IDUG VIRTUAL

Summer 2020 NA **Db2** Tech Conference

Stop REBINDing!!!

Speaker: Terry Purcell
Company: IBM

Platform: Db2 for z/OS

#IDUGDb2

Stop REBINDing!!!

- Or……Stop (Dynamically) PREPARE-ing!!!

- And if you're NOT REBINDing……

Start REBINDing!!!

Agenda

- Why (and why not) REBIND? 
- (Static) Plan management
- Dynamic Plan Stability
- (Other) Enhancements to minimize access path change

Why do we suggest REBINDing?

- Improved performance from new run time
  - SPROCs disabled and puffing required when executing prior release packages
- Exposure to new query optimization and runtime enhancements
  - New access path choices (where the biggest % improvements exist)
  - Improved runtime adaptations and optimizations
- Staying current on runtime structures ensures you're matching the testing Db2 development does
  - Reduce exposure to problems with migrated packages from earlier releases
    - INCORROUTs
    - Thread abends

When (and how?) to REBIND

- Across migration
  - REBIND with APREUSE is recommended immediately after migration
    - REBIND avoids need to puff runtime structure (to new release format)
    - APREUSE reduces risk of access path regression
      - Immediately after migration is usually a time for “stability” to validate new release
    - **Ensure you (initially) keep a prior release runtime**

- …..APREUSE “safety” comes at a cost…..

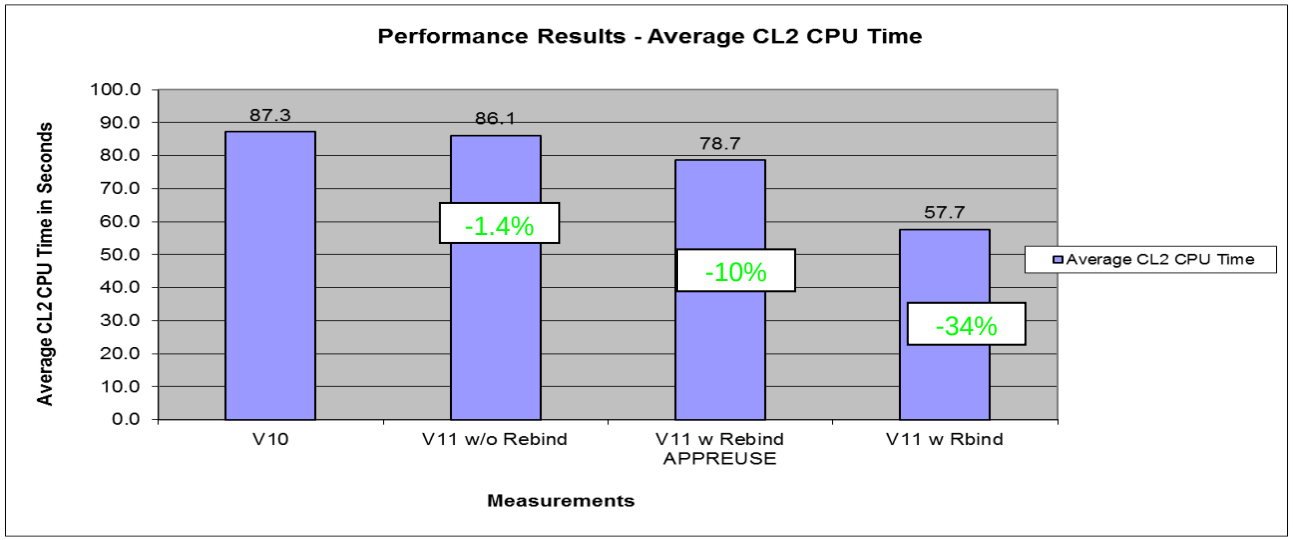
NOTE: APREUSE is a BIND/REBIND option which instructs the optimizer to reuse the prior access path

REBIND with/without APREUSE – V12 message

- Rebind *without* APREUSE to see full potential for CPU savings
  - From Db2 12 - 15+ internal workloads measured
    - Queries with new Access Paths consistently were the biggest winners
  - Typical performance gains for ***new AP's***
    - 10%-99% reduction in CPU and ET
      - New access path and optimized runtime structures
  - Typical performance gains with ***NO AP change***
    - 0%-20% reduction in CPU and ET
      - Optimized runtime structures with existing access path

APREUSE – V11 message

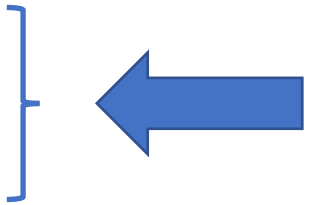
- Db2 11 trade-off of APREUSE
  - An example DB2 11 workload measurement
    - 34% with APREUSE(NONE)
      - New access paths and optimized runtime structures
    - 10% if APREUSE was used
      - Same access path but optimized runtime structures
    - 1.4% if no rebind
      - Prior release runtime structures on new release



NOTE: One workload comparison only – to demonstrate the differences (often as best case)

REBIND – Within a release

- Minimum recommendation
  - One BIND/REBIND within a release (current release runtime structure and prep for next release)
- Across FLs
  - No requirement for REBIND for access path changes
- After maintenance application
  - REBIND (with APREUSE) may be recommended due to Db2 fixes
- After REORG/RUNSTATS
  - REBIND (without APREUSE) not required unless current performance is unacceptable
  - REBIND recommendation is based upon tolerance/ability to address any regressions
    - Fallback via REBIND SWITCH
    - Improving inputs to optimizer with targeted RUNSTATS
    - Other tuning approaches



What is your willingness/ability?

Agenda

- Why (and why not) REBIND?
- (Static) Plan management 
- Dynamic Plan Stability
- (Other) Enhancements to minimize access path change

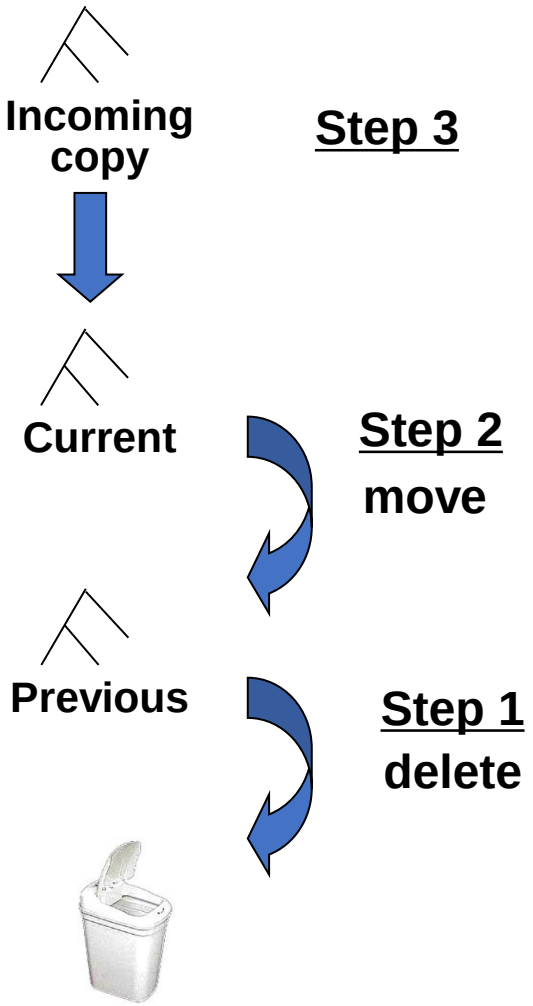
Plan Management (also known as Access Path Stability)

- What are the problems DB2 is attempting to solve?
  - For Static REBINDs at migration or after applying maintenance
    - Recover from an access path regression (REBIND SWITCH)
      - Beginning in DB2 9
    - Allow REBINDs to attempt to preserve the prior access path (APREUSE)
      - Beginning in DB2 10
  - For Static BINDs after application changes
    - Allow BINDs to attempt to preserve the prior access path
      - Beginning in DB2 10

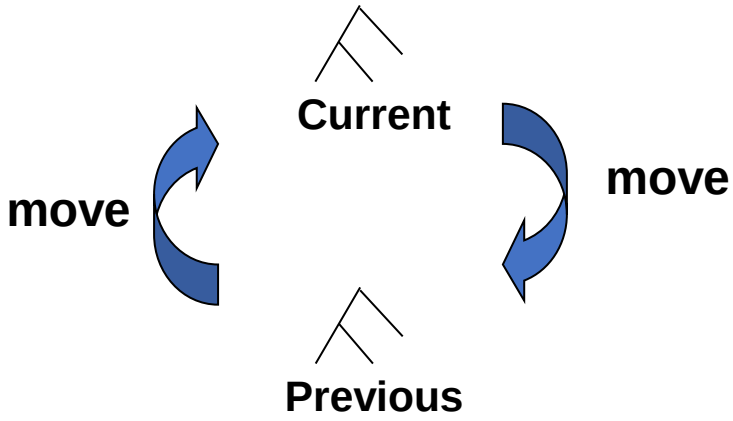
PLANMGMT = BASIC

- BASIC is an available option
  - But not recommended due to ease of disposing of a valid prior copy

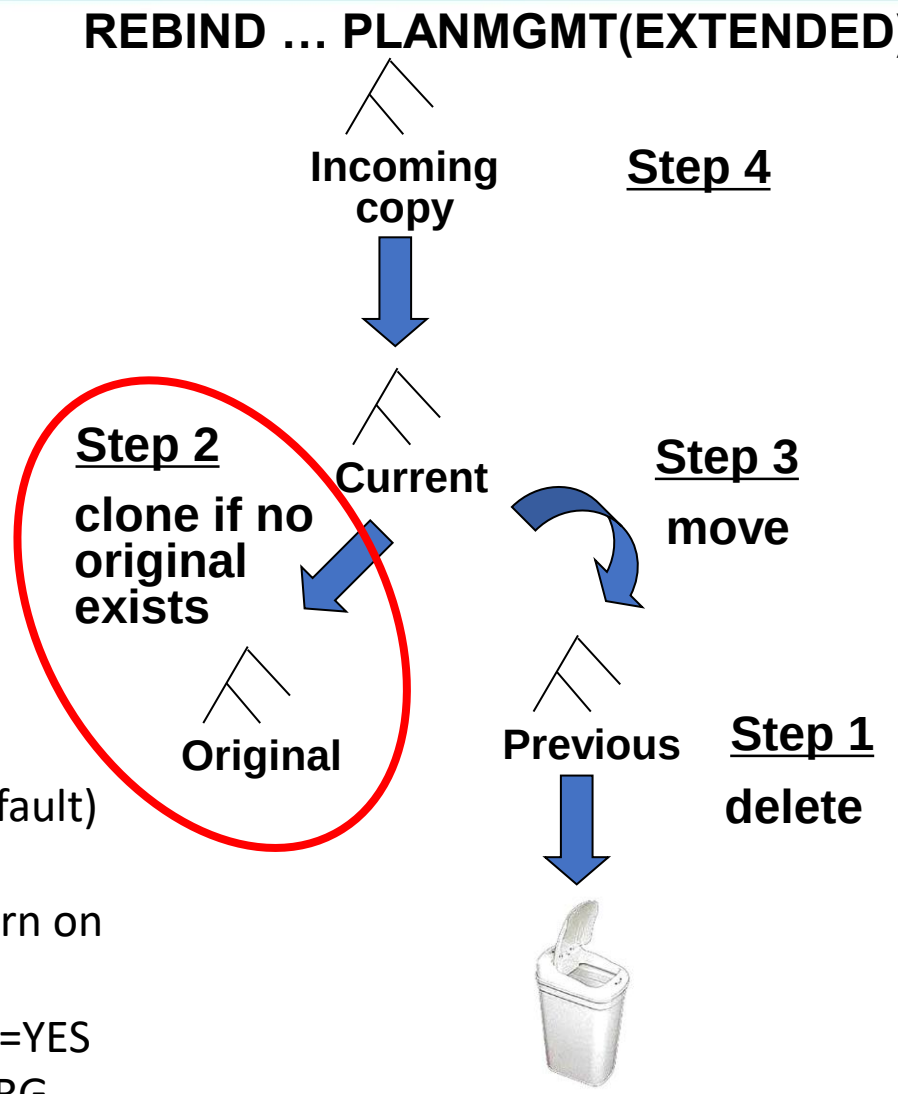
REBIND … PLANMGMT(BASIC)

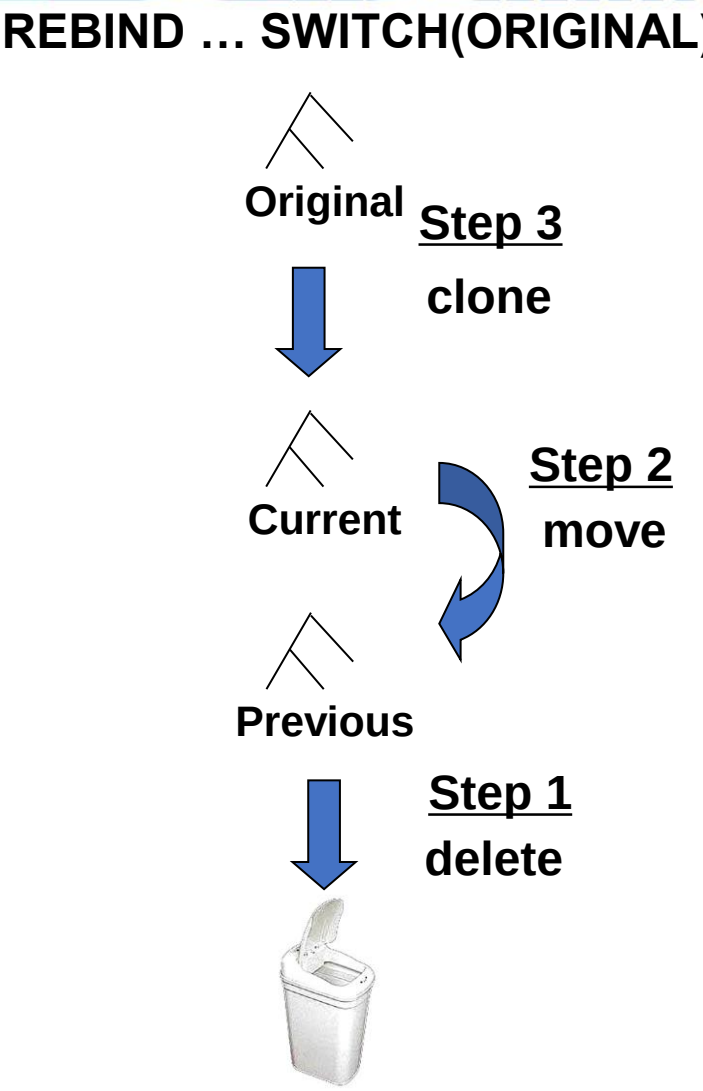


REBIND … SWITCH(PREVIOUS)



PLANMGMT = EXTENDED (Default & Recommended)





- Set ZPARM PLANMGMT=EXTENDED (default)
  - Keep that extra copy!
  - If space is a concern, turn on compression for SPT01
    - COMPRESS_SPT01=YES followed by a REORG

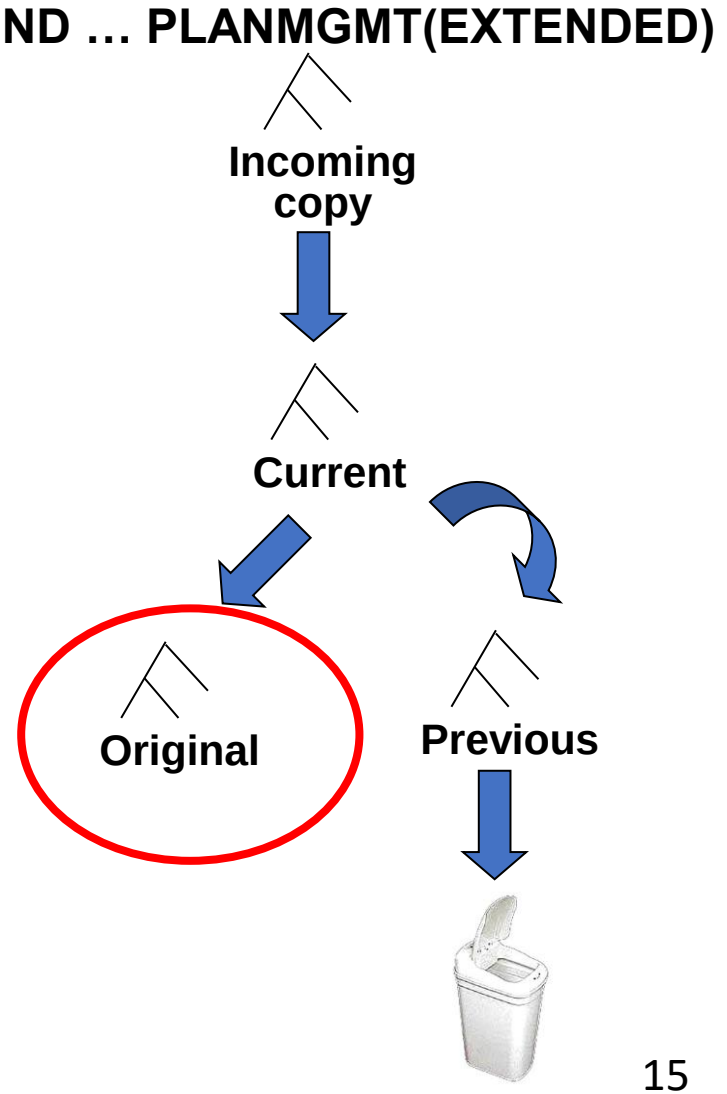
REBIND – Cleanup – Action Required!

- With PLANMGMT(EXTENDED)
  - There is NO automated cleanup
  - ORIGINAL copy may be OLD/STALE
    - Db2 does NOT automatically replace the ORIGINAL. Only the PREVIOUS is replaced at each REBIND.
  - Recommended to regularly cleanup ORIGINAL
  - An “empty” ORIGINAL ensures that next REBIND will create a “new” ORIGINAL
    - ALWAYS keep an up-to-date original

PLANMGMT (EXTENDED) Keeping a valid ORIGINAL

- Across migration (V11 to 12)
  - What if you need to revert to the V11 runtime behavior?
    - Is your “ORIGINAL” valid from V11?
    - Is your ”PREVIOUS” from V11 (and valid)?
      - Depending on number of times REBIND has occurred – may be V12
  - Solution
    - FREE old copies before migration (or before REBIND in new release)

- Within a release
  - After n months of stability – may not need the prior release runtime
  - But regularly FREEing old means next REBIND updates the ORIGINAL



REBIND – APREUSE ERROR vs WARN

- APREUSE(ERROR) is recommended when
  - No tolerance for access path changes
    - NOTE: This doesn't mean NO change – runtime structure behavior can change for existing access path

- APREUSE(WARN) is recommended because
  - ERROR is "all or nothing" – at package level
    - Inability of one SQL to reuse a prior access path will fail the whole package
  - WARN is "per SQL"
    - Inability of an SQL to reuse the prior access path only affects that SQL
  - ERROR must result in an "exact match" to succeed
    - But not all optimizer choices are within APREUSE's control
      - An increase in MATCHCOLS for same index will fail APREUSE(ERROR)
  - WARN can tolerate "some" small differences in access path
    - Increase in MATCHCOLS for same index
    - or multi-index with different order of the "same" indexes are acceptable

- NOTE: APREUSE may fail, but a new access path "could" still be the same as the old

Plan Management – V12 updates

- Free inactive package copies while package is allocated and in use
- Allow selective FREE of ORIGINAL or PREVIOUS or inactive ONLY
- APREUSE source
  - Supports APREUSE of PREVIOUS or ORIGINAL in 1 step
  - Disallows SWITCH to an INVALID copy
- AUTOBIND to use APREUSE(WARN)
  - APAR PH15896 – only requires FL100

Agenda

- Why (and why not) REBIND?
- (Static) Plan management
- Dynamic Plan Stability 
- (Other) Enhancements to minimize access path change

Dynamic SQL challenges

- Dynamic SQL are more susceptible to performance regressions
  - Full prepare at every cache miss
  - Full PREPARE at each member in a data sharing group
  - May undergo optimization process several times a day
    - Static SQL optimized at [RE]BIND time, reused until subsequent [RE]BIND
  - Dynamic SQL exposed to changing optimization inputs
    - Release migration
    - System maintenance
    - Database maintenance (RUNSTATS)
    - System parameter changes
  - Static SQL regression is only exposed at [RE]BIND
    - At [RE]BIND, static regression risk can be mitigated with PLANMGMT and APREUSE

Dynamic plan stability (DPS) - how does it work?

- Provide “static SQL like” performance stability for repeating cached dynamic SQL
  - Persist cached dynamic SQL structures into DB2 catalog
  - Cache-miss -> locate and load cache from catalog structures

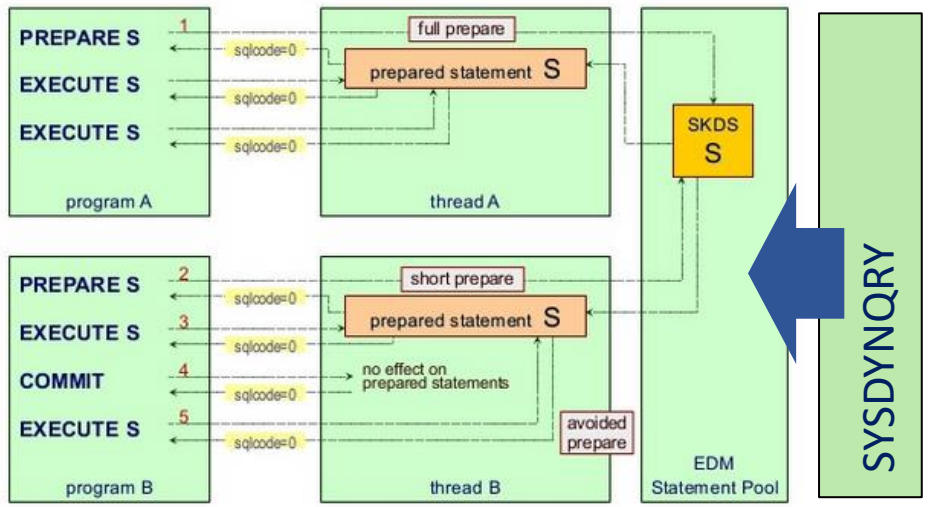
- Allow more stable, predictable performance across
  - Exit/re-entry to statement cache
  - System recycle
  - Database maintenance (RUNSTATS)
  - DB2 maintenance
  - Function level activation
  - Across members of data sharing group

- Potential significant CPU savings at “cache turnover” periods, due to reduced prepares

Dynamic plan stability (DPS) - how does it work?

Stabilize cache structures to DB2 catalog (SYSDYNQRY) and load from there

- Executing a dynamic statement, no DPS
  - Check cache
  - Cache hit -> run cached copy
  - Cache miss > full prepare
- Executing a dynamic statement with DPS:
  - Check cache
  - Cache hit -> run cached copy
  - Cache miss -> SYSDYNQRY hit -> run SYSDYNQRY copy
  - Cache miss > SYSDYNQRY miss > full prepare



Stabilize queries – Capture command

- Dynamic query capture command syntax

```
>>-START DYNQUERYCAPTURE STBLGRP(stabilization-group) ---------------------------->
>--+-THRESHOLD(integer-constant)-|-cache-snap-specification-|--------------><
   |-STMTID--(integer-constant)-----------------------------------------+
   |-STMTTKN-(string-constant)-------------------------------------------'

Cache snap specification
>>--+-----------------+------+-----------------+----+--------------------+--><
    |          .-*-----.   |  |         .-NO--.  |    |         .-LOCAL-.    |
    '-CURSQLID(+-------+)-'  '-MONITOR(-------)-'    '-SCOPE(-+-------+-)-'
              +-SQLID-+               +-YES-+                 +-GROUP-+
```

Capture examples

1. Capture individual statement

   -START DYNQUERYCAPTURE STBLGRP(APP1) STMTID(1253)

   - Stabilize STMTID 1253 into catalog with stabilization group APP1

2. Snapshot capture

   -START DYNQUERYCAPTURE STBLGRP(APP1) THRESHOLD(100) CURSQLID(APP1ID) MONITOR(NO)

   - Capture statements with APP1ID CURSQLID, and number of executions >= 100

3. Capture with monitoring*

   -STARTDYNQUERYCAPTURE STBLGRP(APP1) THRESHOLD(100) CURSQLID(APP1ID) MONITOR(YES)

   - Same as #2, but continue to scan cache and apply threshold until monitor stopped.
   - Performance impact of cache re-scan is negligible.

Remove stabilized queries from catalog

- Command to FREE STABILIZED DYNAMIC QUERY – reasons to FREE
  - A query stabilized with an inefficient access path
  - Query is no longer used

```
SELECT * FROM SYSIBM.SYSDYNQRY
WHERE LASTUSED < CURRENT DATE - 180 DAYS
```

  - Desire for DB2 to consider a new access path
    - No REBIND capability today
  - Clear invalid queries
    - NOTE: Invalidation similar to static SQL invalidation rules

Agenda

- Why (and why not) REBIND?
- (Static) Plan management
- Dynamic Plan Stability
- (Other) Enhancements to minimize access path change

Literal Replacement (or Statement Concentration)

- From V10 - Dynamic SQL with literals can now be re-used in the cache
  - Literals replaced with &
    - Similar to parameter markers but not the same
- Example:
  - WHERE ACCOUNT_NUMBER = 123456
    - This would be replaced by
  - WHERE ACCOUNT_NUMBER = &

- Lookup Sequence
  - Original SQL with literals is looked up in the cache
  - If not found, literals are replaced and new SQL is looked up in the cache
    - Can only match with SQL stored with same attribute, not parameter marker
  - If not found, new SQL is prepared and stored in the cache

Statement Concentration

- Performance Expectation
  - Significant opportunity to avoid PREPAREs for repeating SQL with literals
  - Using parameter marker still provides best performance
  - NOTE: Access path is not optimized for literals
    - Neither are access paths based on parameter markers/host variables
    - Need to use REOPT for that purpose
- Limitations
  - LIKE predicates not supported
  - Literal replacement queries cannot exploit opthints and cannot use EXPLAIN
  - May disable ability to use Index On Expression (IOE) – SUBSTR(COL,4,3) → SUBSTR(COL,&,&)
  - Bindtime pruning may be limited – WHERE 0=1 → WHERE & = &

Statement Concentration – Db2 12 Update

- Initially controlled by
  - Application programmer – Option of PREPARE ATTRIBUTES clause (application SQL call)
  - Client connection (ODBC/JDBC config options)
- Db2 12 adds [RE]BIND PACKAGE option (DBA control)

```
>>-[RE]BIND PACKAGE-------------------------------------------->
>>-+--------------------------+-------------------------------->
   |                   .-NO--.    |
   '-CONCENTRATESTMT(-+-----+-)-'
                     '-YES-'
```

- CONCENTRATESTMT:
  - NO (default)
    - Do not enable statement concentration. Literals in dynamic SQL are not replaced.
  - YES
    - Enable statement concentration. Literals in dynamic SQL are replaced by '&'.

RUNSTATS – Dynamic SQL invalidation Db2 12

- Db2 12 default is that RUNSTATS will NOT invalidate cache
  - Behavior change compared to V11 – always invalidate cache
  - Exceptions
    - UPDATE NONE REPORT NO
    - RESET ACCESS PATH
  - RUNSTATS option to INVALIDATECACHE
    - Default NO
  - Goal/Reason
    - Majority of RUNSTATS executions are simply to keep catalog data current
      - And NOT to change access paths that have acceptable performance
        - Which is the whole theme of this presentation

REBIND PHASE-IN – Db2 12 FL505

- New REBIND behavior
  - To allow BIND/REBIND to complete without requiring quiesce (exclusive access) of package
    - Will create a new CURRENT (phase-in) and existing will be the “phase-out” copy
  - Support for the following options:
    - APREUSE(NONE/WARN/ERROR)
    - & PLANMGMT(EXTENDED)
    - & APREUSESOURCE(CURRENT)
    - the package type is not for a trigger or SQL stored procedure or scalar SQL UDF

Summary

Stop REBINDing!

- Or more correctly – stop introducing new access paths
  - Unless you are able to address any regressions

- Db2 provides many features to reduce the introduction of new access paths
  - Providing stability and reliability in performance

  - REBIND
    - APREUSE - To gain the benefit of new runtime structures – with reduced risk of regression
    - SWITCH - To revert to prior “good” performance
      - REMINDER – Have a recent ORIGINAL!
    - Phase-in (FL505) – new behavior to ensure REBINDs are less disruptive
  - Dynamic Plan Stability
  - Statement concentration
  - RUNSTATS to avoid invalidation by default

IDUG VIRTUAL

Summer 2020 NA **Db2** Tech Conference

Terry Purcell
IBM
tpurcel@us.ibm.com

#IDUGDb2