ID = payment_new.cdid;
    tmbuf->s_H_AMOUNT = 1.00;

    if (payment_new.byname) {
                tmbuf->s_C_ID = 0;
                strcpy(tmbuf->s_C_LAST, payment_new.clast);
    } else {
                tmbuf->s_C_ID = payment_new.cid;
                tmbuf->s_C_LAST[0] = (char)NULL;
    }

    transaction_sleep_do();
    transaction_start(PAYMENT, sizeof(payment), &payment); /* logging */
    payment = payment_new;
    echo_trace ("Waiting for Menu (PAYMENT)");
    usleep(shmglobal->emulex_menu[PAYMENT]*1000000.0+0.9);
    transaction_mark(WHERE_NOW);    /* end of menu, beginning of keying
                                                                    * time */
    echo_trace ("Keying");
    sleep (3);

    transaction_mark(WHERE_NOW);    /* end of keying, beginning of
                                                                    * response time
*/
    echo_trace ("Wait for Response");
    tuxstat = submit_TM_tran("PAYMENT_DSQL");

    usleep(shmglobal->emulex_response[PAYMENT]*1000000.0+0.9);
    transaction_mark(WHERE_NOW);    /* end of response, beginning of
                                                                    * think time */
    if (tuxstat != 0) {
                payment.invalid = 1;
                iprint (IPRINT_ERROR, "Slave %d: Payment tuxedo error (%d)\n",
                        shmentry->num, tuxstat);
                return RTE_ERROR;
    }
    echo_trace ("Thinking");
    transaction_sleep_set(neg_exp_4(shmglobal->think[PAYMENT])*1000.0);
    return (RTE_OK);
}

/*****************************************************************************/
/***                 Stock Level Transaction                           ***/
/*****************************************************************************/
int
StockLevel()
{
    static struct stocklev_struct stocklevel, stocklevel_new;
    char            threshold[32];
    int             resp, charlen;
    double          think_max, think_SL;
    char            const *ptr;
    struct tux_stocklev_struct *tmbuf;
    int tuxstat;
    char servname[30];

    stocklevel_new.invalid = 0;
    stocklevel_new.threshold = uniform(10, 20);          /* uniform no. between 10 and
                                                                                *
20 */

    tmbuf = (struct tux_stocklev_struct *)TMbuffer;

    tmbuf->s_W_ID = WHSEID;
    tmbuf->s_D_ID = DISTRICTID;
    tmbuf->s_threshold = stocklevel_new.threshold;

    transaction_sleep_do();
    transaction_start(STOCKLEV, sizeof(stocklevel), &stocklevel); /* logging */
    stocklevel = stocklevel_new;
    echo_trace ("Waiting for Menu (STOCKLEV)");
    usleep(shmglobal->emulex_menu[STOCKLEV]*1000000.0+0.9);
    transaction_mark(WHERE_NOW);    /* end of menu, beginning of keying
                                                                    * time */
    echo_trace ("Keying");
    sleep (2);

    transaction_mark(WHERE_NOW);    /* end of keying, beginning of
                                                                    * response time
*/
    echo_trace ("Wait for Response");
    tuxstat = submit_TM_tran("STOCKLEV_DSQL");

    usleep(shmglobal->emulex_response[STOCKLEV]*1000000.0+0.9);
    transaction_mark(WHERE_NOW);    /* end of response, beginning of
                                                                    * think time */
    if (tuxstat != 0) {
                stocklevel.invalid = 1;
                iprint (IPRINT_ERROR, "Slave %d: Stock Level tuxedo error (%d)\n",
                        shmentry->num, tuxstat);
                return RTE_ERROR;
    }
    echo_trace ("Thinking");
    transaction_sleep_set(neg_exp_4(shmglobal->think[STOCKLEV])*1000.0);
    return (RTE_OK);
}

/*****************************************************************************/
/***  MAIN()                                                               ***/
/*****************************************************************************/
int
user_transaction()
{
    char            logout[32];
    double          ntask;
    int             resp;

    if (shmentry->flags & TES_FLAG_KEYSTROKE_TIME) {
                int rc;
                /* Wait for specified period of time */
                sleep (shmglobal->keystroke_sleep);
                /* Quit after one transaction */
                shm->lock(shmentry->pid);
                shmentry->flags |= TES_FLAG_DIE;
                shm->unlock(shmentry->pid);
                rc = NewOrder();
                iprint (IPRINT_INFO, "Slave %d: Keystroke timing setting die flag\n",
shmentry->num);
                return rc;
    }
    /*****************************************************************************/
    /***  CHOOSE ONE OF THE TRANSACTIONS                                       ***/
    /*****************************************************************************/
    ntask = (double) uniform(0.0, 100.0);
    if (ntask <= shmglobal->chances[DELIVERY])
                return Delivery();
    ntask -= shmglobal->chances[DELIVERY];
    if (ntask <= shmglobal->chances[ORDSTAT])
                return OrderStatus();
    ntask -= shmglobal->chances[ORDSTAT];
    if (ntask <= shmglobal->chances[PAYMENT])
                return Payment();
    ntask -= shmglobal->chances[PAYMENT];
    if (ntask <= shmglobal->chances[STOCKLEV])
                return StockLevel();
    return NewOrder();

#if 0
    if (resp != RTE_OK) {                   /* logoff if response is not correct */
                strcpy(logout, "9\r");          /* menu option 9 */
                transmit(logout);
                resp = expect("tpcc_cstux_inf:");
                return (ERROR);
    } else
                return (RTE_OK);
#endif
}                                                               /* end of main */

int user_parameter_change(void) {
    #if 0
    int i;
    iprint(IPRINT_TRACE, "Slave %d: total_users = %d\n", shmentry->num);
    iprint(IPRINT_TRACE, "Slave %d: chances     = ", shmentry->num);
    for (i = 0; i < MAX_TRAN_TYPE; i++)
                iprint(IPRINT_TRACE, "%6.2f ", shmglobal->chances[i]);
    iprint(IPRINT_TRACE, "\nSlave %d: think       = ", shmentry->num);
    for (i = 0; i < MAX_TRAN_TYPE; i++)
                iprint(IPRINT_TRACE, "%6.2f ", shmglobal->think[i]);
    iprint(IPRINT_TRACE, "\n");
    #endif
    return RTE_OK;
}

const int MAX_LOGIN_RETRY = 500;
int user_login(char *user, char *password, void *data) {
    extern char ttyname_original[TTYNAME_LEN];
    UserLocal *localdata = (UserLocal *)data;
    int rc;
    int retry = 0;
    int timeout_value = shmglobal->login_timeout;
    char buffer[256];

    if (tpinit((TPINIT *) NULL) == -1) {
                iprint (IPRINT_ERROR, "tpinit(): tperrno = %d\n", tperrno);
                exit (1);
    }
    if ((TMbuffer = tpalloc("CARRAY", NULL, TMinbufsize)) == NULL) {
                iprint(IPRINT_ERROR, "tpalloc(): tperrno: %d\n", tperrno);
                exit(2);
    }

    return RTE_OK;
}
```

```
int user_init () {
    extern int expect_save_active;
    WHSEID = shmlocal->Warehouse;
    DISTRICTID = shmlocal->District;
    status->max_transmit =  shmglobal->keystroke_packet_size;
    expect_save_active = 1;
    return RTE_OK;
}

int user_cleanup () {
    transaction_sleep_do();
    transaction_start(0, 0, NULL); // Just something to clear out the buffer...
    if (tpterm() == -1) {
                iprint(IPRINT_ERROR, "tpterm(): tperrno: %d\n", tperrno);
                exit(3);
    }
    return RTE_OK;
}

/*-----------------------------------------------------------*/
/*              submit TM tran
                */
/*-----------------------------------------------------------*/
int submit_TM_tran(char *servname) {
    transtatus = tpcall(servname, TMbuffer, TMinbufsize, &TMbuffer,
                                        &TMoutbufsize, 0);

    if (transtatus == -1) {
                iprint(IPRINT_ERROR, "tpcall(): tperrno: %d, transtatus: %d\n",
                            tperrno, transtatus);
                return (-1);
    }
    return (0);
}

/*-----------------------------------------------------------*/
/*              submit TM asynchronous transaction                    */
/*-----------------------------------------------------------*/
int submit_TM_atran(char *servname) {
    transtatus = tpacall(servname, TMbuffer, TMinbufsize, TPNOREPLY);
    if (transtatus == -1) {
                iprint(IPRINT_ERROR, "tpacall(): tperrno: %d, transtatus: %d\n",
                            tperrno, transtatus);
                return (-1);
    }
    return (0);
}



/*************************************************************************/
/*                                                                       */
/*                user_tpcc.h                                            */
/*                                                                       */
/*************************************************************************/
#ifndef USER_TPCC_H
#define USER_TPCC_H
/*************************************************************************/
/***   run-time constant for customer last name from 0 to 255,          ***/
/***   run-time constant for customer id from 0 to 1023,               ***/
/***   run-time constant for item id from 0 to 8191.                   ***/
/*************************************************************************/
#define       LASTC       117
#define       CUSTC       319
#define       ITEMC       3849

/*************************************************************************/
/***   response type
                ***/
/*************************************************************************/
/* #define OK              1  */
/* #define ERROR           -1 */

/*************************************************************************/
/***   transaction type
                ***/
/*************************************************************************/
#define       NEWORDER                1
#define       PAYMENT                 2
#define       ORDSTAT                 3
#define       DELIVERY    4
#define       STOCKLEV    5

/*************************************************************************/
/***   transaction structures
                ***/
/*************************************************************************/
struct neword_struct {
    char        invalid;      /* transaction completed sucessfully */
    char        did;
    long        cid;
    long        oid;      /* Order-ID returned from client */
    char        nloop;        /* number of order line, avg = 15 */
    char        oremote;    /* 1 for remote order, 10% */
    char        olremote;   /* number of remote order line, 1% */
    char        rollback;   /* actually saw rollback text on screen */
    struct items_struct {
                char        olswid;
                long        oliid;
                char        olquantity;
    } item[15];
};

struct payment_struct {
    char        invalid;      /* transaction completed sucessfully */
    char did;
    int cid;
    char cwid;
    char cdid;
    char clast[17];
    double amount;
    char byname;                                    /* 1 for by last name, 0 for by id
*/
    char remote;                                    /* 1 for remote wardhouse, 0
otherwise */
};

struct ordstat_struct {
    char        invalid;      /* transaction completed sucessfully */
    char did;
    int cid;
    char clast[17];
    char byname;                                    /* 1 for by last name, 0 for by id
*/
};

struct delivery_struct {
    char        invalid;      /* transaction completed sucessfully */
    char carrier;
};

struct stocklev_struct {
    char        invalid;      /* transaction completed sucessfully */
    char threshold;
};

struct generic_struct {
    char        invalid;      /* transaction completed sucessfully */
};

union transaction_info {
    char   invalid;
    struct generic_struct  generic;
    struct neword_struct   neword;
    struct payment_struct  payment;
    struct ordstat_struct  ordstat;
    struct delivery_struct delivery;
    struct stocklev_struct stocklev;
};

struct UserGlobal {
    int total_users;
    int keystroke_sleep;
    int login_timeout;
    int keystroke_packet_size;
    double chances[MAX_TRAN_TYPE];
    double think[MAX_TRAN_TYPE];
    double emulex_response[MAX_TRAN_TYPE];
    double emulex_menu    [MAX_TRAN_TYPE];
};

struct UserLocal {
    int Warehouse;
    int District;
};

struct user_data_header {
};

extern UserGlobal *shmglobal;
extern UserLocal  *shmlocal;

#endif
```