Telecom One (TM1) Benchmark Description

Version 1.0

Solid Information Technology

20400 Stevens Creek Blvd., Suite 200, Cupertino, CA 95014, USA

Merimiehenkatu 36D, 00150 Helsinki, Finland

Solid K.K., 43th Floor, The Landmark Tower Yokohama,
2-2-1 Minatomirai, Nishi-ku, Yokohama 220-8143, Japan

www.solidtech.com

TM1 Benchmark Description

Version 1.0.7
Last modified on 1 March, 2005

Table of Contents

Introduction

The Telecom One (TM1) benchmark is designed to estimate the performance of an RDBMS/OS/Hardware combination in a typical telco application.

The benchmark generates a measured load on a database server. The load is generated by issuing pre-defined transactions against a specified target database. The target database schema is made to resemble a typical Home Location Register (HLR) database in a mobile phone network. The HLR is a database that mobile network operators use to store information about their customers and the services to which they have subscribed.

The TM1 Benchmark was originally published in a Master's Thesis [1]. The thesis states that the benchmark was modeled after a real test program that used by a telecom equipment manufacturer to evaluate the applicability of various relational database implementations to support control programming in mobile networks. The TM1 benchmark described here adheres fully to the specifications of [1] in terms of the database schema, transactions, population rules, value distributions, transaction mix and the test life cycle. The size of the database populations has been increased, and the configuration parameters have been adjusted to the capabilities of contemporary hardware.

TM1 is based on seven (7) pre-defined transactions that insert, update, delete and query the data in the database. The benchmark is run for two (2) hours and during that period the number of times each transaction is executed follows the transaction probability assigned for the transactions in the transaction mixture (see section *Transactions* for details).

Before each benchmark run, the benchmark schema tables are populated according to strict rules for data granularity, distributions and integrity constraints (see section *Database Schema and Population* for details). This ensures that each benchmark run begins with a consistent database population.

The TM1 results show Maximum Qualified Throughput (MQTh) of the target database system, and the response time distributions per transaction types for all seven types of transactions.

The *Execution and Results* section explains the execution principles of the benchmark. The section *Database Schema and Population* introduces the database schema for TM1 benchmark and the population policy for the database tables involved. The *Transactions* section explains the TM1 transactions are in detail, including SQL syntax. In *Configuration Guidelines,* instructions for configuring the target database systems are laid out. The section *Publishing Results* offers guidance on how to publish results to the general community.

This Benchmark is provided *as is* to the database and telco community for their own uses. Solid cannot guarantee the accuracy of any reported results, nor does it make any warranty about the relevance of such results to any specific application.

Execution and Results

Execution Environment

The execution environment of the benchmark (see Figure 1) follows a typical client/server setting, with the exception that all the clients reside on a single client computer. The system under test (SUT) is run in a dedicated computer if a standalone server is tested. In the case of a hot standby configuration, two computers host the active and standby servers, respectively. The results of each benchmark run are stored in a specified result database (TIRDB). Typically this database is located in a third computer. The following depicts a typical execution environment for a standalone server.

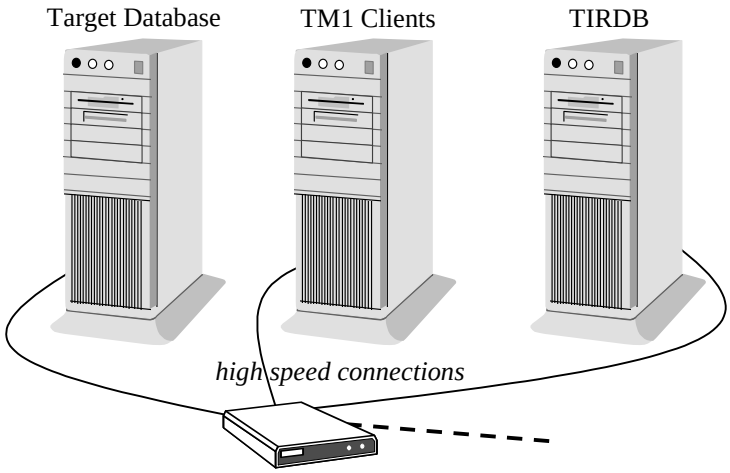

Figure 1. Typical execution environment for TM1 benchmark.

Benchmark Run

A single TM1 run consist of four phases, namely

1. Database creation and population
2. Idle time + ramp-up time
3. Result collection time (actual benchmark test)
4. Result output

These phases are shown in a time line below in Figure 2. The time intervals in the time line give an idea of a typical TM1 run and how it is divided into separate operational intervals. Note that the interval $[t_3,t_4]$ always lasts for 120 minutes (two hours).

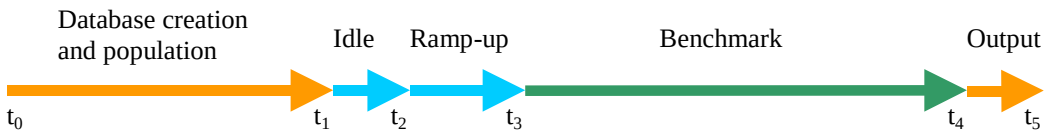

Figure 2. A TM1 run time line.

The benchmark run does not require any user intervention. The software runs from t_0 to t_5 automatically. The process lasts from three (3) hours up, depending on the population size.

Benchmark Results

The TM1 software collects two types of results from the benchmark, namely Maximum Qualified Throughput (MQTh) and transaction response time distributions.

MQTh is the number of successful transactions per time unit. In TM1 we use one second as a time unit, resulting in MQTh/s.

The response time is measured for each individual transaction and reported by transaction type. This provides seven (7) distributions measured with a millisecond resolution. The maximum response time recorded is set to be 10,000 millisecond (10 seconds). Longer response times are discarded.

The results of the benchmark are stored in a special database called TIRDB.

Database Schema and Population

Figure 3 shows the Benchmark database schema used in the TM1 benchmark.

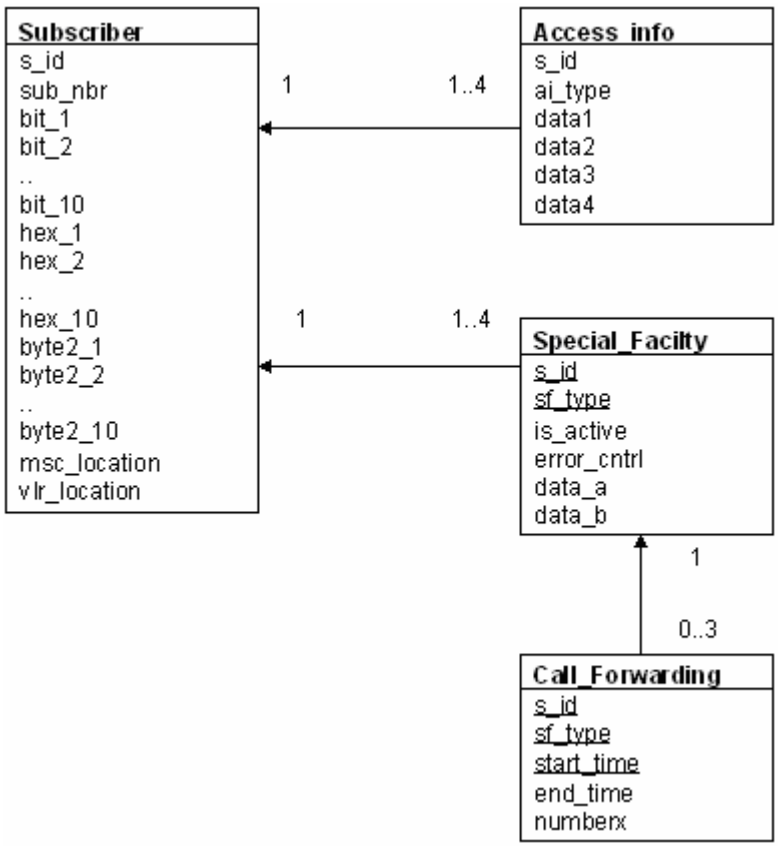

Figure 3. Telecom One (TM1) database schema.

Subscriber Table

- s_id is a unique number between 1 and N where N is the number of subscribers (the *population size*).Typically, the population sizes start at N=100,000 subscribers, and then N is multiplied by factors of 2, 5 and 10 and so forth, for each order of magnitude. During the population, s_id is selected randomly from the set of allowed values.
- sub_nbr is a 15 digit string. It is generated from s_id by transforming s_id to string and padding it with leading zeros.
  For example:
  s_id 123
  sub_nbr "000000000000123"
- bit_X fields are randomly generated either 0 or 1 values.

- hex_X fields are randomly generated numbers between 0 and 15.
- byte2_X fields are randomly generated numbers between 0 and 255.
- msc_location and vlr_location are randomly generated numbers between 1 and $(2^{32} - 1)$.

Access_Info Table

- s_id refers to the s_id in the *Subscriber* table.
- ai_type is a number between 1 and 4. It is randomly chosen, but there can be only one record of each ai_type per each subscriber. So if there are four Access_Info records for a certain subscriber they have values 1,2,3 and 4.
- data1 and data2 are randomly generated numbers between 0 and 255.
- data3 is a 3-character string that is filled with random characters created with upper case A-Z letters.
- data4 is a 5-character string that is filled with random characters created with upper case A-Z letters.

The are between 1 and 4 Access_Info records per *Subscriber* record, so that there are 25 % subscribers with one record, 25% with two records and so on.

Special_Facility Table

- s_id refers to the s_id in the *Subscriber* table.
- sf_type is a number between 1 and 4. It is randomly chosen, but there can be only one record of each sf_type per each subscriber. So if there are four *Special_Facility* records for a certain subscriber they have values 1,2,3 and 4.
- is_active is either 0 or 1. is_active is chosen to be 1 in 85% of the cases and 0 in 15% of the cases.
- error_cntrl and data_a are randomly generated numbers between 0 and 255.
- data_b is a 5-character string that is filled with random characters created with upper case A-Z letters.

There are between 1 and 4 *Special_Facility* records per row in the *Subscriber* table, so that there are 25% subscribers with one record, 25% with two records and so on.

Call_Forwarding Table

- s_id and sf_type refer to the corresponding fields in the *Special_Facility* table.
- start_time is of type integer. It can have value 0, 8 or 16 representing midnight, 8 o'clock or 16 o'clock.
- end_time is of type integer and is start_time + N, where N is randomly generated between 1 and 8.
- numberx is a randomly generated 15 digit string.

There are between zero and 3 *Call_Forwarding* records per *Special_Facility* row, so that there are 25 % *Special_Facility* records without a *Call_Forwarding* record, 25% with one record and so on. Because start_time is part of the primary key, every record must have different start_time.

Initial Data Population

The database is always freshly populated before each benchmark run. This ensures that runs are reproducible, and that each run starts with the correct data distributions.

The *Subscriber* table acts as the main table of the benchmark. After generating a subscriber row, its child records in the other tables are generated and inserted. The number of rows in the *Subscriber* table is used to scale the population size of the other tables. For example, a TM1 with population size of 1,000,000 gives the following table cardinalities for the benchmark:

Subscriber	= 1,000,000 rows
Access_Info	≈ 2,500,000 rows
Special_Facility	≈ 2,500,000 rows
Call_Forwarding	≈ 3,750,000 rows

The population sizes used in typical TM1 runs are: 100,000, 200,000, 500,000, 1,000,000, 2,000,000 and 5,000,000 subscribers.

The attribute values are evenly distributed where appropriate. For example, the length of the time interval in Call_Forwarding is evenly distributed between [1,8] hours.

The initial data is populated using a single client. The benchmark system then waits a preset time for the target DBMS to finish any possible asynchronous tasks, like index structure construction, before continuing to the benchmark itself.

Transactions

The TM1 benchmark runs a mixture of seven (7) transactions issued by ten (10) independent clients. All the clients run the same transaction mixture with the same transaction probabilities as defined below.

Read Transactions (80%):	
GET_SUBSCRIBER_DATA	35 %
GET_NEW_DESTINATION	10 %
GET_ACCESS_DATA	35 %

Write Transactions (20%):	
UPDATE_SUBSCRIBER_DATA	2 %
UPDATE_LOCATION	14 %
INSERT_CALL_FORWARDING	2 %
DELETE_CALL_FORWARDING	2 %

Transactions may not succeed in all cases because random numbers are used to generate keys, and some of the values randomly chosen will not be present in the benchmark database. A transaction returning an acceptable non-fatal error is not considered to be a successful transaction, and thus it is not counted in the measure of Maximum Qualified Throughput.

GET_SUBSCRIBER_DATA

Retrieve one row from the SUBSCRIBER table.

```
SELECT s_id, sub_nbr,
         bit_1, bit_2, bit_3, bit_4, bit_5, bit_6, bit_7,
         bit_8, bit_9, bit_10,
         hex_1, hex_2, hex_3, hex_4, hex_5, hex_6, hex_7,
         hex_8, hex_9, hex_10,
         byte2_1, byte2_2, byte2_3, byte2_4, byte2_5,
         byte2_6, byte2_7, byte2_8, byte2_9, byte2_10,
         msc_location, vlr_location
FROM Subscriber
WHERE s_id = <s_id rnd>;
```

The search key is s_id (primary key). The value range of s_id is [1,P], where P is the size of the *Subscriber* table. All the s_id values in the range [1,P] exist in the table.

For each transaction, s_id is randomly selected from [1,P].

The probability for the transaction to succeed (i.e. a row with the random s_id exists) is **100 %**.

GET_NEW_DESTINATION

Retrieve the current call forwarding destination.

```
SELECT cf.numberx
FROM Special_Facility AS sf, Call_Forwarding AS cf
WHERE
   (sf.s_id = <s_id rnd>
         AND sf.sf_type = <sf_type rnd>
         AND sf.is_active = 1)
   AND (cf.s_id = sf.s_id
         AND cf.sf_type = sf.sf_type)
   AND (cf.start_time \<= <start_time rnd>
         AND <end_time rnd> \< cf.end_time);
```

The value range of s_id is [1,P], where P is the size of the *Subscriber* table. There are between one (1) and four (4) records (average 2.5) in the *Special_Facility* table for each value of s_id in the *Subscriber* table. There are between one (1) and three (3) records (average 1.5) in the *Call_Forwarding* table for each (s_id, sf_type) pair in the *Special_Facility* table.

For each transaction

- s_id is randomly selected from [1,P]
- sf_type is randomly selected from [1,4]
- start_time is randomly selected from {0, 8, 16}
- end_time is randomly selected from [1,24]

The probability for the transaction to succeed (i.e. a row was returned) is **23.9 %**

GET_ACCESS_DATA

Retrieve the access validation data.

```
SELECT data1, data2, data3, data4
FROM Access_Info
WHERE s_id = <s_id rnd>
   AND ai_type = <ai_type rnd>
```

The value range of s_id is [1,P], where P is the size of the *Subscriber* table. The value range of ai_type is [1,4]. There are between one (1) and four (4) rows in the *Access_Info* table for each s_id.

For each transaction

- s_id is randomly selected from [1,P]
- ai_type is randomly selected from [1,4]

The probability for the transaction to succeed (i.e. a row was returned) is **62.5%**.

UPDATE_SUBSCRIBER_DATA

Update the service profile data.

```
UPDATE Subscriber
SET bit_1 = <bit_rnd>
WHERE s_id = <s_id rnd subid>;

UPDATE Special_Facility
SET data_a = <data_a rnd>
WHERE s_id = <s_id value subid>
   AND sf_type = <sf_type rnd>;
```

The value range of s_id is [1,P], where P is the size of the *Subscriber* table. The value range of sf_type is [1,4]. There are between one (1) and four (4) rows in the *Special_Facility* table (average 2.5) for each value of s_id.

For each transaction

- s_id is randomly selected from [1,P]
- sf_type is randomly selected from [1,4]

The probability for the transaction to succeed (i.e. both updates succeed) is **62.5%**.

Note: in the transaction above the keyword *subid* is used as a parameter to carry the value of randomly generated s_id from the first update clause to the second.

UPDATE_LOCATION

Change the location.

```
UPDATE Subscriber
SET vlr_location = <vlr_location rnd>
WHERE sub_nbr = <sub_nbr rndstr>;
```

The column sub_nbr holds a string representation of the s_id number. Its value range is [1,P], where P is the size of the *Subscriber* table.

For each transaction, sub_nbr is randomly selected from its value range.

The probability for the transaction to succeed is **100%**.

INSERT_CALL_FORWARDING

Add a new call forwarding info.

```
SELECT <s_id bind subid s_id>
FROM Subscriber
WHERE sub_nbr = <sub_nbr rndstr>;

SELECT <sf_type bind sfid sf_type>
FROM Special_Facility
WHERE s_id = <s_id value subid>:

INSERT INTO Call_Forwarding
VALUES (<s_id value subid>, <sf_type rnd sf_type>,
   <start_time rnd>, <end_time rnd>, <numberx rndstr>);
```

The column sub_nbr holds a string representation of the s_id number. Its value range is [1,P], where P is the size of the *Subscriber* table. Therefore the first select statement always returns exactly one row.

There are between one (1) and four (4) records in the *Special_Facility* table for each s_id in the *Subscriber* table, Each number of records occurs with equal probability, resulting an average of 2.5 records for each s_id.

The Insert is not guaranteed to succeed because primary key conflicts are possible. Instead of retrieving one of the existing records, the benchmark uses a random sf_type value in the INSERT command. Even using an actual sf_type from the *Special_Facility* table (selected from the result set of the second SELECT) would not guarantee a successful INSERT because the start_time is generated randomly and is part of the *Call_Forwarding* table primary key.

For each transaction

- sub_nbr is randomly selected from its value range
- sf_type is randomly selected from [1,4]
- start_time is randomly selected from {0, 8, 16}
- end_time is randomly selected from [1,24]

- numberx is a string of length 15 characters. A number between [1,P] is randomly generated, converted to string representation and padded with the character zero.

The probability for a successful transaction (i.e. a row was inserted) is **31.25%**.

DELETE_CALL_FORWARDING

Remove a call forwarding info.

```
SELECT <s_id bind subid s_id>
FROM Subscriber
WHERE sub_nbr = <sub_nbr rndstr>;

DELETE FROM Call_Forwarding
WHERE s_id = <s_id value subid>
   AND sf_type = <sf_type rnd>
   AND start_time = <start_time rnd>;
```

The column sub_nbr holds a string representation of the s_id number. Its value range is [1,P], where P is the size of the *Subscriber* table. Therefore the select statement always returns exactly one row.

There are between one (1) and four (4) records in the *Special_Facility* table for each s_id in the *Subscriber* table, Each number of records occurs with equal probability, resulting an average of 2.5 records for each s_id.

There are between zero (0) and three (3) records in the *Call_Forwarding* table for each sf_type value in the *Special_Facility* table. Each number of records occurs with equal probability, resulting an average of 1.5 records for each sf_type.

For each transaction

- s_id is randomly selected from [1,P]
- sf_type is randomly selected from [1,4]
- start_time is randomly selected from {0, 8, 16}

The probability for a successful transaction (i.e. a row was deleted) is **31.25%**.

Configuration Guidelines

In order for the test results to be comparable, the target database products should be configured a maximally similar way. The following are important database system settings that should be taken into account (together with the recommended values):

Database file disk devices
: The number of disk devices used to store the database files. Recommended:1.

Log file disk devices
: The number of disk devices used to store the transaction log files. Recommended: 1 (different than the device for the database files).

Size of the shared buffer pool (database cache)
: The database cache resides in main memory and maintains database pages that are read from or written to disk. Recommended: 0.5 GB.

Checkpoint interval

The time (average) between any two consecutive checkpoints whereby all dirty buffer pages are written to disk. Recommended: 30 min.

Transaction durability level

Some products allow for different log writing modes affecting transaction durability. The strict (full) durability requires that the transaction is written to the log, synchronously, before the transaction's commit is acknowledged by the system. Another way to achieve strict durability is to write the log, synchronously, over the network to another computer, for example, in hot standby configuration. On the other hand, relaxed durability allows for asynchronous log writing (to disk or over network). Recommended: strict.

Transaction isolation level

The isolation level (defined in the SQL standard) dictates how serializable are concurrently executed transactions. The effect of the isolation level is that the higher the level, the less concurrency is allowed in the system. Recommended: repeatable reads.

Disk write-back cache

Contemporary computer disks apply a volatile on-disk buffer for data that is read or written to the disk. While this so-called write-back cache is enabled, the disk device signals that data is written although it may reside still only in the volatile cache. If a power failure happens, the cache-resident data may lost, and thus transaction durability may be compromised[1]. Recommended: write-back cache disabled.

Publishing Results

Any company can use the TM1 Benchmark internally for any purpose at all, with no restrictions. To enhance the credibility of published results, it is recommended that they either be audited or generated by an independent third party. Tests used to compare the performance of different products should be run using identical test-bed configurations. The test environment must be described in sufficient detail that a database professional could reproduce the results.

Common settings that should be included in the report of any TM1 Benchmark include the following:

- The number, size and speed of the disks. How the database data files, indexes, system catalogs, and logs are distributed over the disks.
- Total amount of machine memory, amount of memory used for the database cache.
- Number, model and speed of the CPUs.
- Hardware model description.
- Operating system name and version.
- RDBMS name and version.
- A summary of configuration parameter values, following the list presented in the previous section

[1] Some high-end devices may utilize a persistent write-back cache, whereby an on-board battery secures the data in the buffer during a power outage.

- A copy of a product's configuration file for each product tested and each identifiable configuration used.

References

[1] Toni Strandell: "Open Source Database Systems: Systems study, Performance and Scalability". Master's Thesis, University of Helsinki, Department of Computer Science, May 2003, 54 p. (http://www.cs.helsinki.fi/u/tpstrand/thesis/)

Appendix A: SQL Schema of the TM1 benchmark

```
CREATE TABLE Subscriber (
   s_id INTEGER NOT NULL PRIMARY KEY,
   sub_nbr VARCHAR(15) NOT NULL UNIQUE,
   bit_1 TINYINT,
   bit_2 TINYINT,
   bit_3 TINYINT,
   bit_4 TINYINT,
   bit_5 TINYINT,
   bit_6 TINYINT,
   bit_7 TINYINT,
   bit_8 TINYINT,
   bit_9 TINYINT,
   bit_10 TINYINT,
   hex_1 TINYINT,
   hex_2 TINYINT,
   hex_3 TINYINT,
   hex_4 TINYINT,
   hex_5 TINYINT,
   hex_6 TINYINT,
   hex_7 TINYINT,
   hex_8 TINYINT,
   hex_9 TINYINT,
   hex_10 TINYINT,
   byte2_1 SMALLINT,
   byte2_2 SMALLINT,
   byte2_3 SMALLINT,
   byte2_4 SMALLINT,
   byte2_5 SMALLINT,
   byte2_6 SMALLINT,
   byte2_7 SMALLINT,
   byte2_8 SMALLINT,
   byte2_9 SMALLINT,
   byte2_10 SMALLINT,
   msc_location INTEGER,
   vlr_location INTEGER);



CREATE TABLE Access_Info (
   s_id INTEGER NOT NULL,
   ai_type TINYINT NOT NULL,
   data1 SMALLINT,
   data2 SMALLINT,
   data3 CHAR(3),
   data4 CHAR(5),
   PRIMARY KEY(s_id, ai_type),
   FOREIGN KEY (s_id) REFERENCES Subscriber (s_id));
```

```
CREATE TABLE Special_Facility (
   s_id INTEGER NOT NULL,
   sf_type TINYINT NOT NULL,
   is_active TINYINT NOT NULL,
   error_cntrl SMALLINT,
   data_a SMALLINT,
   data_b CHAR(5),
   PRIMARY KEY (s_id, sf_type),
   FOREIGN KEY (s_id) REFERENCES Subscriber (s_id));


CREATE TABLE Call_Forwarding (
   s_id INTEGER NOT NULL,
   sf_type TINYINT NOT NULL,
   start_time TINYINT NOT NULL,
   end_time TINYINT,
   numberx VARCHAR(15),
   PRIMARY KEY (s_id, sf_type, start_time),
   FOREIGN KEY (s_id, sf_type)
         REFERENCES Special_Facility(s_id, sf_type));
```

www.solidtech.com

Solid World Headquarters

20400 Stevens Creek Blvd.
Suite 200
Cupertino, CA. 95014
USA

Tel +1 408-454-4700
Fax +1 408-454-4900

Solid EMEA Headquarters

Merimiehenkatu 36 D
FIN-00150 Helsinki
Finland

Tel +358 (0) 424 888 81
Fax +358 (0) 9 278 2877

Solid Asia Pacific Headquarters

Solid K.K.
43th Floor
The Landmark Tower Yokohama
2-2-1 Minatomirai, Nishi-ku
Yokohama 220-8143
Japan

Tel +81 (0) 45 224 2525
Fax +81 (0) 45 224 2535