Exploiting DB2 10 new IFCID Technology

Using WorkLoadExpert (WLX)

Roy Boxwell

February 26th 2014

Agenda

- DB2 10 new and enhanced IFCIDs
- WLX architecture
- Live demo
- JAVA interest
- Audit
- Q&A

DB2 10 new and enhanced IFCIDs

What is enhanced or new:

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

② ▪ New IFCIDs 400* and 401 for EDM pool data – let's call it the Static Statement Cache (SSC)
  - 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 a switch to enable the externalization of static SQL metrics

DB2 10 new and enhanced IFCIDs

- The new and enhanced IFCIDs deliver all the important info to enable smart analysis of SQL workload
- The needed IFCIDs are cheap to get and can be very quickly processed via a STC (24x7 to catch all thrown IFCIDs)
- These IFCIDs can be snapped in a continous way, which enables a long period of time to be kept
- Redundant data can be condensed and aggregated

DB2 10 new and enhanced IFCIDs

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.

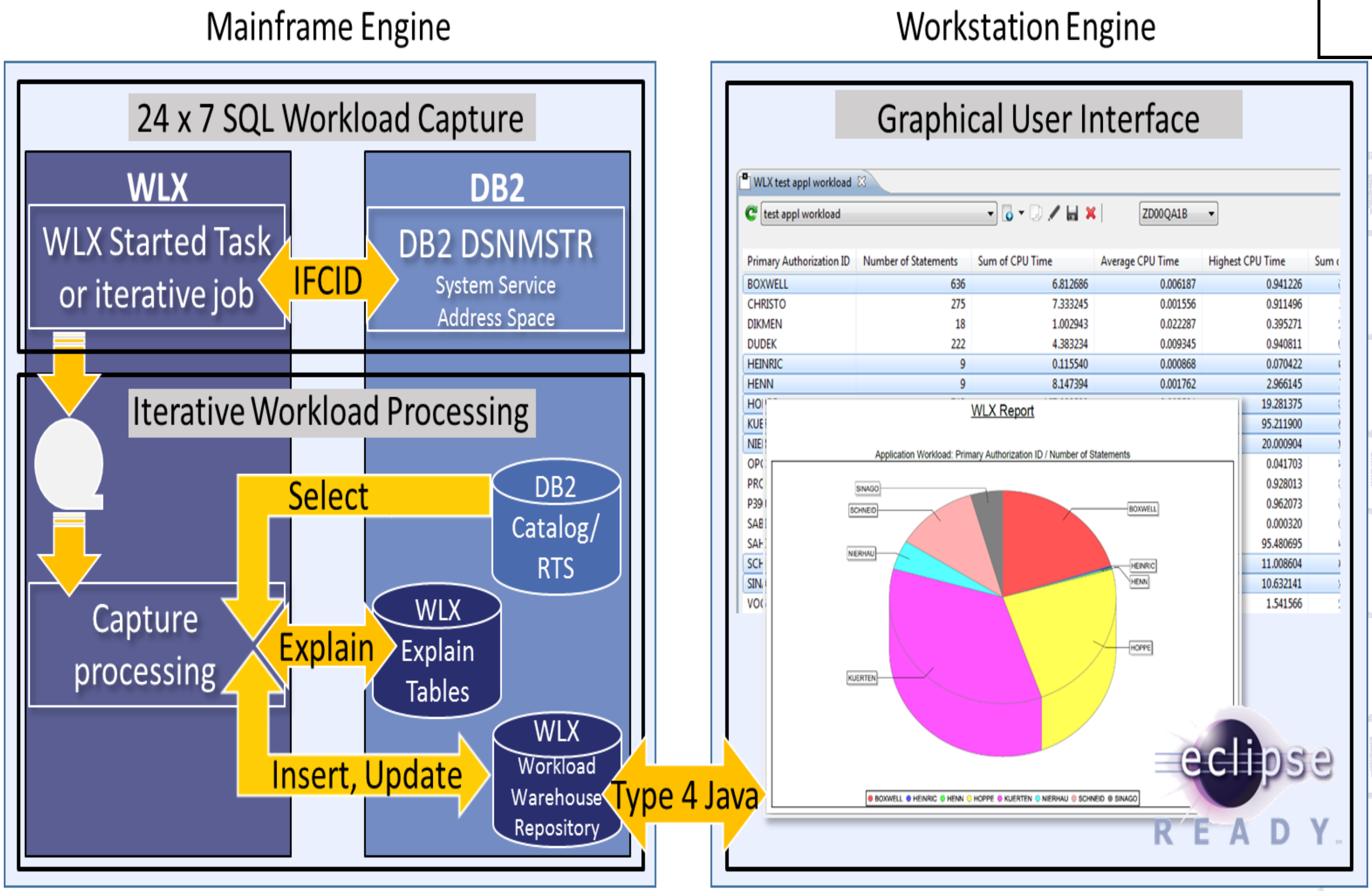
O Counters # OF TIMES THAT A RID LIST WAS NOT USED BECAUSE THE # OF RIDS EXCEEDED ONE OR MORE INTERNAL DB2 LIMITS, AND THE # OF RID BLOCKS EXCEEDED THE VALUE OF SUBSYSTEM PARAMETER MAXTEMPS_RID. # OF TIMES THAT A RID LIST WAS NOT USED BECAUSE NOT ENOUGH STORAGE WAS AVAILABLE TO HOLD THE RID LIST, OR 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*. # OF TIMES THAT RID LIST RETRIEVAL FOR MULTIPLE INDEX ACCESS WAS NOT DONE BECAUSE DB2 COULD NOT DETERMINE THE OUTCOME OF INDEX ANDING OR ORING*.

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.

WLX Architecture


Mainframe Engine
Workstation Engine
24 x 7 SQL Workload Capture
WLX
WLX Started Task or iterative job
IFCID
DB2
DB2 DSNMSTR
System Service Address Space
Iterative Workload Processing
Select
DB2 Catalog/ RTS
Capture processing
Explain
WLX Explain Tables
Insert, Update
WLX Workload Warehouse Repository
Type 4 Java
Graphical User Interface
WLX test appl workload
test appl workload
ZD00QA1B
WLX Report
Application Workload: Primary Authorization ID / Number of Statements
SINAGO
SCHNEID
NIERHAU
KUERTEN
BOXWELL
HEINRIC
HENN
HOPPE
● BOXWELL ● HEINRIC ● HENN ● HOPPE ● KUERTEN ● NIERHAU ● SCHNEID ● SINAGO
eclipse
READY

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

Live Demo

So that‘s enough PPT – Over to the GUI world!

JAVA interest

Eager vs Lazy Loader detection

(JPA – Java Persistence API)

Various JPAs are used and some have Eager Loading or Lazy Loading as an option. Eager instantiates everything for everywhere which can be overkill! Lazy delays the instantiation until used.

As WLX has all of the SQL for a given application it can help make a recommendation of whether or not Eager is better than Lazy. Lazy is „normally“ better.

This needs code changing in Java of course and time to compare the application results before and after.

Audit

Example:

User BOXWELL manages to run three SQLs at the same time from three separate servers in Düsseldorf, München and Sao Paulo.

How is this possible? Has there been a password leak?

Who has run what type of query against my tables?

Questions???

Many thanks for your attention and now….