



# High Speed Tuning for High Speed SQL

Mark Holm  
Chief Technology Officer

## SQL tuning objectives

- Optimize PEOPLE time not MACHINE resources. Focus on:
  - Response time
  - Clock time to complete batch work
- Minimize IT tuning effort
  - Root cause analysis
  - Implementing low-risk, high-impact solutions

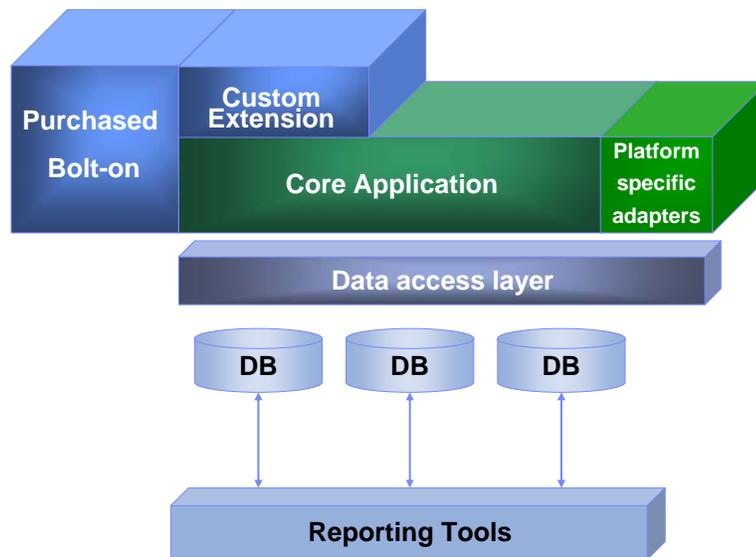


First let's talk about what our objectives should be when tuning SQL.

Historically, performance tuning has focused on reducing the amount of CPU and application used or optimizing disk I/O. While these are great goals, the economics of computing has changed such that the real cost to a business isn't the hardware it is the people using the application. As a result, the real focus of tuning today should be on the bottlenecks that slow people down and therefore on the bottlenecks that slow SQL or the application down. The bottleneck may very well be CPU or disk, but the focus should be to identify why response times are long or batch jobs don't finish fast enough.

A second goal, and this is very important, is to minimize the amount of effort required to do performance tuning for people in IT. I have yet to talk to a shop that has people sitting around twiddling their thumbs. It is a very rare IT department that feels they have as much deep expertise in performance tuning that they would like. Finally, and this is a pretty obvious point, it is important that the low risk and high impact changes be implemented first.

## Packaged Application Architecture Example



Lets take a look at a typical ERP application and how it is put together. The Core Application, the Platform specific adapters, the data access layer and the database are shipped by the vendor. Businesses then either buy bolt-ons from the original vendor or one of their partners and typically extend the application using the built-in development environment provided by the ERP vendor.

Most shops also have a multitude of reporting tools – some green screen, some pc based, some web based.

Most of the database access for the newer applications and definitely all of the graphical or web-based reporting tools is built using SQL.

The end result is a very powerful set of business and reporting functions --- one that is a challenge for IT to manage and properly tune.

## Tuning Obstacles

- The BLACK BOX problem
  - Purchased solutions that can't be changed
  - Mysterious DB2 Query Optimizer
  - Lack of an end-to-end measurement tool
- Time pressure
- Expertise



So what stands in the way of doing SQL performance tuning for many shops? First of all very few companies write all of their own software. Most purchase core business applications that are not designed to allow direct changes to the source code. Essentially the business buys what I'll call a black box application. It performs a set of functions but how those functions are actually implemented is hidden inside the box. If input goes into the box and the output takes a very long time to appear the questions is – How do I speed it up if I can't see inside the application?

A second black box, and the primary subject for today, is the database engine responsible for executing SQL requests. By its very nature, SQL lets the programmer specify WHAT they want to accomplish and not HOW to accomplish it. The responsibility for determining how to extract data or update data is delegated to DB2 and in particular the query optimizer. The query optimizer is in essence writing a program based on the SQL statement. There are many reasons that this hidden program may be inefficient and it's our job to help the query optimizer use the best possible algorithm when it writes these programs on our behalf.

A second obstacle is Time. In many cases performance problems appear suddenly and without warning. The questions is – how can we perform SQL tuning in a minimum amount of time?

The third obstacle is expertise. Becoming an SQL tuning expert involves a long learning curve. The question here is – can performance tuning be done without becoming a DB2 expert?

## The IT Catch-22

- IT inherits the responsibility to:
  - Manage the application
  - Manage the database
  - Ensure good performance
- With little or no control over:
  - Source code
  - Database access (SQL)
  - Database design

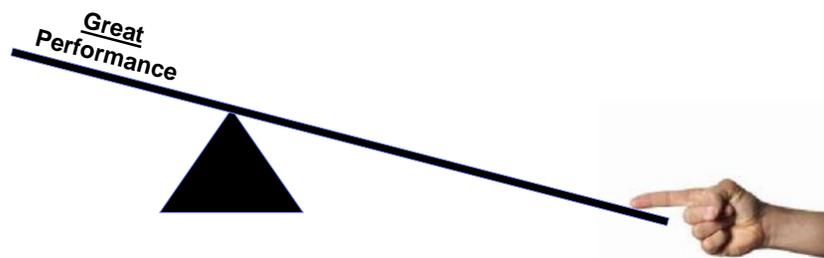


Due to the application architecture and the issues we talked about on the last slide, many IT shops find themselves in a bind. They are responsible for managing an SQL-based application and its database. They are responsible for doing backups, ensuring security, and making the application perform well.

Unfortunately, unlike when companies wrote their own software and had control over the performance sensitive parts of an application – today they do not. Source code can not be changed, SQL can not be changed, and the database can not be changed.

## Good News!

- More control exists than you may be aware of...
- Levers – the most benefit for the least effort
  - Hardware
  - Work management
  - Database changes
  - Commands and APIs
  - SQL



© 2006 Centerfield Technology

The good news is that there are a lot of ways to improve SQL performance without changing source code or SQL – more than you may think. The trick is to become aware of them and understand where they are most applicable. Today we'll go over some of the levers that you have at your disposal and explain what those levers can do for you.

These techniques fall into five categories -- where a database administrator or IT support person has some level of control. We'll focus on the items that are generally simple to try, are generally low risk, yet can provide very measurable improvement.

We'll talk about hardware and when it is appropriate to upgrade. We'll discuss work management configuration and how that influences DB2. We'll spend quite a bit of time on changes that can be made at the database level to speed up SQL. Next we'll talk about a variety of CL commands, APIs, and configuration that can be used in specific situations to boost performance. Lastly, we'll talk about some techniques to think about when you do have control over the SQL – but I'll spend the least amount of time on this topic because as we've already discussed many of you don't have a lot of control over SQL syntax.

## Lever #1 - Hardware

- Obvious choice if:
  - Cost to tune is higher than the hardware expense
  - Tuning options have been exhausted
  - Some piece of hardware will eliminate a **proven** bottleneck – CPU, memory, disk, comm bandwidth
  - Growth is expected so capacity is needed anyway
- But....
  - Don't expect miracles
  - Do your homework to make sure the right problem is being solved

The most obvious way to improve performance is to buy more hardware. This is often an attractive option because it only requires money – and who doesn't like a faster box?

In particular, if it is determined that the people-cost to tune a system is higher than the cost of the hardware it may make sense to upgrade and have those people spend their time on more productive endeavors.

You may also choose to buy hardware if you've convinced yourself that all tuning options have been explored. In reality this is rarely the case but it may be the situation.

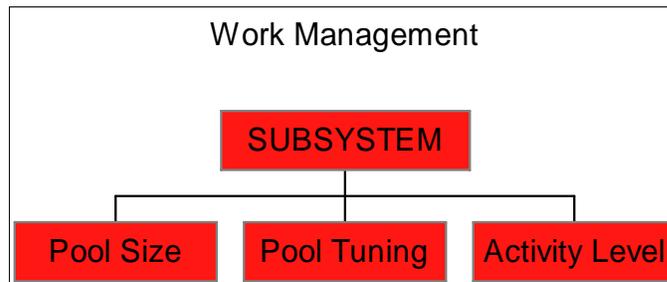
A third case is if it is known that a bottleneck would be eliminated with a focused upgrade – in processor power for a CPU bounds system for example. Obviously it would be best if the root cause of the problem was understood and it was clear the problem was not simply an inefficient SQL statement.

A fourth reason is obviously to upgrade when growth of some type is anticipated. This could be an additional application, more users, or simply keeping more historical data.

There are several caveats for hardware purchases however. The first is expectations. Poor performance can be the result of a complex set of interrelated factors and hardware may only be a partial solution. That leads me to the second point which is to be sure and buy the right hardware for the real problem. If for example, it is decided that end of month batch jobs need to finish in half the time, but they are only using 10% of a single CPU when they run it does not make sense to buy a system that is twice as fast in terms of CPU power. At the most, the batch job will only improve by 5% rather than being twice as fast. The point here is that it is very important to understand the root cause of any performance problem BEFORE you attempt to solve it – otherwise you could spend a lot of time and money for nothing.

## Lever #2 – Work Management

- Work management influences SQL performance
  - Size of pool determines which algorithms are used by the query optimizer
  - Use of Expert Cache \*CALC helps database optimization and runtime
  - Lower activity levels let SQL use more resources (less so w/SQE)

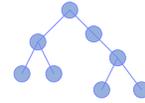


The next layer in the application stack is work management configuration. When the query optimizer takes an SQL statement and converts it into one of these black box programs, it looks at the work management configuration. The size of the pool either encourages or discourages certain approaches or algorithms from being used. The larger the memory pool, the more options the optimizer has at its disposal and therefore in theory it should be able to choose the fastest method to implement the SQL statement.

The second important factor is the use of Expert Cache. Centerfield does a service called a database/ASSESSMENT where we look at a lot of systems including their pool configuration. We find that quite a few shops have adopted the use of the \*CALC setting on their shared pools but there are still systems out there using \*FIXED. If you are using much SQL, it makes sense to use \*CALC because it not only is more friendly to database work, but it also give the optimizer additional information it can use to make decisions.

The third factor that weighs in here, is activity level. The activity level as configured influences the old query optimizer (CQE) by helping it determine what is a fair share of the pool for a particular request.

## Lever #3 – Database changes



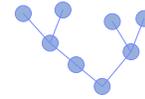
- Building indexes – Obvious ones
  - Simple selection criteria:
    - select CUSTNAME, CUSTADDR from CUSTMAST where CUSTID = ?
    - CUSTID needs an index
  - Joins
    - select ORDERHDR.ITEM, ORDERDTL.QUANTITY from ORDERHDR, ORDERDTL where ORDERHDRID = ORDERDTLID
    - Join columns need an index



## Lever #3 – Database changes

- Building indexes – Less obvious
  - Selection and ordering:
    - select CUSTNAME, CUSTADDR from CUSTMAST where CUSTID = ? ORDER BY CUSTNAME
    - CUSTID, CUSTNAME is a perfect index
  - Index only access
    - select ITEMNAME, ITEMONHAND from ITEM where ITEMID IN (?, ?, ?) ORDER BY ITEMNAME
    - Index that includes ITEMID, ITEMNAME, ITEMONHAND would reduce disk I/O and speed up the query

## Lever #3 – Database changes



- Building indexes – Less obvious
  - High selectivity indexes – Encoded Vectors
    - `select CUSTID, CUSTNAME, CUSTADDR where CUSTSTATE = 'California'` -- 40% of my customers
    - Encoded vector index over CUSTSTATE would be the best performer in this case
- Other ways to get more from indexes
  - Reorganize the data by key
  - Increase logical page size (V5R4)
  - Replace logical file index with SQL structure
  - Use DYNSTL for S/O logical files
  - Convert from 4GB to 1TB

## Lever #3 – Database changes

- Minimizing impact of indexes
  - Selective use of \*DLY and \*RBLD maintenance options
  - Identification and removal of unused indexes
  - Ensure maximum access path sharing
    - Logical file rules different than SQL
    - Same principal – don't have multiple indexes with same leading key fields
  - Use of 1TB indexes to minimize seize contention

## Lever #3 – Database changes

- Space management
  - Utilize “small” data types if possible
    - Packed instead of zoned
    - Integer types
    - Date, time, timestamp instead of packed
    - VARCHAR as appropriate
    - *Minimizes disk and memory footprint*
  - Use of ALLOCATE keyword on variable length character columns.
    - Tradeoff between space and performance. Default is BAD!
    - Performance is generally more important

## Lever #3 – Database changes

- Space management
  - Remove deleted record space
    - Deleted records take up pool space
    - Steals memory from “real” data and index pages
  - Reuse deleted records
    - Reduces/eliminates the need to reclaim space
    - May not get rid of reorganization completely if clustering is desired

## Lever #3 – Database changes

- Managed query tables (MQTs)
  - One of the newest and most promising techniques
  - Pre-calculated results automatically used by DB2
  - Can be a tremendous performance boost if:
    - The user or application can use somewhat stale data
    - The right MQTs exist
    - DB2 has been given permission to use MQTs

## Lever #4 – CL, System values, config files, APIs

- Memory caching

- SETOBJACC command

- Put data, index, or both into a memory pool
- Good tactical way to speed up reporting or batch jobs
- Must be used with CARE!

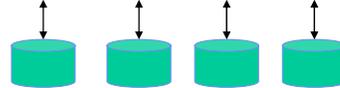
- Bring records API

- Database CPU Parallelism

- Requires DB2 SMP Feature
- Great for CPU bound queries over large files when processor resource exists
- Must be used with CARE!



Select \* from AR where TOTDUE > 5000



## Lever #4 – CL, System values, config files, APIs

Select \* from AR where TOTDUE > 5000



- QAQQINI

- IGNORE\_DERIVED\_INDEX

- Use in legacy environments with S/O logical files
    - Does not help if the query accesses logical files directly

- OPTIMIZATION\_GOAL

- Use in interactive jobs to get data back quickly (\*FIRSTIO)
    - Use in batch jobs to reduce elapsed time (\*ALLIO)

- FORCE\_JOIN\_ORDER

- Use in situations where the optimizer has the wrong order
    - Use with CARE!

## Lever #5 – SQL syntax

- Hints to the optimizer
  - OPTIMIZE FOR n ROWS
  - FETCH FIRST n ROWS ONLY
- Specify columns instead of '\*'
- Avoid leading wildcards (where NAME like '%Joe%')
- Static (compiled) SQL
  - Used ALWCPYDTA(\*OPTIMIZE)
  - Use CLOSSQLCSR(\*ENDJOB) or CLOSSQLCSR(\*ENDACTGRP)
  - Use ALWBLK(\*ALLREAD)
- Use FOR FETCH ONLY

## Summary

- Even packaged applications can be successfully tuned
- Many, many techniques exist
- The challenge is:
  - Determining the leverage points
  - Safely implementing the changes to ensure there is only positive impact to the application



## For further information or help

[www.centerfieldtechnology.com](http://www.centerfieldtechnology.com)

- Newsletter signup on home page
- FAQ: [http://www.centerfieldtechnology.com/asktheexpert\\_viewquestions.asp](http://www.centerfieldtechnology.com/asktheexpert_viewquestions.asp)
- Custom DB2 education
- E-mail: [asktheexpert@centerfieldtechnology.com](mailto:asktheexpert@centerfieldtechnology.com)

[www.ibm.com](http://www.ibm.com)

- [ibm.com/servers/enable/site/education/abstracts/indxng\\_abs.html](http://ibm.com/servers/enable/site/education/abstracts/indxng_abs.html)

### Publications

- <http://publib.boulder.ibm.com/infocenter/series/v5r3/topic/rzahg/rzahgdb.htm>
- <http://publib.boulder.ibm.com/infocenter/series/v5r4/topic/rzahg/rzahgdb.htm>



Thanks for Coming!

