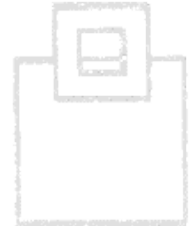
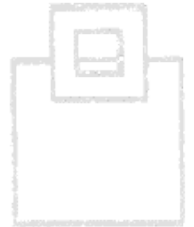


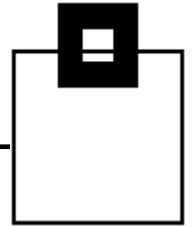
DB2 11 Capturing Tuning and Trending for SQL Workloads - a resource and cost saving approach



Ulf Heinrich
SEGUS Inc



Agenda



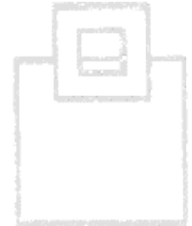
1. DB2 technology enhancements at a glance

- a) Dynamic Statement Cache - introducing IFCIDs 316, 317 and 318
- b) "Static Statement Cache" - introducing the new IFCIDs 400 and 401
- c) Flushed and gone?!? - How to catch what's invalidated from the cache?



2. Data Data Data

- a) What's there for me? The scope of statement level statistics.
- b) What's not there? Limitations of statement level statistics.
- c) How to complete it? - RTS, DB2 Catalog and Explain tables to enrich your SQL workload statistics.

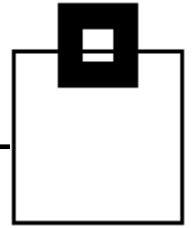


3. Building up a SQL workload warehouse

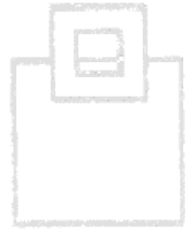
- a) Examples and suggestions what to do with the data.
- b) How to build up a SQL workload warehouse and how to maintain it.
- c) Real world experiences - what others did.



DB2 technology enhancements at a glance



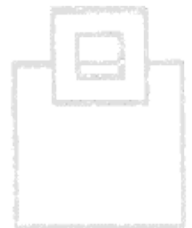
- How many resources do you spend on capturing DB2 SQL workload and its metrics?



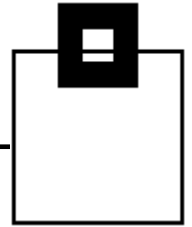
- There seems to be out-of-the-box metrics delivered by DB2, but does it give me all the data I need, when I need it?



- How does the smarter database, how does DB2 11 for z/OS deal with it?...



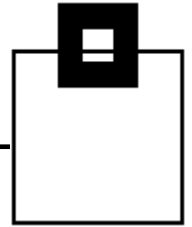
DB2 technology enhancements at a glance



- 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



DB2 technology enhancements at a glance



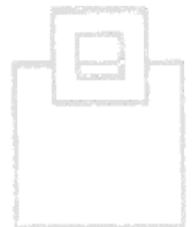
What's exactly new:



- 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 **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

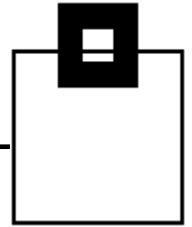
Data Data Data

DSC and EDM provide detailed workload insights:

- SQL text
- Statement ID
- Date/time
- Current status
- Resource consumption
- Identification/ environmental data

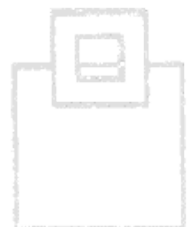


Data Data Data



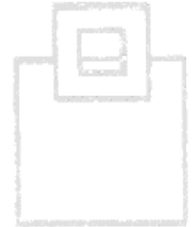
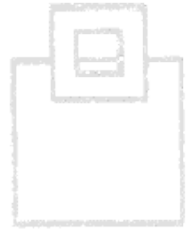
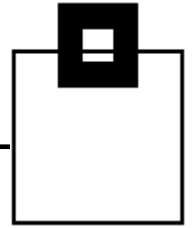
DB2 10 also introduced some additional information from the DSC trace we all know today:

- Wait time accumulation for
 - Latch requests
 - Page latches
 - Drain locks
 - Drains during waits for claims to be released
 - Log writers



Data Data Data

- 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



Dat a Dat a Dat a

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*.

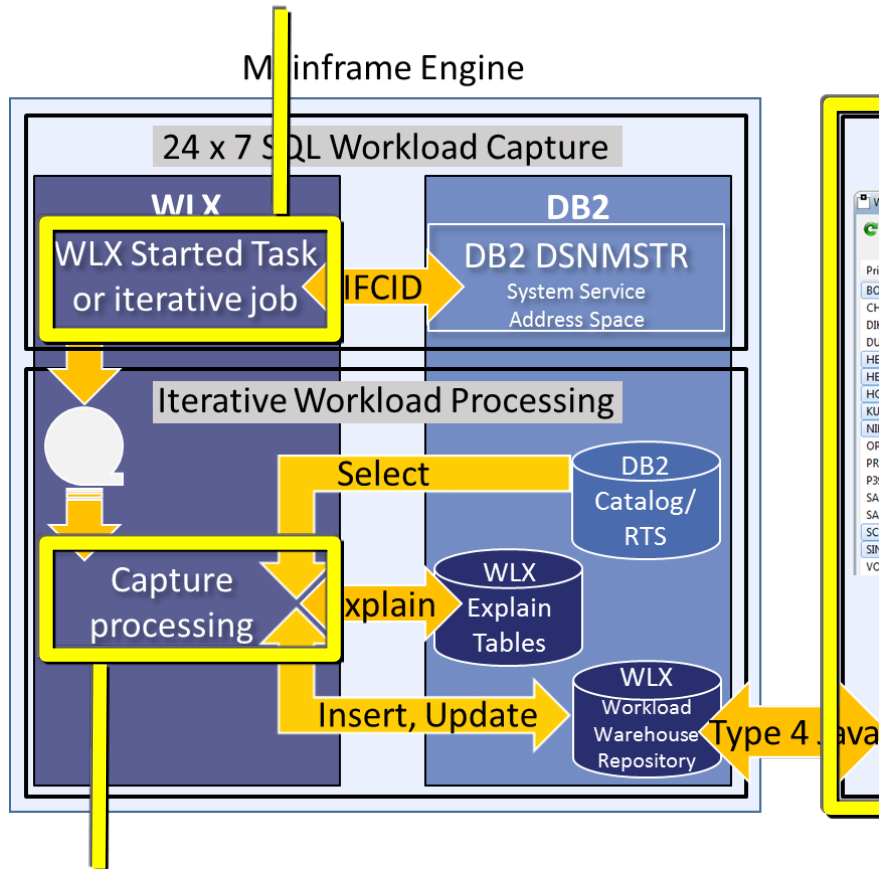
O Counters # OF TIMES THAT A RID LIST WAS NOT USED BECAUSE THE # OF RIDS EXCEEDED ONE OR MORE INTERNAL DB2 LIMITS, AND THE # OF RID BLOCKS EXCEEDED THE VALUE OF SUBSYSTEM PARAMETER MAXTEMP. RID. # OF TIMES THAT A RID LIST WAS NOT USED BECAUSE NOT ENOUGH STORAGE WAS AVAILABLE TO HOLD THE RID LIST, OR 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 THAT A 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.

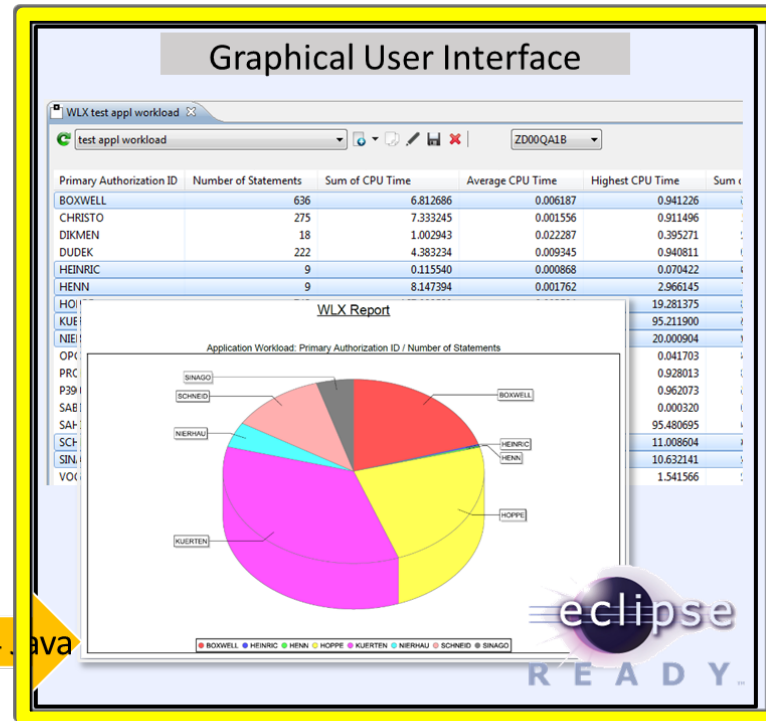
IDENTIFICATION DATA SHARING MEMBER THAT CACHED THE SQL STATEMENT*. PROGRAM NAME. PROGRAM NAME IS THE NAME OF THE PACKAGE OR DBRM THAT PERFORMED THE PREPARE/SQL. PRECOMPILER LINE NUMBER FOR THE PREPARE STATEMENT OR SQL STATEMENT. TRANSACTION NAME. THIS VALUE IS PROVIDED DURING RRS SIGNON OR RE-SIGNON. 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, CURRENT DATA, AND DYNAMIC RULES. 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.

① WLX STC



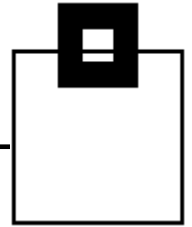
Workstation Engine



② WLX workload processing engine

③ WLX GUI interface

Building up a SQL workload warehouse (WLX)



The WLX STC:

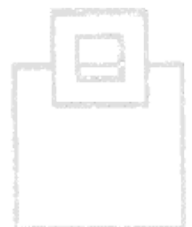
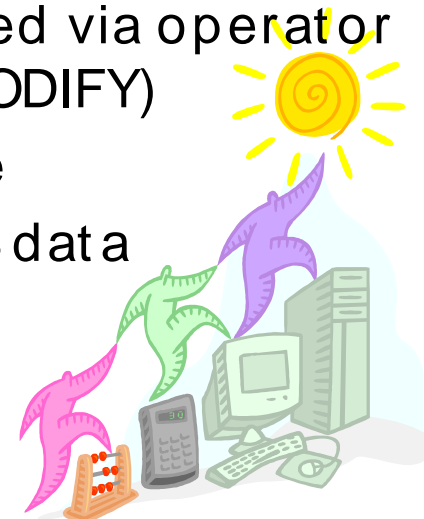
Run a started task 24x7 to catch all the IFCIDs that DB2 will be throwing and store the data.



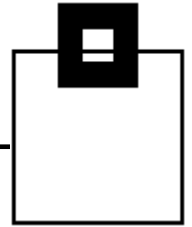
Workload processing engine:

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



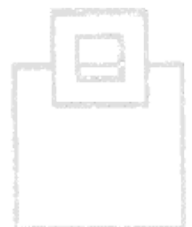
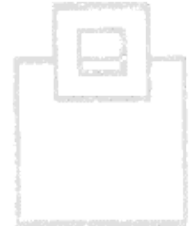
Building up a SQL workload warehouse (WLX)



GUI front end for Eclipse:

Exploit and integrate into Eclipse based GUI front ends

- Eclipse based GUI allow to Plug-In to
 - 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 create PDF reports and allow MS Excel hand over
- Additional plug-ins interface with other tools, such as SEGUS **SQL PerformanceExpert** (SPX) and **Bind ImpactExpert** (BIX)



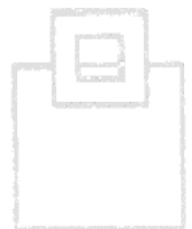
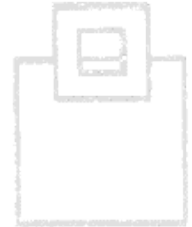
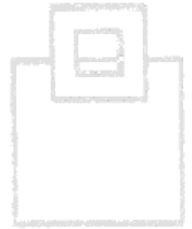
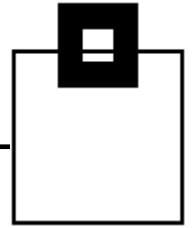
Building up a SQL workload warehouse (WLX)

Enhance your existing SQL Performance Management tools to interface to the DB2 out-of-the-box data.

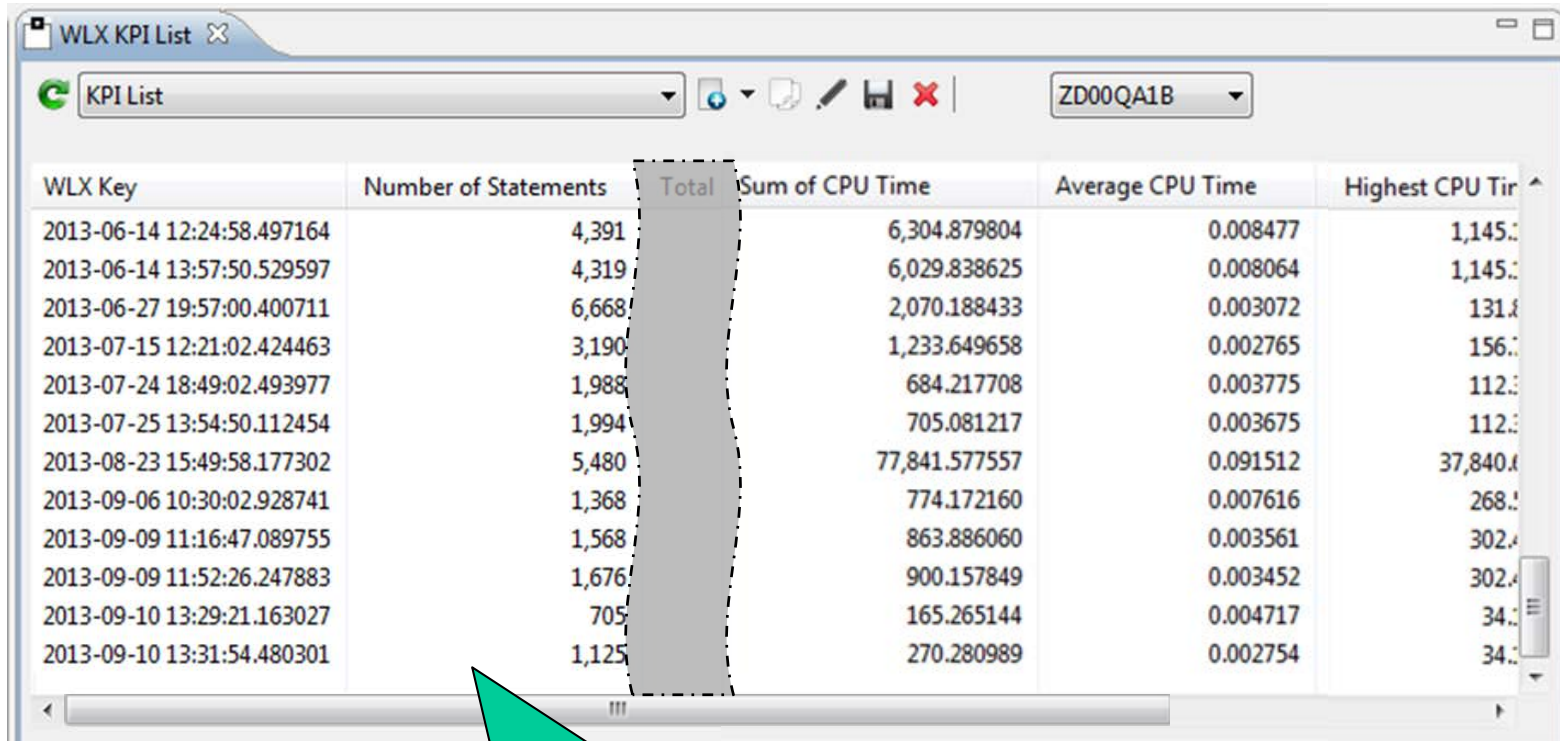
Resulting benefits:

- See any executed SQL in a plex-wide report
- Workload/performance warehouse repository that contains *all* executed SQL
- Powerful history and trending analysis

→ **All of this is now available with the smallest overhead ever possible!**



Building up a SQL workload warehouse (WLX)

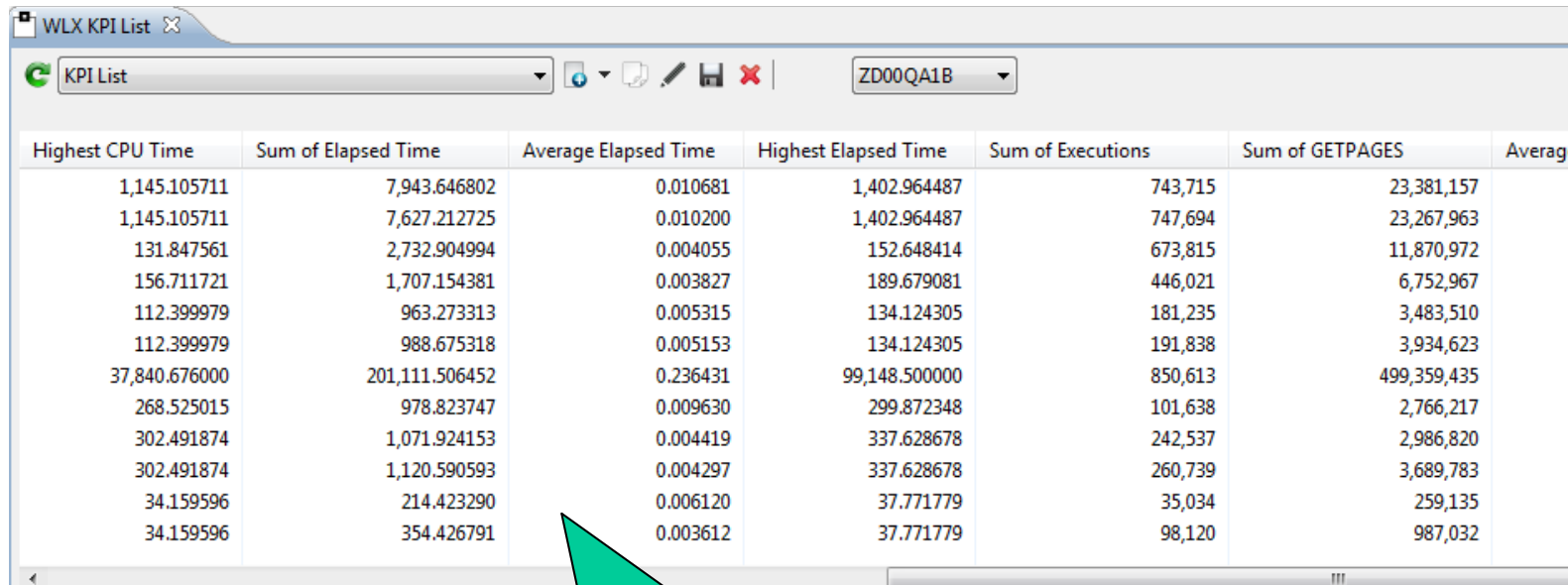


The screenshot shows a software window titled "WLX KPI List". It features a menu bar with a green "C" icon and a dropdown menu labeled "KPI List". To the right of the menu bar are icons for file operations (open, save, print, delete) and a dropdown menu currently set to "ZD00QA1B". The main area contains a table with the following columns: "WLX Key", "Number of Statements", "Total", "Sum of CPU Time", "Average CPU Time", and "Highest CPU Time". The "Total" column is highlighted with a grey background and a dashed border. The table lists 12 rows of data, each representing a specific workload key with its corresponding metrics.

WLX Key	Number of Statements	Total	Sum of CPU Time	Average CPU Time	Highest CPU Time
2013-06-14 12:24:58.497164	4,391		6,304.879804	0.008477	1,145.2
2013-06-14 13:57:50.529597	4,319		6,029.838625	0.008064	1,145.2
2013-06-27 19:57:00.400711	6,668		2,070.188433	0.003072	131.8
2013-07-15 12:21:02.424463	3,190		1,233.649658	0.002765	156.1
2013-07-24 18:49:02.493977	1,988		684.217708	0.003775	112.3
2013-07-25 13:54:50.112454	1,994		705.081217	0.003675	112.3
2013-08-23 15:49:58.177302	5,480		77,841.577557	0.091512	37,840.0
2013-09-06 10:30:02.928741	1,368		774.172160	0.007616	268.9
2013-09-09 11:16:47.089755	1,568		863.886060	0.003561	302.4
2013-09-09 11:52:26.247883	1,676		900.157849	0.003452	302.4
2013-09-10 13:29:21.163027	705		165.265144	0.004717	34.1
2013-09-10 13:31:54.480301	1,125		270.280989	0.002754	34.1

Workload KPIs –
left hand side

Building up a SQL workload warehouse (WLX)

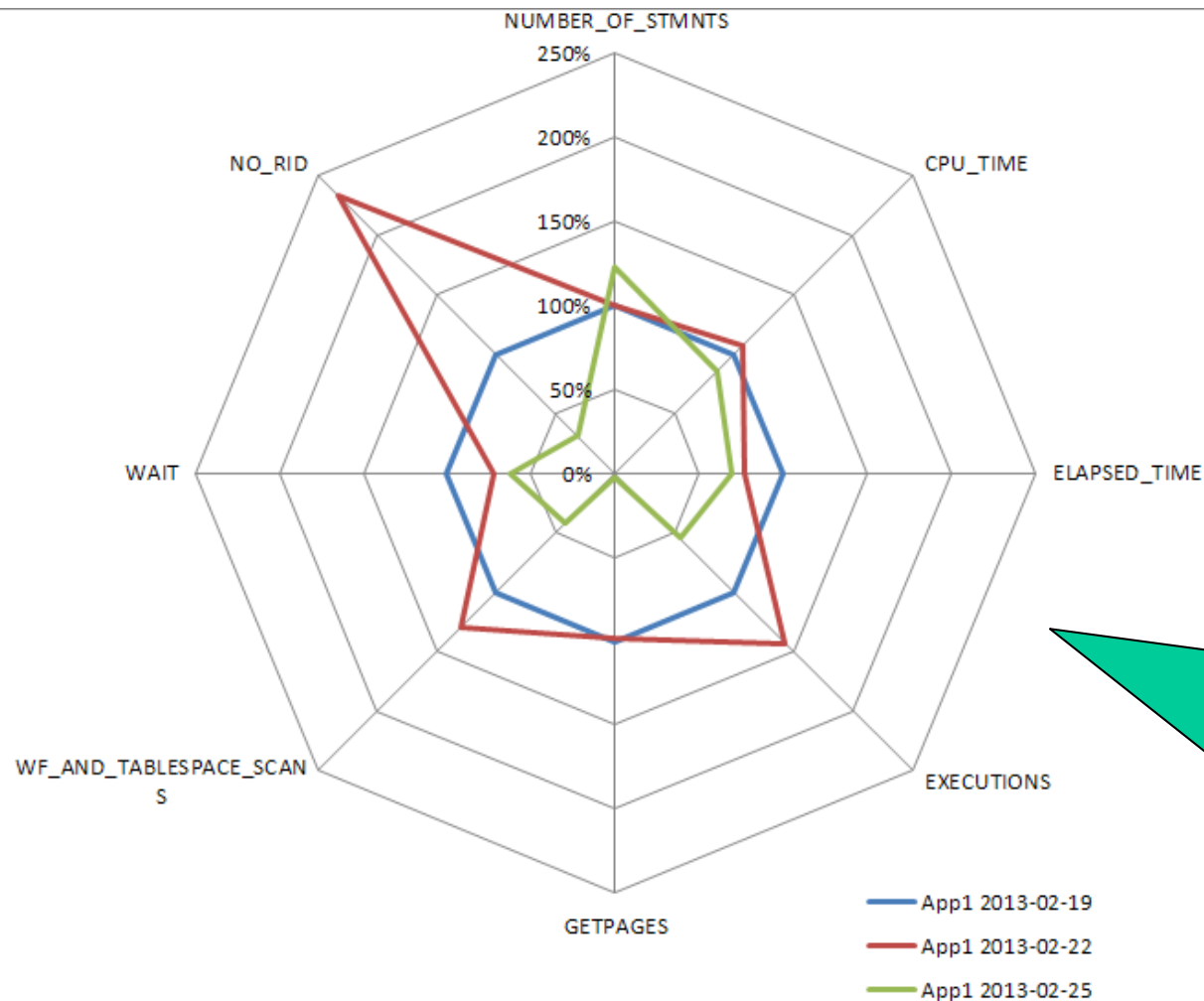


The screenshot shows a software window titled "WLX KPI List". Inside, there's a toolbar with icons for file operations and a dropdown menu showing "ZD00QA1B". Below the toolbar is a table with 7 columns: "Highest CPU Time", "Sum of Elapsed Time", "Average Elapsed Time", "Highest Elapsed Time", "Sum of Executions", "Sum of GETPAGES", and "Average". The table contains 14 rows of data. A green callout bubble points to the right side of the table.

Highest CPU Time	Sum of Elapsed Time	Average Elapsed Time	Highest Elapsed Time	Sum of Executions	Sum of GETPAGES	Average
1,145.105711	7,943.646802	0.010681	1,402.964487	743,715	23,381,157	
1,145.105711	7,627.212725	0.010200	1,402.964487	747,694	23,267,963	
131.847561	2,732.904994	0.004055	152.648414	673,815	11,870,972	
156.711721	1,707.154381	0.003827	189.679081	446,021	6,752,967	
112.399979	963.273313	0.005315	134.124305	181,235	3,483,510	
112.399979	988.675318	0.005153	134.124305	191,838	3,934,623	
37,840.676000	201,111.506452	0.236431	99,148.500000	850,613	499,359,435	
268.525015	978.823747	0.009630	299.872348	101,638	2,766,217	
302.491874	1,071.924153	0.004419	337.628678	242,537	2,986,820	
302.491874	1,120.590593	0.004297	337.628678	260,739	3,689,783	
34.159596	214.423290	0.006120	37.771779	35,034	259,135	
34.159596	354.426791	0.003612	37.771779	98,120	987,032	

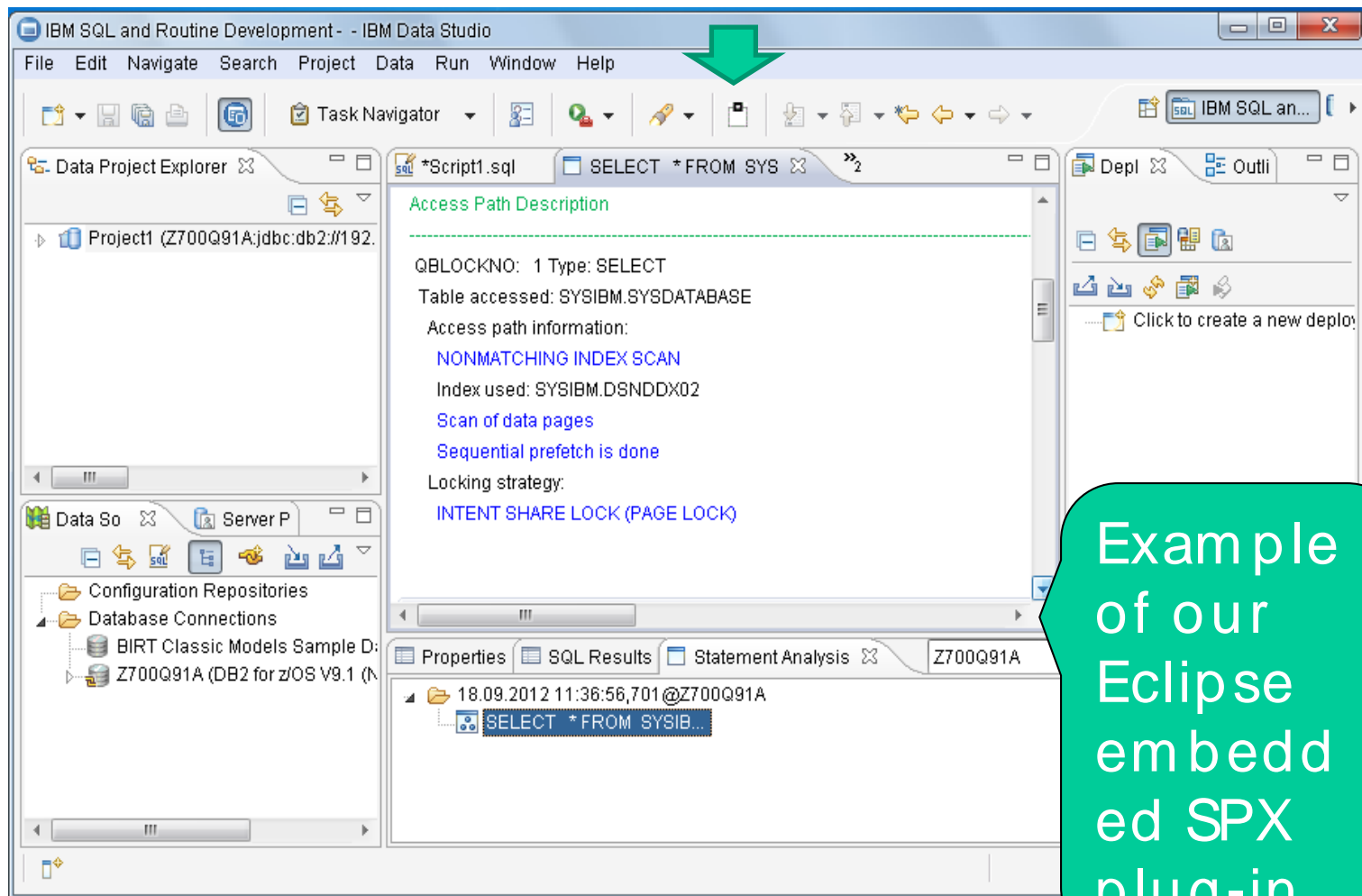
Workload KPIs –
right hand side

Building up a SQL workload warehouse (WLX)



Spider diagram of three application extracts

Building up a SQL workload warehouse (WLX)



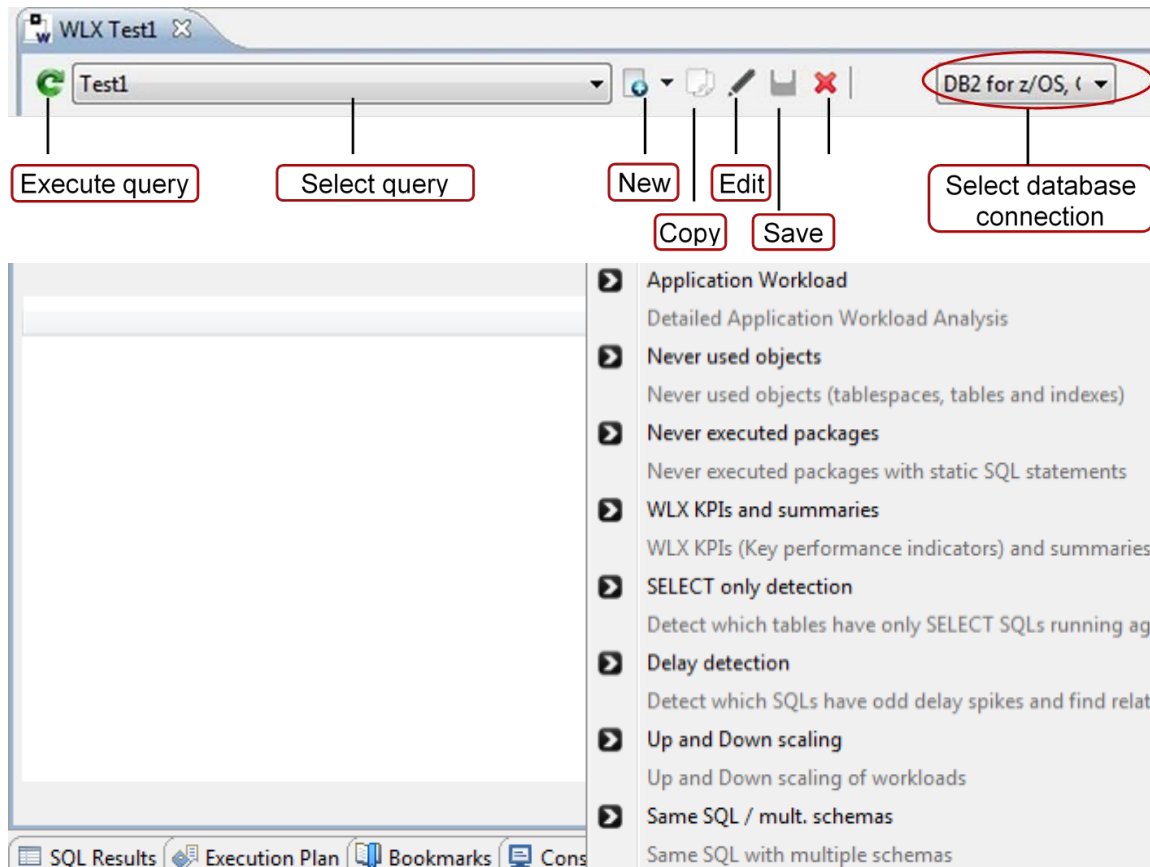
Building up a SQL workload warehouse (WLX)

The screenshot displays the IBM Data Studio interface. The main window shows a SQL script editor with the query `SELECT * FROM SYSIB...`. Below the editor, the 'Violation(s)' pane is active, displaying a table of rule violations. The table has columns: RuleNo, InfoType, Type, and Description. Two violations are listed: RuleNo 9002 with a WARNING type, and RuleNo 9065 with a WARNING type. The description for RuleNo 9065 states: 'SELECT * can lead to unnecessary...'. A green callout bubble points to the 'Type' column of the second violation, containing the text 'SPX rule violation'. The left pane shows the 'Data Project Explorer' with a project named 'Project1 (Z700Q91A;jdbc:db2://192...)'. The bottom pane shows the 'Properties' tab for the selected query, displaying details such as 'TS: DSNDB06 .SYSDBAUT', 'Stats: 2010-05-27-14.11.14.325064', 'Partitions: 0', 'Tables: 2', 'NACTIVEF: 144', and 'Type: Neither a LOB nor a MEMBER CLUSTER.'

RuleNo	InfoType	Type	Description
9002		WARNING	Estimated service units > 25. Try to...
9065		WARNING	SELECT * can lead to unnecessary...

SPX rule violation

Building up a SQL workload warehouse (WLX)



GUI features
– button
overview

Example
use case
drop down
box

Building up a SQL workload warehouse (WLX)

Example of application workload and SQL text drill down

The screenshot displays the Eclipse Database Development environment. The top pane shows the results of a query, and the bottom pane shows the SQL script for the application workload.

Primary Authorization ID	CPU Time	Elapsed Time	Executions	GETPAGES	Synchronous Buffer Reads	Synchronous Buffer Writes	Rows examined	Rows processed	Index scans	WF and Tablespace Scans	Sorts performed	Parallel Group
BOXWELL	6.562974	8.074171	1	13,922	481	0	766,371	3	0	1	1	
BOXWELL	5.982266	17.237795	2	21,640	98	0	881,114	20	0	2	2	
BOXWELL	4.850175	5.619810	1	9,298	13	0	288,444	42	255	8	6	
BOXWELL	4.344587	5.979491	1	13,922	0	0	766,371	12	0	1	1	
BOXWELL	2.736454	3.798705	2	7,059	84	0	170,121	1,699	11,779	0	0	
BOXWELL	1.568500	1.926214	1	3,529	17	0	85,061	1	5,890	0	0	
BOXWELL	1.030146	1.465298	3	7,223	70	0	4,623	9	18	18	12	
BOXWELL	0.909670	1.002262	1	2,792	0	0	0	12	8	6	4	
BOXWELL	0.843634	0.939291	2	3,779	12	0	0	266	4	4	4	
BOXWELL	0.607773	3.096399	1	4,571	442	0	150,679	10	0	1	0	
BOXWELL	0.392377	1.753672	2	2,418	697	0	2,320	24	1,288	12	8	
BOXWELL	0.384967	1.113439	1	2,660	28	0	88,236	963	963	1	0	
BOXWELL	0.379904	0.606013	1	4,974	0	0	265,800	10	0	1	0	
BOXWELL	0.367102	0.411977	3	474	0	0	69	33	75	18	12	

Result counter: 545

Application Workload

Connection profile

Type: Name: Database: Status: Disconnected, Auto Commit

```
WITH
PRIME_KEY (WLX_TIMESTAMP, STMT_GROUP_SSID, STMT_ID, STMT_ORIGIN,
STMT_TIMESTAMP, STMT_TYPE, SCALE_ADJ) AS
(SELECT WLX_TIMESTAMP, STMT_GROUP_SSID, STMT_ID, STMT_ORIGIN, STMT_TIMESTAMP,
STMT_TYPE, CASE WHEN STMT_TIMESTAMP >= (SELECT MAX(IQA0610.WLXT009.WLX_TIMESTAMP)
FROM IQA0610.WLXT009) - 1 HOURS THEN 3.6E+03 / CASE TIMESTAMPODIFF(2, CAST (STMT_STATS_UPD - STMT_TIMESTAMP AS CHAR (22))) WHEN 0 THEN 1 ELSE TIMESTAMPODIFF(2, CAST (STMT_STATS_UPD - STMT_TIMESTAMP AS CHAR (22)))
AND ((1 = 1)
AND (WLX_TIMESTAMP = (SELECT MAX(IQA0610.WLXT009.WLX_TIMESTAMP)
FROM IQA0610.WLXT009))))))
SELECT *
FROM (SELECT DECIMAL(K.SCALE_ADJ, 9, 3) AS SCALE_ADJ, DECIMAL((A.CPU_TIME * 1.000000) / 1E+06, 12, 6) AS CPU_TIME,
DECIMAL((A.CPU_TIME * 1E+02) / (CASE W.DCPU_TIME WHEN 0 THEN 1 ELSE W.DCPU_TIME END),
12, 6) AS PCT_CPU_TIME, CASE WHEN A.STMT_TIMESTAMP >= (SELECT MAX(IQA0610.WLXT009.WLX_TIMESTAMP)
FROM IQA0610.WLXT009) - 1 HOURS THEN DECIMAL(A.CPU_TIME * K.SCALE_ADJ * 1.000000 / 1E+06, 12, 6) ELSE DECIMAL(A.CPU_TIME / (CASE K.SCALE_ADJ WHEN 0 THEN 1 ELSE K.SCALE_ADJ END) * 1.000000 / 1E+06,
12, 6) AS CPU_TIME_ADJ, A.GETP_OPERATIONS, DECIMAL((A.GETP_OPERATIONS * 1E+02) / (CASE W.DGETP_OPERATIONS WHEN 0 THEN 1 ELSE W.DGETP_OPERATIONS END),
12, 6) AS PCT_GETP_OPERATIONS, CASE WHEN A.STMT_TIMESTAMP >= (SELECT MAX(IQA0610.WLXT009.WLX_TIMESTAMP)
FROM IQA0610.WLXT009) - 1 HOURS THEN BITINT(A.GETP_OPERATIONS * K.SCALE_ADJ) ELSE BITINT(A.GETP_OPERATIONS / (CASE K.SCALE_ADJ WHEN 0 THEN 1 ELSE K.SCALE_ADJ END)) END AS GETP_OPERATIONS_ADJ,
DECIMAL((A.ELAPSE_TIME * 1.000000) / 1E+06, 12, 6) AS ELAPSED_TIME,
DECIMAL((A.ELAPSE_TIME * 1E+02) / (CASE W.DELAPSE_TIME WHEN 0 THEN 1 ELSE W.DELAPSE_TIME END),
12, 6) AS PCT_ELAPSED_TIME, CASE WHEN A.STMT_TIMESTAMP >= (SELECT MAX(IQA0610.WLXT009.WLX_TIMESTAMP)
FROM IQA0610.WLXT009) - 1 HOURS THEN DECIMAL(A.ELAPSE_TIME * K.SCALE_ADJ * 1.000000 / 1E+06, 12, 6) ELSE DECIMAL(A.ELAPSE_TIME / (CASE K.SCALE_ADJ WHEN 0 THEN 1 ELSE K.SCALE_ADJ END) * 1.000000 / 1E+06,
12, 6) AS ELAPSED_TIME_ADJ, A.EXECUTIONS, DECIMAL((A.EXECUTIONS * 1E+02) / (CASE W.DEXECUTIONS WHEN 0 THEN 1 ELSE W.DEXECUTIONS END),
12, 6) AS PCT_EXECUTIONS, CASE WHEN A.STMT_TIMESTAMP >= (SELECT MAX(IQA0610.WLXT009.WLX_TIMESTAMP)
FROM IQA0610.WLXT009) - 1 HOURS THEN BITINT(A.EXECUTIONS * K.SCALE_ADJ) ELSE BITINT(A.EXECUTIONS / (CASE K.SCALE_ADJ WHEN 0 THEN 1 ELSE K.SCALE_ADJ END)) END AS EXECUTIONS_ADJ
FROM IQA0610.WLXT009 A, IQA0610.WLXT009 K
WHERE A.STMT_TIMESTAMP >= (SELECT MAX(IQA0610.WLXT009.WLX_TIMESTAMP)
FROM IQA0610.WLXT009) - 1 HOURS
AND ((1 = 1)
AND (WLX_TIMESTAMP = (SELECT MAX(IQA0610.WLXT009.WLX_TIMESTAMP)
FROM IQA0610.WLXT009))))))
```

Database type: Undefined, not connected

Writable Insert 1:1

Building up a SQL workload warehouse (WLX)

Compare view:
Select any two SQLs
to generate graphs

The screenshot displays the Eclipse Database Development environment. The main window shows a table with performance metrics for various SQL statements. A 'Compare view' window is open, comparing two selected SQL statements (Selection 1 and Selection 2) across several metrics. The 'Selection details' pane at the bottom shows the column names and the corresponding SQL queries for both selections.

Sum of Executions	Sum of CPU Time	Average CPU Time	Percentage CPU Time	Sum of GETPAGES
85	307.602598	3.618854	13.55	2.798
76	199.839950	2.629473	8.80	1.947851
76	199.839950	2.629473	8.80	1.947851
76	199.839950	2.629473	8.80	1.947851
76	199.839950	2.629473	8.80	1.947851
76	199.839950	2.629473	8.80	1.947851
8	172.930322	21.616290	7.62	555,208

Sum of Executions	Sum of CPU Time	Average CPU Time	Percentage CPU Time	Sum of GETPAGES	Average G
85	307.602598	3.618854	13.55	2.798277	
76	199.839950	2.629473	8.80	1.947851	2
76	199.839950	2.629473	8.80	1.947851	2
76	199.839950	2.629473	8.80	1.947851	2
76	199.839950	2.629473	8.80	1.947851	2
76	199.839950	2.629473	8.80	1.947851	2

Selection value 1	Selection value 2
76	2
199.839950	158.202192
2.629473	79.101096
8.80	6.97
1.947851	173.132
25,629	86,566
13.95	1.24
299.131558	183.349670

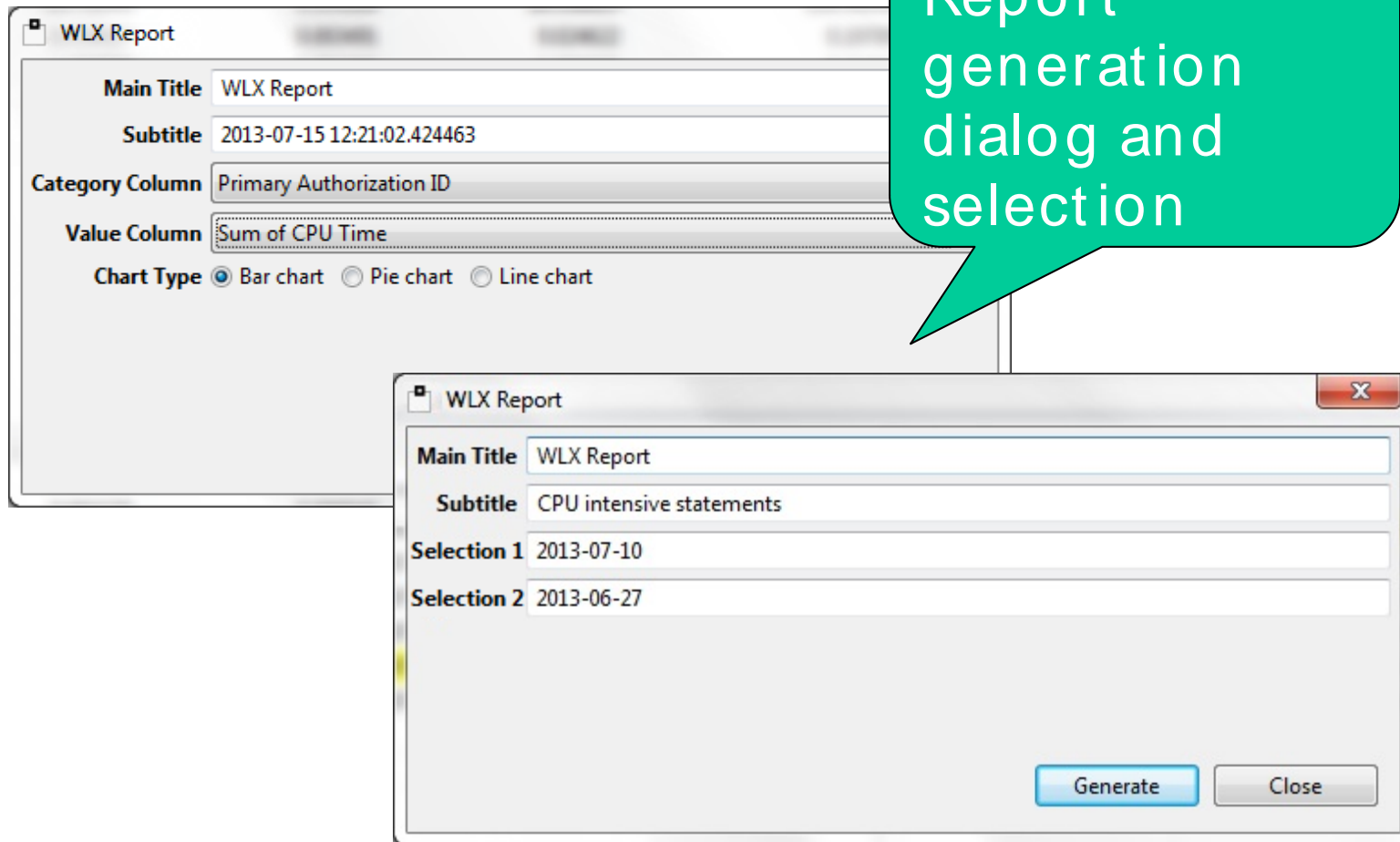
Value 1 SQL:

```
WITH  
INPUT (WLX_TIMESTAMP, STMT_TIMESTAMP, STMT_ORIGIN, STMT_TYPE,  
STMT_ID, STMT_GROUP_SSID, EXECUTIONS, ELAPSE_TIME, CPU_TIME,  
GRP OPERATIONS) AS
```

Value 2 SQL:

```
WITH  
PRIME KEY (WLX_TIMESTAMP, STMT_GROUP_SSID, STMT_ID, STMT_ORIGIN,  
STMT_TIMESTAMP, STMT_TYPE, SCALE_ADJ) AS  
(SELECT WLX_TIMESTAMP, STMT_GROUP_SSID, STMT_ID, STMT_ORIGIN, STMT_TIMESTAMP,
```

Building up a SQL workload warehouse (WLX)



The image shows two overlapping windows for generating a report. The top window, titled 'WLX Report', contains the following fields and options:

- Main Title:** WLX Report
- Subtitle:** 2013-07-15 12:21:02.424463
- Category Column:** Primary Authorization ID
- Value Column:** Sum of CPU Time
- Chart Type:** ☒ Bar chart ☐ Pie chart ☐ Line chart

A green speech bubble points to the top window with the text: "Report generation dialog and selection".

The bottom window, also titled 'WLX Report', contains the following fields and buttons:

- Main Title:** WLX Report
- Subtitle:** CPU intensive statements
- Selection 1:** 2013-07-10
- Selection 2:** 2013-06-27
- Buttons:** Generate, Close

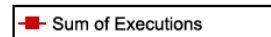
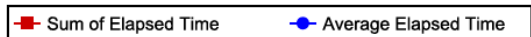
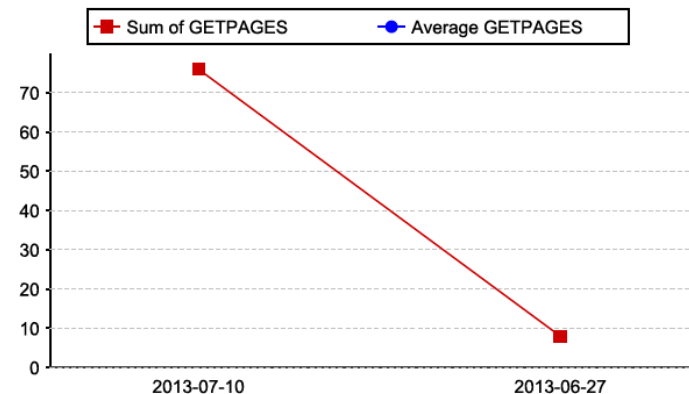
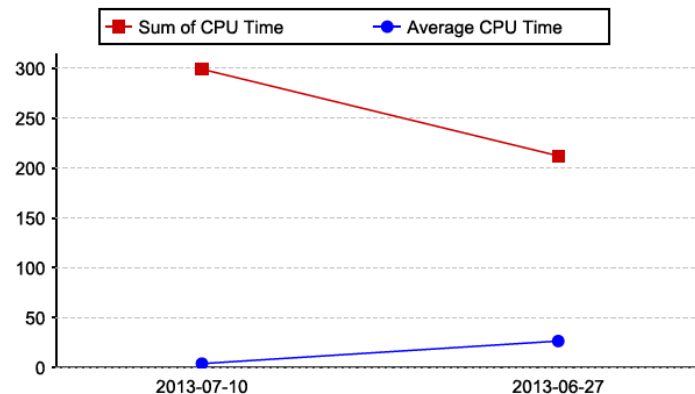
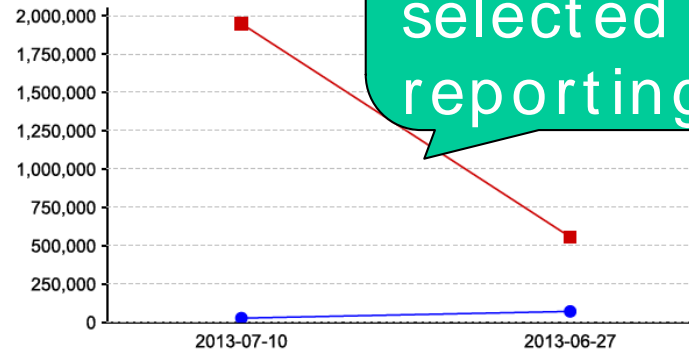
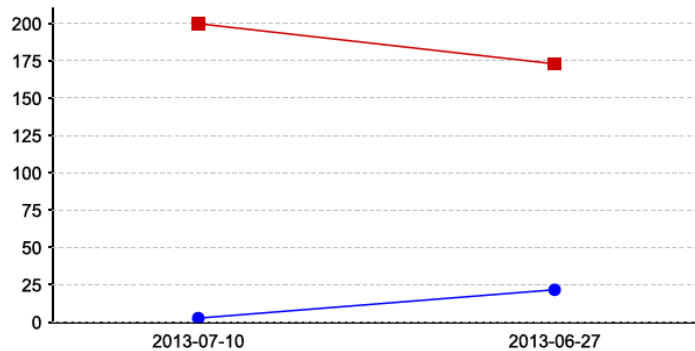
Building up a SQL workload warehouse (WLX)

WLX Report

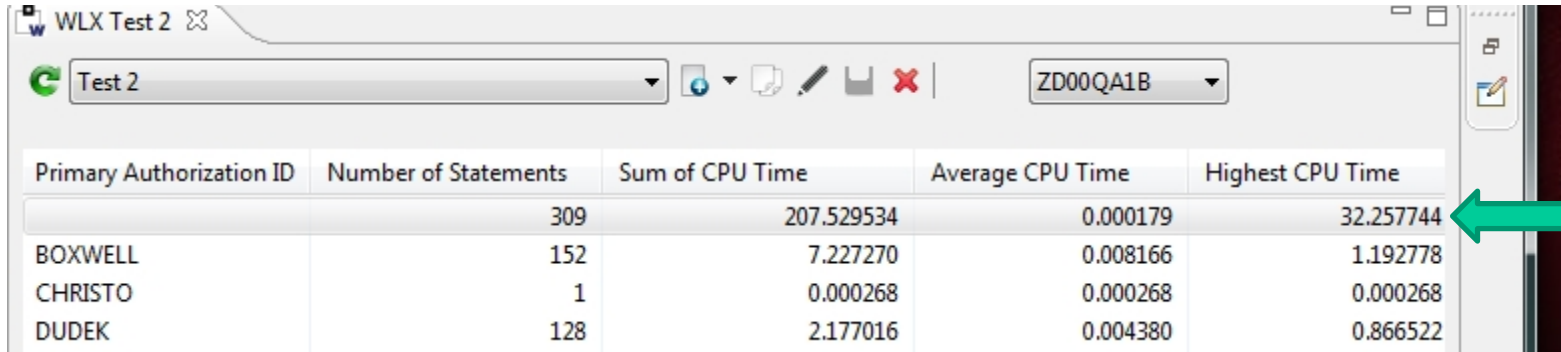
CPU intensive statements

Compare view

Output of
the
selected
reporting



Building up a SQL workload warehouse (WLX)



WLX Test 2

Test 2

ZD00QA1B

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

Primary Authorization ID

Number of Statements

Sum c

Open

309

152

1

128

377

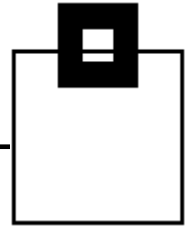
query1

ZD00QA1B

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

Here we have found a bad guy! STOGROUP SQL

Building up a SQL workload warehouse (WLX)

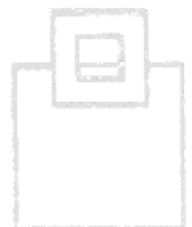


Now we need to see what it is doing...

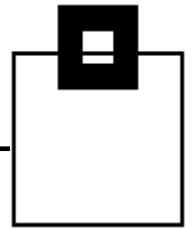
Package or Program	The Collection ID	CPU Time	Elapsed Time
MD2DB06	RTDX0510_PTFTOOL	22.257744	34.707177
MD2DB06	RTDX0510_PTFTOOL	274303	38.114398
M2DBKE09	RTDX0510_PTFTOOL	21.150725	25.142056
MDB2DB06	RTDX0510_PTFTOOL	20.025189	22.226928
MDB2DB26	RTDX0510_PTFTOOL	16.014742	17.335210

```
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
```

Aha! This looks like a great candidate for LEFT OUTER JOIN processing



Building up a SQL workload warehouse (WLX)



WLX Application Select November

Application Select November ZD00QA1B

Primary Auth...	Number ...	Sum of CPU Time...	Average CPU Time...	Highest CPU Time...	Sum of Elapsed Time...	Aver
	26	698.802486	0.155774	540.866522	920.769361	
	20	253.479869	0.121282	187.209637	1,368.946780	
	9	181.070538	0.072982	171.671033	687.489428	
	4	0.161344	0.032268	0.124293	27.289669	
	44	236.363696	0.025426	160.597919	1,263.675875	

Application Usage figures

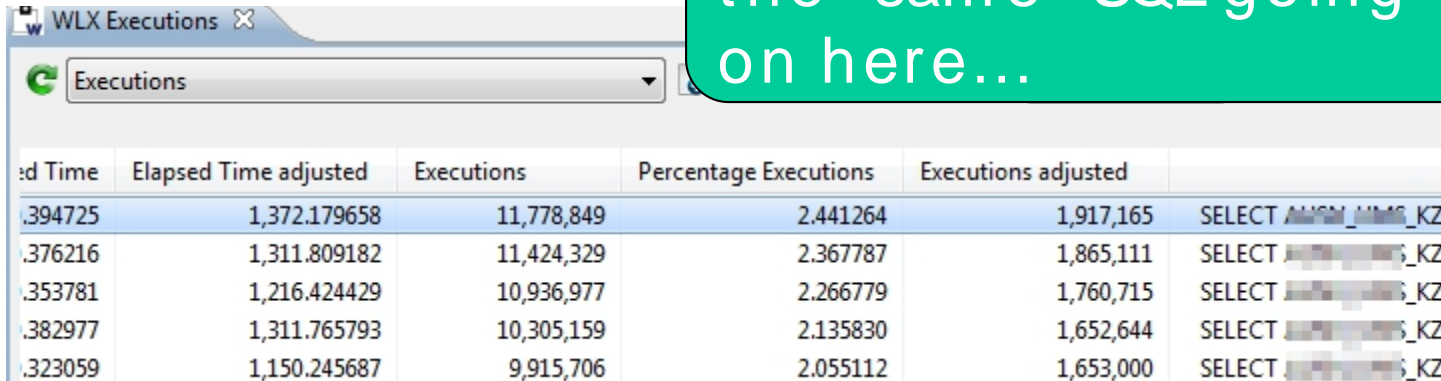
CPU Time	Percentage CPU Time	CPU time adjusted	GETPAGES	Percentage GETPAGES
4,307.680309	3.007077	822.250748	1,105,726,368	7.340946
4,375.207060	3.054216	883.929817	1,093,323,039	7.258600
4,163.849663	2.906673	798.564742	1,060,248,627	7.039018
3,950.536953	2.757765	796.167106	1,018,411,425	6.761259
3,957.145128	2.762378	768.958353	1,010,413,994	6.708164
3,688.444643	2.574805	712.819450	942,433,406	6.256839
8,691.703040	6.067447	396.705305	739,538,587	4.909815
8,728.374272	6.093047	428.755	732,257,998	4.861479
1,406.492725	0.981835			1.159908
3,163.883163	2.208623			0.345301

Adjusted data



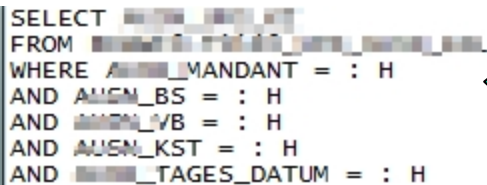
Building up a SQL workload warehouse (WLX)

Lots of executions for the *same* SQL going on here...



The screenshot shows a window titled 'WLX Executions' with a sub-tab 'Executions'. It displays a table with the following columns: 'Elapsed Time', 'Elapsed Time adjusted', 'Executions', 'Percentage Executions', 'Executions adjusted', and a column for the SQL query. The first row is highlighted in blue.

Elapsed Time	Elapsed Time adjusted	Executions	Percentage Executions	Executions adjusted	SQL Query
0.394725	1,372.179658	11,778,849	2.441264	1,917,165	SELECT AUSEN_MANDANT_KZ
0.376216	1,311.809182	11,424,329	2.367787	1,865,111	SELECT AUSEN_MANDANT_KZ
0.353781	1,216.424429	10,936,977	2.266779	1,760,715	SELECT AUSEN_MANDANT_KZ
0.382977	1,311.765793	10,305,159	2.135830	1,652,644	SELECT AUSEN_MANDANT_KZ
0.323059	1,150.245687	9,915,706	2.055112	1,653,000	SELECT AUSEN_MANDANT_KZ



```
SELECT AUSEN_MANDANT_KZ  
FROM AUSEN_MANDANT_KZ  
WHERE AUSEN_MANDANT = : H  
AND AUSEN_BS = : H  
AND AUSEN_VB = : H  
AND AUSEN_KST = : H  
AND AUSEN_TAGES_DATUM = : H
```

Why so often? Discussed with development and find it is a „design“ problem... The query could be run earlier and then only a few times a day instead of millions!

Building up a SQL workload warehouse (WLX)

Often run
BAD SQL

This
workload
splits
into
two
SQLs

Primary Authorization ID	Number of Statements	Sum of CPU Time	Average CPU Time	Highest CPU Time
WLF010	2	51.785070	0.000159	27.820950

Primary Authorization ID	CPU Time	Elapsed Time	Executions	GETPAGES	Synchronous Buff
WLF010	27.820950	33.148750	161,910	2,357,445	
WLF010	23.964120	28.562837	162,626	2,366,231	

Which have this SQL:
Six UNIONS...
DBA rewrote down
to one SELECT and
IN usage

```
SELECT ...  
FROM(  
SELECT MATRIX, 1 as STUFE  
FROM RKNWCO.T4035_MARKT_DEF  
WHERE WM_REGION = ?  
AND WM_REGION_H = ?  
AND WM_REGIONummer = ?  
AND WM_REGION_Nr = ?  
AND WM_REGION_SFALL = ?  
UNION  
SELECT MATRIX, 2 as STUFE  
FROM RKNWCO.T4035_MARKT_DEF  
WHERE WM_REGION = ?  
AND WM_REGION_H = ?  
AND WM_REGIONummer = ?  
AND WM_REGION_Nr = ?  
AND WM_REGION_SFALL = ?  
UNION  
SELECT MATRIX, 3 as STUFE  
FROM RKNWCO.T4035_MARKT_DEF  
WHERE WM_REGION = ?  
AND WM_REGION_H = ?  
AND WM_REGIONummer = 0  
AND WM_REGION_Nr = ?
```

Building up a SQL workload warehouse (WLX)

Index maintenance costs

Description: IndexCostsAnalysis20130807165706

Table creator: IQA0610

Table name: IQATW009

Period 1

Date/Time from: 29.07.2013 00:00

Date/Time to: 29.07.2013 23:59

Period 2

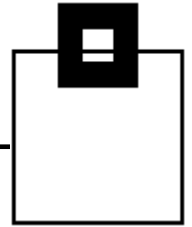
Date/Time from: 01.08.2013 00:00

Date/Time to: 01.08.2013 23:59

OK Cancel

Calendar
date/time
for
from <->
to index
analysis

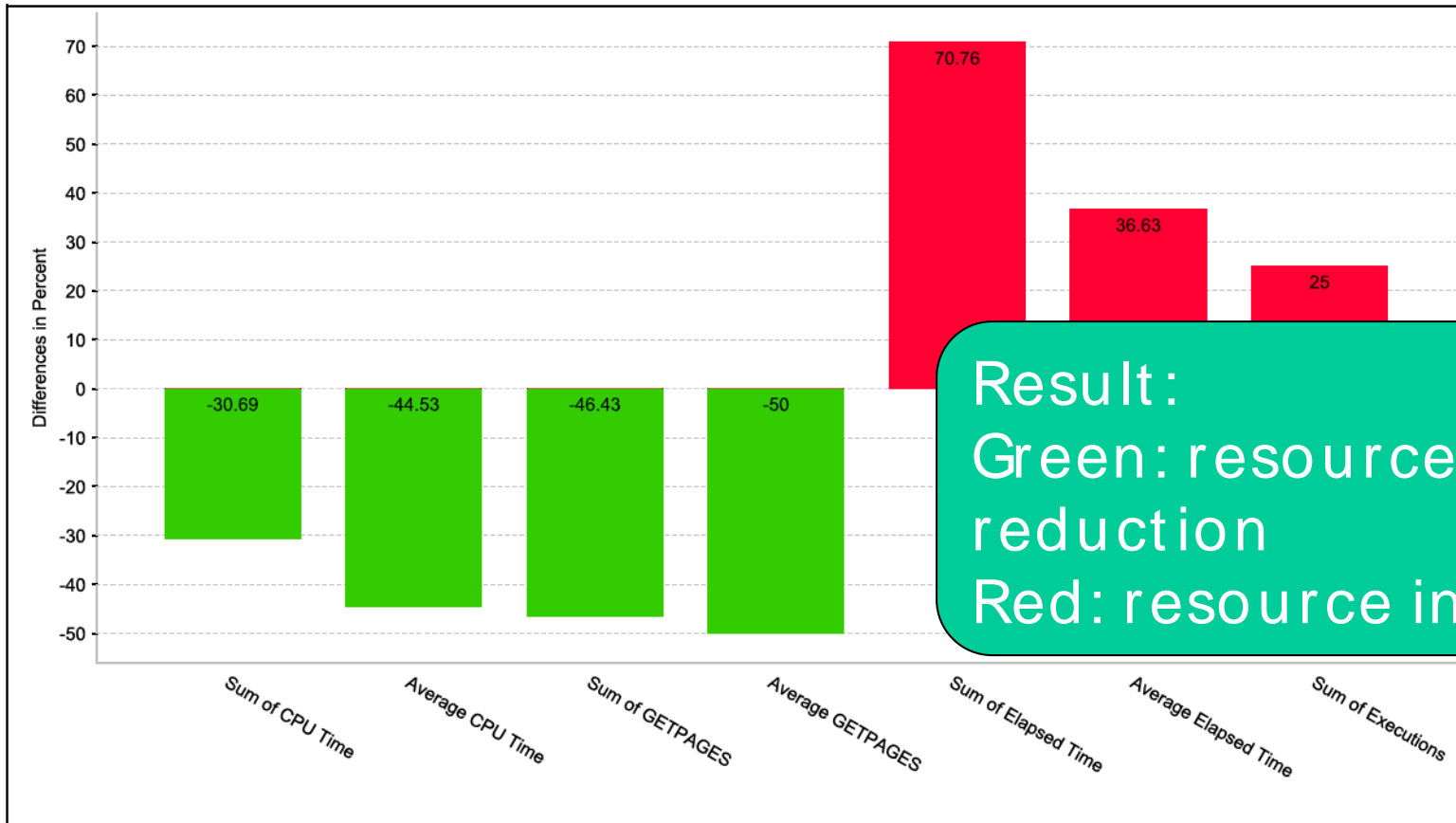
Building up a SQL workload warehouse (WLX)



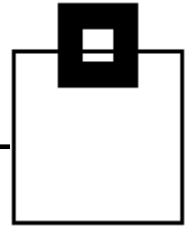
WLX Report

Index Maintenance Costs


Index maintenance costs for table: IQA0610.IQATW008

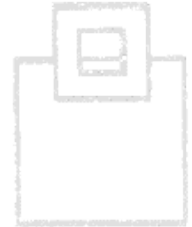
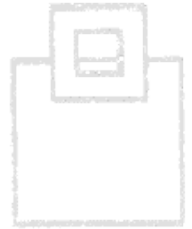


Building up a SQL workload warehouse (WLX)

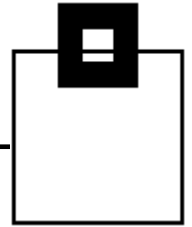


Application Development:

- Application Workload Analysis: E.g. which machine load is produced by a certain Application?
- Explain Tool link (e.g.  SQL Performance Expert, 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

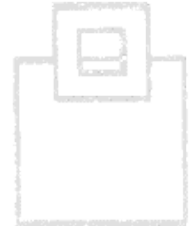


Building up a SQL workload warehouse (WLX)

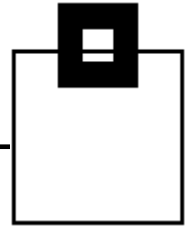


Workload/ Performance management:

- 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



Building up a SQL workload warehouse (WLX)

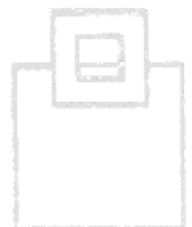


Database Administration:

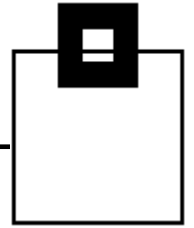
- Find never used Objects (Tables, Indexes, and Tablespaces)
- Find never executed Packages

Audit and Security:

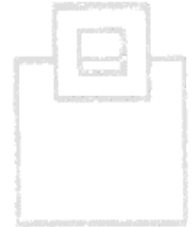
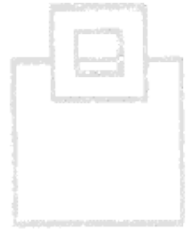
- AUDIT tables being accessed
- AUDIT DB2 data being accessed
- AUDIT data manipulation (insert/update/delete)
- See where updates came from (inside or outside the local network)
- See where data is being accessed from (IP address, intra-/extranet, etc.)
- SQL Text Analysis for free text search (BIF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



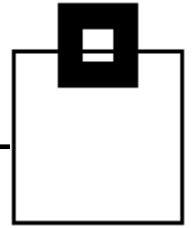
Real life example ...



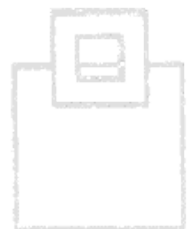
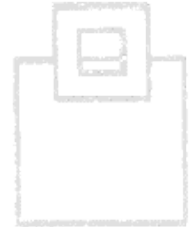
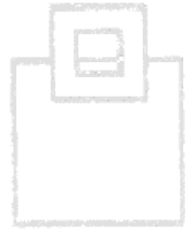
... Quick solution creation and control



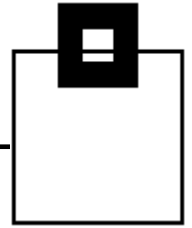
July Problem...



- Thursday night – Production staging of numerous packages
- Friday – Thread time-outs, deadlocks, bad news!
 - First thought: Must be caused by a bad package – All packages checked for bad access paths and everything found was OK
 - Second thought: Open Priority two ticket at IBM in case it is a DB2 problem
 - JAVA trace showed a long running SQL appearing often
 - REORG with inline RUNSTATs the biggest tables used in that SQL
 - Reduce number of available servers to stop problem getting worse (internal throttling of transactions)
 - Full panic mode now enabled
- Saturday – Call in senior DBA from vacation



July Problem...

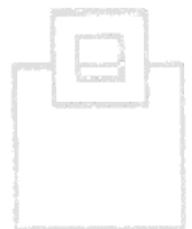
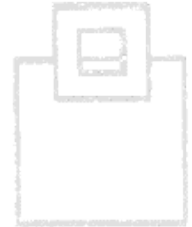
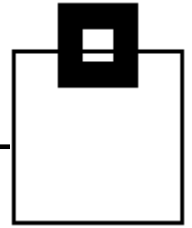


- Saturday Morning 9am – DBA uses WLX to compare SQL workload from Thursday with Friday – Sees bad guy instantly
- 9:10am – DBA uses BIX to confirm that an access path change has caused the problem – Nothing to do with staging, Nothing to do with the large tables
- 9:15am – DBA creates a new “virtual” index using SPX and re-tests – Access path switches back to old correct method
- 9:30am – DBA creates a new index, RUNSTATS it, everything is fixed and the systems are running sweetly again
- 9:45am – DBA goes back on vacation

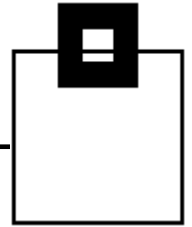


July Problem...

- Monday Morning – Investigation by DBA group starts
- noon - DBA group finds that a badly timed RUNSTATs on Thursday night caused the access path change. New index is OK and in fact the old index can now be dropped
- 3:30pm – Report written for CIO, Problem closed at IBM

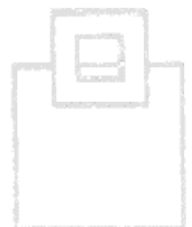


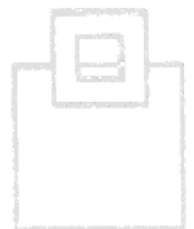
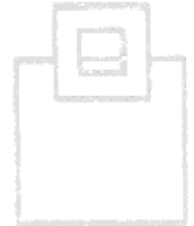
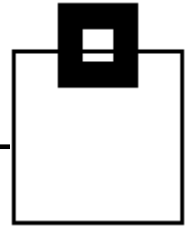
Appendix



■ Problem list :

- PM77114 DB2 10 UK91560 – Abend S04E
- PM78143 DB2 10 UK93065 – SOS – HIPER
- PM80371 DB2 10 UK93127 – Serviceability for SHTE
- PM83370 DB2 10 UK94511 – Fields TB and IX sometimes wrong
- PM85376 DB2 10 UK96310 – Abend S04E
- PM89121 DB2 10 UK95683 – Storage leak leading to abend – HIPER
- PM91159 DB2 10 UK97197 – Improve accuracy of IFCID 316 and 401
- PM92610 DB2 11 UK96376 – Abend with IFCID 400 or 401
- PM93437 DB2 11 UK97361 – IFCID 316 fields length value problem
- PM97922 DB2 10 UI12375 DB2 11 UI12376 – Invalid or empty IFCID 316
- PI07461 OPEN – Inconsistent 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 OPEN MISSING IFCID401





Ulf Heinrich
SEGUS Inc
u.heinrich@segus.com