25 years of missed opportunities?
SQL Tuning Revisited

Roy Boxwell
SOFTWARE ENGINEERING GmbH

Session Code: V05
11th November, 11:30 to 12:30 | Platform: z/OS DB2

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?

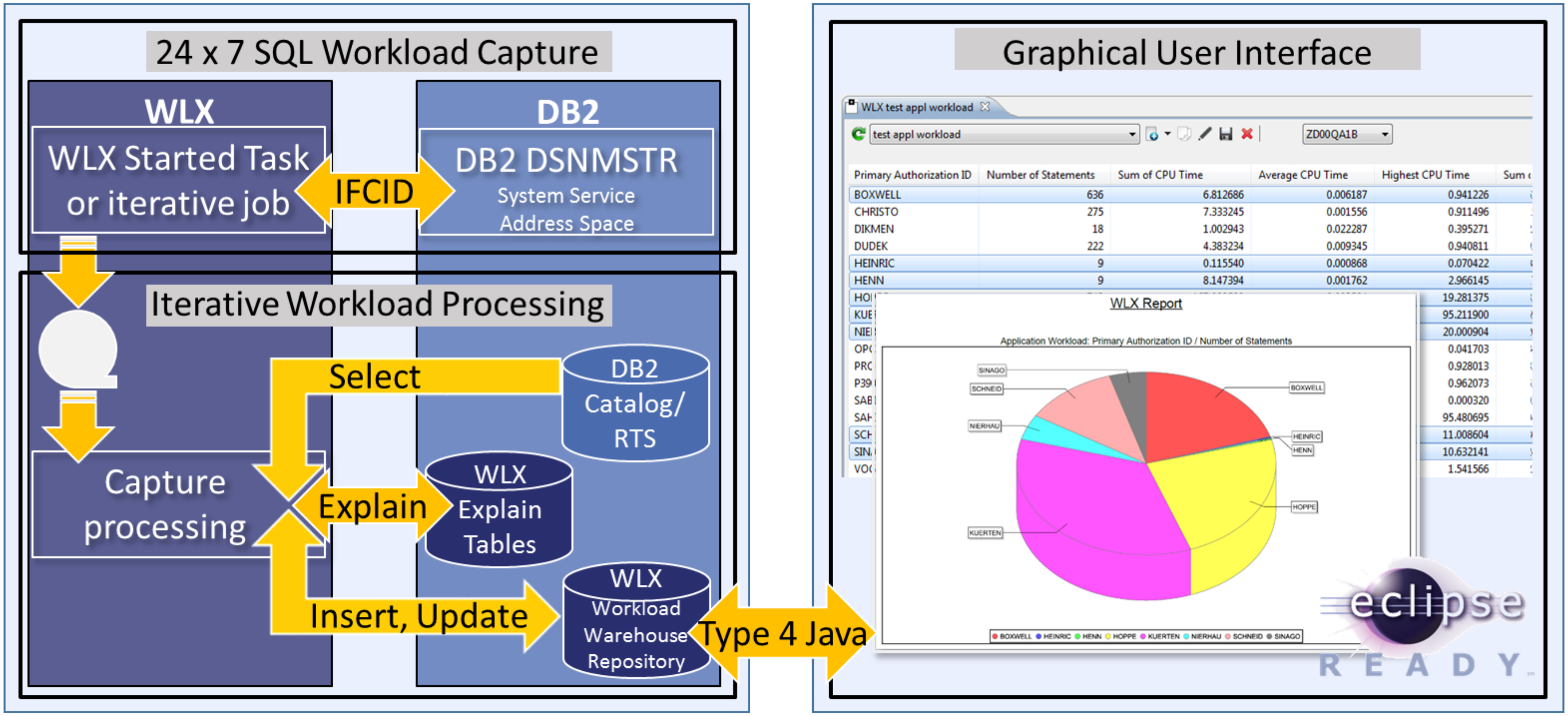
1. Collect as much data as you can
2. Store it in a Data Warehouse
3. Analyze it
4. Take Actions!

WLX Architecture

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
IFCID
DB2
DB2 DSNMSTR
System Service Address Space
Iterative Workload Processing
Select
DB2 Catalog/ RTS
Capture processing
Explain
WLX Explain Tables
Insert, Update
WLX Workload Warehouse Repository
Type 4 Java
Graphical User Interface
WLX test appl workload
test appl workload
ZD00QA1B
WLX Report
Application Workload: Primary Authorization ID / Number of Statements
SINAGO
SCHNEID
NIERHAU
KUERTEN
BOXWELL
HEINRIC
HENN
HOPPE
● BOXWELL ● HEINRIC ● HENN ● HOPPE ● KUERTEN ● NIERHAU ● SCHNEID ● SINAGO
eclipse
READY

Primary Authorization ID	Number of Statements	Sum of CPU Time	Average CPU Time	Highest CPU Time	Sum c
BOXWELL	636	6.812686	0.006187	0.941226	
CHRISTO	275	7.333245	0.001556	0.911496	
DIKMEN	18	1.002943	0.022287	0.395271	
DUDEK	222	4.383234	0.009345	0.940811	
HEINRIC	9	0.115540	0.000868	0.070422	
HENN	9	8.147394	0.001762	2.966145	
HO				19.281375	
KUE				95.211900	
NIE				20.000904	
OP				0.041703	
PR				0.928013	
P39				0.962073	
SAB				0.000320	
SAH				95.480695	
SCH				11.008604	
SIN				10.632141	
VO				1.541566	

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

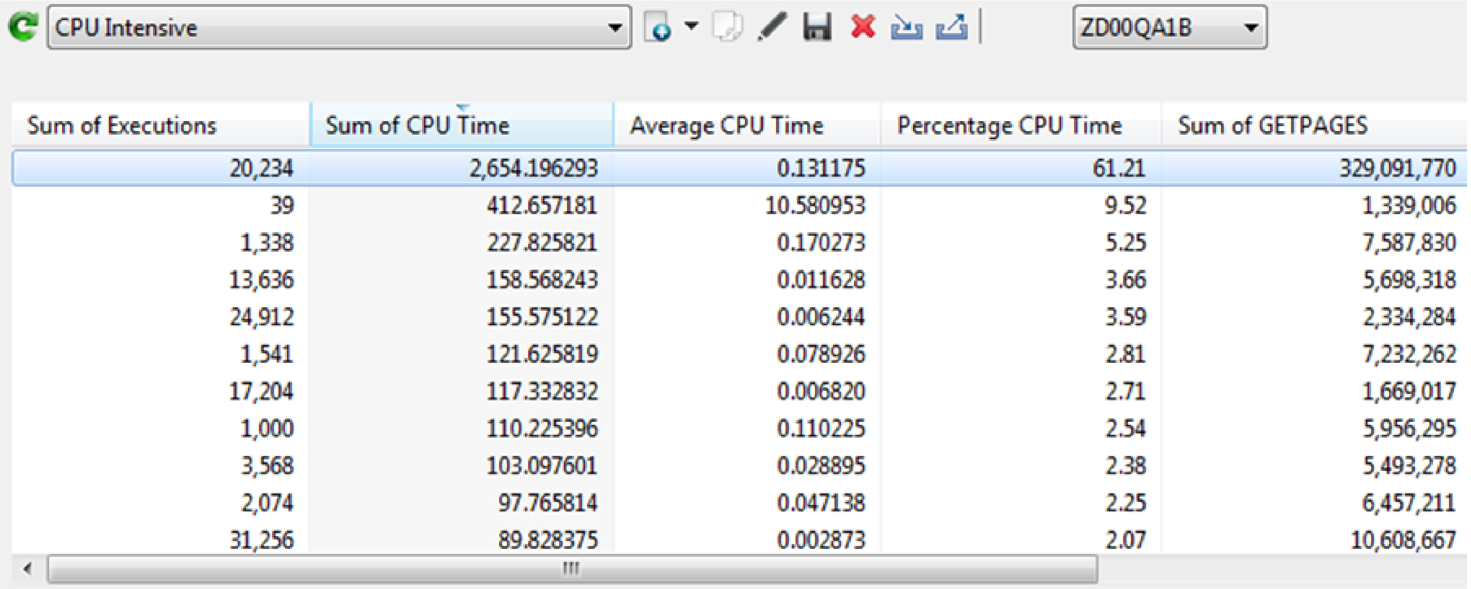
Harvesting the low hanging fruit

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:



Sum of Executions	Sum of CPU Time	Average CPU Time	Percentage CPU Time	Sum of GETPAGES
20,234	2,654.196293	0.131175	61.21	329,091,770
39	412.657181	10.580953	9.52	1,339,006
1,338	227.825821	0.170273	5.25	7,587,830
13,636	158.568243	0.011628	3.66	5,698,318
24,912	155.575122	0.006244	3.59	2,334,284
1,541	121.625819	0.078926	2.81	7,232,262
17,204	117.332832	0.006820	2.71	1,669,017
1,000	110.225396	0.110225	2.54	5,956,295
3,568	103.097601	0.028895	2.38	5,493,278
2,074	97.765814	0.047138	2.25	6,457,211
31,256	89.828375	0.002873	2.07	10,608,667

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

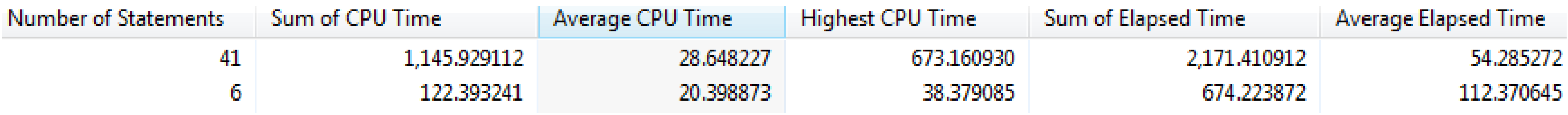
Harvesting the low hanging fruit

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

1. How about Intensive CPU?
2. What about by Application?

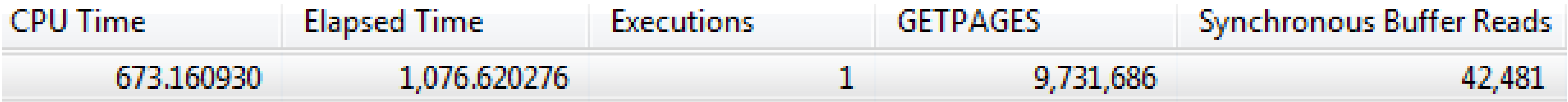
Harvesting the low hanging fruit

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



Number of Statements	Sum of CPU Time	Average CPU Time	Highest CPU Time	Sum of Elapsed Time	Average Elapsed Time
41	1,145.929112	28.648227	673.160930	2,171.410912	54.285272
6	122.393241	20.398873	38.379085	674.223872	112.370645

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

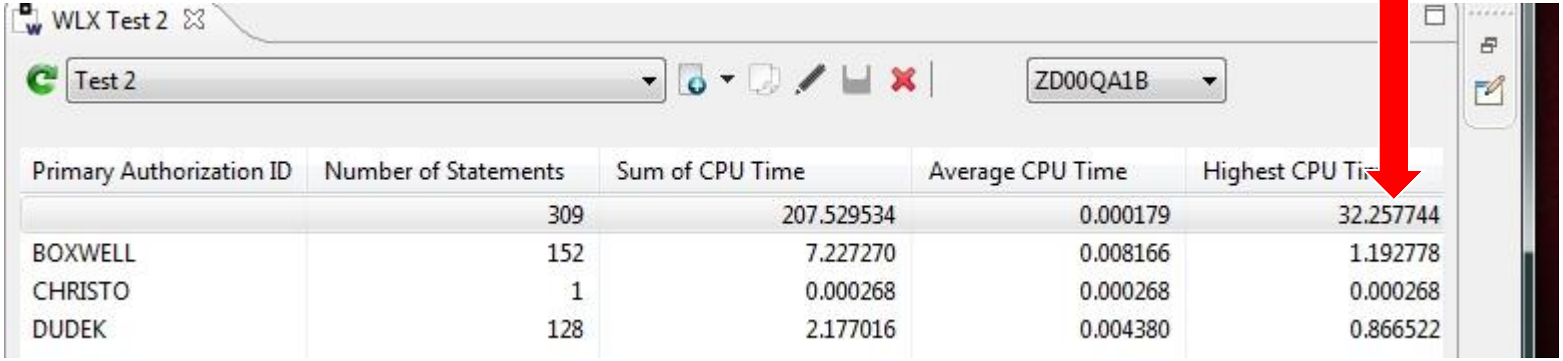


CPU Time	Elapsed Time	Executions	GETPAGES	Synchronous Buffer Reads
673.160930	1,076.620276	1	9,731,686	42,481

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

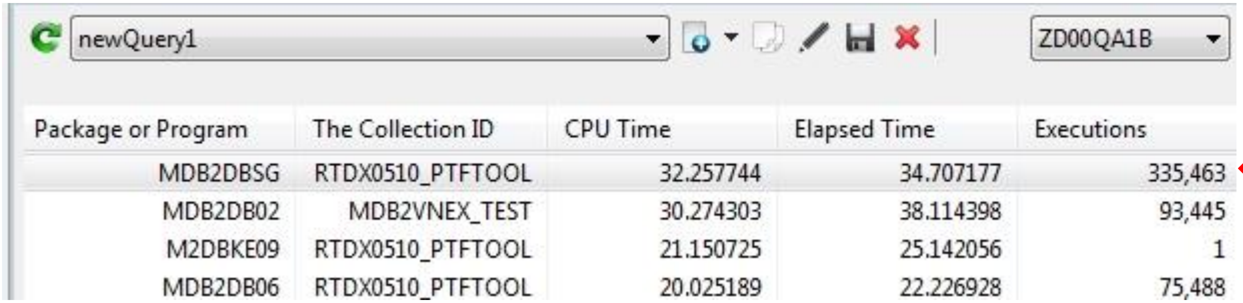
Harvesting the low hanging fruit

Here is a high CPU Static application:



Primary Authorization ID	Number of Statements	Sum of CPU Time	Average CPU Time	Highest CPU Ti
	309	207.529534	0.000179	32.257744
BOXWELL	152	7.227270	0.008166	1.192778
CHRISTO	1	0.000268	0.000268	0.000268
DUDEK	128	2.177016	0.004380	0.866522

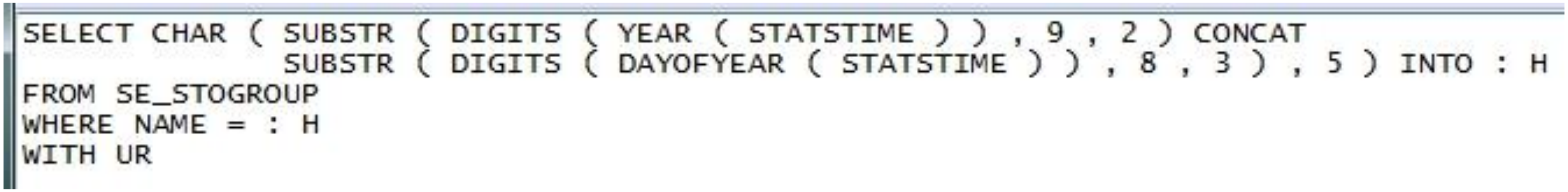
Drill down to Package level:



Package or Program	The Collection ID	CPU Time	Elapsed Time	Executions
MDB2DBSG	RTDX0510_PTFTOOL	32.257744	34.707177	335,463
MDB2DB02	MDB2VNEX_TEST	30.274303	38.114398	93,445
M2DBKE09	RTDX0510_PTFTOOL	21.150725	25.142056	1
MDB2DB06	RTDX0510_PTFTOOL	20.025189	22.226928	75,488

Harvesting the low hanging fruit

Drill down to SQL level:



```
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!

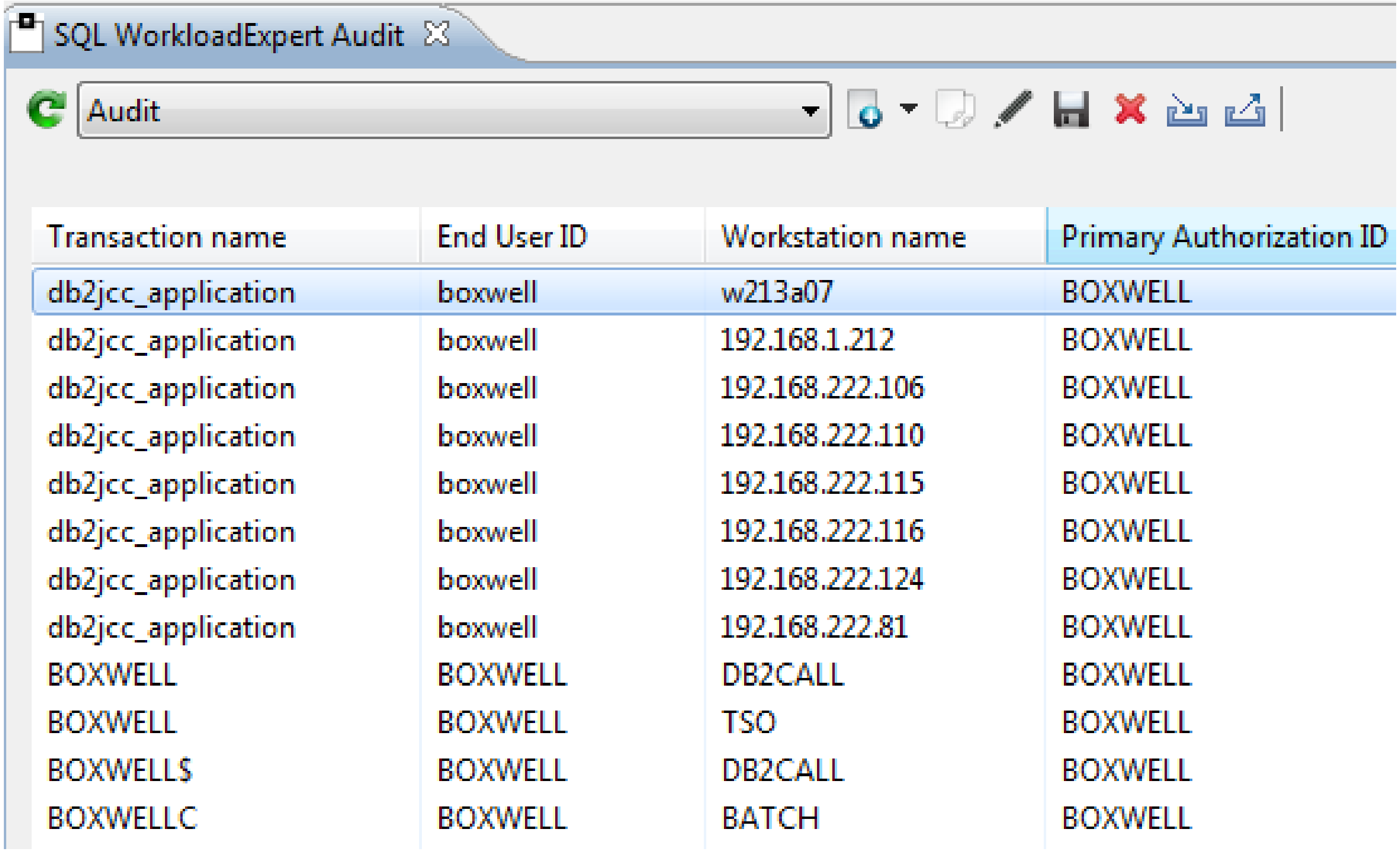
Harvesting the low hanging fruit

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…



Transaction name	End User ID	Workstation name	Primary Authorization ID
db2jcc_application	boxwell	w213a07	BOXWELL
db2jcc_application	boxwell	192.168.1.212	BOXWELL
db2jcc_application	boxwell	192.168.222.106	BOXWELL
db2jcc_application	boxwell	192.168.222.110	BOXWELL
db2jcc_application	boxwell	192.168.222.115	BOXWELL
db2jcc_application	boxwell	192.168.222.116	BOXWELL
db2jcc_application	boxwell	192.168.222.124	BOXWELL
db2jcc_application	boxwell	192.168.222.81	BOXWELL
BOXWELL	BOXWELL	DB2CALL	BOXWELL
BOXWELL	BOXWELL	TSO	BOXWELL
BOXWELL$	BOXWELL	DB2CALL	BOXWELL
BOXWELLC	BOXWELL	BATCH	BOXWELL

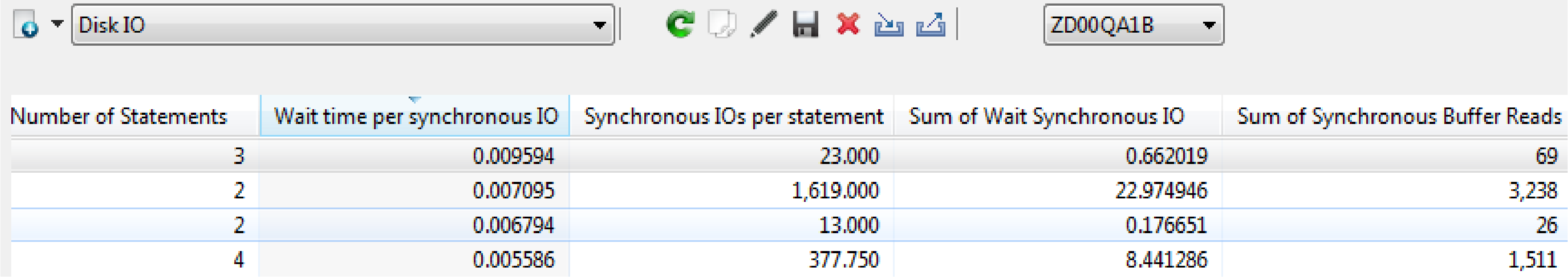
Harvesting the low hanging fruit

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?

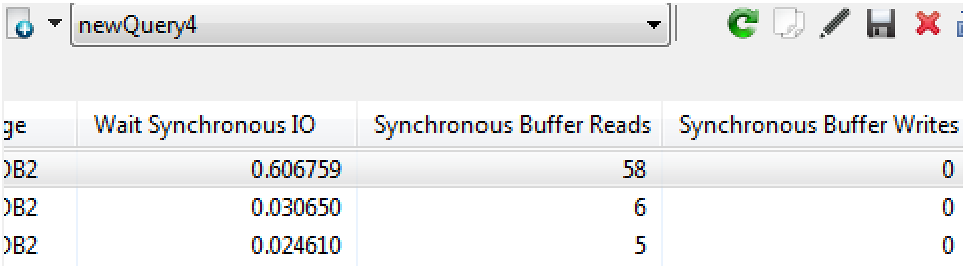
Harvesting the low hanging fruit

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



Number of Statements	Wait time per synchronous IO	Synchronous IOs per statement	Sum of Wait Synchronous IO	Sum of Synchronous Buffer Reads
3	0.009594	23.000	0.662019	69
2	0.007095	1,619.000	22.974946	3,238
2	0.006794	13.000	0.176651	26
4	0.005586	377.750	8.441286	1,511

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



ge	Wait Synchronous IO	Synchronous Buffer Reads	Synchronous Buffer Writes
)B2	0.606759	58	0
)B2	0.030650	6	0
)B2	0.024610	5	0

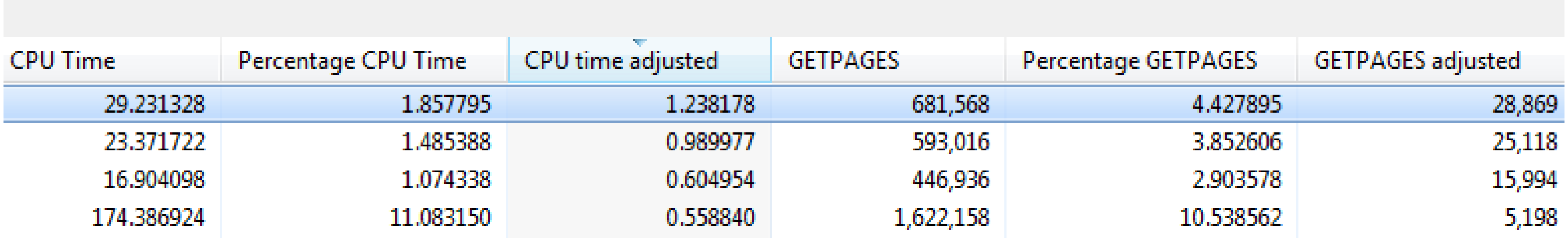
Harvesting the low hanging fruit

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!



CPU Time	Percentage CPU Time	CPU time adjusted	GETPAGES	Percentage GETPAGES	GETPAGES adjusted
29.231328	1.857795	1.238178	681,568	4.427895	28,869
23.371722	1.485388	0.989977	593,016	3.852606	25,118
16.904098	1.074338	0.604954	446,936	2.903578	15,994
174.386924	11.083150	0.558840	1,622,158	10.538562	5,198

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!

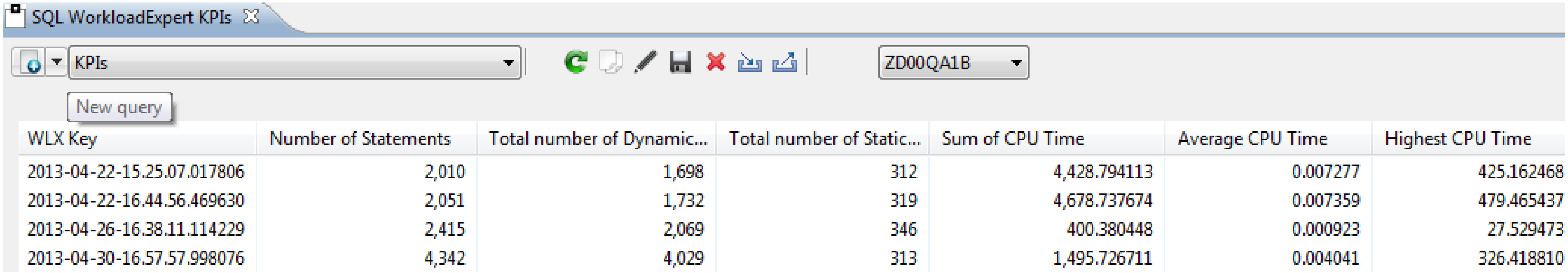
Harvesting the low hanging fruit

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:

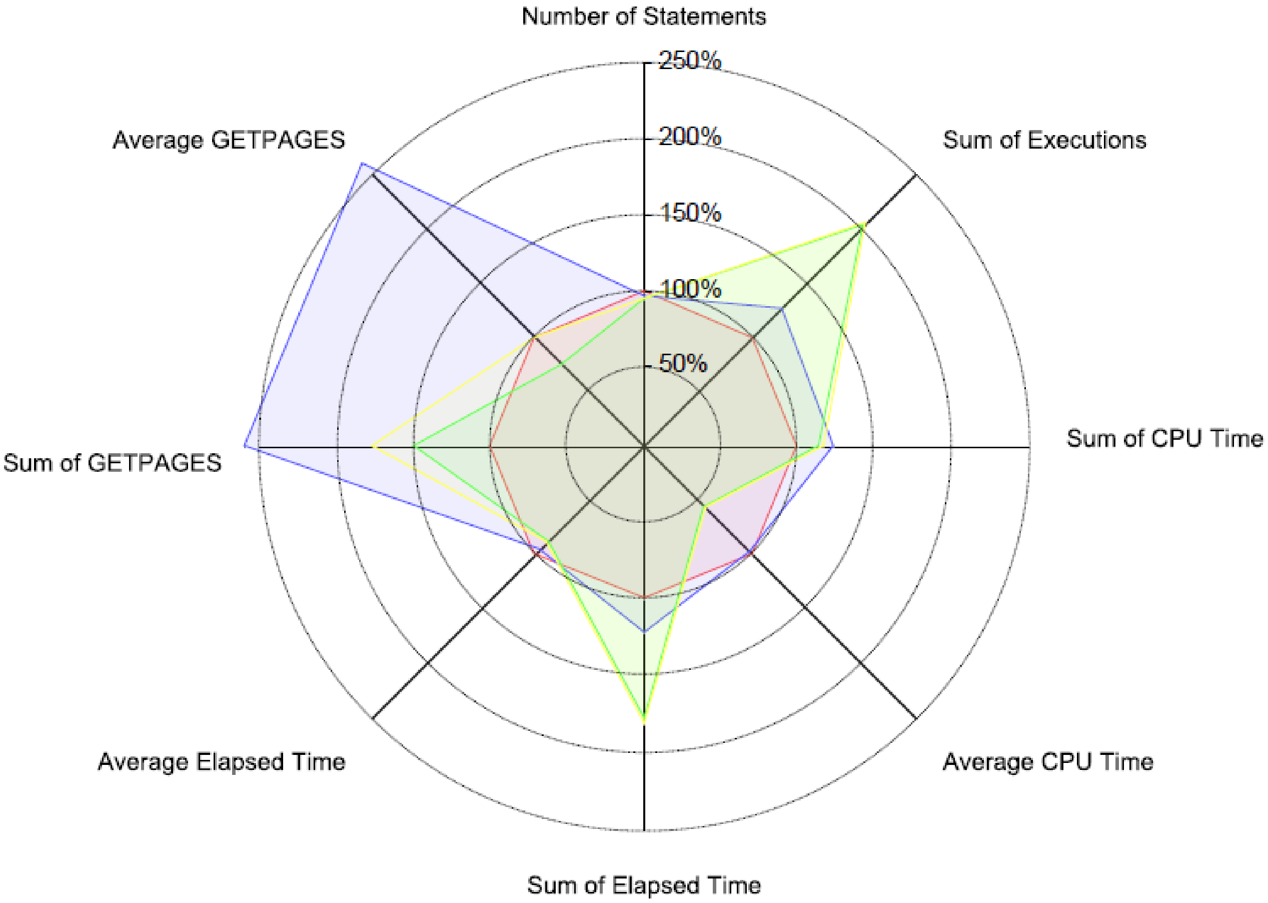


WLX Key	Number of Statements	Total number of Dynamic...	Total number of Static...	Sum of CPU Time	Average CPU Time	Highest CPU Time
2013-04-22-15.25.07.017806	2,010	1,698	312	4,428.794113	0.007277	425.162468
2013-04-22-16.44.56.469630	2,051	1,732	319	4,678.737674	0.007359	479.465437
2013-04-26-16.38.11.114229	2,415	2,069	346	400.380448	0.000923	27.529473
2013-04-30-16.57.57.998076	4,342	4,029	313	1,495.726711	0.004041	326.418810

Not just CPU but GetPages etc. are also available.

Harvesting the low hanging fruit

Then you can play with radar charts:



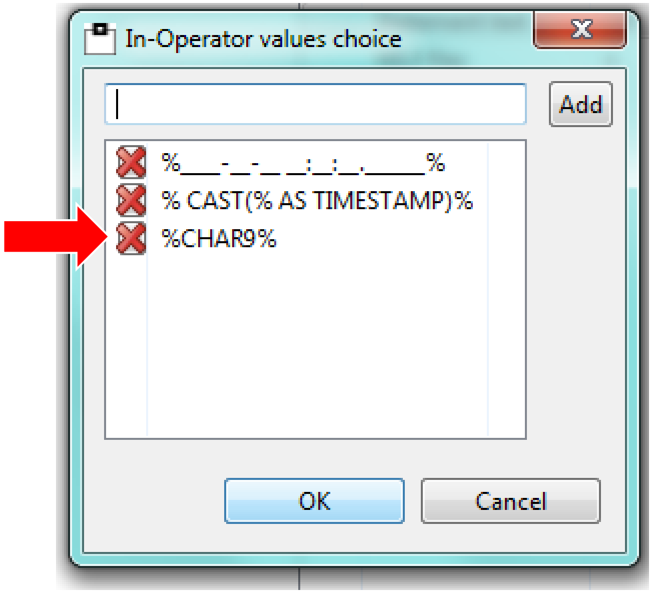
Harvesting the low hanging fruit

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.



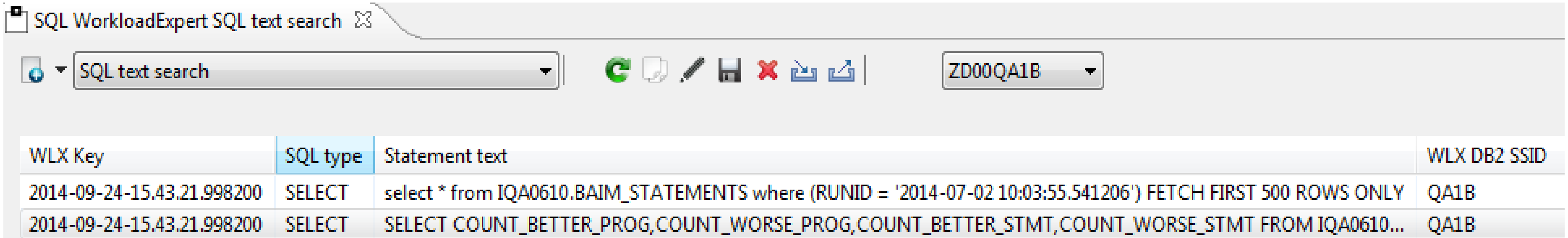
%____-__-__ __:__:__.______%

% CAST(% AS TIMESTAMP)%

%CHAR9%

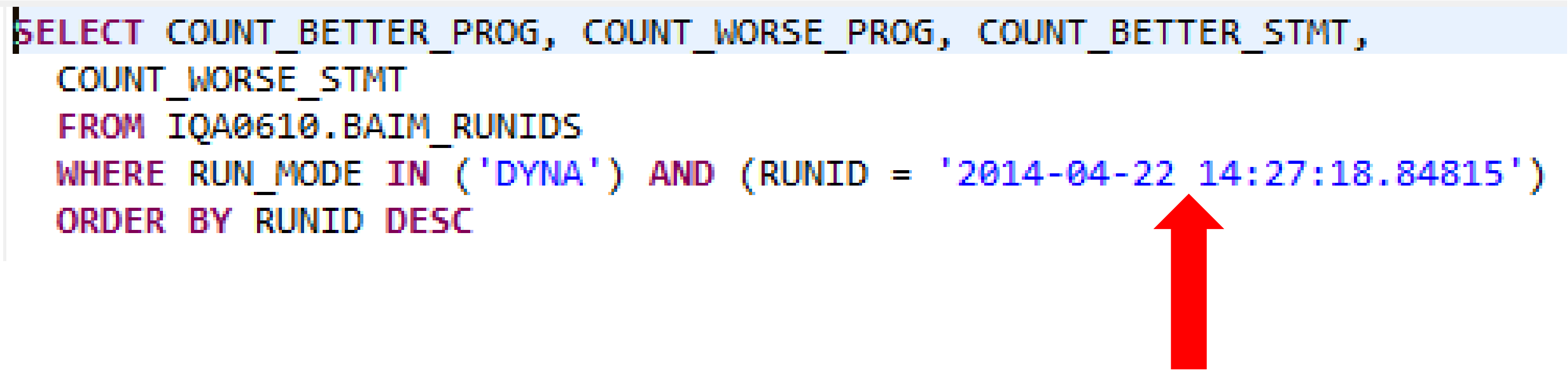
Harvesting the low hanging fruit

And then…



WLX Key	SQL type	Statement text	WLX DB2 SSID
2014-09-24-15.43.21.998200	SELECT	select * from IQA0610.BAIM_STATEMENTS where (RUNID = '2014-07-02 10:03:55.541206') FETCH FIRST 500 ROWS ONLY	QA1B
2014-09-24-15.43.21.998200	SELECT	SELECT COUNT_BETTER_PROG,COUNT_WORSE_PROG,COUNT_BETTER_STMT,COUNT_WORSE_STMT FROM IQA0610...	QA1B

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
```

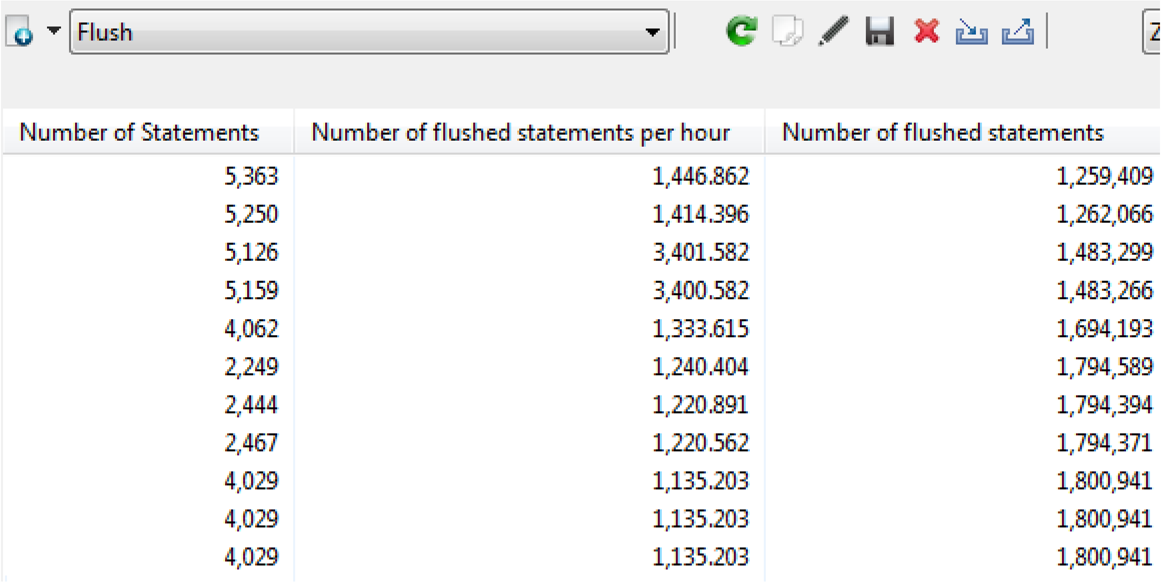
Harvesting the low hanging fruit

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 is:



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

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???

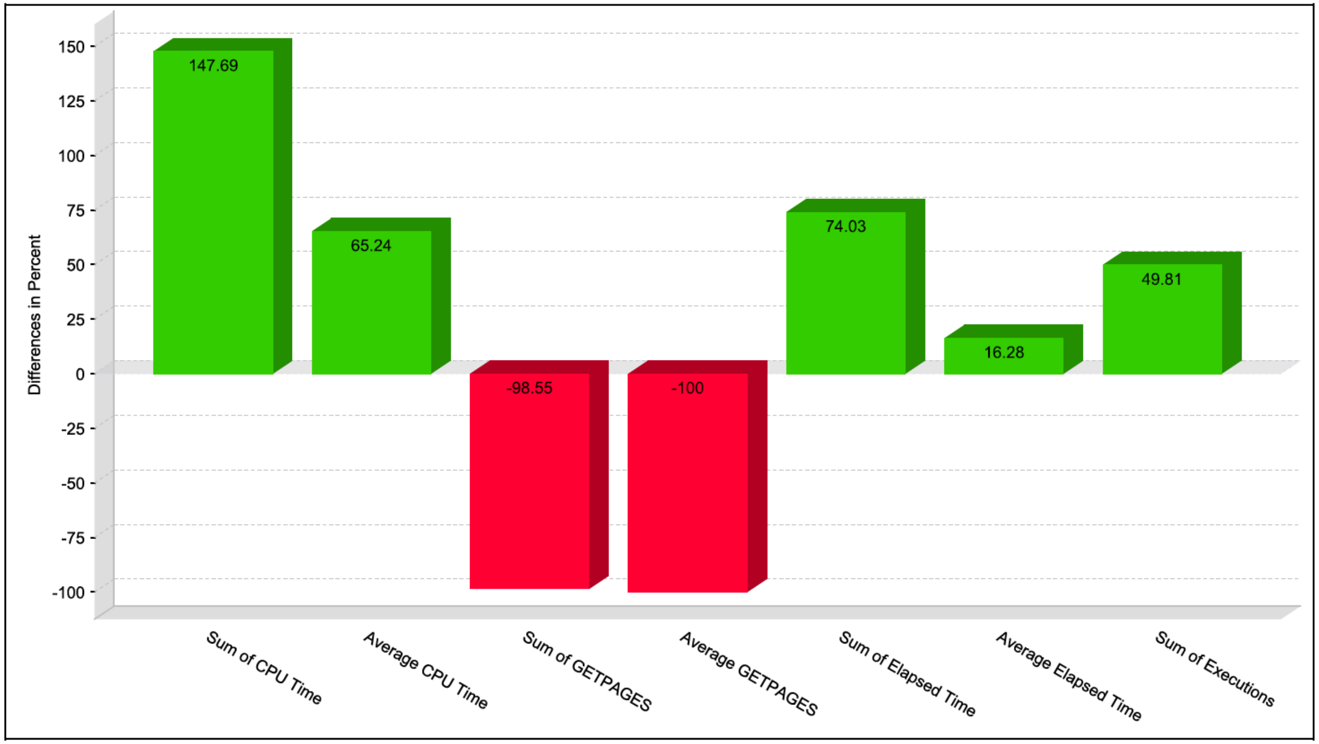
1. How about Intensive CPU?
2. What about by Application?
3. Auditing?
4. Disk I/O Performance?
5. Up and Down Scaling?
6. KPIs for your Enterprise?
7. Searching for SQL?
8. Flushed with success?
9. Index Comparison?

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?

WLX Report

Index maintenance costs for table: IQA0610.IQATW005



iSQL WorkloadExpert™ for DB2 z/OS, © SOFTWARE ENGINEERING GmbH, 2012-2014

Sep 30, 2014 11:49:26 AM

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

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!