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.
   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:

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

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

*This IFCID is not really an IFCID but more of a “switch“ to enable externalization of static SQL metrics
Data Data Data

DSC and EDM provide detailed workload insights:

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

- 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
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 SIMPLE 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. # 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 LIST OVERFLOWED TO A WORK FILE BECAUSE NO RID POOL STORAGE WAS AVAILABLE TO HOLD THE LIST OF RIDs*. # OF TIMES 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 THE 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 LOG REQUESTS*. ACCUMULATED WAIT TIME FOR STORAGE LATCHES*. ACCUMULATED WAIT TIME FOR DRAM LOCKS*. ACCUMULATED WAIT TIME FOR DRAMS 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 WHILE ACTIVITY THAT IS DONE BY ANOTHER THREAD.


ENVIRONMENTAL  REFERENCED TABLE NAME. FOR STATEMENTS THAT REFERENCED 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 NAMEx. BIND OPTIONS: ISOLATION, CURRENTDATA, AND DYNAMICRULES. SPECIAL REGISTER VALUES: CURRENT DEGREE, CURRENT RULES, AND CURRENT PRECISION. WHETHER THE STATEMENT CURSOR IS A HARD 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.
Building up a SQL workload warehouse (WLX)

① WLX STC

Mainframe Engine

24 x 7 SQL Workload Capture

WLX

WLX Started Task or iterative job

IFCID

Iterative Workload Processing

Select

Capture processing

DB2

DB2 DSNMSTR System Service Address Space

WLX

DB2 Catalog/RTS

WLX Explain Tables

Insert, Update

WLX Workload Warehouse Repository

Workstation Engine

Graphical User Interface

WLX Report

Type 4 Java

② WLX workload processing engine

③ WLX GUI interface

© 2014 SOFTWARE ENGINEERING GMBH and SEGUS Inc.
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)

<table>
<thead>
<tr>
<th>WLX Key</th>
<th>Number of Statements</th>
<th>Total</th>
<th>Sum of CPU Time</th>
<th>Average CPU Time</th>
<th>Highest CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-06-14 12:24:58.497164</td>
<td>4,391</td>
<td>6,304.879804</td>
<td>0.008477</td>
<td>1,145.15</td>
<td></td>
</tr>
<tr>
<td>2013-06-14 13:57:50.529597</td>
<td>4,319</td>
<td>6,029.83625</td>
<td>0.008064</td>
<td>1,145.15</td>
<td></td>
</tr>
<tr>
<td>2013-06-27 19:57:00.400711</td>
<td>6,668</td>
<td>2,070.188433</td>
<td>0.003072</td>
<td>131.14</td>
<td></td>
</tr>
<tr>
<td>2013-07-15 12:21:02.424463</td>
<td>3,190</td>
<td>1,233.649658</td>
<td>0.002765</td>
<td>131.14</td>
<td></td>
</tr>
<tr>
<td>2013-07-24 18:49:02.493977</td>
<td>1,988</td>
<td>684.217708</td>
<td>0.003775</td>
<td>131.14</td>
<td></td>
</tr>
<tr>
<td>2013-07-25 15:54:11.12454</td>
<td>1,994</td>
<td>705.081217</td>
<td>0.003675</td>
<td>131.14</td>
<td></td>
</tr>
<tr>
<td>2013-08-03 15:49:58.17302</td>
<td>5,480</td>
<td>77,841.577557</td>
<td>0.091512</td>
<td>37,840.02</td>
<td></td>
</tr>
<tr>
<td>2013-09-02 10:30:02.98741</td>
<td>1,368</td>
<td>774,172.60</td>
<td>0.007616</td>
<td>268.01</td>
<td></td>
</tr>
<tr>
<td>2013-09-03 11:52:26.47833</td>
<td>1,568</td>
<td>863,880.60</td>
<td>0.003561</td>
<td>302.01</td>
<td></td>
</tr>
<tr>
<td>2013-09-09 13:16:30.26273</td>
<td>1,676</td>
<td>900,157.894</td>
<td>0.003452</td>
<td>302.01</td>
<td></td>
</tr>
<tr>
<td>2013-09-10 13:14:05.30831</td>
<td>1,125</td>
<td>165,261.44</td>
<td>0.002754</td>
<td>34.67</td>
<td></td>
</tr>
</tbody>
</table>

Workload KPIs – left hand side
Building up a SQL workload warehouse (WLX)

Workload KPIs – right hand side

© 2014 SOFTWARE ENGINEERING GMBH and SEGUS Inc.
Building up a SQL workload warehouse (WLX)

Spider diagram of three application extracts
Building up a SQL workload warehouse (WLX)

Example of our Eclipse embedded SPX plug-in
Building up a SQL workload warehouse (WLX)

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
Building up a SQL workload warehouse (WLX)

Compare view:
Select any two SQLs to generate graphs

<table>
<thead>
<tr>
<th>Selection 1</th>
<th>Selection 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum of Executions</td>
<td>Sum of CPU Time</td>
</tr>
<tr>
<td>85</td>
<td>109.839050</td>
</tr>
<tr>
<td>76</td>
<td>109.839050</td>
</tr>
<tr>
<td>76</td>
<td>109.839050</td>
</tr>
<tr>
<td>76</td>
<td>109.839050</td>
</tr>
</tbody>
</table>

Selection details:

- Column name
- Sum of Executions
- Sum of CPU Time
- Average CPU Time
- Percentage CPU Time
- Average GETPAGES
- Percentage GETPAGES
- Sum of Elapsed Time
- Value 1
- Value 2
Building up a SQL workload warehouse (WLX)

Report generation dialog and selection
Building up a SQL workload warehouse (WLX)

Output of the selected reporting

© 2014 SOFTWARE ENGINEERING GMBH and SEGUS Inc.
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...

Aha! This looks like a great candidate for LEFT OUTER JOIN processing
Building up a SQL workload warehouse (WLX)

### Application Usage figures

<table>
<thead>
<tr>
<th>Primary Auth</th>
<th>Number</th>
<th>Sum of CPU Time</th>
<th>Average CPU Time</th>
<th>Highest CPU Time</th>
<th>Sum of Elapsed Time</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>26</td>
<td>698.802486</td>
<td>0.155747</td>
<td>540.866522</td>
<td>920.769361</td>
<td></td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>253.479869</td>
<td>0.121892</td>
<td>187.209637</td>
<td>1,368.946780</td>
<td></td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>181.070538</td>
<td>0.072982</td>
<td>171.671033</td>
<td>687.489428</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>0.161344</td>
<td>0.032268</td>
<td>0.124293</td>
<td>27.289669</td>
<td></td>
</tr>
<tr>
<td></td>
<td>44</td>
<td>236.363969</td>
<td>0.052426</td>
<td>160.597919</td>
<td>1,263.675875</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CPU Time</th>
<th>Percentage CPU Time</th>
<th>CPU time adjusted</th>
<th>GETPAGES</th>
<th>Percentage GETPAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>4,307.680309</td>
<td>3.007077</td>
<td>822.250748</td>
<td>1,105,726,368</td>
<td>7.340946</td>
</tr>
<tr>
<td>4,375.207060</td>
<td>3.054216</td>
<td>883.929817</td>
<td>1,093,323,039</td>
<td>7.258600</td>
</tr>
<tr>
<td>4,163.849663</td>
<td>2.906673</td>
<td>798.564742</td>
<td>1,060,248,627</td>
<td>7.039018</td>
</tr>
<tr>
<td>3,950.536953</td>
<td>2.757765</td>
<td>796.167106</td>
<td>1,018,411,425</td>
<td>6.761259</td>
</tr>
<tr>
<td>3,957.145128</td>
<td>2.762378</td>
<td>768.958353</td>
<td>1,010,413,994</td>
<td>6.708164</td>
</tr>
<tr>
<td>3,688.444643</td>
<td>2.574805</td>
<td>712.819450</td>
<td>942,433,406</td>
<td>6.256839</td>
</tr>
<tr>
<td>8,691.703040</td>
<td>6.067447</td>
<td>396.705305</td>
<td>739,538,587</td>
<td>4.909815</td>
</tr>
<tr>
<td>8,728.374272</td>
<td>6.093047</td>
<td>434,796.576</td>
<td>732,257,998</td>
<td>4.861479</td>
</tr>
<tr>
<td>1,406.492725</td>
<td>0.981835</td>
<td>1,159,908</td>
<td>1,159,908</td>
<td>0.345301</td>
</tr>
</tbody>
</table>

Adjusted data
Building up a SQL workload warehouse (WLX)

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

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

Which have this SQL:
Six UNIONS…
DBA rewrote down to one SELECT and IN usage
Building up a SQL workload warehouse (WLX)

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

Result:
Green: resource reduction
Red: resource increase
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