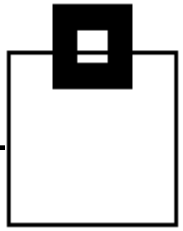
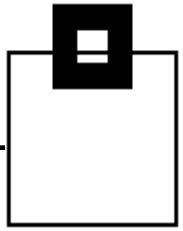


25 years of missed opportunities?

SQL Tuning Revisited

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

AGENDA



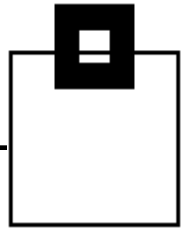
1. Tuning SQL
   - How we have always done it
   - Single SQL, Package, Application…
   - Year 2004 – AP comparisons and simulation

2. Tuning SQL Revisited – A new methodology

3. Harvesting the low hanging fruit

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

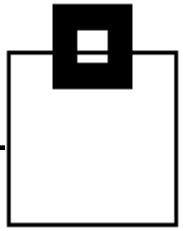
Tuning SQL – How we have always done it



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

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

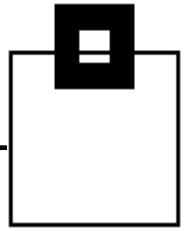
Single SQL, Package, Application…



- 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

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

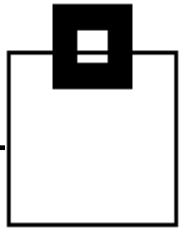
Tuning SQL - Year 2004



- 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

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

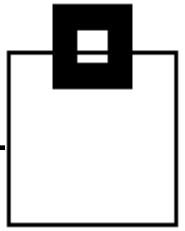
Tuning SQL Revisited



- 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)

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Tuning SQL Revisited

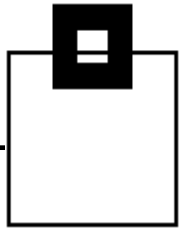


So how to get there?

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

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

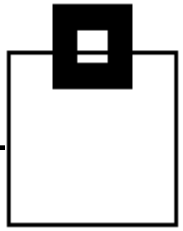
Collect as much data as you can



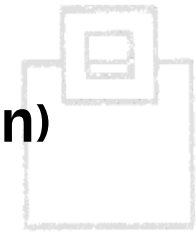
- How many resources do you spend on capturing DB2 SQL workload and its metrics?
- There seems to be out-of-the-box metrics delivered by DB2, but does it give me all the data I need, when I need it?
- How does the smarter database, how does DB2 10, or 11 for z/OS deal with it?...

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

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



© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Collect as much data as you can

What's exactly new since DB2 10:

- 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

- New IFCIDs 400* and 401 additionally EDM pool data – let's call it the **S**tatic **S**tatement **C**ache
  - 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

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

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

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

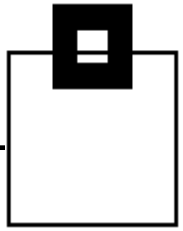
Collect as much data as you can

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

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

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

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Collect as much data as you can

Counters # EXECUTIONS OF THE STATEMENT. FOR A CURSOR STATEMENT, THIS IS THE # OF OPENS. # OF SYNCHRONOUS BUFFER READS PERFORMED FOR STATEMENT. # OF GETPAGE OPERATIONS PERFORMED FOR STATEMENT. # OF ROWS EXAMINED FOR STATEMENT. # OF ROWS PROCESSED FOR STATEMENT - FOR EXAMPLE, THE # OF ROWS RETURNED FOR A SELECT, OR THE NUMBER OF ROWS AFFECTED BY AN INSERT, UPDATE, OR DELETE. # OF SORTS PERFORMED FOR STATEMENT. # OF INDEX SCANS PERFORMED FOR STATEMENT. # OF TABLESPACE SCANS PERFORMED FOR STATEMENT. * # OF PARALLEL GROUPS CREATED FOR STATEMENT. # OF SYNCHRONOUS BUFFER WRITE OPERATIONS PERFORMED FOR STATEMENT. # OF TIMES THAT RID LIST RETRIEVAL FOR MULTIPLE INDEX ACCESS WAS NOT DONE BECAUSE DB2 COULD DETERMINE THE OUTCOME OF INDEX ANDING OR ORING*.

O Counters # OF TIMES THAT A RID LIST WAS NOT USED BECAUSE THE # OF RIDS EXCEEDED ONE OR MORE INTERNAL DB2 LIMITS, AND THE # OF RID BLOCKS EXCEEDED THE VALUE OF SUBSYSTEM PARAMETER MAXTEMPS. RID. # OF TIMES THAT A RID LIST WAS NOT USED BECAUSE NOT ENOUGH STORAGE WAS AVAILABLE TO HOLD THE RID LIST, OR WORK FILE STORAGE OR RESOURCES WERE NOT AVAILABLE. # OF TIMES THAT A RID LIST OVERFLOWED TO A WORK FILE BECAUSE NO RID POOL STORAGE WAS AVAILABLE TO HOLD THE LIST OF RIDS*. # OF TIMES A THAT RID LIST OVERFLOWED TO A WORK FILE BECAUSE THE NUMBER OF RIDS EXCEEDED ONE OR MORE INTERNAL LIMITS*. # OF TIMES THAT APPENDING TO A RID LIST FOR A HYBRID JOIN WAS INTERRUPTED BECAUSE NO RID POOL STORAGE WAS AVAILABLE TO HOLD THE LIST OF RIDS*. # OF TIMES THAT APPENDING TO A RID LIST FOR A HYBRID JOIN WAS INTERRUPTED BECAUSE THE NUMBER OF RIDS EXCEEDED ONE OR MORE INTERNAL LIMITS*.

TIMINGS ACCUMULATED CPU TIME. THIS VALUE INCLUDES CPU TIME THAT IS CONSUMED ON AN IBM SPECIALTY ENGINE. ACCUMULATED ELAPSED TIME USED FOR STATEMENT. ACCUMULATED WAIT TIME FOR LATCH REQUESTS*. ACCUMULATED WAIT TIME FOR PAGE LATCHES*. ACCUMULATED WAIT TIME FOR DRAIN LOCKS* ACCUMULATED WAIT TIME FOR DRAINS DURING WAITS FOR CLAIMS TO BE RELEASED*. ACCUMULATED WAIT TIME FOR LOG WRITERS. ACCUMULATED WAIT TIME FOR SYNCHRONOUS I/O. ACCUMULATED WAIT TIME FOR LOCK REQUESTS. ACCUMULATED WAIT TIME FOR A SYNCHRONOUS EXECUTION UNIT SWITCH. ACCUMULATED WAIT TIME FOR GLOBAL LOCKS. ACCUMULATED WAIT TIME FOR READ ACTIVITY THAT IS DONE BY ANOTHER THREAD. ACCUMULATED WAIT TIME FOR WRITE ACTIVITY THAT IS DONE BY ANOTHER THREAD.

IDENTIFICATION DATA SHARING MEMBER THAT CACHED THE SQL STATEMENT*. PROGRAM NAME. PROGRAM NAME IS THE NAME OF THE PACKAGE OR DBRM THAT PERFORMED THE PREPARE/SQL. PRECOMPILER LINE NUMBER FOR THE PREPARE STATEMENT OR SQL STATEMENT. TRANSACTION NAME. THIS VALUE IS PROVIDED DURING RRS SIGNON OR RE-SIGNON. END USER ID*. THIS VALUE IS PROVIDED DURING RRS SIGNON OR RE-SIGNON. WORKSTATION NAME*. THIS VALUE IS PROVIDED DURING RRS SIGNON OR RE-SIGNON. USER ID. USER ID IS THE PRIMARY AUTH. ID OF THE USER WHO DID THE INITIAL PREPARE. USER GROUP. USER GROUP IS THE CURRENT SQLID OF THE USER WHO DID THE INITIAL PREPARE. USER-PROVIDED IDENTIFICATION STRING.

ENVIRONMENTAL REFERENCED TABLE NAME. FOR STATEMENTS THAT REFERENCE MORE THAN ONE TABLE, ONLY THE NAME OF THE FIRST TABLE THAT IS REFERENCED IS REPORTED. (ALL REFERENCED OBJECTS ARE STORED IN THE WLI DATA MODEL) LITERAL REPLACEMENT FLAG* CURRENT SCHEMA. QUALIFIER THAT IS USED FOR UNQUALIFIED TABLE NAMES. BIND OPTIONS: ISOLATION, CURRENTDATA, AND DYNAMICRULES. SPECIAL REGISTER VALUES: CURRENT DEGREE, CURRENT RULES, AND CURRENT PRECISION. WHETHER THE STATEMENT CURSOR IS A HELD CURSOR. TIMESTAMP WHEN STATISTICS COLLECTION BEGAN. DATA COLLECTION BEGINS WHEN A TRACE FOR IFCID 318 IS STARTED. DATE AND TIME WHEN THE STATEMENT WAS INSERTED INTO THE CACHE IN STORE CLOCK FORMAT. DATE AND TIME WHEN THE STATEMENT WAS UPDATED, IN STORE CLOCK FORMAT. DATE AND TIME WHEN THE STATEMENT WAS UPDATED, IN INTERNAL FORMAT.

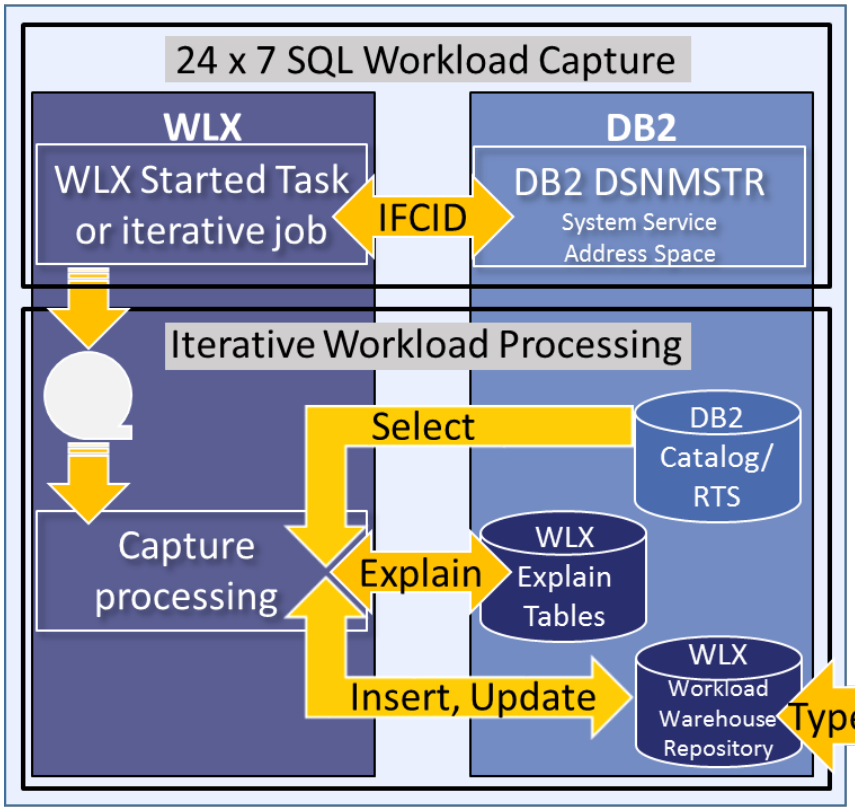
© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

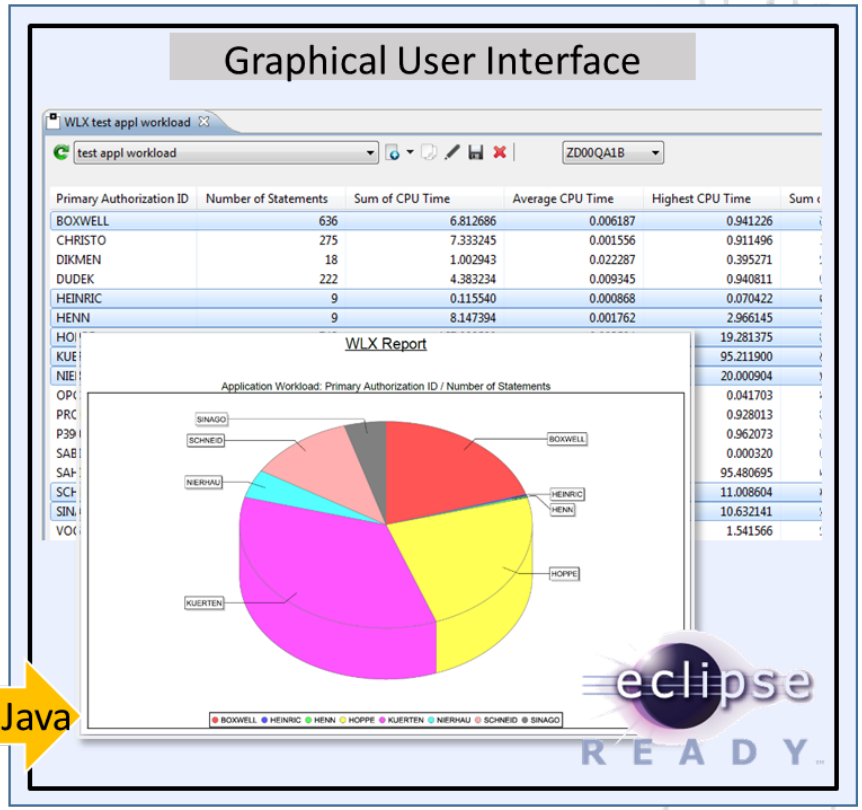
Store it in a SQL Workload Data Warehouse

Capture the hard-to-get SQLs, even the ones that disappear …

Mainframe Engine

Workstation Engine





© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Store it in a SQL Workload Data Warehouse

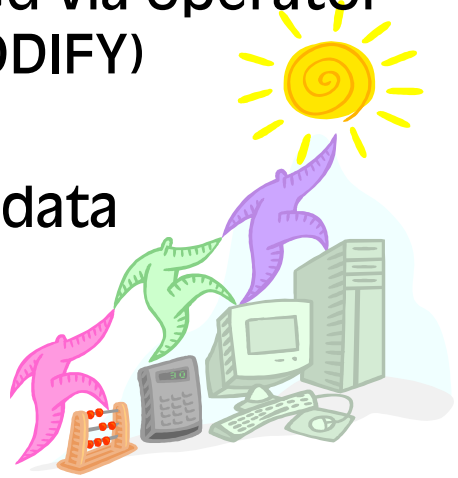
Capture the data e.g. using a STC:

Run a started task 24x7 to catch all the IFCIDs that DB2 will be throwing and store the data.

Process the workload:

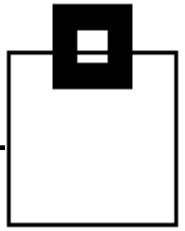
Externalize and process the data, such as every 60 min:

- 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



© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Store it in a SQL Workload Data Warehouse



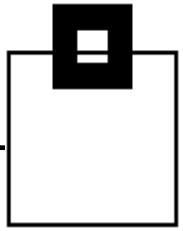
Use a GUI front end, preferably Eclipse:

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

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

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!

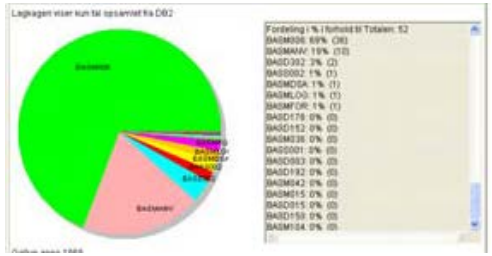
And remember that all of the Type 4 JAVA SQL is ziiP eligible!

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Exploitation example

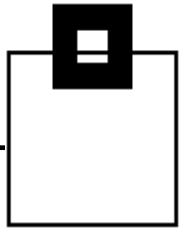
- One of the first customers exploiting this DB2 feature was JN Data from Denmark during the DB2 10 ESP

- Frank Petersen picks up the IFCID records, stores them in a repository and reports nicely in a GUI even supporting mobile devices



© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Harvesting the low hanging fruit



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

1. How about Intensive CPU?

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

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:

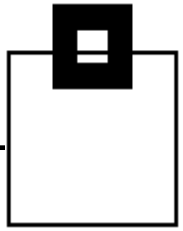
Sum of Executions	Sum of CPU Time	Average CPU Time	Percentage CPU Time	Sum of GETPAGES
20,234	2,654.196293	0.131175	61.21	329,091,770
39	412.657181	10.580953	9.52	1,339,006
1,338	227.825821	0.170273	5.25	7,587,830
13,636	158.568243	0.011628	3.66	5,698,318
24,912	155.575122	0.006244	3.59	2,334,284
1,541	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
```

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

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?

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

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:

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

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

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Harvesting the low hanging fruit

Here is a high CPU Static application:

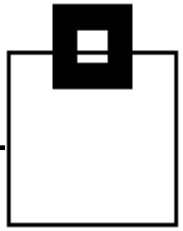
Primary Authorization ID	Number of Statements	Sum of CPU Time	Average CPU Time	Highest CPU Time
	309	207.529534	0.000179	32.257744
BOXWELL	152	7.227270	0.008166	1.192778
CHRISTO	1	0.000268	0.000268	0.000268
DUDEK	128	2.177016	0.004380	0.866522

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

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

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!

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Harvesting the low hanging fruit

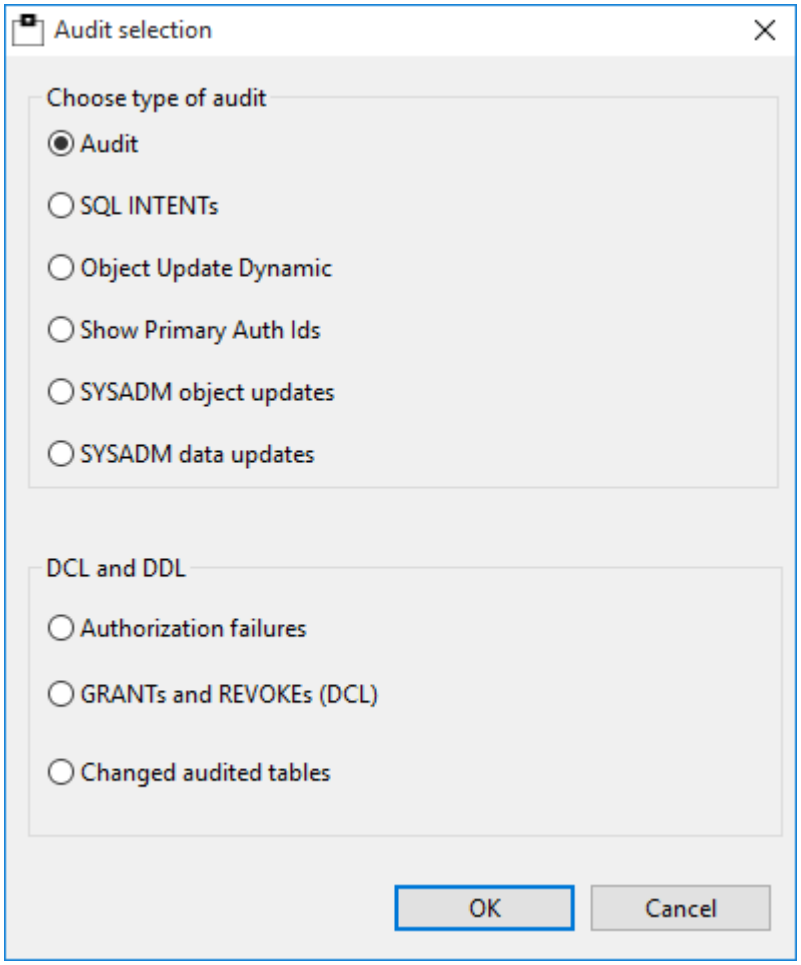
OK, so assuming you have all the data where shall we begin???

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

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

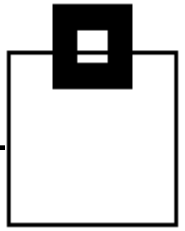
Harvesting the low hanging fruit

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



© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Harvesting the low hanging fruit

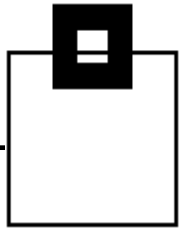


Choose how you like to find out who did what and when…

Transaction name	End User ID	Workstation name	Primary Authorization ID	Current SQL ID	Qualifier	Package	Query type	Intent	Table creator	Table name	Ob
HOPPE	HOPPE	DB2CALL	HOPPE	HOPPE	HOPPE	IQADBACP	SELECT	READ	IQA0610	IQAXI0041	
HOPPE	HOPPE	DB2CALL	HOPPE	HOPPE	HOPPE	IQADBACP	SELECT	READ	IQA0610	IQATI004	
HOPPE	HOPPE	DB2CALL	HOPPE	HOPPE	HOPPE	IQADBACP	SELECT	READ	IQA0610	IQAXI0041	
BOXWELL	BOXWELL	TSO	BOXWELL	BOXWELL	BOXWELL	DSNESM68	SELECT	READ	SYSIBM	SYSDUM...	
BOXWELL	BOXWELL	TSO	BOXWELL	BOXWELL	BOXWELL	DSNESM68	SELECT	READ	MVNXTEST	MVNXX861	
BOXWELL	BOXWELL	TSO	BOXWELL	BOXWELL	BOXWELL	DSNESM68	SELECT	READ	MVNXTEST	MVNXT86	
HOPPE	HOPPE	DB2CALL	HOPPE	PTFADMIN	PTFADMIN	DSNREXX	SELECT	READ	PTFADMIN	MEMBER	
HOPPE	HOPPE	DB2CALL	HOPPE	PTFADMIN	PTFADMIN	DSNREXX	SELECT	READ	PTFADMIN	USERTAB	
HOPPE	HOPPE	DB2CALL	HOPPE	PTFADMIN	PTFADMIN	DSNREXX	SELECT	READ	PTFADMIN	PTF	
HOPPE	HOPPE	DB2CALL	HOPPE	PTFADMIN	PTFADMIN	DSNREXX	SELECT	READ	PTFADMIN	PTFTIN02	
HOPPE	HOPPE	DB2CALL	HOPPE	PTFADMIN	PTFADMIN	DSNREXX	SELECT	READ	PTFADMIN	PTF	
HOPPE	HOPPE	DB2CALL	HOPPE	PTFADMIN	PTFADMIN	DSNREXX	SELECT	READ	PTFADMIN	PTF	
HOPPE	HOPPE	DB2CALL	HOPPE	HOPPE	HOPPE	IQADBACP	SELECT	READ	IQA0610	IQATI006	
HOPPE	HOPPE	DB2CALL	HOPPE	HOPPE	HOPPE	IQADBACP	SELECT	READ	IQA0610	IQAXI0061	
HOPPE	HOPPE	DB2CALL	HOPPE	HOPPE	HOPPE	IQADBACP	SELECT	READ	IQA0610	IQATI007	
HOPPE	HOPPE	DB2CALL	HOPPE	HOPPE	HOPPE	IQADBACP	SELECT	READ	IQA0610	IQAXI0071	

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Harvesting the low hanging fruit

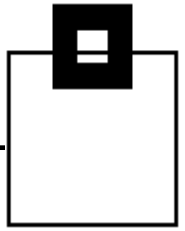


OK, so assuming you have all the data where shall we begin???

1. How about Intensive CPU?
2. What about by Application?
3. Auditing?
4. Disk I/O Performance?

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Harvesting the low hanging fruit



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

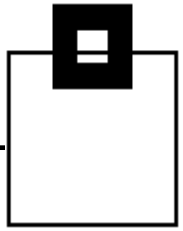
Number of Statements	Wait time per synchronous IO	Synchronous IOs per statement	Sum of Wait Synchronous IO	Sum of Synchronous Buffer Reads
3	0.009594	23.000	0.662019	69
2	0.007095	1,619.000	22.974946	3,238
2	0.006794	13.000	0.176651	26
4	0.005586	377.750	8.441286	1,511

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

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

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?

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Harvesting the low hanging fruit

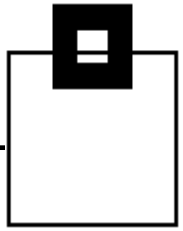
Up and Down scaling is all about getting a “level playing field” when looking at the cache data. Simply displaying the data for SQLs that have been in the cache a week next to SQLs that have been in the cache for only 10 minutes is a bit biased!

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!

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Harvesting the low hanging fruit

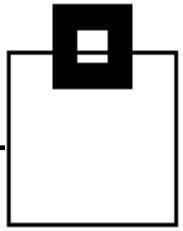


OK, so assuming you have all the data where shall we begin???

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?

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

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:

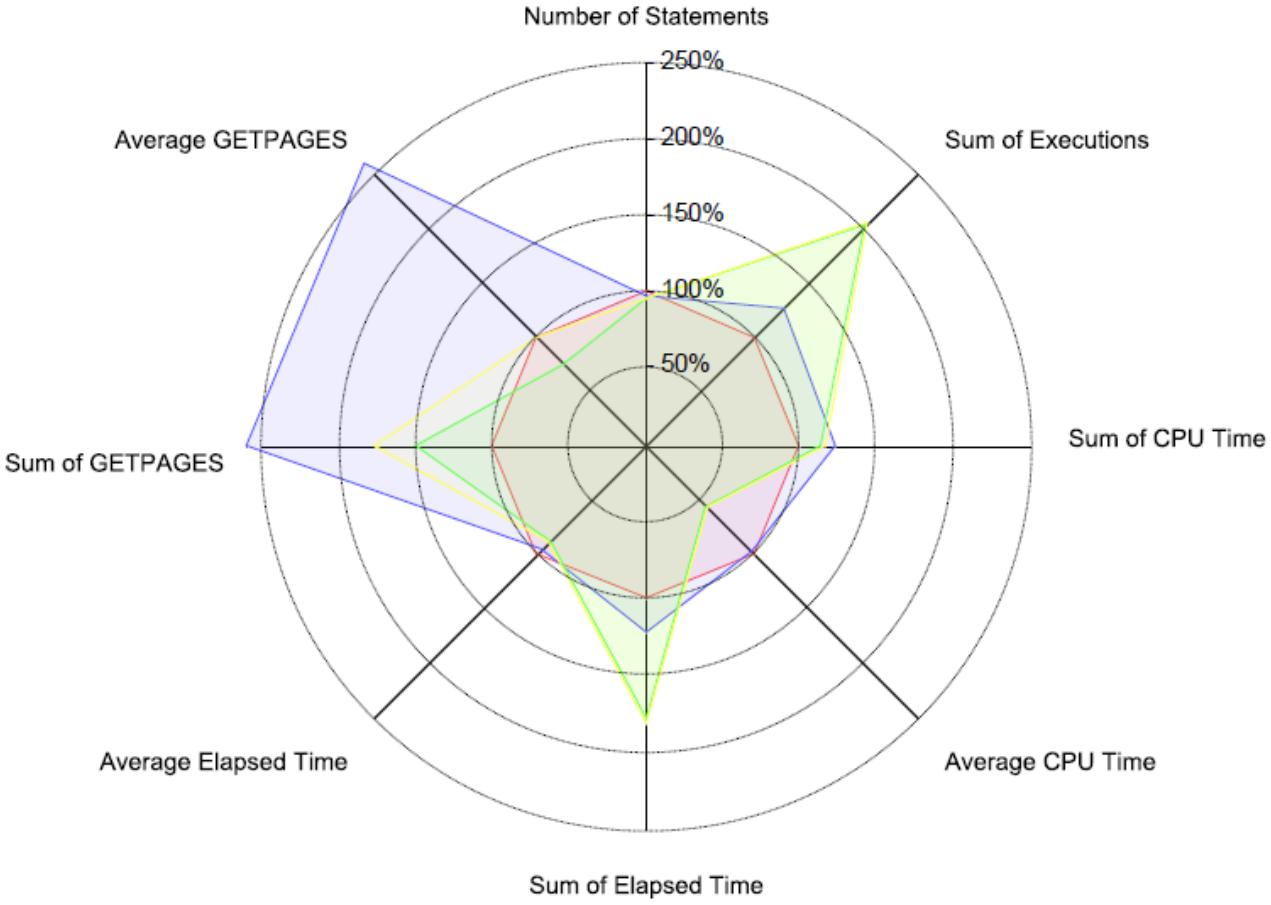
Number of Statements	Total number of Dynamic...	Total number of Static...	Sum of CPU Time	Average CPU Time	Highest CPU Time
2,010	1,698	312	4,428.794113	0.007277	425.162468
2,051	1,732	319	4,678.737674	0.007359	479.465437
2,415	2,069	346	400.380448	0.000923	27.529473
4,342	4,029	313	1,495.726711	0.004041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

Not just CPU but GetPages etc. are also available.

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

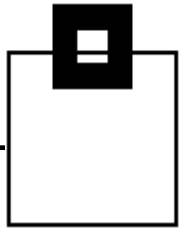
Harvesting the low hanging fruit

Then you can play
with radar charts:



© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Harvesting the low hanging fruit



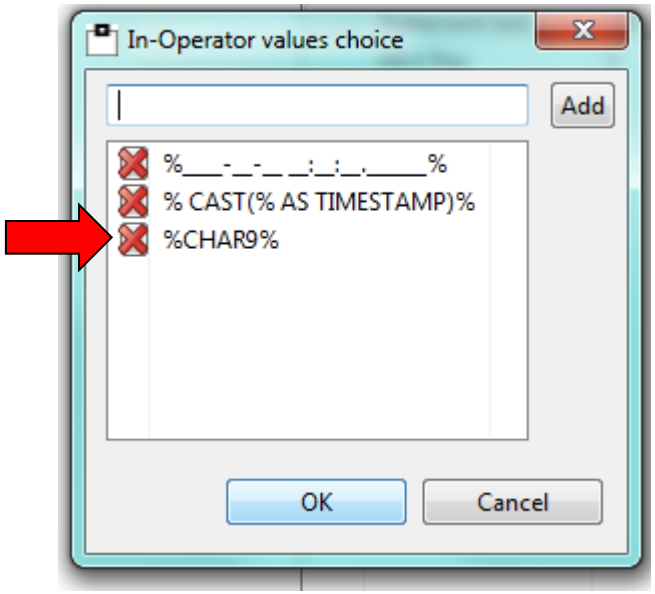
OK, so assuming you have all the data where shall we begin???

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?

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

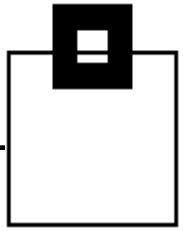
Harvesting the low hanging fruit

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.



© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Harvesting the low hanging fruit

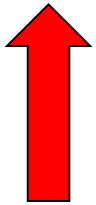


And then…

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

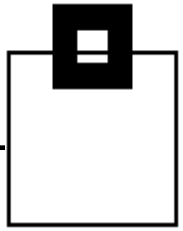
Drill down to get a better view

```
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
```



© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Harvesting the low hanging fruit

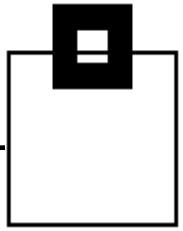


OK, so assuming you have all the data where shall we begin???

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?

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Harvesting the low hanging fruit



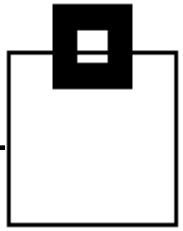
If you are catching and storing all the SQL then you can easily see how good the size and performance of your cache is:

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!

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Harvesting the low hanging fruit



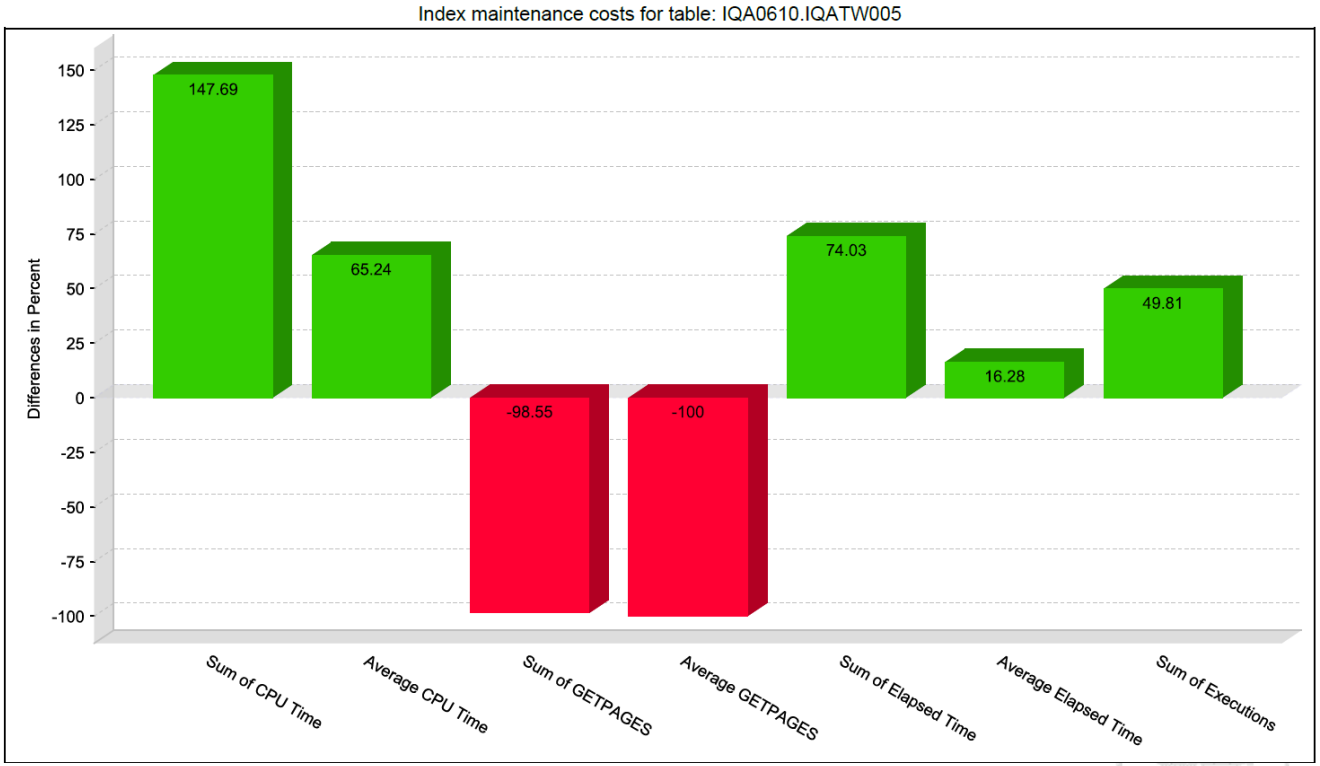
OK, so assuming you have all the data where shall we begin???

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?

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

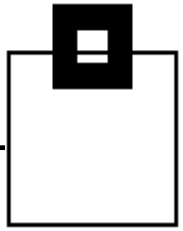
Harvesting the low hanging fruit

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?



© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Harvesting the low hanging fruit

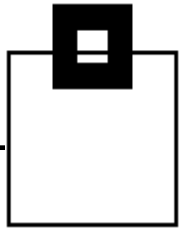


OK, so assuming you have all the data where shall we begin???

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…

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

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

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

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

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

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?

WLX Key	Collection ID	Package	Count
2015-02-11-10.40.01.509983	[redacted]	IQAXPLN	42
2015-02-11-10.40.01.509983	IQA_COLL_610_QB	IQAXPLN	28
2015-02-11-10.40.01.509983	IQA_COLL_610_QB	IQADBACP	14
2015-02-11-10.40.01.509983	IQA_COLLECTION_610	IQAXPLN	14
2015-02-11-10.40.01.509983	IQA_COLLECTION_610	TESTPGM	2
2015-02-27-16.03.31.795257	DSNESPUR	DSNESM68	4
2015-02-27-16.03.31.795257	IQA_COLLECTION_610	IQAXPLN	4
2015-03-04-13.48.32.142444	DSNESPUR	DSNESM68	20
2015-03-04-13.48.32.142444	MDB2VNEX_TEST	O2TESTB	18
2015-03-04-13.48.32.142444	IQA_COLLECTION_610	IQAXPLN	4
2015-03-18-10.23.46.077223	IQA_COLLECTION_610	TESTPGM	5
2015-03-26-17.35.19.367772	IQA_COLLECTION_610	TESTPGM	2

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

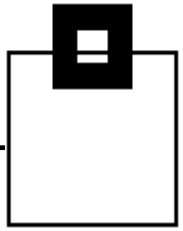
Harvesting the low hanging fruit

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

WLX Key	Collection ID	Package	ICI number	Count	Reason
2015-03-04-13.48.32.142444	DSNESPUR	DSNESM68	1	12	DB2 9 CHAR Usage
2015-03-04-13.48.32.142444	DSNESPUR	DSNESM68	5	8	Keyword CUBE used as unqualified UDF

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

Harvesting the low hanging fruit

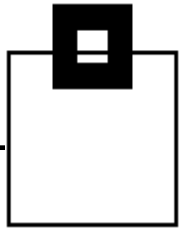


Some real world numbers to amaze and astound:

- 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.

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH

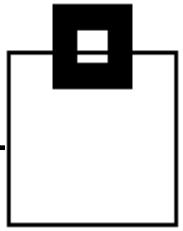
Appendix



- 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

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH



25 years of missed opportunities?
SQL Tuning Revisited

Ulf Heinrich

SEGUS Inc
u.heinrich@segus.com

© 2015 SEGUS & SOFTWARE ENGINEERING GMBH