#SHAREorg

SHARE
Technology · Connections · Results

DB2 for z/OS Migration: Query Performance Considerations

Tom Beavin
IBM Silicon Valley Lab
February 7, 2013
Session 12740

Email: beavin@us.ibm.com

SHARE in San Francisco 2013

Agenda

- Introduction
- Query performance preparation
- Explain enhancements
- Exploiting plan management
- RUNSTATS/REBIND migration recap
- Plan management what next?

Introduction

Rebind Recommended at each new release

- Reasons for DB2 10 REBIND recommendation
  - Improved performance from new run time
    - SPROCs disabled and puffing required when executing prior release packages
    - Maximize DBM1 31-bit VSCR
  - Exposure to new query optimization and runtime enhancements
    - New access path choices
    - Allow RID overflow to workfile
  - Reduce exposure to problems with migrated packages from earlier releases
    - INCORROUTs
    - Thread abends
  - Preparation for further usage of plan management in DB2 10 and beyond
  - REBIND suggestion (obviously) applies to static SQL only
    - Dynamic SQL is exposed immediately

DB2 10 Query Performance – No REBIND required

- No REBIND required for
  - Index list prefetch
  - INSERT index read I/O parallelism
  - Workfile spanned records
  - High performance DBATs
  - Inline LOBs **(New function – requires NFM)**

- **Also – No REBIND required for move from CM to NFM**
  - All access path related enhancements are available in CM

DB2 10 Query Performance – REBIND Required

- REBIND required to take advantage of
  - Use of RELEASE(DEALLOCATE)
  - Early evaluation of residual (stage 2) predicates
  - IN-list improvements (new access method)
  - SQL pagination (new access method)
  - Query parallelism improvements
  - Index include columns (New function – requires NFM)
  - More aggressive view/table expression merge
  - Predicate evaluation enhancements
  - RID list overflow improvements
  - SQLPL performance

Query performance preparation

Migration REBIND Preparation - RUNSTATS

- RUNSTATS changes in V9/10
  - New CLUSTERRATIO & DATAREPEATFACTORF in V9
    - Suggested RUNSTATS before starting REBIND program after migration from V8
  - KEYCARD becomes RUNSTATS default in V10
    - Suggested RUNSTATS before starting REBIND program after migration from V9
      - *If KEYCARD not used prior to V10*
      - *Alternative is to begin using RUNSTATS KEYCARD prior to DB2 10 migration*
        - *So that RUNSTATS not required before REBIND on V10 coming from V9*

Pre-production Proactive Access Path Analysis

- Customers often copy production stats to non-prod
  - In addition to catalog statistics (and zparms), optimizer considers
    - *CPU speed*
    - *# of CPs (for parallelism)*
    - *BP size*
    - *RID pool*
    - *Sort pool*
- Must match statistics and system configuration

- NOTE: Copying stats from V8 production to V9/10 pre-production will not reflect V9/10 production
  - Due to missing (new formula) CR & DRF statistics

Production Modeling

- V9 APAR PM26475 & V10 APAR PM26973
  - Supports optimizer overrides for system settings
  - New zparms
    - *SIMULATED_CPU_SPEED*
    - *SIMULATED_CPU_COUNT*
  - New SYSIBM.DSN_PROFILE_ATTRIBUTES
    - *SORT_POOL_SIZE*
    - *MAX_RIDBLOCKS*
    - *For bufferpools*
      - *Same as the BP names listed in the DSNTIP1 panel*
      - *For example a KEYWORDS value of 'BP8K0' corresponds to BP BP8K0*

Production Modeling – How to obtain values?

- How do I obtain existing production values?
  - BP information available from –DISPLAY BUFFERPOOL command
  - Issue an explain of a dummy statement, and query PLAN_TABLE
    - *Output needs to be converted to INTEGER*

```
EXPLAIN ALL SET QUERYNO=6475 FOR
SELECT * FROM SYSIBM.SYSDUMMY1;

SELECT HEX(SUBSTR(IBM_SERVICE_DATA,17,2)) AS CPU_COUNT,
       HEX(SUBSTR(IBM_SERVICE_DATA,69,4)) AS CPU_SPEED,
       HEX(SUBSTR(IBM_SERVICE_DATA,13,4)) AS MAX_RIDBLOCKS,
       HEX(SUBSTR(IBM_SERVICE_DATA,9,4))  AS SORT_POOL_SIZE
FROM PLAN_TABLE WHERE QUERYNO=6475
```

***NOTE 1: CPU count is only populated if query chooses parallelism**
****NOTE2: Search "Modeling a production environment in a DB2 test subsystem" for more comprehensive SQL**

Production Modeling – How to create a profile?

- How do I create a production profile on my test system?
  - DDL for SYSIBM profile tables is in sample job DSNTIJOS
  - INSERT 1 row into profile table using any unique number

```
INSERT INTO SYSIBM.DSN_PROFILE_TABLE(PROFILEID) VALUES(4713);
```

  - INSERT 1 row for each override

```
INSERT INTO SYSIBM.DSN_PROFILE_ATTRIBUTES
(PROFILEID,KEYWORDS,ATTRIBUTE1,ATTRIBUTE2)
VALUES (4713, 'BP8K0',NULL, 2500);
```

  - Update the CPU zparms
  - Finally, issue -START PROFILE command
    - *NOTE: CPU zparms are active without profile*

Production Modeling – How to validate?

- How do I validate that the profile was used?
  - Execute EXPLAIN
  - In DSN_STATEMNT_TABLE
    - *REASON will contain the value 'PROFILEID nnnn' appended to the existing REASON value for that statement*
  - Original SQL used on production system can also be used on test system to validate output from IBM_SERVICE_DATA

EXPLAIN enhancements

DB2 10 EXPLAIN tables

- Format and CCSID from previous releases is deprecated in V10
  - Cannot use pre V8 format
    - SQLCODE -20008
  - V8 or V9 format
    - Warning SQLCODE +20520 regardless of CCSID EBCDIC or UNICODE
  - Must not use CCSID EBCDIC with V10 format
    - EXPLAIN fails with RC=8 DSNT408I SQLCODE = -878
    - BIND with EXPLAIN fails with RC=8 DSNX200I
- Recommendations
  - Use CCSID UNICODE in all supported releases (V8, V9, V10) due to problems with character truncation and conversion etc
  - Use the V10 extended column format with CCSID UNICODE when
    - Applications access EXPLAIN tables and can only tolerate SQLCODE 0 or +100
  - V10 column format is supported under V8 and V9 with the SPE fallback APAR PK85956 applied with the exception of
    - DSN_STATEMENT_CACHE_TABLE due to the BIGINT columns
  - APAR PK85068 can help migrate V8 or V9 format to the new V10 format with CCSID UNICODE

DB2 10 Retrieving Existing Access Path

- EXPLAIN PACKAGE command
  - Extract PLAN_TABLE information for packages from V9/10
  - Useful if you did not BIND with EXPLAIN(YES)
    - Or PLAN_TABLE entries are lost

```
>>-EXPLAIN----PACKAGE----------->

>>-----COLLECTION--collection-name--PACKAGE--package-name--------->

>----+---------------------------+----+-------------------+-------->
     |                           |    |                   |
     +---VERSION-version-name---+    +---COPY--copy-id---+
```

- COPY-ID can be ‘CURRENT’, ‘PREVIOUS’, ‘ORIGINAL’

DB2 10 What if ? for BIND/REBIND

- BIND/REBIND package EXPLAIN(ONLY) & SQLERROR(CHECK)
  - Existing package copies are not overwritten (new package NOT created)
    - Performs explain or syntax/semantic error checks on SQL
  - Allows you to ask the question “What would the new access path be if I did a BIND/REBIND today?”
    - Without actually creating/overwriting the package
  - Requires BIND, BINDAGENT, or EXPLAIN privilege
  - Externalized in PLAN_TABLE.BIND_EXPLAIN_ONLY=‘Y’
  - NOTE: BIND/REBIND EXPLAIN(ONLY) requires same locking/concurrency requirements as traditional BIND/REBIND

What is the difference of each EXPLAIN usage?

- New DB2 10 options in red
  - BIND/REBIND with EXPLAIN(YES)
    - Generates a new access path, populates PLAN_TABLE and creates new package
  - BIND/REBIND with EXPLAIN(ONLY)
    - Generates a new access path, populates PLAN_TABLE, but does NOT create a new package
  - EXPLAIN PLAN (issued in SPUFI/QMF/DSNTEP2 etc)
    - Generates a new access path and populates PLAN_TABLE
  - EXPLAIN PACKAGE
    - Does not generate new access path. Extracts existing access path from package and populates PLAN_TABLE.
  - EXPLAIN STMTCACHE STMTID/STMTOKEN
    - Does not generate new access path. Extracts existing and populates PLAN_TABLE.

Exploiting plan management

Plan Management (aka Access Path Stability)

- Plan management provides protection from access path (performance) regression across REBIND/BIND
  - Access path fallback to prior (good) access path after REBIND
    - DB2 9 PLANMGMT(EXTENDED/BASIC) with SWITCH capability
  - DB2 10 (APAR PM25679 – July 2011)
    - Freeze access path across BIND/REBIND
      - ***BIND/REBIND PACKAGE … APREUSE(ERROR)***
    - Access path comparison with BIND/REBIND
      - ***BIND/REBIND PACKAGE… APCOMPARE(WARN | ERROR)***

DB2 9 Plan Management (Access Path Stability)

- At REBIND PACKAGE save old copies of package in
  - Directory (SPT01)
  - Catalog tables
- REBIND PACKAGE can be controlled in two ways
  - ZPARM (PLANMGMT)
    - To change system-wide behavior
  - New REBIND option (also called PLANMGMT)
    - To affect packages selectively
- Three flavors of PLANMGMT
  - OFF (V9 default), BASIC, EXTENDED (V10 default)
  - Determines the # of old package copies saved

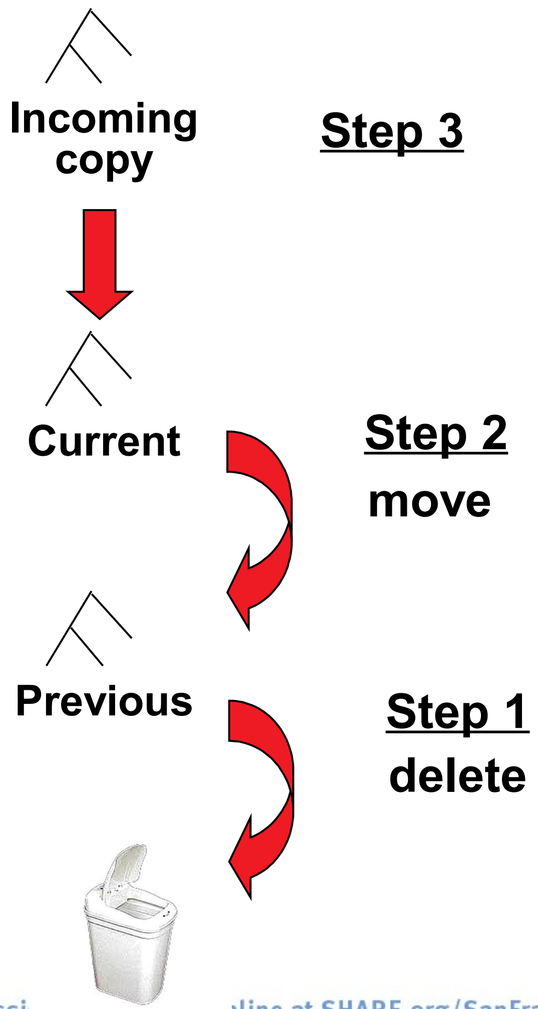
Access Path Stability in DB2 9 …

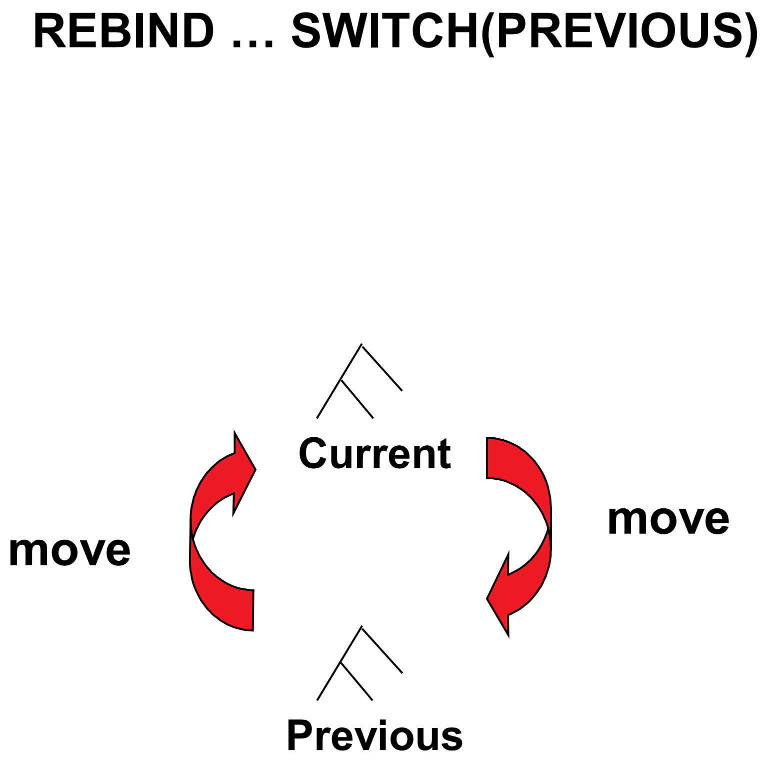
- REBIND PACKAGE … SWITCH
  - SWITCH(PREVIOUS) and SWITCH(ORIGINAL)
  - SWITCH also restores the SYSPACKAGE row
- FREE PACKAGE …
  - PLANMGMTSCOPE(ALL) – Free package completely
  - PLANMGMTSCOPE(PLANMGMTINACTIVE) – Free old copies only
- What's in the catalog tables?
  - SYSPACKAGE reflects current copy only
    - DB2 10 SYSPACKCOPY stores SYSPACKAGE info for previous/original
  - SYSPACKDEP reflects dependencies of all copies
  - Other catalogs (SYSPKSYSTEM, …) reflect metadata for all copies
- Invalidation and Auto Bind
  - Each copy invalidated separately
  - Auto bind only replaces 'current'
- Miscellaneous
  - No V9 support for native SQL stored proc packages (V10 supports native SQL SP's).
  - No support for DBRMs bound directly into PLANs

PLANMGMT = BASIC

- Retains up to 2 copies of a package
  - Current and Previous
- At each REBIND:
  - Any previous copy is discarded
  - Current copy becomes previous copy
  - Incoming copy is the current copy
  - Caveat: ‘Previous’ is discarded, so REBINDing twice with PLANMGMT(BASIC) will cause previous copy to be lost forever
- REBIND … SWITCH(PREVIOUS)
  - Switches between current and previous copy
  - Provides a means of falling back to last used copy
  - SWITCH also restores the SYSPACKAGE record

PLANMGMT = BASIC



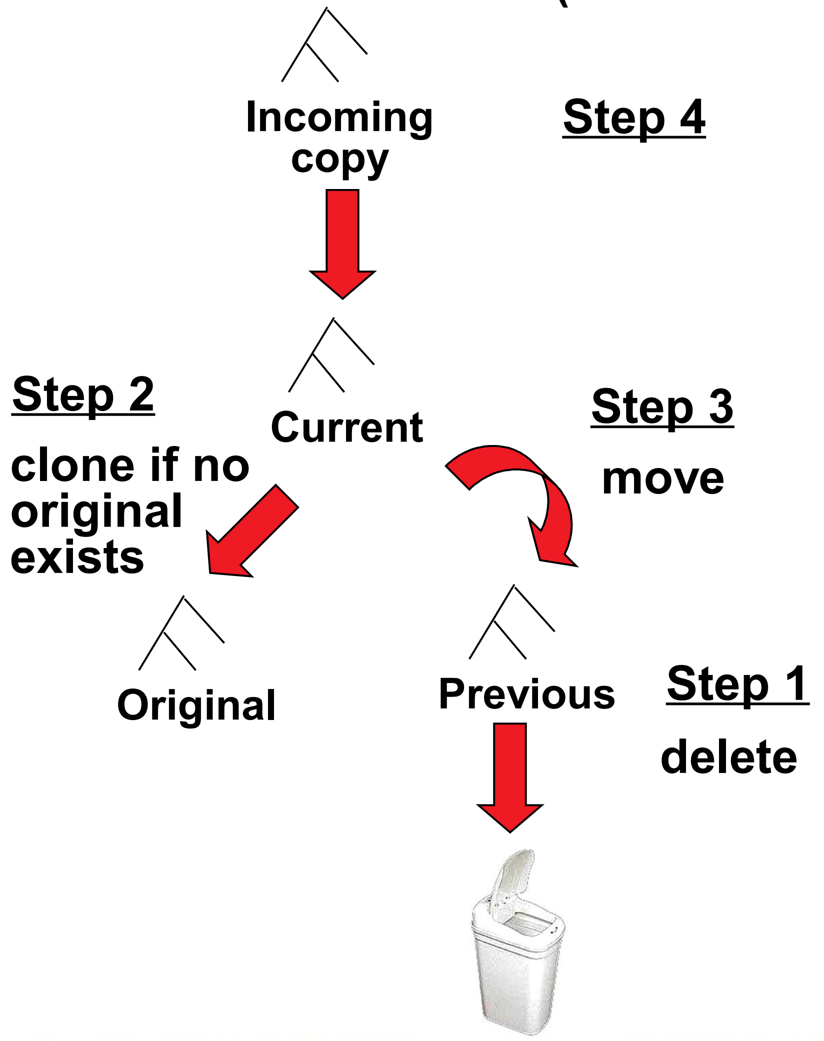


PLANMGMT = EXTENDED (V10 default)

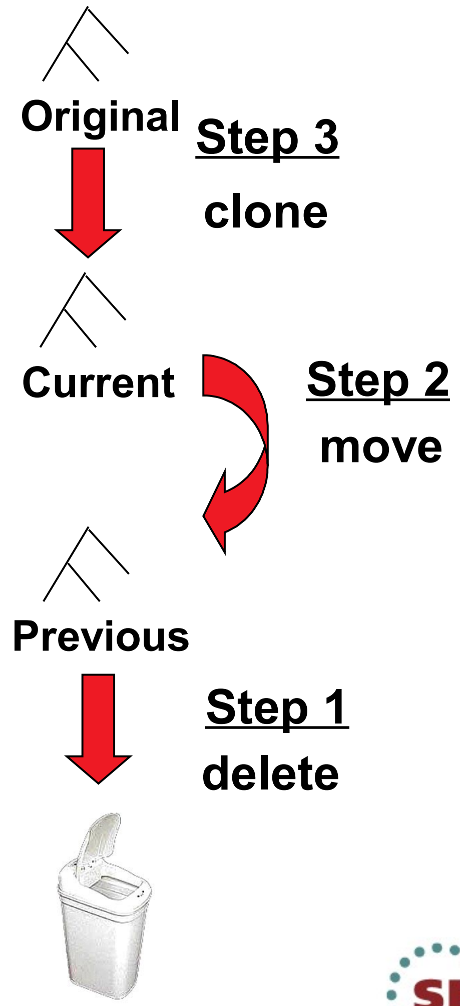
- Retains up to 3 copies of a package
  - Current, Previous and Original
- Original copy is the one that existed from the "beginning"
  - Saved once, never overwritten
    - Unless FREEd, which means new Original will be saved at next REBIND
- At each REBIND:
  - Any previous copy is discarded
  - If there's no original copy, the current copy is saved as original copy
  - Current copy becomes previous copy
  - Incoming copy is the current copy
- REBIND … SWITCH(PREVIOUS)
  - Switches between current and previous copy
- REBIND … SWITCH(ORIGINAL)
  - Current copy moves to previous, and original copy becomes current
  - Provides a mean of falling back to the oldest saved copy

PLANMGMT = EXTENDED

REBIND … PLANMGMT(EXTENDED)



REBIND … SWITCH(ORIGINAL)



Access Path Stability - DB2 9 to 10 Migration

- There is NO capability to FREE only an ORIGINAL copy
  - FREE PACKAGE PLANMGMTSCOPE(PLANMGMTINACTIVE)
    - FREEs both ORIGINAL and PREVIOUS
- But the ORIGINAL can become stale
  - The idea is to keep a “good and stable” backup in case of emergency
  - But it needs to be a recent good/stable backup
- Once you are comfortable with the CURRENT copy
  - Consider FREE PACKAGE PLANMGMTSCOPE(PLANMGMTINACTIVE)
    - Before 1st REBIND in V10
      - *So that 1st REBIND in V10 will save the V9 CURRENT copy to be the ORIGINAL*
- NOTE
  - This recommendation only applies to customers who used PLANMGMT=EXTENDED in V9

Access Path Management – Saving Space

- PLANMGMT(EXTENDED) results in 2 additional package copies

- DB2 10 space saving options for access path management include:

  - APRETAINDUP option of REBIND
    - Default YES
      - *Retain duplicate for BASIC or EXTENDED*
    - Optional NO
      - *Do not retain duplicate access path as PREVIOUS or ORIGINAL*
        - *PREVIOUS/ORIGINAL must be from DB2 9 or later*

  - Inline LOB and compression for SPT01
    - Enabled at V10 ENFM
    - ZPARM COMPRESS_SPT01=YES

Access Path Comparison (APCOMPARE)

- APCOMPARE
  - "Tell me if static SQL statements had changes in access paths"
  - Optionally, stop the BIND/REBIND if there were changes
- New option on
  - BIND PACKAGE
  - REBIND PACKAGE, REBIND TRIGGER and REBIND native SQL proc
- For all statements in the package ...
  - Load old access path from the "EDB"
  - Optimizer generates new access path, as usual
  - Compare old access path with new access path
  - Report results via messages / PLAN_TABLE output
  - Determine REBIND success vs failure

APCOMPARE option values

- APCOMPARE(NONE/NO)
  - No comparison performed
  - This is the default
- APCOMPARE(WARN)
  - RC = 4
  - DB2 will continue processing the package
- APCOMPARE(ERROR)
  - RC = 8
  - DB2 will terminate the processing of the package

Access Path Reuse (APREUSE)

- APREUSE - "Do the BIND/REBIND but <u>try to avoid</u> access path changes"
  - At migration from DB2 9 to 10
  - After service fixes that require the regeneration of runtime structures (++HOLDs directives)
  - To bring invalid packages back to life
  - Due to application changes (BIND PACKAGE)
- New option on
  - BIND PACKAGE
  - REBIND PACKAGE, REBIND TRIGGER PACKAGE, REBIND for native SQL procedures
- For all statements in the package ...
  - Load old access path ("EDB")
  - Feed "EDB" as hint to the optimizer
  - As a final check, compare old access path with new
  - Report results via messages / PLAN_TABLE output
  - Determine REBIND success vs failure
- APREUSE implicitly turns on APCOMPARE

APREUSE option values

- APREUSE(NONE/NO)
  - No reuse performed
  - This is the default

- APREUSE(ERROR)
  - RC = 8
  - DB2 will terminate the processing of the package

APREUSE/APCOMPARE – Things to know

- Requires last BIND/REBIND from DB2 9 or later
  - Starting with DB2 9, EXPLAIN information saved with the package in SPT01
    - NO external PLAN_TABLE records required for APREUSE/APCOMPARE
  - Referred to as "EDB" or "Explain Data Block"
    - EDB is a compact representation of PLAN_TABLE
- Apply to BIND also
  - Attempt to match on SQL text to determine prior query
  - See Appendix for discussion on BIND (as not related to migration)
- IMPORTANT NOTE:
  - Unsuccessful attempt to perform compare/reuse is NOT a failure
    - Due to pre-V9 package
    - Or, mismatched SQL text on BIND

APREUSE/APCOMPARE – Externalization

- Success/Failure messages written to PLAN_TABLE
  - Only if EXPLAIN(YES) or EXPLAIN(ONLY) used
    - NOTE: Upon failure, EXPLAIN(YES) rolls back. Persist for EXPLAIN(ONLY)
  - PLAN_TABLE.REMARKS is populated to indicate failure
  - NOTE: REMARKS column is NOT saved in SPT01
    - EXPLAIN PACKAGE will not externalize REMARKS
  - For APREUSE success - PLAN_TABLE.HINT_USED set to 'APREUSE'
  - See appendix for APCOMPARE/APREUSE externalization detail

APREUSE limitations

- Our hints are just that ... They are "hints"
  - Although we've done a lot of work to make them stick ...
- Hints are not enforceable 100% of the time
  - Incompatibilities between old and new release
    - V10 may merge query blocks, where V9 didn't
    - V10 may turn subqueries into non-correlated, V9 may correlate them
    - REASON = 20, 26, 32, ...
  - Insufficient information in PLAN_TABLE
    - e.g., Don't know which columns to use for merge join
    - e.g., Don't know if what type of parallelism we're using
    - REASON = 50 (comparison failures)
  - Other code limitations
    - e.g. Complex multi-index access trees aren't supported (REASON = 40)
- APREUSE failures are NOT defects ... They' are limitations
  - We have documented this clearly in our books
  - We will not take APARs on APREUSE limitations unless we are sure it is a defect

APREUSE ... Things to watch out for!

- Again, APREUSE will not work 100% of the time
  - Internal testing with hundreds of thousands of queries, failure rate around 1%
    - Although 1 failed query will fail the whole package
  - V10 -> V10 success rate much higher
- APREUSE is not a “sticky” option
  - The default is always NO, not what was used at previous REBIND
  - AUTOBIND always uses APREUSE(NO)
- Spurious REASON=50 (COLUMN_FN_EVAL) differences
  - DB2 9 had a defect that output the wrong value in COLUMN_FN_EVAL
    - Fixed via PM30425 in DB2 9 (requires rebind in DB2 9)
- REASON=48
  - DB2 10 needs virtual table rows to know where subqueries are attached
    - Without these rows, APREUSE may fail with REASON=48
    - Fixed via APAR PM30425 in DB2 9 (requires rebind in DB2 9)

Runtime Structure Implications

- REBIND PLANMGMT copies the runtime structure
  - "Copies" do not have newly generated runtime structures
    - Only "Current" generates a new runtime structure (except for SWITCH)
  - SWITCH will revert to saved runtime structure
    - Any new release runtime benefits are lost if SWITCHed back to prior release runtime structure
- APREUSE generates a new runtime structure
  - Which may benefit from new release runtime optimizations
    - APREUSE success only means access path "framework" is the same

DB2 10 Enhancements Requiring REBIND Which apply to APREUSE?

- REBIND APREUSE can benefit from:
  - Use of RELEASE(DEALLOCATE)
  - Early evaluation of residual (stage 2) predicates
  - Index include columns (assuming no access path change)
  - Predicate evaluation enhancements
  - RID list overflow improvements
- REBIND APREUSE cannot exploit:
  - IN-list improvements
  - SQL pagination
  - Query parallelism improvements
  - Index include columns (if access path change INDEXONLY=N → Y)
  - More aggressive view/table expression merge

RUNSTATS/REBIND migration recap

BIND, REBIND and EXPLAIN usage scenarios …

- **Migration from V9 – Conservative approach**
  - Conservative approach where minimal access path changes are required
    - Step 0 (Optional)
      - *Use REBIND ... EXPLAIN(ONLY) + APREUSE(ERROR)*
      - *Perform an impact analysis before actual REBINDs*
        - *NOTE: PLAN_TABLE output is not representative if APREUSE fails*
    - Step 1: REBIND PACKAGE (*)
      - *Use PLANMGMT(EXTENDED) ... backup of V9 access paths, just in case*

+ EXPLAIN(YES)

+ APREUSE(ERROR)

    - Step 2: For packages that failed Step 1 (i.e., leftovers)
      - *2a: Leave them as is ... they will be at the old level*

        OR
      - *2b. REBIND with PLANMGMT(EXTENDED) + APREUSE(NO)*
        - *This step exposes yourself to access path changes*
        - *But you have a backup*

BIND, REBIND and EXPLAIN usage scenarios …

- **Migration from V9 – Progressive approach**
  - Customer “open” to new access paths
  - But wants insurance against access path regression
    - Step 1: REBIND PACKAGE (*)
      - Use PLANMGMT(EXTENDED) ... backup of V9 access paths, just in case

+ EXPLAIN(YES)

+ APREUSE(NO)

+ APCOMPARE(WARN) … Perform comparisons

    - Step 2 (if required): In the event of regression
      - 2a. REBIND SWITCH(ORIGINAL) … Go back to V9 copy
      - 2b. (optional) EXPLAIN PACKAGE … To see what you have
      - 2c. REBIND PACKAGE APREUSE(ERROR) … to rebase on V10 runtime structures

BIND, REBIND and EXPLAIN usage scenarios…

- **Migration from V8**
  - Customer's only option is exposure to new access paths on REBIND
  - Insurance against access path regression necessary
    - Step 1: REBIND PACKAGE (*)
      - Use PLANMGMT(EXTENDED) ... backup of V8 access paths, just in case

+ EXPLAIN(YES)

    - Step 2 (if required): In the event of regression
      - REBIND SWITCH(ORIGINAL) … Go back to V8 copy
  - If you have V8 PLAN_TABLE records, "roll your own" APREUSE
    - Use OPTHINTS('hintname') the old-fashioned way
      - See existing documentation on how to do this
      - Once successfully rebound on V10 with desired access path, APREUSE can be used for future REBINDs
    - Packages that have hint failures <u>must</u> be rebound without hints
      - These are packages where PLAN_TABLE.HINT_USED = blank

Plan management what next?

Plan Management What Next?

- Future release planning based upon customer feedback
  - Need to break into persistent threads for BIND and REBIND
    - Including REBIND SWITCH
  - APREUSE(WARN) to allow new access path choice if re-use fails
    - Stepping stone to AUTOBIND usage of prior access path if available and REGENERATE for Native SQL Procedures
- We want customers to REBIND on current releases
  - To benefit from performance and quality associated with new runtime structures
  - As such – plan to force REBIND on packages from releases that are out-of-service

Appendix

RUNSTATS/REBIND recap when migrating from V8

- V8 preparation
  - If RUNSTATS will be difficult on large number of objects immediately after migration to V9/10, then REORG and/or RUNSTATS (V8) immediately prior to migration can reduce RUNSTATS need on V9/10 - as RUNSTATS INDEX under V9/10 can be sufficient to capture new CR/DRF
- V8->V9 migration
  - RUNSTATS objects as soon as possible after migration
    - Target dynamic applications first as these are exposed to new access paths immediately
  - Delay static REBINDs until associated objects have RUNSTATS run
- V8->V10 migration
  - RUNSTATS objects as soon as possible after migration
    - Target dynamic applications first as these are exposed to new access paths immediately
    - Equal priority - target static parallelism packages to REBIND to avoid incremental bind at each execution
  - Delay non-parallelism REBINDs until associated objects have RUNSTATS run

RUNSTATS/REBIND recap if migrating from V9

- V9 preparation
  - Begin using KEYCARD on RUNSTATS
  - Ensure all packages have been rebound on V9 to take advantage of plan management APREUSE/APCOMPARE in V10
- V9->V10 migration
  - RUNSTATS on all objects prior to REBINDs if KEYCARD not used pre-V10
  - REBIND static parallelism packages as soon as possible to avoid incremental bind at each execution
  - Delay non-parallelism REBINDs until associated objects have RUNSTATS run
  - BIND/REBIND options APREUSE/APCOMPARE are available on V10 for packages bound on V9

RUNSTATS/REBIND recap if data sharing coexistence from V8

- V8/9 co-existence
  - While in co-existence with V8
    - *Set STATCLUS=STANDARD*
    - *Set ABIND=COEXIST*
    - *Avoid REBIND*
  - Follow V9 migration steps after all members are V9, including setting zparms
    - *Set STATCLUS=ENHANCED*
    - *Set ABIND=YES*

RUNSTATS/REBIND recap if data sharing coexistence from V8 …

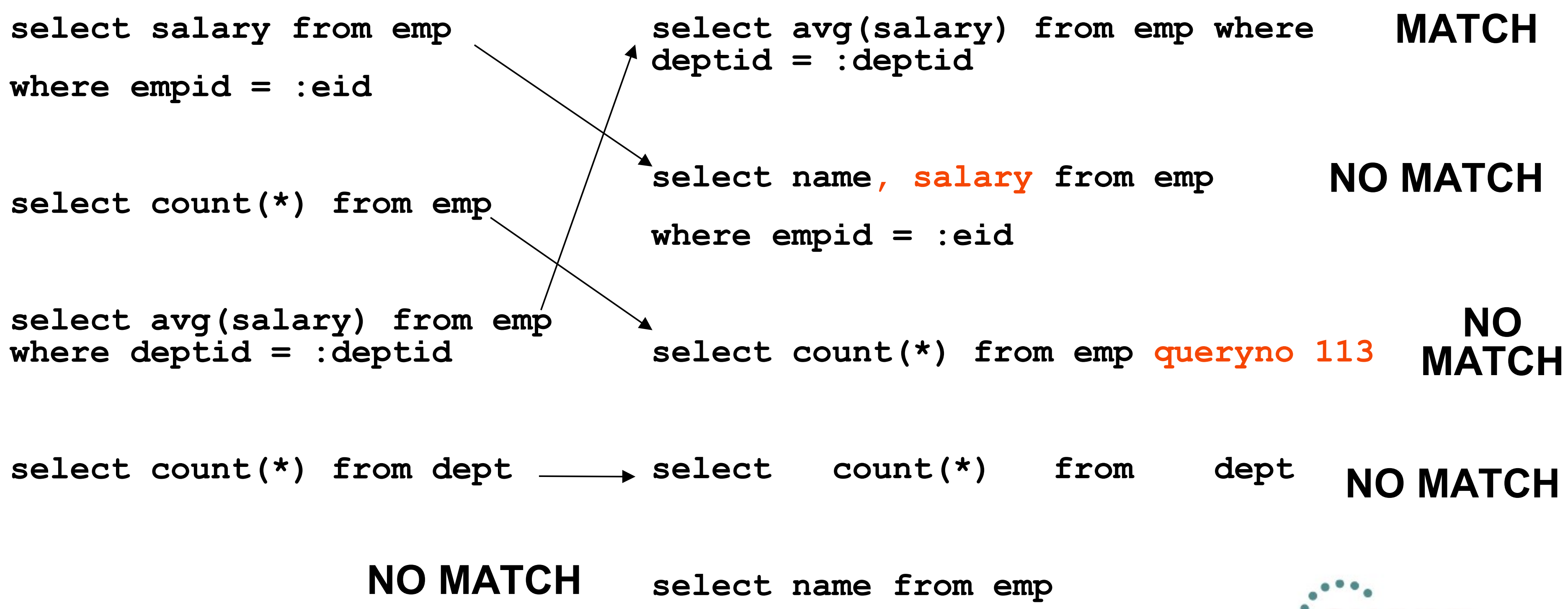
- V8/10 co-existence
  - While in co-existence with V8
    - *Set STATCLUS=STANDARD*
    - *Set ABIND=COEXIST*
    - *Avoid REBIND (see note below about static parallel queries)*
  - What to do with static parallel queries?
    - *Accept incremental bind whenever executed on V10 member*
    - *OR, REBIND with DEGREE('1') to disable parallelism while in co-existence.*
  - Follow V8-V10 migration steps after all members are V10, including setting zparms
    - *Set STATCLUS=ENHANCED*
    - *Set ABIND=YES*

RUNSTATS/REBIND recap if data sharing coexistence from V9

- V9/10 co-existence
  - Set ABIND=COEXIST while in co-existence with V9
  - What to do with static parallel queries?
    - Accept incremental bind whenever executed on V10 member
    - OR, REBIND with DEGREE('1') to disable parallelism while in co-existence.
  - Follow V9-V10 migration steps after all members are V10, including setting zparm
    - Set ABIND=YES

APCOMPARE/REUSE for application changes

- BIND PACKAGE supports APCOMPARE/APREUSE
  - Comparison only performed on statement text that is still the same between the old package and the new DBRM
  - Statement position does not matter



```
select salary from emp

where empid = :eid


select count(*) from emp


select avg(salary) from emp
where deptid = :deptid


select count(*) from dept
```

```
select avg(salary) from emp where
deptid = :deptid

select name, salary from emp

where empid = :eid

select count(*) from emp queryno 113

select    count(*)    from    dept

select name from emp
```

APCOMPARE/REUSE for application changes

- BIND PACKAGE ... REPLACE
  - Replacing an existing version
  - DB2 uses existing version to pick up old access paths
    - (LOCATION, COLLID, NAME, VERSION) used as key
- BIND PACKAGE ... ADD
  - New version being introduced
  - For old access paths, DB2 uses version that has
    - Identical (LOCATION, COLLID, NAME)
    - newest PCTIMESTAMP
- BIND PACKAGE ... COPY ... COPYVER
  - LOCATION, COLLID, NAME, VERSION is supplied by user
- Version used is reported via DSNT294I message
  - “TO PROCESS APCOMPARE AND/OR APREUSE, PRIOR ACCESS PATHS FROM VERSION (old-version) WERE USED.”

APREUSE failures

- APREUSE can fail with some of the same reasons as regular OPTHINTs
- SQLCODE +395 documents hint failure codes
  - 2 TABNO is invalid.
  - 3 TNAME is not specified.
  - 4 TNAME is ambiguous.
  - 5 TABNO doesn't agree with TNAME.
  - 6 QBLOCKNO doesn't agree with TNAME.
  - 7 PAGE_RANGE is invalid.
  - 8 PREFETCH is invalid.
  - 9 METHOD is invalid.
  - ...
  - 48 One or more virtual table rows pertaining to subquery blocks are missing.
  - 50 Output access path different from the hint specification
  - 99 Unexpected error.
- APREUSE adds one reason, REASON = 50
  - Hint was applied successfully but new access path <> hint
  - Detected by comparison

APCOMPARE output

Package summary reported via DSNT285I message

DSNT285I *csect-name bind-type* FOR PACKAGE = *package-name,* USE OF APCOMPARE RESULTS IN

count-1 STATEMENTS WHERE COMPARISON IS SUCCESSFUL

count-2 STATEMENTS WHERE COMPARISON IS NOT SUCCESSFUL

count-3 STATEMENTS WHERE COMPARISON COULD NOT BE PERFORMED.

bind-type

Type of BIND subcommand: BIND or REBIND.

package-name

name of the package in the format 'location-id.collection-id.package-id.(version-id)'.

count 1

The number of statements where previous access path was identical to the incoming access path.

count 2

The number of statements where previous access path was not identical to the incoming access path.

count 3

The number of statements where the comparison could not be performed. This could happen either because the previous access path was not found, or because the no new access path was generated.

System action

If APCOMPARE(WARN) was used, the command proceeds normally. If APCOMPARE(ERROR) was used and the package had a non-zero 'count-2', the command is aborted.

APCOMPARE output …

DSNT285I *csect-name bind-type* FOR PACKAGE = *package-name,* USE OF APCOMPARE RESULTS IN

count-1 STATEMENTS WHERE COMPARISON IS SUCCESSFUL

count-2 STATEMENTS WHERE COMPARISON IS NOT SUCCESSFUL

count-3 STATEMENTS WHERE COMPARISON COULD NOT BE PERFORMED.

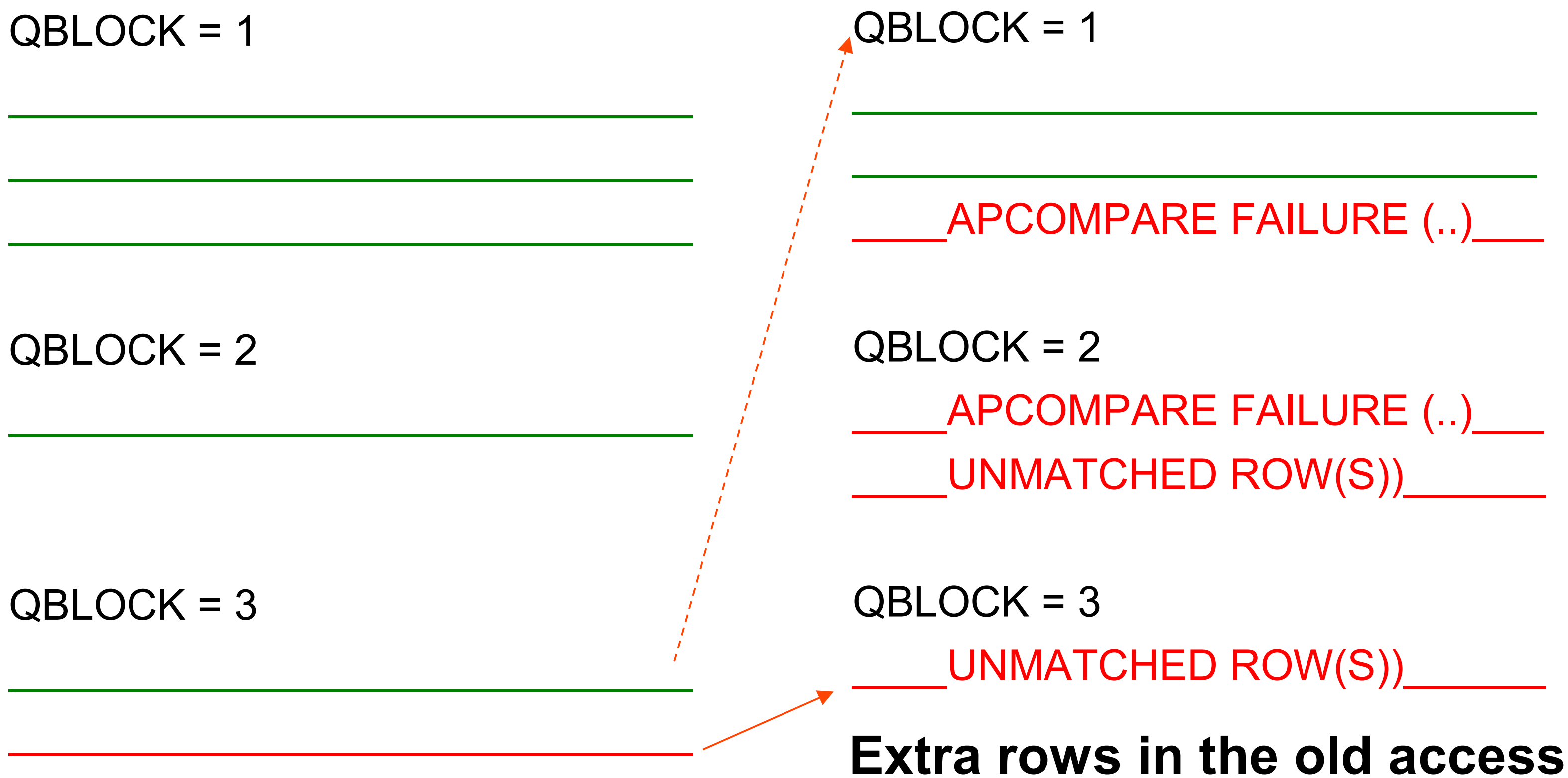
- *Only "EXPLAINABLE" statements are counted*
  - *These are statements that normally have rows in PLAN_TABLE*
- When is count-3 non-zero?
  - Typically when VALIDATE(RUN) was used
  - No access path was generated at BIND time
    - Either no previous access path available
    - OR, no new access path generated
  - SYSPACKSTMT.STATUS <> 'C' (compiled)

APCOMPARE output …

- Comparison details in PLAN_TABLE
  - Only if EXPLAIN(YES) or EXPLAIN(ONLY) used
  - PLAN_TABLE.REMARKS is populated

- Comparison logic
  - For each QBLOCK and PLANNO, compare PLAN_TABLE rows/columns
    - Populate REMARKS for row where difference found
    - REMARKS shows name of PLAN_TABLE column that's different
      - *E.g., "APCOMPARE FAILURE (COLUMN: ACCESSNAME)"*
  - When new/old access path has more rows
    - For e.g., differences in query blocks, extra sorts, etc.
    - "APCOMPARE FAILURE (UNMATCHED ROW(S))"
  - When old access path is unavailable (counted via count-3)
    - "APCOMPARE/APREUSE FAILURE (PREVIOUS ACCESS PATH NOT FOUND)"

APCOMPARE/APREUSE: Comparison Logic



- **Green = Comparison OK**
- **Red = Not OK**

Extra rows in the old access path are reported against existing rows in the current query block.

APREUSE output

Package summary reported via DSNT286I message

DSNT286I *csect-name bind-type* FOR PACKAGE = *package-name,* USE OF APREUSE RESULTS IN

count-1 STATEMENTS WHERE APREUSE IS SUCCESSFUL

count-2 STATEMENTS WHERE APREUSE IS EITHER NOT SUCCESSFUL OR PARTIALLY SUCCESSFUL

count-3 STATEMENTS WHERE APREUSE COULD NOT BE PERFORMED

count-4 STATEMENTS WHERE APREUSE WAS SUPPRESSED BY OTHER HINTS.

bind-type

Type of BIND subcommand: BIND or REBIND.

count 1

The number of statements where previous access path was successfully applied.

count 2

The number of statements where previous access path could not be applied, or was only partially applied.

count 3

The number of statements where the previous access path could not be reused. This could happen either because the previous access path was not found, or because the no new access path was generated.

count 4

The number of statements where the previous access path was not used because of the use of other types of hints such as PLAN_TABLE or SYSIBM.SYSQUERYPLAN hints.

System action

With APREUSE(ERROR), if the package has a non-zero value of count-2, the command is aborted.

APREUSE output …

- SYSPACKAGE.APREUSE
  - ‘N’ for (NO/NONE), ‘E’ for (ERROR)
- SYSPACKSTMT.ACCESS_PATH
  - ‘A’ if APREUSE(ERROR) was used

- Details in PLAN_TABLE
  - Only if EXPLAIN(YES) or EXPLAIN(ONLY) used
  - On success,
    - PLAN_TABLE.HINT_USED set to ‘APREUSE’
  - On failure
    - PLAN_TABLE.HINT_USED set to ‘’ (not set)
    - PLAN_TABLE.REMARKS is populated to indicate failure code
      - *E.g. “APREUSE FAILURE (REASON: 40)”*
    - Reason code documented in books

APREUSE output …

- SYSPACKAGE.APREUSE
  - ‘N’ for (NO/NONE), ‘E’ for (ERROR)
- SYSPACKSTMT.ACCESS_PATH
  - ‘A’ if APREUSE(ERROR) was used

- Details in PLAN_TABLE
  - Only if EXPLAIN(YES) or EXPLAIN(ONLY) used
  - On success,
    - PLAN_TABLE.HINT_USED set to ‘APREUSE’
  - On failure
    - PLAN_TABLE.HINT_USED set to ‘’ (not set)
    - PLAN_TABLE.REMARKS is populated to indicate failure code
      - *e.g. “APREUSE FAILURE (REASON: 40)”*
    - Reason code documented in books

Thank You for listening!

#SHAREorg

SHARE
Technology · Connections · Results

DB2 for z/OS Migration: Query Performance Considerations

Tom Beavin
IBM Silicon Valley Lab
February 7, 2013
Session 12740

Email: beavin@us.ibm.com

SHARE in San Francisco 2013