INTERGRAPH
COMPUTER SYSTEMS

TPC Benchmark™ C Full Disclosure Report

INTERGRAPH

IS-650

Using
Microsoft SQL Server v. 6.5 (SP3)
and
Microsoft Windows NTS v. 4.0 (SP1)

First Edition
July 1997

First Printing July, 1997

Intergraph Corporation believes that the information in this document is accurate as of the publication date. The information discussed in this document is subject to change without notice. Intergraph Corporation is not responsible for any inadvertent errors.

The pricing information is this document is believed to accurately reflect prices in effect of publication date; however, Intergraph Corporation provides no warranty on the pricing information in this document.

Trademarks

Intergraph® and the Intergraph logo are registered trademarks of Intergraph Corporation. InterServe™ is a trademark of Intergraph Corporation.

Pentium® and Pentium® Pro are trademarks of Intel Corporation.

Microsoft®, Windows®, MS-DOS®, and the Microsoft logo are registered trademarks of Microsoft Corporation. Windows NT™ is a trademark of Microsoft Corporation.

TPC Benchmark™ is a trademark of the Transaction Processing Performance Council.

Other brands and product names are trademarks of their respective owners.

Table of Contents

Figures

Tables

Abstract

This report documents Intergraph Corporation's compliance with the specifications of the TPC Benchmark C version 3.3 on the InterServe 650. The database software for the benchmark was Microsoft SQL Server 6.5 (SP3), and the operating system was Microsoft Windows NT Server 4.0 (SP1).

The benchmark was completed on July 30, 1997, and resulted in a score of 5307.23 tpmC, a price performance of $62.93/tpmC with an availability date of July 1997. The standard metrics of tpmC and $/tpmC are reported in accordance with the TPC Benchmark™ C standard.

Preface

According to the TPC Benchmark™ C Standard Specification, test sponsors are required to publish a full disclosure report in order to be compliant with the specification. This report documents Intergraph Corporation's compliance with the specifications of the TPC Benchmark™ C.

According to the *TPC Benchmark™ C Standard Specification*, 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. The purpose of TPC benchmarks is to provide relevant, objective performance data to industry users.

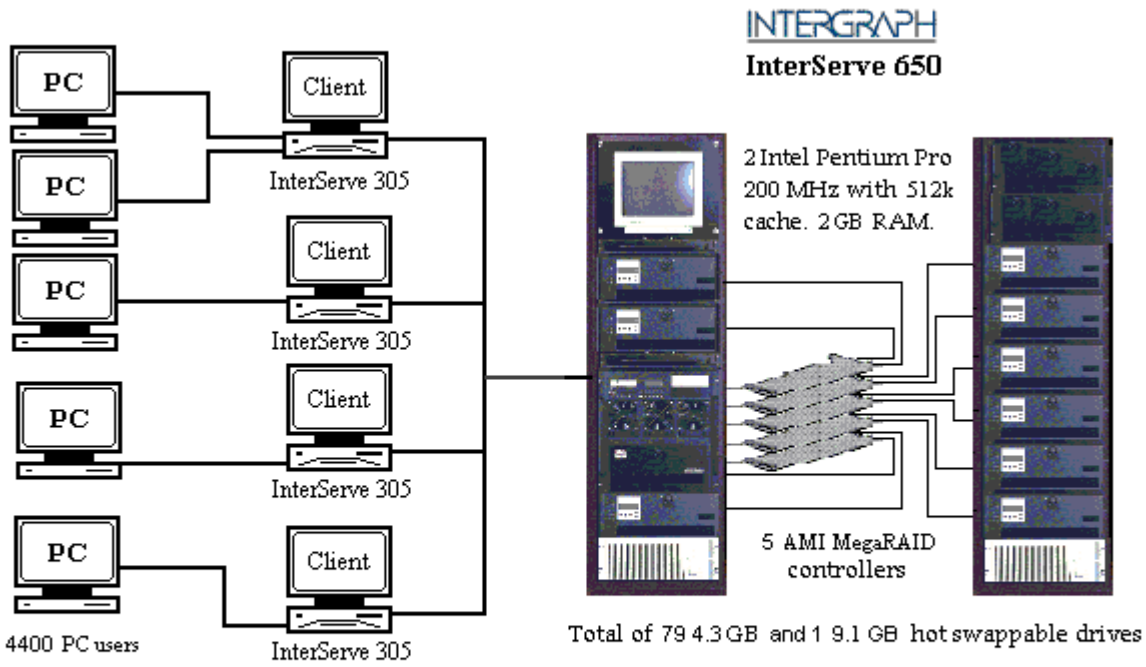
Requests for additional copies of this report should be sent to the following address:

TPC
C/O Shanley Public Relations
777 N. First St., Suite 600
San Jose, CA 95112-6113
USA

INTERGRAPH COMPUTER SYSTEMS	InterServe 650 Server and InterServe 305	TPC-C Rev 3.3 Report Date: 30 July 1997

Total System Cost	TPC-C Throughput	Price /Performance	Availability Date
$334,026	5307.23 tpmC	$62.93	1-July-1997

Processor	Database Manager	Operating System	Other Software	Number of Users
2 Pentium® Pro 200MHz	Microsoft SQL Server 6.5 (SP3)	Microsoft Windows NT 4.0 (SP1)	Microsoft Internet Information Server	4400



System Components	Server Qty	Server Type	Client Qty	Client Type
Processor	2	200 MHz Intel Pentium Pro 512k Cache	4	200 MHz Intel Pentium Pro 256k Cache
Memory	1	2048 MB	4	128 MB
Disk Controllers	5	AMI MegaRAID	4	Integrated Adaptec SCSI
Disk Drives	79 1	Seagate 4.3 GB Hot Swappable Seagate 9.1 GB Hot Swappable	4	Seagate 2.1GB
Total Storage		348.8 GB		8.4 GB

INTERGRAPH COMPUTER SYSTEMS	InterServe 650 Server and InterServe 305	TPC-C Rev 3.3 Report Date: 30 July 1997

Description	Part Number	Third Party Brand	Third Party Pricing	Unit Price	Qty	Extended Price	5 yr. Maint. Price
Server Hardware							
InterServe 650 (256MB,3x4GB,NTS,UPS)	FDPS37300-01			$30,990	1	$30,990	11,449
InterRAID8 + RAID Controller	FDSK473			$7,300	4	$29,200	12,232
InterRAID8 Without Controller	FDSK459			$5,300	5	$26,500	11,592
256MB Memory Upgrade	FMEM151			$2,399	7	$16,793	
4mm Tape Drive	FMTP16100-OD			$1,399	1	$1,399	
15" VGA Monitor	FOPT099			$399	1	$399	121
4.3 GB Hot Swap Drive	FDSK461			$1,199	76	$91,124	
9.1 GB Hot Swap Drive	FDSK470			$2,099	1	$2,099	
					Subtotal	$198,504	35,394
Server Software							
MS SQL Server 6.5 Database		Microsoft		24,999	1	24,999	10,475
MS SQL Workstation (with programmer toolkit)		Microsoft		499	1	499	
Visual C++ 32 Bit Edition (subscription)		Microsoft		499	1	499	
Microsoft NTS 4.0 included with server							
					Subtotal	25,997	10,475
Client Hardware							
InterServe 305 (32MB,2GB)	FDPS445			5,507	4	22,028	5,432
32MB Memory Upgrade	FMEM13400-OD			259	12	3,108	
Intel 10/100BaseT PCI Ethernet Controller	FINF920			150	5	750	
15" SVGA Monitor	FOPT099			399	4	1596	486
					Subtotal	27,482	5,917
Client Software							
Microsoft NTS 4.0 included on Web server							
(includes 5 user licenses)							
					Subtotal	0	0
User Connectivity							
24 Port 10BaseT Hub (includes 10% spares)	AT-3024TR-15	PC Importers		220	203	44,660	NA
(for 4400 users + 10% spares)							
8 Port 100BaseT Hub (includes spare)	AEF-8TX	CompuStar		529	3	1,587	NA
(for server +4 clients + spare)							
					Subtotal	46,247	0
					Other Discounts*	($15,991)	
					Total	**$282,239**	**$51,787**

Notes: * Reseller Discount

Prices used in TPC benchmarks reflect the actual prices a customer would pay for a one-time purchase of the stated components. Individually negotiated discounts are not permitted. Special prices based on assumptions about past or future purchases are not permitted. All discounts reflect standard pricing policies for the listed components. For complete details, see the pricing sections of the TPC benchmark pricing specifications. If you find that the stated prices are not available according to these terms, please inform the TPC at pricing@tpc.org. Thank You.

The benchmark results and test methodology were audited by Francois Raab of Information Paradigm, Inc.

Five Year Cost of Ownership:	**$334,026**
tpmC Rating:	**5307.23**
$ / tpmC:	**$62.93**

Numerical Quantities Summary

MQTH, Computed Maximum Qualified Throughput	5307.23 tpmC
% throughput difference, reported and reproducibility runs	0.14%

Response Times (seconds)	Average	90%	Maximum
New-Order	0.7	0.9	3.6
Payment	0.6	0.8	3.3
Order-Status	0.7	1.0	3.9
Delivery (interactive)	0.4	0.5	0.7
Delivery (deferred)	1.0	2.2	41.4
Stock-Level	1.7	2.6	5.9
Menu	0.4	0.5	0.9

Response time delay added for emulated components (included in response times above)	0.1

Transaction Mix, in percent of total transaction	
New-Order	44.03
Payment	43.45
Order-Status	4.32
Delivery	4.11
Stock-Level	4.09

Keying/Think Times (seconds)	Min.	Average	Max
New-Order	18.0 / 0.1	18.0 / 12.1	18.0 / 120.1
Payment	3.0 / 0.1	3.0 / 12.0	3.0 / 120.1
Order-Status	2.0 / 0.1	2.0 / 10.0	2.0 / 88.1
Delivery (interactive)	2.0 / 0.1	2.0 / 5.0	2.0 / 50.1
Stock-Level	2.0 / 0.1	2.0 / 5.1	2.0 / 47.8

Test Duration (minutes)	
Ramp-up time	28.67
Measurement interval	30.00
Transactions (all types) completed during measurement interval	361643
Ramp down time	38.97

Checkpointing	
Number of checkpoints	1
Checkpoint interval	30 minutes

General Items

Application Code and Definition Statements

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

Appendix A lists the application code used to implement this benchmark.

Test Sponsor

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

This benchmark was sponsored and executed by Intergraph Corporation. The benchmark was developed by Intergraph Corporation and Microsoft Corporation.

Parameter Settings

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

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

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

Appendix D contains the tunable parameters used in this benchmark.

Configuration Diagrams

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

The configuration diagrams for the priced and benchmarked systems are provided on the following pages.

The differences between the benchmarked configuration and the priced configuration include the following:

- Priced configuration contains the hardware required for additional network segments on the clients.

Figure 1: Priced Configuration

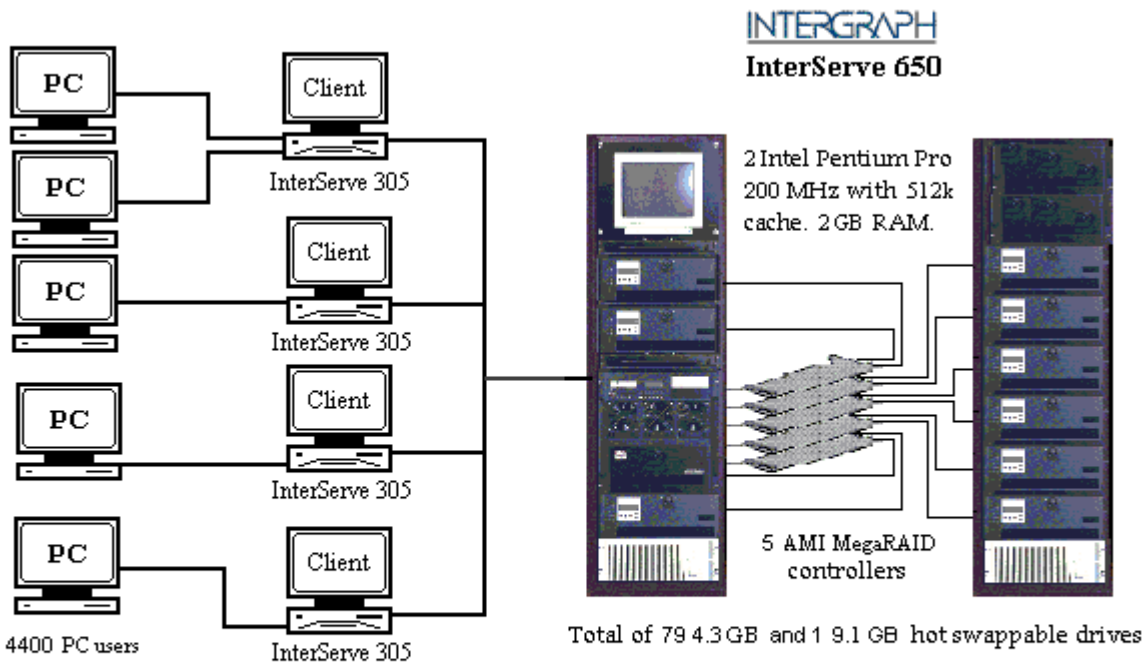
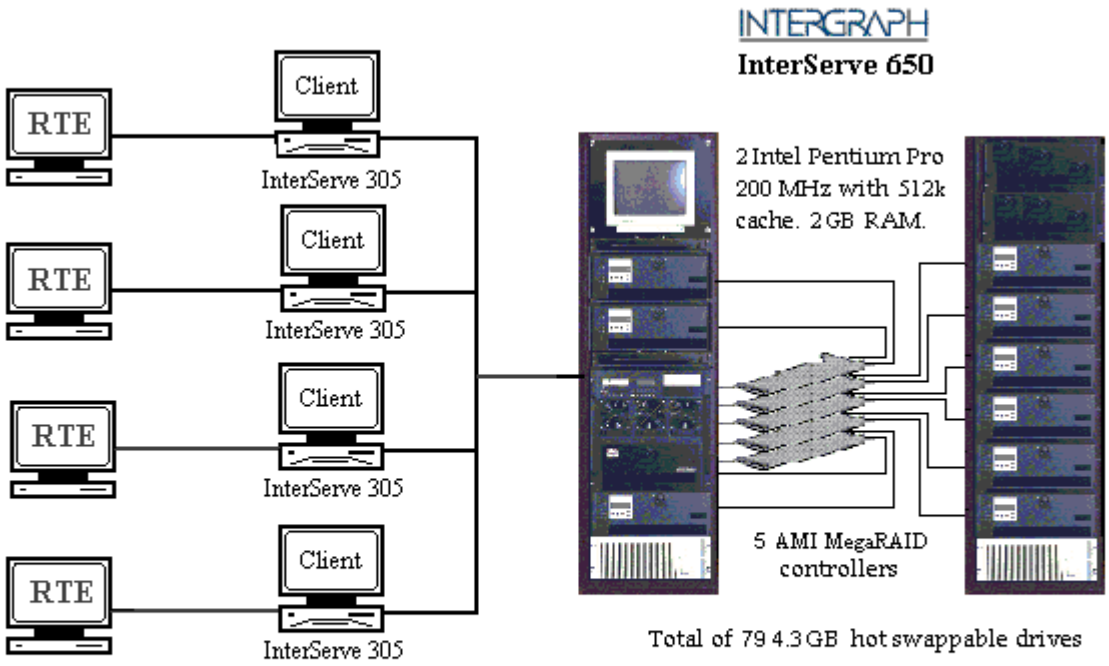


Figure 2: Benchmarked Configuration



Clause 1 Logical Database Design Related Items

Table Definitions

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

Appendix B contains the database definition files that were used to set up the database in this benchmark.

Physical Organization of Database

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

Appendix B contains information detailing the organization and distribution of the database.

Insert and Delete Operations

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

There were no restrictions on insert or delete operations to any tables in the database.

Partitioning

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

Partitioning was not used for this benchmark.

Table Replication

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

No replications were used in this benchmark.

Table Attributes

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

No additional or duplicated attributes were used in this benchmark.

Clause 2 Transaction and Terminal Profiles Related Items

Random Number Generation

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

RTE

Random numbers were generated using the drand48() call. This function generates pseudo-random numbers using the well-known linear congruential algorithm and 48-bit integer arithmetic. Function drand48() returns non-negative double-precision floating-point values uniformly distributed over the interval [0.0, 1.0). Function srand48() is an initialization entry point, which is invoked before drand48() is called.

Database Load

The loader program implements a pseudo random number generator. This generator will run the complete period before repeating. Copied from: Random Numbers Generators: Good Ones Are Hard to Find. Communications of the ACM - October 1988 Volume 31 Number 10.

Input/Output Screen Layout

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

All screen layouts match the TPC-C Benchmark Specification.

Priced Terminal Feature Verification

The method used to verify that the emulated terminals provide all the features described in Clause 2.2.2.4 must be explained. Although not specifically priced, the type and model of the terminals used for the demonstration in 8.1.3.3 must be disclosed and commercially available (including supporting software and maintenance).

The terminal emulator meets the specification. These features were manually verified using a Microsoft Internet Explorer HTTP connection from an Intergraph TD-300 workstation.

Presentation Manager or Intelligent Terminal

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

Application code on the client machine implemented the TPC-C user interface. No presentation manager software or intelligent terminal features were used. The source code for the user interface is listed in Appendix A.

Transaction Statistics

Table 1 lists the numerical quantities required by Clauses 8.1.3.5 to 8.1.3.11.

Table 1: Transaction Statistics

Transaction Type	Statistics	Percentage
New Order	Home warehouse	99.00%
	Remote warehouse	1.00%
	Rolled back transactions	0.99%
	Average items per order	10.01
Payment	Home warehouse	85.02%
	Remote warehouse	14.98%
	Last name access	59.90%
Order Status	Last name access	59.87%
Delivery	Skipped transactions	0%
Transaction Mix	New Order	44.03%
	Payment	43.45%
	Order status	4.32%
	Delivery	4.11%
	Stock level	4.09%

Queuing Mechanism

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

The source code for the delivery process is listed in Appendix A.

Clause 3 Transaction and System Properties Related Items

Transaction System Properties (ACID)

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

Atomicity

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

Completed Transactions

For Completed Transactions: The values of w_ytd, d_ytd, c_balance, c_ytd_payment, and c_payment_cnt of a row were randomly selected from the warehouse table. A Payment transaction was executed on the same warehouse, district, and customer. The transaction was committed. The values of w_ytd, d_ytd, c_balance, c_ytd_payment, and c_payment_cnt were verified that all had been updated appropriately.

Aborted Transactions

For Aborted Transactions: The values of w_ytd, d_ytd, c_balance, c_ytd_payment, and c_payment_cnt of a row were randomly selected from the warehouse table. A Payment transaction was executed on the same warehouse, district, and customer. The transaction was rolled back. The values of w_ytd, d_ytd, c_balance, c_ytd_payment, and c_payment_cnt were verified that none of the values had been changed.

Consistency

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

Consistency conditions 1 to 4 were run and the auditor verified that all four conditions were met.

Isolation

Sufficient conditions must be enabled at either the system or application level to ensure the required isolation defined above (Clause 3.4.1) is obtained.

Isolation conditions 1 to 9 were run and the auditor verified that all seven conditions were met.

Durability

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

Loss of Log Drive and Loss of Memory

The following test was conducted on the fully scaled 440 warehouse database using 4400 emulated terminals:

1. The initial count of the total number of orders was found by the sum of d_next_o_id of all rows in the district table giving the initial count.
2. The test was started and allowed to run at steady state for 10 minutes.
3. One of the log drives was pulled out. Since the disk was mirrored, testing was not interrupted.
4. The test was allowed to run for an additional 10 minutes.
5. The server was powered down.
6. The test was aborted on the RTE.
7. The server was powered back on.
8. Database recovery was done.

9. Several "success" orders recorded by the RTE were verified in the database.
10. Step 1 was repeated to give the total number of orders. The difference from Step 1 was calculated and compared to the number of “success” records in the RTE.

Loss of Data Drive

The following test was conducted on a 10 warehouse database with a load of 100 users. A fully scaled database would also pass this test.

1. The database was dumped to extra disks.
2. The total number of new orders was found by the sum of d_next_o_id of all rows in the district table giving the initial count.
3. The test was started and allowed to run at steady state for 10 minutes.
4. One of the data disks was removed from the RAID cabinet.
5. Errors were reported by Microsoft SQL Server.
6. The RTE was terminated.
7. The data disk was replaced.
8. Microsoft SQL Server was restarted but was unable to recover the database.
9. A dump of the transaction log was taken.
10. The backup of the database was restored and the transaction log was loaded.
11. Several "success" orders recorded by the RTE were verified in the database.
12. Step 1 was repeated to give the total number of orders. The difference from Step 2 was calculated and compared to the number of “success” records in the RTE.

Clause 4 Scaling and Database Population Related Items

Initial Cardinality of Tables

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

The number of rows in each table are shown in Table 2 below:

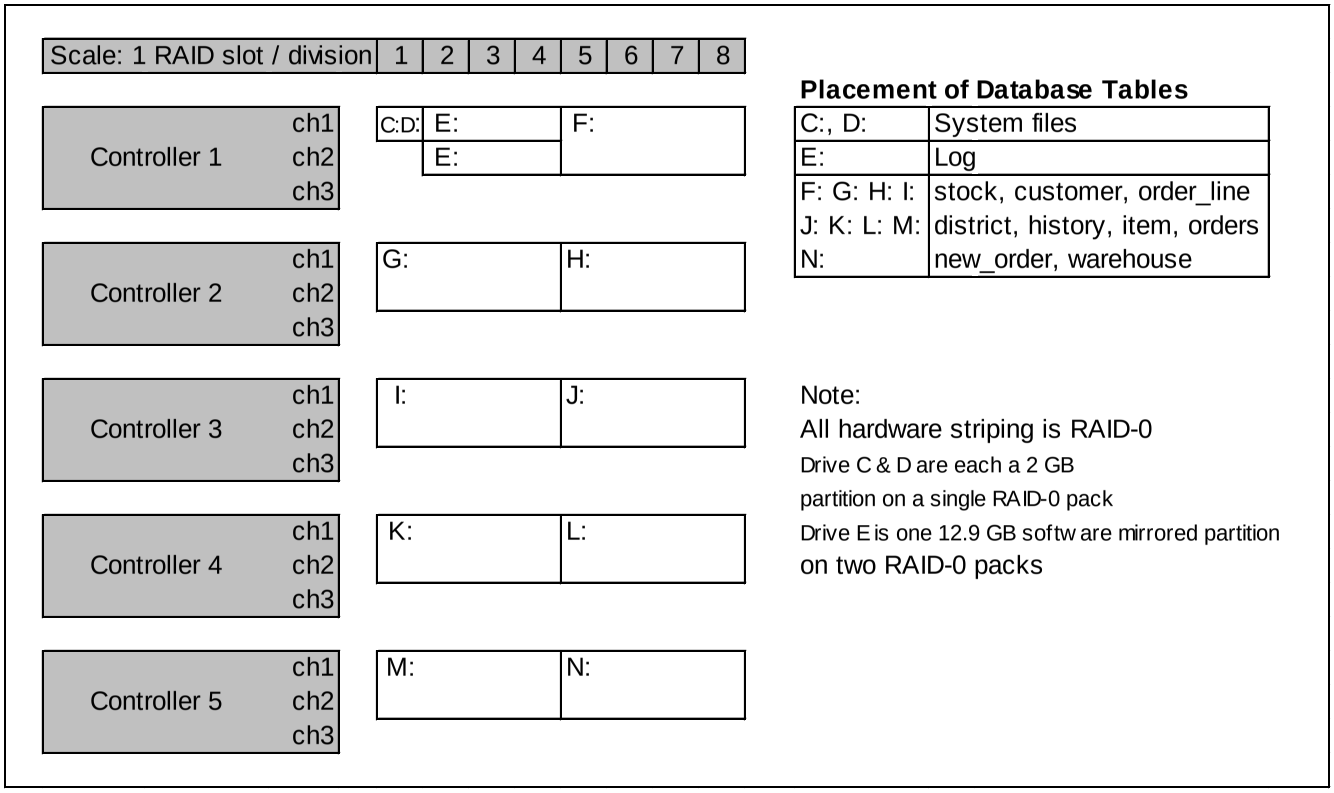
Table 2: Cardinality of Tables

Table	Occurrences
Warehouse	440
District	4,400
Customer	13,200,000
History	13,200,000
Order	13,200,000
New Order	3,960,000
Order Line	132,002,932
Stock	44,000,000
Item	100,000

Database Layout

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

Figure 3: Table Distributions Across Media



The distribution of the database tables over the 80 disks in the priced configuration is an extension of the distribution in the tested system configuration. The one hundred eighty day storage requirements are satisified with the unused space on the priced system.

Type of Database

A statement must be provided that describes:

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

Microsoft SQL Server version 6.5 (a relational database) was used in this benchmark. SQL Server stored procedures were used and invoked through DB-Library function calls.

Database Mapping

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

No partitioning or replication was used.

180 Day Space Computations

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

The details of the 180 day space computations and 8 hours of transaction log space requirements are shown in Appendix D

Clause 5 Performance Metrics and Response Time Related Items

Results

Measured tpmC must be reported.

Measured tpmC 5307.23 tpmC
Price per tpmC $62.93

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 3: Response Times

Type	Average	Maximum	90th percentile
New-Order	0.7	3.6	0.9
Payment	0.6	3.3	0.8
Order-Status	0.7	3.9	1.0
Interactive Delivery	0.4	0.7	0.5
Deferred Delivery	1.0	41.4	2.2
Stock-Level	1.7	5.9	2.6
Menu	0.4	0.9	0.5

Keying and Think Times

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

Table 4: Keying Times

Type	Minimum	Average	Maximum
New-Order	18.0	18.0	18.0
Payment	3.0	3.0	3.0
Order-Status	2.0	2.0	2.0
Interactive Delivery	2.0	2.0	2.0
Stock-Level	2.0	2.0	2.0

Table 5: Think Times

Type	Minimum	Average	Maximum
New-Order	0.1	12.1	120.1
Payment	0.1	12.0	120.1
Order-Status	0.1	10.0	88.1
Interactive Delivery	0.1	5.0	50.1
Stock-Level	0.1	5.1	47.8

An additional time of 100 milliseconds was added to the terminal emulation software to reflect real time latency within a web browser.

Response Time Frequency Distribution Curves

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

Figure 4: New Order Response Time Distribution

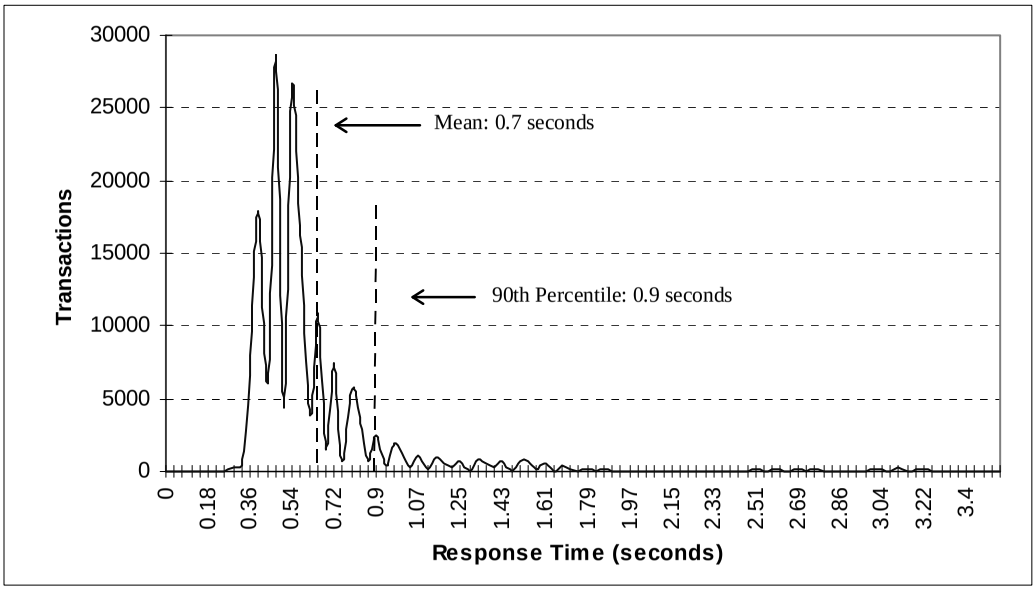


Figure 5: Payment Response Time Distribution

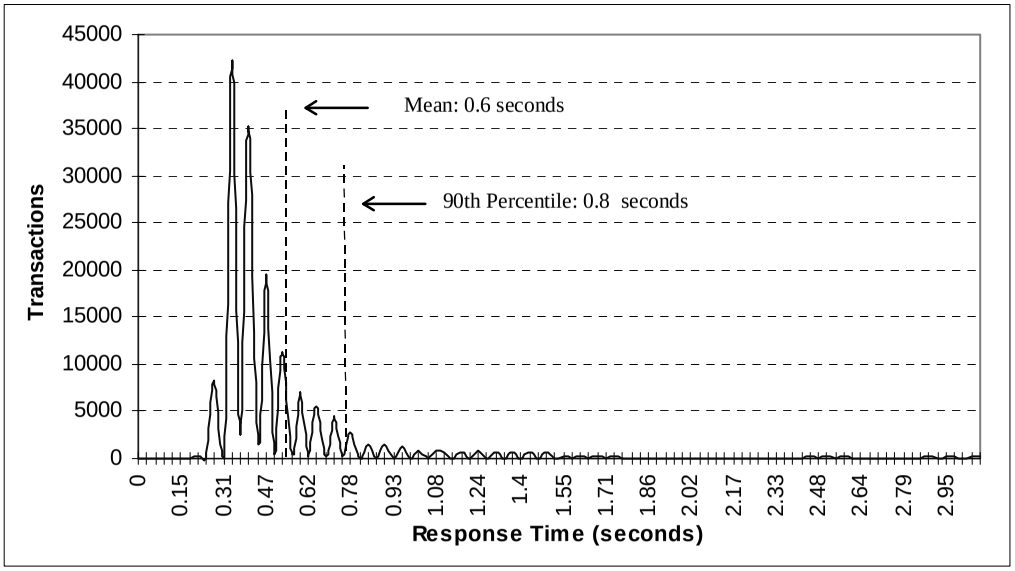


Figure 6: Order Status Response Time Distribution

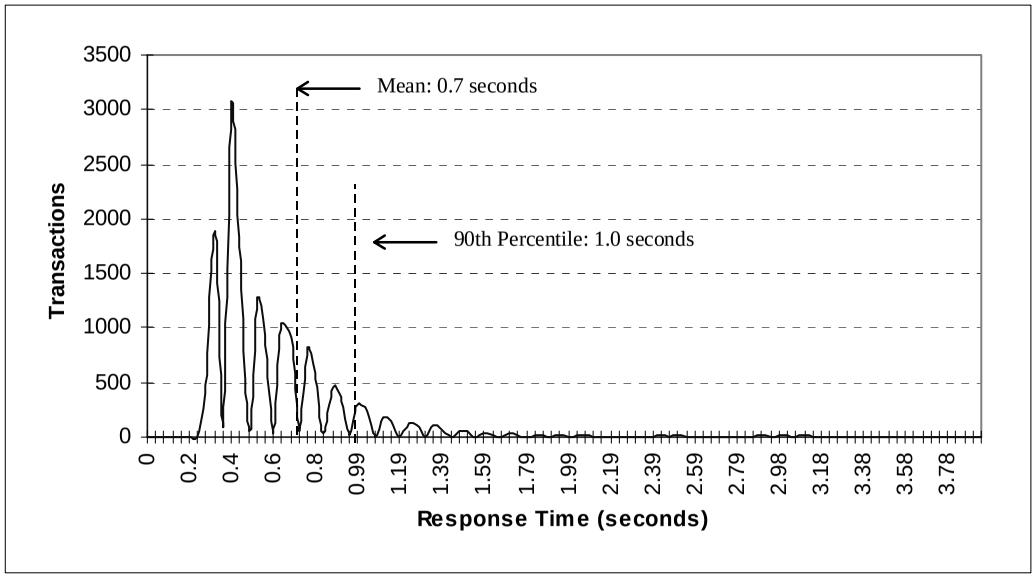


Figure 7: Delivery Response Time Distribution

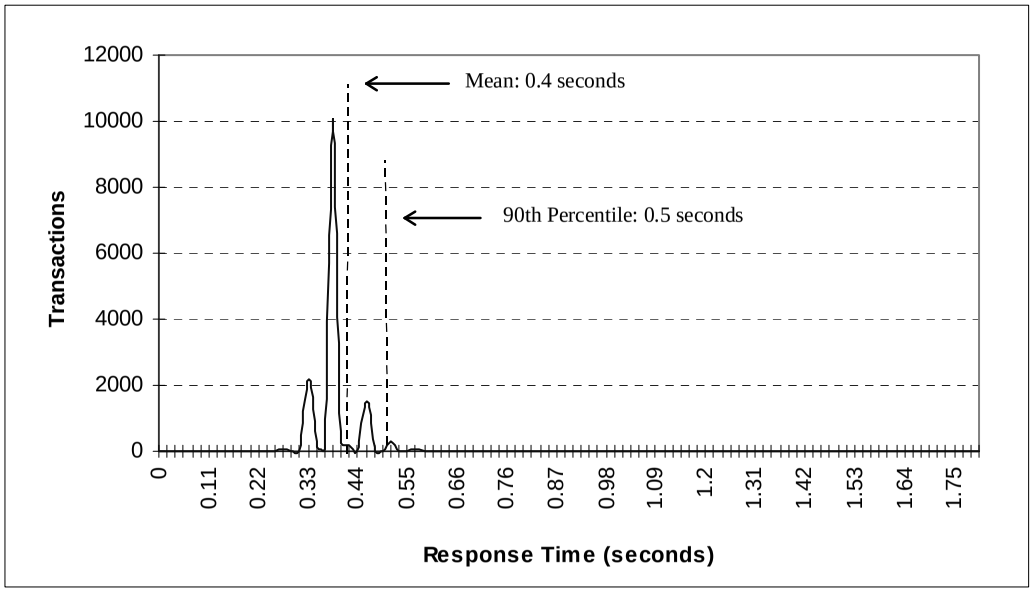
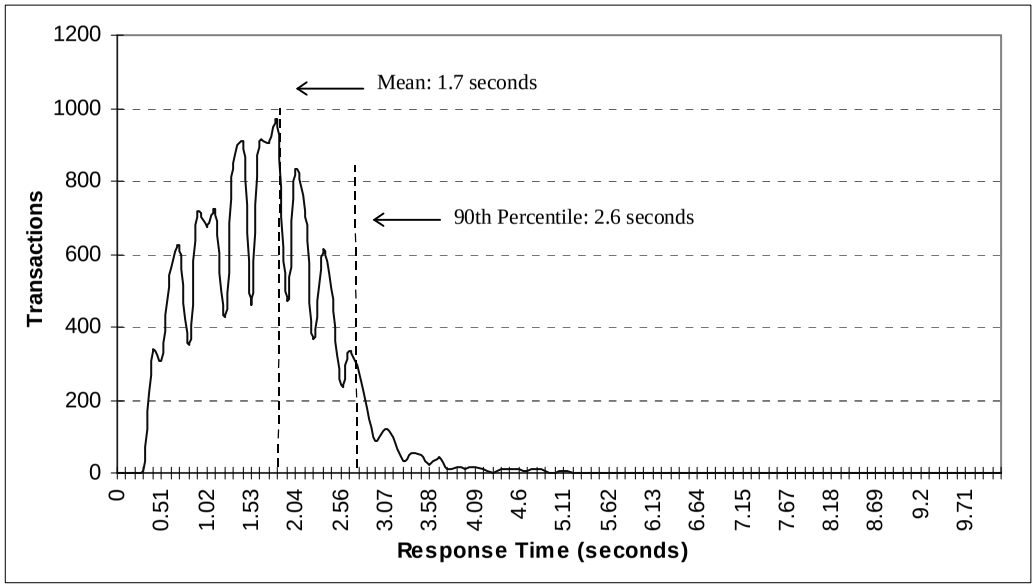


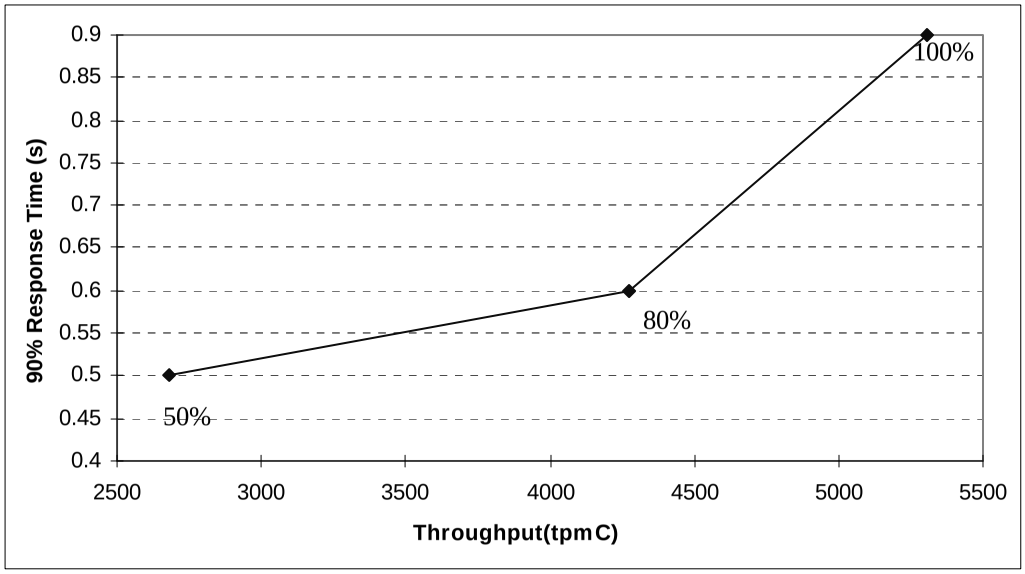
Figure 8: Stock Level Response Time Distribution



Response Time Versus Throughput

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

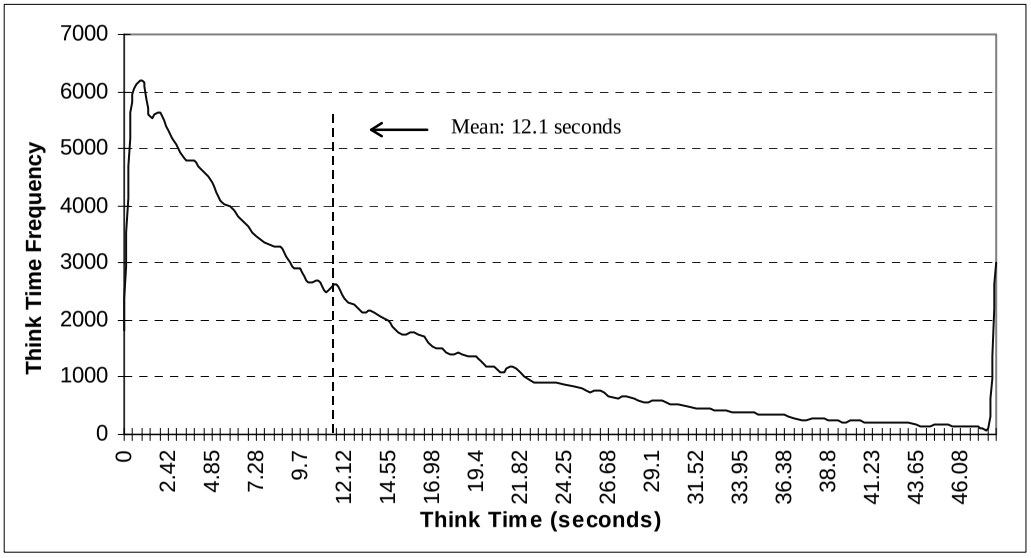
Figure 9: Response Time Versus Throughput



Think Time Frequency Distribution Curves

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

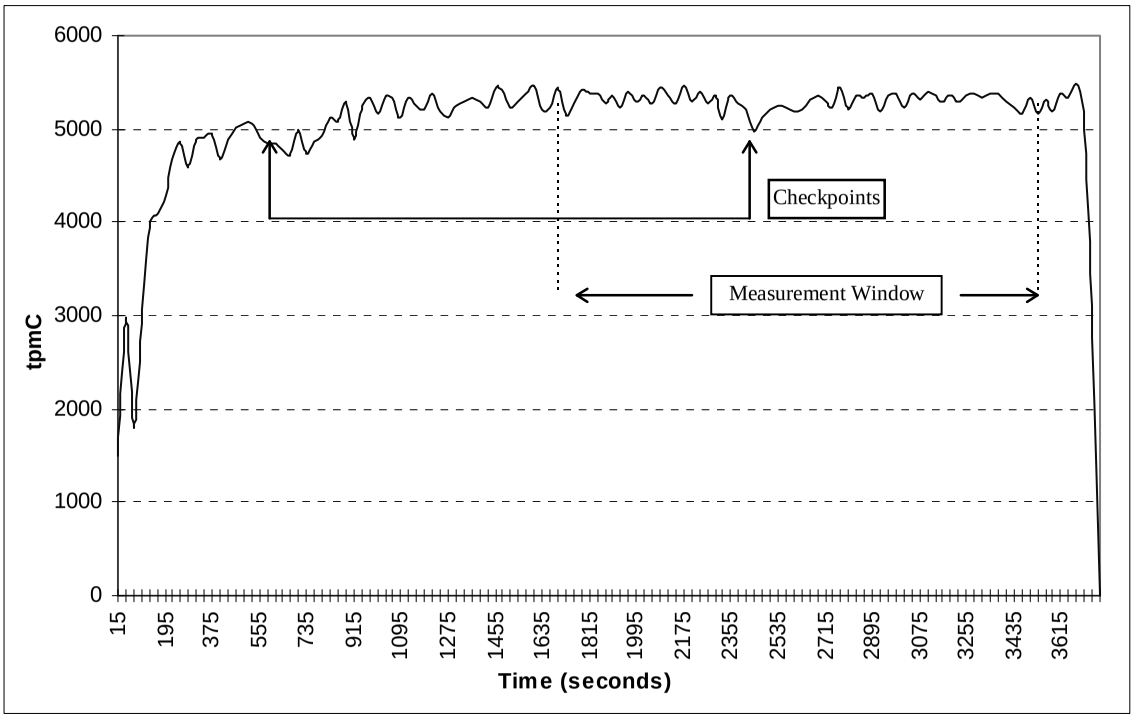
Figure 10: New Order Think Time Distribution



Throughput Versus Elapsed Time

A graph of throughput versus elapsed time (see Clause 5.6.5) must be reported for the New-Order transaction.

Figure 11: Throughput Versus Elapsed Time



Steady State Determination

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

Figure 11, New-Order throughput versus time graph, shows that the system was in steady state at the beginning of the measurement interval.

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.

In Microsoft SQL Server, a checkpoint writes all dirty pages that have been modified to the disks. During this test, SQL Server's recovery interval configuration option was set to the maximum allowable value. Checkpoints were performed by using a Visual Basic application which issued a specified number of checkpoints at specified time intervals.

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 same system for the same length of time as the measured run. The computed throughput for the reproducibility run was within 0.14% of the reported tpmC.

Measurement Period Duration

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

The measurement interval for the reported Maximum Qualified Throughput (tpmC) was 30 minutes.

Regulation of Transaction Mix

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

The "weighted" method used in this benchmark was as described in the specification. The maximum weights were within 5% of the initial value.

Transaction Statistics

The percentage of the total mix for each transaction type must be disclosed.
The percentage of New-Order transactions rolled back as a result of invalid item number must be disclosed.
The average number of order-lines entered per New-Order transaction must be disclosed.
The percentage of remote order-lines entered per New-Order transaction must be disclosed.
The percentage of remote Payment transactions must be disclosed.
The percentage of customer selections by customer last name in the Payment and Order-Status transactions must be disclosed.
The percentage of Delivery transactions skipped due to there being fewer than necessary orders in the New-Order table must be disclosed.

Table 1 lists the statistics required by 8.1.6.14 to 8.1.6.20

Checkpoints

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

The first checkpoint was performed 600 seconds after the start of the benchmark. The second checkpoint was performed 1800 seconds after the first checkpoint and was within the Measurement Interval. This checkpoint occurred 680 seconds after the start of the Measurement Interval.

Clause 6 SUT, Driver, and Communication Definition Related Items

RTE Description

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

A proprietary RTE was used in this benchmark. Appendix A includes a listing of a sample input script.

Emulated Components

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

No emulated components were used in this benchmark.

Configuration Diagrams

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

See "Configuration Diagrams" section under General Items at the beginning of this report.

Network Configuration

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

In the tested configuration, Four (4) 10 megabits/second LAN segments were used to connect four RTE machines to four client machines. One (1) 100 megabits/second LAN segment was used to connect the client machines to the database server.

In the priced configuration Four thousand four hundred (4400) users were spread over five (5) 10 megabits/second network segments, as opposed to four segments in the tested configuration. All client machines were connected to the server by one (1) 100 megabits/second LAN segment.

Network Bandwidth

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

The bandwidth of the network segments in the tested and priced configurations were 10 megabits/second between the users and the clients and 100 megabits/second between the clients and the server. The network utilized and priced is capable of supporting the traffic generated by this benchmark.

Operator Intervention

If the configuration requires operator intervention (see Clause 6.6.6), the mechanism and the frequency of this intervention must be disclosed.

No operator intervention was required.

Clause 7 Pricing Related Items

System Pricing

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

The detailed list of all hardware and programs for the priced configuration is listed in the executive summary section. All third party price quotations are listed in Appendix E.

Support Pricing

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

The total 5-year price support and maintenance price of all hardware and software is listed in the executive summary section. All third party price quotations are listed in Appendix E.

Availability

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

The software and hardware availability is July 1997.

Throughput 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 included.

tpmC	5-Year System Cost	Price/Performance	Availability
5307.23	$334,026	$62.93/tpmC	July 1997

Country Specific Pricing

Additional Clause 7 related items may be included in the Full Disclosure Report for each country specific priced configuration. Country specific pricing is subject to Clause 7.1.7.

All items in this system are priced for the United States of America.

Usage Pricing

For any usage pricing, the sponsor must disclose:

- *Usage level at which the component was priced.*
- *A statement of the company policy allowing such pricing.*

- Windows NT Server pricing policy for users is not dependent upon HTTP connections. Intergraph ships an OEM version of Windows NT which includes 5 user licenses.
- Microsoft Internet Information Server 2.0 is bundled with Windows NT Server 4.0, and Microsoft Internet Explorer is bundled with Windows NT Workstation and Server 4.0 and with Windows 95. Basically, the web server and web browsers come with the operating systems.

Clause 9 Audit Related Items

Auditor's Report

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

The author's name, address, phone number and a copy of his attestation letter appears on the next page.

Information Paradigm

TPC TRANSACTION PROCESSING PERFORMANCE COUNCIL

Certified Auditor

Sponsor: Helen P. Triplett
Server Performance & Analysis
Intergraph Computer Systems
1 Madison Industrial Park
Huntsville, Al 35894

July 24, 1997

I remotely verified the TPC Benchmark™ C performance of the following Client Server configuration:

Platform: InterServe 650 Server c/s
Operating system: Microsoft Windows NT 4.0
Database Manager: Microsoft SQL Server 6.50
Other Software: Microsoft Internet Information Server

The results were:

CPU's Speed	Memory	Disks	NewOrder 90% Response Time	tpmC
Server: InterServe 650 Server				
2 x Pentium Pro (200 MHz - 512K Cache)	2048 MB	79 x 4.3 GB	0.9 Seconds	5307.23
(4) Clients: InterServe 305 (Specification for each)				
1 x Pentium Pro (200 MHz - 256K Cache)	128 MB	1 x 1.0 GB	n/a	n/a

In my opinion, these performance results were produced in compliance with the TPC requirements for Revision 3.3.1 of the benchmark. The following verification items were given special attention:

- The transactions were correctly implemented
- The database records were the proper size
- The database was properly scaled and populated
- The ACID properties were met
- Input data was generated according to the specified percentages

1373 North Franklin Street • Colorado Springs, CO 80903-2527 • **Office:** 719/473-7555 • **Fax:** 719/473-7554

- The transaction cycle times included the required keying and think times
- The reported response times were correctly measured.
- At least 90% of all delivery transactions met the 80 Second completion time limit
- All 90% response times were under the specified maximums
- The measurement interval was representative of steady state conditions
- The reported measurement interval was 30 minutes (1800 seconds)
- One checkpoint was taken during the measurement interval
- Measurement repeatability was verified
- The 180 day storage requirement was correctly computed
- The system pricing was verified for major components and maintenance

Additional Audit Notes:

For availability reasons the 1.0 GB client system disks used in the testing were substituted with 2.1 GB disks for pricing. Based on the characteristics of the disks, and its use as client system disk, it is my opinion that this substitution had no impact on the reported performance.

As documented in the Full Disclosure Report, a dip in performance can be seen on the New-Order throughput versus time graph. It was verified that this dip was the result of starting the NT Performance Monitor on the clients for the purpose of collecting the number of connected users. As such, this dip is not included in the reported measurement interval.

Respectfully Yours,

François Raab
President

InterServe 650 Server (2-cpu)

1373 North Franklin Street • Colorado Springs, CO 80903-2527 • Office: 719/473-7555 • Fax: 719/473-7554

Appendix A: Source Code

SAMPLE USER SCRIPT

```
/s 719
/D </HTML>
/S
GET /scripts/tools/tpcc.dll?c=99&b=New+Order HTTP/1.0
/E 101
/D </HTML>
/s 1800
/S
GET
/scripts/tools/tpcc.dll?f=N&c=99&D=6&CI=2274&OS01=10&OI01=65682&O
Q01=4&OS02=10&OI02=49348&OQ02=10&OS03=10&OI03=90210&OQ03
=2&OS04=10&OI04=93252&OQ04=10&OS05=10&OI05=43221&OQ05=4&
OS06=10&OI06=7909&OQ06=7&OS07=10&OI07=45236&OQ07=2&OS08=
10&OI08=31714&OQ08=7&OS09=10&OI09=47300&OQ09=10&OS10=10&
OI10=40158&OQ10=6&OS11=&OI11=&OQ11=&OS12=&OI12=&OQ12=&O
S13=&OI13=&OQ13=&OS14=&OI14=&OQ14=&OS15=&OI15=&OQ15=
HTTP/1.0
/E 1010
/s 516
/D </HTML>
/S
GET /scripts/tools/tpcc.dll?c=99&b=Payment HTTP/1.0
/E 201
/D </HTML>
/s 300
/S
GET
/scripts/tools/tpcc.dll?f=P&c=99&D=2&CI=&CW=10&CD=2&CL=ANTIANTIE
ING&H=2307.74 HTTP/1.0
/E 203
/s 1232
/D </HTML>
/S
GET /scripts/tools/tpcc.dll?c=99&b=New+Order HTTP/1.0
/E 101
/D </HTML>
/s 1800
/S
GET
/scripts/tools/tpcc.dll?f=N&c=99&D=1&CI=1062&OS01=10&OI01=48222&O
Q01=3&OS02=10&OI02=49242&OQ02=6&OS03=10&OI03=48925&OQ03=
5&OS04=10&OI04=53394&OQ04=7&OS05=10&OI05=72724&OQ05=10&O
S06=10&OI06=40006&OQ06=7&OS07=10&OI07=23782&OQ07=4&OS08=
10&OI08=98022&OQ08=3&OS09=10&OI09=61670&OQ09=8&OS10=10&O
I10=80964&OQ10=1&OS11=10&OI11=87885&OQ11=3&OS12=10&OI12=9
6355&OQ12=4&OS13=10&OI13=90077&OQ13=7&OS14=10&OI14=48341
&OQ14=4&OS15=10&OI15=43702&OQ15=1 HTTP/1.0
/E 1015
/s 1799
/D </HTML>
/S
GET /scripts/tools/tpcc.dll?c=99&b=Payment HTTP/1.0
/E 201
/D </HTML>
/s 300
/S
GET
/scripts/tools/tpcc.dll?f=P&c=99&D=10&CI=&CW=10&CD=10&CL=PRESATI
ONPRES&H=810.41 HTTP/1.0
/E 203
/s 742
/D </HTML>
/S
GET /scripts/tools/tpcc.dll?c=99&b=New+Order HTTP/1.0
/E 101
/D </HTML>
/s 1800
/S
GET
/scripts/tools/tpcc.dll?f=N&c=99&D=3&CI=2214&OS01=10&OI01=96956&O
Q01=5&OS02=10&OI02=70886&OQ02=1&OS03=10&OI03=82006&OQ03=
10&OS04=10&OI04=89830&OQ04=10&OS05=10&OI05=98530&OQ05=5&
OS06=10&OI06=65714&OQ06=2&OS07=10&OI07=72934&OQ07=3&OS08
=10&OI08=47141&OQ08=4&OS09=10&OI09=7268&OQ09=6&OS10=10&O
I10=16596&OQ10=7&OS11=10&OI11=87236&OQ11=1&OS12=10&OI12=4
4708&OQ12=8&OS13=10&OI13=31686&OQ13=6&OS14=&OI14=&OQ14=
&OS15=&OI15=&OQ15= HTTP/1.0
/E 1013
/s 1046
/D </HTML>
/S
GET /scripts/tools/tpcc.dll?c=99&b=New+Order HTTP/1.0
/E 101
/D </HTML>
/s 1800
/S
GET
/scripts/tools/tpcc.dll?f=N&c=99&D=4&CI=220&OS01=10&OI01=8422&OQ0
1=5&OS02=10&OI02=79500&OQ02=6&OS03=10&OI03=65762&OQ03=4&
OS04=10&OI04=90262&OQ04=7&OS05=10&OI05=24629&OQ05=10&OS0
6=10&OI06=73892&OQ06=3&OS07=10&OI07=7136&OQ07=10&OS08=10
&OI08=49342&OQ08=6&OS09=&OI09=&OQ09=&OS10=&OI10=&OQ10=&
OS11=&OI11=&OQ11=&OS12=&OI12=&OQ12=&OS13=&OI13=&OQ13=&
OS14=&OI14=&OQ14=&OS15=&OI15=&OQ15= HTTP/1.0
/E 1008
/s 1303
/D </HTML>
/S
GET /scripts/tools/tpcc.dll?c=99&b=Payment HTTP/1.0
/E 201
/D </HTML>
/s 300
/S
GET
/scripts/tools/tpcc.dll?f=P&c=99&D=9&CI=&CW=10&CD=9&CL=PRESOUG
HTPRI&H=3036.97 HTTP/1.0
/E 203
/s 1028
/D </HTML>
/S
GET /scripts/tools/tpcc.dll?c=99&b=Order-Status HTTP/1.0
/E 301
/D </HTML>
/s 200
/S
GET /scripts/tools/tpcc.dll?f=O&c=99&D=3&CI=2276&CL= HTTP/1.0
/E 302
/s 204
/D </HTML>
/S
GET /scripts/tools/tpcc.dll?c=99&b=New+Order HTTP/1.0
/E 101
/D </HTML>
/s 1800
/S
GET
/scripts/tools/tpcc.dll?f=N&c=99&D=1&CI=2982&OS01=10&OI01=81741&O
Q01=4&OS02=10&OI02=45027&OQ02=5&OS03=10&OI03=97510&OQ03=
7&OS04=10&OI04=71781&OQ04=2&OS05=10&OI05=63634&OQ05=8&OS
06=10&OI06=63398&OQ06=8&OS07=10&OI07=7872&OQ07=8&OS08=&O
I08=&OQ08=&OS09=&OI09=&OQ09=&OS10=&OI10=&OQ10=&OS11=&OI
11=&OQ11=&OS12=&OI12=&OQ12=&OS13=&OI13=&OQ13=&OS14=&OI1
4=&OQ14=&OS15=&OI15=&OQ15= HTTP/1.0
/E 1007
/s 1402
/D </HTML>
/S
GET /scripts/tools/tpcc.dll?c=99&b=Payment HTTP/1.0
/E 201
/D </HTML>
/s 300
/S
GET
/scripts/tools/tpcc.dll?f=P&c=99&D=3&CI=&CW=8&CD=8&CL=PRESESEP
RI&H=1285.41 HTTP/1.0
/E 205
/s 1421
```

RTE PROFILE

```
# sample profile
# sample profile
MAX_TPMC=5000           export MAX_TPMC
ENGINE_USERS=1100       export ENGINE_USERS
INPUT_DIR='pwd'/_input  export INPUT_DIR
OUTPUT_DIR='pwd'/output  export OUTPUT_DIR
LOGIN_PROMPT="Not used"  export LOGIN_PROMPT
LOGIN_TEXT="Not used"    export LOGIN_TEXT
PASSWD_PROMPT="Not used" export PASSWD_PROMPT
PASSWD_TEXT="Not used"   export PASSWD_TEXT
SHELL_PROMPT="Not used"  export SHELL_PROMPT
SHELL_TEXT="Not used"    export SHELL_TEXT
#export DUMP_CORE=1
export PATH=$PATH:~/web/bin
```

CONTEXT.H

```
/* Audited: 28 February 1997 */

/* context.h
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#ifndef __context_h__
#define __context_h__

#include <windows.h>
#include <tpcc/kit/src/tpcc.h>
#include "options.h"

#define E_MAXUSERS -1  /* Error: No free user slots. */
#define E_INVARGS -2      /* Error: Invalid arguments.  */

extern void e_log(char *);
```

```
typedef struct {
                        short   w_id;
                        short   d_id;
#ifdef DB_PRESENT
                        DBPROCESS *dbhandle;
#else
                        long    dbhandle;
#endif DB_PRESENT
                        CRITICAL_SECTION ucsec;
} context;

typedef context user_array[MAX_USERS];

user_array users;
CRITICAL_SECTION gcsec;

void init_user_array(void);
int create_user(short, short);
context *get_user(int);
void delete_user(int);
void cleanup_user_array(void);

#endif __context_h__
```

CONTEXT.C

```
/* Audited: 28 February 1997 */

/* context.c
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#include "context.h"

void init_user_array(void) {
                        int i;
                        InitializeCriticalSection(&gcsec);
                        EnterCriticalSection(&gcsec);
                        for(i = 0; i < MAX_USERS; i++)
                                users[i].w_id = (short)0;
                        LeaveCriticalSection(&gcsec);
}

int create_user(short w_id, short d_id) {
                        int i;
#ifdef DB_PRESENT
                        int spid;
#endif DB_PRESENT
                        if(w_id < 1 || w_id > MAXWH || d_id < 1 || d_id >
10) {
                                return E_INVARGS;
                        }
                        EnterCriticalSection(&gcsec);
                        for(i = 0; i < MAX_USERS; i++) {
                                if(!users[i].w_id) {
                                        users[i].w_id = w_id;
                                        LeaveCriticalSection(&gcsec);
                                        users[i].d_id = d_id;
                                        users[i].dbhandle = NULL;

                        InitializeCriticalSection(&users[i].ucsec);
#ifdef DB_PRESENT

                        if(!SQLOpenConnection(&(users[i].dbhandle),
SERVERNAME,

USEDB,

USERNAME,

USERPASSWD,

"Client",

&spid,

(long *)4096)) {
                                                users[i].dbhandle = NULL;
                                                return MAX_USERS +
TokenIndex;
                                        } else {

                                SQLInitPrivate(users[i].dbhandle, NULL);
                                        }
#endif DB_PRESENT
                                        return i + TokenIndex;
                                }
                        }
                        LeaveCriticalSection(&gcsec);
                        return E_MAXUSERS;
}

context *get_user(int user) {
  return &users[user - TokenIndex];
}

void delete_user(int index) {
                        index -= TokenIndex;
                        if(users[index].w_id) {
#ifdef DB_PRESENT
                                EnterCriticalSection(&users[index].ucsec);
                                SQLExit(users[index].dbhandle);
#endif DB_PRESENT

                                LeaveCriticalSection(&users[index].ucsec);

                                DeleteCriticalSection(&users[index].ucsec);
                                users[index].d_id = 0;
                                users[index].w_id = 0;
                        }
}

void cleanup_user_array(void) {
                        int i;
                        for(i = 0; i < MAX_USERS; i++)
                                delete_user(i);
                        DeleteCriticalSection(&gcsec);
}
```

DEFAULTFUNC.C

```
/* Audited: 28 February 1997 */

/* defaultfunc.c
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#include "defaultfunc.h"

int default_validate(assoc *a, default_data *data, char *output) {
                        int i = 0;
                        char errstr[256];
                        errstr[0] = '\0';
                        data->anumber = -3;
                        data->afloat = HUGE_VAL;
                        data->astring = 0;
                        while((*a)[0][i]) {
                                switch((*a)[0][i][0]) {
                                        case 'n':
                                                data->anumber =
VerifyInt((*a)[1][i], 3);
                                                break;
                                        case 'd':
                                                data->afloat =
VerifyDouble((*a)[1][i], 4);
                                                break;
                                        case 's':
                                                data->astring =
VerifyString((*a)[1][i], 25);
                                                break;
                                        default: break;
                                }
                                i++;
                        }
                        if(data->anumber < 0) {
                                switch(data->anumber) {
                                        case -1:
                                                strcat(errstr, "The Number
field must contain 3 or fewer digits.\r\n");
                                                break;
                                        case -2:
                                                strcat(errstr, "The Number
field must not contain any nondigit characters.\r\n");
                                                break;
                                        case -3:
                                                strcat(errstr, "You must fill in
the Number field.\r\n");
                                                break;
                                        default:
                                                strcat(errstr, "Unknown error
in the Number field.\r\n");
                                                break;
                                }
                        }
                        if(data->afloat == HUGE_VAL) {
                                strcat(errstr, "The Float field must be a
decimal number of up to 2 digit precision, with up to 4 characters
overall.\r\n");
                        }
                        if(!data->astring) {
                                strcat(errstr, "You must enter a string of 25
or fewer characters in the String field.\r\n");
                        }
                        if(errstr[0]) {
                                sprintf(output, errorpage, errstr);
                                return 0;
                        } else return 1;

}


void default_process(default_data *data) {
                        return;
}

void default_format(default_data *data, char *output) {
                        sprintf(output, defaultpage, data->anumber, data-
>afloat, data->astring);
```

```
}

void default_func_main(assoc *a, char *output) {
                        default_data data;
                        data.anumber = 0;
                        data.afloat = 0.0;
                        data.astring = 0;
                        if(!default_validate(a, &data, output)) return;
                        default_process(&data);
                        default_format(&data, output);
}
```

DEFAULTFUNC.H

```
/* Audited: 28 February 1997 */

/* defaultfunc.h
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#ifndef __defaultfunc_h__
#define __defaultfunc_h__

#include "inputparser.h"
#include "functions.h"
#include "errors.h"

#define DEFAULT_FUNC 1

typedef struct {
                        int anumber;
                        char *astring;
                        double afloat;
} default_data;

static char defaultpage[] =
"<HTML><HEAD><TITLE>Default Page</TITLE></HEAD><BODY>"
"<P><H3>This is the Default Page</H3></P><HR>"
"<P>It contains a number, which is %d.</P>"
"<P>It is worth approximately $%lf.</P>"
"<P>The only comment I have is %s.</P>"
"</BODY></HTML>\r\n";

void default_func_main(assoc *, char *);

#endif __defaultfunc_h__
```

DELIVER.C

```
#include <process.h>
#include "tpcc.h"
#include "deliver.h"

#define INCLUDE_DATABASE_CODE

/*
** This program issues the "delivery" transactions.  It receives requests
** through a mailslot from the client processes.  The mailslot is the
** "queue" as required by the spec.
*/

CRITICAL_SECTION ResultsCriticalSection;

DBPROCESS **dbproc;
BOOL *channel_busy;
struct delivery_node *incoming;
int delay;
HANDLE results_file;

void ThreadMain(int index)
{
                        DELIVERY_DATA DeliveryData;
                        SYSTEMTIME now;
                        int i, bytes_read;
                        char output_buffer[80];
                        DeliveryData.w_id = incoming[index].w_id;
                        DeliveryData.o_carrier_id =
incoming[index].o_carrier_id;
#ifdef INCLUDE_DATABASE_CODE
                        SQLInlineDelivery(dbproc[index],&DeliveryData,
DEADLOCK_RETRY,0);
#endif
                        //log the results
                        EnterCriticalSection(&ResultsCriticalSection);
                        sprintf(output_buffer,"QUEUED %04d-%02d-
%02d %02d:%02d:%02d.%03d\r\n",
                                incoming[index].queue_time.wYear,

                        incoming[index].queue_time.wMonth,
                                incoming[index].queue_time.wDay,
                                incoming[index].queue_time.wHour,


                        incoming[index].queue_time.wMinute,

                        incoming[index].queue_time.wSecond,

                        incoming[index].queue_time.wMilliseconds);
                        WriteFile(results_file,output_buffer,strlen(output_
buffer),&bytes_read,NULL);
                        sprintf(output_buffer,"W_ID:%d
Carrier:%d\r\n",incoming[index].w_id,incoming[index].o_carrier_id);
                        WriteFile(results_file,output_buffer,strlen(output_
buffer),&bytes_read,NULL);
                        for (i=0;i<10;i++)
                        {
                                sprintf(output_buffer,"D_ID:%02d
O_ID:%d\r\n",i+1,DeliveryData.DelItems[i].o_id);

                        WriteFile(results_file,output_buffer,strlen(output_
buffer),&bytes_read,NULL);
                        }
                        sprintf(output_buffer,"Status:
%s\r\n",DeliveryData.execution_status);
                        WriteFile(results_file,output_buffer,strlen(output_
buffer),&bytes_read,NULL);
                        sprintf(output_buffer,"THREAD: %d\r\n",index);
                        WriteFile(results_file,output_buffer,strlen(output_
buffer),&bytes_read,NULL);
                        GetLocalTime(&now);
                        sprintf(output_buffer,"FINISHED %04d-%02d-
%02d %02d:%02d:%02d.%03d\r\n",
                                now.wYear,
                                now.wMonth,
                                now.wDay,
                                now.wHour,
                                now.wMinute,
                                now.wSecond,
                                now.wMilliseconds);
                        WriteFile(results_file,output_buffer,strlen(output_
buffer),&bytes_read,NULL);
                        sprintf(output_buffer,"DELTA
%d\r\n",GetTickCount() - incoming[index].tran_start_time);
                        WriteFile(results_file,output_buffer,strlen(output_
buffer),&bytes_read,NULL);
                        LeaveCriticalSection(&ResultsCriticalSection);
                        channel_busy[index] = FALSE;
                        return;
}

int main(int argc, char **argv)
{
                        HANDLE message_handle;
                        int i, bytes_read;
                        char server_name[SERVER_NAME_LEN+1]="";
                        char results_file_name[MAX_PATH+1]="";
                        static int spid;
                        static int thread_count=1;
                        //error handling initialization
                        IngrUtilInit("delivery.err");
                        //parse the arguments
                        for (i=1;i<argc;i++)
                        {
                                if (argv[i][0] != '-' && argv[i][0] != '/')
continue;
                                switch (argv[i][1])
                                {
                                        case 's':
                                        case 'S':
                                                i++;

                                strncpy(server_name,argv[i],sizeof
server_name);
                                                break;
                                        case 'f':
                                        case 'F':
                                                i++;

                                strncpy(results_file_name,argv[i],sizeof
results_file_name);
                                                break;
                                        case 't':
                                        case 'T':
                                                i++;
                                                thread_count = atoi(argv[i]);
                                                break;
                                        default:
                                                printf("Invalid option:
%s\n",argv[i]);
                                                printf("Usage:\n\t%s -S
server_name -F results_file_name [-T threads]\n",argv[0]);
                                                return -1;
                                }
                        }
                        if (server_name[0] == '\0')
                        {
                                printf("Server name switch required\n");
                                return -1;
                        }
                        if (results_file_name[0] == '\0')
                        {
                                printf("Results file name switch
required\n");
                                return -1;
                        }
                        if (thread_count < 1)
                        {
                                printf("Invalid thread count\n");
                                return -1;
```

```
                        }
                        //attach to the database
                        dbproc = (DBPROCESS **) malloc(thread_count
* sizeof (DBPROCESS *));
                        channel_busy = (BOOL *) malloc(thread_count *
sizeof (BOOL));
                        for (i=0;i<thread_count;i++) channel_busy[i] =
FALSE;
#ifdef INCLUDE_DATABASE_CODE
                        SQLInit(NULL);
                        dbsetmaxprocs((short)thread_count);
                        for (i=0;i<thread_count;i++)
                        {
                                SQLOpenConnection(&dbproc[i],
                                        server_name,//database server
name
                                        "tpcc",         //database name
                                        "sa",           //database username
                                        "",                     //database
password
                                        "Delivery",     //application name???
                                        &spid,//?? output field ??
                                        4096);          //packet size
                                SQLInitPrivate(dbproc[i],NULL); //error
and message handling
                        }
#endif
                        //open up the communications for the client
processes to use
                        message_handle =
CreateMailslot(DELIVERY_FILE_NAME,

                        sizeof (struct delivery_node), //max message size

                        10, //wait time ... needed to allow control-c to kill
the process??


                        NULL); //security attributes
                        if (message_handle ==
INVALID_HANDLE_VALUE)
                        {
                                char *message;
                                message =
TranslateErrorCode(GetLastError());

                        UtilFatalError(0,"CreateMailslot()",message);
                        }
                        //create our statistics file
                        InitializeCriticalSection(&ResultsCriticalSection);
                        results_file = CreateFile(results_file_name,

                        GENERIC_WRITE,

                        FILE_SHARE_READ, //so we can type it out
                                                                NULL,
//security

                        CREATE_ALWAYS,

                        FILE_ATTRIBUTE_NORMAL,
                                                                NULL);
                        if (results_file == INVALID_HANDLE_VALUE)
                        {
                                char *message;
                                message =
TranslateErrorCode(GetLastError());
                                UtilFatalError(0,"CreateFile()",message);
                        }
                        //process incoming messages
                        incoming = (struct delivery_node *)
malloc(thread_count * sizeof (struct delivery_node));
                        do
                        {
                                for (i=0;i<thread_count;i++)
                                {
                                        if (!channel_busy[i])
                                        {
                                                channel_busy[i] = TRUE;
timeout_retry:
                                                if
(!ReadFile(message_handle,&incoming[i],sizeof (struct
delivery_node),&bytes_read,NULL))
                                                {//error
                                                        if (GetLastError() ==
ERROR_SEM_TIMEOUT) goto timeout_retry; //timeout allows a control-c to
kill the process??
                                                        else
                                                        {
                                                                char *message;
                                                                message =
TranslateErrorCode(GetLastError());

                                                UtilFatalError(0,"ReadFile() on
Mailslot",message);
                                                        }
                                                }
                                                if (bytes_read == 0) return 0;
//all done???

                                                _beginthread(ThreadMain,0,i);
                                                break;
                                        }
                                }
                                if (i >= thread_count) Sleep(1000); //one
second before trying again to find a free channel
                        } while(1);
}
```

DELIVER.H

```
/* Audited: 28 February 1997 */

/* delivery.h
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#ifndef __delivery_h__
#define __delivery_h__

#include "context.h"
#include <tpcc/kit/src/tpcc.h>
#include "inputparser.h"
#include "output.h"
#include "errors.h"
#include "mailslot.h"
#include "options.h"

#define DELIVERY_FUNC 5

static char dresp[] =
"<HTML><HEAD><TITLE>TPC-C:
Delivery</TITLE></HEAD><BODY><PRE>"
"                       Delivery\r\n"
"Warehouse: XXXX\r\n"
"\r\n"
"Carrier Number: XX\r\n"
"\r\n"
"Execution Status: XXXXXXXXXXXXXXXXXXXXXXXXX"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"</PRE><P><FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">"
"<INPUT TYPE=\"hidden\" NAME=\"c\" VALUE=\"%d\">"
"<INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"New Order\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Payment\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Delivery\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Order-Status\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Stock-Level\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Exit\">"
"</FORM></P></BODY></HTML>\r\n";

#define DW 118
#define DC 142
#define DE 166

extern void e_log(char *);
void delivery_func_main(assoc *, char *);
int delivery_func_parse(assoc *, int *, struct delivery_node *, char *);
int delivery_func_process(struct delivery_node *, int);
void delivery_func_format(char *, struct delivery_node *, int, int);

#endif __delivery_h__
```

ERRORS.H

```
/* Audited: 28 February 1997 */

/* errors.h
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#ifndef __errors_h__
#define __errors_h__

static char errorpage[] =
"<HTML><HEAD><TITLE>TPC-C: Error</TITLE></HEAD><BODY>"
"<p>You did something bad.  The error message was:</p>"
"<PRE>\r\n"
"%s</PRE>"
"<p>Either hit the \"back\" button on your browser and fix the problem, "
"or hit the \"Exit\" button below to terminate this session.  If you believe your
"
"input was not in error, send email to <a
href=\"mailto:rothomas@ingr.com\">Robert "
"Thomas</a> explaining the error you received and the situation that led up
to it.</P>"
```

```
"<HR>"
"<P><FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">"
"<INPUT TYPE=\"hidden\" NAME=\"cookie\" VALUE=\"%d\">"
" <INPUT TYPE=\"submit\" NAME=\"button\" VALUE=\"Exit\">"
"</FORM></P></BODY></HTML>\r\n";

static char dberrpage[] =
"<HTML><HEAD><TITLE>TPC-C: Database
Error</TITLE></HEAD><BODY>"
"<P>The database could not process your request.</P>"
"<P>Press the \"exit\" button below to abort this session.</P><HR>"
"<FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">"
"<INPUT TYPE=\"hidden\" NAME=\"c\" VALUE=\"%d\">"
"<INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Exit\">"
"</FORM></BODY></HTML>";

#define BAD_COOKIE_MSG "o The user authentication is not valid.\r\n
The session cannot proceed.\r\n  Press the 'Exit' button below.\r\n"
#define TOO_LONG_MSG "o The \"%s\" field contained too many
characters.\r\n  The maximum is %d.\r\n"
#define NOT_ISDIGIT_MSG "o The \"%s\" field contained nondigit
characters.\r\n"
#define NO_INPUT_MSG "o You did not fill in the \"%s\" field.\r\n  The field
is required.\r\n"


#endif __errors_h__
```

EXTENSIONS.C

```
/* Audited: 28 February 1997 */

/* extensions.c
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#include "extensions.h"

void register_extensions(void) {
                register_function(default_func_main,
DEFAULT_FUNC);
                register_function(login_func_main,
LOGIN_FUNC);
                register_function(processlogin_func_main,
PROCESSLOGIN_FUNC);
                register_function(query_form_func_main,
QUERY_FORM_FUNC);
                register_function(stock_level_func_main,
STOCKLVL_FUNC);
                register_function(delivery_func_main,
DELIVERY_FUNC);
                register_function(payment_func_main,
PAYMENT_FUNC);
                register_function(order_status_func_main,
ORDERSTAT_FUNC);
                register_function(new_order_func_main,
NEWORDER_FUNC);
                register_function(no_mailslot_func_main,
NOMAILSLOT_FUNC);
}

void init_extensions(void) {
                int rc=0;
                GetRegistryValues();
                init_user_array();
#ifdef DB_PRESENT
                IngrUtilInit("C:\\USERS\\DEFAULT\\DBERR.LOG
");
                rc=dbsetmaxprocs((short)MAX_USERS);
                SQLInit(NULL);
                open_mailslot();
#else
                service_available = 1;
#endif
}

void cleanup_extensions(void) {
                cleanup_user_array();
}
```

EXTENSIONS.H

```
/* Audited: 28 February 1997 */

/* extensions.h
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#ifndef __extensions_h__
#define __extensions_h__

extern void IngrUtilInit(char *);

/* #include headers for your extensions below. */

#include "login.h"
#include "defaultfunc.h"
#include "processlogin.h"
#include "query_form.h"
#include "stocklevel.h"
#include "delivery.h"
#include "payment.h"
#include "orderstatus.h"
#include "neworder.h"
#include "mailslot.h"


/*---- Don't modify anything below this point---------- */
#include "functions.h"
#include "options.h"

void register_extensions(void);
void init_extensions(void);
void cleanup_extensions(void);


#endif __extensions_h__
```

FUNCTIONS.C

```
/* Audited: 28 February 1997 */

/* functions.c
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#include "functions.h"

void init_function_array(void) {
                int i;
                for(i = 0; i < MAX_FUNCS; i++)
                        function_array[i] = (pbfunc)0;
}

int register_function(pbfunc function, int index) {
                if(index > MAX_FUNCS) return
E_OUT_OF_RANGE;
                else if(function_array[index]) return
E_ALREADY_DEFINED;
                else function_array[index] = function;
                return index;
}

/* This function should be modified to correctly select a
   function based on the input. */
int identify_function_index(assoc *a) {
                int i = 0;
                if(!service_available) return
NOMAILSLOT_FUNC;
                while((*a)[0][i]) {
                        if((*a)[0][i][0] == 'f') {
                                switch((*a)[1][i][0]) {
                                        case 'N': return
NEWORDER_FUNC;
                                        case 'D': return
DELIVERY_FUNC;
                                        case 'L': return
PROCESSLOGIN_FUNC;
                                        case 'S': return
STOCKLVL_FUNC;
                                        case 'P': return
PAYMENT_FUNC;
                                        case 'O': return
ORDERSTAT_FUNC;
                                        case 'M': break;
                                    default:  return
DEFAULT_FUNC;
                                }
                        }
                        if((*a)[0][i][0] == 'b') {
                                switch((*a)[1][i][0]) {
                                        case 'E':       return
LOGIN_FUNC;
                                        case 'N':
                                        case 'P':
                                        case 'O':
                                        case 'S':
                                        case 'D':       return
QUERY_FORM_FUNC;
                                        default:        return
DEFAULT_FUNC;
                                }
                        }
                        i++;
                }
                return LOGIN_FUNC;
}
```

FUNCTIONS.H

```
/* Audited: 28 February 1997 */

/* functions.h
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */
```

```
#ifndef __functions_h__
#define __functions_h__

#include "inputparser.h"
#include "extensions.h"


#define MAX_FUNCS 255
#define E_OUT_OF_RANGE -1
#define E_ALREADY_DEFINED -2

typedef void bfunc(assoc *, char *);
typedef bfunc *pbfunc;

pbfunc function_array[MAX_FUNCS];

typedef enum {
} functions;

void init_function_array(void);
int register_function(pbfunc, int);
int identify_function_index(assoc *);



#endif __functions_h__
```

INPUTPARSER.C

```
/* Audited: 28 February 1997 */

/* inputparser.c
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#include "inputparser.h"

char *split(char *first, char sp) {
        int i;
        for(i = 0; i < (int)strlen(first) && first[i] != sp; i++);
        if(i == (int)strlen(first)) return (char *)0;
        else {
                first[i] = '\0';
                return &(first[i+1]);
        }
}

void init_assoc(assoc *a) {
                                int i = 0;
                                for(i = 0; i < MAX_KEYS; i++) {
                                        (*a)[0][i] = (char *)0;
                                        (*a)[1][i] = (char *)0;
                                }
}

void fill_assoc(assoc *a, char *query) {
                                char *val, *rest;
                                int index = 0;
                                if(!query) return;
                                while(query) {
                                        rest = split(query, '&');
                                        val = split(query, '=');
                                        (*a)[0][index] = query;
                                        (*a)[1][index++] = val;
                                        query = rest;
                                }
}

long VerifyLong(char *str, int maxlen) {
        int x;
        if(!str || !(x = strlen(str))) return -3;
        if(x > maxlen) return -1;
        else for(;x;x--) if(!isdigit(str[x-1])) return -2;
        else return atol(str);
                                        return 0L;
}



int VerifyInt(char *str, int maxlen) {
                                        int x;
                                        if(!str || !(x = strlen(str))) return -3;
                                        if(x > maxlen) return -1;
                                        else for(;x;x--) if(!isdigit(str[x-1])) return -2;
                                        else return atoi(str);
                                        return 0;
}

short VerifyShort(char *str, int maxlen) {
        int x;
        if(!str || !(x = strlen(str))) return -3;
        if(x > maxlen) return -1;
        else for(;x;x--) if(!isdigit(str[x-1])) return -2;
        else {
                x = atoi(str);
                return (short)x;
        }
                                        return (short)0;
}

char *VerifyString(char *str, int maxlen) {
        int x;
        if(!str) return (char *)0;
        x = strlen(str);
        if(x > maxlen) return (char *)0;
        else return str;
}


double VerifyDouble(char *str, int maxlen) {
        int x;
        if(!str) return HUGE_VAL;
        x = strlen(str);
        if(x > maxlen) return HUGE_VAL;
        else for(;x;x--) {
                if(isdigit(str[x-1]));
                else if((str[x-1] == '.') && (strlen(str)-x < 3));
                else if((str[x-1] == '-') && (x == 1));
                else if((str[x-1] == '+') && (x == 1));
                else return HUGE_VAL;
        }
        return atof(str);
}
```

INPUTPARSER.H

```
/* Audited: 28 February 1997 */

/* inputparser.h
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#ifndef __inputparser_h__
#define __inputparser_h__

#include <string.h>
#include <ctype.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>

#define MAX_KEYS 100

extern void e_log(char *);

typedef char *assoc[2][MAX_KEYS];

char *split(char *, char);
void init_assoc(assoc *);
void fill_assoc(assoc *, char *);

/* The following are useful generic validation type functions. */

long VerifyLong(char *, int);
int VerifyInt(char *, int);
short VerifyShort(char *, int);
char *VerifyString(char *, int);
double VerifyDouble(char *, int);

#endif __inputparser_h__
```

LOGIN.C

```
/* Audited: 28 February 1997 */

/* login.c
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#include "login.h"

int login_validate(assoc *a) {
                                int i = 0;
                                while((*a)[0][i]) {
                                        switch((*a)[0][i][0]) {
                                                case 'c':
                                                        return VerifyInt((*a)[1][i], 4);
                                                        break;
                                                default:
                                                        break;
                                        }
                                        ++i;
                                }
                                return -1;
}

void login_func_main(assoc *a, char *output) {
                                int cookie = login_validate(a);
                                if(cookie >= 0)
                                        delete_user(cookie);
                                strcpy(output, loginpage);
}
```

LOGIN.H

```
/* Audited: 28 February 1997 */

/* login.h
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#ifndef __login_h__
#define __login_h__

#include "context.h"
#include "inputparser.h"

static char loginpage[] =
"<HTML><HEAD><TITLE>Welcome to TPC-C</TITLE></HEAD><BODY>"
"<P>Please identify your Warehouse and District for this session.</P>"
"<FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">"
"<INPUT TYPE=\"hidden\" NAME=\"f\" VALUE=\"L\">"
"Your Warehouse ID: <INPUT NAME=\"W\" SIZE=4><BR>"
"Your District ID:  <INPUT NAME=\"d\" SIZE=2><BR><HR>"
"<INPUT TYPE=\"submit\">"
"</FORM></BODY></HTML>\r\n";

#define LOGIN_FUNC 0

extern void e_log(char *);

void login_func_main(assoc *, char *);

#endif __login_h__
```

MAILSLOT.C

```
/* Audited: 28 February 1997 */

/* mailslot.c
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#include "mailslot.h"


void open_mailslot(void) {
                        delivery_handle =
CreateFile(DELIVERY_FILE_NAME,

                        GENERIC_WRITE,

                        FILE_SHARE_WRITE | FILE_SHARE_READ,

                        NULL,//security

                        OPEN_EXISTING,

                        FILE_ATTRIBUTE_NORMAL,

                        NULL);//template file
                        if(delivery_handle ==
INVALID_HANDLE_VALUE) {
                                service_available = 0;
                        } else {
                                service_available = 1;
                        }
}

void no_mailslot_func_main(assoc *a, char *output) {
                                sprintf(output, enosvcdmb);
}
```

MAILSLOT.H

```
/* Audited: 28 February 1997 */

/* mailslot.h
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#ifndef __mailslot_h__
#define __mailslot_h__

#include <windows.h>
#include <tpcc/kit/src/tpcc.h>
#include "inputparser.h"
#include "options.h"

#define NOMAILSLOT_FUNC 9
#define DELIVERY_FILE_NAME "\\\\.\\mailslot\\TPCCdelivery"

HANDLE delivery_handle;

int service_available;

static char enosvcdmb[] =
"<HTML><HEAD><TITLE>TPC-C: Service
Unavailable</TITLE></HEAD><BODY>"
"<P>Sorry, the service is unavailable at this time.  The server failed
attempting to open"
" a connection to the delivery process mailbox.  As a result, no transactions
can be"
" performed.  Please try again in an hour.  If the problem persists, email "
"<a href=\"mailto:rothomas@ingr.com\">Robert Thomas</a> and report
seeing this message.</P>"
"</BODY></html>";

void open_mailslot(void);
void no_mailslot_func_main(assoc *, char *);

#endif __mailslot_h__
```

MSTPCC.H

```
#ifndef __damien_tpcc_h__
#define __damien_tpcc_h__

#define DBNTWIN32
// TPC-C Benchmark Kit
//
// Module:  TPCC.H
// Author:  DamienL


// Build number of TPC Benchmark Kit
#define TPCKIT_VER        "2.04"

// General headers
#include <windows.h>
#include <winbase.h>
#include <stdlib.h>
#include <stdio.h>
#include <process.h>
#include <stddef.h>
#include <stdarg.h>
#include <string.h>
#include <signal.h>
#include <time.h>
#include <timeb.h>
#include <types.h>
#include <wincon.h>

#ifdef USE_ODBC
// ODBC headers
#include <sql.h>
#include <sqlext.h>
HENV                                    henv;
#endif

// DB-Library headers
#include <sqlfront.h>
#include <sqldb.h>

// Critical section declarations
CRITICAL_SECTION        ConsoleCritSec;
CRITICAL_SECTION        QueuedDeliveryCritSec;
CRITICAL_SECTION        WriteDeliveryCritSec;
CRITICAL_SECTION        DroppedConnectionsCritSec;
CRITICAL_SECTION        ClientErrorLogCritSec;

// General constants
#define SQLCONN DBPROCESS
#define DUMB_MESSAGE            5701
#define ABORT_ERROR             6104
#define INVALID_ITEM_ID                 0
#define MILLI                   1000
#define MAX_THREADS             2510
#define STATS_MSG_LOW                   3600
#define STATS_MSG_HIGH                  3700
#define SHOWPLAN_MSG_LOW                6200
#define SHOWPLAN_MSG_HIGH               6300
#define FALSE                   0
#define TRUE                    1
#define DEADLOCKWAIT              10
#define UNDEF                                   -1
#define MINPRINTASCII             32
#define MAXPRINTASCII             126

// Default environment constants
#define SERVER                  "argus1"
#define DATABASE                "tpcc"
#define USER                    "sa"
#define PASSWORD                ""
#define SYNCH_SERVERNAME                ""

// Statistic constants
#define INTERVAL                 20   // Total interval of buckets, in sec
#define UNIT                     .1   // Time period of each bucket
#define HIST_MAX                200   // Num of histogram buckets =
INTERVAL/UNIT
#define BUCKET                  100   // Division factor for response time

// Default master arguments
#define ADMIN_DATABASE          "tpcc_admin"
#define RAMP_UP                 600
#define STEADY_STATE            1200
```

```
#define RAMP_DOWN               120
#define NUM_USERS               10
#define NUM_WAREHOUSES          1
#define THINK_TIMES             0
#define DISPLAY_DATA            0
#define DEFMSPACKSIZE           4096
#define TRANSACTION             0
#define CLIENT_MODE             1
#define DEF_WW_T                120
#define DEF_WW_a                1
#define DEADLOCK_RETRY          4
#define DELIVERY_BACKOFF        2
#define DELIVERY_MODE           0
#define NEWORDER_MODE           0
#define DEF_LOAD_MULTIPLIER     1.0
#define DEF_CHECKPOINT_INTERVAL 960
#define DEF_FIRST_CHECKPOINT    240
#define DISABLE_90TH            0
#define                 RESFILENAME
                        "results.txt"
#define SQLSTAT_FILENAME        "sqlstats.txt"
#define ENABLE_SQLSTAT          0
#define SQLSTAT_PERIOD          100
#define SHUTDOWN_SERVER         0
#define AUTO_RUN                0

// Default client arguments
#define NUM_THREADS             10
#define DEFCLPACKSIZE           4096
#define X_FLAG          0
#define Y_FLAG          1
#define NUM_DELIVERIES          2
#define CLIENT_NURAND_C_LAST_C  200
#define CLIENT_NURAND_C_ID_C    500
#define CLIENT_NURAND_OL_I_ID_C 5000
#define DISABLE_DELIVERY_RESFILES 1

// Globals for queued delivery handling
typedef struct          delivery_node *DELIVERY_PTR;
DELIVERY_PTR            delivery_head, delivery_tail;
short                   queued_delivery_cnt;
HANDLE                  hDeliveryMonPipe;
struct delivery_node
{
                        short           w_id;
                        short           o_carrier_id;
                        SYSTEMTIME              queue_time;
                        long            tran_start_time;
                        struct          delivery_node
*next_delivery;
};

// Default loader arguments
#define BATCH           10000
#define DEFLDPACKSIZE           4096
#define ORDERS_PER_DIST         3000
#define LOADER_RES_FILE     "load.out"
#define LOADER_NURAND_C         123
#define DEF_STARTING_WAREHOUSE  1
#define CASE_SENSITIVITY        0

// String length constants
#define SERVER_NAME_LEN     20
#define DATABASE_NAME_LEN   20
#define USER_NAME_LEN       20
#define PASSWORD_LEN        20
#define TABLE_NAME_LEN      20
#define I_NAME_LEN          24
#define I_DATA_LEN          50
#define W_NAME_LEN          10
#define ADDRESS_LEN         20
#define STATE_LEN           2
#define ZIP_LEN             9
#define S_DIST_LEN          24
#define S_DATA_LEN          50
#define D_NAME_LEN          10
#define FIRST_NAME_LEN      16
#define MIDDLE_NAME_LEN     2
#define LAST_NAME_LEN       16
#define PHONE_LEN           16
#define DATETIME_LEN        30
#define CREDIT_LEN          2
#define C_DATA_LEN          250
#define H_DATA_LEN          24
#define DIST_INFO_LEN       24
#define MAX_OL_NEW_ORDER_ITEMS  15
#define MAX_OL_ORDER_STATUS_ITEMS 15
#define BRAND_LEN           1
#define STATUS_LEN          25
#define OL_DIST_INFO_LEN        24

// Transaction types
#define EMPTY           0
#define NEW_ORDER_TRAN          1
#define PAYMENT_TRAN        2
#define ORDER_STATUS_TRAN       3
#define DELIVERY_TRAN       4
#define STOCK_LEVEL_TRAN        5

// Statistic structures
typedef struct
{
    long                    tran_count;
    long                    total_time;
    long                    resp_time;
    long                    resp_min;
    long                    resp_max;
    long                    rolled_back;
    long                    tran_2sec;
    long                    tran_5sec;
    long                    tran_sqr;
                long                num_deadlocks;
    long                    resp_hist[HIST_MAX];
} TRAN_STATS;

typedef struct
{
    TRAN_STATS              NewOrderStats;
    TRAN_STATS              PaymentStats;
    TRAN_STATS              OrderStatusStats;
    TRAN_STATS              QueuedDeliveryStats;
                TRAN_STATS              DeliveryStats;
    TRAN_STATS              StockLevelStats;
} CLIENT_STATS;

// driver structures
typedef struct
{
    char                    *server;
    char                    *database;
    char                    *user;
    char                    *password;
    char                    *table;
    long                    num_warehouses;
    long                    batch;
    long                    verbose;
                            long                    pack_size;
                            char                    *loader_res_file;
                            char                    *synch_servername;
                            long                    case_sensitivity;
                            long                    starting_warehouse;
} TPCCLDR_ARGS;

typedef struct
{
    char                    *server;
    char                    *user;
    char                    *password;
                char                    *admin_database;
    char                    *sqlstat_filename;
                long                    run_id;
} SQLSTAT_ARGS;

typedef struct
{
    SQLCONN                 *sqlconn;
    char                    *server;
    char                    *database;
                char                    *admin_database;
    char                    *user;
    char                    *password;
    long                    ramp_up;
    long                    steady_state;
    long                    ramp_down;
    long                    num_users;
    long                    num_warehouses;
    long                    think_times;
    long                    display_data;
                long                    client_mode;
                long                    tran;
                long                    deadlock_retry;
                long                    delivery_backoff;
                long                    num_deliveries;
                char                    *comment;
                double                  load_multiplier;
                long                    checkpoint_interval;
                long                    first_checkpoint;
                long                    disable_90th;
                char                    *resfilename;
                char                    *sqlstat_filename;
                long                    enable_sqlstat;
                long                    sqlstat_period;
                long                    shutdown_server;
                long                    auto_run;
                long                    dropped_connections;
    short                   spid;
} MASTER_DATA;

typedef struct
{
                long                    num_threads;
    char                    *server;
    char                    *database;
                char                    *admin_database;
    char                    *user;
    char                    *password;
                long                    pack_size;
                short                   x_flag;
                char                    *synch_servername;
                long
                disable_delivery_resfiles;
#ifdef USE_CONMON
                HANDLE                  hConMon;
                short                   con_id;
```

```
                short           con_x;
                short           con_y;
#endif
} GLOBAL_CLIENT_DATA;

typedef struct
{
#ifdef USE_ODBC
                HDBC            hdbc;
                HSTMT                   hstmt;
#else
   SQLCONN                      *sqlconn;
#endif
                short           threadid;
   char                         *server;
   char                         *database;
                char            *admin_database;
   char                         *user;
   char                         *password;
                long            ramp_up;
                long            steady_state;
   long                         ramp_down;
   long                         num_warehouses;
                long            client_mode;
                long            tran;
                long            deadlock_retry;
                long            think_times;
                long            pack_size;
   long                         tran_start_time;
   long                         tran_end_time;
   long                         display_data;
                long            id;
                short           w_id;
   short                        spid;
                long            disable_90th;
                double                  load_multiplier;
                long            num_deliveries;

#ifdef USE_CONMON
                HANDLE                  hConMon;
                short           con_id;
                short           con_x;
                short           con_y;
                short           fTimerStat;
#endif
} CLIENT_DATA;

typedef struct
{
   long                                 o_id;
} DEL_ITEM;

typedef struct
{
#ifdef USE_ODBC
                HDBC            hdbc;
                HSTMT                   hstmt;
#else
                SQLCONN         *sqlconn;
#endif
                SYSTEMTIME                      queue_time;
                SYSTEMTIME
                completion_time;
   long                         tran_start_time;
   long                         tran_end_time;
                short           threadid;
                FILE            *fDelivery;
   short                        spid;
   short                        w_id;
                short           d_id;
                short           o_carrier_id;
                DEL_ITEM        DelItems[10];
                char            *server;
   char                         *database;
                char            *admin_database;
   char                         *user;
   char                         *password;
                long            ramp_up;
   long                         steady_state;
   long                         ramp_down;
                long            pack_size;
                long            id;
                long            disable_90th;
                long            delivery_backoff;
                long
                disable_delivery_resfiles;
} DELIVERY;

typedef struct
{
                long            pipe_num;
} DELIVERY_ARGS;

// transaction structures
typedef struct
{
   short                                ol_supply_w_id;
   long                                 ol_i_id;
   char                                 ol_i_name[I_NAME_LEN+1];
   short                                ol_quantity;
   char
                ol_brand_generic[BRAND_LEN+1];
   double                               ol_i_price;
   double                               ol_amount;
   short                                ol_stock;
   short                                num_warehouses;
} OL_NEW_ORDER_DATA;

typedef struct
{
   short        w_id;
   short        d_id;
   long         c_id;
   short        o_ol_cnt;
                char            c_last[LAST_NAME_LEN+1];
   char         c_credit[CREDIT_LEN+1];
   double       c_discount;
   double       w_tax;
   double       d_tax;
   long                 o_id;
                short           o_commit_flag;
#ifdef USE_ODBC
                TIMESTAMP_STRUCT        o_entry_d;
#else
                DBDATEREC       o_entry_d;
#endif
   short        o_all_local;
   double       total_amount;
                long            num_deadlocks;
                char
                execution_status[STATUS_LEN];
   OL_NEW_ORDER_DATA ol[MAX_OL_NEW_ORDER_ITEMS];
} NEW_ORDER_DATA;

typedef struct
{
   short                w_id;
   short                d_id;
   long                 c_id;
   short                c_d_id;
   short                c_w_id;
   double               h_amount;
#ifdef USE_ODBC
                TIMESTAMP_STRUCT        h_date;
#else
   DBDATEREC            h_date;
#endif
   char                 w_street_1[ADDRESS_LEN+1];
   char                 w_street_2[ADDRESS_LEN+1];
   char                 w_city[ADDRESS_LEN+1];
   char                 w_state[STATE_LEN+1];
   char                 w_zip[ZIP_LEN+1];
   char                 d_street_1[ADDRESS_LEN+1];
   char                 d_street_2[ADDRESS_LEN+1];
   char                 d_city[ADDRESS_LEN+1];
   char                 d_state[STATE_LEN+1];
   char                 d_zip[ZIP_LEN+1];
   char                 c_first[FIRST_NAME_LEN+1];
   char                 c_middle[MIDDLE_NAME_LEN +
1];
   char                 c_last[LAST_NAME_LEN+1];
   char                 c_street_1[ADDRESS_LEN+1];
   char                 c_street_2[ADDRESS_LEN+1];
   char                 c_city[ADDRESS_LEN+1];
   char                 c_state[STATE_LEN+1];
   char                 c_zip[ZIP_LEN+1];
   char                 c_phone[PHONE_LEN+1];
#ifdef USE_ODBC
                TIMESTAMP_STRUCT        c_since;
#else
   DBDATEREC                    c_since;
#endif
   char                 c_credit[CREDIT_LEN+1];
   double               c_credit_lim;
   double               c_discount;
   double               c_balance;
   char                 c_data[200+1];
                long                    num_deadlocks;
                char
execution_status[STATUS_LEN];
} PAYMENT_DATA;

typedef struct
{
                long                    ol_i_id;
   short                ol_supply_w_id;
   short                ol_quantity;
   double               ol_amount;
#ifdef USE_ODBC
                TIMESTAMP_STRUCT        ol_delivery_d;
#else
   DBDATEREC                    ol_delivery_d;
#endif
} OL_ORDER_STATUS_DATA;

typedef struct
{
   short        w_id;
   short        d_id;
   long         c_id;
   char         c_first[FIRST_NAME_LEN+1];
   char         c_middle[MIDDLE_NAME_LEN+1];
   char         c_last[LAST_NAME_LEN+1];
   double        c_balance;
```

```
    long            o_id;
#ifdef USE_ODBC
                    TIMESTAMP_STRUCT    o_entry_d;
#else
    DBDATEREC       o_entry_d;
#endif
    short           o_carrier_id;
    OL_ORDER_STATUS_DATA
OlOrderStatusData[MAX_OL_ORDER_STATUS_ITEMS];
    short           o_ol_cnt;
                    long            num_deadlocks;
                    char
execution_status[STATUS_LEN];
} ORDER_STATUS_DATA;

typedef struct
{
    short                   w_id;
    short                   o_carrier_id;
    SYSTEMTIME              queue_time;
                    long            num_deadlocks;
                    DEL_ITEM                DelItems[10];
                    char
                    execution_status[STATUS_LEN];
} DELIVERY_DATA;


typedef struct
{
    short                           w_id;
    short                           d_id;
    short                           thresh_hold;
    long                            low_stock;
                    long                    num_deadlocks;
                    char
                    execution_status[STATUS_LEN];
} STOCK_LEVEL_DATA;


// For client synchronization
#define LINE_LEN   80
#define NAME_SIZE  25
#define IN_BUF_SIZE     1000
#define OUT_BUF_SIZE    1000
#define TIME_OUT        0
#define PLEASE_READ     1000
#define PLEASE_WRITE    1000

typedef struct _WRTHANDLE
{                   HANDLE          hPipe;
    DWORD                   threadID;
                    CHAR        Name[NAME_SIZE];
                    struct      _WRTHANDLE *    next;
}WRTHANDLE;

// For client console monitor
#ifdef USE_CONMON
#define                 CON_LINE_SIZE           40
#define                 DEADLOCK_X                      17
#define DEADLOCK_Y              4
#define CUR_STATE_X             15
#define CUR_STATE_Y             3
#define                 YELLOW                          0
#define                 RED                             1
#define                 GREEN                           2
int                             total_deadlocks;
#endif
```

```
// Functions in random.c
void                seed();
long                irand();
double              drand();
void                WUCreate();
short               WURand();

// Functions in getargs.c;
void                GetArgsLoader();
void                GetArgsLoaderUsage();
void                GetArgsMaster();
void                GetArgsMasterUsage();
void                GetArgsClient();
void                GetArgsClientUsage();
void                GetArgsDelivery();
void                GetArgsDeliveryUsage();
void                GetArgsSQLStat();
void                GetArgsSQLStatUsage();

// Functions in master.c
void                ReadClientDone();
BOOL    CtrlHandler();

// Functions in client.c
void                ClientMain();
void                DeliveryMain();
void                Delivery();
void                ClientEmulate();
short               ClientSelectTransaction();
void                ClientShuffleDeck();

//Functions in tran.c
BOOL                TranNewOrder();
BOOL                TranPayment();
BOOL                TranOrderStatus();
BOOL                TranDelivery();
BOOL                TranStockLevel();

// Functions in data.c
void                DataNewOrder();
void                DataPayment();
void                DataOrderStatus();
void                DataDelivery();
void                DataStockLevel();
short               DataRemoteWarehouse();

// Functions in time.c
long                TimeNow();
void                TimeInit();
void                TimeKeying();
void                TimeThink();

// Functions in stats.c
void                StatsInit();
void                StatsInitTran();
void                StatsGeneral();
void                StatsDelivery();

// Functions in sqlfuncs.c
BOOL                SQLExec();
BOOL                SQLExecCmd();
BOOL                SQLOpenConnection();
void                SQLClientInit();
int                 SQLMasterInit();
void                SQLDeliveryInit();
int                 SQLClientStats();
int                 SQLDeliveryStats();
void                SQLTranStats();
```

```
void                SQLMasterStats();
void                SQLMasterTranStats();
void                SQLIOStats();
void                SQLCheckpointStats();
void                SQLInitResFile();
void                SQLGetRunId();
BOOL                SQLNewOrder();
BOOL                SQLPayment();
BOOL                SQLOrderStatus();
BOOL                SQLStockLevel();
void    SQLDelivery();
int                 SQLGetCustId();
void                SQLExit();
void                SQLInit();
void                SQLInitPrivate();
void                SQLClientInitPrivate();
void                SQLDeliveryInitPrivate();
int                 SQLMsgHandler();
int                 SQLErrHandler();
int                 SQLClientMsgHandler();
int                 SQLClientErrHandler();
int                 SQLDeliveryMsgHandler();
int                 SQLDeliveryErrHandler();
void                SQLInitDate();
void                SQLShutdown();
#ifdef USE_ODBC
void                ODBCOpenConnection();
void                ODBCOpenDeliveryConnection();
BOOL                ODBCError();
void                ODBCExit();
#endif

// Functions in util.c
void                UtilSleep();
void                UtilPrintNewOrder();
void                UtilPrintPayment();
void                UtilPrintOrderStatus();
void                UtilPrintDelivery();
void                UtilPrintStockLevel();
void                UtilPrintOlTable();
void                UtilError();
void                UtilFatalError();
void                UtilStrCpy();
#ifdef USE_CONMON
void                WriteConsoleString();
#endif
void                WriteDeliveryString();
BOOL                AddDeliveryQueueNode();
BOOL                GetDeliveryQueueNode();


// Functions in strings.c
void                MakeAddress();
void                LastName();
int                 MakeAlphaString();
int                 MakeOriginalAlphaString();
int                 MakeNumberString();
int                 MakeZipNumberString();
void                InitString();
void                InitAddress();
void                PaddString();

// Functions in delivery.c
void                DeliveryHMain();
void                DeliveryH();

#endif __damien_tpcc_h__
```

NEWORDER.C

```
/* Audited: 28 February 1997 */

/* neworder.c
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#include "neworder.h"

int new_order_func_parse(assoc *a, int *cookie, NEW_ORDER_DATA
*data, char *output) {
                        int i = 0;
                        int n = 0;
                        neworder_line lines[15];
                        char errstr[128];
                        char all_errors[1024];
                        errstr[0] = '\0';
                        all_errors[0] = '\0';
                        for(n = 0; n < 15; n++) {
                                lines[n].supply_w_id = -3;
                                lines[n].item_id = -3;
                                lines[n].quantity = -3;
                        }
                        while((*a)[0][i]) {
                                switch((*a)[0][i][0]) {
                                        case 'c':
                                                *cookie = VerifyInt((*a)[1][i],
4);
                                                break;
                                        case 'D':
                                                data->d_id =
VerifyShort((*a)[1][i], 2);
                                                break;
                                        case 'C':
                                                if((*a)[0][i][1] == 'l') {
                                                        data->c_id =
VerifyLong((*a)[1][i], 4);
                                                }
                                                break;
                                        case 'O':
                                                n = atoi(&((*a)[0][i][2]));
                                                if(n < 1 || n > 15) break;
                                                switch((*a)[0][i][1]) {
                                                        case 'S':
                                                                lines[n - 1].swid
= (*a)[1][i];
                                                                lines[n -
1].supply_w_id = VerifyShort(lines[n - 1].swid, 4);
                                                                break;
                                                        case 'I':
                                                                lines[n - 1].iid =
(*a)[1][i];
                                                                lines[n -
1].item_id = VerifyLong(lines[n - 1].iid, 6);
                                                                break;
                                                        case 'Q':
                                                                lines[n -
1].quan = (*a)[1][i];
                                                                lines[n -
1].quantity = VerifyShort(lines[n - 1].quan, 2);
                                                                break;
                                                        default: break;
                                                }
                                                break;
                                        default: break;
                                }
                                ++i;
                        }
                        if(*cookie < 0 || !get_user(*cookie)->w_id) {
                                sprintf(errstr, BAD_COOKIE_MSG);
                                strcat(all_errors, errstr);
                        }
                        switch(data->d_id) {
                                case -1:
                                        sprintf(errstr, TOO_LONG_MSG,
"District ID", 2);
                                        strcat(all_errors, errstr);
                                        break;
                                case -2:
                                        sprintf(errstr, NOT_ISDIGIT_MSG,
"District ID");
                                        strcat(all_errors, errstr);
                                        break;
                                case -3:
                                        sprintf(errstr, NO_INPUT_MSG,
"District ID");
                                        strcat(all_errors, errstr);
                                        break;
                                default:break;
                        }
                        switch(data->c_id) {
                                case -1:
                                        sprintf(errstr, TOO_LONG_MSG,
"Customer ID", 2);
                                        strcat(all_errors, errstr);
                                        break;
                                case -2:
                                        sprintf(errstr, NOT_ISDIGIT_MSG,
"Customer ID");
                                        strcat(all_errors, errstr);
                                        break;
                                case -3:
                                        sprintf(errstr, NO_INPUT_MSG,
"Customer ID");
                                        strcat(all_errors, errstr);
                                        break;
                                default:break;
                        }
                        data->o_ol_cnt = 0;
                        for(n = 0; n < 15; n++) {
                                if(lines[n].supply_w_id == -3 &&
lines[n].item_id == -3 && lines[n].quantity == -3)
                                        continue;
                                switch(lines[n].supply_w_id) {
                                        case -1:
                                                sprintf(errstr,
NLINE_TOO_LONG, n + 1, "Supply Warehouse ID", 4);
                                                strcat(all_errors, errstr);
                                                break;
                                        case -2:
                                                sprintf(errstr,
NLINE_NOT_ISDIGIT, n + 1, "Supply Warehouse ID");
                                                strcat(all_errors, errstr);
                                                break;
                                        case -3:
                                                sprintf(errstr,
NLINE_NO_INPUT, n + 1, "Supply Warehouse ID");
                                                strcat(all_errors, errstr);
                                                break;
                                        default: break;
                                }
                                switch(lines[n].item_id) {
                                        case -1:
                                                sprintf(errstr,
NLINE_TOO_LONG, n + 1, "Item ID", 6);
                                                strcat(all_errors, errstr);
                                                break;
                                        case -2:
                                                sprintf(errstr,
NLINE_NOT_ISDIGIT, n + 1, "Item ID");
                                                strcat(all_errors, errstr);
                                                break;
                                        case -3:
                                                sprintf(errstr,
NLINE_NO_INPUT, n + 1, "Item ID");
                                                strcat(all_errors, errstr);
                                                break;
                                        default: break;
                                }
                                switch(lines[n].quantity) {
                                        case -1:
                                                sprintf(errstr,
NLINE_TOO_LONG, n + 1, "Quantity", 2);
                                                strcat(all_errors, errstr);
                                                break;
                                        case -2:
                                                sprintf(errstr,
NLINE_NOT_ISDIGIT, n + 1, "Quantity");
                                                strcat(all_errors, errstr);
                                                break;
                                        case -3:
                                                sprintf(errstr,
NLINE_NO_INPUT, n + 1, "Quantity");
                                                strcat(all_errors, errstr);
                                                break;
                                        default: break;
                                }
                                data->Ol[data->o_ol_cnt].ol_supply_w_id
= lines[n].supply_w_id;
                                data->Ol[data->o_ol_cnt].ol_i_id =
lines[n].item_id;
                                data->Ol[data->o_ol_cnt].ol_quantity =
lines[n].quantity;
                                data->o_ol_cnt++;
                        }
                        data->w_id = get_user(*cookie)->w_id;
                        data->o_all_local = 1;
                        for(i = 0; i < data->o_ol_cnt; i++) {
                                if(data->Ol[i].ol_supply_w_id != data-
>w_id) {
                                        data->o_all_local = 0;
                                        break;
                                }
                        }
                        if(all_errors[0]) {
                                sprintf(output, errorpage, all_errors);
                                return 0;
                        } else return 1;
}

int new_order_func_process(NEW_ORDER_DATA *data, int cookie) {
#ifdef DB_PRESENT
                        return SQLNewOrder(get_user(cookie)-
>dbhandle, data, DEADLOCK_RETRY);
#else
                        int x;
                        data->o_id = 0;
                        data->o_commit_flag = 1;
                        data->o_entry_d.day = 15;
                        data->o_entry_d.month = 4;
                        data->o_entry_d.year = 1996;
```

```
                    data->o_entry_d.hour = 17;
                    data->o_entry_d.minute = 21;
                    data->o_entry_d.second = 49;
                    strcpy(data->c_last, "Johnson");
                    strcpy(data->c_credit, "B5");
                    data->c_discount = 0.2135;
                    data->w_tax = 8.00;
                    data->d_tax = 3.50;
                    data->o_id = 54658876L;
                    strcpy(data->execution_status, "This thing is
executing");
                    for(x = 0; x < data->o_ol_cnt; x++) {
                         if(data->Ol[x].ol_supply_w_id) {
                              strcpy(data->Ol[x].ol_i_name,
"Some kind of widget");
                              strcpy(data-
>Ol[x].ol_brand_generic, "B");
                              data->Ol[x].ol_i_price = 12.34;
                              data->Ol[x].ol_amount = data-
>Ol[x].ol_quantity * data->Ol[x].ol_i_price;
                              data->total_amount += data-
>Ol[x].ol_amount;
                              data->Ol[x].ol_stock = 321;
                              data->Ol[x].num_warehouses = 5;
                         }
                    }
                    return 1;
#endif DB_PRESENT
}

void new_order_func_format(char *output, NEW_ORDER_DATA *data, int
cookie) {
                    char buf[3000];
                    int x;
                    data->c_discount *= 100;
                    data->w_tax *= 100;
                    data->d_tax *= 100;
                    sprintf(buf, nresp, cookie);
                    for(x = 0; x < data->o_ol_cnt; x++)
                         NewOrderLine(&buf[nlin[x]], &data->Ol[x]);
                    for(; x < 15; x++)
                         AlphaField(&buf[nlin[x]], 78, " ");
                    IntField(&buf[NW], 4, data->w_id);
                    IntField(&buf[ND], 2, data->d_id);
                    DateTimeField(&buf[NDAT], &data->o_entry_d);
                    IntField(&buf[NC], 4, data->c_id);
                    AlphaField(&buf[NN], 16, data->c_last);
                    AlphaField(&buf[NCRED], 2, data->c_credit);
                    DecField(&buf[NDISC], 5, data->c_discount);
                    IntField(&buf[NON], 8, data->o_id);
                    IntField(&buf[NOL], 2, data->o_ol_cnt);
                    DecField(&buf[NWT], 5, data->w_tax);
                    DecField(&buf[NDT], 5, data->d_tax);
                    AlphaField(&buf[NEX], 24, data-
>execution_status);
                    DecField(&buf[NTOT], 8, data->total_amount);
                    FormatHtmlPage(buf, output);
}

void new_order_func_error(char *output, NEW_ORDER_DATA *data, int
cookie) {
                    int x;
                    char buf[3000];
                    sprintf(buf, nresp, cookie);
                    for(x = 0; x < 15; x++)
                         AlphaField(&buf[nlin[x]], 78, " ");
                    IntField(&buf[NW], 4, data->w_id);
                    IntField(&buf[ND], 2, data->d_id);
                    AlphaField(&buf[NDAT], 19, " ");
                    IntField(&buf[NC], 4, data->c_id);
                    AlphaField(&buf[NN], 16, data->c_last);
                    AlphaField(&buf[NCRED], 2, data->c_credit);
                    AlphaField(&buf[NDISC], 5, " ");
                    IntField(&buf[NON], 8, data->o_id);
                    AlphaField(&buf[NOL], 2, " ");
                    AlphaField(&buf[NWT], 5, " ");
                    AlphaField(&buf[NDT], 5, " ");
                    AlphaField(&buf[NEX], 24, data-
>execution_status);
                    AlphaField(&buf[NTOT], 8, " ");
                    FormatHtmlPage(buf, output);
}

void new_order_func_main(assoc *a, char *output) {
                    int cookie;
                    NEW_ORDER_DATA data;
                    if(!new_order_func_parse(a, &cookie, &data,
output)) return;
                    if(!new_order_func_process(&data, cookie)) {
                         new_order_func_error(output, &data,
cookie);
                         return;
                    }
                    new_order_func_format(output, &data, cookie);
}
```

NEWORDER.H

```
/* Audited: 28 February 1997 */

/* neworder.h
  Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#ifndef __neworder_h__
#define __neworder_h__

#include "context.h"
#include <tpcc/kit/src/tpcc.h>
#include "inputparser.h"
#include "output.h"
#include "errors.h"
#include "options.h"

#define NEWORDER_FUNC 8

typedef struct {
                    char *swid;
                    char *iid;
                    char *quan;
                    short supply_w_id;
                    long  item_id;
                    short quantity;
} neworder_line;

#define NLINE_TOO_LONG "o Order Line %d: The \"%s\" field contains too
many characters.\r\n  The maximum is %d.\r\n"
#define NLINE_NOT_ISDIGIT "o Order Line %d: The \"%s\" field contained
nondigit characters.\r\n"
#define NLINE_NO_INPUT "o Order Line %d: You did not fill in the \"%s\"
field.\r\n  The field is required.\r\n"

static char nresp[] =
"<HTML><HEAD><TITLE>TPC-C: New
Order</TITLE></HEAD><BODY><PRE>"
"                              New Order\r\n"
"Warehouse: XXXX   District: XX                    Date:
XXXXXXXXXXXXXXXXXXX\r\n"
"Customer:  XXXX   Name: XXXXXXXXXXXXXXXX   Credit: XX   %%Disc:
XXXXX\r\n"
"Order Number: XXXXXXXX  Number of Lines: XX       W_tax: XXXXX
D_tax: XXXXX\r\n"
"\r\n"
" Supp_W  Item_Id  Item Name              Qty  Stock  B/G  Price
Amount\r\n"
"  XXXX   XXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX  XX   XXX   X
$XXXXXX  $XXXXXXX\r\n"
"  XXXX   XXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX  XX   XXX   X
$XXXXXX  $XXXXXXX\r\n"
"  XXXX   XXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX  XX   XXX   X
$XXXXXX  $XXXXXXX\r\n"
"  XXXX   XXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX  XX   XXX   X
$XXXXXX  $XXXXXXX\r\n"
"  XXXX   XXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX  XX   XXX   X
$XXXXXX  $XXXXXXX\r\n"
"  XXXX   XXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX  XX   XXX   X
$XXXXXX  $XXXXXXX\r\n"
"  XXXX   XXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX  XX   XXX   X
$XXXXXX  $XXXXXXX\r\n"
"  XXXX   XXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX  XX   XXX   X
$XXXXXX  $XXXXXXX\r\n"
"  XXXX   XXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX  XX   XXX   X
$XXXXXX  $XXXXXXX\r\n"
"  XXXX   XXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX  XX   XXX   X
$XXXXXX  $XXXXXXX\r\n"
"  XXXX   XXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX  XX   XXX   X
$XXXXXX  $XXXXXXX\r\n"
"  XXXX   XXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX  XX   XXX   X
$XXXXXX  $XXXXXXX\r\n"
"  XXXX   XXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX  XX   XXX   X
$XXXXXX  $XXXXXXX\r\n"
"  XXXX   XXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX  XX   XXX   X
$XXXXXX  $XXXXXXX\r\n"
"  XXXX   XXXXXX   XXXXXXXXXXXXXXXXXXXXXXXX  XX   XXX   X
$XXXXXX  $XXXXXXX\r\n"
"Execution Status: XXXXXXXXXXXXXXXXXXXXXXXX                 Total:
$XXXXXXXX"
"</PRE><P><FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">"
"<INPUT TYPE=\"hidden\" NAME=\"c\" VALUE=\"%d\">"
"<INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"New Order\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Payment\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Delivery\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Order-Status\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Stock-Level\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Exit\">"
"</FORM></P></BODY></HTML>\r\n";

#define NW 118
#define ND 135
#define NDAT 167
#define NC 199
#define NN 212
#define NCRED 239
#define NDISC 251
#define NON 272
#define NOL 299
#define NWT 316
#define NDT 331
static int nlin[15] = {418, 498, 578, 658, 738, 818, 898, 978, 1058, 1138,
1218, 1298, 1378, 1458, 1538};
#define NEX 1636
```

```
#define NTOT 1688

extern void e_log(char *);
void new_order_func_main(assoc *, char *);
int new_order_func_parse(assoc *, int *, NEW_ORDER_DATA *, char *);
int new_order_func_process(NEW_ORDER_DATA *, int);
void new_order_func_format(char *, NEW_ORDER_DATA *, int);
void new_order_func_error(char *, NEW_ORDER_DATA *, int);

#endif __neworder_h__
```

OPTIONS.C

```
/* Audited: 28 February 1997 */

/* options.c
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#include "options.h"


void GetRegistryValues(void) {
                        int i;
                        DWORD how;
                        HKEY hRegKey;
                        int def_index = 0;
                        int def_warehouse = 100;
                        DWORD type;
                        DWORD size = (DWORD)32;
                        union dtg {BYTE b[32]; char c[32]; DWORD d[8];}
data;

                        RegOpenKeyEx(HKEY_LOCAL_MACHINE,
"SOFTWARE", 0, KEY_READ | KEY_WRITE, &hRegKey);
                        RegCreateKeyEx(hRegKey, "Intergraph", 0,
NULL, REG_OPTION_NON_VOLATILE, KEY_READ | KEY_WRITE, NULL,
&hRegKey, &how);
                        RegCreateKeyEx(hRegKey, "TPC-C ISAPI
Application", 0, NULL, REG_OPTION_NON_VOLATILE, KEY_READ |
KEY_WRITE, NULL, &hRegKey, &how);
                        if(how == REG_CREATED_NEW_KEY) {
                                RegSetValueEx(hRegKey, "ServerName",
0, REG_SZ, (const unsigned char *)"SERVER", 7);
                                RegSetValueEx(hRegKey, "TokenIndex",
0, REG_DWORD, (const unsigned char *)&def_index, 4);
                                RegSetValueEx(hRegKey,
"NumWarehouses", 0, REG_DWORD, (const unsigned char
*)&def_warehouse, 4);
                        }

                        for(i = 0; i < 8; data.d[i++] = (DWORD)0);
                        RegQueryValueEx(hRegKey, "ServerName", 0,
&type, (unsigned char *)&data.b, &size);
                        strcpy(SERVERNAME, data.c);
                        size = (DWORD)32;
                        for(i = 0; i < 8; data.d[i++] = (DWORD)0);
                        RegQueryValueEx(hRegKey, "TokenIndex", 0,
&type, (unsigned char *)&data.b, &size);
                        TokenIndex = data.d[0];
                        size = (DWORD)32;
                        for(i = 0; i < 8; data.d[i++] = (DWORD)0);
                        RegQueryValueEx(hRegKey,
"NumWarehouses", 0, &type, (unsigned char *)&data.b, &size);
                        MAXWH = data.d[0];
}
```

OPTIONS.H

```
/* Audited: 28 February 1997 */

/* options.h
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#ifndef __options_h__
#define __options_h__

#define MAX_USERS 4000
#define DB_PRESENT
#define USEDB "tpcc"
#define USERNAME "sa"
#define USERPASSWD ""
//#define SERVERNAME "SPAT"
//#define MAXWH 10
//#define TokenIndex 0


extern char                     SERVERNAME[32];
extern int                      MAXWH;
extern int                      TokenIndex;

char SERVERNAME[32];
int MAXWH;
int TokenIndex;

/* Do not modify anything below this point. */
#include <windows.h>
#include <winreg.h>
#include <stdio.h>

void GetRegistryValues(void);


#endif __options_h__
```

ORDERSTATUS.C

```
/* Audited: 28 February 1997 */

/* orderstatus.c
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#include "orderstatus.h"

int order_status_func_parse(assoc *a, int *cookie, ORDER_STATUS_DATA
*data, char *output) {
                        int i = 0;
                        int cid = 0;
                        char errstr[128];
                        char all_errors[1024];
                        errstr[0] = '\0';
                        all_errors[0] = '\0';
                        data->c_last[0] = '\0';
                        while((*a)[0][i]) {
                                switch((*a)[0][i][0]) {
                                        case 'c':
                                                *cookie = VerifyInt((*a)[1][i],
4);
                                                break;
                                        case 'C':
                                                switch((*a)[0][i][1]) {
                                                        case 'I':
                                                                if(strlen((*a)[1][i])) cid++;
                                                                data->c_id =
VerifyLong((*a)[1][i], 4);
                                                                break;
                                                        case 'L':
                                                                if(strlen((*a)[1][i])) cid++;
                                                                if(VerifyString((*a)[1][i], 16))
                                                                strcpy(data->c_last, (*a)[1][i]);
                                                                break;
                                                        default: break;
                                                }
                                                break;
                                        case 'D':
                                                data->d_id =
VerifyShort((*a)[1][i], 2);
                                                break;
                                        default: break;
                                }
                                ++i;
                        }
                        if(*cookie < 0 || !get_user(*cookie)->w_id) {
                                sprintf(errstr, BAD_COOKIE_MSG);
                                strcat(all_errors, errstr);
                        }
                        switch(data->d_id) {
                                case -1:
                                        sprintf(errstr, TOO_LONG_MSG,
"District ID", 2);
                                        strcat(all_errors, errstr);
                                        break;
                                case -2:
                                        sprintf(errstr, NOT_ISDIGIT_MSG,
"District ID");
                                        strcat(all_errors, errstr);
                                        break;
                                case -3:
                                        sprintf(errstr, NO_INPUT_MSG,
"District ID");
                                        strcat(all_errors, errstr);
                                        break;
                                default:break;
                        }
                        if(cid != 1)
                                strcat(all_errors, "o You must fill in one
(and only one) of \"Customer ID\" and \"Customer Last Name\".\r\n");
                                else if(!data->c_last[0]) {
                                        switch(data->c_id) {
                                                case -1:
                                                        sprintf(errstr,
TOO_LONG_MSG, "Customer ID", 4);
                                                        strcat(all_errors, errstr);
                                                        break;
                                                case -2:
                                                        sprintf(errstr,
NOT_ISDIGIT_MSG, "Customer ID");
                                                        strcat(all_errors, errstr);
                                                        break;
                                                case -3:
```

```
                                        sprintf(errstr,
TOO_LONG_MSG, "Customer Last Name", 16);
                                        strcat(all_errors, errstr);
                                        break;
                                default: break;
                        }
                }
                data->w_id = get_user(*cookie)->w_id;
                if(all_errors[0]) {
                        sprintf(output, errorpage, all_errors);
                        return 0;
                } else return 1;
}

int order_status_func_process(ORDER_STATUS_DATA *data, int cookie) {
#ifdef DB_PRESENT
                return SQLOrderStatus(get_user(cookie)-
>dbhandle, data, DEADLOCK_RETRY);
#else
                int i;
                if(!data->c_last[0]) strcpy(data->c_last,
"Johnson");
                else data->c_id = 123;
                strcpy(data->c_first, "Frederick");
                strcpy(data->c_middle, "J.");
                data->o_entry_d.day = 15;
                data->o_entry_d.month = 4;
                data->o_entry_d.year = 1996;
                data->o_entry_d.hour = 11;
                data->o_entry_d.minute = 37;
                data->o_entry_d.second = 25;
                data->c_balance = -12345.67;
                data->o_carrier_id = 16;
                data->o_id = 87654321;
                data->o_ol_cnt = 15;
                for(i = 0; i < 15; i++) {
                        data-
>OlOrderStatusData[i].ol_supply_w_id = 5423;
                        data->OlOrderStatusData[i].ol_i_id = 863;
                        data->OlOrderStatusData[i].ol_quantity =
6;
                        data->OlOrderStatusData[i].ol_amount =
0.50;
                        data-
>OlOrderStatusData[i].ol_delivery_d.day = 21;
                        data-
>OlOrderStatusData[i].ol_delivery_d.month = 11;
                        data-
>OlOrderStatusData[i].ol_delivery_d.year = 1996;
                }
                return 1;
#endif DB_PRESENT
}

void order_status_func_format(char *output, ORDER_STATUS_DATA
*data, int cookie) {
                int x;
                char buf[3000];
                sprintf(buf, oresp, cookie);
                IntField(&buf[OW], 4, data->w_id);
                IntField(&buf[OD], 2, data->d_id);
                IntField(&buf[OC], 4, data->c_id);
                AlphaField(&buf[OF], 16, data->c_first);
                AlphaField(&buf[OM], 2, data->c_middle);
                AlphaField(&buf[OL], 16, data->c_last);
                SignedDecField(&buf[OBAL], 9, data-
>c_balance);
                IntField(&buf[ONUM], 8, data->o_id);
                DateTimeField(&buf[ODAT], &data->o_entry_d);
                IntField(&buf[OCAR], 2, data->o_carrier_id);
                for(x = 0; x < data->o_ol_cnt; x++)
                        OrderStatusLine(&buf[olin[x]], &data-
>OlOrderStatusData[x]);
                for(;x < 15; x++)
                        AlphaField(&buf[olin[x]], 56, " ");
                FormatHtmlPage(buf, output);
}

void order_status_func_main(assoc *a, char *output) {
                int cookie;
                ORDER_STATUS_DATA data;
                if(!order_status_func_parse(a, &cookie, &data,
output)) return;
                if(!order_status_func_process(&data, cookie)) {
                        sprintf(output, dberrpage, cookie);
                        return;
                }
                order_status_func_format(output, &data, cookie);
}
```

ORDERSTATUS.H

```
/* Audited: 28 February 1997 */

/* orderstatus.h
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#ifndef __orderstatus_h__
#define __orderstatus_h__

#include "context.h"
#include <tpcc/kit/src/tpcc.h>
#include "inputparser.h"
#include "output.h"
#include "errors.h"
#include "options.h"

#define ORDERSTAT_FUNC 7

static char oresp[] =
"<HTML><HEAD><TITLE>TPC-C: Order-
Status</TITLE></HEAD><BODY><PRE>"
"                              Order-Status\r\n"
"Warehouse: XXXX   District: XX\r\n"
"Customer: XXXX   Name: XXXXXXXXXXXXXXXX XX
XXXXXXXXXXXXXXXX\r\n"
"Cust-Balance: $XXXXXXXXX\r\n"
"\r\n"
"Order-Number: XXXXXXXX   Entry-Date: XXXXXXXXXXXXXXXXXXX
Carrier-Number: XX\r\n"
"Supply-W     Item-Id    Qty     Amount      Delivery-Date\r\n"
"  XXXX       XXXXXX     XX     $XXXXXXXX      XXXXXXXXXX\r\n"
"  XXXX       XXXXXX     XX     $XXXXXXXX      XXXXXXXXXX\r\n"
"  XXXX       XXXXXX     XX     $XXXXXXXX      XXXXXXXXXX\r\n"
"  XXXX       XXXXXX     XX     $XXXXXXXX      XXXXXXXXXX\r\n"
"  XXXX       XXXXXX     XX     $XXXXXXXX      XXXXXXXXXX\r\n"
"  XXXX       XXXXXX     XX     $XXXXXXXX      XXXXXXXXXX\r\n"
"  XXXX       XXXXXX     XX     $XXXXXXXX      XXXXXXXXXX\r\n"
"  XXXX       XXXXXX     XX     $XXXXXXXX      XXXXXXXXXX\r\n"
"  XXXX       XXXXXX     XX     $XXXXXXXX      XXXXXXXXXX\r\n"
"  XXXX       XXXXXX     XX     $XXXXXXXX      XXXXXXXXXX\r\n"
"  XXXX       XXXXXX     XX     $XXXXXXXX      XXXXXXXXXX\r\n"
"  XXXX       XXXXXX     XX     $XXXXXXXX      XXXXXXXXXX\r\n"
"  XXXX       XXXXXX     XX     $XXXXXXXX      XXXXXXXXXX\r\n"
"  XXXX       XXXXXX     XX     $XXXXXXXX      XXXXXXXXXX\r\n"
"  XXXX       XXXXXX     XX     $XXXXXXXX      XXXXXXXXXX"
"</PRE><P><FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">"
"<INPUT TYPE=\"hidden\" NAME=\"c\" VALUE=\"%d\">"
"<INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"New Order\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Payment\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Delivery\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Order-Status\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Stock-Level\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Exit\">"
"</FORM></P></BODY></HTML>\r\n";

#define OW 123
#define OD 140
#define OC 154
#define OF 167
#define OM 184
#define OL 187
#define OBAL 220
#define ONUM 247
#define ODAT 270
#define OCAR 308
static int olin[15] = {371, 429, 487, 545, 603, 661, 719, 777, 835, 893, 951,
1009, 1067, 1125, 1183};

extern void e_log(char *);
void order_status_func_main(assoc *, char *);
int order_status_func_parse(assoc *, int *, ORDER_STATUS_DATA *, char
*);
int order_status_func_process(ORDER_STATUS_DATA *, int);
void order_status_func_format(char *, ORDER_STATUS_DATA *, int);

#endif __orderstatus_h__
```

OUTPUT.C

```
/* Audited: 28 February 1997 */

/* output.c
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#include "output.h"

void AlphaField(char *field, int field_size, char *string)
{
                int i;
                for (i=0;i<field_size;i++)
                {
                        if (string[i] == '\0') break;//end of string
                        field[i] = string[i];
                }
                for (;i<field_size;i++) field[i] = ' '; //space fill the
field
}

void IntField(char *field, int field_size, int value)
{
                int i;
                for (i=field_size-1;i>=0;i--)
                {
                        field[i] = (value % 10) + '0';
                        value /= 10;
                }
}
```

```
void DecField(char *field, int field_size, double value)
{
        int dec, sign, i;
        char *string;
        string = _ecvt(value, field_size-1, &dec, &sign);
        for (i=0;i<(field_size-3)-(dec>=0?dec:0);i++)
        {
                field[i] = '0';
        }
        for (;i<(field_size-3);i++)
        {
                field[i] = *(string++);
        }
        field[i] = '.';
        i++;
        for (;dec<0&&i<field_size;dec++,i++)
        {
                field[i] = '0';
        }
        for (;i<field_size;i++)
        {
                field[i] = *(string++);
        }
}

void SignedDecField(char *field, int field_size, double value)
{
        if (value >= 0.0) {
                field[0] = ' ';
                DecField(&field[1],field_size-1,value);
        } else {
                field[0] = '-';
                DecField(&field[1],field_size-1,-value);
        }
}

void DateField(char *field, DBDATEREC *date)
{
        IntField(field,2,date->day);
        field[2] = '-';
        IntField(&field[3],2,date->month);
        field[5] = '-';
        IntField(&field[6],4,date->year);
}

void DateTimeField(char *field, DBDATEREC *date)
{
        DateField(field,date);
        field[10] = ' ';
        IntField(&field[11],2,date->hour);
        field[13] = ':';
        IntField(&field[14],2,date->minute);
        field[16] = ':';
        IntField(&field[17],2,date->second);
}

void ZipField(char *field, char *zip)
{
        AlphaField(field,5,zip);
        field[5] = '-';
        AlphaField(&field[6],4,&zip[5]);
}

void PhoneField(char *field, char *phone)
{
        AlphaField(field,6,phone);
        field[6] = '-';
        AlphaField(&field[7],3,&phone[6]);
        field[10] = '-';
        AlphaField(&field[11],3,&phone[9]);
        field[14] = '-';
        AlphaField(&field[15],4,&phone[12]);
}

BOOL NewOrderLine(char *field, OL_NEW_ORDER_DATA *oline)
{
        if(!oline->ol_i_id) {
                AlphaField(field, 78, " ");
                return FALSE;
        } else {
                IntField(&field[2], 4, oline-
>ol_supply_w_id);
                IntField(&field[9], 6, oline->ol_i_id);
                AlphaField(&field[18], 24, oline-
>ol_i_name);
                IntField(&field[44], 2, oline->ol_quantity);
                IntField(&field[50], 3, oline->ol_stock);
                AlphaField(&field[57], 1, oline-
>ol_brand_generic);
                field[70] = field[61] = '$';
                DecField(&field[62], 6, oline->ol_i_price);
                DecField(&field[71], 7, oline->ol_amount);
                return TRUE;
        }
        return FALSE;
}

BOOL OrderStatusLine(char *field, OL_ORDER_STATUS_DATA *oline)
{
        if(!oline->ol_i_id) {
                AlphaField(field, 57, " ");
                return FALSE;
        } else {
                IntField(&field[2], 4, oline-
>ol_supply_w_id);
                IntField(&field[13], 6, oline->ol_i_id);
                IntField(&field[24], 2, oline->ol_quantity);
                field[31] = '$';
                DecField(&field[32], 8, oline->ol_amount);
                DateField(&field[46], &oline-
>ol_delivery_d);
                return TRUE;
        }
        return FALSE;
}

int FormatHtmlPage(char *page, char *dest) {
        enum stag {COPY, SCAN} state = COPY;
        int sx = 0, dx = 0;
        while(page[sx]) {
                switch(page[sx]) {
                        case '<':
                                switch(state) {
                                        case COPY:
                                                if(page[sx+1]
== 'P' && page[sx+2] == 'R' && page[sx+3] == 'E' && page[sx+4] == '>') {

                                                while(page[sx++] != '>') dest[dx++] = page[sx-1];

                                                dest[dx++] = page[sx-1];
                                                                state =
SCAN;
                                                } else {
                                                dest[dx++] = page[sx++];
                                                }
                                                break;
                                        case SCAN:
                                                if(page[sx+1]
== '/' && page[sx+2] == 'P' && page[sx+3] == 'R' && page[sx+4] == 'E' &&
page[sx+5] == '>') {

                                                while(page[sx++] != '>') dest[dx++] = page[sx-1];

                                                dest[dx++] = page[sx-1];
                                                                state =
COPY;
                                                } else
if(page[sx+1] == 'I' && page[sx+2] == 'N' && page[sx+3] == 'P' &&
page[sx+4] == 'U' && page[sx+5] == 'T') {

                                                while(page[sx++] != '>') dest[dx++] = page[sx-1];

                                                dest[dx++] = page[sx-1];
                                                } else {

                                                dest[dx++] = '&';

                                                dest[dx++] = 'l';

                                                dest[dx++] = 't';

                                                dest[dx++] = ';';
                                                sx++;
                                                }
                                                break;
                                }
                                break;
                        case '>':
                                switch(state) {
                                        case COPY:
                                                dest[dx++] =
page[sx++];
                                                break;
                                        case SCAN:
                                                dest[dx++] =
'&';
                                                dest[dx++] =
'g';
                                                dest[dx++] = 't';
                                                dest[dx++] = ';';
                                                sx++;
                                                break;
                                }
                                break;
                        case '&':
                                dest[dx++] = '&';
                                dest[dx++] = 'a';
                                dest[dx++] = 'm';
                                dest[dx++] = 'p';
                                dest[dx++] = ';';
                                sx++;
                                break;
                        case '\"':
                                switch(state) {
                                        case COPY:
                                                dest[dx++] =
page[sx++];
                                                break;
                                        case SCAN:
                                                dest[dx++] =
'&';
```

```
                                                dest[dx++] =
'q';
                                                dest[dx++] =
'u';
                                                dest[dx++] =
'o';
                                                dest[dx++] = 't';
                                                dest[dx++] = ';';
                                                sx++;
                                                break;
                                        }
                                        break;
                                default:
                                        dest[dx++] = page[sx++];
                                        break;
                        }
                }
                dest[dx] = '\0';
                return dx;
}
```

OUTPUT.H

```
/* Audited: 28 February 1997 */

/* output.h
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#ifndef __output_h__
#define __output_h__

#include <tpcc/kit/src/tpcc.h>

void AlphaField(char *, int, char *);
void IntField(char *, int , int);
void DecField(char *, int , double);
void SignedDecField(char *, int, double);
void DateField(char *, DBDATEREC *);
void DateTimeField(char *, DBDATEREC *);
void ZipField(char *, char *);
void PhoneField(char *, char *);
BOOL NewOrderLine(char *, OL_NEW_ORDER_DATA *);
BOOL OrderStatusLine(char *, OL_ORDER_STATUS_DATA *);
int FormatHtmlPage(char *, char *);

#endif __output_h__
```

PAYMENT.C

```
/* Audited: 28 February 1997 */

/* payment.c
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#include "payment.h"

int payment_func_parse(assoc *a, int *cookie, PAYMENT_DATA *data, char
*output) {
                int i = 0;
                char errstr[128];
                char all_errors[1024];
                int cid = 0;
                errstr[0] = '\0';
                all_errors[0] = '\0';
                data->c_id = 0;
                data->c_last[0] = '\0';
                while((*a)[0][i]) {
                        switch((*a)[0][i][0]) {
                                case 'c':
                                        *cookie = VerifyInt((*a)[1][i],
4);
                                        break;
                                case 'D':
                                        data->d_id =
VerifyShort((*a)[1][i], 2);
                                        break;
                                case 'C':
                                        switch((*a)[0][i][1]) {
                                                case 'I':
                if(strlen((*a)[1][i])) cid++;
                                                        data->c_id =
VerifyLong((*a)[1][i], 4);
                                                        break;
                                                case 'W':
                                                        data->c_w_id =
VerifyShort((*a)[1][i], 4);
                                                        break;
                                                case 'D':
                                                        data->c_d_id =
VerifyShort((*a)[1][i], 2);
                                                        break;
                                                case 'L':
                if(strlen((*a)[1][i])) cid++;

                if(VerifyString((*a)[1][i], 16))

                strcpy(data->c_last, (*a)[1][i]);
                                                        break;
                                                default: break;
                                        }
                                        break;
                                case 'H':
                                        data->h_amount =
VerifyDouble((*a)[1][i], 7);
                                        break;
                                default: break;
                        }
                        ++i;
                }
                if(*cookie < 0 || !get_user(*cookie)->w_id) {
                        sprintf(errstr, BAD_COOKIE_MSG);
                        strcat(all_errors, errstr);
                }
                if(cid != 1)
                        strcat(all_errors, "o You must fill in one
(and only one) of \"Customer ID\" and \"Customer Last Name\"\r\n");
                else if(!data->c_id && !data->c_last[0])
                        strcat(all_errors, "o The \"Customer Last
Name\" field is too long.  The maximum is 16.\r\n");
                switch(data->d_id) {
                        case -1:
                                sprintf(errstr, TOO_LONG_MSG,
"District ID", 2);
                                strcat(all_errors, errstr);
                                break;
                        case -2:
                                sprintf(errstr, NOT_ISDIGIT_MSG,
"District ID");
                                strcat(all_errors, errstr);
                                break;
                        case -3:
                                sprintf(errstr, NO_INPUT_MSG,
"District ID");
                                strcat(all_errors, errstr);
                                break;
                        default: break;
                }
                switch(data->c_id) {
                        case -1:
                                sprintf(errstr, TOO_LONG_MSG,
"Customer ID", 4);
                                strcat(all_errors, errstr);
                                break;
                        case -2:
                                sprintf(errstr, NOT_ISDIGIT_MSG,
"Customer ID");
                                strcat(all_errors, errstr);
                                break;
                        default:break;
                }
                switch(data->c_w_id) {
                        case -1:
                                sprintf(errstr, TOO_LONG_MSG,
"Customer Warehouse ID", 4);
                                strcat(all_errors, errstr);
                                break;
                        case -2:
                                sprintf(errstr, NOT_ISDIGIT_MSG,
"Customer Warehouse ID");
                                strcat(all_errors, errstr);
                                break;
                        case -3:
                                sprintf(errstr, NO_INPUT_MSG,
"Customer Warehouse ID");
                                strcat(all_errors, errstr);
                                break;
                        default:break;
                }
                switch(data->c_d_id) {
                        case -1:
                                sprintf(errstr, TOO_LONG_MSG,
"Customer District ID", 2);
                                strcat(all_errors, errstr);
                                break;
                        case -2:
                                sprintf(errstr, NOT_ISDIGIT_MSG,
"Customer District ID");
                                strcat(all_errors, errstr);
                                break;
                        case -3:
                                sprintf(errstr, NO_INPUT_MSG,
"Customer District ID");
                                strcat(all_errors, errstr);
                                break;
                        default:break;
                }
                if(data->h_amount == HUGE_VAL)
                        strcat(all_errors, "o The \"Amount Paid\"
field is invalid.\r\n  It should be a decimal number of at most two places,\r\n
without a dollar sign.\r\n  The field cannot contain more than 7
characters.\r\n");
                if(data->d_id >= 0 && (data->d_id < 1 || data-
>d_id > 10))
```

```
                        strcat(all_errors, "o The \"District ID\" field
must be in the range 1-10.\r\n");
                        if(data->c_w_id >= 0 && (data->c_w_id < 1 ||
data->c_w_id > MAXWH)) {
                        sprintf(errstr, "o The \"Customer
Warehouse ID\" field must be in the range 1-%d.\r\n", MAXWH);
                        strcat(all_errors, errstr);
                        }
                        if(data->c_d_id >= 0 && (data->c_d_id < 1 ||
data->c_d_id > 10))
                        strcat(all_errors, "o The \"Customer
District ID\" field must be in the range 1-10.\r\n");
                        data->w_id = get_user(*cookie)->w_id;
                        if(all_errors[0]) {
                        sprintf(output, errorpage, all_errors);
                        return 0;
                        } else return 1;
}

int payment_func_process(PAYMENT_DATA *data, int cookie) {
#ifdef DB_PRESENT
                        return SQLPayment(get_user(cookie)-
>dbhandle, data, DEADLOCK_RETRY);
#else
                        data->c_since.year = 1973;
                        data->c_since.month = 1;
                        data->c_since.day = 9;
                        data->h_date.year = 1996;
                        data->h_date.month = 4;
                        data->h_date.day = 11;
                        data->h_date.hour = 18;
                        data->h_date.minute = 42;
                        data->h_date.second = 9;
                        strcpy(data->w_street_1, "1313 Mockingbird
Ln");
                        strcpy(data->w_street_2, "Suite 666");
                        strcpy(data->w_city, "Huntsville");
                        strcpy(data->w_state, "AL");
                        strcpy(data->w_zip, "358051234");
                        strcpy(data->d_street_1, "1225 Fubar Drive");
                        strcpy(data->d_street_2, "Blicky-Blecky");
                        strcpy(data->d_city, "Hornswaggle");
                        strcpy(data->d_state, "AL");
                        strcpy(data->d_zip, "356259876");
                        strcpy(data->c_first, "Frederick");
                        strcpy(data->c_middle, "J.");
                        if(!data->c_last[0]) strcpy(data->c_last,
"Johnson");
                        else data->c_id = 123;
                        strcpy(data->c_street_1, "6500 Fnord Street");
                        strcpy(data->c_street_2, "Apartment 1492");
                        strcpy(data->c_city, "Fizzywog");
                        strcpy(data->c_state, "TN");
                        strcpy(data->c_zip, "343875678");
                        strcpy(data->c_phone, "ABCDEF3341234567");
                        strcpy(data->c_credit, "D7");
                        data->c_credit_lim = 1234567890.40;
                        data->c_discount = 0.1234;
                        data->c_balance = -12345.76;
                        strcpy(data->c_data, "This customer is a freak.
He frequently carries  automatic weapons, and should be watched closely
at all times.  Hide the silverware.");
                        return 1;
#endif DB_PRESENT
}

void payment_func_format(char *output, PAYMENT_DATA *data, int cookie)
{
                        char buf[3000];
                        sprintf(buf, presp, cookie);
                        data->c_discount *= 100;
  DateTimeField(&buf[PDT], &data->h_date);
                        IntField(&buf[PW], 4, data->w_id);
                        IntField(&buf[PD], 2, data->d_id);
                        AlphaField(&buf[PWA1], 20, data->w_street_1);
                        AlphaField(&buf[PDA1], 20, data->d_street_1);
                        AlphaField(&buf[PWA2], 20, data->w_street_2);
                        AlphaField(&buf[PDA2], 20, data->d_street_2);
                        AlphaField(&buf[PWCT], 20, data->w_city);
                        AlphaField(&buf[PWST], 2, data->w_state);
                        ZipField(&buf[PWZP], data->w_zip);
                        AlphaField(&buf[PDCT], 20, data->d_city);
                        AlphaField(&buf[PDST], 2, data->d_state);
                        ZipField(&buf[PDZP], data->d_zip);
                        IntField(&buf[PC], 4, data->c_id);
                        IntField(&buf[PCW], 4, data->c_w_id);
                        IntField(&buf[PCD], 2, data->c_d_id);
                        AlphaField(&buf[PCF], 16, data->c_first);
                        AlphaField(&buf[PCM], 2, data->c_middle);
                        AlphaField(&buf[PCL], 16, data->c_last);
                        DateField(&buf[PSINCE], &data->c_since);
                        AlphaField(&buf[PCA1], 20, data->c_street_1);
                        AlphaField(&buf[PCRED], 2, data->c_credit);
                        AlphaField(&buf[PCA2], 20, data->c_street_2);
                        DecField(&buf[PDSC], 5, data->c_discount);
                        AlphaField(&buf[PCCT], 20, data->c_city);
                        AlphaField(&buf[PCST], 2, data->c_state);
                        ZipField(&buf[PCZP], data->c_zip);
                        PhoneField(&buf[PPHN], data->c_phone);
                        DecField(&buf[PAMT], 7, data->h_amount);
                        SignedDecField(&buf[PBAL], 14, data-
>c_balance);
                        DecField(&buf[PCLIM], 13, data->c_credit_lim);
                        AlphaField(&buf[PCDAT1], 50, data->c_data);
                        if(strlen(data->c_data) > 50)
                        AlphaField(&buf[PCDAT2], 50, &(data-
>c_data[50]));
                        else AlphaField(&buf[PCDAT2], 50, " ");
                        if(strlen(data->c_data) > 100)
                        AlphaField(&buf[PCDAT3], 50, &(data-
>c_data[100]));
                        else AlphaField(&buf[PCDAT3], 50, " ");
                        if(strlen(data->c_data) > 150)
                        AlphaField(&buf[PCDAT4], 50, &(data-
>c_data[150]));
                        else AlphaField(&buf[PCDAT4], 50, " ");
                        FormatHtmlPage(buf, output);
}

void payment_func_main(assoc *a, char *output) {
                        int cookie;
                        PAYMENT_DATA data;
                        if(!payment_func_parse(a, &cookie, &data,
output)) return;
                        if(!payment_func_process(&data, cookie)) {
                        sprintf(output, dberrpage, cookie);
                        return;
                        }
                        payment_func_format(output, &data, cookie);
}
```

PAYMENT.H

```
/* Audited: 28 February 1997 */

/* payment.h
  Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#ifndef __payment_h__
#define __payment_h__

#include "context.h"
#include <tpcc/kit/src/tpcc.h>
#include "inputparser.h"
#include "output.h"
#include "errors.h"
#include "options.h"

#define PAYMENT_FUNC 6

static char presp[] =
"<HTML><HEAD><TITLE>TPC-C:
Payment</TITLE></HEAD><BODY><PRE>"
"                       Payment\r\n"
"Date: XXXXXXXXXXXXXXXXXX\r\n"
"\r\n"
"Warehouse: XXXX                District: XX\r\n"
"XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXX\r\n"
"XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXX\r\n"
"XXXXXXXXXXXXXXXXXXX XX XXXXXXXXXX
XXXXXXXXXXXXXXXXXXX XX XXXXXXXXX\r\n"
"\r\n"
"Customer: XXXX  Cust-Warehouse: XXXX  Cust-District: XX\r\n"
"Name:   XXXXXXXXXXXXXXX XX XXXXXXXXXXXXXXX     Since:
XXXXXXXXX\r\n"
"       XXXXXXXXXXXXXXXXXXX                Credit: XX\r\n"
"       XXXXXXXXXXXXXXXXXXX                %%Disc:  XXXX\r\n"
"       XXXXXXXXXXXXXXXXXXX XX XXXXXXXXX       Phone:
XXXXXXXXXXXXXXXXX\r\n"
"\r\n"
"Amount Paid:         $XXXXXXX      New Cust-Balance:
$XXXXXXXXXXXXX\r\n"
"Credit Limit:   $XXXXXXXXXXXX\r\n"
"\r\n"
"Cust-Data:
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\r\
n"
"
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\r\
n"
"
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\r\
n"
"
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\r\
n"
"\r\n"
"</PRE><P><FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">"
"<INPUT TYPE=\"hidden\" NAME=\"c\" VALUE=\"%d\">"
"<INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"New Order\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Payment\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Delivery\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Order-Status\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Stock-Level\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Exit\">"
"</FORM></P></BODY></HTML>\r\n";

/* Character indices of field locations */
#define PDT 111
```

```
#define PW 145
#define PD 185
#define PWA1 189
#define PDA1 230
#define PWA2 252
#define PDA2 293
#define PWCT 315
#define PWST 336
#define PWZP 339
#define PDCT 356
#define PDST 377
#define PDZP 380
#define PC 404
#define PCW 426
#define PCD 447
#define PCF 459
#define PCM 476
#define PCL 479
#define PSINCE 508
#define PCA1 528
#define PCRED 577
#define PCA2 589
#define PDSC 638
#define PCCT 653
#define PCST 674
#define PCZP 677
#define PPHN 702
#define PAMT 748
#define PBAL 780
#define PCLIM 813
#define PCDAT1 841
#define PCDAT2 904
#define PCDAT3 967
#define PCDAT4 1030

extern void e_log(char *);
void payment_func_main(assoc *, char *);
int payment_func_parse(assoc *, int *, PAYMENT_DATA *, char *);
int payment_func_process(PAYMENT_DATA *, int);
void payment_func_format(char *, PAYMENT_DATA *, int);

#endif __payment_h__
```

PROCESSLOGIN.C

```
/* Audited: 28 February 1997 */

/* processlogin.c
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#include "processlogin.h"

int processlogin_parse(assoc *a, short *w_id, short *d_id) {
                        int i = 0;
                        while((*a)[0][i]) {
                                switch((*a)[0][i][0]) {
                                        case 'W':
                                                *w_id = VerifyShort((*a)[1][i],
3);
                                                break;
                                        case 'd':
                                                *d_id = VerifyShort((*a)[1][i],
2);
                                                break;
                                        default: break;
                                }
                                ++i;
                        }
                        if(*w_id < 1 || *d_id < 1 || *d_id > 10 || *w_id >
MAXWH)
                                return 0;
                        else
                                return 1;
}


void processlogin_func_main(assoc *a, char *output) {
                        short w_id, d_id;
                        int cookie;
                        if(!processlogin_parse(a, &w_id, &d_id))
                                sprintf(output, logerrpage, MAXWH);
                        else if((cookie = create_user(w_id, d_id)) < 0)
                                sprintf(output, enosvcdbo);
                        else if(cookie >= MAX_USERS + TokenIndex)
                                sprintf(output, noconnpage, MAX_USERS
+ TokenIndex);
                        else
                                sprintf(output, menupage, cookie);
}
```

PROCESSLOGIN.H

```
/* Audited: 28 February 1997 */

/* processlogin.h
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#ifndef __processlogin_h__
#define __processlogin_h__

#include "context.h"
#include "inputparser.h"

extern void e_log(char *);

#define PROCESSLOGIN_FUNC 2

static char logerrpage[] =
"<HTML><HEAD><TITLE>Welcome to TPC-C</TITLE></HEAD><BODY>"
"<P>The Warehouse and/or District ID that you entered is either absent or "
"invalid in some way.  You must provide data for both fields.  The
Warehouse "
"ID an integer in the range 1 to %d.  The District ID must be an integer "
"in the range 1 to 10.</P> "
"<FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">"
"<INPUT TYPE=\"hidden\" NAME=\"f\" VALUE=\"L\">"
"Your Warehouse ID: <INPUT NAME=\"W\" SIZE=4><BR>"
"Your District ID:  <INPUT NAME=\"d\" SIZE=2><BR><HR>"
"<INPUT TYPE=\"submit\">"
"</FORM></BODY></HTML>\r\n";

static char menupage[] =
"<HTML><HEAD><TITLE>TPC-C: Main Menu</TITLE></HEAD><BODY>"
"<P>Please select an action from the menu of buttons below.</P><HR>"
"<FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">"
"<INPUT TYPE=\"hidden\" NAME=\"c\" VALUE=\"%d\">"
"<INPUT TYPE=\"hidden\" NAME=\"f\" VALUE=\"M\">"
"<INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"New Order\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Payment\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Delivery\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Order-Status\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Stock-Level\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Exit\">"
"</FORM></P></BODY></HTML>\r\n";

static char noconnpage[] =
"<HTML><HEAD><TITLE>TPC-C: Can't
Connect</TITLE></HEAD><BODY>"
"<P>Sorry, all %d database connections are currently in use."
"  Please try again later.</P>"
"</BODY></html>\r\n";

static char enosvcdbo[] =
"<HTML><HEAD><TITLE>TPC-C: Service
Unavailable</TITLE></HEAD><BODY>"
"<P>The TPC-C Application Program (TPCC.DLL) failed to establish a
connection to the database"
" for this session.  As a result, no transactions can be processed.  Please try
again later."
"  If the problem persists, email <a
href=\"mailto:rothomas@ingr.com\">Robert Thomas</a> and"
" report seeing this message.</BODY></html>";

void processlogin_func_main(assoc *, char *);
int processlogin_parse(assoc *, short *, short *);

#endif __processlogin_h__
```

QUERY_FORM.C

```
/* Audited: 28 February 1997 */

/* query_form.c
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#include "query_form.h"

void query_form_func_main(assoc *a, char *output) {
                        int i = 0, cookie = -1;
                        char *form = 0;
                        char wid[5];
                        while((*a)[0][i]) {
                                switch((*a)[0][i][0]) {
                                        case 'b':
                                                switch((*a)[1][i][0]) {
                                                        case 'N': form =
noform; break;
                                                        case 'P': form = pform;
break;
                                                        case 'O': form =
oform; break;
                                                        case 'D': form =
dform; break;
                                                        case 'S': form = sform;
break;
                                                        default: sprintf(output,
"Invalid Function Called"); return;
                                                }
                                                break;
                                        case 'c':
                                                cookie = VerifyInt((*a)[1][i],
4);
                                                break;
                                        default: break;
                                }
                                ++i;
                        }
```

```
                    if(cookie < TokenIndex || cookie > MAX_USERS
+ TokenIndex) {
                         sprintf(output, "Invalid cookie value.");
                         return;
                    }
                    if(!get_user(cookie)->w_id) {
                         sprintf(output, "Dead cookie value
recieved.");
                         return;
                    }
                    IntField(wid, 4, get_user(cookie)->w_id);
                    wid[4] = '\0';
                    sprintf(output, form, cookie, wid);
}
```

QUERY_FORM.H

```
/* Audited: 28 February 1997 */

/* query_form.h
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#ifndef __query_form_h__
#define __query_form_h__

#include "context.h"
#include "inputparser.h"
#include "output.h"

#define QUERY_FORM_FUNC 3

extern void e_log(char *);

static char dform[] =
"<HTML><HEAD><TITLE>TPC-C: Delivery</TITLE></HEAD><BODY>"
"<FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">"
"<INPUT TYPE=\"hidden\" NAME=\"f\" VALUE=\"D\">"
"<INPUT TYPE=\"hidden\" NAME=\"c\" VALUE=\"%d\"><PRE>"
"                              Delivery\r\n"
"Warehouse: %4s\r\n"
"\r\n"
"Carrier Number: <INPUT NAME=\"OC\" SIZE=2>\r\n"
"\r\n"
"Execution Status:"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n</PRE>"
"<HR><INPUT TYPE=\"submit\"></FORM></BODY></HTML>\r\n";

static char sform[] =
"<HTML><HEAD><TITLE>TPC-C: Stock-Level</TITLE></HEAD><BODY>"
"<FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">"
"<INPUT TYPE=\"hidden\" NAME=\"f\" VALUE=\"S\">"
"<INPUT TYPE=\"hidden\" NAME=\"c\" VALUE=\"%d\"><PRE>"
"                              Stock-Level\r\n"
"Warehouse: %4s  District:\r\n"
"\r\n"
"Stock Level Threshold: <INPUT NAME=\"t\" SIZE=2>\r\n"
"\r\n"
"low stock:"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n</PRE>"
"<HR><INPUT TYPE=\"submit\"></FORM></BODY></HTML>\r\n";

static char pform[] =
"<HTML><HEAD><TITLE>TPC-C: Payment</TITLE></HEAD><BODY>"
"<FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">"
"<INPUT TYPE=\"hidden\" NAME=\"f\" VALUE=\"P\">"
"<INPUT TYPE=\"hidden\" NAME=\"c\" VALUE=\"%d\"><PRE>"
"                              Payment\r\n"
"Date:\r\n"
"\r\n"
"Warehouse: %4s                  District: <INPUT NAME=\"D\"
SIZE=2>\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"Customer: <INPUT NAME=\"CI\" SIZE=4> Cust-Warehouse: <INPUT
NAME=\"CW\" SIZE=4> Cust-District: <INPUT NAME=\"CD\" SIZE=2>\r\n"
"Name:                 <INPUT NAME=\"CL\" SIZE=16>    Since:\r\n"
"                                          Credit:\r\n"
"                                          %%Disc:\r\n"
"                                          Phone:\r\n"
"\r\n"
"Amount Paid:         $<INPUT NAME=\"H\" SIZE=7>    New Cust-
Balance:\r\n"
"Credit Limit:\r\n"
"\r\n"
"Cust-Data:\r\n\r\n\r\n\r\n"
"\r\n"
"</PRE><HR><INPUT TYPE=\"submit\"></FORM></BODY></HTML>\r\n";

static char oform[] = "<HTML><HEAD><TITLE>TPC-C: Order-
Status</TITLE></HEAD><BODY>"
"<FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">"
"<INPUT TYPE=\"hidden\" NAME=\"f\" VALUE=\"O\">"
"<INPUT TYPE=\"hidden\" NAME=\"c\" VALUE=\"%d\"><PRE>"
"                              Order-Status\r\n"
"Warehouse: %4s   District: <INPUT NAME=\"D\" SIZE=2>\r\n"
"Customer: <INPUT NAME=\"CI\" SIZE=4>  Name:                 <INPUT
NAME=\"CL\" SIZE=16>\r\n"
"Cust-Balance:\r\n"
"\r\n"
"Order-Number:          Entry-Date:                Carrier-Number:\r\n"
"Supply-W     Item-Id    Qty     Amount     Delivery-Date\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"</PRE><HR><INPUT TYPE=\"submit\"></FORM></BODY></HTML>\r\n";

static char noform[] = "<HTML><HEAD><TITLE>TPC-C: New
Order</TITLE></HEAD><BODY>"
"<FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">"
"<INPUT TYPE=\"hidden\" NAME=\"f\" VALUE=\"N\">"
"<INPUT TYPE=\"hidden\" NAME=\"c\" VALUE=\"%d\"><PRE>"
"                              New Order\r\n"
"Warehouse: %4s   District: <INPUT NAME=\"D\" SIZE=2>
Date:\r\n"
"Customer: <INPUT NAME=\"CI\" SIZE=4>  Name:               Credit:
%%Disc:\r\n"
"Order Number:          Number of Lines:          W_tax:          D_tax:\r\n"
"\r\n"
" Supp_W  Item_Id  Item Name               Qty  Stock  B/G  Price
Amount\n"
"  <INPUT NAME=\"OS01\" SIZE=4>   <INPUT NAME=\"OI01\" SIZE=6>
<INPUT NAME=\"OQ01\" SIZE=2>\r\n"
"  <INPUT NAME=\"OS02\" SIZE=4>   <INPUT NAME=\"OI02\" SIZE=6>
<INPUT NAME=\"OQ02\" SIZE=2>\r\n"
"  <INPUT NAME=\"OS03\" SIZE=4>   <INPUT NAME=\"OI03\" SIZE=6>
<INPUT NAME=\"OQ03\" SIZE=2>\r\n"
"  <INPUT NAME=\"OS04\" SIZE=4>   <INPUT NAME=\"OI04\" SIZE=6>
<INPUT NAME=\"OQ04\" SIZE=2>\r\n"
"  <INPUT NAME=\"OS05\" SIZE=4>   <INPUT NAME=\"OI05\" SIZE=6>
<INPUT NAME=\"OQ05\" SIZE=2>\r\n"
"  <INPUT NAME=\"OS06\" SIZE=4>   <INPUT NAME=\"OI06\" SIZE=6>
<INPUT NAME=\"OQ06\" SIZE=2>\r\n"
"  <INPUT NAME=\"OS07\" SIZE=4>   <INPUT NAME=\"OI07\" SIZE=6>
<INPUT NAME=\"OQ07\" SIZE=2>\r\n"
"  <INPUT NAME=\"OS08\" SIZE=4>   <INPUT NAME=\"OI08\" SIZE=6>
<INPUT NAME=\"OQ08\" SIZE=2>\r\n"
"  <INPUT NAME=\"OS09\" SIZE=4>   <INPUT NAME=\"OI09\" SIZE=6>
<INPUT NAME=\"OQ09\" SIZE=2>\r\n"
"  <INPUT NAME=\"OS10\" SIZE=4>   <INPUT NAME=\"OI10\" SIZE=6>
<INPUT NAME=\"OQ10\" SIZE=2>\r\n"
"  <INPUT NAME=\"OS11\" SIZE=4>   <INPUT NAME=\"OI11\" SIZE=6>
<INPUT NAME=\"OQ11\" SIZE=2>\r\n"
"  <INPUT NAME=\"OS12\" SIZE=4>   <INPUT NAME=\"OI12\" SIZE=6>
<INPUT NAME=\"OQ12\" SIZE=2>\r\n"
"  <INPUT NAME=\"OS13\" SIZE=4>   <INPUT NAME=\"OI13\" SIZE=6>
<INPUT NAME=\"OQ13\" SIZE=2>\r\n"
"  <INPUT NAME=\"OS14\" SIZE=4>   <INPUT NAME=\"OI14\" SIZE=6>
<INPUT NAME=\"OQ14\" SIZE=2>\r\n"
"  <INPUT NAME=\"OS15\" SIZE=4>   <INPUT NAME=\"OI15\" SIZE=6>
<INPUT NAME=\"OQ15\" SIZE=2>\r\n"
"Execution Status:                                      Total:  $"
"</PRE><HR><INPUT TYPE=\"submit\"></FORM></BODY></HTML>\r\n";

void query_form_func_main(assoc *, char *);
void query_form_func_parse(assoc *);

#endif __query_form_h__
```

SQLDB.H

```
#ifndef _INC_SQLDB
#define _INC_SQLDB

#ifdef __cplusplus
                        extern "C" {
#endif

/***********************************************************************
*                                                                      *
*      SQLDB.H - DB-Library header file for the Microsoft SQL Server.      *
*                                                                      *
*    Copyright (c) 1989 - 1995 by Microsoft Corp.  All rights reserved.    *
*                                                                      *
***********************************************************************/

// Macros for setting the PLOGINREC
#define DBSETLHOST(a,b)    dbsetlname   ((a), (b), DBSETHOST)
#define DBSETLUSER(a,b)    dbsetlname   ((a), (b), DBSETUSER)
#define DBSETLPWD(a,b)     dbsetlname   ((a), (b), DBSETPWD)
#define DBSETLAPP(a,b)     dbsetlname   ((a), (b), DBSETAPP)
#define BCP_SETL(a,b)      bcp_setl     ((a), (b))
#define DBSETLNATLANG(a,b) dbsetlname   ((a), (b), DBSETLANG)
#define DBSETLPACKET(a,b)  dbsetlpacket ((a), (b))
#define DBSETLSECURE(a)    dbsetlname   ((a), 0,   DBSETSECURE)
#define DBSETLVERSION(a,b) dbsetlname   ((a), 0,  (b))
#define DBSETLTIME(a,b)          dbsetlname    ((a), (LPCSTR)(ULONG)(b),
DBSETLOGINTIME)

/***********************************************************************
* Windows 3.x and Non-Windows 3.x differences.                         *
***********************************************************************/

#ifdef DBMSWIN

extern void SQLAPI dbwinexit(void);

void SQLAPI dblocklib (void);
void SQLAPI dbunlocklib (void);

#define DBLOCKLIB()   dblocklib()
#define DBUNLOCKLIB() dbunlocklib()

#define DBERRHANDLE_PROC FARPROC
#define DBMSGHANDLE_PROC FARPROC

extern DBERRHANDLE_PROC dberrhandle (DBERRHANDLE_PROC);
extern DBMSGHANDLE_PROC dbmsghandle (DBMSGHANDLE_PROC);

#else

#define dbwinexit()

#define DBLOCKLIB()
#define DBUNLOCKLIB()

typedef INT (SQLAPI *DBERRHANDLE_PROC)(PDBPROCESS, INT, INT,
INT, LPCSTR, LPCSTR);
typedef INT (SQLAPI *DBMSGHANDLE_PROC)(PDBPROCESS, DBINT,
INT, INT, LPCSTR, LPCSTR, LPCSTR, DBUSMALLINT);

extern DBERRHANDLE_PROC SQLAPI
dberrhandle(DBERRHANDLE_PROC);
extern DBMSGHANDLE_PROC SQLAPI
dbmsghandle(DBMSGHANDLE_PROC);
extern DBERRHANDLE_PROC SQLAPI dbprocerrhandle(PDBHANDLE,
DBERRHANDLE_PROC);
extern DBMSGHANDLE_PROC SQLAPI dbprocmsghandle(PDBHANDLE,
DBMSGHANDLE_PROC);


#endif


/***********************************************************************
* Function Prototypes                                                  *
***********************************************************************/

// Functions macros
#define DBCMDROW(a)      dbcmdrow(a)
#define DBCOUNT(a)       dbcount (a)
#define DBCURCMD(a)      dbcurcmd(a)
#define DBCURROW(a)      dbcurrow(a)
#define DBDEAD(a)        dbdead(a)
#define DBFIRSTROW(a)    dbfirstrow(a)
#define DBGETTIME()      dbgettime()
#define DBISAVAIL(a)     dbisavail(a)
#define DBLASTROW(a)     dblastrow(a)
#define DBMORECMDS(a)    dbmorecmds(a)
#define DBNUMORDERS(a)   dbnumorders(a)
#define dbrbuf(a)        ((DBINT)dbdataready(a))
#define DBRBUF(a)        ((DBINT)dbdataready(a))
#define DBROWS(a)        dbrows (a)
#define DBROWTYPE(a)     dbrowtype (a)

// Two-phase commit functions
extern RETCODE     SQLAPI abort_xact (PDBPROCESS, DBINT);
extern void        SQLAPI build_xact_string (LPCSTR, LPCSTR, DBINT,
LPSTR);
extern void        SQLAPI close_commit (PDBPROCESS);
extern RETCODE     SQLAPI commit_xact (PDBPROCESS, DBINT);
extern PDBPROCESS  SQLAPI open_commit (PLOGINREC, LPCSTR);
extern RETCODE     SQLAPI remove_xact (PDBPROCESS, DBINT, INT);
extern RETCODE     SQLAPI scan_xact (PDBPROCESS, DBINT);
extern DBINT       SQLAPI start_xact (PDBPROCESS, LPCSTR, LPCSTR,
INT);
extern INT         SQLAPI stat_xact (PDBPROCESS, DBINT);

// BCP functions
extern DBINT       SQLAPI bcp_batch (PDBPROCESS);
extern RETCODE     SQLAPI bcp_bind (PDBPROCESS, LPCBYTE, INT,
DBINT, LPCBYTE, INT, INT, INT);
extern RETCODE     SQLAPI bcp_colfmt (PDBPROCESS, INT, BYTE, INT,
DBINT, LPCBYTE, INT, INT);
extern RETCODE     SQLAPI bcp_collen (PDBPROCESS, DBINT, INT);
extern RETCODE     SQLAPI bcp_colptr (PDBPROCESS, LPCBYTE, INT);
extern RETCODE     SQLAPI bcp_columns (PDBPROCESS, INT);
extern RETCODE     SQLAPI bcp_control (PDBPROCESS, INT, DBINT);
extern DBINT       SQLAPI bcp_done (PDBPROCESS);
extern RETCODE     SQLAPI bcp_exec (PDBPROCESS, LPDBINT);
extern RETCODE     SQLAPI bcp_init (PDBPROCESS, LPCSTR, LPCSTR,
LPCSTR, INT);
extern RETCODE     SQLAPI bcp_moretext (PDBPROCESS, DBINT,
LPCBYTE);
extern RETCODE     SQLAPI bcp_readfmt (PDBPROCESS, LPCSTR);
extern RETCODE     SQLAPI bcp_sendrow (PDBPROCESS);
extern RETCODE     SQLAPI bcp_setl (PLOGINREC, BOOL);
extern RETCODE     SQLAPI bcp_writefmt (PDBPROCESS, LPCSTR);

// Standard DB-Library functions
extern LPCBYTE     SQLAPI dbadata (PDBPROCESS, INT, INT);
extern DBINT       SQLAPI dbadlen (PDBPROCESS, INT, INT);
extern RETCODE     SQLAPI dbaltbind (PDBPROCESS, INT, INT, INT,
DBINT, LPCBYTE);
extern INT         SQLAPI dbaltcolid (PDBPROCESS, INT, INT);
extern DBINT       SQLAPI dbaltlen (PDBPROCESS, INT, INT);
extern INT         SQLAPI dbaltop (PDBPROCESS, INT, INT);
extern INT         SQLAPI dbalttype (PDBPROCESS, INT, INT);
extern DBINT       SQLAPI dbaltutype (PDBPROCESS, INT, INT);
extern RETCODE     SQLAPI dbanullbind (PDBPROCESS, INT, INT,
LPCDBINT);
extern RETCODE     SQLAPI dbbind (PDBPROCESS, INT, INT, DBINT,
LPBYTE);
extern LPCBYTE     SQLAPI dbbylist (PDBPROCESS, INT, LPINT);
extern RETCODE     SQLAPI dbcancel (PDBPROCESS);
extern RETCODE     SQLAPI dbcanquery (PDBPROCESS);
extern LPCSTR      SQLAPI dbchange (PDBPROCESS);
extern RETCODE     SQLAPI dbclose (PDBPROCESS);
extern void        SQLAPI dbclrbuf (PDBPROCESS, DBINT);
extern RETCODE     SQLAPI dbclropt (PDBPROCESS, INT, LPCSTR);
extern RETCODE     SQLAPI dbcmd (PDBPROCESS, LPCSTR);
extern RETCODE     SQLAPI dbcmdrow (PDBPROCESS);
extern BOOL        SQLAPI dbcolbrowse (PDBPROCESS, INT);
extern RETCODE     SQLAPI dbcolinfo (PDBHANDLE, INT, INT, INT,
LPDBCOL);
extern DBINT       SQLAPI dbcollen (PDBPROCESS, INT);
extern LPCSTR      SQLAPI dbcolname (PDBPROCESS, INT);
extern LPCSTR      SQLAPI dbcolsource (PDBPROCESS, INT);
extern INT         SQLAPI dbcoltype (PDBPROCESS, INT);
extern DBINT       SQLAPI dbcolutype (PDBPROCESS, INT);
extern INT         SQLAPI dbconvert (PDBPROCESS, INT, LPCBYTE,
DBINT, INT, LPBYTE, DBINT);
extern DBINT       SQLAPI dbcount (PDBPROCESS);
extern INT         SQLAPI dbcurcmd (PDBPROCESS);
extern DBINT       SQLAPI dbcurrow (PDBPROCESS);
extern RETCODE     SQLAPI dbcursor (PDBCURSOR, INT, INT, LPCSTR,
LPCSTR);
extern RETCODE     SQLAPI dbcursorbind (PDBCURSOR, INT, INT,
DBINT, LPDBINT, LPBYTE);
extern RETCODE     SQLAPI dbcursorclose (PDBHANDLE);
extern RETCODE     SQLAPI dbcursorcolinfo (PDBCURSOR, INT, LPSTR,
LPINT, LPDBINT, LPINT);
extern RETCODE     SQLAPI dbcursorfetch (PDBCURSOR,  INT, INT);
extern RETCODE     SQLAPI dbcursorfetchex (PDBCURSOR, INT, DBINT,
DBINT, DBINT);
extern RETCODE     SQLAPI dbcursorinfo (PDBCURSOR, LPINT,
LPDBINT);
extern RETCODE     SQLAPI dbcursorinfoex (PDBCURSOR,
LPDBCURSORINFO);
extern PDBCURSOR   SQLAPI dbcursoropen (PDBPROCESS, LPCSTR,
INT, INT,UINT, LPDBINT);
extern LPCBYTE     SQLAPI dbdata (PDBPROCESS, INT);
extern BOOL        SQLAPI dbdataready (PDBPROCESS);
extern RETCODE     SQLAPI dbdatecrack (PDBPROCESS,
LPDBDATEREC, LPCDBDATETIME);
extern DBINT       SQLAPI dbdatlen (PDBPROCESS, INT);
extern BOOL        SQLAPI dbdead (PDBPROCESS);
extern void        SQLAPI dbexit (void);
extern RETCODE     SQLAPI dbfcmd (PDBPROCESS, LPCSTR, ...);
extern DBINT       SQLAPI dbfirstrow (PDBPROCESS);
extern void        SQLAPI dbfreebuf (PDBPROCESS);
extern void        SQLAPI dbfreelogin (PLOGINREC);
extern void        SQLAPI dbfreequal (LPCSTR);
extern LPSTR       SQLAPI dbgetchar (PDBPROCESS, INT);
extern SHORT       SQLAPI dbgetmaxprocs (void);
extern INT         SQLAPI dbgetoff (PDBPROCESS, DBUSMALLINT, INT);
extern UINT        SQLAPI dbgetpacket (PDBPROCESS);
extern STATUS      SQLAPI dbgetrow (PDBPROCESS, DBINT);
extern INT         SQLAPI dbgettime (void);
extern LPVOID      SQLAPI dbgetuserdata (PDBPROCESS);
```

```
extern BOOL        SQLAPI dbhasretstat (PDBPROCESS);
extern LPCSTR      SQLAPI dbinit (void);
extern BOOL        SQLAPI dbisavail (PDBPROCESS);
extern BOOL        SQLAPI dbiscount (PDBPROCESS);
extern BOOL        SQLAPI dbisopt (PDBPROCESS, INT, LPCSTR);
extern DBINT       SQLAPI dblastrow (PDBPROCESS);
extern PLOGINREC    SQLAPI dblogin (void);
extern RETCODE      SQLAPI dbmorecmds (PDBPROCESS);
extern RETCODE      SQLAPI dbmoretext (PDBPROCESS, DBINT,
LPCBYTE);
extern LPCSTR      SQLAPI dbname (PDBPROCESS);
extern STATUS      SQLAPI dbnextrow (PDBPROCESS);
extern RETCODE      SQLAPI dbnullbind (PDBPROCESS, INT, LPCDBINT);
extern INT         SQLAPI dbnumalts (PDBPROCESS, INT);
extern INT         SQLAPI dbnumcols (PDBPROCESS);
extern INT         SQLAPI dbnumcompute (PDBPROCESS);
extern INT         SQLAPI dbnumorders (PDBPROCESS);
extern INT         SQLAPI dbnumrets (PDBPROCESS);
extern PDBPROCESS   SQLAPI dbopen (PLOGINREC, LPCSTR);
extern INT         SQLAPI dbordercol (PDBPROCESS, INT);
extern RETCODE      SQLAPI dbprocinfo (PDBPROCESS,
LPDBPROCINFO);
extern void        SQLAPI dbprhead (PDBPROCESS);
extern RETCODE      SQLAPI dbprrow (PDBPROCESS);
extern LPCSTR      SQLAPI dbprtype (INT);
extern LPCSTR      SQLAPI dbqual (PDBPROCESS, INT, LPCSTR);
extern DBINT       SQLAPI dbreadpage (PDBPROCESS, LPCSTR, DBINT,
LPBYTE);
extern DBINT       SQLAPI dbreadtext (PDBPROCESS, LPVOID, DBINT);
extern RETCODE      SQLAPI dbresults (PDBPROCESS);
extern LPCBYTE     SQLAPI dbretdata (PDBPROCESS, INT);
extern DBINT       SQLAPI dbretlen (PDBPROCESS, INT);
extern LPCSTR      SQLAPI dbretname (PDBPROCESS, INT);
extern DBINT       SQLAPI dbretstatus (PDBPROCESS);
extern INT         SQLAPI dbrettype (PDBPROCESS, INT);
extern RETCODE      SQLAPI dbrows (PDBPROCESS);
extern STATUS      SQLAPI dbrowtype (PDBPROCESS);
extern RETCODE      SQLAPI dbrpcinit (PDBPROCESS, LPCSTR,
DBSMALLINT);
extern RETCODE      SQLAPI dbrpcparam (PDBPROCESS, LPCSTR,
BYTE, INT, DBINT, DBINT, LPCBYTE);
extern RETCODE      SQLAPI dbrpcsend (PDBPROCESS);
extern RETCODE      SQLAPI dbrpcexec (PDBPROCESS);
extern void        SQLAPI dbrpwclr (PLOGINREC);
extern RETCODE      SQLAPI dbrpwset (PLOGINREC, LPCSTR, LPCSTR,
INT);
extern INT         SQLAPI dbserverenum (USHORT, LPSTR, USHORT,
LPUSHORT);
extern void        SQLAPI dbsetavail (PDBPROCESS);
extern RETCODE      SQLAPI dbsetmaxprocs (SHORT);
extern RETCODE      SQLAPI dbsetlname (PLOGINREC, LPCSTR, INT);
extern RETCODE      SQLAPI dbsetlogintime (INT);
extern RETCODE      SQLAPI dbsetlpacket (PLOGINREC, USHORT);
extern RETCODE      SQLAPI dbsetnull (PDBPROCESS, INT, INT,
LPCBYTE);
extern RETCODE      SQLAPI dbsetopt (PDBPROCESS, INT, LPCSTR);
extern RETCODE      SQLAPI dbsettime (INT);
extern void        SQLAPI dbsetuserdata (PDBPROCESS, LPVOID);
extern RETCODE      SQLAPI dbsqlexec (PDBPROCESS);
extern RETCODE      SQLAPI dbsqlok (PDBPROCESS);
extern RETCODE      SQLAPI dbsqlsend (PDBPROCESS);
extern RETCODE      SQLAPI dbstrcpy (PDBPROCESS, INT, INT, LPSTR);
extern INT         SQLAPI dbstrlen (PDBPROCESS);
extern BOOL        SQLAPI dbtabbrowse (PDBPROCESS, INT);
extern INT         SQLAPI dbtabcount (PDBPROCESS);
extern LPCSTR      SQLAPI dbtabname (PDBPROCESS, INT);
extern LPCSTR      SQLAPI dbtabsource (PDBPROCESS, INT, LPINT);
extern INT         SQLAPI dbtsnewlen (PDBPROCESS);
extern LPCDBBINARY  SQLAPI dbtsnewval (PDBPROCESS);
extern RETCODE      SQLAPI dbtsput (PDBPROCESS, LPCDBBINARY,
INT, INT, LPCSTR);
extern LPCDBBINARY  SQLAPI dbtxptr (PDBPROCESS, INT);
extern LPCDBBINARY  SQLAPI dbtxtimestamp (PDBPROCESS, INT);
extern LPCDBBINARY  SQLAPI dbtxtsnewval (PDBPROCESS);
extern RETCODE      SQLAPI dbtxtsput (PDBPROCESS, LPCDBBINARY,
INT);
extern RETCODE      SQLAPI dbuse (PDBPROCESS, LPCSTR);
extern BOOL        SQLAPI dbvarylen (PDBPROCESS, INT);
extern BOOL        SQLAPI dbwillconvert (INT, INT);
extern RETCODE      SQLAPI dbwritepage (PDBPROCESS, LPCSTR,
DBINT, DBINT, LPBYTE);
extern RETCODE      SQLAPI dbwritetext (PDBPROCESS, LPCSTR,
LPCDBBINARY, DBTINYINT, LPCDBBINARY, BOOL, DBINT, LPCBYTE);
extern RETCODE      SQLAPI dbupdatetext(PDBPROCESS, LPCSTR,
LPCDBBINARY, LPCDBBINARY, INT, DBINT, DBINT, LPCSTR, DBINT,
LPCDBBINARY);

#ifdef __cplusplus
}
#endif

#endif // _INC_SQLDB
```

SQLFRONT.H

```
#ifndef _INC_SQLFRONT
#define _INC_SQLFRONT

#ifdef DBNTWIN32
                        #ifndef _WINDOWS_
                                #pragma message (__FILE__ " : db-library
error: windows.h must be included before sqlfront.h.")
                        #endif
#endif

#ifdef __cplusplus
                        extern "C" {
#endif

/****************************************************************************
*                                                                           *
*    SQLFRONT.H - DB-Library header file for the Microsoft SQL Server.
*
*                                                                           *
*    Copyright (c) 1989 - 1995 by Microsoft Corp.  All rights reserved.     *
*                                                                           *
* All constant and macro definitions for DB-Library applications
programming *
* are contained in this file.  This file must be included before SQLDB.H and *
* one of the following #defines must be made, depending on the operating
*
* system: DBMSDOS, DBMSWIN or DBNTWIN32.                                    *
*                                                                           *
****************************************************************************/


/****************************************************************************
* Datatype definitions                                                      *
****************************************************************************/

// Note this has changed because Windows 3.1 defines API as 'pascal far'

#if !defined(M_I86SM) && !defined(DBNTWIN32)
#define SQLAPI cdecl far
#else
#define SQLAPI _cdecl
#endif

#ifndef  API
#define  API  SQLAPI
#endif

#ifndef DOUBLE
typedef double DOUBLE;
#endif


/****************************************************************************
* DBPROCESS, LOGINREC and DBCURSOR                                          *
****************************************************************************/

#define DBPROCESS void   // dbprocess structure type
#define LOGINREC  void   // login record type
#define DBCURSOR  void   // cursor record type
#define DBHANDLE  void   // generic handle

// DOS Specific
#ifdef DBMSDOS
typedef DBPROCESS * PDBPROCESS;
typedef LOGINREC  * PLOGINREC;
typedef DBCURSOR  * PDBCURSOR;
typedef DBHANDLE  * PDBHANDLE;
#define PTR *
#endif


// WIN 3.x Specific.  The handle pointers are near for Windows 3.x
#ifdef DBMSWIN
typedef DBPROCESS near * PDBPROCESS;
typedef LOGINREC  near * PLOGINREC;
typedef DBCURSOR  near * PDBCURSOR;
typedef DBHANDLE  near * PDBHANDLE;
#define PTR far *
#endif


// Windows NT Specific
#ifdef DBNTWIN32
typedef DBPROCESS * PDBPROCESS;
typedef LOGINREC  * PLOGINREC;
typedef DBCURSOR  * PDBCURSOR;
typedef DBHANDLE  * PDBHANDLE;
#define PTR *
typedef int (SQLAPI *SQLFARPROC)();
#else
typedef long (far pascal *LGFARPROC)();  // Windows loadable driver fp
#endif


/****************************************************************************
* Win32 compatibility datatype definitions                                  *
* Note: The following datatypes are provided for Win32 compatibility.       *
* Since some of the datatypes are already defined in unrelated include files
*
* there may definition duplication.  Every attempt has been made to check
*
* for such problems.                                                        *
****************************************************************************/

#ifndef DBNTWIN32

#ifndef SHORT
typedef short SHORT;
```

```
#endif

#ifndef INT
typedef int INT;
#endif

#ifndef UINT
typedef unsigned int UINT;
#endif

#ifndef USHORT
typedef unsigned short USHORT;
#endif

#ifndef ULONG
typedef unsigned long ULONG;
#endif

#ifndef CHAR
typedef char CHAR;
#endif

#ifndef LPINT
typedef INT PTR LPINT;
#endif

typedef unsigned char BYTE;

typedef      CHAR PTR LPSTR;
typedef      BYTE PTR LPBYTE;
typedef      void PTR LPVOID;
typedef const CHAR PTR LPCSTR;

typedef int BOOL;

#endif


/*****************************************************************************
* DB-Library datatype definitions                                            *
*****************************************************************************/

#define DBMAXCHAR 256 // Max length of DBVARBINARY and
DBVARCHAR, etc.

#ifndef DBTYPEDEFS    // srv.h (Open Server include) not already included

#define DBTYPEDEFS

#define RETCODE INT
#define STATUS INT

// DB-Library datatypes
typedef char          DBCHAR;
typedef unsigned char   DBBINARY;
typedef unsigned char   DBTINYINT;
typedef short         DBSMALLINT;
typedef unsigned short  DBUSMALLINT;
typedef long          DBINT;
typedef double         DBFLT8;
typedef unsigned char   DBBIT;
typedef unsigned char   DBBOOL;
typedef float         DBFLT4;
typedef long          DBMONEY4;

typedef DBFLT4 DBREAL;
typedef UINT   DBUBOOL;

typedef struct dbdatetime4
{
                        USHORT numdays;       // No of days since Jan-
1-1900
                        USHORT nummins;       // No. of minutes since
midnight
} DBDATETIM4;


typedef struct dbvarychar
{
                        DBSMALLINT  len;
                        DBCHAR      str[DBMAXCHAR];
} DBVARYCHAR;

typedef struct dbvarybin
{
                        DBSMALLINT  len;
                        BYTE        array[DBMAXCHAR];
} DBVARYBIN;

typedef struct dbmoney
{
                        DBINT mnyhigh;
                        ULONG mnylow;
} DBMONEY;

typedef struct dbdatetime
{
                        DBINT dtdays;
                        ULONG dttime;
} DBDATETIME;

// DBDATEREC structure used by dbdatecrack
typedef struct dbdaterec
{
                        INT    year;        // 1753 - 9999
                        INT    quarter;     // 1 - 4
                        INT    month;       // 1 - 12
                        INT    dayofyear;   // 1 - 366
                        INT    day;         // 1 - 31
                        INT    week;        // 1 - 54 (for leap years)
                        INT    weekday;     // 1 - 7  (Mon - Sun)
                        INT    hour;        // 0 - 23
                        INT    minute;      // 0 - 59
                        INT    second;      // 0 - 59
                        INT    millisecond;  // 0 - 999
} DBDATEREC;

#define MAXNUMERICLEN 16
#define MAXNUMERICDIG 38

#define DEFAULTPRECISION 18
#define DEFAULTSCALE     0

typedef struct dbnumeric
{
                        BYTE precision;
                        BYTE scale;
                        BYTE sign; // 1 = Positive, 0 = Negative
                        BYTE val[MAXNUMERICLEN];
} DBNUMERIC;

typedef DBNUMERIC DBDECIMAL;


// Pack the following structures on a word boundary
#ifdef __BORLANDC__
#pragma option -a+
#else
                        #ifndef DBLIB_SKIP_PRAGMA_PACK   // Define
this if your compiler does not support #pragma pack()
                        #pragma pack(2)
                        #endif
#endif

#define MAXCOLNAMELEN 30
#define MAXTABLENAME  30

typedef struct
{
                        DBINT SizeOfStruct;
                        CHAR  Name[MAXCOLNAMELEN+1];
                        CHAR  ActualName[MAXCOLNAMELEN+1];
                        CHAR  TableName[MAXTABLENAME+1];
                        SHORT Type;
                        DBINT UserType;
                        DBINT MaxLength;
                        BYTE  Precision;
                        BYTE  Scale;
                        BOOL  VarLength;     // TRUE, FALSE
                        BYTE  Null;          // TRUE, FALSE or
DBUNKNOWN
                        BYTE  CaseSensitive; // TRUE, FALSE or
DBUNKNOWN
                        BYTE  Updatable;     // TRUE, FALSE or
DBUNKNOWN
                        BOOL  Identity;      // TRUE, FALSE
} DBCOL, PTR LPDBCOL;


#define MAXSERVERNAME 30
#define MAXNETLIBNAME 255
#define MAXNETLIBCONNSTR 255

typedef struct
{
                        DBINT  SizeOfStruct;
                        BYTE   ServerType;
                        USHORT ServerMajor;
                        USHORT ServerMinor;
                        USHORT ServerRevision;
                        CHAR   ServerName[MAXSERVERNAME+1];
                        CHAR   NetLibName[MAXNETLIBNAME+1];
                        CHAR
NetLibConnStr[MAXNETLIBCONNSTR+1];
} DBPROCINFO, PTR LPDBPROCINFO;

typedef struct
{
                        DBINT SizeOfStruct;   // Use
sizeof(DBCURSORINFO)
                        ULONG TotCols;       // Total Columns in cursor
                        ULONG TotRows;       // Total Rows in cursor
                        ULONG CurRow;        // Current actual row in
server
                        ULONG TotRowsFetched; // Total rows actually
fetched
                        ULONG Type;          // See CU_...
                        ULONG Status;        // See CU_...
} DBCURSORINFO, PTR LPDBCURSORINFO;

// Reset default alignment
#ifdef __BORLANDC__
#pragma option -a-
#else
```

```
                        #ifndef DBLIB_SKIP_PRAGMA_PACK   // Define
this if your compiler does not support #pragma pack()
                        #pragma pack()
                        #endif
#endif


#endif // End DBTYPEDEFS


/****************************************************************************
* Pointer Datatypes                                                         *
****************************************************************************/

typedef const LPINT         LPCINT;
typedef const LPBYTE        LPCBYTE ;
typedef      USHORT PTR     LPUSHORT;
typedef const LPUSHORT      LPCUSHORT;
typedef      DBINT PTR      LPDBINT;
typedef const LPDBINT       LPCDBINT;
typedef      DBBINARY PTR   LPDBBINARY;
typedef const LPDBBINARY    LPCDBBINARY;
typedef      DBDATEREC PTR  LPDBDATEREC;
typedef const LPDBDATEREC   LPCDBDATEREC;
typedef      DBDATETIME PTR LPDBDATETIME;
typedef const LPDBDATETIME  LPCDBDATETIME;


/****************************************************************************
* General #defines                                                          *
****************************************************************************/

#define TIMEOUT_IGNORE (ULONG)-1
#define TIMEOUT_INFINITE (ULONG)0
#define TIMEOUT_MAXIMUM (ULONG)1200 // 20 minutes maximum
timeout value

// Used for ServerType in dbgetprocinfo
#define SERVTYPE_UNKNOWN   0
#define SERVTYPE_MICROSOFT 1

// Used by dbcolinfo
enum CI_TYPES { CI_REGULAR=1, CI_ALTERNATE=2, CI_CURSOR=3 };

// Bulk Copy Definitions (bcp)
#define DB_IN               1       // Transfer from client to server
#define DB_OUT              2       // Transfer from server to client

#define BCPMAXERRS  1    // bcp_control parameter
#define BCPFIRST    2    // bcp_control parameter
#define BCPLAST     3    // bcp_control parameter
#define BCPBATCH    4    // bcp_control parameter
#define BCPKEEPNULLS 5   // bcp_control parameter

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

#define TINYBIND        1
#define SMALLBIND       2
#define INTBIND         3
#define CHARBIND        4
#define BINARYBIND      5
#define BITBIND         6
#define DATETIMEBIND    7
#define MONEYBIND       8
#define FLT8BIND        9
#define STRINGBIND      10
#define NTBSTRINGBIND   11
#define VARYCHARBIND    12
#define VARYBINBIND     13
#define FLT4BIND        14
#define SMALLMONEYBIND  15
#define SMALLDATETIBIND 16
#define DECIMALBIND     17
#define NUMERICBIND     18
#define SRCDECIMALBIND  19
#define SRCNUMERICBIND  20
#define MAXBIND         SRCNUMERICBIND

#define DBSAVE          1
#define DBNOSAVE        0

#define DBNOERR         -1
#define DBFINDONE       0x04  // Definately done
#define DBMORE          0x10  // Maybe more commands waiting
#define DBMORE_ROWS     0x20  // This command returned rows

#define MAXNAME         31


#define DBTXTSLEN       8     // Timestamp length

#define DBTXPLEN        16    // Text pointer length

// Error code returns
#define INT_EXIT        0
#define INT_CONTINUE    1
#define INT_CANCEL      2


// dboptions
#define DBBUFFER        0
#define DBOFFSET        1
#define DBROWCOUNT      2
#define DBSTAT          3
#define DBTEXTLIMIT     4
#define DBTEXTSIZE      5
#define DBARITHABORT    6
#define DBARITHIGNORE   7
#define DBNOAUTOFREE    8
#define DBNOCOUNT       9
#define DBNOEXEC        10
#define DBPARSEONLY     11
#define DBSHOWPLAN      12
#define DBSTORPROCID            13

#if defined(DBMSWIN) || defined(DBNTWIN32)
#define DBANSItoOEM             14
#endif

#ifdef DBNTWIN32
#define DBOEMtoANSI             15
#endif

#define DBCLIENTCURSORS 16
#define DBSETTIME 17

// Data Type Tokens
#define SQLTEXT         0x23
#define SQLVARBINARY    0x25
#define SQLINTN         0x26
#define SQLVARCHAR      0x27
#define SQLBINARY       0x2d
#define SQLIMAGE        0x22
#define SQLCHAR         0x2f
#define SQLINT1         0x30
#define SQLBIT          0x32
#define SQLINT2         0x34
#define SQLINT4         0x38
#define SQLMONEY        0x3c
#define SQLDATETIME     0x3d
#define SQLFLT8         0x3e
#define SQLFLTN         0x6d
#define SQLMONEYN       0x6e
#define SQLDATETIMN     0x6f
#define SQLFLT4         0x3b
#define SQLMONEY4       0x7a
#define SQLDATETIM4     0x3a
#define SQLDECIMAL      0x6a
#define SQLNUMERIC      0x6c

// Data stream tokens
#define SQLCOLFMT       0xa1
#define OLD_SQLCOLFMT  0x2a
#define SQLPROCID       0x7c
#define SQLCOLNAME      0xa0
#define SQLTABNAME      0xa4
#define SQLCOLINFO      0xa5
#define SQLALTNAME      0xa7
#define SQLALTFMT       0xa8
#define SQLERROR        0xaa
#define SQLINFO         0xab
#define SQLRETURNVALUE 0xac
#define SQLRETURNSTATUS 0x79
#define SQLRETURN       0xdb
#define SQLCONTROL      0xae
#define SQLALTCONTROL  0xaf
#define SQLROW          0xd1
#define SQLALTROW       0xd3
#define SQLDONE         0xfd
#define SQLDONEPROC     0xfe
#define SQLDONEINPROC  0xff
#define SQLOFFSET       0x78
#define SQLORDER        0xa9
#define SQLLOGINACK     0xad // NOTICE: change to real value

// Ag op tokens
#define SQLAOPCNT               0x4b
#define SQLAOPSUM     0x4d
#define SQLAOPAVG     0x4f
#define SQLAOPMIN     0x51
#define SQLAOPMAX     0x52
#define SQLAOPANY     0x53
#define SQLAOPNOOP    0x56

// Error numbers (dberrs) DB-Library error codes
#define SQLEMEM         10000
#define SQLENULL        10001
#define SQLENLOG        10002
#define SQLEPWD         10003
#define SQLECONN        10004
#define SQLEDDNE        10005
#define SQLENULLO       10006
#define SQLESMSG        10007
#define SQLEBTOK        10008
#define SQLENSPE        10009
#define SQLEREAD        10010
#define SQLECNOR        10011
#define SQLETSIT        10012
```

```
#define SQLEPARM      10013
#define SQLEAUTN      10014
#define SQLECOFL      10015
#define SQLERDCN      10016
#define SQLEICN      10017
#define SQLECLOS      10018
#define SQLENTXT      10019
#define SQLEDNTI      10020
#define SQLETMTD      10021
#define SQLEASEC      10022
#define SQLENTLL      10023
#define SQLETIME      10024
#define SQLEWRIT      10025
#define SQLEMODE      10026
#define SQLEOOB      10027
#define SQLEITIM      10028
#define SQLEDBPS      10029
#define SQLEIOPT      10030
#define SQLEASNL      10031
#define SQLEASUL      10032
#define SQLENPRM      10033
#define SQLEDBOP      10034
#define SQLENSIP      10035
#define SQLECNULL      10036
#define SQLESEOF      10037
#define SQLERPND      10038
#define SQLECSYN      10039
#define SQLENONET      10040
#define SQLEBTYP      10041
#define SQLEABNC      10042
#define SQLEABMT      10043
#define SQLEABNP      10044
#define SQLEBNCR      10045
#define SQLEAAMT      10046
#define SQLENXID      10047
#define SQLEIFNB      10048
#define SQLEKBCO      10049
#define SQLEBBCI      10050
#define SQLEKBCI      10051
#define SQLEBCWE      10052
#define SQLEBCNN      10053
#define SQLEBCOR      10054
#define SQLEBCPI      10055
#define SQLEBCPN      10056
#define SQLEBCPB      10057
#define SQLEVDPT      10058
#define SQLEBIVI      10059
#define SQLEBCBC      10060
#define SQLEBCFO      10061
#define SQLEBCVH      10062
#define SQLEBCUO      10063
#define SQLEBUOE      10064
#define SQLEBWEF      10065
#define SQLEBTMT      10066
#define SQLEBEOF      10067
#define SQLEBCSI      10068
#define SQLEPNUL      10069
#define SQLEBSKERR      10070
#define SQLEBDIO      10071
#define SQLEBCNT      10072
#define SQLEMDBP      10073
#define SQLINIT      10074
#define SQLCRSINV      10075
#define SQLCRSCMD      10076
#define SQLCRSNOIND      10077
#define SQLCRSDIS      10078
#define SQLCRSAGR      10079
#define SQLCRSORD      10080
#define SQLCRSMEM      10081
#define SQLCRSBSKEY      10082
#define SQLCRSNORES      10083
#define SQLCRSVIEW      10084
#define SQLCRSBUFR      10085
#define SQLCRSFROWN      10086
#define SQLCRSBROL      10087
#define SQLCRSFRAND      10088
#define SQLCRSFLAST      10089
#define SQLCRSRO      10090
#define SQLCRSTAB      10091
#define SQLCRSUPDTAB      10092
#define SQLCRSUPDNB      10093
#define SQLCRSVIIND      10094
#define SQLCRSNOUPD      10095
#define SQLCRSOS2      10096
#define SQLEBCSA      10097
#define SQLEBCRO      10098
#define SQLEBCNE      10099
#define SQLEBCSK      10100
#define SQLEUVBF      10101
#define SQLEBIHC      10102
#define SQLEBWFF      10103
#define SQLNUMVAL      10104
#define SQLEOLDVR      10105
#define SQLEBCPS      10106

// The severity levels are defined here
#define EXINFO        1  // Informational, non-error
#define EXUSER        2  // User error
#define EXNONFATAL    3  // Non-fatal error
#define EXCONVERSION  4  // Error in DB-LIBRARY data conversion
#define EXSERVER      5  // The Server has returned an error flag
#define EXTIME        6  // We have exceeded our timeout period while
                         // waiting for a response from the Server - the
                         // DBPROCESS is still alive
#define EXPROGRAM     7  // Coding error in user program
#define EXRESOURCE    8  // Running out of resources - the
DBPROCESS may be dead
#define EXCOMM        9  // Failure in communication with Server - the
DBPROCESS is dead
#define EXFATAL       10 // Fatal error - the DBPROCESS is dead
#define EXCONSISTENCY 11 // Internal software error  - notify MS
Technical Supprt

// Offset identifiers
#define OFF_SELECT    0x16d
#define OFF_FROM      0x14f
#define OFF_ORDER     0x165
#define OFF_COMPUTE   0x139
#define OFF_TABLE     0x173
#define OFF_PROCEDURE 0x16a
#define OFF_STATEMENT 0x1cb
#define OFF_PARAM     0x1c4
#define OFF_EXEC      0x12c

// Print lengths for certain fixed length data types
#define PRINT4    11
#define PRINT2    6
#define PRINT1    3
#define PRFLT8    20
#define PRMONEY   26
#define PRBIT     3
#define PRDATETIME 27
#define PRDECIMAL (MAXNUMERICDIG + 2)
#define PRNUMERIC (MAXNUMERICDIG + 2)

#define SUCCEED  1
#define FAIL     0

#define DBUNKNOWN 2

#define MORE_ROWS    -1
#define NO_MORE_ROWS -2
#define REG_ROW      MORE_ROWS
#define BUF_FULL     -3

// Status code for dbresults(). Possible return values are
// SUCCEED, FAIL, and NO_MORE_RESULTS.
#define NO_MORE_RESULTS 2
#define NO_MORE_RPC_RESULTS 3

// Macros for dbsetlname()
#define DBSETHOST 1
#define DBSETUSER 2
#define DBSETPWD  3
#define DBSETAPP  4
#define DBSETID   5
#define DBSETLANG 6
#define DBSETSECURE 7
#define DBVER42   8
#define DBVER60   9
#define DBSETLOGINTIME 10

// Standard exit and error values
#define STDEXIT  0
#define ERREXIT  -1

// dbrpcinit flags
#define DBRPCRECOMPILE  0x0001
#define DBRPCRESET      0x0004

// dbrpcparam flags
#define DBRPCRETURN     1


// Cursor related constants

// Following flags are used in the concuropt parameter in the dbcursoropen
function
#define CUR_READONLY 1 // Read only cursor, no data modifications
#define CUR_LOCKCC   2 // Intent to update, all fetched data locked when
                       // dbcursorfetch is called inside a transaction block
#define CUR_OPTCC    3 // Optimistic concurrency control, data
modifications
                       // succeed only if the row hasn't been updated since
                       // the last fetch.
#define CUR_OPTCCVAL 4 // Optimistic concurrency control based on
selected column values

// Following flags are used in the scrollopt parameter in dbcursoropen
#define CUR_FORWARD 0       // Forward only scrolling
#define CUR_KEYSET  -1      // Keyset driven scrolling
#define CUR_DYNAMIC 1       // Fully dynamic
#define CUR_INSENSITIVE -2  // Server-side cursors only

// Following flags define the fetchtype in the dbcursorfetch function
#define FETCH_FIRST    1  // Fetch first n rows
#define FETCH_NEXT     2  // Fetch next n rows
#define FETCH_PREV     3  // Fetch previous n rows
#define FETCH_RANDOM   4  // Fetch n rows beginning with given row #
#define FETCH_RELATIVE 5  // Fetch relative to previous fetch row #
#define FETCH_LAST     6  // Fetch the last n rows

// Following flags define the per row status as filled by dbcursorfetch and/or
dbcursorfetchex
```

```
#define FTC_EMPTY        0x00  // No row available
#define FTC_SUCCEED      0x01  // Fetch succeeded, (failed if not set)
#define FTC_MISSING      0x02  // The row is missing
#define FTC_ENDOFKEYSET  0x04  // End of the keyset reached
#define FTC_ENDOFRESULTS 0x08  // End of results set reached

// Following flags define the operator types for the dbcursor function
#define CRS_UPDATE   1  // Update operation
#define CRS_DELETE   2  // Delete operation
#define CRS_INSERT   3  // Insert operation
#define CRS_REFRESH  4  // Refetch given row
#define CRS_LOCKCC   5  // Lock given row

// Following value can be passed to the dbcursorbind function for NOBIND
type
#define NOBIND -2       // Return length and pointer to data

// Following are values used by DBCURSORINFO's Type parameter
#define CU_CLIENT       0x00000001
#define CU_SERVER       0x00000002
#define CU_KEYSET       0x00000004
#define CU_MIXED        0x00000008
#define CU_DYNAMIC      0x00000010
#define CU_FORWARD      0x00000020
#define CU_INSENSITIVE  0x00000040
#define CU_READONLY     0x00000080
#define CU_LOCKCC       0x00000100
#define CU_OPTCC        0x00000200
#define CU_OPTCCVAL     0x00000400

// Following are values used by DBCURSORINFO's Status parameter
#define CU_FILLING      0x00000001
#define CU_FILLED       0x00000002


// Following are values used by dbupdatetext's type parameter
#define UT_TEXTPTR      0x0001
#define UT_TEXT         0x0002
#define UT_MORETEXT     0x0004
#define UT_DELETEONLY   0x0008
#define UT_LOG          0x0010


// The following values are passed to dbserverenum for searching criteria.
#define NET_SEARCH  0x0001
#define LOC_SEARCH  0x0002

// These constants are the possible return values from dbserverenum.
#define ENUM_SUCCESS         0x0000
#define MORE_DATA            0x0001
#define NET_NOT_AVAIL        0x0002
#define OUT_OF_MEMORY        0x0004
#define NOT_SUPPORTED        0x0008
#define ENUM_INVALID_PARAM   0x0010


// Netlib Error problem codes.  ConnectionError() should return one of
// these as the dblib-mapped problem code, so the corresponding string
// is sent to the dblib app's error handler as dberrstr.  Return NE_E_NOMAP
// for a generic DB-Library error string (as in prior versions of dblib).

#define NE_E_NOMAP             0   // No string; uses dblib default.
#define NE_E_NOMEMORY          1   // Insufficient memory.
#define NE_E_NOACCESS          2   // Access denied.
#define NE_E_CONNBUSY          3   // Connection is busy.
#define NE_E_CONNBROKEN        4   // Connection broken.
#define NE_E_TOOMANYCONN       5   // Connection limit exceeded.
#define NE_E_SERVERNOTFOUND    6   // Specified SQL server not
found.
#define NE_E_NETNOTSTARTED     7   // The network has not been
started.
#define NE_E_NORESOURCE        8   // Insufficient network resources.
#define NE_E_NETBUSY           9   // Network is busy.
#define NE_E_NONETACCESS       10  // Network access denied.
#define NE_E_GENERAL           11  // General network error.  Check your
documentation.
#define NE_E_CONNMODE          12  // Incorrect connection mode.
#define NE_E_NAMENOTFOUND      13  // Name not found in directory
service.
#define NE_E_INVALIDCONN       14  // Invalid connection.
#define NE_E_NETDATAERR        15  // Error reading or writing network
data.
#define NE_E_TOOMANYFILES      16  // Too many open file handles.
#define NE_E_CANTCONNECT       17  // SQL Server does not exist or
access denied.

#define NE_MAX_NETERROR        17

#ifdef __cplusplus
}
#endif

#endif // _INC_SQLFRONT
```

SQLFUNCS.C

```
// TPC-C Benchmark Kit
//
// Module:  SQLFUNCS.C
// Author:  DamienL

// Includes
#include "tpcc.h"

long                    client_threads_dropped;
long                    delivery_threads_dropped;

//=======================================================
==============
// Function name: SQLMasterInit
//
//=======================================================
==============

int SQLMasterInit(MASTER_DATA *pMaster)
{
   long                 num_users;
   long                 num_delivery_hdlrs;
   char                 msg[80];
   int                  rc;
                        int         i;
                        char        dbname[30];
                        float       log_size_mb;
                        float       log_used_pct;

#ifdef DEBUG
                        printf("[%ld]DBG: Entering SQLMasterInit()...\n",
(int) GetCurrentThreadId());
#endif

                        // make sure advanced config options are turned
on
                        SQLExecCmd(pMaster->sqlconn,"exec
sp_configure 'show advanced option',1 reconfigure with override");

                        printf("Initializing synchronization tables...\n");

   SQLExecCmd(pMaster->sqlconn,"exec tpcc_sp_master_init");


                        dbcmd(pMaster->sqlconn,
                                "insert into
tpcc_master_sync(ramp_up, steady_state, ramp_down, "
                                "num_warehouses, think_times,
display_data, deadlock_retries, "
                                "client_mode, transaction_type,
next_client_id, next_delivery_id, load_multiplier, "
                                "delivery_backoff, disable_90th,
num_delivery_threads) ");
                        dbfcmd(pMaster->sqlconn,"values (%ld, %ld,
%ld, %ld, %ld, %ld, %ld, %ld, %ld, 0, 0, %f, %ld, %ld, %ld)",
                                pMaster->ramp_up,
                                pMaster->steady_state,
                                pMaster->ramp_down,
                                pMaster->num_warehouses,
                                pMaster->think_times,
                                pMaster->display_data,
                                pMaster->deadlock_retry,
                                pMaster->client_mode,
                                pMaster->tran,
                                pMaster->load_multiplier,
                                pMaster->delivery_backoff,
                                pMaster->disable_90th,
                                pMaster->num_deliveries);
                        SQLExec(pMaster->sqlconn);

}


//=======================================================
==============
//
// Function name: SQLClientInit
//
//=======================================================
==============

void SQLClientInit(CLIENT_DATA *pClient)
{
   char         buffer[400];
                        char                cmd[30];
                        RETCODE     rc;
#ifdef USE_CONMON
                        char        linebuf[CON_LINE_SIZE+1];
#endif

#ifdef DEBUG
                        printf("[%ld]DBG: Entering SQLClientInit()...\n",
(int) GetCurrentThreadId());
#endif

                        sprintf(buffer,"begin tran update
tpcc_master_sync set next_client_id = next_client_id + 1 "
                                "select ramp_up, steady_state,
ramp_down, num_warehouses, "
                                "think_times, display_data,
deadlock_retries, client_mode, "
                                "transaction_type, next_client_id,
load_multiplier, "
```

```
                                        "disable_90th,
num_delivery_threads from tpcc_master_sync commit tran ");

#ifdef USE_ODBC

                        sprintf(cmd,"use %s", pClient->admin_database);
                        rc = SQLExecDirect(pClient->hstmt, cmd,
SQL_NTS);

   if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientStats", "SQLExecDirect() failed.");
                        }

                        SQLFreeStmt(pClient->hstmt, SQL_CLOSE);

                        rc = SQLExecDirect(pClient->hstmt, buffer,
SQL_NTS);
   if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientInit", "SQLExecDirect() failed.");
                        }


                        /* removed becuase of the addition of the set
nocount option on ODBCOpenConnection

                        rc = SQLMoreResults(pClient->hstmt);
   if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientInit", "SQLMoreResults() failed.");
                        }
   */

                        rc = SQLBindCol(pClient->hstmt, 1,
SQL_C_SLONG, &pClient->ramp_up, 0 , NULL);
   if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientInit", "SQLBindCol() failed.");
                        }

                        rc = SQLBindCol(pClient->hstmt, 2,
SQL_C_SLONG, &pClient->steady_state, 0 , NULL);
                        if (rc == SQL_ERROR)
   {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientInit", "SQLBindCol() failed.");
                        }

                        rc = SQLBindCol(pClient->hstmt, 3,
SQL_C_SLONG, &pClient->ramp_down, 0 , NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientInit", "SQLBindCol() failed.");
                        }

                        rc = SQLBindCol(pClient->hstmt, 4,
SQL_C_SLONG, &pClient->num_warehouses, 0 , NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientInit", "SQLBindCol() failed.");
                        }

                        rc = SQLBindCol(pClient->hstmt, 5,
SQL_C_SLONG, &pClient->think_times, 0 , NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientInit", "SQLBindCol() failed.");
                        }

                        rc = SQLBindCol(pClient->hstmt, 6,
SQL_C_SLONG, &pClient->display_data, 0 , NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientInit", "SQLBindCol() failed.");
                        }

                        rc = SQLBindCol(pClient->hstmt, 7,
SQL_C_SLONG, &pClient->deadlock_retry, 0 , NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientInit", "SQLBindCol() failed.");
                        }

                        rc = SQLBindCol(pClient->hstmt, 8,
SQL_C_SLONG, &pClient->client_mode, 0 , NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientInit", "SQLBindCol() failed.");
                        }

                        rc = SQLBindCol(pClient->hstmt, 9,
SQL_C_SLONG, &pClient->tran, 0 , NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientInit", "SQLBindCol() failed.");
                        }

                        rc = SQLBindCol(pClient->hstmt, 10,
SQL_C_SLONG, &pClient->id, 0 , NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientInit", "SQLBindCol() failed.");
                        }

                        rc = SQLBindCol(pClient->hstmt, 11,
SQL_C_DOUBLE, &pClient->load_multiplier, 0 , NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientInit", "SQLBindCol() failed.");
                        }

                        rc = SQLBindCol(pClient->hstmt, 12,
SQL_C_SLONG, &pClient->disable_90th, 0 , NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientInit", "SQLBindCol() failed.");
                        }

                        rc = SQLBindCol(pClient->hstmt, 13,
SQL_C_SLONG, &pClient->num_deliveries, 0 , NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientInit", "SQLBindCol() failed.");
                        }

                        rc = SQLFetch(pClient->hstmt);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientInit", "SQLFetch() failed.");
                        }

                        SQLFreeStmt(pClient->hstmt, SQL_CLOSE);


                        sprintf(cmd,"use %s", pClient->database);
                        rc = SQLExecDirect(pClient->hstmt, cmd,
SQL_NTS);

   if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientStats", "SQLExecDirect() failed.");
                        }

                        SQLFreeStmt(pClient->hstmt, SQL_CLOSE);
#else

                        sprintf(cmd,"use %s",pClient->admin_database);
                        SQLExecCmd(pClient->sqlconn, cmd);
```

```
                        dbcmd(pClient->sqlconn, buffer);
    dbsqlexec(pClient->sqlconn);

    while (dbresults(pClient->sqlconn) != NO_MORE_RESULTS)
    {
                        if (DBROWS(pClient->sqlconn))
                        {
                                dbbind(pClient->sqlconn, 1,
INTBIND, (DBINT) 0,
                                        (BYTE *) &pClient-
>ramp_up);
                                dbbind(pClient->sqlconn, 2,
INTBIND, (DBINT) 0,
                                        (BYTE *) &pClient-
>steady_state);
                                dbbind(pClient->sqlconn, 3,
INTBIND, (DBINT) 0,
                                        (BYTE *) &pClient-
>ramp_down);
                                dbbind(pClient->sqlconn, 4,
INTBIND, (DBINT) 0,
                                        (BYTE *) &pClient-
>num_warehouses);
                                dbbind(pClient->sqlconn, 5,
INTBIND, (DBINT) 0,
                                        (BYTE *) &pClient-
>think_times);
                                dbbind(pClient->sqlconn, 6,
INTBIND, (DBINT) 0,
                                        (BYTE *) &pClient-
>display_data);
                                dbbind(pClient->sqlconn, 7,
INTBIND, (DBINT) 0,
                                        (BYTE *) &pClient-
>deadlock_retry);
                                dbbind(pClient->sqlconn, 8,
INTBIND, (DBINT) 0,
                                        (BYTE *) &pClient-
>client_mode);
                                dbbind(pClient->sqlconn, 9,
INTBIND, (DBINT) 0,
                                        (BYTE *) &pClient->tran);
                                dbbind(pClient->sqlconn, 10,
INTBIND, (DBINT) 0,
                                        (BYTE *) &pClient->id);
                                dbbind(pClient->sqlconn, 11,
FLT8BIND, (DBINT) 0,
                                        (BYTE *) &pClient-
>load_multiplier);
                                dbbind(pClient->sqlconn, 12,
INTBIND, (DBINT) 0,
                                        (BYTE *) &pClient-
>disable_90th);
                                dbbind(pClient->sqlconn, 13,
INTBIND, (DBINT) 0,
                                        (BYTE *) &pClient-
>num_deliveries);
                        }

                        while (dbnextrow(pClient->sqlconn) !=
NO_MORE_ROWS)
                                ;
                }

                sprintf(cmd,"use %s",pClient->database);
                SQLExecCmd(pClient->sqlconn, cmd);
#endif

    return;
}


//=========================================================
==============
//
// Function name: SQLDeliveryInit
//
//=========================================================
==============

void SQLDeliveryInit(DELIVERY *pDeliveryHdlr)
{

    char        buffer[300];
                char                cmd[30];
                RETCODE     rc;

#ifdef DEBUG
                printf("[%ld]DBG: Entering SQLDeliveryInit()...\n",
(int) GetCurrentThreadId());
#endif

                strcpy(buffer,"begin tran update
tpcc_master_sync set next_delivery_id = next_delivery_id + 1 "
                                        "select ramp_up,
steady_state, ramp_down, next_delivery_id, delivery_backoff, "
                                        "disable_90th from
tpcc_master_sync commit tran");

#ifdef USE_ODBC

                sprintf(cmd,"use %s", pDeliveryHdlr-
>admin_database);
                rc = SQLExecDirect(pDeliveryHdlr->hstmt, cmd,
SQL_NTS);

    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                {
                        ODBCError (henv, pDeliveryHdlr->hdbc,
pDeliveryHdlr->hstmt);
                        UtilFatalError(GetCurrentThreadId(),
"SQLDeliveryStats", "SQLExecDirect() failed.");
                }

                SQLFreeStmt(pDeliveryHdlr->hstmt,
SQL_CLOSE);

                rc = SQLExecDirect(pDeliveryHdlr->hstmt, buffer,
SQL_NTS);
    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                {
                        ODBCError (henv, pDeliveryHdlr->hdbc,
pDeliveryHdlr->hstmt);
                        UtilFatalError(GetCurrentThreadId(),
"SQLDeliveryInit", "SQLExecDirect() failed.");
                }

                /* removed becuase of the addition of the set
nocount option on ODBCOpenConnection

                rc = SQLMoreResults(pDeliveryHdlr->hstmt);
    if (rc == SQL_ERROR)
                {
                        ODBCError (henv, pDeliveryHdlr->hdbc,
pDeliveryHdlr->hstmt);
                        UtilFatalError(GetCurrentThreadId(),
"SQLDeliveryInit", "SQLExecDirect() failed.");
                }
                */

                rc = SQLBindCol(pDeliveryHdlr->hstmt, 1,
SQL_C_SLONG, &pDeliveryHdlr->ramp_up, 0 , NULL);
    if (rc == SQL_ERROR)
                {
                        ODBCError (henv, pDeliveryHdlr->hdbc,
pDeliveryHdlr->hstmt);
                        UtilFatalError(GetCurrentThreadId(),
"SQLDeliveryInit", "SQLBindCol() failed.");
                }

                rc = SQLBindCol(pDeliveryHdlr->hstmt, 2,
SQL_C_SLONG, &pDeliveryHdlr->steady_state, 0 , NULL);
                if (rc == SQL_ERROR)
    {
                        ODBCError (henv, pDeliveryHdlr->hdbc,
pDeliveryHdlr->hstmt);
                        UtilFatalError(GetCurrentThreadId(),
"SQLDeliveryInit", "SQLBindCol() failed.");
                }

                rc = SQLBindCol(pDeliveryHdlr->hstmt, 3,
SQL_C_SLONG, &pDeliveryHdlr->ramp_down, 0 , NULL);
                if (rc == SQL_ERROR)
                {
                        ODBCError (henv, pDeliveryHdlr->hdbc,
pDeliveryHdlr->hstmt);
                        UtilFatalError(GetCurrentThreadId(),
"SQLDeliveryInit", "SQLBindCol() failed.");
                }

                rc = SQLBindCol(pDeliveryHdlr->hstmt,  4,
SQL_C_SLONG, &pDeliveryHdlr->id, 0 , NULL);
                if (rc == SQL_ERROR)
                {
                        ODBCError (henv, pDeliveryHdlr->hdbc,
pDeliveryHdlr->hstmt);
                        UtilFatalError(GetCurrentThreadId(),
"SQLDeliveryInit", "SQLBindCol() failed.");
                }

                rc = SQLBindCol(pDeliveryHdlr->hstmt, 5,
SQL_C_SLONG, &pDeliveryHdlr->delivery_backoff, 0 , NULL);
                if (rc == SQL_ERROR)
                {
                        ODBCError (henv, pDeliveryHdlr->hdbc,
pDeliveryHdlr->hstmt);
                        UtilFatalError(GetCurrentThreadId(),
"SQLDeliveryInit", "SQLBindCol() failed.");
                }

                rc = SQLBindCol(pDeliveryHdlr->hstmt, 6,
SQL_C_SLONG, &pDeliveryHdlr->disable_90th, 0 , NULL);
                if (rc == SQL_ERROR)
                {
                        ODBCError (henv, pDeliveryHdlr->hdbc,
pDeliveryHdlr->hstmt);
                        UtilFatalError(GetCurrentThreadId(),
"SQLDeliveryInit", "SQLBindCol() failed.");
                }

                rc = SQLFetch(pDeliveryHdlr->hstmt);
```

```
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, pDeliveryHdlr->hdbc,
pDeliveryHdlr->hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLDeliveryInit", "SQLFetch() failed.");
                        }

                        SQLFreeStmt(pDeliveryHdlr->hstmt,
SQL_CLOSE);

                        sprintf(cmd,"use %s", pDeliveryHdlr->database);
                        rc = SQLExecDirect(pDeliveryHdlr->hstmt, cmd,
SQL_NTS);

    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                        {
                                ODBCError (henv, pDeliveryHdlr->hdbc,
pDeliveryHdlr->hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLDeliveryStats", "SQLExecDirect() failed.");
                        }

                        SQLFreeStmt(pDeliveryHdlr->hstmt,
SQL_CLOSE);

#else

                        sprintf(cmd,"use %s",pDeliveryHdlr-
>admin_database);
                        SQLExecCmd(pDeliveryHdlr->sqlconn, cmd);

                        dbfcmd(pDeliveryHdlr->sqlconn, buffer);
    dbsqlexec(pDeliveryHdlr->sqlconn);

    while (dbresults(pDeliveryHdlr->sqlconn) != NO_MORE_RESULTS)
    {

                                if (DBROWS(pDeliveryHdlr->sqlconn))
                                {
                                        dbbind(pDeliveryHdlr->sqlconn, 1,
INTBIND, (DBINT) 0,
                                                (BYTE *) &pDeliveryHdlr-
>ramp_up);
                                        dbbind(pDeliveryHdlr->sqlconn, 2,
INTBIND, (DBINT) 0,
                                                (BYTE *) &pDeliveryHdlr-
>steady_state);
                                        dbbind(pDeliveryHdlr->sqlconn, 3,
INTBIND, (DBINT) 0,
                                                (BYTE *) &pDeliveryHdlr-
>ramp_down);
                                        dbbind(pDeliveryHdlr->sqlconn, 4,
INTBIND, (DBINT) 0,
                                                (BYTE *) &pDeliveryHdlr-
>id);
                                        dbbind(pDeliveryHdlr->sqlconn, 5,
INTBIND, (DBINT) 0,
                                                (BYTE *) &pDeliveryHdlr-
>delivery_backoff);
                                        dbbind(pDeliveryHdlr->sqlconn, 6,
INTBIND, (DBINT) 0,
                                                (BYTE *) &pDeliveryHdlr-
>disable_90th);
                                }

                        while (dbnextrow(pDeliveryHdlr->sqlconn) !=
NO_MORE_ROWS)
                                ;
    }

                        sprintf(cmd,"use %s",pDeliveryHdlr->database);
                        SQLExecCmd(pDeliveryHdlr->sqlconn, cmd);

#endif

    return;
}


//=====================================================
==============
//
// Function name: SQLNewOrder
//
//=====================================================
==============

#ifdef USE_ODBC
BOOL SQLNewOrder(HDBC           hdbc,
                                        HSTMT           hstmt,
#else
BOOL SQLNewOrder(DBPROCESS      *dbproc,
#endif
#ifdef USE_CONMON
                                        NEW_ORDER_DATA
*pNewOrder,
                                        short           id,
                                short           w_id,
                                        HANDLE          hConMon,
                                        short           con_x,
                                        short           con_y,
                                        short           deadlock_retry)
#else
                                        NEW_ORDER_DATA
*pNewOrder,
                                        short           deadlock_retry)
#endif

{

    RETCODE             rc;
    int             i;
                        int             j;
    DBINT           status;
                        DBINT                           commit_flag;
                        int             tryit;
                        char            printbuf[25];
                        char                    tmpbuf[30];
#ifdef USE_CONMON
                        char            linebuf[CON_LINE_SIZE+1];
#endif
#ifdef USE_ODBC
                        char            buffer[255];
                        BOOL            deadlock_detected;
#else
                        DBDATETIME      datetime;
                        BYTE            *pData;
#endif


#ifdef DEBUG
                        printf("[%ld]DBG: Entering SQLNewOrder()...\n",
(int) GetCurrentThreadId());
#endif

                        pNewOrder->num_deadlocks = 0;

                        strcpy(tmpbuf, "tpcc_neworder");

    for (tryit=0; tryit < deadlock_retry; tryit++)
    {

#ifdef DEBUG
                        printf("[%ld]DBG: Executing NewOrder
transaction...\n", (int) GetCurrentThreadId());
#endif

#ifdef USE_ODBC

                        deadlock_detected = FALSE;

                        sprintf(buffer,"{call %s(?,?,?,?,?,",tmpbuf);
                        for (i = 1; i <= (pNewOrder->o_ol_cnt - 1);
i++)
                                strcat(buffer, "?,?,?,");
                        strcat(buffer, "?,?,?)}");

                        // Bind Parameters
                        rc = SQLBindParameter(hstmt, 1,
SQL_PARAM_INPUT, SQL_C_SSHORT,

                        SQL_SMALLINT, 0, 0, &pNewOrder->w_id, 0,
NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, hdbc, hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLBindParameter() failed.");
                        }

                        rc = SQLBindParameter(hstmt, 2,
SQL_PARAM_INPUT, SQL_C_STINYINT,

                        SQL_TINYINT, 0, 0, &pNewOrder->d_id, 0,
NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, hdbc, hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLBindParameter() failed.");
                        }

                        rc = SQLBindParameter(hstmt, 3,
SQL_PARAM_INPUT, SQL_C_SLONG,

                        SQL_INTEGER, 0, 0, &pNewOrder->c_id, 0,
NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, hdbc, hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLBindParameter() failed.");
                        }

                        rc = SQLBindParameter(hstmt, 4,
SQL_PARAM_INPUT, SQL_C_STINYINT,

                        SQL_TINYINT, 0, 0, &pNewOrder->o_ol_cnt, 0,
NULL);
                        if (rc == SQL_ERROR)
```

```
                {
                        ODBCError (henv, hdbc, hstmt);

                UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLBindParameter() failed.");
                }

                rc = SQLBindParameter(hstmt, 5,
SQL_PARAM_INPUT, SQL_C_STINYINT,

                SQL_TINYINT, 0, 0, &pNewOrder->o_all_local,
0, NULL);
                if (rc == SQL_ERROR)
                {
                        ODBCError (henv, hdbc, hstmt);

                UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLBindParameter() failed.");
                }

                j=0;
                for (i = 0; i < (pNewOrder->o_ol_cnt * 3);
i=i+3)
                {

                        rc = SQLBindParameter(hstmt, i+6,
SQL_PARAM_INPUT, SQL_C_SLONG,

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLBindParameter() failed.");
                                }

                        rc = SQLBindParameter(hstmt, i+7,
SQL_PARAM_INPUT, SQL_C_SSHORT,

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLBindParameter() failed.");
                                }

                        rc = SQLBindParameter(hstmt, i+8,
SQL_PARAM_INPUT, SQL_C_SSHORT,

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLBindParameter() failed.");
                                }

                                j++;

                }

                rc = SQLExecDirect(hstmt, buffer,
SQL_NTS);

                if (rc != SQL_SUCCESS && rc !=
SQL_SUCCESS_WITH_INFO)
                {
                        deadlock_detected = ODBCError
(henv, hdbc, hstmt);
                        if (!deadlock_detected)

                UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLExecDirect() failed.");
                }

                pNewOrder->total_amount=0;

                for (i = 0; i<pNewOrder->o_ol_cnt &&
!deadlock_detected; i++)
                {

                        // Now bind order line results

                        rc = SQLBindCol(hstmt, 1,
SQL_C_CHAR, &pNewOrder->Ol[i].ol_i_name, sizeof(pNewOrder-
>Ol[i].ol_i_name), NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
        hstmt);
SQL_INTEGER, 0, 0, &pNewOrder->Ol[j].ol_i_id, 0, NULL);
                                UtilFatalError(GetCurrentThreadId(),
        "SQLNewOrder", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 2,
        SQL_C_SSHORT, &pNewOrder->Ol[i].ol_stock, 0 , NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
        hstmt);
SQL_SMALLINT, 0, 0, &pNewOrder->Ol[j].ol_supply_w_id, 0, NULL);
                                UtilFatalError(GetCurrentThreadId(),
        "SQLNewOrder", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 3,
        SQL_C_CHAR, &pNewOrder->Ol[i].ol_brand_generic, sizeof(pNewOrder-
        >Ol[i].ol_brand_generic), NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
SQL_SMALLINT, 0, 0, &pNewOrder->Ol[j].ol_quantity, 0, NULL);

                                UtilFatalError(GetCurrentThreadId(),
        "SQLNewOrder", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 4,
        SQL_C_DOUBLE, &pNewOrder->Ol[i].ol_i_price, 0 , NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
        hstmt);

                                UtilFatalError(GetCurrentThreadId(),
        "SQLNewOrder", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 5,
SQL_C_DOUBLE, &pNewOrder->Ol[i].ol_amount, 0 , NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLBindCol() failed.");
                                }

                                // Fetch next row
                                rc = SQLFetch(hstmt);

                                if (rc == SQL_ERROR)
                                {
                                        deadlock_detected =
ODBCError (henv, hdbc, hstmt);
                                        if (!deadlock_detected)

                                UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLFetch() failed.");
                                }

                                pNewOrder->total_amount =
pNewOrder->total_amount
                                                                                + pNe

                                if (!deadlock_detected)
                                {

                                        rc = SQLMoreResults(hstmt);

                                        if (rc == SQL_ERROR)
                                        {
                                                deadlock_detected =
ODBCError (henv, hdbc, hstmt);
                                                if
(!deadlock_detected)

                                UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLMoreResults() failed.");
                                        }

                                }

                        }

                        if (!deadlock_detected)
                        {

                                // Bind return cols
                                rc = SQLBindCol(hstmt, 1,
SQL_C_DOUBLE, &pNewOrder->w_tax, 0, NULL);
                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 2,
SQL_C_DOUBLE, &pNewOrder->d_tax, 0, NULL);
                                if (rc == SQL_ERROR)
```

```
                {
                        ODBCError (henv, hdbc,
hstmt);

                UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLBindCol() failed.");
                }

                rc = SQLBindCol(hstmt, 3,
SQL_C_SLONG, &pNewOrder->o_id, 0, NULL);
                if (rc == SQL_ERROR)
                {
                        ODBCError (henv, hdbc,
hstmt);

                UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLBindCol() failed.");
                }

                rc = SQLBindCol(hstmt, 4,
SQL_C_CHAR, &pNewOrder->c_last, sizeof(pNewOrder->c_last), NULL);
                if (rc == SQL_ERROR)
                {
                        ODBCError (henv, hdbc,
hstmt);

                UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLBindCol() failed.");
                }

                rc = SQLBindCol(hstmt, 5,
SQL_C_DOUBLE, &pNewOrder->c_discount, 0, NULL);
                if (rc == SQL_ERROR)
                {
                        ODBCError (henv, hdbc,
hstmt);

                UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLBindCol() failed.");
                }

                rc = SQLBindCol(hstmt, 6,
SQL_C_CHAR, &pNewOrder->c_credit, sizeof(pNewOrder->c_credit),
NULL);
                if (rc == SQL_ERROR)
                {
                        ODBCError (henv, hdbc,
hstmt);

                UtilFatalError(GetCurrentThreadId() ,
"SQLNewOrder", "SQLBindCol() failed.");
                }

                rc = SQLBindCol(hstmt, 7,
SQL_C_TIMESTAMP, &pNewOrder->o_entry_d, 0, NULL);
                if (rc == SQL_ERROR)
                {
                        ODBCError (henv, hdbc,
hstmt);

                UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLBindCol() failed.");
                }

                rc = SQLBindCol(hstmt, 8,
SQL_C_SLONG, &commit_flag, 0, NULL);
                if (rc == SQL_ERROR)
                {
                        ODBCError (henv, hdbc,
hstmt);

                UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLBindCol() failed.");
                }

                // Now fetch results
                rc = SQLFetch(hstmt);
                if (rc == SQL_ERROR)
                {
                        deadlock_detected =
ODBCError (henv, hdbc, hstmt);
                        if (!deadlock_detected)

                UtilFatalError(GetCurrentThreadId(),
"SQLNewOrder", "SQLFetch() failed.");
                }

        }

        SQLFreeStmt(hstmt, SQL_CLOSE);

#else
        if (dbrpcinit(dbproc, tmpbuf, 0) ==
SUCCEED)
        {
                dbrpcparam(dbproc, NULL, 0,
SQLINT2, -1, -1, (BYTE *) &pNewOrder->w_id);
                dbrpcparam(dbproc, NULL, 0,
SQLINT1, -1, -1, (BYTE *) &pNewOrder->d_id);
                dbrpcparam(dbproc, NULL, 0,
SQLINT4, -1, -1, (BYTE *) &pNewOrder->c_id);
                dbrpcparam(dbproc, NULL, 0,
SQLINT1, -1, -1, (BYTE *) &pNewOrder->o_ol_cnt);
                dbrpcparam(dbproc, NULL, 0,
SQLINT1, -1, -1, (BYTE *) &pNewOrder->o_all_local);

                for (i = 0; i < pNewOrder->o_ol_cnt;
i++)
                {
                        dbrpcparam(dbproc, NULL,
0, SQLINT4, -1, -1, (BYTE *) &pNewOrder->Ol[i].ol_i_id);
                        dbrpcparam(dbproc, NULL,
0, SQLINT2, -1, -1, (BYTE *) &pNewOrder->Ol[i].ol_supply_w_id);
                        dbrpcparam(dbproc, NULL,
0, SQLINT2, -1, -1, (BYTE *) &pNewOrder->Ol[i].ol_quantity);
                }

                if (dbrpcexec(dbproc) ==
SUCCEED)
                {
                        pNewOrder-
>total_amount=0;

                        // Get resutls from order line

                        for (i = 0; i<pNewOrder-
>o_ol_cnt; i++)
                        {

                                if (((rc =
dbresults(dbproc)) != NO_MORE_RESULTS) && (rc != FAIL))
                                {
                                        if
(DBROWS(dbproc) && (dbnumcols(dbproc) == 5))
                                        {
                                                while
(dbnextrow(dbproc) != NO_MORE_ROWS)
                                                {

                                                        if(pData=dbdata(dbproc, 1))
                                                                UtilStrCpy(p

                                                        if(pData=dbdata(dbproc, 2))
                                                                pNewOrder-

                                                        if(pData=dbdata(dbproc, 3))
                                                                UtilStrCpy(p

                                                        if(pData=dbdata(dbproc, 4))
                                                                pNewOrder-

                                                        if(pData=dbdata(dbproc, 5))
                                                                pNewOrder-

                                                        pNewOrder->total_amount = pNewOrder-
>total_amount
                                                                + pNewOrde
                                                }
                                        }
                                }

                                while (((rc =
dbresults(dbproc)) != NO_MORE_RESULTS) && (rc != FAIL))
                                {
                                        if (DBROWS(dbproc)
&& (dbnumcols(dbproc) == 8))
                                        {
                                                while (((rc =
dbnextrow(dbproc)) != NO_MORE_ROWS) && (rc != FAIL))
                                                {

                        if(pData=dbdata(dbproc, 1))

                        pNewOrder->w_tax = (*(DBFLT8 *) pData);

                        if(pData=dbdata(dbproc, 2))

                        pNewOrder->d_tax = (*(DBFLT8 *) pData);

                        if(pData=dbdata(dbproc, 3))

                        pNewOrder->o_id = (*(DBINT *) pData);

                        if(pData=dbdata(dbproc, 4))

                        UtilStrCpy(pNewOrder->c_last, pData,
dbdatlen(dbproc, 4));

                        if(pData=dbdata(dbproc, 5))

                        pNewOrder->c_discount = (*(DBFLT8 *) pData);
```

```
                        if(pData=dbdata(dbproc, 6))

                        UtilStrCpy(pNewOrder->c_credit, pData,
dbdatlen(dbproc, 6));


                        if(pData=dbdata(dbproc, 7))
                                                {

                        datetime = *((DBDATETIME *) pData);

                        dbdatecrack(dbproc, &pNewOrder->o_entry_d,
&datetime);
                                                }


                        if(pData=dbdata(dbproc, 8))

                        commit_flag = (*(DBTINYINT *) pData);
                                                }
                                        }
                                }
                                }
                        }

#endif

#ifdef USE_ODBC
                        if (deadlock_detected)
#else
                        if (SQLDetectDeadlock(dbproc))
#endif
                        {
                                pNewOrder->num_deadlocks++;
#ifdef USE_CONMON
                                sprintf(linebuf,"[%04ld:%04ld]
NewOrder: deadlock:%ld", (int) id,
                                        (int) w_id, (int) pNewOrder-
>num_deadlocks);
                                WriteConsoleString(hConMon,
linebuf, con_x, con_y,RED,TRUE);
                                total_deadlocks++;
                                sprintf(linebuf,"%d",
total_deadlocks);
                                WriteConsoleString(hConMon,
linebuf, DEADLOCK_X, DEADLOCK_Y,RED,TRUE);
#else
                                sprintf(printbuf,"deadlock: retry:
%d",pNewOrder->num_deadlocks);

                        UtilError(GetCurrentThreadId(),"SQLNewOrder",
printbuf);
#endif
                                Sleep(DEADLOCKWAIT*tryit);

                        }
                        else
                        {
                                if (commit_flag == 1)
                                {
                                        pNewOrder->total_amount =
pNewOrder->total_amount *
                                                ((1 +
pNewOrder->w_tax + pNewOrder->d_tax) * (1 - pNewOrder->c_discount));

                                        strcpy(pNewOrder-
>execution_status,"Transaction commited.");
                                        return TRUE;
                                }
                                else
                                {
                                        strcpy(pNewOrder-
>execution_status,"Item number is not valid.");
                                        return FALSE;
                                }
                        }
   }

   // If we reached here, it means we quit after MAX_RETRY deadlocks
                        strcpy(pNewOrder->execution_status,"Hit
deadlock max.  ");
#ifdef USE_CONMON
                        sprintf(linebuf,"[%04ld:%04ld] NewOrder:
deadlock max", (int) id, (int) w_id);
                        WriteConsoleString(hConMon, linebuf, con_x,
con_y,RED,TRUE);
#else
                        UtilError(GetCurrentThreadId(),"SQLNewOrder","
deadlock max retry reached!");
#endif

   return FALSE;
}


//=============================================================
==============
//
// Function name: SQLPayment
//
//=============================================================
==============

#ifdef USE_ODBC
BOOL SQLPayment(HDBC            hdbc,
                                HSTMT           hstmt,
#else
BOOL SQLPayment(DBPROCESS       *dbproc,
#endif
#ifdef USE_CONMON
                                PAYMENT_DATA
*pPayment,
                                short           id,
                                short           w_id,
                                HANDLE          hConMon,
                                short           con_x,
                                short           con_y,
                                short           deadlock_retry)
#else
                        PAYMENT_DATA            *pPayment,

                                short           deadlock_retry)
#endif
{
   RETCODE              rc;
                        int             i;
   int                  tryit;
                        char            cmd_buf[255];
                        char            printbuf[25];
                        BOOL            by_name;
#ifdef USE_CONMON
                        char            linebuf[CON_LINE_SIZE+1];
#endif
#ifdef USE_ODBC
                        char            buffer[255];
                        BOOL            deadlock_detected;
#else
                        DBDATETIME      datetime;
                        BYTE            *pData;
#endif


#ifdef DEBUG
                        printf("[%ld]DBG: Entering SQLPayment()...\n",
(int) GetCurrentThreadId());
#endif

   pPayment->num_deadlocks = 0;

                        if (pPayment->c_id == 0)
                        {
                                by_name = TRUE;
                        }
                        else
                        {
                                by_name = FALSE;
                        }

   for (tryit=0; tryit < deadlock_retry; tryit++)
   {

#ifdef USE_ODBC
                        deadlock_detected = FALSE;
#endif

#ifdef USE_ODBC

                        strcpy(buffer,"{call
tpcc_payment(?,?,?,?,?,?)");
                        if (pPayment->c_id == 0)
                        {
                                strcat(buffer,",?");
                        }
                        strcat(buffer,")}");

                        // Bind Parameters
                        rc = SQLBindParameter(hstmt, 1,
SQL_PARAM_INPUT, SQL_C_SSHORT,

                        SQL_SMALLINT, 0, 0, &pPayment->w_id, 0,
NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, hdbc, hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindParameter() failed.");
                        }

                        rc = SQLBindParameter(hstmt, 2,
SQL_PARAM_INPUT, SQL_C_SSHORT,

                        SQL_SMALLINT, 0, 0, &pPayment->c_w_id, 0,
NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, hdbc, hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindParameter() failed.");
                        }
```

```
                        rc = SQLBindParameter(hstmt, 3,
SQL_PARAM_INPUT, SQL_C_DOUBLE,

                SQL_NUMERIC, 6, 2, &pPayment->h_amount, 0,
NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, hdbc, hstmt);

                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindParameter() failed.");
                        }

                        rc = SQLBindParameter(hstmt, 4,
SQL_PARAM_INPUT, SQL_C_STINYINT,

                SQL_TINYINT, 0, 0, &pPayment->d_id, 0,
NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, hdbc, hstmt);

                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindParameter() failed.");
                        }

                        rc = SQLBindParameter(hstmt, 5,
SQL_PARAM_INPUT, SQL_C_STINYINT,

                SQL_TINYINT, 0, 0, &pPayment->c_d_id, 0,
NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, hdbc, hstmt);

                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindParameter() failed.");
                        }

                        rc = SQLBindParameter(hstmt, 6,
SQL_PARAM_INPUT, SQL_C_SLONG,

                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, hdbc, hstmt);

                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindParameter() failed.");
                        }

                        if (pPayment->c_id == 0)
                        {
                                rc = SQLBindParameter(hstmt, 7,
SQL_PARAM_INPUT, SQL_C_CHAR,

                SQL_CHAR, SQL_NTS, 0, &pPayment->c_last,
sizeof(pPayment->c_last), NULL);
                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindParameter() failed.");
                                }
                        }
                        rc = SQLExecDirect(hstmt, buffer,
SQL_NTS);

                        if (rc != SQL_SUCCESS && rc !=
SQL_SUCCESS_WITH_INFO)
                        {
                                deadlock_detected = ODBCError
(henv, hdbc, hstmt);
                                if (!deadlock_detected)

                        UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLExecDirect() failed.");
                        }

#else
                        //  Execute transaction

                if (dbrpcinit(dbproc, "tpcc_payment", 0) ==
SUCCEED)
                        {
                                dbrpcparam(dbproc, NULL, 0,
SQLINT2, -1, -1, (BYTE *) &pPayment->w_id);
                                dbrpcparam(dbproc, NULL, 0,
SQLINT2, -1, -1, (BYTE *) &pPayment->c_w_id);
                                dbrpcparam(dbproc, NULL, 0,
SQLFLT8, -1, -1, (BYTE *) &pPayment->h_amount);
                                dbrpcparam(dbproc, NULL, 0,
SQLINT1, -1, -1, (BYTE *) &pPayment->d_id);
                                dbrpcparam(dbproc, NULL, 0,
SQLINT1, -1, -1, (BYTE *) &pPayment->c_d_id);
                                dbrpcparam(dbproc, NULL, 0,
SQLINT4, -1, -1, (BYTE *) &pPayment->c_id);
                                if (pPayment->c_id == 0)
                                {
                                        dbrpcparam(dbproc, NULL,
0, SQLCHAR, -1, strlen(pPayment->c_last), pPayment->c_last);
                                }
                        }
#endif

#ifdef USE_ODBC
SQL_INTEGER, SQL_NTS, 0, &pPayment->c_id, 0, NULL);
                        if (!deadlock_detected)
                        {

                                rc = SQLBindCol(hstmt, 1,
SQL_C_SLONG, &pPayment->c_id, 0, NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 2,
SQL_C_CHAR, &pPayment->c_last, sizeof(pPayment->c_last) , NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }
                                rc = SQLBindCol(hstmt, 3,
SQL_C_TIMESTAMP, &pPayment->h_date, 0, NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 4,
SQL_C_CHAR, &pPayment->w_street_1, sizeof(pPayment->w_street_1),
NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 5,
SQL_C_CHAR, &pPayment->w_street_2, sizeof(pPayment->w_street_2),
NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 6,
SQL_C_CHAR, &pPayment->w_city, sizeof(pPayment->w_city), NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 7,
SQL_C_CHAR, &pPayment->w_state, sizeof(pPayment->w_state) , NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 8,
SQL_C_CHAR, &pPayment->w_zip, sizeof(pPayment->w_zip), NULL);

                                if (rc == SQL_ERROR)
                                {
```

```
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 9,
SQL_C_CHAR, &pPayment->d_street_1, sizeof(pPayment->d_street_1) ,
NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 10,
SQL_C_CHAR, &pPayment->d_street_2, sizeof(pPayment->d_street_2),
NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 11,
SQL_C_CHAR, &pPayment->d_city, sizeof(pPayment->d_city), NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 12,
SQL_C_CHAR, &pPayment->d_state, sizeof(pPayment->d_state), NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 13,
SQL_C_CHAR, &pPayment->d_zip, sizeof(pPayment->d_zip), NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 14,
SQL_C_CHAR, &pPayment->c_first, sizeof(pPayment->c_first), NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 15,
SQL_C_CHAR, &pPayment->c_middle, sizeof(pPayment->c_middle),
NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 16,
SQL_C_CHAR, &pPayment->c_street_1, sizeof(pPayment->c_street_1),
NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 17,
SQL_C_CHAR, &pPayment->c_street_2, sizeof(pPayment->c_street_2),
NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 18,
SQL_C_CHAR, &pPayment->c_city, sizeof(pPayment->c_city), NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 19,
SQL_C_CHAR, &pPayment->c_state, sizeof(pPayment->c_state), NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 20,
SQL_C_CHAR, &pPayment->c_zip, sizeof(pPayment->c_zip), NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 21,
SQL_C_CHAR, &pPayment->c_phone, sizeof(pPayment->c_phone) ,
NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 22,
SQL_C_TIMESTAMP, &pPayment->c_since, 0, NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 23,
SQL_C_CHAR, &pPayment->c_credit, sizeof(pPayment->c_credit), NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 24,
SQL_C_DOUBLE, &pPayment->c_credit_lim, 0 , NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                }
```

```
                                        rc = SQLBindCol(hstmt, 25,
SQL_C_DOUBLE, &pPayment->c_discount, 0, NULL);

                                        if (rc == SQL_ERROR)
                                        {
                                                ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                        }

                                        rc = SQLBindCol(hstmt, 26,
SQL_C_DOUBLE, &pPayment->c_balance, 0, NULL);

                                        if (rc == SQL_ERROR)
                                        {
                                                ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                        }

                                        rc = SQLBindCol(hstmt, 27,
SQL_C_CHAR, &pPayment->c_data, sizeof(pPayment->c_data), NULL);

                                        if (rc == SQL_ERROR)
                                        {
                                                ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLBindCol() failed.");
                                        }

                                        rc = SQLFetch(hstmt);

                                        if (rc == SQL_ERROR)
                                        {
                                                deadlock_detected =
ODBCError (henv, hdbc, hstmt);
                                                if (!deadlock_detected)

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLFetch() failed.");
                                        }

                                }

                                SQLFreeStmt(hstmt, SQL_CLOSE);

#else
                                if (dbrpcexec(dbproc) == SUCCEED)
                                {
                                        while (((rc = dbresults(dbproc)) !=
NO_MORE_RESULTS) && (rc != FAIL))
                                        {
                                                if (DBROWS(dbproc) &&
(dbnumcols(dbproc) == 27))
                                                {
                                                        while (((rc =
dbnextrow(dbproc)) != NO_MORE_ROWS) && (rc != FAIL))
                                                        {

if(pData=dbdata(dbproc, 1))
                                                                pPayment->c_id = *((DBINT *) pData);

                                                        if(pData=dbdata(dbproc, 2))

                                                                UtilStrCpy(pPayment->c_last, pData,
dbdatlen(dbproc, 2));

                                                        if(pData=dbdata(dbproc, 3))
                                                                {
                                                                        datetime
= *((DBDATETIME *) pData);
                                                                dbdatecrack(dbproc, &pPayment->h_date,
&datetime);
                                                                }

                                                        if(pData=dbdata(dbproc, 4))

                                                                UtilStrCpy(pPayment->w_street_1, pData,
dbdatlen(dbproc, 4));

                                                        if(pData=dbdata(dbproc, 5))

                                                                UtilStrCpy(pPayment->w_street_2, pData,
dbdatlen(dbproc, 5));

                                                        if(pData=dbdata(dbproc, 6))

                                                                UtilStrCpy(pPayment->w_city, pData,
dbdatlen(dbproc, 6));

                                                        if(pData=dbdata(dbproc, 7))

                                                                UtilStrCpy(pPayment->w_state, pData,
dbdatlen(dbproc, 7));

                                                        if(pData=dbdata(dbproc, 8))

                                                                UtilStrCpy(pPayment->w_zip, pData,
dbdatlen(dbproc, 8));

                                                        if(pData=dbdata(dbproc, 9))

                                                                UtilStrCpy(pPayment->d_street_1, pData,
dbdatlen(dbproc, 9));

                                                        if(pData=dbdata(dbproc, 10))

                                                                UtilStrCpy(pPayment->d_street_2, pData,
dbdatlen(dbproc, 10));

                                                        if(pData=dbdata(dbproc, 11))

                                                                UtilStrCpy(pPayment->d_city, pData,
dbdatlen(dbproc, 11));

                                                        if(pData=dbdata(dbproc, 12))

                                                                UtilStrCpy(pPayment->d_state, pData,
dbdatlen(dbproc, 12));

                                                        if(pData=dbdata(dbproc, 13))

                                                                UtilStrCpy(pPayment->d_zip, pData,
dbdatlen(dbproc, 13));

                                                        if(pData=dbdata(dbproc, 14))

                                                                UtilStrCpy(pPayment->c_first, pData,
dbdatlen(dbproc, 14));

                                                        if(pData=dbdata(dbproc, 15))

                                                                UtilStrCpy(pPayment->c_middle, pData,
dbdatlen(dbproc, 15));

                                                        if(pData=dbdata(dbproc, 16))

                                                                UtilStrCpy(pPayment->c_street_1, pData,
dbdatlen(dbproc, 16));

                                                        if(pData=dbdata(dbproc, 17))

                                                                UtilStrCpy(pPayment->c_street_2, pData,
dbdatlen(dbproc, 17));

                                                        if(pData=dbdata(dbproc, 18))

                                                                UtilStrCpy(pPayment->c_city, pData,
dbdatlen(dbproc, 18));

                                                        if(pData=dbdata(dbproc, 19))

                                                                UtilStrCpy(pPayment->c_state, pData,
dbdatlen(dbproc, 19));

                                                        if(pData=dbdata(dbproc, 20))

                                                                UtilStrCpy(pPayment->c_zip, pData,
dbdatlen(dbproc, 20));

                                                        if(pData=dbdata(dbproc, 21))

                                                                UtilStrCpy(pPayment->c_phone, pData,
dbdatlen(dbproc, 21));

                                                        if(pData=dbdata(dbproc, 22))
                                                                {
                                                                        datetime
= *((DBDATETIME *) pData);

                                                                dbdatecrack(dbproc, &pPayment->c_since,
&datetime);
```

```
                                        }

                        if(pData=dbdata(dbproc, 23))
                        UtilStrCpy(pPayment->c_credit, pData,
dbdatlen(dbproc, 23));

                        if(pData=dbdata(dbproc, 24))
                        pPayment->c_credit_lim = (*(DBFLT8 *) pData);

                        if(pData=dbdata(dbproc, 25))
                        pPayment->c_discount = (*(DBFLT8 *) pData);

                        if(pData=dbdata(dbproc, 26))
                        pPayment->c_balance = (*(DBFLT8 *) pData);

                        if(pData=dbdata(dbproc, 27))
                        UtilStrCpy(pPayment->c_data, pData,
dbdatlen(dbproc, 27));
                                        }
                                }
                        }
                }
#endif

#ifdef USE_ODBC
                if (deadlock_detected)
#else
                if (SQLDetectDeadlock(dbproc))
#endif
                {

                        pPayment->num_deadlocks++;
#ifdef USE_CONMON
                        sprintf(linebuf,"[%04ld:%04ld]
Payment: deadlock:%ld",
                                (int) id, (int) w_id, (int)
pPayment->num_deadlocks);
                        WriteConsoleString(hConMon,
linebuf, con_x, con_y,RED,TRUE);
                        total_deadlocks++;
                        sprintf(linebuf,"%d",
total_deadlocks);
                        WriteConsoleString(hConMon,
linebuf, DEADLOCK_X, DEADLOCK_Y,RED,TRUE);
#else
                        sprintf(printbuf,"deadlock: retry:
%d",pPayment->num_deadlocks);
                        UtilError(GetCurrentThreadId(),"SQLPayment",pr
intbuf);
#endif

                        Sleep(DEADLOCKWAIT*tryit);

                }
                else
                {
                        strcpy(pPayment-
>execution_status,"Transaction commited.");
                        return TRUE;
                }
        }

        // If we reached here, it means we quit after MAX_RETRY deadlocks
                        strcpy(pPayment->execution_status,"Hit
deadlock max.  ");
#ifdef USE_CONMON
                        sprintf(linebuf,"[%04ld:%04ld] Payment: deadlock
max",
                                (int) id, (int) w_id);
                        WriteConsoleString(hConMon, linebuf, con_x,
con_y,RED,TRUE);
#else
                        UtilError(GetCurrentThreadId(),"SQLPayment","d
eadlock max retry reached!");
#endif
        return FALSE;

}


//=======================================================
==============
//
// Function name: SQLOrderStatus
//
//=======================================================
==============

#ifdef USE_ODBC
BOOL SQLOrderStatus(HDBC        hdbc,
                                                HSTMT       hstmt,
#else
BOOL SQLOrderStatus(DBPROCESS   *dbproc,
#endif
#ifdef USE_CONMON
                                ORDER_STATUS_DATA
*pOrderStatus,
                                short           id,
                                short           w_id,
                                HANDLE          hConMon,
                                short           con_x,
                                short           con_y,
                                short           deadlock_retry)
#else
                                ORDER_STATUS_DATA
*pOrderStatus,
                                short           deadlock_retry)
#endif
{
    RETCODE         rc;
    int             tryit;
    int             i;
                        BOOL            not_done;
                        char            cmd_buf[255];
                        char            printbuf[25];
                        BOOL            by_name;
#ifdef USE_CONMON
                        char            linebuf[CON_LINE_SIZE+1];
#endif
#ifdef USE_ODBC
                        char            buffer[255];
                        BOOL            deadlock_detected;
#else
                        DBDATETIME      datetime;
                        BYTE            *pData;
#endif

#ifdef DEBUG
                        printf("[%ld]DBG: Entering
SQLOrderStatus()...\n", (int) GetCurrentThreadId());
#endif

                        pOrderStatus->num_deadlocks = 0;

                        if (pOrderStatus->c_id == 0)
                        {
                                by_name = TRUE;
                        }
                        else
                        {
                                by_name = FALSE;
                        }

    for (tryit=0; tryit < deadlock_retry; tryit++)
    {

#ifdef USE_ODBC
                        deadlock_detected = FALSE;
#endif


#ifdef USE_ODBC

                        strcpy(buffer,"{call
tpcc_orderstatus(?,?,?");
                        if (pOrderStatus->c_id == 0)
                        {
                                strcat(buffer,",?");
                        }
                        strcat(buffer,")}");

                        // Bind Parameters

                        rc = SQLBindParameter(hstmt, 1,
SQL_PARAM_INPUT, SQL_C_SSHORT,

                        SQL_SMALLINT, 0, 0, &pOrderStatus->w_id, 0,
NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, hdbc, hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLBindParameter() failed.");
                        }

                        rc = SQLBindParameter(hstmt, 2,
SQL_PARAM_INPUT, SQL_C_STINYINT,

                        SQL_TINYINT, 0, 0, &pOrderStatus->d_id, 0,
NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, hdbc, hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLBindParameter() failed.");
                        }

                        rc = SQLBindParameter(hstmt, 3,
SQL_PARAM_INPUT, SQL_C_SLONG,
```

```
                        SQL_INTEGER, 0, 0, &pOrderStatus->c_id, 0,
NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, hdbc, hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLBindParameter() failed.");
                        }

                        if (pOrderStatus->c_id == 0)
                        {

                                rc = SQLBindParameter(hstmt, 4,
SQL_PARAM_INPUT, SQL_C_CHAR,

                        SQL_CHAR, SQL_NTS, 0, &pOrderStatus-
>c_last, sizeof(pOrderStatus->c_last), NULL);
                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLBindParameter() failed.");
                                }
                        }

                        rc = SQLExecDirect(hstmt, buffer,
SQL_NTS);

                        if (rc != SQL_SUCCESS && rc !=
SQL_SUCCESS_WITH_INFO)
                        {
                                deadlock_detected = ODBCError
(henv, hdbc, hstmt);
                                if (!deadlock_detected)

                        UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLExecDirect() failed.");
                        }

#else
                        if (dbrpcinit(dbproc, "tpcc_orderstatus", 0)
== SUCCEED)
                        {
                                dbrpcparam(dbproc, NULL, 0,
SQLINT2, -1, -1, (BYTE *) &pOrderStatus->w_id);
                                dbrpcparam(dbproc, NULL, 0,
SQLINT1, -1, -1, (BYTE *) &pOrderStatus->d_id);
                                dbrpcparam(dbproc, NULL, 0,
SQLINT4, -1, -1, (BYTE *) &pOrderStatus->c_id);
                                if (pOrderStatus->c_id == 0)
                                {
                                        dbrpcparam(dbproc, NULL,
0, SQLCHAR, -1, strlen(pOrderStatus->c_last), pOrderStatus->c_last);
                                }
                        }
#endif

#ifdef USE_ODBC

                        not_done = TRUE;
                        i=0;

                        while (not_done && !deadlock_detected)
                        {

                                rc = SQLBindCol(hstmt, 1,
SQL_C_SSHORT, &pOrderStatus->OlOrderStatusData[i].ol_supply_w_id, 0
, NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 2,
SQL_C_SLONG, &pOrderStatus->OlOrderStatusData[i].ol_i_id, 0 , NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 3,
SQL_C_SSHORT, &pOrderStatus->OlOrderStatusData[i].ol_quantity, 0 ,
NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 4,
SQL_C_DOUBLE, &pOrderStatus->OlOrderStatusData[i].ol_amount, 0 ,
NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLBindCol() failed.");
                                }

                                rc = SQLBindCol(hstmt, 5,
SQL_C_TIMESTAMP, &pOrderStatus->OlOrderStatusData[i].ol_delivery_d,
0 , NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThre adId(),
"SQLOrderStatus", "SQLBindCol() failed.");
                                }

                                rc = SQLFetch(hstmt);

                                if (rc == SQL_ERROR)
                                {
                                        deadlock_detected =
ODBCError (henv, hdbc, hstmt);
                                        if (!deadlock_detected)

                                UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLFetch() failed.");
                                }

                                if (rc == SQL_NO_DATA_FOUND)
                                        not_done = FALSE;

                                i++;
                        }

                        pOrderStatus->o_ol_cnt = i-1;

                        if (i==0)
                        {
#ifdef USE_CONMON
                                sprintf(linebuf,"[%04ld:%04ld]
SQLOrderStatus: no orders",
                                        (int) id, (int) w_id);
                                WriteConsoleString(hConMon,
linebuf, con_x, con_y,GREEN,TRUE);
#else
                                UtilError(GetCurrentThreadId(),"SQLOrderStatus
","No orders found for customer");
#endif
                        }
                        else
                        {

                                if (!deadlock_detected)
                                {

                                        rc = SQLMoreResults(hstmt);

                                        if (rc == SQL_ERROR)
                                        {
                                                deadlock_detected =
ODBCError (henv, hdbc, hstmt);
                                                if
(!deadlock_detected)

                                UtilFatalError(GetCurrentThreadId(),
"SQLPayment", "SQLMoreResults() failed.");
                                        }
                                        else
                                        {

                                                if
(!deadlock_detected)
                                                {

                                                        rc =
SQLBindCol(hstmt, 1, SQL_C_SLONG, &pOrderStatus->c_id, 0 , NULL);
                                                        if (rc ==
SQL_ERROR)
                                                        {

                                UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLBindCol() failed.");
                                                        }
```

```
                                                rc =
SQLBindCol(hstmt, 2, SQL_C_CHAR, &pOrderStatus->c_last,
sizeof(pOrderStatus->c_last), NULL);

                                                if (rc ==
SQL_ERROR)
                                                {

                        ODBCError (henv, hdbc, hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLBindCol() failed.");
                                                }

                                                rc =
SQLBindCol(hstmt, 3, SQL_C_CHAR, &pOrderStatus->c_first,
sizeof(pOrderStatus->c_first), NULL);

                                                if (rc ==
SQL_ERROR)
                                                {

                        ODBCError (henv, hdbc, hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLBindCol() failed.");
                                                }

                                                rc =
SQLBindCol(hstmt, 4, SQL_C_CHAR, &pOrderStatus->c_middle,
sizeof(pOrderStatus->c_middle), NULL);

                                                if (rc ==
SQL_ERROR)
                                                {

                        ODBCError (henv, hdbc, hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLBindCol() failed.");
                                                }

                                                rc =
SQLBindCol(hstmt, 5, SQL_C_TIMESTAMP, &pOrderStatus->o_entry_d, 0,
NULL);

                                                if (rc ==
SQL_ERROR)
                                                {

                        ODBCError (henv, hdbc, hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLBindCol() failed.");
                                                }

                                                rc =
SQLBindCol(hstmt, 6, SQL_C_SSHORT, &pOrderStatus->o_carrier_id, 0,
NULL);

                                                if (rc ==
SQL_ERROR)
                                                {

                        ODBCError (henv, hdbc, hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLBindCol() failed.");
                                                }

                                                rc =
SQLBindCol(hstmt, 7, SQL_C_DOUBLE, &pOrderStatus->c_balance, 0,
NULL);

                                                if (rc ==
SQL_ERROR)
                                                {

                        ODBCError (henv, hdbc, hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLBindCol() failed.");
                                                }

                                                rc =
SQLBindCol(hstmt, 8, SQL_C_SLONG, &pOrderStatus->o_id, 0, NULL);

                                                if (rc ==
SQL_ERROR)
                                                {

                        ODBCError (henv, hdbc, hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLBindCol() failed.");
                                                }

                                                rc =
SQLFetch(hstmt);

                                                if (rc ==
SQL_ERROR)
                                                {

                        deadlock_detected = ODBCError (henv, hdbc,
hstmt);                                                 if
(!deadlock_detected)

                        UtilFatalError(GetCurrentThreadId(),
"SQLOrderStatus", "SQLFetch() failed.");
                                                }
                                        }
                                }
                        }

                        SQLFreeStmt(hstmt, SQL_CLOSE);
#else

                        if (dbrpcexec(dbproc) == SUCCEED)
                        {

                                while (((rc = dbresults(dbproc)) !=
NO_MORE_RESULTS) && (rc != FAIL))
                                {

                                        if (DBROWS(dbproc) &&
(dbnumcols(dbproc) == 5))
                                        {
                                                i=0;

                                                while (((rc =
dbnextrow(dbproc)) != NO_MORE_ROWS) && (rc != FAIL))
                                                {

                        if(pData=dbdata(dbproc, 1))

                        pOrderStatus-
>OlOrderStatusData[i].ol_supply_w_id = (*(DBSMALLINT *) pData);


                        if(pData=dbdata(dbproc, 2))

                        pOrderStatus->OlOrderStatusData[i].ol_i_id =
(*(DBINT *) pData);


                        if(pData=dbdata(dbproc, 3))

                        pOrderStatus->OlOrderStatusData[i].ol_quantity
= (*(DBSMALLINT *) pData);


                        if(pData=dbdata(dbproc, 4))

                        pOrderStatus->OlOrderStatusData[i].ol_amount
= (*(DBFLT8 *) pData);


                        if(pData=dbdata(dbproc, 5))
                                                        {
                                                                datetime
= *((DBDATETIME *) pData);

                        dbdatecrack(dbproc, &pOrderStatus-
>OlOrderStatusData[i].ol_delivery_d, &datetime);
                                                        }

                                                        i++;
                                                }
                                                pOrderStatus-
>o_ol_cnt = i;
                                        }
                                        else if (DBROWS(dbproc)
&& (dbnumcols(dbproc) == 8))
                                        {
                                        while (((rc =
dbnextrow(dbproc)) != NO_MORE_ROWS) && (rc != FAIL))
                                                {

                        if(pData=dbdata(dbproc, 1))

                        pOrderStatus->c_id = (*(DBINT *) pData);


                        if(pData=dbdata(dbproc, 2))

                        UtilStrCpy(pOrderStatus->c_last, pData,
dbdatlen(dbproc,2));


                        if(pData=dbdata(dbproc, 3))

                        UtilStrCpy(pOrderStatus->c_first, pData,
dbdatlen(dbproc,3));


                        if(pData=dbdata(dbproc, 4))
```

```
                        UtilStrCpy(pOrderStatus->c_middle, pData,
dbdatlen(dbproc, 4));


                        if(pData=dbdata(dbproc, 5))
                                                {
                                                        datetime
= *((DBDATETIME *) pData);
                        dbdatecrack(dbproc, &pOrderStatus->o_entry_d,
&datetime);
                                                }


                        if(pData=dbdata(dbproc, 6))

                        pOrderStatus->o_carrier_id = (*(DBSMALLINT *)
pData);


                        if(pData=dbdata(dbproc, 7))

                        pOrderStatus->c_balance = (*(DBFLT8 *)
pData);


                        if(pData=dbdata(dbproc, 8))

                        pOrderStatus->o_id = (*(DBINT *) pData);

                                        }
                                }

                                if (i==0)
                                        {
#ifdef USE_CONMON

                        sprintf(linebuf,"[%04ld:%04ld] SQLOrderStatus:
no orders",
                                                (int) id, (int)
w_id);

                        WriteConsoleString(hConMon, linebuf, con_x,
con_y,GREEN,TRUE);
#else

                        UtilError(GetCurrentThreadId(),"SQLOrderStatus
","No orders found for customer");
#endif
                                                }


                                }
                        }
#endif

#ifdef USE_ODBC
                        if (deadlock_detected)
#else
                        if (SQLDetectDeadlock(dbproc))
#endif
                        {

                                pOrderStatus->num_deadlocks++;
#ifdef USE_CONMON
                                sprintf(linebuf,"[%04ld:%04ld]
OrderStatus: deadlock:%ld",
                                                (int) id, (int) w_id, (int)
pOrderStatus->num_deadlocks);
                                WriteConsoleString(hConMon,
linebuf, con_x, con_y,RED,TRUE);
                                total_deadlocks++;
                                sprintf(linebuf,"%d",
total_deadlocks);
                                WriteConsoleString(hConMon,
linebuf, DEADLOCK_X, DEADLOCK_Y,RED,TRUE);
#else
                                sprintf(printbuf,"deadlock: retry:
%d",pOrderStatus->num_deadlocks);
                        UtilError(GetCurrentThreadId(),"SQLOrderStatus
",printbuf);
#endif

                                Sleep(DEADLOCKWAIT*tryit);

                        }
                        else
                        {
                                strcpy(pOrderStatus-
>execution_status,"Transaction commited.");
                                return TRUE;
                        }
    }

    // If we reached here, it means we quit after MAX_RETRY deadlocks
                        strcpy(pOrderStatus->execution_status,"Hit
deadlock max.  ");
#ifdef USE_CONMON
                        sprintf(linebuf,"[%04ld:%04ld] OrderStatus:
deadlock max",
                                (int) id, (int) w_id);
                        WriteConsoleString(hConMon, linebuf, con_x,
con_y,RED,TRUE);
#else
                        UtilError(GetCurrentThreadId(),"SQLOrderStatus
","deadlock max retry reached!");
#endif
    return FALSE;

}


//=============================================================
==============
//
// Function name: SQLStockLevel
//
//=============================================================
==============

#ifdef USE_ODBC
BOOL SQLStockLevel(HDBC                 hdbc,
                                        HSTMT       hstmt,
#else
BOOL SQLStockLevel(DBPROCESS    *dbproc,
#endif
#ifdef USE_CONMON
                        STOCK_LEVEL_DATA
                *pStockLevel,
                        short                           id,
                        short                           w_id,
                        HANDLE
                hConMon,
                        short                           con_x,
                        short                           con_y,
                        short
                deadlock_retry)
#else
                        STOCK_LEVEL_DATA
                *pStockLevel,
                        short
                deadlock_retry)
#endif
{
    int                 tryit;
                RETCODE             rc;
                char                printbuf[25];
#ifdef USE_CONMON
                char                linebuf[CON_LINE_SIZE+1];
#endif
#ifdef USE_ODBC
                char                buffer[255];
                BOOL                deadlock_detected;
#else
                BYTE                *pData;
#endif


#ifdef DEBUG
                printf("[%ld]DBG: Entering SQLStockLevel()...\n",
(int) GetCurrentThreadId());
#endif

                pStockLevel->num_deadlocks = 0;

    for (tryit=0; tryit < deadlock_retry; tryit++)
    {

#ifdef DEBUG
                printf("[%ld]DBG: Executing StockLevel
transaction...\n", (int) GetCurrentThreadId());
#endif

#ifdef USE_ODBC

                deadlock_detected = FALSE;

                strcpy(buffer,"{call
tpcc_stocklevel(?,?,?)}");

                // Bind Parameters
                rc = SQLBindParameter(hstmt, 1,
SQL_PARAM_INPUT, SQL_C_SSHORT,

                SQL_SMALLINT, 0, 0, &pStockLevel->w_id, 0,
NULL);
                if (rc == SQL_ERROR)
                {
                        ODBCError (henv, hdbc, hstmt);

                UtilFatalError(GetCurrentThreadId(),
"SQLStockLevel", "SQLBindParameter() failed.");
                }

                rc = SQLBindParameter(hstmt, 2,
SQL_PARAM_INPUT, SQL_C_STINYINT,

                SQL_TINYINT, 0, 0, &pStockLevel->d_id, 0,
NULL);
                if (rc == SQL_ERROR)
                {
                        ODBCError (henv, hdbc, hstmt);
```

```
                        UtilFatalError(GetCurrentThreadId(),
"SQLStockLevel", "SQLBindParameter() failed.");
                        }

                        rc = SQLBindParameter(hstmt, 3,
SQL_PARAM_INPUT, SQL_C_SSHORT,

                        SQL_SMALLINT, 0, 0, &pStockLevel-
>thresh_hold, 0, NULL);
                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, hdbc, hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLStockLevel", "SQLBindParameter() failed.");
                        }

                        rc = SQLExecDirect(hstmt, buffer,
SQL_NTS);

                        if (rc != SQL_SUCCESS && rc !=
SQL_SUCCESS_WITH_INFO)
                        {
                                deadlock_detected = ODBCError
(henv, hdbc, hstmt);
                                if (!deadlock_detected)

                        UtilFatalError(GetCurrentThreadId(),
"SQLStockLevel", "SQLExecDirect() failed.");
                        }

                        if (!deadlock_detected)
                        {

                                rc = SQLBindCol(hstmt, 1,
SQL_C_SSHORT, &pStockLevel->low_stock, 0, NULL);

                                if (rc == SQL_ERROR)
                                {
                                        ODBCError (henv, hdbc,
hstmt);

                        UtilFatalError(GetCurrentThreadId(),
"SQLStockLevel", "SQLBindCol() failed.");
                                }

                                rc = SQLFetch(hstmt);

                                if (rc == SQL_ERROR)
                                {
                                        deadlock_detected =
ODBCError (henv, hdbc, hstmt);
                                        if (!deadlock_detected)

                        UtilFatalError(GetCurrentThreadId(),
"SQLStockLevel", "SQLFetch() failed.");
                                }

                        }

                        SQLFreeStmt(hstmt, SQL_CLOSE);
#else

                        if (dbrpcinit(dbproc, "tpcc_stocklevel", 0)
== SUCCEED)
                        {
                                dbrpcparam(dbproc, NULL, 0,
SQLINT2, -1, -1, (BYTE *) &pStockLevel->w_id);
                                dbrpcparam(dbproc, NULL, 0,
SQLINT1, -1, -1, (BYTE *) &pStockLevel->d_id);
                                dbrpcparam(dbproc, NULL, 0,
SQLINT2, -1, -1, (BYTE *) &pStockLevel->thresh_hold);

                                if (dbrpcexec(dbproc) ==
SUCCEED)
                                {
                                        while (((rc =
dbresults(dbproc)) != NO_MORE_RESULTS) && (rc != FAIL))
                                        {
                                                if (DBROWS(dbproc))
                                                {
                                                        while (((rc =
dbnextrow(dbproc)) != NO_MORE_ROWS) && (rc != FAIL))
                                                        {

                                if(pData=dbdata(dbproc, 1))

                                pStockLevel->low_stock = *((long *) pData);
                                                        }
                                                }
                                        }
                                }

#endif

#ifdef USE_ODBC
                                if (deadlock_detected)
#else
                                if (SQLDetectDeadlock(dbproc))
#endif
                                {
                                        pStockLevel->num_deadlocks++;
#ifdef USE_CONMON
                                        sprintf(linebuf,"[%04ld:%04ld]
StockLevel: deadlock:%ld",
                                                (int) id, (int) w_id, (int)
pStockLevel->num_deadlocks);
                                        WriteConsoleString(hConMon,
linebuf, con_x, con_y,RED,TRUE);
                                        total_deadlocks++;
                                        sprintf(linebuf,"%d",
total_deadlocks);
                                        WriteConsoleString(hConMon,
linebuf, DEADLOCK_X, DEADLOCK_Y,RED,TRUE);
#else
                                        sprintf(printbuf,"deadlock: retry:
%d",pStockLevel->num_deadlocks);

                                UtilError(GetCurrentThreadId(),"SQLStockLevel",
printbuf);
#endif
                                        Sleep(DEADLOCKWAIT*tryit);

                                }
                                else
                                {
                                        strcpy(pStockLevel-
>execution_status, "Transaction commited.");
                                        return TRUE;
                                }
   }

   // If we reached here, it means we quit after MAX_RETRY deadlocks
                        strcpy(pStockLevel->execution_status, "Hit
deadlock max. ");
#ifdef USE_CONMON
                        sprintf(linebuf,"[%04ld:%04ld] StockLevel:
deadlock max",
                                (int) id, (int) w_id);
                        WriteConsoleString(hConMon, linebuf, con_x,
con_y,RED,TRUE);
#else
                        UtilError(GetCurrentThreadId(),"SQLStockLevel",
"deadlock max retry reached!");
#endif
   return FALSE;

}


//=====================================================
==============
//
// Function name: SQLDelivery
//
//=====================================================
==============

void SQLDelivery(DELIVERY       *pDeliveryHdlr,
                                        TRAN_STATS
                        *pDeliveryStats)
{
                        RETCODE                         rc;
                        int                             i;
                        int
                        deadlock_count;
                        BOOL                    not_done;
                        int
                        deadlock_detected;
                        struct delivery_node    get_node;
                        char                    buf[255];
#ifndef USE_ODBC
                        BYTE                    *pData;
#endif


#ifdef DEBUG
                        printf("[%ld]DBG: Entering SQLDelivery()...\n",
(int) GetCurrentThreadId());
#endif

#ifdef DEBUG
                        sprintf(buf, "[%ld] Retrieving from delivery queue:
Handler(%ld)\n",
                                        (int) GetCurrentThreadId(),
                                        (int) pDeliveryHdlr->id);
                        WriteDeliveryString(buf);
#endif

                        rc = GetDeliveryQueueNode(&get_node);

                        deadlock_count = 0;

                        if (rc==FALSE)
                        {

#ifdef DEBUG
                                sprintf(buf, "[%ld] Sleeping %ld seconds
before attempting another delivery...\n",
                                        (int) GetCurrentThreadId(),
pDeliveryHdlr->delivery_backoff);
```

```
                        WriteDeliveryString(buf);
#endif

                        UtilSleep(pDeliveryHdlr->delivery_backoff);
                        return;
                    }

                    pDeliveryHdlr->w_id = get_node.w_id;
                    pDeliveryHdlr->o_carrier_id = get_node.o_carrier_id;
                    pDeliveryHdlr->queue_time = get_node.queue_time;
                    pDeliveryHdlr->tran_start_time = get_node.tran_start_time;

#ifdef DEBUG
                    sprintf(buf, "[%ld] Starting delivery: Handler(%ld), w_id(%ld), o_carrier_id(%ld), queue_time (%d/%d/%d %d:%d:%d:%d)\n",
                        (int) GetCurrentThreadId(),
                        (int) pDeliveryHdlr->id,
                        (int) pDeliveryHdlr->w_id,
                    (int) pDeliveryHdlr->o_carrier_id,
                        pDeliveryHdlr->queue_time.wMonth,
                        pDeliveryHdlr->queue_time.wDay,
                        pDeliveryHdlr->queue_time.wYear,
                        pDeliveryHdlr->queue_time.wHour,
                        pDeliveryHdlr->queue_time.wMinute,
                        pDeliveryHdlr->queue_time.wSecond,
                        pDeliveryHdlr->queue_time.wMilliseconds);
                    WriteDeliveryString(buf);
#endif

                    not_done = TRUE;

                    // Start new delivery
                    while (not_done)
                    {

                        deadlock_detected = FALSE;

#ifdef USE_ODBC

                        rc = SQLBindParameter(pDeliveryHdlr->hstmt, 1, SQL_PARAM_INPUT, SQL_C_SSHORT,

                        if (rc == SQL_ERROR)
                        {
                            ODBCError (henv, pDeliveryHdlr->hdbc, pDeliveryHdlr->hstmt);

                        UtilFatalError(GetCurrentThreadId(), "SQLDelivery", "SQLBindParameter() failed.");
                        }

                        rc = SQLBindParameter(pDeliveryHdlr->hstmt, 2, SQL_PARAM_INPUT, SQL_C_SSHORT,

                        SQL_SMALLINT, 0, 0, &pDeliveryHdlr->o_carrier_id, 0, NULL);
                        if (rc == SQL_ERROR)
                        {
                            ODBCError (henv, pDeliveryHdlr->hdbc, pDeliveryHdlr->hstmt);

                        UtilFatalError(GetCurrentThreadId(), "SQLDelivery", "SQLBindParameter() failed.");
                        }

                        rc = SQLExecDirect(pDeliveryHdlr->hstmt, "{call tpcc_delivery (?, ?)}", SQL_NTS);

                        if (rc == SQL_ERROR)
                        {
                            deadlock_detected = ODBCError (henv, pDeliveryHdlr->hdbc, pDeliveryHdlr->hstmt);
                            if (!deadlock_detected)

                        UtilFatalError(GetCurrentThreadId(), "SQLDelivery", "SQLExecDirect() failed.");
                        }

                        if (!deadlock_detected)
                        {

                            for (i=0;i<10;i++)
                            {
                                rc = SQLBindCol(pDeliveryHdlr->hstmt, i+1, SQL_C_SLONG, &pDeliveryHdlr->DelItems[i].o_id, 0 , NULL);

                                if (rc == SQL_ERROR)
                                {
                                    ODBCError (henv, pDeliveryHdlr->hdbc, pDeliveryHdlr->hstmt);

                                UtilFatalError(GetCurrentThreadId(), "SQLDelivery", "SQLBindCol() failed.");
                                }
                            }

                            // Fetch next row
                            rc = SQLFetch(pDeliveryHdlr->hstmt);

                            if (rc == SQL_ERROR)
                            {
                                deadlock_detected = ODBCError (henv, pDeliveryHdlr->hdbc, pDeliveryHdlr->hstmt);
                                if (!deadlock_detected)

                            UtilFatalError(GetCurrentThreadId(), "SQLDelivery", "SQLFetch() failed.");
                            SQL_SMALLINT, 0, 0, &pDeliveryHdlr->w_id, 0, NULL);
                            }

                            SQLFreeStmt(pDeliveryHdlr->hstmt, SQL_CLOSE);

#else
                            if (dbrpcinit(pDeliveryHdlr->sqlconn, "tpcc_delivery", 0) == SUCCEED)
                            {
                                dbrpcparam(pDeliveryHdlr->sqlconn, NULL, 0, SQLINT2, -1, -1, (BYTE *) &pDeliveryHdlr->w_id);
                                dbrpcparam(pDeliveryHdlr->sqlconn, NULL, 0, SQLINT1, -1, -1, (BYTE *) &pDeliveryHdlr->o_carrier_id);

                                if (dbrpcexec(pDeliveryHdlr->sqlconn) == SUCCEED)
                                {
                                    while (((rc = dbresults(pDeliveryHdlr->sqlconn)) != NO_MORE_RESULTS)
                                        && (rc != FAIL))
                                    {
                                        while (((rc = dbnextrow(pDeliveryHdlr->sqlconn)) != NO_MORE_ROWS)
                                            && (rc != FAIL))
                                        {
                                            for (i=0;i<10;i++)
                                            {
                                                if(pData=dbdata(pDeliveryHdlr->sqlconn, i+1))

                                                pDeliveryHdlr->DelItems[i].o_id = *((DBINT *) pData);
                                                else

                                                UtilError(GetCurrentThreadId(),"SQLDelivery","dbdata() failed");
                                            }
                                        }
                                    }
                                }
#endif
#ifdef USE_ODBC
                                if (deadlock_detected)
#else
                                if (SQLDetectDeadlock(pDeliveryHdlr->sqlconn))
#endif
                                {
                                    deadlock_count++;
                                    pDeliveryStats->num_deadlocks++;

                                    sprintf(buf, "[%ld] Deadlock detected, retrying... (w_id(%ld), o_carrier(%ld))\n",
                                        pDeliveryHdlr->id,
                                        pDeliveryHdlr->w_id,
                                        pDeliveryHdlr->o_carrier_id);
                                    WriteDeliveryString(buf);

                                    Sleep(DEADLOCKWAIT*deadlock_count);
#ifdef USE_ODBC
                                    SQLFreeStmt(pDeliveryHdlr->hstmt, SQL_CLOSE);
#endif
                                }
                                else
                                {
                                    not_done = FALSE;
                                }

                            }

                            pDeliveryHdlr->tran_end_time = TimeNow();
                            GetLocalTime(&pDeliveryHdlr->completion_time);
```

```
#ifdef DEBUG
                        sprintf(buf, "[%ld] Deliveries completed:
Handler(%ld), w_id(%ld), o_carrier_id(%ld)\n",
                                (int) GetCurrentThreadId(),
                                (int) pDeliveryHdlr->id,
                                (int) pDeliveryHdlr->w_id,
                                (int) pDeliveryHdlr-
>o_carrier_id);
                        WriteDeliveryString(buf);
#endif DEBUG

                        sprintf(buf, "[%ld] w_id(%ld), o_carrier(%ld),
queue depth(%ld), response time(%ld ms)\n",
                                pDeliveryHdlr->id,
                                pDeliveryHdlr->w_id,
                            pDeliveryHdlr->o_carrier_id,
                                queued_delivery_cnt,
                                pDeliveryHdlr-
>tran_end_time - pDeliveryHdlr->tran_start_time);
                        WriteDeliveryString(buf);

                        StatsDelivery(pDeliveryHdlr,
                                pDeliveryStats);

}


//=====================================================
==============
//
// Function name: SQLDetectDeadlock
//
//=====================================================
==============

BOOL SQLDetectDeadlock(DBPROCESS *dbproc)
{

#ifdef DEBUG
                        printf("[%ld]DBG: Entering
SQLDetectDeadlock()...\n", (int) GetCurrentThreadId());
#endif

    if (*((BOOL *) dbgetuserdata(dbproc)) == TRUE)
    {
                        *((BOOL *) dbgetuserdata(dbproc)) =
FALSE;
                        return TRUE;
    }
    else
                        return FALSE;
}


//=====================================================
==============
//
// Function name: SQLExec
//
//=====================================================
==============

BOOL SQLExec(DBPROCESS *dbproc)
{
    int rc;

#ifdef DEBUG
                        printf("[%ld]DBG: Entering SQLExec()...\n", (int)
GetCurrentThreadId());
#endif

    if (DBDEAD(dbproc))
                        UtilFatalError(GetCurrentThreadId(),
"SQLExec","dead dbproc");
    rc = dbsqlexec(dbproc);
    while((rc = dbresults(dbproc)) != NO_MORE_RESULTS)
                        while ((rc = dbnextrow(dbproc)) !=
NO_MORE_ROWS)
                        ;

    return TRUE;
}


//=====================================================
==============
//
// Function name: SQLExecCmd
//
//=====================================================
==============

BOOL SQLExecCmd(DBPROCESS *dbproc, char *cmd)
{
    int rc;

#ifdef DEBUG
                        printf("[%ld]DBG: Entering SQLExecCmd()...\n",
(int) GetCurrentThreadId());
#endif

    if (DBDEAD(dbproc))
                        UtilFatalError(GetCurrentThreadId(),
"SQLExecCmd","dead dbproc");
    rc = dbcmd(dbproc, cmd);
    rc = dbsqlexec(dbproc);
    while((rc = dbresults(dbproc)) != NO_MORE_RESULTS)
                        while ((rc = dbnextrow(dbproc)) !=
NO_MORE_ROWS)
                        ;

    return TRUE;
}


//=====================================================
==============
//
// Function name: SQLOpenConnection
//
//=====================================================
==============

BOOL SQLOpenConnection(DBPROCESS **dbproc,
                                char    *server,
                                char    *database,
                            char    *user,
                            char    *password,
                            char    *app,
                            int     *spid,
                            long    *pack_size)
{
    LOGINREC   *login;

#ifdef DEBUG
                        printf("[%ld]DBG: Entering
SQLOpenConnection()...\n", (int) GetCurrentThreadId());
#endif

    login = dblogin();
    DBSETLUSER(login, user);
    DBSETLPWD(login,  password);
    DBSETLHOST(login, app);

    DBSETLPACKET(login, (unsigned short) pack_size);

    if ((*dbproc = dbopen(login, server )) == NULL) {
                        UtilFatalError(GetCurrentThreadId(),
"SQLOpenConnection", "Could not open connection");
                        return 0;
                    }

    // Use the the right database

    dbuse(*dbproc, database);

    dbsetuserdata(*dbproc, malloc(sizeof(BOOL)));
                        *((BOOL *) dbgetuserdata(*dbproc)) = FALSE;
    dbcmd(*dbproc, "select @@spid");
    dbsqlexec(*dbproc);

    while (dbresults(*dbproc) != NO_MORE_RESULTS)
    {
                        dbbind(*dbproc, 1, SMALLBIND, (DBINT)
0, (BYTE *) spid);
                        while (dbnextrow(*dbproc) !=
NO_MORE_ROWS)
                        ;
    }
    dbcmd(*dbproc, "set nocount on");
    dbsqlexec(*dbproc);
    while (dbresults(*dbproc) != NO_MORE_RESULTS)
    {
                        while (dbnextrow(*dbproc) !=
NO_MORE_ROWS)
                        ;
    }
#ifdef PROFILE
                    SQLExecCmd(*dbproc,"set showplan on set
statistics time on set statistics io on");
#endif
    return TRUE;

};


//=====================================================
==============
//
// Function name: SQLClientStats
//
//=====================================================
==============

int SQLClientStats(CLIENT_DATA      *pClient,
                                CLIENT_STATS   *pStats)
{

                    char        cmd[30];
                    RETCODE     rc;

#ifdef DEBUG
```

```
                        printf("[%ld]DBG: Entering SQLClientStats()...\n",
(int) GetCurrentThreadId());
#endif

#ifdef USE_ODBC

                        sprintf(cmd,"use %s", pClient->admin_database);
                        rc = SQLExecDirect(pClient->hstmt, cmd,
SQL_NTS);

    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientStats", "SQLExecDirect() failed.");
                        }

                        SQLFreeStmt(pClient->hstmt, SQL_CLOSE);

                        SQLTranStats(pClient->hdbc, pClient->hstmt,
&pStats->NewOrderStats,
                                "tpcc_neworder_stats",
"tpcc_neworder_resp_hist", pClient->disable_90th);

    SQLTranStats(pClient->hdbc, pClient->hstmt, &pStats->PaymentStats,
                                "tpcc_payment_stats",
"tpcc_payment_resp_hist", pClient->disable_90th);

    SQLTranStats(pClient->hdbc, pClient->hstmt, &pStats-
>OrderStatusStats,
                                "tpcc_orderstatus_stats",
"tpcc_orderstatus_resp_hist", pClient->disable_90th);

    SQLTranStats(pClient->hdbc, pClient->hstmt, &pStats-
>QueuedDeliveryStats,
                                "tpcc_queued_delivery_stats",
"tpcc_queued_delivery_resp_hist", pClient->disable_90th);

    SQLTranStats(pClient->hdbc, pClient->hstmt, &pStats->StockLevelStats,
                                "tpcc_stocklevel_stats",
"tpcc_stocklevel_resp_hist", pClient->disable_90th);

                        sprintf(cmd,"use %s", pClient->database);
                        rc = SQLExecDirect(pClient->hstmt, cmd,
SQL_NTS);

    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                        {
                                ODBCError (henv, pClient->hdbc, pClient-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLClientStats", "SQLExecDirect() failed.");
                        }

#else

                        sprintf(cmd,"use %s",pClient->admin_database);
                        SQLExecCmd(pClient->sqlconn, cmd);

    SQLTranStats(pClient->sqlconn, &pStats->NewOrderStats,
                                "tpcc_neworder_stats",
"tpcc_neworder_resp_hist", pClient->disable_90th);

    SQLTranStats(pClient->sqlconn, &pStats->PaymentStats,
                                "tpcc_payment_stats",
"tpcc_payment_resp_hist", pClient->disable_90th);

    SQLTranStats(pClient->sqlconn, &pStats->OrderStatusStats,
                                "tpcc_orderstatus_stats",
"tpcc_orderstatus_resp_hist", pClient->disable_90th);

    SQLTranStats(pClient->sqlconn, &pStats->QueuedDeliveryStats,
                                "tpcc_queued_delivery_stats",
"tpcc_queued_delivery_resp_hist", pClient->disable_90th);

    SQLTranStats(pClient->sqlconn, &pStats->StockLevelStats,
                                "tpcc_stocklevel_stats",
"tpcc_stocklevel_resp_hist", pClient->disable_90th);

                        sprintf(cmd,"use %s",pClient->database);
                        SQLExecCmd(pClient->sqlconn, cmd);

#endif

}


//=======================================================
==============
//
// Function name: SQLDeliveryStats
//
//=======================================================
==============

int SQLDeliveryStats(DELIVERY     *pDeliveryHdlr,
                                                TRAN_STATS   *pStats)
{

                        char    cmd[30];
                        RETCODE    rc;

#ifdef DEBUG
                        printf("[%ld]DBG: Entering
SQLDeliveryStats()...\n", (int) GetCurrentThreadId());
#endif

#ifdef USE_ODBC

                        sprintf(cmd,"use %s", pDeliveryHdlr-
>admin_database);
                        rc = SQLExecDirect(pDeliveryHdlr->hstmt, cmd,
SQL_NTS);

    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                        {
                                ODBCError (henv, pDeliveryHdlr->hdbc,
pDeliveryHdlr->hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLDeliveryStats", "SQLExecDirect() failed.");
                        }

                        SQLTranStats(pDeliveryHdlr->hdbc,
pDeliveryHdlr->hstmt, pStats,
                                "tpcc_delivery_stats",
"tpcc_delivery_resp_hist", pDeliveryHdlr->disable_90th);

                        sprintf(cmd,"use %s", pDeliveryHdlr->database);
                        rc = SQLExecDirect(pDeliveryHdlr->hstmt, cmd,
SQL_NTS);

    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                        {
                                ODBCError (henv, pDeliveryHdlr->hdbc,
pDeliveryHdlr->hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLDeliveryStats", "SQLExecDirect() failed.");
                        }

#else

                        sprintf(cmd,"use %s",pDeliveryHdlr-
>admin_database);
                        SQLExecCmd(pDeliveryHdlr->sqlconn, cmd);

    SQLTranStats(pDeliveryHdlr->sqlconn, pStats,
                                "tpcc_delivery_stats",
"tpcc_delivery_resp_hist", pDeliveryHdlr->disable_90th);

                        sprintf(cmd,"use %s",pDeliveryHdlr->database);
                        SQLExecCmd(pDeliveryHdlr->sqlconn, cmd);

#endif

}


//=======================================================
==============
//
// Function name: SQLTranStats
//
//=======================================================
==============

#ifdef USE_ODBC
void SQLTranStats(HDBC   hdbc,
                                                HSTMT  hstmt,
#else
void SQLTranStats(DBPROCESS   *dbproc,
#endif
                                                TRAN_STATS
*pTranStats,
                                                char       *StatsTable,
                                                char       *RespHistTable,
                                                long       disable_90th)
{
    int        i;
#ifdef USE_ODBC
                        RETCODE    rc;
                        char       buffer[255];
#endif

#ifdef DEBUG
                        printf("[%ld]DBG: Entering SQLTranStats()...\n",
(int) GetCurrentThreadId());
#endif

#ifdef USE_ODBC

                        sprintf(buffer,"insert into %s
values(%ld,%ld,%d,%d,%ld,%ld,%ld,%ld,%ld)",
                                                StatsTable,
                                                pTranStats->tran_count,
                                                pTranStats->total_time,
                                                pTranStats->resp_time,
                                                pTranStats->resp_min,
                                                pTranStats->resp_max,
                                                pTranStats->rolled_back,
                                                pTranStats->tran_2sec,
                                                pTranStats->tran_5sec,
                                                pTranStats-
>num_deadlocks);
```

```
                        rc = SQLExecDirect(hstmt, buffer, SQL_NTS);
    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                        {
                                ODBCError (henv, hdbc, hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLTranStats", "SQLExecDirect() failed.");
                        }

                        SQLFreeStmt(hstmt, SQL_CLOSE);

                        if (!disable_90th)
                        {
                                for(i = 0; i < HIST_MAX; i++)
                                {

                                        sprintf(buffer,"insert into %s
values(%ld, %ld)",
                                                        RespHistTable,
                                                        i,
                                                        pTranStats-
>resp_hist[i]);

                                        rc = SQLExecDirect(hstmt, buffer,
SQL_NTS);

                                        if (rc != SQL_SUCCESS && rc !=
SQL_SUCCESS_WITH_INFO)
                                        {
                                                ODBCError (henv, hdbc,
hstmt);

                                UtilFatalError(GetCurrentThreadId(),
"SQLTranStats", "SQLExecDirect() failed.");
                                        }

                                }

                                SQLFreeStmt(hstmt, SQL_CLOSE);
                        }

#else

    dbfcmd(dbproc," insert into %s values(%ld,%ld,%d,%d,",
                                        StatsTable,
                                        pTranStats->tran_count,
                                        pTranStats->total_time,
                                        pTranStats->resp_time,
                                        pTranStats->resp_min);

                        dbfcmd(dbproc,"%ld,%ld,%ld,%ld,%ld)",
                                        pTranStats->resp_max,
                                        pTranStats->rolled_back,
                                        pTranStats->tran_2sec,
                                        pTranStats->tran_5sec,
                                        pTranStats-
>num_deadlocks);

    SQLExec(dbproc);

                        if (!disable_90th)
                        {

                                // Write response histogram
                                for(i = 0; i < HIST_MAX; i++)
                                {
                                        dbfcmd(dbproc, "insert into %s
values(%ld, %ld)",
                                                        RespHistTable, i, pTranStats-
>resp_hist[i]);
                                        SQLExec(dbproc);
                                }
                        }

#endif

}


//==============================================================
//
// Function name: SQLInitResFile
//
//==============================================================

void SQLInitResFile(MASTER_DATA *pMaster,
                                        long    RunId)
{
                        typedef struct
                        {
                                char    name[25];
                                long    value;
                        } CONFIG_STRUCT;

                        char                    configure_name[25];
                        long                    configure_value;
                        int                     i;
                        int                     j;
                        int                     len;
                        char                    date[30];
    char                        version[150];
    FILE                        *fp1;
                        CONFIG_STRUCT   configure_array[100];
                        char                    cmd[250];

#ifdef DEBUG
                        printf("[%ld]DBG: Entering SQLInitResFile()...\n",
(int) GetCurrentThreadId());
#endif

    fp1 = fopen(pMaster->resfilename,"a");
    if (fp1 == NULL)
                                printf("Error in opening result file.\n");

    // Server version
    dbcmd(pMaster->sqlconn,"select convert(char(150),@@version) ");
    dbsqlexec(pMaster->sqlconn);
    while (dbresults(pMaster->sqlconn) != NO_MORE_RESULTS)
    {
                                dbbind(pMaster->sqlconn, 1,
NTBSTRINGBIND, 0, version);
                                while (dbnextrow(pMaster->sqlconn) !=
NO_MORE_ROWS)
                                ;
    }

                        // Server date/time
    dbcmd(pMaster->sqlconn,"select convert(char(30), getdate()) ");
    dbsqlexec(pMaster->sqlconn);
    while (dbresults(pMaster->sqlconn) != NO_MORE_RESULTS)
    {
                                dbbind(pMaster->sqlconn, 1,
NTBSTRINGBIND, 0, date);
                                while (dbnextrow(pMaster->sqlconn) !=
NO_MORE_ROWS)
                                ;
    }

    // Append the results to the file results.dat
    if (fp1 != NULL)
    {

                                fprintf(fp1, "\n\nTPCC BENCHMARK
TEST RUN DETAILED RESULTS\n");
                                fprintf(fp1,
"=======================================\n\n");

                                fprintf(fp1, "Test run id: %ld\n\n",  RunId);

                                if (pMaster->comment)
                                        fprintf(fp1,"Run Comment:
%s\n\n",pMaster->comment);

                                fprintf(fp1,"SQL Server Configuration
Parameters\n");
                                fprintf(fp1,"---------------------------------
\n\n");
                                fprintf(fp1, "Server time: %s\n\n", date);
                                fprintf(fp1, "%s\n",  version);

                                // Get configuration run parameters
                                dbcmd(pMaster->sqlconn,"sp_configure
");
                                dbsqlexec(pMaster->sqlconn);
                                while (dbresults(pMaster->sqlconn) !=
NO_MORE_RESULTS)
                                {
                                        dbbind(pMaster->sqlconn, 1,
NTBSTRINGBIND, 0, configure_name);
                                        dbbind(pMaster->sqlconn, 5,
INTBIND, 0, (BYTE *) &configure_value);
                                        j=0;
                                        while (dbnextrow(pMaster-
>sqlconn) != NO_MORE_ROWS)
                                        {
                                                len =
strlen(configure_name);

                                                for (i=1;i<=(25 - len);i++)

                                        strcat(configure_name, " ");

                                                fprintf(fp1,
"%s%ld\n",configure_name, configure_value);


                                        strcpy(configure_array[j].name, configure_name);
                                                configure_array[j].value =
configure_value;

                                                j++;
                                        }
                                }

                                for (i=0;i<j-1;i++)
                                {
                                        sprintf(cmd, "insert into tpcc_config
values ('%s', %ld, %ld) ",

                                configure_array[i].name, configure_array[i].value,
RunId);
```

```
                                SQLExecCmd(pMaster-
>sqlconn,cmd);
                        }

                        fclose(fp1);

    }
}


//=====================================================
==============
//
// Function name: SQLMasterStats
//
//=====================================================
==============

void SQLMasterStats(MASTER_DATA *pMaster,
                                                long        RunId)
{
                        int    i;
                        char   version[160];
    long   interval;
    long   tran_2sec;
    long   count;
    long   total_tran_cnt;
    long   neworder_tran_cnt;
    long   payment_tran_cnt;
    long   orderstatus_tran_cnt;
                        long   queued_delivery_tran_cnt;
    long   delivery_tran_cnt;
    long   stocklevel_tran_cnt;
                        long   tot_read = 0;
                        long   tot_write = 0;
                        long   total_deadlock_cnt;
    long   neworder_num_deadlocks;
    long   payment_num_deadlocks;
    long   orderstatus_num_deadlocks;
    long   queued_delivery_num_deadlocks;
    long   delivery_num_deadlocks;
    long   stocklevel_num_deadlocks;
    float  neworder_percent;
    float  payment_percent;
    float  orderstatus_percent;
    float  queued_delivery_percent;
    float  stocklevel_percent;
                        FILE   *fp1;
    char   msg[80];

#ifdef DEBUG
                        printf("[%ld]DBG: Entering
SQLMasterStats()...\n", (int) GetCurrentThreadId());
#endif

    fp1 = fopen(pMaster->resfilename,"a");
    if (fp1 == NULL)
                                printf("Error in opening result file.\n");

    count = 20000;

    // Server version
    dbcmd(pMaster->sqlconn,"select convert(char(160),@@version) ");
    dbsqlexec(pMaster->sqlconn);
    while (dbresults(pMaster->sqlconn) != NO_MORE_RESULTS)
    {
                                dbbind(pMaster->sqlconn, 1,
NTBSTRINGBIND, 0, version);
                                while (dbnextrow(pMaster->sqlconn) !=
NO_MORE_ROWS)
                                ;
    }

    // Caculate Transaction percentage mix

                                dbcmd(pMaster->sqlconn,"select
sum(tran_count), sum(num_deadlocks) "
                                        " from tpcc_neworder_stats");
    dbsqlexec(pMaster->sqlconn);
    while (dbresults(pMaster->sqlconn) != NO_MORE_RESULTS)
    {
                                dbbind(pMaster->sqlconn, 1, INTBIND, 0,
(BYTE *) &neworder_tran_cnt);
                                dbbind(pMaster->sqlconn, 2, INTBIND, 0,
(BYTE *) &neworder_num_deadlocks);
                                while (dbnextrow(pMaster->sqlconn) !=
NO_MORE_ROWS)
                                ;
    }

                                dbcmd(pMaster->sqlconn,"select
sum(tran_count), sum(num_deadlocks) "
                                        " from tpcc_payment_stats");

    dbsqlexec(pMaster->sqlconn);
    while (dbresults(pMaster->sqlconn) != NO_MORE_RESULTS)
    {
                                dbbind(pMaster->sqlconn, 1, INTBIND, 0,
(BYTE *) &payment_tran_cnt);
                                dbbind(pMaster->sqlconn, 2, INTBIND, 0,
(BYTE *) &payment_num_deadlocks);
                                while (dbnextrow(pMaster->sqlconn) !=
NO_MORE_ROWS)
                                  ;
    }

    dbcmd(pMaster->sqlconn,"select sum(tran_count), sum(num_deadlocks)
"
                                        " from tpcc_orderstatus_stats");

    dbsqlexec(pMaster->sqlconn);
    while (dbresults(pMaster->sqlconn) != NO_MORE_RESULTS)
    {
                                dbbind(pMaster->sqlconn, 1, INTBIND, 0,
(BYTE *) &orderstatus_tran_cnt);
                                dbbind(pMaster->sqlconn, 2, INTBIND, 0,
(BYTE *) &orderstatus_num_deadlocks);
                                while (dbnextrow(pMaster->sqlconn) !=
NO_MORE_ROWS)
                                ;
    }

    dbcmd(pMaster->sqlconn,"select sum(tran_count), sum(num_deadlocks)
"
                                        " from
tpcc_queued_delivery_stats");

    dbsqlexec(pMaster->sqlconn);
    while (dbresults(pMaster->sqlconn) != NO_MORE_RESULTS)
    {
                                dbbind(pMaster->sqlconn, 1, INTBIND, 0,
(BYTE *) &queued_delivery_tran_cnt);
                                dbbind(pMaster->sqlconn, 2, INTBIND, 0,
(BYTE *) &queued_delivery_num_deadlocks);
                                while (dbnextrow(pMaster->sqlconn) !=
NO_MORE_ROWS)
                                ;
    }

    dbcmd(pMaster->sqlconn,"select sum(tran_count), sum(num_deadlocks)
"
                                        " from tpcc_delivery_stats");

    dbsqlexec(pMaster->sqlconn);
    while (dbresults(pMaster->sqlconn) != NO_MORE_RESULTS)
    {
                                dbbind(pMaster->sqlconn, 1, INTBIND, 0,
(BYTE *) &delivery_tran_cnt);
                                dbbind(pMaster->sqlconn, 2, INTBIND, 0,
(BYTE *) &delivery_num_deadlocks);
                                while (dbnextrow(pMaster->sqlconn) !=
NO_MORE_ROWS)
                                ;
    }

    dbcmd(pMaster->sqlconn,"select sum(tran_count), sum(num_deadlocks)
"
                                        " from tpcc_stocklevel_stats");
    dbsqlexec(pMaster->sqlconn);
    while (dbresults(pMaster->sqlconn) != NO_MORE_RESULTS)
    {
                                dbbind(pMaster->sqlconn, 1, INTBIND, 0,
(BYTE *) &stocklevel_tran_cnt);
                                dbbind(pMaster->sqlconn, 2, INTBIND, 0,
(BYTE *) &stocklevel_num_deadlocks);
                                while (dbnextrow(pMaster->sqlconn) !=
NO_MORE_ROWS)
                                ;
    }

                        // Get total reads and writes
                        dbfcmd(pMaster->sqlconn,"select total_read,
total_write from tpcc_results"
                                        " where run_id = %ld", RunId);
    dbsqlexec(pMaster->sqlconn);
    while (dbresults(pMaster->sqlconn) != NO_MORE_RESULTS)
    {
                                dbbind(pMaster->sqlconn, 1, INTBIND, 0,
(BYTE *) &tot_read);
                                dbbind(pMaster->sqlconn, 2, INTBIND, 0,
(BYTE *) &tot_write);
                                while (dbnextrow(pMaster->sqlconn) !=
NO_MORE_ROWS)
                                ;
    }

    total_tran_cnt = neworder_tran_cnt +
                                    payment_tran_cnt        +
                                    orderstatus_tran_cnt    +
                                    queued_delivery_tran_cnt  +
                                    stocklevel_tran_cnt;

                        total_deadlock_cnt = neworder_num_deadlocks
+
                                    payment_num_deadlocks
+
                                    orderstatus_num_deadlocks
+
                                    queued_delivery_num_deadlocks
+
                                    delivery_num_deadlocks          +
                                    stocklevel_num_deadlocks;

    if (total_tran_cnt == 0)
```

```
                                total_tran_cnt = 1;

    neworder_percent    =
                                ((float) neworder_tran_cnt        / (float)
total_tran_cnt) * 100.0;
    payment_percent     =
                                ((float) payment_tran_cnt         / (float)
total_tran_cnt) * 100.0;
    orderstatus_percent =
                                ((float) orderstatus_tran_cnt     / (float)
total_tran_cnt) * 100.0;
    queued_delivery_percent    =
                                ((float) queued_delivery_tran_cnt / (float)
total_tran_cnt) * 100.0;
    stocklevel_percent  =
                                ((float) stocklevel_tran_cnt      / (float)
total_tran_cnt) * 100.0;

    // Update the result row with summary data
    dbcmd(pMaster->sqlconn,"update tpcc_results ");
    dbcmd(pMaster->sqlconn,"set date_time  = getdate(), ");
    dbfcmd(pMaster->sqlconn," version = '%s', ", version);
    dbfcmd(pMaster->sqlconn," users = %ld, ", pMaster->num_users);
    dbfcmd(pMaster->sqlconn," num_warehouses = %ld, ", pMaster-
>num_warehouses);
                        dbfcmd(pMaster->sqlconn," ramp_time = %ld, ",
pMaster->ramp_up);
                        dbfcmd(pMaster->sqlconn," run_time = %ld, ",
pMaster->steady_state);
                        dbfcmd(pMaster->sqlconn," down_time = %ld, ",
pMaster->ramp_down);
                        dbfcmd(pMaster->sqlconn," first_checkpt = %ld,
", pMaster->first_checkpoint);
                        dbfcmd(pMaster->sqlconn," checkpt_interval =
%ld, ", pMaster->checkpoint_interval);
    dbfcmd(pMaster->sqlconn," total_tran = %ld, ", total_tran_cnt);
                        dbfcmd(pMaster->sqlconn," total_deadlocks =
%ld, ", total_deadlock_cnt);
    dbfcmd(pMaster->sqlconn," neworder_per = %f, ", (double)
neworder_percent);
    dbfcmd(pMaster->sqlconn," payment_per = %f, ", (double)
payment_percent);
    dbfcmd(pMaster->sqlconn," orderstatus_per = %f, ", (double)
orderstatus_percent);
    dbfcmd(pMaster->sqlconn," queued_delivery_per = %f, ", (double)
queued_delivery_percent);
    dbfcmd(pMaster->sqlconn," stocklevel_per  = %f, ", (double)
stocklevel_percent);
                        dbfcmd(pMaster->sqlconn," comment = '%s', ",
pMaster->comment);
                        dbfcmd(pMaster->sqlconn,"
dropped_connections = %ld ", pMaster->dropped_connections);
    dbfcmd(pMaster->sqlconn," where run_id   = %ld  ", RunId);

    dbsqlexec(pMaster->sqlconn);
    while (dbresults(pMaster->sqlconn) != NO_MORE_RESULTS)
                                ;

    // Append the results to the file results.dat
    if (fp1 != NULL)
    {

                            fprintf(fp1, "\nDetailed Test Results for Steady
State\n");
                            fprintf(fp1, "-----------------------------------
\n\n");
                            fprintf(fp1, "Number of users:
%17ld\n",  pMaster->num_users);
                            fprintf(fp1, "Number of warehouses:
%17ld\n",  pMaster->num_warehouses);
                            fprintf(fp1, "Ramp up time:
%17ld\n",  pMaster->ramp_up);
                            fprintf(fp1, "Steady state time:
%17ld\n",  pMaster->steady_state);
                            fprintf(fp1, "Ramp down time:
%17ld\n",  pMaster->ramp_down);
                            fprintf(fp1, "First checkpoint:
%17ld\n",  pMaster->first_checkpoint);
                            fprintf(fp1, "Checkpoint interval:
%17ld\n",  pMaster->checkpoint_interval);
                            fprintf(fp1, "Total trans:          %17ld\n",
total_tran_cnt);
                            fprintf(fp1, "Total deadlocks:
%17ld\n",  total_deadlock_cnt);
                            fprintf(fp1, "New Order percent:
%17.2f%%\n", neworder_percent);
                            fprintf(fp1, "Payment percent:
%17.2f%%\n", payment_percent);
                            fprintf(fp1, "Order Status percent:
%17.2f%%\n", orderstatus_percent);
                            fprintf(fp1, "Delivery percent:
%17.2f%%\n", queued_delivery_percent);
                            fprintf(fp1, "Stock Level percent:
%17.2f%%\n", stocklevel_percent);
                            fprintf(fp1, "Total Reads:          %17ld\n",
tot_read);
                            fprintf(fp1, "Total Writes:         %17ld\n",
tot_write);

                            fprintf(fp1, "\nTransaction  # tran    tpm
tpm/w    tps     Avg    90%%   ");
                            fprintf(fp1, " Min    Max    RB    %%RB  > 2
sec > 5 sec DeadLk\n");
                            fprintf(fp1, "------------ ------- -------- -------- --
-------- ------ ------ ");
                            fprintf(fp1, "------ ------ ----- ----- ------- -------
-------\n");


                            printf("\nTest Results for Steady State");
                            printf("\n----------------------------\n");


                        printf("\nDetailed Test Results for Steady
State\n");
                            printf("-----------------------------------\n\n");
                            printf("Number of users:        %17ld\n",
pMaster->num_users);
                            printf("Number of warehouses:   %17ld\n",
pMaster->num_warehouses);
                            printf("Ramp up time:           %17ld\n",
pMaster->ramp_up);
                            printf("Steady state time:      %17ld\n",
pMaster->steady_state);
                            printf("Ramp down time:         %17ld\n",
pMaster->ramp_down);
                            printf("First checkpoint:       %17ld\n",
pMaster->first_checkpoint);
                            printf("Checkpoint interval:    %17ld\n",
pMaster->checkpoint_interval);
                            printf("Total trans:            %17ld\n",
total_tran_cnt);
                            printf("Total deadlocks:        %17ld\n",
total_deadlock_cnt);
                            printf("New Order percent:
%17.2f%%\n", neworder_percent);
                            printf("Payment percent:
%17.2f%%\n", payment_percent);
                            printf("Order Status percent:
%17.2f%%\n", orderstatus_percent);
                            printf("Delivery percent:       %17.2f%%\n",
queued_delivery_percent);
                            printf("Stock Level percent:
%17.2f%%\n", stocklevel_percent);
                            printf("Total Reads:            %17ld\n",
tot_read);
                            printf("Total Writes:           %17ld\n",
tot_write);

                            printf("\nTransaction  # tran    tpm     tpm/w
tps      Avg    90%%\n");
                            printf("------------ ------- -------- -------- -------
-- ------ ------\n");

                            // Read all the result rows from the 5
stat_tables
                            SQLMasterTranStats(pMaster, fp1, RunId,
"NEW ORDER   ",
                            "tpcc_neworder_stats",
"tpcc_neworder_results",
                            "tpcc_neworder_resp_hist");
                            SQLMasterTranStats(pMaster, fp1, RunId,
"PAYMENT     ",
                            "tpcc_payment_stats",
"tpcc_payment_results",
                            "tpcc_payment_resp_hist");
                            SQLMasterTranStats(pMaster, fp1, RunId,
"ORDER STATUS",
                            "tpcc_orderstatus_stats",
"tpcc_orderstatus_results",
                            "tpcc_orderstatus_resp_hist");
                            SQLMasterTranStats(pMaster, fp1, RunId,
"D DELIVERY  ",
                            "tpcc_queued_delivery_stats",
"tpcc_queued_delivery_results",
                            "tpcc_queued_delivery_resp_hist");
                            SQLMasterTranStats(pMaster, fp1, RunId,
"DELIVERY    ",
                            "tpcc_delivery_stats",
"tpcc_delivery_results",
                            "tpcc_delivery_resp_hist");
                            SQLMasterTranStats(pMaster, fp1, RunId,
"STOCK LEVEL ",
                            "tpcc_stocklevel_stats",
"tpcc_stocklevel_results",
                            "tpcc_stocklevel_resp_hist");
                        }

                        fclose(fp1);

                        sprintf(msg, "\n\nDetailed results written to the
file: %s\n\n", pMaster->resfilename);
    printf(msg);

}


//=======================================================
==============
//
// Function name: SQLMasterTranStats
//
//=======================================================
==============

void SQLMasterTranStats(MASTER_DATA *pMaster,
```

```
                                                FILE    *fp1,
                                                long
RunId,
                                                char
*TranName,
                                                char
*SelTable,
                                                char
*UpdTable,
                                                char
*RespHistTable)
{
    long   tran_count;
    long   total_time;
    long   resp_time;
    long   resp_min;
    long   resp_max;
    long   rolled_back;
    long   tran_2sec;
    long   tran_5sec;
                        long   num_deadlocks;
    long   bucket;
    long   value;
    long   per_90;
    double tps;
    double tpm;
    double avg_res;
    double ninety_percentile;
                        double rolled_back_percent;
    char   msg[80];
                        char   cvtbuf[20];
                        char   fail_flag[8];

#ifdef DEBUG
                        printf("[%ld]DBG: Entering
SQLMasterTranStats()...\n", (int) GetCurrentThreadId());
#endif

                        tran_count = 0;
    total_time = 0;
    resp_time = 0;
    resp_min = 0;
    resp_max = 0;
    rolled_back = 0;
    tran_2sec = 0;
    tran_5sec = 0;
                        num_deadlocks = 0;

    dbcmd(pMaster->sqlconn,"select sum(tran_count), sum(total_time),");
    dbcmd(pMaster->sqlconn,"min(resp_min), max(resp_max),
sum(rolled_back),");
    dbcmd(pMaster->sqlconn,"sum(tran_2sec), sum(tran_5sec),
sum(num_deadlocks) ");
    dbfcmd(pMaster->sqlconn," from %s", SelTable);

    dbsqlexec(pMaster->sqlconn);
    dbresults(pMaster->sqlconn);

    dbbind(pMaster->sqlconn, 1, INTBIND,  0, (BYTE  *) &tran_count);
    dbbind(pMaster->sqlconn, 2, INTBIND,  0, (BYTE  *) &total_time);
    dbbind(pMaster->sqlconn, 3, INTBIND,  0, (BYTE  *) &resp_min);
    dbbind(pMaster->sqlconn, 4, INTBIND,  0, (BYTE  *) &resp_max);
    dbbind(pMaster->sqlconn, 5, INTBIND,  0, (BYTE  *) &rolled_back);
    dbbind(pMaster->sqlconn, 6, INTBIND,  0, (BYTE  *) &tran_2sec);
    dbbind(pMaster->sqlconn, 7, INTBIND,  0, (BYTE  *) &tran_5sec);
                        dbbind(pMaster->sqlconn, 8, INTBIND,  0, (BYTE
*) &num_deadlocks);

    while (dbnextrow(pMaster->sqlconn) != NO_MORE_ROWS)
                                ;

    // Compute TPS and avg response time
    tps = (float) tran_count / (float) pMaster->steady_state;
    tpm = tps * 60.0;
    if (tran_count == 0)
                                avg_res = 0.0;
    else
                                avg_res = ((float) total_time / (float)
tran_count)/1000.0;

                        if (tran_count != 0)
                        {
                                rolled_back_percent =  (double)
rolled_back/tran_count;
                        }
                        else
                        {
                                rolled_back_percent = (double) 0L;
                        }

    // Read histogram of response time
    per_90 = 0;
    dbfcmd(pMaster->sqlconn, "select bucket, sum(bucket_value) from %s ",
                                RespHistTable);
    dbcmd(pMaster->sqlconn, "group by bucket");
    dbsqlexec(pMaster->sqlconn);
    dbresults(pMaster->sqlconn);

    dbbind(pMaster->sqlconn, 1, INTBIND, 0, (BYTE *) &bucket);
    dbbind(pMaster->sqlconn, 2, INTBIND, 0, (BYTE *) &value);

    while (dbnextrow(pMaster->sqlconn) != NO_MORE_ROWS)
    {
                                per_90 = per_90 + value;
                                if (per_90 >= (tran_count * .9))
                                {
                                ninety_percentile = (double)
(bucket+1)*0.1;
                                per_90 = 0;
                                }

    }

                        strcpy(fail_flag,"(Pass)");

                        if (avg_res > ninety_percentile)
                        {
                                strcpy(fail_flag,"(Fail)");
                        }
                        else
                        {

                                if ((strcmp(TranName, "NEW ORDER   ")
== 0) &&
                                        (ninety_percentile > 5))
                                {
                                        strcpy(fail_flag,"(Fail)");
                                }

                                if ((strcmp(TranName, "PAYMENT     ") ==
0) &&
                                        (ninety_percentile > 5))
                                {
                                        strcpy(fail_flag,"(Fail)");
                                }

                                if ((strcmp(TranName, "ORDER STATUS")
== 0) &&
                                        (ninety_percentile > 5))
                                {
                                        strcpy(fail_flag,"(Fail)");
                                }

                                if ((strcmp(TranName, "D DELIVERY  ")
== 0) &&
                                        (ninety_percentile > 5))
                                {
                                        strcpy(fail_flag,"(Fail)") ;
                                }

                                if ((strcmp(TranName, "DELIVERY") == 0)
&&
                                        (ninety_percentile > 80))
                                {
                                        strcpy(fail_flag,"(Fail)");
                                }

                                if ((strcmp(TranName, "STOCK LEVEL ")
== 0) &&
                                        (ninety_percentile > 20))
                                {
                                        strcpy(fail_flag,"(Fail)");
                                }
                        }

    dbfcmd(pMaster->sqlconn,"update %s set ", UpdTable);
    dbfcmd(pMaster->sqlconn,"       total_tran      = %ld, ", tran_count);
    dbfcmd(pMaster->sqlconn,"       avg_res         = %f,  ", (double) avg_res);
                                dbfcmd(pMaster->sqlconn,"       ninetyth_per    =
%f,  ", (double) ninety_percentile);
    dbfcmd(pMaster->sqlconn,"       tps             = %f,  ", (double) tps);
    dbfcmd(pMaster->sqlconn,"       tpm             = %f,  ", (double) tpm);
    dbfcmd(pMaster->sqlconn,"       min_res         = %f,  ", (double)
resp_min/1000.0);
    dbfcmd(pMaster->sqlconn,"       max_res         = %f,  ", (double)
resp_max/1000.0);
    dbfcmd(pMaster->sqlconn,"       rolled_back     = %ld, ", rolled_back);
                                dbfcmd(pMaster->sqlconn,"       rolled_back_per =
%f, ",  (double) rolled_back_percent);
    dbfcmd(pMaster->sqlconn,"       tran_2sec       = %ld, ", tran_2sec);
    dbfcmd(pMaster->sqlconn,"       tran_5sec       = %ld, ", tran_5sec);
                                dbfcmd(pMaster->sqlconn,"       num_deadlocks
= %ld  ", num_deadlocks);
    dbfcmd(pMaster->sqlconn," where run_id   = %ld  ", RunId);

    SQLExec(pMaster->sqlconn);

                                fprintf(fp1,"%s %7ld %8.2f %8.2f %10.2f %6.2f
%6.2f %6.2f %6.2f %5ld %5.2f %7ld %7ld %7ld  %s\n",
                                        TranName,
                                        tran_count,
                                        tpm,
                                        (float) tpm/pMaster-
>num_warehouses,
                                        tps,
                                        avg_res,
                                        ninety_percentile,
                                        (double) resp_min/1000.0,
                                        (double) resp_max/1000.0,
                                        rolled_back,
                                        rolled_back_percent,
                                        tran_2sec,
                                        tran_5sec,
                                        num_deadlocks,
```

```
                                        fail_flag);

                        printf("%s %7ld %8.2f %8.2f %10.2f %6.2f %6.2f
%s\n",
                                        TranName,
                                        tran_count,
                                        tpm,
                                        (float) tpm/pMaster-
>num_warehouses,
                                        tps,
                                        avg_res,
                                        ninety_percentile,
                                        fail_flag);

}


//=====================================================
==============
//
// Function name: SQLMasterIOStats
//
//=====================================================
==============

void SQLIOStats(MASTER_DATA *pMaster, int RunId, char *msg)
{
                        char            stat_name[30];
                        char                            tmpbuf[30];
                        float           value;
                        char            dbname[30];
                        float           log_size_mb;
                        float           log_used_pct;
                        int                             i;
                        FILE                            *fp1;

#ifdef DEBUG
                        printf("[%ld]DBG: Entering
SQLMasterIOStats()...\n", (int) GetCurrentThreadId());
#endif

    dbcmd(pMaster->sqlconn,"update tpcc_results ");
    dbcmd(pMaster->sqlconn,"set total_read = @@total_read -
isnull(total_read, 0), ");
    dbcmd(pMaster->sqlconn,"  total_write = @@total_write -
isnull(total_write,0) ");
    dbfcmd(pMaster->sqlconn," where run_id = %ld", RunId);
    dbsqlexec(pMaster->sqlconn);
    while (dbresults(pMaster->sqlconn) != NO_MORE_RESULTS)
                        ;

    fp1 = fopen(pMaster->resfilename,"a");
    if (fp1 == NULL)
                        printf("Error in opening result file.\n");

                        printf("%s",msg);
                        fprintf(fp1,"%s",msg);

                        dbcmd(pMaster->sqlconn, "dbcc
sqlperf(iostats)");
                        dbsqlexec(pMaster->sqlconn);
                        while (dbresults(pMaster->sqlconn) !=
NO_MORE_RESULTS)
                        {
                                if (dbnumcols(pMaster->sqlconn) == 2)
                                {
                                        dbbind(pMaster->sqlconn, 1,
NTBSTRINGBIND, 0, stat_name);
                                        dbbind(pMaster->sqlconn, 2,
FLT4BIND, 0, (BYTE *) &value);
                                        while (dbnextrow(pMaster-
>sqlconn) != NO_MORE_ROWS)
                                        {
                                                strcpy(tmpbuf,"");
                                                if (strlen(stat_name) < 30)
                                                {
                                                        for(i=0;i< (30 -
strlen(stat_name)); i++)
                                                                strcat(tmpbuf,"
");
                                                        strcat(stat_name,
tmpbuf);
                                                }

                                                printf("%s%12.4f\n",
stat_name, value);
                                                fprintf(fp1,"%s%12.4f\n",
stat_name, value);
                                        }
                                }
                        }

                        dbcmd(pMaster->sqlconn, "dbcc
sqlperf(logspace)");
                        dbsqlexec(pMaster->sqlconn);
                        while (dbresults(pMaster->sqlconn) !=
NO_MORE_RESULTS)
                        {
                                if (dbnumcols(pMaster->sqlconn) == 4)
                                {
                                        dbbind(pMaster->sqlconn, 1,
NTBSTRINGBIND, 0, dbname);
                                        dbbind(pMaster->sqlconn, 2,
FLT4BIND, 0, (BYTE *) &log_size_mb);
                                        dbbind(pMaster->sqlconn, 3,
FLT4BIND, 0, (BYTE *) &log_used_pct);
                                        while (dbnextrow(pMaster-
>sqlconn) != NO_MORE_ROWS)
                                        {
                                                if
(strcmp(dbname,"tpcc")==0)
                                                {
                                                        printf("Database tpcc
log size (MB)   %12.4f\n",log_size_mb);
                                                        printf("Database tpcc
log used (%%)    %12.4f\n\n",log_used_pct);
                                                        fprintf(fp1, "Database
tpcc log size (MB)   %12.4f\n",log_size_mb);
                                                        fprintf(fp1, "Database
tpcc log used (%%)    %12.4f\n",log_used_pct);
                                                }
                                        }
                                }

                                fclose(fp1);

}


//=====================================================
==============
//
// Function name: SQLShutdown
//
//=====================================================
==============

void SQLShutdown(MASTER_DATA *pMaster)
{
                        char            cmd[255];

#ifdef DEBUG
                        printf("[%ld]DBG: Entering SQLShutdown()...\n",
(int) GetCurrentThreadId());
#endif

                        sprintf(cmd, "use %s checkpoint use master
dump tran %s with no_log shutdown ",
                                        pMaster->database,
                                        pMaster->database);

    dbcmd(pMaster->sqlconn,cmd);
    dbsqlexec(pMaster->sqlconn);
    while (dbresults(pMaster->sqlconn) != NO_MORE_RESULTS)
                                ;

}


//=====================================================
==============
//
// Function name: SQLCheckpointStats
//
//=====================================================
==============

void SQLCheckpointStats(MASTER_DATA *pMaster, char *msg)
{
                        FILE                            *fp1;

#ifdef DEBUG
                        printf("[%ld]DBG: Entering
SQLCheckpointStats()...\n", (int) GetCurrentThreadId());
#endif

    fp1 = fopen(pMaster->resfilename,"a");
    if (fp1 == NULL)
                        printf("Error in opening result file.\n");

                        fprintf(fp1,"%s",msg);

                        fclose(fp1);

}

//=====================================================
==============
//
// Function name: SQLGetRunId
//
//=====================================================
==============

void SQLGetRunId(DBPROCESS *sqlconn,
                                        int     *pRunId)
{

#ifdef DEBUG
                        printf("[%ld]DBG: Entering SQLGetRunId()...\n",
(int) GetCurrentThreadId());
#endif
```

```
    dbcmd(sqlconn,"update tpcc_run_id set val=val + 1 ");
    dbsqlexec(sqlconn);
    dbresults(sqlconn);
    dbcmd(sqlconn,"select val from tpcc_run_id");
    dbsqlexec(sqlconn);
    dbresults(sqlconn);
    dbbind(sqlconn, 1, INTBIND, 0, (BYTE *) pRunId);
    while (dbnextrow(sqlconn) != NO_MORE_ROWS)
                                        ;

    // Insert run_id into results table
    dbfcmd(sqlconn,"insert into tpcc_results(run_id) values(%ld) ",
                                *pRunId);
    dbfcmd(sqlconn,"insert into tpcc_neworder_results(run_id) values(%ld) ",
*pRunId);
    dbfcmd(sqlconn,"insert into tpcc_payment_results(run_id) values(%ld) ",
    dbfcmd(sqlconn,"insert into tpcc_orderstatus_results(run_id) values(%ld)
", *pRunId);
    dbfcmd(sqlconn,"insert into tpcc_delivery_results(run_id) values(%ld) ",
*pRunId);
    dbfcmd(sqlconn,"insert into tpcc_queued_delivery_results(run_id)
values(%ld) ", *pRunId);
    dbfcmd(sqlconn,"insert into tpcc_stocklevel_results(run_id) values(%ld) ",
*pRunId);
    dbsqlexec(sqlconn);
    while (dbresults(sqlconn) != NO_MORE_RESULTS)
                                        ;

}


//=====================================================
==============
//
// Function name: SQLErrHandler
//
//=====================================================
==============

int SQLErrHandler(SQLCONN *dbproc,
                                                int     severity,
                                                int     err,
                                                int     oserr,
                                                char    *dberrstr,
                                        char    *oserrstr)
{
                                char msg[256];

#ifdef DEBUG
                                printf("[%ld]DBG: Entering SQLErrHandler()...\n",
(int) GetCurrentThreadId());
#endif

                                sprintf(msg, "(%ld) : %s\n", err, dberrstr);
    UtilError(GetCurrentThreadId(), "DB-Library",msg);

    if (oserr != DBNOERR)
    {
                                        sprintf(msg, "(%ld) : %s\n", oserr,
oserrstr);
                                        UtilError(GetCurrentThreadId(), "OS
Error",msg);
    }

                                if ((dbproc == NULL) || (DBDEAD(dbproc)))
                                {
//                                      ExitThread(-1);
                                }

                                return (INT_CANCEL);

}


//=====================================================
==============
//
// Function name: SQLMsgHandler
//
//=====================================================
==============

*pRunId)int SQLMsgHandler(SQLCONN   *dbproc,
                                                        DBINT     msgno,
                                                        int       msgstate,
                                                        int       severity,
                                                        char      *msgtext)
{
    char msg[256];

 #ifdef DEBUG
                                printf("[%ld]DBG: Entering
SQLClientMsgHandler()...\n", (int) GetCurrentThreadId());
                                printf("[%ld]DBG: \tmsgno = %ld\n", (int)
GetCurrentThreadId(), (int) msgno);
                                printf("[%ld]DBG: \tmsgstate = %ld\n", (int)
GetCurrentThreadId(), (int) msgstate);
                                printf("[%ld]DBG: \tseverity = %ld\n", (int)
GetCurrentThreadId(), (int) severity);
                                printf("[%ld]DBG: \t%s\n", (int)
GetCurrentThreadId(), msgtext);
 #endif

    if ( (msgno == 5701) || (msgno == 2528) || (msgno == 5703) || (msgno ==
6006) )
                                {
                                        return(INT_CONTINUE);
                                }

    // deadlock message
    if (msgno == 1205)
    {

                                // set the deadlock indicator
                                if (dbgetuserdata(dbproc) != NULL)
                                *((BOOL *) dbgetuserdata(dbproc)) =
TRUE;
                                else
                                {
                                printf("\nError, dbgetuserdata returned
NULL.\n");
                                }

                                return(INT_CONTINUE);


    }

#ifdef PROFILE
    if ( (msgno == 0) ||
                                ((msgno > STATS_MSG_LOW) &&
(msgno < STATS_MSG_HIGH)) ||
                                ((msgno > SHOWPLAN_MSG_LOW) &&
(msgno < SHOWPLAN_MSG_HIGH)))
    {
                                printf("[%ld] %s\n",  (int)
GetCurrentThreadId(), msgtext);

                                return (INT_CONTINUE);
    }
    else
    {
#endif

                                if (msgno == 0)
                                {
                                        return(INT_CONTINUE);
                                }
                                else
                                {
                                        sprintf(msg, "(%ld) : %s\n", msgno,
msgtext);
                                        UtilError(GetCurrentThreadId(),
"SQL Server Message", msg);
                                        //ExitThread(-1);
                                }

#ifdef PROFILE
    }
#endif

                        return (INT_CANCEL);
}


//=====================================================
==============
//
// Function name: SQLClientErrHandler
//
//=====================================================
==============

int SQLClientErrHandler(SQLCONN *dbproc,
                                                int     severity,
                                                int     err,
                                                int     oserr,
                                                char    *dberrstr,
                                        char    *oserrstr)
{
                        char msg[256];
                        FILE *fp1;
                        char timebuf[128];
                        char datebuf[128];

#ifdef DEBUG
                        printf("[%ld]DBG: Entering
SQLClientErrHandler()...\n", (int) GetCurrentThreadId());
#endif

                        _strtime(timebuf);
                        _strdate(datebuf);

                        sprintf(msg, "%s %s : DBLibrary (%ld) %s\n",
datebuf, timebuf, err, dberrstr);
    UtilError(GetCurrentThreadId(), "DB-Library",msg);

                        EnterCriticalSection(&ClientErrorLogCritSec);
                        fp1 = fopen("client.err","a");
                        if (fp1 == NULL)
                                printf("Error in opening errorlog file.\n");
                        fprintf(fp1, msg);
```

```
                        fclose(fp1);
                        LeaveCriticalSection(&ClientErrorLogCritSec);

    if (oserr != DBNOERR)
    {
                                sprintf(msg, "%s %s : OSError (%ld)
%s\n", datebuf, timebuf, oserr, oserrstr);
                                UtilError(GetCurrentThreadId(), "OS
Error",msg);


                        EnterCriticalSection(&ClientErrorLogCritSec);
                                fp1 = fopen("client.err","a");
                                if (fp1 == NULL)
                                        printf("Error in opening errorlog
file.\n");
                                fprintf(fp1, msg);
                                fclose(fp1);

                        LeaveCriticalSection(&ClientErrorLogCritSec);
    }


                        if ((dbproc == NULL) || (DBDEAD(dbproc)))
                        {

                        InterlockedIncrement(&client_threads_dropped);
                                //ExitThread(-1);
                        }

                        return (INT_CANCEL);

}


//=======================================================
==============
//
// Function name: SQLClientMsgHandler
//
//=======================================================
==============

int SQLClientMsgHandler(SQLCONN   *dbproc,
                                                DBINT     msgno,
                                                int       msgstate,
                                                int       severity,
                                              char      *msgtext)
{
    char msg[256];
                        FILE   *fp1;
                        char timebuf[128];
                        char datebuf[128];

 #ifdef DEBUG
                        printf("[%ld]DBG: Entering
SQLClientMsgHandler()...\n", (int) GetCurrentThreadId());
                        printf("[%ld]DBG: \tmsgno = %ld\n", (int)
GetCurrentThreadId(), (int) msgno);
                        printf("[%ld]DBG: \tmsgstate = %ld\n", (int)
GetCurrentThreadId(), (int) msgstate);
                        printf("[%ld]DBG: \tseverity = %ld\n", (int)
GetCurrentThreadId(), (int) severity);
                        printf("[%ld]DBG: \t%s\n", (int)
GetCurrentThreadId(), msgtext);
 #endif

    if ( (msgno == 5701) || (msgno == 2528) || (msgno == 5703) || (msgno ==
6006) )
                {
                        return(INT_CONTINUE);
                }

    // deadlock message
    if (msgno == 1205)
    {

                        // set the deadlock indicator
                        if (dbgetuserdata(dbproc) != NULL)
                        *((BOOL *) dbgetuserdata(dbproc)) =
TRUE;
                        else
                        {
                        printf("\nError, dbgetuserdata returned
NULL.\n");
                        }

                        return(INT_CONTINUE);

    }

#ifdef PROFILE
    if ( (msgno == 0) ||
                        ((msgno > STATS_MSG_LOW) &&
(msgno < STATS_MSG_HIGH)) ||
                        ((msgno > SHOWPLAN_MSG_LOW) &&
(msgno < SHOWPLAN_MSG_HIGH)))
    {
                        printf("[%ld] %s\n",  (int)
GetCurrentThreadId(), msgtext);

                        return (INT_CONTINUE);
    }
    else
    {
#endif

                        if (msgno == 0)
                        {
                                return(INT_CONTINUE);
                        }
                        else
                        {
                                _strtime(timebuf);
                                _strdate(datebuf);

                                sprintf(msg, "%s %s : SQLServer
(%ld) %s\n", datebuf, timebuf, msgno, msgtext);
                                UtilError(GetCurrentThreadId(),
"SQL Server Message", msg);


                        EnterCriticalSection(&ClientErrorLogCritSec);
                                fp1 = fopen("client.err","a");
                                if (fp1 == NULL)
                                        printf("Error in opening
errorlog file.\n");
                                fprintf(fp1, msg);
                                fclose(fp1);

                        LeaveCriticalSection(&ClientErrorLogCritSec);


                        InterlockedIncrement(&client_threads_dropped);
                                //ExitThread(-1);
                        }
#ifdef PROFILE
    }
#endif

                                return (INT_CANCEL);
}


//=======================================================
==============
//
// Function name: SQLDeliveryErrHandler
//
//=======================================================
==============

int SQLDeliveryErrHandler(SQLCONN *dbproc,
                                                int     severity,
                                                int     err,
                                                int     oserr,
                                                char    *dberrstr,
                                              char    *oserrstr)
{
                        char msg[256];
                        FILE   *fp1;
                        char timebuf[128];
                        char datebuf[128];

#ifdef DEBUG
                        printf("[%ld]DBG: Entering
SQLClientErrHandler()...\n", (int) GetCurrentThreadId());
#endif

                        _strtime(timebuf);
                        _strdate(datebuf);

                        sprintf(msg, "%s %s : DBLibrary (%ld) %s\n",
datebuf, timebuf, err, dberrstr);
    UtilError(GetCurrentThreadId(), "DB-Library",msg);

                        EnterCriticalSection(&ClientErrorLogCritSec);
                        fp1 = fopen("delivery.err","a");
                        if (fp1 == NULL)
                                printf("Error in opening errorlog file.\n");
                        fprintf(fp1, msg);
                        fclose(fp1);
                        LeaveCriticalSection(&ClientErrorLogCritSec);

    if (oserr != DBNOERR)
    {
                                sprintf(msg, "%s %s : OSError (%ld)
%s\n", datebuf, timebuf, oserr, oserrstr);
                                UtilError(GetCurrentThreadId(), "OS
Error",msg);


                        EnterCriticalSection(&ClientErrorLogCritSec);
                                fp1 = fopen("delivery.err","a");
                                if (fp1 == NULL)
                                        printf("Error in opening errorlog
file.\n");
                                fprintf(fp1, msg);
                                fclose(fp1);

                        LeaveCriticalSection(&ClientErrorLogC ritSec);
    }
```

```
                        if ((dbproc == NULL) || (DBDEAD(dbproc)))
                        {

                        InterlockedIncrement(&delivery_threads_droppe
d);
                                //ExitThread(-1);
                        }

                        return (INT_CANCEL);

}


//=====================================================
==============
//
// Function name: SQLDeliveryMsgHandler
//
//=====================================================
==============

int SQLDeliveryMsgHandler(SQLCONN   *dbproc,
                                                DBINT     msgno,
                                                int       msgstate,
                                                int       severity,
                                        char      *msgtext)
{
    char msg[256];
                        FILE   *fp1;
                        char timebuf[128];
                        char datebuf[128];

 #ifdef DEBUG
                        printf("[%ld]DBG: Entering
SQLClientMsgHandler()...\n", (int) GetCurrentThreadId());
                        printf("[%ld]DBG: \tmsgno = %ld\n", (int)
GetCurrentThreadId(), (int) msgno);
                        printf("[%ld]DBG: \tmsgstate = %ld\n", (int)
GetCurrentThreadId(), (int) msgstate);
                        printf("[%ld]DBG: \tseverity = %ld\n", (int)
GetCurrentThreadId(), (int) severity);
                        printf("[%ld]DBG: \t%s\n", (int)
GetCurrentThreadId(), msgtext);
 #endif

    if ( (msgno == 5701) || (msgno == 2528) || (msgno == 5703) || (msgno ==
6006) )
                        {
                                return(INT_CONTINUE);
                        }

    // deadlock message
    if (msgno == 1205)
    {

                        // set the deadlock indicator
                        if (dbgetuserdata(dbproc) != NULL)
                        *((BOOL *) dbgetuserdata(dbproc)) =
TRUE;
                        else
                        {
                        printf("\nError, dbgetuserdata returned
NULL.\n");
                        }

                        return(INT_CONTINUE);
    }

#ifdef PROFILE
    if ( (msgno == 0) ||
                                ((msgno > STATS_MSG_LOW) &&
(msgno < STATS_MSG_HIGH)) ||
                                ((msgno > SHOWPLAN_MSG_LOW) &&
(msgno < SHOWPLAN_MSG_HIGH)))
    {
                                printf("[%ld] %s\n",  (int)
GetCurrentThreadId(), msgtext);

                                return (INT_CONTINUE);
    }
    else
    {
#endif

                                if (msgno == 0)
                                {
                                        return(INT_CONTINUE);
                                }
                                else
                                {
                                        _strtime(timebuf);
                                        _strdate(datebuf);

                                        sprintf(msg, "%s %s : SQLServer
(%ld) %s\n", datebuf, timebuf, msgno, msgtext);
                                        UtilError(GetCurrentThreadId(),
"SQL Server Message", msg);

                        EnterCriticalSection(&ClientErrorLogCritSec);
                                        fp1 = fopen("delivery.err","a");
                                        if (fp1 == NULL)
                                                printf("Error in opening
errorlog file.\n");
                                        fprintf(fp1, msg);
                                        fclose(fp1);

                        LeaveCriticalSection(&ClientErrorLogCritSec);


                        InterlockedIncrement(&delivery_threads_droppe
d);
                                        //ExitThread(-1);
                                }

#ifdef PROFILE
    }
#endif

                        return (INT_CANCEL);
}


//=====================================================
==============
//
// Function name: SQLExit
//
//=====================================================
==============

void SQLExit(SQLCONN    *dbproc)
{
#ifdef DEBUG
                        printf("[%ld]DBG: Entering SQLExit()...\n", (int)
GetCurrentThreadId());
#endif
                        dbclose(dbproc);
}


//=====================================================
==============
//
// Function name: SQLInit
//
//=====================================================
==============

void SQLInit(HINSTANCE hInst)
{

#ifdef DEBUG
                        printf("[%ld]DBG: Entering SQLInit()...\n", (int)
GetCurrentThreadId());
#endif

    dbinit();
    dbmsghandle((DBMSGHANDLE_PROC)SQLMsgHandler);
    dberrhandle((DBERRHANDLE_PROC)SQLErrHandler);
}


//=====================================================
==============
//
// Function name: SQLInitPrivate
//
//=====================================================
==============

void SQLInitPrivate(PDBPROCESS dbproc, HINSTANCE hInst)
{

#ifdef DEBUG
                        printf("[%ld]DBG: Entering SQLInitPrivate()...\n",
(int) GetCurrentThreadId());
#endif

    dbprocmsghandle(dbproc, (DBMSGHANDLE_PROC)SQLMsgHandler);
    dbprocerrhandle(dbproc, (DBERRHANDLE_PROC)SQLErrHandler);
}


//=====================================================
==============
//
// Function name: SQLClientInitPrivate
//
//=====================================================
==============

void SQLClientInitPrivate(PDBPROCESS dbproc, HINSTANCE hInst)
{

#ifdef DEBUG
                        printf("[%ld]DBG: Entering SQLInitPrivate()...\n",
(int) GetCurrentThreadId());
#endif
```

```
    dbprocmsghandle(dbproc,
(DBMSGHANDLE_PROC)SQLClientMsgHandler);
    dbprocerrhandle(dbproc,
(DBERRHANDLE_PROC)SQLClientErrHandler);
}


//=====================================================
==============
//
// Function name: SQLDeliveryPrivate
//
//=====================================================
==============

void SQLDeliveryInitPrivate(PDBPROCESS dbproc, HINSTANCE hInst)
{

#ifdef DEBUG
                        printf("[%ld]DBG: Entering SQLInitPrivate()...\n",
(int) GetCurrentThreadId());
#endif

    dbprocmsghandle(dbproc,
(DBMSGHANDLE_PROC)SQLDeliveryMsgHandler);
    dbprocerrhandle(dbproc,
(DBERRHANDLE_PROC)SQLDeliveryErrHandler);
}


//=====================================================
==============
//
// Function name: SQLInitDate
//
//=====================================================
==============

#ifdef USE_ODBC
void SQLInitDate(TIMESTAMP_STRUCT *pDate)
#else
void SQLInitDate(DBDATEREC *pDate)
#endif
{

#ifdef DEBUG
                        printf("[%ld]DBG: Entering SQLInitDate()...\n",
(int) GetCurrentThreadId());
#endif

    pDate->month = 1;
    pDate->day   = 1;
    pDate->year  = 1990;
                        pDate->hour = 0;
                        pDate->minute = 0;
                        pDate->second = 0;
}



#ifdef USE_ODBC
//=====================================================
==============
//
// Function name: ODBCOpenConnection
//
//=====================================================
==============

void ODBCOpenConnection(CLIENT_DATA *Client)
{

                        RETCODE         rc;
                        char            buffer[30];

#ifdef DEBUG
                        printf("[%ld]DBG: Entering
ODBCOpenConnection()...\n", (int) GetCurrentThreadId());
#endif

                        rc = SQLAllocConnect(henv, &Client->hdbc);

                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, Client->hdbc, Client-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"ODBCOpenConnection", "SQLAllocConnect() failed.");
                        }

                        rc = SQLSetConnectOption (Client->hdbc,
SQL_PACKET_SIZE, Client->pack_size);

                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, Client->hdbc, Client-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"ODBCOpenConnection", "SQLSetConnectOption() failed.");
                        }

                        rc = SQLConnect(Client->hdbc,
                                        Client->server,
                                        SQL_NTS,
                                        Client->user,
                                        SQL_NTS,
                                        Client->password,
                                        SQL_NTS);

    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                        {
                                ODBCError (henv, Client->hdbc, Client-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"ODBCOpenConnection", "Could not open connection");
                        }

                        rc = SQLAllocStmt(Client->hdbc, &Client-
>hstmt);

                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, Client->hdbc, Client-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLOpenConnection", "SQLAllocStmt() failed.");
                        }

                        sprintf(buffer,"use %s", Client->database);

                        rc = SQLExecDirect(Client->hstmt, buffer,
SQL_NTS);

    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                        {
                                ODBCError (henv, Client->hdbc, Client-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLOpenConnection", "SQLExecDirect() failed.");
                        }

                        SQLFreeStmt(Client->hstmt, SQL_CLOSE);

                        sprintf(buffer,"set nocount on");

                        rc = SQLExecDirect(Client->hstmt, buffer,
SQL_NTS);

    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                        {
                                ODBCError (henv, Client->hdbc, Client-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLOpenConnection", "SQLExecDirect() failed.");
                        }

                        SQLFreeStmt(Client->hstmt, SQL_CLOSE);

                        sprintf(buffer,"select @@spid");

                        rc = SQLExecDirect(Client->hstmt, buffer,
SQL_NTS);

    if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                        {
                                ODBCError (henv, Client->hdbc, Client-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLOpenConnection", "SQLExecDirect() failed.");
                        }

                        rc = SQLBindCol(Client->hstmt, 1,
SQL_C_SSHORT, &Client->spid, 0, NULL);

    if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, Client->hdbc, Client-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLOpenConnection", "SQLBindCol() failed.");
                        }

                        rc = SQLFetch(Client->hstmt);

                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, Client->hdbc, Client-
>hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLOpenConnection", "SQLFetch() failed.");
                        }

                        SQLFreeStmt(Client->hstmt, SQL_CLOSE);

}


//=====================================================
==============
//
// Function name: ODBCOpenDeliveryConnection
//
```

```
//=========================================================
==============

void ODBCOpenDeliveryConnection(DELIVERY *DeliveryHdlr)
{

                        RETCODE         rc;
                        char            buffer[30];

#ifdef DEBUG
                        printf("[%ld]DBG: Entering
ODBCOpenDeliveryConnection()...\n", (int) GetCurrentThreadId());
#endif

                        rc = SQLAllocConnect(henv, &DeliveryHdlr-
>hdbc);

                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, DeliveryHdlr->hdbc,
DeliveryHdlr->hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"ODBCOpenConnection", "SQLAllocConnect() failed.");
                        }

                        rc = SQLSetConnectOption (DeliveryHdlr->hdbc,
SQL_PACKET_SIZE, DeliveryHdlr->pack_size);

                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, DeliveryHdlr->hdbc,
DeliveryHdlr->hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"ODBCOpenConnection", "SQLSetConnectOption() failed.");
                        }

                        rc = SQLConnect(DeliveryHdlr->hdbc,
                                                DeliveryHdlr->server,
                                                SQL_NTS,
                                                DeliveryHdlr->user,
                                                SQL_NTS,
                                                DeliveryHdlr-
>password,
                                                SQL_NTS);

   if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                        {
                                ODBCError (henv, DeliveryHdlr->hdbc,
DeliveryHdlr->hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"ODBCOpenConnection", "Could not open connection");
                        }

                        rc = SQLAllocStmt(DeliveryHdlr->hdbc,
&DeliveryHdlr->hstmt);

                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, DeliveryHdlr->hdbc,
DeliveryHdlr->hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLOpenConnection", "SQLAllocStmt() failed.");
                        }

                        sprintf(buffer,"use %s", DeliveryHdlr->database);

                        rc = SQLExecDirect(DeliveryHdlr->hstmt, buffer,
SQL_NTS);

   if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                        {
                                ODBCError (henv, DeliveryHdlr->hdbc,
DeliveryHdlr->hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLOpenConnection", "SQLExecDirect() failed.");
                        }

                        SQLFreeStmt(DeliveryHdlr->hstmt,
SQL_CLOSE);

                        sprintf(buffer,"set nocount on", DeliveryHdlr-
>database);

                        rc = SQLExecDirect(DeliveryHdlr->hstmt, buffer,
SQL_NTS);

   if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                        {
                                ODBCError (henv, DeliveryHdlr->hdbc,
DeliveryHdlr->hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLOpenConnection", "SQLExecDirect() failed.");
                        }

                        SQLFreeStmt(DeliveryHdlr->hstmt,
SQL_CLOSE);

                        sprintf(buffer,"select @@spid");

                        rc = SQLExecDirect(DeliveryHdlr->hstmt, buffer,
SQL_NTS);

   if (rc != SQL_SUCCESS && rc != SQL_SUCCESS_WITH_INFO)
                        {
                                ODBCError (henv, DeliveryHdlr->hdbc,
DeliveryHdlr->hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLOpenConnection", "SQLExecDirect() failed.");
                        }

                        rc = SQLBindCol(DeliveryHdlr->hstmt, 1,
SQL_C_SSHORT, &DeliveryHdlr->spid, 0, NULL);

   if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, DeliveryHdlr->hdbc,
DeliveryHdlr->hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLOpenConnection", "SQLBindCol() failed.");
                        }

                        rc = SQLFetch(DeliveryHdlr->hstmt);

                        if (rc == SQL_ERROR)
                        {
                                ODBCError (henv, DeliveryHdlr->hdbc,
DeliveryHdlr->hstmt);
                                UtilFatalError(GetCurrentThreadId(),
"SQLOpenConnection", "SQLFetch() failed.");
                        }

                        SQLFreeStmt(DeliveryHdlr->hstmt,
SQL_CLOSE);

}

//=========================================================
==============
//
// Function name: ODBCError
//
//=========================================================
==============

BOOL ODBCError (HENV henv,
                                        HDBC hdbc,
                                        HSTMT hstmt)

{
                        RETCODE     rc;
                        SDWORD      lNativeError;
                        char        szState[6];
                        char
szMsg[SQL_MAX_MESSAGE_LENGTH];
                        BOOL        deadlock_detected;
                        char        timebuf[128];
                        char        datebuf[128];
                        FILE        *fp1;
                        char        msg[255];
                        BOOL        bKillThread;

                        deadlock_detected = FALSE;
                        bKillThread = FALSE;

                        rc = SQLError(henv, hdbc, hstmt,
                                                szState,
&lNativeError,
                                                szMsg, sizeof(szMsg),
NULL);

                        while(rc == SQL_SUCCESS)
                        {

                                if (lNativeError == 1205)
                                {
                                        deadlock_detected = TRUE;
                                }
                                else
                                {
                                        _strtime(timebuf);
                                        _strdate(datebuf);

                                        sprintf(msg, "%s %s : ODBC Error:
State=%s, Error=%ld, %s\n",
                                                                datebuf,
timebuf, szState, lNativeError, szMsg);


                        EnterCriticalSection(&ClientErrorLogCritSec);
                                fp1 = fopen("client.err","a");
                                if (fp1 == NULL)
                                        printf("Error in opening
errorlog file.\n");
                                fprintf(fp1, msg);
                                fclose(fp1);

                        LeaveCriticalSection(&ClientErrorLogCritSec);

                                printf("%s", msg);

                                bKillThread = TRUE;
                        }
```

```
                        rc = SQLError(henv, hdbc, hstmt,
                                szState, &lNativeError,
                                szMsg, sizeof(szMsg),
NULL);
                }


                if (bKillThread == TRUE)
                {

                InterlockedIncrement(&client_threads_dropped);
                        //ExitThread(-1);
                }

                return deadlock_detected;

}


//=====================================================
=============
//
// Function name: ODBCExit
//
//=====================================================
=============
void ODBCExit(HDBC    hdbc,
                                HSTMT   hstmt)
{

#ifdef DEBUG
                printf("[%ld]DBG: Entering ODBCExit()...\n", (int)
GetCurrentThreadId());
#endif

                SQLFreeStmt(hstmt, SQL_DROP);
                SQLDisconnect(hdbc);
                SQLFreeConnect(hdbc);
}

#endif
```

STOCKLEVEL.C

```
/* Audited: 28 February 1997 */

/* stocklevel.c
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#include "stocklevel.h"

int stock_level_func_parse(assoc *a, int *cookie, STOCK_LEVEL_DATA
*data, char *output) {
                int i = 0;
                char errstr[128];
                char all_errors[1024];
                errstr[0] = '\0';
                all_errors[0] = '\0';
                while((*a)[0][i]) {
                        switch((*a)[0][i][0]) {
                                case 'c':
                                        *cookie = VerifyInt((*a)[1][i],
4);
                                        break;
                                case 't':
                                        data->thresh_hold =
VerifyShort((*a)[1][i], 2);
                                        break;
                                default: break;
                        }
                        ++i;
                }
                if(*cookie < 0 || !get_user(*cookie)->w_id) {
                        sprintf(errstr, BAD_COOKIE_MSG);
                        strcat(all_errors, errstr);
                }
                switch(data->thresh_hold) {
                        case -1:
                                sprintf(errstr, TOO_LONG_MSG,
"threshhold", 2);
                                strcat(all_errors, errstr);
                                break;
                        case -2:
                                sprintf(errstr, NOT_ISDIGIT_MSG,
"threshhold");
                                strcat(all_errors, errstr);
                                break;
                        case -3:
                                sprintf(errstr, NO_INPUT_MSG,
"threshhold");
                                strcat(all_errors, errstr);
                                break;
                        default:break;
                }
                data->w_id = get_user(*cookie)->w_id;
                data->d_id = get_user(*cookie)->d_id;
                if(all_errors[0]) {
                        sprintf(output, errorpage, all_errors);
                        return 0;
                } else return 1;
}

int stock_level_func_process(STOCK_LEVEL_DATA *data, int cookie) {
#ifdef DB_PRESENT
                return SQLStockLevel(get_user(cookie)-
>dbhandle, data, DEADLOCK_RETRY);
#else
                data->low_stock = 123;
                return 1;
#endif DB_PRESENT
}

void stock_level_func_format(char *output, STOCK_LEVEL_DATA *data, int
cookie) {
                char buf[3000];
                sprintf(buf, sresp, cookie);
                IntField(&buf[SW], 4, data->w_id);
                IntField(&buf[SD], 2, data->d_id);
                IntField(&buf[ST], 2, data->thresh_hold);
                IntField(&buf[SL], 3, data->low_stock);
                FormatHtmlPage(buf, output);
}

void stock_level_func_main(assoc *a, char *output) {
                int cookie;
                STOCK_LEVEL_DATA data;
                if(!stock_level_func_parse(a, &cookie, &data,
output)) return;
                if(!stock_level_func_process(&data, cookie)) {
                        sprintf(output, dberrpage, cookie);
                        return;
                }
                stock_level_func_format(output, &data, cookie);
}
```

STOCKLEVEL.H

```
/* Audited: 28 February 1997 */

/* stocklevel.h
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#ifndef __stocklevel_h__
#define __stocklevel_h__

#include "context.h"
#include <tpcc/kit/src/tpcc.h>
#include "inputparser.h"
#include "output.h"
#include "errors.h"
#include "options.h"

#define STOCKLVL_FUNC 4

static char sresp[] =
"<HTML><HEAD><TITLE>TPC-C: Stock-
Level</TITLE></HEAD><BODY><PRE>"
"                       Stock-Level\r\n"
"Warehouse: XXXX  District: XX\r\n"
"\r\n"
"Stock Level Threshold: XX\r\n"
"\r\n"
"low stock: XXX"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"\r\n"
"</PRE><P><FORM ACTION=\"tpcc.dll\" METHOD=\"GET\">"
"<INPUT TYPE=\"hidden\" NAME=\"c\" VALUE=\"%d\">"
"<INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"New Order\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Payment\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Delivery\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Order-Status\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Stock-Level\">"
" <INPUT TYPE=\"submit\" NAME=\"b\" VALUE=\"Exit\">"
"</FORM></P></BODY></HTML>\r\n";

#define SW 121
#define SD 137
#define ST 166
#define SL 183

extern void e_log(char *);
void stock_level_func_main(assoc *, char *);
int stock_level_func_parse(assoc *, int *, STOCK_LEVEL_DATA *, char *);
```

```
int stock_level_func_process(STOCK_LEVEL_DATA *, int);
void stock_level_func_format(char *, STOCK_LEVEL_DATA *, int);

#endif __stocklevel_h__
```

STUBS.C

```
/* Audited: 2 May 1996 */

/* stubs.c */
/* Copyright 1996 Intergraph Corp.  Huntsville, AL  USA */

/*
**  This file contains routines which take the place of Microsoft routines.
**  For the most part, none of the routines in this file actually do anything,
**  but are here to make the linker happy.
*/

#include <windows.h>

/* First, the "delivery" routines */
void GetDeliveryQueueNode()
{
}

void StatsDelivery()
{
}

void TimeNow()
{
}

void WriteDeliveryString()
{
}

/* Some "utility" routines */
#include <stdio.h>
static FILE *error_file;
static char *unique_filename;
static SYSTEMTIME now;

void IngrUtilInit(char *filename)
{
                        unique_filename = filename;
}

void UtilError(int thread, char *msg1, char *msg2)
{
                        if (error_file == 0) error_file =
fopen(unique_filename,"w");
                        GetLocalTime(&now);
                        if (error_file != 0)
                        {
                                fprintf(error_file,"%02d:%02d:%02d.%03d
%d/%d/%d\n",
                                        now.wHour, now.wMinute,
now.wSecond, now.wMilliseconds,
                                        now.wMonth, now.wDay,
now.wYear);
                                fprintf(error_file,"%s: %s\n",msg1,msg2);
                                fflush(error_file);
                        }
}

void UtilFatalError(int thread, char *msg1, char *msg2)
{
                        if (error_file == 0) error_file =
fopen(unique_filename,"w");
                        GetLocalTime(&now);
                        if (error_file != 0)
                        {
                                fprintf(error_file,"%02d:%02d:%02d.%03d
%d/%d/%d\n",
                                        now.wHour, now.wMinute,
now.wSecond, now.wMilliseconds,
                                        now.wMonth, now.wDay,
now.wYear);
                                fprintf(error_file,"%s: %s\n",msg1,msg2);
                        }
                        return;
}

void UtilSleep()
{
}

void UtilStatus()
{
}

void UtilStrCpy(char * destination, char * source, int length)
{
   strncpy(destination, source, length);
   destination[length] = '\0';
}


/* My own (somewhat) useful little routines. */
1LPSTR TranslateErrorCode(ULONG errorcode)
{
   DWORD LanguageId;
   LPSTR SystemMessage = 0;
   LPVOID SystemMessageArray[1] = { &SystemMessage };
   static char default_buffer[32];

   LanguageId = MAKELANGID(LANG_ENGLISH,
SUBLANG_ENGLISH_US);

   /* Translate the System Error Code into a string. */
   FormatMessage(FORMAT_MESSAGE_FROM_SYSTEM |

FORMAT_MESSAGE_ALLOCATE_BUFFER,
                        NULL,
                        errorcode,
                        LanguageId,
                        (LPSTR) &SystemMessage,
                        (unsigned long) NULL,
                        NULL);

   if (SystemMessage != 0) return SystemMessage;
   else
   {
      HINSTANCE lib;
      lib = LoadLibrary("ntdll.dll");
                        if (lib != NULL)
                        {

                        FormatMessage(FORMAT_MESSAGE_ALLOCA
TE_BUFFER |

FORMAT_MESSAGE_FROM_HMODULE,
                                lib,
                                errorcode,
                                LanguageId,
                                (LPSTR) &SystemMessage,
                                (unsigned long) NULL,
                                NULL);
                        if (SystemMessage != 0) return
SystemMessage;
                        }
   }

   sprintf(default_buffer, "error:%#x", errorcode);
   return default_buffer;
}
```

TPCC.C

```
/* Audited: 28 February 1997 */

/* tpcc.c
   Copyright (c) 1997 Intergraph Corp.
 */

#include "tpcc.h"

FILE *logfile;

void e_log(char *s) {
                        time_t timeval = time(0);
                        char ctimestr[26];
                        strcpy(ctimestr, ctime(&timeval));
                        ctimestr[24] = '\0';
                        fprintf(logfile, "%s | %s\n", ctimestr, s);
                        fflush(logfile);
}

BOOL APIENTRY DllMain(HANDLE hModule, ULONG reason, LPVOID
lpReserved) {
                        switch(reason) {
                                case DLL_PROCESS_ATTACH:
                                        logfile =
fopen("C:\\USERS\\DEFAULT\\HTTPERR.LOG", "w+");
                                        init_function_array();
                                        register_extensions();
                                        init_extensions();
                                        break;
                                case DLL_PROCESS_DETACH:
                                        cleanup_extensions();
                                        fclose(logfile);
                                        break;
                                case DLL_THREAD_ATTACH:
                                        break;
                                case DLL_THREAD_DETACH:
                                        break;
                                default: break;
                        }
                        return TRUE;
}

BOOL WINAPI GetExtensionVersion(HSE_VERSION_INFO *version) {
                        version->dwExtensionVersion =
HSE_VERSION_MAJOR << 16 | HSE_VERSION_MINOR;
                        strcpy(version->lpszExtensionDesc, "Intergraph
TPC-C Web Client");
                        return TRUE;
}
```

```
DWORD WINAPI HttpExtensionProc(LPEXTENSION_CONTROL_BLOCK
ecb) {
                        char querystring[1024];
                        assoc a;
                        char output[3000];
                        char header[256];
                        int length, hlen, function_index;
                        init_assoc(&a);
                        strcpy(querystring, ecb->lpszQueryString);
                        fill_assoc(&a, querystring);
   function_index = identify_function_index(&a);
                        if(function_array[function_index]) {
                                (*function_array[function_index])(&a,
output);
                        } else {
                                strcpy(output, enofuncent);
                        }
                        length = strlen(output);
                        sprintf(header, "Content-type:
text/html\r\nContent-length: %d\r\n\r\n", length);
                        hlen = strlen(header);
                        ecb->ServerSupportFunction(ecb->ConnID,
HSE_REQ_SEND_RESPONSE_HEADER, (LPVOID) NULL, &hlen,
(LPDWORD)header);
                        ecb->WriteClient(ecb->ConnID, output, &length,
(DWORD) NULL);
                        return
HSE_STATUS_SUCCESS_AND_KEEP_CONN;
}
```

TPCC.DEF

```
; Audited: 28 February 1997

; tpcc.def
;   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA


; TPCC.def : declares the module parameters for the DLL.

LIBRARY                 "TPCC"

EXPORTS
                        HttpExtensionProc
                        GetExtensionVersion
```

TPCC.H

```
/* Audited: 28 February 1997 */

/* tpcc.h
   Copyright (c) 1997 Intergraph Corp.  Huntsville, AL USA
 */

#ifndef __tpcc2_h__
#define __tpcc2_h__

#include <windows.h>
#include <HttpExt.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
#include "functions.h"
#include "inputparser.h"
#include "extensions.h"

static char enofuncent[] =
"<HTML><HEAD><TITLE>Function Not Found</TITLE></HEAD><BODY>"
"The URL you submitted contained an invalid query, which referenced a
nonexistent function."
"Don't do whatever it is you did.</BODY></HTML>";

#endif __tpcc2_h__
```

Appendix B: Database Design

CREATEDB.SQL

```
/*  TPC-C Benchmark Kit                     */
/*                                          */
/*  CREATEDB.SQL                            */
/*                                          */
/*  This script is used to create the database    */

use master
go

if exists ( select name from sysdatabases where name = "tpcc" )
                        drop database tpcc
go

create database tpcc

                        on
                          tpcdata1=1361,
                          tpcdata2=1361,
                          tpcdata3=1361,
                          tpcdata4=1361,
                          tpcdata5=1361,
                          tpcdata6=1361,
                          tpcdata7=1361,
                          tpcdata8=1361,
                          tpcdata9=1361,

                          tpcdata1=1361,
                          tpcdata2=1361,
                          tpcdata3=1361,
                          tpcdata4=1361,
                          tpcdata5=1361,
                          tpcdata6=1361,
                          tpcdata7=1361,
                          tpcdata8=1361,
                          tpcdata9=1361,

                          tpcdata1=1361,
                          tpcdata2=1361,
                          tpcdata3=1361,
                          tpcdata4=1361,
                          tpcdata5=1361,
                          tpcdata6=1361,
                          tpcdata7=1361,
                          tpcdata8=1361,
                          tpcdata9=1361

                          log on tpclog=3200
go
```

DISKINIT.SQL

```
/*  TPC-C Benchmark Kit                     */
/*                                          */
/*  DISKINIT.SQL                        */
/*                                          */
/*  This script is used create devices          */

use master
go

disk init name = "tpclog",
                                physname = "E:",
                                vdevno   = 14,
                                size     = 1638400
go

disk init name = "tpcdata1",
                                physname = "F:",
                                vdevno   = 15,
                                size     = 2091780
go

disk init name = "tpcdata2",
                                physname = "G:",
                                vdevno   = 16,
                                size     = 2091780
go

disk init name = "tpcdata3",
                                physname = "H:",
                                vdevno   = 17,
                                size     = 2091780
go

disk init name = "tpcdata4",
                                physname = "I:",
                                vdevno   = 18,
                                size     = 2091780
go

disk init name = "tpcdata5",
                                physname = "J:",
                                vdevno   = 19,
                                size     = 2091780
go

disk init name = "tpcdata6",
                                physname = "K:",
                                vdevno   = 20,
                                size     = 2091780
go

disk init name = "tpcdata7",
                                physname = "L:",
                                vdevno   = 21,
                                size     = 2091780
go

disk init name = "tpcdata8",
                                physname = "M:",
                                vdevno   = 22,
                                size     = 2091780
go

disk init name = "tpcdata9",
                                physname = "N:",
                                vdevno   = 23,
                                size     = 2091780
go
```

DBOPT1.SQL

```
/*  TPC-C Benchmark Kit                     */
/*                                          */
/*  DBOPT1.SQL                          */
/*                                          */
/*  Set database options for database load          */


use master
go

sp_dboption tpcc,'select into/bulkcopy',true
go

sp_dboption tpcc,'trunc. log on chkpt.',true
go

use tpcc
go

checkpoint
go

use tpcc_admin
go

sp_dboption tpcc,'trunc. log on chkpt.',true
go
```

DBOPT2.SQL

```
/*  TPC-C Benchmark Kit                     */
/*                                          */
/*  DBOPT2.SQL                          */
/*                                          */
/*  Reset database options after database load          */


use master
go

sp_dboption tpcc,'select ',false
go

sp_dboption tpcc,'trunc. ',false
go

use tpcc
go

checkpoint
go
```

DEILVERY.SQL

```
/*  File:     DELIVERY.SQL                  */
/*            Microsoft TPC-C Kit Ver. 3.00.000         */
/*            Audited 08/23/96, By Francois Raab          */
/*                                          */
/*            Copyright Microsoft, 1996             */
/*                                          */
/*  Purpose:    Delivery transaction for Microsoft TPC-C Benchmark Kit    */
```

```
/*  Author:     Damien Lindauer                         */
/*              damienl@Microsoft.com                   */


use tpcc
go

/* delivery transaction */

if exists (select name from sysobjects where name = "tpcc_delivery" )
                        drop procedure tpcc_delivery
go

create proc tpcc_delivery               @w_id           smallint,

                        @o_carrier_id   smallint
as

declare @d_id tinyint,
        @o_id int,
        @c_id int,
        @total numeric(12,2),
        @oid1 int,
        @oid2 int,
        @oid3 int,
        @oid4 int,
        @oid5 int,
        @oid6 int,
        @oid7 int,
        @oid8 int,
        @oid9 int,
        @oid10 int

select @d_id = 0

begin tran d

    while (@d_id < 10)
    begin

        select @d_id  = @d_id + 1,
               @total = 0,
               @o_id  = 0

        select @o_id = min(no_o_id)
        from new_order holdlock
        where no_w_id = @w_id and
                                        no_d_id = @d_id

        if (@@rowcount <> 0)
        begin

            /* claim the order for this district */

            delete new_order
            where no_w_id = @w_id and
                                        no_d_id = @d_id and
                                            no_o_id = @o_id

            /* set carrier_id on this order (and get customer id) */

            update orders
                                        set o_carrier_id = @o_carrier_id,
               @c_id       = o_c_id
            where o_w_id = @w_id and
                                        o_d_id = @d_id and
                                            o_id   = @o_id

            /* set date in all lineitems for this order (and sum amounts) */

            update order_line
                                        set ol_delivery_d = getdate(),
                                        @total        = @total + ol_amount
            where ol_w_id = @w_id and
                ol_d_id = @d_id and
                                            ol_o_id = @o_id

            /* accummulate lineitem amounts for this order into customer */

                                        update customer
                                            set c_balance      = c_balance +
@total,
                                                c_delivery_cnt = c_delivery_cnt
+ 1
            where c_w_id = @w_id and
                                                c_d_id = @d_id and
                                                    c_id   = @c_id

        end

        select @oid1 = case @d_id when  1  then @o_id else @oid1 end,
               @oid2 = case @d_id when  2  then @o_id else @oid2 end,
               @oid3 = case @d_id when  3  then @o_id else @oid3 end,
               @oid4 = case @d_id when  4  then @o_id else @oid4 end,
               @oid5 = case @d_id when  5  then @o_id else @oid5 end,
               @oid6 = case @d_id when  6  then @o_id else @oid6 end,
               @oid7 = case @d_id when  7  then @o_id else @oid7 end,
               @oid8 = case @d_id when  8  then @o_id else @oid8 end,
               @oid9 = case @d_id when  9  then @o_id else @oid9 end,
               @oid10 = case @d_id when 10 then @o_id else @oid10 end

    end

commit tran d

select @oid1,
       @oid2,
       @oid3,
       @oid4,
       @oid5,
       @oid6,
       @oid7,
       @oid8,
       @oid9,
       @oid10

go
```

IDXCUSCL.SQL

```
/*  TPC-C Benchmark Kit                                 */
/*                                                      */
/*  IDXCUSCL.SQL                                        */
/*                                                      */
/*  Creates clustered index on customer (noseg)         */


use tpcc
go


if exists ( select name from sysindexes where name = 'customer_c1' )
                        drop index customer.customer_c1
go

select getdate()
go
create unique clustered index customer_c1 on customer(c_w_id, c_d_id,
c_id)
                        with sorted_data
go
select getdate()
go
```

IDXCUSNC.SQL

```
/*  TPC-C Benchmark Kit                                 */
/*                                                      */
/*  IDXCUSNC.SQL                                        */
/*                                                      */
/*  Creates non-clustered index on customer (noseg)     */


use tpcc
go


if exists ( select name from sysindexes where  name = 'customer_nc1' )
   drop index customer.customer_nc1
go

select getdate()
go
create unique nonclustered index customer_nc1 on customer(c_w_id,
c_d_id, c_last, c_first, c_id)
go
select getdate()
go
```

IDXDISCL.SQL

```
/*  TPC-C Benchmark Kit                                 */
/*                                                      */
/*  IDXDISCL.SQL                                        */
/*                                                      */
/*  Creates clustered index on district (noseg)         */


use tpcc
go


if exists ( select name from sysindexes where name = 'district_c1' )
                        drop index district.district_c1
go

select getdate()
go
create unique clustered index  district_c1 on district(d_w_id, d_id)
                        with fillfactor=1
go
```

```
select getdate()
go
```

IDXITMCL.SQL

```
/*  TPC-C Benchmark Kit                                 */
/*                                                      */
/*  IDXITMCL.SQL                                        */
/*                                                      */
/*  Creates clustered index on item (noseg)             */

use tpcc
go

if exists ( select name from sysindexes where  name = 'item_c1' )
                        drop index item.item_c1
go

select getdate()
go
create unique clustered index item_c1 on item(i_id)
                        with sorted_data
go
select getdate()
go
```

IDXNODCL.SQL

```
/*  TPC-C Benchmark Kit                                 */
/*                                                      */
/*  IDXNODCL.SQL                                        */
/*                                                      */
/*  Creates clustered index on new_order (noseg)        */

use tpcc
go

if exists ( select name from sysindexes where  name = 'new_order_c1' )
                        drop index new_order.new_order_c1
go

select getdate()
go
create unique clustered index new_order_c1 on new_order(no_w_id,
no_d_id, no_o_id)
                        with sorted_data
go
select getdate()
go
```

IDXODLCL.SQL

```
/*  TPC-C Benchmark Kit                                 */
/*                                                      */
/*  IDXODLCL.SQL                                        */
/*                                                      */
/*  Creates clustered index on order_line (noseg)       */

use tpcc
go

if exists ( select name from sysindexes where  name = 'order_line_c1' )
                        drop index order_line.order_line_c1
go

select getdate()
go
create unique clustered index order_line_c1 on order_line(ol_w_id, ol_d_id,
ol_o_id, ol_number)
                        with sorted_data
go
select getdate()
go
```

IDXORDCL.SQL

```
/*  TPC-C Benchmark Kit                                 */
/*                                                      */
/*  IDXORDCL.SQL                                        */
/*                                                      */
/*  Creates clustered index on orders (noseg)           */

use tpcc
go

if exists ( select name from sysindexes where  name = 'orders_c1' )
                        drop index orders.orders_c1
go

select getdate()
go
create unique clustered index orders_c1 on orders(o_w_id, o_d_id, o_id)
                        with sorted_data
go
select getdate()
go
```

IDXSTKCL.SQL

```
/*  TPC-C Benchmark Kit                                 */
/*                                                      */
/*  IDXSTKCL.SQL                                        */
/*                                                      */
/*  Creates clustered index on stock (noseg)            */

use tpcc
go

if exists ( select name from sysindexes where  name = 'stock_c1' )
                        drop index stock.stock_c1
go

select getdate()
go
create unique clustered index stock_c1 on stock(s_i_id, s_w_id)
                        with sorted_data
go
select getdate()
go
```

IDXWARCL.SQL

```
/*  TPC-C Benchmark Kit                                 */
/*                                                      */
/*  IDXWARCL.SQL                                        */
/*                                                      */
/*  Creates clustered index on warehouse (noseg)        */

use tpcc
go

if exists ( select name from sysindexes where name = 'warehouse_c1' )
                        drop index warehouse.warehouse_c1
go

select getdate()
go
create unique clustered index warehouse_c1 on warehouse(w_id)
                        with fillfactor=1
go
select getdate()
go
```

NEWORD.SQL

```
/*  File:     NEWORD.SQL                                */
/*            Microsoft TPC-C Kit Ver. 3.00.000         */
/*            Audited 08/23/96, By Francois Raab        */
/*                                                      */
/*            Copyright Microsoft, 1996                 */
/*                                                      */
/*  Purpose:    New-Order transaction for Microsoft TPC-C Benchmark Kit
*/
/*  Author:     Damien Lindauer                         */
/*              damienl@Microsoft.com                   */

use tpcc
go

/* new-order transaction stored procedure */

if exists ( select name from sysobjects where name = "tpcc_neworder" )
                        drop procedure tpcc_neworder
```

```
go

create proc tpcc_neworder
                                                @w_id
smallint,
                                                @d_id
tinyint,
                                                @c_id
int,
                        @o_ol_cnt       tinyint,

                        @o_all_local    tinyint,
                                                @i_id1
int = 0, @s_w_id1 smallint = 0, @ol_qty1 smallint = 0,
                                                @i_id2
int = 0, @s_w_id2 smallint = 0, @ol_qty2 smallint = 0,
                                                @i_id3
int = 0, @s_w_id3 smallint = 0, @ol_qty3 smallint = 0,
                                                @i_id4
int = 0, @s_w_id4 smallint = 0, @ol_qty4 smallint = 0,
                                                @i_id5
int = 0, @s_w_id5 smallint = 0, @ol_qty5 smallint = 0,
                                                @i_id6
int = 0, @s_w_id6 smallint = 0, @ol_qty6 smallint = 0,
                                                @i_id7
int = 0, @s_w_id7 smallint = 0, @ol_qty7 smallint = 0,
                                                @i_id8
int = 0, @s_w_id8 smallint = 0, @ol_qty8 smallint = 0,
                                                @i_id9
int = 0, @s_w_id9 smallint = 0, @ol_qty9 smallint = 0,
                                                @i_id10
int = 0, @s_w_id10 smallint = 0, @ol_qty10 smallint = 0,
                                                @i_id11
int = 0, @s_w_id11 smallint = 0, @ol_qty11 smallint = 0,
                                                @i_id12
int = 0, @s_w_id12 smallint = 0, @ol_qty12 smallint = 0,
                                                @i_id13
int = 0, @s_w_id13 smallint = 0, @ol_qty13 smallint = 0,
                                                @i_id14
int = 0, @s_w_id14 smallint = 0, @ol_qty14 smallint = 0,
                                                @i_id15
int = 0, @s_w_id15 smallint = 0, @ol_qty15 smallint = 0

as
declare @w_tax          numeric(4,4),
        @d_tax          numeric(4,4),
        @c_last         char(16),
        @c_credit       char(2),
        @c_discount     numeric(4,4),
        @i_price        numeric(5,2),
        @i_name         char(24),
        @i_data         char(50),
        @o_entry_d      datetime,
        @remote_flag    int,
        @s_quantity     smallint,
        @s_data         char(50),
        @s_dist         char(24),
                                @li_no          int,
                                @o_id                   int,
                                @commit_flag    tinyint,
        @li_id          int,
        @li_s_w_id      smallint,
        @li_qty         smallint,
                                @ol_number              int,
                                @c_id_local             int

begin
                        begin transaction n

/* get order date */

select @o_entry_d = getdate()

/* get district tax and next availible order id and update */

update district
                        set @d_tax      = d_tax,
                           @o_id        = d_next_o_id,
                                d_next_o_id = d_next_o_id + 1
                           where d_w_id = @w_id and
                                d_id   = @d_id

                        /*  process orderlines */

select  @li_no = 0

                        /*  set commit flag */
                        select @commit_flag = 1

while (@li_no < @o_ol_cnt)
                        begin

   select @li_no = @li_no + 1

                                /*  Set i_id, s_w_id, and qty for this lineitem */

                                select @li_id = case @li_no
                                      when 1 then @i_id1
                                                when 2 then @i_id2
                                                when 3 then @i_id3
                                                when 4 then @i_id4
                                                when 5 then @i_id5
                                                when 6 then @i_id6
                                                when 7 then @i_id7
                                                when 8 then @i_id8
                                                when 9 then @i_id9
                                                when 10 then @i_id10
                                                when 11 then @i_id11
                                                when 12 then @i_id12
                                                when 13 then @i_id13
                                                when 14 then @i_id14
                                                when 15 then @i_id15
                                                end

                                select @li_s_w_id = case @li_no
                                                when 1 then @s_w_id1
                                                when 2 then @s_w_id2
                                                when 3 then @s_w_id3
                                                when 4 then @s_w_id4
                                                when 5 then @s_w_id5
                                                when 6 then @s_w_id6
                                                when 7 then @s_w_id7
                                                when 8 then @s_w_id8
                                                when 9 then @s_w_id9
                                                when 10 then @s_w_id10
                                                when 11 then @s_w_id11
                                                when 12 then @s_w_id12
                                                when 13 then @s_w_id13
                                                when 14 then @s_w_id14
                                                when 15 then @s_w_id15
                                                end

                                select @li_qty = case  @li_no
                                                when 1 then @ol_qty1
                                                when 2 then @ol_qty2
                                                when 3 then @ol_qty3
                                                when 4 then @ol_qty4
                                                when 5 then @ol_qty5
                                                when 6 then @ol_qty6
                                                when 7 then @ol_qty7
                                                when 8 then @ol_qty8
                                                when 9 then @ol_qty9
                                                when 10 then @ol_qty10
                                                when 11 then @ol_qty11
                                                when 12 then @ol_qty12
                                                when 13 then @ol_qty13
                                                when 14 then @ol_qty14
                                                when 15 then @ol_qty15
                                                end

        /* get item data (no one updates item) */

        select @i_price = i_price,
              @i_name  = i_name,
              @i_data  = i_data
        from item (tablock holdlock)
                                where i_id = @li_id

                                /* if there actually is an item with this id, go to
work */

                                        if (@@rowcount > 0)
                                        begin
                update stock set s_ytd          = s_ytd + @li_qty,
                                @s_quantity  = s_quantity,
                                s_quantity   = s_quantity - @li_qty +
                                   case when (s_quantity - @li_qty < 10) then 91 else 0
end,
                                s_order_cnt  = s_order_cnt + 1,
                                s_remote_cnt = s_remote_cnt + case
                                   when (@li_s_w_id = @w_id) then 0 else 1 end,
                                @s_data      = s_data,
                                @s_dist      = case @d_id
                                                                        when 1  then
s_dist_01
                                                                        when 2  then
s_dist_02
                                                                        when 3  then
s_dist_03
                                                                        when 4  then
s_dist_04
                                                                        when 5  then
s_dist_05
                                                                        when 6  then
s_dist_06
                                                                        when 7  then
s_dist_07
                                                                        when 8  then
s_dist_08
                                                                        when 9  then
s_dist_09
                                                                        when 10 then
s_dist_10
                                                                end
                        where s_i_id = @li_id and
                                                                s_w_id = @li_s_w_id

                                                        /* insert order_line data (using data
from item and stock) */

                insert into order_line values(@o_id,            /* from district update */
                                        @d_id,                  /* input param          */
```

```
                        @w_id,          /* input param      */
                        @li_no,         /* orderline number */
                        @li_id,         /* lineitem id      */
                        @li_s_w_id,     /* lineitem warehouse */
                        "jan 1, 1900",  /* constant         */
                        @li_qty,        /* lineitem qty     */
                        @i_price * @li_qty, /* ol_amount    */
                        @s_dist)        /* from stock       */

        /* send line-item data to client */

                                select @i_name,
            @s_quantity,
            b_g = case when ( (patindex("%ORIGINAL%",@i_data) > 0) and

(patindex("%ORIGINAL%",@s_data) > 0) )
                    then "B" else "G" end,
            @i_price,
            @i_price * @li_qty

      end
      else
                                begin

                                  /*  no item found - triggers rollback
condition */

                                    select "",0,"",0,0
                                    select @commit_flag = 0

                                end
                            end

    /* get customer last name, discount, and credit rating */

    select @c_last     = c_last,
           @c_discount = c_discount,
           @c_credit   = c_credit,
                                @c_id_local = c_id
    from customer holdlock
    where c_id   = @c_id and
          c_w_id = @w_id and
          c_d_id = @d_id

    /* insert fresh row into orders table */

    insert into orders values (@o_id,
                                    @d_id,
                                   @w_id,
                                        @c_id_local,
                                        @o_entry_d,
                                           0,
                                           @o_ol_cnt,
                                           @o_all_local)

                            /* insert corresponding row into new-order table
*/

                            insert into new_order values (@o_id,

@d_id,

@w_id)

                            /*  select warehouse tax */

                            select @w_tax = w_tax
                            from warehouse holdlock
                            where w_id = @w_id

                            if (@commit_flag = 1)
                                    commit transaction n
                            else
                                    /* all that work for nuthin!!! */
                                    rollback transaction n

                            /* return order data to client */
                            select @w_tax,
                                   @d_tax,
                                     @o_id,
                                     @c_last,
                                     @c_discount,
                                     @c_credit,
                                     @o_entry_d,
                                     @commit_flag

end

go
```

ORDSTAT.SQL

```
/*  File:      ORDSTAT.SQL                                  */
/*             Microsoft TPC-C Kit Ver. 3.00.000            */
/*             Audited 08/23/96, By Francois Raab           */
/*                                                          */
/*             Copyright Microsoft, 1996                    */
/*                                                          */
/*  Purpose:    Order-Status transaction for Microsoft TPC-C Benchmark Kit
*/
/*  Author:     Damien Lindauer                              */
/*             damienl@Microsoft.com                        */


use tpcc
go

if exists ( select name from sysobjects where name = "tpcc_orderstatus" )
                            drop procedure   tpcc_orderstatus
go

create proc tpcc_orderstatus @w_id               smallint,
                                                    @d_id               tinyint,
                                                    @c_id               int,
                                                    @c_last      char(16) = ""

as

declare @c_balance              numeric(12,2),
        @c_first                char(16),
        @c_middle               char(2),
        @o_id                       int,
        @o_entry_d              datetime,
        @o_carrier_id           smallint,
        @val                        smallint,
        @cnt                        smallint

begin tran o

                            if (@c_id = 0)
                                    begin
                                    /* get customer id and info using last name
                                    */

                                    select @cnt = count(*)
                                    from customer holdlock
                                    where c_last = @c_last and
                                      c_w_id = @w_id and
                                c_d_id = @d_id

                                    select @val = (@cnt + 1) / 2
                                    set rowcount @val

                                    select @c_id = c_id,
                                           @c_balance = c_balance,
                                           @c_first   = c_first,
                                           @c_last    = c_last,
                                           @c_middle  = c_middle
                                    from customer holdlock
                                    where c_last = @c_last and
                                      c_w_id = @w_id and
                                c_d_id = @d_id
                                    order by c_w_id, c_d_id, c_last, c_first

                                    set rowcount 0
                                    end

                            else
                                    begin

                                    /*  get customer info if by id*/

                                    select @c_balance = c_balance,
                                         @c_first   = c_first,
                                         @c_middle  = c_middle,
                                         @c_last    = c_last
                                    from customer holdlock
                                    where c_id   = @c_id and
                                        c_d_id = @d_id and
                                        c_w_id = @w_id

                                    select @cnt = @@rowcount

                                    end

                            /* if no such customer */
                            if (@cnt = 0)
                            begin
                                    raiserror("Customer not found",18,1)
                                    goto custnotfound
                            end

                            /*  get order info */

                            select @o_id = o_id,
                                   @o_entry_d    = o_entry_d,
                              @o_carrier_id = o_carrier_id
                            from orders holdlock
                            where o_c_id = @c_id and
                          o_d_id = @d_id and
                          o_w_id = @w_id

                            /*  select order lines for the current order */

                            select ol_supply_w_id,
                              ol_i_id,
                                 ol_quantity,
                                 ol_amount,
                                 ol_delivery_d
```

```
                        from order_line holdlock
                        where ol_o_id = @o_id and
                            ol_d_id = @d_id and
                            ol_w_id = @w_id

custnotfound:

commit tran o

/*  return data to client  */

select @c_id,
                            @c_last,
                            @c_first,
                            @c_middle,
                            @o_entry_d,
                            @o_carrier_id,
                            @c_balance,
                            @o_id

go
```

PAYMENT.SQL

```
/*  File:       PAYMENT.SQL                                     */
/*              Microsoft TPC-C Kit Ver. 3.00.000               */
/*              Audited 08/23/96, By Francois Raab              */
/*                                                              */
/*              Copyright Microsoft, 1996                       */
/*                                                              */
/*  Purpose:    Payment transaction for Microsoft TPC-C Benchmark Kit      */
/*  Author:     Damien Lindauer                                 */
/*              damienl@Microsoft.com                           */


use tpcc
go

if exists (select name from sysobjects where name = "tpcc_payment" )
                        drop procedure tpcc_payment
go

create proc tpcc_payment @w_id          smallint,
                                                        @c_w_id
smallint,
                                                        @h_amount
numeric(6,2),
                                                        @d_id
tinyint,
                                                        @c_d_id
tinyint,
                                                        @c_id
int,
                                                        @c_last
char(16) = ""


as
declare  @w_street_1     char(20),
        @w_street_2     char(20),
        @w_city         char(20),
        @w_state        char(2),
        @w_zip          char(9),
        @w_name         char(10),
        @d_street_1     char(20),
        @d_street_2     char(20),
        @d_city         char(20),
        @d_state        char(2),
        @d_zip          char(9),
        @d_name         char(10),
        @c_first        char(16),
        @c_middle       char(2),
        @c_street_1     char(20),
        @c_street_2     char(20),
        @c_city         char(20),
        @c_state        char(2),
        @c_zip          char(9),
        @c_phone        char(16),
        @c_since        datetime,
        @c_credit       char(2),
        @c_credit_lim   numeric(12,2),
        @c_balance      numeric(12,2),
        @c_discount     numeric(4,4),
        @data1          char(250),
        @data2          char(250),
        @c_data_1       char(250),
        @c_data_2       char(250),
        @datetime       datetime,
        @w_ytd          numeric(12,2),
        @d_ytd          numeric(12,2),
        @cnt            smallint,
        @val            smallint,
        @screen_data            char(200),
                                @d_id_local  tinyint,
                                @w_id_local             smallint,
                                @c_id_local  int

select @screen_data = ""

begin tran p

                        /* get payment date */

                        select @datetime = getdate()

                        if (@c_id = 0)
                        begin
                                /* get customer id and info using last name
*/

                                select @cnt = count(*)
                                from customer holdlock
                                where c_last = @c_last and
                                    c_w_id = @c_w_id and
                                    c_d_id = @c_d_id

                                select @val = (@cnt + 1) / 2
                                set rowcount @val

                                select @c_id = c_id
                                from customer holdlock
                                where c_last = @c_last and
                                    c_w_id = @c_w_id and
                                    c_d_id = @c_d_id
                                order by c_w_id, c_d_id, c_last, c_first

                                set rowcount 0
                        end

                        /* get customer info and update balances */

                        update customer set
                                c_balance     = c_balance = c_balance -
@h_amount,
                                c_payment_cnt  = c_payment_cnt + 1,
                                c_ytd_payment  = c_ytd_payment +
@h_amount,
                                @c_first       = c_first,
                                @c_middle      = c_middle,
                            @c_last        = c_last,
                            @c_street_1    = c_street_1,
                                @c_street_2    = c_street_2,
                                @c_city        = c_city,
                                @c_state       = c_state,
                                @c_zip         = c_zip,
                                @c_phone       = c_phone,
                                @c_credit      = c_credit,
                                @c_credit_lim  = c_credit_lim,
                                @c_discount    = c_discount,
                                @c_since       = c_since,
                                @data1         = c_data_1,
                                @data2         = c_data_2,
                                @c_id_local    = c_id
                        where c_id   = @c_id and
                             c_w_id = @c_w_id and
                             c_d_id = @c_d_id

                        /* if customer has bad credit get some more info
*/

                        if (@c_credit = "BC")
                        begin

                                /* compute new info */

                                select @c_data_2 =
substring(@data1,209,42) +

substring(@data2, 1, 208)
                                select @c_data_1 =
convert(char(5),@c_id) +

convert(char(4),@c_d_id) +

convert(char(5),@c_w_id) +

convert(char(4),@d_id) +

convert(char(5),@w_id) +

convert(char(19),@h_amount) +
                                                        substring(@data1, 1,
208)

                                /* update customer info */

                                update customer set
                                      c_data_1 = @c_data_1,
                                           c_data_2 = @c_data_2
                                where c_id   = @c_id and
                                          c_w_id = @c_w_id and
                                     c_d_id = @c_d_id

                                select @screen_data = substring
(@c_data_1,1,200)
                        end


                        /* get district data and update year-to-date */
```

```
update district
        set d_ytd       = d_ytd + @h_amount,
                @d_street_1 = d_street_1,
                @d_street_2 = d_street_2,
                @d_city     = d_city,
                @d_state    = d_state,
                @d_zip      = d_zip,
                @d_name     = d_name,
                @d_id_local = d_id
        where d_w_id = @w_id and
              d_id   = @d_id

/* get warehouse data and update year-to-date */

update warehouse
set w_ytd       = w_ytd + @h_amount,
        @w_street_1 = w_street_1,
    @w_street_2 = w_street_2,
    @w_city     = w_city,
    @w_state    = w_state,
    @w_zip      = w_zip,
    @w_name     = w_name,
        @w_id_local = w_id
where w_id = @w_id

/* create history record */

insert into history values  (@c_id_local,

@c_d_id,

@c_w_id,

@d_id_local,

@w_id_local,

@datetime,

@h_amount,

@w_name + "    " + @d_name)

commit tran p

/* return data to client */

select  @c_id,
                @c_last,
                @datetime,
                @w_street_1,
                @w_street_2,
                @w_city,
                @w_state,
                @w_zip,
                @d_street_1,
                @d_street_2,
                @d_city,
                @d_state,
                @d_zip,
                @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,
                @screen_data

go
```

PINTABLE.SQL

```
/*  TPC-C Benchmark Kit                                         */
/*                                                              */
/*  PINTABLE.SQL                                                */
/*                                                              */
/*  This script file is used to 'pin' certain tables in the data cache      */

use tpcc
go

exec sp_tableoption "district","pintable",true
exec sp_tableoption "warehouse","pintable",true
exec sp_tableoption "new_order","pintable",true
exec sp_tableoption "item","pintable",true
go
```

RUNCFG.SQL

```
/*  TPC-C Benchmark Kit                                         */
/*                                                              */
/*  RUNCFG.SQL                                                  */
/*                                                              */
/*  This script file is used to set server configuration parameters for test runs
*/


exec sp_configure "show advanced option", 1
go

reconfigure with override
go

exec sp_configure "affinity mask",0
exec sp_configure "hash buckets",265003
exec sp_configure "logwrite sleep (ms)",-1
exec sp_configure "max async IO",64
exec sp_configure "max lazywrite IO",32
exec sp_configure "max worker threads",100
exec sp_configure "memory",30000
exec sp_configure "free buffers",2000
exec sp_configure "priority boost",0
exec sp_configure "procedure cache",2
exec sp_configure "RA worker threads",0
exec sp_configure "recovery interval",32767
exec sp_configure "set working set size",0
exec sp_configure "SMP concurrency",-1
exec sp_configure "spin counter",10000
exec sp_configure "tempdb in ram (MB)",5
exec sp_configure "user connections",150
go

reconfigure with override
go

shutdown
go
```

SHUTDOWN.SQL

```
/*  TPC-C Benchmark Kit                                         */
/*                                                              */
/*  SHUTDOWN.SQL                                                */
/*                                                              */
/*  This script file is used to shutdown the server gracefully              */


use tpcc
go

checkpoint
go

use tpcc_admin
go

checkpoint
go

dump tran tpcc with no_log
go

dump tran tpcc_admin with no_log
go

shutdown
go
```

STOCKLEV.SQL

```
/*  File:       STOCKLEV.SQL                                    */
/*              Microsoft TPC-C Kit Ver. 3.00.000               */
/*              Audited 08/23/96, By Francois Raab              */
/*                                                              */
/*              Copyright Microsoft, 1996                       */
/*                                                              */
/*  Purpose:    Stock-Level transaction for Microsoft TPC-C Benchmark Kit  */
/*  Author:     Damien Lindauer                                 */
/*              damienl@Microsoft.com                           */


use tpcc
go

/* stock-level transaction stored procedure */

if exists (select name from sysobjects where name = "tpcc_stocklevel" )
                        drop procedure tpcc_stocklevel
go

create proc tpcc_stocklevel         @w_id         smallint,
                                                  @d_id
                        tinyint,
                                                  @threshhold         smallint
as
declare @o_id_low int,
```

```
    @o_id_high int

  select @o_id_low  = (d_next_o_id - 20),
                      @o_id_high = (d_next_o_id - 1)
  from district
  where d_w_id = @w_id and
                      d_id   = @d_id

  select count(distinct(s_i_id))
                      from  stock, order_line
  where ol_w_id   =  @w_id and
                      ol_d_id   =  @d_id and
                      ol_o_id between @o_id_low and @o_id_high
and
      s_w_id    =  ol_w_id and
                      s_i_id    =  ol_i_id and
                      s_quantity <  @threshhold

go
```

TABLES.SQL

```
/*  TPC-C Benchmark Kit                                 */
/*                                          */
/*  TABLES.SQL                                  */
/*                                          */
/*  Creates TPC-C tables (noseg)                        */


use tpcc
go

checkpoint
go


if exists ( select name from sysobjects where name = 'warehouse' )
                      drop table warehouse
go

create table warehouse
(
                      w_id                  smallint,
                      w_name                      char(10),
                      w_street_1            char(20),
                      w_street_2            char(20),
                      w_city                char(20),
                      w_state                     char(2),
                      w_zip                 char(9),
                      w_tax                 numeric(4,4),
                      w_ytd                 numeric(12,2)
)
go


if exists ( select name from sysobjects where name = 'district' )
                      drop table district
go

create table district
(
                      d_id                  tinyint,
                      d_w_id                      smallint,
                      d_name                      char(10),
                      d_street_1            char(20),
                      d_street_2            char(20),
                      d_city                char(20),
                      d_state                     char(2),
                      d_zip                 char(9),
                      d_tax                 numeric(4,4),
                      d_ytd                 numeric(12,2),
                      d_next_o_id           int
)
go


if exists ( select name from sysobjects where name = 'customer' )
                      drop table customer
go

create table customer
(
                      c_id                  int,
                      c_d_id                      tinyint,
                      c_w_id                      smallint,
                      c_first               char(16),
                      c_middle              char(2),
                      c_last                char(16),
                      c_street_1            char(20),
                      c_street_2            char(20),
                      c_city                char(20),
                      c_state                     char(2),
                      c_zip                 char(9),
                      c_phone                     char(16),
                      c_since                     datetime,
                      c_credit              char(2),
                      c_credit_lim          numeric(12,2),
                      c_discount            numeric(4,4),
                      c_balance             numeric(12,2),
                      c_ytd_payment         numeric(12,2),
                      c_payment_cnt         smallint,
                      c_delivery_cnt        smallint,
                      c_data_1              char(250),
                      c_data_2              char(250)
)
go


if exists ( select name from sysobjects where name = 'history' )
                      drop table history
go

create table history
(
                      h_c_id                      int,
                      h_c_d_id              tinyint,
                      h_c_w_id              smallint,
                      h_d_id                      tinyint,
                      h_w_id                      smallint,
                      h_date                      datetime,
                      h_amount              numeric(6,2),
                      h_data                      char(24)
)
go


if exists ( select name from sysobjects where name = 'new_order' )
                      drop table new_order
go

create table new_order
(
                      no_o_id                     int,
                      no_d_id                     tinyint,
                      no_w_id                     smallint
)
go


if exists ( select name from sysobjects where name = 'orders' )
                      drop table orders
go

create table orders
(
                      o_id                  int,
                      o_d_id                      tinyint,
                      o_w_id                      smallint,
                      o_c_id                      int,
                      o_entry_d             datetime,
                      o_carrier_id          tinyint,
                      o_ol_cnt              tinyint,
                      o_all_local           tinyint
)
go


if exists ( select name from sysobjects where name = 'order_line' )
                      drop table order_line
go

create table order_line
(
                      ol_o_id                     int,
                      ol_d_id                     tinyint,
                      ol_w_id                     smallint,
                      ol_number             tinyint,
                      ol_i_id               int,
                      ol_supply_w_id        smallint,
                      ol_delivery_d         datetime,
                      ol_quantity           smallint,
                      ol_amount             numeric(6,2),
                      ol_dist_info          char(24)
)
go


if exists ( select name from sysobjects where name = 'item' )
                      drop table item
go

create table item
(
                      i_id                  int,
                      i_im_id                     int,
                      i_name                      char(24),
                      i_price                     numeric(5,2),
                      i_data                char(50)
)
go


if exists ( select name from sysobjects where name = 'stock' )
                      drop table stock
go

create table stock
(
                      s_i_id                int,
                      s_w_id                      smallint,
                      s_quantity            smallint,
                      s_dist_01             char(24),
```

```
                    s_dist_02           char(24),
                    s_dist_03           char(24),
                    s_dist_04           char(24),
                    s_dist_05           char(24),
                    s_dist_06           char(24),
                    s_dist_07           char(24),
                    s_dist_08           char(24),
                    s_dist_09           char(24),
                    s_dist_10           char(24),
                    s_ytd               int,
                    s_order_cnt         smallint,
                    s_remote_cnt        smallint,
                    s_data                  char(50)
)
go
```

TPCCBCP.SQL

```
/*  TPC-C Benchmark Kit                                 */
/*                                                      */
/*  TPCCBCP.SQL                                         */
/*                                                      */
/*  This script file sets the table lock option for bulk load       */


use tpcc
go

exec sp_tableoption "warehouse","table lock on bulk load",true
exec sp_tableoption "district","table lock on bulk load",true
exec sp_tableoption "stock","table lock on bulk load",true
exec sp_tableoption "item","table lock on bulk load",true
exec sp_tableoption "customer","table lock on bulk load",true
exec sp_tableoption "history","table lock on bulk load",true
exec sp_tableoption "orders","table lock on bulk load",true
exec sp_tableoption "order_line","table lock on bulk load",true
exec sp_tableoption "new_order","table lock on bulk load",true
go
```

TPCCIRL.SQL

```
/*  TPC-C Benchmark Kit                                 */
/*                                                      */
/*  TPCCIRL.SQL                                         */
/*                                                      */
/*  This script file sets the insert row lock option on selected tables      */


use tpcc
go

exec sp_tableoption "history","insert row lock",true
exec sp_tableoption "new_order","insert row lock",true
exec sp_tableoption "orders","insert row lock",true
exec sp_tableoption "order_line","insert row lock",true
go
```

MAKEFILE.X86

```
!include $(TPC_DIR)\build\ntintel\tpc.inc

CUR_DIR = $(TPC_DIR)\src

CLIENT_EXE     = $(EXE_DIR)\client.exe
MASTER_EXE     = $(EXE_DIR)\master.exe
TPCCLDR_EXE    = $(EXE_DIR)\tpccldr.exe
DELIVERY_EXE   = $(EXE_DIR)\delivery.exe
sqlstat_EXE    = $(EXE_DIR)\sqlstat.exe

all : $(CLIENT_EXE) $(MASTER_EXE) $(TPCCLDR_EXE)
$(DELIVERY_EXE) $(sqlstat_EXE)

$(OBJ_DIR)\client.obj : $(CUR_DIR)\client.c $(INC_DIR)\tpcc.h
    $(CC) $(CFLAGS) /Fo$(OBJ_DIR)\client.obj $(CUR_DIR)\client.c

$(OBJ_DIR)\master.obj : $(CUR_DIR)\master.c $(INC_DIR)\tpcc.h
    $(CC) $(CFLAGS) /Fo$(OBJ_DIR)\master.obj $(CUR_DIR)\master.c

$(OBJ_DIR)\tpccldr.obj : $(CUR_DIR)\tpccldr.c $(INC_DIR)\tpcc.h
    $(CC) $(CFLAGS) /Fo$(OBJ_DIR)\tpccldr.obj $(CUR_DIR)\tpccldr.c

$(OBJ_DIR)\stats.obj : $(CUR_DIR)\stats.c $(INC_DIR)\tpcc.h
    $(CC) $(CFLAGS) /Fo$(OBJ_DIR)\stats.obj $(CUR_DIR)\stats.c

$(OBJ_DIR)\getargs.obj : $(CUR_DIR)\getargs.c   $(INC_DIR)\tpcc.h
    $(CC) $(CFLAGS) /Fo$(OBJ_DIR)\getargs.obj $(CUR_DIR)\getargs.c

$(OBJ_DIR)\util.obj : $(CUR_DIR)\util.c  $(INC_DIR)\tpcc.h
    $(CC) $(CFLAGS) /Fo$(OBJ_DIR)\util.obj $(CUR_DIR)\util.c

$(OBJ_DIR)\time.obj : $(CUR_DIR)\time.c $(INC_DIR)\tpcc.h
    $(CC) $(CFLAGS) /Fo$(OBJ_DIR)\time.obj $(CUR_DIR)\time.c

$(OBJ_DIR)\random.obj : $(CUR_DIR)\random.c $(INC_DIR)\tpcc.h
    $(CC) $(CFLAGS) /Fo$(OBJ_DIR)\random.obj $(CUR_DIR)\random.c

$(OBJ_DIR)\strings.obj : $(CUR_DIR)\strings.c $(INC_DIR)\tpcc.h
    $(CC) $(CFLAGS) /Fo$(OBJ_DIR)\strings.obj $(CUR_DIR)\strings.c

$(OBJ_DIR)\sqlfuncs.obj : $(CUR_DIR)\sqlfuncs.c  $(INC_DIR)\tpcc.h
    $(CC) $(CFLAGS) /Fo$(OBJ_DIR)\sqlfuncs.obj $(CUR_DIR)\sqlfuncs.c

$(OBJ_DIR)\tran.obj : $(CUR_DIR)\tran.c  $(INC_DIR)\tpcc.h
    $(CC) $(CFLAGS) /Fo$(OBJ_DIR)\tran.obj $(CUR_DIR)\tran.c

$(OBJ_DIR)\data.obj : $(CUR_DIR)\data.c  $(INC_DIR)\tpcc.h
    $(CC) $(CFLAGS) /Fo$(OBJ_DIR)\data.obj $(CUR_DIR)\data.c

$(OBJ_DIR)\delivery.obj : $(CUR_DIR)\delivery.c  $(INC_DIR)\tpcc.h
    $(CC) $(CFLAGS) /Fo$(OBJ_DIR)\delivery.obj $(CUR_DIR)\delivery.c

$(OBJ_DIR)\sqlstat.obj : $(CUR_DIR)\sqlstat.c  $(INC_DIR)\tpcc.h
    $(CC) $(CFLAGS) /Fo$(OBJ_DIR)\sqlstat.obj $(CUR_DIR)\sqlstat.c

$(EXE_DIR)\client.exe : $(OBJ_DIR)\client.obj $(OBJ_DIR)\tran.obj
$(OBJ_DIR)\sqlfuncs.obj $(OBJ_DIR)\random.obj $(OBJ_DIR)\util.obj
$(OBJ_DIR)\data.obj $(OBJ_DIR)\getargs.obj $(OBJ_DIR)\time.obj
$(OBJ_DIR)\stats.obj $(OBJ_DIR)\strings.obj
    $(LL) -entry:mainCRTStartup -out:$(EXE_DIR)\client.exe          \
    $(OBJ_DIR)\client.obj $(OBJ_DIR)\tran.obj   $(OBJ_DIR)\sqlfuncs.obj   \
    $(OBJ_DIR)\random.obj $(OBJ_DIR)\util.obj   $(OBJ_DIR)\data.obj     \
    $(OBJ_DIR)\getargs.obj $(OBJ_DIR)\time.obj  $(OBJ_DIR)\stats.obj    \
    $(OBJ_DIR)\strings.obj                                              \
    $(DB_LIB)\ntwdblib.lib $(NTLIBS)

$(EXE_DIR)\master.exe : $(OBJ_DIR)\master.obj $(OBJ_DIR)\sqlfuncs.obj
$(OBJ_DIR)\util.obj $(OBJ_DIR)\getargs.obj $(OBJ_DIR)\time.obj
$(OBJ_DIR)\stats.obj
    $(LL) -entry:mainCRTStartup -out:$(EXE_DIR)\master.exe          \
    $(OBJ_DIR)\master.obj $(OBJ_DIR)\sqlfuncs.obj $(OBJ_DIR)\util.obj   \
    $(OBJ_DIR)\getargs.obj $(OBJ_DIR)\time.obj   $(OBJ_DIR)\stats.obj   \
    $(DB_LIB)\ntwdblib.lib $(NTLIBS)

$(EXE_DIR)\tpccldr.exe : $(OBJ_DIR)\tpccldr.obj $(OBJ_DIR)\getargs.obj
$(OBJ_DIR)\util.obj $(OBJ_DIR)\time.obj $(OBJ_DIR)\random.obj
$(OBJ_DIR)\strings.obj
    $(LL) -entry:mainCRTStartup -out:$(EXE_DIR)\tpccldr.exe         \
    $(OBJ_DIR)\tpccldr.obj $(OBJ_DIR)\getargs.obj $(OBJ_DIR)\strings.obj \
    $(OBJ_DIR)\util.obj $(OBJ_DIR)\time.obj $(OBJ_DIR)\random.obj       \
    $(DB_LIB)\ntwdblib.lib $(NTLIBS)

$(EXE_DIR)\delivery.exe : $(OBJ_DIR)\delivery.obj
$(OBJ_DIR)\sqlfuncs.obj $(OBJ_DIR)\util.obj $(OBJ_DIR)\getargs.obj
$(OBJ_DIR)\time.obj $(OBJ_DIR)\stats.obj
    $(LL) -entry:mainCRTStartup -out:$(EXE_DIR)\delivery.exe         \
    $(OBJ_DIR)\delivery.obj $(OBJ_DIR)\sqlfuncs.obj $(OBJ_DIR)\util.obj
\
    $(OBJ_DIR)\getargs.obj $(OBJ_DIR)\time.obj  $(OBJ_DIR)\stats.obj     \
    $(DB_LIB)\ntwdblib.lib $(NTLIBS)

$(EXE_DIR)\sqlstat.exe : $(OBJ_DIR)\sqlstat.obj $(OBJ_DIR)\sqlfuncs.obj
$(OBJ_DIR)\util.obj $(OBJ_DIR)\getargs.obj $(OBJ_DIR)\time.obj
$(OBJ_DIR)\stats.obj
    $(LL) -entry:mainCRTStartup -out:$(EXE_DIR)\sqlstat.exe          \
    $(OBJ_DIR)\sqlstat.obj $(OBJ_DIR)\sqlfuncs.obj $(OBJ_DIR)\util.obj     \
    $(OBJ_DIR)\getargs.obj $(OBJ_DIR)\time.obj  $(OBJ_DIR)\stats.obj     \
    $(DB_LIB)\ntwdblib.lib $(NTLIBS)
```

RANDOM.C

```
/*                      FILE:           RANDOM.C
 *                                      Microsoft TPC-C Kit Ver.
3.00.000
 *                                      Audited 08/23/96, By
Francois Raab
 *
 *                                      Copyright Microsoft, 1996
 *
 *                      PURPOSE:   Random number generation
functions for Microsoft TPC-C Benchmark Kit
 *                      Author:         Damien Lindauer
 *                                      damienl@Microsoft.com
 */

// Includes
#include "tpcc.h"
#include "math.h"

// Defines
#define A          16807
#define M       2147483647
#define Q         127773      /* M div A */
#define R           2836      /* M mod A */
#define Thread              __declspec(thread)

// Globals
long                    Thread Seed = 0;      /* thread local seed */

/*****************************************************************************
*                                                                            *
* random -                                                                   *
*       Implements a GOOD pseudo random number generator.  This
generator     *
*       will/should? run the complete period before repeating.               *
*                                                                            *
```

```
* Copied from:                                                  *
*      Random Numbers Generators: Good Ones Are Hard to Find.
*
*      Communications of the ACM - October 1988 Volume 31 Number 10
*
*                                                              *
* Machine Dependencies:                                        *
*      long must be 2 ^ 31 - 1 or greater.                     *
*                                                              *
*******************************************************************************/

/*******************************************************************************
* seed - load the Seed value used in irand and drand.  Should be used
before  *
*      first call to irand or drand.                           *
*******************************************************************************/

void seed(long val)
{

#ifdef DEBUG
   printf("[%ld]DBG: Entering seed()...\n", (int) GetCurrentThreadId());
                                printf("Old Seed %ld New Seed %ld\n",Seed,
val);
#endif

   if ( val < 0 )
                                   val = abs(val);

   Seed = val;

}


/*******************************************************************************
*                                                              *
* irand - returns a 32 bit integer pseudo random number with a period of     *
*      1 to 2 ^ 32 - 1.                                        *
*                                                              *
* parameters:                                                  *
*      none.                                                   *
*                                                              *
* returns:                                                     *
*      32 bit integer - defined as long ( see above ).         *
*                                                              *
* side effects:                                                *
*      seed get recomputed.                                    *
*******************************************************************************/

long irand()
{
   register long    s;     /* copy of seed */
   register long    test;  /* test flag */
   register long    hi;    /* tmp value for speed */
   register long    lo;    /* tmp value for speed */

#ifdef DEBUG
   printf("[%ld]DBG: Entering irand()...\n", (int) GetCurrentThreadId());
#endif

   s = Seed;
   hi = s / Q;
   lo = s % Q;

   test = A * lo - R * hi;
   if ( test > 0 )
                                   Seed = test;
   else
                                   Seed = test + M;

   return( Seed );
}


/*******************************************************************************
*                                                              *
* drand - returns a double pseudo random number between 0.0 and 1.0.
*
*      See irand.                                              *
*******************************************************************************/
double drand()
{

#ifdef DEBUG
   printf("[%ld]DBG: Entering drand()...\n", (int) GetCurrentThreadId());
#endif

   return( (double)irand() / 2147483647.0);
}


//=============================================================
==============
// Function   : RandomNumber
//
// Description:
//=============================================================
==============
long RandomNumber(long lower, long upper)
{
   long rand_num;

#ifdef DEBUG
   printf("[%ld]DBG: Entering RandomNumber()...\n", (int)
GetCurrentThreadId());
#endif

                                if ( upper == lower ) /* pgd 08-13-96 perf
enhancement */
                                   return lower;

                                upper++;

   if ( upper <= lower )
                                   rand_num = upper;
   else
                                   rand_num = lower + irand() % (upper -
lower); /* pgd 08-13-96 perf enhancement */


#ifdef DEBUG
   printf("[%ld]DBG: RandomNumber between %ld & %ld ==> %ld\n",
                                                   (int) GetCurrentThreadId(),
lower, upper, rand_num);
#endif


   return rand_num;
}


#if 0

//Orginal code pgd 08/13/96

long RandomNumber(long lower,
                                                   long upper)
{
   long rand_num;

#ifdef DEBUG
   printf("[%ld]DBG: Entering RandomNumber()...\n", (int)
GetCurrentThreadId());
#endif

                                upper++;

   if ((upper <= lower))
                                   rand_num = upper;
   else
                                   rand_num = lower + irand() % ((upper >
lower) ? upper - lower : upper);


#ifdef DEBUG
   printf("[%ld]DBG: RandomNumber between %ld & %ld ==> %ld\n",
                                                   (int) GetCurrentThreadId(),
lower, upper, rand_num);
#endif


   return rand_num;
}
#endif

//=============================================================
==============
// Function   : NURand
//
// Description:
//=============================================================
==============
long NURand(int iConst,
                                   long x,
                                   long y,
                                   long C)
{
   long rand_num;

#ifdef DEBUG
   printf("[%ld]DBG: Entering NURand()...\n", (int) GetCurrentThreadId());
#endif

   rand_num = (((RandomNumber(0,iConst) | RandomNumber(x,y)) + C) %
(y-x+1))+x;

#ifdef DEBUG
   printf("[%ld]DBG: NURand: num = %d\n", (int) GetCurrentThreadId(),
rand_num);
#endif

   return rand_num;
}
```

STRINGS.C

```
/*                              FILE:           STRINGS.C
 *                                              Microsoft TPC-C Kit Ver.
3.00.000
```

```
 *                                      Audited 08/23/96, By
Francois Raab
 *
 *                                      Copyright Microsoft, 1996
 *
 *                      PURPOSE:        String generation functions for
Microsoft TPC-C Benchmark Kit
 *                      Author:         Damien Lindauer
 *                                      damienl@Microsoft.com
 */

// Includes
#include "tpcc.h"
#include <string.h>
#include <ctype.h>

//=======================================================
==============
//
// Function name: MakeAddress
//
//=======================================================
==============

void MakeAddress(char *street_1,
                                        char *street_2,
                                        char *city,
                                        char *state,
                                        char *zip)
{

#ifdef DEBUG
    printf("[%ld]DBG: Entering MakeAddress()\n", (int) GetCurrentThreadId());
#endif

    MakeAlphaString (10, 20, ADDRESS_LEN, street_1);
    MakeAlphaString (10, 20, ADDRESS_LEN, street_2);
    MakeAlphaString (10, 20, ADDRESS_LEN, city);
    MakeAlphaString ( 2,  2, STATE_LEN, state);
    MakeZipNumberString( 9,  9, ZIP_LEN, zip);

#ifdef DEBUG
    printf("[%ld]DBG: MakeAddress: street_1: %s, street_2: %s, city: %s,
state: %s, zip: %s\n",
                                (int) GetCurrentThreadId(),
street_1, street_2, city, state, zip);
#endif

    return;

}


//=======================================================
==============
//
// Function name: LastName
//
//=======================================================
==============

void LastName(int  num,
                                char *name)
{
    int                         i;
                int             len;
    static char *n[] =
    {
                                "BAR" , "OUGHT", "ABLE" , "PRI"  ,
"PRES",
                                "ESE" , "ANTI" , "CALLY", "ATION",
"EING"
    };

#ifdef DEBUG
    printf("[%ld]DBG: Entering LastName()\n", (int) GetCurrentThreadId());
#endif

    if ((num >= 0) && (num < 1000))
    {
                                strcpy(name, n[(num/100)%10]);
                                strcat(name, n[(num/10)%10]);
                                strcat(name, n[(num/1)%10]);

                                if (strlen(name) < LAST_NAME_LEN)
                                {
                                        PaddString(LAST_NAME_LEN,
name);
                                }

    }
    else
    {
                                printf("\nError in LastName()... num <%ld>
out of range (0,999)\n", num);
                                exit(-1);
    }


#ifdef DEBUG
    printf("[%ld]DBG: LastName: num = [%d] ==> [%d][%d][%d]\n",
                                        (int) GetCurrentThreadId(), num,
num/100, (num/10)%10, num%10);
                                printf("[%ld]DBG: LastName: String = %s\n", (int)
GetCurrentThreadId(), name);
#endif

    return;

}


//=======================================================
==============
//
// Function name: MakeAlphaString
//
//=======================================================
==============

//philipdu 08/13/96 Changed MakeAlphaString to use A-Z, a-z, and 0-9 in
//accordance with spec see below:
//The spec says:
//4.3.2.2                       The notation random a-string [x .. y]
//(respectively, n-string [x .. y]) represents a string of random alphanumeric
//(respectively, numeric) characters of a random length of minimum x,
maximum y,
//and mean (y+x)/2. Alphanumerics are A..Z, a..z, and 0..9.  The only other
//requirement is that the character set used "must be able to represent a
minimum
//of 128 different characters".  We are using 8-bit chars, so this is a non
issue.
//It is completely unreasonable to stuff non-printing chars into the text fields.
//-CLevine 08/13/96

int MakeAlphaString( int  x, int  y, int  z, char *str)
{
                int             len;
                int             i;
                static  char chArray[] =
"0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstu
vwxyz";
                static  int     chArrayMax = 61;

#ifdef DEBUG
    printf("[%ld]DBG: Entering MakeAlphaString()\n", (int)
GetCurrentThreadId());
#endif

                len= RandomNumber(x, y);

                for (i=0; i<len; i++)
                        str[i] = chArray[RandomNumber(0,
chArrayMax)];
                if ( len < z )
                        memset(str+len, ' ', z - len);
                str[len] = 0;

    return len;
}

#if 0
//philipdu 08/13/96 Orginal MakeAlphaString

int MakeAlphaString( int  x,
                                        int  y,
                                             int  z,
                                        char *str)
{
                int             len;
                int             i;

#ifdef DEBUG
    printf("[%ld]DBG: Entering MakeAlphaString()\n", (int)
GetCurrentThreadId());
#endif

                len= RandomNumber(x, y);

                for (i=0; i<len; i++)
                {
                        str[i] = RandomNumber(MINPRINTASCII,
MAXPRINTASCII);
                }

                str[len] = '\0';

                if (len < z)
                {
                        PaddString(z, str);
                }

    return (len);
}
#endif


//=======================================================
==============
//
// Function name: MakeOriginalAlphaString
//
```

```
//=====================================================
==============

int MakeOriginalAlphaString(int  x,
                                                    int  y,
                                                         int  z,
                                                    char *str,
                                                   int  percent)
{
   int                          len;
   int                          val;
   int                          start;

#ifdef DEBUG
   printf("[%ld]DBG: Entering MakeOriginalAlphaString()\n", (int)
GetCurrentThreadId());
#endif

   // verify prercentage is valid
   if ((percent < 0) || (percent > 100))
   {
                                printf("MakeOrigianlAlphaString: Invalid
percentage: %d\n", percent);
                                exit(-1);
   }

   // verify string is at least 8 chars in length
   if ((x + y) <= 8)
   {
                                printf("MakeOriginalAlphaString: string
length must be >= 8\n");
                                exit(-1);
   }

   // Make Alpha String
   len = MakeAlphaString(x,y, z, str);

   val = RandomNumber(1,100);
   if (val <= percent)
   {
                                start = RandomNumber(0, len - 8);
                                strncpy(str + start, "ORIGINAL", 8);
   }


#ifdef DEBUG
   printf("[%ld]DBG: MakeOriginalAlphaString: : %s\n",
                                        (int) GetCurrentThreadId(), str);
#endif

   return strlen(str);
}


//=====================================================
==============
//
// Function name: MakeNumberString
//
//=====================================================
==============
int MakeNumberString(int  x, int  y, int  z, char *str)
{
                        char tmp[16];

                        //MakeNumberString is always called
MakeZipNumberString(16, 16, 16, string)

                        memset(str, '0', 16);
                        itoa(RandomNumber(0, 99999999), tmp, 10);
                        memcpy(str, tmp, strlen(tmp));

                        itoa(RandomNumber(0, 99999999), tmp, 10);
                        memcpy(str+8, tmp, strlen(tmp));

                        str[16] = 0;

   return 16;
}

#if 0
int MakeNumberString(int  x,
                                                    int  y,
                                                         int  z,
                                                    char *str)
{
   int                          len;
   int                          i;

#ifdef DEBUG
   printf("[%ld]DBG: Entering MakeNumberString()\n", (int)
GetCurrentThreadId());
#endif

   len = RandomNumber(x,y);


   for (i=0; i < len; i++)
   {
                                str[i] = (char) (RandomNumber(48,57));
   }

   str[len] = '\0';

                                PaddString(z, str);

   return strlen(str);
}

#endif

//=====================================================
==============
//
// Function name: MakeZipNumberString
//
//=====================================================
==============
int MakeZipNumberString(int  x, int  y, int  z, char *str)
{
                        char tmp[16];

                        //MakeZipNumberString is always called
MakeZipNumberString(9, 9, 9, string)

                        strcpy(str, "000011111");

                        itoa(RandomNumber(0, 9999), tmp, 10);
                        memcpy(str, tmp, strlen(tmp));

   return 9;
}

#if 0
//pgd 08/14/96 Orginal Code Below
int MakeZipNumberString(int  x,
                                                    int  y,
                                                         int  z,
                                                    char *str)
{
   int                          len;
   int                          i;

#ifdef DEBUG
   printf("[%ld]DBG: Entering MakeZipNumberString()\n", (int)
GetCurrentThreadId());
#endif

   len = RandomNumber(x-5,y-5);

   for (i=0; i < len; i++)
   {
                                str[i] = (char) (RandomNumber(48,57));
   }

   str[len] = '\0';

                                strcat(str, "11111");

                                PaddString(z, str);

   return strlen(str);
}
#endif

//=====================================================
==============
//
// Function name: InitString
//
//=====================================================
==============
void InitString(char *str, int len)
{
   int i;

#ifdef DEBUG
   printf("[%ld]DBG: Entering InitString()\n", (int) GetCurrentThreadId());
#endif

                                memset(str, ' ', len);
   str[len] = 0;
}

#if 0
//Orginal pgd 08/14/96
void InitString(char *str, int len)
{
   int i;

#ifdef DEBUG
   printf("[%ld]DBG: Entering InitString()\n", (int) GetCurrentThreadId());
#endif


   for (i=0; i< len; i++)
                                str[i] = ' ';
   str[len] = '\0';
}

#endif

//=====================================================
==============
```

```
// Function name: InitAddress
//
// Description:
//
//=============================================================
==============

void InitAddress(char *street_1, char *street_2, char *city, char *state, char
*zip)
{
                        int i;


                        memset(street_1, ' ', ADDRESS_LEN+1);
                        memset(street_2, ' ', ADDRESS_LEN+1);
                        memset(city, ' ', ADDRESS_LEN+1);

   street_1[ADDRESS_LEN+1] = 0;
   street_2[ADDRESS_LEN+1] = 0;
   city[ADDRESS_LEN+1]     = 0;

                        memset(state, ' ', STATE_LEN+1);
   state[STATE_LEN+1] = 0;

                        memset(zip, ' ', ZIP_LEN+1);
   zip[ZIP_LEN+1] = 0;
}

#if 0
//Orginal pgd 08/14/96
void InitAddress(char *street_1,
                                        char *street_2,
                                        char *city,
                                        char *state,
                                        char *zip)
{
                        int i;

#ifdef DEBUG
   printf("[%ld]DBG: Entering InitAddress()\n", (int) GetCurrentThreadId());
#endif

   for (i=0; i< ADDRESS_LEN+1; i++)
   {
                                street_1[i] = ' ';
                                street_2[i] = ' ';
                                city[i]     = ' ';
   }

   street_1[ADDRESS_LEN+1] = '\0';
   street_2[ADDRESS_LEN+1] = '\0';
   city[ADDRESS_LEN+1]     = '\0';

   for (i=0; i< STATE_LEN+1; i++)
                                state[i] = ' ';
   state[STATE_LEN+1] = '\0';

   for (i=0; i< ZIP_LEN+1; i++)
                                zip[i] = ' ';
   zip[ZIP_LEN+1] = '\0';
}

#endif
//=============================================================
==============
//
// Function name: PaddString
//
//=============================================================
==============

void PaddString(int max, char *name)
{
                        int             i;
                        int             len;

                        len = strlen(name);
                        if ( len < max )
                                memset(name+len, ' ', max - len);
                        name[max] = 0;

                        return;
}

#if 0
                        //pgd 08/14/96 Orginal code below
                        void PaddString(int             max,
                                                char    *name)
                        {

                                int             i;
                                int             len;

                        #ifdef DEBUG
                                printf("[%ld]DBG: Entering
PaddString()\n", (int) GetCurrentThreadId());
                        #endif

                                len = strlen(name);

                                for (i=1;i<=(max - len);i++)
                                {
                                        strcat(name, " ");
                                }

                        }
#endif
```

TIME.C

```
// TPC-C Benchmark Kit
//
// Module:  TIME.C
// Author:  DamienL


// Includes
#include "tpcc.h"

// Globals
static long start_sec;


//=============================================================
==============
//
// Function name: TimeNow
//
//=============================================================
==============

long TimeNow()
{
  long                  time_now;
  struct                _timeb el_time;

#ifdef DEBUG
   printf("[%ld]DBG: Entering TimeNow()\n", (int) GetCurrentThreadId());
#endif

  _ftime(&el_time);

  time_now = ((el_time.time - start_sec) * 1000) + el_time.millitm;

  return time_now;
}


//=============================================================
==============
//
// Function name: TimeInit
//
// This function is used to normalize the seconds component of
// elapsed time so that it will not overflow, when converted to milli seconds
//
//=============================================================
==============

void TimeInit()
{

   struct  _timeb  norm_time;

#ifdef DEBUG
   printf("[%ld]DBG: Entering TimeInit()\n", (int) GetCurrentThreadId());
#endif

   _ftime(&norm_time);
   start_sec = norm_time.time;
}


//=============================================================
==============
//
// Function name: TimeKeying
//
//=============================================================
==============

void TimeKeying(int             TranType,
                                        double          load_multiplier)
{

#ifdef DEBUG
   printf("[%ld]DBG: Entering TimeKeying()\n", (int) GetCurrentThreadId());
#endif
                        switch (TranType)
                        {
                           case NEW_ORDER_TRAN:
                                UtilSleepMs( (long)
((load_multiplier * 18)*1000) );
                                break;

                           case PAYMENT_TRAN:
                                UtilSleepMs( (long)
((load_multiplier * 3)*1000) );
                                break;

                           case ORDER_STATUS_TRAN:
                           case DELIVERY_TRAN:
```

```
                        case STOCK_LEVEL_TRAN:
                                UtilSleepMs( (long)
((load_multiplier * 2)*1000) );
                                break;

                        default:
                                printf("TimeKeying: Error - default
reached!\n");

                }
}


//=================================================
=============
//
// Function name: TimeThink
//
//=================================================
=============

void TimeThink(int              TranType,
                        double      load_multiplier)
{

#ifdef DEBUG
    printf("[%ld]DBG: Entering TimeThink()\n", (int) GetCurrentThreadId());
#endif

                switch (TranType)
                {
                    case NEW_ORDER_TRAN:
                    case PAYMENT_TRAN:
                                UtilSleepMs( (long)
((load_multiplier * 12)*1000) );
                                break;

                    case ORDER_STATUS_TRAN:
                                UtilSleepMs( (long)
((load_multiplier * 10)*1000) );
                                break;

                    case DELIVERY_TRAN:
                    case STOCK_LEVEL_TRAN:
                                UtilSleepMs( (long)
((load_multiplier * 5)*1000) );
                                break;

                    default:
                                printf("TimeThink: Error - default
reached!\n");

                }
}
```

TPCC.H

```
/*              FILE:       TPCC.H
 *                          Microsoft TPC-C Kit Ver.
3.00.000
 *                          Audited 08/23/96, By
Francois Raab
 *
 *                          Copyright Microsoft, 1996
 *
 *              PURPOSE:    Header file for Microsoft TPC-C
Benchmark Kit
 *              Author:     Damien Lindauer
 *                          damienl@Microsoft.com
 */

// Build number of TPC Benchmark Kit
#define TPCKIT_VER      "3.00.02"

// General headers
#include <windows.h>
#include <winbase.h>
#include <stdlib.h>
#include <stdio.h>
#include <process.h>
#include <stddef.h>
#include <stdarg.h>
#include <string.h>
#include <signal.h>
#include <time.h>
#include <timeb.h>
#include <types.h>
#include <wincon.h>

#ifdef USE_ODBC
// ODBC headers
#include <sql.h>
#include <sqlext.h>
HENV                                    henv;
#endif

// DB-Library headers
#include <sqlfront.h>
#include <sqldb.h>

#include "trans.h"                      //pgd 5-6-96 split transaction
structs defination into own header

                        //for tpcform.c i.e. telnet application

// Critical section declarations
CRITICAL_SECTION        ConsoleCritSec;
CRITICAL_SECTION        QueuedDeliveryCritSec;
CRITICAL_SECTION        WriteDeliveryCritSec;
CRITICAL_SECTION        DroppedConnectionsCritSec;
CRITICAL_SECTION        ClientErrorLogCritSec;

// General constants
#define SQLCONN DBPROCESS
#define DUMB_MESSAGE            5701
#define ABORT_ERROR             6104
#define INVALID_ITEM_ID             0
#define MILLI                1000
#define MAX_THREADS             2510
#define STATS_MSG_LOW               3600
#define STATS_MSG_HIGH              3700
#define SHOWPLAN_MSG_LOW            6200
#define SHOWPLAN_MSG_HIGH           6300
#define FALSE                   0
#define TRUE                    1
#define UNDEF                           -1
#define MINPRINTASCII            32
#define MAXPRINTASCII            126

// Default environment constants
#define SERVER                  ""
#define DATABASE                "tpcc"
#define USER                    "sa"
#define PASSWORD                ""
#define SYNCH_SERVERNAME                ""

// Statistic constants
#define INTERVAL                20   // Total interval of buckets, in sec
#define UNIT                    .1   // Time period of each bucket
#define HIST_MAX                200   // Num of histogram buckets =
INTERVAL/UNIT
#define BUCKET                  100   // Division factor for response time

// Default master arguments
#define ADMIN_DATABASE          "tpcc_admin"
#define RAMP_UP                 600
#define STEADY_STATE            1200
#define RAMP_DOWN               120
#define NUM_USERS               10
#define NUM_WAREHOUSES                  1
#define THINK_TIMES                     0
#define DISPLAY_DATA            0
#define DEFMSPACKSIZE            4096
#define TRANSACTION             0
#define CLIENT_MODE             1
#define DEF_WW_T                        120
#define DEF_WW_a                        1
#define DEADLOCK_RETRY                  4
#define DELIVERY_BACKOFF                2
#define DELIVERY_MODE                   0
#define NEWORDER_MODE                   0
#define DEF_LOAD_MULTIPLIER         1.0
#define DEF_CHECKPOINT_INTERVAL   960
#define DEF_FIRST_CHECKPOINT        240
#define DISABLE_90TH                    0
#define                 RESFILENAME
                        "results.txt"
#define SQLSTAT_FILENAME                "sqlstats.txt"
#define ENABLE_SQLSTAT                  0
#define SQLSTAT_PERIOD                  100
#define SHUTDOWN_SERVER                 0
#define AUTO_RUN                        0
#define DISABLE_SQLPERF                 0

// Default client arguments
#define NUM_THREADS                 10
#define X_FLAG              0
#define Y_FLAG              1
#define NUM_DELIVERIES                  2
#define CLIENT_NURAND                223
#define DISABLE_DELIVERY_RESFILES   1
#define ENABLE_QJ                       0

// Globals for queued delivery handling
typedef struct          delivery_node *DELIVERY_PTR;
DELIVERY_PTR            delivery_head, delivery_tail;
short                           queued_delivery_cnt;
HANDLE                          hDeliveryMonPipe;
struct delivery_node
{
                        short                   w_id;
                        short                   o_carrier_id;
                        SYSTEMTIME                      queue_time;
                        long                    tran_start_time;
                        struct                  delivery_node
*next_delivery;
};

// Default loader arguments
#define BATCH               10000
#define DEFLDPACKSIZE           4096
```

```
#define ORDERS_PER_DIST          3000
#define LOADER_RES_FILE    "load.out"
#define LOADER_NURAND_C          123
#define DEF_STARTING_WAREHOUSE     1
#define BUILD_INDEX                1
#define INDEX_SCRIPT_PATH   "scripts"

// Transaction types
#define EMPTY                    0
#define NEW_ORDER_TRAN           1
#define PAYMENT_TRAN             2
#define ORDER_STATUS_TRAN        3
#define DELIVERY_TRAN            4
#define STOCK_LEVEL_TRAN         5

// Statistic structures
typedef struct
{
   long                          tran_count;
   long                          total_time;
   long                          resp_time;
   long                          resp_min;
   long                          resp_max;
   long                          rolled_back;
   long                          tran_2sec;
   long                          tran_5sec;
   long                          tran_sqr;
                    long                 num_deadlocks;
   long                          resp_hist[HIST_MAX];
} TRAN_STATS;

typedef struct
{
   TRAN_STATS                    NewOrderStats;
   TRAN_STATS                    PaymentStats;
   TRAN_STATS                    OrderStatusStats;
   TRAN_STATS                    QueuedDeliveryStats;
               TRAN_STATS                 DeliveryStats;
   TRAN_STATS                    StockLevelStats;
} CLIENT_STATS;

// driver structures
typedef struct
{
   char                          *server;
   char                          *database;
   char                          *user;
   char                          *password;
   char                          *table;
   long                          num_warehouses;
   long                          batch;
   long                          verbose;
                    long                 pack_size;
                    char                 *loader_res_file;
                    char                 *synch_servername;
                    long                 case_sensitivity;
                    long                 starting_warehouse;
                    long                 build_index;
                    char                 *index_script_path;
} TPCCLDR_ARGS;

typedef struct
{
   char                          *server;
   char                          *user;
   char                          *password;
                    char                 *admin_database;
   char                          *sqlstat_filename;
                    long                 run_id;
} SQLSTAT_ARGS;

typedef struct
{
   SQLCONN                       *sqlconn;
   char                          *server;
   char                          *database;
                    char                 *admin_database;
   char                          *user;
   char                          *password;
   long                          ramp_up;
   long                          steady_state;
   long                          ramp_down;
   long                          num_users;
   long                          num_warehouses;
   long                          think_times;
   long                          display_data;
                    long                 client_mode;
                    long                 tran;
                    long                 deadlock_retry;
                    long                 delivery_backoff;
                    long                 num_deliveries;
                    char                 *comment;
                    double                    load_multiplier;
                    long                 checkpoint_interval;
                    long                 first_checkpoint;
                    long                 disable_90th;
                    char                 *resfilename;
                    char                 *sqlstat_filename;
                    long                 enable_sqlstat;
                    long                 sqlstat_period;
                    long                 shutdown_server;
                    long                 auto_run;
                    long                 dropped_connections;
   short                         spid;
                    long                 disable_sqlperf;
} MASTER_DATA;

typedef struct
{
                    long                 num_threads;
   char                          *server;
   char                          *database;
                    char                 *admin_database;
   char                          *user;
   char                          *password;
                    long                 pack_size;
                    short                x_flag;
                    char                 *synch_servername;
                    long
                    disable_delivery_resfiles;
                    long                 enable_qj;
#ifdef USE_CONMON
                    HANDLE                    hConMon;
                    short                con_id;
                    short                con_x;
                    short                con_y;
#endif
} GLOBAL_CLIENT_DATA;

typedef struct
{
#ifdef USE_ODBC
                    HDBC                 hdbc;
                    HSTMT                     hstmt;
#else
   SQLCONN                       *sqlconn;
#endif
                    short                threadid;
   char                          *server;
   char                          *database;
                    char                 *admin_database;
   char                          *user;
   char                          *password;
   long                          ramp_up;
   long                          steady_state;
   long                          ramp_down;
   long                          num_warehouses;
                    long                 client_mode;
                    long                 tran;
                    long                 deadlock_retry;
                    long                 think_times;
                    long                 pack_size;
   long                          tran_start_time;
   long                          tran_end_time;
   long                          display_data;
                    long                 id;
                    short                w_id;
   short                         spid;
                    long                 disable_90th;
                    double                    load_multiplier;
                    long                 num_deliveries;
                    long                 enable_qj;
#ifdef USE_CONMON
                    HANDLE                    hConMon;
                    short                con_id;
                    short                con_x;
                    short                con_y;
                    short                threadid;
#endif
} CLIENT_DATA;

typedef struct
{
#ifdef USE_ODBC
                    HDBC                 hdbc;
                    HSTMT                     hstmt;
#else
                    SQLCONN              *sqlconn;
                    SYSTEMTIME                queue_time;
                    SYSTEMTIME
                    completion_time;
   long                          tran_start_time;
   long                          tran_end_time;
                    short                threadid;
                    FILE                 *fDelivery;
   short                         spid;
   short                         w_id;
                    short                d_id;
                    short                o_carrier_id;
                    DEL_ITEM             DelItems[10];
                    char                 *server;
   char                          *database;
                    char                 *admin_database;
   char                          *user;
   char                          *password;
   long                          ramp_up;
   long                          steady_state;
   long                          ramp_down;
                    long                 pack_size;
                    long                 id;
                    long                 disable_90th;
                    long                 delivery_backoff;
```

```
                        long
                        disable_delivery_resfiles;
                        long                    enable_qj;
} DELIVERY;

typedef struct
{
                        long                    pipe_num;
} DELIVERY_ARGS;

// For client synchronization
#define LINE_LEN   80
#define NAME_SIZE  25
#define IN_BUF_SIZE     1000
#define OUT_BUF_SIZE    1000
#define TIME_OUT        0
#define PLEASE_READ     1000
#define PLEASE_WRITE    1000

typedef struct _WRTHANDLE
{                       HANDLE          hPipe;
   DWORD                        threadID;
                        CHAR        Name[NAME_SIZE];
                        struct      _WRTHANDLE *    next;
}WRTHANDLE;

// For client console monitor
#ifdef USE_CONMON
#define                 CON_LINE_SIZE           40
#define                 DEADLOCK_X                      17
#define DEADLOCK_Y              4
#define CUR_STATE_X             15
#define CUR_STATE_Y             3
#define                 YELLOW                          0
#define                 RED                             1
#define                 GREEN                           2
int                             total_deadlocks;
#endif

// Functions in random.c
void                    seed();
long                    irand();
double                  drand();
void                    WUCreate();
short                   WURand();

// Functions in getargs.c;
void                    GetArgsLoader();
void                    GetArgsLoaderUsage();
void                    GetArgsMaster();
void                    GetArgsMasterUsage();
void                    GetArgsClient();
void                    GetArgsClientUsage();
void                    GetArgsDelivery();
void                    GetArgsDeliveryUsage();
void                    GetArgsSQLStat();
void                    GetArgsSQLStatUsage();

// Functions in master.c
void                    ReadClientDone();
BOOL    CtrlHandler();

// Functions in client.c
void                    ClientMain();
void                    DeliveryMain();
void                    Delivery();
void                    ClientEmulate();
short                   ClientSelectTransaction();
void                    ClientShuffleDeck();

//Functions in tran.c
BOOL                    TranNewOrder();
BOOL                    TranPayment();
BOOL                    TranOrderStatus();
BOOL                    TranDelivery();
BOOL                    TranStockLevel();

// Functions in data.c
void                    DataNewOrder();
void                    DataPayment();
void                    DataOrderStatus();
void                    DataDelivery();
void                    DataStockLevel();
short                   DataRemoteWarehouse();

// Functions in time.c
long                    TimeNow();
void                    TimeInit();
void                    TimeKeying();
void                    TimeThink();

// Functions in stats.c
void                    StatsInit();
void                    StatsInitTran();
void                    StatsGeneral();
void                    StatsDelivery();

// Functions in sqlfuncs.c
BOOL                    SQLExec();
BOOL                    SQLExecCmd();
BOOL                    SQLOpenConnection();
void                    SQLClientInit();
int                     SQLMasterInit();
void                    SQLDeliveryInit();
int                     SQLClientStats();
int                     SQLDeliveryStats();
void                    SQLTranStats();
void                    SQLMasterStats();
void                    SQLMasterTranStats();
void                    SQLIOStats();
void                    SQLCheckpointStats();
void                    SQLInitResFile();
void                    SQLGetRunId();
BOOL                    SQLNewOrder();
BOOL                    SQLPayment();
BOOL                    SQLOrderStatus();
BOOL                    SQLStockLevel();
void    SQLDelivery();
int                     SQLGetCustId();
void                    SQLExit();
void                    SQLInit();
void                    SQLInitPrivate();
void                    SQLClientInitPrivate();
void                    SQLDeliveryInitPrivate();
int                     SQLMsgHandler();
int                     SQLErrHandler();
int                     SQLClientMsgHandler();
int                     SQLClientErrHandler();
int                     SQLDeliveryMsgHandler();
int                     SQLDeliveryErrHandler();
void                    SQLInitDate();
void                    SQLShutdown();
#ifdef USE_ODBC
void                    ODBCOpenConnection();
void                    ODBCOpenDeliveryConnection();
BOOL                    ODBCError();
void                    ODBCExit();
#endif

// Functions in util.c
void                    UtilSleep();
void                    UtilPrintNewOrder();
void                    UtilPrintPayment();
void                    UtilPrintOrderStatus();
void                    UtilPrintDelivery();
void                    UtilPrintStockLevel();
void                    UtilPrintOlTable();
void                    UtilError();
void                    UtilFatalError();
void                    UtilStrCpy();
#ifdef USE_CONMON
void                    WriteConsoleString();
#endif
void                    WriteDeliveryString();
BOOL                    AddDeliveryQueueNode();
BOOL                    GetDeliveryQueueNode();


// Functions in strings.c
void                    MakeAddress();
void                    LastName();
int                     MakeAlphaString();
int                     MakeOriginalAlphaString();
int                     MakeNumberString();
int                     MakeZipNumberString();
void                    InitString();
void                    InitAddress();
void                    PaddString();

// Functions in delivery.c
void                    DeliveryHMain();
void                    DeliveryH();
```

TPCCLDR.C

```
/*                      FILE:           TPCCLDR.C
 *                                      Microsoft TPC-C Kit Ver.
3.00.000
 *                                      Audited 08/23/96, By
Francois Raab
 *
 *                                      Copyright Microsoft, 1996
 *
 *                      PURPOSE:    Database loader for Microsoft TPC-
C Benchmark Kit
 *                      Author:         Damien Lindauer
 *                                      damienl@Microsoft.com
 */


// Includes
#include "tpcc.h"
#include "search.h"

// Defines
#define MAXITEMS                100000
#define CUSTOMERS_PER_DISTRICT  3000
#define DISTRICT_PER_WAREHOUSE  10
#define ORDERS_PER_DISTRICT     3000
#define MAX_CUSTOMER_THREADS    2
#define MAX_ORDER_THREADS       3
```

```
#define MAX_MAIN_THREADS       4

// Functions declarations
long NURand();
void LoadItem();
void LoadWarehouse();

void Stock();
void District();

void LoadCustomer();
void CustomerBufInit();
void CustomerBufLoad();
void LoadCustomerTable();
void LoadHistoryTable();

void LoadOrders();
void OrdersBufInit();
void OrdersBufLoad();
void LoadOrdersTable();
void LoadNewOrderTable();
void LoadOrderLineTable();
void GetPermutation();
void CheckForCommit();
void OpenConnections();

void BuildIndex();

void CurrentDate();

// Shared memory structures

typedef struct
{
   long          ol;
   long          ol_i_id;
   short         ol_supply_w_id;
   short         ol_quantity;
   double        ol_amount;
   char          ol_dist_info[DIST_INFO_LEN+1];
                 // Added to insure ol_delivery_d set properly
during load
                 char                    ol_delivery_d[30];
} ORDER_LINE_STRUCT;

typedef struct
{
   long          o_id;
   short         o_d_id;
   short         o_w_id;
   long          o_c_id;
   short         o_carrier_id;
   short         o_ol_cnt;
   short         o_all_local;
                 ORDER_LINE_STRUCT   o_ol[15];
} ORDERS_STRUCT;

typedef struct
{
   long                          c_id;
   short                         c_d_id;
   short                         c_w_id;
                 char
                 c_first[FIRST_NAME_LEN+1];
   char
                 c_middle[MIDDLE_NAME_LEN+1];
   char
                 c_last[LAST_NAME_LEN+1];
   char
                 c_street_1[ADDRESS_LEN+1];
   char
                 c_street_2[ADDRESS_LEN+1];
   char                          c_city[ADDRESS_LEN+1];
   char                          c_state[STATE_LEN+1];
   char                          c_zip[ZIP_LEN+1];
   char                          c_phone[PHONE_LEN+1];
   char                          c_credit[CREDIT_LEN+1];
   double                        c_credit_lim;
   double                        c_discount;
   double                        c_balance;
   double                        c_ytd_payment;
                 short                   c_payment_cnt;
                 short                   c_delivery_cnt;
                 char
                 c_data_1[C_DATA_LEN+1];
   char                          c_data_2[C_DATA_LEN+1];
                 double                  h_amount;
   char                          h_data[H_DATA_LEN+1];
} CUSTOMER_STRUCT;

typedef struct
{
                 char
                 c_last[LAST_NAME_LEN+1];
                 char
                 c_first[FIRST_NAME_LEN+1];
   long                          c_id;
} CUSTOMER_SORT_STRUCT;

typedef struct
{
   long          time_start;
} LOADER_TIME_STRUCT;


// Global variables
char          errfile[20];
DBPROCESS     *i_dbproc1;
DBPROCESS     *w_dbproc1, *w_dbproc2;
DBPROCESS     *c_dbproc1, *c_dbproc2;
DBPROCESS     *o_dbproc1, *o_dbproc2, *o_dbproc3;
ORDERS_STRUCT   orders_buf[ORDERS_PER_DISTRICT];
CUSTOMER_STRUCT customer_buf[CUSTOMERS_PER_DISTRICT];
long          main_threads_completed;
long          customer_threads_completed;
long          order_threads_completed;
long          orders_rows_loaded;
long          new_order_rows_loaded;
long          order_line_rows_loaded;
long          history_rows_loaded;
long          customer_rows_loaded;
long          stock_rows_loaded;
long          district_rows_loaded;
long          item_rows_loaded;
long          warehouse_rows_loaded;
long          main_time_start;
long          main_time_end;
TPCCLDR_ARGS    *aptr, args;


//==============================================================
==============
//
// Function name: main
//
//==============================================================
==============

int main(int  argc, char **argv)
{

                 DWORD
dwThreadID[MAX_MAIN_THREADS];
                 HANDLE          hThread[MAX_MAIN_THREADS];
                 FILE           *fLoader;
                 char            buffer[255];
                 int             main_threads_started;
                 RETCODE    retcode;
                 LOGINREC *login;

   printf("\n*********************************************************");
                 printf("\n*                                                *");
                 printf("\n*  Microsoft SQL Server 6.5
*");
                 printf("\n*                                                *");
                 printf("\n*  TPC-C BENCHMARK KIT:  Database
loader         *");
                 printf("\n*  Version %s                              *",
TPCKIT_VER);
                 printf("\n*                                                *");
                 printf("\n*********************************************************
****\n\n");

   // process command line arguments

   aptr = &args;
   GetArgsLoader(argc, argv, aptr);

                 if (aptr->build_index = 0)
                        printf("data load only\n");
                 if (aptr->build_index = 1)
                        printf("data load and index creation\n");

                 // install dblib error handlers

                 dbmsghandle((DBMSGHANDLE_PROC)SQLMs
gHandler);
                 dberrhandle((DBERRHANDLE_PROC)SQLErrHa
ndler);

                 // open connections to SQL Server

                 OpenConnections();

                 // open file for loader results
   fLoader = fopen(aptr->loader_res_file, "a");

   if (fLoader == NULL)
                 {
                        printf("Error, loader result file open
failed.");
                        exit(-1);
                 }

   // start loading data

   sprintf(buffer,"TPC-C load started for %ld warehouses:  ", aptr-
>num_warehouses);
                 if(aptr->build_index = 0)
                        strcat(buffer, "data load only\n");
                 if (aptr->build_index = 1)
                        strcat(buffer, "data load and index
creation\n");
```

```
                printf("%s",buffer);
                fprintf(fLoader,"%s",buffer);

                main_time_start = (TimeNow() / MILLI);

                // start parallel load threads

                main_threads_completed = 0;
                main_threads_started = 0;

                if ((aptr->table == NULL) || !(strcmp(aptr-
>table,"item")))
                {
                        fprintf(fLoader, "\nStarting loader threads
for: item\n");

                        hThread[0] = CreateThread(NULL,
0,
(LPTHREAD_START_ROUTINE) LoadItem,
NULL,
0,
&dwThreadID[0]);

                        if (hThread[0] == NULL)
                        {
                                printf("Error, failed in creating
creating thread = 0.\n");
                                exit(-1);
                        }

                        main_threads_started++;

                }

                if ((aptr->table == NULL) || !(strcmp(aptr-
>table,"warehouse")))
                {
                        fprintf(fLoader, "Starting loader threads
for: warehouse\n");

                        hThread[1] = CreateThread(NULL,
0,
(LPTHREAD_START_ROUTINE) LoadWarehouse,
NULL,
0,
&dwThreadID[1]);

                        if (hThread[1] == NULL)
                        {
                                printf("Error, failed in creating
creating thread = 1.\n");
                                exit(-1);
                        }

                        main_threads_started++;
                }

    if ((aptr->table == NULL) || !(strcmp(aptr->table,"customer")))
                        {
                                fprintf(fLoader, "Starting loader threads
for: customer\n");

                                hThread[2] = CreateThread(NULL,
0,
(LPTHREAD_START_ROUTINE) LoadCustomer,
NULL,
0,
&dwThreadID[2]);

                                if (hThread[2] == NULL)
                                {
                                        printf("Error, failed in creating
creating main thread = 2.\n");
                                        exit(-1);
                                }

                                main_threads_started++;
                        }

    if ((aptr->table == NULL) || !(strcmp(aptr->table,"orders")))
                        {
                                fprintf(fLoader, "Starting loader threads
for: orders\n");

                                hThread[3] = CreateThread(NULL,
0,
(LPTHREAD_START_ROUTINE) LoadOrders,
NULL,
0,
&dwThreadID[3]);

                                if (hThread[3] == NULL)
                                {
                                        printf("Error, failed in creating
creating main thread = 3.\n");
                                        exit(-1);
                                }

                        main_threads_started++;

                        }

                        while (main_threads_completed !=
main_threads_started)
                                Sleep(1000L);

    main_time_end = (TimeNow() / MILLI);

    sprintf(buffer,"\nTPC-C load completed successfully in %ld minutes.\n",
                                (main_time_end -
main_time_start)/60);

                        printf("%s",buffer);
                        fprintf(fLoader, "%s", buffer);

                        fclose(fLoader);

                        dbexit();

    exit(0);
}


//=======================================================
==============
//
// Function name: LoadItem
//
//=======================================================
==============

void LoadItem()
{
    long   i_id;
                        long   i_im_id;
    char   i_name[I_NAME_LEN+1];
    double i_price;
    char   i_data[I_DATA_LEN+1];
                        char   name[20];
                        long   time_start;

                        printf("\nLoading item table...\n");

                        // Seed with unique number
                        seed(1);

                        InitString(i_name, I_NAME_LEN+1);
                        InitString(i_data, I_DATA_LEN+1);

                        sprintf(name, "%s..%s", aptr->database, "item");
                        bcp_init(i_dbproc1, name, NULL, "logs\\item.err",
DB_IN);

                        bcp_bind(i_dbproc1, (BYTE *) &i_id,      0, -1,
NULL, 0, 0, 1);
                        bcp_bind(i_dbproc1, (BYTE *) &i_im_id,   0, -1,
NULL, 0, 0, 2);
                        bcp_bind(i_dbproc1, (BYTE *) i_name,     0,
I_NAME_LEN, NULL, 0, 0, 3);
                        bcp_bind(i_dbproc1, (BYTE *) &i_price,   0, -1,
NULL, 0, SQLFLT8, 4);
                        bcp_bind(i_dbproc1, (BYTE *) i_data,     0,
I_DATA_LEN, NULL, 0, 0, 5);

                        time_start = (TimeNow() / MILLI);

                        item_rows_loaded = 0;

                        for (i_id = 1; i_id <= MAXITEMS; i_id++)
                        {
                                i_im_id = RandomNumber(1L, 10000L);

                                MakeAlphaString(14, 24, I_NAME_LEN,
i_name);

                                i_price = ((float) RandomNumber(100L,
10000L))/100.0;

                                MakeOriginalAlphaString(26, 50,
I_DATA_LEN, i_data, 10);

                                if (!bcp_sendrow(i_dbproc1))
```

```
                                        printf("Error, LoadItem() failed
calling bcp_sendrow().  Check error file.\n");
                                item_rows_loaded++;
                                CheckForCommit(i_dbproc1,
item_rows_loaded, "item", &time_start);
                        }

                        bcp_done(i_dbproc1);
                        dbclose(i_dbproc1);

                        printf("Finished loading item table.\n");

                        if (aptr->build_index == 1)
                                BuildIndex("idxitmcl");

                        InterlockedIncrement(&main_threads_completed)
;
}


//=======================================================
==============
//
// Function   : LoadWarehouse
//
// Loads WAREHOUSE table and loads Stock and District as Warehouses
are created
//
//=======================================================
==============

void LoadWarehouse()
{
    short  w_id;
    char   w_name[W_NAME_LEN+1];
    char   w_street_1[ADDRESS_LEN+1];
    char   w_street_2[ADDRESS_LEN+1];
    char   w_city[ADDRESS_LEN+1];
    char   w_state[STATE_LEN+1];
    char   w_zip[ZIP_LEN+1];
    double w_tax;
    double w_ytd;
                        char   name[20];
                        long   time_start;


                        printf("\nLoading warehouse table...\n");

                        // Seed with unique number
                        seed(2);

                        InitString(w_name, W_NAME_LEN+1);
                        InitAddress(w_street_1, w_street_2, w_city,
w_state, w_zip);

                        sprintf(name, "%s..%s", aptr->database,
"warehouse");
                        bcp_init(w_dbproc1, name, NULL,
"logs\\whouse.err", DB_IN);

                        bcp_bind(w_dbproc1, (BYTE *) &w_id,        0, -1,
NULL, 0, 0, 1);
                        bcp_bind(w_dbproc1, (BYTE *) w_name,       0,
W_NAME_LEN,   NULL, 0, 0, 2);
                        bcp_bind(w_dbproc1, (BYTE *) w_street_1,   0,
ADDRESS_LEN,  NULL, 0, 0, 3);
                        bcp_bind(w_dbproc1, (BYTE *) w_street_2,   0,
ADDRESS_LEN,  NULL, 0, 0, 4);
                        bcp_bind(w_dbproc1, (BYTE *) w_city,       0,
ADDRESS_LEN,  NULL, 0, 0, 5);
                        bcp_bind(w_dbproc1, (BYTE *) w_state,      0,
STATE_LEN,    NULL, 0, 0, 6);
                        bcp_bind(w_dbproc1, (BYTE *) w_zip,        0,
ZIP_LEN,      NULL, 0, 0, 7);
                        bcp_bind(w_dbproc1, (BYTE *) &w_tax,       0, -
1,          NULL, 0, SQLFLT8, 8);
                        bcp_bind(w_dbproc1, (BYTE *) &w_ytd,       0, -
1,          NULL, 0, SQLFLT8, 9);

                        time_start = (TimeNow() / MILLI);

                        warehouse_rows_loaded = 0;

                        for (w_id = aptr->starting_warehouse; w_id <
aptr->num_warehouses+1; w_id++)
                        {

                                MakeAlphaString(6,10, W_NAME_LEN,
w_name);

                                MakeAddress(w_street_1, w_street_2,
w_city, w_state, w_zip);

                                w_tax = ((float)
RandomNumber(0L,2000L))/10000.00;

                                w_ytd = 300000.00;

                                if (!bcp_sendrow(w_dbproc1))
                                printf("Error, LoadWarehouse() failed
calling bcp_sendrow().  Check error file.\n");
                                warehouse_rows_loaded++;
                                CheckForCommit(i_dbproc1,
warehouse_rows_loaded, "warehouse", &time_start);
                        }

                        bcp_done(w_dbproc1);
                        dbclose(w_dbproc1);

                        printf("Finished loading warehouse table.\n");

                        if (aptr->build_index == 1)
                                BuildIndex("idxwarcl");

                        stock_rows_loaded = 0;
                        district_rows_loaded = 0;

                        District(w_id);
                        Stock(w_id);

                        InterlockedIncrement(&main_threads_completed)
;
}


//=======================================================
==============
//
// Function   : District
//
//=======================================================
==============

void District()
{
    short  d_id;
    short  d_w_id;
    char   d_name[D_NAME_LEN+1];
    char   d_street_1[ADDRESS_LEN+1];
    char   d_street_2[ADDRESS_LEN+1];
    char   d_city[ADDRESS_LEN+1];
    char   d_state[STATE_LEN+1];
    char   d_zip[ZIP_LEN+1];
    double d_tax;
    double d_ytd;
                        char   name[20];
    long   d_next_o_id;
    int    rc;
                        long   time_start;
                        int          w_id;

                        for (w_id = aptr->starting_warehouse; w_id <
aptr->num_warehouses+1; w_id++)
                        {

                                printf("...Loading district table: w_id =
%ld\n", w_id);

                                // Seed with unique number
                                seed(4);

                                InitString(d_name, D_NAME_LEN+1);

                                InitAddress(d_street_1, d_street_2, d_city,
d_state, d_zip);

                                sprintf(name, "%s..%s", aptr->database,
"district");
                                rc = bcp_init(w_dbproc2, name, NULL,
"logs\\district.err", DB_IN);

                                bcp_bind(w_dbproc2, (BYTE *) &d_id,
0, -1,          NULL, 0, 0, 1);
                                bcp_bind(w_dbproc2, (BYTE *) &d_w_id,
0, -1,          NULL, 0, 0, 2);
                                bcp_bind(w_dbproc2, (BYTE *) d_name,
0, D_NAME_LEN,    NULL, 0, 0, 3);
                                bcp_bind(w_dbproc2, (BYTE *)
d_street_1,     0, ADDRESS_LEN,   NULL, 0, 0, 4);
                                bcp_bind(w_dbproc2, (BYTE *)
d_street_2,     0, ADDRESS_LEN,   NULL, 0, 0, 5);
                                bcp_bind(w_dbproc2, (BYTE *) d_city,
0, ADDRESS_LEN,   NULL, 0, 0, 6);
                                bcp_bind(w_dbproc2, (BYTE *) d_state,
0, STATE_LEN,     NULL, 0, 0, 7);
                                bcp_bind(w_dbproc2, (BYTE *) d_zip,
0, ZIP_LEN,       NULL, 0, 0, 8);
                                bcp_bind(w_dbproc2, (BYTE *) &d_tax,
0, -1,          NULL, 0, SQLFLT8, 9);
                                bcp_bind(w_dbproc2, (BYTE *) &d_ytd,
0, -1,          NULL, 0, SQLFLT8, 10);
                                bcp_bind(w_dbproc2, (BYTE *)
&d_next_o_id,   0, -1,          NULL, 0, 0, 11);

                                d_w_id = w_id;

                                d_ytd  = 30000.0;

                                d_next_o_id = 3001L;

                                time_start = (TimeNow() / MILLI);

                                for (d_id = 1; d_id <=
DISTRICT_PER_WAREHOUSE; d_id++)
```

```
                {

            MakeAlphaString(6,10,D_NAME_LEN, d_name);

                MakeAddress(d_street_1,
d_street_2, d_city, d_state, d_zip);

                d_tax = ((float)
RandomNumber(0L,2000L))/10000.00;

                if (!bcp_sendrow(w_dbproc2))
                    printf("Error, District() failed
calling bcp_sendrow().  Check error file.\n");
                district_rows_loaded++;
                CheckForCommit(w_dbproc2,
district_rows_loaded, "district", &time_start);
                }

                rc = bcp_done(w_dbproc2);
            }

            printf("Finished loading district table.\n");

            if (aptr->build_index == 1)
                BuildIndex("idxdiscl");

    return;
}


//=======================================================
==============
//
// Function   : Stock
//
//=======================================================
==============

void Stock()
{
    long  s_i_id;
    short s_w_id;
    short s_quantity;
    char  s_dist_01[S_DIST_LEN+1];
    char  s_dist_02[S_DIST_LEN+1];
    char  s_dist_03[S_DIST_LEN+1];
    char  s_dist_04[S_DIST_LEN+1];
    char  s_dist_05[S_DIST_LEN+1];
    char  s_dist_06[S_DIST_LEN+1];
    char  s_dist_07[S_DIST_LEN+1];
    char  s_dist_08[S_DIST_LEN+1];
    char  s_dist_09[S_DIST_LEN+1];
    char  s_dist_10[S_DIST_LEN+1];
    long  s_ytd;
    short s_order_cnt;
    short s_remote_cnt;
    char  s_data[S_DATA_LEN+1];
    short i;
    short len;
    int   rc;
            char  name[20];
            long  time_start;

            // Seed with unique number
            seed(3);

            sprintf(name, "%s..%s", aptr->database, "stock");

            rc = bcp_init(w_dbproc2, name, NULL,
"logs\\stock.err", DB_IN);

            bcp_bind(w_dbproc2, (BYTE *) &s_i_id,      0, -1,
NULL, 0, 0, 1);
            bcp_bind(w_dbproc2, (BYTE *) &s_w_id,      0, -
1,      NULL, 0, 0, 2);
            bcp_bind(w_dbproc2, (BYTE *) &s_quantity,  0, -
1,      NULL, 0, 0, 3);
            bcp_bind(w_dbproc2, (BYTE *) s_dist_01,    0,
S_DIST_LEN, NULL, 0, 0, 4);
            bcp_bind(w_dbproc2, (BYTE *) s_dist_02,    0,
S_DIST_LEN, NULL, 0, 0, 5);
            bcp_bind(w_dbproc2, (BYTE *) s_dist_03,    0,
S_DIST_LEN, NULL, 0, 0, 6);
            bcp_bind(w_dbproc2, (BYTE *) s_dist_04,    0,
S_DIST_LEN, NULL, 0, 0, 7);
            bcp_bind(w_dbproc2, (BYTE *) s_dist_05,    0,
S_DIST_LEN, NULL, 0, 0, 8);
            bcp_bind(w_dbproc2, (BYTE *) s_dist_06,    0,
S_DIST_LEN, NULL, 0, 0, 9);
            bcp_bind(w_dbproc2, (BYTE *) s_dist_07,    0,
S_DIST_LEN, NULL, 0, 0, 10);
            bcp_bind(w_dbproc2, (BYTE *) s_dist_08,    0,
S_DIST_LEN, NULL, 0, 0, 11);
            bcp_bind(w_dbproc2, (BYTE *) s_dist_09,    0,
S_DIST_LEN, NULL, 0, 0, 12);
            bcp_bind(w_dbproc2, (BYTE *) s_dist_10,    0,
S_DIST_LEN, NULL, 0, 0, 13);
            bcp_bind(w_dbproc2, (BYTE *) &s_ytd,       0, -1,
NULL, 0, 0, 14);
            bcp_bind(w_dbproc2, (BYTE *) &s_order_cnt, 0,
-1,       NULL, 0, 0, 15);
            bcp_bind(w_dbproc2, (BYTE *) &s_remote_cnt,
0, -1,       NULL, 0, 0, 16);
            bcp_bind(w_dbproc2, (BYTE *) s_data,       0,
S_DATA_LEN,   NULL, 0, 0, 17);

            s_ytd = s_order_cnt = s_remote_cnt = 0;

            time_start = (TimeNow() / MILLI);

            printf("...Loading stock table\n");

            for (s_i_id=1; s_i_id <= MAXITEMS; s_i_id++)
            {

                for (s_w_id = aptr->starting_warehouse;
s_w_id < aptr->num_warehouses+1; s_w_id++)
                {

                    s_quantity =
RandomNumber(10L,100L);
                    len =
MakeAlphaString(24,24,S_DIST_LEN, s_dist_01);
                    len =
MakeAlphaString(24,24,S_DIST_LEN, s_dist_02);
                    len =
MakeAlphaString(24,24,S_DIST_LEN, s_dist_03);
                    len =
MakeAlphaString(24,24,S_DIST_LEN, s_dist_04);
                    len =
MakeAlphaString(24,24,S_DIST_LEN, s_dist_05);
                    len =
MakeAlphaString(24,24,S_DIST_LEN, s_dist_06);
                    len =
MakeAlphaString(24,24,S_DIST_LEN, s_dist_07);
                    len =
MakeAlphaString(24,24,S_DIST_LEN, s_dist_08);
                    len =
MakeAlphaString(24,24,S_DIST_LEN, s_dist_09);
                    len =
MakeAlphaString(24,24,S_DIST_LEN, s_dist_10);

                    len =
MakeOriginalAlphaString(26,50, S_DATA_LEN, s_data,10);

                    if (!bcp_sendrow(w_dbproc2))
                    printf("Error, Stock() failed calling
bcp_sendrow().  Check error file.\n");
                    stock_rows_loaded++;
                    CheckForCommit(w_dbproc2,
stock_rows_loaded, "stock", &time_start);

                }
            }

            bcp_done(w_dbproc2);
            dbclose(w_dbproc2);

            printf("Finished loading stock table.\n");

            if (aptr->build_index == 1)
                BuildIndex("idxstkcl");

            return;
}


//=======================================================
==============
//
// Function   : LoadCustomer
//
//=======================================================
==============

void LoadCustomer()
{
            LOADER_TIME_STRUCT
customer_time_start;
            LOADER_TIME_STRUCT     history_time_start;
            short                  w_id;
    short   d_id;
            DWORD
dwThreadID[MAX_CUSTOMER_THREADS];
            HANDLE
hThread[MAX_CUSTOMER_THREADS];
            char                   name[20];
            char                           buf[250];

            printf("\nLoading customer and history
tables...\n");

            // Seed with unique number
            seed(5);

            // Initialize bulk copy
            sprintf(name, "%s..%s", aptr->database,
"customer");
            bcp_init(c_dbproc1, name, NULL,
"logs\\customer.err", DB_IN);

            sprintf(name, "%s..%s", aptr->database,
"history");
```

```
                bcp_init(c_dbproc2, name, NULL,
"logs\\history.err", DB_IN);

                customer_rows_loaded    = 0;
                history_rows_loaded     = 0;

                CustomerBufInit();

                customer_time_start.time_start = (TimeNow() /
MILLI);
                history_time_start.time_start = (TimeNow() /
MILLI);

                for (w_id = aptr->starting_warehouse; w_id <=
aptr->num_warehouses; w_id++)
                {
                        for (d_id = 1L; d_id <=
DISTRICT_PER_WAREHOUSE; d_id++)
                        {

                                CustomerBufLoad(d_id, w_id);

                                // Start parallel loading threads
here...

                                customer_threads_completed=0;

                                // Start customer table thread

                                printf("...Loading customer table for:
d_id = %d, w_id = %d\n", d_id, w_id);

                                hThread[0] = CreateThread(NULL,
0,
(LPTHREAD_START_ROUTINE) LoadCustomerTable,
&customer_time_start,
0,
&dwThreadID[0]);

                                if (hThread[0] == NULL)
                                {
                                        printf("Error, failed in
creating creating thread = 0.\n");
                                        exit(-1);
                                }

                                // Start History table thread

                                printf("...Loading history table for:
d_id = %d, w_id = %d\n", d_id, w_id);

                                hThread[1] = CreateThread(NULL,
0,
(LPTHREAD_START_ROUTINE) LoadHistoryTable,
&history_time_start,
0,
&dwThreadID[1]);

                                if (hThread[1] == NULL)
                                {
                                        printf("Error, failed in
creating creating thread = 1.\n");
                                        exit(-1);
                                }

                                while
(customer_threads_completed != 2)
                                        Sleep(1000L);
                        }
                }

                // flush the bulk connection
                bcp_done(c_dbproc1);
                bcp_done(c_dbproc2);

                sprintf(buf,"update customer set c_first = 'C_LOAD = %d' where c_id = 1
and c_w_id = 1 and c_d_id = 1",LOADER_NURAND_C);
                dbcmd(c_dbproc1, buf);
                dbsqlexec(c_dbproc1);
                while (dbresults(c_dbproc1) !=
NO_MORE_RESULTS);

                dbclose(c_dbproc1);
                dbclose(c_dbproc2);

                printf("Finished loading customer table.\n");

                if (aptr->build_index == 1)
                        BuildIndex("idxcuscl");

                if (aptr->build_index == 1)
                        BuildIndex("idxcusnc");

                InterlockedIncrement(&main_threads_completed)
;

                return;

        }

        //=============================================================
        ==============

        void CustomerBufInit()
        {
                int     i;

                for (i=0;i<CUSTOMERS_PER_DISTRICT;i++)
                {
                        customer_buf[i].c_id = 0;
                        customer_buf[i].c_d_id = 0;
                        customer_buf[i].c_w_id = 0;

                        strcpy(customer_buf[i].c_first,"");
                        strcpy(customer_buf[i].c_middle,"");
                        strcpy(customer_buf[i].c_last,"");
                        strcpy(customer_buf[i].c_street_1,"");
                        strcpy(customer_buf[i].c_street_2,"");
                        strcpy(customer_buf[i].c_city,"");
                        strcpy(customer_buf[i].c_state,"");
                        strcpy(customer_buf[i].c_zip,"");
                        strcpy(customer_buf[i].c_phone,"");
                        strcpy(customer_buf[i].c_credit,"");

                        customer_buf[i].c_credit_lim = 0;
                        customer_buf[i].c_discount = (float) 0;
                        customer_buf[i].c_balance = 0;
                        customer_buf[i].c_ytd_payment = 0;
                        customer_buf[i].c_payment_cnt = 0;
                        customer_buf[i].c_delivery_cnt = 0;

                        strcpy(customer_buf[i].c_data_1,"");
                        strcpy(customer_buf[i].c_data_2,"");

                        customer_buf[i].h_amount = 0;

                        strcpy(customer_buf[i].h_data,"");
                }
        }

//=============================================================
==============
//
// Function   : CustomerBufLoad
//
// Fills shared buffer for HISTORY and CUSTOMER
//=============================================================
==============

void CustomerBufLoad(int d_id, int w_id)
{
                long                            i;
    CUSTOMER_SORT_STRUCT    c[CUSTOMERS_PER_DISTRICT];


                for (i=0;i<CUSTOMERS_PER_DISTRICT;i++)
                {
                        if (i < 1000)
                                LastName(i, c[i].c_last);
                        else

                        LastName(NURand(255,0,999,LOADER_NURA
ND_C), c[i].c_last);


                        MakeAlphaString(8,16,FIRST_NAME_LEN,
c[i].c_first);

                        c[i].c_id = i+1;

                }

                printf("...Loading customer buffer for: d_id = %d,
w_id = %d\n",
                        d_id, w_id);

                for (i=0;i<CUSTOMERS_PER_DISTRICT;i++)
                {

                        customer_buf[i].c_d_id = d_id;
                        customer_buf[i].c_w_id = w_id;
                        customer_buf[i].h_amount = 10.0;
                        customer_buf[i].c_ytd_payment  = 10.0;
                        customer_buf[i].c_payment_cnt  = 1;
                        customer_buf[i].c_delivery_cnt = 0;

                        // Generate CUSTOMER and HISTORY
data

                        customer_buf[i].c_id = c[i].c_id;

                        strcpy(customer_buf[i].c_first, c[i].c_first);
                        strcpy(customer_buf[i].c_last, c[i].c_last);

                        customer_buf[i].c_middle[0] = 'O';
```

```
                customer_buf[i].c_middle[1] = 'E';

                MakeAddress(customer_buf[i].c_street_1,
customer_buf[i].c_street_2,
                        customer_buf[i].c_city,
                        customer_buf[i].c_state,
                        customer_buf[i].c_zip);

                MakeNumberString(16, 16, PHONE_LEN,
customer_buf[i].c_phone);

                if (RandomNumber(1L, 100L) > 10)
                        customer_buf[i].c_credit[0] = 'G';
                else
                        customer_buf[i].c_credit[0] = 'B';
                customer_buf[i].c_credit[1] = 'C';

                customer_buf[i].c_credit_lim = 50000.0;
                customer_buf[i].c_discount = ((float)
RandomNumber(0L, 5000L)) / 10000.0;
                customer_buf[i].c_balance = -10.0;

                MakeAlphaString(250, 250,
C_DATA_LEN, customer_buf[i].c_data_1);
                MakeAlphaString(50, 250, C_DATA_LEN,
customer_buf[i].c_data_2);

                // Generate HISTORY data
                MakeAlphaString(12, 24, H_DATA_LEN,
customer_buf[i].h_data);

        }
}

//=======================================================
==============
//
// Function   : LoadCustomerTable
//
//=======================================================
==============

void LoadCustomerTable(LOADER_TIME_STRUCT *customer_time_start)
{
                int     i;
    long    c_id;
    short   c_d_id;
    short   c_w_id;
    char    c_first[FIRST_NAME_LEN+1];
    char    c_middle[MIDDLE_NAME_LEN+1];
    char    c_last[LAST_NAME_LEN+1];
    char    c_street_1[ADDRESS_LEN+1];
    char    c_street_2[ADDRESS_LEN+1];
    char    c_city[ADDRESS_LEN+1];
    char    c_state[STATE_LEN+1];
    char    c_zip[ZIP_LEN+1];
    char    c_phone[PHONE_LEN+1];
    char    c_credit[CREDIT_LEN+1];
    double  c_credit_lim;
    double  c_discount;
    double  c_balance;
    double  c_ytd_payment;
    short   c_payment_cnt;
    short   c_delivery_cnt;
    char    c_data_1[C_DATA_LEN+1];
    char    c_data_2[C_DATA_LEN+1];
                char    name[20];
                char    c_since[50];

    bcp_bind(c_dbproc1, (BYTE *) &c_id,         0, -1,          NULL,0,0, 1);
    bcp_bind(c_dbproc1, (BYTE *) &c_d_id,       0, -1,          NULL,0,0, 2);
    bcp_bind(c_dbproc1, (BYTE *) &c_w_id,       0, -1,          NULL,0,0, 3);
    bcp_bind(c_dbproc1, (BYTE *) c_first,       0, FIRST_NAME_LEN,
NULL,0,0, 4);
    bcp_bind(c_dbproc1, (BYTE *) c_middle,      0,
MIDDLE_NAME_LEN,NULL,0,0, 5);
    bcp_bind(c_dbproc1, (BYTE *) c_last,        0, LAST_NAME_LEN,
NULL,0,0, 6);
    bcp_bind(c_dbproc1, (BYTE *) c_street_1,    0, ADDRESS_LEN,
NULL,0,0, 7);
    bcp_bind(c_dbproc1, (BYTE *) c_street_2,    0, ADDRESS_LEN,
NULL,0,0, 8);
    bcp_bind(c_dbproc1, (BYTE *) c_city,        0, ADDRESS_LEN,
NULL,0,0, 9);
    bcp_bind(c_dbproc1, (BYTE *) c_state,       0, STATE_LEN,
NULL,0,0,10);
    bcp_bind(c_dbproc1, (BYTE *) c_zip,         0, ZIP_LEN,
NULL,0,0,11);
    bcp_bind(c_dbproc1, (BYTE *) c_phone,       0, PHONE_LEN,
NULL,0,0,12);
                bcp_bind(c_dbproc1, (BYTE *) c_since,       0,
50,         NULL,0,SQLCHAR,13);
    bcp_bind(c_dbproc1, (BYTE *) c_credit,      0, CREDIT_LEN,
NULL,0,0,14);
    bcp_bind(c_dbproc1, (BYTE *) &c_credit_lim, 0, -1,
NULL,0,SQLFLT8,15);
    bcp_bind(c_dbproc1, (BYTE *) &c_discount,   0, -1,
NULL,0,SQLFLT8,16);
    bcp_bind(c_dbproc1, (BYTE *) &c_balance,    0, -1,
NULL,0,SQLFLT8,17);
    bcp_bind(c_dbproc1, (BYTE *) &c_ytd_payment, 0, -1,
NULL,0,SQLFLT8,18);
    bcp_bind(c_dbproc1, (BYTE *) &c_payment_cnt, 0, -1,
NULL,0,0,19);
    bcp_bind(c_dbproc1, (BYTE *) &c_delivery_cnt,0, -1,
NULL,0,0,20);
    bcp_bind(c_dbproc1, (BYTE *) c_data_1,      0, C_DATA_LEN,
NULL,0,0,21);
    bcp_bind(c_dbproc1, (BYTE *) c_data_2,      0, C_DATA_LEN,
NULL,0,0,22);

    for (i = 0; i < CUSTOMERS_PER_DISTRICT; i++)
    {
                c_id = customer_buf[i].c_id;
                c_d_id = customer_buf[i].c_d_id;
                c_w_id = customer_buf[i].c_w_id;

                strcpy(c_first, customer_buf[i].c_first);
                strcpy(c_middle,
customer_buf[i].c_middle);
                strcpy(c_last, customer_buf[i].c_last);
                strcpy(c_street_1,
customer_buf[i].c_street_1);
                strcpy(c_street_2,
customer_buf[i].c_street_2);
                strcpy(c_city, customer_buf[i].c_city);
                strcpy(c_state, customer_buf[i].c_state);
                strcpy(c_zip, customer_buf[i].c_zip);
                strcpy(c_phone,
customer_buf[i].c_phone);
                strcpy(c_credit, customer_buf[i].c_credit);

                CurrentDate(&c_since);
                c_credit_lim =
customer_buf[i].c_credit_lim;
                c_discount = customer_buf[i].c_discount;
                c_balance = customer_buf[i].c_balance;
                c_ytd_payment =
customer_buf[i].c_ytd_payment;
                c_payment_cnt =
customer_buf[i].c_payment_cnt;
                c_delivery_cnt =
customer_buf[i].c_delivery_cnt;

                strcpy(c_data_1,
customer_buf[i].c_data_1);
                strcpy(c_data_2,
customer_buf[i].c_data_2);

                // Send data to server
                if (!bcp_sendrow(c_dbproc1))
                printf("Error, LoadCustomerTable() failed
calling bcp_sendrow().  Check error file.\n");
                customer_rows_loaded++;
                CheckForCommit(c_dbproc1,
customer_rows_loaded, "customer", &customer_time_start->time_start);
    }

                InterlockedIncrement(&customer_threads_compl
eted);

}


//=======================================================
==============
//
// Function   : LoadHistoryTable
//
//=======================================================
==============

void LoadHistoryTable(LOADER_TIME_STRUCT *history_time_start)
{
                int     i;
    long    c_id;
    short   c_d_id;
    short   c_w_id;
                double  h_amount;
    char    h_data[H_DATA_LEN+1];
                char    h_date[50];

    bcp_bind(c_dbproc2, (BYTE *) &c_id,         0, -1,          NULL, 0, 0, 1);
    bcp_bind(c_dbproc2, (BYTE *) &c_d_id,       0, -1,          NULL, 0, 0, 2);
    bcp_bind(c_dbproc2, (BYTE *) &c_w_id,       0, -1,          NULL, 0, 0, 3);
    bcp_bind(c_dbproc2, (BYTE *) &c_d_id,       0, -1,          NULL, 0, 0, 4);
    bcp_bind(c_dbproc2, (BYTE *) &c_w_id,       0, -1,          NULL, 0, 0, 5);
    bcp_bind(c_dbproc2, (BYTE *) h_date,        0, 50,          NULL, 0,
SQLCHAR, 6);
    bcp_bind(c_dbproc2, (BYTE *) &h_amount,     0, -1,          NULL, 0,
SQLFLT8, 7);
    bcp_bind(c_dbproc2, (BYTE *) h_data,        0, H_DATA_LEN,   NULL, 0,
0, 8);

    for (i = 0; i < CUSTOMERS_PER_DISTRICT; i++)
    {
                c_id = customer_buf[i].c_id;
                c_d_id = customer_buf[i].c_d_id;
                c_w_id = customer_buf[i].c_w_id;
                h_amount = customer_buf[i].h_amount;
                strcpy(h_data, customer_buf[i].h_data);
```

```
                                CurrentDate(&h_date);

                                // send to server
                                if (!bcp_sendrow(c_dbproc2))
                                printf("Error, LoadHistoryTable() failed
calling bcp_sendrow().  Check error file.\n");
                                history_rows_loaded++;
                                CheckForCommit(c_dbproc2,
history_rows_loaded, "history", &history_time_start->time_start);
   }

                        InterlockedIncrement(&customer_threads_compl
eted);

}


//=====================================================
===-================
//
// Function   : LoadOrders
//
//=====================================================
==================

void LoadOrders()
{
                        LOADER_TIME_STRUCT      orders_time_start;
                        LOADER_TIME_STRUCT
new_order_time_start;
                        LOADER_TIME_STRUCT
order_line_time_start;
                        short                   w_id;
   short                d_id;
                        DWORD
                        dwThreadID[MAX_ORDER_THREADS];
                        HANDLE
                        hThread[MAX_ORDER_THREADS];
                        char                    name[20];


                        printf("\nLoading orders...\n");

                        // seed with unique number
                        seed(6);

                        // initialize bulk copy
                        sprintf(name, "%s..%s", aptr->database,
"orders");
                        bcp_init(o_dbproc1, name, NULL,
"logs\\orders.err", DB_IN);

                        sprintf(name, "%s..%s", aptr->database,
"new_order");
                        bcp_init(o_dbproc2, name, NULL,
"logs\\neword.err", DB_IN);

                        sprintf(name, "%s..%s", aptr->database,
"order_line");
                        bcp_init(o_dbproc3, name, NULL,
"logs\\ordline.err", DB_IN);

                        orders_rows_loaded      = 0;
                        new_order_rows_loaded   = 0;
                        order_line_rows_loaded  = 0;

                        OrdersBufInit();

                        orders_time_start.time_start = (TimeNow() /
MILLI);
                        new_order_time_start.time_start = (TimeNow() /
MILLI);
                        order_line_time_start.time_start = (TimeNow() /
MILLI);

                        for (w_id = aptr->starting_warehouse; w_id <=
aptr->num_warehouses; w_id++)
                        {
                                for (d_id = 1L; d_id <=
DISTRICT_PER_WAREHOUSE; d_id++)
                                {

                                        OrdersBufLoad(d_id, w_id);

                                        // start parallel loading threads
here...

                                        order_threads_completed=0;

                                        // start Orders table thread

                                        printf("...Loading Order Table for:
d_id = %d, w_id = %d\n", d_id, w_id);

                                        hThread[0] = CreateThread(NULL,
0,
(LPTHREAD_START_ROUTINE) LoadOrdersTable,
&orders_time_start,
0,
&dwThreadID[0]);

                                        if (hThread[0] == NULL)
                                        {
                                                printf("Error, failed in
creating creating thread = 0.\n");
                                                exit(-1);
                                        }

                                        // start NewOrder table thread

                                        printf("...Loading New-Order Table
for: d_id = %d, w_id = %d\n", d_id, w_id);

                                        hThread[1] = CreateThread(NULL,
0,
(LPTHREAD_START_ROUTINE) LoadNewOrdersTable,
&new_order_time_start,
0,
&dwThreadID[1]);

                                        if (hThread[1] == NULL)
                                        {
                                                printf("Error, failed in
creating creating thread = 1.\n");
                                                exit(-1);
                                        }

                                        // start Order-Line table thread

                                        printf("...Loading Order-Line Table
for: d_id = %d, w_id = %d\n", d_id, w_id);

                                        hThread[2] = CreateThread(NULL,
0,
                                                                                                &dwThread
                                        if (hThread[2] == NULL)
                                        {
                                                printf("Error, failed in
creating creating thread = 2.\n");
                                                exit(-1);
                                        }

                                        while (order_threads_completed !=
3)
                                                Sleep(1000L);

                                }
                        }

                        printf("Finished loading orders.\n");

                        InterlockedIncrement(&main_threads_completed)
;

   return;
}

        //=====================================================
        ==============
        // Function : OrdersBufInit
        //
        // Clear shared buffer for ORDERS, NEWORDER, and ORDERLINE
        //
        //=====================================================
        ==============

        void OrdersBufInit()
        {
                int     i;
                int             j;

                for (i=0;i<ORDERS_PER_DISTRICT;i++)
                {
                        orders_buf[i].o_id = 0;
                        orders_buf[i].o_d_id = 0;
                        orders_buf[i].o_w_id = 0;
                        orders_buf[i].o_c_id = 0;
                        orders_buf[i].o_carrier_id = 0;
                        orders_buf[i].o_ol_cnt = 0;
                        orders_buf[i].o_all_local = 0;

                        for (j=0;j<=14;j++)
                        {
                                orders_buf[i].o_ol[j].ol = 0;
                                orders_buf[i].o_ol[j].ol_i_id = 0;
                                orders_buf[i].o_ol[j].ol_supply_w_id
        = 0;
                                orders_buf[i].o_ol[j].ol_quantity = 0;
                                orders_buf[i].o_ol[j].ol_amount = 0;

                        strcpy(orders_buf[i].o_ol[j].ol_dist_info,"");
                        }
                }

        }
0,
(LPTHREAD_START_ROUTINE) LoadOrderLineTable,
&order_line_time_start,
```

```
//=========================================================
==============
//
// Function   : OrdersBufLoad
//
// Fills shared buffer for ORDERS, NEWORDER, and ORDERLINE
//
//=========================================================
==============

void OrdersBufLoad(int d_id, int w_id)
{

                int     cust[ORDERS_PER_DIST+1];
                long    o_id;
                short   ol;

                printf("...Loading Order Buffer for: d_id = %d,
w_id = %d\n",
                        d_id, w_id);

                GetPermutation(cust, ORDERS_PER_DIST);

                for
(o_id=0;o_id<ORDERS_PER_DISTRICT;o_id++)
                {

                        // Generate ORDER and NEW-ORDER
data

                        orders_buf[o_id].o_d_id = d_id;
                        orders_buf[o_id].o_w_id = w_id;
                        orders_buf[o_id].o_id = o_id+1;
                        orders_buf[o_id].o_c_id = cust[o_id+1];
                        orders_buf[o_id].o_ol_cnt =
RandomNumber(5L, 15L);

                        if (o_id < 2100)
                        {
                                orders_buf[o_id].o_carrier_id =
RandomNumber(1L, 10L);
                                orders_buf[o_id].o_all_local  = 1;
                        }
                        else
                        {
                                orders_buf[o_id].o_carrier_id = 0;
                                orders_buf[o_id].o_all_local  = 1;
                        }

                        for
(ol=0;ol<orders_buf[o_id].o_ol_cnt;ol++)
                        {

                                orders_buf[o_id].o_ol[ol].ol = ol+1;
                                orders_buf[o_id].o_ol[ol].ol_i_id =
RandomNumber(1L, MAXITEMS);

                        orders_buf[o_id].o_ol[ol].ol_supply_w_id = w_id;

                        orders_buf[o_id].o_ol[ol].ol_quantity = 5;
                                MakeAlphaString(24, 24,
OL_DIST_INFO_LEN, &orders_buf[o_id].o_ol[ol].ol_dist_info);

                                // Generate ORDER-LINE data
                                if (o_id < 2100)
                                {

                        orders_buf[o_id].o_ol[ol].ol_amount = 0;
                                        // Added to insure
ol_delivery_d set properly during load

                                CurrentDate(&orders_buf[o_id].o_ol[ol].ol_deliver
y_d);
                                }
                                else
                                {

                                orders_buf[o_id].o_ol[ol].ol_amount =
RandomNumber(1,999999)/100.0;
                                        // Added to insure
ol_delivery_d set properly during load

                                strcpy(orders_buf[o_id].o_ol[ol].ol_delivery_d,"D
ec 31, 1889");
                                }
                        }
                }
}


//=========================================================
==============
//
// Function   : LoadOrdersTable
//
//=========================================================
==============

void LoadOrdersTable(LOADER_TIME_STRUCT *orders_time_start)
{
                int     i;
    long        o_id;
                short           o_d_id;
                short           o_w_id;
    long        o_c_id;
    short       o_carrier_id;
    short       o_ol_cnt;
    short       o_all_local;
                char            o_entry_d[50];

                // bind ORDER data
    bcp_bind(o_dbproc1, (BYTE *) &o_id,         0, -1,         NULL, 0, 0, 1);
    bcp_bind(o_dbproc1, (BYTE *) &o_d_id,       0, -1,         NULL, 0, 0, 2);
    bcp_bind(o_dbproc1, (BYTE *) &o_w_id,       0, -1,         NULL, 0, 0, 3);
    bcp_bind(o_dbproc1, (BYTE *) &o_c_id,       0, -1,         NULL, 0, 0, 4);
    bcp_bind(o_dbproc1, (BYTE *) o_entry_d,     0, 50,         NULL, 0,
SQLCHAR, 5);
    bcp_bind(o_dbproc1, (BYTE *) &o_carrier_id,  0, -1,        NULL, 0, 0, 6);
    bcp_bind(o_dbproc1, (BYTE *) &o_ol_cnt,     0, -1,         NULL, 0, 0, 7);
    bcp_bind(o_dbproc1, (BYTE *) &o_all_local,   0, -1,        NULL, 0, 0, 8);

    for (i = 0; i < ORDERS_PER_DISTRICT; i++)
    {
                        o_id        = orders_buf[i].o_id;
                        o_d_id      = orders_buf[i].o_d_id;
                        o_w_id      = orders_buf[i].o_w_id;
                        o_c_id      = orders_buf[i].o_c_id;
                        o_carrier_id = orders_buf[i].o_carrier_id;
                        o_ol_cnt    = orders_buf[i].o_ol_cnt;
                        o_all_local  = orders_buf[i].o_all_local;
                        CurrentDate(&o_entry_d);

                        // send data to server
                        if (!bcp_sendrow(o_dbproc1))
                        printf("Error, LoadOrdersTable() failed
calling bcp_sendrow().  Check error file.\n");
                        orders_rows_loaded++;
                        CheckForCommit(o_dbproc1,
orders_rows_loaded, "ORDERS", &orders_time_start->time_start);
    }

                if ((o_w_id == aptr->num_warehouses) &&
(o_d_id == 10))
                {
                        bcp_done(o_dbproc1);
                        dbclose(o_dbproc1);

                        if (aptr->build_index == 1)
                                BuildIndex("idxordcl");
                }

                InterlockedIncrement(&order_threads_completed
);

}


//=========================================================
==============
//
// Function   : LoadNewOrderTable
//
//=========================================================
==============

void LoadNewOrderTable(LOADER_TIME_STRUCT
*new_order_time_start)
{
    int         i;
    long        o_id;
    short       o_d_id;
    short       o_w_id;

                // Bind NEW-ORDER data
    bcp_bind(o_dbproc2, (BYTE *) &o_id,         0, -1, NULL, 0, 0, 1);
    bcp_bind(o_dbproc2, (BYTE *) &o_d_id,       0, -1, NULL, 0, 0, 2);
    bcp_bind(o_dbproc2, (BYTE *) &o_w_id,       0, -1, NULL, 0, 0, 3);

                for (i = 2100; i < 3000; i++)
                {
                        o_id    = orders_buf[i].o_id;
                        o_d_id  = orders_buf[i].o_d_id;
                        o_w_id  = orders_buf[i].o_w_id;

                        if (!bcp_sendrow(o_dbproc2))
                        printf("Error, LoadNewOrderTable() failed
calling bcp_sendrow().  Check error file.\n");
                        new_order_rows_loaded++;
                        CheckForCommit(o_dbproc2,
new_order_rows_loaded, "NEW_ORDER", &new_order_time_start-
>time_start);

                }

                if ((o_w_id == aptr->num_warehouses) &&
(o_d_id == 10))
                {
                        bcp_done(o_dbproc2);
                        dbclose(o_dbproc2);

                        if (aptr->build_index == 1)
                                BuildIndex("idxnodcl");
                }
```

```
                        InterlockedIncrement(&order_threads_completed
);

}


//=======================================================
==============
//
// Function   : LoadOrderLineTable
//
//=======================================================
==============

void LoadOrderLineTable(LOADER_TIME_STRUCT *order_line_time_start)
{
                        int         i,j;
   long       o_id;
                        short       o_d_id;
                        short       o_w_id;
   long       ol;
                        long        ol_i_id;
   short      ol_supply_w_id;
   short      ol_quantity;
   double      ol_amount;
   short      o_all_local;
   char       ol_dist_info[DIST_INFO_LEN+1];
                        char        ol_delivery_d[50];

                        // bind ORDER-LINE data
   bcp_bind(o_dbproc3, (BYTE *) &o_id,              0, -1, NULL, 0, 0, 1);
   bcp_bind(o_dbproc3, (BYTE *) &o_d_id,            0, -1, NULL, 0, 0, 2);
   bcp_bind(o_dbproc3, (BYTE *) &o_w_id,            0, -1, NULL, 0, 0, 3);
   bcp_bind(o_dbproc3, (BYTE *) &ol,                0, -1, NULL, 0, 0, 4);
   bcp_bind(o_dbproc3, (BYTE *) &ol_i_id,           0, -1, NULL, 0, 0, 5);
   bcp_bind(o_dbproc3, (BYTE *) &ol_supply_w_id,    0, -1, NULL, 0, 0,
6);
                        bcp_bind(o_dbproc3, (BYTE *) ol_delivery_d,
   bcp_bind(o_dbproc3, (BYTE *) &ol_quantity,       0, -1, NULL, 0, 0, 8);
   bcp_bind(o_dbproc3, (BYTE *) &ol_amount,         0, -1, NULL, 0,
SQLFLT8, 9);
   bcp_bind(o_dbproc3, (BYTE *) ol_dist_info,       0, DIST_INFO_LEN,
NULL, 0, 0, 10);

                        for (i = 0; i < ORDERS_PER_DISTRICT; i++)
   {
                                o_id    = orders_buf[i].o_id;
                                o_d_id  = orders_buf[i].o_d_id;
                                o_w_id  = orders_buf[i].o_w_id;

                                for (j=0; j < orders_buf[i].o_ol_cnt; j++)
                                {
                                        ol          = orders_buf[i].o_ol[j].ol;
                                        ol_i_id     =
orders_buf[i].o_ol[j].ol_i_id;
                                        ol_supply_w_id =
orders_buf[i].o_ol[j].ol_supply_w_id;
                                        ol_quantity    =
orders_buf[i].o_ol[j].ol_quantity;
                                        ol_amount      =
orders_buf[i].o_ol[j].ol_amount;
                                        //  Changed to insure ol_delivery_d
set properly (now set in OrdersBufLoad)
                                        //  CurrentDate(&ol_delivery_d);

                                strcpy(ol_delivery_d,orders_buf[i].o_ol[j].ol_deliv
ery_d);
                                strcpy(ol_dist_info,orders_buf[i].o_ol[j].ol_dist_inf
o);

                                if (!bcp_sendrow(o_dbproc3))
                                        printf("Error,
LoadOrderLineTable() failed calling bcp_sendrow().  Check error file.\n");
                                order_line_rows_loaded++;
                                CheckForCommit(o_dbproc3,
order_line_rows_loaded, "ORDER_LINE", &order_line_time_start-
>time_start);
                                }

                        }

                        if ((o_w_id == aptr->num_warehouses) &&
(o_d_id == 10))
                        {
                                bcp_done(o_dbproc3);
                                dbclose(o_dbproc3);

                                if (aptr->build_index == 1)
                                        BuildIndex("idxodlcl");
                        }

                        InterlockedIncrement(&order_threads_completed
);

}


//=======================================================
==============
//
// Function   : GetPermutation
//
//=======================================================
0, 50, NULL, 0, SQLCHAR, 7);

   void GetPermutation(int perm[], int n)
   {
                        int i, r, t;

                        for (i=1;i<=n;i++)
                                perm[i] = i;

                        for (i=1;i<=n;i++)
                        {
                                r = RandomNumber(i,n);
                                t = perm[i];
                                perm[i] = perm[r];
                                perm[r] = t;
                        }

   }


//=======================================================
==============
//
// Function   : CheckForCommit
//
//=======================================================
==============

void CheckForCommit(DBPROCESS *dbproc,
                                        int rows_loaded,
                                                char *table_name,
                                                        long *time_start)
{
   long                 time_end, time_diff;

   // commit every "batch" rows

   if ( !(rows_loaded % aptr->batch) )
   {
                                bcp_batch(dbproc);

                                time_end = (TimeNow() / MILLI);
                                time_diff = time_end - *time_start;

                                printf("-> Loaded %ld rows into %s in %ld
sec - Total = %d (%.2f rps)\n",
                                        aptr->batch,
                                        table_name,
                                                time_diff,
                                        rows_loaded,
                                                (float) aptr->batch / (time_diff
? time_diff : 1L));

                                *time_start = time_end;
   }

   return;
}


//=======================================================
==============
//
// Function   : OpenConnections
//
//=======================================================
==============

void OpenConnections()
{
                        RETCODE    retcode;

                        LOGINREC *login;

   login = dblogin();

   retcode = DBSETLUSER(login, aptr->user);
                        if (retcode == FAIL)
                        {
                                printf("DBSETLUSER failed.\n");
                        }
                        retcode = DBSETLPWD(login, aptr->password);
                        if (retcode == FAIL)
                        {
                                printf("DBSETLPWD failed.\n");
                        }

                        retcode = DBSETLPACKET(login, (USHORT)
aptr->pack_size);
                        if (retcode == FAIL)
                        {
                                printf("DBSETLPACKET failed.\n");
                        }

                        printf("DB-Library packet size: %ld\n",aptr-
>pack_size);

   // turn connection into a BCP connection
```

```
    retcode = BCP_SETL(login, TRUE);
                                if (retcode == FAIL)
                                {
                                        printf("BCP_SETL failed.\n");
                                }

    // open connections to SQL Server */

    if ((i_dbproc1 = dbopen(login, aptr->server)) == NULL)
    {
                                printf("Error on login 1 to server %s.\n",
aptr->server);
                                exit(-1);
    }

    if ((w_dbproc1 = dbopen(login, aptr->server)) == NULL)
    {
                                printf("Error on login 2 to server %s.\n",
aptr->server);
                                exit(-1);
    }

    if ((w_dbproc2 = dbopen(login, aptr->server)) == NULL)
    {
                                printf("Error on login 3 to server %s.\n",
aptr->server);
                                exit(-1);
    }

    if ((c_dbproc1 = dbopen(login, aptr->server)) == NULL)
    {
                                printf("Error on login 4 to server %s.\n",
aptr->server);
                                exit(-1);
    }

    if ((c_dbproc2 = dbopen(login, aptr->server)) == NULL)
    {
                                printf("Error on login 5 to server %s.\n",
aptr->server);
                                exit(-1);
    }

    if ((o_dbproc1 = dbopen(login, aptr->server)) == NULL)
    {
                                printf("Error on login 6 to server %s.\n",
aptr->server);
                                exit(-1);
    }

    if ((o_dbproc2 = dbopen(login, aptr->server)) == NULL)
    {
                                printf("Error on login 7 to server %s.\n",
aptr->server);
                                exit(-1);
    }

    if ((o_dbproc3 = dbopen(login, aptr->server)) == NULL)
    {
                                printf("Error on login 8 to server %s.\n",
aptr->server);
                                exit(-1);
    }

}
```

```
//=====================================================
==============
//
// Function name: SQLErrHandler
//
//=====================================================
==============

int SQLErrHandler(SQLCONN *dbproc,
                                                int     severity,
                                                int     err,
                                                int     oserr,
                                                char    *dberrstr,
                                                char    *oserrstr)
{
                                char msg[256];
                                FILE *fp1;
                                char timebuf[128];
                                char datebuf[128];

                                _strtime(timebuf);
                                _strdate(datebuf);

                                sprintf(msg, "%s %s : DBLibrary (%ld) %s\n",
datebuf, timebuf, err, dberrstr);
    printf("%s",msg);

                                fp1 = fopen("logs\tpccldr.err","a");
                                if (fp1 == NULL)
                                {
                                        printf("Error in opening errorlog file.\n");
                                }
                                else
                                {
                                        fprintf(fp1, msg);
                                        fclose(fp1);
                                }

    if (oserr != DBNOERR)
    {
                                        sprintf(msg, "%s %s : OSError (%ld)
%s\n", datebuf, timebuf, oserr, oserrstr);
                                        printf("%s",msg);

                                        fp1 = fopen("logs\\tpccldr.err","a");
                                        if (fp1 == NULL)
                                        {
                                                printf("Error in opening errorlog
file.\n");
                                        }
                                        else
                                        {
                                                fprintf(fp1, msg);
                                                fclose(fp1);
                                        }
    }

                                if ((dbproc == NULL) || (DBDEAD(dbproc)))
                                {
                                        exit(-1);
                                }

                                return (INT_CANCEL);

}
```

```
//=====================================================
==============
//
// Function name: SQLMsgHandler
//
//=====================================================
==============

int SQLMsgHandler(SQLCONN   *dbproc,
                                                DBINT     msgno,
                                                int       msgstate,
                                                int       severity,
                                                char      *msgtext)
{
    char msg[256];
                                FILE   *fp1;
                                char timebuf[128];
                                char datebuf[128];

    if ( (msgno == 5701) || (msgno == 2528) || (msgno == 5703) || (msgno ==
6006) )
                                {
                                        return(INT_CONTINUE);
                                }

                                if (msgno == 0)
                                {
                                        return(INT_CONTINUE);
                                }
                                else
                                {
                                        _strtime(timebuf);
                                        _strdate(datebuf);

                                        sprintf(msg, "%s %s : SQLServer (%ld)
%s\n", datebuf, timebuf, msgno, msgtext);

                                        printf("%s",msg);

                                        fp1 = fopen("logs\\tpccldr.err","a");
                                        if (fp1 == NULL)
                                        {
                                                printf("Error in opening errorlog
file.\n");
                                        }
                                        else
                                        {
                                                fprintf(fp1, msg);
                                                fclose(fp1);
                                        }

                                        exit(-1);
                                }

                                return (INT_CANCEL);
}



//=====================================================
==============
//
// Function name: CurrentDate
//
//=====================================================
==============

void CurrentDate(char       *datetime)
```

```
{

                        char timebuf[128];
                        char datebuf[128];

                        _strtime(timebuf);
                        _strdate(datebuf);

                        sprintf(datetime, "%s %s", datebuf, timebuf);

}


//=====================================================
=============
//
// Function name: BuildIndex
//
//=====================================================
=============

void BuildIndex(char        *index_script)
{
                        char    cmd[256];

                        printf("Starting index creation:
%s\n",index_script);

                        sprintf(cmd, "isql -S%s -U%s -P%s -e -
i%s\\%s.sql >> logs\\%s.out",
                                        aptr->server,
                                        aptr->user,
                                        aptr->password,
                                        aptr->index_script_path,
                                        index_script,
                                        index_script);

                        system(cmd);

                        printf("Finished index creation:
%s\n",index_script);
}
```

UTIL.C

```
// TPC-C Benchmark Kit
//
// Module:  UTIL.C
// Author:  DamienL


// Includes
#include "tpcc.h"


//=====================================================
=============
//
// Function name: UtilSleep
//
//=====================================================
=============

void UtilSleep(long  delay)
{

#ifdef DEBUG
   printf("[%ld]DBG: Entering UtilSleep()\n", (int) GetCurrentThreadId());
#endif

#ifdef DEBUG
   printf("[%ld]DBG: Sleeping for %ld seconds...\n", (int)
GetCurrentThreadId(), delay);
#endif

   Sleep(delay * 1000);

}


//=====================================================
=============
//
// Function name: UtilSleep
//
//=====================================================
=============

void UtilSleepMs(long  delay)
{

#ifdef DEBUG
   printf("[%ld]DBG: Entering UtilSleepMs()\n", (int) GetCurrentThreadId());
#endif

#ifdef DEBUG
   printf("[%ld]DBG: Sleeping for %ld milliseconds...\n", (int)
GetCurrentThreadId(), delay);
#endif

   Sleep(delay);

}

//=====================================================
=============
//
// Function name: UtilPrintNewOrder
//
//=====================================================
=============
void UtilPrintNewOrder(NEW_ORDER_DATA *pNewOrder)
{
   int i;

#ifdef DEBUG
   printf("[%ld]DBG: Entering UtilPrintNewOrder()\n", (int)
GetCurrentThreadId());
#endif

                        EnterCriticalSection(&ConsoleCritSec);

                        printf("\n[%04ld]\tNewOrder Transaction\n\n",
(int) GetCurrentThreadId());

   printf("Warehouse: %ld\n"
                                "District: %ld\n"
                                "Date: %02ld/%02ld/%04ld
%02ld:%02ld:%02ld\n\n"
                                "Customer Number:  %ld\n"
                                "Customer Name: %s\n"
                                "Customer Credit: %s\n"
                                "Cusotmer Discount: %02.2f%%\n\n"
                                "Order Number: %ld\n"
                                "Warehouse Tax: %02.2f%%\n"
                                "District Tax: %02.2f%%\n\n"
                                "Number of Order Lines: %ld\n\n",
                                        (int)    pNewOrder->w_id,
                                        (int)    pNewOrder->d_id,
                                        (char *) pNewOrder-
>o_entry_d.month,
                                        (char *) pNewOrder-
>o_entry_d.day,
                                        (char *) pNewOrder-
>o_entry_d.year,
                                        (char *) pNewOrder-
>o_entry_d.hour,
                                        (char *) pNewOrder-
>o_entry_d.minute,
                                        (char *) pNewOrder-
>o_entry_d.second,
                                        (int)    pNewOrder->c_id,
                                        (char *) pNewOrder->c_last,
                                        (char *) pNewOrder->c_credit,
                                        (float)  pNewOrder->c_discount,
                                        (int)    pNewOrder->o_id,
                                        (float)  pNewOrder->w_tax,
                                        (float)  pNewOrder->d_tax,
                                        (int)    pNewOrder->o_ol_cnt);

                        printf("Supp_W Item_Id Item Name
Qty Stock B/G Price    Amount  \n");
                        printf("------ ------- ------------------------- --- ----- --- -
------- --------\n");

   for (i=0;i < pNewOrder->o_ol_cnt;i++)
   {
                                printf("%04ld   %06ld  %24s   %02ld
%03ld   %1s  %8.2f %9.2f\n",
                                (int)    pNewOrder->Ol[i].ol_supply_w_id,
                                (int)    pNewOrder->Ol[i].ol_i_id,
                                        (char *) pNewOrder-
>Ol[i].ol_i_name,
                                (int)    pNewOrder->Ol[i].ol_quantity,
                                (int)    pNewOrder->Ol[i].ol_stock,
                                (char *) pNewOrder-
>Ol[i].ol_brand_generic,
                                (float)  pNewOrder->Ol[i].ol_i_price,
                                (float)  pNewOrder->Ol[i].ol_amount);
   }

                        printf("\nTotal: $%05.2f\n\n",
                                        (float)  pNewOrder->total_amount);

                        printf("Execution Status: %s\n\n",
                                        (char *) pNewOrder-
>execution_status);

                        LeaveCriticalSection(&ConsoleCritSec);

}


//=====================================================
=============
//
// Function name: UtilPrintPayment
//
//=====================================================
=============
```

```
void UtilPrintPayment(PAYMENT_DATA *pPayment)
{

                        char    tmp_data[201];
   char    data_line_1[51];
                        char    data_line_2[51];
                        char    data_line_3[51];
                        char    data_line_4[51];

#ifdef DEBUG
   printf("[%ld]DBG: Entering UtilPrintPayment()\n", (int)
GetCurrentThreadId());
#endif

   EnterCriticalSection(&ConsoleCritSec);

                        printf("\n[%04ld]\tPayment Transaction\n\n", (int)
GetCurrentThreadId());

   printf("Date: %02ld/%02ld/%04ld %02ld:%02ld:%02ld\n\n",
                            (int)   pPayment->h_date.month,
                                    (int)   pPayment->h_date.day,
                            (int)   pPayment->h_date.year,
                                    (int)   pPayment->h_date.hour,
                                    (int)   pPayment->h_date.minute,
                                    (int)   pPayment->h_date.second);

                        printf("Warehouse: %ld\n"
                            "District: %ld\n\n",
                                    (int)   pPayment->w_id,
                                    (int)   pPayment->d_id);

                        printf("Warehouse Address Street 1: %s\n"
                            "Warehouse Address Street 2: %s\n",
                                    (char *) pPayment->w_street_1,
                                    (char *) pPayment->w_street_2);

                        printf("Warehouse Address City: %s\n"
                           "Warehouse Address State: %s\n"
                           "Warehouse Address Zip: %s\n\n",
                                    (char *) pPayment->w_city,
                                    (char *) pPayment->w_state,
                                    (char *) pPayment->w_zip);

                        printf("District Address Street 1: %s\n"
                            "District Address Street 2: %s\n",
                                    (char *) pPayment->d_street_1,
                                 (char *) pPayment->d_street_2);

                        printf("District Address City: %s\n"
                           "District Address State: %s\n"
                           "District Address Zip: %s\n\n",
                                 (char *) pPayment->d_city,
                                    (char *) pPayment->d_state,
                                    (char *) pPayment->d_zip);

                        printf("Customer Number: %ld\n"
                            "Customer Warehouse: %ld\n"
                            "Customer District: %ld\n",
                                    (int)   pPayment->c_id,
                                    (int)   pPayment->c_w_id,
                                    (int)   pPayment->c_d_id);

                        printf("Customer Name:   %s %s %s\n"
                            "Customer Since:  %02ld-%02ld-
%04ld\n",
                                    (char *) pPayment->c_first,
                                    (char *) pPayment->c_middle,
                                    (char *) pPayment->c_last,
                                    (int)   pPayment->c_since.month,
                                    (int)   pPayment->c_since.day,
                                    (int)   pPayment->c_since.year);

                        printf("Customer Address Street 1: %s\n"
                           "Customer Address Street 2: %s\n"
                           "Customer Address City: %s\n"
                           "Customer Address State: %s\n"
                           "Customer Address Zip: %s\n"
                                "Customer Phone Number: %s\n\n"
                           "Customer Credit: %s\n"
                           "Customer Discount: %02.2f%%\n",
                                    (char *) pPayment->c_street_1,
                                    (char *) pPayment->c_street_2,
                                    (char *) pPayment->c_city,
                                    (char *) pPayment->c_state,
                                    (char *) pPayment->c_zip,
                                    (char *) pPayment->c_phone,
                                    (char *) pPayment->c_credit,
                                    (double) pPayment->c_discount);

                        printf("Amount Paid: $%04.2f\n"
                           "New Customer Balance: $%10.2f\n",
                                    (float)  pPayment->h_amount,
                                    (double) pPayment->c_balance);

   printf("Credit Limit: $%10.2f\n\n",
                                    (double) pPayment->c_credit_lim);

   if (strcmp(pPayment->c_data," ") != 0)
                        {
                                strcpy(tmp_data, pPayment->c_data);
                                strncpy(data_line_1, tmp_data, 50);
data_line_1[50] = '\0';
                                strncpy(data_line_2, &tmp_data[50], 50);
data_line_2[50] = '\0';
                                strncpy(data_line_3, &tmp_data[100], 50);
data_line_3[50] = '\0';
                                strncpy(data_line_4, &tmp_data[150], 50);
data_line_4[50] = '\0';

                        }
                        else
                        {
                                strcpy(data_line_1, " ");
strcpy(data_line_2, " ");
                                strcpy(data_line_3, " ");
strcpy(data_line_4, " ");
                        }

                        printf("                ----------------------------------------
--------\n");
   printf("Customer Data: |%50s|\n", data_line_1);
                        printf("               |%50s|\n", data_line_2);
                        printf("               |%50s|\n", data_line_3);
                        printf("               |%50s|\n", data_line_4);
                        printf("                ----------------------------------------
--------\n\n");

                        printf("Execution Status: %s\n\n",
                                    (char *) pPayment->execution_status);

                        LeaveCriticalSection(&ConsoleCritSec);
}

//=========================================================
==============
//
// Function name: UtilPrintOrderStatus
//
//=========================================================
==============

void UtilPrintOrderStatus(ORDER_STATUS_DATA *pOrderStatus)
{
   int i;

#ifdef DEBUG
   printf("[%ld]DBG: Entering UtilPrintOrderStatus()\n", (int)
GetCurrentThreadId());
#endif

                        EnterCriticalSection(&ConsoleCritSec);

                        printf("\n[%04ld]\tOrder-Status Transaction\n\n",
(int) GetCurrentThreadId());

                        printf("Warehouse: %ld\n"
                            "District: %ld\n\n",
                                    (int) pOrderStatus->w_id,
                                    (int) pOrderStatus->d_id);

                        printf("Customer Number: %ld\n"
                            "Customer Name: %s %s %s\n\n",
                                    (int) pOrderStatus->c_id,
                                    (char *) pOrderStatus->c_first,
                                    (char *) pOrderStatus->c_middle,
                                    (char *) pOrderStatus->c_last);

                        printf("Customer Balance: $%5.2f\n\n",
                                    (double) pOrderStatus-
>c_balance);

                        printf("Order Number: %ld\n"
                            "Entry Date: %02ld/%02ld/%04ld
%02ld:%02ld:%02ld\n"
                            "Carrier Number: %ld\n\n"
                            "Number of order lines: %ld\n\n",
                                    (int) pOrderStatus->o_id,
                                    (int) pOrderStatus-
>o_entry_d.month,
                                    (int) pOrderStatus->o_entry_d.day,
                                    (int) pOrderStatus->o_entry_d.year,
                                    (int) pOrderStatus->o_entry_d.hour,
                                    (int) pOrderStatus-
>o_entry_d.minute,
                                    (int) pOrderStatus-
>o_entry_d.second,
                                    (int) pOrderStatus->o_carrier_id,
                                    (int) pOrderStatus->o_ol_cnt);

   printf ("Supply-W    Item-Id    Delivery-Date    Qty   Amount    \n");
                                    printf ("--------  -------  -------------  ---  ----------
\n");

   for (i=0;i < pOrderStatus->o_ol_cnt; i++)
   {
                                        printf("%04ld       %06ld
%02ld/%02ld/%04ld      %02ld    %9.2f\n",
                                                (int) pOrderStatus-
>OlOrderStatusData[i].ol_supply_w_id,
```

```
                                        (int) pOrderStatus-
>OlOrderStatusData[i].ol_i_id,
                                        (int) pOrderStatus-
>OlOrderStatusData[i].ol_delivery_d.month,
                                        (int) pOrderStatus-
>OlOrderStatusData[i].ol_delivery_d.day,
                                        (int) pOrderStatus-
>OlOrderStatusData[i].ol_delivery_d.year,
                                        (int) pOrderStatus-
>OlOrderStatusData[i].ol_quantity,
                                        (double) pOrderStatus-
>OlOrderStatusData[i].ol_amount);
    }

                        if (pOrderStatus->o_ol_cnt == 0)
                                printf("\nNo Order-Status items.\n\n");

                        printf("\nExecution Status: %s\n\n",
                                (char *) pOrderStatus-
>execution_status);

                        LeaveCriticalSection(&ConsoleCritSec);

}


//=======================================================
==============
//
// Function name: UtilPrintDelivery
//
//=======================================================
==============

void UtilPrintDelivery(DELIVERY_DATA *pQueuedDelivery)
{

#ifdef DEBUG
    printf("[%ld]DBG: Entering UtilPrintDelivery()\n", (int)
GetCurrentThreadId());
#endif

                        EnterCriticalSection(&ConsoleCritSec);

                        printf("\n[%04ld]\tDelivery Transaction\n\n", (int)
GetCurrentThreadId());

    printf("Warehouse: %ld\n", (int) pQueuedDelivery->w_id);

                        printf("Carrier Number: %ld\n\n", (int)
pQueuedDelivery->o_carrier_id);

    printf("Execution Status: %s\n\n", (char *) pQueuedDelivery-
>execution_status);

                        LeaveCriticalSection(&ConsoleCritSec);

}


//=======================================================
==============
//
// Function name: UtilPrintStockLevel
//
//=======================================================
==============

void UtilPrintStockLevel(STOCK_LEVEL_DATA *pStockLevel)
{

#ifdef DEBUG
    printf("[%ld]DBG: Entering UtilPrintStockLevel()\n", (int)
GetCurrentThreadId());
#endif

                        EnterCriticalSection(&ConsoleCritSec);

                        printf("\n[%04ld]\tStock-Level Transaction\n\n",
(int) GetCurrentThreadId());

                        printf("Warehouse: %ld\nDistrict: %ld\n",
                                (int) pStockLevel->w_id,
                                (int) pStockLevel->d_id);

                        printf("Stock Level Threshold: %ld\n\n", (int)
pStockLevel->thresh_hold);

                        printf("Low Stock Count: %ld\n\n", (int)
pStockLevel->low_stock);

                        printf("Execution Status: %s\n\n", (char *)
pStockLevel->execution_status);

                        LeaveCriticalSection(&ConsoleCritSec);
}

//=======================================================
==============
//
// Function name: UtilError
//
//=======================================================
==============
void UtilError(long threadid, char * header, char *msg)
{

#ifdef DEBUG
    printf("[%ld]DBG: Entering UtilError()\n", (int) GetCurrentThreadId());
#endif

    printf("[%ld] %s: %s\n", (int) threadid, header, msg);
}


//=======================================================
==============
//
// Function name: UtilFatalError
//
//=======================================================
==============

void UtilFatalError(long threadid, char * header, char *msg)
{

#ifdef DEBUG
    printf("[%ld]DBG: Entering UtilFatalError()\n", (int) GetCurrentThreadId());
#endif

    printf("[Thread: %ld]... %s: %s\n", (int) threadid, header, msg);
    exit(-1);
}

//=======================================================
==============
//
// Function name: UtilStrCpy
//
//=======================================================
============
void UtilStrCpy(char * pDest, char * pSrc, int n)
{

#ifdef DEBUG
    printf("[%ld]DBG: Entering UtilStrCpy()\n", (int) GetCurrentThreadId());
#endif

    strncpy(pDest, pSrc, n);
    pDest[n] = '\0';
}


#ifdef USE_CONMON
//=======================================================
==============
//
// Function name: WriteConsoleString
//
//=======================================================
============
void WriteConsoleString(HANDLE hConMon, char *str, short x, short y,
short color, BOOL pad)
{
                        COORD   dwWriteCoord = {0, 0};
                        DWORD   cCharsWritten;
                        LPVOID  dummy;
                        int     len, i;

#ifdef DEBUG
    printf("[%ld]DBG: Entering WriteConsoleString()\n", (int)
GetCurrentThreadId());
#endif

                        dwWriteCoord.X = x;
                        dwWriteCoord.Y = y;

                        if (pad)
                        {
                                len = strlen(str);
                                if (len < CON_LINE_SIZE)
                                {
                                        for(i=1;i<CON_LINE_SIZE-len;i++)
                                        {
                                                strcat(str," ");
                                        }
                                }
                        }

                        EnterCriticalSection(&ConsoleCritSec);

                        switch (color)
    {
                                case YELLOW:
                                        SetConsoleTextAttribute(hConMon,
                                                FOREGROUND_INTENSITY
| FOREGROUND_GREEN | FOREGROUND_RED |
BACKGROUND_BLUE);
                                        break;

                                case RED:
                                        SetConsoleTextAttribute(hConMon,
```

```
                                        FOREGROUND_INTENSITY
| FOREGROUND_RED | BACKGROUND_BLUE);
                                break;

                                case GREEN:
                                        SetConsoleTextAttribute(hConMon,
                                                FOREGROUND_INTENSITY
| FOREGROUND_GREEN | BACKGROUND_BLUE);
                                        break;
   }

                        SetConsoleCursorPosition(hConMon,
dwWriteCoord);
                        WriteConsole(hConMon, str, strlen(str),
&cCharsWritten, dummy);

                        LeaveCriticalSection(&ConsoleCritSec);

}
#endif


//=====================================================
==============
//
// Function name: AddDeliveryQueueNode
//
//=====================================================
==============

BOOL AddDeliveryQueueNode(DELIVERY_PTR node_to_add)
{

                        DELIVERY_PTR    local_node;
#ifdef DEBUG
                        DELIVERY_PTR    ptrtmp;
                        short           i;
#endif

                        EnterCriticalSection(&QueuedDeliveryCritSec);

                        if ((local_node = malloc(sizeof(struct
delivery_node)) ) == NULL)
                        {
                                printf("ERROR:  problem allocating
memory for delivery queue.\n");
                                exit(-1);
                        }
                        else
                        {
                                memcpy(local_node, node_to_add, sizeof
(struct delivery_node));

                                if (queued_delivery_cnt == 0)
                                {
                                        delivery_head = local_node;
                                        delivery_head->next_delivery =
NULL;
                                        delivery_tail = delivery_head;
                                }
                                else
                                {
                                        local_node->next_delivery = NULL;
                                        delivery_tail->next_delivery =
local_node;
                                        delivery_tail = local_node;
                                }
                        }
                                queued_delivery_cnt++;
#ifdef DEBUG
                                i=0;
                                printf("Add to delivery list:
%ld\n",queued_delivery_cnt);
                                ptrtmp=delivery_head;
                                while (ptrtmp != NULL)
                                {
                                        i++;
                                        printf("%ld - w_id %ld - o_carrier_id %ld -
queue_time %d/%d/%d %d:%d:%d:%d\n",
                                                i, ptrtmp->w_id, ptrtmp-
>o_carrier_id,
                                                ptrtmp-
>queue_time.wMonth,
                                                ptrtmp->queue_time.wDay,
                                                ptrtmp->queue_time.wYear,
                                                ptrtmp->queue_time.wHour,
                                                ptrtmp-
>queue_time.wMinute,
                                                ptrtmp-
>queue_time.wSecond,
                                                ptrtmp-
>queue_time.wMilliseconds);

                                        ptrtmp=ptrtmp->next_delivery;
                                }
#endif

                                LeaveCriticalSection(&QueuedDeliveryCritSec);

   return TRUE;

}


//=====================================================
==============
//
// Function name: GetDeliveryQueueNode
//
//=====================================================
==============

BOOL GetDeliveryQueueNode(DELIVERY_PTR node_to_get)
{

                        DELIVERY_PTR            local_node;
                        BOOL                    rc;
#ifdef DEBUG
                        DELIVERY_PTR            ptrtmp;
                        short                   i;
#endif

                        EnterCriticalSection(&QueuedDeliveryCritSec);

                        if (queued_delivery_cnt == 0)
                        {

#ifdef DEBUG
                                printf("No delivery nodes found.\n");
#endif
                                rc = FALSE;
                        }
                        else
                        {
                                memcpy(node_to_get, delivery_head,
sizeof(struct delivery_node));

                                if (queued_delivery_cnt == 1)
                                {
                                        free(delivery_head);
                                        delivery_head = NULL;
                                        queued_delivery_cnt = 0;
                                }
                                else
                                {
                                        local_node = delivery_head;
                                        delivery_head = delivery_head-
>next_delivery;
                                        free(local_node);
                                        queued_delivery_cnt--;
                                }
#ifdef DEBUG
                                i=0;
                                printf("Get from delivery list:
%ld\n",queued_delivery_cnt);
                                ptrtmp=delivery_head;
                                while (ptrtmp != NULL)
                                {
                                        i++;
                                        printf("%ld - w_id %ld - o_carrier_id
%ld - queue_time %d/%d/%d %d:%d:%d:%d\n",
                                                i, ptrtmp->w_id,
ptrtmp->o_carrier_id,
                                                ptrtmp-
>queue_time.wMonth,
                                                ptrtmp-
>queue_time.wDay,
                                                ptrtmp-
>queue_time.wYear,
                                                ptrtmp-
>queue_time.wHour,
                                                ptrtmp-
>queue_time.wMinute,
                                                ptrtmp-
>queue_time.wSecond,
                                                ptrtmp-
>queue_time.wMilliseconds);

                                        ptrtmp=ptrtmp->next_delivery;
                                }
#endif

                                rc = TRUE;

                        }

                        LeaveCriticalSection(&QueuedDelivery CritSec);

   return rc;

}


//=====================================================
==============
//
// Function name: WriteDeliveryString
//
//=====================================================
==============
```

```
void WriteDeliveryString(char   buf[255])
{

                        DWORD   bytesWritten;
                        DWORD   retCode;

#ifdef DEBUG
   printf("[%ld]DBG: Entering UtilDeliveryMsg()\n", (int)
GetCurrentThreadId());
#endif

                        EnterCriticalSection(&WriteDeliveryCritSec);

   retCode = WriteFile (hDeliveryMonPipe, buf, PLEASE_WRITE,
                                &bytesWritten, NULL);

                        LeaveCriticalSection(&WriteDeliveryCritSec);

}
```

Appendix C: Tunable Parameters

Microsoft Windows NT v4.0 Tunable Parameters:

System\CurrentControlSet\Control\SessionManager

Key Name: SYSTEM\CurrentControlSet\Control\Session Manager
Class Name: <NO CLASS>
Last Write Time: 7/3/97 - 11:34 AM
Value 0
Name: BootExecute
Type: REG_MULTI_SZ
Data: autocheck autochk *

Value 1
Name: CriticalSectionTimeout
Type: REG_DWORD
Data: 0x278d00

Value 2
Name: EnableMCA
Type: REG_DWORD
Data: 0x1

Value 3
Name: EnableMCE
Type: REG_DWORD
Data: 0

Value 4
Name: ExcludeFromKnownDlls
Type: REG_MULTI_SZ
Data:

Value 5
Name: GlobalFlag
Type: REG_DWORD
Data: 0

Value 6
Name: HeapDeCommitFreeBlockThreshold
Type: REG_DWORD
Data: 0

Value 7
Name: HeapDeCommitTotalFreeThreshold
Type: REG_DWORD
Data: 0

Value 8
Name: HeapSegmentCommit
Type: REG_DWORD
Data: 0

Value 9
Name: HeapSegmentReserve
Type: REG_DWORD
Data: 0

Value 10
Name: LicensedProcessors
Type: REG_DWORD
Data: 0x4

Value 11
Name: ObjectDirectories
Type: REG_MULTI_SZ
Data: \Windows
\RPC Control

Value 12
Name: ProcessorControl
Type: REG_DWORD
Data: 0x2

Value 13
Name: ProtectionMode
Type: REG_DWORD
Data: 0

Value 14
Name: RegisteredProcessors
Type: REG_DWORD
Data: 0x4

Value 15
Name: ResourceTimeoutCount
Type: REG_DWORD
Data: 0x9e340

Key Name: SYSTEM\CurrentControlSet\Control\Session Manager\DOS Devices
Class Name: <NO CLASS>
Last Write Time: 7/25/96 - 12:07 PM
Value 0
Name: AUX
Type: REG_SZ
Data: \DosDevices\COM1

Value 1
Name: MAILSLOT
Type: REG_SZ
Data: \Device\MailSlot

Value 2
Name: NUL
Type: REG_SZ
Data: \Device\Null

Value 3
Name: PIPE
Type: REG_SZ
Data: \Device\NamedPipe

Value 4
Name: PRN
Type: REG_SZ
Data: \DosDevices\LPT1

Value 5
Name: UNC
Type: REG_SZ
Data: \Device\Mup

Key Name: SYSTEM\CurrentControlSet\Control\Session Manager\Environment
Class Name: <NO CLASS>
Last Write Time: 7/16/97 - 10:04 AM
Value 0
Name: ComSpec
Type: REG_EXPAND_SZ
Data: %SystemRoot%\system32\cmd.exe

Value 1
Name: NUMBER_OF_PROCESSORS
Type: REG_SZ
Data: 2

Value 2
Name: OS
Type: REG_SZ
Data: Windows_NT

Value 3
Name: Os2LibPath
Type: REG_EXPAND_SZ
Data: %SystemRoot%\system32\os2\dll;

Value 4
Name: Path
Type: REG_EXPAND_SZ
Data: %SystemRoot%\system32;%SystemRoot%;;D:\MSSQL\BINN

Value 5
Name: PROCESSOR_ARCHITECTURE
Type: REG_SZ
Data: x86

Value 6
Name: PROCESSOR_IDENTIFIER
Type: REG_SZ
Data: x86 Family 6 Model 1 Stepping 9, GenuineIntel

Value 7
Name: PROCESSOR_LEVEL
Type: REG_SZ
Data: 6

Value 8
Name: PROCESSOR_REVISION
Type: REG_SZ
Data: 0109

Value 9
Name: windir
Type: REG_EXPAND_SZ
Data: %SystemRoot%

Key Name: SYSTEM\CurrentControlSet\Control\Session Manager\Executive
Class Name: <NO CLASS>
Last Write Time: 5/12/97 - 4:13 PM
Value 0
Name: AdditionalCriticalWorkerThreads
Type: REG_DWORD
Data: 0

Value 1
Name: AdditionalDelayedWorkerThreads
Type: REG_DWORD
Data: 0

Value 2

Name: PriorityQuantumMatrix
Type: REG_BINARY
Data:
00000000 c0 da 03 58 00 00 00 00 - 19 5f bc 01 ...X....._..

Key Name: SYSTEM\CurrentControlSet\Control\Session Manager\FileRenameOperations
Class Name: <NO CLASS>
Last Write Time: 7/25/96 - 12:07 PM

Key Name: SYSTEM\CurrentControlSet\Control\Session Manager\KnownDLLs
Class Name: <NO CLASS>
Last Write Time: 5/12/97 - 11:06 AM
Value 0
Name: advapi32
Type: REG_SZ
Data: advapi32.dll

Value 1
Name: comdlg32
Type: REG_SZ
Data: comdlg32.dll

Value 2
Name: crtdll
Type: REG_SZ
Data: crtdll.dll

Value 3
Name: DllDirectory
Type: REG_EXPAND_SZ
Data: %SystemRoot%\system32

Value 4
Name: gdi32
Type: REG_SZ
Data: gdi32.dll

Value 5
Name: kernel32
Type: REG_SZ
Data: kernel32.dll

Value 6
Name: lz32
Type: REG_SZ
Data: lz32.dll

Value 7
Name: ole32
Type: REG_SZ
Data: ole32.dll

Value 8
Name: oleaut32
Type: REG_SZ
Data: oleaut32.dll

Value 9
Name: olecli32
Type: REG_SZ
Data: olecli32.dll

Value 10
Name: olecnv32
Type: REG_SZ
Data: olecnv32.dll

Value 11
Name: olesvr32
Type: REG_SZ
Data: olesvr32.dll

Value 12
Name: olethk32
Type: REG_SZ
Data: olethk32.dll

Value 13
Name: rpcrt4
Type: REG_SZ
Data: rpcrt4.dll

Value 14
Name: shell32
Type: REG_SZ
Data: shell32.dll

Value 15
Name: user32
Type: REG_SZ
Data: user32.dll

Value 16
Name: version
Type: REG_SZ
Data: version.dll

Key Name: SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management
Class Name: <NO CLASS>
Last Write Time: 7/14/97 - 3:16 PM
Value 0
Name: ClearPageFileAtShutdown
Type: REG_DWORD
Data: 0

Value 1
Name: DisablePagingExecutive
Type: REG_DWORD
Data: 0

Value 2
Name: IoPageLockLimit
Type: REG_DWORD
Data: 0

Value 3
Name: LargeSystemCache
Type: REG_DWORD
Data: 0

Value 4
Name: NonPagedPoolQuota
Type: REG_DWORD
Data: 0

Value 5
Name: NonPagedPoolSize
Type: REG_DWORD
Data: 0

Value 6
Name: PagedPoolQuota
Type: REG_DWORD
Data: 0

Value 7
Name: PagedPoolSize
Type: REG_DWORD
Data: 0

Value 8
Name: PagingFiles
Type: REG_MULTI_SZ
Data: C:\pagefile.sys 819 819

Value 9
Name: SecondLevelDataCache
Type: REG_DWORD
Data: 0x1

Value 10
Name: SystemPages
Type: REG_DWORD
Data: 0

Key Name: SYSTEM\CurrentControlSet\Control\Session Manager\SubSystems
Class Name: <NO CLASS>
Last Write Time: 5/12/97 - 11:06 AM
Value 0
Name: Debug
Type: REG_EXPAND_SZ
Data:

Value 1
Name: Kmode
Type: REG_EXPAND_SZ
Data: %SystemRoot%\system32\win32k.sys

Value 2
Name: Optional
Type: REG_MULTI_SZ
Data: Os2
Posix

Value 3
Name: Os2
Type: REG_EXPAND_SZ
Data: %SystemRoot%\system32\os2ss.exe

Value 4
Name: Posix
Type: REG_EXPAND_SZ
Data: %SystemRoot%\system32\psxss.exe

Value 5
Name: Required
Type: REG_MULTI_SZ
Data: Debug
Windows

Value 6
Name: Windows
Type: REG_EXPAND_SZ

Data: %SystemRoot%\system32\csrss.exe
ObjectDirectory=\Windows SharedSection=1024,3072 Windows=On
SubSystemType=Windows ServerDll=basesrv,1
ServerDll=winsrv:UserServerDllInitialization,3
ServerDll=winsrv:ConServerDllInitialization,2 ProfileControl=Off
MaxRequestThreads=16

CurrentControlSet\Services\ InetInfo\Parameters

Key Name: SYSTEM\CurrentControlSet\Services\InetInfo
Class Name: <NO CLASS>
Last Write Time: 11/14/96 - 5:24 AM

Key Name:
SYSTEM\CurrentControlSet\Services\InetInfo\Parameters
Class Name: <NO CLASS>
Last Write Time: 7/2/97 - 1:24 PM
Value 0
Name: BandwidthLevel
Type: REG_DWORD
Data: 0xffffffff

Value 1
Name: ListenBackLog
Type: REG_DWORD
Data: 0x19

Value 2
Name: MaxPoolThreads
Type: REG_DWORD
Data: 0x14

Value 3
Name: MemoryCacheSize
Type: REG_DWORD
Data: 0x5dc000

Value 4
Name: PoolThreadsLilmit
Type: REG_DWORD
Data: 0x1fe

Value 5
Name: ThreadTimeout
Type: REG_DWORD
Data: 0x15180

Key Name:
SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\Filter
Class Name: <NO CLASS>
Last Write Time: 7/2/97 - 9:42 AM
Value 0
Name: FilterType
Type: REG_DWORD
Data: 0

Value 1
Name: NumDenySites
Type: REG_DWORD
Data: 0

Value 2
Name: NumGrantSites
Type: REG_DWORD
Data: 0

Key Name:
SYSTEM\CurrentControlSet\Services\InetInfo\Parameters\MimeMap
Class Name: <NO CLASS>
Last Write Time: 11/14/96 - 5:24 AM
Value 0
Name: application/envoy,evy,,5
Type: REG_SZ
Data:

Value 1
Name: application/mac-binhex40,hqx,,4
Type: REG_SZ
Data:

Value 2
Name: application/msword,doc,,5
Type: REG_SZ
Data:

Value 3
Name: application/msword,dot,,5
Type: REG_SZ
Data:

Value 4
Name: application/octet-stream,*,,5
Type: REG_SZ
Data:

Value 5
Name: application/octet-stream,bin,,5
Type: REG_SZ
Data:

Value 6
Name: application/octet-stream,exe,,5
Type: REG_SZ
Data:

Value 7
Name: application/oda,oda,,5
Type: REG_SZ
Data:

Value 8
Name: application/pdf,pdf,,5
Type: REG_SZ
Data:

Value 9
Name: application/postscript,ai,,5
Type: REG_SZ
Data:

Value 10
Name: application/postscript,eps,,5
Type: REG_SZ
Data:

Value 11
Name: application/postscript,ps,,5
Type: REG_SZ
Data:

Value 12
Name: application/rtf,rtf,,5
Type: REG_SZ
Data:

Value 13
Name: application/winhlp,hlp,,5
Type: REG_SZ
Data:

Value 14
Name: application/x-bcpio,bcpio,,5
Type: REG_SZ
Data:

Value 15
Name: application/x-cpio,cpio,,5
Type: REG_SZ
Data:

Value 16
Name: application/x-csh,csh,,5
Type: REG_SZ
Data:

Value 17
Name: application/x-director,dcr,,5
Type: REG_SZ
Data:

Value 18
Name: application/x-director,dir,,5
Type: REG_SZ
Data:

Value 19
Name: application/x-director,dxr,,5
Type: REG_SZ
Data:

Value 20
Name: application/x-dvi,dvi,,5
Type: REG_SZ
Data:

Value 21
Name: application/x-gtar,gtar,,9
Type: REG_SZ
Data:

Value 22
Name: application/x-hdf,hdf,,5
Type: REG_SZ
Data:

Value 23
Name: application/x-latex,latex,,5
Type: REG_SZ
Data:

Value 24
Name: application/x-msaccess,mdb,,5
Type: REG_SZ
Data:

Value 25
Name: application/x-mscardfile,crd,,5
Type: REG_SZ
Data:

Value 26
Name: application/x-msclip,clp,,5
Type: REG_SZ
Data:

Value 27
Name: application/x-msexcel,xla,,5
Type: REG_SZ
Data:

Value 28
Name: application/x-msexcel,xlc,,5
Type: REG_SZ
Data:

Value 29
Name: application/x-msexcel,xlm,,5
Type: REG_SZ
Data:

Value 30
Name: application/x-msexcel,xls,,5
Type: REG_SZ
Data:

Value 31
Name: application/x-msexcel,xlt,,5
Type: REG_SZ
Data:

Value 32
Name: application/x-msexcel,xlw,,5
Type: REG_SZ
Data:

Value 33
Name: application/x-msmediaview,m13,,5
Type: REG_SZ
Data:

Value 34
Name: application/x-msmediaview,m14,,5
Type: REG_SZ
Data:

Value 35
Name: application/x-msmetafile,wmf,,5
Type: REG_SZ
Data:

Value 36
Name: application/x-msmoney,mny,,5
Type: REG_SZ
Data:

Value 37
Name: application/x-mspowerpoint,ppt,,5
Type: REG_SZ
Data:

Value 38
Name: application/x-msproject,mpp,,5
Type: REG_SZ
Data:

Value 39
Name: application/x-mspublisher,pub,,5
Type: REG_SZ
Data:

Value 40
Name: application/x-msterminal,trm,,5
Type: REG_SZ
Data:

Value 41
Name: application/x-msworks,wks,,5
Type: REG_SZ
Data:

Value 42
Name: application/x-mswrite,wri,,5
Type: REG_SZ
Data:

Value 43
Name: application/x-netcdf,cdf,,5
Type: REG_SZ
Data:

Value 44
Name: application/x-netcdf,nc,,5
Type: REG_SZ
Data:

Value 45
Name: application/x-perfmon,pma,,5
Type: REG_SZ
Data:

Value 46
Name: application/x-perfmon,pmc,,5
Type: REG_SZ
Data:

Value 47
Name: application/x-perfmon,pml,,5
Type: REG_SZ
Data:

Value 48
Name: application/x-perfmon,pmr,,5
Type: REG_SZ
Data:

Value 49
Name: application/x-perfmon,pmw,,5
Type: REG_SZ
Data:

Value 50
Name: application/x-sh,sh,,5
Type: REG_SZ
Data:

Value 51
Name: application/x-shar,shar,,5
Type: REG_SZ
Data:

Value 52
Name: application/x-sv4cpio,sv4cpio,,5
Type: REG_SZ
Data:

Value 53
Name: application/x-sv4crc,sv4crc,,5
Type: REG_SZ
Data:

Value 54
Name: application/x-tar,tar,,5
Type: REG_SZ
Data:

Value 55
Name: application/x-tcl,tcl,,5
Type: REG_SZ
Data:

Value 56
Name: application/x-tex,tex,,5
Type: REG_SZ
Data:

Value 57
Name: application/x-texinfo,texi,,5
Type: REG_SZ
Data:

Value 58
Name: application/x-texinfo,texinfo,,5
Type: REG_SZ
Data:

Value 59
Name: application/x-troff,roff,,5
Type: REG_SZ
Data:

Value 60
Name: application/x-troff,t,,5
Type: REG_SZ
Data:

Value 61
Name: application/x-troff,tr,,5
Type: REG_SZ
Data:

Value 62
Name: application/x-troff-man,man,,5
Type: REG_SZ
Data:

Value 63
Name: application/x-troff-me,me,,5
Type: REG_SZ
Data:

Value 64
Name: application/x-troff-ms,ms,,5
Type: REG_SZ
Data:

Value 65
Name: application/x-ustar,ustar,,5
Type: REG_SZ
Data:

Value 66
Name: application/x-wais-source,src,,7
Type: REG_SZ
Data:

Value 67
Name: application/zip,zip,,9
Type: REG_SZ
Data:

Value 68
Name: audio/basic,au,,<
Type: REG_SZ
Data:

Value 69
Name: audio/basic,snd,,<
Type: REG_SZ
Data:

Value 70
Name: audio/x-aiff,aif,,<
Type: REG_SZ
Data:

Value 71
Name: audio/x-aiff,aifc,,<
Type: REG_SZ
Data:

Value 72
Name: audio/x-aiff,aiff,,<
Type: REG_SZ
Data:

Value 73
Name: audio/x-pn-realaudio,ram,,<
Type: REG_SZ
Data:

Value 74
Name: audio/x-wav,wav,,<
Type: REG_SZ
Data:

Value 75
Name: image/bmp,bmp,,:
Type: REG_SZ
Data:

Value 76
Name: image/cis-cod,cod,,5
Type: REG_SZ
Data:

Value 77
Name: image/gif,gif,,g
Type: REG_SZ
Data:

Value 78
Name: image/ief,ief,,:
Type: REG_SZ
Data:

Value 79
Name: image/jpeg,jpe,,:
Type: REG_SZ
Data:

Value 80
Name: image/jpeg,jpeg,,:
Type: REG_SZ
Data:

Value 81
Name: image/jpeg,jpg,,:
Type: REG_SZ
Data:

Value 82
Name: image/tiff,tif,,:
Type: REG_SZ
Data:

Value 83
Name: image/tiff,tiff,,:
Type: REG_SZ
Data:

Value 84
Name: image/x-cmu-raster,ras,,:
Type: REG_SZ
Data:

Value 85
Name: image/x-cmx,cmx,,5
Type: REG_SZ
Data:

Value 86
Name: image/x-portable-anymap,pnm,,:
Type: REG_SZ
Data:

Value 87
Name: image/x-portable-bitmap,pbm,,:
Type: REG_SZ
Data:

Value 88
Name: image/x-portable-graymap,pgm,,:
Type: REG_SZ
Data:

Value 89
Name: image/x-portable-pixmap,ppm,,:
Type: REG_SZ

Data:

Value 90
Name: image/x-rgb,rgb,,:
Type: REG_SZ
Data:

Value 91
Name: image/x-xbitmap,xbm,,:
Type: REG_SZ
Data:

Value 92
Name: image/x-xpixmap,xpm,,:
Type: REG_SZ
Data:

Value 93
Name: image/x-xwindowdump,xwd,,:
Type: REG_SZ
Data:

Value 94
Name: text/html,htm,,h
Type: REG_SZ
Data:

Value 95
Name: text/html,html,,h
Type: REG_SZ
Data:

Value 96
Name: text/html,stm,,h
Type: REG_SZ
Data:

Value 97
Name: text/plain,bas,,0
Type: REG_SZ
Data:

Value 98
Name: text/plain,c,,0
Type: REG_SZ
Data:

Value 99
Name: text/plain,h,,0
Type: REG_SZ
Data:

Value 100
Name: text/plain,txt,,0
Type: REG_SZ
Data:

Value 101
Name: text/richtext,rtx,,0
Type: REG_SZ
Data:

Value 102
Name: text/tab-separated-values,tsv,,0
Type: REG_SZ
Data:

Value 103
Name: text/x-setext,etx,,0
Type: REG_SZ
Data:

Value 104
Name: video/mpeg,mpe,,;
Type: REG_SZ
Data:

Value 105
Name: video/mpeg,mpeg,,;
Type: REG_SZ
Data:

Value 106
Name: video/mpeg,mpg,,;
Type: REG_SZ
Data:

Value 107
Name: video/quicktime,mov,,;
Type: REG_SZ
Data:

Value 108
Name: video/quicktime,qt,,;
Type: REG_SZ
Data:

Value 109
Name: video/x-msvideo,avi,,<
Type: REG_SZ
Data:

Value 110
Name: video/x-sgi-movie,movie,,<
Type: REG_SZ
Data:

Value 111
Name: x-world/x-vrml,flr,,5
Type: REG_SZ
Data:

Value 112
Name: x-world/x-vrml,wrl,,5
Type: REG_SZ
Data:

Value 113
Name: x-world/x-vrml,wrz,,5
Type: REG_SZ
Data:

Value 114
Name: x-world/x-vrml,xaf,,5
Type: REG_SZ
Data:

Value 115
Name: x-world/x-vrml,xof,,5
Type: REG_SZ
Data:

Key Name:
SYSTEM\CurrentControlSet\Services\InetInfo\Performance
Class Name: <NO CLASS>
Last Write Time: 7/2/97 - 9:42 AM
Value 0
Name: Close
Type: REG_SZ
Data: CloseINFOPerformanceData

Value 1
Name: Collect
Type: REG_SZ
Data: CollectINFOPerformanceData

Value 2
Name: First Counter
Type: REG_DWORD
Data: 0x738

Value 3
Name: First Help
Type: REG_DWORD
Data: 0x739

Value 4
Name: Last Counter
Type: REG_DWORD
Data: 0x756

Value 5
Name: Last Help
Type: REG_DWORD
Data: 0x757

Value 6
Name: Library
Type: REG_SZ
Data: infoctrs.DLL

Value 7
Name: Open

Type: REG_SZ
Data: OpenINFOPerformanceData

CurrentControlSet\Services\W3SVC

Key Name: SYSTEM\CurrentControlSet\Services\W3SVC
Class Name: <NO CLASS>
Last Write Time: 12/16/96 - 9:21 AM
Value 0
Name: DependOnGroup
Type: REG_MULTI_SZ
Data:

Value 1
Name: DependOnService
Type: REG_MULTI_SZ
Data: RPCSS
NTLMSSP

Value 2
Name: DisplayName
Type: REG_SZ
Data: World Wide Web Publishing Service

Value 3
Name: ErrorControl
Type: REG_DWORD
Data: 0

Value 4
Name: ImagePath
Type: REG_EXPAND_SZ
Data: C:\WINNT\System32\inetsrv\inetinfo.exe

Value 5
Name: ObjectName
Type: REG_SZ
Data: LocalSystem

Value 6
Name: Start
Type: REG_DWORD
Data: 0x2

Value 7
Name: Type
Type: REG_DWORD
Data: 0x20

Key Name: SYSTEM\CurrentControlSet\Services\W3SVC\Enum
Class Name: <NO CLASS>
Last Write Time: 3/3/97 - 2:16 AM
Value 0
Name: 0
Type: REG_SZ
Data: Root\LEGACY_W3SVC\0000

Value 1
Name: Count
Type: REG_DWORD
Data: 0x1

Value 2
Name: NextInstance
Type: REG_DWORD
Data: 0x1

Key Name:
SYSTEM\CurrentControlSet\Services\W3SVC\Parameters
Class Name: <NO CLASS>
Last Write Time: 2/11/97 - 2:32 PM
Value 0
Name: AccessDeniedMessage
Type: REG_SZ
Data: Error: Access is Denied.

Value 1
Name: AdminEmail
Type: REG_SZ
Data: Admin@corp.com

Value 2
Name: AdminName
Type: REG_SZ
Data: Administrator

Value 3
Name: AnonymousUserName
Type: REG_SZ
Data: Administrator

Value 4
Name: Authorization
Type: REG_DWORD
Data: 0x5

Value 5
Name: CacheExtensions
Type: REG_DWORD
Data: 0x1

Value 6
Name: CheckForWAISDB
Type: REG_DWORD
Data: 0

Value 7
Name: ConnectionTimeOut
Type: REG_DWORD
Data: 0x4e20

Value 8
Name: DebugFlags
Type: REG_DWORD
Data: 0x8

Value 9
Name: Default Load File
Type: REG_SZ
Data: Default.htm

Value 10
Name: Dir Browse Control
Type: REG_DWORD
Data: 0x4000001e

Value 11
Name: Filter DLLs
Type: REG_SZ
Data: C:\WINNT\System32\inetsrv\sspifilt.dll

Value 12
Name: GlobalExpire
Type: REG_DWORD
Data: 0xffffffff

Value 13
Name: InstallPath
Type: REG_SZ
Data: C:\WINNT\System32\inetsrv

Value 14
Name: LogFileDirectory
Type: REG_EXPAND_SZ
Data: %SystemRoot%\System32\LogFiles

Value 15
Name: LogFileFormat
Type: REG_DWORD
Data: 0

Value 16
Name: LogFilePeriod
Type: REG_DWORD
Data: 0x1

Value 17
Name: LogFileTruncateSize
Type: REG_DWORD
Data: 0x1388000

Value 18
Name: LogSqlDataSource
Type: REG_SZ
Data: HTTPLOG

Value 19
Name: LogSqlPassword

```
  Type:       REG_SZ
  Data:       sqllog

Value 20
  Name:       LogSqlTableName
  Type:       REG_SZ
  Data:       Internetlog

Value 21
  Name:       LogSqlUserName
  Type:       REG_SZ
  Data:       InternetAdmin

Value 22
  Name:       LogType
  Type:       REG_DWORD
  Data:       0

Value 23
  Name:       MajorVersion
  Type:       REG_DWORD
  Data:       0x2

Value 24
  Name:       MaxConnections
  Type:       REG_DWORD
  Data:       0x186a0

Value 25
  Name:       MinorVersion
  Type:       REG_DWORD
  Data:       0

Value 26
  Name:       NTAuthenticationProviders
  Type:       REG_SZ
  Data:       NTLM

Value 27
  Name:       ScriptTimeout
  Type:       REG_DWORD
  Data:       0x384

Value 28
  Name:       SecurePort
  Type:       REG_DWORD
  Data:       0x1bb

Value 29
  Name:       ServerComment
  Type:       REG_SZ
  Data:

Value 30
  Name:       ServerSideIncludesEnabled
  Type:       REG_DWORD
  Data:       0x1

Value 31
  Name:       ServerSideIncludesExtension
  Type:       REG_SZ
  Data:       .stm


Key Name:
SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\Deny IP List
Class Name:       <NO CLASS>
Last Write Time:  12/16/96 - 4:28 PM

Key Name:
SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\Grant IP List
Class Name:       <NO CLASS>
Last Write Time:  12/16/96 - 4:28 PM

Key Name:
SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\Script Map
Class Name:       <NO CLASS>
Last Write Time:  12/16/96 - 9:21 AM
Value 0
  Name:       .idc
  Type:       REG_SZ
  Data:       C:\WINNT\System32\inetsrv\httpodbc.dll


Key Name:
SYSTEM\CurrentControlSet\Services\W3SVC\Parameters\Virtual Roots
Class Name:       <NO CLASS>
Last Write Time:  12/16/96 - 4:28 PM
Value 0
  Name:       /,
  Type:       REG_SZ
  Data:       C:\InetPub\wwwroot,,1

Value 1
  Name:       /iisadmin,
  Type:       REG_SZ
  Data:       C:\WINNT\System32\inetsrv\iisadmin,,1

Value 2
  Name:       /Scripts,
  Type:       REG_SZ
  Data:       C:\InetPub\scripts,,5


Key Name:
SYSTEM\CurrentControlSet\Services\W3SVC\Performance
Class Name:       <NO CLASS>
Last Write Time:  12/16/96 - 9:21 AM
Value 0
  Name:       Close
  Type:       REG_SZ
  Data:       CloseW3PerformanceData

Value 1
  Name:       Collect
  Type:       REG_SZ
  Data:       CollectW3PerformanceData

Value 2
  Name:       First Counter
  Type:       REG_DWORD
  Data:       0x758

Value 3
  Name:       First Help
  Type:       REG_DWORD
  Data:       0x759

Value 4
  Name:       Last Counter
  Type:       REG_DWORD
  Data:       0x790

Value 5
  Name:       Last Help
  Type:       REG_DWORD
  Data:       0x791

Value 6
  Name:       Library
  Type:       REG_SZ
  Data:       w3ctrs.DLL

Value 7
  Name:       Open
  Type:       REG_SZ
  Data:       OpenW3PerformanceData


Key Name:
SYSTEM\CurrentControlSet\Services\W3SVC\Security
Class Name:       <NO CLASS>
Last Write Time:  12/16/96 - 9:21 AM
Value 0
  Name:       Security
  Type:       REG_BINARY
  Data:
00000000  01 00 14 80 c0 00 00 00 - cc 00 00 00 14 00 00 00
................
00000010  34 00 00 00 02 00 20 00 - 01 00 00 00 02 80 18 00  4.....
........
00000020  ff 01 0f 00 01 01 00 00 - 00 00 00 01 00 00 00 00
................
00000030  20 02 00 00 02 00 8c 00 - 05 00 00 00 00 00 18 00
...............
00000040  8d 01 02 00 01 01 00 00 - 00 00 00 01 00 00 00 00
................
00000050  ff ff ff ff 00 00 1c 00 - fd 01 02 00 01 02 00 00  ................
00000060  00 00 00 05 20 00 00 00 - 23 02 00 00 98 97 14 00  ....
...#.......
00000070  00 00 1c 00 ff 01 0f 00 - 01 02 00 00 00 00 00 05  ................
00000080  20 00 00 00 20 02 00 00 - 98 97 14 00 00 00 1c 00   ...
...........
```

```
00000090   ff 01 0f 00 01 02 00 00 - 00 00 00 05 20 00 00 00  ............
...
000000a0   25 02 00 00 98 97 14 00 - 00 00 18 00 fd 01 02 00
%...............
000000b0   01 01 00 00 00 00 00 05 - 12 00 00 00 25 02 00 00
............%...
000000c0   01 01 00 00 00 00 00 05 - 12 00 00 00 01 01 00 00
................
000000d0   00 00 00 05 12 00 00 00 -                         ........
```

Key Name:
SYSTEM\CurrentControlSet\Services\W3SVC\W3SAMP
Class Name: <NO CLASS>
Last Write Time: 12/16/96 - 9:21 AM

SOFTWARE\MICROSOFT\ MSSQLServer

Key Name: SOFTWARE\Microsoft\MSSQLServer
Class Name: <NO CLASS>
Last Write Time: 5/12/97 - 4:26 PM

Key Name: SOFTWARE\Microsoft\MSSQLServer\Client
Class Name: <NO CLASS>
Last Write Time: 5/12/97 - 4:30 PM

Key Name: SOFTWARE\Microsoft\MSSQLServer\Client\ConnectTo
Class Name: <NO CLASS>
Last Write Time: 6/17/97 - 5:25 PM
Value 0
Name: DSQUERY
Type: REG_SZ
Data: DBNMPNTW

Key Name: SOFTWARE\Microsoft\MSSQLServer\Client\DB-Lib
Class Name: <NO CLASS>
Last Write Time: 6/17/97 - 4:23 PM
Value 0
Name: AutoAnsiToOem
Type: REG_SZ
Data: ON

Value 1
Name: UseIntlSettings
Type: REG_SZ
Data: ON

Key Name: SOFTWARE\Microsoft\MSSQLServer\MSSQLServer
Class Name: <NO CLASS>
Last Write Time: 6/19/97 - 1:06 PM
Value 0
Name: AuditLevel
Type: REG_DWORD
Data: 0

Value 1
Name: DefaultDomain
Type: REG_SZ
Data: TORNADO1

Value 2
Name: DefaultLogin
Type: REG_SZ
Data: guest

Value 3
Name: ImpersonateClient
Type: REG_DWORD
Data: 0

Value 4
Name: ListenOn
Type: REG_MULTI_SZ
Data: SSNMPN60,\\.\pipe\sql\query
SSMSSO60,1433

Value 5
Name: LoginMode
Type: REG_DWORD
Data: 0

Value 6
Name: MailAccountName
Type: REG_SZ
Data:

Value 7
Name: MailPassword
Type: REG_SZ
Data:

Value 8
Name: Map#
Type: REG_SZ
Data: -

Value 9
Name: Map$
Type: REG_SZ
Data:

Value 10
Name: Map_
Type: REG_SZ
Data: \

Value 11
Name: ResourceMgrID
Type: REG_SZ
Data: {BA50C160-D3AA-11D0-ACA5-000000000000}

Value 12
Name: SetHostname
Type: REG_DWORD
Data: 0

Value 13
Name: Tapeloadwaittime
Type: REG_DWORD
Data: 0xffffffff

Key Name:
SOFTWARE\Microsoft\MSSQLServer\MSSQLServer\CurrentVersion
Class Name: <NO CLASS>
Last Write Time: 6/17/97 - 4:23 PM
Value 0
Name: CurrentVersion
Type: REG_SZ
Data: 6.50.252

Value 1
Name: RegisteredOrganization
Type: REG_SZ
Data: Ingr

Value 2
Name: RegisteredOwner
Type: REG_SZ
Data: Ingr

Value 3
Name: RegisteredProductID
Type: REG_SZ
Data:

Value 4
Name: SerialNumber
Type: REG_DWORD
Data: 0x818f0040

Value 5
Name: SoftwareType
Type: REG_SZ
Data: System

Key Name:
SOFTWARE\Microsoft\MSSQLServer\MSSQLServer\Parameters
Class Name: <NO CLASS>
Last Write Time: 6/19/97 - 1:06 PM
Value 0
Name: SQLArg0
Type: REG_SZ
Data: -dD:\MSSQL\DATA\MASTER.DAT

Value 1
Name: SQLArg1
Type: REG_SZ
Data: -eD:\MSSQL\LOG\ERRORLOG

Key Name: SOFTWARE\Microsoft\MSSQLServer\Replication
Class Name: <NO CLASS>
Last Write Time: 5/23/97 - 4:37 PM
Value 0
Name: DistributionDB
Type: REG_SZ
Data:

Value 1
Name: WorkingDirectory
Type: REG_SZ
Data: D:\MSSQL\REPLDATA

Key Name: SOFTWARE\Microsoft\MSSQLServer\Setup
Class Name: <NO CLASS>
Last Write Time: 6/19/97 - 1:06 PM
Value 0
Name: CRC
Type: REG_SZ
Data: 130876933

```
Value 1
  Name:          SetupStatus
  Type:          REG_SZ
  Data:          Installed

Value 2
  Name:          SQLPath
  Type:          REG_SZ
  Data:          D:\MSSQL


Key Name:        SOFTWARE\Microsoft\MSSQLServer\SQL Interface
Class Name:      REG_MULTI_SZ
Last Write Time:   6/4/97 - 2:40 PM

Key Name:        SOFTWARE\Microsoft\MSSQLServer\SQL
Interface\Graph Control
Class Name:      REG_MULTI_SZ
Last Write Time:   6/4/97 - 2:40 PM

Key Name:        SOFTWARE\Microsoft\MSSQLServer\SQL Service
Manager
Class Name:      <NO CLASS>
Last Write Time:   6/17/97 - 5:27 PM
Value 0
  Name:          Action Verify
  Type:          REG_DWORD
  Data:          0

Value 1
  Name:          Background Interval
  Type:          REG_DWORD
  Data:          0x5

Value 2
  Name:          DefaultSvc
  Type:          REG_SZ
  Data:          MSSQLServer

Value 3
  Name:          Foreground Interval
  Type:          REG_DWORD
  Data:          0x2

Value 4
  Name:          Remote
  Type:          REG_DWORD
  Data:          0x1

Value 5
  Name:          Services
  Type:          REG_MULTI_SZ
  Data:          MSSQLServer
                 SQLExecutive
                 MSDTC


Value 6
  Name:          WindowDimensions
  Type:          REG_SZ
  Data:          0,374,277,275,214


Key Name:        SOFTWARE\Microsoft\MSSQLServer\SQLExecutive
Class Name:      <NO CLASS>
Last Write Time:   6/17/97 - 4:23 PM
Value 0
  Name:          CmdExecAccount
  Type:          REG_BINARY
  Data:
00000000   5e 96 b2 4b 1a b2 f3 de - 6f 3b 59 ee 97 9f 3e 88  ^..K....o;Y...>.

Value 1
  Name:          MailAutoStart
  Type:          REG_DWORD
  Data:          0x1

Value 2
  Name:          NonAlertableErrors
  Type:          REG_SZ
  Data:          1204,4002

Value 3
  Name:          RestartSQLServer
  Type:          REG_DWORD
  Data:          0x1

Value 4
  Name:          RestartSQLServerInterval
  Type:          REG_DWORD
  Data:          0x5

Value 5
  Name:          ServerHost
  Type:          REG_SZ
  Data:

Value 6
  Name:          SyshistoryLimitRows
  Type:          REG_DWORD
  Data:          0x1

Value 7
  Name:          SyshistoryMaxRows
  Type:          REG_DWORD
  Data:          0x3e8

Value 8
  Name:          TaskHistoryMaxRows
  Type:          REG_DWORD
  Data:          0x64


Key Name:
SOFTWARE\Microsoft\MSSQLServer\SQLExecutive\Subsystems
Class Name:      <NO CLASS>
Last Write Time:   6/5/97 - 4:48 PM
Value 0
  Name:          CmdExec
  Type:          REG_SZ
  Data:
D:\MSSQL\BINN\CMDEXEC.DLL,CmdExecStart,CmdEvent,CmdExecStop,1
0

Value 1
  Name:          Distribution
  Type:          REG_SZ
  Data:
D:\MSSQL\BINN\SQLREPL.DLL,distribution_start,distribution_event,distribu
tion_stop,100

Value 2
  Name:          LogReader
  Type:          REG_SZ
  Data:
D:\MSSQL\BINN\SQLREPL.DLL,logreader_start,logreader_event,logreader
_stop,25

Value 3
  Name:          Sync
  Type:          REG_SZ
  Data:
D:\MSSQL\BINN\SQLREPL.DLL,sync_start,sync_event,sync_stop,100
```

Microsoft SQL Server 6.5 Tunable Parameters

```
name                           minimum     maximum     config_value
 run_value
 ------------------------------ ----------- ----------- ------------
 -----------
affinity mask                          0  2147483647           0
           0
allow updates                          0           1           0
           0
backup buffer size                     1          32           1
           1
backup threads                         0          32           5
           5
cursor threshold                      -1  2147483647          -1
          -1
database size                          2       10000           2
           2
default language                       0        9999           0
           0
default sortorder id                   0         255          50
          50
fill factor                            0         100           0
           0
free buffers                          20      524288        5000
        5000
hash buckets                        4999      265003      265003
      265003
language in cache                      3         100           3
           3
LE threshold maximum                   2      500000         200
         200
LE threshold minimum                   2      500000          20
          20
LE threshold percent                   1         100           0
           0
locks                               5000  2147483647        5000
        5000
LogLRU buffers                         0  2147483647         256
         256
logwrite sleep (ms)                   -1         500          -1
          -1
max async IO                           1        1024          16
          16
max lazywrite IO                       1        1024          30
          30
max text repl size                     0  2147483647       65536
       65536
max worker threads                    10        1024          40
          40
media retention                        0         365           0
           0
memory                              2800     1048576      880000
      880000
nested triggers                        0           1           1
```

network packet size	1	512	32767	4096
open databases	4096	5	32767	20
open objects	20	100	2147483647	500
priority boost	500	0	1	0
procedure cache	0	1	99	2
Protection cache size	2	1	8192	15
RA cache hit limit	15	1	255	4
RA cache miss limit	4	1	255	3
RA delay	3	0	500	15
RA pre-fetches	15	1	1000	3
RA slots per thread	3	1	255	5
RA worker threads	5	0	255	0
recovery flags	0	0	1	0
recovery interval	0	1	32767	32767
remote access	32767	0	1	1
remote conn timeout	1	-1	32767	10
remote login timeout	10	0	2147483647	5
remote proc trans	5	0	1	0
remote query timeout	0	0	2147483647	0
remote sites	0	0	256	10
resource timeout	10	5	2147483647	10
set working set size	10	0	1	0
show advanced options	0	0	1	1
SMP concurrency	1	-1	64	-1
sort pages	-1	64	511	64
spin counter	64	1	2147483647	10000
tempdb in ram (MB)	10000	0	2044	5
time slice	5	50	1000	100
user connections	100	5	32767	4460
user options	4460	0	4095	0

0

AMI MegaRAID CONFIGURATION

****

Adapter 0
****
Number Of Logical Drives: 2.

Logical Drive
State : Optimal
RAID TYPE : 0
Write Policy : Write Thru
Read Policy : No Read Ahead
Cache Policy : Direct I/O
Stripe Size : 64K Byte
No. of Stripes : 8
Size : 33080MB
Component Physical Drives :

RANK 0
CHANNEL : 0, TARGET : 0
CHANNEL : 1, TARGET : 0
CHANNEL : 0, TARGET : 1
CHANNEL : 1, TARGET : 1
CHANNEL : 0, TARGET : 2
CHANNEL : 1, TARGET : 2
CHANNEL : 0, TARGET : 4
CHANNEL : 1, TARGET : 4

Logical Drive
State : Optimal
RAID TYPE : 0
Write Policy : Write Thru
Read Policy : No Read Ahead
Cache Policy : Direct I/O
Stripe Size : 64K Byte
No. of Stripes : 8
Size : 33080MB
Component Physical Drives :

RANK 0
CHANNEL : 0, TARGET : 5
CHANNEL : 1, TARGET : 5
CHANNEL : 0, TARGET : 6
CHANNEL : 1, TARGET : 6
CHANNEL : 0, TARGET : 8
CHANNEL : 0, TARGET : 9
CHANNEL : 1, TARGET : 8
CHANNEL : 1, TARGET : 9

@
****
Adapter 1
****
Number Of Logical Drives: 2.

Logical Drive
State : Optimal
RAID TYPE : 0
Write Policy : Write Thru
Read Policy : No Read Ahead
Cache Policy : Direct I/O
Stripe Size : 64K Byte
No. of Stripes : 8
Size : 33080MB
Component Physical Drives :

RANK 0
CHANNEL : 0, TARGET : 0
CHANNEL : 1, TARGET : 0
CHANNEL : 0, TARGET : 1
CHANNEL : 1, TARGET : 1
CHANNEL : 0, TARGET : 2
CHANNEL : 1, TARGET : 2
CHANNEL : 0, TARGET : 4
CHANNEL : 1, TARGET : 4

Logical Drive
State : Optimal
RAID TYPE : 0
Write Policy : Write Thru
Read Policy : No Read Ahead
Cache Policy : Direct I/O
Stripe Size : 64K Byte
No. of Stripes : 8
Size : 33176MB
Component Physical Drives :

RANK 0
CHANNEL : 0, TARGET : 5
CHANNEL : 1, TARGET : 5
CHANNEL : 0, TARGET : 6
CHANNEL : 0, TARGET : 8
CHANNEL : 1, TARGET : 6
CHANNEL : 1, TARGET : 8
CHANNEL : 0, TARGET : 9
CHANNEL : 1, TARGET : 9

@
****
Adapter 2
****
Number Of Logical Drives: 2.

Logical Drive
State : Optimal
RAID TYPE : 0
Write Policy : Write Thru

```
Read Policy        : No Read Ahead
Cache Policy       : Direct I/O
Stripe Size        : 64K Byte
No. of Stripes     : 8
Size               : 33176MB
Component Physical Drives :

RANK 0
      CHANNEL : 0, TARGET : 0
      CHANNEL : 1, TARGET : 0
      CHANNEL : 0, TARGET : 1
      CHANNEL : 1, TARGET : 1
      CHANNEL : 0, TARGET : 2
      CHANNEL : 1, TARGET : 2
      CHANNEL : 0, TARGET : 4
      CHANNEL : 1, TARGET : 4

Logical Drive
State              : Optimal
RAID TYPE          : 0
Write Policy       : Write Thru
Read Policy        : No Read Ahead
Cache Policy       : Direct I/O
Stripe Size        : 64K Byte
No. of Stripes     : 8
Size               : 33176MB
Component Physical Drives :

RANK 0
      CHANNEL : 0, TARGET : 5
      CHANNEL : 1, TARGET : 5
      CHANNEL : 0, TARGET : 6
      CHANNEL : 1, TARGET : 6
      CHANNEL : 0, TARGET : 8
      CHANNEL : 1, TARGET : 8
      CHANNEL : 0, TARGET : 9
      CHANNEL : 1, TARGET : 9

@
**********
Adapter 3
**********
Number Of Logical Drives: 2.

Logical Drive
State              : Optimal
RAID TYPE          : 0
Write Policy       : Write Thru
Read Policy        : No Read Ahead
Cache Policy       : Direct I/O
Stripe Size        : 64K Byte
No. of Stripes     : 8
Size               : 33176MB
Component Physical Drives :

RANK 0
      CHANNEL : 0, TARGET : 0
      CHANNEL : 1, TARGET : 0
      CHANNEL : 0, TARGET : 1
      CHANNEL : 1, TARGET : 1
      CHANNEL : 0, TARGET : 2
      CHANNEL : 1, TARGET : 2
      CHANNEL : 0, TARGET : 4
      CHANNEL : 1, TARGET : 4

Logical Drive
State              : Optimal
RAID TYPE          : 0
Write Policy       : Write Thru
Read Policy        : No Read Ahead
Cache Policy       : Direct I/O
Stripe Size        : 64K Byte
No. of Stripes     : 8
Size               : 33176MB
Component Physical Drives :

RANK 0
      CHANNEL : 0, TARGET : 5
      CHANNEL : 1, TARGET : 5
      CHANNEL : 0, TARGET : 6
      CHANNEL : 1, TARGET : 6
      CHANNEL : 0, TARGET : 8
      CHANNEL : 1, TARGET : 8
      CHANNEL : 0, TARGET : 9
      CHANNEL : 1, TARGET : 9

@
**********
Adapter 4
**********
Number Of Logical Drives: 5.

Logical Drive
State              : Optimal
RAID TYPE          : 0
Write Policy       : Write Thru
Read Policy        : No Read Ahead
Cache Policy       : Direct I/O
Stripe Size        : 64K Byte
No. of Stripes     : 1
Size               : 4148MB
Component Physical Drives :

RANK 0
      CHANNEL : 2, TARGET : 0

Logical Drive
State              : Optimal
RAID TYPE          : 0
Write Policy       : Write Thru
Read Policy        : No Read Ahead
Cache Policy       : Direct I/O
Stripe Size        : 64K Byte
No. of Stripes     : 8
Size               : 33080MB
Component Physical Drives :

RANK 0
      CHANNEL : 1, TARGET : 1
      CHANNEL : 2, TARGET : 1
      CHANNEL : 1, TARGET : 2
      CHANNEL : 2, TARGET : 2
      CHANNEL : 1, TARGET : 4
      CHANNEL : 2, TARGET : 4
      CHANNEL : 1, TARGET : 5
      CHANNEL : 2, TARGET : 5

Logical Drive
State              : Optimal
RAID TYPE          : 0
Write Policy       : Write Thru
Read Policy        : No Read Ahead
Cache Policy       : Direct I/O
Stripe Size        : 64K Byte
No. of Stripes     : 3
Size               : 12441MB
Component Physical Drives :

RANK 0
      CHANNEL : 1, TARGET : 6
      CHANNEL : 1, TARGET : 8
      CHANNEL : 1, TARGET : 9

Logical Drive
State              : Optimal
RAID TYPE          : 0
Write Policy       : Write Thru
Read Policy        : No Read Ahead
Cache Policy       : Direct I/O
Stripe Size        : 64K Byte
No. of Stripes     : 3
Size               : 12441MB
Component Physical Drives :

RANK 0
 CHANNEL : 2, TARGET : 8
```

CHANNEL : 2, TARGET : 9

Logical Drive
State : Optimal
RAID TYPE : 0
Write Policy : Write Thru
Read Policy : Read Ahead
Cache Policy : Caching I/O
Stripe Size : 64K Byte
No. of Stripes : 1
Size : 8677MB
Component Physical Drives :

RANK 0
CHANNEL : 1, TARGET : 0

@

Appendix D: Disk Storage Calculations

Disk Storage									
Note: Numbers are in KB unless otherwise specified									
Warehouse configured:		**440**							
Thoughput (tpmC):		**5,307.23**		**TpmC/Wh**	**12.06**				
Table	**Rows**	**Data**	**Index**	**5% Space**	**Daily Growth**				
Warehouse	440	880	8	44					
District	4,400	8,800	40	442					
Customer	13,200,000	8,801,760	683,144	474,245					
Orders	13,200,000	343,200	2,074	0	66,634				
Order_line	132,002,932	7,337,790	47,966	0	1,425,378				
New_order	3,960,000	44,000	268	2,213					
Stock	44,000,000	14,669,600	81,056	737,533					
Item	100,000	9,100	46	457					
history	13,200,000	660,002	0	0	127,374				
Total		**31,875,132**	**814,602**	**1,214,935**	**1,619,387**				
Database	**Allocated**								
Master DB & etc	29,696	master + model + msdb + pubs							
TPCC DB	40,905,728.00								
Total_Allocated	**40,935,424**								
Dynamic space	8,340,992	Sum of Data for Order, Order_line and History							
Static space	25,563,677	Sum of all data and index (including the rootdbs) + 5% - Dynamic space							
Free space	7,030,755	Total space allocated to DBMS - Dynamic and static spaces							
Daily growth	1,609,730	(Dynamic space / (W*62.5))* tpmC							
Daily spread	4,616,161	Free Space - 1.5*Daily growth (zero if negative)				p.s. Since MS SQL Server can be configured to			
180 day space	315,314,998	Static space + 180 * (Daily growth + Daily Spread)				eleminate daily spread, zero is assumed in here			
180 day (GB)	**300.71 GB**								
			after	before	diff	log/no in KB			
log per new order	5.29		2510.7207	0.001952	2510.718752	5.294289902			
8 hrs log space	13,487,047								
Total									
Space Usage	**(GB)**								
180-day space	300.71 GB								
Logs (mirrored)	25.72 GB								
swap	0.80 GB		**Currently pricing**						
OS and MSSQL	0.13 GB		**Size:**	**4.04 GB**	**8.47 GB**				
Total	**327.36 GB**		**Quantity:**	**79**	**1**	**Total Storage:**	327.39 GB		

Appendix E: Third Party Letters and Price Quotations

VECTOR

PROPOSAL: 70723-NS
DATE: 7/29/97

TO:

SUBMITTED BY:

Charles H Roberman SIGNATURE

Charles H. Roberman
Area Manager *croberman@vector.com*

ATTN:
PHONE:
FAX:

PHONE: 281-440-8340, x.224
FAX: 281-440-8460

VECTOR IS PLEASED TO PROPOSE AS FOLLOWS:

ITEM	DESCRIPTION	QTY	UNIT PRICE	EXTENSION
1	FDPS373 InterServe 650 (256MB,3x4GB,UPS, no rack,NTS) List price is $30,990.	1	$24790	$24790
2	FDPS445 InterServe 305 (32MB,2GB,NTS) List price is $5,507.	4	4405	17620
3	FDSK473 InterRAID-8 Rackmount with AMI controller List price is $7,300.	4	7130	28520
4	FDSK459 InterRAID-8 Rackmount without AMI controller List price is $5,300	5	5175	25875
5	FMEM151 256MB Memory Upgrade List price is $2,399.	7	2200	15400
6	FMEM134 32MB Memory Upgrade List price is $259.	12	240	2880
7	FMTP161 4mm Tape Drive List price is $1,399.	1	1365	1365
8	FOPT099 15" Monitor List price is $399.	5	390	1950
9	FDSK461 4GB Hot swap Drive List price is $1,199.	76	1170	88920
10	FDSK470 9GB Hot Swap Drive List price is $2,099.	1	2050	2050
11	FINF920 Intel 10/100BaseT PCI NIC List price is $150.	5	145	725
	Total			$209995

Terms of Sale:

- This is an offer to enter into a contract with Vector Technology Corp. at its home office address and is valid for thirty (30) days.
- All prices are F.O.B. point of origin and do not include freight, installation, sales taxes, excise taxes, duties, tariffs, or other charges levied by federal, state, or local governmental authority.
- Terms of payment are net thirty (30) days with established credit.
- A finance charge of 1 1/2% per month, which is an annual rate of 18%, will be charged on all past-due accounts.
- Title shall become that of the Customer upon delivery to common carrier or a licensed trucker, which shall constitute delivery to the Customer.
- Please refer to current VECTOR installation, maintenance, technical services, and warranty policies if applicable.
- Any transaction between VECTOR and Customer shall be governed and construed in accordance to the laws of the State of Texas.
- **Vector Technology Corp. shall in no event be liable for special, indirect or consequential damages of any kind.**

DELIVERY TIME: 15 days ARO

TERMS: _____ FOB: Origin

ACCEPTED BY: _____
SIGNATURE

_____
NAME/TITLE

P.O.# _____ DATE: _____

Vector Technology Corporation - Home Office: 15111 Mintz Lane - Houston, Texas 77014 (281)440-8340

QUTATION.XLS

QUOTATION

COMPANY NAME :

From: NEVEEN MOURAD

Attn.: NIK SIMPSON

Date : 35487

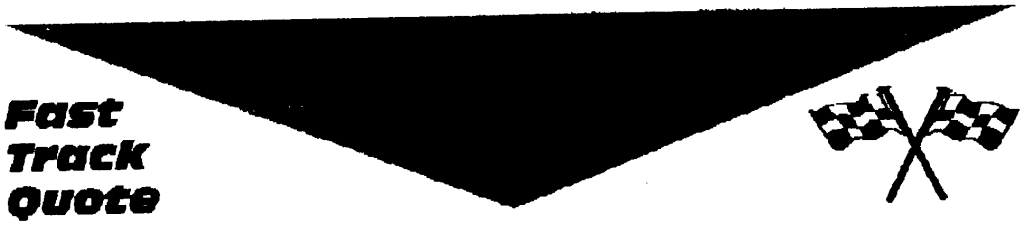
Phone : 205-730-6286

Fax # : 205-730-6239

Line #	Feature	DESK.	Quantity	PRICE	EX.PRICE
1	NET WORK CARD	AEF-8TX 8 PORT 100 BASE T HUB	3	$529.00	$1,587.00
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					$0.00
14					$0.00
15					$0.00
16					$0.00
17					$0.00
18					$0.00
19					$0.00
20					$0.00
Comment :		Sub Total			$1,587.00
		SHIPPING			
		TAX			
		Gross Price			$1,587.00

PAYMENT TERM : COD CASH

Page 1



RESELLER

Name Nick Simpson

Address Intergraph Computer Systems

Phone 205-730-4286

FAX 205-730-6239

Quantity	Item	Description	Unit Price	Total
203	906024-10EH	24 port 10-base T	$220	$44660
			Total	$44660

Microdyne

Microdyne Use Only - Please forward to Fast Track Bid Desk (800) 255-3967, Fax: (703) 329-3716

Microdyne Sales Rep: CRYSTAL WHITLEY Date: 7/27/92

Approval: Crystal Whitley Date: 7/27/92

Tech Data

Tech Data Use Only - Please forward to Danny Conlan (813) 539-7429, Fax: (813) 538-7801

Tech Data Rep: Date:

Approval: Date:

Redmond, WA 98052-6399 Fax 206 936 7329

Microsoft®

July 21, 1997

Mr. John Lam
Server Performance and Analysis
Intergraph Computer Systems
1 Madison Industrial Parkway
Huntsville, AL 35894

via FAX: (205) 730-6239

Dear John,

Here is the information you requested regarding pricing of certain Microsoft products:

Microsoft SQL Server 6.5 software, unlimited user license	$24999
Windows NT Server 4.0 software, incl 5 CALs	$809
Microsoft SQL Workstation (includes programmers toolkit)	$499
Visual C++ 5.0 Professional Edition	$499
5-yr maintenance for above software @ $2095/yr	$10475

This quote is valid for the next 60 days. Please let me know if I can be of any further assistance.

Best regards,

Sid Arora
Product Manager, Microsoft SQL Server
Personal and Business Systems Division

Microsoft Corporation is an equal opportunity employer.

Redmond, WA 98052 6399 Fax 206 936 7329

Microsoft

July 21, 1997

Mr. John Lam
Server Performance and Analysis
Intergraph Computer Systems
1 Madison Industrial Parkway
Huntsville, AL 35894

via FAX: (205) 730-6239

Dear John,

Microsoft has received your request for permission to disclose the results of TPC-C benchmark tests conducted by Intergraph with Microsoft SQL Server 6.5 on the following system:

Intergraph IS650 server, 2-processors, Pentium Pro, 200 MHz
Test results: 5307 tpmC @ $63/tpmC approx.

Microsoft hereby grants Intergraph permission to disclose these results to third parties and acknowledges that Intergraph has formally requested permission to do so in accordance with the license agreement for Microsoft SQL Server software.

Best regards,

Sid Arora
Product Manager, Microsoft SQL Server
Personal and Business Systems Division