25 years of missed opportunities?

*SQL Tuning Revisited*
AGENDA

1. Tuning SQL
   - How we have always done it
   - Single SQL, Package, Application...
   - Year 2004 – AP comparisons and simulation

2. Tuning SQL Revisited – A new methodology

3. Harvesting the low hanging fruit
Tuning SQL – How we have always done it

- Get an SQL from development
- EXPLAIN it
- Tune it if required
- Stage to next Level (Dev -> QA, QA -> Prod)
- Fire fight
Single SQL, Package, Application...

- Get an SQL, Package or list of Packages from development
- Fight for (and against!) Dynamic SQL
- EXPLAIN them all
- See if any have gone belly up
- Tune it if required and if you have the time
- Stage to next Level (Dev -> QA, QA -> Prod)
- Fire fight
Tuning SQL - Year 2004

- Get an SQL, Package or list of Packages from development
- Propagate Production statistics down the environment chain (Prod -> QA, Prod -> Dev)
- Simulate Hardware, ZPARMS, and BUFFERPOOLS
- Fight for (and against!) Dynamic SQL
- EXPLAIN them all
- Compare with existing Access Paths – Reject any that have got worse
- Tune it if required and if you have the time
- Stage to next Level (Dev -> QA, QA -> Prod)
- Fire fight
Tuning SQL Revisited

- Get *all* Dynamic and Static SQL running in the Plex
- Propagate Production statistics down the environment chain (Prod -> QA, Prod -> Dev)
- Simulate Hardware, ZPARMS, and BUFFERPOOLS
- EXPLAIN them all
- Compare with existing Access Paths – Tune any that have got worse
  - Pick the „low hanging fruit“
- Stage to next Level (Dev -> QA, QA -> Prod)
Tuning SQL Revisited

So how to get there?

1. Collect as much data as you can
2. Store it in a Data Warehouse
3. Analyze it
4. Take Actions!
Collect as much data as you can

- How many resources do you spend on capturing DB2 SQL workload and its metrics?

- There seems to be out-of-the-box metrics delivered by DB2, but does it give me all the data I need, when I need it?

- How does the smarter database, how does DB2 10, or 11 for z/OS deal with it?...
Collect as much data as you can

- DB2 10 Monitoring Enhancements and Changes:
  - Statement Level Statistics
    - Enhanced messages and traces to capture statement level information
  - Statement information in real-time
    - STMT_ID – unique statement identifier assigned when statement first inserted into DSC
    - Statement type – static or dynamic
    - Bind TS – 10 byte TS when stmt was bound, or prepared
  - Statement level execution statistics (per execution)
  - New Monitor class 29 for statement detail level monitoring
    - Monitor Class 29 (overhead is ~1-3%)
    - New for statement level detail
Collect as much data as you can

What’s exactly new since DB2 10:

1. IFCID 316 was enhanced to externalize the data from the Dynamic Statement Cache (DSC) when a flushing situation occurs (LRU, RUNSTATs, ALTER, DROP, REVOKE, …)
   – NO DATA LOSS

2. New IFCIDs 400* and 401 additionally EDM pool data
   – let’s call it the Static Statement Cache
   - Memory resident storage of static SQL statements
   - Like with the enhanced 316, data is externalized when the EDM pool is full. – NO DATA LOSS

*This IFCID is not really an IFCID but more of a „switch“ to enable externalization of static SQL metrics
Collect as much data as you can

DSC and EDM provide detailed workload insights:
- SQL text
- Statement ID
- Date/time
- Current status
- Resource consumption
- Identification/environmental data
Collect as much data as you can

DB2 10 also introduced some additional information from the DSC trace we all know today:

- Wait time accumulation for
  - Latch requests
  - Page latches
  - Drain locks
  - Drains during waits for claims to be released
  - Log writers
Collect as much data as you can

- Date and time in store clock format for Stmt insertion and update (along with internal format)
- Number of times that
  - a RID list overflowed because of
    - storage shortage
    - # of RIDs exceeded internal limit(s)
  - a RID list append for a hybrid join interrupted
    - because of RID pool storage shortage
    - # of RIDs exceeded internal limit(s)
  - a RID list retrieval failed for multiple IX access. The result of IX AND/OR-ing could not be determined
Collect as much data as you can
Store it in a SQL Workload Data Warehouse

Captures the hard-to-get SQLs, even the ones that disappear ...

Mainframe Engine

Workstation Engine

24 x 7 SQL Workload Capture

WLX
WLX Started Task or iterative job

DB2
DB2 DSNMSTR
System Service Address Space

Iterative Workload Processing

Select

DB2 Catalog/RTS

WLX Explain Tables

WLX Workload Warehouse Repository

Insert, Update

Type 4 Java

Graphical User Interface

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH
Store it in a SQL Workload Data Warehouse

Capture the data e.g. using a STC:
Run a started task 24x7 to catch all the IFCIDs that DB2 will be throwing and store the data.

Process the workload:
Externalize and process the data, such as every 60 min:
• customizable (e.g. 30 - 180 minutes)
• allow Ad hoc data refresh triggered via operator command for the started task (MODIFY)
• capture the SQL Text at trace time
• gather additional catalog and RTS data
• add explain data if needed
Store it in a SQL Workload Data Warehouse

Use a GUI front end, preferably Eclipse:
Exploit and integrate into Eclipse based GUI front ends

- GUIs can come as a Plug-in for
  - IBM Rational
  - IBM Data Studio
  - Eclipse native
- Existing DB2 connections are used to connect to the mainframe
- Interactive dialogs allow complex and powerful analysis
- Export features can create PDF reports and allow MS Excel hand over
- Additional plug-ins interface with other tools, such as SQL PerformanceExpert (SPX) and Bind ImpactExpert (BIX)
Store it in a SQL Workload Data Warehouse

Make the SQL Workload Warehouse Repository a set of DB2 tables that can also be created in LUW on a x86 server (E.g. DB2 Express-C).

If this is done then you can simply unload from the z/OS DB2 tables and then load the LUW Tables directly from within the GUI which enables you to run all the analytics queries “locally”.

This can obviously save a lot of space on the z/OS side!

And remember that all of the Type 4 JAVA SQL is ziiP eligible!
Analyze it

GUI features – button overview
Analyze it

Various Use Case’s drop down box
Analyze it

Example of application workload and SQL text drill down
Analyze it

Compare view:
Select any two SQLs for detailed comparison
Analyze it

Report generation dialog and selection
Harvesting the low hanging fruit

OK, so assuming you have all the data where shall we begin???

1. How about Intensive CPU?
Harvesting the low hanging fruit

The definition is simply the percentage of the CPU for a given period of time for an SQL. Here is two days of data:

As you can see one SQL executed over 20,000 times and soaked up the lion’s share of the machine! Drilling down reveals the SQL:

```sql
WHERE KA_BEARB_ZK <> '1' AND KA_BEARB_ZK <> 'L' WITH_CS
```
Harvesting the low hanging fruit

OK, so assuming you have all the data where shall we begin???

1. How about Intensive CPU?
2. What about by Application?
Harvesting the low hanging fruit

The Application definition is simply the Primary Authorization Id or the Collection/Package. Here is one snapshot of data:

<table>
<thead>
<tr>
<th>Number of Statements</th>
<th>Sum of CPU Time</th>
<th>Average CPU Time</th>
<th>Highest CPU Time</th>
<th>Sum of Elapsed Time</th>
<th>Average Elapsed Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>1,145.929112</td>
<td>28.648227</td>
<td>673.160930</td>
<td>2,171.410912</td>
<td>54.285272</td>
</tr>
<tr>
<td>6</td>
<td>122.393241</td>
<td>20.398873</td>
<td>38.379085</td>
<td>674.223872</td>
<td>112.370645</td>
</tr>
</tbody>
</table>

The average CPU is pretty high and the „highest“ is very high! Drilling on down:

Only one execution for this guy and the SQL was a pretty horrible three table join with about 20 predicates.
Harvesting the low hanging fruit

Here is a high CPU Static application:

<table>
<thead>
<tr>
<th>Primary Authorization ID</th>
<th>Number of Statements</th>
<th>Sum of CPU Time</th>
<th>Average CPU Time</th>
<th>Highest CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOXWELL</td>
<td>152</td>
<td>7.227270</td>
<td>0.008166</td>
<td>1.192778</td>
</tr>
<tr>
<td>CHRISTO</td>
<td>1</td>
<td>0.000268</td>
<td>0.000268</td>
<td>0.000268</td>
</tr>
<tr>
<td>DUDEK</td>
<td>128</td>
<td>2.177016</td>
<td>0.004380</td>
<td>0.866522</td>
</tr>
</tbody>
</table>

Drill down to Package level:

<table>
<thead>
<tr>
<th>Package or Program</th>
<th>The Collection ID</th>
<th>CPU Time</th>
<th>Elapsed Time</th>
<th>Executions</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDB2DBSG</td>
<td>RTDX0510_PFTOOL</td>
<td>32.257744</td>
<td>34.707177</td>
<td>335,463</td>
</tr>
<tr>
<td>MDB2DB02</td>
<td>MDB2VNEX_TEST</td>
<td>30.27403</td>
<td>38.114398</td>
<td>93,445</td>
</tr>
<tr>
<td>M2DBKE09</td>
<td>RTDX0510_PFTOOL</td>
<td>21.150725</td>
<td>25.142056</td>
<td>1</td>
</tr>
<tr>
<td>MDB2DB06</td>
<td>RTDX0510_PFTOOL</td>
<td>20.025189</td>
<td>22.226928</td>
<td>75,488</td>
</tr>
</tbody>
</table>
Drill down to SQL level:

```
SELECT CHAR ( SUBSTR ( DIGITS ( YEAR ( STATTIME ) ) , 9 , 2 ) CONCAT
SUBSTR ( DIGITS ( DAYOFYEAR ( STATTIME ) ) , 8 , 3 ) , 5 ) INTO : H
FROM SE_STOGROUP
WHERE NAME = : H
WITH UR
```

For every physical object a select from SYSSTOGROUP… Rewrite to a LEFT OUTER JOIN and the problem is solved!
Harvesting the low hanging fruit

OK, so assuming you have all the data where shall we begin???

1. How about Intensive CPU?
2. What about by Application?
3. Auditing???
Harvesting the low hanging fruit

Choose how you like to find out who did what and when...

Choose type of audit
- Audit
- SQL INTENTS
- Object Update Dynamic
- Show Primary Auth Ids
- SYSADM object updates
- SYSADM data updates

DCL and DDL
- Authorization failures
- GRANTs and REVOKEs (DCL)
- Changed audited tables

OK  Cancel
Harvesting the low hanging fruit

Choose how you like to find out who did what and when...
Harvesting the low hanging fruit

OK, so assuming you have all the data where shall we begin???

1. How about Intensive CPU?
2. What about by Application?
3. Auditing?
4. Disk I/O Performance?
Harvesting the low hanging fruit

Any Wait time per synchronous IO over 0.002 seconds is bad:

<table>
<thead>
<tr>
<th>Number of Statements</th>
<th>Wait time per synchronous IO</th>
<th>Synchronous IOs per statement</th>
<th>Sum of Wait Synchronous IO</th>
<th>Sum of Synchronous Buffer Reads</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.009594</td>
<td>23.000</td>
<td>0.662019</td>
<td>69</td>
</tr>
<tr>
<td>2</td>
<td>0.007095</td>
<td>1,619.000</td>
<td>22.974946</td>
<td>3,238</td>
</tr>
<tr>
<td>2</td>
<td>0.006794</td>
<td>13.000</td>
<td>0.176651</td>
<td>26</td>
</tr>
<tr>
<td>4</td>
<td>0.005586</td>
<td>377.750</td>
<td>8.441286</td>
<td>1,511</td>
</tr>
</tbody>
</table>

For OLTP transactions any with more than one Synchronous IOs per statement is “sub optimal”! Drill down shows details:

<table>
<thead>
<tr>
<th>ge</th>
<th>Wait Synchronous IO</th>
<th>Synchronous Buffer Reads</th>
<th>Synchronous Buffer Writes</th>
</tr>
</thead>
<tbody>
<tr>
<td>B2</td>
<td>0.606759</td>
<td>58</td>
<td>0</td>
</tr>
<tr>
<td>B2</td>
<td>0.030650</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>B2</td>
<td>0.024610</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>
Harvesting the low hanging fruit

OK, so assuming you have all the data where shall we begin???

1. How about Intensive CPU?
2. What about by Application?
3. Auditing?
4. Disk I/O Performance?
5. Up and Down Scaling?
Harvesting the low hanging fruit

Up and Down scaling is all about getting a “level playing field” when looking at the cache data. Simply displaying the data for SQLs that have been in the cache a week next to SQLs that have been in the cache for only 10 minutes is a bit biased!

Here you can easily see the “normal Top 10” values and the “adjusted” values. Your “Top 10” suddenly contains completely new candidates that you were *never* even aware of!

<table>
<thead>
<tr>
<th>CPU Time</th>
<th>Percentage CPU Time</th>
<th>CPU time adjusted</th>
<th>GETPAGES</th>
<th>Percentage GETPAGES</th>
<th>GETPAGES adjusted</th>
</tr>
</thead>
<tbody>
<tr>
<td>29.231328</td>
<td>1.857795</td>
<td>1.238178</td>
<td>681,568</td>
<td>4.427895</td>
<td>28,869</td>
</tr>
<tr>
<td>23.371722</td>
<td>1.485388</td>
<td>0.989977</td>
<td>593,016</td>
<td>3.852606</td>
<td>25,118</td>
</tr>
<tr>
<td>16.904098</td>
<td>1.074338</td>
<td>0.604954</td>
<td>446,936</td>
<td>2.903578</td>
<td>15,994</td>
</tr>
<tr>
<td>174.386924</td>
<td>11.083150</td>
<td>0.558840</td>
<td>1,622,158</td>
<td>10.538562</td>
<td>5,198</td>
</tr>
</tbody>
</table>
Harvesting the low hanging fruit

OK, so assuming you have all the data where shall we begin???

1. How about Intensive CPU?
2. What about by Application?
3. Auditing?
4. Disk I/O Performance?
5. Up and Down Scaling?
6. KPIs for your Enterprise?
Harvesting the low hanging fruit

Naturally all this data also lets you build up a great set of KPIs to keep track of how many, what type, and how CPU & I/O hungry everything is:

<table>
<thead>
<tr>
<th>Number of Statements</th>
<th>Total number of Dynamic...</th>
<th>Total number of Static...</th>
<th>Sum of CPU Time</th>
<th>Average CPU Time</th>
<th>Highest CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,010</td>
<td>1,698</td>
<td>312</td>
<td>4,428.794113</td>
<td>0.007277</td>
<td>425.162468</td>
</tr>
<tr>
<td>2,051</td>
<td>1,732</td>
<td>319</td>
<td>4,678.737674</td>
<td>0.007359</td>
<td>479.465437</td>
</tr>
<tr>
<td>2,415</td>
<td>2,069</td>
<td>346</td>
<td>400.380448</td>
<td>0.000923</td>
<td>27.529473</td>
</tr>
<tr>
<td>4,342</td>
<td>4,029</td>
<td>313</td>
<td>1,495.726711</td>
<td>0.004041</td>
<td>326.418810</td>
</tr>
</tbody>
</table>

Not just CPU but GetPages etc. are also available.
Harvesting the low hanging fruit

Then you can play with radar charts:
Harvesting the low hanging fruit

OK, so assuming you have all the data where shall we begin???

1. How about Intensive CPU?
2. What about by Application?
3. Auditing?
4. Disk I/O Performance?
5. Up and Down Scaling?
6. KPIs for your Enterprise?
7. Searching for SQL?
Harvesting the low hanging fruit

These days it is sometimes pretty neat to see what text is in the SQL. Currently two things spring to mind, first is CHAR9 usage and then dodgy Timestamp casting.
Harvesting the low hanging fruit

And then...

SQL Workload Expert SQL text search

<table>
<thead>
<tr>
<th>WLX Key</th>
<th>SQL type</th>
<th>Statement text</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-09-24-15.43.21.998200</td>
<td>SELECT</td>
<td><code>select * from IQA0610.BAIM_STATEMENTS where (RUNID = '2014-07-02 10:03:55.541206') FETCH FIRST 500 ROWS ONLY</code></td>
</tr>
<tr>
<td>2014-09-24-15.43.21.998200</td>
<td>SELECT</td>
<td><code>SELECT COUNT_BETTER_PROG,COUNT_WORSE_PROG,COUNT_BETTER_STMT,COUNT_WORSE_STMT FROM IQA0610...</code></td>
</tr>
</tbody>
</table>

Drill down to get a better view

```sql
SELECT COUNT_BETTER_PROG, COUNT_WORSE_PROG, COUNT_BETTER_STMT,
       COUNT_WORSE_STMT
FROM IQA0610.BAIM_RUNIDS
WHERE RUN_MODE IN ('Dyna') AND (RUNID = '2014-04-22 14:27:18.84815')
ORDER BY RUNID DESC
```
Harvesting the low hanging fruit

OK, so assuming you have all the data where shall we begin???

1. How about Intensive CPU?
2. What about by Application?
3. Auditing?
4. Disk I/O Performance?
5. Up and Down Scaling?
6. KPIs for your Enterprise?
7. Searching for SQL?
8. Flushed with success?
Harvesting the low hanging fruit

If you are catching and storing all the SQL then you can easily see how good the size and performance of your cache is:

<table>
<thead>
<tr>
<th>Number of Statements</th>
<th>Number of flushed statements per hour</th>
<th>Number of flushed statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,363</td>
<td>1,446.862</td>
<td>1,259,409</td>
</tr>
<tr>
<td>5,250</td>
<td>1,414.396</td>
<td>1,262,066</td>
</tr>
<tr>
<td>5,126</td>
<td>3,401.582</td>
<td>1,483,299</td>
</tr>
<tr>
<td>5,159</td>
<td>3,400.582</td>
<td>1,483,266</td>
</tr>
<tr>
<td>4,062</td>
<td>1,333.615</td>
<td>1,694,193</td>
</tr>
<tr>
<td>2,249</td>
<td>1,240.404</td>
<td>1,794,589</td>
</tr>
<tr>
<td>2,444</td>
<td>1,220.891</td>
<td>1,794,394</td>
</tr>
<tr>
<td>2,467</td>
<td>1,220.562</td>
<td>1,794,371</td>
</tr>
<tr>
<td>4,029</td>
<td>1,135.203</td>
<td>1,800,941</td>
</tr>
<tr>
<td>4,029</td>
<td>1,135.203</td>
<td>1,800,941</td>
</tr>
<tr>
<td>4,029</td>
<td>1,135.203</td>
<td>1,800,941</td>
</tr>
</tbody>
</table>

Rule of thumb is to make the EDMSTMTC as big as it can be! 200,000 is a good start!
Harvesting the low hanging fruit

OK, so assuming you have all the data where shall we begin???

1. How about Intensive CPU?
2. What about by Application?
3. Auditing?
4. Disk I/O Performance?
5. Up and Down Scaling?
6. KPIs for your Enterprise?
7. Searching for SQL?
8. Flushed with success?
9. Index Comparison?
Harvesting the low hanging fruit

Compare KPIs before and after Index creation. Especially twinned with Virtual Index usage this is a real winner! Did that new Index help or hinder my DB2?
Harvesting the low hanging fruit

OK, so assuming you have all the data where shall we begin???

1. How about Intensive CPU?
2. What about by Application?
3. Auditing?
4. Disk I/O Performance?
5. Up and Down Scaling?
6. KPIs for your Enterprise?
7. Searching for SQL?
8. Flushed with success?
9. Index Comparison?
10. Miscellaneous other possibilities…
Harvesting the low hanging fruit

Again, if you are catching and storing all the SQL then you can do:

- Sub-system loading checking
- Delay detection
- Object Quiet Times – Alter & Reorg
- Find all non-executed Packages - Free
- Never executed SQLs within executed Packages - Delete
- Never referenced Tables/Indexes - Drop
- Select only usage of objects – Locksize tuning
Harvesting the low hanging fruit

Why stop with just these IFCIDs? If you have a technology for high speed catching and writing why not expand it to handle:

172 – Deadlocks
196 – Timeouts
337 – Lock Escalations
359 – Index page Splits
366/376 – BIF Usage
Harvesting the low hanging fruit

BIF Usage is a major area of concern and so how do you check what is currently running in your shop?
Harvesting the low hanging fruit

BIF Usage is a major area of concern and so how do you check what is currently running in your shop?

<table>
<thead>
<tr>
<th>WLX Key</th>
<th>Collection ID</th>
<th>Package</th>
<th>ICI number</th>
<th>Count</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015-03-04-13.48.32.142444</td>
<td>DSNESPUR</td>
<td>DSNESM68</td>
<td>1</td>
<td>12</td>
<td>DB2 9 CHAR Usage</td>
</tr>
<tr>
<td>2015-03-04-13.48.32.142444</td>
<td>DSNESPUR</td>
<td>DSNESM68</td>
<td>5</td>
<td>8</td>
<td>Keyword CUBE used as unqualified UDF</td>
</tr>
</tbody>
</table>
Harvesting the low hanging fruit

Some real world numbers to amaze and astound:

- On one member of a Data Sharing group the SQLs that normally ran fast were running 45% slower than on other members. After using WLX it was discovered that this member had orders of magnitude more updates – Increase Log Buffer, Active Log, and Archive Log sizes then redirect some traffic. Et Voila!

- 450,000,000 Get pages per hour saved! -- New index created which gave a knock on performance boost effect to the whole DB2 sub-system.

- CPU Reduction from 17,111 seconds per hour to 16 seconds per hour! – One “Bad Guy” query tuned.

- Elapsed time from 30,000 seconds per hour to 30 seconds per hour! – Another single SQL “Bad Guy” query tuned.
Appendix

- DB2 APARs to check for:

  - PI07461 DB2 10 UI19041 DB2 11 UI19042 – Incons. QA0401EU, GL and GP
  - PI09147 DB2 10 UI15679 DB2 11 UI15680 – Abend S04E
  - PI09408 DB2 10 UI15740 DB2 11 UI15741 – Abend S04E
  - PI09788 DB2 11 UI15739 – SOS with IFCID400
  - PI16183 DB2 10 UI18350 DB2 11 UI18352 - Missing IFCID401
  - PI18260 DB2 11 UI20560 – QA0401EXR is not initialized
  - PI35766 OPEN Elapsed time incorrect for parallel queries
  - PI46967 OPEN Invalid IFCID 401 after IDAA APAR PI23083/PI30005
So now you know...

- Of course it is easier with **SQL WorkLoadExpert for DB2 z/OS**
  - Data Warehouse
  - Extensible and Extendable
  - Low CPU cost
- For Single SQL tuning it links to **SQLPerformanceExpert for DB2 z/OS**
  - (Also from ISPF/SPUFI etc.)
- Both work with **BindImpactExpert for DB2 z/OS** for Access Path comparison (Rebinds/Binds/APARs) and application release control (DSC Management etc.)
WLX typical use cases

Application Development:

- Application Workload Analysis: E.g. which machine load is produced by a certain Application?
- Explain Tool link (e.g. SQL PerformanceExpert, IBM DataStudio)
- Show same SQL on Multiple Schemas to detect “heavy-hitters”
- SQL Text Analysis for free text search (e.g.: BIF [Built-in Function] and UDF [User-Defined Functions] -usage, Java-formatted timestamps, etc.)
- View to detect “heavy-hitters” resulting from identical statements using different predicates
- Find unused (orphaned) SQL
WLX typical use cases

Workload/Performance management:

- Workload-Change, Problem-Detection and Trending, Comparison of CPU consumption, I/O, execution rates, current KPIs and deltas – Calculated and summarized to the costs of multiple apps
- Disc Problem Detection – I/O Rates
- SQL KPIs – Background Noise and Exceptions
- SELECT Only Table Detection (READ only activity)
- Delay Detection (All queries which are delayed)
- Up and Down Scaling of SQL Workloads
- DSC Flush Analysis
- CPU Intensive Statements
- Index Maintenance Costs
WLX functional packages of use cases

Database Administration:
- Find never used Objects (Tables, Indexes, and Tablespaces)
- Find never executed Packages and never executed SQLs within executed Packages

Audit and Security:
- AUDIT tables being accessed
- AUDIT DB2 data being accessed
- AUDIT data manipulation (insert/update/delete)
- See where updates came from (inside or outside the local network)
- See where data is being accessed from (IP address, intra-/extranet, etc.)
- SQL Text Analysis for free text search (BIF [Built-in Function] and UDF [User-Defined Functions] -usage, Java-formatted timestamps, etc.)
25 years of missed opportunities?  
*SQL Tuning Revisited*

Ulf Heinrich

SEGUS Inc  
*u.heinrich@segus.com*