IDUG
The Worldwide DB2 User Community

EMEA Conference 2010

Quality Assurance for Dynamic SQL

Craig S. Mullins
Mullins Consulting, Inc. (for InSoft Software GmbH)

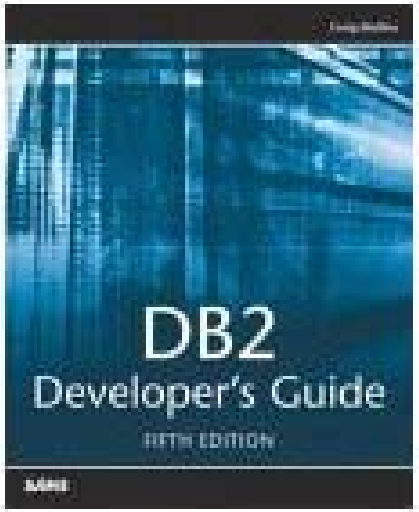
Session Code: V12
Tuesday 2010-Nov-09 @ 9:45 – 10:45 | Platform: DB2 for z/OS

Author

- **This presentation was prepared by:**

- Craig S. Mullins
- President & Principal Consultant

- **Mullins Consulting, Inc.**
- **15 Coventry Ct**
- **Sugar Land, TX 77479**
- **Tel: 281-494-6153**
- **Fax: 281.491.0637**
- **Skype: cs.mullins**
- **E-mail: craig@craigsmullins.com**



Agenda

- Dynamic SQL – The Basics
  - Static vs. Dynamic SQL
  - When to Use Dynamic vs. Static SQL
- Drivers of Dynamic SQL Growth
  - Packaged Applications (e.g. ERP)
  - Modern Application Development
    - GUI, web, etc.
- Performance Issues
  - Understanding the Dynamic Statement Cache
  - BIND options
- Solving Your Dynamic SQL Woes with InSoft's QA+

Solving Your Dynamic SQL Woes with InSoft's QA+

DB/IQ - Family

DB/IQ Family in Version 4.92

- ✉ QA Quality Assurance (Base Product)
- ✉ IA+ Index Administrator + (Add-On)
- ✉ PM PackMan - Package Management
- ✉ QA+ QA Plus (Add-On)
- ✉ WL+ WorkLoad Detector + (Add-On)

INSOFT SOFTWARE GMBH

Types of SQL

Functionality	
DCL	Control of data and security
DDL	Data definition
DML	Data manipulation

Execution Type	
Production	Planned
Ad hoc	Unplanned

Existence	
Embedded	Requires a program
Stand-alone	No program used

Dynamism	
Dynamic SQL	Changeable at run time
Static SQL	Unchangeable at run time

Dynamic SQL – Basics

Static SQL

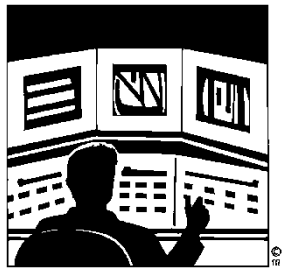
- Is unique to DB2
- Access paths established before execution
- Creates a consistent access path
- Requires a BIND to create SQL access paths

Dynamic SQL:

- Usually more flexible but access path can be inconsistent
- Not as common
  - At least in traditional COBOL applications
- A “mini-bind” is done automatically by DB2 before execution

Dynamic SQL versus Static SQL

Capability	Dynamic SQL	Static SQL
Access paths created at:	Run time	BIND time[1]
Access paths in PLAN_TABLE		✓
BIND not required before execution	✓	
Dynamic Statement Cache	✓	
Uses latest RUNSTATS[2]	✓	
SQL errors detected at BIND		✓
Authorization to tables not required		✓
Flexible: SQL can change	✓	
Supported in interpreted languages (e.g. REXX)	✓	



So to Summarize…

With Static SQL:

- **Beneficial** because Access paths are formulated at BIND time and EXECUTE authority on plans and packages eliminates need for privileges for all objects in the SQL
- **But** binding before running program can become burdensome and many of current development tools provide better support for dynamic APIs

With Dynamic SQL:

- **Beneficial** because IDEs support Java better, the Dynamic Statement Cache can alleviate performance penalty of re-formulating access paths and dynamic SQL will always use the latest RUNSTATS (which may produce a better access paths)
- **But** compiling statements each time they are executed increases total statement execution time and SQL errors may not be detected until the program is executed.

Static vs. Dynamic SQL: When To Use Each

- **Performance sensitivity of the SQL statement**
  - Dynamic SQL will incur a higher initial cost per SQL statement due to the need to prepare the SQL before use. But once prepared, the difference in execution time for dynamic SQL compared to static SQL diminishes.
- **Data uniformity**
  - Dynamic SQL can result in more efficient access paths than static SQL is whenever data is:
    1. Non-uniformly distributed. (e.g. cigar smokers skews male)
    2. Correlated (e.g. CITY, STATE, and ZIP_CODE data will be correlated)
- **Use of range predicates**
  - The more frequently you need to use range predicates (<, >, <=, >=, BETWEEN, LIKE) the more you should favor dynamic SQL.
  - The optimizer can take advantage of distribution statistics & histogram statistics to formulate better access paths because the actual range will be known.

When to Use Static vs. Dynamic SQL (continued)

- **Repetitious Execution**
  - As the frequency of execution increases, then you should favor static SQL (or perhaps dynamic SQL with local dynamic statement caching (KEEPDYNAMIC YES).
  - The cost of the PREPARE becomes a smaller and smaller percentage of the overall run time of the statement the more frequently it runs (if the cached prepare is reused).
- **Nature of Query**
  - When you need all or part of the SQL statement to be generated during application execution favor dynamic over static SQL.
- **Run Time Environment**
  - Dynamic SQL can be the answer when you need to build an application where the database objects may not exist at precompile time. Dynamic might be a better option than static specifying VALIDATE(RUN).
- **Frequency of RUNSTATS**
  - When your application needs to access data that changes frequently and dramatically, it makes sense to consider dynamic SQL.

Drivers of Dynamic SQL Growth

- Packaged applications use dynamic SQL
  - SAP R/3, Peoplesoft, Siebel, etc.
  - Easier to support multiple DBMSes that way

- Many newer applications use dyanmic SQL
  - Developed on distributed platforms and for the web
    - New developers are more familiar with GUI-based programming environments
    - Many of the current development tools provide better support for dynamic APIs (like JDBC), than they do for static SQL
    - Many developers never even sign on to the mainframe
      - Java and .NET developers

Let's Talk About Dynamic SQL Performance

- Many aspects of tuning dynamic SQL are similar to tuning static SQL
  - Find the problematic SQL
  - Determine who executed it and from which program
    - if possible
  - EXPLAIN the statement
  - Tune the statement
  - Repeat as needed

We'll speak about each of these on the ensuing slides.

- Building best practices for tuning dynamic SQL

Finding the Problematic Dynamic SQL

There are no DBRMs, packages, or access paths in PLAN_TABLE(s) to utilize.

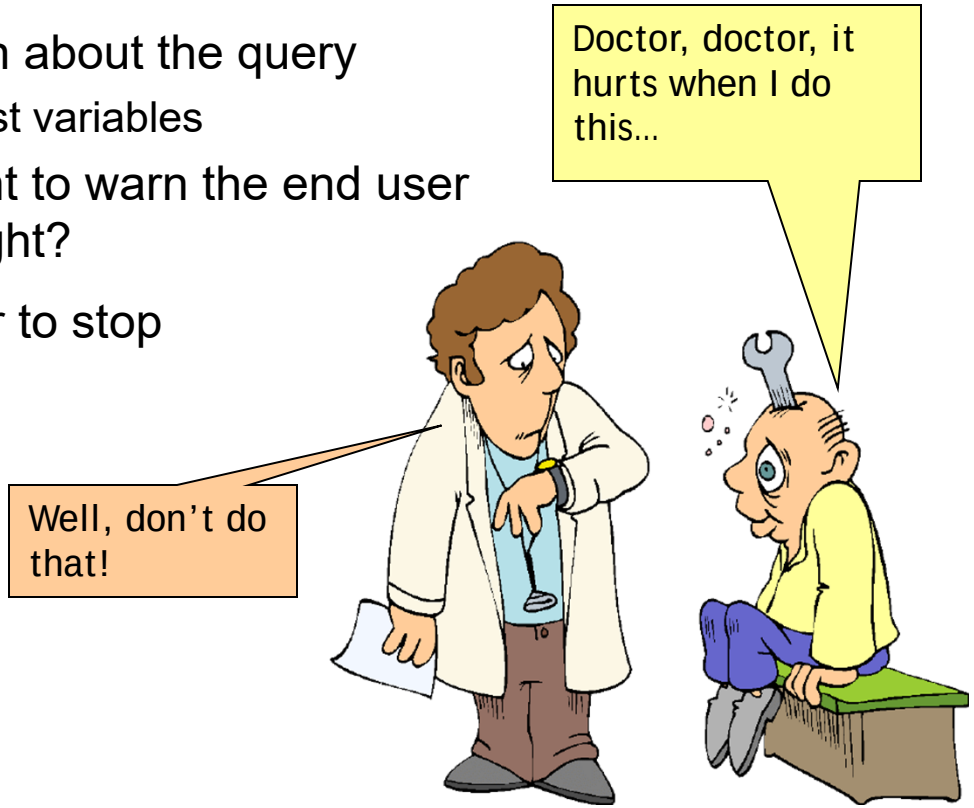
So, how do you find problems?

- A problem usually starts with a phone call
- Could also arise from performance reports and/or performance monitor(s)
  - Typically after a problem occurs, though
  - May require costly traces

Who Executed the Dynamic SQL?

Can you determine *who* executed the *problem* dynamic SQL statement(s)?

- Why would you want/need to know this?
  - Can help to gather information about the query
    - parameter markers and host variables
  - If it is still running you will want to warn the end user before you kill the thread… right?
  - Remediate by getting the user to stop running that particular query
    …or form of the query.



Who Executed the Dynamic SQL? (continued)

But it can be difficult to determine *who* executed dynamic SQL

- Why?
  - Much dynamic SQL comes from an application server
  - Single connection ID problem
- There are four client identifiers that *can* be passed to DB2 for z/OS when dynamic SQL is executed:
  - Client workstation name
  - Client transaction name
  - Client user
  - Client accounting information.

These can be set using DRDA or JDBC…

Tuning/Fixing Dynamic SQL Statements

- Take advantage of information in the Dynamic Statement Cache
  - We will cover the DSC in more detail momentarily
- Create a statement cache table
  - Run EXPLAIN STMTCACHE STMTID
    - Needs SYSADM authority
- Apply traditional SQL tuning best practices
  - If you are able to modify the SQL
- What if you cannot modify the SQL?
  - Add or modify indexes
  - Freshed statistics by executing RUNSTATS
  - Reorganize table spaces and/or indexes
  - Lobby your application vendors to change the SQL

An Additional Dynamic SQL Tuning Thought

Consider stored procedures for frequently executed dynamic SQL statements

- Stored procedures can reduce network traffic
- Native SQL stored procedures are relatively easy to code and are more efficient than earlier types of stored procedures

Caveats:

- Requires modification of the application
  - To call the stored procedure instead of issuing the dynamic SQL
- Few tools to manage DB2 stored procedures
- Requires mainframe skills
  - May not be helpful if all programmers use an IDE

Dynamic SQL Best Practices

- Threshold and parameter controlled alerting
  - Tag “problem” SQL and watch over time
  - Compare past/average versus current
- Retain historical details
  - Easy identification of poor performing SQL
  - Average CPU and elapsed times for each SQL
  - Group top x SQL statements for tuning
    - Concentrate on biggest consumers for biggest return on tuning investment
  - Identify trends

Introduction to Dynamic Statement Caching (DSC)

- DB2 uses the DSC to minimize (perhaps even eliminate) dynamic prepares
- There are three (*OK, four*) types of caching supported:
  - No caching
  - Local Dynamic Statement Caching
  - Global Dynamic Statement Caching
  - Full Caching
- The prepared SQL and statement text for dynamic SQL statements are cached in the DBM1address space
  - Local Statement Cache
  - Global Dynamic Statement Cache

Implementing Dynamic Statement Caching

- Controlled by several different parameters
  - BIND options
  - DSNZPARMs
  - Application constructs
- Will discuss the parameters on the upcoming slides in the appropriate place

No Caching

- This is the default
  - And the way DB2 worked before the DSC was introduced
- Prepared statements do not persist across COMMIT
  - The prepared mini-plan is discarded after a commit
    - Except for CURSOR with HOLD statements
  - The next time the statement is executed it will have to be prepared again

Local Dynamic Statement Caching

- Eliminates need for the application to issue multiple PREPAREs for the same statement
  - Implicit PREPAREs are done by DB2
- To enable Local Statement Caching
  - Set KEEPDYNAMIC(YES) Bind Parameter
  - The MAXKEEPD DSNZPARM controls maximum prepared statements
    - Does not affect statement text which is always kept
- Not really a big performance help
  - Useful because programmers do not need to keep track of when COMMITs are issued in order to perform another PREPARE
    - DB2 does the implicit PREPARE
  - Some reduction in message traffic in a distributed environment is possible

Global Dynamic Statement Caching

- Will reuse prepared statements across units of work
  - Within and across program executions
  - Must be the EXACT same SQL statement, though
  - Prepared statement cached in global dynamic statement cache
    - Skeleton Dynamic Statement (SKDS)
    - Short Prepare
- To enable Global Statement Caching
  - Set DSNZPARM CACHEDYN=YES
- Global Dynamic Statement Caching can offer significant performance improvement for applications with frequent reuse of dynamic SQL
  - No coding changes required

Prerequisites for Global Dynamic Caching

- Statement text must be EXACTLY the same
  - Can use parameter markers
  - Literals will not work (unless it is always the SAME literal)[1]
- Other things that must be the same or compatible
  - Bind rules
  - Special registers
  - Authorizations
- Packaged application vendors (e.g. ERP) have designed their applications to make use of DSC

[1] DB2 V10 changes this where literals can be treated like variables

Are These Statements The Same?

```
SELECT COL1, COL2
FROM   TEST_TABLE
WHERE  COL3 = ?
AND    COL4 = ?
```

```
SELECT COL1, COL2
FROM   TEST_TABLE
WHERE  COL3 = ?
AND    COL4 = ?
```

How About These?

```
SELECT COL1, COL2
FROM   TEST_TABLE
WHERE  COL3 = ? AND COL4 = ?
```

```
SELECT COL1,
       COL2
FROM   TEST_TABLE
WHERE  COL3 = ?
AND    COL4 = ?
```

DB2 10: Literals can be treated as "the same"

- As we just learned, in order to take advantage of the DSC, the dynamic SQL statement must be exactly the same
- If a literal changes, it is not the same. For example:

NOT THE SAME

```
SELECT NAME, ADDRESS
  FROM   CUST
WHERE  CUSTNO = 1234;
```

```
SELECT NAME, ADDRESS
  FROM   CUST
WHERE  CUSTNO = 5678
```

- As of DB2 V10, dynamic SQL with literals can be reused in the DSC
  - It is still generally better to use parameter markers for dynamic SQL than to use literals and rely on this update though

Full Caching *(Both Local and Global)*

- Combines the benefits of Local and Global Dynamic Statement Caching
  - Can completely avoid PREPARE operations
  - Prepared statement kept in local thread storage and not invalidated across commits
    - Prepare Avoidance
- Enabling global statement caching
  - CACHEDYN=YES
  - MAXKEEPD>0
  - KEEPDYNAMIC(YES)

Flushing the DSC

Invalidate the dynamic statement cache by:

1. Executing RUNSTATS
2. RUNSTATS UPDATE NONE REPORT NO
   - Causes any statement in the DSC which is dependent on the affected table space or index space to be removed from the cache.
   - This is done to allow users who manually update DB2 Catalog statistics to invalidate the related dynamic SQL in the cache
     - The next prepare will re-evaluate the access paths.

- The granularity is at the table space and index level
  - Not the table level

Reoptimization (REOPT)

Reoptimization settings can make dynamic SQL more static …and sometimes vice versa.

- You can gain additional optimization for (mostly dynamic) SQL using the REOPT parameter of the BIND command.
- REOPT specifies whether to have DB2 determine an access path at run time by using the values of host variables, parameter markers, and special registers.
- As of DB2 9, there are four options from which to choose when specifying REOPT.

REOPT Parameter Choices

NOREOPT(VARS) can be specified as a synonym of REOPT(NONE).

REOPT (NONE) –PREPARE determines the access path and no reoptimization is performed. The bound statement can be moved to the dynamic statement cache (DSC), if the cache is being used.

DB2 V8

REOPT (ONCE) – PREPARE determines an initial access path before the host variable values are available. When the statement is first executed and the host variable values are known, the statement is reoptimized one time. The hope is that the one-time reoptimization will provide a better access path than the initial PREPARE. The statement can be placed in the dynamic statement cache and reused multiple times.

REOPT(VARS) can be specified as a synonym of REOPT(ALWAYS).

REOPT (ALWAYS) – The SQL statement is re-optimized each time it is executed, always using the latest host variable values.

DB2 9

REOPT (AUTO) – Leave it up to DB2 (autonomic?). If changes in the filter factors for the statement predicates warrant, DB2 can re-prepare the statement. The newly prepared statement would be executed and would replace the prepared statement currently in the Global Dynamic Statement cache.

REOPT Applicability: Dynamic vs. Static SQL

REOPT Parameter	Dynamic SQL	Static SQL
NONE	YES	YES
ALWAYS	YES	YES
ONCE	YES	NO
AUTO	YES	NO

Consider binding static programs with REOPT(ALWAYS) when the values for your program's host variables or special registers are volatile and make a significant difference for access paths.

ONCE and AUTO are not valid for static SQL because they work with the dynamic statement cache, which does not apply to static SQL.

Dynamic SQL Best Practices

- Understand the difference between static & dynamic
  - Use appropriate parameters for each
- Implement Dynamic Statement Cache
  - Train programmers how to write code that benefits from DSC
- Build threshold and parameter-controlled alerts
  - Tag “problem” SQL and watch over time
  - Compare past/average versus current
- Retain historical details
  - Easy identification of poor performing SQL
  - Average CPU and elapsed times for each SQL
  - Group “Top *x*” SQL statements for tuning
    - Concentrate on biggest consumers for biggest return on tuning investment
  - Identify trends

And Now ...
Solving Your Dynamic SQL Woes Using InSoft QA+

- Benefits
- Facilities
- Cache Monitor
- Accumulation
- Neutralization
- Tagging
- Reports
- Searching

QA+ Benefits

- Monitor all activities in the DB2 Statement Cache consequently
- Accumulate and build a DB2 Performance Data Warehouse
- Report vital information on performance problems and heavy consumers
- Issue alerts on all critical dynamic SQL executed (local, remote, connected client)
- One service to monitor all dynamic SQL - local applications, client-server, web-based, QMF, ERP-applications and ...
- Produce a comprehensive picture of the dynamic SQL as basis for tuning
- Increase CPU savings and delay expensive upgrades
- No expensive DB2 Trace required (318 for statistics is recommended)
- Available for DB2 V8 and V9

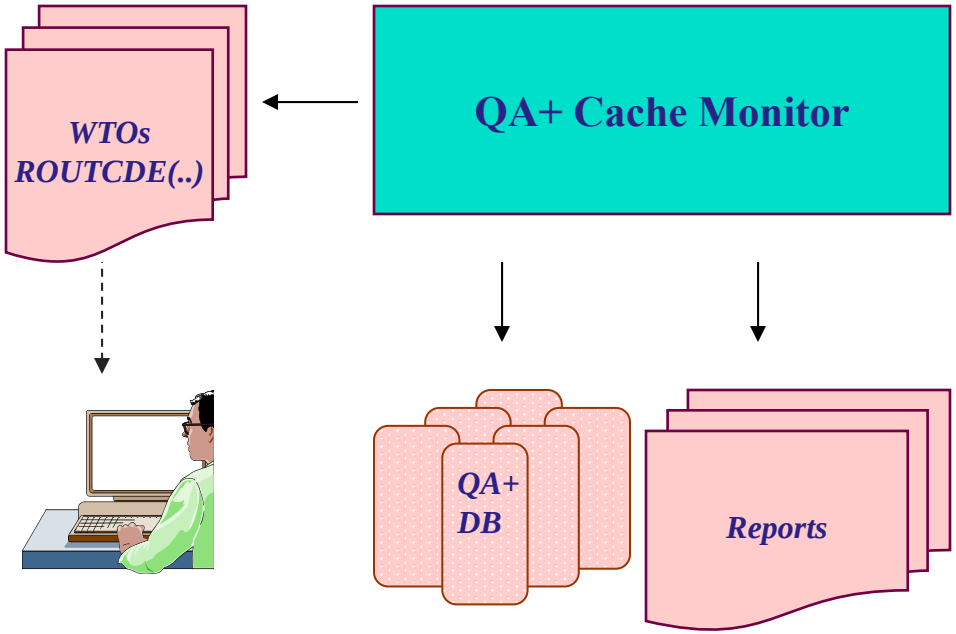
QA+ Facilities

- Threshold and parameter controlled Alert System - automatically registers all “problem” SQL!
- “snap shots” @ pre-defined periods for a given duration
- All dynamic SQL consolidated and benchmarked together with the DB2 access paths
- “same” SQL statements optionally neutralized to reduce volume of data, compare Access Paths and CPU usage
- Average CPU and elapse times for each SQL; easy recognition of poor performing SQL
- Detailed explain analysis
- View results per DB2 Group, sub system, user identifiers or an accumulated combination of all

QA+ Facilities (continued)

- Easy navigation dialogue - locate most frequent & most costly SQL. E.g. SQL with highest no. of rows returned, Getpages, heaviest CPU consumers, hottest objects, indexes not used ...
- Tune the top “n” hot SQL statements
- Tag “problem” SQL and follow/ compare progress thru time.
- Reports include - Period Analysis, Trends, CPU Consumers, Objects accessed ...
- System reports include Cache activity and recommended EDMCACHE size.
- Groups can be (re)defined to reveal CPU consumers per User-Id. (Primary or Secondary) and / or Tokens
- Interface to QA enables Quality Assurance for all monitored SQL and full explain functionality for tuning purposes
- Interface to IA+ (Index Advisor) enables “index tuning”

QA+ Cache Monitor



Job runs for ... minutes or unlimited
Do until time expired or quiesce pending

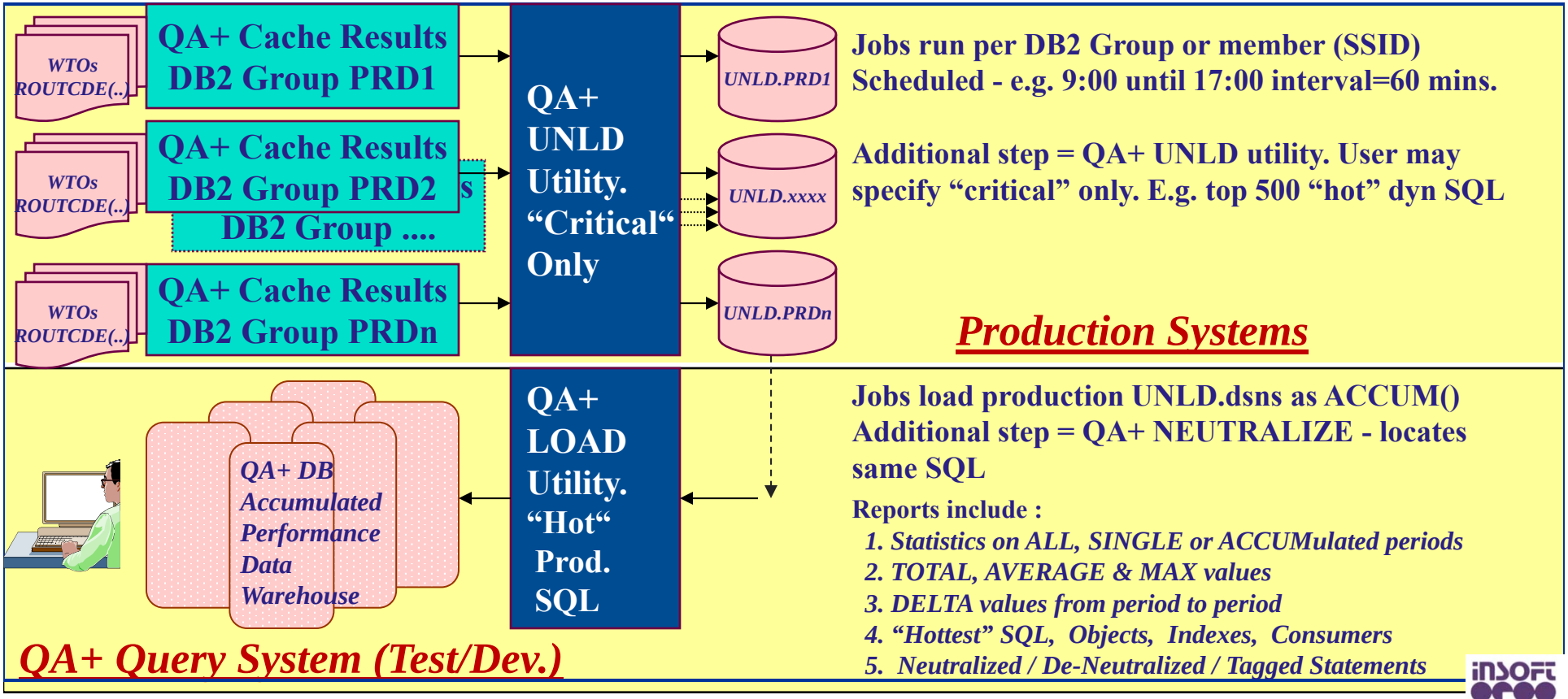
1. *Obtains Cache snap-shot*
2. *EXPLAIN all "new" STMTs*
3. *Check ALERTs pending*
4. *Save all required data in QA+ DB*
5. *Job waits ... minutes (1 - 60 mins.)*

Continue

Reports include :

1. *Statistics on ALL + SINGLE periods*
2. *TOTAL, AVERAGE & MAX values*
3. *DELTA values from period to period*
4. *"Hottest" SQL, Objects, Indexes, Consumers*
5. *Non-used Indexes*

QA+ Accumulation



QA+ Neutralization

- Automatically consolidates “like” SQL statements
  in multiple phases:
  1. Group DML accessing same base object names
  2. Eliminate ‘xxx’ and other literals
  3. Optionally eliminate object “creators” and “schemas”
  4. Process all “inter-changeables” - e.g.
     a) WHERE A=1 AND B=2 compared to WHERE B=2 AND A=1
     b) SELECT Block1 UNION SELECT Block2 or Block2 / Block1
     c) and many more ...
  5. Compute SIMILARITY FACTOR. 100% = identical, 95% etc.

- DB/IQ QA+ is the only z/OS DB2 product going beyond item 2)

QA+ Tagging

- User has found interesting dynamic SQL and tags it with “tag name”
- All “like” SQL statements are also tagged with “tag name” when found
- Applies to all sub systems participating in QA+ database
- DB/IQ QA+ is the only z/OS DB2 product tagging “like” dynamic SQL

QA+ Reports

- Reports can be generated at any time
- All reports are optional
- Most reports also put to a data set -> “host” and CSV format
- According to multiple criteria (e.g. AVG_Timings, Totals,...)
  - totals and averages (over all and single periods)
  - "hottest" objects (qualified and unqualified) over all periods
  - "hottest" and “unused” indexes over all periods
  - "hot" consumers / tokens over all periods
  - "hottest" SQL over all periods (opt. with SQL text and explain)
  - "hottest" SQL for single periods
  - "SAME Hot" statements over ALL periods
  - "hot" SQL. CPU >= 5 * higher than DB2 estimated
  - "Tagged” statements over ALL periods – with SQL text & explain

QA+ Report Sample Report 03

```
================================================================================================================
DB/IQ QA+ Cache Monitor IDBTBQP  V. 4.92 running in DB8G. "Hot" Objects (Qual.)   over ALL Periods
                      MINEXEC(10)      MAXRANK(50)      HOTCOLUMN(AVGCPU)
================================================================================================================
  Rank Object                                        STMTIDs      EXECs   CPU Time     Elapse Exec/CPUs Exec/ELAP
  ---- ------------------------------------------- -------- ---------- ---------- ---------- --------- ---------
     1 SYSIBM.SYSPACKAGE                                 25         16        2.1        9.6   0.13309   0.59859
     2 INSOFT.TDB1QAPLUSSTMTD                            12         24        3.0        7.4   0.12348   0.30715
     3 INSOFT.TDB1QAPLUSPERIODS                          12         24        3.0        7.4   0.12348   0.30715
     4 SYSIBM.SYSTRIGGERS                                14         30        0.3        1.3   0.01094   0.04200
     5 SYSIBM.SYSSYNONYMS                                 8         80        0.4        1.9   0.00557   0.02371
     6 SYSIBM.SYSSEQUENCES                                6         57        0.3        0.7   0.00517   0.01311
                                                                                                          ...
    16 SYSIBM.SYSCHECKS                                   2        764        0.3        1.6   0.00044   0.00206
    17 INSOFT.TDB1TUSE_MA                                34       6286        2.3       17.6   0.00037   0.00281
                                                                                                          ...
    28 SYSIBM.SYSTABLES                               25299      37395        5.0       27.7   0.00013   0.00074
                                                                                                          ...
    50 INSOFT.TDB1TUSE2                                   2          2        0.5        2.6       n/a       n/a
```

Cache Monitor Batch Output - "hottest" Objects found over all measured periods

AVG - calculated min. 10 EXECs

Ranking -9999 unlimited. 50=top 50.

QA+ Report Sample Report 08

```
================================================================================================================================
DB/IQ QA+ Cache Monitor IDBTBQP  V. 4.92 running in DB8G. "Hot" CONSUMERS over ALL Periods.
                      MINEXEC(10)      MAXRANK(50)      HOTCOLUMN(AVGCPU)
================================================================================================================================
```

Rank	CONSUMER	TYPE	EXECs	GetPage	RowExam	RowProc	Sorts	IXScans	TSScans	CPU Time	Elapse	Exec/CPUs	Exec/ELAP
1	INSOFT	CURSQLID	177590	18621K	243469K	604216	25528	106983	89345	4:03.1	16:02.4	0.00137	0.00542
2	INSO32	CURSQLID	1004	8160	18350	1559	393	306	1112	0.2	0.5	0.00019	0.00053
3	INSO42	CURSQLID	244	1120	973	650	110	20	332	0.0	0.2	0.00017	0.00066
4	INSO22	CURSQLID	323561	1507035	1784900	457949	12956	386948	53354	24.5	1:43.6	0.00008	0.00032
1	JAVA1	GROUPID	1272	29146	37859	11481	508	18734	1455	0.9	3.2	0.00068	0.00250
2	ALLUSERS	GROUPID	502399	20138K	245273K	1064374	38987	494257	144143	4:27.8	17:46.7	0.00053	0.00212
3	INSOFTDB	GROUPID	501127	20108K	245235K	1052893	38479	475523	142688	4:26.9	17:43.5	0.00053	0.00212
1	INSO42	PRIMAUTH	245	20712	19291	9809	111	18339	333	0.7	2.5	0.00268	0.01040
2	INSO22	PRIMAUTH	501127	20108K	245235K	1052893	38479	475523	142688	4:26.9	17:43.5	0.00053	0.00212
3	INSO32	PRIMAUTH	1027	8434	18568	1672	397	395	1122	0.2	0.6	0.00020	0.00061

Cache Monitor Batch Output - "hottest" Consumers found over all measured periods

QA+ Report Sample Report 09

```
================================================================================================================================
DB/IQ QA+ Cache Monitor IDBTBQP  V. 4.92 running in DB8G. "Hot" Statements over ALL Periods. Accum-id: INSOPROD
                        MINEXEC(10)      MAXRANK(50)      HOTCOLUMN(AVGCPU)
================================================================================================================================
```

Report via Accumulation

Rank	Ident1	Ident2	STMTID	PROGNAME	TYP	EXECs	GetPage	RowExam	RowProc	Sorts	IXScans	TSScans	CPU-T.	Elapse	Exec/CPUs	Exec/ELA
1	NRW	Köln	143411	IDBUPS3	SEL	12762	16201K	241580K	12762	0	0	12762	2:44.3	6:40.9	0.01288	0.0314
2	BERLIN	Kreuzbrg	247	IDBUBI30	SEL	397	353330	873003	216	397	0	588	2.0	5.1	0.00504	0.0129
3	BERLIN	Kreuzbrg	242	IDBUBI35	SEL	120	108128	31800	160	360	120	360	0.5	1.3	0.00428	0.0110
4	BERLIN	Kreuzbrg	202	IDBUB30	SEL	28	25184	18592	24	56	0	76	0.1	0.3	0.00426	0.0110
5	NRW	Köln	143365	IDBUB30	SEL	30	269	60	30	30	60	60	0.0	0.4	0.00145	0.0141
6	NRW	Köln	143382	IDBUBI35	SEL	1256	13456	51764	25254	1256	29022	1256	1.4	10.2	0.00110	0.0080
7	BERLIN	Kreuzbrg	220	IDBUBI35	SEL	350	2612	1050	7000	350	1400	350	0.3	0.5	0.00079	0.0014
8	BERLIN	Kreuzbrg	203	IDBUB30	SEL	45	176	90	45	45	90	90	0.0	0.1	0.00062	0.0013
9	NRW	Köln	143364	IDBUB30	SEL	17	528	289	15	17	0	34	0.0	0.1	0.00057	0.0070
10	BERLIN	Kreuzbrg	226	IDBUBI35	SEL	579	8531	579	579	579	1158	579	0.3	0.8	0.00045	0.0013
11	NRW	Köln	143389	IDBUBI35	SEL	24	1128	8924	24	0	0	392	0.0	0.1	0.00040	0.0036
12	NRW	Köln	143386	IDBUBI35	SEL	1321	13314	27413	26092	1321	2642	1321	0.5	3.6	0.00039	0.0027
13	NRW	Köln	143388	IDBUBI35	SEL	185	2474	186	186	185	371	185	0.1	0.8	0.00039	0.0043
14	NRW	Köln	143384	IDBUBI35	SEL	1277	12821	26647	25370	1277	2554	1277	0.5	3.7	0.00037	0.0029
15	NRW	Köln	143363	IDBUB30	SEL	1608	953	0	120	0	0	1608	0.6	4.3	0.00035	0.0026
16	BERLIN	Kreuzbrg	162	IDBTBQP1	SEL	160	969	787	304	161	0	322	0.1	0.1	0.00033	0.0008
17	BERLIN	Kreuzbrg	224	IDBUBI35	SEL	426	2599	426	7684	426	852	426	0.1	0.3	0.00033	0.0008
18	NRW	Köln	143420	IDBUBI20	SEL	55	342	263	263	55	0	110	0.0	0.0	0.00031	0.0003

Identify system reference

Cache Monitor Batch Output - "Hottest" SQL over ALL periods

QA+ Report Sample Report 14

```
================================================================================================================
DB/IQ QA+ Cache Monitor IDBTBQP  V. 4.92 running in DB8G. Tagged Statements over ALL Periods. Accum-id: INSOPROD
================================================================================================================
TAG: MYTAG_002
STMTID: 242          Progname: IDBUBI35  Cached: 2009-09-11-13.33.50.152657
Ident1: BERLIN       Ident2: Kreuzbrg
Execs: 120           GetPage: 108128     RowExam: 31800       RowProc: 160
Sorts: 360           IXScans: 120        TSScans: 360
CPU Time: 0.5        Elapse: 1.3         Exec/CPUs: 0.00428   Exec/ELAP: 0.01101

       SELECT F.RELNAME, F.COLNAME, F.COLSEQ, R.DELETERULE, R.REFTBCREATOR, ...
              ... more
---    statement EXPLAINed as follows                                      ---
              ... more
--------------------------------------------------------------------------------------------
TAG: MYTAG_002
  STMTID: 143403       Progname: IDBUBI35  Cached: 2009-10-09-15.11.26.294770
  Ident1: NRW          Ident2: Köln
  Execs: 164           GetPage: 982        RowExam: 1678        RowProc: 254
  Sorts: 164           IXScans: 637        TSScans: 164
  CPU Time: 0.0        Elapse: 0.2         Exec/CPUs: 0.00018   Exec/ELAP: 0.00130

         SELECT F.RELNAME, F.COLNAME, F.COLSEQ, R.DELETERULE, R.REFTBCREATOR, ...
              ... more
```

tagged as **MYTAG_002** – STMTID **242** was cached in DB9G (identified by BERLIN/Kreuzbrg) 2009-09-11-13:33

tagged as **MYTAG_002** – STMTID 143403 was cached in DB8G (identified by NRW/Köln) 2009-10-09-15:11

QA+ Search Engine

```
InSoft Software --------- QA Cache Search thru SnapShots --INSO22--2009/1 18:35
===>
Snap shots between: 2009-10-13-00.00 and 2009-12-31-23.59
ACCUM-ID          Ident 1               Ident 2
PRIMAUTH *        CURSQLID   *          User Group DBAS
PROGRAM  *        EDM STMTID            User Tags
TOKEN
                   Following values num or numK or numM
SQL EXEC        >=      1000   GetPage Requests >=          0
SyncReads       >=         0   BuffWrites       >=          0
RowsExamined    >=         0   RowsProcessed    >=          0
Sorts           >=         0   PGRP_Created     >=          0
IX_Scans        >=         0   TS_Scans         >=        100
                   Following values in seconds
Tot_Elapse      >=    5.00000  Tot_CPU          >=    0.50000
Tot_SYNCIO      >=    0.00000  Tot_Lock         >=    0.00000
Tot_SYNCEX      >=    0.00000  Tot_Global       >=    0.00000
Tot_OthReads    >=    0.00000  Tot_OtherWrit    >=    0.00000
Avg_Elapse      >=    0.00000  Avg_CPU          >=    0.00000
Avg_SYNCIO      >=    0.00000  Avg_LOCK         >=    0.00000
Avg_SYNCEX      >=    0.00000  Avg_Global       >=    0.00000
Avg_OthReads    >=    0.00000  Avg_OtherWrit    >=    0.00000
String within SQL text %DSN_%
```

Search by Accumulation or when captured

Search USER Group or TAGGED Statements

Search for a specific statement ID

Search for statements exceeding specified values

Cache Monitor Dialogue - Search Input

Questions ?

Please visit us @ Booth 7

www.insoft-software.com

INSOFT SOFTWARE GMBH

IDUG
The Worldwide DB2 User Community

EMEA Conference 2010

Craig S. Mullins
Mullins Consulting, Inc.
craig@craigsmullins.com

Colin Oakhill
Insoft Software
co@insoft-software.de

Session V12

Title Quality Assurance for Dynamic SQL