



IBM Software Group

# DB2 for z/OS Getting the Most From SQL Optimization Hints, Tips and Best Practices

*Yoichi Tsuji, Silicon Valley Laboratory*



@business on demand.

# Disclaimer

© Copyright IBM Corporation 2008. All rights reserved.  
U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

**THE INFORMATION CONTAINED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION CONTAINED IN THIS PRESENTATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED. IN ADDITION, THIS INFORMATION IS BASED ON IBM'S CURRENT PRODUCT PLANS AND STRATEGY, WHICH ARE SUBJECT TO CHANGE BY IBM WITHOUT NOTICE. IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, THIS PRESENTATION OR ANY OTHER DOCUMENTATION. NOTHING CONTAINED IN THIS PRESENTATION IS INTENDED TO, NOR SHALL HAVE THE EFFECT OF, CREATING ANY WARRANTIES OR REPRESENTATIONS FROM IBM (OR ITS SUPPLIERS OR LICENSORS), OR ALTERING THE TERMS AND CONDITIONS OF ANY AGREEMENT OR LICENSE GOVERNING THE USE OF IBM PRODUCTS AND/OR SOFTWARE.**

IBM, the IBM logo, ibm.com, DB2, and z/OS are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at [www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml)



# Agenda

- *Short introduction of SQL optimization in DB2 for z/OS*
  - ▶ *Factors in access path selection*
- *Stabilizing access paths*
  - ▶ *Understanding the workload*
- *Influencing access path selection*
  - ▶ *Tuning the optimizer*
- *Future visions to get the best query performance with stable access path selection*



## Our goals – a level setting

- Maintaining good query performance is an important part of reducing TCO
- DB2 for z/OS provides the leading edge SQL optimization technologies
  - ▶ Tradition from the era of 'System R', the long history of development and service
  - ▶ Accumulated on IBM's experiences with the customer base of the world largest data base users on the mainframe (System 370 to System z)
- IBM's main themes of on-going development of optimizer
  - ▶ More efficient SQL and data base operations of DB2 for z/OS (not just bind time optimization)
  - ▶ Better stabilizing the access path selection
  - ▶ More features and tools to understand and tune the access paths easier



## Short introduction of DB2 for z/OS query optimization

- ***Basic flow of SQL optimization in DB2 for z/OS***
- ***Behind the scene – what the optimizer does***
- ***Cost based access path selection***
- ***How the costing parameters are determined***
- ***Why statistics are important***
- ***Other factors influencing the access path selection***



# Basic flow of SQL (query) optimization

- Steps in compilation of an SQL statement
  - ▶ Query semantics analysis and transformation
  - ▶ Access path generation and selection
  - ▶ Parallelism optimization
  - ▶ Plan generation based on the selected access path



# Query optimization – behind the scene

- Based on the SQL semantics and the imbedded ‘logic’
  - ▶ Main techniques in the internal query transformation (rewrite)
    - ▶ Predicate transitive closures, pushdown and bubble-up
    - ▶ Outer join reduction (full outer join to left/right outer join to inner join)
    - ▶ View and table expression merge (or materialization)
    - ▶ Union distribution and query block pruning
    - ▶ Parallel groups (based on the mode of parallelism operation)
- Based on heuristics
  - ▶ Subquery to join transformation → more cost based in DB2 9
  - ▶ Some index selection criteria (e.g., unique vs. non-unique indexes)
  - ▶ Star join detection
- Based on cost estimation – access path selection
  - ▶ Index selection (including considerations on sort avoidance)
  - ▶ Join types (nested loop, sort merge, hybrid)
  - ▶ Join sequence
  - ▶ Determination of the parallelism plan (especially in DB2 9)



## Cost based access path selection

- Evaluation of the 'elapsed time' ← CPU and I/O wait time
  - ▶ A plan with the minimum elapsed time wins
- Cost evaluation based on estimating the 'access footprint' and the 'access pattern' to data for the CPU and I/O costs
  - ▶ Simulates the query execution
    - ✓ How many GETPAGES and how many rows to be processed ?
    - ✓ How efficient the prefetches are, and how many synchronous I/O's ?
- Most influential parameters in the cost estimation
  - ▶ Selectivity of the predicates (on tables and on indexes)
  - ▶ Access 'order' to the objects
    - ✓ Particularly on the inner table of a join
    - ✓ Is an index accessed in order or not ? Is the data accessed in order or not (cluster ratio if the table rows are accessed via an index) ?
    - ✓ Is a sort needed, or can it be avoided ? Which is more efficient ?
  - ▶ Access density (or sparsity) to the objects



# How the costing parameters are determined

- Statistics, statistics, statistics, ...
  - ▶ Quantifying the costs to compare possible plans and select the best
- Major statistics referenced from the catalog tables
  - ▶ Statistics on a table (SYSIBM.SYSTABLES)
    - Pages, cardinality
  - ▶ Statistics on an index (SYSIBM.SYSINDEXES)
    - Levels, leaf pages, full-key cardinality
    - Cluster ratio
    - DataRepeatFactorF ← DB2 9 (supplementing the cluster ratio)
  - ▶ Statistics on a column (SYSIBM.SYSCOLUMNNS)
    - Column cardinality, LOW2KEY, HIGH2KEY
  - ▶ Distribution and correlation statistics (SYSIBM.SYSCOLDIST)
    - Single/multi-column frequencies and multi-column cardinalities
    - Histograms ← DB2 9



# Why statistics are important to query optimization

- Access path selection is “cost based” and the cost evaluation is based on “statistics” → to quantifying costs of possible paths for comparison.
- Without right statistics, optimizer has to use “ambiguous guesses”.

▶ For example,

- for a query:

```
SELECT * FROM Employee
WHERE LastName=? AND Dept=?
```

- if there are indexes on the table Employee

```
XDept on (Dept)
```

```
XName on (LastName)
```

which index will be a better choice to run this query faster ?



## Other factors influencing the access path selection

- The size of the bufferpools
  - ▶ BP hit estimation based on the VPSIZE → I/O cost estimation
  - ▶ Possible BP hits are evaluated when pages are accessed repeatedly.
- RIDpool size
  - ▶ Availability of list prefetch (sorted RID list access, including hybrid joins)
- MIPS ratio of the processor
  - ▶ The CPU cost estimation is based on the processor's MIPS
- The number of processors
  - ▶ Impacts the degree of CPU parallelism
  - ▶ Can impact the access path selection ← DB2 9 (and later)
- Indexes in pending state (dynamic prepare)
  - ▶ Indexes in pending states at prepare time of a dynamic query will not be picked.



## Stabilizing the access paths and keeping the query performance high

- **Questions ...**
  - ▶ *Why can access paths be changed when rebind ?*
  - ▶ *Queries started to run slower even if the access paths are the same*
- **Analyze and understand the workload and the data base schema**
- **How to stabilize the access paths and query performance**
  - ▶ *Review the queries, review indexes, ... → Any tuning possibility ?*
  - ▶ *Are there sufficient statistics ?*
  - ▶ *If the query performance is not satisfactory ...*
- **Saving and restoring access paths**
  - ▶ *Best practices*
  - ▶ *“Plan stability” feature in DB2 9 and beyond*



# Why can access paths be changed after REBIND ?

- On release migration – major influential factors
  - ▶ Changes in predicate processing
    - ✓ More predicates treated as stage 1 and indexable (V8 and DB2 9)
    - ✓ Extended support of partition pruning (V8 and DB2 9)
  - ▶ Introduction of new or extended access paths
    - ✓ Multi-column sort merge join (V8 CM)
    - ✓ Global optimization and generalized sparse index (DB2 9 CM)
  - ▶ Changes in the “cost model”
    - ✓ Finned PMR fixes, extended forward-fits of PTF’s in earlier releases
    - ✓ Enabling PTF’s, if they were disabled by default in earlier releases
  - ▶ Support of new functions (with changes in the database schema)
    - ✓ Index-only access support for varchar columns (V8 NFM with non-padded index)
    - ✓ Index-on-expression → a predicate with an expression to be indexable (DB2 9 NFM)



## Why can access paths be changed after REBIND ?

- Possible changes in statistics (if RUNSTATS is re-run before REBIND)
  - ▶ The size of a table and its column statistics can impact the selectivity estimation of predicates. The buffer pool hitting estimation could also be changed.
  - ▶ Index cluster ratio can change after many inserts, updates, deletes.
  - ▶ Other statistics can also be changed (index levels, etc.)
- Possible changes in the optimization logic after applying PTF's.
  - ▶ Changes in the statistics interpretation to better cope with certain predicate combinations, join conditions, etc.
    - ✓ Sometimes, more statistics are needed (e.g., distribution statistics).

→ In any case, changes “usually” work better.



## A query runs slower with the same access path

- A typical reason: The “actual” cluster ratio of some indexes have changed (not the statistics in the catalog) – if the query is returning about the same number of rows as before.
  - ▶ The actual clustering status of an index can change after many inserts, updates and/or deletes of rows.
  - ▶ If a “good” access path was based on a high index cluster ratio (statistics) and, if the real clustering has become lower, the access path might not be “good” any more.
  - ▶ An extra caution will be needed on the cluster ratio of a cluster index.
    - ✓ If the cluster index is used in access paths of many queries, a lower clustering can cause excessive synchronous I/O’s, especially for report generation queries.
    - ✓ If the actual cluster ratio becomes lower (say, 90% or lower than the CLUSTERRATIOF in SYSINDEXES), it will be a good idea to run REORG to restore the clustering.
    - ✓ Can be checked by running RUNSTATS on indexes with REPORT YES and UPDATE NONE, if updates in catalog is not desired (however, it will invalidate statements in dynamic statement cache referencing the index in the access paths).
- Another case: The actual distribution of data was changed.
  - ▶ Existing access paths might not be “good” any more.



# Understanding the queries and the workload

- The first step to achieve high performance of query execution
  - ▶ Some tuning opportunities can exist in the application or query level.
  - ▶ Also consider query level tuning if a query runs slow, for the access path stability.
- Characteristics of queries (a workload = a set of application queries)
  - ▶ Are there particular patterns in the queries ? For example,
    - ✓ Frequently used predicate patterns among the queries
    - ✓ Frequently appearing joins (with common join predicates)
  - ▶ Views and table expressions ? Subqueries ?
- Type of application / application architectures
  - ▶ General purpose vender package or simple application layers ?
  - ▶ Batch, OLTP, or BI / warehousing type queries ? Static or dynamic ?
  - ▶ Use of temporary tables, set operators (e.g., UNION or UNION ALL), ...
- Queries and the data base schema
  - ▶ Indexes and common predicates
  - ▶ Level of normalization (foreign – primary key relationship and predicates)



# Analyzing a query or a workload (1)

- EXPLAIN

- ▶ The basic tool to analyze the access path
- ▶ For critical queries, it is recommended to save the current access path information provided by EXPLAIN
  - ✓ Useful to compare the access path with the original when it changes.
  - ✓ Also useful when a use of optimization hint is considered to restore the original access path.
  - ✓ Valuable information for IBM service if a PMR is opened for performance regression.
  - ✓ In order to avoid the PLAN\_TABLE growing too big, the contents could be copied for an archive (e.g., tape).



## Analyzing a query or a workload (2)

- Optimization Service Center or “IBM OCS” (V8 and later)
  - ▶ Offered as part of DB2 Accessories Suite for z/OS (no charge) – Info and download at  

```
http://www-01.ibm.com/software/data/db2/zos/downloads/osc.html
```
  - ▶ A workstation tool for monitoring and tuning the SQL statements.
    - ✓ For a single query or a workload (a set of queries).
    - ✓ Largely improved visual explain to view and investigate access paths.
    - ✓ Formatted query reports with annotation.
    - ✓ Statistics Advisor to check missing statistics (optionally runs RUNSTATS).
    - ✓ Proactively monitors the health of SQL workloads and issues alerts when a statement exceeds exception thresholds.
  - ▶ More reference: Redbook, “IBM DB2 9 for z/OS: New Tools for Query Optimization“, SG24-7421-00 (now applicable to DB2 for z/OS V8).
- Optimization Expert or “IBM OE” is a superset of OSC (charged)
  - ▶ More advisory features, including index advisor.



## Analyzing a query or a workload (3)

- Check-points in query or workload analysis
  - ▶ Are the predicates stage 1 ?
  - ▶ Can non-Boolean term predicates be made stage-1 ?
  - ▶ Are there redundant (same and duplicated) predicates ?
  - ▶ Are the subqueries (in predicates) working efficiently ?
  - ▶ Are there views or table expression materialized ?
- Check-points for queries and data base schema
  - ▶ Are the common predicates covered by indexes ?
  - ▶ Are the index design allow as much matching predicates ?
  - ▶ Do the cluster indexes indeed provide high cluster ratio ?



# TUNING QUERIES

- Go one step ahead of the optimizer
  - ▶ When designing new applications
  - ▶ When some queries run slower
- Several topics are presented here
  - ▶ Often appear as real problems
  - ▶ Some have been covered by DB2, in V8 or 9

- General guidelines and more examples:

## Performance Monitoring and Tuning

- ✓ Up to V8, a part of “DB2 for z/OS Administration Guide”
- ✓ DB2 9, an independent manual

contains wide range of query writing techniques



# Tuning queries – Are the predicates stage 1 ?

- Some simple predicates might not be stage 1.

- ▶ Data type or length mismatch

- Examples

```
Dept = 'B34' where Dept is CHAR(3)
```

```
or Dept = :HV when :HV is defined as CHAR(5)
```

- ▶ The predicate is stage 2 (V7). However, with the arguments having the same length (same data type in this example),

```
Dept = 'B34'
```

```
or Dept = :HV3 when :HV3 is defined as CHAR(3)
```

- ▶ the predicate becomes stage 1, and indexable (if there is an index on Dept column).

- The conditions are relaxed in V8 (more predicates as stage 1).



## Tuning queries – Are the predicates stage 1 ?

- Some simple predicates might not be stage 1.
- Another example:
  - **:HV BETWEEN OrderDate AND ShipDate**
  - ▶ The predicate is stage 2. However, if it is rewritten as
    - **:HV >= OrderDate AND :HV <= ShipDate**
    - ▶ the predicates become stage 1, and indexable (if there is an index on columns OrderDate and ShipDate).
- Why not done by DB2 ? → A future consideration.



## Tuning queries – Can the predicates be stage 1 ?

- A complex case:

```
(T1.C11 = :HV11 AND T2.C21 = :HV21)
```

```
OR (T1.C12 = :HV12 AND T2.C22 = :HV22)
```

- ▶ The predicate is stage 2 and will be applied only after tables T1 and T2 are joined. With extra Boolean term predicates:

```
(T1.C11 = :HV11 OR T1.C12 = :HV12)
```

```
AND (T2.C21 = :HV21 OR T2.C22 = :HV22)
```

```
AND ( (T1.C11 = :HV11 AND T2.C21 = :HV21)
```

```
OR (T1.C12 = :HV12 AND T2.C22 = :HV22) )
```

- ▶ the added predicates can be applied to T1 and T2 before the tables are joined (the filtering can be applied earlier to make the join operation more efficient).
- ▶ In the above rewrite, the original predicate must remain, because the added predicates alone will return “more rows” (“weakened” predicates, or partial conjunctive normalization).
- The predicate rewrite of this type is supported by V8 and later (“local” predicates only)
  - ▶ V8 for star-join queries only
  - ▶ DB2 9 for general queries



# Tuning queries – redundant predicates

- If a predicate is duplicated, estimation error in the selectivity increases.
  - ▶ Can cause instability in access path selection.
- Example

```
AND Accout.CID = Client.CID
```

```
AND Accout.CID = Client.CID
```

```
AND Client.CID = ?
```

Avoid including duplicated predicates in a query

```
AND Accout.CID = Client.CID
```

```
AND Client.CID = ?
```



## Tuning queries – Correlated or non-correlated subquery (1)

- If a non-correlated subquery scans a large amount of data, a costly materialization can be avoided by converting it to a correlated form. This rewrite will be effective when
  - ▶ The parent table is small, or has high filtering local predicates.
  - ▶ The child table has an index on the correlated column and a high filtering matching index scan is possible with the added predicate.

- Example

```
SELECT * FROM Small_Table S
WHERE S.C1 IN (SELECT B.C1 FROM Big_Table B) ;
```

- ▶ The query is quivalent to

```
SELECT * FROM Small_Table S
WHERE EXISTS (SELECT 1 FROM Big_Table B
              WHERE B.C1 = S.C1) ;
```

- If the column B.C1 has a high column cardinality and at a leading position of an index on BigTable, the rewritten query will be very efficient.
- DB2 9 supports this type of query transformation with “global optimization”.



## Tuning queries – Correlated or non-correlated subquery (2)

- On the other hand, if the parent table is large, a correlated subquery attached to the parent table will be executed many times. In this case, “decorrelation” will be effective when
  - ▶ The decorrelated subquery is much smaller than the parent.
  - ▶ The parent table has an index on the correlated column.
  - ▶ By the decorrelation, materialization will occur but it is expected that the small work file will be scanned first then the large parent is joined.

- Example

```
SELECT * FROM Big_Table B
WHERE EXISTS (SELECT 1 FROM Small_Table S
              WHERE S.C1 = B.C1) ;
```

- ▶ The query is quivalent to

```
SELECT * FROM Big_Table B
WHERE B.C1 IN (SELECT S.C1 FROM Small_Table S) ;
```

- If the column B.C1 has a high column cardinality and at a leading position of an index on BigTable, the rewritten query will be very efficient (type “N” index access will be used).
- DB2 9 supports this type of query transformation with “global optimization”.



## Tuning queries – merge or materialize view or table expression

- Two way considerations on views or table expressions – merge or materialize
- (1) Can a materialized view or table expression be merged ?
  - ▶ A view or a table expression with “SELECT DISTINCT” or “GROUP BY” is materialized, in principle.
  - ▶ If such a view or a table expression is materialized and results in a large work file, can “DISTINCT” or “GROUP BY” be moved outside of the view or the table expression ?
  - ▶ If they can be merged, an extra overhead of materialization can be eliminated and potentially more efficient access path could be chosen.
- (2) Any advantage to materialize a view or a table expression ?
  - ▶ In contrast, there can be cases where, by pushing “SELECT DISTINCT” or “GROUP BY” into a view or a table expression, they can be applied before materialization and the resulted work file becomes small. The referencing query can take the advantage of joining the small work file.
  - ▶ This can be used as a technique to apply DISTINCT or GROUP BY early in the query processing by creating an artificial table expression to avoid causing a big “fan-out” before sorting for DISTINCT or GROUP BY in the original query (if the sort is needed).
  - ▶ DB2 9 supports “sparse index” on a materialized work file.



## Tuning queries – predicate push-down

- Simple predicates are “pushed down” into a materialized view or table expression where possible. Some predicates such as LIKE and complex predicates are not.
- Can a complex predicate be “pushed down” manually ?
- Example

```
SELECT *
FROM (SELECT REGION, YEAR, QTR, SUM(SALES)
      FROM SALES_TABLE
      GROUP BY REGION, YEAR, QTR) QTR_SALES
WHERE (QTR = 1 OR YEAR < '1999') ;
```

- ▶ The predicate can be moved into the table expression.

```
SELECT *
FROM (SELECT REGION, YEAR, QTR, SUM(SALES)
      FROM SALES_TABLE
      WHERE (QTR = 1 OR YEAR < '1999')
      GROUP BY REGION, YEAR, QTR) QTR_SALES ;
```

- The predicate can be applied within the table expression
  - ▶ If the sort for GROUP BY is needed, the sort will be more efficient with less number of rows.
  - ▶ The size of materialized work file will be smaller and require less space.
- V7 APAR PQ73454 to allow push down of IN-lists
  - ▶ In V7, disabled by default. A zparm INLISTP has to be set to enable the enhancement.
  - ▶ A short summary of INLISTP is described later



# Index design considerations

- General performance trade-off
  - ▶ Adding indexes incurs operational overheads
  - ▶ Adding a good index (if missing) can improve performance of query execution
- Is there a “good” index on a large table accessed by many queries, or by a query running very frequently, or by critical report generation (long running) queries ?
  - ▶ Providing matching keys with high filtering (high cardinality on the index key or keys) for a pattern of predicates commonly appears, especially for multiple EQUAL predicates
  - ▶ Providing the order of ORDER BY or GROUP BY so that the SORT can be avoided
    - ✓ Sort avoidance will depend on the cost evaluation → if the index cluster ratio is low, a sort can be more cost effective to avoid a large amount of synchronous I/O's caused by the index scan.
  - ▶ Possibility of index-only access
- Optimization Expert (IBM OE, offered with extra charge) will help analysis on a workload
  - ▶ Index Advisor



# STATISTICS

- Statistics is the key part of query optimization
  - ▶ Access path selection is based on the statistics to evaluate the costs of access paths
  - ▶ Right amount of statistics (necessary and sufficient) ?
    - ✓ Analyze the workload
    - ✓ IBM OSC Statistics Advisor will provide guidance (V8 and later)
- Strongly recommended to keep the statistics up to date
  - ▶ A trade-off exists
    - ✓ Frequent collection of statistics can change access paths on REBIND, especially for dynamically prepared statements. Also, the cost of running RUNSTATS can be high.
    - ✓ For a data base for transaction processing, a large amount of transaction records may be inserted to the tables daily. This can cause a shift of data distribution and an old access path might not work well.
- Out of date statistics will make diagnostics of access path issues hard.



# Statistics collection timing

- If the current query performance is stable and satisfactory,
  - ▶ Collection of statistics could be done less frequently.
- If some queries run slower
  - ▶ Identify the tables and indexes where statistics might have been changed.
  - ▶ Rerun RUNSTATS on the tables and indexes.
  - ▶ There might be columns where distribution statistics or multi-column cardinalities can help improving and stabilizing the access paths
  - ▶ Be sensitive to index cluster ratio, especially for cluster indexes (REORG ?)
- If a data base is very active, especially a workload mainly consisting of dynamic statements,
  - ▶ Collect statistics on the active tables and their indexes (monthly, weekly or even daily)
- If mass INSERT or LOAD is applied to a table (e.g., DW or MQT's)
  - ▶ Collect statistics on the table and its indexes each time when it is refreshed.



## Tips on statistics collection

- IBM generally recommends to keep statistics up to date
  - ▶ It also helps diagnosing an access path problem with the current status of the data base (especially when IBM service is contacted).
- If RUNSTATS is run on a table or a table space
  - ▶ Also collect statistics on all indexes
  - ▶ Not recommended to run RUNSTATS only on an index or a subset of indexes on a table
  - ▶ Inconsistent statistics can cause instability in access path selection
- RUNSTATS for V8 and later has options to collect distribution statistics on any combination of columns
- RUNSTATS options REPORT(YES) UPDATE(NONE) can be used to check the current statistics without updating the catalogs.
  - ▶ With the options, RUNSTATS invalidate statements in the dynamic statement cache referencing the objects.



# Queries still run slow after collecting enough stats

- Statistics are statistics
    - ▶ Do not necessarily represent the real distribution of data
    - ▶ Statistical inference do not necessarily capture the real data access patterns
  - Statistics are not applicable to certain situations
    - ▶ Predicates with host variables (parameter markers) or complex expressions
    - ▶ For example, for a predicate `ReceiveDate > :HV1`
      - ✓ The selectivity can not be accurately evaluated until the value of the host variable is known, or REOPT is used (otherwise, a 'guess').
  - Statistical inference errors can accumulate for complex queries
    - ▶ Data skews are hard to recognize under join predicates
- Is there any possibility to tune the query ?
- Consider the possibilities :
- Use optimization hint
  - Use options to influence the optimizer's behavior



# Saving and restoring access paths

## → Recommended process on a release migration

- Previous access plan will be lost by REBIND or BIND REPLACE.

- To save a copy the “current” access plan

- ▶ Use BIND into an alternative collection, instead of REBIND
- ▶ For example, with a package PROG1

```
BIND PACKAGE(NEWCOLL) COPY(OLDCOLL.PROG1) EXPLAIN(YES)
```

Or

```
BIND PACKAGE(NEWCOLL) MEMBER(PROG1) EXPLAIN(YES)
```

- Select the new collection using package lists

```
REBIND PLAN(APPLPLAN) PKLIST(NEWCOLL.*, OLDCOLL.*)
```

- Falling back to the “prior” access plan (using the above REBIND command)

```
FREE PACKAGE(NEWCOLL.PROG1)
```

- If precompile/compile of a program is needed, the above process will not work

- ▶ SQLCODE -805 or -818.
- ▶ Consider using PLAN HINTs for the access paths restoration
- ▶ A good practice is to keep the plan table entries with EXPLAIN(YES)

- Details: [http://www.db2now.com/purcell-Fuh\\_REBIND\\_IDUGspring2006.pdf](http://www.db2now.com/purcell-Fuh_REBIND_IDUGspring2006.pdf)

- The “plan stability” feature of DB2 9 automates the process.



## DB2 9 “plan stability” feature (1)

- PK52523 (DB2 9)
- New options for REBIND
  - ▶ REBIND ... PLANMGMT(BASIC)
    - ✓ Any previous copy is discarded
    - ✓ Current copy becomes “previous”
    - ✓ Incoming copy becomes “current”
  - ▶ REBIND ... PLANMGMT(EXTENDED)
    - ✓ In addition to BASIC (“previous” and “current”), “original” copy is saved.
    - ✓ If there is no “original” at REBIND, the current copy is saved as “original”
    - ✓ “Original” is saved once, never overwritten.
  - ▶ REBIND ... SWITCH(PREVIOUS) → restores the “previous” plan
    - ✓ Switches the “current” and “previous”
  - ▶ REBIND ... SWITCH(ORIGINAL) → restores the “original” plan
    - ✓ “Current” moves to “previous”, “original” becomes “current”
    - ✓ “Original” is unchanged (“current” = “original”) after this SWITCH.



## DB2 9 “plan stability” feature (2)

- DASD requirements
  - ▶ PLANMGMT(BASIC) requires (up to) 2x space in SPT01
  - ▶ PLANMGMT(EXTENDED) (up to) 3x space in SPT01
  - ▶ Similar overhead in SYSPACKDEP
- REBIND time
  - ▶ Depending on the application, 10~30% overhead
- A caution: Rebinding SQL statements are required by some PTF's
  - ▶ Fixing an incorrect output symptom, or ABEND.
  - ▶ Certain access paths might be disabled → previous access path may become invalid
  - ▶ Restoring an earlier access path can have a potential risk, if done in middle of a release life cycle.



## DB2 9 “plan stability” feature -- Future visions

- Usability, applicability and manageability
  - ▶ Central repository of access paths
  - ▶ Versioning
- Plan stability for dynamic SQL
  - ▶ Extend the benefit of static SQL
  - ▶ Dynamic SQL behaves like static SQL (as if there were “persistent dynamic SQL cache”)
  - ▶ Capture “critical” dynamic SQL statements based on “scoping”
- More flexible access path restoration for REBIND and BIND
  - ▶ REBIND → DB2 “attempts” to lockdown access paths for all statements in the package
  - ▶ BIND → DB2 “attempts” to lockdown access paths for statements that haven’t changed.



# Influencing access path selection

- ***Optimization hints***
  - ▶ *Optimizer “tries” to generate the given access path*
- ***OPTIMIZE FOR n ROWS***
  - ▶ *Can avoid SORTs*
- ***REOPT options***
  - ▶ *NONE, ALWAYS, ONCE and AUTO*
- ***Remark on “trick predicates”***
- ***Remarks on VOLATILE table***
- ***Special ZPARMs for optimizer***

→ ***Optimizer “persuasion” techniques***



# Optimization hints (1)

- An alternative to “saving and restoring the access plan”
- Save the EXPLAIN results (PLAN\_TABLE) → a good practice
  - ▶ Particularly for critical packages or queries
  - ▶ For possible use with optimization hints (or to compare with the new access paths)
- One way to restore good access path if query performance degrades, or to explore better access paths
- Useful for static SQL
  - ▶ Hard to use “ad hoc” dynamic queries (e.g., queries generated by an application system)
- SQLCODEs
  - ▶ +394 Hint was applied successfully
  - ▶ +395 A part or the entire hint was not applied
    - Optimizer was not able to use the hint because of some constraints in access path generation



## Optimization hints (2)

- Future direction → usability, applicability and manageability
  - ▶ A centralized repository of optimization hints.
  - ▶ A hint is applied by statement text (static or dynamic).
    - No need to change applications to use optimization hints.
  - ▶ The use of a hint can be disabled.
    - If a hint was used for a static statement and the hint is disabled, the package will be invalidated.
    - For a dynamic statement, it will be removed from the dynamic statement cache.



# OPTIMIZE FOR n ROWS (1)

- A part of the SQL syntax for SELECT statement
- Access path is optimized to return “n” rows quickly, rather than returning the full result set.
  - ▶ Optimizer assumes that the full result set is returned, in contrast to FETCH FIRST n ROWS ONLY.
- General impacts to access path selection
  - ▶ Depending on the specified “n” rows,
    - ✓ More likely to generate a “pipeline plan of nested loop joins”
    - ✓ Large sort will be avoided, including list prefetch (sorted RID list access)
  - ▶ OPTIMIZE FOR 1 ROWS
    - ✓ No sort, no list prefetch
  - ▶ Effective only in the top query block (not in subqueries)
  - ▶ Not effective if GROUP BY or ORDER BY sort is needed (if there is no index supporting GROUP BY or ORDER BY)



## OPTIMIZE FOR n ROWS (2)

- Effective when an application provides a buffer (outside of DB2)
  - ▶ Repeat fetches until it is filled, then process the fetched rows (asynchronous block fetch operation performed by the application).
  - ▶ Once the buffered rows are processed, next round of fetches.
- Effective when sorts are not desired
  - ▶ To reduce or eliminate sort merge joins, hybrid joins and/or list prefetch.
  - ▶ For a smaller “n”, only smaller sort or list prefetch will be used in the plan.
  - ▶ For “n=1”, no sort nor list prefetch.
- Major enhancement in V8
  - ▶ PQ87390 (UQ91022) Aug., 2004
  - ▶ Stabilized access path selection with performance improvements for queries with OPTIMIZE FOR n ROWS



# REOPT option (1)

- Why REOPT
  - ▶ Actual values of parameter markers or host variables are used in the optimization
    - More accurate estimation of selectivity (no guess by unknowns)
- REOPT – a BIND option
  - ▶ NONE      The default
  - ▶ ALWAYS    For static and dynamic SQL
    - ✓ The selectivity estimation is always most accurate.
    - ✓ For dynamic SQL, the benefit of caching is lost
    - ✓ For static SQL, the “predictability” of access path is lost
    - ✓ Re-optimization overhead for each execution
  - ▶ ONCE      (V8) For dynamic SQL with dynamic statement cache
    - ✓ Optimize once when the statement is first executed (at OPEN)
    - ✓ Values of parameter markers are used for optimization. The access path is saved in the global statement cache.
    - ✓ Effective to applications where the values used in optimization are “typical” for the repeated runs of the statement.



## REOPT (2)

- ▶ AUTO (DB2 9) For dynamic SQL
  - ✓ First execution of a statement is the same as REOPT (ONCE)
  - ✓ Re-optimize the statement when a significant change occurs in selectivity of a predicate(s) from the time of the last optimization.
  - ✓ Some overhead to compute and compare the predicates' selectivity (even when not re-optimized).
  - ✓ New access path only replaces the local copy (not the global statement cache).
- Future directions
  - ▶ REOPT ONCE and AUTO to support static SQL
  - ▶ Retain multiple access paths to minimize the optimization overhead for REOPT AUTO
    - For dynamic SQL → in the cache
    - For static SQL → in a repository (disk)



## “Trick predicates”

- Sometimes, one or more predicates not impacting the SQL semantics could be used to influence the access path selection.
- **Use the “trick” as the last resort** (when nothing else works to improve the performance of a query).
  - ▶ IBM generally recommends the use of an optimization hint, if possible.
- A typical example to avoid using an index
  - ▶ A dynamic query has a Boolean term predicate (CheckTime <= ?).
  - ▶ The access path uses an index on CheckTime (even with REOPT for a large data skew on the column).
  - ▶ In reality, the predicate is not selective and the index access is not efficient.
  - ▶ To avoid the predicate being indexable, a redundant term can be added:  
((CheckTime <= ?) OR (1=0)) → no more indexable
  - ▶ This does not change the meaning of the predicate.
- DB2 9 provides histogram statistics. With REOPT ONCE or AUTO, the problem of the dynamic query can be solved by collecting histogram statistics on the column.



## VOLATILE table (1)

- DB2 V8 introduced a keyword VOLATILE in the CREATE TABLE statement.
- The keyword VOLATILE indicates that the statistics are not trustable for the table.
  - ▶ Frequent insert, delete, update or LOAD will be applied to the table.
- On a VOLATILE table,
  - ▶ A strong preference is given to a “full matching” index scan by the optimizer (“more matching” next) regardless of the cost estimation.
  - ▶ At the same time, the access to the table rows in the order of index key is also enforced.
- Consequently, list prefetch is disabled on a VOLATILE table.
  - ▶ List prefetch accesses to the table rows in the order of sorted RIDs (i.e., the physical RID order). This will not guarantee the access in the index key order.



## VOLATILE table (2)

- List prefetch can be a good access type with an index having a relatively low cluster ratio.
- When designing a table as a candidate for VOLATILE, and if there is a possibility of having list prefetch on the table as a good access type
  - ▶ Multiple indexes will be created on the table and some likely have low cluster ratio (other than the cluster index).
  - ▶ Some keys to access the table (particularly join keys) are on the (likely) low clustering indexes.
- It is recommended not to use the VOLATILE keyword when creating the table (unless it is “needed”).
- If the table will have only the cluster index that covers the main access keys to the table, VOLATILE can be a good option.



## VOLATILE table (3)

- An alternative (without VOLATILE keyword)
  - ▶ For a table that can change its statistics very frequently, and it is desired to have a matching index access to the table, allowing list prefetch,
    - ✓ Set a small value to a ZPARM NPGTHRSH (if not set already)
      - For example, 2.
    - ✓ Manually update NPAGES (INT) and NPAGESF (FLOAT) fields in SYSIBM.SYSTABLES for the table.
      - For example, 1 to NPAGES, 1.0 to NPAGESF (the values must be consistent)
      - If NPGTHRSH was already set to a non-zero value, set a smaller value to NPAGES and NPAGESF fields of SYSTABLES.
    - ✓ Setting NPGTHRSH could cause a potential side effect but keeping its value small, it will be minimum.
  - ▶ The ZPARM NPGTHRSH will be discussed later again.



# Special ZPARAMs

- When a PTF change in the optimizer area can impact broad range of workloads, a new subsystem parameter is added in the PTF as a ‘serviceability ZAPRM’.
  - ▶ The PTF change is disabled by default.
  - ▶ To enable the PTF change, the ZPARAM has to be set.
  - ▶ The ZPARAM is usually a YES/NO switch
  - ▶ Usually “hidden” (not on the system parameter panel, not in the IBM supplied JCL DSNTIJUZ)
- This is for “plan stability”.
  - ▶ General users do not need to enable the change
    - ✓ Unless the problem described in the PTF (APAR) was experienced
  - ▶ IBM service will provide a guidance, if a PMR problem is to be solved by enabling the PTF change.
- In most cases, the ZPARAM is effective to the target releases of the PTF.
  - ▶ In the next release, once the PTF fix is verified good in the field, the ZPARAM will be removed.
- The “special” ZPARAMs are not usually described in IBM manuals.
  - ▶ Because they are primarily for serviceability and transient in the nature



# Special ZPARMs

- INLISTP ← PQ73454
  - ▶ V7 (default 0, disabled), V8 and 9 (default 50, enabled)
  - ▶ Controls
    - ✓ Push-down of IN-list predicates to a materialized view or table expression.
    - ✓ Predicate bubble-up from a correlated subquery
  - ▶ INLISTP > 0 → enable “bubble-up” of simple predicates
  - ▶ INLISTP > 1 → enable IN-list “push-down” and “bubble-up” when the number of list elements is less than or equal to the specified INLISTP value.
- NPGTHRSH
  - ▶ V6, V7, V8 and 9 (default 0, i.e., disabled)
  - ▶ When a value larger than 0 is set, an index access with most matching columns will be used as a preferred access type on a table whose NPAGES (and/or NPAGESF) is less than the specified NPGTHRSH value.
  - ▶ Setting a large value to NPGTHRSH can cause side effects.



# Special ZPARAMs

- OPTXIIRC ← PK30857
  - ▶ V8 (default NO, i.e., disabled), removed in DB2 9
  - ▶ Controls enablement of an improved index access costing method
  - ▶ Remark: Set OPTXOIRC = YES if the performance of multiple queries regressed due to access path change.
- OPTCCOS1 ← PQ65335
  - ▶ V7 (default NO, i.e., disabled), removed in V8
  - ▶ Controls enablement of an improved use of multi-column cardinality statistics
- OPTCCOS2 ← PQ84158
  - ▶ V7 (default NO, i.e., disabled), removed in V8
  - ▶ Controls enablement of an improved correlated subquery costing
- OPTCCOS3 ← PK14923
  - ▶ V7 (default NO, i.e., disabled), removed in V8
  - ▶ Controls enablement of an improved index access costing with screening predicates



# Special ZPARMs

- PARAMDEG ← an external system parameter
  - ▶ V7, V8 and 9 (default 0, this parameter not to be used)
  - ▶ Specifies the maximum degree of parallelism
- OPTSUBQ1 ← PQ50462 (PQ80660, PQ81790, PQ84668)
  - ▶ V7 (default NO, i.e., disabled), removed in V8
  - ▶ Controls improved non-correlated subquery costing methods
- STATCLUS
  - ▶ DB2 9 (default ENHANCED to use new cluster ratio formula)
  - ▶ Controls RUNSTATS compute the cluster ratio of an index using the new formula. To use the prior (V8 and earlier) formula, set  
STATCLUS = STANDARD
- OPTIXOPREF ← PK51734
  - ▶ V8 and 9 (default OFF, i.e., disabled)
  - ▶ If enabled, DB2 will “favor” index-only access



# Conclusion – query optimization challenges

- Access path selection is based on the statistics
  - ▶ Statistics do not necessarily represent the real data distributions
  - ▶ Missing statistics, undetectable data skews and correlations, .....
- Statistics not always applicable
  - ▶ Host variables or parameter markers
  - ▶ Selectivity for expressions, user-defined table functions, ...
- Cost estimation and comparison
  - ▶ Many combinations (access types and Join types on different indexes, join sequences, .....
  - ▶ Limitations in algorithms and modeling

→ Optimizer does a good job, usually ..... but not perfect (yet) !



## Conclusion – getting the most from the optimizer

- Avoid optimizer making guesses !
  - ▶ Keep the statistics current
  - ▶ Whenever possible, provide “good” SQL statements
    - Better application of predicates
    - Minimize impact of host variables and parameter markers
  - ▶ Evaluate indexes to best suit the workload
- Prepare backup in advance
  - ▶ Always keep the EXPLAIN information for critical packages
  - ▶ Save access paths for backup
- Best use of options if applicable
  - ▶ REOPT
  - ▶ Optimization hints
  - ▶ OPTIMIZE FOR n ROWS, some tuning ZPARMs



# THANK YOU

Any question ?

Yoichi Tsuji

[tsujiy@us.ibm.com](mailto:tsujiy@us.ibm.com)

