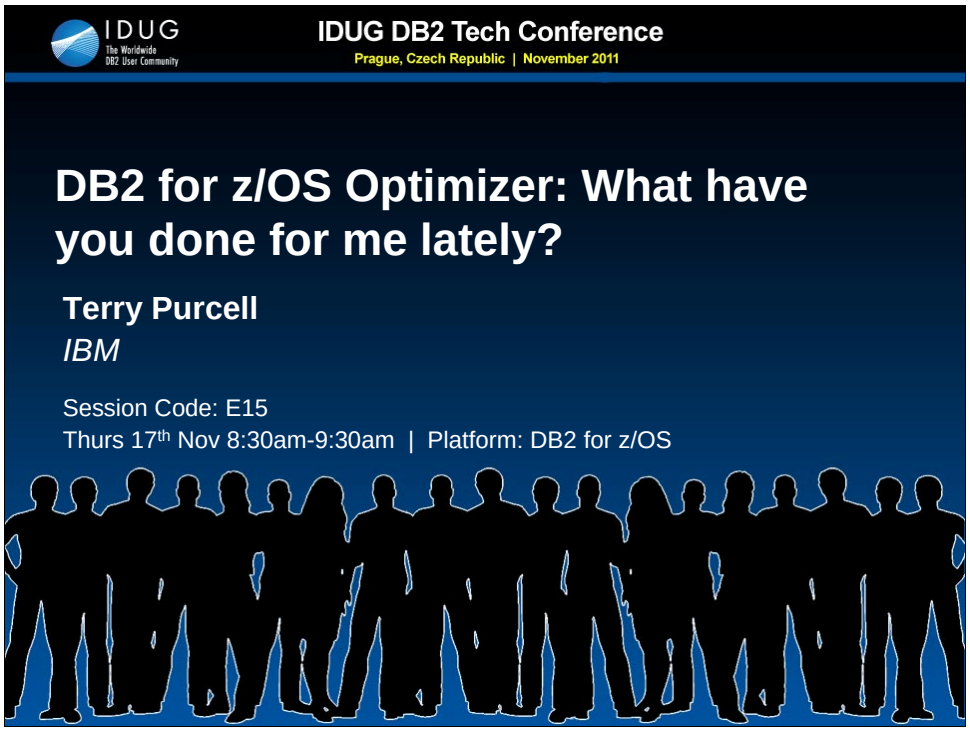


You can always read about the features/enhancements of a new release, such as DB2 10 for z/OS, in the manuals, in redbooks or by attending highlight presentations. But what you generally miss is the deep dive on these enhancements, and more importantly - what important enhancements have been delivered that are too small to make the 1 hour highlight presentations? And what enhancements have been delivered in the service stream???

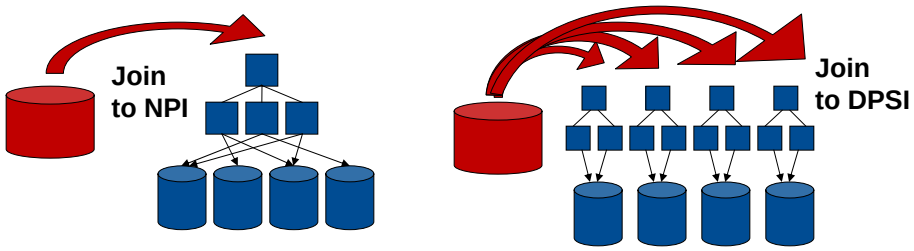
Agenda

- Any important fixes?
- OK, what do you need to remind me of?
- What's new for DB2 9 that you haven't told me?
- What's new for DB2 10 since GA?

DPSI Costing

- DPSIs have always had 1 clear sweet spot
  - When limited partitions qualify
- Sour spot has always been joins to a DPSI

- DB2 9 APAR PM25934
  - Improves cost recognition of DPSI probes for join



There is often a struggle to find the sweet spot where DPSIs can replace NPIs (NPSIs) to improve utility availability/performance without compromising query performance.

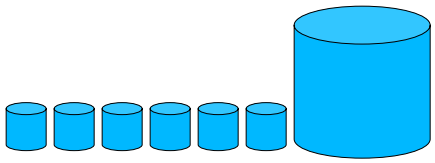
Unless there are local predicates against the partition limit keys to eliminate unnecessary partitions from the query, then DPSIs require all parts to be probed. Performance degradation can be experienced with DPSIs when these are accessed as the inner table of a join.

APAR PM25934 improves the optimizers recognition of the I/O overhead associated with accessing DPSIs, and therefore the optimizer is more likely to choose access paths that avoid the repeated probing of DPSIs.

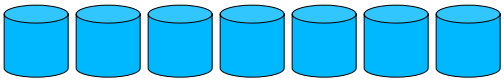
NPIs still have an advantage for query performance in that they are one index b-tree structure, but APAR PM25934 may allow some customers to exploit DPSIs in more situations. As usual, your mileage may vary, and thorough performance testing is recommended before implementing any index change.

Improved histogram collection

- Histograms delivered in DB2 9
  - Equal-depth histograms – each quantile should have an even distribution
    - Problem related to final quantile having the largest percentage



- DB2 9 APAR PM27249
  - Corrects algorithm to ensure more even distribution



Histogram statistics represent the distribution of values across a range, and are beneficial to capture this distribution for higher cardinality columns when the data may not be distributed equally throughout the range.

Prior to APAR PM27249, RUNSTATS statistics collected may produce histograms where the last quantile is considerably larger than other quantiles. PM27249 improves RUNSTATS histogram statistics collection to more evenly distribute the values throughout the quantiles.

Index on Expression

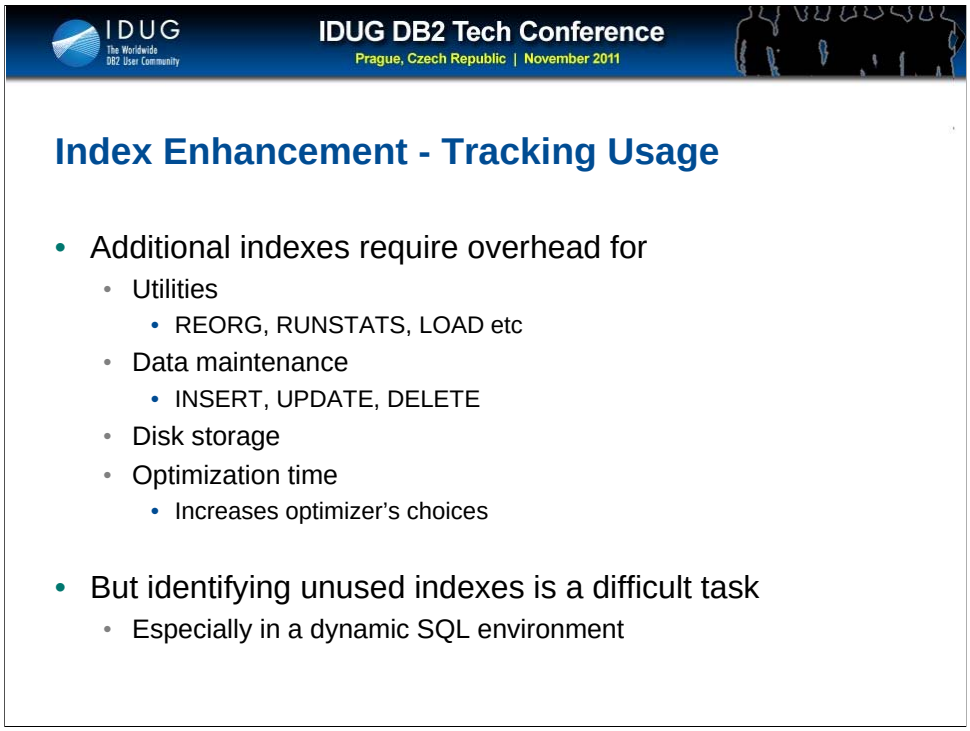
- DB2 9 delivered index on expression
  - However LIKE with parameter marker/host var was not indexable
    - UPPER(LASTNAME) LIKE ?
  - DB2 9 APAR PK96294 resolves indexability

DB2 9 delivered the capability to create indexes containing a column expression. One use case that expressions are used in SQL queries is for case-insensitive searches. For example, LASTNAME column is stored as mixed upper/lower case and to simplify the WHERE clause predicate, the search is performed as a single case – UPPER for example.

Prior to APAR PK96294, UPPER(LASTNAME) LIKE ? – using host variable or parameter marker, was not indexable. PK96294 provides indexability for Index on Expression with LIKE and parameter marker/host variable.

Agenda

- Any important fixes?
- OK, what do you need to remind me of?
- What's new for DB2 9 that you haven't told me?
- What's interesting in DB2 10
  - That I can't easily read about myself?



Additional indexes require overhead. When a table has many indexes on it, it is possible not all of them are useful. Sometimes indexes are created by default, or indexes are created sporadically over time – and the context or reason may later be forgotten.

If you can reduce the number of indexes on a table to only those necessary, you can reduce overhead that was associated with those unused indexes – such as insert/update/delete, utility overhead, disk storage and query optimization time.

Identifying unused indexes is particularly challenging in a dynamic SQL environment where queries are not captured in the catalog as they are for static SQL.

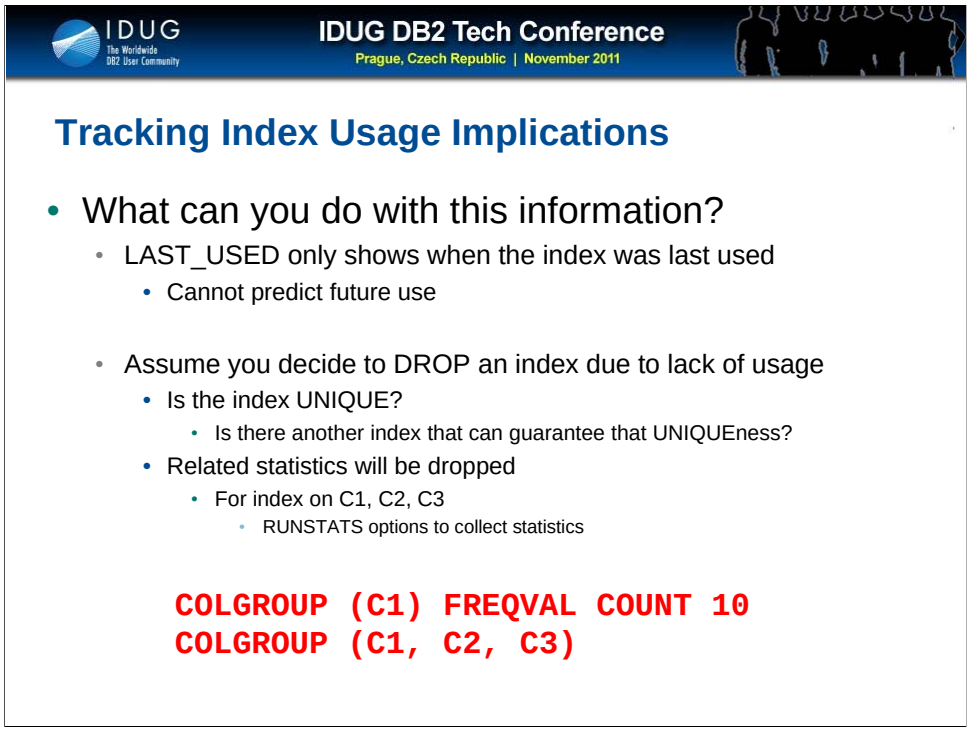
Tracking Index Usage

- RTS records the index last used date.
  - SYSINDEXSPACESTATS.LASTUSED
    - Updated once in a 24 hour period
      - RTS service task updates at 1st externalization interval (set by STATSINT) after 12PM.
    - if the index is used by DB2, update occurs.
    - If the index was not used, no update.
- "Used", as defined by DB2 as:
  - As an access path for query or fetch.
  - For searched UPDATE / DELETE SQL statement.
  - As a primary index for referential integrity.
  - To support foreign key access

DB2 9 tracks index usage in RTS. Once in each 24 hour period, the LASTUSED date will be updated if that index was “used” during that day. DB2 records this info but does not act on it.

What is considered as “used”?

If you insert a record in the table, the index is of course updated. But index maintenance is not considered as “used”. Only index usage in support of an access path is considered “used”.



Tracking index usage via LAST_USED is interesting in determining what indexes may not be used by your applications.

It should be noted that LAST_USED is not a guarantee of the requirement or redundancy of any index in your environment, and any decision to drop an index based upon LAST_USED should first consider other factors.

• LAST_USED shows the last used date and obviously is unable to predict potential future usage.

• Indexes may be required for enforcing a business rule such as uniqueness. Dropping a unique index is unwise unless there is a complementary or subset index that is also unique.

• Dropping an index also removes statistics that could have been associated with the index, and these statistics may have been beneficial for optimizer in choosing access paths. The fact that statistics were or were not beneficial for optimizer is not recorded by DB2. The be safe, it is possible to recollect the RUNSTATS using the COLGROUP to replace those removed when the index was dropped.

Clusterratio Enhancement

- New Clusterratio formula in DB2 9
  - Including new DATAREPEATFACTOR statistic
    - Differentiates density and sequential

Dense (and sequential)

Sequential (not dense)

- Controlled by zparm STATCLUS
  - ENHANCED is default
  - STANDARD disables, and is NOT recommended
- Recommend RUNSTATS before mass REBIND in first version AFTER V8

DB2 9 introduces a new RUNSTATS clusterratio (CR) formula and new statistics DATAREPEATFACTORF (DRF) which is beneficial to optimizer in estimating the data I/O cost associated with access to a table via an index. This change is to resolve deficiencies in the CR formula in V8 and prior.

CR is a measure of whether subsequent rows will benefit from dynamic prefetch. The addition of DRF allows optimizer to determine if those rows are on different pages, and whether those pages are within the range of pages already prefetched.

The new CR/DRF formula is controlled by zparm STATCLUS, and although STATCLUS=STANDARD results in CR calculation as per V8 and prior, this does not result in V8 optimizer behavior. Instead, the result would be V8 RUNSTATS with V9 (or V10) optimizer. Therefore, IBM strongly recommends the default STATCLUS=ENHANCED, unless otherwise instructed by DB2 support/development.

To take advantage of the new statistics formulas with the new optimizer, it is recommended to run RUNSTATS as soon as possible after migration to V9 or from V8→10, and BEFORE REBIND. The exception is datasharing co-existence where it is recommended to delay RUNSTATS until after all members are migrated off V8.

Clusterratio Impacts

- Clusterratio may be
  - Higher for indexes
    - With many duplicates (lower colcardf)
      - In recognition of sequential RIDs
    - On smaller tables
      - Less clusterratio degradation from random inserts
    - Indexes that are reverse sequential
  - Lower for random indexes
    - No benefit from dynamic prefetch
- Clusterratio(CR)/DataRepeatfactor (DRF) patterns

	High DRF	Low DRF
High CR	Sequential but not dense	Density matching clustering or small table
Low CR	Random index	Unlikely

The question is often asked – which indexes are likely to see a CR change from V8→9 or V8→10? The answer is likely to be all indexes except the clustering index (or any index that closely aligns with the clustering columns).

Some indexes will increase their CR in recognition of duplicate RID chains that would benefit from dynamic prefetch. Or and index may increase it's CR due to recognition that it is a DESC version of the clustering (or like the clustering index) and also smaller tables may see their indexes having higher CRs – and thus these indexes may not degrade their CRs as quickly due to unclustered inserts. This will help smaller tables avoid the problem of reverting from index access to tablespace scan due to unclustered inserts. Increases in CR may also result in less focus on list prefetch when dynamic prefetch is a more appropriate choice.

Indexes that are not expected to obtain any benefit from dynamic prefetch will see their CRs reduce under the new formula compared with V8.

Agenda

- Any important fixes?
- OK, what do you need to remind me of?
- What's new for DB2 9 that you haven't told me?
- What's new for DB2 10 since GA?

Production Modelling

- Have you ever copied production statistics to your dev/test systems for access path analysis
  - Only to have “system” factors result in different access paths than production?
  - In addition to catalog statistics, optimizer considers
    - CPU speed
    - # of CPs (for parallelism)
    - BP size
    - RID pool
    - Sort pool

When the optimizer is choosing an access path, it takes into consideration the SQL, available objects (indexes, MQTs etc), catalog statistics and environment information such as CPU speed, BP size, RID pool etc. While it is relatively easy for customers to create a development/test environment that simulates production by copying statistics from production to dev/test, which is done when a customer cannot create a full-sized copy including data.

Customers however have run into situations where replicating the statistics alone does not always result in the same access paths between the copy environment and production – due to other environmental reasons such as different CPU speed or BP sizes.

Production Modelling

- V9 APAR PM26475 & V10 APAR PM26973
  - Supports optimizer overrides for these system settings
  - New zparms
    - SIMULATED_CPU_SPEED
    - SIMULATED_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.

V9 APAR PM26475 & V10 APAR PM26973 adds the capability for simulating environment variables such as CPU speed, number of CPUs, sort pool, RID pool and BP sizes – so that a dev/test environment can be setup to simulate the system settings from production, without requiring that the non-production has the same CPU model and memory available as production.

In conjunction with copying statistics from production, this feature should allow customers to setup a non-production to mirror production.

Production Modelling – 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,63,2)) AS MAX_RIDBLOCKS,
        HEX(SUBSTR(IBM_SERVICE_DATA,9,4))  AS SORT_POOL_SIZE
 FROM PLAN_TABLE WHERE QUERYNO=6475;
```

*NOTE: CPU count is only populated if query chooses parallelism

To be able to set these values to match production, it is important to know exactly what the current production values are.

BP sizes can be obtained via a –DISPLAY BUFFERPOOL command. RID & sort can be retrieved from ZPARM values, or can be retrieved from the IBM_SERVICE_DATA column of the PLAN_TABLE – this method is recommended since it also provides the CPU speed and count which are not available from ZPARMs.

CPU count is only populated when a query chooses parallelism, and thus the example provided in this slide (against SYSDUMMY1) may not be sufficient to choose parallelism and an explain of a query against a larger table my be required. The SELECT for IBM_SERVICE_DATA could instead use WHERE PARALLELISM_MODE='C'.

Note also that CPU count is only important for optimizer in evaluating parallelism plans, and thus if you are not interested in parallelism, CPU_COUNT is not important to include in your simulated production environment.

Production Modelling – 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

This outlines the steps required to create and start the production modelling profile. The APAR closing text also includes a more detailed explanation of the steps required.

Production Modelling – 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

After attempting to simulate production, it is important to validate that you were successful. This slide highlights the 2 methods to validate that a production profile was exploited.

Agenda

- Any important fixes?
- OK, what do you need to remind me of?
- What's new for DB2 9 that you haven't told me?
- What's new for DB2 10 since GA?

Access Path Reuse / Compare For Packages

- APAR PM25679 (closed July 2011)
  - Introduces APREUSE & APCOMPARE options to BIND/REBIND
  - **BIND/REBIND PACKAGE … APREUSE(ERROR)**
    - DB2 attempts to reuse prior access paths
  - **BIND/REBIND PACKAGE… APCOMPARE(WARN | ERROR)**
    - DB2 issues a warning/error for each statement that has an access path change
- Prior package must have been bound on V9 or V10 (not V8 or prior)

Access Path Comparison (APCOMPARE)

- APCOMPARE
  - "Tell me if static SQL statements had changes in access paths"
  - Optionally, stop the BIND/REBIND if their were changes
- New option on
  - BIND PACKAGE
  - REBIND PACKAGE
  - REBIND TRIGGER PACKAGE
- Only works when package bound on V9 or higher
  - Starting with DB2 9, EXPLAIN information saved with the package in SPT01
  - Referred to as "EDB" or "Explain Data Block"
    - EDB is a compact representation of PLAN_TABLE
- 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

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 (contd.)

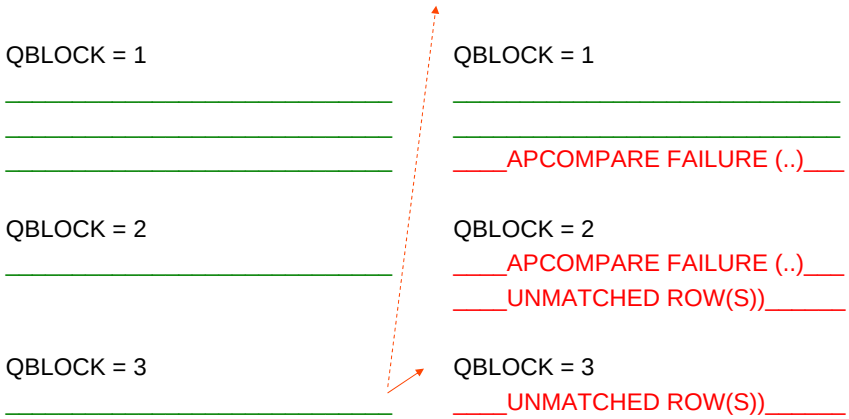
DSNT285I *csect-name bind-type* FOR PACKAGE = *package-name,* USE OF APCOMPARE RESULTS IN
count-1 STATEMENTS WHERE COMPARISON IS SUCCESSFUL
count-2 STATEMENTS WHERE COMPARISON IS NOT SUCCESSFUL
count-3 STATEMENTS WHERE COMPARISON COULD NOT BE PERFORMED.

- *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 (contd.)

- 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: Comparison Logic



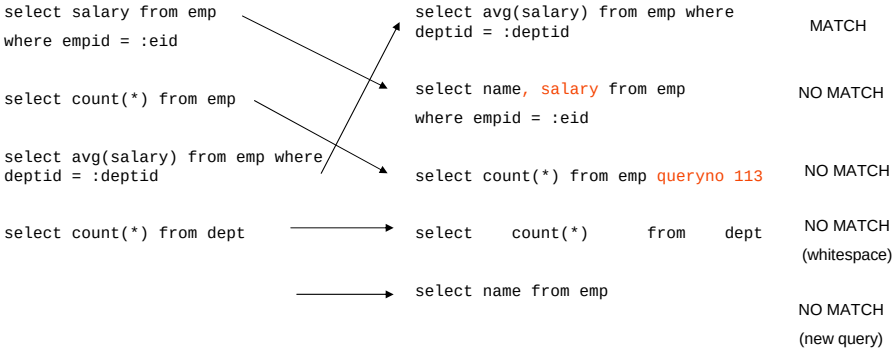
Extra rows in the old access path are reported against existing rows in the current query block.

Green = Comparison OK

Red = Not OK

APCOMPARE for application changes

- BIND PACKAGE supports APCOMPARE
  - Comparison only performed on statements that are still same between the old package and the new DBRM
  - Statement position does not matter



APCOMPARE 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.”

Access Path Reuse (APREUSE)

APREUSE - "Do the BIND/REBIND but try to avoid 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 and REBIND TRIGGER PACKAGE

- Only works when package bound on V9 or higher
  - Because EDB only saved starting with DB2 9

- 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
  - To be sure, new access path is compared with hint

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 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 (contd.)

- 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 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

APREUSE limitations

- Our hints are just that ... they're "hints"
  - Although we've done a lot of work to make them stick ...
- Hints aren't enforceable 100% of the time
  - Incompatibilities between old and new release
    - V10 may merge query blocks, V9 won'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're limitations
  - We're documented this clearly in our books
  - We won't take APARs on APREUSE limitations unless we're sure it's a defect

APREUSE ... Things to watch out for!

- Again, APREUSE won't work 100% of the time
  - Internal testing with hundreds of thousands of queries, failure rate around 1%
  - 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
  - Causes spurious differences
  - Fixed via PM30425 in DB2 9
    - Unless package is rebound with APAR, the differences will still show up
- 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 PM30425 in DB2 9
    - Again, needs package to be rebound

Migration Plan: V9 -> V10

- Conservative approach
  - Customer wants minimal access paths changes
- 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'll at old level
  OR
  2b. REBIND with PLANMGMT(EXTENDED) + APREUSE(NO)
    - This step exposes them to access path changes
    - But they have a backup
- Optional Step 0
  - Use REBIND ... EXPLAIN(ONLY) + APREUSE(ERROR)
  - Perform an impact analysis before actual REBINDs
- It's best to automate these steps!

Migration Plan: V9 -> V10 (contd.)

- "Progressive" approach
  - Customer open to new access paths
  - But wants insurance in case of regressions
- REBIND PACKAGE (*)
  - Use PLANMGMT(EXTENDED) ... For backups of V9
    + APREUSE(NO) ... Want to experience new access paths
    + APCOMPARE(WARN)... Perform comparisons
    + EXPLAIN(YES) ... Want details of access path changes
- In the event of a regression
  - REBIND SWITCH(ORIGINAL) ... Go back to V9 copy
  - EXPLAIN PACKAGE ... To see what you have
  - REBIND PACKAGE APREUSE(ERROR) ... To rebase on V10

Migration Plan: V8 -> V10

- APREUSE doesn't work on V8 packages ☹
  - But customers can "roll their own" APREUSE using OPTHINTs
  - APREUSE is essentially syntactic sugar over hints
- "Roll your own" APREUSE
  - Must have PLAN_TABLE records for packages
    - Either from old REBINDs with EXPLAIN(YES)
    - Or, from a fresh REBIND, perhaps into a dummy collection
  - Use PLANMGMT(EXTENDED) ... for backups
  - Use OPTHINTS('hintname') the old-fashioned way
    - See existing documentation on how to do this
  - Packages that have hint failures must be rebound without hints
    - When hints don't work, access path may be worse
    - These are packages where PLAN_TABLE.HINT_USED = blank

IDUG
The Worldwide DB2 User Community

IDUG DB2 Tech Conference

Prague, Czech Republic | November 2011

Terry Purcell

IBM

tpurcel@us.ibm.com

Session E15

DB2 for z/OS Optimizer: What have you done for me lately?