25 years of missed opportunities?
SQL Tuning Revisited.

Roy Boxwell
SOFTWARE ENGINEERING GmbH

Session Code: V05
Tuesday May 5th, 1:00 to 2:00 pm | Platform: z/OS DB2
AGENDA

• Tuning SQL
  • How we have always done it
  • Single SQL, Package, Application…
  • Year 2004 – AP comparisons and simulation

• Tuning SQL Revisited – A new methodology

• 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!
WLX Architecture

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

Mainframe Engine

Workstation Engine

Graphical User Interface

24 x 7 SQL Workload Capture

WLX
WLX Started Task or iterative job

IFCID

DB2
DB2 DSNMSTR
System Service Address Space

Iterative Workload Processing

Capture processing

Select

DB2 Catalog/RTS

WLX Catalog/RTS

Explain

WLX Explain Tables

Insert, Update

WLX Workload Warehouse Repository

Type 4 Java
WLX Architecture

The Workload Warehouse Repository is 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!
WLX Architecture
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:

```
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:

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:

Drill down to Package level:

![Data Table Image]
Harvesting the low hanging fruit

Drill down to SQL level:

```sql
SELECT CHAR ( SUBSTR ( DIGITS ( YEAR ( STATSTIME ) ), 9, 2 ) CONCAT
SUBSTR ( DIGITS ( DAYOFYEAR ( STATSTIME ) ), 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...
Harvesting the low hanging fruit

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

<table>
<thead>
<tr>
<th>Transaction name</th>
<th>End User ID</th>
<th>Workstation name</th>
<th>Primary Authorization ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>db2jcc_application</td>
<td>boxwell</td>
<td>w213a07</td>
<td>BOXWELL</td>
</tr>
<tr>
<td>db2jcc_application</td>
<td>boxwell</td>
<td>192.168.1.212</td>
<td>BOXWELL</td>
</tr>
<tr>
<td>db2jcc_application</td>
<td>boxwell</td>
<td>192.168.222.106</td>
<td>BOXWELL</td>
</tr>
<tr>
<td>db2jcc_application</td>
<td>boxwell</td>
<td>192.168.222.110</td>
<td>BOXWELL</td>
</tr>
<tr>
<td>db2jcc_application</td>
<td>boxwell</td>
<td>192.168.222.115</td>
<td>BOXWELL</td>
</tr>
<tr>
<td>db2jcc_application</td>
<td>boxwell</td>
<td>192.168.222.116</td>
<td>BOXWELL</td>
</tr>
<tr>
<td>db2jcc_application</td>
<td>boxwell</td>
<td>192.168.222.124</td>
<td>BOXWELL</td>
</tr>
<tr>
<td>db2jcc_application</td>
<td>boxwell</td>
<td>192.168.222.81</td>
<td>BOXWELL</td>
</tr>
<tr>
<td>BOXWELL</td>
<td>BOXWELL</td>
<td>DB2CALL</td>
<td>BOXWELL</td>
</tr>
<tr>
<td>BOXWELL</td>
<td>BOXWELL</td>
<td>TSO</td>
<td>BOXWELL</td>
</tr>
<tr>
<td>BOXWELL$</td>
<td>BOXWELL</td>
<td>DB2CALL</td>
<td>BOXWELL</td>
</tr>
<tr>
<td>BOXWELLCC</td>
<td>BOXWELL</td>
<td>BATCH</td>
<td>BOXWELL</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?
Harvesting the low hanging fruit

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

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

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...

Drill down to get a better view

```
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(s) are:

Rule of thumb is to make the EDMSTMTTC 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?
Harvesting the low hanging fruit

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

• Both work with BindImpactExpert for DB2 z/OS for Access Path comparison and release control
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
Roy Boxwell
SOFTWARE ENGINEERING GmbH
R.Boxwell@seg.de

Session: V05
25 years of missed opportunities?
SQL Tuning Revisited

Please fill out your session evaluation before leaving!