


Database Performance Tuning and Query Optimization

Ernesto García Cali

Related papers

[Download a PDF Pack](#) of the best related papers 



[Foundations and Trends R in Databases](#)[Architecture of a Database System](#)

Sreenivas Chowdary

[Architecture of a Database System](#)

Chuyang Lin



Database Performance Tuning and Query Optimization

Sadhana J. Kamatkar¹(✉), Ajit Kamble¹, Amelec Vilorio²,
Lissette Hernández-Fernández², and Ernesto García Calí²

¹ University of Mumbai, Mumbai, India

sjkamatkar@mu.ac.in, ajit@ucc.mu.ac.in

² Universidad de la Costa, Barranquilla, Colombia

{avilorio7, lhernand3l, egarcia29}@cuc.edu.co

Abstract. Today, IT professionals are challenged with the task of ongoing improvements to achieve goals of businesses. Unfortunately, some factor/Resources, skill environment does not dynamically grow as fast as business needs. That sequence of events creates major obstacles for DB infrastructure, deployment, administration and maintenance. This paper discusses the performance issues, different bottlenecks such as CPU bottlenecks, Memory structures, Input output capacity issue, Database Design issues and Indexing issues. Also this paper address Tuning stages and how SQL queries can be optimized for better performance. In this paper we are focusing on query tuning tips & tricks which can be applied to gain immediate performance gain by creating Query Execution Flow Chart. We demonstrate the application of this technique in an Employee Biometric Attendance Management System.

Keywords: Database management system · RDBMS · Database tuning
Query optimization · Database performance · Optimization techniques

1 Introduction

The goal of database performance tuning is to minimize the response time of queries by making the best use of system resources. The best use of these resources involves minimizing network traffic, disk I/O, and CPU time. This goal can only be achieved by understanding the logical and physical structure of data, the applications used on system, and how the conflicting uses of database might affect performance. Although newer relational databases and faster hardware run most SQL queries with a significantly small response time, there is always room for improvement. SQL performance tuning can be an incredibly difficult task, particularly when working with large- scale data where even the most minor change can have a dramatic - positive or negative impact on performance. Since most relational databases share same design concepts under their hood, this paper is not specific to any particular vendor. Although in our example we talk about five databases, these tips apply to a wider range of RDBMS. Database Management System (DBMS) is the main source of any Organization, Institute or Company to run their business. One of the core value of any organization is customer centricity, performance tuning is highly desirable and Customers' needs to be

satisfied. Database performance tuning encompasses the steps one can take to optimize performance with the goal of maximizing the use of system resources for greater efficiency [1–4]. By fine-tuning certain database elements such as index use, query structure, data models system configuration (e.g., hardware and OS settings) and application design, one can significantly impact the overall performance of application [5–8].

2 Performance Tuning

Database Tuning is the activity of making a database application run more quickly. “More quickly” usually means higher throughput, though it may mean lower response time for time-critical applications. SQL Statements are used to retrieve data from the database. We can get same results by writing different SQL queries. But use of the best query is important when performance is considered [9].

2.1 Performance Issues

See Fig. 1.

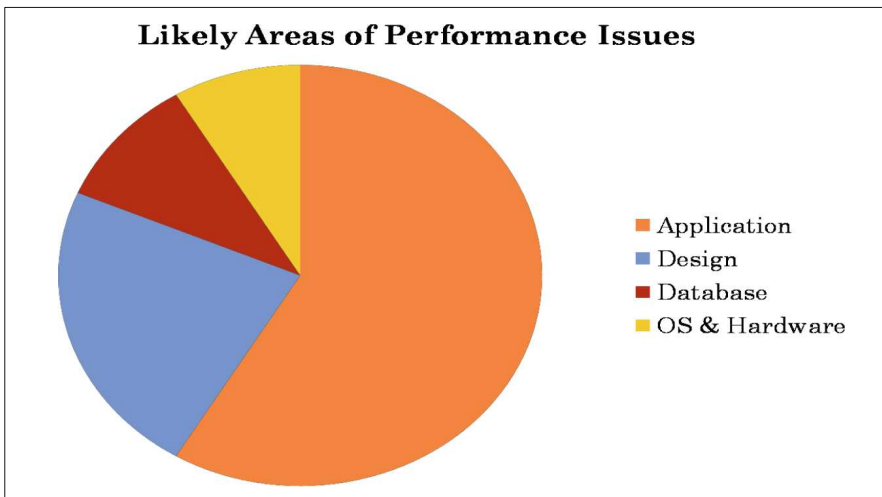


Fig. 1. Performance issues

2.2 Typical Bottlenecks

One of the first tasks in database tuning is to understand the causes of the problems and find the current bottlenecks and different factors [10].

- CPU Bottlenecks

CPU performance bottlenecks occur for a variety of reasons. They include having a non-optimal query plan, an inadequate design application or database design, poor SQL

Server configuration or a lack of hardware resources. Review the operation system CPU and processor counters for Processor Queue Length to verify that the number of threads waiting for CPU cycles is eight or less. If this number is greater than 12, it means that the CPU is causing the performance issue [11].

Once one have identified a CPU bottleneck, use sys.dm_os_wait_stats dynamic management view (DMV) to identify the top ten worst-performing queries for the CPU, as shown below.

Algorithm 1. CPU performance bottlenecks

```
SELECT TOP 10 (a.total_worker_time / a.execution_count) AS [Avg_CPU_Time]
Convert (VARCHAR, Last_Execution_Time) AS [Last_Execution_Time]
Total_Physical_Reads
SUBSTRING (b.TEXT, a.statement_start_offset / 2, (
CASE
WHEN a.statement_end_offset = - 1
THEN len(convert(NVARCHAR(max), b.TEXT)) * 2
ELSE a.statement_end_offset
END - a.statement_start_offset
) / 2) AS [Query_Text]
,dbname = Upper(db_name(b.dbid))
,b.objectid AS 'Object_ID', B.* FROM sys.dm_exec_query_stats a
CROSS APPLY sys.dm_exec_sql_text(a.sql_handle) AS b
ORDER BY [Avg_CPU_Time] DESC;
```

One can then tune these queries and the underlying indexes to resolve the CPU bottleneck. Also, configure SQL Server to use all available CPU machines. One can also scale up SQL Server system by adding additional CPUs or upgrading to a new server with more and faster CPUs.

- **Memory Structures**

Memory affects SQL Server performance more than any other piece of hardware. Therefore, it is necessary to monitor memory usage regularly on SQL Server systems to ensure that the percentage of memory available is higher than 20%. If users are experiencing performance issues and the percentage of available memory drops below 20%, then the problem is insufficient memory allocation. Observe the average page life expectancy performance counter and make sure it is always above 300 s (5 min). Anything less indicates either poor index design leading to increased disk input/output (I/O) and less effective use of memory or an actual shortage of memory. Monitor the paging rates on the SQL Server system, and make sure they are regularly above 1,000 pages per second. Typically, small OLTP transactions do not require large memory grants. Anything greater than a memory grant of zero for an OLTP transaction indicates low memory in a SQL Server system [12].

One way to handle memory bottlenecks is to find memory-intensive processes, which can be used to identify potential application problems such as memory leaks. One can also review queries to optimize performance to consume less memory. Another approach is to scale up the SQL Server environment by adding more physical

memory (RAM) to the SQL Server. Scaling up is usually a good approach to address any performance bottleneck related to memory.

- I/O Capacity issue

Compared to other hardware resources, storage input/output is usually the slowest of the system resources in SQL Server. Therefore, it is essential to investigate whether one can optimize the design and configuration of the storage system to achieve scalability and high performance, disk counters for Average Disk Sec/Read and Average Disk.

Sec/Write. Make sure that the time a read or write takes is, ideally, less than 12 ms for OLTP systems and higher for decision support systems.

As with memory, the easiest way to solve a disk I/O performance bottleneck is to scale up the SQL Server environment by replacing existing disks with faster disks that can better cope with the I/O load and that distribute the I/O load across multiple spindles. Also, defragment the data disk regularly [13, 14].

- Database Design issues

Poor database design leads to inadequate database performance. For example, the highly normalized database is associated with complex relational joins. This result in long-running queries that waste system resources such as CPU, memory and disk I/O. Thus a highly normalized database degrades SQL Server and database performance significantly. The general rule for writing efficient queries is to redesign the database if any operation requires five or more table joins.

- Indexing Issues

Indexes are the solution to many performance problems, but having too many indexes on frequently updated tables can incur additional overhead because SQL Server performs extra work to keep indexes up-to-date during insert/update/delete operations. Thus the SQL Server database engine needs more time when updating data in the table based on the number and complexity of the indexes. Also, index maintenance can increase CPU and I/O usage, which can be harmful to performance in a write-intensive system. One has to remove any duplicate and redundant indexes as they are a draining the system resources.

2.3 Tuning Stages

These tuning stages describe the phases of the software cycle and the order in which those phases are executed. Each phase produces deliverables required by the next phase in the life cycle [15–17]. Requirements are translated into design. Code is produced according to the design which is called development phase. After coding and development, the testing verifies the deliverable of the implementation phase against requirements. The management plays an important role to control and manage the system as per the guidelines, to enhance the performance of the system (Fig. 2).

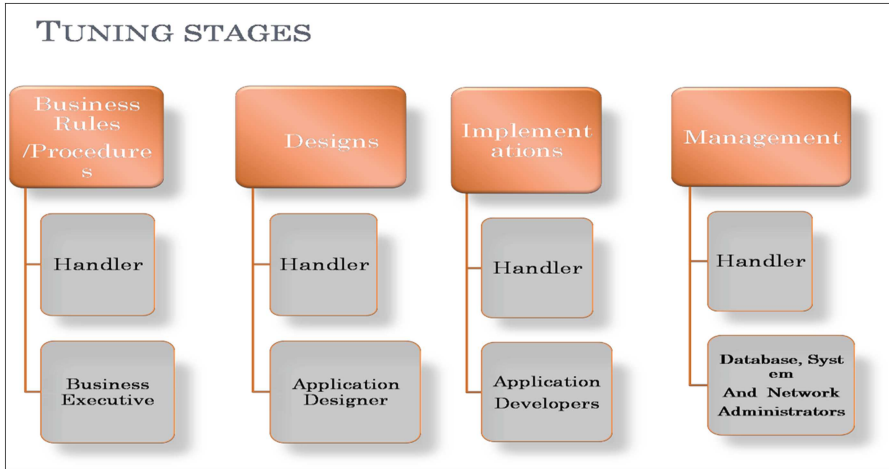


Fig. 2. Tuning stages

3 Query Optimization

Query optimization is the overall process of choosing the most efficient means of executing a SQL statement.

SQL is a nonprocedural language, so the optimizer is free to merge, reorganize, and process in any order. The database optimizes each SQL statement based on statistics collected about the accessed data. The following are the causes for bottlenecks [18].

3.1 Causes

CPU Bottlenecks, database upgrade issue, database configuration issue, poor database design, large table, indexing issue, key issue, unmanaged complex query, bad coding, and data loads consuming lots of resources and time.

3.2 Query Optimization Techniques

There are some query optimization techniques as follows [15]: The SQL query becomes faster if one uses the actual columns names in SELECT statement instead of than '*'; HAVING clause is used to filter the rows after all the rows are selected. It is just like a filter; Try to minimize the number of sub query block in one's query; Use operator EXISTS, IN and table joins appropriately in one's query; Try to use UNION ALL in place of UNION; Use DROP TABLE then CREATE TABLE instead of DELETE FROM to remove all data from a table etc.

4 Employee Biometric Attendance Management System

Mumbai University has many more application with large Database and Applications. Like “Employee Biometric Attendance Management System”. The Database of this system stores the large amount of Employee attendance data and Leave Application data. We generate many reports such as monthly report of “Employee Attendance Statistics Report” (Fig. 4). In this Report the following are required fields: employee code; employee name department name designation; total working days’ total present days early going; late coming; total working hours and total leave taken.

4.1 Challenges

The BioHRM database has 90 tables which are inter related to each other. Those Tables have large amount of employees’ attendance and leave application data. And BioHRM System has categories into two category one is Teaching Staff (Total staff-230) and another is Non-Teaching Staff (Total staff-1140). Writing the query to get employee attendance statistics with their respective leaves taken in that particular month becomes complex. It’s taking more time to produce or generate reports and consume lots of resources.

Tables_in_biohrm	
advancetype	employeeleavetravellog
applicationmenu	employeeloan
applicationmodule	employeeloaninstallment
applicationsecurity	employeeloaninstallmentlog
applicationsecuritylog	employeeloanlog
attendance	employeeelog
bloodgroup	employeeemessaging
branch	employeeemessaginglog
caste	employeeeparticulars
castelog	employeeeparticularslog
category	employeequalification
city	employeequalificationlog
citylog	employeeerelation
company	employeeerelationlog
country	employeesalary
countrylog	employeesalarydetails
department	employeesalarydetailslog
departmentlog	employeesalaryhistory
designation	employeesalaryhistorylog
designationlog	employeesalarylog
division	employeeestatus
divisionlog	employeeestatuslog
documentcategory	gender
documentcategorylog	holiday
documenttype	holidaylog
documenttypelog	language
documentupload	languagelog
documentuploadlog	leaveapplication
employee	leaveapplicationlog
employeeadvance	leavetraveltype
employeeadvancelog	leavetype
employeeattendance	leavetype log
employeeattendance_old	maritalstatus
employeeattendancelog	maritalstatuslog
employeecontact	particulartype
employeecontactlog	preference
employeeedocument	qualification
employeeedocumentlog	qualificationlog
employeeemail	relation
employeeesms	relationlog
employeeexperience	rti
employeeexperiencelog	rtilog
employeehierarchy	rtipost
	salaryhead
	salaryheadlimit
	salaryheadlimitlog

Fig. 3. BioHRM database

To overcome the above challenges, we have tried to overcome the causes of bottleneck and optimize the query execution techniques (Fig. 3).

4.2 Simplified Query Execution with Help of Flow Chart

The BioHRM database has 90 tables which are inter related to each other, therefore writing a Query becomes complex work. Employee Attendance Management System (BioHRM) requires to generate many reports such as Daily Attendance report, Report of Late Coming employees, early going employees, over time report, shift wise report, leave taken report, Leave balance report, etc. To generate these reports, it is essential to study the Entity Relation Diagram and Data Flow Diagram.

4.2.1 Entity Relation Diagram

Then it has become very easy to write the following complex query, to generate monthly Employee Attendance Statistics Report.

Algorithm 2. Employee Attendance Statistics Report

```

SELECT
original.empID,original.empCode,original.Name,original.departmentname,original.designationname
,original.desigID,original.TotalWorkingDays,original.PresentDays,original.LateComing,original.EarlyGoing,original.TotalHours,original.TotalMin,leavedata.LeavesTaken FROM
(SELECT employee.employeeid as empID, employee.employeecode as empCode,
Concat(firstname,Space(1), middlename, Space(1),lastname) as Name,
departmentname,designationname,employeehistory.designationid as desigID,
24 As TotalWorkingDays, count(employeeattendance.leavetypeid) as PresentDays ,
count(if(employeeattendance.intime1 > '10:45:00',1 ,NULL)) AS LateComing,
COUNT(if(employeeattendance.outtime1 < '17:45:00', 1 , NULL )) As EarlyGoing,
sum(minute(timediff( employeeattendance.outtime1, employeeattendance.intime1 ))/60 ) as
TotalMin, sum(hour(timediff( employeeattendance.outtime1, employeeattendance.intime1 )) ) as
Totalhours FROM employeeattendance
INNER JOIN employee ON employee.employeeid = employeeattendance.employeeid
INNER JOIN employeehistory ON employeehistory.employeehistoryid =
employeeattendance.employeehistoryid
LEFT JOIN leavetype ON employeeattendance.leavetypeid = leavetype.leavetypeid LEFT JOIN
department ON employeehistory.departmentid = department.departmentid LEFT JOIN designation
ON employeehistory.designationid = designation.designationid
WHERE employeeattendance.attendancedate between '2018-01-01 00:00:00' and '2018-01-
30 00:00:00' and employeeattendance.leavetypeid = 1
Group By employee.employeeid order by departmentname
) original
LEFT JOIN (
SELECT
employeeid,sum(floor(floor(hour(timediff(leaveapplication.todate,leaveapplication.fromdate)))/24)+
1) as LeavesTaken FROM leaveapplication WHERE fromdate between '2018-01-01 00:00:00' and
'2018-01-30 00:00:00' and
floor(floor(hour(timediff(leaveapplication.todate,leaveapplication.fromdate)))/24)+1 GROUP by
employeeid
)leavedata
ON original.empID = leavedata.employeeid

```

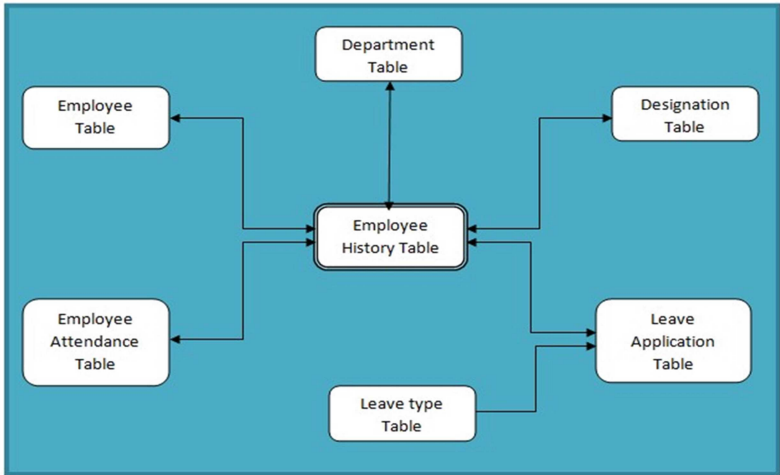



Fig. 4. Entity relation diagram

4.2.2 Data Flow Diagram

See Fig. 5.

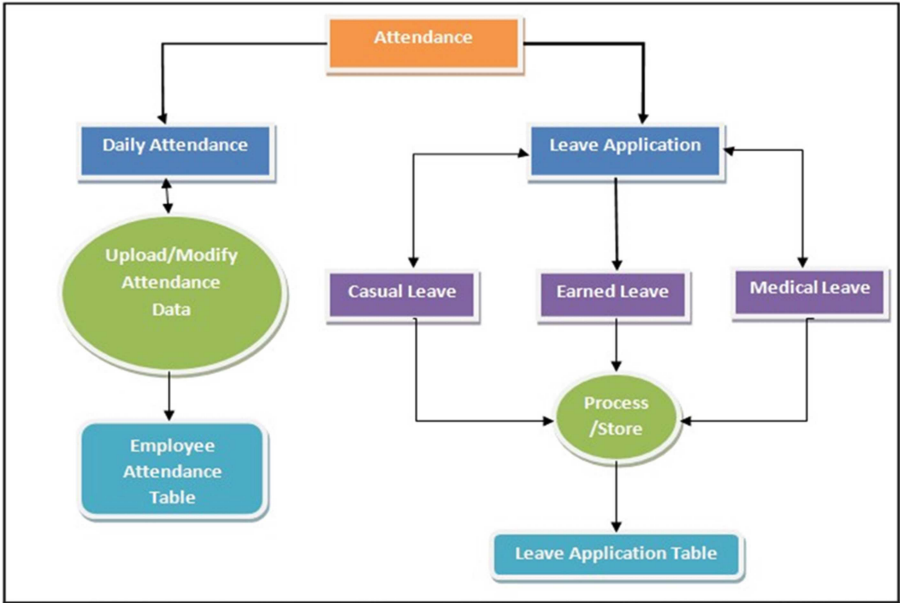


Fig. 5. Data flow diagram

5 Conclusion

There are many factors or Administration management techniques which play an important role to maintain and improve the performance and efficiency of the Systems such as - System resources availability, Integration with system, Skill Manpower and Different Techniques to give better results. Using this we achieve good performance in reliable time and improve system productivity. The purpose of this paper is to provide SQL scenarios to serve as a quick and easy reference guide during the development phase and maintenance of the database queries. This paper discusses the performance issues and different bottlenecks. This paper address Tuning stages at the different phases of the software cycle and how SQL queries can be optimized for better performance. In this paper we are focusing on performance gain by creating Query Execution Flow Chart. We have explained this technique by giving real life example of an Employee Biometric Attendance Management System.

References

1. Oracle: Oracle Database Administrator's Guide 11 g Release 2 (11.2) (2013)
2. Shasha, D.: Tuning databases for high performance. *AMC Comput. Surv.* Baltimore **28**
3. Groff, J.R., Weinberg, P.N.: *SQL: The Complete Reference*, Second Edition
4. Choudhuri, S., Narasayya, V.: Self-tuning database systems: a decade progress. *Microsoft Research* (2007)
5. Satish, S.K., Saraswatipura, M.K., Shastry, S.C.: DB2 performance enhancements using materialized query table for LUW systems. In: *ICONS 2007 Second International Conference*, April 2007
6. Wiese, D., Rabinovitch, G.: Knowledge Management in autonomic database performance tuning, 20–25 April 2009
7. Zhang, G., Chen, M., Liu, L.: A model for application-oriented database performance tuning
8. Rupley Jr., M.L.: *Introduction to Query Processing and Optimization*. Indiana University at South Bend (2008)
9. Agarwal, S., Bruno, N., Chaudhari, S.: AutoAdmin: self tuning database system technology. *IEEE Data Eng. Bull.* (2006)
10. Auto Admin: Self-Tuning database systems technology. *IEEE* (2006)
11. SQL Memory Management in Oracle9i, Hong Kong, China (2002)
12. Foundations of Automated Database Tuning. In: *VLDB 2006*, Seoul, Korea, VLDB Endowment. *ACM*, 12–15 September 2006
13. Burleson, D.: *Oracle Tuning, The Definitive Reference* Second Edition
14. Bruno, N., Chaudhuri, S.: Exploiting statistics on query expressions for optimization. In: *Proceedings of ACM SIGMOD Conference*. *ACM* (2002)
15. Top 10 performance tuning tips for relational databases. <http://web.synametrics.com/top10performancetips.htm>
16. SQL Server Optimization Tips (2017). http://santhoshgudise.weebly.com/uploads/8/5/4/7/8547208/sql_server_optimization_tips-1.doc
17. Farooq, B.: Five tips to avoid a performance bottleneck or other SQL Server snares. <http://searchsqlserver.techtarget.com/tip/Five-tips-to-avoid-a-performance-bottleneck-or-other-SQL-Server-snares>
18. Vilorio, A., Robayo, P.V.: Virtual network level of application composed IP networks connected with systems-(NETS Peer-to-Peer). *Indian J. Sci. Technol.* **9**(46) (2016)