Holistic Approach to DBMS Tuning

Ninoslav Čerkez[1], Zoran Skočir[2]
IN2 d.o.o.[1]
Savska 41/1, HR-10000 Zagreb[1]
University of Zagreb, Faculty of Electrical Engineering and Computing [2]
Unska 3, HR-10000 Zagreb[2]
Telephone: +385 1 6121650/107[1], +385 1 6129 626[2]
E-mail: ninoslav.cerkez@in2.hr, zoran.skocir@fer.hr

Abstract: **Having the right approach in tuning database management systems (DBMSs) is a crucial task in achieving fast, reliable and scalable DBMSs. In this paper a holistic approach (HA) is proposed as the way to achieve this goal, which is more efficient than some partial approach. Our experience says that it is good because it helps us and it asks us to better understand our goals, the application that we tune RDBMSs and our customers.**

1. INTRODUCTION

The common approach to the problem of tuning database systems, which is used in practice, is: leave it to the end of development. From our experience, these are some of the reasons for this approach:

- DBMSs today are designed well and they have better and better tools for tuning themselves
- Developers think that tuning a database is the database administrator's (DBA's) job
- The problem is too complex and there is no correct and final approach to that problem (if there were such an approach, it would be applied to the software application)

This approach, which we call Holistic Approach (HA), is much better and has a deeper impact on developing fast, scalable and reliable applications. It is better because it asks us to think about all the phases that are important for developing a database application: conceptual, logical and physical models of our application, concepts and features of DBMSs that we work with, architectural design of the DBMS for our application, advantages and disadvantages of each parser, methods for writing good SQL and procedural code and methods to simulate a system with mathematical tools.

The paper describes elements of the HA, with all its phases, to achieve the goal of fast, reliable and scalable DBMSs.

Section 2 gives us prerequisites that every developer should accomplish before tuning. In Section 3, performance toolkits are listed that we can use. Section 4 is dedicated to macro-level architecture decisions when we configure DBMSs. In Section 5 we have presented the way that DBMSs process statements. Section 6 considers questions about optimizers in DBMSs. Section 7 give us a method for effective schema design. Section 8 is about effective SQL and Section 9 is about effective procedural code. Section 10 shows how queuing theory can be used in performance tuning. The conclusion given in Section 11 is followed by references.

All phases are tested in a production and test environment at IN2 d.o.o. As a test environment, the Education Department infrastructure was used, and the INopis2, INsurance2, INvest2 and Public Sector project environments as production environments. Oracle DBMS (Rel.8.1.7. and 9i Rel.9.2.) was used.

2. PREREQUISITES FOR TUNING

It is incredible how many people try to tune that with which they are unfamiliar. The HA suggests that one should read the documentation when tuning a DBMS, and to do so in this order [1]: Concepts Guide, New Features Guide, Application Developers Guide, Performance Tuning Guide and Reference, Backup and Recovery Concepts and Administrators Guide.

With the Concepts and New Features guides, a developer can learn the idea of a DBMS and new features that can be used compared to the previous release. The Application Developers Guide gives instructions for making a good start in development. Performance Tuning and Administrators guides give instructions for tuning a DBMS and the core steps in DBMS administration. Backup and Recovery Concepts is useful for learning the backup and recovery process and getting helpful tips.

Test results in IN2's internal educational training during two months for a group of 6 new developers show that those who attend have better results (more than 15%) in future courses (especially SQL). The tests are repeated during a two-year period (June 2002 – June 2004) for groups of two to six employees and they were in the 15-20% range. We only excluded Backup and Recovery Concepts from the test participants, since they are specific to DBAs.

3. EFFICIENT PERFORMANCE TOOLKIT

The HA suggests learning the proper use of a performance toolkit as the next step. Today nearly everyone is using a GUI of some sort, but the truth is that command-line tools, like SQL*Plus, are still relevant and better than many GUI tools. To make sure the application is fast and as scalable as possible, toolkits should be used in the proper way. This is probably the best approach [1]:

- Single-user mode

  We should use EXPLAIN PLAN, AUTOTRACE and TKPROF

- Multi-user mode

  We should use STATPACK.

Finally, everyone should consider JDeveloper as a fast up-and-coming tool, which can effectively be used for tuning too. The future belongs to this tool [2].

Test results at IN2 are as follows:

- If an employee was a new developer and participated in internal educational training, they were able to study problems with SQL queries immediately afterwards.
- If an employee had to go on a project and had no knowledge about the required toolkit, they had constant problems in analyzing problems with slow queries, until the moment they learned to use the toolkit.

4. ARCHITECTURE DECISIONS

Topics that are relevant for achieving the sort of scalability and desired performance are [1]:

- Handling storage space effectively
- Achieving high concurrency
- Parallel operations

This is a kind of macro-level architecture decision, and the focus is not on the process architecture and memory structures but deciding when to use a given solution.

4.1. Shared Server vs. Dedicated Server Connections

The common server configuration is with a dedicated server. Shared server configuration is more complex, and it's known as a multithreaded server or MTS.

We should use a dedicated server, unless we have more concurrent connections to our database than our operating system can handle. We should consider a shared server when another connection is either not possible or will affect performance. When the system is not overloaded with processes/threads, then it would be faster with a dedicated server. With a dedicated server configuration, each client has its own process on a UNIX platform or a thread on a Windows platform. Thus, a dedicated server will receive our SQL and execute it for us (there is a one-to-one mapping between processes in the database and a client process). On most conventional database servers today (typically, machines with 2-4 CPUs), we would need 200–400 concurrent connections to even consider using a shared server instead.

Our test results show that dedicated servers are still good choice for most cases. We didn't have a situation in which we should have used a shared server. The reason is, perhaps, that we don't have such a production database in Croatia.

4.2. Partitioning

With partitioning we can achieve:

- Increased availability
  We are managing things in smaller chunks.
- Easier administration
  Operations on smaller objects are easier, faster and less resource-intensive.
- Increased performance
  Measurable performance is the hardest goal of these three.

4.3. Parallel Processing

Oracle DBMSs support a wide variety of parallel operations:

- Parallel Query
- Parallel DML
- Parallel DDL

Parallelism is just a tool and can increase or decrease processing time. It will definitely increase processing time if we use it for small problems. Thus, a parallel query is not a scalable solution, since it was initially designed so that a single user in a data warehouse could have all the resources on a machine.

In practice we didn't have good results with parallel queries in Oracle DBMS Rel. 8i. In our test cases with those databases it just ruined performance. With databases from Rel. 9i, it showed much better results.

5. HOW DBMSs PROCESS STATEMENTS?

Statement processing [3] is fundamental for deciding how to write fast and scalable statements. Since parsing is the first step in statement execution, it's worth remembering that it is the most expensive step, because it includes syntactical analysis, semantic analysis and shared pool checks. One of the main goals in statement processing is to choose soft parsing instead of hard parsing. Soft parsing greatly increases scalability because it uses previously saved results of query optimizations in the area of memory called a shared pool.

V$SQL and V$SQL_SHARED_CURSOR can give us all the information we need about statements and their chances of being soft parsed.

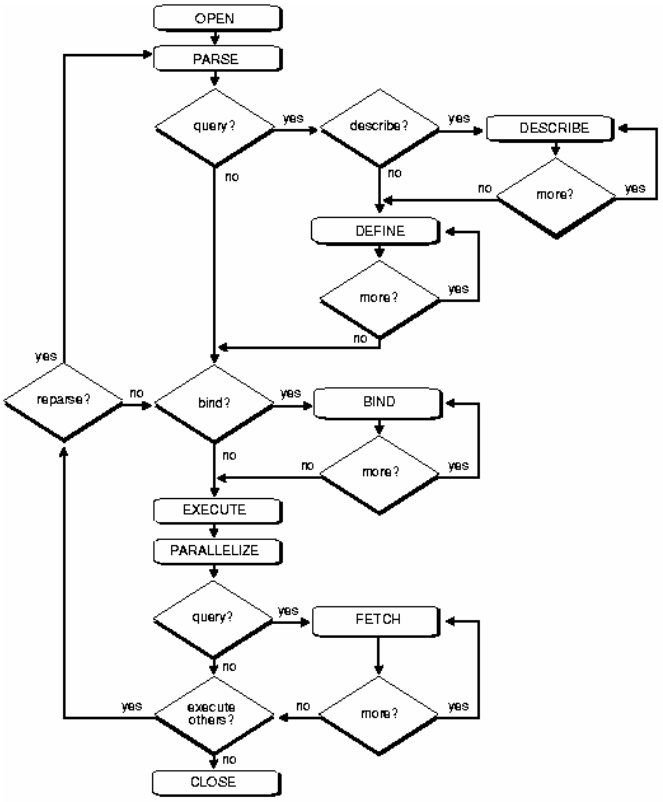

Figure 1 - Processing statement

Figure 1 shows that parsing a statement is the most important part of executing a statement. For that reason it's highly important to use bind variables properly, since their use/non-use can dramatically impact performance.

6. OPTIMIZERS

The title of this chapter will not be in plural form for much longer. The rule-based optimizer (RBO) is about to become obsolete and be replaced by the cost-based optimizer (CBO) [1].

With the RBO, query plans are generated according to a predefined set of rules, and with CBO, query plans are generated based on statistics and costs associated with performing certain operations. CBO generates most of the possible ways of processing a query, and each step in the generated query plan is assigned a cost. The query with the lowest cost „wins“, and that will be the chosen query plan. We should adjust the OPTIMIZER_INDEX_CACHING and OPTIMIZER_INDEX_COST_ADJ parameters. With default settings, we have suitable situations for reporting/warehousing systems, and with the opposite parameters we have situations for transactional or *OLTP* systems.

Our test results show the following common problems with CBO (Table 1).

Table 1 - Problems with CBO

Problem	% Cases
Work without performing analysis	35%
Analyzing with wrong data	22%
Mixing the optimizers in joins	18%
Choosing wrong index	15%
Others	10%

7. EFFICIENT SCHEMA DESIGN

Efficient data modeling is more or less a revealed secret. After conceptual modeling, the next step is schema design, and that is the topic that will be up-to-date every time, since every DBMS with every new release includes new features, which can and should be analyzed if we want to achieve the best results.

These topics are some of the most important [1]:

- **Where to put data integrity code?**

There are some developers that think it’s okay to include data integrity on the client side, but the majority are for putting data integrity into the database (and tests show this is better). It’s true that enforcing data integrity through the database makes the system slower, but the benefits are more important, and time consumption is not so relevant in this case.

- **Correct datatype**

Using an incorrect datatype can definitely and even dramatically decrease database performances. So the general approach is to use the correct datatype for all data. Date for dates, varchar2 for strings, and number for numbers. Also, we should use proper storage for every datatype and consider the business logic for all data.

- **Proper table types**

Most people know only about two types of tables: heap-organized (standard database table) and B*Tree indexes.

Oracle also offers hash-clustered, index-organized and external tables.

With hash-clustered and index-organized tables one can make queries much faster. Of course, there are some side effects, like slower insertion into these kinds of tables. However, when properly used, they will help us to have an application with a much faster response time. External tables are a present and future replacement for SQL Loader. They are more developer-friendly than SQL Loader.

Test results at IN2 are as follows:

- Data integrity code is always better when put in the database. The time benefits that we achieve when we put rules on the client side are minor. The costs of carrying out all client forms when those rules are changed are dangerous for efficiency and are hard to track. We have tried client rules with few often used forms and we gave up soon. So, our results are clear: all rules go into the database.

8. EFFICIENT SQL

SQL is very easy to learn. It has a small number of rules and a small number of reserved words. But, for professional work with SQL, there are many things to keep in mind. We think that a good approach to writing SQL is the best way to avoid problems.

8.1. Method: Incremental approach

We prefer a slightly modified incremental approach [6], which gives us control when writing queries.

This method has the following steps:

a) Take an *ER* or *UML* diagram for the tables we use in the query.
   In this way we have a visual test on the entities and attributes that correspond to the physical tables and columns we need. Then we write clauses in the following given order.

b) FROM clause
   We identify tables that hold the data we need and join them correctly. The focus is on the joins and we only add the primary-key and foreign-key columns necessary to verify that we've done the job correctly.

c) WHERE clause
   We add necessary conditions for our data.

d) GROUP BY clause
   Only after we are sure we have the correct data do we summarize it.

e) HAVING clause
   Once we have summarized the data correctly, we can eliminate any summary rows we don't want in the result set.

f) SELECT statement
   Now we add the columns we need.

g) ORDER BY clause

Also, we have to keep in mind to test all the complex parts of the query separately (subqueries, every query in a union, and raw data before applying built-in functions). It's worth knowing that this method follows the order that a DBMS uses to process a query.

8.2. Guideline for writing SQL in detail

The HA suggests using standards in every step of building applications. For starters, we think that the Oracle Custom Development Method (CDM) [7] is the best way to begin. In the end, many companies or even their project teams have their own standards, but CDM is a great standard to follow, even without any changes. It's not just used when writing SQL statements, but all other steps in developing a database application. Our favourite suggestions are:

- When there is no other way, we should use a *dummy where clause* to enable an index
- When we use the ANY, SOME or ALL operators we should be aware that the optimizer expands conditions that use these operators
- We should be aware that when we use IN/EXISTS and NOT IN/NOT EXISTS, every operator has a place where it's best-used, but there is no absolute rule when to use one over another [1] [4]
- We should always avoid implicit datatype conversion

Test results at IN2 are as follows:

- The incremental approach in writing SQL has no concurrency. All participants take it as the normal way of writing SQL after a very short period of time.
- CDM is our standard in developing IT systems, and we absolutely use its rules for writing SQL. As an addition for implementing rules, we use CDM RuleFrame [7] as an automatic framework for implementing rules.

9. EFFICIENT PROCEDURAL CODE

PL/SQL is the most productive language if we want to manipulate database data [1][7].

It has many advantages over the other procedural languages that work with databases.

- Its datatypes are SQL datatypes.

There is no conversion between its programming language types and database types, because they are the same.

- PL/SQL knows how to implicitly declare record types when we work with a query.
- PL/SQL is a portable and reusable language.
- It supports bind variables as a way to support scalability and performance.

- PL/SQL packages support overloading, encapsulation, breaking dependency chains and other features.

However, there are many things to remember when writing PL/SQL code:

- Whenever possible, it's better to use static SQL, since it's faster and it's parsed only once
- We should be aware when we use dynamic SQL, either DBMS_SQL or native SQL
- When we process a larger number of rows, we should consider using BULK processing, especially in ETL processes (extract, transform, load)
- We should be aware of the advantages and limits of REF cursors
- We should always use %TYPE and %ROWTYPE when declaring variables

Our test results are as follows:

- We use CDM standards in writing PL/SQL too, and developers who constantly use this method are less error prone and they achieve better and reliable code. Also, they can manage harder and more complex tasks because they have a proven method.

10. BEYOND PERFORMANCE: QUEUING THEORY MODEL

One of the largest benefits of using a queuing model is that it structures our thinking about response time [5]. It reveals the concrete mathematical relationship between the workload, service rate, and expectation parameters. Also, it highlights that the way to optimize the business value of a system is to consider all of these parameters.

M/M/m is a model that can be used to experiment with parameters that would be very expensive to manipulate in the real world. The first M stands for request interarrival time and is an exponentially distributed random variable. The second M stands for the service time and is also an exponentially distributed random variable. m is the number of parallel service channels inside the system, all of which are identically capable of providing service to any arriving service request.

For example, the M/M/m model gives us a concrete answer to the question: Which is better, a system with a single very fast CPU or a system with m > 1 slower CPUs? It shows that this depends on the arrival rate (λ) (Figure 2). The figure shows that a computer with a single fast CPU (bold curve) produces better response times for low arrival rates, but a computer with two slower CPUs (faint curve) produces better response times for higher arrival rates. λ_{eq} is the break-point arrival rate between two curves.μ stands for the average service rate (requests per time unit).

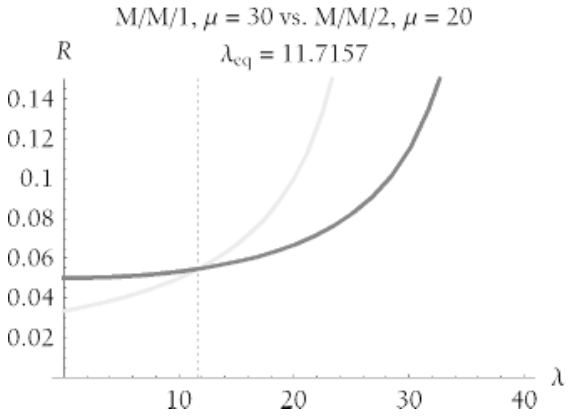

Figure 2 - Queuing theory model example

In practice that means that long nightly batch jobs run faster on the system with the single fast CPU. When a single-threaded job runs solo on a system, the arrival rate is low. It will run the fastest on a fast CPU. The multi-CPU system provides better response times to online users during their busiest work hours. Multi-CPU systems scale better to higher arrival rates than single-CPU systems.

11. CONCLUSION

Tuning database applications and DBMSs is a very complicated task and it should include the HA. One should consider all aspects of application development (data modeling, schema design, business logic, etc.), good programming skills (SQL, PL/SQL, etc.), good knowledge of DBMSs (optimizers, DBMS structure, statement processing, and performance toolkits) and finally to consider mathematical models that can save time and money. Our practical experience shows that the HA is excellent for testing either production environment. Its value is in its methodological approach which is considering all phases of database application development.

REFERENCES

[1] Thomas Kyte: "Effective Oracle by Design", McGraw-Hill/Osborne, 2003.
[2] Peter Koletzke, Dr. Paul Dorsey, Dr. Avrom Faderman: "JDeveloper Handbook", McGraw-Hill/Osborne, 2003.
[3] otn.oracle.com – Application Developer's Guide – Fundamentals Rel2 (9.2)
[4] Dan Tow: "SQL Tuning"- O'Reilly&Associates, 2004
[5] Cary Millsap:"Optimizing Oracle Performance" - O'Reilly&Associates, 2003.
[6] Jonathan Gennick: "An Incremental Approach to Developing SQL Queries"- Oracle Magazine – July/August 2000.
[7] www.oracle.com – Oracle Method, Release 6.0.0 February 2000.