



# Db2 z/OS Dynamic SQL Monitoring: Best Practices CCDUG 2026

Toine Michielse ... presenting material mostly  
created by Michal Bialecki

[toine.michielse@broadcom.com](mailto:toine.michielse@broadcom.com) / [michal.bialecki@broadcom.com](mailto:michal.bialecki@broadcom.com)

# What is dynamic SQL

- Static program or stored procedure executing dynamic SQL

- PREPARE & EXECUTE

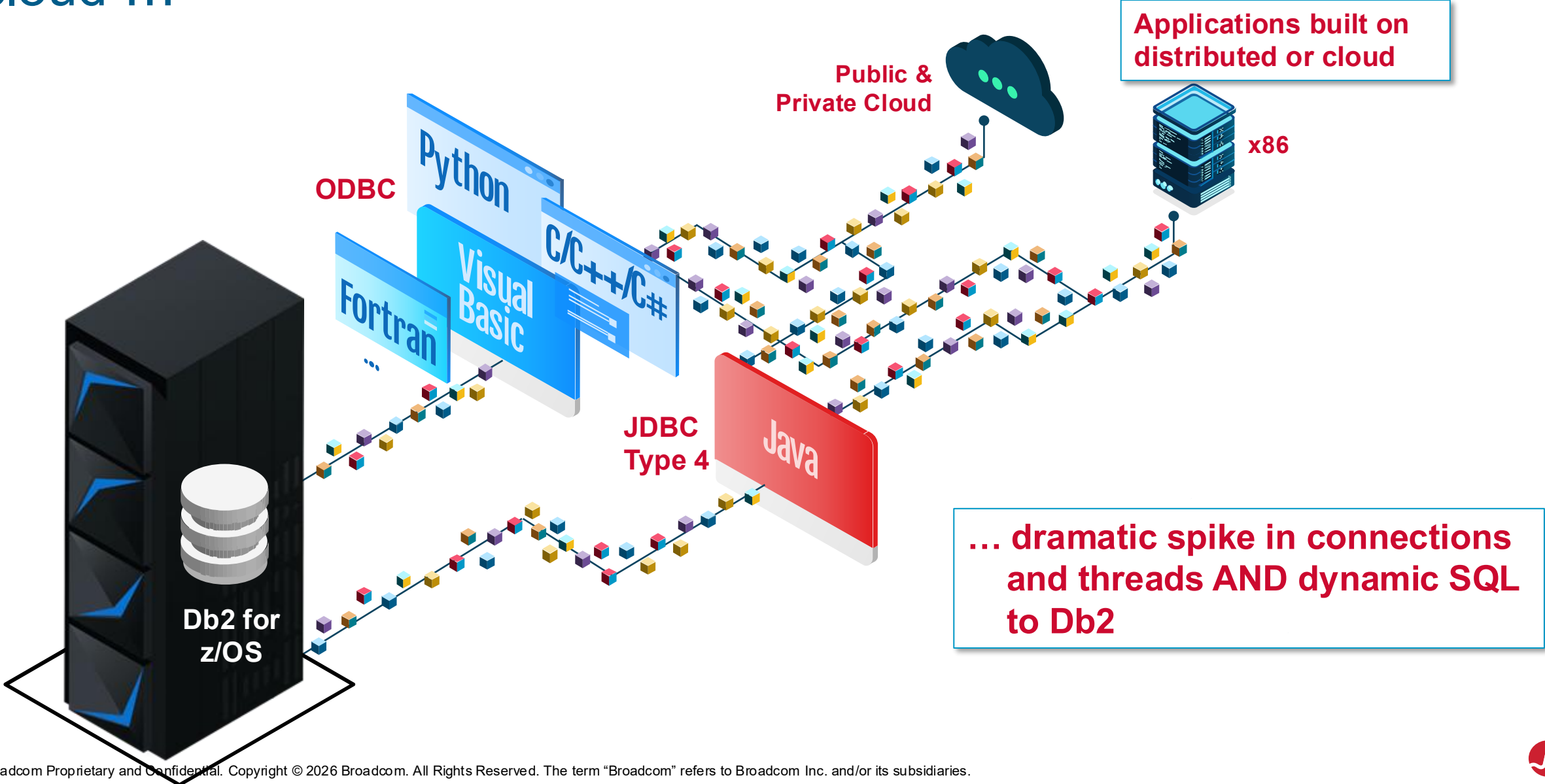
```
DECLARE C1 CURSOR FOR DYNSQL;  
DYNSQL_SELECT = 'SELECT X, Y, Z FROM TABLE_A WHERE X < 9';  
EXEC SQL PREPARE DYNSQL FROM :DYNSQL_SELECT;  
EXEC SQL OPEN C1;
```

- EXECUTE IMMEDIATE (can't be SELECT), and destroyed after execution

```
EXEC SQL EXECUTE IMMEDIATE 'DELETE FROM DSN8C10.DEPT
```



# Exponential growth in applications built off platform on x86 or cloud ...



# .....A puzzle



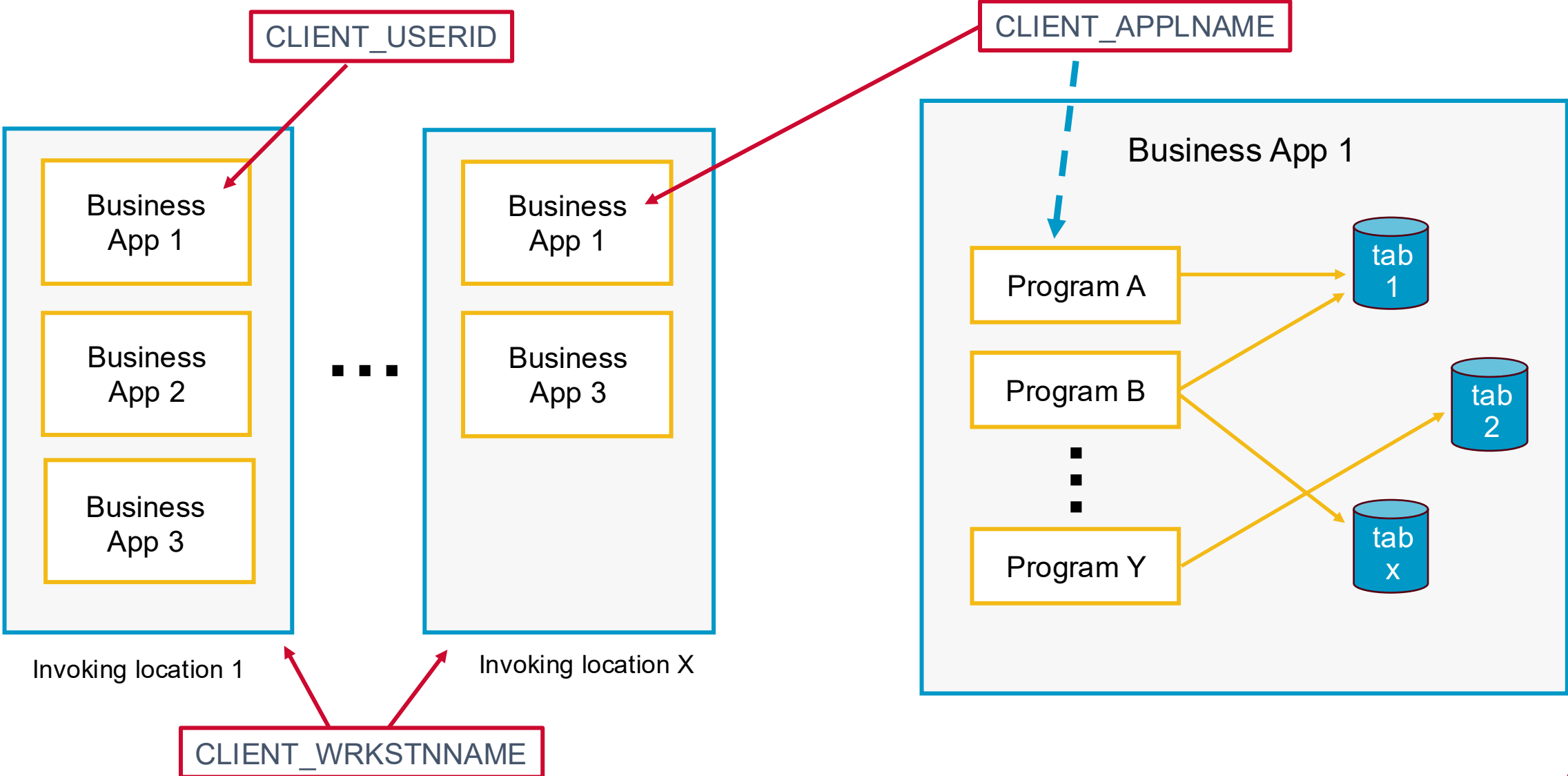
# Start with proper program identification

Once distributed applications have been identified, there are numerous benefits, including:

- Enable profiling in Db2 for z/OS
- Enable workload protection using WLM classification
- Enable different level of monitoring, dash boarding and capacity insights
- Enable proper charge back



# Positioning applications in a distributed world



# Dynamic SQL vs Static SQL

|                                     | Dynamic SQL  | Static SQL  |
|-------------------------------------|--|---|
| Access path                         | Determined during PREPARE time   | Determined at BIND / REBIND time *) with REOPT <> NONE also during run time if SQL with hostvariables |
| Executable form                     | Cached in dynamic statement cache  | Stored as Package   |
| Survives Db2 restart                | No, unless Plan Stability is used  | Yes   |
| Can go back to previous access path | No, but with Plan Stability - stores invalid copy (COPYID 4), where access path can be EXPLAINED | Yes, PLANMGMT(EXTENDED/BASIC)   |
| Phase-in                            | No   | Yes, with PLANMGMT(EXTENDED)  |
| Object dependencies                 | Not recorded, *) unless Plan Stability is used - SYSIBM.SYSDYNQRYDEP                             | Recorded in SYSIBM.SYSPACKDEP   |
| Invalidation                        | Yes, re-PREPARE needed   | Yes, AUTOBIND or REBIND needed  |
| Authorization                       | DYNAMICRULES (RUN) – object level<br>DYNAMICRULES (BIND) – package level                         | Privileges at package level (OWNER)   |

# Dynamic Statement Cache (DSC)

- Dynamic SQL needs to be PREPARED (bound) to generate an access path and executable form (runtime structure) of SQL
  - Access path selection process can be expensive, with complex SQL and over-collected statistics
  - the cost is incurred many times, if no caching
- Dynamic Statement Cache (for each Db2 member) is enabled by:
  - ZPARMs CACHEDYN=YES and EDMSTMTC > 0
- Dynamic Statement Cache (DSC) stores executable form of SQL, ready to be reused, matched on:
  - **Exact SQL text** (with literals concentration if used) – blanks and any variation matters, eg PREPARE ATTRIBUTES need to be same
  - Authorization id (authid), some special registers & bind options, APPLCOMPAT - need to be same (<https://www.ibm.com/docs/en/db2-for-zos/13.0.0?topic=cache-conditions-statement-sharing>)
  - **It is NOT matched on:**
    - CLIENT\_USERID - The client user ID name information.
    - CLIENT\_APPLNAME - The client application name information.
    - CLIENT\_WRKSTNNAME - The client workstation name information
    - Db2 tries to reuse cached SQL no matter from where it comes



# Dynamic Statement Cache (DSC) ..

- Statements in DSC are invalidated due to:
  - Changing dependent objects referenced by the statement eg ALTER, REVOKE or DROP
  - Authorization and access control changes
  - Collecting statistics for objects referenced by the statement with the RUNSTATS utility or inline STATISTICS in other utilities with the INVALIDATECACHE YES / UPDATE NONE REPORT NO / RESET ACCESS PATH option

NOTE: Invalidation does not remove statement from DSC, only marks it as invalid (not re-usable)
- Statement in DSC are removed:
  - When SQL is not in use and DSC pool size is hitting max - in LRU (least recently used) order
  - Db2 shutdown
    - BUT... Dynamic SQL plan stability (Db2 z/OS 12) can reload stabilized dynamic SQL from Db2 catalog tables at first PREPARE attempt after Db2 restart, or removal from DSC

# DSC – No caching

CACHEDYN=NO



## PROGA / USER1

PREPARE SQL1; Full prepare and copy to thread

EXECUTE SQL1; Execute of prepared SQL copy on thread

EXECUTE SQL1; Execute of prepared SQL copy on thread



## PROGB / USER1

PREPARE SQL1; Full prepare and copy to thread

EXECUTE SQL1; Execute of prepared SQL copy on thread

COMMIT; Deletion of prepared SQL copy from thread

EXECUTE SQL1; Stmt not prepared SQLCODE -514 / -518

PREPARE SQL1; Full prepare and copy to thread

EXECUTE SQL1; Execute of prepared SQL copy on thread



# DSC – Global caching

CACHEDYN=YES & EDMSTMTC > 0



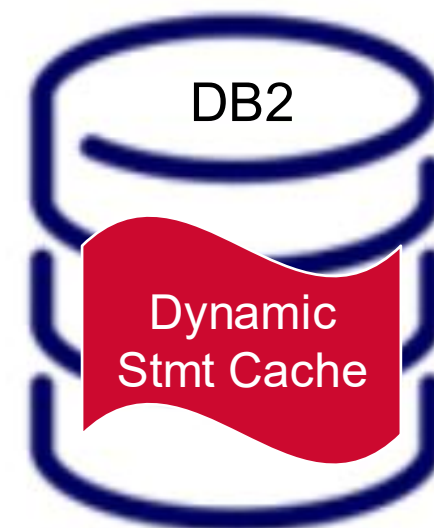
## PROGA / USER1

|               |  |
|---------------|--|
| PREPARE SQL1; | Full prepare and copy to thread          |
| EXECUTE SQL1; | Execute of prepared SQL – copy on thread |
| EXECUTE SQL1; | Execute of prepared SQL – copy on thread |



## PROGB / USER1

|               |   |
|---------------|---|
| PREPARE SQL1; | Short prepare from DSC – copy to thread   |
| EXECUTE SQL1; | Execute of prepared SQL – copy on thread  |
| COMMIT;       | Deletion of prepared SQL – copy on thread |
| EXECUTE SQL1; | Stmt not prepared SQLCODE -514 / -518     |
| PREPARE SQL1; | Short prepare from DSC – copy to thread   |
| EXECUTE SQL1; | Execute of prepared SQL – copy to thread  |



# DSC – Global caching (different user)

CACHEDYN=YES & EDMSTMTC > 0



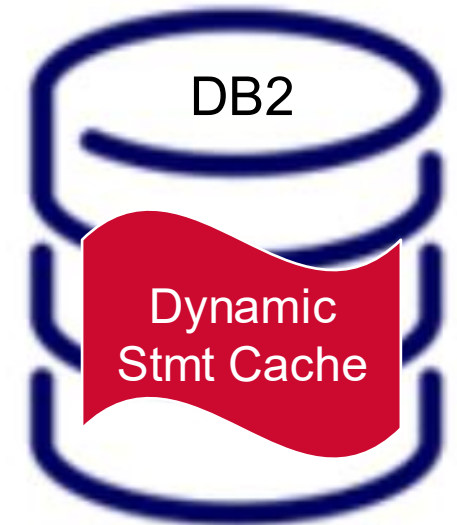
## PROGA / USER1

|               |  |
|---------------|--|
| PREPARE SQL1; | Full prepare and copy to thread          |
| EXECUTE SQL1; | Execute of prepared SQL – copy on thread |
| EXECUTE SQL1; | Execute of prepared SQL – copy on thread |



## PROGB / USER2

|               |   |
|---------------|---|
| PREPARE SQL1; | Full prepare and copy to thread           |
| EXECUTE SQL1; | Execute of prepared SQL – copy on thread  |
| COMMIT;       | Deletion of prepared SQL – copy on thread |
| EXECUTE SQL1; | Stmt not prepared SQLCODE -514 / -518     |
| PREPARE SQL1; | Short prepare from DSC – copy to thread   |
| EXECUTE SQL1; | Execute of prepared SQL – copy on thread  |



# Dynamic Statement Cache (DSC) pool monitoring



- DSC HitRatio > 90% - otherwise increase EDMSTMTC
  - QISEDSEI (The number of inserts into the dynamic statement cache)  
/ QISEDSEI (The number of requests for the dynamic statement cache)
- Equivalent in Broadcom SYSVIEW for Db2:
  - EDM\_INSERTS\_DYN / EDM\_DYN\_REQUESTS

SYSVIEW for Db2: 1. System Statistics → 3. EDM Pool 42

20.0.15 DTGPPR20

1 SnapShot 2 Buffer Pool 3 EDM Pool 4 Logs 5 Acct Sum 6 More...

R/EDMPOOL EDM Pool - Accum Row 2/23

| EDM Pool     | Total Pages | StIn HWM | StIn LWM | Full Fail | Requests | Loads | Pct Loads | Accum      |
|--------------|-------------|----------|----------|-----------|----------|-------|-----------|------------|
|              |             |          |          |           |          |       |           | 1234567890 |
| SKEL         | 3750        | 3750     |          | 0         |          |       |           |            |
| . SKCT Used  | 27          | 27       |          |           | 3169     | 14    | 0.4       | █          |
| . SKPT Used  | 300         | 300      |          |           | 265      | 46    | 17.4      | █          |
| . SKPT Steal | 327         |          | 327      |           |          |       |           |            |
| DBD          | 3750        | 3750     |          | 0         |          |       |           |            |
| . DBD Used   | 355         | 355      |          |           | 9532     | 33    | 0.3       | █          |
| . DBD Steal  | 227         |          | 224      |           |          |       |           |            |
| STMT         | 7500        |          |          | 0         |          |       |           |            |
| . STMT Used  | 1580        | 1580     |          |           | 505      | 194   | 38.4      | █          |
| . STMT Steal | 1550        |          | 1550     |           |          |       |           |            |

# DSC – Local Statement Cache pool (on thread)

- Enabled by KEEP DYNAMIC(YES) BIND option
- Not part of EDM POOLS, allocated on thread level storage in DBM1
  - Additional real storage requirement (per thread)
  - MAXKEEPD controls number of statements kept on thread level (default is 5k), FIFO order
- Comes with several implications:
  - Application logic/code needs to be **re-written** not to do subsequent PREPAREs after 1<sup>st</sup> PREPARE
  - Thread stays ACTIVE until it is deallocated (no pooling / INACTIVE)
  - Sysplex workload balancing functionality is practically limited **to Automatic Client Re-routing**,
    - does not rebalance work to different members at commit/rollback level as thread stays active and statement is “glued” to thread
    - rebalance happens every 200 (v12) /500 (v13) COMMITs
      - for “real” Sysplex WLB, every COMMIT/ROLLBACK

# DSC – Local caching (with NO app changes)

CACHEDYN=YES / EDMSTMTC > 0 / **KEEPDYNAMIC(YES)**



## PROGA / USER1

PREPARE SQL1; **Full** prepare and copy to thread

EXECUTE SQL1; Execute of prepared SQL – copy on thread

EXECUTE SQL1; Execute of prepared SQL – copy on thread



## PROGB / USER1

PREPARE SQL1; **Short** prepare and copy to thread

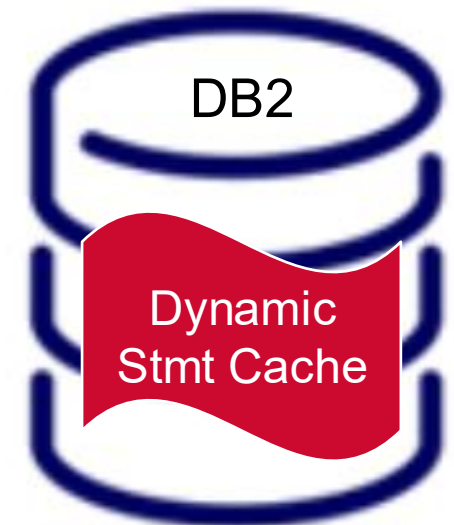
EXECUTE SQL1; Execute of prepared SQL – copy on thread

COMMIT; No deletion

EXECUTE SQL1; **Avoided** prepare – execution of copy on thread

PREPARE SQL1; **Short** prepare from DSC – copy to thread

EXECUTE SQL1; **Avoided** prepare – execution of copy on thread



NO APP CHANGE  
= NO BENEFIT !

# DSC – Local caching (with app changes)

CACHEDYN=YES / EDMSTMTC > 0 / **KEEPDYNAMIC(YES)**



## PROGA / USER1

PREPARE SQL1; **Full** prepare and copy to thread

EXECUTE SQL1; Execute of prepared SQL – copy on thread

EXECUTE SQL1; Execute of prepared SQL – copy on thread



## PROGB / USER1

PREPARE SQL1; **Short** prepare and copy to thread

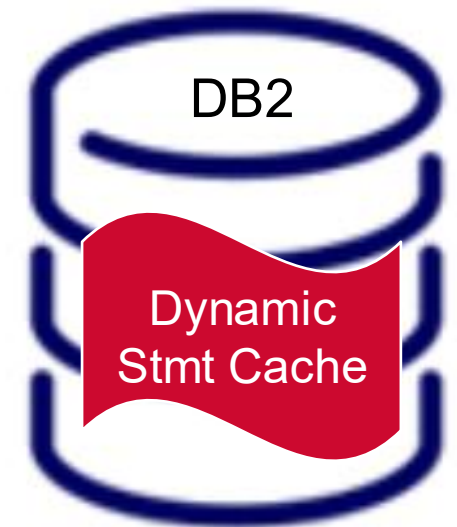
EXECUTE SQL1; Execute of prepared SQL – copy on thread

COMMIT; No deletion

EXECUTE SQL1; **Avoided** prepare – execution of copy on thread

~~PREPARE SQL1; **Short** prepare from DSC – copy to thread~~

EXECUTE SQL1; **Avoided** prepare – execution of copy on thread



# DSC + Dynamic Plan Stability

CACHEDYN=YES / EDMSTMTC > 0 / CACHEDYN\_STABILIZATION=BOTH



## PROGA / USER1

|               |  |
|---------------|--|
| PREPARE SQL1; | Full prepare and copy to thread          |
| EXECUTE SQL1; | Execute of prepared SQL – copy on thread |
| EXECUTE SQL1; | Execute of prepared SQL – copy on thread |

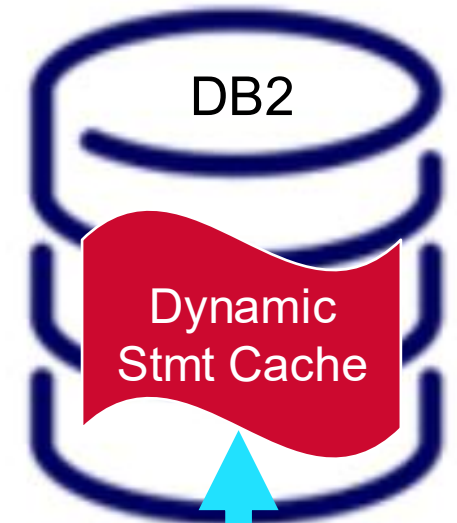
-START DYNQUERYCAPTURE... STMTID(n) which stores PREPARED copy of SQL with STMTID(n) in SYSIBM.SYSDYNQRY catalog table

Let's say that SQL is now removed from DSC



## PROGB / USER1

|               |   |
|---------------|---|
| PREPARE SQL1; | Load from SYSDYNQRY to DSC and Short prepare and copy to thread |
| EXECUTE SQL1; | Execute of prepared SQL – copy on thread                        |



SYSIBM.SYSDYNQRY

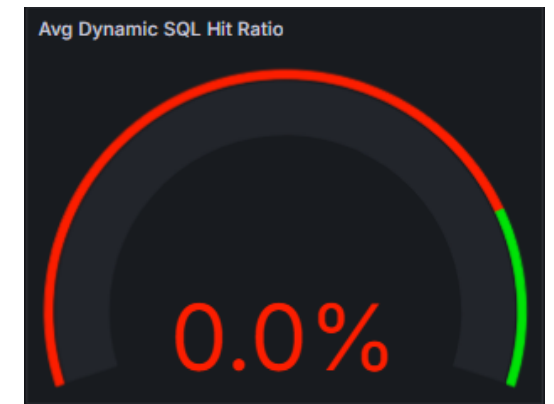
# DSC – Local Statement Cache pool (on thread)

- Local Statement Cache hit ratio > 70% otherwise increase MAXKEEPD



SYSVIEW for Db2: 1. System Statistics → 6. More → 16. Dynamic Prepare / Row Access

| R/SYSDYNP                                   | Dynamic Prepare / Row Access - Accum | Row 1/25<br>Accum |
|---|--------------------------------------|-------------------|
| <b>DYNAMIC PREPARE</b>                      |                                      |                   |
| Explicit PREPAREs issued                    | 175923                               | 0                 |
| Preps restricted index pend.                | 0                                    | 0                 |
| <b>GLOBAL DYNAMIC CACHE</b>                 |                                      |                   |
| Cache search requests                       | 247785                               | 0                 |
| Statements found in cache                   | 222704                               | 0                 |
| Statements not found in cache               | 24435                                | 0                 |
| Cache inserts - full prepares               | 23432                                | 0                 |
| Hit ratio without stabilized (%)            | 90.2                                 | 0                 |
| <b>LOCAL DYNAMIC CACHE</b>                  |                                      |                   |
| Implicit prepare performed                  | 0                                    | 0                 |
| Prepare avoided                             | 0                                    | 0                 |
| Stmts discarded - MAXKEEPD                  | 0                                    | 0                 |
| Stmts purged - dep. object                  | 22854                                | 0                 |
| Hit ratio (%)                               | 0.0                                  | 0                 |
| <b>STABILIZED DYNAMIC STATEMENTS</b>        |                                      |                   |
| Statement search requests                   | 25081                                | 0                 |
| Possible catalog row found                  | 805                                  | 0                 |
| Matched through text/bind                   | 664                                  | 0                 |
| Statements found                            | 664                                  | 0                 |
| Prepares satisfied                          | 646                                  | 0                 |
| Hit ratio incl. stabilized (%)              | 90.5                                 | 0                 |
| <b>DIRECT ROW ACCESS</b>                    |                                      |                   |
| Number of Times Successful                  |                                      | 0                 |
| Reverted to Using Index                     |                                      | 0                 |
| Reverted to Using TS Scan                   |                                      | 0                 |
| <b>TOTAL ROWS ACCESSED</b>                  |                                      |                   |
| Number of Rows Fetched                      |                                      | 19212K            |
| Number of Rows Inserted                     |                                      | 4746463           |
| Number of Rows Updated                      |                                      | 3187696           |
| Number of Rows Deleted                      |                                      | 154948            |
| <b>UNCOMMITTED ROW ACCESS</b>               |                                      |                   |
| Insert/Skipped                              |                                      | 0                 |
| Delete/Accessed                             |                                      | 0                 |
| Update/Accessed                             |                                      | 0                 |
| <b>CONCENTRATE STATEMENTS WITH LITERALS</b> |                                      |                   |
| Statements parsed                           |                                      | 0                 |
| Literal replaced                            |                                      | 0                 |
| Matching statements found                   |                                      | 0                 |
| Statement reuse failed                      |                                      | 0                 |



# DSC – Benefits (Global / Local)

|   | CPU cost                                | CPU savings per PREPARE |
|---|---|-------------------------|
| FULL PREPARE (no caching)                           | ~ 500.000-5.000.000 *) CPU instructions | 0.0 %                   |
| SHORT PREPARE (global caching)                      | ~ 10.000*) CPU instructions             | 98.0 %                  |
| AVOIDED PREPARE (global + local caching)            | ~ 2.000*) CPU instructions              | 99.6 %                  |
| PLAN STABILITY = Db2 Catalog I/O<br>+ SHORT PREPARE | IO cost + ~ 10.000*) CPU instructions   |                         |

\*numbers can vary depending on SQL

- When does it benefit to use Global Caching? **ALWAYS** when using dynamic SQL
- When does it benefit to use Local Caching? **SELECTIVELY**
  - for short / high frequently executed SQLs
  - when you do not use sysplexWLB balancing at transaction / commit / rollback boundary
  - when you can re-write application not to issue 2nd PREPARE before next EXECUTE
    - this is usually a challenge to do ...
    - Websphere can avoid re-prepares transparently with set Data Source properties keepDynamic=1

# DSC – Benefits of Dynamic Plan Stability?

- **Savings** thanks to dynamic plan stability as with a **short PREPARE** minus cost of I/O to load from SYSDYNQRY unless statement is in the DSC already
- When does it benefit to use Dynamic Plan Stability? **SELECTIVELY**
  - For **frequently** used SQL
    - based on selection filter by DSN\_STATEMENT\_CACHE\_TABLE.STAT\_EXEC or DSC-related IFCID 316 trace record – for example, **more than 1000** statement executions
  - When we want to **preserve dynamic statement access path** from removal from DSC:
    - Due to Db2 restarts - to avoid “tsunami” of PREPAREs
    - LRUed from Dynamic Statement Cache due to age, for example:
      - when workload profile changes from day/night or from normal day -> end of month processing, etc
  - When we are **sure that access path of statement is GOOD** (proven with EXPLAIN ..)
    - There is no easy way to revert to previous access path like with PLANMGMT for static packages
      - But at least you can EXPLAIN .. STABILIZED DYNAMIC QUERY COPYID(INVALID) previous access path

# DSC – Limitations of Dynamic Plan Stability

- Be aware of limitations:
  - Once statement is invalidated, it needs to be stabilized again
    - invalidation rules are same as for static packages
    - no “autobind” / “autostabilize” for Dynamic Plan Stability
  - Excluded SQL with CONCENTRATE STATEMENT WITH LITERALS
    - <https://ideas.ibm.com/ideas/DB24ZOS-I-1249>
  - Excluded REOPT (AUTO)
  - Excluded transformed SQL referencing system temporal, application temporal, or archived transparency tables



# Monitoring Dynamic Statements

# Monitoring Dynamic Statements / SQLs - prereqs

- IFCID 316/317/318 shall be turned ON to collect execution statistics, totals and full SQL statements:
  - IFCID 316 is to collect executions statistics and first 60 bytes of SQL
  - IFCID 317 is a toggle switch to get full SQL
  - IFCID 318 is a toggle switch to externalize runtime statistics (and totals)
  - Normally any monitoring tool turns those IFCIDs on, but also can be turned on manually, for “home grown” monitoring
  - Traces also can be directed to SMF, however don’t expect traces to be in immediately “live” with full SQL
    - When IFCID 316 is collected to SMF, then trace record is written:
      - ONLY when statement is LRU removed from Dynamic Statement cache,
      - **with partial SQL (first 60 bytes), regardless if IFCID 317 is turned on**
    - Invalidation does not remove statement from DSC, so it won’t also go to SMF then
    - During DB2 shutdown, no info is collected to SMF
- Access path info
  - externalizing dynamic statement cache **does not collect access path info** in PLAN\_TABLE and other explain tables
    - unless CURRENT EXPLAIN MODE = YES | EXPLAIN (but then prepared statements are not saved into the DSC)
  - This needs to be collected additionally per specified STMTID / STMTOKEN

# Monitoring Dynamic Statements / SQLs - prereqs

- How often shall I externalize Dynamic Statement Cache and access path info?
  - Simple answer: before unused statement info gets removed - LRUed
    - can be hours, or days, or minutes?
  - Before any workload switch, e.g. day / night / weekend / end of month
    - when we know that significantly different SQLs will be executing (so old ones will be LRUed)
  - Most tooling will do it with specified set interval, e.g. every 60 mins
  - Focus on top consumers, eg top30

```
== Dynamic Statement Text Capture Parameters ==  
HIST-IFC317-AVG-CPU-TOP=30,30  
HIST-IFC317-AVG-ELAP-TOP=30,30  
HIST-IFC317-TOT-ELAP-TOP=30,30  
HIST-IFC317-TOT-CPU-TOP=30,30
```

- How to externalize Dynamic Statement Cache
  - EXPLAIN STMTCACHE ALL (on every Db2 member separately)
- How to externalize access path info
  - EXPLAIN STMTCACHE STMTID / STMTOKEN
  - Note: this Access Path is from prepare time, when statement was entered into cache, not from explain time
    - If you have run RUNSTATS after statement was PREPARED, access path may differ comparing to EXPLAIN FOR explainable-sql-statement
      - V12 RUNSTATS by default does not invalidate a statement in DSC

# Monitoring Dynamic Statements / SQLs – how to

- Crucial to identify who is coming from where:
  - Enforce pgm annotation: CLIENT\_USERID, CLIENT\_APPLNAME, CLIENT\_WRKSTNNAME
    - may require application changes, jdbc driver properties, system profiles
    - have as “catch all” entry in RLF for those who don’t follow annotation?
      - DSNRLMTxx resource limit tables with RLFEUID / RLFEUWN / RLFIP set to blank apply to all users/all workstations,
      - except those who are specified explicitly (and you need to allow those “well behaving ones” to execute)
      - start with warning, then error to enable blocking rules
      - would need serious management support across teams and execution
  - OR system profiles to limit numbers of threads per non-annotated clients? – penalty box

| What the profile controls | DSN_PROFILE_ATTRIBUTES.KEYWORDS values | Applicable DSN_PROFILE_TABLE filtering categories  |
|---------------------------|--|--|
| Remote threads            | MONITOR THREADS                        | <ul style="list-style-type: none"> <li>– LOCATION only</li> <li>– PRDID only</li> <li>– AUTHID, ROLE, or both</li> <li>– COLLID, PKGNAME, or both</li> <li>– One of CLIENT_APPLNAME, CLIENT_USERID, or CLIENT_WRKSTNNAME</li> </ul> <p>The filtering values are not case-sensitive, and profiles can match regardless of the case of the input values.</p> |

- other ways to identify who is coming from where – eg technical userid per application,
  - if this granularity is enough for your needs?

# Monitoring Dynamic Statements / SQLs – how to

- Know your topN dynamic SQLs patterns / values / access path
- Store in database / graphical form / continuously monitor and alert
- Investigate outliers
  - EASILY SAID? DON'T BE AFRAID
    - start small from prototype, maybe from top5 ?
    - make an aim to tune few SQLs/week?

SYSVIEW for Db2: 10. SQL Exec Current Stats

```

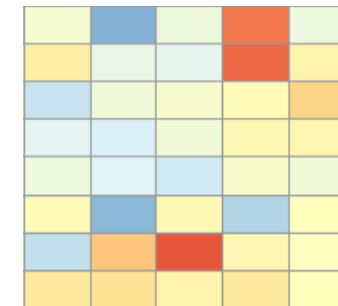
Menu  Print  Tools  Help  SYSVIEW for DB2  D13A CA31  08/08/25 09:35:26
                20.0.16  D13APR20
  1 DynCache  2 DynCache-Int  3 Static SQL-EDM  4 Static SQL-Int
                FOCUS OFF
R/DYNSQLST  Dynamic SQL Statements in Cache  Item 1/4114
SORT 7 D
Actions: S=Select for statement detail

```

| SQL Text<br>(16 bytes) | Cur<br>Usrs | Act<br>Cpys | Exec | Accum<br>Elapsed | Accum<br>CPU | Accum<br>zIIP | Program<br>Name | Stmt<br>Num | Getpgs  | Avg/Exec<br>Elapsed | Avg/Exec<br>CPU | Avg/Exec<br>zIIP |
|------------------------|-------------|-------------|------|------------------|--------------|---------------|-----------------|-------------|---------|---------------------|-----------------|------------------|
| SELECT count(*)        | 0           | 0           | 81   | 13:03.499        | 21:59.181    | 21:18.097     | THREAD1         | 71          | 72555K  | 9.672               | 16.286          | 15.778           |
| SELECT COUNT(*)        | 0           | 0           | 1    | 41.508           | 2.038        | 1.497         | SYSLH200        | 1           | 2539953 | 41.508              | 2.038           | 1.497            |
| SELECT JOBQUAL,        | 0           | 0           | 1    | 52.281           | 2.268        | 1.467         | SYSLH200        | 1           | 2539468 | 52.281              | 2.268           | 1.467            |
| SELECT COUNT(*)        | 0           | 0           | 1    | 43.255           | 2.136        | 1.436         | SYSLH200        | 1           | 2539810 | 43.255              | 2.136           | 1.436            |
| SELECT JOBQUAL,        | 0           | 0           | 1    | 50.132           | 2.327        | 1.401         | SYSLH200        | 1           | 2539413 | 50.132              | 2.327           | 1.401            |

# Monitoring on SQL level - know your topN dynamic SQLs

- Know what is a norm for topN dynamic SQL
  - TopN Executions
  - TopN CPU consumers
  - TopN getpages, sync I/O
  - Have a scoring table what is a norm and what is off
  - Scale / normalize:
    - Avg CPU per execution,
    - Avg Sync IO per execution, avg getpages per execution,
    - Avg numbers of getpages / rows returned
      - (\*credits to Thomas Bauman, IDUG 2025 NA and IDUG EMEA 2025 )
    - any indicator that would matter to you eg heatmap



# Monitoring on SQL level - know your topN dynamic SQLs

- EXPLAIN STMTCACHE ALL populates DSN\_STATEMENT\_CACHE\_TABLE
  - SYSADM or SQLADM needed to externalize all users
  - Needs to execute on Db2 member

- To populate PLAN\_TABLE additional step is needed

- EXPLAIN STMTCACHE STMTID [STMTTOKEN]

- STMTID or STMTTOKEN can be retrieved from DSN\_STATEMENT\_CACHE\_TABLE
- Query example to generate all EXPLAINS at once:

```
TRUNCATE DSN_STATEMENT_CACHE_TABLE; EXPLAIN STMTCACHE ALL ;
```

```
SELECT 'EXPLAIN STMTCACHE STMTID ' || CONCAT STRIP(CHAR(STMT_ID)) || CONCAT ' ;'  
FROM DSN_STATEMENT_CACHE_TABLE ;
```

- Output from above can be used to do the real EXPLAIN (after minor tweaks, like removing headers) and will populate PLAN\_TABLES and other explain tables (note: this is normal explain, NOT from time when statement was PREPARED)

```
----- RC/SQL - Browse  
COMMAND ===>  
28 ROWS RETRIEVED  
#1  
EXPLAIN STMTCACHE STMTID 2;  
EXPLAIN STMTCACHE STMTID 3;  
EXPLAIN STMTCACHE STMTID 4;  
EXPLAIN STMTCACHE STMTID 5;  
EXPLAIN STMTCACHE STMTID 6;  
EXPLAIN STMTCACHE STMTID 7;  
EXPLAIN STMTCACHE STMTID 8;
```

- Use the following predicates to join the DSN\_STATEMENT\_CACHE\_TABLE and the PLAN\_TABLE:

DSN\_STATEMENT\_CACHE\_TABLE.STMTID = PLAN\_TABLE.QUERYNO AND

DSN\_STATEMENT\_CACHE\_TABLE.CACHED\_TS = PLAN\_TABLE.BIND\_TIME AND

DSN\_STATEMENT\_CACHE\_TABLE.EXPANSION\_REASON = PLAN\_TABLE.EXPANSION\_REASON

# Monitoring on SQL level - know your topN dynamic SQLs

- Examples:

- selecting top30 CPU intensive SQLs and normalizing :

- SELECT PROGRAM\_NAME, PRMAUTH, CLIENT\_USERID, CLIENT\_APPLNAME, CLIENT\_WRKSTNNAME, STMT\_ID, STMT\_TEXT, **STAT\_CPU / STAT\_EXECB AS AVG\_CPU\_PER\_SQL** FROM DSN\_STATEMENT\_CACHE\_TABLE WHERE STAT\_EXECB > 1000 ORDER BY **AVG\_CPU\_PER\_SQL** FETCH FIRST 30 ROWS ONLY;

- Selecting top30 zIIP intensive SQLs and normalizing

- SELECT PROGRAM\_NAME, PRMAUTH, CLIENT\_USERID, CLIENT\_APPLNAME, CLIENT\_WRKSTNNAME, STMT\_ID, STMT\_TEXT, **STAT\_ZIIP\_CPU / STAT\_EXECB AS AVG\_ZIIP\_PER\_SQL** FROM DSN\_STATEMENT\_CACHE\_TABLE WHERE STAT\_EXECB > 1000 ORDER BY **AVG\_ZIIP\_PER\_SQL** FETCH FIRST 30 ROWS ONLY;

- \*) STAT\_ZIIP\_CPU (new V13 APAR PH64742)

# Monitoring on SQL level - know your topN dynamic SQLs

- Examples:

- selecting top30 sync IO intensive SQLs:

- SELECT PROGRAM\_NAME, PRMAUTH, CLIENT\_USERID, CLIENT\_APPLNAME,  
CLIENT\_WRKSTNNAME, STMT\_ID, STMT\_TEXT,  
STAT\_CPU / STAT\_EXECB as CPU\_PER\_SQL,  
**STAT\_SYNRB / STAT\_EXECB as SYNCIO\_PER\_SQL**,  
STAT\_GPAGB/STAT\_EXECB as GETPAGES\_PER\_SQL  
FROM DSN\_STATEMENT\_CACHE\_TABLE  
WHERE STAT\_EXECB > 100  
ORDER BY **SYNCIO\_PER\_SQL**  
FETCH FIRST 30 ROWS ONLY;

- Selecting SQLs doing tablespace scans:

- SELECT PROGRAM\_NAME, PRMAUTH, CLIENT\_USERID, CLIENT\_APPLNAME,  
CLIENT\_WRKSTNNAME, STMT\_ID, STMT\_TEXT, STAT\_CPU / STAT\_EXECB as CPU\_PER\_SQL,  
**STAT\_RSCNB**  
FROM DSN\_STATEMENT\_CACHE\_TABLE  
WHERE STAT\_EXECB > 100 and **STAT\_RSCNB > 0**  
ORDER BY **STAT\_RSCNB**  
;

# Other interesting fields in DSN\_STATEMENT\_CACHE\_TABLE

|                   |   |  |
|-------------------|---|--|
| STAT_EXECB        |   | The number of times this statement has been run  |
| STAT_GPAGB        |   | The number of getpage operations that are performed for the statement.   |
| <b>STAT_SYNRB</b> |   | The number of synchronous buffer reads that are performed for the statement.                                       |
| STAT_SORTB        |   | The number of sorts that are performed for the statement   |
| <b>STAT_PROWB</b> |   | The number of rows that are processed for the statement (* <b>STAT_EROWB</b> )                                     |
| STAT_INDXB        |   | The number of index scans that are performed for the statement   |
| <b>STAT_RSCNB</b> |   | The number of table space scans that are performed for the statement   |
| STAT_RIDLIMTB     | ⚠ | The number of times a RID list was not used because the number of RIDs would have exceeded Db2 limits              |
| STAT_RIDSTORB     | ⚠ | The number of times a RID list was not used because there is not enough storage available to hold the list of RIDs |
| CLIENT_USERID     |   | The client user ID name information <i>BEWARE: at cache time only, it is just 1<sup>st</sup> SQL value</i>         |
| CLIENT_APPLNAME   |   | The client application name information <i>BEWARE: at cache time only, it is just 1<sup>st</sup> SQL value</i>     |
| CLIENT_WRKSTNNAME |   | The client workstation name information <i>BEWARE: at cache time only, it is just 1<sup>st</sup> SQL value</i>     |

# Apple (SQL) to apple (SQL) compare

- Invention is on your side, you can store it in a time-series table (timestamp value DSN\_STATEMENT\_CACHE\_TABLE.CACHED\_TS) for easy compare what has changed overtime, heatmaps, charts, etc

| STMTID | STMT      | R-SCAN / TABLE SCAN ACCESS PATH |                  |                  |                  |                  |                  |                  |  |
|--------|-----------|---------------------------------|------------------|------------------|------------------|------------------|------------------|------------------|--|
|        |           | 04/05/2025 13:43                | 05/05/2025 13:43 | 06/05/2025 13:43 | 07/05/2025 13:43 | 08/05/2025 13:43 | 09/05/2025 13:43 | 10/05/2025 13:43 |  |
| 1      | SELECT1.. | 1                               | 1                | 0                | 1                | 0                | 1                | 1                |  |
| 2      | SELECT2.. | 1                               | 0                | 0                | 0                | 0                | 0                | 0                |  |
| 3      | SELECT3.. | 0                               | 1                | 1                | 0                | 1                | 1                | 0                |  |
| 4      | SELECT4.. | 1                               | 1                | 0                | 0                | 1                | 1                | 1                |  |
| 5      | SELECT5.. | 0                               | 0                | 1                | 1                | 0                | 0                | 1                |  |

- Challenge: compare APPLE to APPLE, same SQL to the same SQL
  - **STMTID** is just a sequence number, order of inserted SQLs into cache and **NOT a unique identifier**

# Apple (SQL) to apple (SQL) compare

- Are all apples the same?

```
SELECT * FROM SYSIBM.SYSTABLES;
```

```
Select * FROM sysibm.systables;
```

```
SELECT *  
FROM SYSIBM. SYSTABLES;
```

```
SELECT *          FROM SYSIBM. SYSTABLES          ;
```

```
SELECT *          FROM SYSIBM.SYSTABLES;
```



- They all will cache as separate cache entries, so can make things harder to analyze and compare, and focus on same SQL access path

- DB2 v12 comes to rescue - DSN\_STATEMENT\_CACHE\_TABLE.STMT\_HASHID2 is normalized SQL, so you can GROUP BY STMT\_HASHID2 same statements

STMT\_HASHID2

CHAR(8) FOR BIT  
DATA

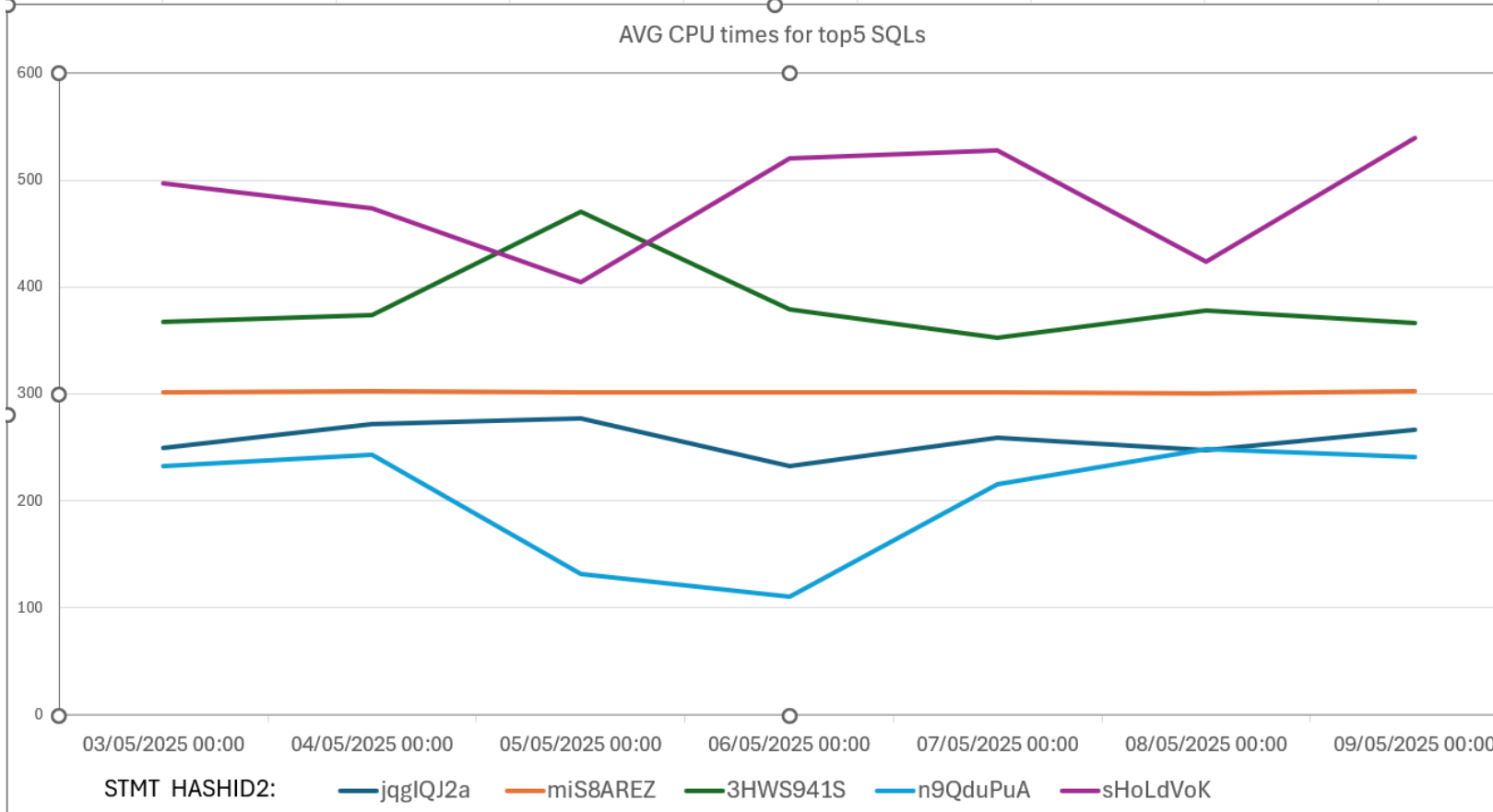
Used to identify an SQL statement. Based on normalized SQL statement text.

- Includes certain BIND options for static SQL, and PREPARE attributes for dynamic SQL.
- Includes COLLID and PACKAGE name for static SQL.
- Excludes COLLID and PACKAGE name for dynamic SQL.
- Excludes VERSION name for static SQL.

- it is not yet part of IFCID 316 trace, so home-grown monitors won't get it. Raised an AHA item (see pdf's end)

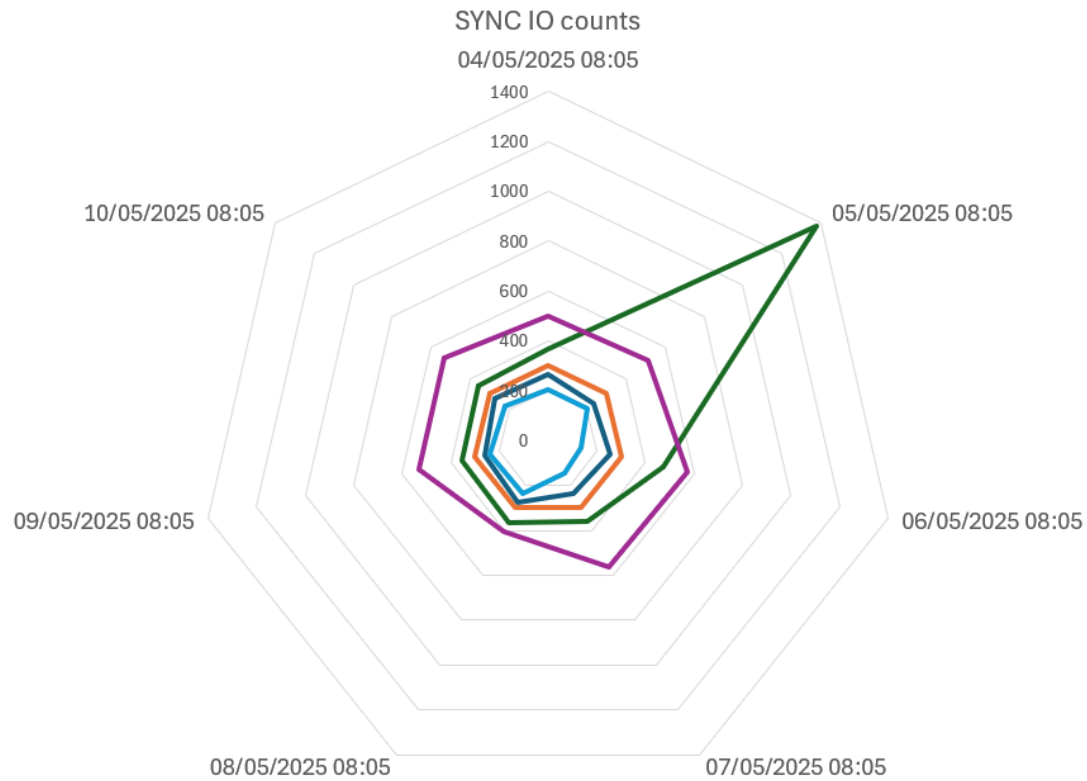
# Apple (SQL) to apple (SQL) compare

| STMT_HASHID2 | 03/05/2025 22:42 | 04/05/2025 22:42 | 05/05/2025 22:42 | 06/05/2025 22:42 | 07/05/2025 22:42 | 08/05/2025 22:42 | 09/05/2025 22:42 |
|--------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|
| jqglQJ2a     | 250              | 272              | 277              | 233              | 259              | 247              | 266              |
| miS8AREZ     | 302              | 303              | 301              | 302              | 301              | 300              | 303              |
| 3HWS941S     | 367              | 374              | 470              | 379              | 352              | 378              | 366              |
| n9QduPuA     | 233              | 243              | 132              | 110              | 215              | 248              | 241              |
| sHoLdVoK     | 497              | 474              | 404              | 520              | 528              | 424              | 539              |



# Apple (SQL) to apple (SQL) compare

| STMT_HASHID2 | 04/05/2025 08:05 | 05/05/2025 08:05 | 06/05/2025 08:05 | 07/05/2025 08:05 | 08/05/2025 08:05 | 09/05/2025 08:05 | 10/05/2025 08:05 |
|--------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|
| jqglQJ2a     | 267              | 236              | 254              | 234              | 278              | 263              | 273              |
| miS8AREZ     | 300              | 302              | 303              | 301              | 301              | 300              | 300              |
| 3HWS941S     | 369              | 1379             | 476              | 363              | 366              | 355              | 355              |
| n9QduPuA     | 204              | 204              | 133              | 147              | 238              | 238              | 223              |
| sHoLdVoK     | 497              | 511              | 573              | 565              | 406              | 531              | 533              |



STMT\_HASHID2: —jqglQJ2a —miS8AREZ —3HWS941S —n9QduPuA —sHoLdVoK

# Apple (SQL) to apple (SQL) compare

SYNC IO heatmap example

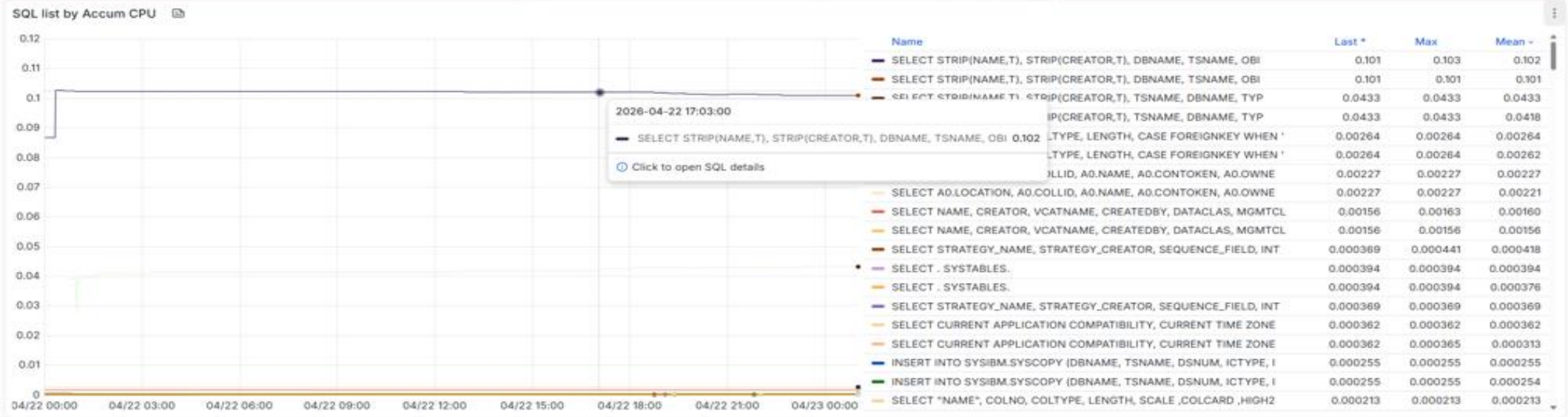
| SQL       | STMT_HASHID2 | 11/05/2025 15:07 | 12/05/2025 15:07 | 13/05/2025 15:07 | 14/05/2025 15:07 | 15/05/2025 15:07 | 16/05/2025 15:07 | 17/05/2025 15:07 | 18/05/2025 15:07 | 19/05/2025 15:07 | 20/05/2025 15:07 | 21/05/2025 15:07 | 22/05/2025 15:07 | 23/05/2025 15:07 | 24/05/2025 15:07 | 25/05/2025 15:07 | 26/05/2025 15:07 | 27/05/2025 15:07 | 28/05/2025 15:07 | 29/05/2025 15:07 | 30/05/2025 15:07 | 01/06/2025 15:07 | 02/06/2025 15:07 | 03/06/2025 15:07 | 04/06/2025 15:07 |
|-----------|--------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|------------------|
| SELECT1.. | jqgIQJ2a     | 246              | 261              | 246              | 246              | 252              | 276              | 251              | 265              | 269              | 258              | 244              | 265              | 248              | 254              | 260              | 248              | 262              | 266              | 239              | 251              | 240              | 239              | 264              | 276              |
| SELECT2.. | miS8AREZ     | 303              | 301              | 301              | 300              | 302              | 303              | 301              | 301              | 303              | 302              | 300              | 302              | 300              | 301              | 302              | 303              | 303              | 303              | 300              | 303              | 301              | 303              | 301              | 301              |
| SELECT3.. | 3HWS941S     | 377              | 1355             | 477              | 381              | 383              | 365              | 350              | 371              | 374              | 370              | 368              | 383              | 374              | 378              | 373              | 354              | 379              | 356              | 366              | 372              | 369              | 362              | 355              | 380              |
| SELECT4.. | n9QduPuA     | 250              | 220              | 127              | 127              | 242              | 205              | 224              | 244              | 232              | 241              | 215              | 238              | 212              | 202              | 226              | 231              | 211              | 201              | 213              | 235              | 227              | 217              | 230              | 225              |
| SELECT5.. | sHoLdVoK     | 510              | 534              | 457              | 541              | 451              | 445              | 536              | 500              | 511              | 405              | 424              | 596              | 545              | 577              | 472              | 553              | 444              | 469              | 462              | 559              | 505              | 589              | 520              | 471              |
| SELECT6.. | gh87adsd     | 473              | 473              | 423              | 425              | 424              | 521              | 498              | 593              | 562              | 468              | 594              | 513              | 557              | 562              | 467              | 542              | 588              | 561              | 443              | 509              | 406              | 491              | 494              | 471              |
| SELECT7.. | y67as9e1     | 577              | 532              | 542              | 545              | 495              | 531              | 436              | 459              | 466              | 557              | 539              | 569              | 410              | 588              | 475              | 409              | 444              | 434              | 494              | 562              | 553              | 434              | 519              | 495              |
| SELECT8.. | hijaow7e     | 545              | 551              | 458              | 428              | 409              | 485              | 521              | 504              | 406              | 530              | 470              | 594              | 435              | 486              | 534              | 460              | 567              | 441              | 415              | 499              | 544              | 590              | 497              | 488              |



# Sneak preview....

| Total Executions   | Accum CPU Time  | Total Getpages  | Total Tablespace Scans  |
|--|---|---|---|
| SELECT AQ.TYPE FROM SYSIBM.SYSTABLES AQ WHERE AQ.CREATOR = '1'<br><b>752925</b>  | SELECT STRIP(NAME,T), STRIP(CREATOR,T), DBNAME, TSNAME, OBI<br><b>5.46 mins</b> | SELECT AQ.LOCATION, AQ.COLLD, AQ.NAME, AQ.CONTOKEN, AQ.OWNE<br><b>16.6 Mil</b>    | SELECT STRIP(NAME,T), STRIP(CREATOR,T), DBNAME, TSNAME, OBI<br><b>3.26 K</b>  |
| SELECT ASBANTER, ASBANTEE, ASBANTETYPE, ASBANTOR, A<br><b>175848</b>             | SELECT AQ.LOCATION, AQ.COLLD, AQ.NAME, AQ.CONTOKEN, AQ.OWNE<br><b>48.5 s</b>    | SELECT STRIP(NAME,T), STRIP(CREATOR,T), DBNAME, TSNAME, OBI<br><b>11.6 Mil</b>    | SELECT NAME, CREATOR, VCATNAME, CREATEDBY, DATACLAS, MGMTCL<br><b>3.08 K</b>  |
| SELECT 'T' FROM sys_jnl_c0 WHERE JRNNAME='1' AND JNL_SYSTEM='1'<br><b>118384</b> | SELECT STRIP(NAME,T), STRIP(CREATOR,T), TSNAME, DBNAME, TYP<br><b>27.4 s</b>    | SELECT AQ.TYPE FROM SYSIBM.SYSTABLES AQ WHERE AQ.CREATOR = '1'<br><b>2.26 Mil</b> | SELECT DISTINCT AQ.SQNAME FROM SYSIBM.SYSVOLUMES AQ WHERE AQ<br><b>2.27 K</b> |
| SELECT ADSEQUENCEID, AD.DCREATOR, AD.BMREQD, AD.DNAME, AD<br><b>108708</b>       | SELECT AQ.TYPE FROM SYSIBM.SYSTABLES AQ WHERE AQ.CREATOR = '1'<br><b>7.69 s</b> | SELECT STRIP(NAME,T), STRIP(CREATOR,T), TSNAME, DBNAME, TYP<br><b>2.01 Mil</b>    | SELECT .<br><b>1.70 K</b>   |

| Avg Accum CPU Time   | Avg Getpages   | Avg Tablespace Scans  |
|--|--|---|
| SELECT STRIP(NAME,T), STRIP(CREATOR,T), DBNAME, TSNAME, OBI<br><b>101 ms</b> | SELECT STRIP(NAME,T), STRIP(CREATOR,T), TSNAME, DBNAME, TYP<br><b>3.61 K</b> | SELECT STRIP(NAME,T), STRIP(CREATOR,T), DBNAME, TSNAME, OBI<br><b>3.26 K</b>  |
| SELECT STRIP(NAME,T), STRIP(CREATOR,T), TSNAME, DBNAME, TYP<br><b>49 ms</b>  | SELECT STRIP(NAME,T), STRIP(CREATOR,T), DBNAME, TSNAME, OBI<br><b>3.60 K</b> | SELECT NAME, CREATOR, VCATNAME, CREATEDBY, DATACLAS, MGMTCL<br><b>3.08 K</b>  |
| SELECT AQ.LOCATION, AQ.COLLD, AQ.NAME, AQ.CONTOKEN, AQ.OWNE<br><b>2 ms</b>   | SELECT AQ.LOCATION, AQ.COLLD, AQ.NAME, AQ.CONTOKEN, AQ.OWNE<br><b>790</b>    | SELECT DISTINCT AQ.SQNAME FROM SYSIBM.SYSVOLUMES AQ WHERE AQ<br><b>2.27 K</b> |
| SELECT NAME, CREATOR, VCATNAME, CREATEDBY, DATACLAS, MGMTCL<br><b>1 ms</b>   | SELECT . SYSTABLES.<br><b>116</b>  | SELECT .<br><b>1.70 K</b>   |



SELECT STRIP(NAME,T), STRIP(CREATOR,T), DBNAME, TSNAME, OBI

SQL Statistics

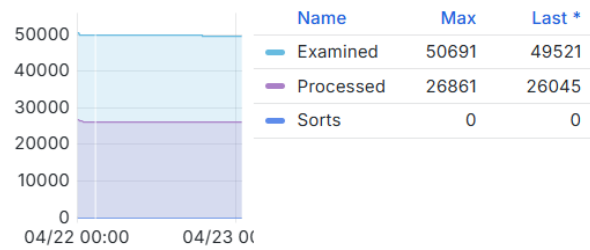
Users: 0    Executions: 3217    Accel Executions: 0    Copies: 0    Unique: 1D92    Accel Unique: 0    Stabilized: 31A

Identification Fields

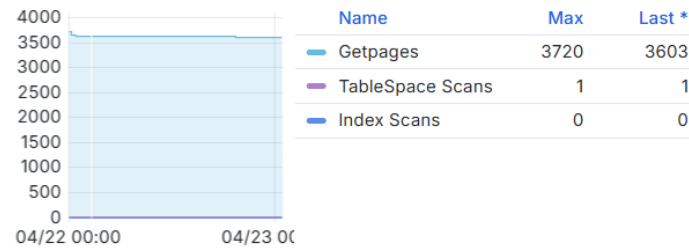
UserID

Program Name

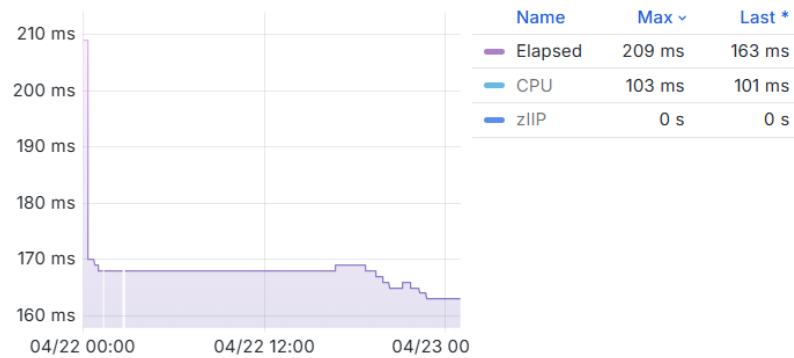
Avg Rows Activity



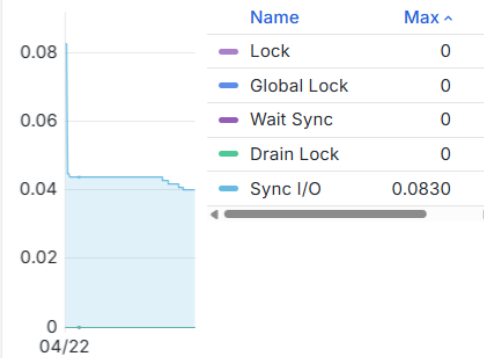
Avg I/O Activity



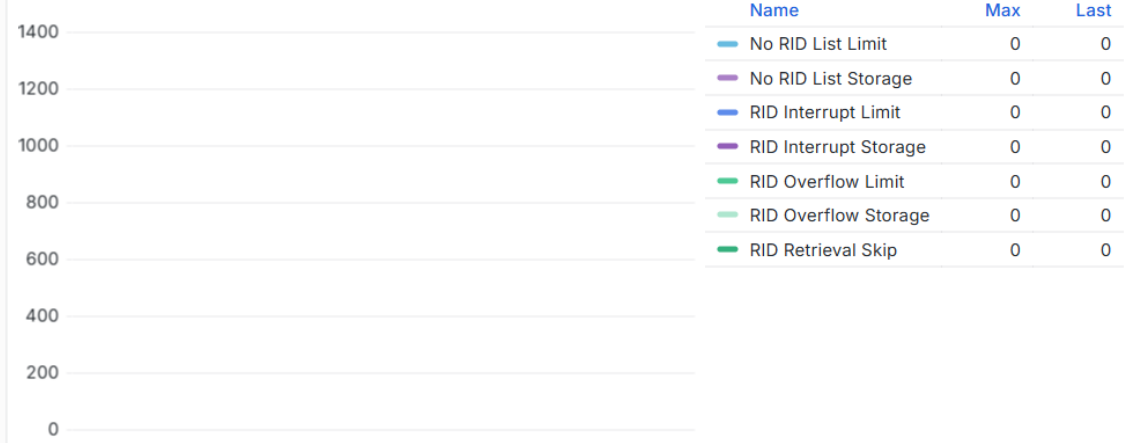
Avg Db2 Times



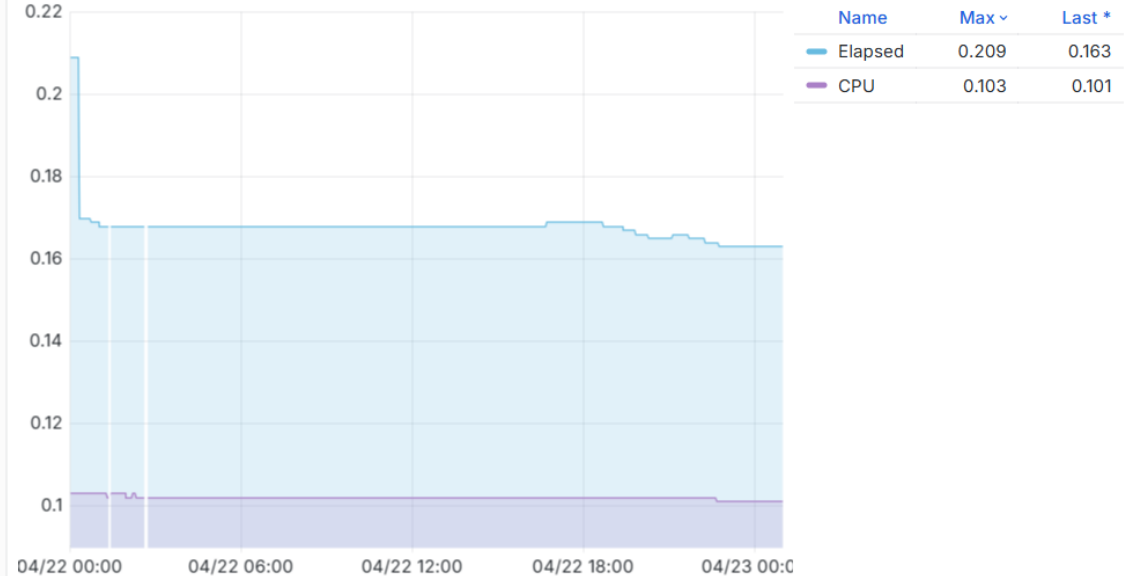
Avg Wait Times



Total RID Statistics



Avg Accel Times



# Thank You!!!