

TPC Benchmark™ H

Full Disclosure Report for



PRIMERGY RX300 S3

Using EXASolution 2.0

December 10, 2007

First Edition

First Edition – December 10, 2007

PRIMERGY RX300 S3
using
EXASolution 2.0

Fujitsu Siemens Computers GmbH believes that the information in this document is accurate as of the publication date. The information in this document is subject to change without notice. We assume no responsibility for any errors that may appear in this document. The pricing information in this document is believed to accurately reflect the current prices as of the publication date. However, we provide no warranty of the pricing information in this document.

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

All performance data contained in this report were obtained in a rigorously controlled environment. Results obtained in other operating environments may vary significantly. We do not warrant or represent that a user can or will achieve similar performance. No warranty of system performance or price/performance is expressed or implied in this report.

Copyright © 2007 Fujitsu Siemens Computers GmbH. All rights reserved.

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

PRIMERGY and PRIMERGY RX300 S3 are trademarks of Fujitsu Siemens Computers GmbH.

EXASOL, EXASolution and EXACluster OS are trademarks of EXASOL AG.

Intel and Xeon are trademarks of Intel Corporation.

TPC Benchmark and TPC-H are trademarks or registered trademarks of the Transaction Processing Performance Council (TPC).

All other brand or product names mentioned are considered trademarks or registered trademarks of their respective owners.

Abstract

Overview

This report documents the methodology and results of the TPC Benchmark H test conducted with the EXASolution 2.0 based on a PRIMERGY Server Cluster in conformance with the requirements of the TPC-H Benchmark Specification.

The benchmark results are summarized in the following table.

Software	Hardware	Total System Cost	QphH @1000GB	Price/QphH @1000GB
EXASolution 2.0	PRIMERGY RX300 S3 Cluster	\$ 993,077	580,729.8	\$ 1.72

The Transaction Processing Performance Council (TPC) developed the TPC-H Benchmark. The TPC was founded to define transactions processing benchmarks and to disseminate objective, verifiable performance data to the industry.

Standard and Executive Summary Statements

The Executive Summary and Numerical Quantities Summary of the benchmark results for the PRIMERGY RX300 S3 Cluster can be found in the following pages.

Auditor

In order to verify compliance to the TPC-H benchmark specification, Francois Raab, Infosizing, Inc., audited the benchmark configuration, environment, and methodology used to produce and validate the test results, and the pricing model used to calculate the price/performance.



PRIMERGY RX300 S3
using EXASolution 2.0

TPC-H Rev. 2.6.0

Report Date
December 10, 2007

Total System Cost

Composite Query per Hour Metric

Price / Performance

\$ 993,077

580,729.8
QphH@1000GB

\$ 1.72
\$ / QphH@1000GB

Database Size

Database Manager

Operating System

Other Software

Availability Date

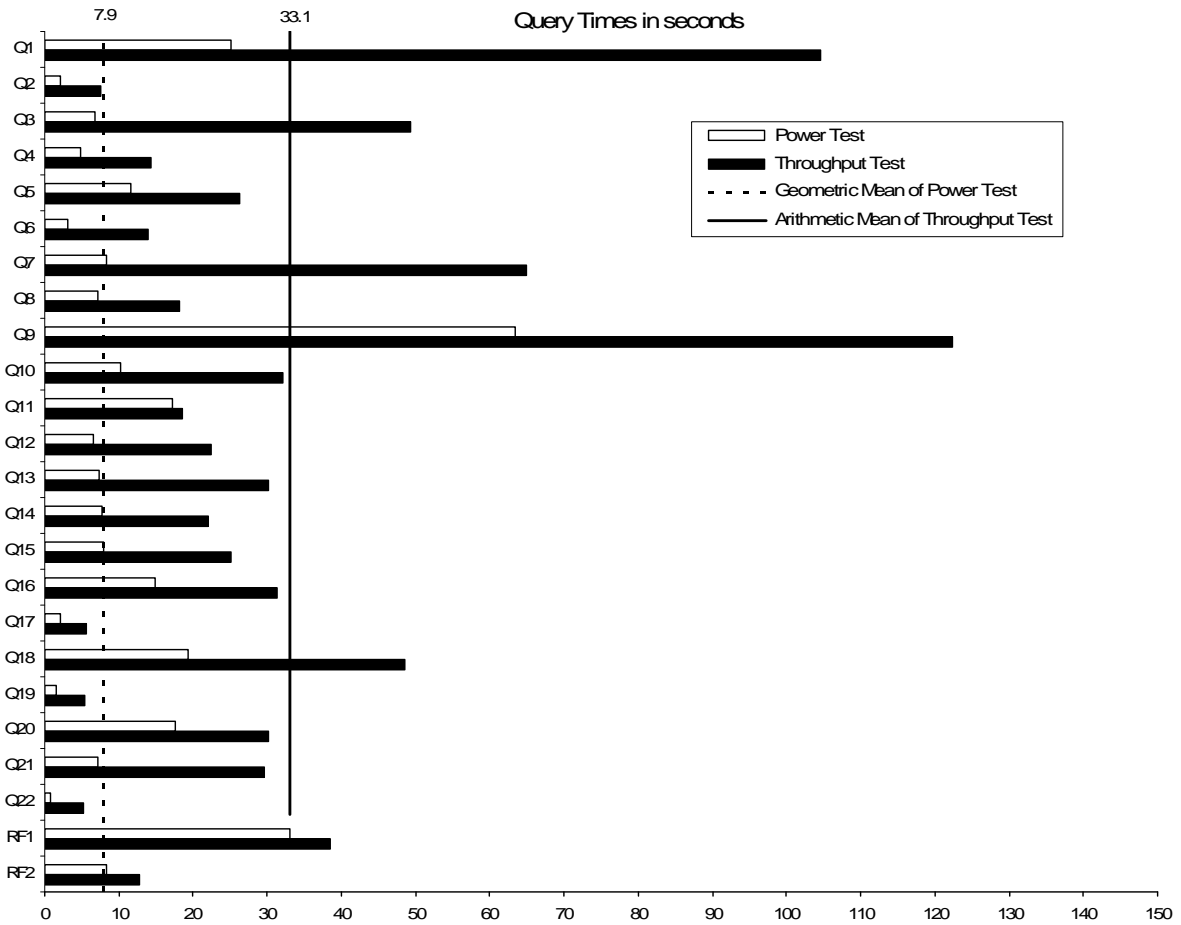
1000 GB

EXASolution 2.0

EXACluster OS 1.3

C++ Compiler

December 10, 2007



Database Load Time = 1.92

Included Backup: N

Data Storage / Database Size = 32.63

RAID (Base tables only): Y

RAID (Base tables and auxiliary data structures): Y

RAID (All): Y

System Configuration:

40 x PRIMERGY RX300 S3 Server, each with:
 2 Intel XEON X5365 QC 3.0 GHz processors (each is 1 chip, 4 cores, 4 threads)
 16 GB RAM
 6x 146 GB (10k rpm) internal SAS disks

Total Storage: 32,634 GB
 (1 GB = 1024 * 1024 * 1024 bytes)



PRIMERGY RX300 S3
using EXASolution 2.0

TPC-H Rev. 2.6.0

Report Date
December 10, 2007

Description	Part Number	Pricing	Unit Price	Qty	Extended Price	3 yr. Maint. Price	
System Hardware							
PRIMERGY RX 300 S3							
2x 3.0 Ghz Quadcore CPUs (Intel® Xeon 5365)							
12x 2.5" SAS/S-ATA HotSwap Slots							
8x 2GB Base FBDIMM – 667MHz	PY RX300S3 2.5/X5365	1	11,731	40	469,240		
6x HD SAS 146GB 10.000 rpm hot plug 2.5"							
RAID Ctrl SAS onboard 256MB iTBBU LSI							
2x Ethernet Cable (LAN-CAT 5)							
TopUp Service 3 years, 24x7, 4h response time						40,824	
Server Discount (36%)						-14,697	
HP ProCurve 5406zl (1000Base-T Ethernet Switch, 48 ports)	J8699A#ABB	1	8,945	1	8,945		
HP ProCurve Switch 5400ZL 24 port modul	J8702A	1	4,479	2	8,958		
HP eCare Pack ProCurve Chassis 6 Switch 5406zl, 3 years, 24x7, 4h response time	UE251E	1				1,396	
Switch Discount						-48	
					Subtotal	\$ 315,615	27,475

Storage

No external storage required

Software

EXASolution 2.0 licence for 3 years	EXA-40N-16G	2	692,213	1	692,213		
EXASOL Discount (10%)		2			-69,222		
EXASOL Premium Support (3.9%)	EXA-SUP-P	2				26,996	
					Subtotal	\$ 622,991	\$ 26,996
					Total	\$ 938,606	\$ 54,471

3-Year Cost of Ownership \$ 993,077

QphH Rating: 580,729.8

\$/QphH@1000GB: 1.72

Price Key: 1-FSC, contact: Julian Sayer, Julian.Sayer@fujitsu-siemens.com
2-EXASOL, contact: Olga Sapozhnykova, sales@exasol.com

All discounts are based on list prices and for similar quantities and configurations.

Results independently audited by: Francois Raab of InfoSizing, Inc. (www.sizing.com)

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

Numerical Quantities

Measurement Results

Database Scale Factor	1000 GB
Total Data Storage / Database Size	32.63
Start of Database Load	17:53:41
End of Database Load	19:49:00
Database Load Time	1h 55m 19 s
Query Streams for Throughput Test	7
TPC-H Power	456,232.5
TPC-H Throughput	739,200.0
TPC-H Composite Query-per-Hour Metric (QphH@1000GB)	580,729.8
Total System Price Over 3 Years	\$993,077
TPC-H Price/ Performance Metric (\$/QphH@1000GB)	\$1.72

Measurement Interval

Measurement Interval in Throughput Test (Ts)	750 seconds
--	-------------

Duration of Stream Execution

Stream ID	Seed	Start Date	Start Time	End Date	End Time	Duration
Stream 0	1111194900	2007-11-11	19:49:15	2007-11-11	19:54:11	4mins:56secs
Stream 1	1111194901	2007-11-11	19:54:11	2007-11-11	20:05:59	11 mins:48secs
Stream 2	1111194902	2007-11-11	19:54:15	2007-11-11	20:06:31	12 mins:16secs
Stream 3	1111194903	2007-11-11	19:54:15	2007-11-11	20:06:16	12 mins:01secs
Stream 4	1111194904	2007-11-11	19:54:16	2007-11-11	20:06:21	12 mins:05secs
Stream 5	1111194905	2007-11-11	19:54:17	2007-11-11	20:06:37	12 mins:20secs
Stream 6	1111194906	2007-11-11	19:54:18	2007-11-11	20:06:20	12 mins:02secs
Stream 7	1111194907	2007-11-11	19:54:18	2007-11-11	20:06:41	12 mins:23secs
Refresh		2007-11-11	19:54:11	2007-11-11	20:00:12	6 mins:01secs

TPC-H Timing Intervals (in seconds)

Query	Power	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7	Min Q _i	Avg Q _i	Max Q _i
1	25.2	100.8	98.8	104.0	99.9	109.0	108.6	111.5	98.8	104.7	111.5
2	2.0	7.3	9.8	6.0	9.0	7.0	7.0	6.8	6.0	7.6	9.8
3	6.8	45.1	48.4	47.6	52.2	64.8	47.8	38.7	38.7	49.2	64.8
4	4.9	12.4	12.3	15.4	15.5	15.0	13.1	16.1	12.3	14.3	16.1
5	11.5	29.3	24.1	29.9	29.2	24.9	22.8	23.4	22.8	26.2	29.9
6	3.1	19.4	6.6	16.4	13.1	14.9	14.0	12.7	6.6	13.9	19.4
7	8.3	76.2	74.6	47.9	62.2	65.3	71.1	58.1	47.9	65.0	76.2
8	7.1	15.4	21.0	17.7	19.4	16.2	15.0	22.9	15.0	18.2	22.9
9	63.3	94.6	123.2	135.7	118.8	111.5	124.8	147.6	94.6	122.3	147.6
10	10.3	34.3	35.5	28.7	30.6	30.9	36.1	29.3	28.7	32.2	36.1
11	17.1	21.8	17.2	17.4	16.3	19.3	16.5	21.4	16.3	18.6	21.8
12	6.6	25.1	25.1	25.3	24.8	22.4	23.1	10.9	10.9	22.4	25.3
13	7.4	30.4	32.2	28.4	29.6	29.5	30.6	30.4	28.4	30.1	32.2
14	7.8	22.6	25.4	19.9	21.7	25.0	19.8	20.5	19.8	22.1	25.4
15	7.9	23.5	28.4	24.2	23.3	34.7	24.0	18.5	18.5	25.2	34.7
16	14.9	30.9	33.2	30.6	30.8	33.2	31.5	29.2	29.2	31.3	33.2
17	2.1	4.4	6.1	7.2	7.9	3.8	3.8	5.8	3.8	5.6	7.9
18	19.4	48.9	44.9	50.0	46.2	48.0	46.6	55.3	44.9	48.5	55.3
19	1.5	4.7	5.8	4.3	6.7	5.5	4.7	6.1	4.3	5.4	6.7
20	17.7	37.6	28.9	27.4	27.2	26.7	26.7	37.0	26.7	30.2	37.6
21	7.2	19.2	26.8	30.3	37.1	27.0	29.9	36.5	19.2	29.5	37.1
22	0.9	3.5	7.5	6.2	3.4	5.8	5.4	4.2	3.4	5.1	7.5
RF1	33.1	41.7	39.7	34.0	39.0	38.5	39.4	37.3	34.0	38.5	41.7
RF2	8.4	19.2	16.5	9.0	11.7	12.6	11.0	9.6	9.0	12.8	19.2

Table of Contents

ABSTRACT	3
OVERVIEW	3
STANDARD AND EXECUTIVE SUMMARY STATEMENTS	3
AUDITOR.....	3
TABLE OF CONTENTS	8
1 GENERAL ITEMS	12
1.1 TEST SPONSOR	12
1.2 PARAMETER SETTINGS	12
1.3 CONFIGURATION ITEMS	12
2 CLAUSE 1: LOGICAL DATABASE DESIGN RELATED ITEMS	14
2.1 DATABASE DEFINITION STATEMENTS	14
2.2 PHYSICAL ORGANIZATION	14
2.3 HORIZONTAL PARTITIONING	14
2.4 REPLICATION	14
3 CLAUSE 2: QUERIES AND REFRESH FUNCTIONS	15
3.1 QUERY LANGUAGE	15
3.2 VERIFYING METHOD FOR RANDOM NUMBER GENERATION.....	15
3.3 GENERATING VALUES FOR SUBSTITUTION PARAMETERS.....	15
3.4 QUERY TEXT AND OUTPUT DATA FROM QUALIFICATION DATABASE.....	15
3.5 QUERY SUBSTITUTION PARAMETERS AND SEEDS USED.....	15
3.6 ISOLATION LEVEL	15
3.7 SOURCE CODE OF REFRESH FUNCTIONS.....	15
4 CLAUSE 3: DATABASE SYSTEM PROPERTIES	16
4.1 ACID PROPERTIES	16
4.2 ATOMICITY REQUIREMENTS	16
4.3 CONSISTENCY REQUIREMENTS	16
4.4 ISOLATION REQUIREMENTS.....	17
4.5 DURABILITY REQUIREMENTS.....	19
5 CLAUSE 4: SCALING AND DATABASE POPULATION	20
5.1 ENDING CARDINALITY OF TABLES	20
5.2 DISTRIBUTION OF TABLES AND LOGS ACROSS MEDIA	20
5.3 MAPPING OF DATABASE PARTITIONS/REPLICATION	20
5.4 IMPLEMENTATION OF RAID.....	20
5.5 DBGEN MODIFICATIONS	21
5.6 DATA BASE LOAD TIME.....	21
5.7 DATA STORAGE RATIO	21
5.8 DATABASE LOAD MECHANISM DETAILS AND ILLUSTRATION	21
5.9 QUALIFICATION DATABASE CONFIGURATION.....	22
5.10 BASE DATA COMPARISON.....	22
5.11 REFERENTIAL INTEGRITY OF INITIAL DATABASE LOAD	22
6 CLAUSE 5: PERFORMANCE METRICS AND EXECUTION RULES RELATED ITEMS	23
6.1 SYSTEM ACTIVITY BETWEEN LOAD AND PERFORMANCE TESTS	23
6.2 STEPS IN THE POWER TEST.....	23
6.3 TIMING INTERVAL FOR EACH QUERY AND REFRESH FUNCTIONS.....	23
6.4 NUMBER OF STREAMS FOR THE THROUGHPUT TEST	23
6.5 START AND END DATE/TIME OF EACH QUERY STREAM	23
6.6 TOTAL ELAPSED TIME OF THE MEASUREMENT INTERVAL	23
6.7 REFRESH FUNCTION START DATE/TIME AND FINISH DATE/TIME	23
6.8 TIMING INTERVALS FOR EACH QUERY AND EACH REFRESH FUNCTION FOR EACH STREAM.....	23
6.9 PERFORMANCE METRICS.....	24
6.10 THE PERFORMANCE METRIC AND NUMERICAL QUANTITIES FROM BOTH RUNS	24

6.11 SYSTEM ACTIVITY BETWEEN PERFORMANCE TESTS	24
7 CLAUSE 6: SUT AND DRIVER IMPLEMENTATION RELATED ITEMS	25
7.1 DRIVER	25
7.2 IMPLEMENTATION SPECIFIC LAYER (ISL)	25
7.3 PROFILE-DIRECTED OPTIMIZATION	25
8 CLAUSE 7: PRICING	26
8.1 HARDWARE AND SOFTWARE USED IN THE PRICED SYSTEM	26
8.2 TOTAL THREE YEAR PRICE	26
8.3 AVAILABILITY DATE	26
9 CLAUSE 8: AUDITOR'S INFORMATION AND ATTESTATION LETTER	27
9.1 AUDITOR'S REPORT	27
APPENDIX A: SYSTEM AND DATABASE CONFIGURATIONS AND PARAMETERS	28
APPENDIX B: DATABASE BUILD SCRIPTS	29
APPENDIX C: ACID SCRIPTS	31
APPENDIX D: QUERY TEXT AND QUERY OUTPUT	77
APPENDIX E: SEED AND INPUT PARAMETERS	88
APPENDIX F: BENCHMARK PROGRAMS AND SCRIPTS	90
APPENDIX G: PRICING INFORMATION	109

Benchmark Sponsor: Detlev Seidel
 Primary TPC Representative
 Fujitsu Siemens Computers GmbH
 Heinz-Nixdorf-Ring 1
 33106 Paderborn

December 3, 2007

I verified the TPC Benchmark™ H performance of the following configuration:

Platform: **Fujitsu Siemens PRIMERGY RX300 S3, 40-node Cluster**
 Database Manager: **EXASolution 2.0**
 Operating System: **EXACluster OS 1.3**

The results were:

CPU (Speed)	Memory	Disks	QphH@1000GB
PRIMERGY RX300 S3, 40-node cluster (each node with)			
2 x Intel XEON X5365 Quad (3.0GHz)	16GB Main	6 x 146GB SAS	580,729.8

In my opinion, this performance result was produced in compliance with the TPC’s requirements for the benchmark. The following verification items were given special attention:

- The database records were defined with the proper layout and size
- The database population was generated using DBGEN
- The database was properly scaled to 1,000GB and populated accordingly
- The compliance of the database auxiliary data structures was verified

- The database load time was correctly measured and reported
- The required ACID properties were verified and met
- The query input variables were generated by QGEN
- The query text was produced using minor modifications
- The execution of the queries against the SF1 database produced compliant answers
- A compliant implementation specific layer was used to drive the tests
- The throughput tests involved 7 query streams
- The ratio between the longest and the shortest query was such that no query timing was adjusted
- The execution times for queries and refresh functions were correctly measured and reported
- The repeatability of the measured results was verified
- The required amount of database log was configured
- The system pricing was verified for major components and maintenance
- The major pages from the FDR were verified for accuracy

Additional Audit Notes:

None.

Respectfully Yours,



François Raab
President

1 General Items

1.1 Test Sponsor

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

This TPC-H benchmark is sponsored by Fujitsu Siemens Computers GmbH. The benchmark implementation was developed and engineered by EXASOL AG.

1.2 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*
- *Optimizer/Query execution options*
- *Query processing tool/language configuration parameters*
- *Recovery/commit options*
- *Consistency/locking options*
- *Operating system and configuration parameters*
- *Configuration parameters and options for any other software component incorporated into the pricing structure*
- *Compiler optimization options*

This requirement can be satisfied by providing a full list of all parameters and options, as long as all those which have been modified from their default values have been clearly identified and these parameters and options are only set once.

Details of system and database configuration and parameters are provided in Appendix A.

1.3 Configuration Items

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

- *Number and type of processors.*
- *Size of allocated memory, and any specific mapping/partitioning of memory unique to the test.*
- *Number and type of disk units (and controllers, if applicable).*
- *Number of channels or bus connections to disk units, including their protocol type.*
- *Number of LAN (e.g. Ethernet) Connections, including routers, workstations, terminals, etc., that were physically used in the test or are incorporated into the pricing structure.*
- *Type and the run-time execution location of software components (e.g., DBMS, query processing tools/languages, middle-ware components, software drivers, etc.).*

The System Under Test (SUT), depicted in Figure 1.1, that was used to obtain the results in this benchmark consists of a cluster of 40 PRIMERGY RX300 S3 servers, each with:

System components

- 2 QC CPU (Intel® Xeon X5365, 3GHz)
- 16 GB RAM
- 6 146 GB SAS 10.000 rpm
- 1 RAID controller onboard (BBU included)
- 2 Ethernet Ports
- 1 HP ProCurve 5406zl (1000Base-T Ethernet Switch, 96 ports)

Network is Gigabit Ethernet, there is one physical network for initial data distribution, synchronization and administration. Each server contains 6 physical disks. 3 pairs - each of 2 disks - are mirrored among the controller (Hardware RAID 1) and then striped (Software RAID 0).

Priced configuration and benchmarked configuration are identical.

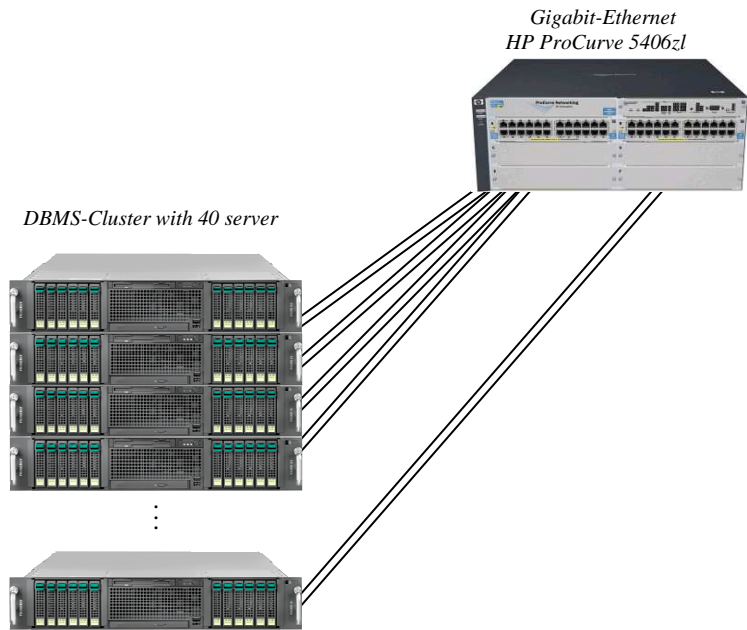


Figure 1.1: Benchmarked and priced system configuration

2 Clause 1: Logical Database Design Related Items

2.1 Database Definition Statements

Listings must be provided for all table definition statements and all other statements used to set up the test and qualification databases (8.1.2.1).

Appendix B contains the build scripts that define the tables and indices for the TPC-H database.

2.2 Physical Organization

The physical organization of tables and indices, within the test and qualification databases, must be disclosed. If the column ordering of any table is different from that specified in Clause 1.4, it must be noted.

Physical organization requires no user input. All the database data is placed on the same partition.

2.3 Horizontal Partitioning

Horizontal partitioning of tables and rows in the test and qualification databases (see Clause 1.5.4) must be disclosed.

Horizontal partitioning is used. The data is automatically distributed on the cluster nodes using a hash algorithm. The columns used for the hashing are controlled by DDL statements (see Appendix B).

2.4 Replication

Any replication of physical objects must be disclosed and must conform to the requirements of Clause 1.5.6.

No objects were physically replicated.

3 Clause 2: Queries and Refresh Functions

3.1 Query Language

The query language used to implement the queries must be identified.

SQL was the query language uniquely used throughout this benchmark.

3.2 Verifying Method for Random Number Generation

The method of verification for the random number generation must be described unless the supplied DBGEN and QGEN were used.

TPC supplied versions 2.6.0 Build 1 of DBGEN and version 2.6.0 Build 1 of QGEN were used in this benchmark. The DBGEN tool was patched by EXASOL changing the output format (see section 5.5).

3.3 Generating Values for Substitution Parameters

The method used to generate values for substitution parameters must be disclosed. If QGEN is not used for this purpose, then the source code of any non-commercial tool used must be disclosed. If QGEN is used, the version number, release number, modification number, and patch level of QGEN must be disclosed.

QGEN version 2.6.0 Build 1 was used to generate the substitution parameters.

3.4 Query Text and Output Data from Qualification Database

The executable query text used for query validation must be disclosed along with the corresponding output data generated during the execution of the query text against the qualification database. If minor modifications (see Clause 2.2.3) have been applied to any functional query definition or approved variants in order to obtain executable query text, these modifications must be disclosed and justified. The justification for a particular minor query modification can apply collectively to all queries for which it has been used. The output data for the power and throughput tests must be made available electronically upon request.

Appendix D contains the actual query text and query output. Minor modifications have been applied and were commented as such.

3.5 Query Substitution Parameters and Seeds Used

The query substitution parameters used for all performance tests must be disclosed in tabular format, along with the seeds used to generate these parameters.

Appendix E contains the seed and query substitution parameters.

3.6 Isolation Level

The isolation level used to run the queries must be disclosed. If the isolation level does not map closely to the levels defined in Clause 3.4, additional descriptive detail must be provided.

The queries and transactions were run with the isolation level 3 (phenomena P1 through P4, as defined in the TPC-H specification, are prevented from occurring).

3.7 Source Code of Refresh Functions

The details of how the refresh functions were implemented must be disclosed (including source code of any non-commercial program used).

Appendix F contains the source code of the refresh functions.

4 Clause 3: Database System Properties

4.1 ACID Properties

The ACID (Atomicity, Consistency, Isolation, and Durability) properties of transaction processing systems must be supported by the system under test during the timed portion of this benchmark. Since TPC-H is not a transaction processing benchmark, the ACID properties must be evaluated outside the timed portion of the test.

Appendix C contains the source code of the ACID test scripts.

4.2 Atomicity Requirements

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

4.2.1 Atomicity of the Completed Transactions

Perform the ACID Transaction for a randomly selected set of input data and verify that the appropriate rows have been changed in the ORDERS, LINEITEM, and HISTORY tables.

The following steps were performed to verify the atomicity of the completed ACID transactions:

1. The total price from the ORDERS table and the extended price from the LINEITEM table were retrieved for a randomly selected order key.
2. One ACID Transaction was performed using the order key from step 1.
3. The ACID Transaction was committed.
4. The total price from the ORDERS table and the extended price from the LINEITEM table were retrieved for the same order key.
5. It was verified that the appropriate rows had been changed.

4.2.2 Atomicity of Aborted Transactions

Perform the ACID Transaction for a randomly selected set of input data, substituting a ROLLBACK of the transaction for the COMMIT of the transaction. Verify that the appropriate rows have not been changed in the ORDERS, LINEITEM, and HISTORY tables.

The following steps were performed to verify the atomicity of the completed ACID transactions:

1. The total price from the ORDERS table and the extended price from the LINEITEM table were retrieved for a randomly selected order key.
2. One ACID Transaction was performed using the order key from step 1. The transaction was stopped prior to the commit.
3. The ACID Transaction was rolled back.
4. The total price from the ORDERS table and the extended price from the LINEITEM table were retrieved for the same order key.
5. It was verified that the appropriate rows had not been changed.

4.3 Consistency Requirements

Consistency is the property of the application that requires any execution of transactions to take the database from one consistent state to another. A consistent state for the TPC-H database is defined to exist when:

$$O_TOTALPRICE = \sum(\text{trunc}(\text{trunc}(L_EXTENDEDPRICE * (1 - L_DISCOUNT), 2) * (1 + L_TAX), 2))$$

For each ORDER and LINEITEM defined by (O_ORDERKEY = L_ORDERKEY).

4.3.1 Consistency Test

Verify that ORDERS and LINEITEM tables are initially consistent, submit the prescribed number of ACID Transactions with randomly selected input parameters, and re-verify the consistency of the ORDERS and LINEITEM.

The following queries were executed before and after the durability tests to show that the database was always in a consistent state both initially and after submitting transactions:


```

SELECT *
FROM (
    SELECT o_orderkey, o_totalprice - sum(trunc(trunc(l_extendedprice *
        (1-l_discount),2)*(1+l_tax),2)) part_res
    FROM orders, lineitem
    WHERE o_orderkey=l_orderkey
    GROUP BY o_orderkey, o_totalprice
) WHERE not part_res=0;

```

EXASolution 2.0 has serializable isolation level with table level lock concurrency control. The ACID Transaction of stream0 was expanded with 5 seconds delay after the update and before commit. Since only one update transaction can execute at any one time, the delay should guaranty that the active update transaction is "in-flight" at the time of the failure. To improve the throughput of the concurrent streams a random delay was used after COMMIT (0-5 seconds) and after transaction conflicts (0-2 seconds).

The following steps were performed to verify the consistency of ACID transactions:

1. The consistency of the ORDERS and LINEITEM tables was verified.
2. 1000 transactions for each of the 11+1 execution streams were prepared.
3. After that at least 100 ACID transactions were submitted from each of 11+1 execution streams.
4. A durability failure was induced by pulling a disk drive or switching off the power to one or more servers.
5. The consistency of the ORDERS and LINEITEM tables was re-verified.

4.4 Isolation Requirements

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

The steps of the isolation tests were adopted to the EXASOL isolation environment.

4.4.1 Isolation Test 1 – Read-Write Conflict with Commit

Demonstrate isolation for the read-write conflict of a read-write transaction and a read-only transaction when the read-write transaction is committed

The following steps were performed to satisfy the test of isolation for a read-only and a read-write committed transaction:

1. Start a query and verify that the row was retrieved.
2. Start an update transaction, read and update the same row. Wait before commit.
3. Start the same query and verify that the row retrieved has not changed.
4. Commit the update transaction
5. Start the same query and verify that the new row is retrieved

4.4.2 Isolation Test 2 – Read-Write Conflict with Rollback

Demonstrate isolation for the read-write conflict of a read-write transaction and a read-only transaction when the read-write transaction is rolled back.

The following steps were performed to satisfy the test of isolation for a read-only and a rolled back read-write transaction:

1. Start a query and verify that the row was retrieved.
2. Start an update transaction, read and update the same row. Wait before commit.
3. Start the same query and verify that the row retrieved has not changed.
4. Rollback the update transaction
5. Start the same query and verify that the old row (step 1) is retrieved

4.4.3 Isolation Test 3 – Write-Write Conflict with Commit

Demonstrate isolation for the write-write conflict of two update transactions when the first transaction is committed.

The following steps were performed to verify isolation of two update transactions:

1. Start a query and verify that the row was retrieved.
2. Start an update transaction, read and update the same row. Wait before commit.

3. Start another update transaction, read and try to update the same row and verify that the transaction is forced to rollback.
4. commit the update transaction
5. Start the same query and verify that the new row is retrieved

4.4.4 Isolation Test 4 – Write-Write Conflict with Rollback

Demonstrate isolation for the write-write conflict of two update transactions when the first transaction is rolled back.

The following steps were performed to verify isolation of two update transactions after the first one is rolled back:

1. Start a query and verify that the row was retrieved.
2. Start an update transaction, read and update the same row. Wait before commit.
3. Start another update transaction, read and try to update the same row and verify that the transaction is forced to rollback.
4. Rollback the update transaction
5. Start the same query and verify that the new row retrieved in step 1 has not changed

4.4.5 Isolation Test 5 – Concurrent Read and Write Transactions on Different Tables

Demonstrate the ability of read and write transactions affecting different database tables to make progress concurrently.

The following steps were performed to demonstrate the ability of read and write transactions affecting different tables to make progress concurrently:

6. Create a copy of tables Lineitem and Orders. The original tables are referred to as TablePair A. The duplicate tables are referred to as TablePair B.
7. Start a query against TablePair A and verify that the row was retrieved.
8. Start a query against TablePair B and verify that the row was retrieved.
9. Start an update transaction, read and update the same row in TablePair A. Wait before commit.
10. Start an update transaction, read and update the same row in TablePair B and commit.
11. Commit the update transaction against TablePair A.
12. Start a query against TablePair A and verify that the updated row was retrieved.
13. Start a query against TablePair B and verify that the updated row was retrieved.

4.4.6 Isolation Test 6 – Update Transactions during Continuous Read-Only Query Stream

Demonstrate the continuous submission of arbitrary (read-only) queries against one or more tables of the database does not indefinitely delay update transactions affecting those tables from making progress.

The following query was used:

```
SELECT l1.l_quantity,
       SUM(l2.l_extendedprice),
       SUM(l3.l_extendedprice),
       SUM(l3.l_quantity)
FROM   lineitem l1, lineitem l2, lineitem l3
WHERE  l1.l_shipdate <= DATE '1998-12-01' -0
      AND l1.l_orderkey = l2.l_orderkey
      AND l1.l_linenumber = l2.l_linenumber
      AND l1.l_extendedprice = l3.l_extendedprice
      AND l3.l_quantity < 30
GROUP BY l1.l_quantity;
COMMIT;
```

1. A Transaction, T1, which executed the above query against the qualification database, was started using a randomly selected DELTA.
2. An ACID Transaction, T2, was started for a randomly selected O_KEY, L_KEY and DELTA.
3. T2 completed and appropriate rows in the ORDERS, LINEITEM and HISTORY tables had been changed.
4. T1 was still executing.
5. Transaction T1 completed executing the query.

4.5 Durability Requirements

The tested system must guarantee durability: the ability to preserve the effects of committed transactions and insure database consistency after recovery from any one of the failures listed in Clause 3.5.2.

4.5.1 Permanent Unrecoverable Failure of Any Durable Medium

Guarantee the database and committed updates are preserved across a permanent irrecoverable failure of any single durable medium containing TPC-H database tables or recovery log tables.

During the durability test, one disk was removed. The test continued uninterrupted, because of the RAID protection (see also section 1.3). Upon replacement/re-insert of the drive, the data files were rebuild to a consistent state by the RAID controller. The durability success file and the HISTORY table were compared and the counts matched.

4.5.2 System Crash

Guarantee the database and committed updates are preserved across an instantaneous interruption (system crash/system hang) in processing which requires the system to reboot to recover.

The system crash and memory failure tests were combined. Two system crashes were performed by turning off the power during the durability test: one test for only one node and the other test for the whole cluster.

When power was restored, in both cases the system rebooted automatically and the database was restarted manually. The durability success file and the HISTORY table were compared and the counts matched.

4.5.3 Memory Failure

Guarantee the database and committed updates are preserved across failure of all or part of memory (loss of contents). See the previous section.

The system crash and memory failure tests were combined as explained in section 4.5.2.

5 Clause 4: Scaling and Database Population

5.1 Ending Cardinality of Tables

The cardinality (e.g., the number of rows) of each table of the test database, as it existed at the completion of the database load (see clause 4.2.5) must be disclosed.

Table 5.1 lists the TPC Benchmark H defined tables and the row count for each table as they existed upon completion of the build.

Table Name	Cardinality
LINEITEM	5,999,989,709
ORDER	1,500,000,000
PARTSUPP	800,000,000
PART	200,000,000
CUSTOMER	150,000,000
SUPPLIER	10,000,000
NATION	25
REGION	5

Table 5.1: Initial Number of Rows

5.2 Distribution of Tables and Logs Across Media

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

Each server contains 6 physical disks. 3 pairs - each of 2 disks - are mirrored among the controller (Hardware RAID 1) and then striped (Software RAID 0). Controller cache is switched on.

The resulting device is divided into 4 partitions (see Table 5.2). All benchmark- and database-relevant data is stored in partition Data (Tables, Indexes, Temp, Logs, Flat Files).

Partition	Size
Boot	0.1 GB
System	46 GB
Swap	22 GB
Data	334 GB

Table 5.2: Disk layout schema for one DBMS Cluster node

5.3 Mapping of Database Partitions/Replication

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

No database partitions/replications were used in this benchmark.

5.4 Implementation of RAID

Implementations may use some form of RAID to ensure high availability. If used for data, auxiliary storage (e.g. indexes) or temporary space, the level of RAID must be disclosed for each device.

Please refer to chapter 5.2.

5.5 DBGEN Modifications

The version number, release number, modification number, and patch level of DBGEN must be disclosed. Any modifications to the DBGEN (see Clause 4.2.1) source code must be disclosed. In the event that a program other than DBGEN was used to populate the database, it must be disclosed in its entirety.

TPC supplied versions 2.6.0 Build 1 of DBGEN with a minor EXASOL patch was used in this benchmark. The patch resulted in the following changes of the output format:

- Quotes for strings
- Separators

See Appendix F for details.

5.6 Data Base Load Time

The database load time for the test database (see Clause 4.3) must be disclosed.

The database load time was 1h55m19sec.

5.7 Data Storage Ratio

The data storage ratio must be disclosed. It is computed by dividing the total data storage of the priced configuration (expressed in GB) by the size chosen for the test database as defined in 4.1.3.1. The ratio must be reported to the nearest 1/100th, rounded up.

Disk Type	GB per disk*	# of disks	Total (GB)**
Internal	146 GB	240	32633.54

Scale Factor	Data Storage Ratio
1000	32.63

* Disk manufacturer definition of 1 GB is 10^9 bytes

** In this calculation 1 GB is defined as 2^{30} bytes

5.8 Database Load Mechanism Details and Illustration

The details of the database load must be disclosed, including a block diagram illustrating the overall process. Disclosure of the load procedure includes all steps, scripts, input and configuration files required to completely reproduce the test and qualification databases.

The database was loaded using data generation stored on the flat files all on the tested and priced configuration. DBGEN was used to create the flat files.

The block diagram in figure 5.8.1 describes the process used to load the database.

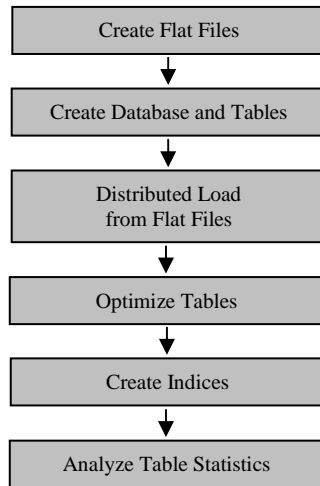


Figure 5.8.1: Database Load Process

5.9 Qualification Database Configuration

Any differences between the configuration of the qualification database and the test database must be disclosed.

The qualification database used identical scripts to create and load the data with changes to adjust for the database scale factor.

5.10 Base Data Comparison

Verify that the rows in the loaded database after the performance test are correct by comparing some small number of rows extracted at random from any two files of the corresponding Base, Insert and Delete reference data set files for each table and the corresponding rows of the database.

Verified according to the specification.

5.11 Referential Integrity of Initial Database Load

Verify referential integrity in the database after the initial load.

Verified according to the specification.

6 Clause 5: Performance Metrics and Execution Rules Related Items

6.1 System Activity between Load and Performance Tests

Any system activity on the SUT that takes place between the conclusion of the load test and the beginning of the performance test must be fully disclosed.

There is no activity on the SUT between the conclusion of the load test and the beginning of the performance test. Benchmark scripts are started just after the load was completed.

6.2 Steps in the Power Test

The details of the steps followed to implement the power test (e.g., system boot, database restart, etc.) must be disclosed.

The following steps were used to implement the power test:

1. Database was started
2. RF1 Refresh Transaction
3. Stream 00 Execution
4. RF2 Refresh Transaction

6.3 Timing Interval for Each Query and Refresh Functions

The timing intervals (see Clause 5.3.6) for each query of the measured set for both refresh functions must be reported for the power test.

The timing intervals for each query and both update functions are given in the Numerical Quantities Summary earlier in this document.

6.4 Number of Streams for the Throughput Test

The number of execution streams used for the throughput test must be disclosed.

7 query streams and 1 refresh stream were used for the throughput test.

6.5 Start and End Date/Time of Each Query Stream

The start time and finish time for each query stream must be reported for the throughput test.

The throughput test start time and finish time for each stream are given in the Numerical Quantities Summary earlier in this document.

6.6 Total Elapsed Time of the Measurement Interval

The total elapsed time of the measurement interval (see Clause 5.3.5) must be reported for the throughput test.

The total elapsed time of the throughput test is given in the Numerical Quantities Summary earlier in this document.

6.7 Refresh Function Start Date/Time and Finish Date/Time

Start and finish time for each update function in the update stream must be reported for the throughput test.

The refresh function start and finish date/time are given in the Numerical Quantities Summary earlier in this document.

6.8 Timing Intervals for Each Query and Each Refresh Function for Each Stream

The timing intervals (see Clause 5.3.6) for each query of each stream and for each refresh function must be reported for the throughput test.

The timing intervals for each query and each update function are given in the Numerical Quantities Summary earlier in this document.

6.9 Performance Metrics

The computed performance metric, related numerical quantities and price performance metric must be reported.

The performance metrics, and the numbers, on which they are based, is given in the Numerical Quantities Summary earlier in this document.

6.10 The Performance Metric and Numerical Quantities from Both Runs

A description of the method used to determine the reproducibility of the measurement results must be reported. This must include the performance metrics (QppH and QthH) from reproducibility runs.

Run ID	QppH@1000GB	QptH@1000GB	QphH@1000GB
Run 1	456,232.5	739,200.0	580,729.8
Run 2	489,069.1	728,515.1	596,903.9
% Difference	6.71 %	1.47 %	2.71 %

6.11 System Activity between Performance Tests

Any activity on the SUT that takes place between the conclusion of Run 1 and the beginning of Run 2 must be disclosed.

There was no system activity between Run 1 and Run 2.

7 Clause 6: SUT and Driver Implementation Related Items

7.1 Driver

A detailed description of how the driver performs its functions must be supplied, including any related source code or scripts. This description should allow an independent reconstruction of the driver.

All stream executions are performed by a script. QGEN is used to produce query text.

For each power-test run:

1. The shell script for RF1 is started
2. Then the queries as generated by QGEN are submitted in the order defined by Clause 5.3.5.4 from the driver.
3. The shell script for RF2 is started.

For each throughput-test run:

1. The queries as generated by QGEN are submitted in the order defined by Clause 5.3.5.4 from the driver in several streams.
2. Parallely, pairs of RF1/RF2 are executed sequentially in one update stream.

The source code of the used scripts are disclosed on Appendix F.

7.2 Implementation Specific Layer (ISL)

If an implementation specific layer is used, then a detailed description of how it performs its functions must be supplied, including any related source code or scripts. This description should allow an independent reconstruction of the implementation-specific layer.

This benchmark test did not make use of an ISL.

7.3 Profile-Directed Optimization

If profile-directed optimization as described in Clause 5.2.9 is used, such use must be disclosed.

Profile-directed optimization was not used in this benchmark test.

8 Clause 7: Pricing

8.1 Hardware and Software Used in the Priced System

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

A detailed list of hardware and software used in the priced system is included in the pricing sheet in the executive summary. All prices are currently effective. Third-party price quotations are included in Appendix G.

8.2 Total Three Year Price

The total 3-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.

A detailed pricing sheet of all the hardware and software used in this configuration and the 3-year maintenance costs, demonstrating the computation of the total 3-year price of the configuration, is included in the executive summary at the beginning of this document.

8.3 Availability Date

The committed delivery date for general availability of products used in the priced calculations must be reported. When the priced system includes products with different availability dates, the availability date reported on the executive summary must be the date by which all components are committed to being available. The full disclosure report must report availability dates individually for at least each of the categories for which a pricing subtotal must be provided.

Component	Availability Date
Cluster Hardware	Now (date of publication)
EXASolution 2.0	Now (date of publication)

9 Clause 8: Auditor's Information and Attestation Letter

9.1 Auditor's Report

The auditor's agency name, address, phone number, and Attestation letter with a brief audit summary report indicating compliance must be included in the full disclosure report. A statement should be included specifying who to contact in order to obtain further information regarding the audit process.

This implementation of the TPC Benchmark H was audited by Francois Raab for InfoSizing.

Further information regarding the audit process may be obtained from:

Francois Raab

InfoSizing, Inc.

125 West Monroe Street

Colorado Springs, CO 80907

(719) 473-7555

(719) 473-7554

Email: francois@sizing.com

The auditor's attestation letter is included after the table of contents.

TPC Benchmark H Full Disclosure Report and other information can be downloaded from the Transaction Processing Performance Council website at www.tpc.org.

Appendix A: System and Database Configurations and Parameters

Kernel-Parameters:

```
kernel.shmall = 16777216
kernel.shmmax = 2147483648
kernel.shmni = 16777216
kernel.msgmnb = 1073741824
kernel.msgmni = 256
kernel.msgmax = 1073741824
vm.swappiness = 0
vm.vfs_cache_pressure = 100
vm.overcommit_ratio = 50
vm.overcommit_memory = 0
vm.nr_hugepages = 5500
```

Database Parameters:

```
-dictionarythreshold 300
-replicationborder 10100000
-slb 11000
```

Appendix B: Database Build Scripts

```
# -----
# create_schema.sql
# -----

CREATE SCHEMA tpc;
OPEN SCHEMA tpc;
CREATE OR REPLACE TABLE NATION ( N_NATIONKEY DEC(11), N_NAME CHAR(25), N_REGIONKEY DEC(11), N_COMMENT VARCHAR(152) );
CREATE OR REPLACE TABLE REGION ( R_REGIONKEY DEC(11), R_NAME CHAR(25), R_COMMENT VARCHAR(152) );
CREATE OR REPLACE TABLE PART ( P_PARTKEY DEC(11), P_NAME VARCHAR(55), P_MFGR CHAR(25), P_BRAND CHAR(10), P_TYPE VARCHAR(25),
P_SIZE DEC(10), P_CONTAINER CHAR(10), P_RETAILPRICE DECIMAL(12,2), P_COMMENT VARCHAR(23) );
CREATE OR REPLACE TABLE SUPPLIER ( S_SUPPKEY DEC(11), S_NAME CHAR(25), S_ADDRESS VARCHAR(40), S_NATIONKEY DEC(11), S_PHONE
CHAR(15), S_ACCTBAL DECIMAL(12,2), S_COMMENT VARCHAR(101) );
CREATE OR REPLACE TABLE PARTSUPP ( PS_PARTKEY DEC(11), PS_SUPPKEY DEC(11), PS_AVAILQTY DEC(10), PS_SUPPLYCOST DECIMAL(12,2),
PS_COMMENT VARCHAR(199) );
CREATE OR REPLACE TABLE CUSTOMER ( C_CUSTKEY DEC(11), C_NAME VARCHAR(25), C_ADDRESS VARCHAR(40), C_NATIONKEY DEC(11), C_PHONE
CHAR(15), C_ACCTBAL DECIMAL(12,2), C_MKTSEGMENT CHAR(10), C_COMMENT VARCHAR(117) );
CREATE OR REPLACE TABLE ORDERS ( O_ORDERKEY DEC(11), O_CUSTKEY DEC(11), O_ORDERSTATUS CHAR(1), O_TOTALPRICE DECIMAL(12,2),
O_ORDERDATE DATE, O_ORDERPRIORITY CHAR(15), O_CLERK CHAR(15), O_SHIPPRIORITY DEC(10), O_COMMENT VARCHAR(79) );
CREATE OR REPLACE TABLE LINEITEM ( L_ORDERKEY DEC(11), L_PARTKEY DEC(11), L_SUPPKEY DEC(11), L_LINENUMBER DEC(10), L_QUANTITY
DECIMAL(12,2), L_EXTENDEDPRICE DECIMAL(12,2), L_DISCOUNT DECIMAL(12,2), L_TAX DECIMAL(12,2), L_RETURNFLAG
CHAR(1), L_LINestatus CHAR(1), L_SHIPDATE DATE, L_COMMITDATE DATE, L_RECEIPTDATE DATE, L_SHIPINSTRUCT CHAR(25),
L_SHIPMODE CHAR(10), L_COMMENT VARCHAR(44) );

COMMIT;
ALTER TABLE CUSTOMER set primary join attribute C_CUSTKEY;
ALTER TABLE ORDERS set primary join attribute O_CUSTKEY;
ALTER TABLE LINEITEM set primary join attribute L_ORDERKEY;
ALTER TABLE PARTSUPP set primary join attribute PS_PARTKEY, PS_SUPPKEY;
ALTER TABLE PART set primary join attribute P_PARTKEY;
ALTER TABLE SUPPLIER set primary join attribute S_SUPPKEY;
COMMIT;

# -----
# create_user.sql
# -----

create user tpcuser identified by tpcuser;
grant create session to tpcuser;
grant create table to tpcuser;
grant create view to tpcuser;
grant create schema to tpcuser;

-- to get debug output of system tables
grant select any dictionary to tpcuser;
commit;

# -----
# create_indices.sql
# -----
set autocommit off;
open schema tpc;

optimize table part;
commit;
optimize table partsupp;
commit;
optimize table supplier;
commit;
optimize table customer;
commit;
optimize table orders;
commit;
optimize table lineitem;
commit;
enforce index lineitem_on_suppkey on lineitem (l_suppkey);
enforce index lineitem_on_partkey_suppkey on lineitem (l_partkey, l_suppkey);
enforce index lineitem_on_partkey on lineitem (l_partkey);
enforce index lineitem_on_orderkey on lineitem (l_orderkey);
enforce index nation_on_nationkey on nation (n_nationkey);
enforce index region_on_regionkey on region (r_regionkey);
enforce index supplier_on_suppkey on supplier (s_suppkey);
enforce index supplier_on_nationkey on supplier (s_nationkey);
enforce index customer_on_custkey on customer (c_custkey);
enforce index customer_on_nationkey on customer (c_nationkey);
enforce index part_on_partkey on part (p_partkey);
enforce index partsupp_on_partkey_suppkey on partsupp (ps_partkey, ps_suppkey);
enforce index partsupp_on_partkey on partsupp (ps_partkey);
enforce index partsupp_on_suppkey on partsupp (ps_suppkey);
enforce local index orders_on_orderkey_l on orders (o_orderkey);
enforce global index orders_on_orderkey_g on orders (o_orderkey);
enforce index orders_on_custkey on orders (o_custkey);
commit;
analyze table part estimate statistics;
analyze table partsupp estimate statistics;
analyze table supplier estimate statistics;
analyze table customer estimate statistics;
analyze table orders estimate statistics;
analyze table lineitem estimate statistics;
analyze table nation estimate statistics;
analyze table region estimate statistics;
commit;
```

```

# -----
# load_init_parallel.sh
# -----

#!/bin/bash

if [ $# -ne 7 ]; then
    echo "Syntax: $0 <dw> <user> <password> <schema> <SF> <instances-per-node> <logdir>"
    exit 1
fi

DW_ENV=$1
user=$2
pass=$3
schema=$4
SF=$5
INST_PN=$6
LOGDIR="$8"
export TPC_SHAREDIR=/shares/dwstaging/benchmark/shares

DW_STATE="`dw $DW_ENV show state`" || {
    echo "DW with environment $DW_ENV does not work." >&2
    exit 1
}

DW_CONN="`echo $DW_STATE | sed 's/^.*host //; s/^\([^ ]+\)` at live port \/1:/'`"
DW_NODES="`dw $DW_ENV show nodes | wc -l`"

TPC_LOCALDIR="/shares/dwstaging/local/benchmark/TPC_Database_SF${SF}_${INST_PN}x_${DW_NODES}N"

NATION_COLUMNS="N_NATIONKEY:DECIMAL(11)|N_NAME:CHAR(25)|N_REGIONKEY:DECIMAL(11)|N_COMMENT:VARCHAR(152)"
REGION_COLUMNS="R_REGIONKEY:DECIMAL(11)|R_NAME:CHAR(25)|R_COMMENT:VARCHAR(152)"
PART_COLUMNS="P_PARTKEY:DECIMAL(11)|P_NAME:VARCHAR(55)|P_MFGR:CHAR(25)|P_BRAND:CHAR(10)|P_TYPE:VARCHAR(25)|P_SIZE:DECIMAL(10)|P_CONTAINER:CHAR(10)|P_RETAILPRICE:DECIMAL(12,2)|P_COMMENT:VARCHAR(23)"
SUPPLIER_COLUMNS="S_SUPPKEY:DECIMAL(11)|S_NAME:CHAR(25)|S_ADDRESS:VARCHAR(40)|S_NATIONKEY:DECIMAL(11)|S_PHONE:CHAR(15)|S_ACCTBAL:DECIMAL(12,2)|S_COMMENT:VARCHAR(101)"
PARTSUPP_COLUMNS="PS_PARTKEY:DECIMAL(11)|PS_SUPPKEY:DECIMAL(11)|PS_AVAILQTY:DECIMAL(10)|PS_SUPPLYCOST:DECIMAL(12,2)|PS_COMMENT:VARCHAR(199)"
CUSTOMER_COLUMNS="C_CUSTKEY:DECIMAL(11)|C_NAME:VARCHAR(25)|C_ADDRESS:VARCHAR(40)|C_NATIONKEY:DECIMAL(11)|C_PHONE:CHAR(15)|C_A_CCTBAL:DECIMAL(12,2)|C_MKTSEGMENT:CHAR(10)|C_COMMENT:VARCHAR(117)"
ORDERS_COLUMNS="O_ORDERKEY:DECIMAL(11)|O_CUSTKEY:DECIMAL(11)|O_ORDERSTATUS:CHAR(1)|O_TOTALPRICE:DECIMAL(12,2)|O_ORDERDATE:DATE|O_ORDERPRIORITY:CHAR(15)|O_CLERK:CHAR(15)|O_SHIPPRIORITY:DECIMAL(10)|O_COMMENT:VARCHAR(79)"
LINEITEM_COLUMNS="L_ORDERKEY:DECIMAL(11)|L_PARTKEY:DECIMAL(11)|L_SUPPKEY:DECIMAL(11)|L_LINENUMBER:DECIMAL(10)|L_QUANTITY:DECIMAL(12,2)|L_EXTENDEDPRICE:DECIMAL(12,2)|L_DISCOUNT:DECIMAL(12,2)|L_TAX:DECIMAL(12,2)|L_RETURNFLAG:CHAR(1)|L_LINESTATUS:CHAR(1)|L_SHIPDATE:DATE|L_COMMITDATE:DATE|L_RECEIPTDATE:DATE|L_SHIPINSTRUCT:CHAR(25)|L_SHIPMODE:CHAR(10)|L_COMMENT:VARCHAR(44)"

# since LINEITEM is the largest table by far it is loaded parallel to the other tables
dw "$DW_ENV" loader -p -q -l csv2dwa -s $TPC_LOCALDIR/metafile.LINEITEM.csv -d $user:$schema:$pass@$DW_CONN -a -t LINEITEM -c "`eval echo \${LINEITEM_COLUMNS}`" | sed -e "s/\([0-9]\{0-9\}*\)[ \t]*rows loaded\./LINEITEM rows loaded; \1/g" || { echo "Initial loading failed in table LINEITEM." ; exit 1 ; } &

for table in PART SUPPLIER PARTSUPP CUSTOMER ORDERS
do
    nice dw "$DW_ENV" loader -p -q -l csv2dwa -s $TPC_LOCALDIR/metafile.$table.csv -d $user:$schema:$pass@$DW_CONN -a -t $table -c "`eval echo \${$table_COLUMNS}`" | sed -e "s/\([0-9]\{0-9\}*\)[ \t]*rows loaded\./$table rows loaded; \1/g" || { echo "Initial loading failed in table $table." ; exit 1 ; }
done

for table in NATION REGION
do
    nice dw "$DW_ENV" loader -p -q -l csv2dwa -s ${TPC_SHAREDIR}/Database_SF${SF}/metafile.$table.csv -d $user:$schema:$pass@$DW_CONN -a -t $table -c "`eval echo \${$table_COLUMNS}`" | sed -e "s/\([0-9]\{0-9\}*\)[ \t]*rows loaded\./$table rows loaded; \1/g" || { echo "Initial loading failed in table $table." ; exit 1 ; }
done

# wait until all tables where loaded
wait
sync
exit 0

```

Appendix C: ACID Scripts

```
-----
# tpc_h_load_db_for_acid_tests.sh
# -----
#!/bin/bash
# param <dw environment> <roothost> <scalefactor> <instances per node> <logdir>

if [ $# -ne 5 ]
then
    echo "Usage: $0 <dw environment> <roothost> <scalefactor> <instances per node> <logdir>"
    exit 1
fi

ENVIRONMENT="$1"
ROOTHOST="$2"
SF="$3"
STREAMS="$4"
INST_PN="$4"
LOGDIR="$5/`date +%y%m%d%H%M%S`"
LOGFILE="$LOGDIR/test.log"
ROOTPORT=26822

export TPC_BASEDIR=/shares/dwstaging/benchmark
export TPC_SCRIPTSDIR=$TPC_BASEDIR/scripts
export TPC_SHAREDIR=$TPC_BASEDIR/shares
export TPC_SQLDIR=$TPC_BASEDIR/sql
export DSS_CONFIG=$TPC_BASEDIR/tpch_20070105
export TPC_ACID_SQLDIR=$TPC_BASEDIR/ACID/sql

### Startup of database
mkdir -p $LOGDIR
echo "Loading of TPC H database started" >$LOGFILE
echo "-----" >>$LOGFILE
echo "SCRIPT : $0" >> $LOGFILE
echo "SCALE : $SF" >>$LOGFILE
echo "LOGDIR : $LOGDIR" >>$LOGFILE
echo "SCRIPTS : $TPC_SCRIPTSDIR" >>$LOGFILE
echo "-----" >>$LOGFILE

echo >> $LOGFILE
echo "Start of Test at ; `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\(...\)\...../1/g`" >>$LOGFILE

### create DB and user

dw $ENVIRONMENT create
cat $TPC_SQLDIR/create_user.sql | dwapplus -p 36822 -h $ROOTHOST -u sys -P exasol >>$LOGDIR/create_user.log 2>&1
dw $ENVIRONMENT shutdown
dw $ENVIRONMENT startup

##### Phase I (Load-Test)

### Initialization of database-schema

TIMESFILE=$LOGDIR/alltimes.log
LOAD_START=`date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\(...\)\...../1/g`"
echo -n "Load Test; $LOAD_START " > $TIMESFILE
echo "Start of Load test at ; $LOAD_START" >> $LOGFILE
cat $TPC_SQLDIR/create_schema.sql | dwapplus -p $ROOTPORT -h $ROOTHOST -u tpcuser -P tpcuser >>$LOGDIR/create_schema.log 2>&1

### Load

$TPC_SCRIPTSDIR/load_init.sh $ENVIRONMENT tpcuser tpcuser tpc $SF $INST_PN $LOGDIR >>$LOGFILE 2>&1

### Initialization of indices

cat $TPC_SQLDIR/create_indices.sql | dwapplus -p $ROOTPORT -h $ROOTHOST -u tpcuser -P tpcuser >>$LOGDIR/create_indices.log 2>&1
LOAD_END=`date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\(...\)\...../1/g`"
echo "End of Load test at ; $LOAD_END" >> $LOGFILE
echo -n " ; $LOAD_END " >> $TIMESFILE
LOAD_TIME=""`$TPC_SCRIPTSDIR/timediff "$LOAD_START" "$LOAD_END`""
echo " ; $LOAD_TIME" >> $TIMESFILE
sync

# execute tables.sql
cat $TPC_SQLDIR/tables.sql | dwapplus -p $ROOTPORT -h $ROOTHOST -u tpcuser -P tpcuser >>$LOGDIR/tables.log 2>&1

# execute dbcheck.sql
cat $TPC_SQLDIR/dbcheck.sql | dwapplus -p $ROOTPORT -h $ROOTHOST -u tpcuser -P tpcuser >>$LOGDIR/dbcheck.log 2>&1

# create history table
cat $TPC_ACID_SQLDIR/create_history_table.sql | dwapplus -p $ROOTPORT -h $ROOTHOST -u tpcuser -P tpcuser >>$LOGFILE 2>&1

echo "End of Test at ; " `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\(...\)\...../1/g`" >>$LOGFILE

# -----
# run_acid.sh
# -----
#!/bin/bash
# params <roothost> <scalefactor> <streams> <logdir> <durability_tests>

if [ ! $# -eq 5 ]
then
    echo "usage: $0 <roothost> <scalefactor> <streams> <logdir> <durability_tests>"
    exit 1
fi
```

```

ROOTHOST="$1"
SCALEFACTOR="$2"
STREAMS="$3"
LOGDIR="$4/ACID"
NUM_DURABILITY="$5"

ROOTPORT=26822

mkdir -p $LOGDIR

ACIDDIR="/shares/dwstaging/benchmark/ACID"
TMPDIR=$LOGDIR/tmp
mkdir -p $TMPDIR

echo "TPC Benchmark H - ACID-Test" | tee $LOGDIR/ACID.log
echo "===== " | tee -a $LOGDIR/ACID.log
echo | tee -a $LOGDIR/ACID.log

## create the history table
#echo "creating history table" | tee -a $LOGDIR/ACID.log
#dwaplus -h $ROOTHOST -p $ROOTPORT -u sys -P exasol -f $ACIDDIR/sql/create_history_table.sql >> /dev/null
echo | tee -a $LOGDIR/ACID.log

## ATOMICITY TEST
A_TRANSACTIONS_COMMIT=25
A_TRANSACTIONS_ROLLBACK=25

$ACIDDIR/scripts/run_atomicity_test.sh $ROOTHOST $SCALEFACTOR $LOGDIR $A_TRANSACTIONS_COMMIT $A_TRANSACTIONS_ROLLBACK | tee -
a $LOGDIR/ACID.log

## CONSISTENCY TEST
# $ACIDDIR/scripts/run_consistency_test.sh $ROOTHOST $SCALEFACTOR $STREAMS $LOGDIR | tee -a $LOGDIR/ACID.log

## ISOLATION TEST
$ACIDDIR/scripts/run_isolation_test.sh $ROOTHOST $SCALEFACTOR $LOGDIR | tee -a $LOGDIR/ACID.log

## DURABILITY TEST
$ACIDDIR/scripts/run_durability_test.sh $ROOTHOST $SCALEFACTOR $STREAMS $LOGDIR $NUM_DURABILITY | tee -a $LOGDIR/ACID.log

echo | tee -a $LOGDIR/ACID.log

echo "===== " | tee -a $LOGDIR/ACID.log
echo "Testresult ACID Test" | tee -a $LOGDIR/ACID.log
echo "===== " | tee -a $LOGDIR/ACID.log

if [ -z "`grep 'failed' $LOGDIR/ACID.log`" ]
then
    echo "ACID test passed" | tee -a $LOGDIR/ACID.log
else
    echo "ACID test failed" | tee -a $LOGDIR/ACID.log
fi

# -----
# run_atomicity_test.sh
# -----
#!/bin/bash
# params <roothost> <scalefactor> <logdir> <transactionsForTest1> <transactionsForTest2>

ROOTHOST="$1"
SCALEFACTOR="$2"
LOGDIR="$3/atomicity"
TRANSACTIONSCOMMIT="$4"
TRANSACTIONSROLLBACK="$5"

mkdir -p $LOGDIR
mkdir -p $LOGDIR/tmp
ACIDDIR="/shares/dwstaging/benchmark/ACID"
LOGFILE=$LOGDIR/atomicity.log

echo "*****" | tee $LOGFILE
echo "ATOMICITY_TEST" | tee -a $LOGFILE
echo "*****" | tee -a $LOGFILE

$ACIDDIR/scripts/atomicity_test_1.sh $ROOTHOST $SCALEFACTOR $LOGDIR $TRANSACTIONSCOMMIT | tee -a $LOGFILE
$ACIDDIR/scripts/atomicity_test_2.sh $ROOTHOST $SCALEFACTOR $LOGDIR $TRANSACTIONSROLLBACK | tee -a $LOGFILE

if [ -z "`grep 'failed' $LOGFILE`" ]
then
    echo "Atomicity test passed" | tee -a $LOGFILE
else
    echo "Atomicity test failed" | tee -a $LOGFILE
fi

echo | tee -a $LOGFILE

# -----
# atomicity_test_1.sh
# -----
#!/bin/bash
# executes atomicity test 1
# param: <roothost> <scalefactor> <logdir> <numberOfTransactions>

ROOTHOST="$1"
SCALEFACTOR="$2"
LOGDIR="$3"
NUMUPDATES="$4"

ACIDDIR="/shares/dwstaging/benchmark/ACID"

```



```

$ACIDDIR/scripts/atomicity_test_mainpart.sh COMMIT $ROOTHOST $SCALEFACTOR $LOGDIR 1 $NUMUPDATES
LINESCHANGED=`wc -l $LOGDIR/diff_atomicity_1.diff`

if [ $LINESCHANGED -eq $((4+1+$NUMUPDATES+4+(4*$NUMUPDATES)+2+(2*$NUMUPDATES)) ) ]
then
    echo "Atomicity Test 1 passed ($LINESCHANGED differences in diff file)"
else
    if [ ! $LINESCHANGED -eq 4 ]
    then
        echo "Atomicity Test 1 resulted in changed lines (might have failed), but diff file contains different
number of lines ($LINESCHANGED) then expected ($((4+1+$NUMUPDATES+4+(4*$NUMUPDATES)+2+(2*$NUMUPDATES))) ). Check diff file,
please, since test might have succeeded."
    else
        echo "Atomicity Test 1 failed ($LINESCHANGED differences in diff file which very probably means no lines
where changed)."

```

```

# generate transaction for isolation test 1
$ACIDDIR/scripts/create_random_transactions.sh 0 1 $$F $HOST $TMPDIR $LOGDIR/transaction_keys_isolation1.keys
# execute isolation test 1
$ACIDDIR/scripts/isolation1.sh $LOGDIR/transaction_keys_isolation1.keys $HOST $LOGDIR | tee -a $LOGDIR/isolation.log

## ISOLATION TEST 2
# generate transaction for isolation test 2
$ACIDDIR/scripts/create_random_transactions.sh 0 1 $$F $HOST $TMPDIR $LOGDIR/transaction_keys_isolation2.keys
# execute isolation test 2
$ACIDDIR/scripts/isolation2.sh $LOGDIR/transaction_keys_isolation2.keys $HOST $LOGDIR | tee -a $LOGDIR/isolation.log

## ISOLATION TEST 3
# generate transaction for isolation test 3
$ACIDDIR/scripts/create_random_transactions.sh 0 1 $$F $HOST $TMPDIR $LOGDIR/transaction_keys_isolation3.keys
# execute isolation test 3
$ACIDDIR/scripts/isolation3.sh $LOGDIR/transaction_keys_isolation3.keys $HOST $LOGDIR | tee -a $LOGDIR/isolation.log

## ISOLATION TEST 4
# generate transaction for isolation test 4
$ACIDDIR/scripts/create_random_transactions.sh 0 1 $$F $HOST $TMPDIR $LOGDIR/transaction_keys_isolation4.keys
# execute isolation test 4
$ACIDDIR/scripts/isolation4.sh $LOGDIR/transaction_keys_isolation4.keys $HOST $LOGDIR | tee -a $LOGDIR/isolation.log

## ISOLATION TEST 5
# generate transaction for isolation test 5
$ACIDDIR/scripts/create_random_transactions.sh 0 1 $$F $HOST $TMPDIR $LOGDIR/transaction_keys_isolation5.keys
# execute isolation test 5
$ACIDDIR/scripts/isolation5.sh $LOGDIR/transaction_keys_isolation5.keys $HOST $LOGDIR | tee -a $LOGDIR/isolation.log

## ISOLATION TEST 6
# generate transaction for isolation test 6
$ACIDDIR/scripts/create_random_transactions.sh 0 1 $$F $HOST $TMPDIR $LOGDIR/transaction_keys_isolation6.keys
# execute isolation test 6
$ACIDDIR/scripts/isolation6.sh $LOGDIR/transaction_keys_isolation6.keys $HOST $LOGDIR | tee -a $LOGDIR/isolation.log

## determine test result
if [ -z "`grep 'failed' $LOGDIR/isolation.log`" ]
then
    echo "Isolation test passed"
else
    echo "Isolation test failed. See $LOGDIR/isolation.log for details."
fi

# -----
# isolation1.sh
# -----
#!/bin/bash
# params: <inputfile> <host> <logdir> <keypress>

INPUTFILE="$1"
HOST="$2"
LOGDIR="$3"
KEYPRESS="$4"

ROOTPORT=26822

OKEY=( `cat $INPUTFILE` )
SCRIPTSDIR="/shares/dwstaging/benchmark/ACID/scripts"
LOGFILE=$LOGDIR/isolation1.log

PIPENAME=$LOGDIR/tmp/ilpipe

mkfifo $PIPENAME

echo "-----" | tee $LOGFILE
echo "Isolation test 1" | tee -a $LOGFILE
echo "-----" | tee -a $LOGFILE
echo >> $LOGFILE
echo -n "O_KEY, L_KEY, DELTA = " >> $LOGFILE
cat $INPUTFILE >> $LOGFILE
echo >> $LOGFILE

# Step 1: start ACID query
echo "Step 1: ACID query" | tee -a $LOGFILE
echo Before query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/\(\.\.\.\)\.\.\.\./\1/g` >> $LOGFILE
RESULTS1="$SCRIPTSDIR/acidquery.sh $HOST $LOGDIR $OKEY | tee -a $LOGFILE"
echo After query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/\(\.\.\.\)\.\.\.\./\1/g` >> $LOGFILE
# Step 2: start update transaction on same row, wait before commit
# acidtransaction with wait and commit
if [ ! -z $KEYPRESS ]
then
    echo "Press enter to start transaction (Step 2)"
    read $X
fi

echo "Step 2: ACID transaction" | tee -a $LOGFILE
cat $PIPENAME | $SCRIPTSDIR/transaction_isolation/acidtransaction $HOST $ROOTPORT COMMIT WAIT $INPUTFILE
$LOGDIR/isolation1_return_t1.log tpcuser tpcuser 2>&1 | sed -u "s/^/Transaction 1:/g" | tee -a $LOGFILE | grep -A 1 "ERROR"
&
# wait for transaction to reach wait for commit
sleep 10

# Step 3: start ACID query again, verify results are still the old
if [ ! -z $KEYPRESS ]
then
    echo "Press enter to start ACID query again (Step 3)"
    read $X
fi

echo "Step 3: Same ACID query again" | tee -a $LOGFILE
echo Before query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/\(\.\.\.\)\.\.\.\./\1/g` >> $LOGFILE

```

```

RESULTS2="`$SCRIPTSDIR/acidquery.sh $HOST $LOGDIR $OKEY | tee -a $LOGFILE`"
echo After query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\(.....\)...../1/g'` >> $LOGFILE
if [ "$RESULTS1" != "$RESULTS2" ]
then
    echo "Failed: Results differ from results of first query." | tee -a $LOGFILE
    TESTRESULT="Test failed"
fi

# Step 4: commit the update transaction
if [ ! -z $KEYPRESS ]
then
    echo "Press enter to commit ACID transaction (Step 4)"
    read $X
fi
echo "Step 4: Commit of ACID transaction" | tee -a $LOGFILE
# send newline to acidtransaction 1 (to commit)
FD=8
eval "exec $FD>>$PIPENAME" # open pipe
echo >& $FD
eval "exec $FD>>- " >/dev/null 2>&1 # close pipe (EOF)
# wait for commit to finish
sleep 10

# Step 5: start ACID query a third time and verify that the new row is retrieved
if [ ! -z $KEYPRESS ]
then
    echo "Press enter to start ACID query a third time (Step 5)"
    read $X
fi
echo "Step 5: Same ACID query again" | tee -a $LOGFILE
echo Before query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\(.....\)...../1/g'` >> $LOGFILE
RESULTS3="`$SCRIPTSDIR/acidquery.sh $HOST $LOGDIR $OKEY | tee -a $LOGFILE`"
echo After query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\(.....\)...../1/g'` >> $LOGFILE
if [ "$RESULTS2" = "$RESULTS3" ]
then
    echo "Failed: Results are the same as those from the second query." | tee -a $LOGFILE
    TESTRESULT="Test failed"
fi

ERRORS="`grep 'ERROR' $LOGFILE`"
if [ \(( -z "$TESTRESULT" \) -a \(( -z "$ERRORS" \) ) ]
then
    echo "Isolation test 1 passed" | tee -a $LOGFILE
else
    echo "Isolation test 1 failed" | tee -a $LOGFILE
fi

rm -f $PIPENAME

# -----
# isolation2.sh
# -----
#!/bin/bash
# params: <inputfile> <host> <logdir> <keypress>

INPUTFILE="$1"
HOST="$2"
LOGDIR="$3"
KEYPRESS="$4"

ROOTPORT=26822

OKEY=( `cat $INPUTFILE` )
SCRIPTSDIR="/shares/dwstaging/benchmark/ACID/scripts"
LOGFILE=$LOGDIR/isolation2.log

PIPENAME=$LOGDIR/tmp/i2pipe
mkfifo $PIPENAME

echo "-----" | tee $LOGFILE
echo "Isolation test 2" | tee -a $LOGFILE
echo "-----" | tee -a $LOGFILE
echo >> $LOGFILE
echo -n "O_KEY, L_KEY, DELTA = " >> $LOGFILE
cat $INPUTFILE >> $LOGFILE
echo >> $LOGFILE

# Step 1: start ACID query
echo "Step 1: ACID query" | tee -a $LOGFILE
echo Before query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\(.....\)...../1/g'` >> $LOGFILE
RESULTS1="`$SCRIPTSDIR/acidquery.sh $HOST $LOGDIR $OKEY | tee -a $LOGFILE`"
echo After query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\(.....\)...../1/g'` >> $LOGFILE

# Step 2: start update transaction on same row, wait before commit
# acidtransaction with wait and commit
if [ ! -z $KEYPRESS ]
then
    echo "Press enter to start transaction (Step 2)"
    read $X
fi

echo "Step 2: ACID transaction" | tee -a $LOGFILE
cat $PIPENAME | $SCRIPTSDIR/transaction_isolation/acidtransaction $HOST $ROOTPORT ROLLBACK WAIT $INPUTFILE
$LOGDIR/isolation2_return_t1.log tpcuser tpcuser 2>&1 | sed -u 's/^Transaction 1:/g' | tee -a $LOGFILE | grep -A 1 "ERROR" &
# wait for transaction to reach wait for commit
sleep 10

# Step 3: start ACID query again, verify results are still the old
if [ ! -z $KEYPRESS ]

```

```

then
    echo "Press enter to start ACID query again (Step 3)"
    read $X
fi
echo "Step 3: Same ACID query again" | tee -a $LOGFILE
echo Before query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/\(\...\)\...../1/g'` >> $LOGFILE
RESULTS2=`$SCRIPTSDIR/acidquery.sh $HOST $LOGDIR $OKEY | tee -a $LOGFILE`
echo After query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/\(\...\)\...../1/g'` >> $LOGFILE
if [ "$RESULTS1" != "$RESULTS2" ]
then
    echo "Failed: Results differ from results of first query." | tee -a $LOGFILE
    TESTRESULT="Test failed"
fi

# Step 4: rollback the update transaction
if [ ! -z $KEYPRESS ]
then
    echo "Press enter to rollback ACID transaction (Step 4)"
    read $X
fi
echo "Step 4: Rollback of ACID transaction" | tee -a $LOGFILE
# send newline to acidtransaction 1 (to rollback)
FD=8
eval "exec $FD>>$PIPIENAME" # open pipe
echo >& $FD
eval "exec $FD>>- >/dev/null 2>&1 # close pipe (EOF)
# wait for commit to finish
sleep 10

# Step 5: start ACID query a third time and verify that the new row is retrieved
if [ ! -z $KEYPRESS ]
then
    echo "Press enter to start ACID query a third time (Step 5)"
    read $X
fi
echo "Step 5: Same ACID query again" | tee -a $LOGFILE
echo Before query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/\(\...\)\...../1/g'` >> $LOGFILE
RESULTS3=`$SCRIPTSDIR/acidquery.sh $HOST $LOGDIR $OKEY | tee -a $LOGFILE`
echo After query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/\(\...\)\...../1/g'` >> $LOGFILE
if [ "$RESULTS2" != "$RESULTS3" ]
then
    echo "Failed: Results differ from those of the second query." | tee -a $LOGFILE
    TESTRESULT="Test failed"
fi

# determine test result
ERRORS="grep 'ERROR' $LOGFILE"
if [ \(! -z "$TESTRESULT" \) -a \(! -z "$ERRORS" \) ]
then
    echo "Isolation test 2 passed" | tee -a $LOGFILE
else
    echo "Isolation test 2 failed" | tee -a $LOGFILE
fi

rm -f $PIPIENAME

# -----
# isolation3.sh
# -----
#!/bin/bash
# params: <inputfile> <host> <logdir> <keypress>

INPUTFILE="$1"
HOST="$2"
LOGDIR="$3"
KEYPRESS="$4"

ROOTPORT=26822

OKEY=( `cat $INPUTFILE` )
SCRIPTSDIR="/shares/dwstaging/benchmark/ACID/scripts"
LOGFILE=$LOGDIR/isolation3.log

PIPIENAME=$LOGDIR/tmp/i3pipe

mkfifo $PIPIENAME

echo "-----" | tee $LOGFILE
echo "Isolation test 3" | tee -a $LOGFILE
echo "-----" | tee -a $LOGFILE
echo >> $LOGFILE
echo -n "O_KEY, L_KEY, DELTA = " >> $LOGFILE
cat $INPUTFILE >> $LOGFILE
echo >> $LOGFILE

# Step 1: start a query and verify that the row was retrieved
echo "Step 1: ACID query" | tee -a $LOGFILE
echo Before query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/\(\...\)\...../1/g'` >> $LOGFILE
RESULTS1=`$SCRIPTSDIR/acidquery.sh $HOST $LOGDIR $OKEY | tee -a $LOGFILE`
echo After query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/\(\...\)\...../1/g'` >> $LOGFILE

# Step 2: start update transaction, wait before commit
if [ ! -z $KEYPRESS ]
then
    echo "Press enter to start ACID transaction 1 (Step 2)"
    read $X
fi
echo "Step 2: ACID transaction 1 until commit" | tee -a $LOGFILE
cat $PIPIENAME | $SCRIPTSDIR/transaction_isolation/acidtransaction $HOST $ROOTPORT COMMIT WAIT $INPUTFILE
$LOGDIR/isolation3_return.tl.log tpcuser tpcuser | sed -u 's/^/Transaction 1:/g' | tee -a $LOGFILE | grep -A 1
"(ERROR/ROLLBACK)" &

```

```

sleep 10
if [ ! -z "`grep '\(ERROR\|ROLLBACK\)\' $LOGFILE`" ]
then
    ERROR="$ERROR Transaction 1"
fi

# Step 3: start another update transaction on same row. Verify rollback
if [ ! -z "$KEYPRESS" ]
then
    echo "Press enter to start ACID transaction 2 (Step 3)"
    read $X
fi
echo "Step 3: ACID transaction 2 with COMMIT (expecting ROLLBACK exception)" | tee -a $LOGFILE
$SCRIPTSDIR/transaction_isolation/acidtransaction $HOST $ROOTPORT COMMIT NOWAIT $INPUTFILE $LOGDIR/isolation3_return_t2.log
tpcuser tpcuser | sed 's/^/Transaction 2:/g' | tee -a $LOGFILE | grep -A 1 "ROLLBACK" > $LOGDIR/isolation3_rollback_t2.log
ROLLBACK="`grep 'ROLLBACK' $LOGDIR/isolation3_rollback_t2.log`"
if [ -z "$ROLLBACK" ]
then
    ERROR="$ERROR No Rollback in Transaction 2"
fi

if [ ! -z "`grep 'ERROR' $LOGFILE`" ]
then
    ERROR="$ERROR Transaction 2"
fi

# Step 4: commit the update transaction
if [ ! -z "$KEYPRESS" ]
then
    echo "Press enter to commit ACID transaction 1 (Step 4)"
    read $X
fi
echo "Step 4: Commit of ACID transaction 1" | tee -a $LOGFILE
# send newline to acidtransaction 1 (to commit)
FD=8
eval "exec $FD>>$PIPENAME" # open pipe
echo >& $FD
eval "exec $FD>>- " >/dev/null 2>&1 # close pipe (EOF)
sleep 10
if [ ! -z "`grep '\(ERROR\|ROLLBACK\)\' $LOGFILE | grep 'Transaction 1\'`" ]
then
    ERROR="$ERROR Transaction 1"
fi

# Step 5: start the same query again and verify that the new row was retrieved
if [ ! -z "$KEYPRESS" ]
then
    echo "Press enter to start ACID query again (Step 5)"
    read $X
fi
echo "Step 5: ACID query again" | tee -a $LOGFILE
echo "Before query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/\(\\.\.\.\.\)\.\.\.\.\./\1/g`" >> $LOGFILE
RESULTS2="$SCRIPTSDIR/acidquery.sh $HOST $LOGDIR $OKEY | tee -a $LOGFILE`"
echo "After query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/\(\\.\.\.\.\)\.\.\.\.\./\1/g`" >> $LOGFILE
if [ "$RESULTS1" = "$RESULTS2" ]
then
    ERROR="$ERROR Results compare"
fi

# determine test result
if [ ! -z "$ERROR" ]
then
    echo "Isolation test 3 failed." | tee -a $LOGFILE
    echo "$ERROR"
else
    echo "Isolation test 3 passed." | tee -a $LOGFILE
fi

rm -f $PIPENAME

# -----
# isolation4.sh
# -----
#!/bin/bash
# params: <inputfile> <host> <logdir> <keypress>

INPUTFILE="$1"
HOST="$2"
LOGDIR="$3"
KEYPRESS="$4"

ROOTPORT=26822

OKEY=( `cat $INPUTFILE` )
SCRIPTSDIR="/shares/dwstaging/benchmark/ACID/scripts"
LOGFILE="$LOGDIR/isolation4.log"

PIPENAME=$LOGDIR/tmp/i4pipe

mkfifo $PIPENAME

echo "-----" | tee $LOGFILE
echo "Isolation test 4" | tee -a $LOGFILE
echo "-----" | tee -a $LOGFILE
echo >> $LOGFILE
echo -n "O_KEY, L_KEY, DELTA = " >> $LOGFILE
cat $INPUTFILE >> $LOGFILE
echo >> $LOGFILE

# Step 1: start a query and verify that the row was retrieved
echo "Step 1: ACID query" | tee -a $LOGFILE

```

```

echo Before query `date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(....\))...../1/g'` >> $LOGFILE
RESULTS1=`$SCRIPTSDIR/acidquery.sh $HOST $LOGDIR $OKEY | tee -a $LOGFILE`
echo After query `date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(....\))...../1/g'` >> $LOGFILE

# Step 2: start update transaction, wait before commit
if [ ! -z "$KEYPRESS" ]
then
    echo "Press enter to start ACID transaction 1 (Step 2)"
    read $X
fi
echo "Step 2: ACID transaction 1 until commit" | tee -a $LOGFILE
cat $PIPENAME | $SCRIPTSDIR/transaction_isolation/acidtransaction $HOST $ROOTPORT ROLLBACK WAIT $INPUTFILE
$LOGDIR/isolation4_return_t1.log tpcuser tpcuser | sed -u 's/^/Transaction 1:/g' | tee -a $LOGFILE | grep -A 1 "ERROR" &
sleep 10
if [ ! -z "`grep '(ERROR|ROLLBACK)' $LOGFILE`" ]
then
    ERROR="$ERROR Transaction 1"
fi

# Step 3: start another update transaction on same row. Verify rollback
if [ ! -z "$KEYPRESS" ]
then
    echo "Press enter to start ACID transaction 2 (Step 3)"
    read $X
fi
echo "Step 3: ACID transaction 2 with COMMIT (expecting ROLLBACK exception)" | tee -a $LOGFILE
$SCRIPTSDIR/transaction_isolation/acidtransaction $HOST $ROOTPORT COMMIT NOWAIT $INPUTFILE $LOGDIR/isolation4_return_t2.log
tpcuser tpcuser | sed -u 's/^/Transaction 2:/g' | tee -a $LOGFILE | grep -A 1 "ROLLBACK" > $LOGDIR/isolation4_rollback_t2.log
ROLLBACK=`grep 'ROLLBACK' $LOGDIR/isolation4_rollback_t2.log`
if [ -z "$ROLLBACK" ]
then
    ERROR="$ERROR No Rollback in Transaction 2"
fi
if [ ! -z "`grep 'ERROR' $LOGFILE`" ]
then
    ERROR="$ERROR Transaction 2"
fi

# Step 4: commit the update transaction
if [ ! -z "$KEYPRESS" ]
then
    echo "Press enter to rollback ACID transaction 1 (Step 4)"
    read $X
fi
echo "Step 4: Rollback of ACID transaction 1" | tee -a $LOGFILE
# send newline to acidtransaction 1 (to rollback)
FD=8
eval "exec $FD>>$PIPENAME" # open pipe
echo >& $FD
eval "exec $FD>>-" >/dev/null 2>&1 # close pipe (EOF)
sleep 10
if [ ! -z "`grep 'ERROR' $LOGFILE | grep 'Transaction 1'`" ]
then
    ERROR="$ERROR Transaction 1"
fi

# Step 5: start the same query again and verify that the new row was retrieved
if [ ! -z "$KEYPRESS" ]
then
    echo "Press enter to start ACID query again (Step 5)"
    read $X
fi
echo "Step 5: ACID query again" | tee -a $LOGFILE
echo Before query `date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(....\))...../1/g'` >> $LOGFILE
RESULTS2=`$SCRIPTSDIR/acidquery.sh $HOST $LOGDIR $OKEY | tee -a $LOGFILE`
echo After query `date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(....\))...../1/g'` >> $LOGFILE
if [ "$RESULTS1" != "$RESULTS2" ]
then
    ERROR="$ERROR Results compare"
fi

## determine test result
if [ ! -z "$ERROR" ]
then
    echo "Isolation test 4 failed." | tee -a $LOGFILE
    echo "$ERROR"
else
    echo "Isolation test 4 passed." | tee -a $LOGFILE
fi

rm -f $PIPENAME

# -----
# isolation5.sh
# -----
#!/bin/bash
# params: <inputfile> <host> <logdir>

INPUTFILE="$1"
HOST="$2"
LOGDIR="$3"

ROOTPORT=26822

OKEY=( `cat $INPUTFILE` )
SCRIPTSDIR="/shares/dwstaging/benchmark/ACID/scripts"
LOGFILE=$LOGDIR/isolation5.log

PIPENAME=$LOGDIR/tmp/i5pipe

mkfifo $PIPENAME

```

```

echo "-----" | tee $LOGFILE
echo "Isolation test 5" | tee -a $LOGFILE
echo "-----" | tee -a $LOGFILE
echo >> $LOGFILE
echo -n "O_KEY, L_KEY, DELTA = " >> $LOGFILE
cat $INPUTFILE >> $LOGFILE
echo >> $LOGFILE

echo "Copying lineitem to lineitem2, orders to orders2, history to history2" | tee -a $LOGFILE

# before: Copying lineitem to lineitem2
#         Copying orders to orders2
#         Copying history to history2
$SCRIPTSDIR/copyTablesForIsolation5.sh $HOST $LOGDIR/tmp >> $LOGFILE

# Step 1: Start query against first table
echo "Step 1: ACID query on LINEITEM" | tee -a $LOGFILE
echo Before query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\.....)\...../1/g' >> $LOGFILE
RESULT1Q1="$SCRIPTSDIR/acidquery.sh $HOST $LOGDIR $OKEY | tee -a $LOGFILE"
echo After query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\.....)\...../1/g' >> $LOGFILE

# Step 2: Start query against second table
echo "Step 2: ACID query on LINEITEM2" | tee -a $LOGFILE
echo Before query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\.....)\...../1/g' >> $LOGFILE
RESULT1Q2="$SCRIPTSDIR/acidquery.sh $HOST $LOGDIR $OKEY LINEITEM2 | tee -a $LOGFILE"
echo After query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\.....)\...../1/g' >> $LOGFILE

# Step 3: Start update transaction against first tables, wait before commit
if [ ! -z "$KEYPRESS" ]
then
    echo "Press enter to start ACID transaction 1 (on LINEITEM, ORDERS) (Step 3)"
    read $X
fi
echo "Step 3: ACID transaction 1 (on LINEITEM, ORDERS) until commit" | tee -a $LOGFILE
cat $PIPENAME | $SCRIPTSDIR/transaction_isolation/acidtransaction $HOST $ROOTPORT COMMIT WAIT $INPUTFILE
$LOGDIR/isolation5_return_t1.log tpcuser tpcuser | sed -u 's/^/Transaction 1:/g' | tee -a $LOGFILE | grep -A 1
"\(ROLLBACK\|ERROR\)" &
sleep 10
if [ ! -z "`grep '\(ROLLBACK\|ERROR\)' $LOGFILE`" ]
then
    ERROR="$ERROR Transaction1"
fi

# Step 4: Start update transaction against second tables and commit
if [ ! -z "$KEYPRESS" ]
then
    echo "Press enter to start ACID transaction 2 (on LINEITEM2, ORDERS2) (Step 4)"
    read $X
fi
echo "Step 4: ACID transaction 2 (on LINEITEM2, ORDERS2) with COMMIT" | tee -a $LOGFILE
$SCRIPTSDIR/transaction_isolation/acidtransaction $HOST $ROOTPORT COMMIT NOWAIT $INPUTFILE $LOGDIR/isolation5_return_t2.log
tpcuser tpcuser lineitem2 orders2 history2 | sed -u 's/^/Transaction 2:/g' | tee -a $LOGFILE | grep -A 1
"\(ROLLBACK\|ERROR\)"
if [ ! -z "`grep '\(ROLLBACK\|ERROR\)' $LOGFILE | grep 'Transaction 2'`" ]
then
    ERROR="$ERROR Transaction2"
fi

# Step 5: Commit first update transaction
if [ ! -z "$KEYPRESS" ]
then
    echo "Press enter to commit ACID transaction 1 (Step 5)"
    read $X
fi
echo "Step 5: Commit of ACID transaction 1" | tee -a $LOGFILE
# send newline to acidtransaction 1 (to rollback)
FD=8
eval "exec $FD>>$PIPENAME" # open pipe
echo >& $FD
eval "exec $FD>>- >/dev/null 2>&1 # close pipe (EOF)"
sleep 10
if [ ! -z "`grep '\(ROLLBACK\|ERROR\)' $LOGFILE | grep 'Transaction 1'`" ]
then
    ERROR="$ERROR Transaction1"
fi

# Step 6: Start query against first table again (results should differ from first time)
if [ ! -z "$KEYPRESS" ]
then
    echo "Press enter to start ACID query on LINEITEM again (Step 6)"
    read $X
fi
echo "Step 6: ACID query again (on LINEITEM)" | tee -a $LOGFILE
echo Before query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\.....)\...../1/g' >> $LOGFILE
RESULT2Q1="$SCRIPTSDIR/acidquery.sh $HOST $LOGDIR $OKEY | tee -a $LOGFILE"
echo After query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\.....)\...../1/g' >> $LOGFILE
if [ "$RESULT1Q1" = "$RESULT2Q1" ]
then
    ERROR="$ERROR Compare results for first table"
fi

# Step 7: Start query against second table again (results should differ from first time)
if [ ! -z "$KEYPRESS" ]
then
    echo "Press enter to start ACID query on LINEITEM2 again (Step 7)"
    read $X
fi
echo "Step 7: ACID query again (on LINEITEM2)" | tee -a $LOGFILE
echo Before query `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\.....)\...../1/g' >> $LOGFILE

```

```

RESULT2Q2="" $SCRIPTSDIR/acidquery.sh $HOST $LOGDIR $OKEY LINEITEM2 | tee -a $LOGFILE`"
echo After query `date +%m/%d/%Y %k:%M:%S.%N' | sed 's/\(\..\..\)\.\.\.\./\1/g'` >> $LOGFILE
if [ "$RESULT1Q2" = "$RESULT2Q2" ]
then
    ERROR="$ERROR Compare results for second table"
fi

if [ -z "$ERROR" ]
then
    echo "Isolation test 5 passed"
else
    echo "Isolation test 5 failed"
    echo $ERROR
fi

echo "Deleting lineitem2, orders2, history2" | tee -a $LOGFILE
$SCRIPTSDIR/deleteCopiedTablesForIsolation5.sh $HOST $LOGDIR/tmp >> $LOGFILE

rm -f $PIPENAME

# -----
# copyTablesForIsolation5.sh
# -----
#!/bin/bash
# params: <HOST> <TMPDIR>

HOST=$1
TMPDIR=$2

ROOTPORT=26822

TMPFILE=$TMPDIR/tmp_copy.sql

echo "set autocommit off;" > $TMPFILE
echo "open schema tpc;" > $TMPFILE
echo "create table LINEITEM2 as select * from lineitem;" >> $TMPFILE
echo "create table ORDERS2 as select * from orders;" >> $TMPFILE
echo "create table HISTORY2 as select * from history;" >> $TMPFILE
echo "commit;" >> $TMPFILE

dwaplus -h $HOST -p $ROOTPORT -u tpcuser -P tpcuser -f $TMPFILE
rm -f $TMPFILE

# -----
# deleteCopiedTablesForIsolation5.sh
# -----
#!/bin/bash
# params: <HOST> <TMPDIR>

HOST=$1
TMPDIR=$2

ROOTPORT=26822

TMPFILE=$TMPDIR/tmp_copy.sql

echo "set autocommit off;" > $TMPFILE
echo "open schema tpc;" >> $TMPFILE
echo "drop table LINEITEM2 cascade;" >> $TMPFILE
echo "drop table ORDERS2 cascade;" >> $TMPFILE
echo "drop table HISTORY2 cascade;" >> $TMPFILE
echo "commit;" >> $TMPFILE

dwaplus -h $HOST -p $ROOTPORT -u tpcuser -P tpcuser -f $TMPFILE
rm -f $TMPFILE

# -----
# isolation6.sh
# -----
#!/bin/bash
# params: <inputfile> <host> <logdir>

INPUTFILE="$1"
HOST="$2"
LOGDIR="$3"

ROOTPORT=26822

OKEY=( `cat $INPUTFILE` )
SCRIPTSDIR="/shares/dwstaging/benchmark/ACID/scripts"
LOGFILE=$LOGDIR/isolation6.log

echo "-----" | tee $LOGFILE
echo "Isolation test 6" | tee -a $LOGFILE
echo "-----" | tee -a $LOGFILE
echo >> $LOGFILE
echo -n "O_KEY, L_KEY, DELTA = " >> $LOGFILE
cat $INPUTFILE >> $LOGFILE
echo >> $LOGFILE

# Step 1: TPC-H query 1 with DELTA=0
echo "Step 1: TPC-H query 1 with DELTA=0" | tee -a $LOGFILE
$SCRIPTSDIR/tpchquery1ForIsolation6.sh $HOST $LOGDIR 0 >> $LOGFILE &
# wait until the query should be started
sleep 2

# Step 2: ACID transaction
echo "Step 2: ACID transaction" | tee -a $LOGFILE
STARTTIME_TA=`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/\(\..\..\)\.\.\.\./\1/g'`"

```



```

echo "starttime of ACID transaction: $STARTTIME_TA" >> $LOGFILE
$SCRIPTSDIR/transaction_isolation/acidtransaction $HOST $ROOTPORT COMMIT NOWAIT $INPUTFILE
$LOGDIR/tmp/isolation6_return.t1.log tpcuser tpcuser | sed -u 's/^/Transaction 2:/g' | tee -a $LOGFILE | grep
"\(ROLLBACK\|ERROR\)"
ENDTIME_TA="date +%m/%d/%Y %k:%M:%S.%N" | sed 's/\(.\{.\}\)...../1/g'`"
echo "endtime of ACID transaction: $ENDTIME_TA" >> $LOGFILE
wait

if [ ! -z "`grep '\(ROLLBACK\|ERROR\)' $LOGFILE | grep 'Transaction 2'" ]
then
    ERROR="ERROR Transaction 2"
fi

# ensure the acid transaction completed within the execution time of the query
# i.e. starttime_query < start_time_transaction, endtime_transaction < endtime_query
STARTTIME_QU="grep 'Q1' $LOGDIR/tmp/Q1Delta.log2 | sed 's/[^;]*; \([^;]*\);.*\1/'"
ENDTIME_QU="grep 'Q1' $LOGDIR/tmp/Q1Delta.log2 | sed 's/[^;]*;[^;]*; \([^;]*\);.*\1/'"

echo "STARTTIME_TA=$STARTTIME_TA" >> $LOGFILE
echo "ENDTIME_TA=$ENDTIME_TA" >> $LOGFILE
echo "STARTTIME_QU=$STARTTIME_QU" >> $LOGFILE
echo "ENDTIME_QU=$ENDTIME_QU" >> $LOGFILE

# convert times to format "yyyymmddhhmissff3" for string comparison
START_QU="echo $STARTTIME_QU | sed 's/\([0-9][0-9]\)\.\.\.\(\([0-9][0-9]\)\.\.\.\(\([0-9][0-9][0-9][0-9]\)\)\3\1\1\2/' | sed 's/[
\t:./]*//g'"
END_QU="echo $ENDTIME_QU | sed 's/\([0-9][0-9]\)\.\.\.\(\([0-9][0-9]\)\.\.\.\(\([0-9][0-9][0-9][0-9]\)\)\3\1\1\2/' | sed 's/[
\t:./]*//g'"
START_TA="echo $STARTTIME_TA | sed 's/\([0-9][0-9]\)\.\.\.\(\([0-9][0-9]\)\.\.\.\(\([0-9][0-9][0-9][0-9]\)\)\3\1\1\2/' | sed 's/[
\t:./]*//g'"
END_TA="echo $ENDTIME_TA | sed 's/\([0-9][0-9]\)\.\.\.\(\([0-9][0-9]\)\.\.\.\(\([0-9][0-9][0-9][0-9]\)\)\3\1\1\2/' | sed 's/[
\t:./]*//g'"

echo "START_TA=$START_TA" >> $LOGFILE
echo "END_TA=$END_TA" >> $LOGFILE
echo "START_QU=$START_QU" >> $LOGFILE
echo "END_QU=$END_QU" >> $LOGFILE

if [ $START_TA -le $START_QU ]
then
    ERROR="$ERROR Transaction started before query"
fi

if [ $END_TA -ge $END_QU ]
then
    ERROR="$ERROR Transaction ended after query"
fi

if [ -z "$ERROR" ]
then
    echo "Isolation test 6 passed." | tee -a $LOGFILE
else
    echo "Isolation test 6 failed." | tee -a $LOGFILE
    echo "$ERROR" | tee -a $LOGFILE
fi

# -----
# tpchquery1ForIsolation6.sh
# -----
#!/bin/bash
# params: <HOST> <LOGDIR> <DELTA>

HOST="$1"
LOGDIR="$2"
DELTA="$3"

TMPDIR="$LOGDIR/tmp"
SQL="/shares/dwstaging/benchmark/ACID/sql/Q1Delta.sql"

QUERYSTREAMDIR="/shares/dwstaging/benchmark/scripts/query_streams/"

ROOTPORT=26822

# transform inputfile...
cp $SQL $TMPDIR/Q1NOReformat.sql
cat $TMPDIR/Q1NOReformat.sql | sed "s/:q/1/g" | sed "s/:s/1/g" | sed "s/:o//g" | sed "s/:e/COMMIT;/g" | sed "s/:1/$DELTA/g" |
sed "s/:r$/g" > $TMPDIR/Q1Delta.sql

# starting query
$QUERYSTREAMDIR/querystreamex $TMPDIR/Q1Delta.sql $TMPDIR/Q1Delta.log $TMPDIR/Q1Delta.log2 1 $HOST $ROOTPORT tpc tpcuser
tpcuser

# -----
# Q1Delta.sql
# -----
-- $ID$
-- TPC-H/TPC-R Pricing Summary Report Query (Q1)
-- Functional Query Definition
-- Approved February 1998

-- Minor modification - date arithmetic for days ( 2.2.3.3 c )

-- TPC-H Query :q :s
:0

select
    l1.l_quantity,
    sum(l2.l_extendedprice),
    sum(l3.l_extendedprice),
    sum(l3.l_quantity)

```

```

from
    lineitem l1, lineitem l2, lineitem l3
where
    l1.l_shipdate <= date '1998-12-01' - 0
    and l1.l_orderkey = l2.l_orderkey
    and l1.l_linenumber = l2.l_linenumber
    and l1.l_extendedprice = l3.l_extendedprice
    and l3.l_quantity<30
group by l1.l_quantity
;
:e

# -----
# transaction_isolation/acidtransaction.cpp
# -----
#include "querystream.h"

#include <iostream>
using std::cerr;
using std::cout;
using std::cin;
using std::endl;
using std::ostream;

#include <iomanip>

#include <fstream>
using std::ofstream;
using std::ifstream;

#include <string>
using std::string;

#include <sstream>
using std::ostringstream;

// for rand()
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>
#include <math.h>

#define MAX_LINE_SIZE 10000

#include <sys/time.h>

double EPS=0.00001;

double trunc( double d, size_t p )
{
    // calculating precFactor as 10^p (p>=0)
    unsigned long precFactor = 1;
    for ( size_t i=1; i<=p; i++ )
    {
        precFactor *= 10;
    }

    double truncation = (trunc(d*precFactor))/((double)precFactor);
    if ( ((truncation + (1.0/precFactor))-d) < EPS )
        truncation += (1.0/precFactor);
    return truncation;
}

/** \brief Calculates the time passed between t1 and t2. */
double timediff( timeval t1, timeval t2 )
{
    double passed = 1.0e-6 * ( t2.tv_usec - t1.tv_usec );
    passed += ( t2.tv_sec - t1.tv_sec );
    if ( passed < 0 )
        passed = -passed;
    return passed;
}

/** \brief Deletes from a given string all occurrences of another string at beginning and end. */
string trim( string &s, const string drop = " " )
{
    string r = s.erase( s.find_last_not_of(drop)+1 );
    return r.erase( 0, r.find_first_not_of(drop) );
}

void printTimestamp( ostream &mystream )
{
    timeval current_time;
    gettimeofday( &current_time, 0 );
    struct tm tm_time;
    localtime_r( &current_time.tv_sec, &tm_time );
    int year = 1900 + tm_time.tm_year;
    int month = tm_time.tm_mon;
    int day = tm_time.tm_mday;
    int hour = tm_time.tm_hour;
    int minute = tm_time.tm_min;
    int seconds = tm_time.tm_sec;
    mystream << std::setw(4) << std::setfill('0') << year << "-" << std::setw(2) << std::setfill('0') << month << "-" <<
    std::setw(2) << std::setfill('0') << day << " " << std::setw(2) << std::setfill('0') << hour << ":" << std::setw(2) <<
    std::setfill('0') << minute << ":" << std::setw(2) << std::setfill('0') << seconds << "." << std::setw(6) <<
    std::setfill('0') << current_time.tv_usec << "/";
}

void processRetSize( int retSize, int expectedRetSize, QueryStream &qs )
{
    if ( retSize != expectedRetSize )

```

```

    {
        if ( retSize == -3 )
        {
            cerr << "Unexpected ROLLBACK" << endl;
            throw rollbackException();
        }
        cerr << "!!! ERROR Unexpected return size. Expected " << expectedRetSize << " but got " << retSize << "...
        exiting. !!!" << endl;
        qs.disconnectFromServer();
        exit(-1);
    }
}

// executes a COMMIT
void executeCommit( QueryStream &qs )
{
    printTimestamp( cout );
    cout << endl;
    int retsize = qs.execute( "COMMIT;" );
    processRetSize( retsize, -1, qs );
}

// executes a ROLLBACK
void executeRollback( QueryStream &qs )
{
    printTimestamp( cout );
    cout << endl;
    int retsize = qs.execute( "ROLLBACK;" );
    processRetSize( retsize, -1, qs );
}

// Waits for a keypress
void doWait()
{
    printTimestamp( cout );
    cout << ": Press a key, please, to continue" << endl;
    cin.clear();
    char ch = cin.get();
}

// Does not wait
void noWait()
{
    printTimestamp( cout );
    cout << endl;
    system( "sleep 2" );
}

// executes the ACID-transaction for given values O_KEY, L_KEY and DELTA
// until the point where a COMMIT or a ROLLBACK can be executed.
// (old) values of rprice, quantity, tax, disc, extprice and ototal
// are returned.
// With exception of the update to writelock some table at the beginning
// quite identical to the pseudocode in the specification.
void performAcidTransactionVariant1( QueryStream &qs, string O_KEY, string L_KEY, int DELTA, int &pkey, int &skey, double
&rprice, double &quantity, double &tax, double &disc, double &extprice, double &ototal, string lineitemname, string
ordersname, string historyname )
{
    ostringstream queryText("");
    queryText.setf( std::ios::fixed );
    queryText.precision( 2 );
    int retSize;

    // Read from orders
    queryText.str("");
    queryText << "SELECT o_totalprice, o_orderkey FROM " << ordersname << " WHERE o_orderkey=" << O_KEY << ";";
    retSize = qs.execute( queryText.str() );
    processRetSize( retSize, 1, qs );
    ototal = qs.getDouble( 0 );
    qs.freeFetchArray();

    // Read from lineitem
    queryText.str("");
    queryText << "SELECT l_quantity, l_extendedprice, l_partkey, l_suppkey, l_tax, l_discount, l_orderkey, l_linenummer FROM
" << lineitemname << " WHERE l_orderkey=" << O_KEY << " AND l_linenummer=" << L_KEY << ";";
    retSize = qs.execute( queryText.str() );
    processRetSize( retSize, 1, qs );
    quantity = qs.getDouble( 0 );
    extprice = qs.getDouble( 1 );
    pkey = qs.getInt( 2 );
    skey = qs.getInt( 3 );
    tax = qs.getDouble( 4 );
    disc = qs.getDouble( 5 );
    qs.freeFetchArray();

    // Calculate new values
    ototal = ototal - trunc( trunc( extprice * ( 1-disc ), 2 ) * (1+tax), 2 );
    rprice = trunc( extprice/quantity, 2 );
    double cost = trunc( rprice * DELTA, 2 );
    double new_extprice = extprice + cost;
    double new_ototal = trunc( new_extprice * (1.0-disc), 2 );
    new_ototal = trunc( new_ototal * (1.0+tax), 2 );
    new_ototal = ototal + new_ototal;

    // Update lineitem
    queryText.str("");
    queryText << "UPDATE " << lineitemname << " ";
    queryText << "SET l_extendedprice=" << new_extprice << " ";
    queryText << "l_quantity=" << quantity + DELTA << " ";
    queryText << "WHERE l_orderkey=" << O_KEY << " AND l_linenummer=" << L_KEY << ";";
    retSize = qs.execute( queryText.str() );
}

```

```

processRetSize( retSize, -1, qs );

// Update orders
queryText.str("");
queryText << "UPDATE " << ordersname << " ";
queryText << "SET o_totalprice=" << new_ototal << " ";
queryText << "WHERE o_orderkey=" << O_KEY << " ";
retSize = qs.execute( queryText.str() );
processRetSize( retSize, -1, qs );

// Insert int history
queryText.str("");
queryText << "INSERT INTO " << historyname << " VALUES ( " << pkey << ", " << skey << ", " << O_KEY << ", " << L_KEY <<
", " << DELTA << ", current_timestamp);";
retSize = qs.execute( queryText.str() );
processRetSize( retSize, -1, qs );
}

int main( int argc, char** argv )
{
    cout.setf( std::ios::fixed );
    cout.precision( 2 );
    if ( argc != 9 && argc != 10 && argc != 11 && argc!=12 && argc != 13 && argc != 14 )
    {
        cerr << "usage: " << argv[0] << " <host> <port> <commitaction> <waitaction> <inputfile> <logfile> <user> <password>
[<lineitemname> <ordersname> <historyname>] [<printafter> [<successfile>]]" << endl;
        cerr << endl;
        cerr << "\thost      name of host to connect to" << endl;
        cerr << "\tport      port number on host to connect to" << endl;
        cerr << "\tcommit_type what to do at end of transaction. May be DECIDE (wait for input then) or anything
else (especially COMMIT or ROLLBACK)." << endl;
        cerr << "\tlogfile    the logfile to write output to." << endl;
        cerr << "\tuser      name of user to connect to DWA" << endl;
        cerr << "\tpassword  password used to authenticate user" << endl;
        cerr << "\tlineitem  name of the lineitem table to use (default: LINEITEM)" << endl;
        cerr << "\torders    name of the orders table to use (default: ORDERS)" << endl;
        cerr << "\thistory  name of the history table to use (default: HISTORY)" << endl;
        cerr << "\tprintafter number of transactions (committed or rolled back) after which to print a text (default: -1 =
print no text)" << endl;
        cerr << "\tsuccessfile successfile used in durability test (default: no success file)" << endl;
        return 1;
    }
    Configuration config;
    config.host = argv[1];
    config.port = argv[2];

    config.user      = argv[7];
    config.password = argv[8];

    std::string actionName = argv[3];
    std::string waitName = argv[4];

    string inputfilename = argv[5];
    string logfilename = argv[6];

    string lineitemname = "LINEITEM";
    string ordersname = "ORDERS";
    string historyname = "HISTORY";

    if ( argc == 12 || argc == 13 || argc == 14 )
    {
        lineitemname = argv[9];
        ordersname = argv[10];
        historyname = argv[11];
    }

    int printAfter = -1;
    if ( argc == 10 || argc == 11 )
    {
        printAfter = atoi( argv[9] );
    }
    else if ( argc == 13 || argc == 14 )
    {
        printAfter = atoi( argv[12] );
    }

    bool useSuccessfile = false;
    ofstream success;
    if ( argc == 11 )
    {
        useSuccessfile = true;
        success.open( argv[10] );
        success << "| PKEY | SKEY | O_KEY | L_KEY | DELTA | CURRENT_DATE_TIME |" << endl;
    }
    else if ( argc == 14 )
    {
        useSuccessfile = true;
        success.open( argv[13] );
    }

    void (*commitaction)( QueryStream &qs );

    if ( actionName == "COMMIT" )
        commitaction = &executeCommit;
    else if ( actionName == "ROLLBACK" )
        commitaction = &executeRollback;
    else
    {
        cerr << "action must be one of COMMIT, ROLLBACK" << endl;
        exit( -1 );
    }
}

```

```

}

bool wait = false;
void (*waitaction)();
if ( waitName == "WAIT" )
{
    waitaction = &doWait;
    wait = true;
}
else if ( waitName == "NOWAIT" )
{
    waitaction = &noWait;
}
else
{
    cerr << "wait must be one of WAIT, NOWAIT" << endl;
    exit( -1 );
}

// open streams
ofstream logfile( logfile.c_str() );
logfile << "| RPRICE | QUANTITY | TAX | DISC | EXTPRICE | OTOTAL |" << endl;

//ofstream streamlogfile("acidtransaction.log");
ifstream inputfile( inputfilename.c_str() );

// open querystream
QueryStream qs( "ACID" );//, &streamlogfile );

// connect to server
if ( qs.connectToServer( config ) )
{
    cout << "MAIN: Connected to server" << endl;
}
else
{
    cerr << "MAIN: Error connecting to server. Trying to disconnect." << endl;
    qs.disconnectFromServer();
    return -1;
}

// open TPC-schema
int retSize;

retSize = qs.execute( "open schema tpc;" );
processRetSize( retSize, -1, qs );

retSize = qs.execute( "COMMIT;" );
processRetSize( retSize, -1, qs );

// execute ACID-Transactions as long as the input file contains more values for O_KEY, L_KEY, DELTA
// in case of waiting just the transaction for the first triple is executed.
string O_KEY, L_KEY;
int DELTA;

int committedOrRolledBack = 0;

do
{
    if ( committedOrRolledBack == printAfter )
    {
        cout << "committed or rolled back " << printAfter << " transactions" << endl;
    }
    // read next values from inputfile
    inputfile >> O_KEY;
    inputfile >> L_KEY;
    inputfile >> DELTA;
    if ( inputfile.eof() )
        break;

    // execute the ACID transaction with O_KEY, L_KEY and DELTA
    // but do neither COMMIT nor ROLLBACK
    double rprice, quantity, tax, disc, extprice, ototal;
    int pkey, skey;
    try
    {
        committedOrRolledBack++;
        performAcidTransactionVariant1( qs, O_KEY, L_KEY, DELTA, pkey, skey, rprice, quantity, tax,
disc, extprice, ototal, lineitemname, ordersname, historyname );
    }
    catch( rollbackException )
    {
        continue;
    }
    // wait or no wait (waitaction is a function pointer to doWait() or noWait() )
    waitaction();

    // commit or rollback (commitaction is a function pointer to executeCommit( qs ) or executeRollback( qs )
    commitaction( qs );

    if ( useSuccessfile )
    {
        // write to successfile
        success << "| " << pkey << " | " << skey << " | " << O_KEY << " | " << L_KEY << " | " << DELTA
<< " | ";

        timeval current_time;
        gettimeofday( &current_time, 0 );
        struct tm tm_time;
        localtime_r( &current_time.tv_sec, &tm_time );
        int year = 1900 + tm_time.tm_year;
        int month = tm_time.tm_mon;

```

```

        int day = tm_time.tm_mday;
        int hour = tm_time.tm_hour;
        int minute = tm_time.tm_min;
        int seconds = tm_time.tm_sec;
        success << std::setw(4) << std::setfill('0') << year << "-" << std::setw(2) << std::setfill('0')
<< month << "-" << std::setw(2) << std::setfill('0') << day << " " << std::setw(2) << std::setfill('0') << hour << ":" <<
std::setw(2) << std::setfill('0') << minute << ":" << std::setw(2) << std::setfill('0') << seconds << "." << std::setw(6) <<
std::setfill('0') << current_time.tv_usec << " |" << endl;
    }

    // write to durability success file (if needed)

    // write values to logfile (values returned by driver)
    logfile << "|" << rprice << " |" << quantity << " |" << tax << " |" << disc << " |" << extprice << "
| " << ototal << " |" << endl;
    logfile.flush();
} while ( !inputfile.eof() && !wait );

    // disconnect and exit
if ( qs.disconnectFromServer() )
{
    //cout << "MAIN: successfully disconnected." << endl;
}
else
{
    cout << "MAIN: Error while disconnecting." << endl;
}
return 0;
}
}
-----
# transaction_isolation/querystream.h
-----
#ifndef QUERYSTREAM_H
#define QUERYSTREAM_H

#include <string>
#include <iostream>

#include "dwaCInterface.h"

/** \brief The maximum length of a column name. */
static const size_t maxColumnNameSize = 40;

/** \brief A structure to describe connection information to a database.
 * Used in QueryStream::connectToServer().
 * \sa QueryStream::connectToServer() */
struct Configuration
{
    /** \brief The hostname of the server. */
    char *host;
    /** \brief The port of the server to connect to. */
    char *port;
    /** \brief The user name used for authentication to the database. */
    char *user;
    /** \brief The password of the user. */
    char *password;
};

/** \brief This exception is thrown when a transaction was automatically rolled back.*/
class rollbackException
{
};

/** \brief A structure used to fetch data from a database.
 * Each database column is represented by one FetchStruct.
 * Only one of the arrays of ODBC-C-datatypes is initialized and filled per column.
 * That one is identified by the c_datatype. */
struct FetchStruct
{
    /** \brief Array for columns of SQLCHAR* type
 * (one large block, easy access by sqlCharArrayAccess). */
    SQLCHAR *sqlCharArray;
    /** \brief To quickly access the i-th string in the sqlCharArray array. */
    SQLCHAR* *sqlCharArrayAccess;
    /** \brief Array for columns of SQLSMALLINT type. */
    SQLSMALLINT *sqlSmallInt;
    /** \brief Array for columns of SQLUSMALLINT type. */
    SQLUSMALLINT *sqlUSmallInt;
    /** \brief Array for columns of SQLINTEGER type. */
    SQLINTEGER *sqlInteger;
    /** \brief Array for columns of SQLUINTEGER type. */
    SQLUINTEGER *sqlUInteger;
    /** \brief Array for columns of SQLBIGINT type. */
    SQLBIGINT *sqlBigInt;
    /** \brief Array for columns of SQLUBIGINT type. */
    SQLUBIGINT *sqlUBigInt;
    /** \brief Array for columns of SQLREAL type. */
    SQLREAL *sqlReal;
    /** \brief Array for columns of SQLDOUBLE type. */
    SQLDOUBLE *sqlDouble;
    /** \brief Array for columns of SQLCHAR type. */
    SQLCHAR *sqlChar;
    /** \brief Array for columns of SQLSCHAR type. */
    SQLSCHAR *sqlSChar;
    /** \brief Array for columns of SQL_DATE_STRUCT type. */
    SQL_DATE_STRUCT *sqlDate;
    /** \brief Array for columns of SQL_NUMERIC_STRUCT type. */
    SQL_NUMERIC_STRUCT *sqlNumeric;

    /** \brief The length of a returned data and an indicator whether it is NULL. */

```

```

SQLLEN *resultLen;

/** \brief The name of the column. */
SQLCHAR columnName[maxColumnNameSize];
/** \brief The length of the column name. */
SQLSMALLINT nameLength;
/** \brief The SQL datatype ID of the column. */
SQLSMALLINT datatype;
/** \brief The ODBC C datatype ID of the column.
 * Might be set by QueryStream::find_ODBC_C_Type( SQLSMALLINT sqlType ) */
SQLSMALLINT c_datatype;
/** \brief The size of the column. */
SQLINTEGER columnSize;
/** \brief The number of decimal digits of the column. */
SQLSMALLINT decimalDigits;
/** \brief Whether the column allows NULL values. */
SQLSMALLINT nullable;
};

/** \brief The class represents a query stream to a server.
 * This query stream is slightly different from that one used by the Performance test
 * since it is for implementing the ACID-Transaction for which queries are only
 * executed to get (or change) single lines whose values must be accessible since
 * they will be changed.
 *
 * The execute(...) function was slightly changed to fetch at most two rows and not output or delete
 * the result set. Furthermore a new function freeFetchArray() was introduced to allow the user of
 * this class to free the fetchMe array. Last but not least the fetchMe array of fetched rows is
 * publicly accessible by several getDatatype(column) functions to allow reading of returned data.
 *
 * In addition to that the mapping of C-Datatypes to SQL-Types might have changed slightly to ensure
 * mapping of some columns to numerical C-Types. This affects find_ODBC_C_Type(...). */
class QueryStream
{
public:
    /** \brief Constructor.
     * \param streamId The identifier used to identify this query stream.
     * \param logfile The stream to be used for logging. */
    QueryStream( std::string streamId, std::ostream* logfile = &std::cout );
    /** \brief Tries to establish a connection to a server.
     * \param configuration The connection and login information for the server.
     * \return true if the connection could be established, false otherwise. */
    bool connectToServer( Configuration configuration );
    /** \brief Executes a given statement or query.
     * \param statement The text of the statement or query to execute.
     * \param fetchSize The number of rows to fetch with a single fetch command
     * (bigger fetch size might reduce overhead, but increases memory usage).
     * \return The number of rows in resultset (-2 (error), -1 (no resultset), 0,1 or 2 (means more than one)).
     */
    int execute( std::string statement );
    /** \brief Closes the connection to the server.
     * \return true if the connection was closed successfully, false if errors occurred. */
    bool disconnectFromServer();
    /** \brief frees the memory allocated to store the resultset.
     * Call this exactly if execute did not return -1. */
    void freeFetchArray();

    int getInt( size_t columnNumber );

    double getDouble( size_t columnNumber );
protected:
    /** \brief The maximum size of a string. */
    static const size_t maxStringSize = 200;
    /** \brief The number of rows to fetch in a single fetch command. */
    SQLINTEGER fetchBlockSize;
    /** \brief The number to use for fetchBlockSize if it is not
     * given as argument to execute(). */
    static const size_t standardFetchBlockSize = 100;

    /** \brief Binds a column to an entry in the fetchMe array.
     * This must be done before fetching a resultset.
     * This function arranges memory allocation and binding function.
     * \param colNo The number of the resultset column to bind to.
     * \param localColNo The index in the fetchMe array the resultset data shall be stored.
     * \param datatype The SQL datatype of the column to bind to. */
    void bindColumn( int colNo, int localColNo, SQLSMALLINT datatype );

    /** \brief Allocates memory for the buffers needed in binding resultset columns.
     * The array corresponding to the given datatype of the given index
     * of the fetchMe array is initialized to the size of fetchBlockSize.
     * If the datatype is a string datatype, the string buffer will have the
     * size maxStringSize.
     * \param localColNo The index in the fetchMe array the data shall be allocated for.
     * \param datatype The ODBC-C-datatype memory shall be allocated for. */
    void allocateMemoryForBind( int localColNo, SQLSMALLINT datatype );

    /** \brief Maps an SQL-datatype-ID to a ODBC-C-datatype-ID to which it can be converted.
     * This is needed for binding columns and fetching results from a remote database.
     * \param sqlType The SQL-datatype-ID whose corresponding ODBC-C-datatype-ID is wanted.
     * \return The ODBC-C-datatype-ID for the given SQL-datatype-ID */
    SQLSMALLINT find_ODBC_C_Type( SQLSMALLINT sqlType );

    /** \brief Format resultset header for output. */
    void outputHeader();

    /** \brief Formats fetched rows for output.
     * The data in fetchMe is printed for each column and all fetched rows. */
    void outputFetchedRows();

    /** \brief The stream identifier. */
    std::string myID;
};

```

```

/** \brief The environment handle used. */
SQLHENV henv;
/** \brief The connection handle used. */
SQLHDBC hdbc;
/** \brief The only statement handle used. */
SQLHSTMT hstmt;

/** \brief The structure used to fetch result sets of queries. */
FetchStruct *fetchMe;

/** \brief The number of rows in a result set. */
SQLSMALLINT numRowsCols;

/** \brief The number of rows fetched by last fetch command. */
SQLINTEGER rowsFetched;

/** \brief The stream used for logging results.*/
std::ostream* logfile;

private:
/** \brief Parses the return code of an statement and gives additional information on error. */
SQLRETURN execCritical(SQLRETURN retcode, SQLSMALLINT handle_type = 0, SQLHANDLE handle = 0);
};

#endif // QUERYSTREAM_H
# -----
# transaction_isolation/querystream.cpp
# -----
#include "querystream.h"
#include <iostream>

#ifdef DEBUG
#ifdef DEBUG
#define DEBUGOUT( X ) cout << "DEBUG:" << X
#else
#define DEBUGOUT( X )
#endif
#endif

#define LOG *logfile

using std::cout;
using std::endl;
using std::cerr;

#include <string>
using std::string;

SQLRETURN QueryStream::execCritical(SQLRETURN retcode, SQLSMALLINT handle_type, SQLHANDLE handle)
{
    switch(retcode)
    {
        case SQL_INVALID_HANDLE:
            cout << "invalid handle" << endl;
            throw "ERROR use of invalid handle!\n";
            break;
        case SQL_ERROR:
            {
                SQLCHAR err[1000];
                SQLCHAR state[6];
                SQLINTEGER native;
                SQLSMALLINT len;
                SQLSMALLINT i=1;
                while ( handle !=0 && DWAGetDiagRec(handle_type, handle, i, state, &native,err,1000,&len) == SQL_SUCCESS )
                {
                    cout << state << ":" << err << endl;
                    i++;
                    if ( string( (const char*) err).find( "GlobalTransactionRollback" ) )
                    {
                        throw rollbackException();
                    }
                }
                cout << "SQL error" << endl;
                throw "An error occured while executing command!\n";
                break;
            }
        case SQL_SUCCESS_WITH_INFO:
            cout << "Problem: Has Info, which should not be" << endl;
            throw "Problem: Has Info, which should not be!\n";
            break;
        case SQL_STILL_EXECUTING:
            cout << "Still executing. This should not be should only";
            cout << "return on successful execution!" << endl;
            throw "Unexpectedly still executing!\n";
            break;
        case SQL_NO_DATA:
        case SQL_SUCCESS:
            break;
        default:
            // should not reach here...
            cout << "unexpected return code: " << retcode << endl;
            throw "unexpected return code!\n";
            break;
    }
    return retcode;
}

QueryStream::QueryStream( std::string streamId, std::ostream* logfile_ )
{
    myID = streamId;
    henv = NULL;

```



```

    hdbc = NULL;
    hstmt = NULL;
    logfile = logfile_;
}

bool QueryStream::connectToServer( Configuration cfg )
{
    char localeAttr[]="deu";
    try
    {
        // Allocating environment handle
        execCritical(DWAAllocHandle(SQL_HANDLE_ENV, NULL, &henv));
        // Setting up environment to german localization
        execCritical(DWASetEnvAttr(henv, DWA_ENV_ATTR_SET_LOCALE_ALL, localeAttr, SQL_NTS));
        // Allocating handle for connection
        execCritical(DWAAllocHandle(SQL_HANDLE_DBC, henv, &hdbc));
        // Turning autocommit off... is this necessary?
        execCritical(DWASetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT, (void*)SQL_AUTOCOMMIT_OFF, SQL_NTS));
        // Setting up clientname
        execCritical(DWASetConnectAttr(hdbc, DWA_CLIENT_NAME, (void*)myID.c_str(), SQL_NTS));
        // Connect to server
        execCritical(DWAServerConnect(hdbc, cfg.host, SQL_NTS, cfg.port, SQL_NTS, cfg.user, SQL_NTS, cfg.password, SQL_NTS));
    }
    catch(...)
    {
        // might be nice to clean up handles
        return false;
    }
    return true;
}

int QueryStream::execute( std::string statement )
{
    // the number of rows in the resultSet
    int resultSetSize = -1;

    try
    {
        // put the statement in the logfile
        LOG << statement << endl;
        // Allocating statement handle
        execCritical(DWAAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt), SQL_HANDLE_DBC, hdbc);
        SQLCHAR* statementtext = (SQLCHAR*) statement.c_str();
        // execute the statement / query
        execCritical(DWAExecDirect(hstmt, statementtext, statement.size()), SQL_HANDLE_STMT, hstmt);
        // get the affected rows (only insert, update, delete)
        SQLINTEGER affectedrows;
        execCritical(DWARowCount(hstmt,&affectedrows), SQL_HANDLE_STMT, hstmt);
        // get the number of columns in the resultSet (if any)
        execCritical(DWANumResultCols(hstmt,&numResultCols), SQL_HANDLE_STMT, hstmt);
        // no result columns means no query
        if ( numResultCols <= 0 )
            LOG << "affected rows: " << affectedrows << "\n\n";

        // if a resultSet was constructed, retrieve and print it
        if ( numResultCols > 0 )
            // a resultSet was constructed
            {
                LOG << '\n';
                resultSetSize = 0;
                // setting number of rows to fetch with each fetch
                SQLINTEGER fetchsize = 2; // This is to store the result set and be able to ensure only one line was received
                fetchBlockSize = 2;
                execCritical(DWASetStmtAttr(hstmt, SQL_ATTR_ROW_ARRAY_SIZE,
                    (SQLPOINTER) fetchsize, 0 ), SQL_HANDLE_STMT, hstmt);
                //cout << "Fetch block size (SQL_ATTR_ROW_ARRAY_SIZE) set to " << fetchsize << endl;
                // setting position to store the fetch size for a query
                execCritical(DWASetStmtAttr(hstmt, SQL_ATTR_ROWS_FETCHED_PTR,
                    (SQLPOINTER) &rowsFetched, 0), SQL_HANDLE_STMT, hstmt);
                // allocate the fetch struct to be large enough for the columns
                fetchMe = new FetchStruct[numResultCols];
                // set all pointers to 0 to make it easy to free the correct memory later
                for ( int i=0; i<numResultCols; i++ )
                {
                    fetchMe[i].sqlCharArray = 0;
                    fetchMe[i].sqlCharArrayAccess = 0;
                    fetchMe[i].sqlSmallInt = 0;
                    fetchMe[i].sqlUSmallInt = 0;
                    fetchMe[i].sqlInteger = 0;
                    fetchMe[i].sqlUInteger = 0;
                    fetchMe[i].sqlBigInt = 0;
                    fetchMe[i].sqlUBigInt = 0;
                    fetchMe[i].sqlReal = 0;
                    fetchMe[i].sqlDouble = 0;
                    fetchMe[i].sqlChar = 0;
                    fetchMe[i].sqlSChar = 0;
                    fetchMe[i].sqlDate = 0;
                    fetchMe[i].sqlNumeric = 0;
                    fetchMe[i].resultLen = 0;
                }
                execCritical(DWASetStmtAttr(hstmt, SQL_ATTR_ROWS_FETCHED_PTR,
                    (SQLPOINTER) &rowsFetched, 0), SQL_HANDLE_STMT, hstmt);
                // determine the datatype of the column and additional information.
                for ( SQLSMALLINT colNr=0; colNr<numResultCols; colNr++ )
                {
                    // Get column description for the column with number colNr
                    DWADescribeCol( hstmt, colNr+1, fetchMe[colNr].columnName,
                        maxColumnNameSize, &fetchMe[colNr].nameLength,
                        &fetchMe[colNr].datatype, &fetchMe[colNr].columnSize,
                        &fetchMe[colNr].decimalDigits, &fetchMe[colNr].nullable );
                    fetchMe[colNr].c_datatype = find_ODBC_C_Type( fetchMe[colNr].datatype );
                }
            }
    }
}

```

```

        // bind the column to an entry in the fetchMe array
        bindColumn( colNr+1, colNr, fetchMe[colNr].c_datatype );
    }
    // output the resultset header
    outputHeader();
    // fetch the columns
    do
    {
        // fetch columns
        execCritical(DWAFetch( hstmt ), SQL_HANDLE_STMT, hstmt);
        // output columns
        outputFetchedRows();
        // correct the result set size
        resultSetSize += rowsFetched;
    } while ( rowsFetched == fetchBlockSize );
    // add the total resultset size to the logfile
    LOG << "\nresultset consists of " << resultSetSize << " rows in " << numResultCols << " columns\n\n";
}
}
catch(rollbackException)
{
    // free the handle for the statement again
    cout << "\nROLLBACK EXECUTING STATEMENT:\n" << statement << '\n' << endl;
    execCritical(DWAFreeHandle( SQL_HANDLE_STMT, hstmt ), SQL_HANDLE_STMT, hstmt);
    return -3;
}
catch( char * c )
{
    LOG << "ERROR EXECUTING STATEMENT\n\n";
    cout << "\nERROR EXECUTING STATEMENT:\n" << statement << '\n' << endl;
    cout << c << endl;
    execCritical(DWAFreeHandle( SQL_HANDLE_STMT, hstmt ), SQL_HANDLE_STMT, hstmt);
    return -2;
}
catch(...)
{
    LOG << "ERROR EXECUTING STATEMENT\n\n";
    cout << "\nERROR EXECUTING STATEMENT:\n" << statement << '\n' << endl;
    execCritical(DWAFreeHandle( SQL_HANDLE_STMT, hstmt ), SQL_HANDLE_STMT, hstmt);
    return -2;
}
execCritical(DWAFreeHandle( SQL_HANDLE_STMT, hstmt ), SQL_HANDLE_STMT, hstmt);
return resultSetSize;
}

void QueryStream::freeFetchArray()
{
    for ( int i=0; i<numResultCols; i++ )
    {
        if ( fetchMe[i].sqlCharArray != 0 )
        {
            delete[] fetchMe[i].sqlCharArray;
            delete[] fetchMe[i].sqlCharArrayAccess;
        }
        if ( fetchMe[i].sqlSmallInt != 0 ) delete[] fetchMe[i].sqlSmallInt;
        if ( fetchMe[i].sqlUSmallInt != 0 ) delete[] fetchMe[i].sqlUSmallInt;
        if ( fetchMe[i].sqlInteger != 0 ) delete[] fetchMe[i].sqlInteger;
        if ( fetchMe[i].sqlUIInteger != 0 ) delete[] fetchMe[i].sqlUIInteger;
        if ( fetchMe[i].sqlBigInt != 0 ) delete[] fetchMe[i].sqlBigInt;
        if ( fetchMe[i].sqlUBigInt != 0 ) delete[] fetchMe[i].sqlUBigInt;
        if ( fetchMe[i].sqlReal != 0 ) delete[] fetchMe[i].sqlReal;
        if ( fetchMe[i].sqlDouble != 0 ) delete[] fetchMe[i].sqlDouble;
        if ( fetchMe[i].sqlChar != 0 ) delete[] fetchMe[i].sqlChar;
        if ( fetchMe[i].sqlSChar != 0 ) delete[] fetchMe[i].sqlSChar;
        if ( fetchMe[i].sqlDate != 0 ) delete[] fetchMe[i].sqlDate;
        if ( fetchMe[i].sqlNumeric != 0 ) delete[] fetchMe[i].sqlNumeric;
        if ( fetchMe[i].resultLen != 0 ) delete[] fetchMe[i].resultLen;
    }
    delete[] fetchMe;
}

int QueryStream::getInt( size_t columnNumber )
{
    size_t i = columnNumber;
    if ( i >= numResultCols )
    {
        cout << "column out of range" << endl;
        throw "column out of range";
    }

    if ( fetchMe[i].sqlCharArray != 0 ) return atoi( (char*) fetchMe[i].sqlCharArrayAccess[0] );

    if ( fetchMe[i].sqlSmallInt != 0 ) return fetchMe[i].sqlSmallInt[0];
    if ( fetchMe[i].sqlUSmallInt != 0 ) return fetchMe[i].sqlUSmallInt[0];
    if ( fetchMe[i].sqlInteger != 0 ) return fetchMe[i].sqlInteger[0];
    if ( fetchMe[i].sqlUIInteger != 0 ) return fetchMe[i].sqlUIInteger[0];
    if ( fetchMe[i].sqlBigInt != 0 ) return fetchMe[i].sqlBigInt[0];
    if ( fetchMe[i].sqlUBigInt != 0 ) return fetchMe[i].sqlUBigInt[0];
    if ( fetchMe[i].sqlReal != 0 ) return (int) fetchMe[i].sqlReal[0];
    if ( fetchMe[i].sqlDouble != 0 ) return (int) fetchMe[i].sqlDouble[0];
    if ( fetchMe[i].sqlChar != 0 ) return fetchMe[i].sqlChar[0];
    if ( fetchMe[i].sqlSChar != 0 ) return fetchMe[i].sqlSChar[0];

    cout << "cannot cast datatype to Int" << endl;
    throw "cannot cast datatype to Int";
}

double QueryStream::getDouble( size_t columnNumber )
{

```

```

        size_t i = columnNumber;
        if ( i >= numResultCols )
        {
            cout << "column out of range" << endl;
            throw "column out of range";
        }

        if ( fetchMe[i].sqlCharArray != 0 ) return atof( (char*) fetchMe[i].sqlCharArrayAccess[0] );

        if ( fetchMe[i].sqlSmallInt != 0 ) return fetchMe[i].sqlSmallInt[0];
        if ( fetchMe[i].sqlUSmallInt != 0 ) return fetchMe[i].sqlUSmallInt[0];
        if ( fetchMe[i].sqlInteger != 0 ) return fetchMe[i].sqlInteger[0];
        if ( fetchMe[i].sqlUInteger != 0 ) return fetchMe[i].sqlUInteger[0];
        if ( fetchMe[i].sqlBigint != 0 ) return fetchMe[i].sqlBigint[0];
        if ( fetchMe[i].sqlUBigint != 0 ) return fetchMe[i].sqlUBigint[0];
        if ( fetchMe[i].sqlReal != 0 ) return fetchMe[i].sqlReal[0];
        if ( fetchMe[i].sqlDouble != 0 ) return fetchMe[i].sqlDouble[0];
        if ( fetchMe[i].sqlChar != 0 ) return fetchMe[i].sqlChar[0];
        if ( fetchMe[i].sqlSchar != 0 ) return fetchMe[i].sqlSchar[0];

        cout << "cannot cast datatype to Int" << endl;
        throw "cannot cast datatype to Int";
    }
}

bool QueryStream::disconnectFromServer()
{
    try
    {
        // disconnecting from server
        execCritical(DWADisconnect(hdbc));
        // freeing connection handle
        execCritical(DWAFreeHandle(SQL_HANDLE_DBC, hdbc));
        // freeing environment handle
        execCritical(DWAFreeHandle(SQL_HANDLE_ENV, henv));
    }
    catch(...)
    {
        cout << "Error while disconnecting from server" << endl;
        return false;
    }
    return true;
}

void QueryStream::outputHeader()
{
    int headerLength = 1;
    LOG << "|";
    for ( size_t col=0; col<numResultCols; col++ )
    {
        LOG << fetchMe[col].columnName << "|";
        headerLength += fetchMe[col].nameLength + 1;
    }
    LOG << '\n';
    for ( int i=0; i<headerLength; i++ )
        LOG << '=';
    LOG << '\n';
    (LOG).flush();
}

void QueryStream::outputFetchedRows()
{
    for ( size_t rows=0; rows<rowsFetched; rows++ )
    {
        LOG << "|";
        for ( size_t cols=0; cols<numResultCols; cols++ )
        {
            if ( fetchMe[cols].resultLen[rows] == SQL_NULL_DATA )
            {
                LOG << "|";
                continue;
            }
            switch( fetchMe[cols].c_datatype )
            {
                // string data
                case SQL_C_CHAR:
                case SQL_C_BINARY:
                //case SQL_C_XML:
                //case SQL_C_VARBOOKMARK:
                    LOG << fetchMe[cols].sqlCharArrayAccess[rows] << "|";
                    break;

                // small int
                case SQL_C_SSHORT:
                    LOG << fetchMe[cols].sqlSmallInt[rows] << "|";
                    break;

                // unsigned small int
                case SQL_C_USHORT:
                    LOG << fetchMe[cols].sqlUSmallInt[rows] << "|";
                    break;

                // long int
                case SQL_C_SLONG:
                    LOG << fetchMe[cols].sqlInteger[rows] << "|";
                    break;

                // unsigned long int
                case SQL_C_ULONG:
                    LOG << fetchMe[cols].sqlUInteger[rows] << "|";
                    break;
            }
        }
    }
}

```

```

// float
case SQL_C_FLOAT:
    LOG << fetchMe[cols].sqlReal[rows] << "|";
break;

// double
case SQL_C_DOUBLE:
    LOG << fetchMe[cols].sqlDouble[rows] << "|";
break;

// unsigned char
case SQL_C_BIT:
case SQL_C_UTINYINT:
    LOG << fetchMe[cols].sqlChar[rows] << "|";
break;

// signed char
case SQL_C_STINYINT:
    LOG << fetchMe[cols].sqlSChar[rows] << "|";
break;

// big int
case SQL_C_SBIGINT:
    LOG << fetchMe[cols].sqlBigInt[rows] << "|";
break;

// unsigned big int
case SQL_C_UBIGINT:
    LOG << fetchMe[cols].sqlUBigInt[rows] << "|";
break;

// date
case SQL_C_TYPE_DATE:
{
    // format date to YYYY-MM-DD for output
    if ( fetchMe[cols].sqlDate[rows].year < 1000 )
        LOG << "0";
    if ( fetchMe[cols].sqlDate[rows].year < 100 )
        LOG << "0";
    if ( fetchMe[cols].sqlDate[rows].year < 10 )
        LOG << "0";
    LOG << fetchMe[cols].sqlDate[rows].year << "-";

    if ( fetchMe[cols].sqlDate[rows].month < 10 )
        LOG << "0";
    LOG << fetchMe[cols].sqlDate[rows].month << "-";

    if ( fetchMe[cols].sqlDate[rows].day < 10 )
        LOG << "0";
    LOG << fetchMe[cols].sqlDate[rows].day << "|";
}
break;

// numeric, i.e. decimal
// not used since it is eaier by far to map decimals to doubles
case SQL_C_NUMERIC:
{
    // format decimal for output
    int scale = fetchMe[cols].sqlNumeric[rows].scale;
    char text[3*SQL_MAX_NUMERIC_LEN+2+scale];
    int pos_T = 3*SQL_MAX_NUMERIC_LEN+scale;
    int pos_N = SQL_MAX_NUMERIC_LEN-1;
    text[3*SQL_MAX_NUMERIC_LEN+scale] = '\0';
    //int add = 0;
    //if ( fetchMe[cols].sqlNumeric[rows].sign == 0 )
    //    add=1;
    while ( fetchMe[cols].sqlNumeric[rows].val[pos_N] == 0 && pos_N > 0 )
        pos_N--;

    while ( pos_N >= 0 )
    {
        int current = fetchMe[cols].sqlNumeric[rows].val[pos_N];
        int newVal = 0;
        int currentPos = pos_N;
        while ( currentPos >= 0 )
        {
            //if ( currentPos == 0 )
            //{
            //    current += add;
            //    add = 0;
            //}
            if ( current < 10 )
            {
                fetchMe[cols].sqlNumeric[rows].val[currentPos] = newVal;
                currentPos--;

                if ( currentPos >= 0 )
                {
                    current = current * 256 + fetchMe[cols].sqlNumeric[rows].val[currentPos];
                    newVal = 0;
                }
            }
            else
            {
                newVal = current / 10;
                current = current % 10;
            }
        }
        if ( fetchMe[cols].sqlNumeric[rows].val[pos_N] == 0 )

```

```

        {
            pos_N--;
        }
        text[pos_T--] = '0' + current;
        if ( scale == 1 )
        {
            text[pos_T--] = '.';
        }
        scale--;
    }
    if ( scale==0)
        text[pos_T--] = '0';
    while ( scale > 0 )
    {
        text[pos_T--] = '0';
        if ( scale == 1 )
        {
            text[pos_T--] = '.';
            text[pos_T--] = '0';
        }
        scale--;
    }

    if ( fetchMe[cols].sqlNumeric[rows].sign == 0 )
        text[pos_T--] = '-';
    LOG << text+pos_T+1 << " |";
}
break;

// other datatype
default:
    cout << "output: datatype " << fetchMe[cols].c_datatype << " not supported yet" << endl;
    throw fetchMe[cols].c_datatype;
break;
}
}
LOG << '\n';
(LOG).flush();
}
}

void QueryStream::bindColumn( int colNo, int localColNo, SQLSMALLINT c_datatype )
{
    // allocate memory for the column to bind
    allocateMemoryForBind( localColNo, c_datatype );
    // bind the column
    switch ( c_datatype )
    {
        // string data
        case SQL_C_CHAR:
        case SQL_C_BINARY:
        //case SQL_C_XML:
        //case SQL_C_VARBOOKMARK:
            execCritical(DWABindCol(
                hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlCharArray,
                maxStringSize*sizeof(SQLCHAR), fetchMe[localColNo].resultLen));
            break;

        // small int
        case SQL_C_SSHORT:
            execCritical(DWABindCol(
                hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlSmallInt,
                sizeof(SQLSMALLINT), fetchMe[localColNo].resultLen));
            break;

        // unsigned small int
        case SQL_C_USHORT:
            execCritical(DWABindCol(
                hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlUSmallInt,
                sizeof(SQLUSMALLINT), fetchMe[localColNo].resultLen));
            break;

        // long int
        case SQL_C_SLONG:
            execCritical(DWABindCol(
                hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlInteger,
                sizeof(SQLINTEGER), fetchMe[localColNo].resultLen));
            break;

        // unsigned long int
        case SQL_C_ULONG:
            execCritical(DWABindCol(
                hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlUIInteger,
                sizeof(SQLUIINTEGER), fetchMe[localColNo].resultLen));
            break;

        // float
        case SQL_C_FLOAT:
            execCritical(DWABindCol(
                hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlReal,
                sizeof(SQLREAL), fetchMe[localColNo].resultLen));
            break;

        // double
        case SQL_C_DOUBLE:
            execCritical(DWABindCol(
                hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlDouble,
                sizeof(SQLDOUBLE), fetchMe[localColNo].resultLen));
            break;

        // unsigned char
    }
}

```

```

case SQL_C_BIT:
case SQL_C_UTINYINT:
    execCritical(DWABindCol(
        hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlChar,
        sizeof(SQLCHAR), fetchMe[localColNo].resultLen));
break;

// signed char
case SQL_C_STINYINT:
    execCritical(DWABindCol(
        hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlSmallInt,
        sizeof(SQLCHAR), fetchMe[localColNo].resultLen));
break;

// big int
case SQL_C_SBIGINT:
    execCritical(DWABindCol(
        hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlBigInt,
        sizeof(SQLBIGINT), fetchMe[localColNo].resultLen));
break;

// unsigned big int
case SQL_C_UBIGINT:
    execCritical(DWABindCol(
        hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlUBigInt,
        sizeof(SQLUBIGINT), fetchMe[localColNo].resultLen));
break;

// date
case SQL_C_TYPE_DATE:
    execCritical(DWABindCol(
        hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlDate,
        sizeof(SQL_DATE_STRUCT), fetchMe[localColNo].resultLen));
break;

// numeric, i.e. decimal
case SQL_C_NUMERIC:
    execCritical(DWABindCol(
        hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlNumeric,
        sizeof(SQL_NUMERIC_STRUCT), fetchMe[localColNo].resultLen));
break;

// other datatype
default:
    throw c_datatype;
break;
}
}

void QueryStream::allocateMemoryForBind( int localColNo, SQLSMALLINT datatype )
{
    // allocate memory for resultLen field used to indicate NULL values (among others)
    fetchMe[localColNo].resultLen = new SQLLEN[fetchBlockSize];

    // allocate memory depending on the column datatype
    switch ( datatype )
    {
        // string data
        case SQL_C_CHAR:
        case SQL_C_BINARY:
        //case SQL_C_XML:
        //case SQL_C_VARBOOKMARK:
            fetchMe[localColNo].sqlCharArray = new SQLCHAR[fetchBlockSize*maxStringSize];
            fetchMe[localColNo].sqlCharArrayAccess = new SQLCHAR*[fetchBlockSize];
            for ( size_t i=0; i<fetchBlockSize; i++ )
            {
                fetchMe[localColNo].sqlCharArrayAccess[i] = fetchMe[localColNo].sqlCharArray+i*maxStringSize;
            }
            break;

        // small int
        case SQL_C_SSHORT:
            fetchMe[localColNo].sqlSmallInt = new SQLSMALLINT[fetchBlockSize];
            break;

        // unsigned small int
        case SQL_C_USHORT:
            fetchMe[localColNo].sqlUSmallInt = new SQLUSMALLINT[fetchBlockSize];
            break;

        // long int
        case SQL_C_SLONG:
            fetchMe[localColNo].sqlInteger = new SQLINTEGER[fetchBlockSize];
            break;

        // unsigned long int
        case SQL_C_ULONG:
            fetchMe[localColNo].sqlUInteger = new SQLUINTEGER[fetchBlockSize];
            break;

        // float
        case SQL_C_FLOAT:
            fetchMe[localColNo].sqlReal = new SQLREAL[fetchBlockSize];
            break;

        // double
        case SQL_C_DOUBLE:
            fetchMe[localColNo].sqlDouble = new SQLDOUBLE[fetchBlockSize];
            break;

        // unsigned char
    }
}

```

```

    case SQL_C_BIT:
    case SQL_C_UTINYINT:
        fetchMe[localColNo].sqlChar = new SQLCHAR[fetchBlockSize];
        break;

    // signed char
    case SQL_C_STINYINT:
        fetchMe[localColNo].sqlSChar = new SQLSCHAR[fetchBlockSize];
        break;

    // big int
    case SQL_C_SBIGINT:
        fetchMe[localColNo].sqlBigInt = new SQLBIGINT[fetchBlockSize];
        break;

    // unsigned big int
    case SQL_C_UBIGINT:
        fetchMe[localColNo].sqlUBigInt = new SQLUBIGINT[fetchBlockSize];
        break;

    // date
    case SQL_C_TYPE_DATE:
        fetchMe[localColNo].sqlDate = new SQL_DATE_STRUCT[fetchBlockSize];
        break;

    // numeric, i.e. decimal
    case SQL_C_NUMERIC:
        fetchMe[localColNo].sqlNumeric = new SQL_NUMERIC_STRUCT[fetchBlockSize];
        break;

    // other datatype
    default:
        cout << "memory allocation: datatype " << datatype << " not supported yet" << endl;
        throw datatype;
        break;
    }
}

SQLSMALLINT QueryStream::find_ODBC_C_Type( SQLSMALLINT sqlType )
{
    switch( sqlType )
    {
        case SQL_CHAR:
        case SQL_VARCHAR:
        case SQL_LONGVARCHAR:
        case SQL_WCHAR:
        case SQL_WVARCHAR:
        case SQL_WLONGVARCHAR:
            return SQL_C_CHAR;
        break;

        case SQL_DECIMAL:
        case SQL_NUMERIC:
            //return SQL_C_NUMERIC;
            return SQL_C_DOUBLE;
            //return SQL_C_CHAR;
        break;

        case SQL_SMALLINT:
            return SQL_C_SSHORT;
        break;

        case SQL_INTEGER:
            return SQL_C_SLONG;
        break;

        case SQL_REAL:
        case SQL_FLOAT:
        case SQL_DOUBLE:
            return SQL_C_DOUBLE;
        break;

        case SQL_BIT:
            return SQL_C_BIT;
        break;

        case SQL_TINYINT:
            return SQL_C_STINYINT;
        break;

        case SQL_BIGINT:
            return SQL_C_SBIGINT;
        break;

        case SQL_BINARY:
        case SQL_VARBINARY:
        case SQL_LONGVARBINARY:
            return SQL_C_BINARY;
        break;

        case SQL_TYPE_DATE:
            return SQL_C_TYPE_DATE;
        break;

        default:
            cout << "unsupported datatype, don't know corresponding C type: " << sqlType << endl;
            throw sqlType;
        break;
    }
}

```

```

# -----
# run_durability_test.sh (consistency check included)
# -----
#!/bin/bash
# params: <host> <scalefactor> <streams> <logdir> <number_of_durability_tests>

HOST="$1"
SF="$2"
STREAMS="$3"
LOGDIR="$4/durability"
NUM_TESTS="$5"
mkdir -p $LOGDIR
mkdir -p $LOGDIR/tmp
LOGFILE=$LOGDIR/durability.log
ACIDDIR="/shares/dwstaging/benchmark/ACID"
TRANSACTIONS_FOR_DURABILITY=400

ROOTPORT=26822

echo "*****" | tee $LOGFILE
echo "DURABILITY TEST" | tee -a $LOGFILE
echo "*****" | tee -a $LOGFILE

## create table history2 (needed for easier comparison of history table and success files)
echo "creating table history2 (needed for easier comparison of history table and success files)" | tee -a $LOGFILE
dwaplus -h $HOST -p $ROOTPORT -u tpcuser -P tpcuser -f $ACIDDIR/sql/create_history_table2.sql >> /dev/null

## start one durability test after another
for (( ID=1;ID<=$NUM_TESTS; ID++ ))
do
    $ACIDDIR/scripts/durability_test_main.sh $HOST $SF $STREAMS $LOGDIR $TRANSACTIONS_FOR_DURABILITY $ID | tee -a
$LOGFILE
done

## check whether whole durability tests has passed or failed
if [ -z "`grep 'failed' $LOGFILE`" ]
then
    echo "Durability test passed." | tee -a $LOGFILE
else
    echo "Durability test failed. See $LOGDIR/durability.log for details." | tee -a $LOGFILE
fi

# -----
# durability_test_main.sh
# -----
#!/bin/bash
# executes a durability test
# param: <roothost> <scalefactor> <streams> <logdir> <numberoftransactionsperstream> <testid>

ROOTHOST="$1"
SCALEFACTOR="$2"
STREAMS="$3"
LOGDIR="$4"
NUMTRANSACTIONS="$5"
ID="$6"
LOGFILE="$LOGDIR/durability${ID}.log"

ROOTPORT=26822

ACIDDIR="/shares/dwstaging/benchmark/ACID"
INFORM_AFTER_NUM_TRANSACTIONS=100

export ADDITIONALWAITTIME=5
# unset ADDITIONALWAITTIME

export WAITAFTERROLLBACK=8
# unset WAITAFTERROLLBACK

export WAITAFTERCOMMIT=15
# unset WAITAFTERCOMMIT

# export WAITBEFORECOMMIT=1
unset WAITBEFORECOMMIT

echo "-----" | tee $LOGFILE
echo "Durability test $ID" | tee -a $LOGFILE
echo "-----" | tee -a $LOGFILE

echo "deleting old entries in history table" | tee -a $LOGFILE
## delete all entries in history table
dwaplus -h $ROOTHOST -p $ROOTPORT -u tpcuser -P tpcuser -f $ACIDDIR/sql/clear_history.sql >> $LOGFILE

## generate random orderkeys for use in consistency check
for (( i=0; i<=$STREAMS; i++ ))
do
    echo "creating $NUMTRANSACTIONS random transactions for stream $i" | tee -a $LOGFILE
    $ACIDDIR/scripts/create_random_transactions.sh \
        $i $NUMTRANSACTIONS $SCALEFACTOR $ROOTHOST \
        $LOGDIR/tmp $LOGDIR/durability${ID}_keys$i.log >> $LOGFILE
done

## check initial consistency
INCONSISTENT=(`$ACIDDIR/scripts/check_consistency.sh $ROOTHOST $LOGDIR/durability${ID}_initial_consistency.log`)

if [ $INCONSISTENT -gt 0 ]
then
    echo "Durability Test $ID failed ( Initial consistency check failed, $INCONSISTENT Orderkeys inconsistent )." | tee
-a $LOGFILE
    exit

```



```

fi

## start the transactions on all streams
echo "starting stream 0" | tee -a $LOGFILE
$ACIDDIR/scripts/transaction/acidtransaction $ROOTHOST $ROOTPORT COMMIT NOWAIT $LOGDIR/durability${ID}_keys0.log
$LOGDIR/durability${ID}_return0.log tpcuser tpcuser $LOGDIR/durability${ID}_values0.log $INFORM_AFTER_NUM_TRANSACTIONS
$LOGDIR/durability${ID}_success0.log \
2>/dev/null | tee $LOGDIR/durability${ID}_transaction_0.log &
for (( i=1; i<=$STREAMS; i++ ))
do
    echo "starting stream $i" | tee -a $LOGFILE
    $ACIDDIR/scripts/transaction/acidtransaction $ROOTHOST $ROOTPORT COMMIT NOWAIT $LOGDIR/durability${ID}_keys$i.log
    $LOGDIR/durability${ID}_return$i.log tpcuser tpcuser $LOGDIR/durability${ID}_values$i.log $INFORM_AFTER_NUM_TRANSACTIONS
    $LOGDIR/durability${ID}_success$i.log \
    2>/dev/null | tee $LOGDIR/durability${ID}_transaction_$i.log | grep "\ (ERROR\|ROLLBACK\|committed or
    rolled back)" | tee $LOGDIR/tmp/dur${ID}_t$i.err | grep "\ (ERROR\|committed or rolled back)" \
    | sed -u "s/^/STREAM $i:/g" | tee -a $LOGFILE &
done

echo "waiting until $INFORM_AFTER_NUM_TRANSACTIONS transactions have been committed or rolled back by each stream." | tee -a
$LOGFILE

## wait until failure may be caused
# some loop
DONE="NOT DONE"
while [ ! -z "$DONE" ]
do
    DONE=""
    for (( i=0; i<=$STREAMS; i++ ))
    do
        CURRENT=`grep 'committed or rolled back' $LOGDIR/durability${ID}_transaction_$i.log`
        if [ -z "$CURRENT" ]
        then
            DONE="NOT DONE"
        fi
    done
    sleep 10
done

## ask for keypress before failure (to check for errors in streamlogs)
echo "Press enter key, please, before causing an error to allow checking for errors (which cause the test to fail)" | tee -a
$LOGFILE
read $X
# check for errors in all stream logs
for (( i=0; i<=$STREAMS; i++ ))
do
    CURRENT=`grep 'ERROR' $LOGDIR/durability${ID}_transaction_$i.log`
    if [ ! -z "$CURRENT" ]
    then
        ERROR="$ERROR:Error in stream $i"
    fi
done
echo "date +%m/%d/%Y %k:%M:%S.%N" | sed 's/\(\\.\.\.\)\.\.\.\./\1/g`: Error checking completed" | tee -a $LOGFILE
## kill the streams (if still running)
echo "Press enter key to kill running transactions" | tee -a $LOGFILE
read $X
echo "date +%m/%d/%Y %k:%M:%S.%N" | sed 's/\(\\.\.\.\)\.\.\.\./\1/g`: killing all acidtransactions which might still be
running" | tee -a $LOGFILE
killall acidtransaction | tee -a $LOGFILE
echo "date +%m/%d/%Y %k:%M:%S.%N" | sed 's/\(\\.\.\.\)\.\.\.\./\1/g`: killed all acidtransactions" | tee -a $LOGFILE

## wait for keypress after recovery
echo "Press enter key, please, when recovery is completed." | tee -a $LOGFILE
read $X
echo "date +%m/%d/%Y %k:%M:%S.%N" | sed 's/\(\\.\.\.\)\.\.\.\./\1/g`: Checking consistency" | tee -a $LOGFILE

## check consistency again
INCONSISTENT=(`$ACIDDIR/scripts/check_consistency.sh $ROOTHOST $LOGDIR/durability${ID}_final_consistency.log`)
if [ $INCONSISTENT -gt 0 ]
then
    echo "Durability Test $ID failed ( Final consistency check failed, $INCONSISTENT Orderkeys inconsistent )."
    exit
fi

## clear second history table (easier comparison)
echo "clearing table history2" | tee -a $LOGFILE
dwaplus -h $ROOTHOST -p $ROOTPORT -u tpcuser -P tpcuser -f $ACIDDIR/sql/clear_history2.sql >> $LOGFILE

## insert all entries from all success files in second history table
echo "inserting entries from all success files in table history2" | tee -a $LOGFILE
SQLFILE=$LOGDIR/durability${ID}_history.sql
echo "set autocommit off" > $SQLFILE
echo "open schema tpc;" >> $SQLFILE
for (( i=0; i<=$STREAMS; i++ ))
do
    cat $LOGDIR/durability${ID}_success$i.log | sed "s//,/g" | grep -v CURRENT_DATE_TIME | sed "s/, \(\.\.\.\)\.\.\.\./\1/g"
| sed "s/^/,/insert into history2 values (/g" | sed "s/,/\,/'/;g" >> $SQLFILE
done
echo "commit;" >> $SQLFILE

# load sql into history2
cat $SQLFILE | dwaplus -h $ROOTHOST -p $ROOTPORT -u tpcuser -P tpcuser >> $LOGDIR/durability${ID}_load_into_history2.log

## compare history tables
echo "comparing history tables" | tee -a $LOGFILE
cat $ACIDDIR/sql/compare_history1.sql | dwaplus -h $ROOTHOST -p $ROOTPORT -u tpcuser -P tpcuser >
$LOGDIR/durability${ID}_compare_history1.log
cat $ACIDDIR/sql/compare_history2.sql | dwaplus -h $ROOTHOST -p $ROOTPORT -u tpcuser -P tpcuser >
$LOGDIR/durability${ID}_compare_history2.log
DIFFERENCES1=`grep 'rows in resultset\.' $LOGDIR/durability${ID}_compare_history1.log | sed 's/rows in resultset\./g`))

```

```

DIFFERENCES2=`grep 'rows in resultset\.' $LOGDIR/durability${ID}_compare_history2.log | sed 's/rows in resultset\.//g'`

echo "storing history table in $LOGDIR/durability${ID}_store_history.log" | tee -a $LOGFILE
cat $ACIDDIR/sql/store_history.sql | dwapplus -h $ROOTHOST -p $ROOTPORT -u tpcuser -P tpcuser >
$LOGDIR/durability${ID}_store_history.log
cat $ACIDDIR/sql/store_history2.sql | dwapplus -h $ROOTHOST -p $ROOTPORT -u tpcuser -P tpcuser >
$LOGDIR/durability${ID}_store_history2.log

## very probably history tables are identical... but there might be one or two additional entries
## if there are differences in history tables, take a look at them yourself to decide whether test was ok
if [ $DIFFERENCES1 -gt 0 ]
then
    echo "Durability test $ID might have failed. See $LOGDIR/durability${ID}_compare_history1.log for details." | tee -a
$LOGFILE
    echo "You might also want to take a look at $LOGDIR/durability${ID}_store_history.log and
$LOGDIR/durability${ID}_store_history2.log." | tee -a $LOGFILE
    exit
fi

if [ $DIFFERENCES2 -gt 0 ]
then
    echo "Durability test $ID might have failed. See $LOGDIR/durability${ID}_compare_history2.log for details." | tee -a
$LOGFILE
    echo "You might also want to take a look at $LOGDIR/durability${ID}_store_history.log and
$LOGDIR/durability${ID}_store_history2.log." | tee -a $LOGFILE
    exit
fi

## passed the durability test if there were no errors until now
if [ -z "$ERROR" ]
then
    echo "Durability Test $ID passed." | tee -a $LOGFILE
else
    echo "Durability Test $ID failed due to errors." | tee -a $LOGFILE
    echo "$ERROR" | tee -a $LOGFILE
fi

# -----
# check_consistency.sh
# -----
#!/bin/bash
# param <roothost> <logfile>

ROOTHOST="$1"
LOGFILE="$2"
ROOTPORT=26822

SQLDIR="/shares/dwstaging/benchmark/ACID/sql"

## execute both created queries
cat $SQLDIR/check_consistency.sql | dwapplus -h $ROOTHOST -p $ROOTPORT -u sys -P exasol > $LOGFILE
SELECTED=`cat $LOGFILE | grep "rows in resultset" | sed "s/rows in resultset//g"`

echo "$SELECTED"

# -----
# check_consistency.sql
# -----
set autocommit off;

open schema tpc;

select *
from (
    select o_orderkey, o_totalprice - sum( trunc( trunc ( l_extendedprice * (1-l_discount),2)*(1+l_tax),2)) part_res
    from orders, lineitem
    where o_orderkey=l_orderkey
    group by o_orderkey, o_totalprice
) where not part_res=0;

exit;

# -----
# compare_history1.sql
# -----
set autocommit off;
open schema tpc;
select * from history h1 where not exists
(
    select *
    from history2 h2
    where h1.h_p_key=h2.h_p_key
    and h1.h_s_key = h2.h_s_key
    and h1.h_o_key=h2.h_o_key
    and h1.h_l_key=h2.h_l_key
    and h1.h_delta=h2.h_delta
);
commit;

# -----
# compare_history2.sql
# -----
set autocommit off;
open schema tpc;
select * from history2 h2 where not exists
(
    select *

```

```

        from history h1
        where h1.h_p_key=h2.h_p_key
            and h1.h_s_key = h2.h_s_key
            and h1.h_o_key=h2.h_o_key
            and h1.h_l_key=h2.h_l_key
            and h1.h_delta=h2.h_delta
    );
commit;
# -----
# create_random_transactions.sh
# -----
#!/bin/bash
# params: <stream> <count> <scalefactor> <host> <tmpdir> <outfile>
STREAM=$1
COUNT=$2
SF=$3
HOST=$4
TMPFILE=$5/transactions_${STREAM}.sql
OUTFILE=$6
ROOTPORT=26822

echo "set autocommit off;" > $TMPFILE
echo >> $TMPFILE
echo "open schema tpc;" >> $TMPFILE
echo >> $TMPFILE
echo "create table indices${STREAM} ( i dec(10) );" >> $TMPFILE
echo -n "insert into indices${STREAM} values " >> $TMPFILE
for (( i=1; i<=$COUNT; i++ ))
do
    echo -n "$i"
    if [ $i != $COUNT ]
    then
        echo ","
    fi
done >> $TMPFILE

echo ";" >> $TMPFILE
echo >> $TMPFILE

SEED=$RANDOM

echo "create table rnd${STREAM} as select random(DEC(12), $SEED, 0, 1, ${SF}*150000) rnd, i from indices${STREAM};" >> $TMPFILE
echo "COMMIT;" >> $TMPFILE
echo >> $TMPFILE

echo "update rnd${STREAM} set rnd=trunc(rnd/8)*32+mod(rnd,8);" >> $TMPFILE
echo "COMMIT;" >> $TMPFILE
echo >> $TMPFILE

echo "create table rndwithln${STREAM} as select rnd, i, max(l_linenumber) ln from lineitem, rnd${STREAM} where l_orderkey=rnd
group by i, rnd order by i;" >> $TMPFILE
echo "COMMIT;" >> $TMPFILE
echo >> $TMPFILE

SEED=$RANDOM
echo "update rndwithln${STREAM} set ln=random( dec(10), $SEED, 0, 1, ln );" >> $TMPFILE
echo "COMMIT;" >> $TMPFILE
echo >> $TMPFILE

SEED=$RANDOM
echo "create table values${STREAM} as select rnd O_KEY, ln L_KEY, random( dec(10), $SEED, 0, 1, 100 ) DELTA, i from
rndwithln${STREAM};" >> $TMPFILE
echo "COMMIT;" >> $TMPFILE

echo "drop table indices${STREAM};" >> $TMPFILE
echo "drop table rnd${STREAM};" >> $TMPFILE
echo "drop table rndwithln${STREAM};" >> $TMPFILE
echo "COMMIT;" >> $TMPFILE

echo >> $TMPFILE
echo "select O_KEY,L_KEY,DELTA from values${STREAM} order by i;" >> $TMPFILE
echo >> $TMPFILE
echo "drop table values${STREAM};" >> $TMPFILE
echo >> $TMPFILE
echo "COMMIT;" >> $TMPFILE
echo >> $TMPFILE
echo "exit;" >> $TMPFILE
echo >> $TMPFILE

# execute SQLFILE with dwapplus. Filter output so only the values for O_KEY, L_KEY and DELTA remain
dwapplus -h $HOST -p $ROOTPORT -u tpcuser -P tpcuser \
-f $TMPFILE | grep " " | grep -v "O_KEY" \
> $OUTFILE

# -----
# store_values_for_orderkeys.sh
# -----
#!/bin/bash
# param <roothost> <orderkeysfile> <outfile> <logdir>
# precondition:
# - DWA must be running and accepting connections on <roothost>:$ROOTPORT
# - the path to <outfile> should exist

ROOTHOST="$1"
ORDERKEYS="$2"
OUTFILE="$3"
LOGDIR="$4"

ROOTPORT=26822

ACIDDIR="/shares/dwstaging/benchmark/ACID"

```

```

TMPDIR=$LOGDIR/tmp

mkdir -p $TMPDIR

echo "OPEN SCHEMA tpc;" > $TMPDIR/tmp_queryfile.sql
echo >> $TMPDIR/tmp_queryfile.sql
echo "SELECT * FROM history ORDER BY h_date_t;" >> $TMPDIR/tmp_queryfile.sql
# echo $TMPDIR/tmp_queryfile.sql
echo "SELECT * FROM lineitem WHERE l_orderkey in (" >> $TMPDIR/tmp_queryfile.sql
# insert the orderkeys as a comma-separated list into the queryfile
cat $ORDERKEYS | sed "s/[0-9] /&A/g" | sed "s/A.*$/,/g" | tr -d '\n' | sed "s/,,$//g" | sed "s/,/,,\n/g" >>
$TMPDIR/tmp_queryfile.sql
echo ")" >> $TMPDIR/tmp_queryfile.sql
echo "ORDER BY l_orderkey, l_linenumber;" >> $TMPDIR/tmp_queryfile.sql
echo >> $TMPDIR/tmp_queryfile.sql
echo "SELECT * FROM orders WHERE o_orderkey in (" >> $TMPDIR/tmp_queryfile.sql
# insert the orderkeys as a comma-separated list into the queryfile
cat $ORDERKEYS | sed "s/[0-9] /&A/g" | sed "s/A.*$/,/g" | tr -d '\n' | sed "s/,,$//g" | sed "s/,/,,\n/g" >>
$TMPDIR/tmp_queryfile.sql
#cat $ORDERKEYS >> $TMPDIR/tmp_queryfile.sql
echo ")" >> $TMPDIR/tmp_queryfile.sql
echo "ORDER BY o_orderkey;" >> $TMPDIR/tmp_queryfile.sql
echo >> $TMPDIR/tmp_queryfile.sql
echo "EXIT;" >> $TMPDIR/tmp_queryfile.sql

cat $TMPDIR/tmp_queryfile.sql | dwpplus -h $ROOTHOST -p $ROOTPORT -u sys -P exasol > $OUTFILE
# rm -f $TMPDIR/tmp_queryfile.sql

# -----
# acidquery.sh
# -----
#!/bin/bash
# params <HOST> <LOGDIR> <OKEY>

HOST=$1
LOGDIR=$2
OKEY=$3

ROOTPORT=26822

TMPFILE=$LOGDIR/query_tmp.sql

rm -f $TMPFILE

echo "set autocommit off" > $TMPFILE
echo "open schema tpc;" >> $TMPFILE
echo "select sum(trunc(trunc(l_extendedprice*(1-l_discount),2)*(1+l_tax),2))" >> $TMPFILE
echo "from lineitem" >> $TMPFILE
echo "where l_orderkey=$OKEY;" >> $TMPFILE

dwpplus -h $HOST -p $ROOTPORT -u tpcuser -P tpcuser -f $TMPFILE | grep "\((SUM|/--|\ \)"

#rm -r $TMPFILE

# -----
# transaction/acidtransaction.cpp
# -----
#include "querystream.h"

#include <iostream>
using std::cerr;
using std::cout;
using std::cin;
using std::endl;
using std::ostream;

#include <iomanip>

#include <fstream>
using std::ofstream;
using std::ifstream;

#include <string>
using std::string;

#include <sstream>
using std::ostringstream;

// for rand()
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>

#define MAX_LINE_SIZE 10000

#include <sys/time.h>

#include <stdlib.h>
#include <math.h>

double EPS=0.00001;
char* waitBeforeEnv;
char* waitAfterRBEEnv;
char* waitAfterCommitEnv;
bool enableStreamWait;
char* streamWaitTime;

ofstream valueLog;

double trunc( double d, size_t p )
{

```

```

    // calculating precFactor as 10^p (p>=0)
    unsigned long precFactor = 1;
    for ( size_t i=1; i<=p; i++ )
    {
        precFactor *= 10;
    }

    double truncation = (trunc(d*precFactor))/((double)precFactor);
    if ( ((truncation + (1.0/precFactor))-d) < EPS )
    truncation += (1.0/precFactor);
    return truncation;
}

/** \brief Calculates the time passed between t1 and t2. */
double timediff( timeval t1, timeval t2 )
{
    double passed = 1.0e-6 * ( t2.tv_usec - t1.tv_usec );
    passed += ( t2.tv_sec - t1.tv_sec );
    if ( passed < 0 )
        passed = -passed;
    return passed;
}

/** \brief Deletes from a given string all occurrences of another string at beginning and end. */
string trim( string &s, const string drop = " " )
{
    string r = s.erase( s.find_last_not_of(drop)+1 );
    return r.erase( 0, r.find_first_not_of(drop) );
}

void printTimestamp( ostream &mystream )
{
    timeval current_time;
    gettimeofday( &current_time, 0 );
    struct tm tm_time;
    localtime_r( &current_time.tv_sec, &tm_time );
    int year = 1900 + tm_time.tm_year;
    int month = tm_time.tm_mon;
    int day = tm_time.tm_mday;
    int hour = tm_time.tm_hour;
    int minute = tm_time.tm_min;
    int seconds = tm_time.tm_sec;
    mystream << std::setw(4) << std::setfill('0') << year << "-" << std::setw(2) << std::setfill('0') << month << "-" <<
    std::setw(2) << std::setfill('0') << day << " " << std::setw(2) << std::setfill('0') << hour << ":" << std::setw(2) <<
    std::setfill('0') << minute << ":" << std::setw(2) << std::setfill('0') << seconds << "." << std::setw(6) <<
    std::setfill('0') << current_time.tv_usec;
}

void processRetSize( int retSize, int expectedRetSize, QueryStream &qs )
{
    if ( retSize != expectedRetSize )
    {
        if ( retSize == -3 )
        {
            cerr << "Unexpected ROLLBACK" << endl;
            throw rollbackException();
        }
        cerr << "!!! ERROR Unexpected return size. Expected " << expectedRetSize << " but got " << retSize << "...
        exiting. !!!" << endl;
        qs.disconnectFromServer();
        exit(-1);
    }
}

// executes a COMMIT
void executeCommit( QueryStream &qs )
{
    if ( enableStreamOWait )
    {
        if ( streamOWaitTime != NULL )
        {
            string sleeptime = string("sleep ") + string( streamOWaitTime );
            system( sleeptime.c_str() );
        }
    }
    printTimestamp( cout );
    cout << endl;
    int retsize = qs.execute( "COMMIT;" );
    processRetSize( retsize, -1, qs );
    cout << "commit done at ";
    printTimestamp( cout );
    cout << endl;
}

// executes a ROLLBACK
void executeRollback( QueryStream &qs )
{
    printTimestamp( cout );
    cout << endl;
    int retsize = qs.execute( "ROLLBACK;" );
    processRetSize( retsize, -1, qs );
}

// Waits for a keypress
void dwait()
{
    printTimestamp( cout );
    cout << ": Press a key, please, to continue" << endl;
    cin.clear();
    char ch = cin.get();
}

```

```

// Does not wait
void nowait()
{
    printTimestamp( cout );
    cout << endl;
    if ( waitBeforeEnv != NULL )
    {
        string sleepTime = string("sleep ") + string(waitBeforeEnv);
        system( sleepTime.c_str() );
    }
}

// executes the ACID-transaction for given values O_KEY, L_KEY and DELTA
// until the point where a COMMIT or a ROLLBACK can be executed.
// (old) values of rprice, quantity, tax, disc, extprice and ototal
// are returned.
// With exception of the update to writelock some table at the beginning
// quite identical to the pseudocode in the specification.
void performAcidTransactionVariant1( QueryStream &qs, string O_KEY, string L_KEY, int DELTA, int &pkey, int &skey, double
&rprice, double &quantity, double &tax, double &disc, double &extprice, double &ototal, string lineitemname, string
ordersname, string historyname )
{
    ostringstream queryText("");
    queryText.setf( std::ios::fixed );
    queryText.precision( 2 );
    int retSize;

    // Read from orders
    queryText.str("");
    queryText << "SELECT o_totalprice, o_orderkey FROM " << ordersname << " WHERE o_orderkey=" << O_KEY << ";";
    retSize = qs.execute( queryText.str() );
    processRetSize( retSize, 1, qs );
    ototal = qs.getDouble( 0 );
    qs.freeFetchArray();
    cout << "SELECT done at ";
    printTimestamp( cout );
    cout << endl;

    // Read from lineitem
    queryText.str("");
    queryText << "SELECT l_quantity, l_extendedprice, l_partkey, l_suppkey, l_tax, l_discount, l_orderkey, l_linenummer FROM
" << lineitemname << " WHERE l_orderkey=" << O_KEY << " AND l_linenummer=" << L_KEY << ";";
    retSize = qs.execute( queryText.str() );
    processRetSize( retSize, 1, qs );
    quantity = qs.getDouble( 0 );
    extprice = qs.getDouble( 1 );
    pkey = qs.getInt( 2 );
    skey = qs.getInt( 3 );
    tax = qs.getDouble( 4 );
    disc = qs.getDouble( 5 );
    qs.freeFetchArray();
    cout << "SELECT done at ";
    printTimestamp( cout );
    cout << endl;

    double old_ototal = ototal;

    // Calculate new values
    ototal = ototal - trunc( trunc( extprice * ( 1-disc ), 2 ) * (1+tax), 2 );
    rprice = trunc( extprice/quantity, 2 );
    double cost = trunc( rprice * DELTA, 2 );
    double new_extprice = extprice + cost;
    double new_ototal = trunc( new_extprice * (1.0-disc), 2 );
    new_ototal = trunc( new_ototal * (1.0+tax), 2 );
    new_ototal = ototal + new_ototal;

    valueLog << "| old | " << DELTA << " | " << extprice << " | " << quantity << " | " << disc << " | " << tax << " | "
<< old_ototal << " |" << endl;
    valueLog << "| new | " << DELTA << " | " << new_extprice << " | " << quantity + DELTA << " | " << disc << " | " <<
tax << " | " << new_ototal << " |" << endl;

    // Update lineitem
    queryText.str("");
    queryText << "UPDATE " << lineitemname << " ";
    queryText << "SET l_extendedprice=" << new_extprice << " ";
    queryText << "l_quantity=" << quantity + DELTA << " ";
    queryText << "WHERE l_orderkey=" << O_KEY << " AND l_linenummer=" << L_KEY << ";";
    retSize = qs.execute( queryText.str() );
    processRetSize( retSize, -1, qs );
    cout << "UPDATE done at ";
    printTimestamp( cout );
    cout << endl;

    // Update orders
    queryText.str("");
    queryText << "UPDATE " << ordersname << " ";
    queryText << "SET o_totalprice=" << new_ototal << " ";
    queryText << "WHERE o_orderkey=" << O_KEY << ";";
    retSize = qs.execute( queryText.str() );
    processRetSize( retSize, -1, qs );
    cout << "UPDATE done at ";
    printTimestamp( cout );
    cout << endl;

    // Insert int history
    queryText.str("");
    queryText << "INSERT INTO " << historyname << " VALUES ( " << pkey << " , " << skey << " , " << O_KEY << " , " << L_KEY <<
" , " << DELTA << " , current_timestamp);";
    retSize = qs.execute( queryText.str() );
    processRetSize( retSize, -1, qs );
}

```

```

        cout << "INSERT done at ";
        printTimestamp( cout );
        cout << endl;
    }

int main( int argc, char** argv )
{
    srand( time(0) );
    cout.setf( std::ios::fixed );
    cout.precision( 2 );
    if ( argc != 10 && argc != 11 && argc != 12 && argc!=13 && argc != 14 && argc != 15 )
    {
        cerr << "usage: " << argv[0] << " <host> <port> <commitaction> <waitaction> <inputfile> <logfile> <user> <password>
<valueolog> [<lineitemname> <ordersname> <historyname>] [<printafter> [<successfile>]]" << endl;
        cerr << endl;
        cerr << "\thost      name of host to connect to" << endl;
        cerr << "\tport      port number on host to connect to" << endl;
        cerr << "\tcommit_type what to do at end of transaction. May be DECIDE (wait for input then) or anything
else (especially COMMIT or ROLLBACK)." << endl;
        cerr << "\tlogfile    the logfile to write output to." << endl;
        cerr << "\tuser      name of user to connect to DWA" << endl;
        cerr << "\tpassword  password used to authenticate user" << endl;
        cerr << "\tvalueolog  logfile used for old and new values of delta, l_extendedprice and all the other values which
are part of the transaction" << endl;
        cerr << "\tlineitem  name of the lineitem table to use (default: LINEITEM)" << endl;
        cerr << "\torders   name of the orders table to use (default: ORDERS)" << endl;
        cerr << "\thistory  name of the history table to use (default: HISTORY)" << endl;
        cerr << "\tprintafter number of transactions (committed or rolled back) after which to print a text (default: -1 =
print no text)" << endl;
        cerr << "\tsuccessfile successfile used in durability test (default: no success file)" << endl;
        cerr << endl << "\tWAITBEFORECOMMIT (environment variable): Before a commit is done wait this time (in
seconds)." << endl;
        cerr << "\tWAITAFTERCOMMIT (environment variable): After a successful commit was done wait this time
(in seconds)." << endl;
        cerr << "\tWAITAFTERROLLBACK (environment variable): After an automatic rollback wait this time (in
seconds)." << endl;
        cerr << "\tADDITIONALWAITTIME (environment variable): If <printafter> transactions were successully
committed and if this is stream 0 wait this additional time (in seconds) before each commit." << endl;

        return 1;
    }
    Configuration config;
    config.host = argv[1];
    config.port = argv[2];

    config.user = argv[7];
    config.password = argv[8];

    std::string actionName = argv[3];
    std::string waitName = argv[4];

    string inputfilename = argv[5];
    string logfilename = argv[6];

    string lineitemname = "LINEITEM";
    string ordersname = "ORDERS";
    string historyname = "HISTORY";

    string valueslogname = argv[9];

    if ( argc == 13 || argc == 14 || argc == 15 )
    {
        lineitemname = argv[10];
        ordersname = argv[11];
        historyname = argv[12];
    }

    int printAfter = -1;
    if ( argc == 11 || argc == 12 )
    {
        printAfter = atoi( argv[10] );
    }
    else if ( argc == 14 || argc == 15 )
    {
        printAfter = atoi( argv[13] );
    }

    bool useSuccessfile = false;
    ofstream success;
    if ( argc == 12 )
    {
        useSuccessfile = true;
        success.open( argv[11] );
    }
    else if ( argc == 15 )
    {
        useSuccessfile = true;
        success.open( argv[14] );
    }
    if (useSuccessfile)
    {
        success << "| PKEY | SKEY | O_KEY | L_KEY | DELTA | CURRENT_DATE_TIME |" << endl;
    }

    void (*commitaction)( QueryStream &qs );

    if ( actionName == "COMMIT" )
        commitaction = &executeCommit;
    else if ( actionName == "ROLLBACK" )
        commitaction = &executeRollback;

```

```

else
{
    cerr << "action must be one of COMMIT, ROLLBACK" << endl;
    exit( -1 );
}

bool wait = false;
void (*waitaction)();
if ( waitName == "WAIT" )
{
    waitaction = &doWait;
    wait = true;
}
else if ( waitName == "NOWAIT" )
{
    waitaction = &noWait;
}
else
{
    cerr << "wait must be one of WAIT, NOWAIT" << endl;
    exit( -1 );
}

// open streams
ofstream logfile( logfilename.c_str() );
logfile << " | RPRICE | QUANTITY | TAX | DISC | EXTPRICE | OTOTAL |" << endl;

valueLog.open( valuesLogname.c_str() );
valueLog.setf( std::ios::fixed );
valueLog.precision( 2 );
valueLog << " | old / new | DELTA | L_EXTENDED | L_QUANTITY | DISC | TAX | O_TOTAL |" << endl;
//ofstream streamlogfile("acidtransaction.log");
ifstream inputFile( inputfilename.c_str() );

bool isStream0 = false;
enableStream0Wait = false;
// checking whether we have stream 0. This is checked via the inputfilename which must be of form *s$i.log ($i is
extracted)
if ( ( inputfilename.size()>5      &&      inputfilename[inputfilename.size()-5]!='0'      &&      !      isdigit(
inputfilename[inputfilename.size()-6] ) ) )
{
    isStream0 = true;
    stream0WaitTime = getenv("ADDITIONALWAITTIME");

    if ( stream0WaitTime != NULL )
        cout << "will sleep additional " << stream0WaitTime << " seconds before a commit when enough
transactions were committed." << endl;
}

// get the wait delays before commits and after (automatic) rollbacks
waitBeforeEnv = getenv("WAITBEFORECOMMIT");
if ( waitBeforeEnv != NULL )
    cout << "will sleep " << waitBeforeEnv << " seconds before each commit" << endl;

waitAfterRBEEnv = getenv("WAITAFTERROLLBACK");
if ( waitAfterRBEEnv != NULL )
    cout << "will sleep " << waitAfterRBEEnv << " seconds after each automatic rollback" << endl;

waitAfterCommitEnv = getenv("WAITAFTERCOMMIT");
if ( waitAfterCommitEnv != NULL )
    cout << "will sleep " << waitAfterCommitEnv << " seconds after each successful commit" << endl;

// open querystream
QueryStream qs( "ACID" );//, &streamlogfile );

// connect to server
bool connected = false;
while ( ! connected )
{
    if ( qs.connectToServer( config ) )
        connected = true;
    else
        cout << "Error connecting to server. Trying to reconnect." << endl;
}

// open TPC-schema
int retSize;

retSize = qs.execute( "open schema tpc;" );
processRetSize( retSize, -1, qs );

retSize = qs.execute( "COMMIT;" );
processRetSize( retSize, -1, qs );

// execute ACID-Transactions as long as the input file contains more values for O_KEY, L_KEY, DELTA
// in case of waiting just the transaction for the first triple is executed.
string O_KEY, L_KEY;
int DELTA;

int committedOrRolledBack = 0;

bool gotRollback = false;

do
{
    if ( committedOrRolledBack == printAfter )
    {
        cout << "committed or rolled back " << printAfter << " transactions" << endl;
        if ( isStream0 )
            enableStream0Wait = true;
    }
}

```



```

}
// read next values from inputfile
if ( ! gotRollback )
{
    inputfile >> O_KEY;
    inputfile >> L_KEY;
    inputfile >> DELTA;
    if ( inputfile.eof() )
        break;
}

// execute the ACID transaction with O_KEY, L_KEY and DELTA
// but do neither COMMIT nor ROLLBACK
double rprice, quantity, tax, disc, extprice, ototal;
int pkey, skey;
try
{
    performAcidTransactionVariant1( qs, O_KEY, L_KEY, DELTA, pkey, skey, rprice, quantity, tax,
disc, extprice, ototal, lineitemname, ordersname, historyname );
    committedOrRolledBack++;

    // got no rollback exception
    gotRollback=false;
}
catch( rollbackException )
{
    // got rollback exception
    gotRollback=true;
    valueLog << "transaction rolled back automatically" << endl;
    if ( waitAfterRBEEnv != NULL )
    {
        string sleepTime;
        double randomTime = ((double) rand())/((double)RAND_MAX+1) * atof( waitAfterRBEEnv );
        /*if ( committedOrRolledBack > 0.8*printAfter && committedOrRolledBack < printAfter )
        {
            char duration[10];
            sprintf( duration, "%.1f", atof( waitAfterRBEEnv ) * 5 );
            sleepTime = string("sleep ") + string(duration);
        }
        else
        {
            sleepTime = string("sleep ") + string(waitAfterRBEEnv);
        }*/
        char random_sleep[10];
        sprintf( random_sleep, "%.2f", randomTime );
        sleepTime = string("sleep ") + string(random_sleep);
        system( sleepTime.c_str() );
    }
    continue;
}
// wait or no wait (waitaction is a function pointer to doWait() or noWait() )
waitaction();

// commit or rollback (commitaction is a function pointer to executeCommit( qs ) or executeRollback( qs )
commitaction( qs );
valueLog << actionName << " successful" << endl;
if ( useSuccessfile )
{
    // write to successfile
    success << " | " << pkey << " | " << skey << " | " << O_KEY << " | " << L_KEY << " | " << DELTA
<< " | ";

    timeval current_time;
    gettimeofday( &current_time, 0 );
    struct tm tm_time;
    localtime_r( &current_time.tv_sec, &tm_time );
    int year = 1900 + tm_time.tm_year;
    int month = tm_time.tm_mon;
    int day = tm_time.tm_mday;
    int hour = tm_time.tm_hour;
    int minute = tm_time.tm_min;
    int seconds = tm_time.tm_sec;
    success << std::setw(4) << std::setfill('0') << year << "-" << std::setw(2) << std::setfill('0')
<< month << "-" << std::setfill('0') << day << " " << std::setw(2) << std::setfill('0') << hour << ":" <<
std::setw(2) << std::setfill('0') << minute << ":" << std::setw(2) << std::setfill('0') << seconds << "." << std::setw(6) <<
std::setfill('0') << current_time.tv_usec << " |" << endl;
}

// write to durability success file (if needed)

// write values to logfile (values returned by driver)
logfile << " | " << rprice << " | " << quantity << " | " << tax << " | " << disc << " | " << extprice << "
| " << ototal << " |" << endl;
logfile.flush();

// wait a little bit before next transaction to allow other (rolled back) transaction streams to get
through
if ( waitAfterCommitEnv != NULL )
{
    string sleepTime;
    double randomTime = ((double) rand())/((double)RAND_MAX+1) * atof( waitAfterCommitEnv );
    /*if ( committedOrRolledBack > 0.8*printAfter && committedOrRolledBack < printAfter )
    {
        char duration[10];
        sprintf( duration, "%.2f", atof( waitAfterCommitEnv ) * 5 );
        sleepTime = string("sleep ") + string(duration);
    }
    else
    {
        sleepTime = string("sleep ") + string(waitAfterCommitEnv);
    }*/
    char random_sleep[10];

```

```

        sprintf( random_sleep, "%.2f", randomTime );
        sleepTime = string("sleep ") + string(random_sleep);
        system( sleepTime.c_str() );
    }

    } while ( !inputfile.eof() && !wait );

    // disconnect and exit
    if ( qs.disconnectFromServer() )
    {
        //cout << "MAIN: successfully disconnected." << endl;
    }
    else
    {
        cout << "MAIN: Error while disconnecting." << endl;
    }
    return 0;
}
# -----
# transaction/querystream.h
# -----
#ifndef QUERYSTREAM_H
#define QUERYSTREAM_H

#include <string>
#include <iostream>

#include "dwaCInterface.h"

/** \brief The maximum length of a column name. */
static const size_t maxColumnNameSize = 40;

/** \brief A structure to describe connection information to a database.
 * Used in QueryStream::connectToServer().
 * \sa QueryStream::connectToServer() */
struct Configuration
{
    /** \brief The hostname of the server. */
    char *host;
    /** \brief The port of the server to connect to. */
    char *port;
    /** \brief The user name used for authentication to the database. */
    char *user;
    /** \brief The password of the user. */
    char *password;
};

/** \brief This exception is thrown when a transaction was automatically rolled back.*/
class rollbackException
{
};

/** \brief A structure used to fetch data from a database.
 * Each database column is represented by one FetchStruct.
 * Only one of the arrays of ODBC-C-datatypes is initialized and filled per column.
 * That one is identified by the c_datatype. */
struct FetchStruct
{
    /** \brief Array for columns of SQLCHAR* type
     *(one large block, easy access by sqlCharArrayAccess). */
    SQLCHAR *sqlCharArray;
    /** \brief To quickly access the i-th string in the sqlCharArray array. */
    SQLCHAR* *sqlCharArrayAccess;
    /** \brief Array for columns of SQLSMALLINT type. */
    SQLSMALLINT *sqlSmallInt;
    /** \brief Array for columns of SQLUSMALLINT type. */
    SQLUSMALLINT *sqlUSmallInt;
    /** \brief Array for columns of SQLINTEGER type. */
    SQLINTEGER *sqlInteger;
    /** \brief Array for columns of SQLUINTEGER type. */
    SQLUINTEGER *sqlUInteger;
    /** \brief Array for columns of SQLBIGINT type. */
    SQLBIGINT *sqlBigInt;
    /** \brief Array for columns of SQLUBIGINT type. */
    SQLUBIGINT *sqlUBigInt;
    /** \brief Array for columns of SQLREAL type. */
    SQLREAL *sqlReal;
    /** \brief Array for columns of SQLDOUBLE type. */
    SQLDOUBLE *sqlDouble;
    /** \brief Array for columns of SQLCHAR type. */
    SQLCHAR *sqlChar;
    /** \brief Array for columns of SQLSCHAR type. */
    SQLSCHAR *sqlSChar;
    /** \brief Array for columns of SQL_DATE_STRUCT type. */
    SQL_DATE_STRUCT *sqlDate;
    /** \brief Array for columns of SQL_NUMERIC_STRUCT type. */
    SQL_NUMERIC_STRUCT *sqlNumeric;

    /** \brief The length of a returned data and an indicator whether it is NULL. */
    SQLLEN *resultLen;

    /** \brief The name of the column. */
    SQLCHAR columnName[maxColumnNameSize];
    /** \brief The length of the column name. */
    SQLSMALLINT nameLength;
    /** \brief The SQL datatype ID of the column. */
    SQLSMALLINT datatype;
    /** \brief The ODBC C datatype ID of the column.
     * Might be set by QueryStream::find_ODBC_C_Type( SQLSMALLINT sqlType ) */
    SQLSMALLINT c_datatype;
};

```

```

    /** \brief The size of the column. */
    SQLINTEGER columnSize;
    /** \brief The number of decimal digits of the column. */
    SQLSMALLINT decimalDigits;
    /** \brief Whether the column allows NULL values. */
    SQLSMALLINT nullable;
};

/** \brief The class represents a query stream to a server.
 * This query stream is slightly different from that one used by the Performance test
 * since it is for implementing the ACID-Transaction for which queries are only
 * executed to get (or change) single lines whose values must be accessible since
 * they will be changed.
 *
 * The execute(...) function was slightly changed to fetch at most two rows and not output or delete
 * the result set. Furthermore a new function freeFetchArray() was introduced to allow the user of
 * this class to free the fetchMe array. Last but not least the fetchMe array of fetched rows is
 * publicly accessible by several getDatatype(column) functions to allow reading of returned data.
 *
 * In addition to that the mapping of C-Datatypes to SQL-Types might have changed slightly to ensure
 * mapping of some columns to numerical C-Types. This affects find_ODBC_C_Type(...). */
class QueryStream
{
public:
    /** \brief Constructor.
     * \param streamId The identifier used to identify this query stream.
     * \param logfile The stream to be used for logging. */
    QueryStream( std::string streamId, std::ostream* logfile = &std::cout );
    /** \brief Tries to establish a connection to a server.
     * \param configuration The connection and login information for the server.
     * \return true if the connection could be established, false otherwise. */
    bool connectToServer( Configuration configuration );
    /** \brief Executes a given statement or query.
     * \param statement The text of the statement or query to execute.
     * \param fetchSize The number of rows to fetch with a single fetch command
     * (bigger fetch size might reduce overhead, but increases memory usage).
     * \return The number of rows in resultset (-2 (error), -1 (no resultset), 0,1 or 2 (means more than one)).
     */
    int execute( std::string statement );
    /** \brief Closes the connection to the server.
     * \return true if the connection was closed successfully, false if errors occurred. */
    bool disconnectFromServer();
    /** \brief frees the memory allocated to store the resultset.
     * Call this exactly if execute did not return -1. */
    void freeFetchArray();

    int getInt( size_t columnNumber );

    double getDouble( size_t columnNumber );
protected:
    /** \brief The maximum size of a string. */
    static const size_t maxStringSize = 200;
    /** \brief The number of rows to fetch in a single fetch command. */
    SQLINTEGER fetchBlockSize;
    /** \brief The number to use for fetchBlockSize if it is not
     * given as argument to execute(). */
    static const size_t standardFetchBlockSize = 100;

    /** \brief Binds a column to an entry in the fetchMe array.
     * This must be done before fetching a resultset.
     * This function arranges memory allocation and binding function.
     * \param colNo The number of the resultset column to bind to.
     * \param localColNo The index in the fetchMe array the resultset data shall be stored.
     * \param datatype The SQL datatype of the column to bind to. */
    void bindColumn( int colNo, int localColNo, SQLSMALLINT datatype );

    /** \brief Allocates memory for the buffers needed in binding resultset columns.
     * The array corresponding to the given datatype of the given index
     * of the fetchMe array is initialized to the size of fetchBlockSize.
     * If the datatype is a string datatype, the string buffer will have the
     * size maxStringSize.
     * \param localColNo The index in the fetchMe array the data shall be allocated for.
     * \param datatype The ODBC-C-datatype memory shall be allocated for. */
    void allocateMemoryForBind( int localColNo, SQLSMALLINT datatype );

    /** \brief Maps an SQL-datatype-ID to a ODBC-C-datatype-ID to which it can be converted.
     * This is needed for binding columns and fetching results from a remote database.
     * \param sqlType The SQL-datatype-ID whose corresponding ODBC-C-datatype-ID is wanted.
     * \return The ODBC-C-datatype-ID for the given SQL-datatype-ID */
    SQLSMALLINT find_ODBC_C_Type( SQLSMALLINT sqlType );

    /** \brief Format resultset header for output. */
    void outputHeader();

    /** \brief Formats fetched rows for output.
     * The data in fetchMe is printed for each column and all fetched rows. */
    void outputFetchedRows();

    /** \brief The stream identifier. */
    std::string myID;

    /** \brief The environment handle used. */
    SQLHENV henv;
    /** \brief The connection handle used. */
    SQLHDBC hdbc;
    /** \brief The only statement handle used. */
    SQLHSTMT hstmt;

    /** \brief The structure used to fetch result sets of queries. */
    FetchStruct *fetchMe;

```

```

    /** \brief The number of rows in a result set. */
    SQLSMALLINT numResultCols;

    /** \brief The number of rows fetched by last fetch command. */
    SQLINTEGER rowsFetched;

    /** \brief The stream used for logging results.*/
    std::ostream* logfile;

private:
    /** \brief Parses the return code of an statement and gives additional information on error. */
    SQLRETURN execCritical(SQLRETURN retcode, SQLSMALLINT handle_type = 0, SQLHANDLE handle = 0);
};

#endif // QUERYSTREAM_H
# -----
# transaction/querystream.cpp
# -----
#include "querystream.h"
#include <iostream>

#ifdef DEBUG
#ifdef DEBUG
#define DEBUGOUT( X ) cout << "DEBUG:" << X
#else
#define DEBUGOUT( X )
#endif
#endif

#define LOG *logfile

using std::cout;
using std::endl;
using std::cerr;

#include <string>
using std::string;

SQLRETURN QueryStream::execCritical(SQLRETURN retcode, SQLSMALLINT handle_type, SQLHANDLE handle)
{
    switch(retcode)
    {
    case SQL_INVALID_HANDLE:
        cout << "invalid handle" << endl;
        throw "ERROR use of invalid handle!\n";
        break;
    case SQL_ERROR:
        {
            SQLCHAR err[1000];
            SQLCHAR state[6];
            SQLINTEGER native;
            SQLSMALLINT len;
            SQLSMALLINT i=1;
            while ( handle !=0 && DWAGetDiagRec(handle_type, handle, i, state, &native,err,1000,&len) == SQL_SUCCESS )
            {
                cout << state << ":" << err << endl;
                i++;
                if ( string( (const char*) err).find( "GlobalTransactionRollback" ) )
                {
                    throw rollbackException();
                }
            }
            cout << "SQL error" << endl;
            throw "An error occured while executing command!\n";
            break;
        }
    case SQL_SUCCESS_WITH_INFO:
        cout << "Problem: Has Info, which should not be" << endl;
        throw "Problem: Has Info, which should not be!\n";
        break;
    case SQL_STILL_EXECUTING:
        cout << "Still executing. This should not be should only";
        cout << "return on successful execution!" << endl;
        throw "Unexpectedly still executing!\n";
        break;
    case SQL_NO_DATA:
    case SQL_SUCCESS:
        break;
    default:
        // should not reach here...
        cout << "unexpected return code: " << retcode << endl;
        throw "unexpected return code!\n";
        break;
    }
    return retcode;
}

QueryStream::QueryStream( std::string streamId, std::ostream* logfile_ )
{
    myID = streamId;
    henv = NULL;
    hdbc = NULL;
    hstmt = NULL;
    logfile = logfile_;
}

bool QueryStream::connectToServer( Configuration cfg )
{
    char localeAttr[]="deu";
    try
    {
        // Allocating environment handle

```

```

execCritical(DWAAllocHandle(SQL_HANDLE_ENV, NULL, &henv));
// Setting up environment to german localization
execCritical(DWASetEnvAttr(henv, DWA_ENV_ATTR_SET_LOCALE_ALL, localeAttr, SQL_NTS));
// Allocating handle for connection
execCritical(DWAAllocHandle(SQL_HANDLE_DBC, henv, &hdbc));
// Turning autocommit off... is this necessary?
execCritical(DWASetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT, (void*)SQL_AUTOCOMMIT_OFF, SQL_NTS));
// Setting up clientname
execCritical(DWASetConnectAttr(hdbc, DWA_CLIENT_NAME, (void*)myID.c_str(), SQL_NTS));
// Connect to server
execCritical(DWAServerConnect(hdbc, cfg.host, SQL_NTS, cfg.port, SQL_NTS, cfg.user, SQL_NTS, cfg.password, SQL_NTS));
}
catch(...)
{
// might be nice to clean up handles
return false;
}
return true;
}

int QueryStream::execute( std::string statement )
{
// the number of rows in the resultSet
int resultSetSize = -1;

try
{
// put the statement in the logfile
LOG << statement << endl;
// Allocating statement handle
execCritical(DWAAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt), SQL_HANDLE_DBC, hdbc);
SQLCHAR* statementtext = (SQLCHAR*) statement.c_str();
// execute the statement / query
execCritical(DWAExecDirect(hstmt, statementtext, statement.size() ), SQL_HANDLE_STMT, hstmt );
// get the affected rows (only insert, update, delete)
SQLINTEGER affectedrows;
execCritical(DWARowCount(hstmt,&affectedrows), SQL_HANDLE_STMT, hstmt);
// get the number of columns in the resultSet (if any)
execCritical(DWANumResultCols(hstmt,&numResultCols), SQL_HANDLE_STMT, hstmt);
// no result columns means no query
if ( numResultCols <= 0 )
LOG << "affected rows: " << affectedrows << "\n\n";

// if a resultSet was constructed, retrieve and print it
if ( numResultCols > 0 )
// a resultSet was constructed
{
LOG << '\n';
// setting number of rows to fetch with each fetch
SQLINTEGER fetchsize = 2; // This is to store the result set and be able to ensure only one line was received
fetchBlockSize = 2;
execCritical(DWASetStmtAttr(hstmt, SQL_ATTR_ROW_ARRAY_SIZE,
(SQLPOINTER) fetchsize , 0 ), SQL_HANDLE_STMT, hstmt);
//cout << "Fetch block size (SQL_ATTR_ROW_ARRAY_SIZE) set to " << fetchsize << endl;
// setting position to store the fetch size for a query
execCritical(DWASetStmtAttr(hstmt, SQL_ATTR_ROWS_FETCHED_PTR,
(SQLPOINTER) &rowsFetched, 0), SQL_HANDLE_STMT, hstmt);
// allocate the fetch struct to be large enough for the columns
fetchMe = new FetchStruct[numResultCols];
// set all pointers to 0 to make it easy to free the correct memory later
for ( int i=0; i<numResultCols; i++ )
{
fetchMe[i].sqlCharArray = 0;
fetchMe[i].sqlCharArrayAccess = 0;
fetchMe[i].sqlSmallInt = 0;
fetchMe[i].sqlUSmallInt = 0;
fetchMe[i].sqlInteger = 0;
fetchMe[i].sqlUInteger = 0;
fetchMe[i].sqlBigInt = 0;
fetchMe[i].sqlUBigInt = 0;
fetchMe[i].sqlReal = 0;
fetchMe[i].sqlDouble = 0;
fetchMe[i].sqlChar = 0;
fetchMe[i].sqlSChar = 0;
fetchMe[i].sqlDate = 0;
fetchMe[i].sqlNumeric = 0;
fetchMe[i].resultLen = 0;
}
execCritical(DWASetStmtAttr(hstmt, SQL_ATTR_ROWS_FETCHED_PTR,
(SQLPOINTER) &rowsFetched, 0), SQL_HANDLE_STMT, hstmt);
// determine the datatype of the column and additional information.
for ( SQLSMALLINT colNr=0; colNr<numResultCols; colNr++ )
{
// Get column description for the column with number colNr
DWADescribeCol( hstmt, colNr+1, fetchMe[colNr].columnName,
maxColumnNameSize, &(fetchMe[colNr].nameLength),
&(fetchMe[colNr].datatype), &(fetchMe[colNr].columnSize),
&(fetchMe[colNr].decimalDigits), &(fetchMe[colNr].nullable) );
fetchMe[colNr].c_datatype = find_ODBC_C_Type( fetchMe[colNr].datatype );

// bind the column to an entry in the fetchMe array
bindColumn( colNr+1, colNr, fetchMe[colNr].c_datatype );
}
// output the resultSet header
outputHeader();
// fetch the columns
do
{
// fetch columns
execCritical(DWAFetch( hstmt ), SQL_HANDLE_STMT, hstmt);
}
}
}

```

```

        // output columns
        outputFetchedRows();
        // correct the result set size
        resultSetSize += rowsFetched;
    } while ( rowsFetched == fetchBlockSize );
    // add the total resultset size to the logfile
    LOG << "\nresultset consists of " << resultSetSize << " rows in " << numResultCols << " columns\n\n";
}
}

catch(rollbackException)
{
    // free the handle for the statement again
    cout << "\nROLLBACK EXECUTING STATEMENT:\n" << statement << '\n' << endl;
    execCritical(DWAFreeHandle( SQL_HANDLE_STMT, hstmt ), SQL_HANDLE_STMT, hstmt);
    return -3;
}
catch( char * c )
{
    LOG << "ERROR EXECUTING STATEMENT\n\n";
    cout << "\nERROR EXECUTING STATEMENT:\n" << statement << '\n' << endl;
    cout << c << endl;
    execCritical(DWAFreeHandle( SQL_HANDLE_STMT, hstmt ), SQL_HANDLE_STMT, hstmt);
    return -2;
}
catch(...)
{
    LOG << "ERROR EXECUTING STATEMENT\n\n";
    cout << "\nERROR EXECUTING STATEMENT:\n" << statement << '\n' << endl;
    execCritical(DWAFreeHandle( SQL_HANDLE_STMT, hstmt ), SQL_HANDLE_STMT, hstmt);
    return -2;
}
execCritical(DWAFreeHandle( SQL_HANDLE_STMT, hstmt ), SQL_HANDLE_STMT, hstmt);
return resultSetSize;
}

void QueryStream::freeFetchArray()
{
    for ( int i=0; i<numResultCols; i++ )
    {
        if ( fetchMe[i].sqlCharArray != 0 )
        {
            delete[] fetchMe[i].sqlCharArray;
            delete[] fetchMe[i].sqlCharArrayAccess;
        }
        if ( fetchMe[i].sqlSmallInt != 0 ) delete[] fetchMe[i].sqlSmallInt;
        if ( fetchMe[i].sqlUSmallInt != 0 ) delete[] fetchMe[i].sqlUSmallInt;
        if ( fetchMe[i].sqlInteger != 0 ) delete[] fetchMe[i].sqlInteger;
        if ( fetchMe[i].sqlUInteger != 0 ) delete[] fetchMe[i].sqlUInteger;
        if ( fetchMe[i].sqlBigInt != 0 ) delete[] fetchMe[i].sqlBigInt;
        if ( fetchMe[i].sqlUBigInt != 0 ) delete[] fetchMe[i].sqlUBigInt;
        if ( fetchMe[i].sqlReal != 0 ) delete[] fetchMe[i].sqlReal;
        if ( fetchMe[i].sqlDouble != 0 ) delete[] fetchMe[i].sqlDouble;
        if ( fetchMe[i].sqlChar != 0 ) delete[] fetchMe[i].sqlChar;
        if ( fetchMe[i].sqlSChar != 0 ) delete[] fetchMe[i].sqlSChar;
        if ( fetchMe[i].sqlDate != 0 ) delete[] fetchMe[i].sqlDate;
        if ( fetchMe[i].sqlNumeric != 0 ) delete[] fetchMe[i].sqlNumeric;
        if ( fetchMe[i].resultLen != 0 ) delete[] fetchMe[i].resultLen;
    }
    delete[] fetchMe;
}

int QueryStream::getInt( size_t columnNumber )
{
    size_t i = columnNumber;
    if ( i >= numResultCols )
    {
        cout << "column out of range" << endl;
        throw "column out of range";
    }

    if ( fetchMe[i].sqlCharArray != 0 ) return atoi( (char*) fetchMe[i].sqlCharArrayAccess[0] );

    if ( fetchMe[i].sqlSmallInt != 0 ) return fetchMe[i].sqlSmallInt[0];
    if ( fetchMe[i].sqlUSmallInt != 0 ) return fetchMe[i].sqlUSmallInt[0];
    if ( fetchMe[i].sqlInteger != 0 ) return fetchMe[i].sqlInteger[0];
    if ( fetchMe[i].sqlUInteger != 0 ) return fetchMe[i].sqlUInteger[0];
    if ( fetchMe[i].sqlBigInt != 0 ) return fetchMe[i].sqlBigInt[0];
    if ( fetchMe[i].sqlUBigInt != 0 ) return fetchMe[i].sqlUBigInt[0];
    if ( fetchMe[i].sqlReal != 0 ) return (int) fetchMe[i].sqlReal[0];
    if ( fetchMe[i].sqlDouble != 0 ) return (int) fetchMe[i].sqlDouble[0];
    if ( fetchMe[i].sqlChar != 0 ) return fetchMe[i].sqlChar[0];
    if ( fetchMe[i].sqlSChar != 0 ) return fetchMe[i].sqlSChar[0];

    cout << "cannot cast datatype to Int" << endl;
    throw "cannot cast datatype to Int";
}

double QueryStream::getDouble( size_t columnNumber )
{
    size_t i = columnNumber;
    if ( i >= numResultCols )
    {
        cout << "column out of range" << endl;
        throw "column out of range";
    }

    if ( fetchMe[i].sqlCharArray != 0 ) return atof( (char*) fetchMe[i].sqlCharArrayAccess[0] );

    if ( fetchMe[i].sqlSmallInt != 0 ) return fetchMe[i].sqlSmallInt[0];
    if ( fetchMe[i].sqlUSmallInt != 0 ) return fetchMe[i].sqlUSmallInt[0];
}

```

```

if ( fetchMe[i].sqlInteger != 0 ) return fetchMe[i].sqlInteger[0];
if ( fetchMe[i].sqlUInteger != 0 ) return fetchMe[i].sqlUInteger[0];
if ( fetchMe[i].sqlBigInt != 0 ) return fetchMe[i].sqlBigInt[0];
if ( fetchMe[i].sqlUBigInt != 0 ) return fetchMe[i].sqlUBigInt[0];
if ( fetchMe[i].sqlReal != 0 ) return fetchMe[i].sqlReal[0];
if ( fetchMe[i].sqlDouble != 0 ) return fetchMe[i].sqlDouble[0];
if ( fetchMe[i].sqlChar != 0 ) return fetchMe[i].sqlChar[0];
if ( fetchMe[i].sqlSChar != 0 ) return fetchMe[i].sqlSChar[0];

    cout << "cannot cast datatype to Int" << endl;
    throw "cannot cast datatype to Int";
}

bool QueryStream::disconnectFromServer()
{
    try
    {
        // disconnecting from server
        execCritical(DWADisconnect(hdbc));
        // freeing connection handle
        execCritical(DWAFreeHandle(SQL_HANDLE_DBC, hdbc));
        // freeing environment handle
        execCritical(DWAFreeHandle(SQL_HANDLE_ENV, henv));
    }
    catch(...)
    {
        cout << "Error while disconnecting from server" << endl;
        return false;
    }
    return true;
}

void QueryStream::outputHeader()
{
    int headerLength = 1;
    LOG << "|";
    for ( size_t col=0; col<numResultCols; col++ )
    {
        LOG << fetchMe[col].columnName << "|";
        headerLength += fetchMe[col].nameLength + 1;
    }
    LOG << '\n';
    for ( int i=0; i<headerLength; i++ )
        LOG << '=';
    LOG << '\n';
    (LOG).flush();
}

void QueryStream::outputFetchedRows()
{
    for ( size_t rows=0; rows<rowsFetched; rows++ )
    {
        LOG << "|";
        for ( size_t cols=0; cols<numResultCols; cols++ )
        {
            if ( fetchMe[cols].resultLen[rows] == SQL_NULL_DATA )
            {
                LOG << "|";
                continue;
            }
            switch( fetchMe[cols].c_datatype )
            {
                // string data
                case SQL_C_CHAR:
                case SQL_C_BINARY:
                //case SQL_C_XML:
                //case SQL_C_VARBOOKMARK:
                    LOG << fetchMe[cols].sqlCharArrayAccess[rows] << "|";
                    break;

                // small int
                case SQL_C_SSHORT:
                    LOG << fetchMe[cols].sqlSmallInt[rows] << "|";
                    break;

                // unsigned small int
                case SQL_C_USHORT:
                    LOG << fetchMe[cols].sqlUSmallInt[rows] << "|";
                    break;

                // long int
                case SQL_C_SLONG:
                    LOG << fetchMe[cols].sqlInteger[rows] << "|";
                    break;

                // unsigned long int
                case SQL_C_ULONG:
                    LOG << fetchMe[cols].sqlUInteger[rows] << "|";
                    break;

                // float
                case SQL_C_FLOAT:
                    LOG << fetchMe[cols].sqlReal[rows] << "|";
                    break;

                // double
                case SQL_C_DOUBLE:
                    LOG << fetchMe[cols].sqlDouble[rows] << "|";
                    break;
            }
        }
    }
}

```

```

// unsigned char
case SQL_C_BIT:
case SQL_C_UTINYINT:
    LOG << fetchMe[cols].sqlChar[rows] << "|";
break;

// signed char
case SQL_C_STINYINT:
    LOG << fetchMe[cols].sqlSChar[rows] << "|";
break;

// big int
case SQL_C_SBIGINT:
    LOG << fetchMe[cols].sqlBigInt[rows] << "|";
break;

// unsigned big int
case SQL_C_UBIGINT:
    LOG << fetchMe[cols].sqlUBigInt[rows] << "|";
break;

// date
case SQL_C_TYPE_DATE:
{
    // format date to YYYY-MM-DD for output
    if ( fetchMe[cols].sqlDate[rows].year < 1000 )
        LOG << "0";
    if ( fetchMe[cols].sqlDate[rows].year < 100 )
        LOG << "0";
    if ( fetchMe[cols].sqlDate[rows].year < 10 )
        LOG << "0";
    LOG << fetchMe[cols].sqlDate[rows].year << "-";

    if ( fetchMe[cols].sqlDate[rows].month < 10 )
        LOG << "0";
    LOG << fetchMe[cols].sqlDate[rows].month << "-";

    if ( fetchMe[cols].sqlDate[rows].day < 10 )
        LOG << "0";
    LOG << fetchMe[cols].sqlDate[rows].day << "|";
}
break;

// numeric, i.e. decimal
// not used since it is eaier by far to map decimals to doubles
case SQL_C_NUMERIC:
{
    // format decimal for output
    int scale = fetchMe[cols].sqlNumeric[rows].scale;
    char text[3*SQL_MAX_NUMERIC_LEN+2+scale];
    int pos_T = 3*SQL_MAX_NUMERIC_LEN+scale;
    int pos_N = SQL_MAX_NUMERIC_LEN-1;
    text[3*SQL_MAX_NUMERIC_LEN+1+scale] = '\0';
    //int add = 0;
    //if ( fetchMe[cols].sqlNumeric[rows].sign == 0 )
    //    add=1;
    while ( fetchMe[cols].sqlNumeric[rows].val[pos_N] == 0 && pos_N > 0 )
        pos_N--;

    while ( pos_N >= 0 )
    {
        int current = fetchMe[cols].sqlNumeric[rows].val[pos_N];
        int newVal = 0;
        int currentPos = pos_N;
        while ( currentPos >= 0 )
        {
            //if ( currentPos == 0 )
            //{
            //    current += add;
            //    add = 0;
            //}
            if ( current < 10 )
            {
                fetchMe[cols].sqlNumeric[rows].val[currentPos] = newVal;
                currentPos--;

                if ( currentPos >= 0 )
                {
                    current = current * 256 + fetchMe[cols].sqlNumeric[rows].val[currentPos];
                    newVal = 0;
                }
            }
            else
            {
                newVal = current / 10;
                current = current % 10;
            }
        }
        if ( fetchMe[cols].sqlNumeric[rows].val[pos_N] == 0 )
        {
            pos_N--;
        }
        text[pos_T--] = '0' + current;
        if ( scale == 1 )
        {
            text[pos_T--] = '.';
        }
        scale--;
    }
    if ( scale==0)

```



```

        text[pos_T--] = '0';
while ( scale > 0 )
{
    text[pos_T--] = '0';
    if ( scale == 1 )
    {
        text[pos_T--] = '.';
        text[pos_T--] = '0';
    }
    scale--;
}

if ( fetchMe[cols].sqlNumeric[rows].sign == 0 )
    text[pos_T--] = '-';
LOG << text+pos_T+1 << "|";
}
break;

// other datatype
default:
    cout << "output: datatype " << fetchMe[cols].c_datatype << " not supported yet" << endl;
    throw fetchMe[cols].c_datatype;
break;
}
}
LOG << '\n';
    (LOG).flush();
}
}

void QueryStream::bindColumn( int colNo, int localColNo, SQLSMALLINT c_datatype )
{
    // allocate memory for the column to bind
    allocateMemoryForBind( localColNo, c_datatype );
    // bind the column
    switch ( c_datatype )
    {
        // string data
        case SQL_C_CHAR:
        case SQL_C_BINARY:
        //case SQL_C_XML:
        //case SQL_C_VARBOOKMARK:
            execCritical(DWABindCol(
                hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlCharArray,
                maxStringSize*sizeof(SQLCHAR), fetchMe[localColNo].resultLen));
            break;

        // small int
        case SQL_C_SSHORT:
            execCritical(DWABindCol(
                hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlSmallInt,
                sizeof(SQLSMALLINT), fetchMe[localColNo].resultLen));
            break;

        // unsigned small int
        case SQL_C_USHORT:
            execCritical(DWABindCol(
                hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlUSmallInt,
                sizeof(SQLUSMALLINT), fetchMe[localColNo].resultLen));
            break;

        // long int
        case SQL_C_SLONG:
            execCritical(DWABindCol(
                hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlInteger,
                sizeof(SQLINTEGER), fetchMe[localColNo].resultLen));
            break;

        // unsigned long int
        case SQL_C_ULONG:
            execCritical(DWABindCol(
                hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlUInteger,
                sizeof(SQLUINTEGER), fetchMe[localColNo].resultLen));
            break;

        // float
        case SQL_C_FLOAT:
            execCritical(DWABindCol(
                hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlReal,
                sizeof(SQLREAL), fetchMe[localColNo].resultLen));
            break;

        // double
        case SQL_C_DOUBLE:
            execCritical(DWABindCol(
                hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlDouble,
                sizeof(SQLDOUBLE), fetchMe[localColNo].resultLen));
            break;

        // unsigned char
        case SQL_C_BIT:
        case SQL_C_UTINYINT:
            execCritical(DWABindCol(
                hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlChar,
                sizeof(SQLCHAR), fetchMe[localColNo].resultLen));
            break;

        // signed char
        case SQL_C_STINYINT:
            execCritical(DWABindCol(
                hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlSmallInt,

```

```

        sizeof(SQLCHAR), fetchMe[localColNo].resultLen));
break;

// big int
case SQL_C_SBIGINT:
    execCritical(DWABindCol(
        hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlBigInt,
        sizeof(SQLBIGINT), fetchMe[localColNo].resultLen));
break;

// unsigned big int
case SQL_C_UBIGINT:
    execCritical(DWABindCol(
        hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlUBigInt,
        sizeof(SQLUBIGINT), fetchMe[localColNo].resultLen));
break;

// date
case SQL_C_TYPE_DATE:
    execCritical(DWABindCol(
        hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlDate,
        sizeof(SQL_DATE_STRUCT), fetchMe[localColNo].resultLen));
break;

// numeric, i.e. decimal
case SQL_C_NUMERIC:
    execCritical(DWABindCol(
        hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlNumeric,
        sizeof(SQL_NUMERIC_STRUCT), fetchMe[localColNo].resultLen));
break;

// other datatype
default:
    throw c_datatype;
break;
}
}

void QueryStream::allocateMemoryForBind( int localColNo, SQLSMALLINT datatype )
{
    // allocate memory for resultLen field used to indicate NULL values (among others)
    fetchMe[localColNo].resultLen = new SQLLEN[fetchBlockSize];

    // allocate memory depending on the column datatype
    switch ( datatype )
    {
        // string data
        case SQL_C_CHAR:
        case SQL_C_BINARY:
        //case SQL_C_XML:
        //case SQL_C_VARBOOKMARK:
            fetchMe[localColNo].sqlCharArray = new SQLCHAR[fetchBlockSize*maxStringSize];
            fetchMe[localColNo].sqlCharArrayAccess = new SQLCHAR*[fetchBlockSize];
            for ( size_t i=0; i<fetchBlockSize; i++ )
            {
                fetchMe[localColNo].sqlCharArrayAccess[i] = fetchMe[localColNo].sqlCharArray+i*maxStringSize;
            }
            break;

        // small int
        case SQL_C_SSHORT:
            fetchMe[localColNo].sqlSmallInt = new SQLSMALLINT[fetchBlockSize];
            break;

        // unsigned small int
        case SQL_C_USHORT:
            fetchMe[localColNo].sqlUSmallInt = new SQLUSMALLINT[fetchBlockSize];
            break;

        // long int
        case SQL_C_SLONG:
            fetchMe[localColNo].sqlInteger = new SQLINTEGER[fetchBlockSize];
            break;

        // unsigned long int
        case SQL_C_ULONG:
            fetchMe[localColNo].sqlUInteger = new SQLUINTEGER[fetchBlockSize];
            break;

        // float
        case SQL_C_FLOAT:
            fetchMe[localColNo].sqlReal = new SQLREAL[fetchBlockSize];
            break;

        // double
        case SQL_C_DOUBLE:
            fetchMe[localColNo].sqlDouble = new SQLDOUBLE[fetchBlockSize];
            break;

        // unsigned char
        case SQL_C_BIT:
        case SQL_C_UTINYINT:
            fetchMe[localColNo].sqlChar = new SQLCHAR[fetchBlockSize];
            break;

        // signed char
        case SQL_C_STINYINT:
            fetchMe[localColNo].sqlSChar = new SQLCHAR[fetchBlockSize];
            break;

        // big int

```

```

    case SQL_C_SBIGINT:
        fetchMe[localColNo].sqlBigInt = new SQLBIGINT[fetchBlockSize];
        break;

    // unsigned big int
    case SQL_C_UBIGINT:
        fetchMe[localColNo].sqlUBigInt = new SQLUBIGINT[fetchBlockSize];
        break;

    // date
    case SQL_C_TYPE_DATE:
        fetchMe[localColNo].sqlDate = new SQL_DATE_STRUCT[fetchBlockSize];
        break;

    // numeric, i.e. decimal
    case SQL_C_NUMERIC:
        fetchMe[localColNo].sqlNumeric = new SQL_NUMERIC_STRUCT[fetchBlockSize];
        break;

    // other datatype
    default:
        cout << "memory allocation: datatype " << datatype << " not supported yet" << endl;
        throw datatype;
        break;
    }
}

SQLSMALLINT QueryStream::find_ODBC_C_Type( SQLSMALLINT sqlType )
{
    switch( sqlType )
    {
        case SQL_CHAR:
        case SQL_VARCHAR:
        case SQL_LONGVARCHAR:
        case SQL_WCHAR:
        case SQL_WVARCHAR:
        case SQL_WLONGVARCHAR:
            return SQL_C_CHAR;
        break;

        case SQL_DECIMAL:
        case SQL_NUMERIC:
            //return SQL_C_NUMERIC;
            return SQL_C_DOUBLE;
            //return SQL_C_CHAR;
        break;

        case SQL_SMALLINT:
            return SQL_C_SSHORT;
        break;

        case SQL_INTEGER:
            return SQL_C_SLONG;
        break;

        case SQL_REAL:
        case SQL_FLOAT:
        case SQL_DOUBLE:
            return SQL_C_DOUBLE;
        break;

        case SQL_BIT:
            return SQL_C_BIT;
        break;

        case SQL_TINYINT:
            return SQL_C_STINYINT;
        break;

        case SQL_BIGINT:
            return SQL_C_SBIGINT;
        break;

        case SQL_BINARY:
        case SQL_VARBINARY:
        case SQL_LONGVARBINARY:
            return SQL_C_BINARY;
        break;

        case SQL_TYPE_DATE:
            return SQL_C_TYPE_DATE;
        break;

        default:
            cout << "unsupported datatype, don't know corresponding C type: " << sqlType << endl;
            throw sqlType;
            break;
    }
}

# -----
# create_history_table.sql
# -----
OPEN SCHEMA tpc;

-- KEYS DEC(13)
-- INTEGERS DEC(10)
-- DECIMALS DEC(12,2)

```

```

CREATE OR REPLACE TABLE HISTORY( H_P_KEY DEC(13), H_S_KEY DEC(13), H_O_KEY DEC(13), H_L_KEY DEC(10), H_DELTA DEC(10),
H_DATE_T TIMESTAMP );

COMMIT;
exit;
# -----
# create_history_table2.sql
# -----
OPEN SCHEMA tpc;

-- KEYS      DEC(13)
-- INTEGERS DEC(10)
-- DECIMALS DEC(12,2)

CREATE OR REPLACE TABLE HISTORY2( H_P_KEY DEC(13), H_S_KEY DEC(13), H_O_KEY DEC(13), H_L_KEY DEC(10), H_DELTA DEC(10),
H_DATE_T TIMESTAMP );

COMMIT;
exit;
# -----
# clear_history.sql
# -----
set autocommit off;
open schema tpc;
delete * from history;
commit;
exit;

# -----
# clear_history2.sql
# -----
set autocommit off;
open schema tpc;
delete * from history2;
commit;
exit;

# -----
# store_history.sql
# -----
set autocommit off
open schema tpc;
select * from history;
rollback;
exit

# -----
# store_history2.sql
# -----
set autocommit off
open schema tpc;
select * from history2;
rollback;
exit

```

Appendix D: Query Text and Query Output

```

*****
* TPC-H Query 1 0 *
*****
-- Minor modification - date arithmetic for days ( 2.2.3.3 c )
select
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
from
    lineitem
where
    l_shipdate <= date '1998-12-01' - 90
group by
    l_returnflag,
    l_linestatus
order by
    l_returnflag,
    l_linestatus;

/L_RETURNFLAG/L_LINESTATUS/SUM_QTY/SUM_BASE_PRICE/SUM_DISC_PRICE/SUM_CHARGE/AVG_QTY/AVG_PRICE/AVG_DISC/COUNT_ORDER/
=====
/A/F/37734107.00/56586554400.73/53758257134.87/55909065222.83/25.52/38273.13/0.05/1478493.00/
/N/F/991417.00/1487504710.38/1413082168.05/1469649223.19/25.52/38284.47/0.05/38854.00/
/N/O/74476040.00/111701729697.74/106118230307.61/110367043872.50/25.50/38249.12/0.05/2920374.00/
/R/F/37719753.00/56568041380.90/53741292684.60/55889619119.83/25.51/38250.85/0.05/1478870.00/

resultset consists of 4 rows in 10 columns

*****
* TPC-H Query 2 0 *
*****
-- Minor modification - result set limit ( 2.1.2.9.3 )
select
    s_acctbal,
    s_name,
    n_name,
    p_partkey,
    p_mfgr,
    s_address,
    s_phone,
    s_comment
from
    part,
    supplier,
    partsupp,
    nation,
    region
where
    p_partkey = ps_partkey
    and s_suppkey = ps_suppkey
    and p_size = 15
    and p_type like '%BRASS'
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'EUROPE'
    and ps_supplycost = (
        select
            min(ps_supplycost)
        from
            partsupp,
            supplier,
            nation,
            region
        where
            p_partkey = ps_partkey
            and s_suppkey = ps_suppkey
            and s_nationkey = n_nationkey
            and n_regionkey = r_regionkey
            and r_name = 'EUROPE'
    )
order by
    s_acctbal desc,
    n_name,
    s_name,
    p_partkey
LIMIT 100
;

/S_ACCTBAL/S_NAME/N_NAME/P_PARTKEY/P_MFGR/S_ADDRESS/S_PHONE/S_COMMENT/
=====
/9938.53/Supplier#000005359          /UNITED KINGDOM          /185358.00/Manufacturer#4          /QKuHYh,vZGiwu2FWEJoLDx04/33-
429-790-6131/uriously regular requests hag/
/9937.84/Supplier#000005969          /ROMANIA          /108438.00/Manufacturer#1
/ANDENSOSmk,miq23Xfb5Rwt6dvUcvvt6Qa/29-520-692-3537/efully express instructions. regular requests against the slyly fin/
/9936.22/Supplier#000005250          /UNITED KINGDOM          /249.00/Manufacturer#4          /B3rqp0xbSEim4Mpy2RH J/33-320-
228-2957/etec about the furiously final accounts. slyly ironic pinto beans sleep inside the furiously/

```

```

/9923.77|Supplier#000002324 |GERMANY |29821.00|Manufacturer#4 |y3OD9UyWSTok|17-779-299-
1839|ackages boost blithely. blithely regular deposits c/
/9871.22|Supplier#000006373 |GERMANY |43868.00|Manufacturer#5 |J8fcXWstQM|17-813-485-
8637|etect blithely bold asymptotes. fluffily ironic platelets wake furiously; blit/
----- rows deleted -----
/7912.91|Supplier#000004211 |GERMANY |184210.00|Manufacturer#4
/2wQRVovHrm3,v03IKzfTD,1PysFXQFFOG|17-266-947-7315|ay furiously regular platelets. cou/
/7894.56|Supplier#000007981 |GERMANY |85472.00|Manufacturer#4 |NSJ96vMROAbeXP|17-963-404-
3760|ic platelets affix after the furiously/
/7887.08|Supplier#000009792 |GERMANY |164759.00|Manufacturer#3 |Y28ITVeYrit3kIGdV2K8fSZ
V2UqT5H10tz|17-988-938-4296|ckly around the carefully fluffy theodolites. slyly ironic pack/
/7871.50|Supplier#000007206 |RUSSIA |104695.00|Manufacturer#1 |3w fNCnrVmvJjE95sgWZzvW|32-
432-452-7731|ironic requests. furiously final theodolites cajole. final, express packages sleep. quickly reg/
/7852.45|Supplier#000005864 |RUSSIA |8363.00|Manufacturer#4 |WCNEBPZeSXh3h,c|32-454-883-
3821|usly unusual pinto beans. brave ideas sleep carefully quickly ironi/
/7850.66|Supplier#000001518 |UNITED KINGDOM |86501.00|Manufacturer#1 |ONda3YjIHKJOC|33-730-383-
3892|ifts haggle fluffily pending pai/
/7843.52|Supplier#000006683 |FRANCE |11680.00|Manufacturer#4
/2Z0JGkiv0YI00cFwUGfviIbhzCdy|16-464-517-8943| express, final pinto beans x-ray slyly asymptotes. unusual, unusual/

```

resultset consists of 100 rows in 8 columns

```

*****
* TPC-H Query 3 0 *
*****
-- Minor modification - result set limit ( 2.1.2.9.3 )
select
    l_orderkey,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    o_orderdate,
    o_shippriority
from
    customer,
    orders,
    lineitem
where
    c_mktsegment = 'BUILDING'
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate < date '1995-03-15'
    and l_shipdate > date '1995-03-15'
group by
    l_orderkey,
    o_orderdate,
    o_shippriority
order by
    revenue desc,
    o_orderdate
LIMIT 10
;

```

```

/L_ORDERKEY/REVENUE/O_ORDERDATE/O_SHIPPRIORITY/
=====
/2456423.00|406181.01|1995-03-05|0.00|
/3459808.00|405838.70|1995-03-04|0.00|
/492164.00|390324.06|1995-02-19|0.00|
/1188320.00|384537.94|1995-03-09|0.00|
/2435712.00|378673.06|1995-02-26|0.00|
/4878020.00|378376.80|1995-03-12|0.00|
/5521732.00|375153.92|1995-03-13|0.00|
/2628192.00|373133.31|1995-02-22|0.00|
/993600.00|371407.46|1995-03-05|0.00|
/2300070.00|367371.15|1995-03-13|0.00|

```

resultset consists of 10 rows in 4 columns

```

*****
* TPC-H Query 4 0 *
*****
-- Minor modification - date arithmetic for month and years ( 2.2.3.3 c )
select
    o_orderpriority,
    count(*) as order_count
from
    orders
where
    o_orderdate >= date '1993-07-01'
    and o_orderdate < add_months( date '1993-07-01' , 3 )
    and exists (
        select
            *
        from
            lineitem
        where
            l_orderkey = o_orderkey
            and l_commitdate < l_receiptdate
    )
group by
    o_orderpriority
order by
    o_orderpriority;

```

```

/O_ORDERPRIORITY/ORDER_COUNT/
=====
/1-URGENT |10594.00|
/2-HIGH |10476.00|
/3-MEDIUM |10410.00|
/4-NOT SPECIFIED|10556.00|
/5-LOW |10487.00|

```

resultset consists of 5 rows in 2 columns

```

*****
* TPC-H Query 5 0 *
*****
-- Minor modification - date arithmetic for month and years ( 2.2.3.3 c )
select
    n_name,
    sum(l_extendedprice * (1 - l_discount)) as revenue
from
    customer,
    orders,
    lineitem,
    supplier,
    nation,
    region
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and l_suppkey = s_suppkey
    and c_nationkey = s_nationkey
    and s_nationkey = n_nationkey
    and n_regionkey = r_regionkey
    and r_name = 'ASIA'
    and o_orderdate >= date '1994-01-01'
    and o_orderdate < add_months( date '1994-01-01' , 12)
group by
    n_name
order by
    revenue desc;

|N_NAME|REVENUE|
=====
|INDONESIA|55502041.17|
|VIETNAM|55295087.00|
|CHINA|53724494.26|
|INDIA|52035512.00|
|JAPAN|45410175.70|

resultset consists of 5 rows in 2 columns

*****
* TPC-H Query 6 0 *
*****
-- Minor modification - date arithmetic for month and years ( 2.2.3.3 c )
select
    sum(l_extendedprice * l_discount) as revenue
from
    lineitem
where
    l_shipdate >= date '1994-01-01'
    and l_shipdate < add_months( date '1994-01-01' , 12)
    and l_discount between .06 - 0.01 and .06 + 0.01
    and l_quantity < 24;

|REVENUE|
=====
|123141078.23|

resultset consists of 1 rows in 1 columns

*****
* TPC-H Query 7 0 *
*****
select
    supp_nation,
    cust_nation,
    l_year,
    sum(volume) as revenue
from
    (
        select
            n1.n_name as supp_nation,
            n2.n_name as cust_nation,
            extract(year from l_shipdate) as l_year,
            l_extendedprice * (1 - l_discount) as volume
        from
            supplier,
            lineitem,
            orders,
            customer,
            nation n1,
            nation n2
        where
            s_suppkey = l_suppkey
            and o_orderkey = l_orderkey
            and c_custkey = o_custkey
            and s_nationkey = n1.n_nationkey
            and c_nationkey = n2.n_nationkey
            and (
                (n1.n_name = 'FRANCE' and n2.n_name = 'GERMANY')
                or (n1.n_name = 'GERMANY' and n2.n_name = 'FRANCE')
            )
            and l_shipdate between date '1995-01-01' and date '1996-12-31'
    ) as shipping
group by
    supp_nation,
    cust_nation,
    l_year
order by

```

```

    supp_nation,
    cust_nation,
    l_year;

/SUPP_NATION/CUST_NATION/L_YEAR/REVENUE/
=====
/FRANCE                /GERMANY                |1995.00|54639732.73|
/FRANCE                /GERMANY                |1996.00|54633083.31|
/GERMANY               /FRANCE                 |1995.00|52531746.67|
/GERMANY               /FRANCE                 |1996.00|52520549.02|

```

resultset consists of 4 rows in 4 columns

```

*****
* TPC-H Query 8 0 *
*****

```

```

select
    o_year,
    sum(case
        when nation = 'BRAZIL' then volume
        else 0
    end) / sum(volume) as mkt_share
from
    (
        select
            extract(year from o_orderdate) as o_year,
            l_extendedprice * (1 - l_discount) as volume,
            n2.n_name as nation
        from
            part,
            supplier,
            lineitem,
            orders,
            customer,
            nation n1,
            nation n2,
            region
        where
            p_partkey = l_partkey
            and s_suppkey = l_suppkey
            and l_orderkey = o_orderkey
            and o_custkey = c_custkey
            and c_nationkey = n1.n_nationkey
            and n1.n_regionkey = r_regionkey
            and r_name = 'AMERICA'
            and s_nationkey = n2.n_nationkey
            and o_orderdate between date '1995-01-01' and date '1996-12-31'
            and p_type = 'ECONOMY ANODIZED STEEL'
    ) as all_nations
group by
    o_year
order by
    o_year;

```

```

/O_YEAR/MKT_SHARE/
=====
|1995.00|0.03|
|1996.00|0.04|

```

resultset consists of 2 rows in 2 columns

```

*****
* TPC-H Query 9 0 *
*****

```

```

select
    nation,
    o_year,
    sum(amount) as sum_profit
from
    (
        select
            n_name as nation,
            extract(year from o_orderdate) as o_year,
            l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
        from
            part,
            supplier,
            lineitem,
            partsupp,
            orders,
            nation
        where
            s_suppkey = l_suppkey
            and ps_suppkey = l_suppkey
            and ps_partkey = l_partkey
            and p_partkey = l_partkey
            and o_orderkey = l_orderkey
            and s_nationkey = n_nationkey
            and p_name like '%green%'
    ) as profit
group by
    nation,
    o_year
order by
    nation,
    o_year desc;

```

```

/NATION/O_YEAR/SUM_PROFIT/

```



```

=====
/ALGERIA          |1998.00|31342867.23|
/ALGERIA          |1997.00|57138193.02|
/ALGERIA          |1996.00|56140140.13|
/ALGERIA          |1995.00|53051469.65|
/ALGERIA          |1994.00|53867582.13|
/ALGERIA          |1993.00|54942718.13|
----- rows deleted -----
/UNITED STATES    |1992.00|48671944.50|
/VIETNAM          |1998.00|30442736.06|
/VIETNAM          |1997.00|50309179.79|
/VIETNAM          |1996.00|50488161.41|
/VIETNAM          |1995.00|49658284.61|
/VIETNAM          |1994.00|50596057.26|
/VIETNAM          |1993.00|50953919.15|
/VIETNAM          |1992.00|49613838.32|

```

resultset consists of 175 rows in 3 columns

```

*****
* TPC-H Query 10 0 *
*****
-- Minor modification - date arithmetic for month and years ( 2.2.3.3 c )
-- Minor modification - result set limit ( 2.1.2.9.3 )
select
    c_custkey,
    c_name,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    c_acctbal,
    n_name,
    c_address,
    c_phone,
    c_comment
from
    customer,
    orders,
    lineitem,
    nation
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate >= date '1993-10-01'
    and o_orderdate < add_months( date '1993-10-01' , 3)
    and l_returnflag = 'R'
    and c_nationkey = n_nationkey
group by
    c_custkey,
    c_name,
    c_acctbal,
    c_phone,
    n_name,
    c_address,
    c_comment
order by
    revenue desc
LIMIT 20
;

```

```

/C_CUSTKEY/C_NAME/REVENUE/C_ACCTBAL/N_NAME/C_ADDRESS/C_PHONE/C_COMMENT/
=====
/57040.00/Customer#000057040/734235.25/632.87/JAPAN          |EioyZjf4pp|22-895-641-3466|sits. slyly regular
requests sleep alongside of the regular inst|
/143347.00/Customer#000143347/721002.69/2557.47/EGYPT      |laReFYv,Kw4|14-742-935-3718|ggle carefully enticing
requests. final deposits use bold, bold pinto beans. ironic, idle re|
/60838.00/Customer#000060838/679127.31/2454.77/BRAZIL     |64EaJ5vMAHWJlBOxJk1pNc2RjWE|12-913-494-9813| need
to boost against the slyly regular account|
/101998.00/Customer#000101998/637029.57/3790.89/UNITED KINGDOM |01c9CILnNtfoQYmZj|33-593-865-6378|ress foxes wake
slyly after the bold excuses. ironic platelets are furiously carefully bold theodolites|
/125341.00/Customer#000125341/633508.09/4983.51/GERMANY    |S29ODD6bceU8QSuuEJznkNaK|17-582-695-5962|arefully
even depths. blithely even excuses sleep furiously. foxes use except the dependencies. ca|
/25501.00/Customer#000025501/620269.78/7725.04/ETHIOPIA   |W556MXuoiaYCCZamJI,Rn0B4ACUGdkQ8DZ|15-874-808-
6793|he pending instructions wake carefully at the pinto beans. regular, final instructions along the slyly fina|
/115831.00/Customer#000115831/596423.87/5098.10/FRANCE    |rFeBbEYk dl ne7zV5EDrmig1oK09wV7pxqCgIc|16-715-
386-3788|l somas sleep. furiously final deposits wake blithely regular pinto b|
/84223.00/Customer#000084223/594998.02/528.65/UNITED KINGDOM |nAVZCs6BaWap rrm27N 2qBnzc5WBauxbA|33-442-824-8191|
slyly final deposits haggle regular, pending dependencies. pending escapades wake |
/54289.00/Customer#000054289/585603.39/5583.02/IRAN       |vXCxocS0U0Bad5JQI ,oobkZ|20-834-292-4707|ely special
foxes are quickly finally ironic p|
/39922.00/Customer#000039922/584878.11/7321.11/GERMANY    |Zgy4s5012GKN4pLDPBU8m342gIw6R|17-147-757-8036|y
final requests. furiously final foxes cajole blithely special platelets. f|
/6226.00/Customer#00006226/576783.76/2230.09/UNITED KINGDOM |8gPu8,NPGkfyQQ0hcIYUGPIEWc,ybP5g,|33-657-701-
3391|ending platelets along the express deposits cajole carefully final |
/922.00/Customer#00000922/576767.53/3869.25/GERMANY       |Az9RFaut7NkPnc5zSD2PwHgvvr4jRzq|17-945-916-
9648|luffily Fluffy deposits. packages c|
/147946.00/Customer#000147946/576455.13/2030.13/ALGERIA   |iANyZHjqhy7A7ah0pTrYyh7|10-886-956-3143|ithely
ironic deposits haggle blithely ironic requests. quickly regu|
/115640.00/Customer#000115640/569341.19/6436.10/ARGENTINA  |Vtgfia9qI 7EPghecU1X|11-411-543-4901|ost slyly
along the patterns; pinto be|
/73606.00/Customer#000073606/568656.86/1785.67/JAPAN      |xuR0Tro5yChDFOCrjkd2o1|22-437-653-6966|he furiously
regular ideas. slowly|
/110246.00/Customer#000110246/566842.98/7763.35/VIETNAM   |7Kzf1gX MDOq7sOkI|31-943-426-9837|egular deposits
serve blithely above the fl|
/142549.00/Customer#000142549/563537.24/5085.99/INDONESIA   |ChqEoK430ysjdHbtKCP6dKqjNyvv9|19-955-562-
2398|sleep pending courts. ironic deposits against the carefully unusual platelets cajole carefully express accounts.|
/146149.00/Customer#000146149/557254.99/1791.55/ROMANIA    |s87fvzFQpU|29-744-164-6487| of the slyly silent
accounts. quickly final accounts across the |
/52528.00/Customer#000052528/556397.35/551.79/ARGENTINA   |NFztyTOR10UOJ|11-208-192-3205| deposits hinder.
blithely pending asymptotes breach slyly regular re|
/23431.00/Customer#000023431/554269.54/3381.86/ROMANIA    |HgiV0phqhaIa9aydNoI1b|29-915-458-2654|nusual, even
instructions: furiously stealthy n|

```

resultset consists of 20 rows in 8 columns

```
*****
* TPC-H Query 11 0 *
*****
-- Minor modification - Renamed keyword value to value_ ( 2.2.3.3 k )
select
    ps_partkey,
    sum(ps_supplycost * ps_availqty) as value_
from
    partsupp,
    supplier,
    nation
where
    ps_suppkey = s_suppkey
    and s_nationkey = n_nationkey
    and n_name = 'GERMANY'
group by
    ps_partkey having
        sum(ps_supplycost * ps_availqty) > (
            select
                sum(ps_supplycost * ps_availqty) * 0.0001000000
            from
                partsupp,
                supplier,
                nation
            where
                ps_suppkey = s_suppkey
                and s_nationkey = n_nationkey
                and n_name = 'GERMANY'
        )
order by
    value_ desc;
```

```
/PS_PARTKEY|VALUE_|
=====
|129760.00|17538456.86|
|166726.00|16503353.92|
|191287.00|16474801.97|
|161758.00|16101755.54|
|34452.00|15983844.72|
|139035.00|15907078.34|
|9403.00|15451755.62|
|154358.00|15212937.88|
----- rows deleted -----
|122819.00|7888881.02|
|154731.00|7888301.33|
|101674.00|7879324.60|
|51968.00|7879102.21|
|72073.00|787736.11|
|5182.00|7874521.73|
```

resultset consists of 1048 rows in 2 columns

```
*****
* TPC-H Query 12 0 *
*****
-- Minor modification - date arithmetic for month and years ( 2.2.3.3 c )
select
    l_shipmode,
    sum(case
        when o_orderpriority = '1-URGENT'
            or o_orderpriority = '2-HIGH'
        then 1
        else 0
    end) as high_line_count,
    sum(case
        when o_orderpriority <> '1-URGENT'
            and o_orderpriority <> '2-HIGH'
        then 1
        else 0
    end) as low_line_count
from
    orders,
    lineitem
where
    o_orderkey = l_orderkey
    and l_shipmode in ('MAIL', 'SHIP')
    and l_commitdate < l_receiptdate
    and l_shipdate < l_commitdate
    and l_receiptdate >= date '1994-01-01'
    and l_receiptdate < add_months( date '1994-01-01' , 12)
group by
    l_shipmode
order by
    l_shipmode;
```

```
/L_SHIPMODE|HIGH_LINE_COUNT|LOW_LINE_COUNT|
=====
|MAIL      |6202.00|9324.00|
|SHIP     |6200.00|9262.00|
```

resultset consists of 2 rows in 3 columns

```
*****
* TPC-H Query 13 0 *
*****
```

```

-- Minor modification - Outer join syntax ( 2.2.3.3 q )
-- Minor modification - Naming of the columns of the sub select - different syntax for select-list AS clause ( 2.2.3.3 b )
select
    c_count,
    count(*) as custdist
from
    (
        select
            c_custkey,
            count(o_orderkey) c_count
        from
            customer, orders
        where
            c_custkey = o_custkey(+)
            and o_comment(+) not like '%special%requests%'
        group by
            c_custkey
    ) as c_orders
group by
    c_count
order by
    custdist desc,
    c_count desc;

```

```

|C_COUNT|CUSTDIST|
=====
|0.00|50005.00|
|9.00|6641.00|
|10.00|6532.00|
|11.00|6014.00|
|8.00|5937.00|
|12.00|5639.00|
|13.00|5024.00|
|19.00|4793.00|
|7.00|4687.00|
|17.00|4587.00|
|18.00|4529.00|
|20.00|4516.00|
|15.00|4505.00|
|14.00|4446.00|
|16.00|4273.00|
|21.00|4190.00|
|22.00|3623.00|
|6.00|3265.00|
|23.00|3225.00|
|24.00|2742.00|
|25.00|2086.00|
|5.00|1948.00|
|26.00|1612.00|
|27.00|1179.00|
|4.00|1007.00|
|28.00|893.00|
|29.00|593.00|
|3.00|415.00|
|30.00|376.00|
|31.00|226.00|
|32.00|148.00|
|2.00|134.00|
|33.00|75.00|
|34.00|50.00|
|35.00|37.00|
|1.00|17.00|
|36.00|14.00|
|38.00|5.00|
|37.00|5.00|
|40.00|4.00|
|41.00|2.00|
|39.00|1.00|

```

resultset consists of 42 rows in 2 columns

```

*****
* TPC-H Query 14 0 *
*****
-- Minor modification - date arithmetic for month and years ( 2.2.3.3 c )
select
    100.00 * sum(case
        when p_type like 'PROMO%'
            then l_extendedprice * (1 - l_discount)
        else 0
    end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from
    lineitem,
    part
where
    l_partkey = p_partkey
    and l_shipdate >= date '1995-09-01'
    and l_shipdate < add_months( date '1995-09-01', 1);

```

```

|PROMO_REVENUE|
=====
|16.38|

```

resultset consists of 1 rows in 1 columns

```

*****
* TPC-H Query 15 0 *
*****
-- Minor modification - date arithmetic for month and years ( 2.2.3.3 c )
create view revenue0 (supplier_no, total_revenue) as
    select
        l_suppkey,

```

```

        sum(l_extendedprice * (1 - l_discount))
    from
        lineitem
    where
        l_shipdate >= date '1996-01-01'
        and l_shipdate < add_months(date '1996-01-01' , 3)
    group by
        l_suppkey;

select
    s_suppkey,
    s_name,
    s_address,
    s_phone,
    total_revenue
from
    supplier,
    revenue0
where
    s_suppkey = supplier_no
    and total_revenue = (
        select
            max(total_revenue)
        from
            revenue0
    )
order by
    s_suppkey;

/S_SUPPKEY/S_NAME/S_ADDRESS/S_PHONE/TOTAL_REVENUE/
=====
/8449.00/Supplier#00008449      /Wp34zim9qYFbvctdW/20-469-856-8873/1772627.21/

```

resultset consists of 1 rows in 5 columns

drop view revenue0;

 * TPC-H Query 16 0 *

```

select
    p_brand,
    p_type,
    p_size,
    count(distinct ps_suppkey) as supplier_cnt
from
    partsupp,
    part
where
    p_partkey = ps_partkey
    and p_brand <> 'Brand#45'
    and p_type not like 'MEDIUM POLISHED%'
    and p_size in (49, 14, 23, 45, 19, 3, 36, 9)
    and ps_suppkey not in (
        select
            s_suppkey
        from
            supplier
        where
            s_comment like '%Customer%Complaints%'
    )
group by
    p_brand,
    p_type,
    p_size
order by
    supplier_cnt desc,
    p_brand,
    p_type,
    p_size;

```

```

/P_BRAND/P_TYPE/P_SIZE/SUPPLIER_CNT/
=====
/Brand#41 /MEDIUM BRUSHED TIN/3.00/28.00/
/Brand#54 /STANDARD BRUSHED COPPER/14.00/27.00/
/Brand#11 /STANDARD BRUSHED TIN/23.00/24.00/
/Brand#11 /STANDARD BURNISHED BRASS/36.00/24.00/
/Brand#15 /MEDIUM ANODIZED NICKEL/3.00/24.00/
/Brand#15 /SMALL ANODIZED BRASS/45.00/24.00/
/Brand#15 /SMALL BURNISHED NICKEL/19.00/24.00/
----- rows deleted -----
/Brand#51 /SMALL PLATED BRASS/23.00/3.00/
/Brand#52 /MEDIUM BRUSHED BRASS/45.00/3.00/
/Brand#53 /MEDIUM BRUSHED TIN/45.00/3.00/
/Brand#54 /ECONOMY POLISHED BRASS/9.00/3.00/
/Brand#55 /PROMO PLATED BRASS/19.00/3.00/
/Brand#55 /STANDARD PLATED TIN/49.00/3.00/

```

resultset consists of 18314 rows in 4 columns

 * TPC-H Query 17 0 *

```

select
    sum(l_extendedprice) / 7.0 as avg_yearly
from
    lineitem,
    part
where
    p_partkey = l_partkey
    and p_brand = 'Brand#23'
    and p_container = 'MED BOX'
    and l_quantity < (
        select
            0.2 * avg(l_quantity)
        from
            lineitem
    )

```

```

        where      l_partkey = p_partkey
    );

/AVG_YEARLY/
=====
/348406.05/

resultset consists of 1 rows in 1 columns

*****
* TPC-H Query 18 0 *
*****
-- Minor modification - result set limit ( 2.1.2.9.3 )
select
    c_name,
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice,
    sum(l_quantity)
from
    customer,
    orders,
    lineitem
where
    o_orderkey in (
        select  l_orderkey
        from    lineitem
        group by l_orderkey having sum(l_quantity) > 300
    )
    and c_custkey = o_custkey
    and o_orderkey = l_orderkey
group by
    c_name,
    c_custkey,
    o_orderkey,
    o_orderdate,
    o_totalprice
order by
    o_totalprice desc,
    o_orderdate
LIMIT 100
;

/C_NAME/C_CUSTKEY/O_ORDERKEY/O_ORDERDATE/O_TOTALPRICE/SUM(LINEITEM.L_QUANTITY)/
=====
/Customer#000128120/128120.00/4722021.00/1994-04-07/544089.09/323.00/
/Customer#000144617/144617.00/3043270.00/1997-02-12/530604.44/317.00/
/Customer#000013940/13940.00/2232932.00/1997-04-13/522720.61/304.00/
/Customer#000066790/66790.00/2199712.00/1996-09-30/515531.82/327.00/
/Customer#000046435/46435.00/4745607.00/1997-07-03/508047.99/309.00/
/Customer#000015272/15272.00/3883783.00/1993-07-28/500241.33/302.00/
/Customer#000146608/146608.00/3342468.00/1994-06-12/499794.58/303.00/
----- rows deleted -----
/Customer#000149842/149842.00/5156581.00/1994-05-30/411329.35/302.00/
/Customer#000010129/10129.00/5849444.00/1994-03-21/409129.85/309.00/
/Customer#000069904/69904.00/1742403.00/1996-10-19/408513.00/305.00/
/Customer#000017746/17746.00/6882.00/1997-04-09/408446.93/303.00/
/Customer#000013072/13072.00/1481925.00/1998-03-15/399195.47/301.00/
/Customer#000082441/82441.00/857959.00/1994-02-07/382579.74/305.00/
/Customer#000088703/88703.00/2995076.00/1994-01-30/363812.12/302.00/

resultset consists of 57 rows in 6 columns

*****
* TPC-H Query 19 0 *
*****
select
    sum(l_extendedprice* (1 - l_discount)) as revenue
from
    lineitem,
    part
where
    (
        p_partkey = l_partkey
        and p_brand = 'Brand#12'
        and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        and l_quantity >= 1 and l_quantity <= 1 + 10
        and p_size between 1 and 5
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    )
    or
    (
        p_partkey = l_partkey
        and p_brand = 'Brand#23'
        and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        and l_quantity >= 10 and l_quantity <= 10 + 10
        and p_size between 1 and 10
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
    )
    or
    (
        p_partkey = l_partkey
        and p_brand = 'Brand#34'
        and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        and l_quantity >= 20 and l_quantity <= 20 + 10
        and p_size between 1 and 15
        and l_shipmode in ('AIR', 'AIR REG')
    )

```

```

        and l_shipinstruct = 'DELIVER IN PERSON'
    );

/REVENUE/
=====
/3083843.06/

resultset consists of 1 rows in 1 columns

*****
* TPC-H Query 20 0 *
*****
-- Minor modification - date arithmetic for month and years ( 2.2.3.3 c )
select
    s_name,
    s_address
from
    supplier,
    nation
where
    s_suppkey in (
        select
            ps_suppkey
        from
            partsupp
        where
            ps_partkey in (
                select
                    p_partkey
                from
                    part
                where
                    p_name like 'forest%'
            )
            and ps_availqty > (
                select
                    0.5 * sum(l_quantity)
                from
                    lineitem
                where
                    l_partkey = ps_partkey
                    and l_suppkey = ps_suppkey
                    and l_shipdate >= date '1994-01-01'
                    and l_shipdate < add_months( date '1994-01-01' , 12)
            )
    )
    and s_nationkey = n_nationkey
    and n_name = 'CANADA'
order by
    s_name;

```

```

/S_NAME/S_ADDRESS/
=====
/Supplier#000000020      /iybAE,RmTymrZVYafZva2SH,j/
/Supplier#000000091      /YV45D7TkfdQan0OZ7q9QxkyGUapU1oOWU6q3/
/Supplier#000000197      /YC2Acon6kjY3zj3Fbxs2k4VdF7X0cd2F/
/Supplier#000000226      /83qOdU2EYRdPQAqHEtn GRZEd/
/Supplier#000000285      /Br7eInntlyxrw6ImgPJ7YdhFDjuBf/
/Supplier#000000378      /FfbhyCwVvcPrO8ltp9/
/Supplier#000000402      /i9Sw4DoyMhzhKXCH9By,AYSgmD/
----- rows deleted -----
/Supplier#000009862      /rJzweWeN58/
/Supplier#000009868      /ROjGgx5gvtkmmUuoeyy7v/
/Supplier#000009869      /ucLqzrpbTRMewGSM29t0rNTM30g1Tu3Xgg3mKag/
/Supplier#000009899      /7XdpAhrzr1t,UQFZE/
/Supplier#000009974      /7wJ,J5DKcxSU4Kp1cQLpbcAvB5AsvKT/

```

resultset consists of 204 rows in 2 columns

```

*****
* TPC-H Query 21 0 *
*****
-- Minor modification - result set limit ( 2.1.2.9.3 )
select
    s_name,
    count(*) as numwait
from
    supplier,
    lineitem l1,
    orders,
    nation
where
    s_suppkey = l1.l_suppkey
    and o_orderkey = l1.l_orderkey
    and o_orderstatus = 'F'
    and l1.l_receiptdate > l1.l_commitdate
    and exists (
        select
            *
        from
            lineitem l2
        where
            l2.l_orderkey = l1.l_orderkey
            and l2.l_suppkey <> l1.l_suppkey
    )
    and not exists (
        select
            *
        from
            lineitem l3
        where
            l3.l_orderkey = l1.l_orderkey
            and l3.l_suppkey <> l1.l_suppkey
            and l3.l_receiptdate > l3.l_commitdate
    )
    and s_nationkey = n_nationkey
    and n_name = 'SAUDI ARABIA'
group by
    s_name
order by
    numwait desc,
    s_name
LIMIT 100
;

```

```

/S_NAME/NUMWAIT/
=====
/Supplier#000002829      |20.00/
/Supplier#000005808      |18.00/
/Supplier#000000262      |17.00/
/Supplier#000000496      |17.00/
/Supplier#000002160      |17.00/
/Supplier#000002301      |17.00/
/Supplier#000002540      |17.00/
----- rows deleted -----
/Supplier#000001337      |12.00/
/Supplier#000001916      |12.00/
/Supplier#000001925      |12.00/
/Supplier#000002039      |12.00/
/Supplier#000002357      |12.00/
/Supplier#000002483      |12.00/

```

resultset consists of 100 rows in 2 columns

```

*****
* TPC-H Query 22 0 *
*****

```

-- Minor modification - use of substring instead of substr (2.2.3.3 p)

```

select
  centrycode,
  count(*) as numcust,
  sum(c_acctbal) as totacctbal
from
  (
    Select  substring(c_phone from 1 for 2) as centrycode,
           c_acctbal
    from    customer
    where   substring(c_phone from 1 for 2) in
           ('13', '31', '23', '29', '30', '18', '17')
    and c_acctbal > (
      select  avg(c_acctbal)
      from    customer
      where   c_acctbal > 0.00
      and substring(c_phone from 1 for 2) in
           ('13', '31', '23', '29', '30', '18', '17')
    )
    and not exists (
      select  *
      from    orders
      where   o_custkey = c_custkey
    )
  ) as custsale
group by centrycode
order by centrycode;

```

```

/CNTRYCODE/NUMCUST/TOTACCTBAL/
=====
/13|888.00|6737713.99/
/17|861.00|6460573.72/
/18|964.00|7236687.40/
/23|892.00|6701457.95/
/29|948.00|7158866.63/
/30|909.00|6808436.13/
/31|922.00|6806670.18/

```

resultset consists of 7 rows in 3 columns

Appendix E: Seed and Input Parameters

```

#-----
# Stream 0 - Seed = 1111194900
#-----
14 1994-01-01
2 22 NICKEL AFRICA
9 azure
20 orange 1995-01-01 ROMANIA
6 1996-01-01 0.03 25
17 Brand#24 MED BOX
18 314
8 CHINA ASIA STANDARD BURNISHED TIN
21 IRAN
13 express packages
3 FURNITURE 1995-03-04
22 16 20 23 19 13
18 28
16 Brand#42 ECONOMY ANODIZED 22 23
49 21 48 15 7
41
4 1993-07-01
11 MOROCCO 0.0000001000
15 1994-01-01
1 80
10 1994-05-01
19 Brand#34 Brand#34 Brand#42 4 16
30
5 AFRICA 1996-01-01
7 EGYPT CHINA
12 MAIL SHIP 1994-01-01

#-----
# Stream 1 - Seed = 1111194901
#-----
21 CANADA
3 MACHINERY 1995-03-20
18 312
5 AMERICA 1996-01-01
11 CANADA 0.0000001000
7 VIETNAM IRAN
6 1996-01-01 0.08 24
20 bisque 1993-01-01 INDIA
17 Brand#21 MED PACK
12 TRUCK SHIP 1994-01-01
16 Brand#22 STANDARD PLATED 23 22
28 6 37 36 43
49
15 1997-01-01
13 express requests
10 1993-02-01
2 10 TIN EUROPE
8 IRAN MIDDLE EAST PROMO BRUSHED TIN
14 1994-01-01
19 Brand#31 Brand#12 Brand#41 9 17
26
9 wheat
22 29 34 10 16 32
24 19
1 88
4 1996-02-01

#-----
# Stream 2 - Seed = 1111194902
#-----
6 1996-01-01 0.06 24
17 Brand#23 MED CAN
14 1994-01-01
16 Brand#12 MEDIUM POLISHED 7 27
47 1 49 16 39
19 Brand#33 Brand#55 Brand#35 4 18
22
10 1993-11-01
9 steel
2 48 COPPER AFRICA
15 1995-01-01
8 BRAZIL AMERICA PROMO PLATED TIN
5 ASIA 1996-01-01
22 13 24 34 19 29
17 23
12 RAIL SHIP 1994-01-01
7 JORDAN BRAZIL
13 express requests
18 313
1 96
4 1993-11-01
20 lemon 1997-01-01 UNITED KINGDOM
3 BUILDING 1995-03-06
11 MOZAMBIQUE 0.0000001000
21 SAUDI ARABIA

```

```

#-----
# Stream 3 - Seed = 1111194903
#-----
8 ROMANIA EUROPE PROMO ANODIZED TIN
5 MIDDLE EAST 1996-01-01
4 1996-06-01
6 1996-01-01 0.03 25
17 Brand#25 JUMBO BOX
7 ETHIOPIA ROMANIA
1 104
18 315
22 23 17 30 32 31
13 34
14 1994-01-01
9 sienna
10 1994-09-01
15 1997-01-01
11 EGYPT 0.0000001000
20 spring 1995-01-01 JORDAN
2 36 STEEL EUROPE
21 JAPAN
19 Brand#45 Brand#33 Brand#35 9 19
29
13 special requests
16 Brand#42 PROMO ANODIZED 32 45
25 41 43 9 2
44
12 AIR REG AIR 1994-01-01
3 MACHINERY 1995-03-22

#-----
# Stream 4 - Seed = 1111194904
#-----
5 AFRICA 1996-01-01
21 EGYPT
14 1995-01-01
19 Brand#42 Brand#11 Brand#34 5 20
25
15 1995-01-01
17 Brand#22 JUMBO PACK
12 REG AIR RAIL 1995-01-01
6 1996-01-01 0.09 24
4 1994-03-01
9 rosy
8 IRAQ MIDDLE EAST ECONOMY POLISHED
NICKEL
16 Brand#32 SMALL BURNISHED 35 16
47 2 42 32 29
46
11 PERU 0.0000001000
2 23 NICKEL AMERICA
10 1993-06-01
18 312
1 112
13 special requests
7 RUSSIA IRAQ
22 12 22 16 32 33
23 17
3 BUILDING 1995-03-08
20 forest 1993-01-01 BRAZIL

#-----
# Stream 5 - Seed = 1111194905
#-----
21 VIETNAM
15 1993-01-01
4 1996-10-01
6 1997-01-01 0.06 24
7 KENYA CANADA
16 Brand#12 LARGE POLISHED 22 37
21 14 43 26 27
44
19 Brand#44 Brand#54 Brand#23 10 10
22
18 314
14 1995-01-01
22 23 25 22 33 15
19 30
11 ETHIOPIA 0.0000001000
13 special requests
3 HOUSEHOLD 1995-03-24
1 120
2 11 TIN EUROPE
5 AMERICA 1997-01-01
8 CANADA AMERICA ECONOMY BURNISHED NICKEL
20 powder 1997-01-01 PERU
12 FOB REG AIR 1995-01-01
17 Brand#24 JUMBO CAN
10 1994-03-01
9 plum

```



```

#-----
# Stream 6 - Seed = 1111194906
#-----
10 1994-12-01
3 BUILDING 1995-03-10
15 1995-01-01
13 special accounts
6 1997-01-01 0.03 25
8 SAUDI ARABIA MIDDLE EAST LARGE
BRUSHED NICKEL
9 orchid
7 FRANCE SAUDI ARABIA
4 1994-07-01
11 CHINA 0.0000001000
22 17 22 20 14 21
24 28
18 315
12 MAIL REG AIR 1995-01-01
1 68
5 ASIA 1997-01-01
16 Brand#42 PROMO BRUSHED 31 23
40 30 4 20 24 2
2 49 COPPER AMERICA
14 1995-01-01
19 Brand#51 Brand#32 Brand#22 5 11
29
20 chartreuse 1995-01-01 GERMANY
17 Brand#25 WRAP BOX
21 JORDAN

```

```

#-----
# Stream 7 - Seed = 1111194907
#-----
18 313
8 JAPAN ASIA LARGE PLATED NICKEL
20 midnight 1994-01-01 RUSSIA
21 FRANCE
2 37 STEEL MIDDLE EAST
4 1997-02-01
22 13 15 29 32 12
22 27
17 Brand#22 WRAP PACK
1 76
11 FRANCE 0.0000001000
9 misty
19 Brand#53 Brand#25 Brand#22 1 12
25
3 HOUSEHOLD 1995-03-26
13 special accounts
5 EUROPE 1997-01-01
7 UNITED KINGDOM JAPAN
10 1993-09-01
16 Brand#32 MEDIUM BURNISHED 7 29
13 45 37 39 46
17
6 1997-01-01 0.09 25
14 1995-01-01
15 1993-01-01
12 TRUCK REG AIR 1995-01-01

```

Appendix F: Benchmark programs and Scripts

```
-----
#-----
# tpc_h_run_full_parallel.sh (Main-Script)
#-----
-----

#!/bin/bash
# param <dw environment> <roothost> <scalefactor> <streams> <instances per node> <refresh
functions while queries> <logdir>
echo $*
if [ $# -ne 7 ]
then
echo "Usage: $0 <dw environment> <roothost> <scalefactor> <streams> <instances per node>
<refresh functions while queries> <logdir>"
echo "<refresh functions while queries> may be yes or no"
exit 1
fi

ENVIRONMENT="$1"
ROOTHOST="$2"
SF="$3"
STREAMS="$4"
INST_PN="$5"
RF_WHILE_Q="$6"
LOGDIR="$7/"`date +%Y%m%d%H%M%S``"
LOGFILE="$LOGDIR/test.log"
ROOTPORT=26822

export TPC_BASEDIR=/shares/dwstaging/benchmark
export TPC_SCRIPTSDIR=$TPC_BASEDIR/scripts
export TPC_SHAREDIR=$TPC_BASEDIR/shares
export TPC_SQLDIR=$TPC_BASEDIR/sql
export DSS_CONFIG=$TPC_BASEDIR/tpch_20070105

### Startup of database
mkdir -p $LOGDIR
mkdir -p $LOGDIR/run1
mkdir -p $LOGDIR/run2
echo "TPC Benchmark H started" >>$LOGFILE
echo "-----" >>$LOGFILE
echo "SCRIPT : $0" >> $LOGFILE
echo "ROOTHOST: $ROOTHOST" >>$LOGFILE
echo "SCALE : $SF" >>$LOGFILE
echo "STREAMS : $STREAMS" >> $LOGFILE
echo "INSTPN : $INST_PN" >>$LOGFILE
echo "LOGDIR : $LOGDIR" >>$LOGFILE
echo "SCRIPTS : $TPC_SCRIPTSDIR" >>$LOGFILE
echo "-----" >>$LOGFILE

echo >> $LOGFILE
echo "Start of Test at; `date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\(...\))...../1/g'`"
>>$LOGFILE

##### Phase I (Load-Test)

### Initialization of database-schema and user

dw $ENVIRONMENT create
dwaplus -u sys -P exasol -h $ROOTHOST -p 36822 -f
/shares/dwstaging/benchmark/sql/create_user.sql
```

```
dw $ENVIRONMENT shutdown
dw $ENVIRONMENT startup

TIMESFILE=$LOGDIR/run1/alltimes.log
LOAD_START="`date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\(...\))...../1/g'`"
echo -n "Load Test; $LOAD_START " > $TIMESFILE
>>$LOGFILE
dwaplus -p $ROOTPORT -h $ROOTHOST -u tpcuser -P tpcuser -f $TPC_SQLDIR/create_schema.sql
>>$LOGDIR/create_schema.log 2>&1

### Load

$TPC_SCRIPTSDIR/load_init_parallel.sh $ENVIRONMENT tpcuser tpcuser tpc $SF $INST_PN $LOGDIR
>>$LOGFILE 2>&1
sync

### Initialization of indices
echo "Start of database optimization at ; `date +%m/%d/%Y %k:%M:%S.%N` | sed
's/(\(...\))...../1/g'`" >> $LOGFILE
optimizeLineitem()
{
cat $TPC_SQLDIR/create_indices.sql | grep -i -v -e NATION -e REGION -e " PART" -e ORDERS -e
PARTSUPP -e CUSTOMER -e SUPPLIER -e ANALYZE | dwaplus -p $ROOTPORT -h $ROOTHOST -u tpcuser -P
tpcuser 1>>$LOGDIR/optimizeLineitem.log 2>&1
}

optimizeOther()
{
cat $TPC_SQLDIR/create_indices.sql | grep -i -v -e LINEITEM -e ANALYZE | dwaplus -p
$ROOTPORT -h $ROOTHOST -u tpcuser -P tpcuser 1>>$LOGFILE 2>&1
}

optimizeLineitem &
optimizeOther &
wait

cat $LOGDIR/optimizeLineitem.log >> $LOGFILE
rm -f $LOGDIR/optimizeLineitem.log

cat $TPC_SQLDIR/create_indices.sql | grep -i -v -e ENFORCE -e OPTIMIZE -e SELECT | dwaplus -p
$ROOTPORT -h $ROOTHOST -u tpcuser -P tpcuser 1>>$LOGFILE 2>&1

echo "End of database optimization at ; `date +%m/%d/%Y %k:%M:%S.%N` | sed
's/(\(...\))...../1/g'`" >> $LOGFILE
sync

LOAD_END="`date +%m/%d/%Y %k:%M:%S.%N` | sed 's/(\(...\))...../1/g'`"
echo "End of Load test at ; $LOAD_END" >> $LOGFILE
echo -n "; $LOAD_END " >> $TIMESFILE
LOAD_TIME="``$TPC_SCRIPTSDIR/timediff "$LOAD_START" "$LOAD_END"``"
echo "; $LOAD_TIME" >> $TIMESFILE
sync

# execute tables.sql
cat $TPC_SQLDIR/tables.sql | dwaplus -p $ROOTPORT -h $ROOTHOST -u tpcuser -P tpcuser
>>$LOGDIR/tables.log 2>&1

### Create queries for the query-streams

echo "Generating substitution parameters for performance test" >> $LOGFILE
$TPC_SCRIPTSDIR/make_queries.sh $LOGDIR/run1 $SF $STREAMS "$LOAD_END" >> $LOGFILE
cp $LOGDIR/run1/stream*.sql $LOGDIR/run2
```

```

echo "Phase I finished at ; " `date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
>>$LOGFILE
sync

##### Phase II (Power-Test 1)

U=1

echo "Start of Phase II at ; " `date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
>>$LOGFILE

START=`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
echo -n "RF1 for stream/pair ; 0 ; $U ; $START " >> $TIMESFILE
echo "Start of RF1 for stream/pair ; 0 ; $U ; $START" >> $LOGFILE
$TPC_SCRIPTSDIR/load_rfl.sh $U $ENVIRONMENT tpcuser tpcuser tpc $$F $INST_PN $LOGDIR >>$LOGFILE
2>&1
END=`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
echo "End of RF1 for stream/pair ; 0 ; $U ; $END" >> $LOGFILE
echo -n "; $END " >> $TIMESFILE
TIMEUSED=""`$TPC_SCRIPTSDIR/timediff "$START" "$END"``"
echo "; $TIMEUSED" >> $TIMESFILE

echo "Start of querystream 0 at ; " `date +%m/%d/%Y %k:%M:%S.%N' | sed
's/(\(.....\))...../1/g'` >> $LOGFILE
$TPC_SCRIPTSDIR/query_streams/querystreamex $LOGDIR/run1/stream0.sql $LOGDIR/run1/stream0.log
$LOGDIR/run1/stream0.log2 0 $ROOTHOST $ROOTPORT tpcuser tpcuser >>$LOGFILE 2>&1
echo "End of querystream 0 at ; " `date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
>> $LOGFILE

START=`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
echo -n "RF2 for stream/pair ; 0 ; $U ; $START " >> $TIMESFILE
echo "Start of RF2 for stream/pair ; 0 ; $U ; $START" >>$LOGFILE
$TPC_SCRIPTSDIR/query_streams/deletestreamex $TPC_SHAREDIR/Database_SF$$SF/delete.1 1 tpc
$ROOTHOST $ROOTPORT tpcuser tpcuser >>$LOGFILE 2>&1
END=`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
echo "End of RF2 for stream/pair ; 0 ; $U ; $END" >>$LOGFILE
echo -n "; $END " >> $TIMESFILE
TIMEUSED=""`$TPC_SCRIPTSDIR/timediff "$START" "$END"``"
echo "; $TIMEUSED" >> $TIMESFILE

echo "End of Phase II at ; " `date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
>>$LOGFILE
sync

##### Phase III (Throughput-Test 1)

TP_START=`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
echo "Start of Phase III at ; $TP_START" >>$LOGFILE

do
for ((S=1;S<=$STREAMS+1;S++))
do
echo "Start of querystream $$S at ; " `date +%m/%d/%Y %k:%M:%S.%N' | sed
's/(\(.....\))...../1/g'` >> $LOGFILE
($TPC_SCRIPTSDIR/query_streams/querystreamex $LOGDIR/run1/stream$$S.sql
$LOGDIR/run1/stream$$S.log $LOGDIR/run1/stream$$S.log2 $$S $ROOTHOST $ROOTPORT tpcuser tpcuser
>>$LOGFILE 2>&1 ; echo "End of querystream $$S at ; " `date +%m/%d/%Y %k:%M:%S.%N' | sed
's/(\(.....\))...../1/g'` >>$LOGFILE) &
done
if [ "$RF_WHILE_Q" != yes ] ; then
wait
fi
STREAM_U=1
for ((U=2;U<=$STREAMS+1;U++))

```

```

do
START=`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
echo -n "RF1 for stream/pair ; $STREAM_U ; $U ; $START " >> $TIMESFILE
echo "Start of RF1 for stream/pair ; $STREAM_U ; $U ; $START" >> $LOGFILE
$TPC_SCRIPTSDIR/load_rfl.sh $U $ENVIRONMENT tpcuser tpcuser tpc $$F $INST_PN $LOGDIR
>>$LOGFILE 2>&1
END=`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
echo "End of RF1 for stream/pair ; $STREAM_U ; $U ; $END" >>$LOGFILE
echo -n "; $END " >> $TIMESFILE
TIMEUSED=""`$TPC_SCRIPTSDIR/timediff "$START" "$END"``"
echo "; $TIMEUSED" >> $TIMESFILE

START=`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
echo -n "RF2 for stream/pair ; $STREAM_U ; $U ; $START " >> $TIMESFILE
echo "Start of RF2 for stream/pair ; $STREAM_U ; $U ; $START" >> $LOGFILE
$TPC_SCRIPTSDIR/query_streams/deletestreamex $TPC_SHAREDIR/Database_SF$$SF/delete.$U
$U tpc $ROOTHOST $ROOTPORT tpcuser tpcuser >>$LOGFILE 2>&1
END=`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
echo "End of RF2 for stream/pair ; $STREAM_U ; $U ; $END" >>$LOGFILE
echo -n "; $END " >> $TIMESFILE
TIMEUSED=""`$TPC_SCRIPTSDIR/timediff "$START" "$END"``"
echo "; $TIMEUSED" >> $TIMESFILE

STREAM_U=$((STREAM_U+1))

done
if [ "$RF_WHILE_Q" = yes ] ; then
wait
fi

TP_END=`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
TPTIME=""`$TPC_SCRIPTSDIR/timediff "$TP_START" "$TP_END"``"
echo "Throughput Test ; $TP_START ; $TP_END ; $TPTIME" >> $TIMESFILE
echo "End of Phase III at ; $TP_END" >>$LOGFILE
sync

##### Phase IV (Power-Test 2)

echo "Start of Phase IV at ; " `date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
>>$LOGFILE
TIMESFILE=$LOGDIR/run2/alltimes.log

((U=$STREAMS+2))
START=`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
echo -n "RF1 for stream/pair ; 0 ; $U ; $START " >> $TIMESFILE
echo "Start of RF1 for stream/pair ; 0 ; $U ; $START" >> $LOGFILE
$TPC_SCRIPTSDIR/load_rfl.sh $U $ENVIRONMENT tpcuser tpcuser tpc $$F $INST_PN $LOGDIR >>$LOGFILE
2>&1
END=`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
echo "End of RF1 for stream/pair ; 0 ; $U ; $END" >> $LOGFILE
echo -n "; $END " >> $TIMESFILE
TIMEUSED=""`$TPC_SCRIPTSDIR/timediff "$START" "$END"``"
echo "; $TIMEUSED" >> $TIMESFILE

echo "Start of querystream 0 at ; " `date +%m/%d/%Y %k:%M:%S.%N' | sed
's/(\(.....\))...../1/g'` >> $LOGFILE
$TPC_SCRIPTSDIR/query_streams/querystreamex $LOGDIR/run2/stream0.sql $LOGDIR/run2/stream0.log
$LOGDIR/run2/stream0.log2 0 $ROOTHOST $ROOTPORT tpcuser tpcuser >>$LOGFILE 2>&1
echo "End of querystream 0 at ; " `date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
>> $LOGFILE

START=`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(.....\))...../1/g'`
echo -n "RF2 for stream/pair ; 0 ; $U ; $START " >> $TIMESFILE
echo "Start of RF2 for stream/pair ; 0 ; $U ; $START" >>$LOGFILE

```

```

$TPC_SCRIPTSDIR/query_streams/deletestreamex $TPC_SHAREDIRE/Database_SFSSF/delete.$U $U tpc
$ROOTHOST $ROOTPORT tpcuser tpcuser >>$LOGFILE 2>&l
END="`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(....\))...../1/g'`"
echo "End of RF2 for stream/pair ; 0 ; $U ; $END" >>$LOGFILE
echo -n " ; $END " >> $TIMESFILE
TIMEUSED="``$TPC_SCRIPTSDIR/timediff "$START" "$END"``"
echo " ; $TIMEUSED" >> $TIMESFILE

echo "End of Phase IV at ; " `date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(....\))...../1/g'`
>>$LOGFILE
sync

##### Phase V (Throughput-Test 2)

TP_START="`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(....\))...../1/g'`"
echo "Start of Phase V at ; $TP_START" >>$LOGFILE

for ((S=1;S<$STREAMS+1;S++))
do
    echo "Start of querystream $S at ; `date +%m/%d/%Y %k:%M:%S.%N' | sed
's/(\(....\))...../1/g'`" >> $LOGFILE
    ($TPC_SCRIPTSDIR/query_streams/querystreamex $LOGDIR/run2/stream$S.sql
$LOGDIR/run2/stream$S.log $LOGDIR/run2/stream$S.log2 $S $ROOTHOST $ROOTPORT tpcuser tpcuser
>>$LOGFILE 2>&l ; echo "End of querystream $S at ; `date +%m/%d/%Y %k:%M:%S.%N' | sed
's/(\(....\))...../1/g'`" >>$LOGFILE) &
done
if [ "$RF_WHILE_Q" != yes ] ; then
    wait
fi
STREAM_U=1
for ((U=$STREAMS+3;U<=$STREAMS+$STREAMS+2;U++))
do
    START="`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(....\))...../1/g'`"
    echo -n "RF1 for stream/pair ; $STREAM_U ; $U ; $START " >> $TIMESFILE
    echo "Start of RF1 for stream/pair ; $STREAM_U ; $U ; $START" >> $LOGFILE
    $TPC_SCRIPTSDIR/load_rf1.sh $U $ENVIRONMENT tpcuser tpcuser tpc $SF $INST_PN $LOGDIR
>>$LOGFILE 2>&l
    END="`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(....\))...../1/g'`"
    echo "End of RF1 for stream/pair ; $STREAM_U ; $U ; $END" >>$LOGFILE
    echo -n " ; $END " >> $TIMESFILE
    TIMEUSED="``$TPC_SCRIPTSDIR/timediff "$START" "$END"``"
    echo " ; $TIMEUSED" >> $TIMESFILE

    START="`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(....\))...../1/g'`"
    echo -n "RF2 for stream/pair ; $STREAM_U ; $U ; $START " >> $TIMESFILE
    echo "Start of RF2 for stream/pair ; $STREAM_U ; $U ; $START" >> $LOGFILE
    $TPC_SCRIPTSDIR/query_streams/deletestreamex $TPC_SHAREDIRE/Database_SFSSF/delete.$U
$U tpc $ROOTHOST $ROOTPORT tpcuser tpcuser >>$LOGFILE 2>&l
    END="`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(....\))...../1/g'`"
    echo "End of RF2 for stream/pair ; $STREAM_U ; $U ; $END" >>$LOGFILE
    echo -n " ; $END " >> $TIMESFILE
    TIMEUSED="``$TPC_SCRIPTSDIR/timediff "$START" "$END"``"
    echo " ; $TIMEUSED" >> $TIMESFILE

    STREAM_U=$((STREAM_U+1))
done
if [ "$RF_WHILE_Q" = yes ] ; then
    wait
fi

TP_END="`date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(....\))...../1/g'`"
TPTIME="``$TPC_SCRIPTSDIR/timediff "$TP_START" "$TP_END"``"
echo "Throughput Test ; $TP_START ; $TP_END ; $TPTIME" >> $TIMESFILE

```

```

echo "End of Phase V at ; $TP_END" >>$LOGFILE
sync

### Extract and format important times for presentation
$TPC_SCRIPTSDIR/calculate_results.sh $LOGDIR/run1 $STREAMS $TPC_SCRIPTSDIR
>$LOGDIR/run1/result.csv
$TPC_SCRIPTSDIR/calculate_results.sh $LOGDIR/run2 $STREAMS $TPC_SCRIPTSDIR
>$LOGDIR/run2/result.csv

# execute dbcheck.sql
cat $TPC_SQLDIR/dbcheck.sql | dwaplus -p $ROOTPORT -h $ROOTHOST -u tpcuser -P tpcuser
>>$LOGDIR/dbcheck.log 2>&l

#dw $ENVIRONMENT shutdown

echo "End of Test at ; " `date +%m/%d/%Y %k:%M:%S.%N' | sed 's/(\(....\))...../1/g'`
>>$LOGFILE

# -----
# make_queries.sh
# -----

#!/bin/bash
# params: <logdir> <scalefactor> <streamsTP> <endtime of load>
LOGDIR=$1
SF=$2
STREAMS=$3
LOADEND="$4"

RANDOM=`echo "$LOADEND" | sed 's%(\.)/(\.)/... \(\.):\(\.):\(\.)\...%12\3\4\5%' |
sed 's%0%'`
#RANDOM=`date +%m-d%H%M%S`

export DSS_CONFIG=/shares/dwstaging/benchmark/tpch_20070105
cd $DSS_CONFIG
for ((U=0;U<=$STREAMS;U++))
do
    echo "QGen seed; stream $U; $RANDOM"
    $DSS_CONFIG/qgen -s $SF -c -i $TPC_SQLDIR/initstream.sql -p $U -r $RANDOM >
$LOGDIR/stream$U.sql
dos2unix $LOGDIR/stream$U.sql
((RANDOM=RANDOM+1))
done

# -----
# initstream.sql
# -----
OPEN SCHEMA tpc;
COMMIT;

# -----
# load_rf1.sh (RF1)
# -----

#!/bin/bash
if [ $# -ne 8 ] ; then
    echo "Syntax: $0 <num> <dw> <user> <password> <schema> <SF> <instances-per-node>
<logdir>"
    exit 1
fi

```

```

num="$1"
DW_ENV="$2"
user="$3"
pass="$4"
schema="$5"
SF="$6"
INST_PN="$7"
LOGDIR="$8"
export TPC_SHAREDIR=/shares/dwstaging/benchmark/shares

DW_STATE="`dw $DW_ENV show state`" || {
    echo "DW with environment $DW_ENV does not work." >&2
    exit 1
}

DW_CONN="`echo $DW_STATE | sed 's/^.*host //; s/^\([^ ]*\)$ at live port /\1:/'`"
DW_NODES="`dw $DW_ENV show nodes | wc -l`"

TPC_LOCALDIR="/shares/dwstaging/local/benchmark/TPC_Database_SF${SF}_${INST_PN}x_${DW_NODES}N"

ORDERS_COLUMNS="O_ORDERKEY:DECIMAL(11)|O_CUSTKEY:DECIMAL(11)|O_ORDERSTATUS:CHAR(1)|O_TOTALPRICE:DECIMAL(12,2)|O_ORDERDATE:DATE|O_ORDERPRIORITY:CHAR(15)|O_CLERK:CHAR(15)|O_SHIPPRIORITY:DECIMAL(10)|O_COMMENT:VARCHAR(79)"
LINEITEM_COLUMNS="L_ORDERKEY:DECIMAL(11)|L_PARTKEY:DECIMAL(11)|L_SUPPKEY:DECIMAL(11)|L_LINENUMBER:DECIMAL(10)|L_QUANTITY:DECIMAL(12,2)|L_EXTENDEDPRICE:DECIMAL(12,2)|L_DISCOUNT:DECIMAL(12,2)|L_TAX:DECIMAL(12,2)|L_RETURNFLAG:CHAR(1)|L_LINestatus:CHAR(1)|L_SHIPDATE:DATE|L_COMMITDATE:DATE|L_RECEIPTDATE:DATE|L_SHIPINSTRUCT:CHAR(25)|L_SHIPMODE:CHAR(10)|L_COMMENT:VARCHAR(44)"

( dw "$DW_ENV" loader -p -q -l csv2dwa \
  -s $TPC_LOCALDIR/metafile.RF1.csv.u$num \
  -d $user:$schema:$pass@$DW_CONN -a \
  -t ORDERS -c " `eval echo \\${ORDERS_COLUMNS}` " \
  -t LINEITEM -c " `eval echo \\${LINEITEM_COLUMNS}` " \
  | tr -d '\n' | sed -e 's/rows loaded\\.//g' | sed -e 's/^\([0-9]*\) \([0-9]*\) \$/ORDERS rows loaded; \1\nLINEITEM rows loaded; \2\n/' \
  ) || { echo "RF1 loading failed (step 1 of 1)." ; exit 1 ; }
exit 0

# -----
# deletestream.cpp (RF2)
# -----

#include "querystream_for_delete.h"

#include <iostream>
using std::cout;
using std::cerr;
using std::endl;

#include <fstream>
using std::ofstream;
using std::ifstream;

#include <string>
using std::string;

#include <sstream>
using std::ostringstream;

#include <sys/time.h>

/** \brief Calculates the time passed between t1 and t2. */

```

```

double timediff( timeval t1, timeval t2 )
{
    double passed = 1.0e-6 * ( t2.tv_usec - t1.tv_usec );
    passed += ( t2.tv_sec - t1.tv_sec );
    if ( passed < 0 )
        passed = -passed;
    return passed;
}

/** \brief Deletes from a given string all occurrences of another string at beginning and end.
*/
string trim( string &s, const string drop = " " )
{
    string r = s.erase( s.find_last_not_of(drop)+1 );
    return r.erase( 0, r.find_first_not_of(drop) );
}

const int KEY_COUNT_PER_STEP = 1500000;

int main( int argc, char** argv )
{
    timeval stream_starttime;
    timeval stream_endtime;
    timeval starttime;
    timeval endtime;

    bool firsttime = true;
    if ( argc!=8 && argc!=9 )
    {
        cout << "usage: " << argv[0] << " <inputfile> <streamid> <schema> <host> <port> <user> <password> [<logfile>]" << endl;
        cout << endl;
        cout << "\tinputfile name of the file containing the queries to execute" << endl;
        cout << "\tstreamid an identifier used for this stream" << endl;
        cout << "\thost name of host to connect to" << endl;
        cout << "\tport port number on host to connect to" << endl;
        cout << "\tschema name of a schema to open prior to any statements in inputfile (default: open no schema)" << endl;
        cout << "\tuser name of user to connect to DWA (default: sys)" << endl;
        cout << "\tpassword password used to authenticate user (default: exasol)" << endl;
        return 1;
    }
    Configuration config;
    config.host = argv[4];
    config.port = argv[5];
    config.user = argv[6];
    config.password = argv[7];

    std::string initialSchema = argv[3];
    std::string myStreamID = argv[2];

    bool hasLog = false;
    ofstream logfile;

    if ( argc == 9 )
    {
        gettimeofday(&stream_starttime, 0);

        logfile.open( argv[8] );
        hasLog = true;
    }

    if ( hasLog )

```

```

{
    logfile.setf(std::ios::fixed);
    logfile.precision(2);

    logfile << "=====";
    for ( int i=0; i< myStreamID.size(); i++ )
        logfile << "=";
    logfile << '\n';
    logfile << "= Logfile for delete stream " << myStreamID << " =\n";
    logfile << "=====";
    for ( int i=0; i< myStreamID.size(); i++ )
        logfile << "=";
    logfile << "\n" << endl;
}

ifstream inputfile( argv[1] );
if ( !inputfile )
{
    cerr << "delete stream " << myStreamID << ": couldn't open inputfile '" <<
argv[1] << "' << endl;
    if ( hasLog )
        logfile << "delete stream " << myStreamID << ": couldn't open
inputfile '" << argv[1] << "' << endl;
    exit( -1 );
}

QueryStream qs( myStreamID, hasLog ? &logfile : NULL );

//cout << "hostname: " << config.host << endl;
//cout << "port : " << config.port << endl;
//cout << "user : " << config.user << endl;
//cout << "password: " << config.password << endl;
if ( ! qs.connectToServer( config ) )
{
    qs.disconnectFromServer();
    return -1;
}

// open the selected schema if any was given
if ( initialSchema != "" )
{
    Handle schemaHandle;
    std::string openSchemaStatement = "open schema " + initialSchema + ";";
    qs.prepare( schemaHandle, openSchemaStatement);
    qs.execute( schemaHandle, 0);
    qs.done( schemaHandle );
}

gettimeofday(&starttime, 0 );

uint keys[KEY_COUNT_PER_STEP];
size_t keysRead = 0;

// read orderkey values from file
string statementOrders = "delete from orders where o_orderkey = ?;";
string statementLineitem = "delete from lineitem where l_orderkey = ?;";

Handle ordersHandle;
Handle lineitemHandle;

//cout << statementOrders << endl;
qs.prepare( ordersHandle, statementOrders );

//cout << statementLineitem << endl;

```

```

qs.prepare( lineitemHandle, statementLineitem );

//cout << statementOrders << endl;
qs.bindIntParameter( ordersHandle, keys, KEY_COUNT_PER_STEP );
//cout << statementLineitem << endl;
qs.bindIntParameter( lineitemHandle, keys, KEY_COUNT_PER_STEP );

while ( !inputfile.eof() )
{
    keysRead = 0;
    while ( (keysRead < KEY_COUNT_PER_STEP) && (! inputfile.eof()) )
    {
        inputfile >> keys[keysRead];
        if ( inputfile.eof() )
            break;
        keysRead++;
    }

    //cout << keysRead << " keys read:" << endl;
    //if ( hasLog )
    //{
        //    for ( int i=0; i<keysRead; i++ )
        //        logfile << keys[i] << endl;
    //}
    //cout << statementOrders << endl;
    qs.execute( ordersHandle, keysRead );
    //cout << statementLineitem << endl;
    qs.execute( lineitemHandle, keysRead );
}

//cout << "done" << endl;
qs.done( ordersHandle );
qs.done( lineitemHandle );

//cout << statementOrders << " done" << endl;

//cout << "commit" << endl;
Handle commitHandle;
string commit = "commit;";
qs.prepare( commitHandle, commit );
qs.execute( commitHandle, 0);
qs.done( commitHandle );
//cout << "commit done" << endl;

if ( hasLog )
{
    gettimeofday( &endtime, 0 );
    logfile << "\n-----\n";
    logfile << "- Summary -\n";
    logfile << "-----\n";
    logfile << "started at: " << ctime(&(starttime.tv_sec))
    << " (epoch: " << starttime.tv_sec << " s, " << starttime.tv_usec
    << " usec"<< ")";
    logfile << "finished at: " << ctime(&(endtime.tv_sec))
    << " (epoch: " << endtime.tv_sec << " s, " << endtime.tv_usec
    << " usec" << ")";
    logfile << "time used : " << timediff(starttime, endtime) << " s\n";

    logfile << "=====\n";
    logfile << "** End of delete stream *\n";
    logfile << "=====\n";
}

```

```

qs.disconnectFromServer();

    if ( hasLog )
    {
        gettimeofday( &stream_endtime, 0 );
        logfile << "-----";
        for ( int i=0; i<myStreamID.size(); i++ )
            logfile << "-";
        logfile << "\n- Summary of deletestream " << myStreamID << " -\n";
        logfile << "-----";
        for ( int i=0; i<myStreamID.size(); i++ )
            logfile << "-";
        logfile << "\n\nstarted at: " << ctime(&(stream_starttime.tv_sec))
            << " (epoch: " << stream_starttime.tv_sec << " s, "
            << stream_starttime.tv_usec << " usec"<< ") \n";
        logfile << "finished at: " << ctime(&(stream_endtime.tv_sec))
            << " (epoch: " << stream_endtime.tv_sec << " s, "
            << stream_endtime.tv_usec << " usec" << ") \n";
        logfile << "time used : " << timediff(stream_starttime, stream_endtime)
            << " s\n\n";
    }

return 0;
}

# -----
# querystream_for_delete.h
# -----

#ifndef QUERYSTREAM_H
#define QUERYSTREAM_H

#include <string>
#include <iostream>

#include "dwaCInterface.h"

/** \brief The maximum length of a column name. */
static const size_t maxColumnNameSize = 40;

/** \brief A structure to describe connection information to a database.
 * Used in QueryStream::connectToServer().
 * \sa QueryStream::connectToServer() */
struct Configuration
{
    /** \brief The hostname of the server. */
    char *host;
    /** \brief The port of the server to connect to. */
    char *port;
    /** \brief The user name used for authentication to the database. */
    char *user;
    /** \brief The password of the user. */
    char *password;
};

struct Handle
{
    size_t max_size;
    /** \brief The only statement handle used. */
    SQLHSTMT hstmt;
    //SQLSMALLINT *statusArray;
    SQLSMALLINT parameterProcessed;
};

```

```

SQLINTEGER *indArray;

Handle();
~Handle();
};

/** \brief The class represents a query stream to a server.
 * The query stream can be used to send any query as text to the server and
 * retrieve the result. */
class QueryStream
{
public:
    /** \brief Constructor.
     * \param streamId The identifier used to identify this query stream.
     * \param logfile The stream to be used for logging. */
    QueryStream( std::string streamId, std::ostream* logfile = &std::cout );
    /** \brief Tries to establish a connection to a server.
     * \param configuration The connection and login information for the server.
     * \return true if the connection could be established, false otherwise. */
    bool connectToServer( Configuration configuration );

    bool prepare( Handle &handle, std::string statement );
    bool bindIntParameter( Handle &handle, uint *parameterArray, size_t size );
    bool execute( Handle &handle, size_t current_size );
    bool done( Handle &handle );
    /** \brief Closes the connection to the server.
     * \return true if the connection was closed successfully, false if errors occurred. */
    bool disconnectFromServer();
protected:
    void setSize( Handle &handle, long long int new_size );
    /** \brief The maximum size of a string. */
    static const size_t maxStringSize = 200;

    /** \brief The stream identifier. */
    std::string myID;

    /** \brief The environment handle used. */
    SQLHENV henv;
    /** \brief The connection handle used. */
    SQLHDBC hdbc;

    /** \brief The stream used for logging results.*/
    std::ostream* logfile;
};

#endif // QUERYSTREAM_H

# -----
# querystream_for_delete.cpp
# -----

#include "querystream_for_delete.h"
#include <iostream>

#ifdef DEBUG
#define DEBUGOUT( X ) cout << "DEBUG:" << x
#else
#define DEBUGOUT( X )
#endif

#define LOG *logfile

```

```

using std::cout;
using std::endl;
using std::cerr;

void errorDiagnosis( void *handle )
{
    if ( handle != 0 )
    {
        SQLCHAR state[6];
        SQLINTEGER nativeCode;
        SQLCHAR message[100];
        SQLSMALLINT messageLength;
        SQLSMALLINT i=0;
        int retcode;
        cout << "doing error diagnosis" << endl;
        while( ( retcode = DWAGetDiagRec( SQL_HANDLE_STMT, handle, i, state,
&nativeCode, message, 100, &messageLength ) ) != SQL_SUCCESS )
        {
            cout << "message: " << message << endl;
            i++;
        }
    }
}

SQLRETURN execCritical(SQLRETURN retcode, void* handle = 0)
{
    switch(retcode)
    {
        case SQL_INVALID_HANDLE:
            cerr << "invalid handle" << endl;
            errorDiagnosis( handle );
            throw "ERROR use of invalid handle!\n";
            break;
        case SQL_ERROR:
            cerr << "SQL error" << endl;
            errorDiagnosis( handle );
            throw "An error occured while executing command!\n";
            break;
        case SQL_SUCCESS_WITH_INFO:
            cerr << "Problem: Has Info, which should not be" << endl;
            errorDiagnosis( handle );
            throw "Problem: Has Info, which should not be!\n";
            break;
        case SQL_STILL_EXECUTING:
            cerr << "Still executing. This should not be should only";
            cerr << "return on successful execution!" << endl;
            errorDiagnosis( handle );
            throw "Unexpectedly still executing!\n";
            break;
        case SQL_NO_DATA:
        case SQL_SUCCESS:
            break;
        default:
            // should not reach here...
            cerr << "unexpected return code: " << retcode << endl;
            errorDiagnosis( handle );
            throw "unexpected return code!\n";
            break;
    }
    return retcode;
}

Handle::~Handle()

```

```

{
    max_size = 0;
    hstmt = NULL;
    indArray = NULL;
    //statusArray = NULL;
    parameterProcessed = 0;
}

Handle::~Handle()
{
    if ( indArray != 0 )
    {
        free( indArray );
        indArray = 0;
    }
    /*if ( statusArray != 0 )
    {
        free( statusArray );
        statusArray = 0;
    }*/
}

QueryStream::QueryStream( std::string streamId, std::ostream* logfile_ )
{
    myID = streamId;
    henv = NULL;
    hdbc = NULL;
    logfile = logfile_;
}

bool QueryStream::connectToServer( Configuration cfg )
{
    char localeAttr[]="deu";
    try
    {
        // Allocating environment handle
        execCritical(DWAAllocHandle(SQL_HANDLE_ENV, NULL, &henv));
        // Setting up environment localization
        execCritical(DWASetEnvAttr(henv, DWA_ENV_ATTR_SET_LOCALE_ALL, localeAttr, SQL_NTS));
        // Allocating handle for connection
        execCritical(DWAAllocHandle(SQL_HANDLE_DBC, henv, &hdbc));
        // Turning autocommit off... is this necessary?
        execCritical(DWASetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT, (void*)SQL_AUTOCOMMIT_OFF,
SQL_NTS) );
        // Setting up clientname
        execCritical(DWASetConnectAttr(hdbc, DWA_CLIENT_NAME, (void*)myID.c_str(), SQL_NTS));
        // Connect to server
        execCritical(DWAServerConnect(hdbc, cfg.host, SQL_NTS, cfg.port, SQL_NTS, cfg.user,
SQL_NTS, cfg.password, SQL_NTS), henv);
    }
    catch(...)
    {
        // It would be nice to clean up handles
        return false;
    }
    return true;
}

bool QueryStream::prepare( Handle &handle, std::string statement )
{
    handle.max_size = 0;
    try
    {

```



```

        // allocate statement handle
        execCritical(DWAAllocHandle(SQL_HANDLE_STMT, hdbc, &handle.hstmt));
        // enable column-wise binding
        //execCritical(DWASetStmtAttr(          SQL_ATTR_PARAM_BIND_TYPE,
SQL_PARAMETER_BIND_BY_COLUMN, 0 ))
        // call prepare
        execCritical(DWAPrepare(handle.hstmt,          (SQLCHAR*)statement.c_str(),
statement.size() ));
    }
    catch (...)
    {
        if ( logfile ) LOG << "Error preparing statement: " << statement << endl;
        cerr << "Error preparing statement: " << statement << endl;
        return false;
    }
}
return true;
}

void QueryStream::setSize( Handle &handle, long long int new_size )
{
    // set the number of rows in the parameter
    execCritical( DWASetStmtAttr( handle.hstmt, SQL_ATTR_PARAMSET_SIZE, (SQLPOINTER)
new_size, 0 ), handle.hstmt );
}

bool QueryStream::bindParam( Handle &handle, uint *parameterArray, size_t size )
{
    // setting size intern
    handle.max_size = size;

    try
    {
        // binding parameter, setting sizes etc
        setSize( handle, size );
        //handle.statusArray = (SQLUSMALLINT*) malloc( size * sizeof( SQLUSMALLINT)
);

        handle.indArray = (SQLINTEGER*) malloc( size * sizeof( SQLINTEGER ) );
        for ( int i=0; i<size; i++ )
        {
            handle.indArray[i] = 0;
        }
        // execCritical( DWASetStmtAttr( handle.hstmt, SQL_ATTR_PARAM_STATUS_PTR,
handle.statusArray, 0 ) );
        execCritical( DWASetStmtAttr( handle.hstmt, SQL_ATTR_PARAMS_PROCESSED_PTR,
&handle.parameterProcessed, 0 ) );
        execCritical( DWABindParameter( handle.hstmt, 1, SQL_PARAM_INPUT,
SQL_C_ULONG, SQL_BIGINT, 0, 0, (SQLPOINTER) parameterArray, sizeof(uint),handle.indArray ) );
    }
    catch (...)
    {
        if ( logfile ) LOG << "Error binding parameter" << endl;
        cerr << "Error binding parameter" << endl;
        return false;
    }
}
return true;
}

bool QueryStream::execute( Handle &handle, size_t currentSize )
{
    // if necessary resize parameters (just smaller allowed)
    // call execute
    // check return values
    try

```

```

    {
        // rebinding if necessary
        if ( currentSize < handle.max_size )
        {
            setSize( handle, currentSize );
        }
        //cout << "execute" << endl;
        execCritical( DWASetStmtAttr( handle.hstmt, handle.hstmt );
        /*for ( int i=0; i<currentSize; i++ )
        {
            cout << "i=" << i << endl;
            execCritical( handle.statusArray[i], handle.hstmt );
        }*/
        //cout << "after execute" << endl;
    }
    catch(...)
    {
        if ( logfile ) LOG << "ERROR EXECUTING PREPARED STATEMENT\n\n";
        cerr << "\nERROR EXECUTING PREPARED STATEMENT\n" << endl;
        return false;
    }
}
return true;
}

bool QueryStream::done( Handle &handle )
{
    // deallocate handle
    try
    {
        if ( handle.indArray != 0 )
        {
            free( handle.indArray );
            handle.indArray = 0;
        }
        /*if ( handle.statusArray != 0 )
        {
            free( handle.statusArray );
            handle.statusArray = 0;
        }*/
        execCritical(DWAFreeHandle(SQL_HANDLE_STMT, handle.hstmt));
    }
    catch (...)
    {
        cerr << "error in done" << endl;
        return false;
    }
}
return true;
}

bool QueryStream::disconnectFromServer()
{
    try
    {
        // disconnecting from server
        execCritical(DWADisconnect(hdbc));
        //cout << "Successfully disconnected from server." << endl;
        // freeing connection handle
        execCritical(DWAFreeHandle(SQL_HANDLE_DBC, hdbc));
        //cout << "Freeing connection handle: ok" << endl;
        // freeing environment handle
        execCritical(DWAFreeHandle(SQL_HANDLE_ENV, henv));
        //cout << "Freeing environment handle: ok" << endl;
    }
}

```

```

catch(...)
{
    cerr << "Error while disconnecting from server" << endl;
    return false;
}
return true;
}

# -----
# tpchquerystreammain.cpp
# -----

#include "querystream.h"

#include <iostream>
using std::cout;
using std::endl;

#include <fstream>
using std::ofstream;
using std::ifstream;
using std::ostream;

#include <string>
using std::string;

#include <sstream>
using std::ostringstream;

#include <iomanip>

#define MAX_LINE_SIZE 10000

#include <sys/time.h>

/** \brief Calculates the time passed between t1 and t2. */
double timediff( timeval t1, timeval t2 )
{
    double passed = 1.0e-6 * ( t2.tv_usec - t1.tv_usec );
    passed += ( t2.tv_sec - t1.tv_sec );
    if ( passed < 0 )
        passed = -passed;
    return passed;
}

/** \brief Deletes from a given string all occurrences of another string at beginning and end. */
string trim( string &s, const string drop = " " )
{
    string r = s.erase( s.find_last_not_of(drop)+1 );
    return r.erase( 0, r.find_first_not_of(drop) );
}

void timestamp( timeval mytime, ostream &out )
{
    struct tm tm_time;
    localtime_r( &mytime.tv_sec, &tm_time );
    int year = 1900 + tm_time.tm_year;
    // month in 0 (JAN)..11 (DEC). Add one to get the number of the month
    int month = tm_time.tm_mon+1;
    int day = tm_time.tm_mday;
    int hour = tm_time.tm_hour;

```

```

int minute = tm_time.tm_min;
int seconds = tm_time.tm_sec;
int milisec = mytime.tv_usec / 1000;
out << std::setw(2) << std::setfill('0') << month << "/" << std::setw(2) <<
std::setfill('0') << day << "/" << std::setw(4) << std::setfill('0') << year << " " <<
std::setw(2) << std::setfill('0') << hour << ":" << std::setw(2) << std::setfill('0') << minute
<< ":" << std::setw(2) << std::setfill('0') << seconds << "." << std::setw(3) <<
std::setfill('0') << milisec;
}

int main( int argc, char** argv )
{
    timeval stream_starttime;
    timeval stream_endtime;
    timeval starttime;
    timeval endtime;

    gettimeofday(&stream_starttime, 0);

    bool firsttime = true;
    if ( argc != 7 && argc!=8 && argc != 9 && argc != 10 )
    {
        cout << "usage: " << argv[0] << " <inputfile> <logfile> <shortlog> <streamid> <host>
<port> [<schema>] [<user> <password>]" << endl;
        cout << endl;
        cout << "\tinputfile name of the file containing the queries to execute" << endl;
        cout << "\tlogfile name of file to print output to" << endl;
        cout << "\tshortlog name of file to print a very short version of the log
to (just times)" << endl;
        cout << "\tstreamid an identifier used for this stream" << endl;
        cout << "\thost name of host to connect to" << endl;
        cout << "\tport port number on host to connect to" << endl;
        cout << "\tschema name of a schema to open prior to any statements in
inputfile (default: open no schema)" << endl;
        cout << "\tuser name of user to connect to DWA (default: sys)" << endl;
        cout << "\tpassword password used to authenticate user (default: exasol)" << endl;
        return 1;
    }
    Configuration config;
    config.host = argv[5];
    config.port = argv[6];
    if ( argc == 9 )
    {
        config.user = argv[7];
        config.password = argv[8];
    }
    else if ( argc == 10 )
    {
        config.user = argv[8];
        config.password = argv[9];
    }
    else
    {
        config.user = "sys";
        config.password = "exasol";
    }

    std::string initialSchema = "";
    if ( argc == 8 || argc == 10 )
    {
        initialSchema = argv[7];
    }
}

```



```

SQLUBIGINT      *sqlUBigInt;
/** \brief Array for columns of SQLREAL type. */
SQLREAL        *sqlReal;
/** \brief Array for columns of SQLDOUBLE type. */
SQLDOUBLE      *sqlDouble;
/** \brief Array for columns of SQLCHAR type. */
SQLCHAR        *sqlChar;
/** \brief Array for columns of SQLSCHAR type. */
SQLSCHAR       *sqlSChar;
/** \brief Array for columns of SQL_DATE_STRUCT type. */
SQL_DATE_STRUCT *sqlDate;
/** \brief Array for columns of SQL_NUMERIC_STRUCT type. */
SQL_NUMERIC_STRUCT *sqlNumeric;

/** \brief The length of a returned data and an indicator whether it is NULL. */
SQLLEN *resultLen;

/** \brief The name of the column. */
SQLCHAR columnName[maxColumnNameSize];
/** \brief The length of the column name. */
SQLSMALLINT nameLength;
/** \brief The SQL datatype ID of the column. */
SQLSMALLINT datatype;
/** \brief The ODBC C datatype ID of the column.
 * Might be set by QueryStream::find_ODBC_C_Type( SQLSMALLINT sqlType ) */
SQLSMALLINT c_datatype;
/** \brief The size of the column. */
SQLINTEGER columnSize;
/** \brief The number of decimal digits of the column. */
SQLSMALLINT decimalDigits;
/** \brief Whether the column allows NULL values. */
SQLSMALLINT nullable;
};

/** \brief The class represents a query stream to a server.
 * The query stream can be used to send any query as text to the server and
 * retrieve the result. */
class QueryStream
{
public:
    /** \brief Constructor.
     * \param streamId The identifier used to identify this query stream.
     * \param logfile The stream to be used for logging. */
    QueryStream( std::string streamId, std::ostream* logfile = &std::cout );
    /** \brief Tries to establish a connection to a server.
     * \param configuration The connection and login information for the server.
     * \return true if the connection could be established, false otherwise. */
    bool connectToServer( Configuration configuration );
    /** \brief Executes a given statement or query.
     * \param statement The text of the statement or query to execute.
     * \param fetchSize The number of rows to fetch with a single fetch command
     * (bigger fetch size might reduce overhead, but increases memory usage).
     * \return true if the statement was executed successfully, false if some error
     * occurred while executing the statement or fetching the result set. */
    bool execute( std::string statement, long int fetchSize = standardFetchBlockSize );
    /** \brief Closes the connection to the server.
     * \return true if the connection was closed successfully, false if errors occurred. */
    bool disconnectFromServer();
protected:
    /** \brief The maximum size of a string. */
    static const size_t maxStringSize = 200;
    /** \brief The number of rows to fetch in a single fetch command. */

```

```

SQLINTEGER fetchBlockSize;
/** \brief The number to use for fetchBlockSize if it is not
 * given as argument to execute(). */
    static const size_t standardFetchBlockSize = 100;

/** \brief Binds a column to an entry in the fetchMe array.
 * This must be done before fetching a resultset.
 * This function arranges memory allocation and binding function.
 * \param colNo The number of the resultset column to bind to.
 * \param localColNo The index in the fetchMe array the resultset data shall be stored.
 * \param datatype The SQL datatype of the column to bind to. */
void bindColumn( int colNo, int localColNo, SQLSMALLINT datatype );

/** \brief Allocates memory for the buffers needed in binding resultset columns.
 * The array corresponding to the given datatype of the given index
 * of the fetchMe array is initialized to the size of fetchBlockSize.
 * If the datatype is a string datatype, the string buffer will have the
 * size maxStringSize.
 * \param localColNo The index in the fetchMe array the data shall be allocated for.
 * \param datatype The ODBC-C-datatype memory shall be allocated for. */
void allocateMemoryForBind( int localColNo, SQLSMALLINT datatype );

/** \brief Maps an SQL-datatype-ID to a ODBC-C-datatype-ID to which it can be converted.
 * This is needed for binding columns and fetching results from a remote database.
 * \param sqlType The SQL-datatype-ID whose corresponding ODBC-C-datatype-ID is wanted.
 * \return The ODBC-C-datatype-ID for the given SQL-datatype-ID */
SQLSMALLINT find_ODBC_C_Type( SQLSMALLINT sqlType );

/** \brief Format resultset header for output. */
void outputHeader();

/** \brief Formats fetched rows for output.
 * The data in fetchMe is printed for each column and all fetched rows. */
void outputFetchedRows();

/** \brief The stream identifier. */
std::string myID;

/** \brief The environment handle used. */
SQLHENV henv;
/** \brief The connection handle used. */
SQLHDBC hdbc;
/** \brief The only statement handle used. */
SQLHSTMT hstmt;

/** \brief The structure used to fetch result sets of queries. */
FetchStruct *fetchMe;

/** \brief The number of rows in a result set. */
SQLSMALLINT numRows;

/** \brief The number of rows fetched by last fetch command. */
SQLINTEGER rowsFetched;

/** \brief The stream used for logging results.*/
std::ostream* logfile;
};

#endif // QUERYSTREAM_H

# -----

```

```

# querystream.cpp
# -----

#include "querystream.h"
#include <iostream>

#ifdef DEBUG
#define DEBUGOUT( X ) cout << "DEBUG:" << x
#else
#define DEBUGOUT( X )
#endif

#define LOG *logfile

using std::cout;
using std::endl;
using std::cerr;

SQLRETURN execCritical(SQLRETURN retcode, void* handle = 0)
{
    switch(retcode)
    {
        case SQL_INVALID_HANDLE:
            cerr << "invalid handle" << endl;
            throw "ERROR use of invalid handle!\n";
            break;
        case SQL_ERROR:
            cerr << "SQL error" << endl;
            throw "An error occurred while executing command!\n";
            break;
        case SQL_SUCCESS_WITH_INFO:
            cerr << "Problem: Has Info, which should not be" << endl;
            throw "Problem: Has Info, which should not be!\n";
            break;
        case SQL_STILL_EXECUTING:
            cerr << "Still executing. This should not be should only";
            cerr << "return on successful execution!" << endl;
            throw "Unexpectedly still executing!\n";
            break;
        case SQL_NO_DATA:
        case SQL_SUCCESS:
            break;
        default:
            // should not reach here...
            cerr << "unexpected return code: " << retcode << endl;
            throw "unexpected return code!\n";
            break;
    }
    return retcode;
}

QueryStream::QueryStream( std::string streamId, std::ostream* logfile_ )
{
    myID = streamId;
    henv = NULL;
    hdbc = NULL;
    hstmt = NULL;
    logfile = logfile_;
}

bool QueryStream::connectToServer( Configuration cfg )
{

```

```

    char localeAttr[]="deu";
    try
    {
        // Allocating environment handle
        execCritical(DWAAllocHandle(SQL_HANDLE_ENV, NULL, &henv));
        // Setting up environment localization
        execCritical(DWASetEnvAttr(henv, DWA_ENV_ATTR_SET_LOCALE_ALL, localeAttr, SQL_NTS));
        // Allocating handle for connection
        execCritical(DWAAllocHandle(SQL_HANDLE_DBC, henv, &hdbc));
        // Turning autocommit off... is this necessary?
        execCritical(DWASetConnectAttr(hdbc, SQL_ATTR_AUTOCOMMIT, (void*)SQL_AUTOCOMMIT_OFF,
SQL_NTS) );
        // Setting up clientname
        execCritical(DWASetConnectAttr(hdbc, DWA_CLIENT_NAME, (void*)myID.c_str(), SQL_NTS));
        // Connect to server
        execCritical(DWAServerConnect(hdbc, cfg.host, SQL_NTS, cfg.port, SQL_NTS, cfg.user,
SQL_NTS, cfg.password, SQL_NTS), henv);
    }
    catch(...)
    {
        // It would be nice to clean up handles
        return false;
    }
    return true;
}

bool QueryStream::execute( std::string statement, long int fetchSize )
{
    try
    {
        // setting the fetchBlockSize (transmitted to DWA later on)
        this->fetchBlockSize = fetchSize;
        // put the statement in the logfile
        LOG << statement << '\n';
        // Allocating statement handle
        execCritical(DWAAllocHandle(SQL_HANDLE_STMT, hdbc, &hstmt));
        DEBUGOUT( "Allocating statement handle: ok" << endl );
        SQLCHAR* statementtext = (SQLCHAR*) statement.c_str();
        // execute the statement / query
        execCritical(DWAExecDirect(hstmt, statementtext, statement.size() ) );
        // get the affected rows (only insert, update, delete)
        SQLINTEGER affectedrows;
        execCritical(DWARowCount(hstmt,&affectedrows));
        // get the number of columns in the resultset (if any)
        execCritical(DWANumResultCols(hstmt,&numResultCols));
        // no result columns means no query
        if ( numResultCols <= 0 )
            LOG << "affected rows: " << affectedrows << "\n\n";

        // if a resultset was constructed, retrieve and print it
        if ( numResultCols > 0 )
            // a resultset was constructed
            {
                LOG << '\n';
                // the number of rows in the resultSet
                int resultSetSize = 0;
                // setting number of rows to fetch with each fetch
                SQLINTEGER fetchsize = fetchBlockSize;
                execCritical(DWASetStmtAttr(hstmt, SQL_ATTR_ROW_ARRAY_SIZE,
(SQLPOINTER) fetchsize , 0 ));
                //cout << "Fetch block size (SQL_ATTR_ROW_ARRAY_SIZE) set to " << fetchsize <<
endl;
            }
    }

```

```

// setting position to store the fetch size for a query
execCritical(DWASetStmtAttr(hstmt, SQL_ATTR_ROWS_FETCHED_PTR,
                          (SQLPOINTER) &rowsFetched, 0));

// allocate the fetch struct to be large enough for the columns
fetchMe = new FetchStruct(numResultCols);
// set all pointers to 0 to make it easy to free the correct memory later
for ( int i=0; i<numResultCols; i++ )
{
    fetchMe[i].sqlCharArray = 0;
    fetchMe[i].sqlCharArrayAccess = 0;

    fetchMe[i].sqlSmallInt = 0;
    fetchMe[i].sqlUSmallInt = 0;
    fetchMe[i].sqlInteger = 0;
    fetchMe[i].sqlUIInteger = 0;
    fetchMe[i].sqlBigInt = 0;
    fetchMe[i].sqlUBigInt = 0;
    fetchMe[i].sqlReal = 0;
    fetchMe[i].sqlDouble = 0;
    fetchMe[i].sqlChar = 0;
    fetchMe[i].sqlSChar = 0;
    fetchMe[i].sqlDate = 0;
    fetchMe[i].sqlNumeric = 0;
    fetchMe[i].resultLen = 0;
}
execCritical(DWASetStmtAttr(hstmt, SQL_ATTR_ROWS_FETCHED_PTR,
                          (SQLPOINTER) &rowsFetched, 0));
// determine the datatype of the column and additional information.
for ( SQLSMALLINT colNr=0; colNr<numResultCols; colNr++ )
{
    // Get column description for the column with number colNr
    DWADescribeCol( hstmt, colNr+1, fetchMe[colNr].columnName,
                  maxColumnNameSize, &(fetchMe[colNr].nameLength),
                  &(fetchMe[colNr].datatype), &(fetchMe[colNr].columnSize),
                  &(fetchMe[colNr].decimalDigits), &(fetchMe[colNr].nullable) );
    fetchMe[colNr].c_datatype = find_ODBC_C_Type( fetchMe[colNr].datatype );

    // bind the column to an entry in the fetchMe array
    bindColumn( colNr+1, colNr, fetchMe[colNr].c_datatype );
}
// output the resultset header
outputHeader();
// fetch the columns and print them to output
do
{
    // fetch columns
    execCritical(DWAFetch( hstmt ));
    // output columns
    outputFetchedRows();
    // correct the result set size
    resultSetSize += rowsFetched;
} while ( rowsFetched == fetchBlockSize );
// add the total resultset size to the logfile
LOG << "\nresultset consists of " << resultSetSize << " rows in " << numResultCols
<< " columns\n\n";
// free memory for column variables again
for ( int i=0; i<numResultCols; i++ )
{
    if ( fetchMe[i].sqlCharArray != 0 )
    {
        delete[] fetchMe[i].sqlCharArray;
        delete[] fetchMe[i].sqlCharArrayAccess;
    }
}

```

```

}
if ( fetchMe[i].sqlSmallInt != 0 ) delete[] fetchMe[i].sqlSmallInt;
if ( fetchMe[i].sqlUSmallInt != 0 ) delete[] fetchMe[i].sqlUSmallInt;
if ( fetchMe[i].sqlInteger != 0 ) delete[] fetchMe[i].sqlInteger;
if ( fetchMe[i].sqlUIInteger != 0 ) delete[] fetchMe[i].sqlUIInteger;
if ( fetchMe[i].sqlBigInt != 0 ) delete[] fetchMe[i].sqlBigInt;
if ( fetchMe[i].sqlUBigInt != 0 ) delete[] fetchMe[i].sqlUBigInt;
if ( fetchMe[i].sqlReal != 0 ) delete[] fetchMe[i].sqlReal;
if ( fetchMe[i].sqlDouble != 0 ) delete[] fetchMe[i].sqlDouble;
if ( fetchMe[i].sqlChar != 0 ) delete[] fetchMe[i].sqlChar;
if ( fetchMe[i].sqlSChar != 0 ) delete[] fetchMe[i].sqlSChar;
if ( fetchMe[i].sqlDate != 0 ) delete[] fetchMe[i].sqlDate;
if ( fetchMe[i].sqlNumeric != 0 ) delete[] fetchMe[i].sqlNumeric;
if ( fetchMe[i].resultLen != 0 ) delete[] fetchMe[i].resultLen;
}
delete[] fetchMe;
}
// free the handle for the statement again
execCritical(DWAFreeHandle( SQL_HANDLE_STMT, hstmt ));
}
catch(...)
{
    LOG << "ERROR EXECUTING STATEMENT\n\n";
    cerr << "\nERROR EXECUTING STATEMENT:\n" << statement << '\n' << endl;
    return false;
}
return true;
}

bool QueryStream::disconnectFromServer()
{
    try
    {
        // disconnecting from server
        execCritical(DWADisconnect(hdbc));
        //cout << "Successfully disconnected from server." << endl;
        // freeing connection handle
        execCritical(DWAFreeHandle(SQL_HANDLE_DBC, hdbc));
        //cout << "Freeing connection handle: ok" << endl;
        // freeing environment handle
        execCritical(DWAFreeHandle(SQL_HANDLE_ENV, henv));
        //cout << "Freeing environment handle: ok" << endl;
    }
    catch(...)
    {
        cerr << "Error while disconnecting from server" << endl;
        return false;
    }
    return true;
}

void QueryStream::outputHeader()
{
    int headerLength = 1;
    LOG << "|";
    for ( size_t col=0; col<numResultCols; col++ )
    {
        LOG << fetchMe[col].columnName << "|";
        headerLength += fetchMe[col].nameLength + 1;
    }
    LOG << '\n';
    for ( int i=0; i<headerLength; i++ )

```

```

        LOG << '=';
    LOG << '\n';
}

void QueryStream::outputFetchedRows()
{
    for ( size_t rows=0; rows<rowsFetched; rows++ )
    {
        LOG << "|";
        for ( size_t cols=0; cols<numResultCols; cols++ )
        {
            if ( fetchMe[cols].resultLen[rows] == SQL_NULL_DATA )
            {
                LOG << "|";
                continue;
            }
            switch( fetchMe[cols].c_datatype )
            {
                // string data
                case SQL_C_CHAR:
                case SQL_C_BINARY:
                //case SQL_C_XML:
                //case SQL_C_VARBOOKMARK:
                    LOG << fetchMe[cols].sqlCharArrayAccess[rows] << "|";
                    break;

                // small int
                case SQL_C_SSHORT:
                    LOG << fetchMe[cols].sqlSmallInt[rows] << "|";
                    break;

                // unsigned small int
                case SQL_C_USHORT:
                    LOG << fetchMe[cols].sqlUSmallInt[rows] << "|";
                    break;

                // long int
                case SQL_C_SLONG:
                    LOG << fetchMe[cols].sqlInteger[rows] << "|";
                    break;

                // unsigned long int
                case SQL_C_ULONG:
                    LOG << fetchMe[cols].sqlUInteger[rows] << "|";
                    break;

                // float
                case SQL_C_FLOAT:
                    LOG << fetchMe[cols].sqlReal[rows] << "|";
                    break;

                // double
                case SQL_C_DOUBLE:
                    LOG << fetchMe[cols].sqlDouble[rows] << "|";
                    break;

                // unsigned char
                case SQL_C_BIT:
                case SQL_C_UTINYINT:
                    LOG << fetchMe[cols].sqlChar[rows] << "|";
                    break;
            }
        }
    }
}

```

```

// signed char
case SQL_C_STINYINT:
    LOG << fetchMe[cols].sqlSChar[rows] << "|";
    break;

// big int
case SQL_C_SBIGINT:
    LOG << fetchMe[cols].sqlBigInt[rows] << "|";
    break;

// unsigned big int
case SQL_C_UBIGINT:
    LOG << fetchMe[cols].sqlUBigInt[rows] << "|";
    break;

// date
case SQL_C_TYPE_DATE:
{
    // format date to YYYY-MM-DD for output
    if ( fetchMe[cols].sqlDate[rows].year < 1000 )
        LOG << "0";
    if ( fetchMe[cols].sqlDate[rows].year < 100 )
        LOG << "0";
    if ( fetchMe[cols].sqlDate[rows].year < 10 )
        LOG << "0";
    LOG << fetchMe[cols].sqlDate[rows].year << "-";

    if ( fetchMe[cols].sqlDate[rows].month < 10 )
        LOG << "0";
    LOG << fetchMe[cols].sqlDate[rows].month << "-";

    if ( fetchMe[cols].sqlDate[rows].day < 10 )
        LOG << "0";
    LOG << fetchMe[cols].sqlDate[rows].day << "|";
}
break;

// numeric, i.e. decimal
// not used since decimal is mapped to double
case SQL_C_NUMERIC:
{
    // format decimal for output
    int scale = fetchMe[cols].sqlNumeric[rows].scale;
    char text[3*SQL_MAX_NUMERIC_LEN+2+scale];
    int pos_T = 3*SQL_MAX_NUMERIC_LEN+scale;
    int pos_N = SQL_MAX_NUMERIC_LEN-1;
    text[3*SQL_MAX_NUMERIC_LEN+1+scale] = '\0';
    //int add = 0;
    //if ( fetchMe[cols].sqlNumeric[rows].sign == 0 )
    //    add=1;
    while ( fetchMe[cols].sqlNumeric[rows].val[pos_N] == 0 && pos_N > 0 )
        pos_N--;

    while ( pos_N >= 0 )
    {
        int current = fetchMe[cols].sqlNumeric[rows].val[pos_N];
        int newVal = 0;
        int currentPos = pos_N;
        while ( currentPos >= 0 )
        {
            //if ( currentPos == 0 )
            //{

```



```

        // current += add;
        // add = 0;
        //}
        if ( current < 10 )
        {
            fetchMe[cols].sqlNumeric[rows].val[currentPos] = newVal;
            currentPos--;

            if ( currentPos >= 0 )
            {
                current = current * 256 +
fetchMe[cols].sqlNumeric[rows].val[currentPos];
                newVal = 0;
            }
            else
            {
                newVal = current / 10;
                current = current % 10;
            }
        }
        if ( fetchMe[cols].sqlNumeric[rows].val[pos_N] == 0 )
        {
            pos_N--;
        }
        text[pos_T--] = '0' + current;
        if ( scale == 1 )
        {
            text[pos_T--] = '.';
        }
        scale--;
    }
    if ( scale==0 )
        text[pos_T--] = '0';
    while ( scale > 0 )
    {
        text[pos_T--] = '0';
        if ( scale == 1 )
        {
            text[pos_T--] = '.';
            text[pos_T--] = '0';
        }
        scale--;
    }

    if ( fetchMe[cols].sqlNumeric[rows].sign == 0 )
        text[pos_T--] = '-';
    LOG << text+pos_T+1 << "/";
}
break;

// other datatype
default:
    cout << "output: datatype " << fetchMe[cols].c_datatype << " not supported
yet" << endl;
    throw fetchMe[cols].c_datatype;
break;
}
}
LOG << '\n';
}
}

```

```

}
void QueryStream::bindColumn( int colNo, int localColNo, SQLSMALLINT c_datatype )
{
    // allocate memory for the column to bind
    allocateMemoryForBind( localColNo, c_datatype );
    // bind the column
    switch ( c_datatype )
    {
        // string data
        case SQL_C_CHAR:
        case SQL_C_BINARY:
        //case SQL_C_XML:
        //case SQL_C_VARBOOKMARK:
        execCritical(DWABindCol(
            hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlCharArray,
            maxStringSize*sizeof(SQLCHAR), fetchMe[localColNo].resultLen));
        << endl );
        DEBUGOUT( "bound C-String to column " << colNo << " (index " << localColNo << ")"
        break;

        // small int
        case SQL_C_SSHORT:
        execCritical(DWABindCol(
            hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlSmallInt,
            sizeof(SQLSMALLINT), fetchMe[localColNo].resultLen));
        << endl );
        DEBUGOUT( "bound short int to column " << colNo << " (index " << localColNo << ")"
        break;

        // unsigned small int
        case SQL_C_USHORT:
        execCritical(DWABindCol(
            hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlUSmallInt,
            sizeof(SQLUSMALLINT), fetchMe[localColNo].resultLen));
        << endl );
        DEBUGOUT( "bound unsigned short int to column " << colNo << " (index " <<
localColNo << ")" << endl );
        break;

        // long int
        case SQL_C_SLONG:
        execCritical(DWABindCol(
            hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlInteger,
            sizeof(SQLINTEGER), fetchMe[localColNo].resultLen));
        << endl );
        DEBUGOUT( "bound long int to column " << colNo << " (index " << localColNo << ")"
        break;

        // unsigned long int
        case SQL_C_ULONG:
        execCritical(DWABindCol(
            hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlUInteger,
            sizeof(SQLUIINTEGER), fetchMe[localColNo].resultLen));
        << endl );
        DEBUGOUT( "bound unsigned long int to column " << colNo << " (index " << localColNo
<< ")" << endl );
        break;

        // float
        case SQL_C_FLOAT:
        execCritical(DWABindCol(
            hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlReal,
            sizeof(SQLREAL), fetchMe[localColNo].resultLen));
    }
}

```

```

        DEBUGGOUT( "bound float to column " << colNo << " (index " << localColNo << ")" <<
endl );
break;

// double
case SQL_C_DOUBLE:
    execCritical(DWABindCol(
        hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlDouble,
        sizeof(SQLDOUBLE), fetchMe[localColNo].resultLen));
    DEBUGGOUT( "bound double to column " << colNo << " (index " << localColNo << ")" <<
endl );
break;

// unsigned char
case SQL_C_BIT:
case SQL_C_UTINYINT:
    execCritical(DWABindCol(
        hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlChar,
        sizeof(SQLCHAR), fetchMe[localColNo].resultLen));
    DEBUGGOUT( "bound char to column " << colNo << " (index " << localColNo << ")" <<
endl );
break;

// signed char
case SQL_C_STINYINT:
    execCritical(DWABindCol(
        hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlSmallInt,
        sizeof(SQLCHAR), fetchMe[localColNo].resultLen));
    DEBUGGOUT( "bound signed char to column " << colNo << " (index " << localColNo <<
)" << endl );
break;

// big int
case SQL_C_SBIGINT:
    execCritical(DWABindCol(
        hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlBigInt,
        sizeof(SQLBIGINT), fetchMe[localColNo].resultLen));
    DEBUGGOUT( "bound signed big int to column " << colNo << " (index " << localColNo <<
)" << endl );
break;

// unsigned big int
case SQL_C_UBIGINT:
    execCritical(DWABindCol(
        hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlUBigInt,
        sizeof(SQLUBIGINT), fetchMe[localColNo].resultLen));
    DEBUGGOUT( "bound unsigned big int to column " << colNo << " (index " << localColNo
<< ")" << endl );
break;

// date
case SQL_C_TYPE_DATE:
    execCritical(DWABindCol(
        hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlDate,
        sizeof(SQL_DATE_STRUCT), fetchMe[localColNo].resultLen));
    DEBUGGOUT( "bound date to column " << colNo << " (index " << localColNo << ")" <<
endl );
break;

// numeric, i.e. decimal
case SQL_C_NUMERIC:
    execCritical(DWABindCol(

```

```

        hstmt, colNo, c_datatype, (SQLPOINTER) fetchMe[localColNo].sqlNumeric,
        sizeof(SQL_NUMERIC_STRUCT), fetchMe[localColNo].resultLen));
    DEBUGGOUT( "bound numeric to column " << colNo << " (index " << localColNo << ")" <<
endl );
break;

// other datatype
default:
    DEBUGGOUT( "column binding: datatype " << c_datatype << " not supported yet" << endl
);
    throw c_datatype;
break;
}
}

void QueryStream::allocateMemoryForBind( int localColNo, SQLSMALLINT datatype )
{
    // allocate memory for resultLen field used to indicate NULL values (among others)
    fetchMe[localColNo].resultLen = new SQLLEN[fetchBlockSize];

    // allocate memory depending on the column datatype
    switch ( datatype )
    {
        // string data
        case SQL_C_CHAR:
        case SQL_C_BINARY:
        //case SQL_C_XML:
        //case SQL_C_VARBOOKMARK:
            fetchMe[localColNo].sqlCharArray = new SQLCHAR[fetchBlockSize*maxStringSize];
            fetchMe[localColNo].sqlCharArrayAccess = new
SQLCHAR*[fetchBlockSize];
            for ( size_t i=0; i<fetchBlockSize; i++ )
            {
                fetchMe[localColNo].sqlCharArrayAccess[i] =
fetchMe[localColNo].sqlCharArray+i*maxStringSize;
            }
            break;

        // small int
        case SQL_C_SSHORT:
            fetchMe[localColNo].sqlSmallInt = new SQLSMALLINT[fetchBlockSize];
            break;

        // unsigned small int
        case SQL_C_USHORT:
            fetchMe[localColNo].sqlUSmallInt = new SQLUSMALLINT[fetchBlockSize];
            break;

        // long int
        case SQL_C_SLONG:
            fetchMe[localColNo].sqlInteger = new SQLINTEGER[fetchBlockSize];
            break;

        // unsigned long int
        case SQL_C_ULONG:
            fetchMe[localColNo].sqlUInteger = new SQLUInteger[fetchBlockSize];
            break;

        // float
        case SQL_C_FLOAT:
            fetchMe[localColNo].sqlReal = new SQLREAL[fetchBlockSize];
            break;

```

```

// double
case SQL_C_DOUBLE:
    fetchMe[localColNo].sqlDouble = new SQLDOUBLE[fetchBlockSize];
break;

// unsigned char
case SQL_C_BIT:
case SQL_C_UTINYINT:
    fetchMe[localColNo].sqlChar = new SQLCHAR[fetchBlockSize];
break;

// signed char
case SQL_C_STINYINT:
    fetchMe[localColNo].sqlSChar = new SQLSCHAR[fetchBlockSize];
break;

// big int
case SQL_C_SBIGINT:
    fetchMe[localColNo].sqlBigInt = new SQLBIGINT[fetchBlockSize];
break;

// unsigned big int
case SQL_C_UBIGINT:
    fetchMe[localColNo].sqlUBigInt = new SQLUBIGINT[fetchBlockSize];
break;

// date
case SQL_C_TYPE_DATE:
    fetchMe[localColNo].sqlDate = new SQL_DATE_STRUCT[fetchBlockSize];
break;

// numeric, i.e. decimal
case SQL_C_NUMERIC:
    fetchMe[localColNo].sqlNumeric = new SQL_NUMERIC_STRUCT[fetchBlockSize];
break;

// other datatype
default:
    cout << "memory allocation: datatype " << datatype << " not supported yet" << endl;
    throw datatype;
break;
}
}

SQLSMALLINT QueryStream::find_ODBC_C_Type( SQLSMALLINT sqlType )
{
    switch( sqlType )
    {
        case SQL_CHAR:
        case SQL_VARCHAR:
        case SQL_LONGVARCHAR:
        case SQL_WCHAR:
        case SQL_WVARCHAR:
        case SQL_WLONGVARCHAR:
            return SQL_C_CHAR;
        break;

        case SQL_DECIMAL:
        case SQL_NUMERIC:
            //return SQL_C_NUMERIC;
            return SQL_C_DOUBLE;
    }
}

```

```

//return SQL_C_CHAR;
break;

case SQL_SMALLINT:
    return SQL_C_SSHORT;
break;

case SQL_INTEGER:
    return SQL_C_SLONG;
break;

case SQL_REAL:
case SQL_FLOAT:
case SQL_DOUBLE:
    return SQL_C_DOUBLE;
break;

case SQL_BIT:
    return SQL_C_BIT;
break;

case SQL_TINYINT:
    return SQL_C_STINYINT;
break;

case SQL_BIGINT:
    return SQL_C_SBIGINT;
break;

case SQL_BINARY:
case SQL_VARBINARY:
case SQL_LONGVARBINARY:
    return SQL_C_BINARY;
break;

case SQL_TYPE_DATE:
    return SQL_C_TYPE_DATE;
break;

default:
    cout << "unsupported datatype, don't know corresponding C type: " << sqlType <<
endl;
    throw sqlType;
break;
}
}

#-----
# Diff of DGen-Files
# -----

# dss.h

462c462
< #define SEPARATOR '|' /* field spearator for generated flat files */
---
> #define SEPARATOR ',' /* field spearator for generated flat files */
482a483
> #define PR_HUGE_LAST(f, str) dbg_print(DT_HUGE, f, (void *)str, 0, 0)

```

```

# makefile.suite

76c70
< CC      =
---
> CC      = /usr/bin/gcc
82,84c76,78
< DATABASE=
< MACHINE =
< WORKLOAD =
---
> DATABASE= DWA
> MACHINE = LINUX
> WORKLOAD = TPCB
88c82
< CFLAGS  = -O -DDBNAM="dss\" -D$(MACHINE) -D$(DATABASE)
-D$(WORKLOAD)
---
> CFLAGS  = -O1 -DDBNAM="dss\" -D$(MACHINE) -D$(DATABASE)
-D$(WORKLOAD) -DTERABYTE -DEOL_HANDLIN -DEOL_HANDLING

print.c

114c114
<          fprintf(target, "%s", (char *)data);
---
>          fprintf(target, "%s", (char *)data);
153c153
<          fprintf(target, "%c", *(char *)data);
---
>          fprintf(target, "%c", *(char *)data);
484c484
<          PR_HUGE(dfp, &new);
---
>          PR_HUGE_LAST(dfp, &new);

# tpcd.h

51a52,60
> #ifdef DWA
> #define GEN_QUERY_PLAN "-- GEN_QUERY_PLAN"
> #define START_TRAN    ""
> #define END_TRAN      "COMMIT;"
> #define SET_OUTPUT    ""
> #define SET_ROWCOUNT "LIMIT %d\n"
> #define SET_DBASE     "OPEN SCHEMA %s;\n"
> #endif
>

```

Appendix G: Pricing Information

EXASOL AG
Neumeyerstrasse 48
D-90411 Nürnberg
T +49 911 23991 0
F +49 911 23991 5241



October 15, 2007

Fujitsu Siemens Computers GmbH
Heinz-Nixdorf-Ring 1
33106 Paderborn

Here is the requested price list:

Product	Part Number	Price	Quantity	Discount	Extended Price
Cluster-Software EXASolution 2.0 for 40 nodes with 16 GB License for 3 years (EXACluster OS 1.3 included)	EXA-40N-16G	692,213 \$	1	10%	622,991 \$
Premium-Support 24x7 (3.9%)	EXA-SUP-P	26,996 \$	1		26,996 \$
Total					649,987 \$

This quote is valid for 60 days.

EXASOL Pricing Contact:
Olga Sapozhnykova
sales@exasol.com