IDUG
The Worldwide DB2 User Community
Regional Events 2010

Can I control a dynamic world?
Dynamic SQL Management for DB2 z/OS

Ulf Heinrich
SEGUS Inc

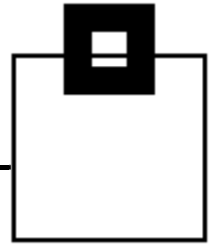
Platform: z/OS

AGENDA

- Dynamic SQL at a glance:
  - Characteristics
  - DB2 setup and support
  - DB2 commands and features

- See the possibilities:
  - How to exploit dynamic SQL successfully
  - How to manage dynamic SQL reliably

Dynamic SQL at a glance



Characteristics:

- It's flexible
  → SQL statements can be built and executed on the fly
- It's dynamic
  → access paths are determined ad hoc
- It's state of the art
  → widely supported in today's programming languages
- It's difficult to control
  → Statement and access path is only available at runtime
- It's expensive
  → Optimization and tuning is difficult

Dynamic SQL at a glance

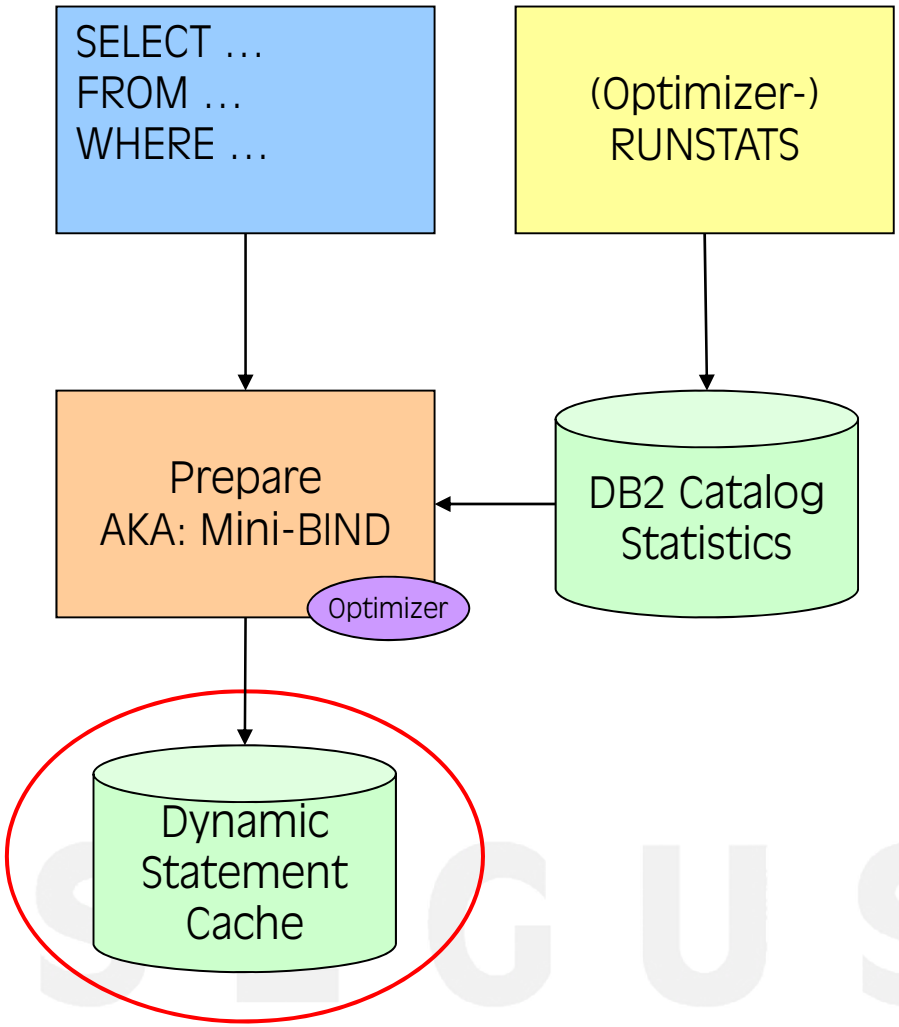
Characteristics:

IBM Says:

"In general, an application using dynamic SQL has a higher startup (or initial) cost per SQL statement due to the need to compile the SQL statements before using them. Once compiled, the execution time for dynamic SQL compared to static SQL should be equivalent and, in some cases, faster due to better access plans being chosen by the optimizer. Each time a dynamic statement is executed, the initial compilation cost becomes less of a factor. If multiple users are running the same dynamic application with the same statements, only the first application to issue the statement realizes the cost of statement compilation."

Dynamic SQL at a glance

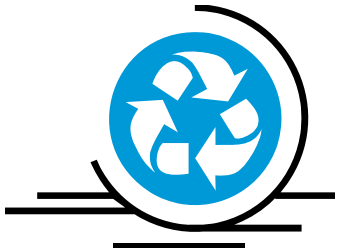
Characteristics:



Access Paths for dynamic SQL are determined on the fly and stored in the DSC.

RUNSTATS, ALTERS, DB2 RESTART invalidates and flushes the DSC for an object.

Dynamic SQL at a glance



Characteristics:

DB2 can recycle a cached statement if

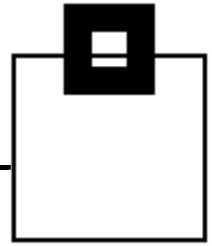
- The statement is 100% identical
  - Thus Literals usually compromise caching

  → Use parameter markers!
- Bind rules, Special registers, Authorizations are compatible/same

If this doesn't fit to your application, you may not benefit from the dynamic statement cache at all.

ERP/CRM vendors like SAP use the DSC extensively and fully exploit it

Dynamic SQL at a glance



Characteristics:

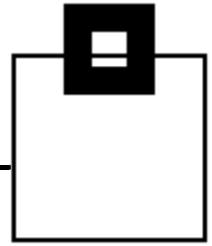
The DSC is where dynamic SQL statements, and *only* Dynamic SQL statements, reside once they have been PREPAREd if certain ZPARM and/or BIND options are in use.

The next time the exact same statement is to be PREPAREd the cache is searched and, if all is valid and certain ZPARM and/or BIND options are in use, then the PREPARE is avoided

→ Thus saving lots of CPU time.

Ideally an SQL statement should stay in the cache forever, but the real world shows that two days of residency or latency is typical.

Dynamic SQL at a glance



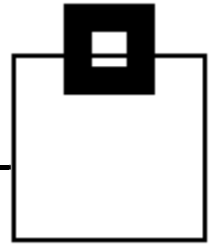
Characteristics:

The DSC is in fact two areas of memory:

- The Local statement cache in the DBM1 space with a FIFO queue
  → gives you a small benefit (PREPARE once across COMMITs)

- The Global statement cache in the EDM Pool above "The Bar" with a sophisticated LRU queue
  → gives you a big benefit (allows to reuse a statement PREPAREd by another thread)

Dynamic SQL at a glance



DB2 Setup and Support:

CACHEDYN->		NO	YES
KEEPDYNAMIC	NO	• No skeletons cached in EDMP • Only full prepares • No prepared statements kept across commits (note1) • No statement strings kept across commits NONE	• Skeletons cached in EDMP • 1st prepare full; others short (note 2) • No prepared statements kept across commits (note 1) • No statement strings kept across commits Global
	YES	• No skeletons cached in EDMP • Only full prepares • No prepared statements kept across commits (note 1) • Stmt strings kept across commits – implicit prepares LOCAL	• Skeletons cached in EDMP • 1st prepare full; others short (note 2) • Prepared stmts across commits – avoids prepares (note 3) • Stmt strings kept across commits – implicit prepares FULL AKA: Prepare Avoidance

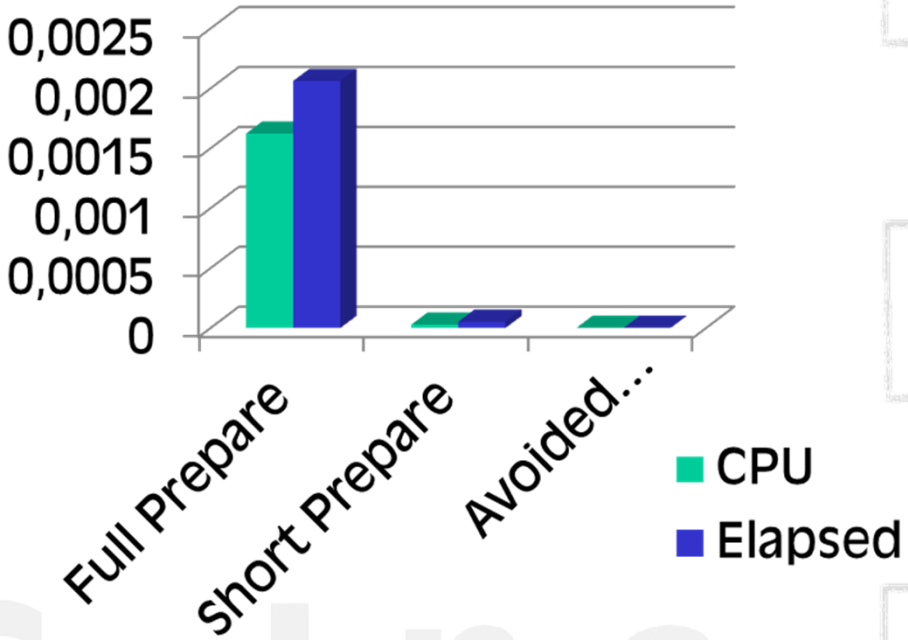
Note 1: unless a cursor WITH HOLD is open, Note 2: unless invalidated or flushed out due to LRU, Note 3: assuming MAXKEEPD > 0

Dynamic SQL at a glance

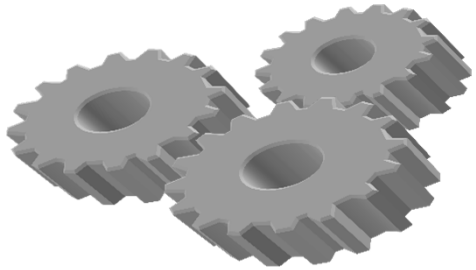
DB2 Setup and Support:

The right setup saves a lot of money

- Exploit the full flavor of caching
  - MAXKEEPD>0
  - CACHEDYN=YES
  - KEEPDYNAMIC(YES)



Dynamic SQL at a glance



DB2 Setup and Support:

1. Turn on the LOCAL and GLOBAL cache!
   Bind with KEEPDYNAMIC(YES)
   ZPARM CACHEDYN = YES
   ZPARM MAXKEEPD >= 5000 (For large SAP start at 8000)

2. Enforce the use of QUERYNO in dynamic SQL
   It is the *only* way to be able to use HINTs
   It is the *best* way to enable trend analysis

3. Try and enforce use of parameter markers where appropriate…
   … as always with DB2 „It depends…“

5. Set ZPARM EDMSTMTC to a “good” caching size

4. Switch OFF the RLF for dynamic SQL unless you really need it!

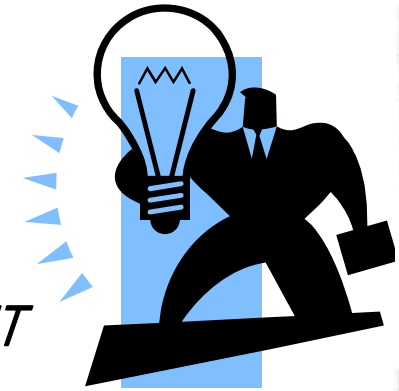
Dynamic SQL at a glance

So far, so good …

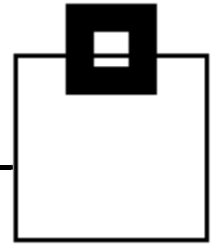
Knowing how to handle it, opens up great opportunities for

- Packaged applications like SAP
- Less-mainframe-skilled developers
- Interactive multi-platform solutions
- The mainframe competing with the distributed environment
- Cost efficient and well performing applications

→ The key is the Dynamic Statement Cache …



Dynamic SQL at a glance



DB2 Commands and Features:

Apart from the obvious DB2 group restart or IPL any type of RUNSTATS is poison to the DSC!

- RUNSTATS TABLESPACE will delete any statements that use any object within the TABLESPACE regardless of any TABLE (xxx.yyy) syntax.
- RUNSTATS INDEX will delete statements that use that index.

REOPT causes re-optimization of an access path

- ALWAYS → creates a fresh AP at each execution (no caching!)
- ONCE → creates AP at first execution and stores it in the DSC
- AUTO → DB2 decides whether a new AP is beneficial based on the parameter values
  - (Should) combine the advantages of REOPT(ALWAYS) and REOPT(ONCE)

Dynamic SQL at a glance

DB2 Commands and Features:

Different to static SQL, the dynamic world allows to “re-explain” an access path using for example SPUFI, DSNTEP2:

- EXPLAIN STMTCACHE ALL
- EXPLAIN STMTCACHE STMTID
- EXPLAIN STMTCACHE STMTTOKEN

→ EXPLAIN STMTCACHE does not go through the EXPLAIN process, but tells you exactly about the current access!

See the possibilities

How to exploit dynamic SQL successfully:

The DSC gives you insight:

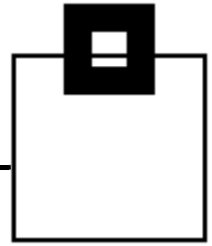
Memory resident storage of prepared dynamic SQL statements

- SQL text
- Statement ID
- Date/time, current status
- Resource consumption

→ Start IFCID traces 316, 317 (318) to access the data

…once the data is gathered, decide on what you want to analyze…

See the possibilities



How to exploit dynamic SQL successfully:

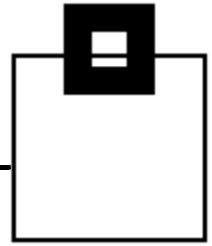
...if you want/need to see everything:

EXPLAIN STMTCACHE ALL

- Externalizes all statements in the dynamic statement cache
  - <current sqlid>. DSN_STATEMENT_CACHE _TABLE
  - one row for each cached statement
- Uses LOBs (STMT_TEXT is a 2M CLOB so be careful with that)
- Needs to run against all members in a data sharing environment

→ The data externalized is mostly identical to IFCID 316, 317

See the possibilities



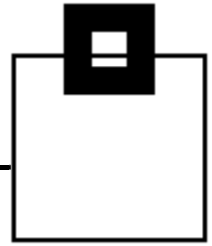
How to exploit dynamic SQL successfully:

...if you want/need to see details on a single statement:

EXPLAIN STMTCACHE STMTID

- Externalizes more details on the specified statement ID
  - The ID is an integer that uniquely identifies a statement in the dynamic statement cache.
  - E.g. from DSN_STATEMENT_CACHE_TABLE STMT_ID column
  - E.g. through IFI monitor facilities from IFCID 316 or 124
  - E.g. from diagnostic IFCID trace records such as 172, 196, and 337.
- Inserts data into PLAN, DSN_DYNAMIC_STATEMNT, DSN_STATEMENT, and DSN_FUNCTION tables

See the possibilities



How to exploit dynamic SQL successfully:

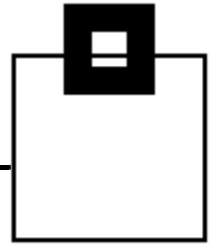
...if you want/need to see details on a group of statements:

EXPLAIN STMTCACHE STMTTOKEN

- Externalizes more details on all cached statements associated with the specified token
- STMTTOKEN has to be set by the application program
  - RRSAF SET_ID
  - sqleseti API

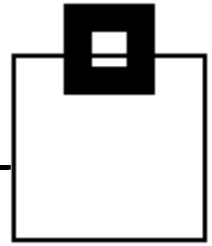
- Inserts data into PLAN, DSN_DYNAMIC_STATEMNT, DSN_STATEMENT, and DSN_FUNCTION tables

See the possibilities



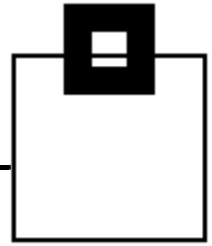
```
Analyze for DB2 z/OS --- Dynamic Statement Cache (1/8) -- Statement 1 from 117
Command ===> _____________________________________________ Scroll ===> CSR
                                                          DB2: Q91A

Primary cmd: END, F(ilter), Z(oom), L(ocate) getpages
Line    cmd: Z(oom), A(nalyze), E(dit statement), S(tatement text), T(able),
             X(EXecute)

    StmtID     Program   Lineno     UserID    Qualifier Executes   Getpages   S
    --------- -------- --------- -------- -------- --------- --------- -
_         2162 IQADBACP      1086 NEUMANN  NEUMANN          14       245 V
_         2164 IQADBACP      1094 NEUMANN  NEUMANN          36       222 V
_         2152 IQADBACP      1086 NEUMANN  NEUMANN           3        61 V
_         2154 IQADBACP      1086 NEUMANN  NEUMANN           7        48 V
_         2247 IQADBACP      1042 NEUMANN  NEUMANN           1        48 V
_         2250 IQADBACP      1042 NEUMANN  NEUMANN           1        48 V
_         2192 IQADBACP      1082 NEUMANN  NEUMANN          10        47 V
_         2208 IQADBACP      1042 NEUMANN  NEUMANN           1        47 V
_         2138 IQADBACP      1082 NEUMANN  NEUMANN          12        39 V
_         2150 IQADBACP      1086 NEUMANN  NEUMANN           3        24 V
_         2155 IQADBACP      1086 NEUMANN  NEUMANN           7        24 V
_         2253 IQADBACP      1022 NEUMANN  NEUMANN           1        23 V
_         2255 IQADBACP      1090 NEUMANN  NEUMANN           3        21 V
_         2256 IQADBACP      1094 NEUMANN  NEUMANN           4        20 V
_         2142 IQADBACP      1086 NEUMANN  NEUMANN           8        18 V
_         2112 IQADBACP      1082 NEUMANN  NEUMANN           0        16 V
```

See the possibilities



```
Analyze for DB2 z/OS --- Dynamic Statement Cache (4/8) -- Statement 1 from 117
Command ===> _____________________________________________ Scroll ===> CSR
                                                          DB2: Q91A

Primary cmd: END, F(ilter), Z(oom), L(ocate) getpages
Line    cmd: Z(oom), A(nalyze), E(dit statement), S(tatement text), T(able),
             X(EXecute)
               Synchr.    Synchr.   Rows       Rows       Index     Tablespc.
    StmtID     Buffer Rd  Buffer Wr examined   processed  Scans     Scans
    --------- ---------- -------- ---------- ---------- ---------- ----------
_         2162          0        0         74         37         52         15
_         2164          0        0          0        185          0         74
_         2152          4        0         38         19         30          4
_         2154          0        0         16          8         16          0
_         2247          0        0        101          2          2          1
_         2250          0        0        101          2          2          1
_         2192          0        0        844          2          4         11
_         2208          4        0        100         39          2          1
_         2138          0        0         13         13         13          0
_         2150          0        0          8          4          8          0
_         2155          0        0          8          0          8          0
_         2253          0        0          3          1          0          0
_         2255          0        0          0          7          3          3
_         2256          0        0          0          2          0          8
_         2142          0        0          0          9          0          9
_         2112          0        0          1          0          0          1
```

See the possibilities



```
Analyze for DB2 z/OS --- Dynamic Statement Cache (6/8) -- Statement 1 from 117
Command ===> _____________________________________________ Scroll ===> CSR
                                                          DB2: Q91A

Primary cmd: END, F(ilter), Z(oom), L(ocate) getpages
Line    cmd: Z(oom), A(nalyze), E(dit statement), S(tatement text), T(able),
             X(EXecute)

    StmtID     Total CPU       Average CPU     Total Elapse    Average Elapse
    ---------- HHHH:MM:SS.ttt  HHHH:MM:SS.ttt  HHHH:MM:SS.ttt  HHHH:MM:SS.ttt
_         2162          0.040           0.003           0.373           0.027
_         2164          0.047           0.001           0.128           0.004
_         2152          0.014           0.005           0.104           0.035
_         2154          0.007           0.001           0.007           0.001
_         2247          0.006           0.006           0.006           0.006
_         2250          0.006           0.006           0.006           0.006
_         2192          0.005           0.001           0.005           0.001
_         2208          0.013           0.013           0.089           0.089
_         2138          0.004               -           0.004               -
_         2150          0.002           0.001           0.002           0.001
_         2155          0.002               -           0.002               -
_         2253              -               -               -               -
_         2255          0.004           0.001           0.004           0.001
_         2256          0.004           0.001           0.004           0.001
_         2142          0.002               -           0.002               -
_         2112              -               -               -               -
```

See the possibilities

How to manage dynamic SQL reliably:

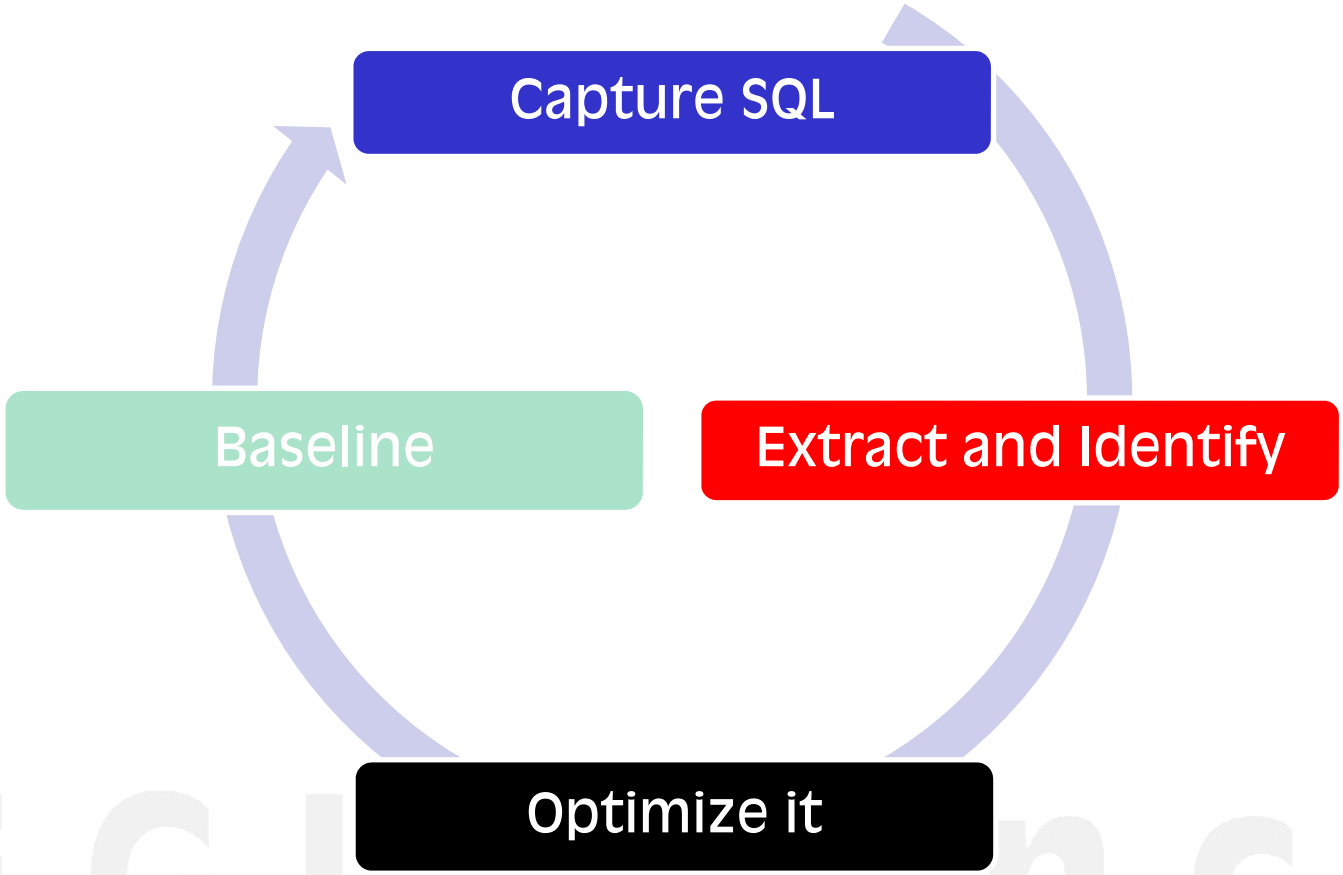
Dynamic SQL can be managed …. but it takes some work:

- Analyze dynamic SQL now and over time
- Tune dynamic SQL now and over time

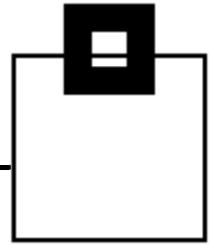
- Keep it simple and do it the familiar way – like your static
  1. Find the candidates
  2. Analyze and understand it
  3. Optimize it
  4. Create a baseline to understand trends and changes

See the possibilities

How to manage dynamic SQL reliably:



See the possibilities



How to manage dynamic SQL reliably:

Step 1 – Find the candidates:

Capture SQL

DSC Capture:
- ❑ Online extraction
- ❑ Batch extraction

Threshold based SQL extract and SQL filtering

Sort statements by
CPU utilization, frequency, timestamps

See the possibilities

How to manage dynamic SQL reliably:

Step 1 – Find the candidates:

Keep in mind, there are no DBRMs, packages, (source code)

→ Use your DB2 monitor

→ Grab the data from the DSC

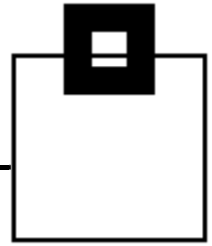
Find a starting point

→ E.g. get the top ten bad ones

→ Sort statements by

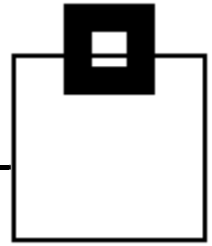
- CPU utilization
- Execution frequency
- Timestamp

See the possibilities



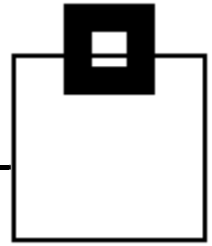
```
Analyze for DB2 z/OS ----- Limit DSC Snapshot --------------------------------
Command ===> ____________________________________________________ DB2: Q91A

Primary cmd: END

MEMBER          : ________  Blank(Connected DB2) / *(All members) / member name

NO LIMITATION   : X
HIGHEST VALUES  : _
EXCEED THRESHOLD: _      THRESHOLD: _________

For limitation to highest values or exceeding of specified threshold
EXECUTIONS      : _      ROWS PROCESSED    : _      SORTS                 : _
BUFFER READS    : _      ROWS EXAMINED     : _      PARALLEL GROUPS       : _
BUFFER WRITES   : _      INDEX SCANS       : _      RID EXCEED DB2 LIMITS : _
GETPAGES        : _      TABLE SPACE SCANS : _      RID EXCEED STORAGE    : _

For limitation to highest values only
ELAPSE TIME     : _      CPU TIME          : _

WAIT TIME FOR ...
SYNCRONOUS I/O  : _      SYNCR. EXECUTION  : _      READS  OTHER THREADS  : _
LOCK AND LATCH  : _      GLOBAL LOCKS      : _      WRITES OTHER THREADS  : _
```

See the possibilities



```
Analyze for DB2 z/OS ----- Filter Dynamic Statement Cache --------------------
Command ===> _____________________________________________________ DB2: Q91A

Primary cmd: END

FIRST TABLE : _____________________________________________________________
CREATOR       _____________________________________________________________

FIRST TABLE : _____________________________________________________________
NAME          _____________________________________________________________

QUALIFIER   : _____________________________________________________________
              _____________________________________________________________

PRIMARY     : _____________________________________________________________
AUTHID        _____________________________________________________________

SELECT  X     CURRENT USERS   between ________ and ________ (Integer)
INSERT  X     STMT COUNT      between ________ and ________ (Integer)
UPDATE  X     AVG CPU TIME    between ________ and ________ (MM:SS.TTT)
DELETE  X     AVG ELAPSE TIME between ________ and ________ (MM:SS.TTT)
              AVG GETPAGES    between ________ and ________ (Integer)
Total stmts         104
OUTPUT LIMIT:     10000   0 - 25000 Max number of statements to be displayed
```

See the possibilities



How to manage dynamic SQL reliably:

Step 1 – Find the candidates – you may need to aggregate!

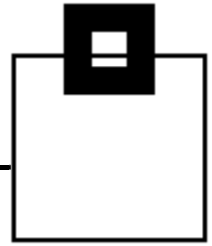
Level 1: Ignore values, spacing, cursor names, select clauses

….

SQL-Text	Count	CPU-Time
SELECT … WHERE COL = ‘ABC‘	1	1s
SELECT … WHERE COL = ‘BCD‘	1	1s
SELECT … WHERE COL = ‘CDE‘	1	1s
SELECT … WHERE COL = ‘DEF‘	1	1s
SELECT … WHERE COL = ‘EFG‘	1	1s
…		

SQL-Text	Count	CPU-Time
SELECT … WHERE COL = ‘ABC‘	10.000	10.000s
…		

See the possibilities



How to manage dynamic SQL reliably:

Step 1 – Find the candidates – you may need to aggregate!

… Level 2: Level1 + operators in predicates

Level 3: Level2 + right hand side

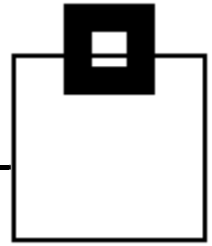
Level 4: Aggregate on object level

Optionally (for all levels): table creator

```
SELECT COLX, COL2 FROM CRE1.TAB1 WHERE COL5 = 'ABC'
SELECT COL1, COL2 FROM CRE1.TAB1 WHERE COL5 = '123'      => Level 1
SELECT COL1, COL2 FROM CRE1.TAB1 WHERE COL5 > '123'      => Level 2
SELECT COL1, COL2 FROM CRE1.TAB1 WHERE COL5 > :HV1       => Level 3
SELECT COL1, COL2 FROM CRE1.TAB1 WHERE COL7 > :HV1       => Level 4

SELECT COL1, COL2 FROM CRE2.TAB1 WHERE COL5 > :HV1       => Level 3
                                                         + tbcreator
```

```
SELECT COL1, COL2 FROM CRE1.TAB1 WHERE COL = 'ABC'
```

See the possibilities



How to manage dynamic SQL reliably:

Step 2 – Analyze and understand it:

Extract and Identify

- Sort statements by CPU utilization, frequency, timestamps
- Identify bad statements with dynamic explain

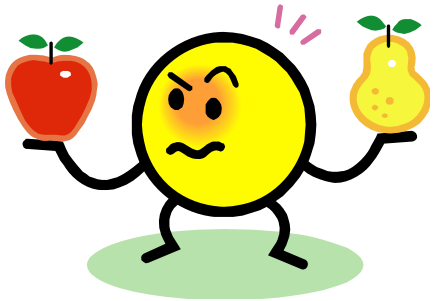
See the possibilities

How to manage dynamic SQL reliably:

Step 2 – Analyze and understand it:

- Run EXPLAIN STMTCACHE STMTID
  - SQLCODE -20248 if statement no longer exists in the cache
- Match the column STMT_TOKEN in your PLAN_TABLE to the statement token of the statement
- COLLID Column contains “DSNDYNAMICSQLCACHE”
- Consider **all** the relevant factors
  - Schema
  - Object maintenance status
  - Statistics
  - SQL

See the possibilities



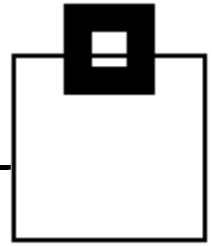
How to manage dynamic SQL reliably:

Step 2 – Analyze and understand it:

Match all information:

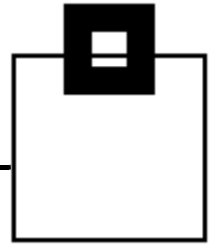
PLAN_TABLE	DSN_STATEMENT_CACHE_TABLE	
QUERYNO	STMT_ID	- Statement Identifier
STMT_TOKEN	STMT_TOKEN	- Identification Token
COLLID	COLLID	- "DSNDYNAMICSQLCACHE"
BIND_TIME	CACHED_TS	- Cache TS of the stmt

See the possibilities



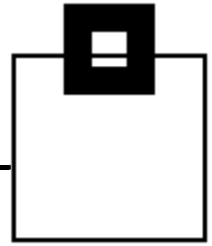
```
Analyze for DB2 z/OS ----- Explain Data (1/5) ----------------- Entry 1 from 1
Command ===> _____________________________________________ Scroll ===> CSR
                                                            DB2: Q91A
Primary cmd: END, T(Explain Text), V(iolations), R(unstats), P(redicates),
             S(tatement Text), PR(int Reports), Z(oom), SAVExxx, SHOWxxx
Line    cmd: Z(oom), I(ndexes of table), S(hort catalog), T(able), X(IndeX)

DSN = NEUMANN.ADB2.IN                                  Member = DELME
Stmt =         1
Milliseconds:        433  Service Units:       1696  Cost Category: B

  QBNO QBTYPE CREATOR  TABLE NAME         ACCS MTCH IX  METH PRNT TABL PRE  MXO
  PLNO TABNO  XCREATOR INDEX NAME         TYPE COLS ON   OD  QBLK TYPE FTCH PSQ
  ---- ------ -------- ------------------ ---- ---- --- ---- ---- ---- ---- ---
_    1 SELECT SYSIBM   SYSTABLES           R      0  N     0    0   T    S    0
     1 1
  ---- ------ -------- ------------------ ---- ---- --- ---- ---- ---- ---- ---
```

See the possibilities



```
Analyze for DB2 z/OS ----- Index Overview -------------------- Index 1 from 3
Command ===> _____________________________________________ Scroll ===> CSR
                                                            DB2: Q91A
Primary cmd: END, CAN(cel), SE(tup), Z(oom), L(ocate) creator
Line    cmd: C(olumns), D(atabase), K(PacKages), P(artitions), T(able), Z(oom)

    IX Creator  IX Name              Created by  Created timestamp
    TB Creator  TB Name              Database    Statstime
                                     Indexspace
    ------------------------------------------------------------------
_   SYSIBM      DSNDTX01             SYSIBM      0001-01-01-00.00.00.000000
    SYSIBM      SYSTABLES            DSNDB06     0001-01-01-00.00.00.000000
                                     DSNDTX01
    ------------------------------------------------------------------
_   SYSIBM      DSNDTX02             SYSIBM      0001-01-01-00.00.00.000000
    SYSIBM      SYSTABLES            DSNDB06     0001-01-01-00.00.00.000000
                                     DSNDTX02
    ------------------------------------------------------------------
_   SYSIBM      DSNDTX03             SYSIBM      2003-09-21-23.27.05.275288
    SYSIBM      SYSTABLES            DSNDB06     0001-01-01-00.00.00.000000
                                     DSNDTX03
    ------------------------------------------------------------------
```

See the possibilities



How to manage dynamic SQL reliably:

Step 3 – Optimize it:

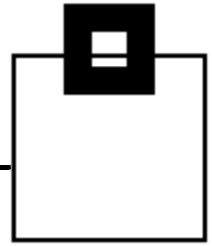
Optimize it

Apply rules based system to identify SQL problems

- ❑ Online
- ❑ Batch for mass analysis

Dynamically Explain SQL Statement

See the possibilities

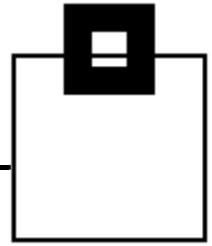


How to manage dynamic SQL reliably:

Step 3 – Optimize it:

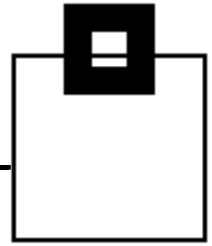
- Apply the traditional SQL tuning practices
  - Add or adjust indexes
  - Run REORG to improve index or tablespace processing
  - Run RUNSTATS to update catalog statistics
  - Improve the query (or talk to your application vendor)

See the possibilities



```
Analyze for DB2 z/OS ----- Violations -------------- LINE 00000001 COL 001 080
Command ===> _____________________________________________ Scroll ===> CSR
                                                          DB2: Q91A
Primary cmd: END, E(xplain Data), T(Explain Text), R(unstats), P(redicates),
             S(tatement Text), PR(int Reports), SAVExxx, SHOWxxx

DSN = NEUMANN.ADB2.IN                                  MEMBER = DELME
STMT =         1

------------------------------------------------------------------------------
                ----- RULE-NO.:  9072   (WARNING) -----
Predicate is stage 2 (neither stage 1 nor indexable)). QBLOCKNO: 1, Access:
STAGE2, Predicate: 5 BETWEEN SYSIBM.SYSTABLES.DBID AND SYSIBM.SYSTABLES.OBID
Try to rewrite the predicate as stage 1 or indexable or try to add another (
stage 1 or indexable) predicate for this column(s) to the WHERE or ON clause.
                ----- RULE-NO.:  9201   (WARNING) -----
A predicate like: '(EXPR) BETWEEN COL1 AND COL2' should be rewritten like:
'(EXPR) >= COL1 AND (EXPR) <= COL2'.
Then the predicates are INDEXABLE.
                ----- RULE-NO.:  9065   (WARNING) -----
SELECT * can lead to unnecessary data transfer. QBLOCKNO(s) affected: 1.
Select only columns which are really used by your application.
                ----- RULE-NO.:  9070   (SEVERE-ERROR) -----
Runstats check found critical rule violations.
Please look into the runstats report.
                ----- RULE-NO.:  9099   (WARNING) -----
```

See the possibilities



How to manage dynamic SQL reliably:

Step 4 – Create a Baseline

Baseline

- Store plan table entry for dynamic SQL statement
- Apply the way of analysis of static SQL
- Do comparison over time, to speed up statement analysis
- Match SQL statement across application version

See the possibilities

How to manage dynamic SQL reliably:

Step 4 – Create a Baseline

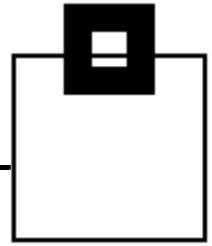
- Keep control of your environment
  - Quickly identify and understand performance degradation
  - Protect your production environment

Find out who executed the statement(s) and fix the problem not the symptom

- Using program, user ID, qualifier, …
- Teach/enable the programmers and/or apply reliable QA

→ Make dynamic SQL management a best practice

See the possibilities



How to manage dynamic SQL reliably:

Dynamic SQL management and protection:

Protecting your production environment from unforeseen performance degradations requires quality assurance.

A trend analysis system allows to pre-check the results from a

- New application version
- New DB2 version (or APARs affecting performance)
- New statistics
  - RUNSTATS can be painful in a dynamic environment
    - ANY RUNSTATS INVALIDATES THE AP IN THE DSC INCLUDING

```
RUNSTATS    UPDATE(NONE)
            HISTORY(NO)
            REPORT(NO)
```

See the possibilities

How to manage dynamic SQL reliably:

Dynamic SQL management and protection: setting up QA

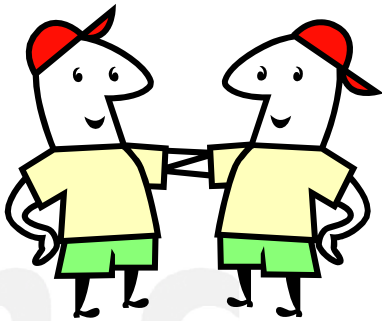
DB2P: Production

1. Take a snapshot of the Dynamic Statement Cache
2. Explain of all captured statements to central PLAN_TABLE

DB2Q: Quality Assurance

1. Homogeneous System Copy (or Catalog Statistics) of DB2P
2. Import the snapshot of the Dynamic Statement Cache
3. Explain all statements (needed)
4. Compare original and new

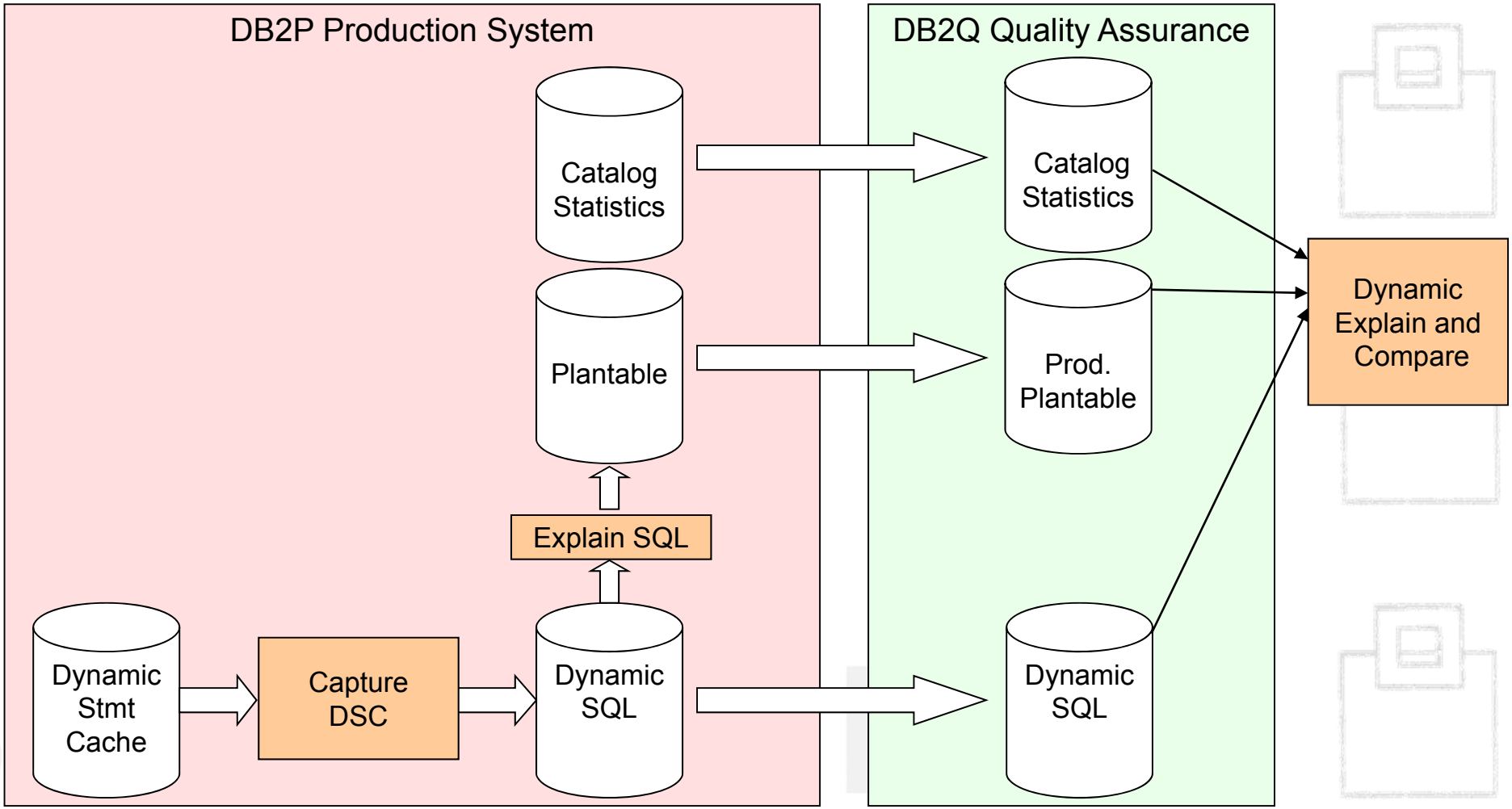
→ Allows to pre-check in a QA environment



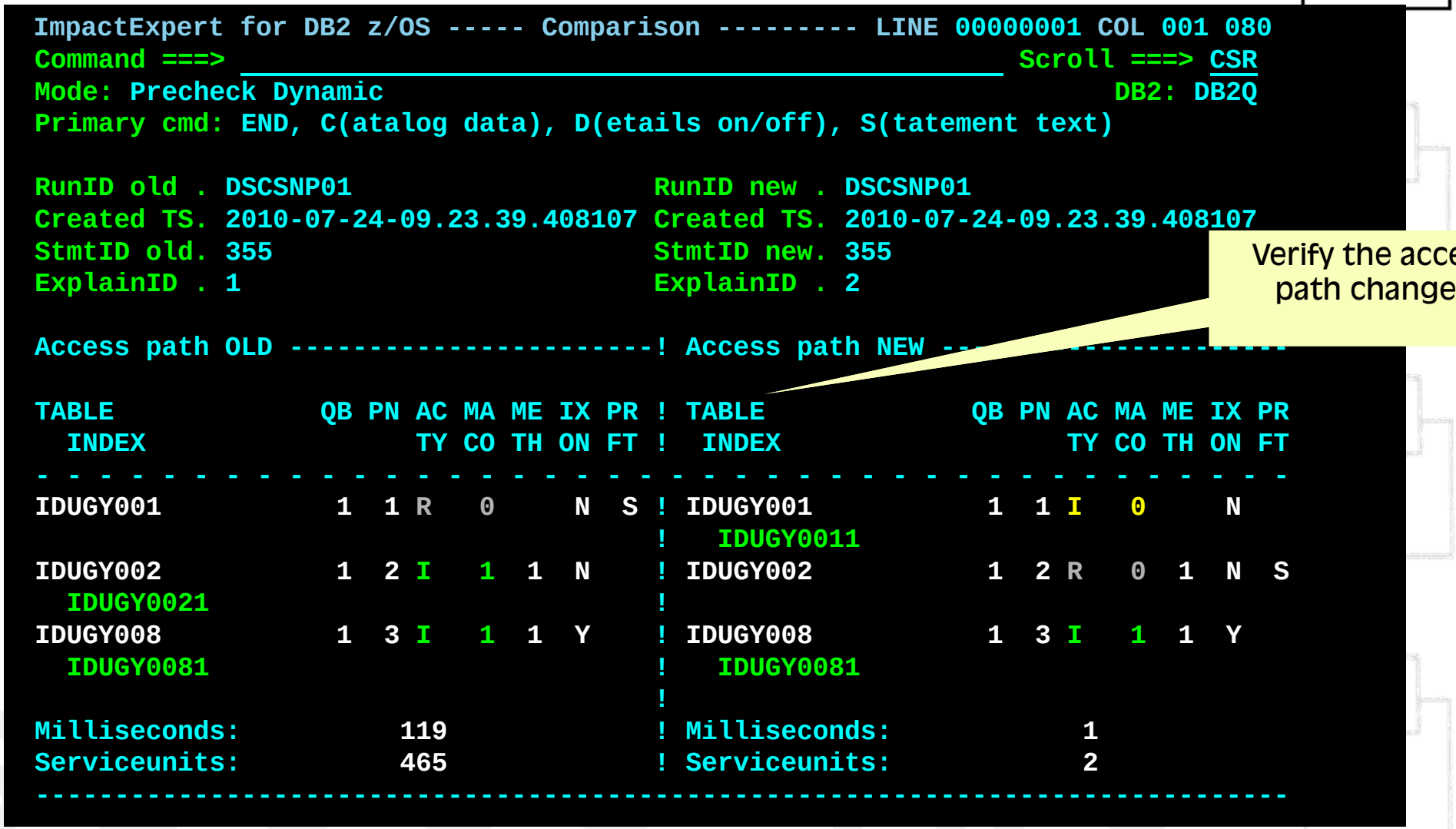
See the possibilities

How to manage dynamic SQL reliably:

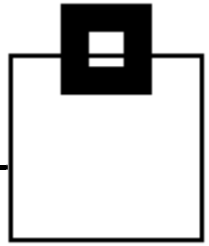
Dynamic SQL management and protection: setting up QA



See the possibilities



See the possibilities



How to manage dynamic SQL reliably:

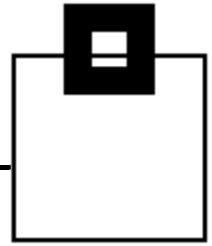
If you have difficulties finding the initiator of performance problems:

- Often dynamic SQL gets executed by an application server/common authorization ID
- Middleware usually connects to DB2 using common authorization/RACF ID
- The IP address may only show your DB2 Connect gateway
- User requires access to all objects (there is no package execution)

→Identifying the user performing a dynamic SQL can be challenging

- Since DB2 V8 there are client identification registers
  - CURRENT CLIENT_USERID
  - CURRENT CLIENT_WRKSTNNAME
  - CURRENT CLIENT_APPLNAME
  - CURRENT CLIENT_ACCTNG

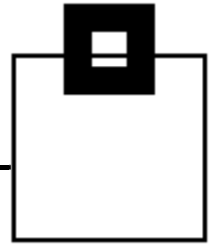
See the possibilities



How to manage dynamic SQL reliably:

- DB2 V8 (and above) client identification registers can bet set via
  - SQLESETI
    - SQLE_CLIENT_INFO_USERID
    - SQLE_CLIENT_INFO_WRKSTNNAME
    - SQLE_CLIENT_INFO_APPLNAME
    - SQLE_CLIENT_INFO_ACCTSTR
  - JDBC
    - DB2Connection.setDB2ClientUser(String info)
    - DB2Connection.setDB2ClientWorkstation(String info)
    - DB2Connection.setDB2ClientApplicationInformation(String info)
    - DB2Connection.setDB2ClientAccountingInformation(String info)
  - RRS sign on
    - RRS DSNRLI

See the possibilities

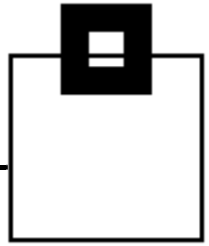


How to manage dynamic SQL reliably:

- DRDA Applications can use EXCSQLSET
  - SET CLIENT USERID „userid"
  - SET CLIENT WRKSTNNAME „wrkstn"
  - SET CLIENT APPLNAME „applname"
  - SET CLIENT ACCTNG „accounting"

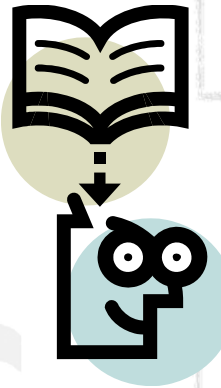
- Web Application Server provides –setClientInformation API
  - WSConnection.CLIENT_ID
  - WSConnection.CLIENT_LOCATION
  - WSConnection.CLIENT_APPLICATION_NAME
  - WSConnection.CLIENT_ACCOUNTING_INFO

See the possibilities

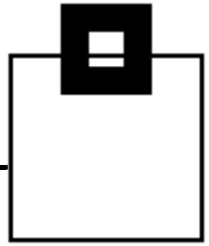


Conclusion:

- Dynamic SQL is more difficult to manage than static SQL
- Using the Dynamic Statement Cache can be a performance boost
- Access paths can get lost without changing anything
- Well known QA procedures for static SQL fit for dynamic SQL, but require adjustments
- Protecting dynamic environments is *just* more complex
- Security and authorization for dynamic SQL requires special considerations and adjustments



See the possibilities



What you can expect from exploiting it right:

- Flexibility in developing and running your applications
- Even more insight out of the box than in your static world
- Cost efficiency in development and operations

See the possibilities

References:

- IBM Redbook – Squeezing the Most out of Dynamic SQL

IDUG
The Worldwide DB2 User Community

Regional Events 2010

Ulf Heinrich

SEGUS Inc

u.heinrich@segus.com

Dynamic SQL Management and Protection