6th International Conference on Applied Informatics
Eger, Hungary, January 27–31, 2004.

Database systems benchmarking

András Gábor

University of Debrecen, Institute of Informatics,
Department of Information Technology
e-mail: gabora@ inf.unideb.hu

Abstract

Database systems play a fundamental role in information technology ever since the early days. Different models were in use before the relational database model emerged and took the leading role. Today all business and most non-business applications use RDBMS software products, which became just as important as the hardware or the operating system.

The available RDBMS software products vary in many ways. They differ in prices, performance, manageability, scalability, functional capabilities, etc. Choosing the appropriate one for an information system can be a hard task. Vendors are trying to convince us about the superiority of their product. Therefore measurements are required to allow for an honest comparison. The subjects of traditional measurements have been performance, price and sometimes stability. In this paper I will give an overview of the standard traditional benchmarks that have been used.

These traditional benchmarks measure the relational capabilities of the databases. As today most products perform the most basic relational features reliably and fast enough to fulfil most application needs, the database development efforts have shifted towards providing full-featured DBMSs.

There is a need for benchmarking the new features, because their utilization saves development time, increases maintainability and evolvability and sometimes performance as well. In the second part of the paper I will enumerate such features and propose a benchmark approach to rate them.

1. Benchmark properties

Definition: A *benchmark* is a means of measuring a product or a system against a set of conditions. A benchmark has to specify:

- the target properties of the system under test (SUT) that the benchmark measures,

- the result metrics and methods of the measurements,

- the operations and their workload used in the benchmark,

- the environment conditions in which the benchmark is run.

Therefore a *database benchmark* is a benchmark measuring a *database system*. The Benchmark Handbook [Gray93] points out that a benchmark should also be scalable and portable, which is not always the case in the real world.
In order to get interpretable results, the operations and the environment model a typical scenario in an application domain. Hence, the results are relevant to that domain, and hardly applicable (if at all) for other purposes. This way we have to talk about *domain benchmarks*.

Categorizations

We can differentiate benchmarks based on their application domain, the type of the underlying DBMS and the level of standardization.
The most relevant domains are:

- OLTP (On-Line Transaction Processing)

- DSS (Decision Support System)

- General SQL

- Full text retrieval

- Geographical data

- Engineering data, CAD/CAM

- Web benchmarks

Based on the database type:

- Relational

- Object-oriented

Another categorization aspect is to consider how standard a benchmark is. We can talk about *standard*, *semi-standard* and *custom benchmarks*.

Portions of the benchmarks are run according to standardized rules. But most of the times these standard benchmarks don't fit in our application model. So instead we can either use a modified version of these benchmarks or design and implement a totally independent, custom benchmark. While standard benchmarks are easy to implement because they are already defined, the results of these might be irrelevant to our actual application scenario. On the other hand it can be hard and expensive to design a custom benchmark.
In this article we consider only standard benchmarks. Custom benchmarking is discussed in [Gray93] *Chapter 11 - Doing Your Own Benchmark*.

A third categorization perspective is whether a benchmark is *vendor-independent* or *vendor- specific*. Benchmarks in the latter category can be subcategorized based on the vendor. These benchmarks are tightly coupled to the vendor's products and are non-portable to other systems.

2. Traditional benchmarks

The first standard benchmarks appeared in the 1980's. In 1984 an effort was made by Jim Gray to create an agreement on a standard performance benchmarks. The article [ANON] defined three major tests. By that time OLTP was the most important application domain. As a result one of the three tests - *DebitCredit* - was and OLTP benchmark. It simulated transactions in an accounting system with 3 major files (branches, tellers, accounts) and a history file. The other two tests were called *Sort* (100 million records disk sort) and *Scan* (Mini batch transactions of access and modify sequentially).

The benchmark rated the tested system according to *performance*, *price*, and *price/performance*. The elapsed time gives the performance metric in TPS, the number of processed transactions per minute. The cost is the so-called five-year capital exclusive of communications lines, terminals, development and operations. The price/performance is the quotient of the previous two.
Although *DebitCredit* was very simple and non-scalable, it was based on real-world application.

The other early benchmark was developed in 1983 at the University of Wisconsin. It is referred to as the *Wisconsin* benchmark. It is a systematic benchmark on a synthetic data. It also has four files (tables), but has more transactions that aren't specific to any application domain. Thus this benchmark measures the overall performance of the database. The performance metric is the elapsed time.

Originally the benchmark was not scalable but later version resolved this problem. The description of the benchmark can be found in [Gray93] *Chapter 4 - The Wisconsin Benchmark: Past, Present, and Future.*

Transaction Processing Performance Council[1]

The database vendors started using benchmarks and publishing their results. But these results were unchecked and unreliable. Therefore 8 leading companies founded the *Transaction Processing Performance Council (TPC)* in 1988. The organization today has around 30 companies that include database vendors and hardware suppliers as well.

The organization has defined many benchmarks that have established a basis for more honest and reliable comparison of database systems as every benchmark. The definitions of the benchmarks are available to anyone freely [TPC] but only

[1] TPC and its benchmarks are trademarks of TPC. For the precise definition of the benchmarks referred to in this paper please visit `http://www.tpc.org/`

audited hence validated results can be published.
Table 1 lists the TPC benchmarks.

Benchmark name	**Application domain**	**Validity**	**Latest version**	**Performance metric(s)**
TPC-A	OLTP	1989 Nov – 1995 Jun	2.0	tpsA
TPC-B	OLTP	1990 Aug – 1995 Jun	2.0	tpsB
TPC-C	OLTP	1992 July –	02.máj	tpmC
TPC-D	DSS	1995 Apr – 1999 Jun	01.febr	QppD@Size, QthD@Size, QphD@Size
TPC-H	DSS (ad hoc)	1999 Feb –	2.1.0	QphH@Size
TPC-R	DSS (reporting)	1999 Feb –	2.1.0	QphR@Size
TPC-W	Web applications	1999 Dec –	08.jan	WIPS, WIPSb, WIPSo

Table 1 – List of TPC benchmarks

A brief summary of the above benchmarks follows here, the full specifications and the latest results are available at TPC's website `http://www.tpc.org`.

The first benchmarks TPC-A and TPC-B were both simple OLTP benchmarks based on the concepts of the DebitCredit. The main difference between them is while TPC-A results are measured at the terminal, TPC-B is run in batch mode on the server side omitting the network traffic and performing a stress test. By today both of these have become obsolete and only the TPC-C remained active evolving through many versions.

The TPC-C is an enhancement over the TPC-A and works with a more complex database model. It simulates computer environment of an order-entry application where a population of users is running transactions against the database. A user randomly chooses one out of five different transactions according to the specified distribution. User actions are simulated by specifying a mean keying time (entering data) before submitting a transaction and think time (evaluating results) after the results arrive and the user restarts its cycle.

The metrics of the benchmark are tpmC (transactions per minute TPC-C) and $/tpmC, where the price involves the hardware, software and maintanance costs

All published TPC benchmarks must adhere to a specified format. A *Full Disclosure Report* includes relevant information on the computer environment of the test and the price calculations. By studying the environment and the database settings we can find best practices for tuning the databases for the benchmark's domain.

As computational power increased and data storage became cheap Decision Support Systems (DSS) nowadays play a major role among database applications. These applications are about issuing complex queries against databases.

The first DSS TPC benchmark was TPC-D. During the benchmark complex queries are run and infrequent updates are also required. It can be run in either sequentially to measure the *Power* (in QppD@Size) or concurrently to measure the *Throughput* (in QthD@Size). The combined metric QphD@Size is called the Composite Query-Per-Hour Rating. The @Size part of the metric emphasizes that only results achieved at the same database size are meaningfully comparable. This size is referred to as the *Scale Factor*, which can be one of the following in GB: 1, 10, 30, 100, 300, 1000, 3000, 10000.
It has been soon realized that better results can be achieved if we exploit the fact that all possible queries are known in advance. Better optimizer strategies, better data placement will produce very good results that don't reflect the DBMS's capabilities to answer ad hoc queries.
This realization led to the split of the TPC-D benchmark into: TPC-H where no query-based optimizations are allowed to measure ad hoc query processing; and into TPC-R which measures the reporting capabilities as every optimization is allowed provided that it is stated in the full disclosure report.

Other benchmarks

TPC's benchmarks are reported most often, but there are also some other widely used tests, mainly for private measurements.
The *ANSI SQL Standard Scalable and Portable Benchmark* – AS^3AP – is a test that measures general performance. The benchmark uses a data mode scalable in size and requires set of fixed queries and transactions to be run. The main metric is called the *equal database size* , which is the largest database size on which the test set was able to complete in under 12 hours. When we compare two results, we divide the two results and get the *equal database ratio*.
The *Set Query* benchmark as its name implies measures the query processing power of a DBMS. The queries of the test are from three application domains: Document Search, Direct Marketing, DSS and Management Reporting. The elapsed time of each query is measured separately to identify certain weaknesses of the system. In order to compare systems a combine metric is used called *Dollar Price per Query Per Second* ($PRICE/QPS). This is a price/performance metric similar to TPC's approach.
The definition of the AS^3AP and the Set Query is available in [Gray93].

Benchmark tools

As previously stated benchmarks are available as specifications, while the implementation is up to us. This is usually very time and resource consuming procedure that we cannot afford. Fortunately there are tools available to run benchmarks.

We picked out one commercial tool, called Quest Benchmark Factory^{TM} by the Quest Software. Evaluation versions are available from `http://www.quest.com/` .

This tool

- can be used with different vendors (basically any that has ODBC support)

- implements many standard benchmarks including: TPC-C, Set Query, Wisconsin, AS^{3}AP

- can be used to design and run custom benchmarks

There are also open source tools:

- Open Source Database Benchmark available at `http://osdb.sourceforge.net/` . Available only for some database products at the moment.

- OpenLink ODBC-Bench, JDBC-Bench available from `http://www.openlinksw.com/`. We can run TPC-A and TPC-C like tests.

According to our knowledge the Quest tool is widely used in the industry. We also have good experiences.

Benchmark values

The appearance and wide usage of benchmarks was beneficial for the industry as it established accepted bases for comparing database systems and the competition for the best result urged DBMS vendors to improve their products.

But be careful when interpreting results. Keep in mind that these are achieved in pretuned conditions and don't necessarily reflect the average processing capabilities. Misleading interpretations used for marketing can undermine the importance of benchmarks and the faith in their reliability.

3. Features benchmark proposal

The available current benchmarks focus on performance only. Development issues and maintenance costs are set aside. This is the case in spite of the fact that DBMS developments today have shifted to increasing added functionality in DBMS products. Therefore we propose to define benchmarks that also take the implementation aspect into account.

Some new features are common across several products and a portion of these is even standardized by the SQL:1999 standard [SQL99]. However developers don't seem to use and accept these. The reason for this might be lack of knowledge and mistrust in the achievable application benefits. Reported benchmark results could dissolve doubt and show application scenario case studies propagating the use of better techniques.

Overview

A *features benchmark* includes:

- A modeled application scenario where the features could be used beneficially.
- An implementation using old techniques.
- Multiple implementations that utilize the sets of tested features.
- Technology scores for each feature in the implementations. The more useful a feature the more score is given. The value should be a real number greater than 1.
- Transactions and their run specifications.

When we run a test in a system, we choose an implementation. We run the transactions and measure the performance. Then we combine the performance and the technology scores into the *Adjusted Performance Result*:

$$APR = pm \cdot \prod_{f \in F} TS_f,$$

where

- pm is the performance measured (in any metric relevant to the scenario).
- F is bag[2] of features in the implementation.
- TS_f is the technology score (real number) i.e. the impact of feature f.

However it is not yet clear whether simple multiplication is the best operation when multiple features are used.

Features

Lets look at some of the available features:

1. *Collection types.* Nested relational models have been know in theory since 1983 [NFNF]. By that time DBMSs were focusing on stability. However implementations have been available in some DBMSs for a while (Oracle, Informix) and other products might include them soon. Collection attributes allow us to avoid redundancy without normalization.
2. *Object types (User-defined types), object tables.* By using object types, methods and inheritance we can easily model the real world entities. It is easier to extend the model later. Foreign keys can also be replaced by object references if needed.

[2]bag: a set where an element can occur multiple times

3. *Virtual Private Database.* A virtual private table is a table that contains a subset of rows in a larger table. For each user/group we assign a criteria that specifies which rows belong to their version of the table. The DBMS translates SQL statements issued on a virtual table into statements that contain the specified criteria. This way the protection mechanism is transparent to the user, even if he selects, deletes, etc. all rows only the rows that are accessible to him are affected. By using this technique we can use one single table to store all similar corporate data. We don't have to collect all data into one large table when we want to analyze aggregated data as it is always stored so while applications don't have to be aware of this. However handling a large table with access criterions instead of a few smaller ones with access rights probably result in a slightly poorer performance.

4. *Domain indexes.* Some applications require storage of metadata information along with data in order to make data access fast and easy. An example of this is when storing documents and along with them we store their contained words referencing the container documents. Normally the "word index" must be extracted and maintained by us. By using domain indexes we can separate the indexing routines into a reusable code. The application doesn't anymore have to maintain the index structure itself still it can exploit it in queries by including special operators in the search conditions.

5. *Table functions.* Table functions allow us to generate rows programmatically for queries used in SQL statements. These rows are totally dynamic and temporary. An example of using table functions is when we want to join a table to some temporary data. Traditionally we would store the temporary data in a temporary table for the duration of the join, while using table functions allow us to retrieving the temporary data on the fly.

6. *Aggregate clauses and analytical functions.* By using advanced aggregate clauses we can declaratively program complex aggregate queries. In analytical queries we can define sets of rows as environments of the actual row in the result set, then on these sets the analytical functions perform some aggregate computation and the result is assigned to the actual row. Sometimes it is possible to write an equivalent query without these features, but it usually is much more complex, harder to understand and modify plus it performs much worse. In other cases there is no equivalent query but we can perform the computation procedurally. These solutions can be compared.

There are many other features not listed. However the features in the above list demonstrate that there are good solutions that should be adopted by developers.

Future work

We have to fully specify benchmarks for most relevant features. Based on the experiences we can assign appropriate technology scores.

A comparison formula or method is needed that includes a custom weight factor representing the importance of performance proportional to the used technology. Find scenarios where new features are inappropriate to identify their limitations.

References

[SQL99] Jim Melton, *Advanced SQL: 1999 - Understanding Object-Relational and Other Advanced Features*, Morgan Kaufmann, 2003

[Gray93] Jim Gray ed., *The Benchmark Handbook*, Morgan Kaufmann Publishers Inc., 1993, `http://www.benchmarkresources.com`

[ANON] Anon et.al. (Jim Gray really), *A Measure of Transaction Processing Power*; Datamation, 1985, p. 112., also Tandem Technical Report TR 85.2.

[NFNF] Arisawa H., Moriya K., Miura T.: *Operations and the Properties On Non-First-Normal-Form Relational Databases*, Proceedings of 9th International Conference On Very Large Data Bases. October 1983, pp 197-204.

[Book02] Gábor András, Juhász István: *PL/SQL-programozás. Alkalmazásfejlesztés ORACLE9i-ben*, Panem, 2002, Budapest.
Title in English: *PL/SQL-programming. Application Development in ORACLE9i*

[Book03] Gábor András, Gunda Lénárd, Juhász István, Kollár Lajos, Mohai Gábor, Vágner Anikó: *Az Oracle és a WEB. Haladó Oracle9i ismeretek*, Panem, 2003, Budapest
Title in English: *Oracle and the WEB. Advanced Oracle9i*

[ORA9] Oracle9*i* Application Developer's Guide - Object-Relational Features

[TPC] The Transaction Processing Performance Council, `http://www.tpc.org`

[QUEST] Quest Software - Benchmark Factory, `http://www.quest.com`, `http://www.benchmarkfactory.com`

[OSDB] The Open Source Database Benchmark, `http://osdb.sourceforge.net`

[OpenL] Openlink ODBC Bench, JDBC Bench, `http://www.openlinksw.com`