2010 IDUG North America

What's new from the optimizer in DB2 10 for z/OS?

Terry Purcell
IBM Silicon Valley Lab
tpurcel@us.ibm.com

Session Code: B02

Tues, May 11th 1:45 PM - 2:45 PM
Platform: DB2 for z/OS

Disclaimer/Trademarks

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements, or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

The information on the new product is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The information on the new product is for informational purposes only and may not be incorporated into any contract. The information on the new product is not a commitment, promise, or legal obligation to deliver any material, code or functionality. The development, release, and timing of any features or functionality described for our products remains at our sole discretion. *

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious, and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks The following terms are trademarks or registered trademarks of other companies and have been used in at least one of the pages of the presentation:

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both: AIX, AS/400, DataJoiner, DataPropagator, DB2, DB2 Connect, DB2 Extenders, DB2 OLAP Server, DB2 Universal Database, Distributed Relational Database Architecture, DRDA, eServer, IBM, IMS, iSeries, MVS, Net.Data, OS/390, OS/400, PowerPC, pSeries, RS/6000, SQL/400, SQL/DS, Tivoli, VisualAge, VM/ESA, VSE/ESA, WebSphere, z/OS, zSeries

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.
Intel and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.
UNIX is a registered trademark of The Open Group in the United States and other countries.
Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
Other company, product, or service names may be trademarks or service marks of others.

Agenda

- Access path management
  - Dynamic Statement Cache Enhancements
  - Access Path Stability
  - Instance Based Statement Hints
- Query performance improvements
  - Safe Query Optimization
  - Aggressive View Merge
  - IN List Processing
  - SQL Pagination
  - Parallelism Enhancements

Dynamic Statement Cache

- Introduced in DB2 V5
- Re-uses SQL and access path
  - If identical SQL string
  - If same user,…
- Avoids full prepare (like a BIND)
- Good programming practice to use parameter marker (?)
  - ? are parameter markers
  - Ensures SQL is always the same
- Not all programs use ?
  - Ruby On Rails generates literals not ?
  - So SQL can not be re-used in Cache

4

Prior to DB2 V5 the use of dynamic SQL was frowned upon by most DBA's as it meant that every piece of dynamic SQL had to undergo a process similar to a Bind for DB2 to assess the access path. The introduction of the dynamic statement cache meant that the access path was calculated the first time the SQL was executed but every other time the same SQL was executed by the same user a prepare could be avoided as the access path had been remembered. This meant that using dynamic SQL could be almost as efficient as static SQL – great news for companies running SAP, SIEBEL, Peoplesoft or other similar systems.

In order to take advantage of re-use in the cache programs had to be coded with parameter markers, these are similar to using host variables in static SQL. This meant that SQL would always be identical even if the values in the parameter markers changed.

But not all SQL uses parameter markers, program generators such as Ruby On Rails generates dynamic SQL with literals rather than using parameter markers. Because the literals are likely to be different with every execution of the SQL then very little re-use of the SQL in the cache can take place.

A prepare must take place for each unique piece of SQL so the whole system can run slower than a similar system using parameter markers.

Literal Replacement

- Dynamic SQL with literals can now be re-used in the cache
  - Literals replaced with & (similar to parameter markers but not the same)
- To enable either you:-
  - Put CONCENTRATE STATEMENTS WITH LITERALS in the ATTRSTRING in the PREPARE
  - Or set LITERALREPLACEMENT in the ODBC initialization file
  - Or set the keyword enableLiteralReplacement='YES' in the JCC Driver
- 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
    - Additional match on literal usability
    - Can only match with SQL stored with same attribute, not parameter marker
  - If not found, new SQL is prepared and stored in the cache

5

In DB2 x more SQL can now be re-used in the cache.

To enable this you can do one of the following.

On the client the PREPARE statement can be changed to include the 'CONCENTRATE STATEMENTS WITH LITERALS'

The JCC Driver on the client side can be changed to include the keyword "enableliteralReplacement='YES'"

Set LITERALREPLACEMENT in the ODBC initialization file in z/OS – this will enable all SQL coming into DB2 through ODBC to have literal replacement enabled.

Literal Replacement …

- Example:

  SELECT BALANCE FROM ACCOUNT WHERE ACCOUNT_NUMBER = 123456

  - This would be replaced by

  SELECT BALANCE FROM ACCOUNT WHERE ACCOUNT_NUMBER = &

- Performance Expectation
  - Using parameter marker provides still best performance
  - Biggest performance gain for small SQL with literals that have a cache hit now, but did not before
  - Determined access path is not optimized to provided literals,
    - Need to use REOPT for that purpose

Access Path Stability

- Query optimization depends on many inputs.
  - Some of which may be missing/incomplete.
  - Small changes in the environment can change (degrade) an access path.
- Re-optimization is not always the preferred solution.
  - Static SQL typically insulated from erratic access path changes.
  - Dynamic SQL far more prone to access path degradation.
- When access paths change due to re-optimization (REBIND or PREPARE), often difficult to revert to a prior access path.
- In DB2 V9 (APAR PK52522)
  - Solution provided for fallback to prior path static SQL across REBIND.
- DB2 10 for z/OS
  - extends the V9 solution unifying the treatment of static and dynamic.

7

Query optimization in DB2 depends on many inputs, and even minor changes in the database environment can cause access paths to change. In addition, query optimizers aren't perfect, and given that they often work with incomplete information (typically statistics), suboptimal access paths aren't uncommon. Sometimes, a re-optimization yields access paths that are worse than before, causing performance regressions that may result in application outages and eventually loss of productivity/revenue for database users. This application instability due to changes in access paths is a long-standing issue facing many DB2 users. Specifically:

• Users do not have a easy way of guaranteeing that query access paths will not change unexpectedly. This problem affects static and dynamic SQL although it is more severe for dynamic SQL.

• Static SQL queries are usually insulated from erratic changes in the access paths because DB2 allows pre-compiling/binding these queries such that an executable form is created and stored in the

database. This executable form is subsequently reused for multiple submissions of that query. However, there are situations when static SQL queries are re-optimized and access paths regress.

This usually happens after an explicit user action (via BIND/REPLACE or REBIND) but it could also be triggered automatically due to a change in the database (such as dropped indexes).

• Dynamic SQL queries are more prone to wild swings in access paths. This is because DB2 does not retain access paths of dynamic queries on disk. Dynamic SQL access paths are typically

cached in memory, and once an access path is purged from the cache, a new one needs to be generated (PREPARE). There is no guarantee that the same access path will be generated again, or

that the newer access path would be better.

• Users don't necessarily know what access path was in play when a query actually regressed, or better yet, what was the optimal access path prior to the regression. This is often the case if the user hasn't

taken steps to retain access path descriptions (EXPLAIN records). For static SQL, an access path could have been previously saved by the user if a query was optimized with a specified option

(EXPLAIN(YES)) when performing a BIND/REBIND. However, due to the nature of dynamic SQL, it is rare for customers to have access path history for their dynamic queries. Even if the user can recreate

the "bad" access path, it may be impossible to determine the "good" access path that existed prior to the regression. Consequently, problem diagnosis becomes extremely hard since a user is unable

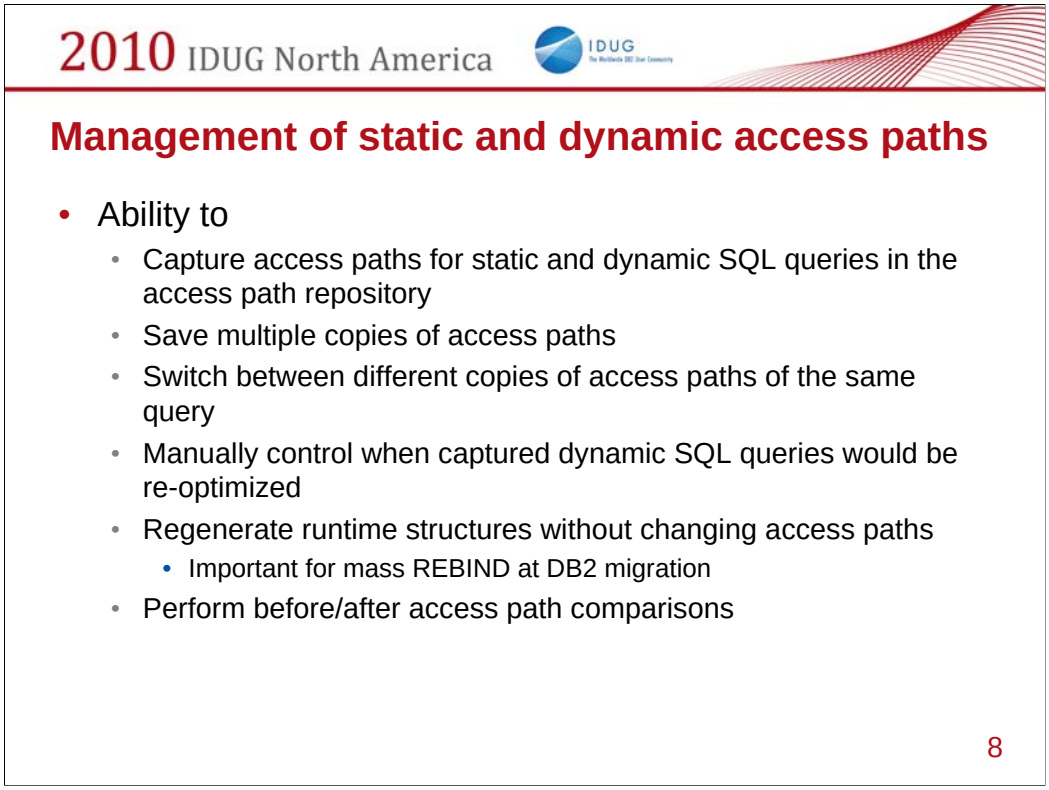
to compare the new access path with an older one.

• When access paths do change due to re-optimization (REBIND or PREPARE) and a regression has been detected, users do not have an easy way of reverting to prior access paths.

• Sometimes, users need to REBIND packages merely to regenerate the executable forms that are saved persistently on disk. This may be necessary to correct defective runtime structures created due to software bugs, or to allow the exploitation of new features (such as 64 bit support). Unfortunately, users don't have an easy mechanism to "lock down" their existing access paths such that the regenerated

structures still represent the old access paths.

In DB2 9, DCR DK266 (APAR PK52522), a limited solution was delivered to address this issue of query plan instability in static SQL statements. This line item proposes a comprehensive framework that significantly extends that solution, and does so in a fashion that unifies the treatment of static and



Access Path stability function introduces a framework for the management of access paths of static and dynamic SQL queries. It comprises of an access path repository along with operations/commands that manage this repository. It is being developed in conjunction with a companion line item, Statement level hints. It will deliver support for server-wide hints along with critical infrastructure that is needed by plan stability. This function will deliver the following:

• Ability to capture access paths for static and dynamic SQL queries in the access path repository

• Support the ability to save multiple copies of access paths

• Support the ability for users to switch between different copies of access paths of the same query

• Allow users to manually control when captured dynamic SQL queries would be re-optimized

• Regenerate runtime structures without changing access paths

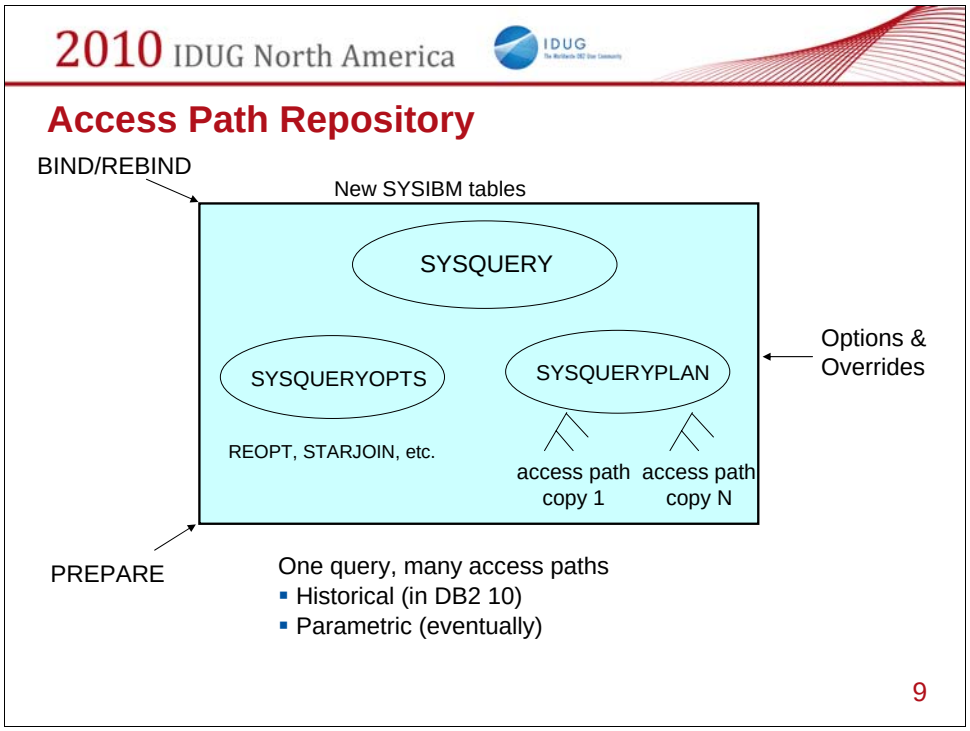
• Extend REBIND PACKAGE ... PLANMGMT and SWITCH options to native SQL stored procedure packages

The repository is populated with access paths of static and dynamic SQL queries. In addition, it can also be populated via hints. Whether static and dynamic SQL access paths are captured into a repository is governed by two settings – PLANMGMT and PLANMGMTSCOPE. These settings can be specified via several different mechanisms. Collectively, these mechanisms support varying levels of granularity. These are listed in the order of precedence, from lowest to highest.

• Via subsystem parameters (ZPARMs) called PLANMGMT and PLANMGMTSCOPE. PLANMGMT is an existing ZPARM introduced in DB2 for z/OS 9. PLANMGMTSCOPE is new ZPARM. These

ZPARMs have a member-wide impact.

• Via bind options, also called PLANMGMT and PLANMGMTSCOPE. These options apply to BIND



The access path repository holds important query metadata (such as query text), query access paths and other information (such as optimization options). The repository supports versioning by maintaining copies of access paths and associated options for each query. Note that we use the term 'copy' instead of 'version' in order to avoid any confusion with package versions that are also supported by DB2.

The repository comprises of several new catalog tables. The central tables are:

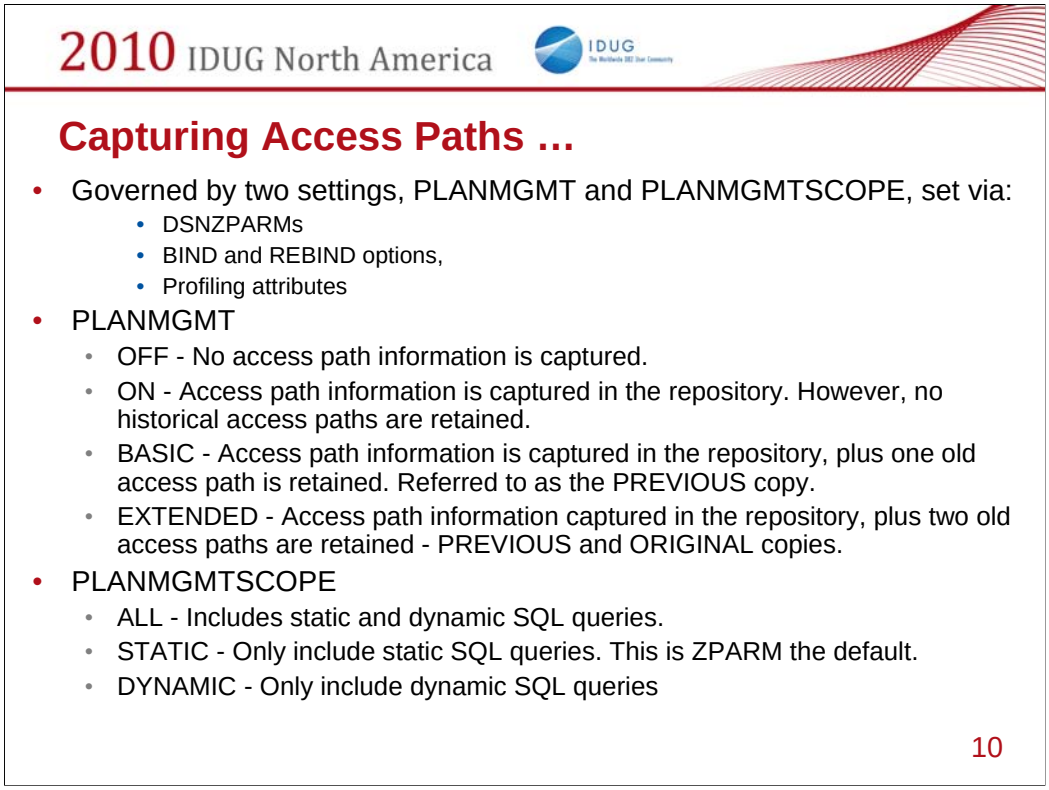
• SYSIBM.SYSQUERY is the central table of the access path repository. It holds one row for each static or dynamic SQL query that's being stabilized. It also hold rows that represent user-specified hints.

For static and dynamic queries, much of the information stored in this table is common to all copies.

• SYSIBM.SYSQUERYPLAN holds access paths descriptions (i.e. EXPLAIN records) for each query represented in SYSQUERY. SYSQUERYPLAN contains a set of columns that are derived from the

existing PLAN_TABLE schema. In the presence of versioning, a query in SYSQUERY can have more than one access path for it.

• SYSIBM.SYSQUERYOPTS holds miscellaneous information about each copy of a query in SYSQUERY.



PLANMGMT can have one of the following settings. Not all commands will support each of these options:

• OFF - No access path information is captured.

• ON - Access path information is captured in the repository. However, no historical access paths are retained.

• BASIC - Access path information is captured in the repository. In addition, one old access path is retained. This is referred to as the PREVIOUS copy.

• EXTENDED - Access path information captured in the repository. Two old access paths are retained. These are referred to as the PREVIOUS and ORIGINAL copies.

In DB2 10 for z/OS, the default value of the PLANMGMT sub-system parameter will be EXTENDED. This is a change from DB2 9 for z/OS (which used a default of OFF).

PLANMGMTSCOPE determines whether the PLANMGMT setting applies to static SQL, dynamic SQL, or both. It only comes into play when PLANMGMT is set to a value other than OFF. PLANMGMTSCOPE provides a finer level of granularity and can have the following values:

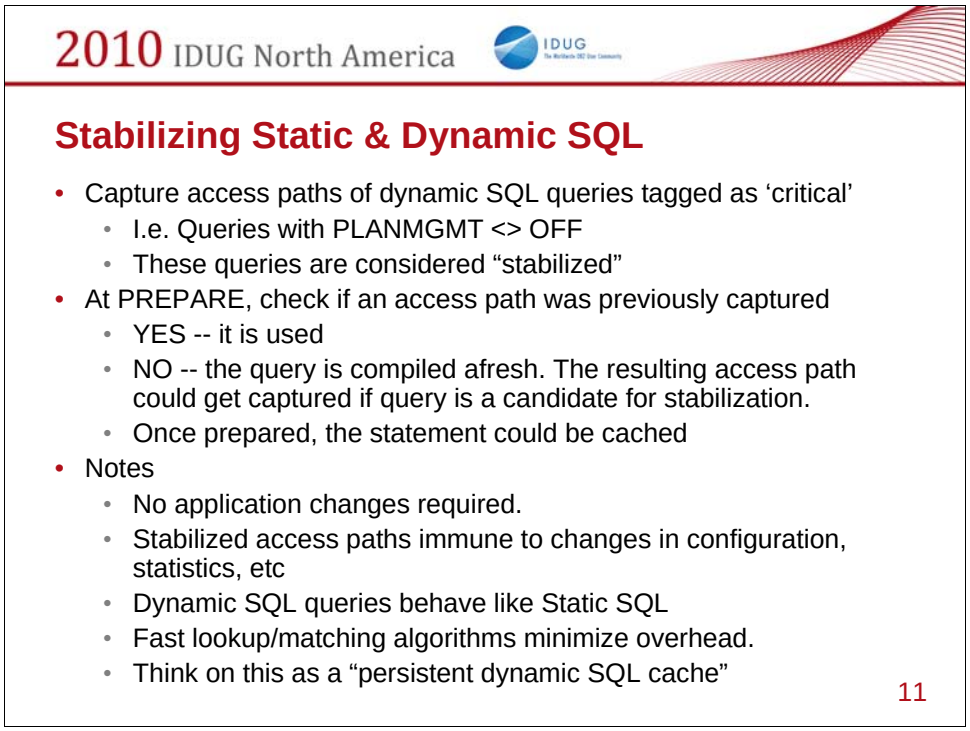
• ALL - Includes static and dynamic SQL queries.

• STATIC - Only include static SQL queries. This is the default value of the ZPARM.

• DYNAMIC - Only include dynamic SQL queries

Static SQL access paths are captured when PLANMGMT is set to a value other than OFF, and PLANMGMTSCOPE is either ALL or STATIC. These access paths are invoked when a user issues a BIND

PACKAGE or REBIND PACKAGE command. .



Dynamic SQL access paths are captured at *full PREPARE*. Irrespective of the PLANMGMT/PLANMGMTSCOPE settings, DB2 first checks the access path repository for the presence of a previously

saved access path. If such an access path exists and it is marked as valid (i.e. SYSQUERYOPTS VALID = 'Y'), that access path is used during query compilation. However, if no valid access path

exists, DB2 generates a new access path. This access path may be captured into the repository if PLANMGMT is set to a value other than OFF, and PLANMGMTSCOPE is either ALL or DYNAMIC. Any

older access paths may be retained depending on the exact value of PLANMGMT.

For dynamic SQL, only the following types of queries are captured: SELECT, INSERT with fullselect (i.e. INSERTs that do not use the VALUES clause), UPDATE, DELETE, MERGE and TRUNCATE.

REBINDing Dynamic SQL

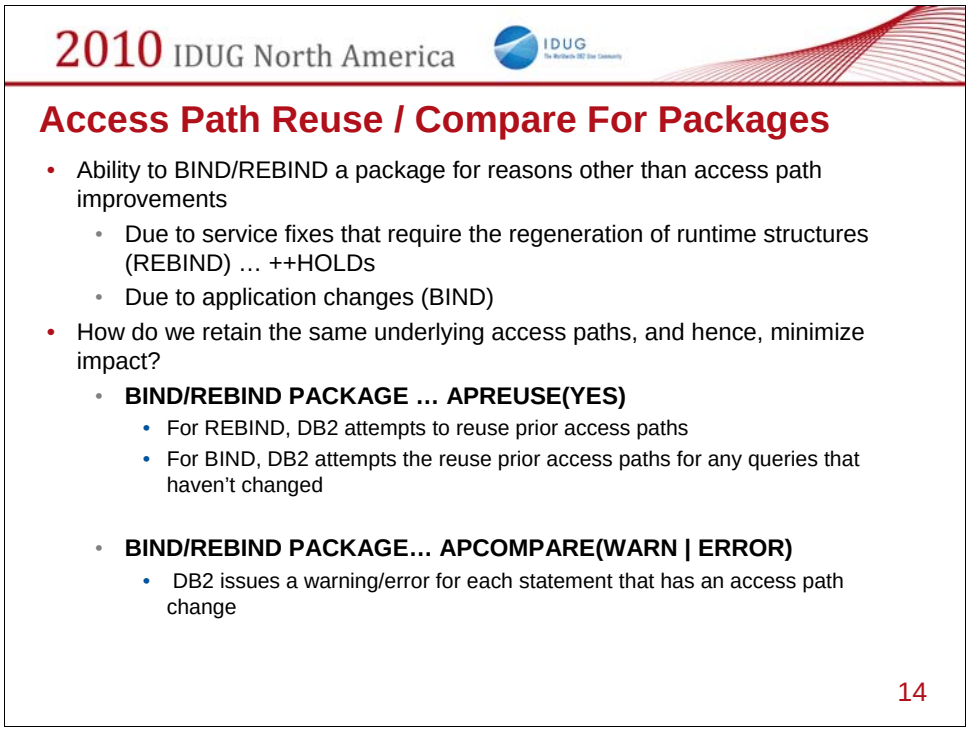
- Captured access paths may not stay optimal forever
  - Periodic tuning/maintenance may open up more optimal access paths
- New mechanism to REBIND captured dynamic SQL statements
  - Generate new access paths, either immediately or at next full PREPARE
  - Previous access paths can be retained as a backup
  - Similar to REBIND PACKAGE
- New mechanism to switch to a prior access path
  - Used when new access paths cause performance regressions
  - Similar to REBIND PACKAGE … SWITCH(PREVIOUS/ORIGINAL)
- Support varied scope
  - All queries
  - Queries that originated from a package
  - Queries that any user-definable criteria

Access Path "Fallback"

- For each stabilized package/query, DB2 retains a few copies of old query plans
  - CURRENT – The active copy
  - PREVIOUS – The copy in use before the last REBIND. Transient.
  - ORIGINAL – The oldest/first query plan. Never overwritten.
- When query plans cause performance regressions …
  - A DBA can revert to a prior query plan

- Static SQL
  - Reverts back to an older copy of the one or more package
  - REBIND PACKAGE (HRCOLL.*) … SWITCH(PREVIOUS)
- Dynamic SQL
  - Reverts back to an older copy of a dynamic SQL query
  - REBIND QUERY … SWITCH(PREVIOUS | ORIGINAL)

  - REBIND QUERY FILTER('REGRESSED') … SWITCH(ORIGINAL)
  - REBIND QUERY PACKAGE(HRCOLL.*) … SWITCH(PREVIOUS)



Access path comparison and lockdown

Sometimes, users need to BIND/REBIND their static SQL packages and yet maintain the exact same access paths as before. They'd like to merely regenerate the executable forms that are saved persistently on disk. This may be necessary to correct defective runtime structures due to software bugs, or to allow the exploitation of features that may have become available in newer releases of DB2 (such as 64 bit support). To support this need, BIND PACKAGE and REBIND PACKAGE commands will be extended with two new options:

• APREUSE

This option has two legal values - YES and NO. The default is APREUSE(NO). When APREUSE(YES) is specified, DB2 will attempt to reuse existing access paths. This enforcement

is not guaranteed in all cases. Indeed, there may be isolated situations esp. when the two packages are from different releases wherein such enforcement may not work. If DB2 is not able to reuse

an old access path, a DSNT286I message is issued with the appropriate reason code, and DB2 continues processing.

• APCOMPARE

This option has three legal values - WARN, ERROR and NONE. With APCOMPARE(WARN) and APCOMPARE(ERROR), DB2 compares the old and new access paths for each matching statement. If the access paths are structurally dissimilar, DB2 will raise a DSNT285I message. If APCOMPARE(WARN) is the option specified, DB2 will continue processing.

If APCOMPARE(ERROR) was used, DB2 will terminate the processing of the package. With APCOMPARE(NONE) , DB2 does not perform any access path comparison.

AP Reuse/Compare – Anticipated Scenario

- Mass rebind recommended after DB2 migration
  - 1st reason to regenerate runtime structures under new release
    - Regain loss of runtime optimizations such as SPROCs (approx 7% CPU)
  - Concern is access path regression
- Suggested scenario
  - REBIND APREUSE(YES) APCOMPARE(ERROR)
    - Allow DB2 to attempt to reuse the prior path (not guaranteed)
    - Fail the REBIND if access path changes
  - Anticipated that majority of REBINDs will succeed
    - Regaining lost CPU due to invalidated runtime structures
    - Mitigating risk of access path regression
  - After migration stabilizes
    - Explore new access path possibilities

Getting information on packages

- SYSIBM.SYSPACKCOPY
  - New catalog table
  - Hold SYSPACKAGE-style metadata for any previous or original package copies
  - No longer need to SWITCH to see information on inactive copies
- EXPLAIN PACKAGE
  - Extract PLAN_TABLE information for one or more packages and their copies
  - The package/copy must be created on DB2 9 or later
  - Useful if you didn't BIND with EXPLAIN(YES)

Freeing captured access paths

For static SQL, the existing FREE PACKAGE command is used to delete a package from the system. DB2 also supports an existing PLANMGMTSCOPE option to delete only the older copies of a package. With this line item, when a FREE PACKAGE command deletes a package copy, it will also delete any rows in the access path repository that refer these copies.

For dynamic SQL queries, a new FREE QUERY command will be introduced. FREE QUERY will purge one or more queries from the access path repository. If any of the specified queries are resident in the dynamic SQL cache, they will be purged from the cache. The exact set of queries that will be re-optimized can be specified via a command line option. These options are similar to those supported for REBIND QUERY:

• Free one/all dynamic SQL queries that have been captured in the repository. For e.g.:

FREE QUERY QUERYID(6557)

FREE QUERY QUERYID(ALL)

• Free only a subset of the dynamic SQL queries via a filtering criteria. For e.g.:

FREE QUERY FILTER('UNUSED')

DB2 will free all queries for which SYSQUERY.FILTER is set to the value specified on the command line (i.e. 'UNUSED')

The same FREE QUERY command can also be used to free static SQL queries and hints. Much like FREE PACKAGE, FREE QUERY will also support an optional PLANMGMTSCOPE option. PLANMGMTSCOPE has two legal values - ALL and INACTIVE. If the PLANMGMTSCOPE clause

Robust hints system

- Current limitations in hint matching
  - QUERYNO is used to link queries to their hints – a bit fragile
  - For dynamic SQL, require a change to apps – can be impractical
- New mechanisms being considered:
  - Associate query text with its corresponding hint … more robust
  - Hints enforced for the entire DB2 subsystem, irrespective of static vs. dynamic, etc.
  - Hints integrated into the same access path repository
- PLAN_TABLE isn't going away
- Only the "hint lookup" mechanism is being improved.

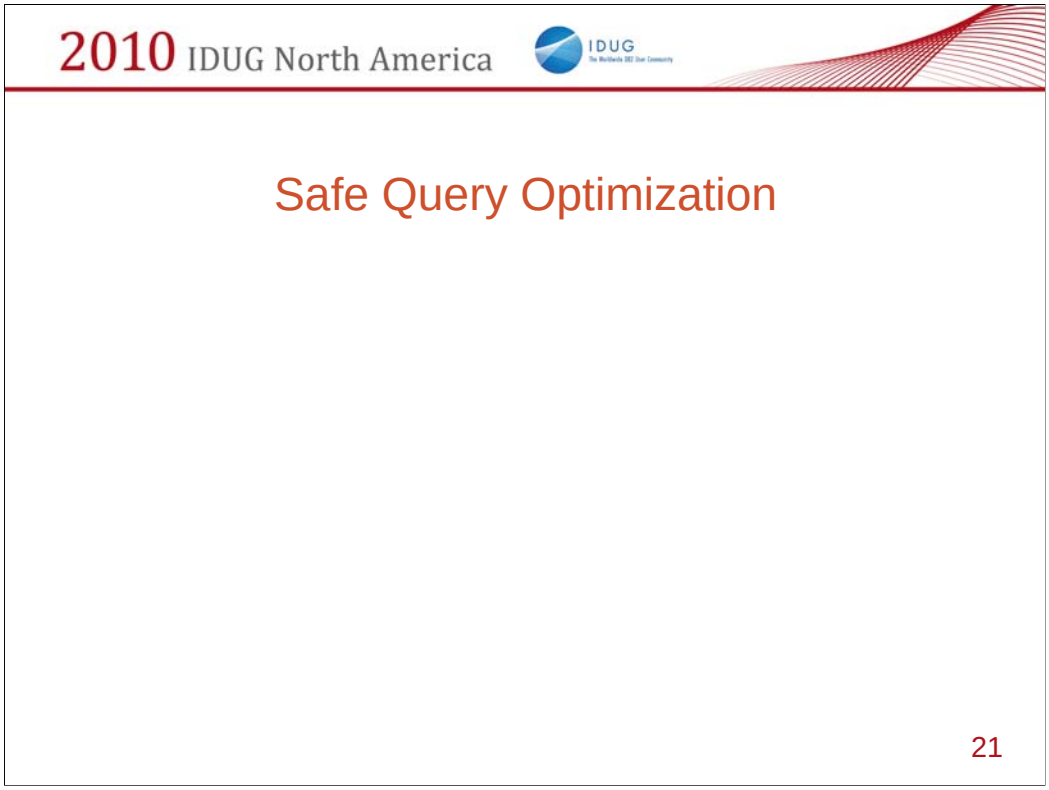
Robust hints system (cont.)

Steps to use new hints:

- Populate a user table called DSN_USERQUERY_TABLE with query text of one or more queries.
- Populate PLAN_TABLE with the corresponding hints, OR,
- Run new command, BIND QUERY, to integrate the hint into the repository.
- FREE QUERY can be used to remove the hint.

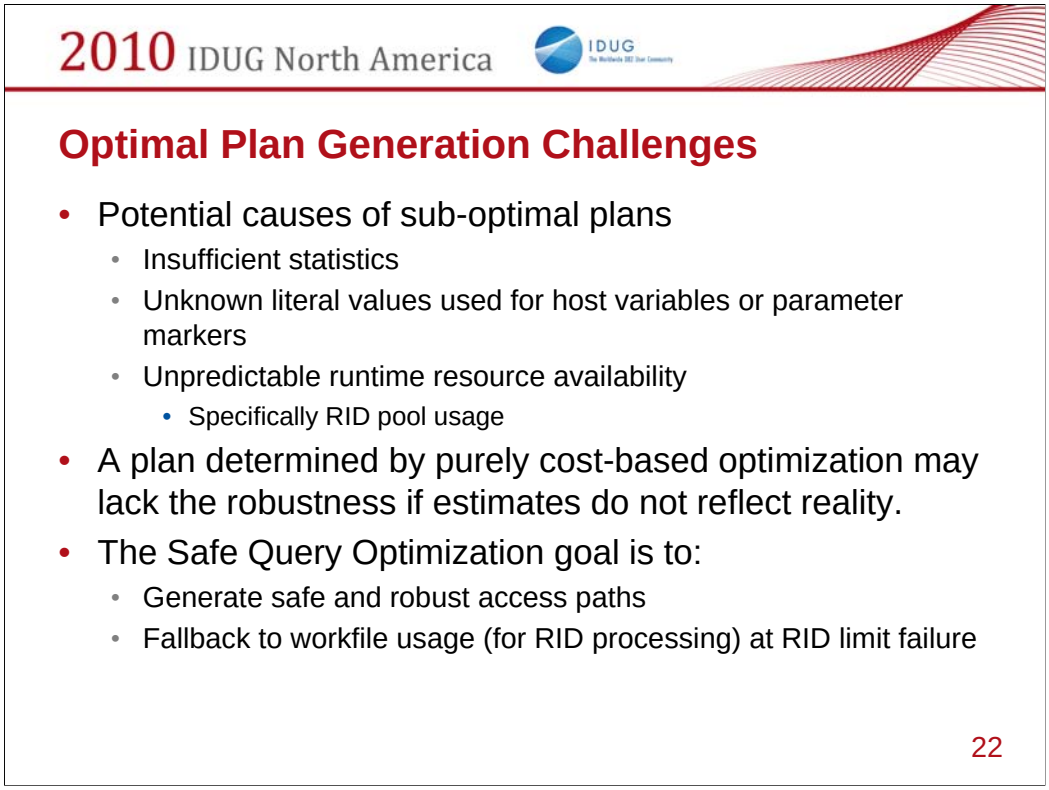
Statement-level BIND options

- DB2 supports a few ways to influence query processing behavior
  - ZPARMs to influence all queries on the subsystem
  - BIND options for queries in a entire package
- However, a statement-level granularity is sometimes needed
  - E.g., a package bound with REOPT(NONE) may have one statement that needs REOPT(ALWAYS)
- New mechanisms for statement-level bind options:
  - Similar to mechanism used for hints
  - DSN_USERQUERY_TABLE can also hold per-statement options



Line items 868

Safe Query Optimization – Runtime Trending Analysis



Abstract

Cost-based optimization may not always generate the optimal plan due to a number of reasons, such as:

• Insufficient statistics, for example, join selectivity of predicate 'T1.C1=T2.C1'.

• Non-substantiated query optimization assumption, for example, RANGE predicate default selectivity table.

• Unpredictable runtime resource availability, for example, RID pool usage.

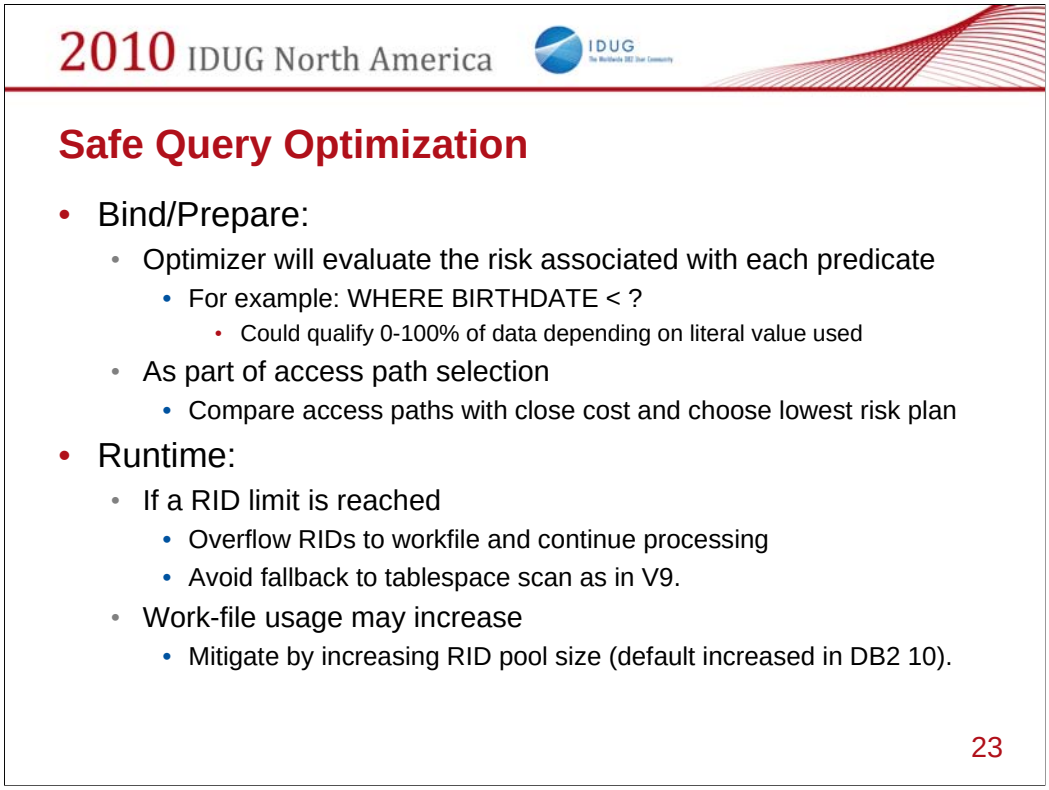
• etc.

In all, the plan picked by purely cost-based optimization lacks the robustness to prepare for various scenarios on some queries. Line item 868, Safe Optimization has the goal to:

• Generate safe and robust access paths.

• Detect and correct query execution exceptions.

• Introduce autonomic behavior that will allow DB2 to minimize the impact of bind-time unknowns.



Functional Description

This line item is designed to provide safe and robust plan during access path selection and plan execution. At bind time, it is done by extending current cost-based optimization with more statistics and more intelligence. At execution time, it is done by improving the current exception handling.

3.40.2.1. Cost Uncertainty

To overcome bindtime unknowns, query optimization has to make assumptions. Each assumption introduces cost uncertainty to related plans. And it has different impacts on different plans. This

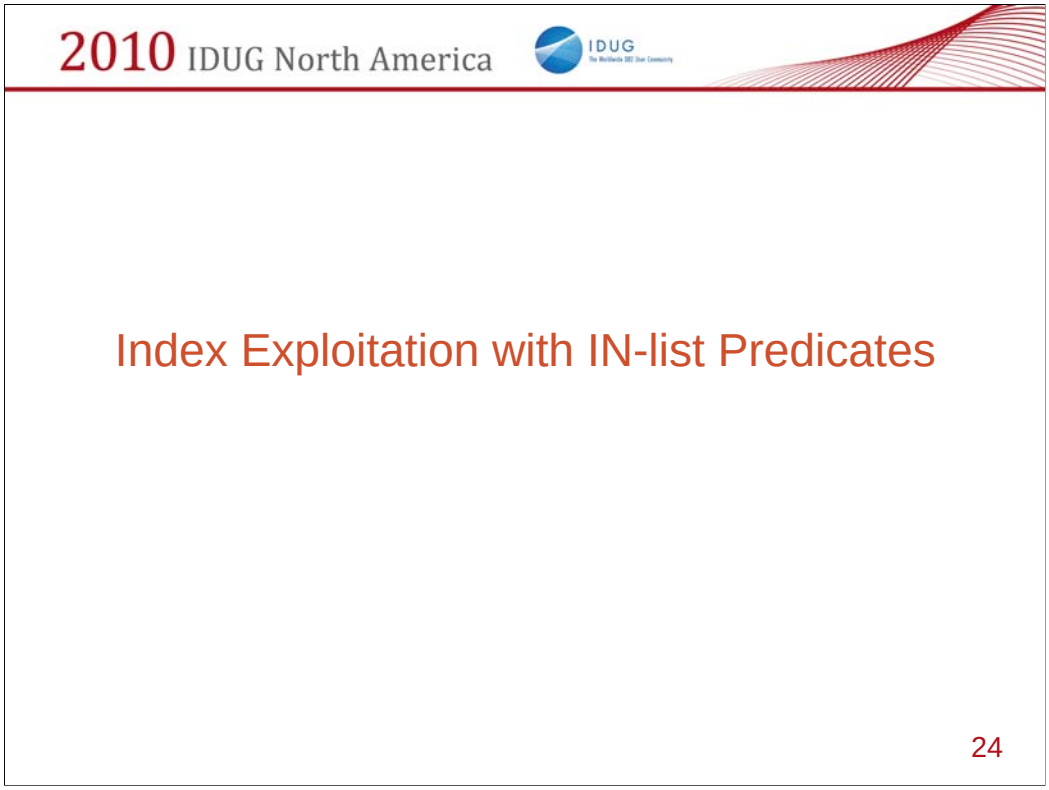
line item introduces a technique to quantify the cost uncertainty and incorporate it into cost-based optimizer during access path selection, especially during picking the right index and deciding the right join

sequence.

3.40.2.2. RID access

RID access, including List prefetch and Hybrid Join(HBJ), becomes a performance challenge when the RID-pool overflows. When it happens, RID access falls back to tablespace scan and it loses all index filtering. In attempting to prevent RID pool overflow, query optimization has introduced threshold checking logic at bindtime. However, it still can not completely avoid plan fallback. Besides that, it could block some queries from exploiting RID access.

This line item is designed to address RID access issue. At bindtime, query optimization allows RID access as long as it is the cheapest plan. All existing thresholds of RID access will be removed. At runtime, RID list is processed incrementally when it approaches RID pool overflow, so that RID access never needs to fall back to a tablespace scan (V9). In some cases, workfile usage increase is expected.



Line Item 904, Aggressive view merge, and index exploitation with IN-list predicate aim to improve performance for complex queries involving IN-list predicates, table expressions, views, subqueries, and outer

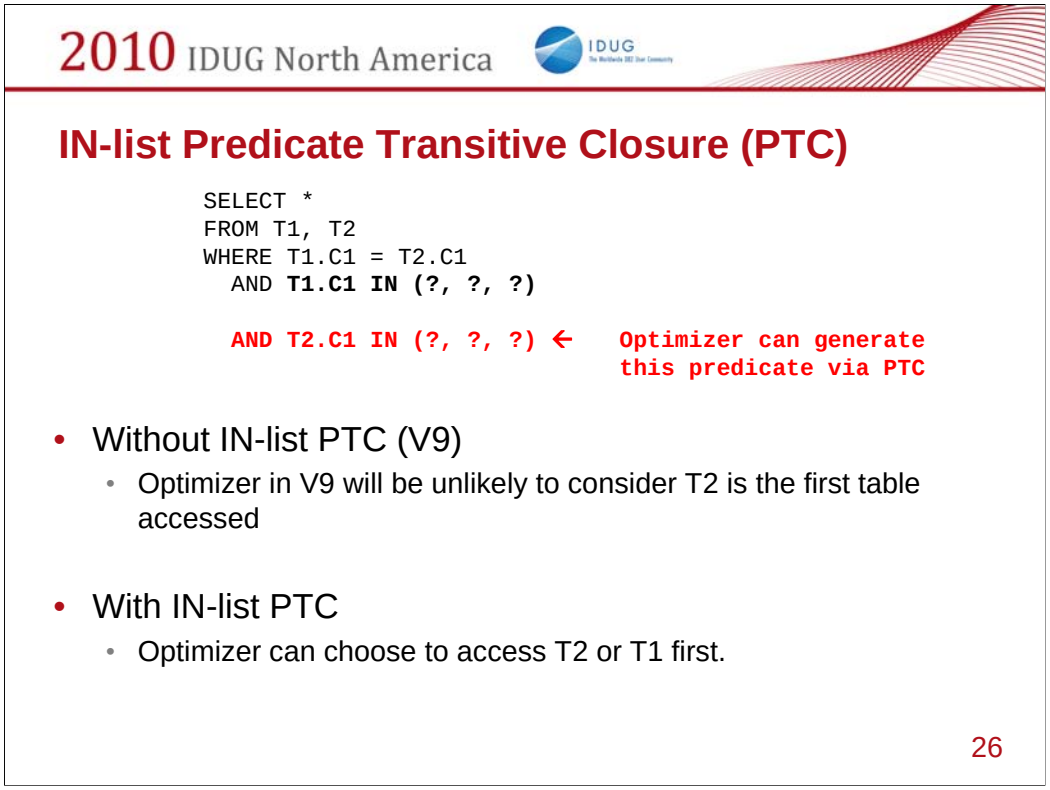
join etc.

Index Exploitation with IN-list Predicates

- For IN-list predicates, DB2 V10 adds
  - Transitive closure support for IN-list predicates
  - Matching on multiple IN-list predicates via single index access
  - List prefetch support
- IN-list predicates used extensively in some workloads
  - For example: SAP R/3

25

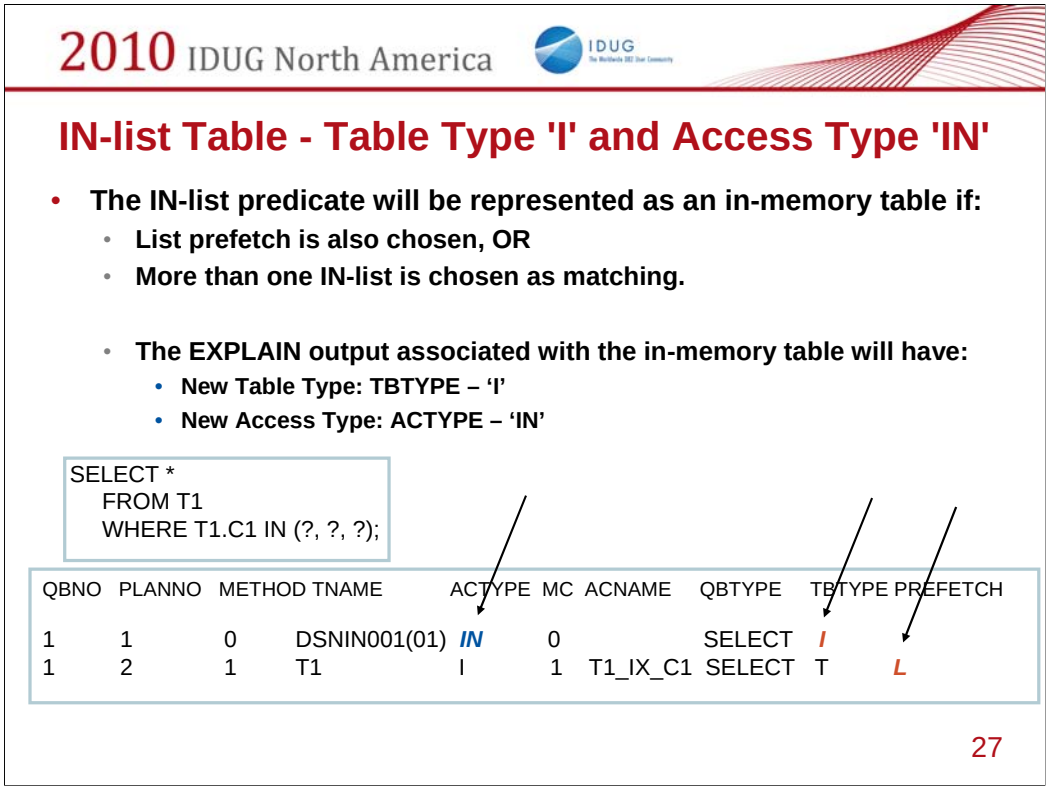
For IN-list predicates, we add support for matching on multiple IN-list predicates via single index access, and list prefetch. Transitive closure support for IN-list predicate is also included.



Even If there is an index on T2(C1), if there is no transitive closure for IN-list predicate, Optimizer normally will not consider using this index when T2 is the outer table.

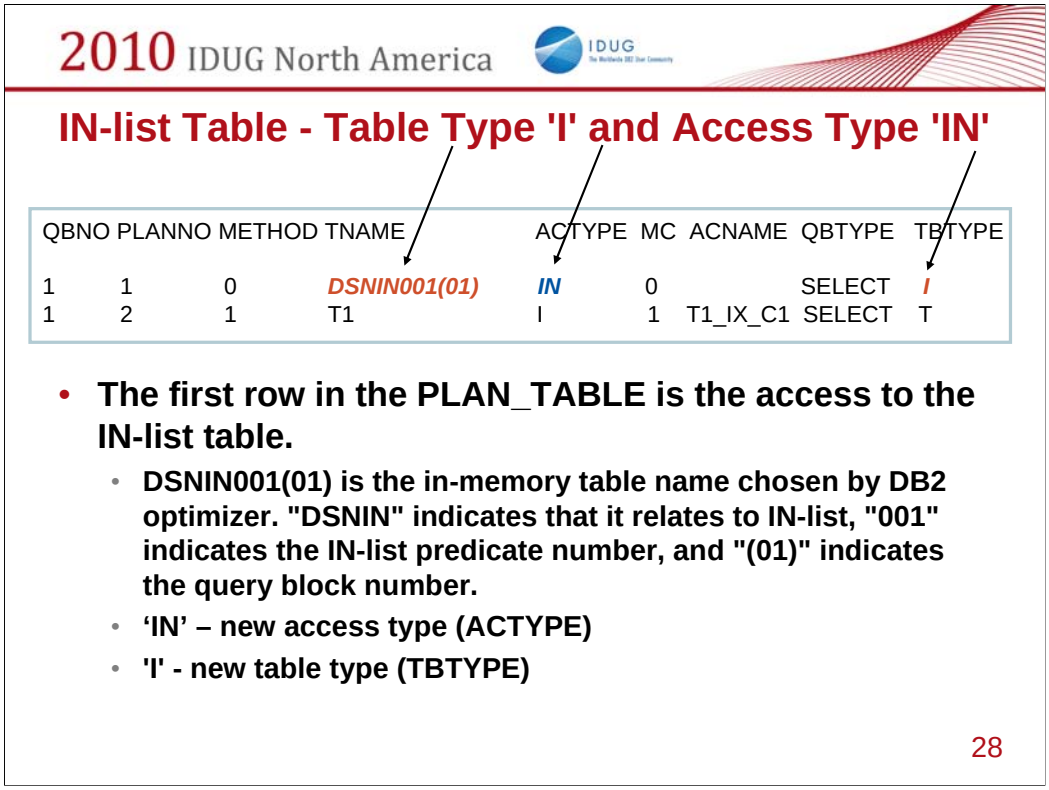
When T2 is the inner table, a sort merge join will normally not be, considered as a good access path because of lacking local predicate to reduce inner work file size.

Both these issues are addressed with transitive closure.



If IN-list predicate is selected as the matching predicate, it will be accessed as an in-memory table.

In the EXPLAIN output in the PLAN_TABLE, this access to the in-memory table is associated with a new table type 'I' and new access type 'IN'.



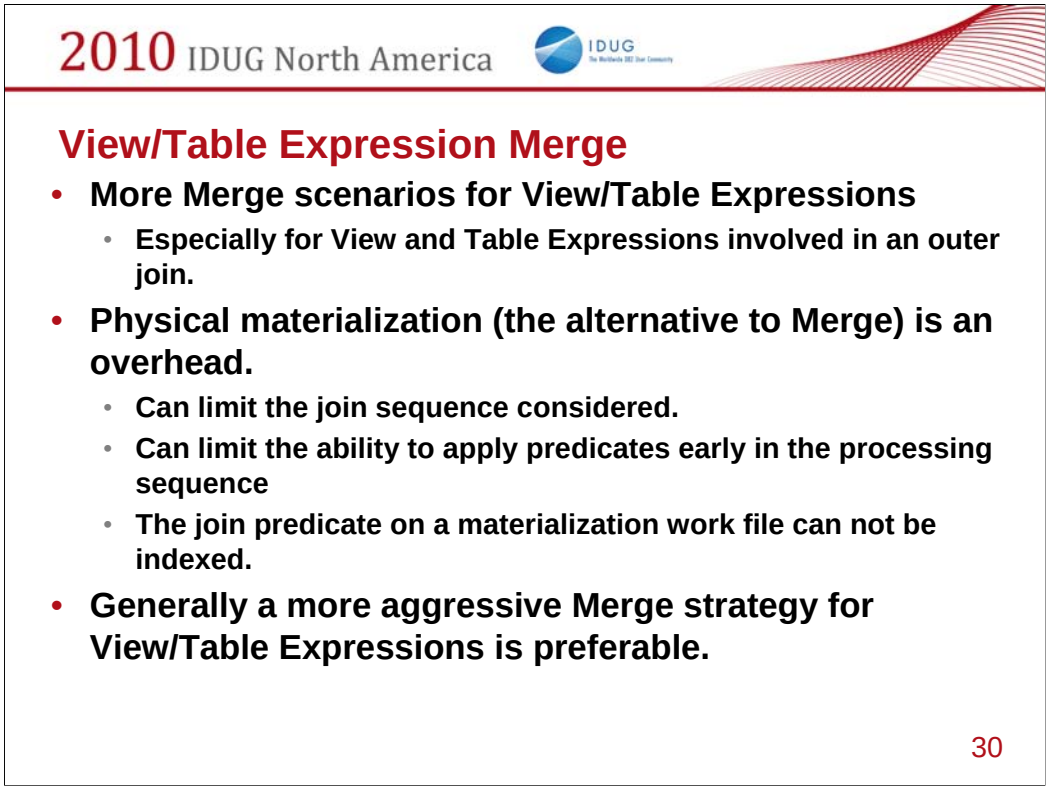
If IN-list predicate is selected as the matching predicate, it will be accessed as an in-memory table.

In the EXPLAIN output in the PLAN_TABLE, this access to the in-memory table is associated with a new table type 'I' and new access type 'IN'.

DSNIN001(01) is the in-memory table name named by DB2 optimizer. The IN-list table name will use a simple convention: "DSNIN" indicates that it relates to IN-list, "001" indicates the IN-list predicate number, and "(01)" in the middle indicates the query block number.

Line Item 904, Aggressive view merge, and index exploitation with IN-list predicate aim to improve performance for complex queries involving IN-list predicates, table expressions, views, subqueries, and outer

join etc.

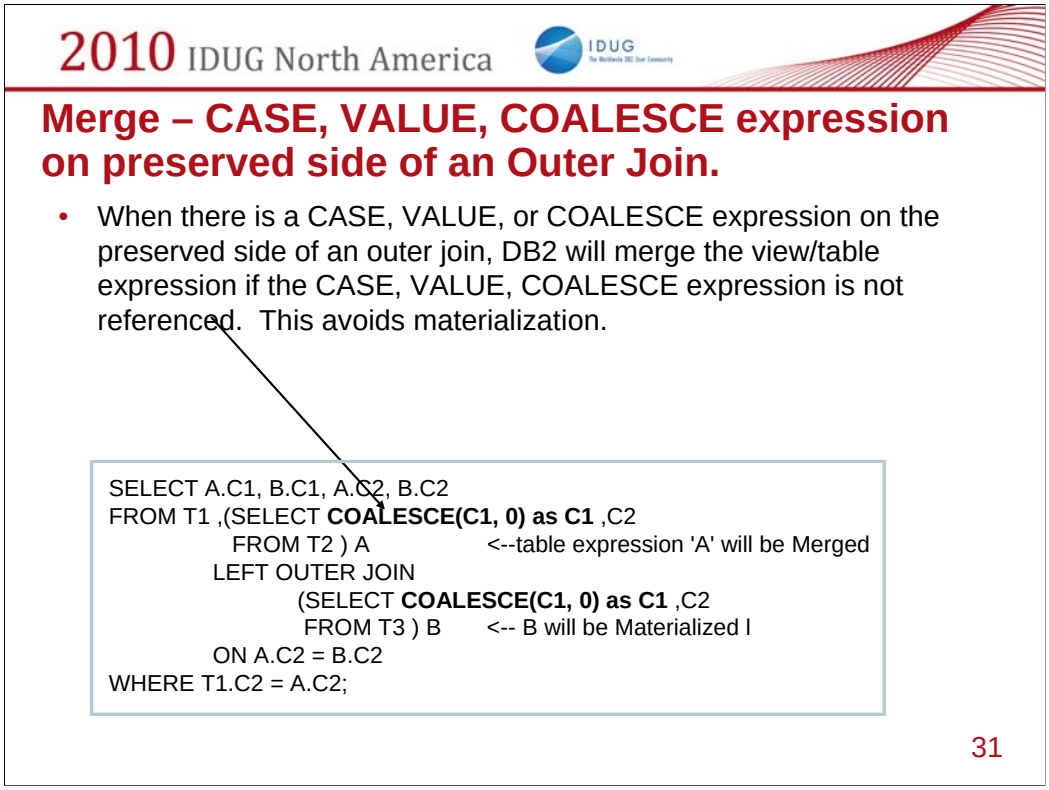


This version 10 section documents more Merge scenarios for View/Table expression, especially for View and Table Expressions involved in outer join. Physical materialization itself is an overhead, in addition, it also limits join sequence considered. Plus, the join predicate on materialization work file is not indexable. So normally more aggressive merge for View and Table Expression is always preferable.

In DB2 V9 the word ‘materialize’ is defined as:

“The process of putting rows from a view or nested table expression into a work file for additional processing by a query.”

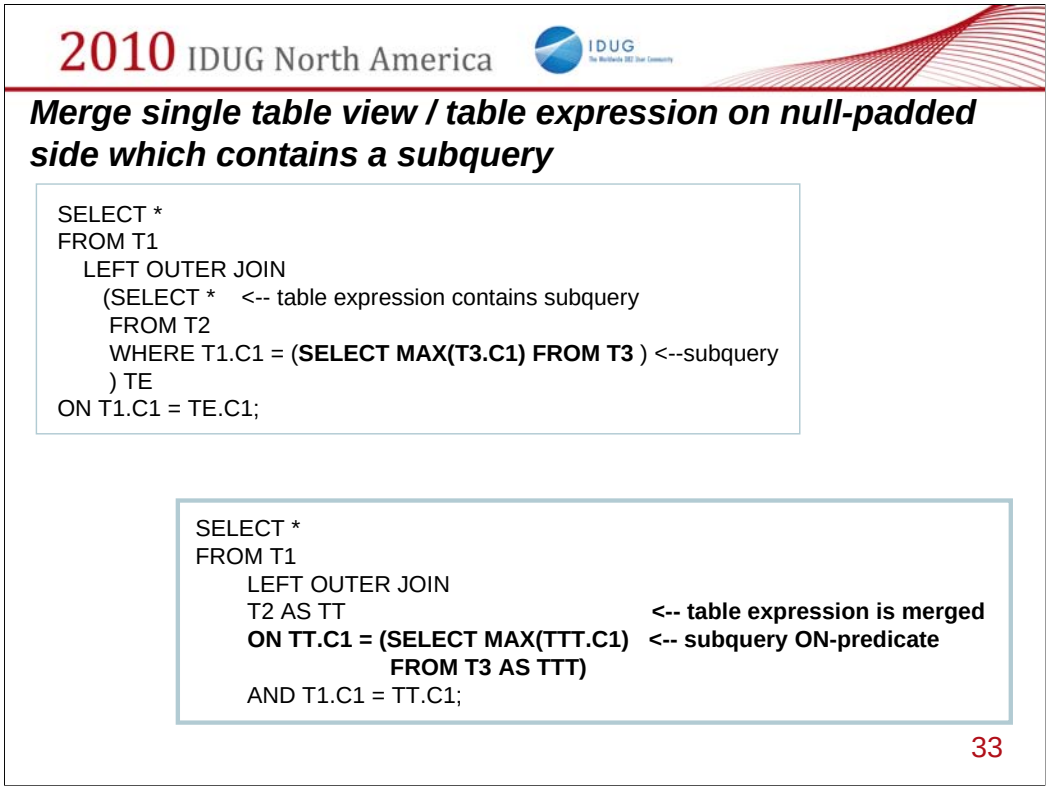
In DB2 V10, there are additional areas where can be avoided.



When there are CASE, VALUE, COALESCE expressions on the preserved side of an outer join, DB2 will be enhanced to merge the view/table expression if the CASE, VALUE, COALESCE expression is not referenced.

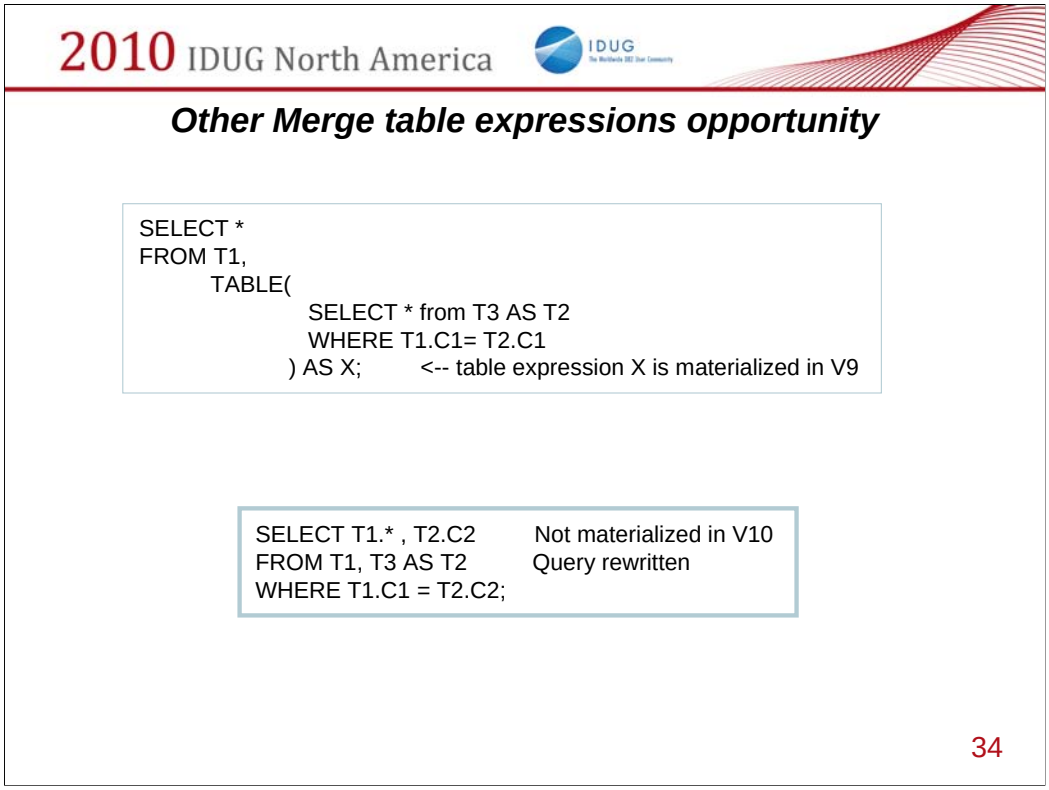
Merge single table view / table expression on null-padded side which contains a subquery

- For left/right outer joins, after query transformation, DB2 will allow subquery predicates in the on-clause.
- DB2 can Merge certain single table views / table expressions on null-padded side which contains a subquery.
- Performance may be improved due to materialization avoidance.



For left/right outer joins, after query transformation, DB2 will allow subquery predicates in the on-clause for LEFT / RIGHT outer join. Thus DB2 could merge certain single table views / table expressions on null-padded side which contains a subquery. These views and table expressions must only contain a reference to a single table. DB2 will do so by converting the subquery predicate to a "before join" predicate.

When the table in the table expression is very large, performance will be improved due to lack of materialization.



In the V10 FPS there is reference to 'sideway reference'. The term sideway reference appears in the V8 documentation. In neither place is it defined. Following is the documentation from the V10 FPS.

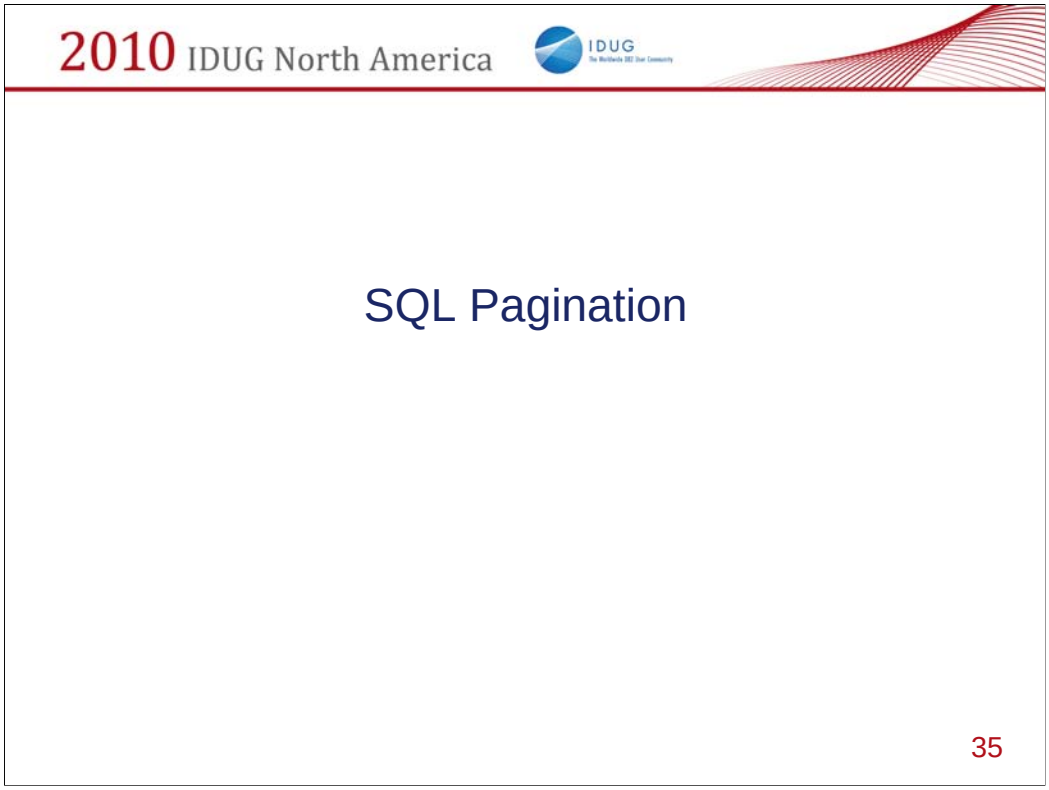
Merge table expressions that contain sideway references

DB2 will try to merge table expression that contain a sideway reference

```
SELECT *
FROM T1,
   TABLE(
         SELECT * from T3 AS T2
         WHERE T1.C1= T2.C1
        ) AS X;          <-- table expression X is materialized due to side way reference
```

With the enhancement in this line item, DB2 will not materialize the table expression. Instead, the table expression will be merged into the parent query block, with the query being rewritten as follows:

```
SELECT T1.* , T2.C2
FROM T1, T3 AS T2
WHERE T1.C1 = T2.C2;
```



LI901

Data-dependent pagination is the term used to address an application's need to access part of DB2 result set based on a logical key value. For example, application accesses a phone book sorted by last name and first name, however, application is only interested in retrieving the records from a certain name, 'John Smith'. Pagination support in V 10 addresses the performance issues for this requirement in V9.

SQL Pagination targets 2 classes of OR queries:

- Cursor scrolling SQL
  - Retrieve next n rows
  - Common in COBOL/CICS and any screen scrolling application
- Complex OR predicates against the same index
  - Common in SAP
- In both cases:
  - The OR (disjunct) predicate refers to a single table only.
  - Each OR predicate can be mapped to the same index.
  - Each disjunct has at least one matching predicate.

In the next visual, we will go from this generalized case to some specific examples.

Simple scrolling – Index matching and ORDER BY

- Scroll forward through the PHONEBOOK to obtain the next 20 rows from the current position – JONES, WENDY
  - Assumes index is available on (LASTNAME, FIRSTNAME)
- WHERE clause may appear as:

```
WHERE (LASTNAME='JONES' AND FIRSTNAME>'WENDY')
   OR (LASTNAME>'JONES')
ORDER BY LASTNAME, FIRSTNAME;
```

```
WHERE ((LASTNAME='JONES' AND FIRSTNAME>'WENDY')
   OR (LASTNAME>'JONES')
  AND (LASTNAME >= 'JONES'))
ORDER BY LASTNAME, FIRSTNAME;
```

← Extra predicate required in V9 for matching index access

37

The new method is to convert this OR predicate into a range list with two ranges. Therefore, there will be at most 2 index probes, the 1st probe is for LASTNAME='JONES' and FIRSTNAME>'WENDY'.

All remaining 'JONES' will be scanned through the PHONEBOOK table until there are no more rows that qualify the 1st probe predicates or the application stops fetching. These rows appear in the index in ascending order which satisfies the ORDER BY ordering and thus there is no requirement to sort the rows.

Multiple OR ranges against same index

Assume an index on (LASTNAME, FIRSTNAME).

Given the following complex WHERE clause:

```
WHERE (LASTNAME='JONES' AND FIRSTNAME='WENDY')
OR (LASTNAME='SMITH' AND FIRSTNAME='JOHN');
```

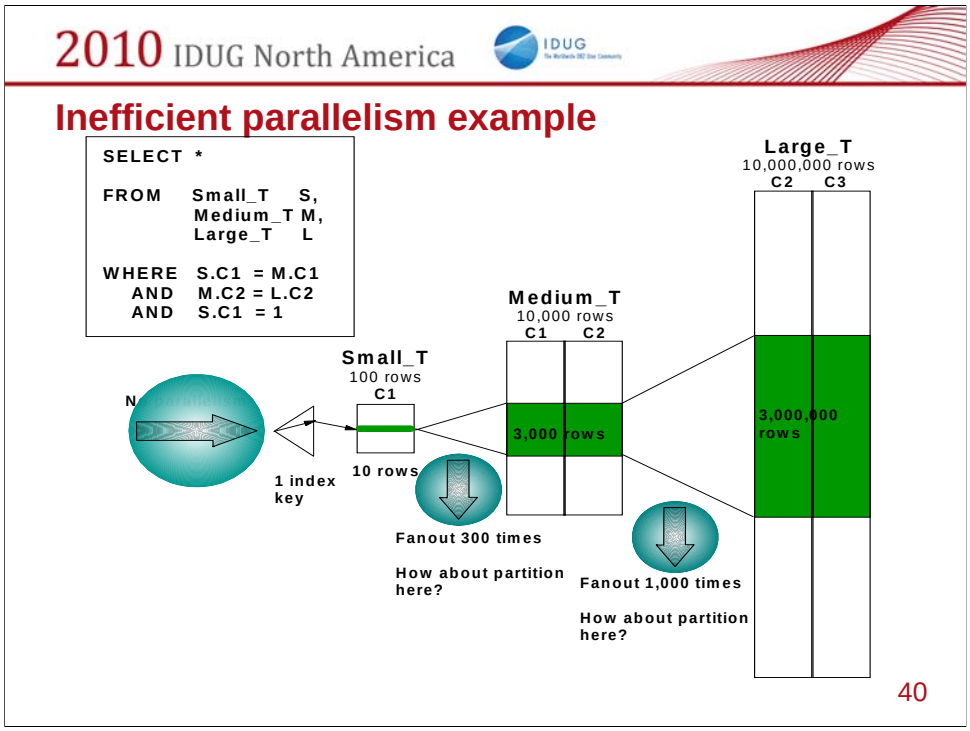
SQL pagination will allow single matching index access for this type of OR condition – without list prefetch.

V9 requires multi-index access with list prefetch for matching index access.

38

The new method is to convert this OR predicate into a range list with two ranges. There will be 2 index probes, the 1st one is (LASTNAME='JONES' AND FIRSTNAME='WENDY'), the 2nd one is (LASTNAME='SMITH' AND FIRSTNAME='JOHN'). Both index probes have two matching columns on the index.

Parallelism Enhancements



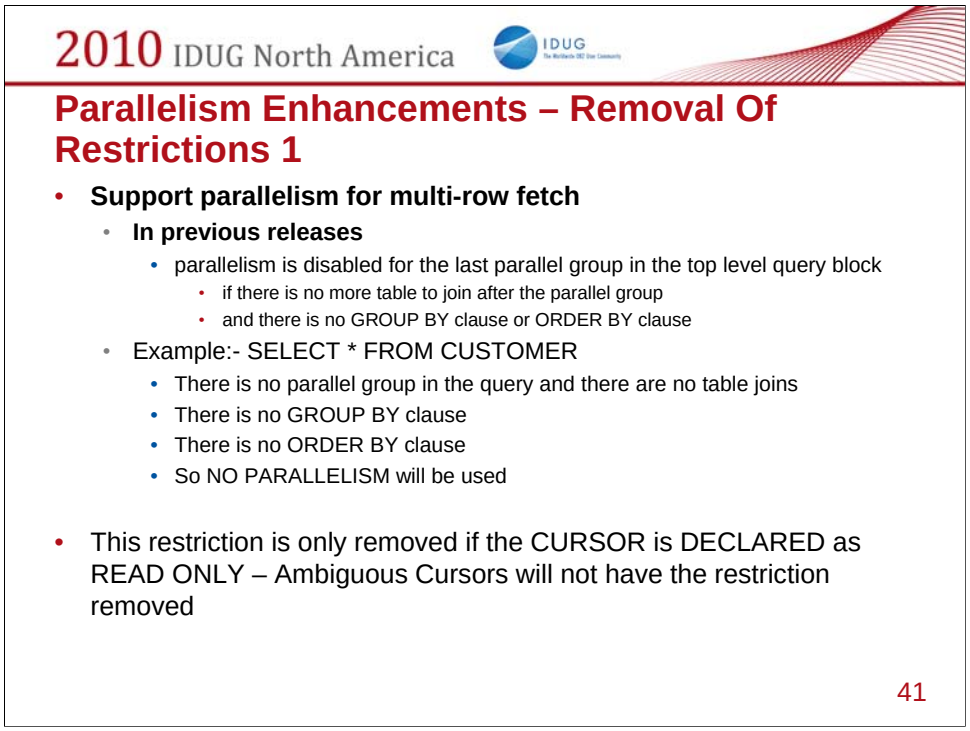
This example shows a join of 3 tables,Partition on leading table but the leading table is small and is not suitable for partitioning as it has a limited number of index keys

Limited number of inlist elements

Limited number of pages

Leading table is workfile

Big fan out after join



In previous releases, when multi-row fetch is used, parallelism is disabled for the last parallel group in the

top level query block if there is no more table to join after the parallel group and there is no GROUP BY

clause or ORDER BY clause. For example, for a simple query SELECT * FROM TABLE, if multi-row fetch is

used, then parallelism is disabled. This restriction forces customer to choose between multi-row fetch and

parallelism.

DB2 will remove this restriction when the cursor is read only. The restriction still exists for

an ambiguous cursor.

Parallelism Enhancements – Removal Of Restrictions 2

- Allow parallelism if a parallel group contains a work file
  - DB2 generates temporary a work file when view or table expression is materialized
  - This type of work file can not be shared among child task in previous releases of DB2, hence parallelism is disabled
  - DB2 10 will make the work file shareable
    - only applies to CP mode parallelism and no full outer join case

2010 IDUG North America

Parallelism Enhancements - Effectiveness

- Previous Releases of DB2 use Key Range Partitioning
  - Key Ranges Decided at Bind Time
  - Based on Statistics (low2key, high2key, column cardinality)
  - Assumes uniform data distribution
  - If the Statistics are out of date or data is not uniformly distributed what happens to performance

43

The problems with key range partitioning after sort composite

The key ranges are decided at bind time by optimizer based on statistics (low2key, high2key, and column

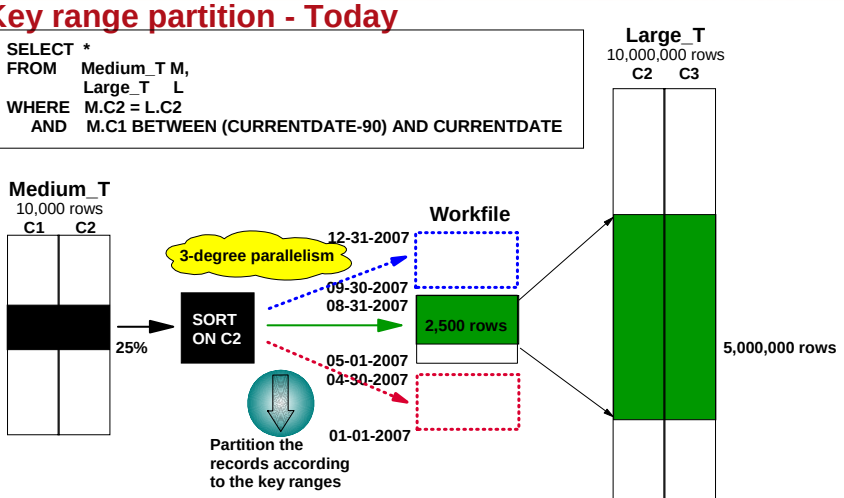
cardinality) and the assumption of uniform data distribution within low2key and high2key. This makes

DB2 too dependant on the availability and accuracy of the statistics. Also the assumption of uniform data

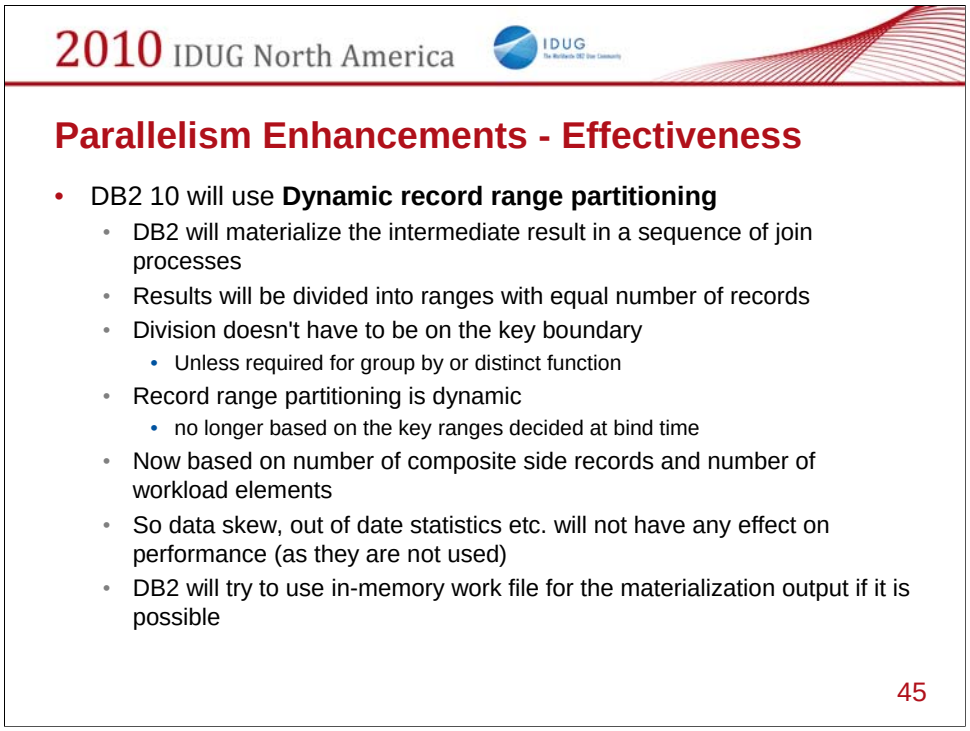
distribution does not always stand. DB2 is too vulnerable to performance disasters with key range partitioning.

Key range partition - Today

```
SELECT  *
FROM    Medium_T M,
        Large_T    L
WHERE   M.C2 = L.C2
  AND   M.C1 BETWEEN (CURRENTDATE-90) AND CURRENTDATE
```



M.C1 is date column, assume currentdate is 8-31-2007, after the between predicate is applied, only rows with date between 06-03-2007 and 8-31-2007 survived, but optimizer chops up the key ranges within the whole year after the records are sorted :-(



What is dynamic record range partitioning and why is it good?

DB2 will materialize the intermediate result in a sequence of join process, and the results

will be divided into ranges with equal number of records. This division doesn't have to be on the key

boundary unless it is required for group by or distinct function. Record range partitioning is dynamic because

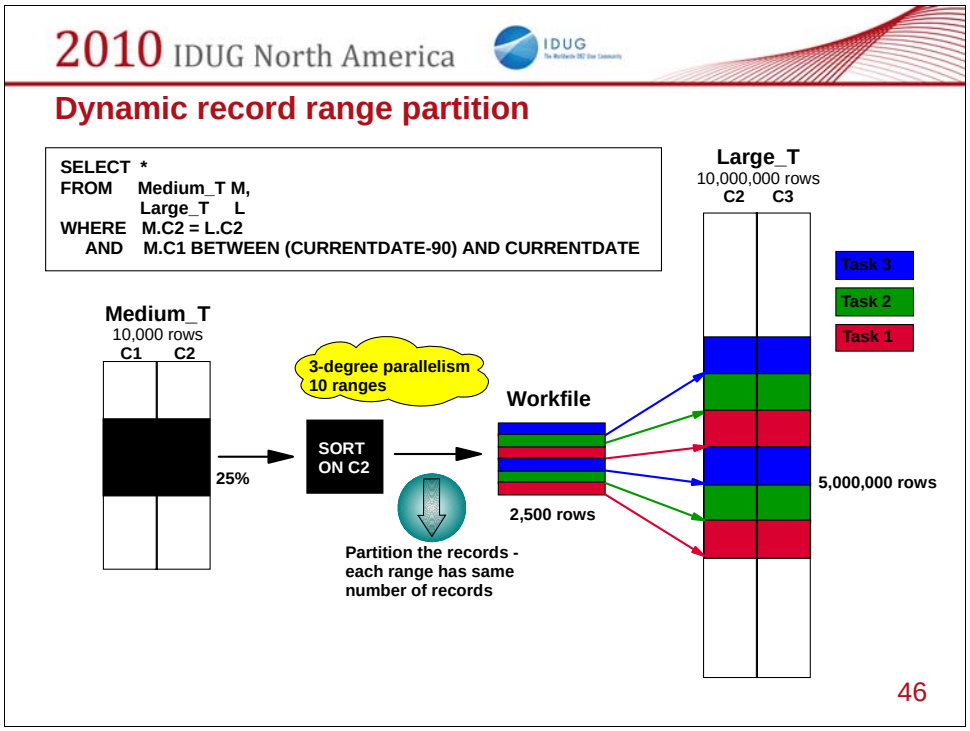
partitioning is no longer based on the key ranges decided at bind time, instead, it is based on the

number of composite side records and the number of workload elements. All the problems associated

with key partitioning, such as the limited number of distinct values, lacking of statistics, data skew or data

correlation, are bypassed and for sure the composite side records are distributed evenly.

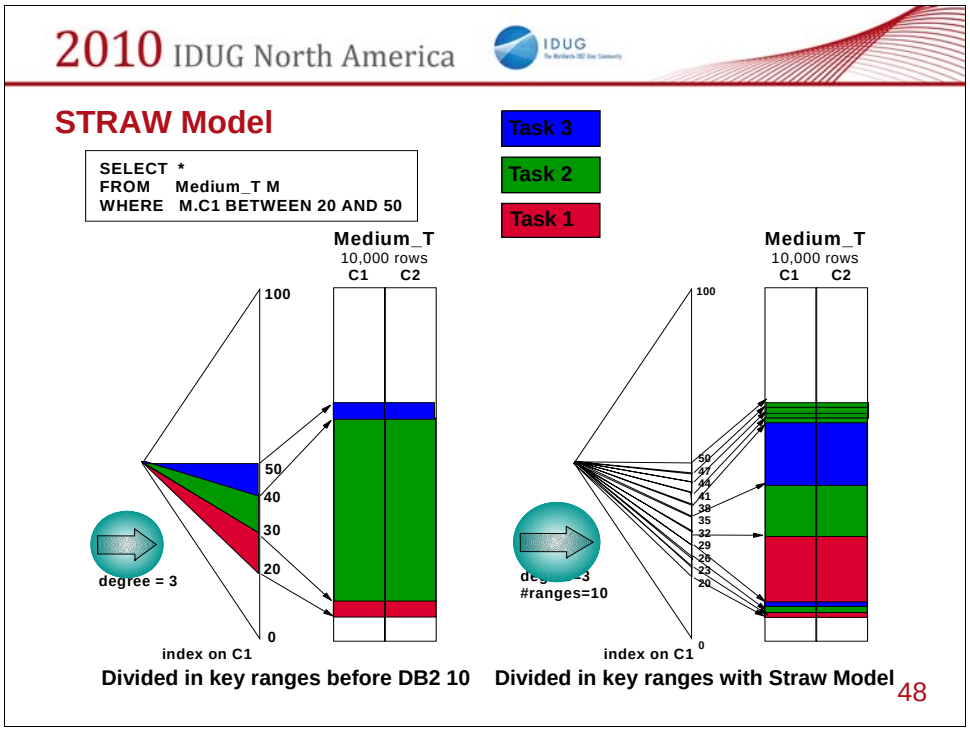
Moreover, DB2 will try to use in-memory work file for the materialization output if it is possible.



Partitioning now takes place on the work file after it is sorted – that way each parallel task will have the same number of rows to process.

Parallelism Enhancements - Effectiveness - Straw Model

- Previous releases of DB2 divide the number of keys or pages by the number representing the parallel degree
  - One task is allocated per degree of parallelism
  - The range is processed and the task ends
  - Tasks may take different times to process
- DB2 10 will use the Straw Model workload distribution method
  - More key or page ranges will be allocated than the number of parallel degrees
  - The same number of tasks as before are allocated (same as degree)
  - Once a task finishes it's smaller range it will process another range
  - Even if data is skewed this new process should make processing faster



The degree of parallelism is 3 – in the old method the key ranges are divided into 3 tasks but the middle task will take the longest time to process as it has far more rows to process. In the Straw model the key ranges are split into 10 even though the degree of parallelism is still 3. The 3 tasks are allocated the smaller key ranges and when each one finishes it will process another key range. In the old method the second task processed most of the rows but in the Straw Model all 3 tasks will share the work making the query run quicker.

Session Code: B02

What's new from the optimizer in DB2 10 for z/OS?

Terry Purcell
tpurcel@us.ibm.com