PDUG

Warsaw 25-26 September 2025

DYNAMIC SQL MONITORING: BEST PRACTICES

Michal.Bialecki
BROADCOM

POLISH DB2 USERS GROUP

Let me quickly introduce myself

- Michal Bialecki, Db2 technical consultant @Broadcom
  - Previously: DBA, defect support, consulting, performance studies and health checks, co-founder of PDUG (Polish Db2 Users group)
- Worked for many many++ years for IBM as Db2 for z/OS L2 defect support
  - and last 5 years in DB2 z/OS SWAT team
- Joined Broadcom in April 2025
- My passions:
  - climbing / snowboard

BROADCOM

Agenda

- Dynamic SQL overview
- Monitoring dynamic SQL

BROADCOM

Dynamic SQL overview

What is dynamic SQL

- SPUFI / DSNTEP2/4 / DSNTIAUL / REXX / **JDBC** / CLI / Db2 Interactive
- **Other remote databases, cloud pods?**

```
IQPSQL3O 20.0.16 -------   ISQL Online
COMMAND ===>

Option    ===> *
DB2 SSID  ===> DT31
-------------------------------------

OPTIONS:                                C
 S  - SQL Execution
 D  - Dataset I/O
 BP - Batch Processor/Submit
 E  - Edit SQL



SQL TO BE EXECUTED:

SELECT COUNT(*) FROM SYSIBM.SYSTABLES;
```

- 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), destroyed after execution

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


BROADCOM

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 it used	Yes
Can go back to previous access path	No, but with Plan Stability - stores invalid copy (COPYID 4), where access path is can be EXPLAINed	Yes, PLANMGMT(EXTENDED/BASIC)
Phase-in	No	Yes, with PLANMGMT(EXTENDED)
Object dependencies	Not recorded, *) unless Plan Stability it used - SYSIBM.SYSDYNQRYDEP	Recorded in SYSIBM.SYSPACKDEP
Invalidation	Yes, re-PREPARE needed	Yes, AUTOBIND or REBIND needed
Authorization	DYNAMICRULES (RUN) – object level 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 variations matters, eg PREPARE ATTRIBUTES need to be same
  - Authorisation 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 non-executable

- Statement in DSC are removed:
  - When SQL is not in use and DSC pool size in 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

BROADCOM

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

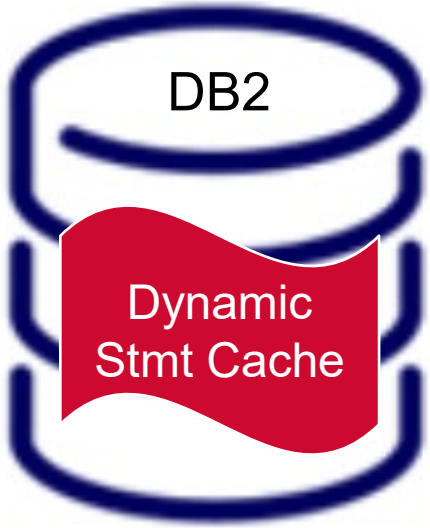
PROGB / USER1	
PREPARE SQL1;	Full prepare and copy to thread
EXECUTE SQL1;	Execute of prepared SQL copy on thread
COMMIT;	Deletion of prepared SQL copy from thread
EXECUTE SQL1;	Stmt not prepared SQLCODE -514 / -518
PREPARE SQL1;	Full prepare and copy to thread
EXECUTE SQL1;	Execute of prepared SQL copy 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

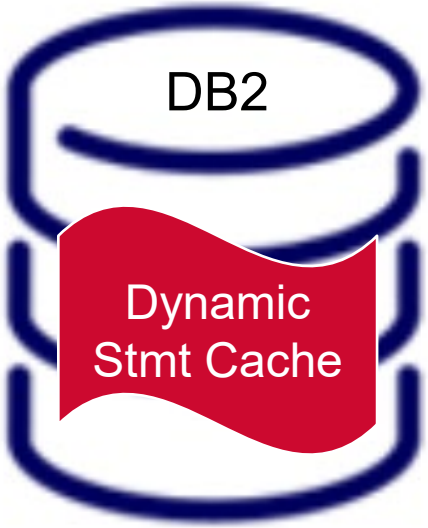


DSC – Global caching (different user)

CACHEDYN=YES & EDMSTMTC > 0

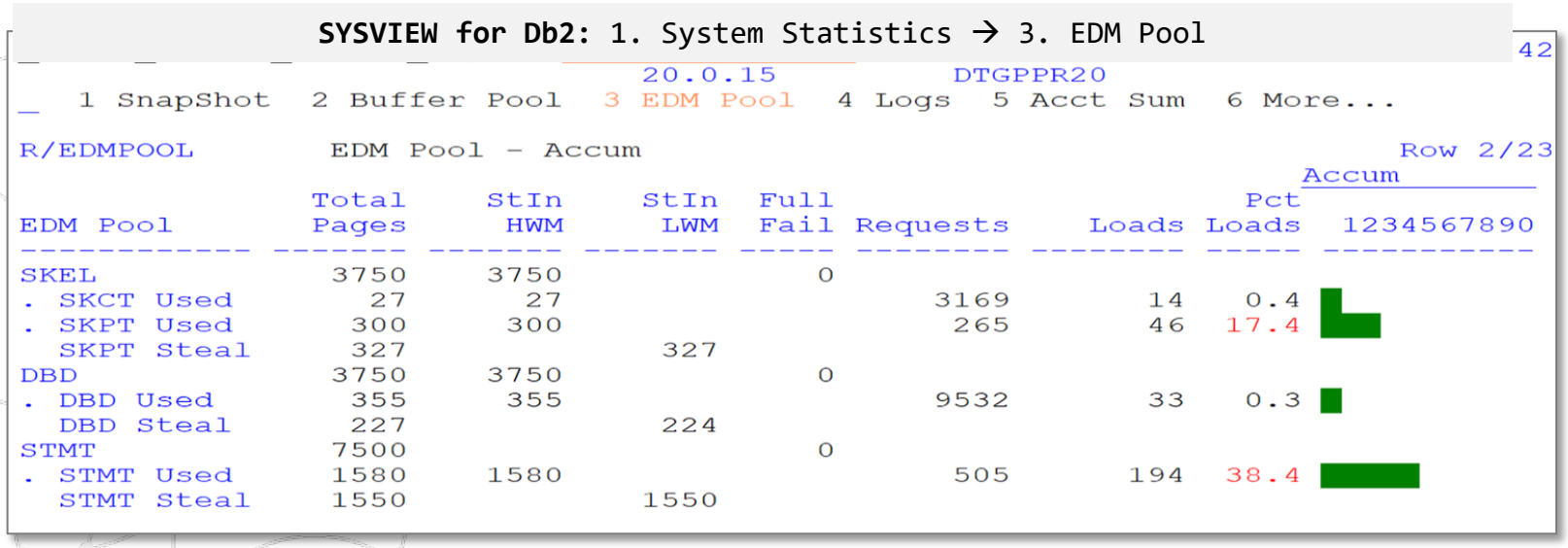
PROGA / USER1	
PREPARE SQL1;	Full prepare and copy to thread
EXECUTE SQL1;	Execute of prepared SQL – copy on thread
EXECUTE SQL1;	Execute of prepared SQL – copy on thread

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
  - QISEDSI (The number of inserts into the dynamic statement cache) / QISEDSG (The number of requests for the dynamic statement cache)
- Equivalent in Broadcom SYSVIEW:
  STMT Pct Loads < 10% (100%-hit ratio)
  - EDM_INSERTS_DYN / EDM_DYN_REQUESTS



DSC – Local Statement Cache pool (on thread)

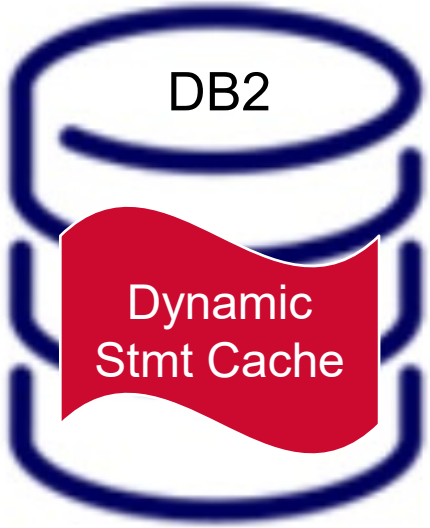
- Enabled by KEEPDYNAMIC(YES) BIND option
- Not part of EDM POOLs, allocated on thread level storage in DBM1
  - Additional real storage requirement (per thread)
  - MAXKEEPD controls number of statement kept on thread level (default is 5k), FIFO order
- Comes with several implications:
  - Application logic/code needs to be **re-written** not to do subsequent PREPAREs after 1st PREPARE
  - Thread stays ACTIVE until it is deallocated (no pooling / INACTIVE)
  - Limits Sysplex workload balancing to Automatic Client Re-routing only,
    - does not rebalance work to different members at commit/rollback level as thread stays active and statement is “glued” to thread

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 !

BROADCOM

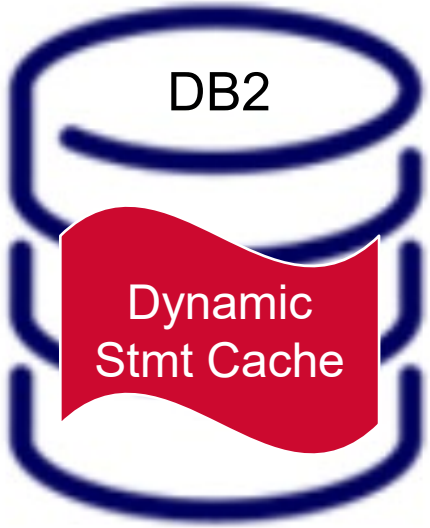
DSC – Local caching (with app changes)

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

PREPARE REMOVED FROM APPLICATION CODE



BROADCOM

DSC – Local Statement Cache pool (on thread)

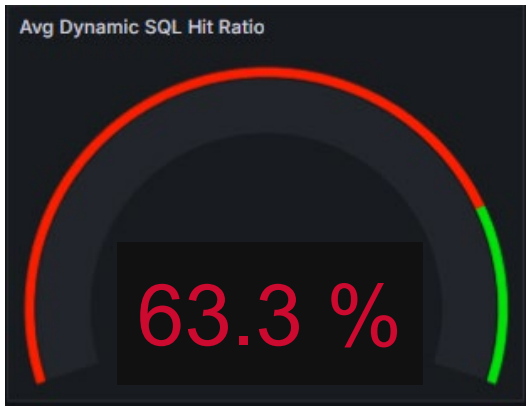
- Local Statement Cache hit ratio > 70% otherwise increase MAXKEEPD
  - IPREP_FOUND / (IPREP_FOUND + IPREP_NOFND)

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

```
R/SYSDYNP       Dynamic Prepare / Row Access - Accum                Row 1/25
                                                                    Accum
DYNAMIC PREPARE                                 DIRECT ROW ACCESS
  Explicit PREPAREs issued          175923      Number of Times Successful      0
  Preps restricted index pend.           0      Reverted to Using Index         0
                                                Reverted to Using TS Scan       0
GLOBAL DYNAMIC CACHE
  Cache search requests             247785      TOTAL ROWS ACCESSED
  Statements found in cache         222704      Number of Rows Fetched      19212K
  Statements not found in cache      24435      Number of Rows Inserted    4746463
  Cache inserts - full prepares      23432      Number of Rows Updated     3187696
  Hit ratio without stabilized (%)    90.2      Number of Rows Deleted      154948

LOCAL DYNAMIC CACHE                             UNCOMMITTED ROW ACCESS
  Implicit prepare performed             0      Insert/Skipped                  0
  Prepare avoided                        0      Delete/Accessed                 0
  Stmts discarded - MAXKEEPD             0      Update/Accessed                 0
  Stmts purged - dep. object         22854
  Hit ratio (%)                        0.0

STABILIZED DYNAMIC STATEMENTS                   CONCENTRATE STATEMENTS WITH LITERALS
  Statement search requests          25081      Statements parsed               0
  Possible catalog row found           805      Literal replaced                0
  Matched through text/bind            664      Matching statements found       0
  Statements found                     664      Statement reuse failed          0
  Prepares satisfied                   646
  Hit ratio incl. stabilized (%)      90.5
```



BROADCOM

DSC + Dynamic Plan Stability

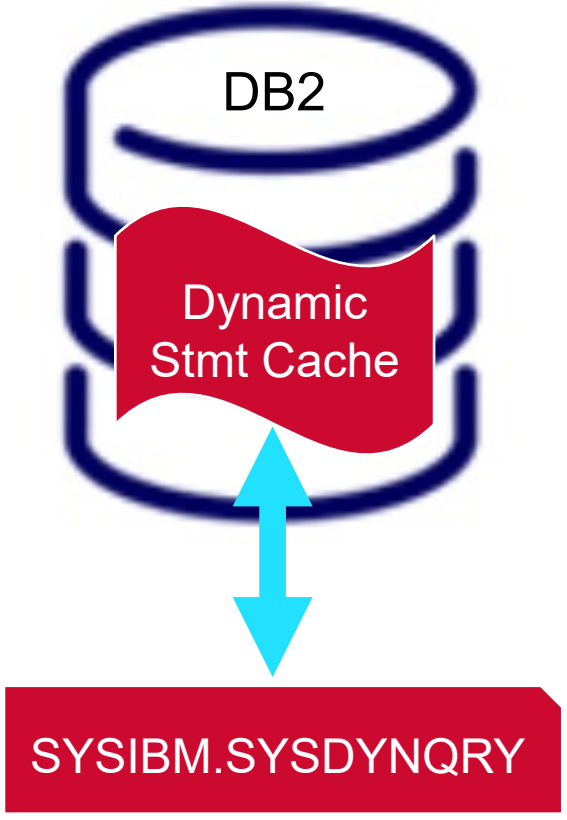
CACHEDYN=YES / EDMSTMTC > 0 / **CACHEDYN_STABILIZATION=BOTH**

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



DSC – Benefits (Global / Local)

		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 + SHORT PREPARE	IO + ~ 10.000*) CPU instructions	

*numbers can vary depending on SQL

- When it benefits to use Global Caching? **ALWAYS** when using dynamic SQL
- When it benefits 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 – Benefit of Dynamic Plan Stability?

- **Savings** thanks to dynamic plan stability are same as with a **short PREPARE** minus cost of I/O to load from SYSDYNQRY unless statement is in the DSC already
- When it benefits 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” for Dynamic Plan Stability
  - Excluded SQL with CONCENTRATE STATEMENT WITH LITERALS and REOPT (AUTO)
  - Excluded transformed SQL referencing system temporal, application temporal, or archived transparency tables

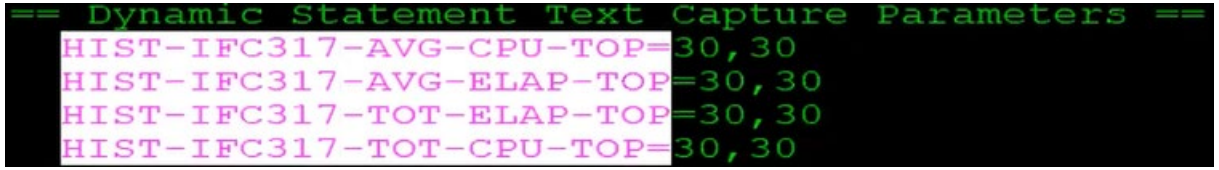
Monitoring dynamic SQL

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 and “live”:
    - 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 in PLAN_TABLE** and other explain tables
  - This needs to be collected additionally / intelligently per STMTID

Monitoring Dynamic Statements / SQLs - prereqs

- How often shall I externalize Dynamic Statement Cache ?
  - Simple answer: before unused statement info gets removed - LRUed
    - can be hours, or days, or minutes?
  - Before any workload switch, eg 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



- 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 doesn’t follow annotation?
      - DSNRLMTxx resource limit tables with RLFEUID / RLFEUWN / RLFIP set to blank applies to apply to all users/all workstations,
      - except those who are specified explicitely (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	– LOCATION only – PRDID only – AUTHID, ROLE, or both – COLLID, PKGNAME, or both – One of CLIENT_APPLNAME, CLIENT_USERID, or CLIENT_WRKSTNNAME The filtering values are not case-sensitive, and profiles can match regardless of the case of the input values.

  - 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
  - EASY 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 20.0.16 D13A CA31 D13APR20 08/08/25 09:35:26

1 DynCache 2 DynCache-Int 3 Static SQL-EDM 4 Static SQL-Int

R/DYNSQLST Dynamic SQL Statements in Cache FOCUS OFF Item 1/4114

SORT 7 D

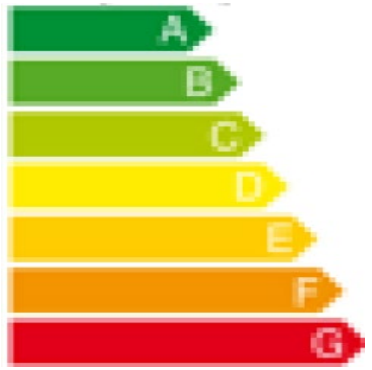
Actions: S=Select for statement detail

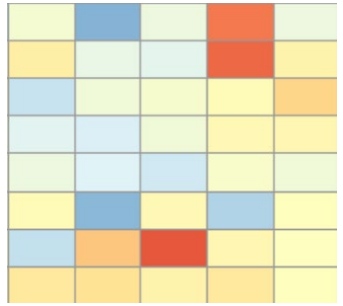
SQL Text (16 bytes)	Cur Usrs	Act Cpys	Exec	Accum Elapsed	Accum CPU	Accum zIIP	Program Name	Stmt Num	Getpgs	Avg/Exec Elapsed	Avg/Exec CPU	Avg/Exec zIIP
SELECT count(*)	0	0	81	13:03.499	21:59.181	21:18.097	THREAD1	71	72555K	9.672	16.286	15.778
SELECT COUNT(*)	0	0	1	41.508	2.038	1.497	SYSLH200	1	2539953	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

BROADCOM

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, A12 IDUG 2025 NA)
    - 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 **each** 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_TABLE
      NOTE: PLAN_TABLE will be populated with info when statement was PREPARED, not from current EXPLAIN

```
---------------- 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, PRIMAUTH, 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 > 100
      ORDER BY **AVG_CPU_PER_SQL DESC**
      FETCH FIRST 30 ROWS ONLY;
  - Selecting top30 zIIP intensive SQLs and normalizing
    - SELECT PROGRAM_NAME, PRIMAUTH, 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 > 100
      ORDER **AVG_ZIIP_PER_SQL DESC**
      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, PRIMAUTH, 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 DESC**
      FETCH FIRST 30 ROWS ONLY;
  - Selecting SQLs doing tablespace scans:
    - SELECT PROGRAM_NAME, PRIMAUTH, 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 DESC**;

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.	
STAT_SYNRB	The number of synchronous buffer reads that are performed for the statement.	
STAT_SORTB	The number of sorts that are performed for the statement	
STAT_PROWB	The number of rows that are processed for the statement	
STAT_INDXB	The number of index scans that are performed for the statement	
STAT_RSCNB 	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	*BEWARE: at cache time only, it is just 1st SQL value*
CLIENT_APPLNAME	The client application name information	*BEWARE: at cache time only, it is just 1st SQL value*
CLIENT_WRKSTNNAME	The client workstation name information	*BEWARE: at cache time only, it is just 1st SQL value*

Apple (SQL) to apple (SQL) compare

- Invention is on your side, you can store it in a time-series tables (timestamp value DSN_STATEMENT_CACHE_TABLE.CACHED_TS) for easy compare what has changed overtime, heatmaps, charts, etc

		STAT_RSCNB > 0	R-SCAN / TABLE SCAN ACCESS PATH					
STMTID	STMT	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 same SQL
  - STMTID is just a sequence number, order of inserted SQLs into cache and **NOT a unique identifier**
  - **WARNING: if based on STMTID,** above table may have misleading info, if SQL was removed and re-inserted into cache, eg after DB2 restart (sequence number – STMTID can/will change)

Apple (SQL) to apple (SQL) compare

- Are all apples same?

```
SELECT * FROM SYSIBM.SYSTABLES;
Select * FROM sysibm.systables;
SELECT *
FROM SYSIBM. SYSTABLES;
SELECT *                          FROM SYSIBM. SYSTABLES          ;
SELECT *                          FROM SYSIBM.SYSTABLES;
```

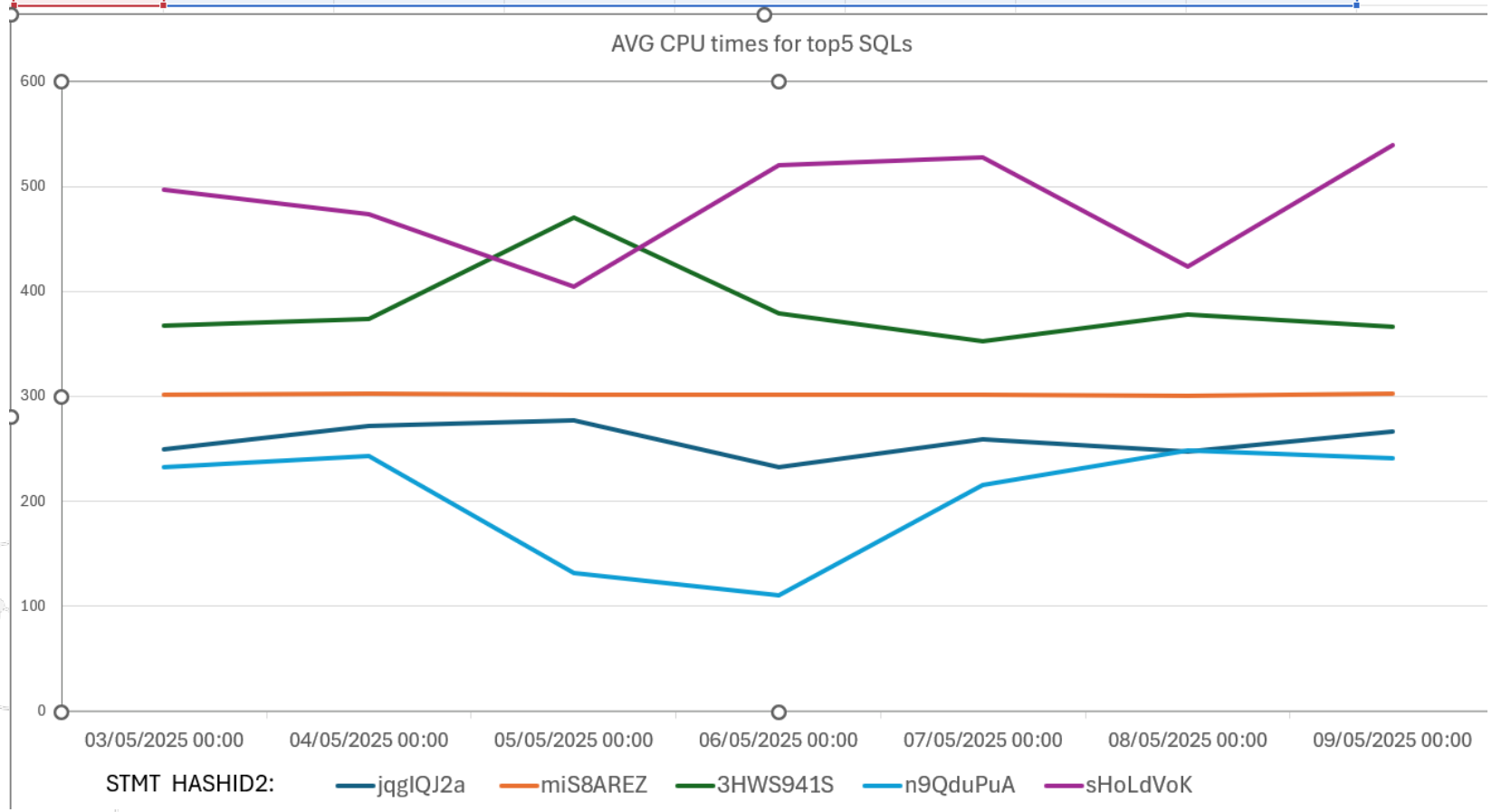
  - They all will cache as separate cache entries, so can make things harder to analyze and compare, and focus on same SQL access path
- DSN_STATEMENT_CACHE_TABLE.QUERY_HASH is hashed SQL identifier (unique)
  - Only for dynamic plan stability, and case/spacing sensitive
- V12 comes to rescue - DSN_STATEMENT_CACHE_TABLE.STMT_HASHID2 is normalized SQL (case and spacing insensitive), 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.

BROADCOM

Apple (SQL) to apple (SQL) compare

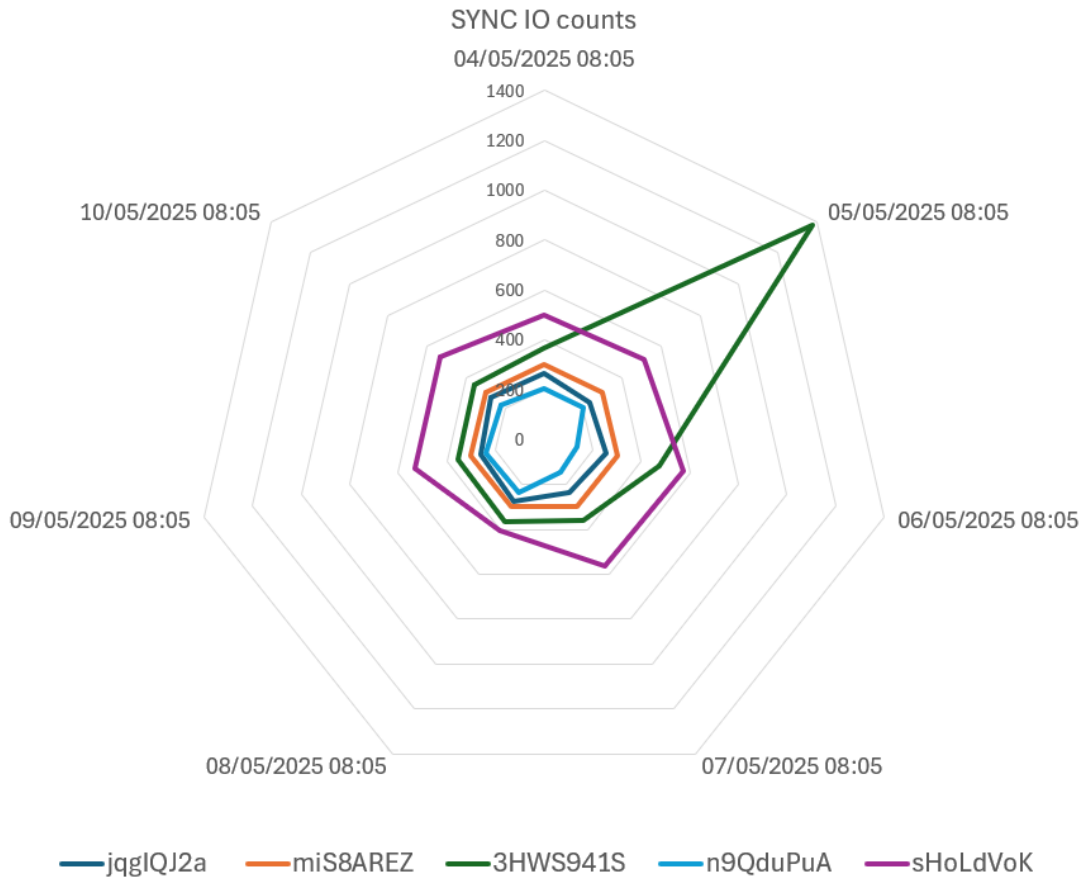
STMT_HASHID2	03/05/2025 22:42	04/05/2025 22:42	05/05/2025 22:42	06/05/2025 22:42	07/05/2025 22:42	08/05/2025 22:42	09/05/2025 22:42
jqglQJ2a	250	272	277	233	259	247	266
miS8AREZ	302	303	301	302	301	300	303
3HWS941S	367	374	470	379	352	378	366
n9QduPuA	233	243	132	110	215	248	241
sHoLdVoK	497	474	404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



BROADCOM

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

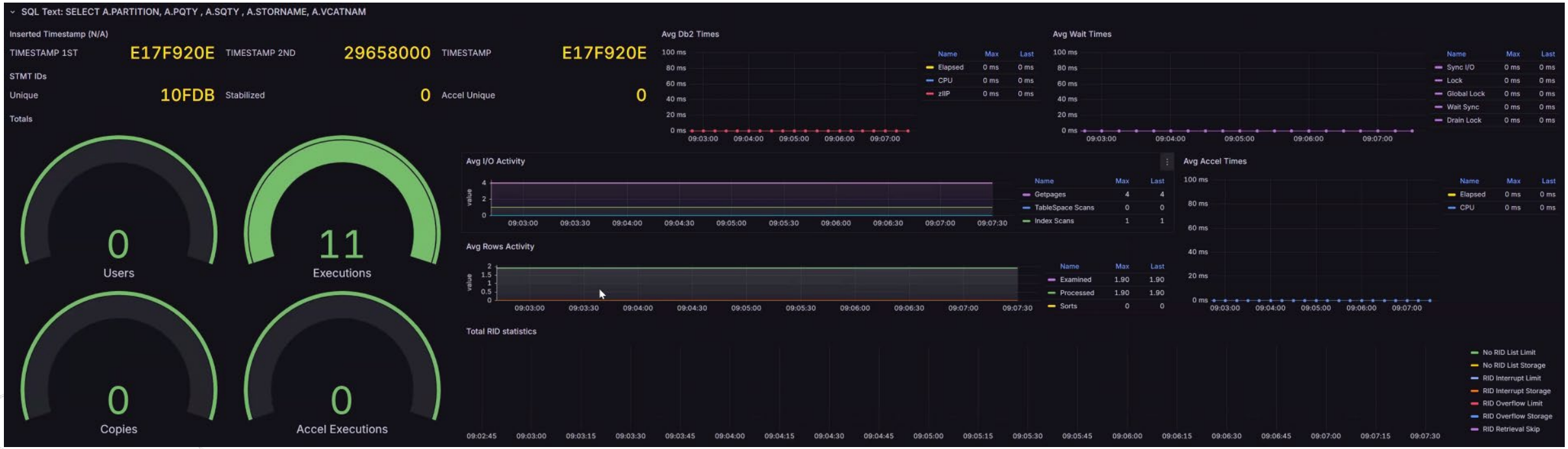


BROADCOM

Apple (SQL) to apple (SQL) compare

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..	jqglQJ2a	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..	hjjaow7e	545	551	458	428	409	485	521	504	406	530	470	594	435	486	534	460	567	441	415	499	544	590	497	488

Focus on particular statement using SYSVIEW for Db2 + Grafana



- Indicators to watch (custom selection):
  - AVG Db2 times changing over time
  - RID access abandoned + Tablespace scans
    - would indicate more data fetched, so do not fit into RID list?
  - Increased sync IO
    - BP too small, or monopolized by others, more data ?
  - change in number of rows being fetched,
    - maybe our access path now shall be different / rePREPARED?

Please VOTE - AHA IDEA for STMT_HASHID2 in IFCID 316

- https://ibm-data-and-ai.ideas.ibm.com/ideas/DB24ZOS-I-1796

Home / Search results / DB24ZOS-I-1796

Add a new idea ✓ Subscribed

5

VOTE

Status	Under review
Workspace	Db2 for z/OS
Created by	Guest
Created on	Aug 8, 2025

Add STMT_HASHID2 in IFCID 316

See this idea on ideas.ibm.com

Can you add STMT_HASHID2 (referred in DSN_STATEMENT_CACHE_TABLE) field to the IFCID 316 record? It would be very handy in identifying normalized SQL statement text(s).

BROADCOM

Meet the Experts and Your Peers in October

- In-person event in Plano, TX: October 15-17
- No registration fee!
- Network with peers and Mainframe technical experts
- Over 120 technical how-to sessions, product update sessions and hands-on workshops

BROADCOM

BROADCOM®
MAINFRAME SOFTWARE

Thank you

Michal.Bialecki@broadcom.com