#SHAREorg

SHARE
Technology · Connections · Results

Do these DB2 10 for z/OS Optimizer Enhancments apply to me?

Andrei Lurie
IBM Silicon Valley Lab

February 4, 2013
Session Number 12739

SHARE in San Francisco 2013

Agenda

- Introduction
- IN-list and complex ORs
- Predicate simplification
- Stage 2 predicate pushdown
- View/Table expression merge
- Misc optimizations
- Summary

DB2 10 and Query Performance

- Major **focus of DB2 10 was "out-of-the-box" performance improvements**
  - Query performance has a large impact on application/system performance (and thus, TCO)
  - Bigger focus on performance than prior releases of DB2
- 3-pronged approach for DB2 10 query performance
  - Plan management and query stabilization
  - Runtime optimizations (regardless of access path choice)
  - New optimizer choices

DB2 10 Optimizer changes overview

- Plan management and Query Stabilization
  - Concentrate statements with literals
  - Plan management – APREUSE/APCOMPARE
  - EXPLAIN enhancements
  - Statement level hints and options
- Runtime optimizations
  - Predicate evaluation enhancements
  - RID overflow to work file
- New access path choices
  - Multi-IN list matches
  - Range-list access
- Misc
  - RUNSTATS performance and usability
  - Even distribution for parallelism and removal of limitations

See session 12740 "DB2 for z/OS Migration – Query Performance Considerations"

Focus of this presentation

Agenda

- Introduction
- IN-list and complex ORs
- Predicate simplification
- Stage 2 predicate pushdown
- View/Table expression merge
- Misc optimizations
- Summary

Performance enhancement for complex OR & long IN-lists

- Improvement to early-out of (non-matching) ORs and IN-lists
  - For example: `WHERE C1 IN (1,2,3,...,100)`
    - DB2 9 stops predicate comparison once a match was found, but cycles through the predicate tree to find the end.
    - DB2 10 stops predicate comparison and jumps to end
- Predicate application (code path) is also reduced in DB2 10
- Runtime optimization - no REBIND needed
- Not visible in PLAN_TABLE

IN-list Table
Table Type 'I' and Access Type 'IN'

- The IN-list predicate will be represented as an in-memory table if:
  - List prefetch is chosen, OR
  - More than one IN-list is chosen as matching
- The EXPLAIN output associated with the in-memory table will have:
  - New Table Type: TBTYPE – 'I'
  - New Access Type: ACTYPE – 'IN'

```
SELECT … FROM T1 WHERE T1.C1 IN (?, ?, ?);
```

QBNO	PLANNO	METHOD	TNAME	ACTYPE	MC	ACNAME	QBTYPE	TBTYPE	PREFETCH
1	1	0	DSNIN001(01)	***IN***	0		SELECT	***I***	
1	2	1	T1	I	1	T1_IX_C1	SELECT	T	***L***

Multi IN-List Example

- Ability to match on more than one IN-list
  - DB2 9 could only match on the 1st IN-list
  - DB2 10 can match on all three columns in this example

```
SELECT … FROM T1
WHERE C1 IN (?, ?)
  AND C2 IN (?, ?)
  AND C3 = ?
```

```
index I1 on T1 (C1,C2,C3)
column  cardinality

C1                   2
C2                  20
C3          10,000,000
```

```
Pln                                          I SORT COMP  T
Nr Nr   M Table          AC  MC Index        O UJOG UJOG  T
-- --- -- ------------   --  -- ------       - ---- ----  -
 1   1  0 DSNIN002(01)   IN   0 I1           N NNNN NNNN  I
 1   1  1 DSNIN003(01)   IN   0 I1           N NNNN NNNN  I
 1   1  1 T1             I    3 I1           N NNNN NNNN  T
```

IN-list Predicate Transitive Closure (PTC)

```
SELECT …
FROM T1, T2
WHERE T1.C1 = T2.C1
  AND T1.C1 IN (?, ?, ?)
  AND T2.C1 IN (?, ?, ?)   ← Optimizer can generate
                              this predicate via PTC
```

- Without IN-list PTC (DB2 9)
  - Optimizer will be unlikely to consider T2 as the first table accessed
- With IN-list PTC (DB2 10)
  - Optimizer can choose to access T2 or T1 first
- PTC already supported for =, BETWEEN, <, <=, >, >=

Reducing Matchcols for IN-lists

```
SELECT …
FROM T1
WHERE C1 = ?
  AND C2 IN (?, ?, ?, ?, ?) ← Optimizer may
                               choose not to match
```

- If the equals (=) predicates provide strong filtering
  - Optimizer may choose not to match on the IN-list
    - Instead apply as index screening
    - To avoid overhead of additional index probing
- Example above
  - MATCHCOLS reduced from 2 to 1
  - ACCESSTYPE changed from “N” to “I”
- Optimizer already trims IN-lists if equals predicates are unique

Range-list access
Targets two types of OR queries

- Cursor scrolling (pagination) SQL
  - Retrieve next n rows
  - Common in COBOL/CICS and any screen scrolling application
  - Not to be confused with “scrollable cursors”
    - Hence term pagination to avoid confusion (???)
- Complex OR predicates against the same columns
  - 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.

Simple scrolling
Index matching and ORDER BY

- Scroll forward to obtain the next 20 rows
  - WHERE clause may appear as:

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

  - Assumes index is available on (LASTNAME, FIRSTNAME)
  - DB 10 supports
    - Single matching index access with sort avoided
  - DB2 9 requires
    - Multi-index access, list prefetch and sort

Simple scrolling – Pre-DB2 10 Solution

- To avoid multi-index access in online transactions for scrolling SQL
  - Customers added redundant predicate to support single matching index
  - And – potentially OPTIMIZE FOR n ROWS

```
WHERE ((LASTNAME='JONES' AND FIRSTNAME>'WENDY')
   OR (LASTNAME>'JONES'))
AND LASTNAME >= 'JONES'
ORDER BY LASTNAME, FIRSTNAME
OPTIMIZE FOR 1 ROW;
```

  - DB2 10 supports
    - MATCHCOLS=2 on 1st OR, and MC=1 on 2nd OR
  - DB2 9 supports
    - MATCHCOLS=1 on predicate in **red**

- NOTE: APAR PM56355 to encourage range-list access with OFnR and extra predicate

Complex OR predicates against same index

- Given WHERE clause
  - And index on one or both columns

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

- DB2 9 requires
  - Multi-index access with list prefetch
- DB2 10 supports
  - Matching single (range-list) index access – no list prefetch
  - Or, Multi-index access with list prefetch

Range-list – PLAN_TABLE representation

- Order of PLAN_TABLE entries is by coding sequence
  - Determination of execution sequence deferred to runtime
    - When all host variables/parameter markers are resolved
  - For this example, coding sequence does not match execution sequence

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

QBlockno	Planno	Accessname	Access_Type	Matchcols	Mixopseq
1	1	IX1	NR	1	1
1	1	IX1	NR	2	2

New access type (NR = IN-List Range)

Coding sequence

Range-list – not always chosen

- Range-list (ACCESSTYPE='NR') is a new optimizer choice
  - Does not mean "ALWAYS" chosen
    - This is a cost-based choice
- Range-list is a good choice when
  - Single matching index access is needed to avoid:
    - RID processing
    - Sort (if order is needed)
  - Ideal for online (CICS/COBOL or web) screen scrolling
- Biggest challenge for optimizer
  - Programs that FETCH first n rows, but do NOT have OPTIMIZE or FETCH FIRST clause

Agenda

- Introduction
- IN-list and complex ORs
- Predicate simplification
- Stage 2 predicate pushdown
- View/Table expression merge
- Misc optimizations
- Summary

Remove “Always False” Predicates

- Remove “Always False” literal predicates
  - Literal “IN” or “=“ only (no host vars or REOPT)
    - Original “OR” is stage 2
      - Disables index access and many query transformations

```
WHERE ('A' = 'B' OR COL1 IN ('B', 'C'))
```

  - Becomes….

```
WHERE COL1 IN ('B', 'C')
```

  - Documented tricks are NOT pruned (OR 0=1)
    - Available in V8/9 via zparm PREDPRUNE

Other variations of “Always False/True”

- Removal of “Always False” literal predicates
  - Does NOT apply for “Always False” within “AND”

```
WHERE ((COL1= 'C' AND 'A' = 'B') OR COL1 ='B')
```

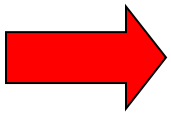
- “Always True” predicates are NOT pruned
  - These will be executed by DB2 for each row processed

```
WHERE 1=1
```

Remove Unnecessary Tables

- Remove unnecessary LEFT OUTER JOIN tables
  - If no columns are SELECTed from the RIGHT table, then
    - The right table is unnecessary if no duplicates because:
      - Unique index on join key of right table
      - Or, SELECT DISTINCT

```
SELECT DISTINCT T1.C3
FROM T1 LEFT OUTER JOIN T2
ON T1.C2 = T2.C2
WHERE T1.C1 = ?
```



```
SELECT DISTINCT T1.C3
FROM T1
WHERE T1.C1 = ?
```

  - NOTE: “Removed” tables will NOT appear in explain (PLAN_TABLE)

» Available in V8/9 via zparm PREDPRUNE

Agenda

- Introduction
- IN-list and complex ORs
- Predicate simplification
- Stage 2 predicate pushdown
- View/Table expression merge
- Misc optimizations
- Summary

Stage 2 predicate “pushed down” to IM/DM

- Most Stage 2 (residual) predicates can execute as index screening (indexable) or stage 1 (sargable)
  - CPU time improvement
  - Reduced data getpages if stage 2 predicate becomes index screening
  - Applies to
    - Arithmetic/datetime expressions, scalar built-in functions, CAST (essentially all expressions without subqueries or CASE)
  - Does not apply to
    - Cannot be OR’d with an IN predicate
    - List prefetch
  - Eligible for DM (stage 1) pushdown but NOT IM pushdown
    - Join predicates
    - OR’d predicates that span different predicate stages
- Externalized in DSN_FILTER_TABLE column PUSHDOWN

Stage 2 predicate “pushed down” to IM/DM

- Timing
  - Index matching
  - Index screening
  - Stage 2 pushed down to IM — No data access
  - Stage 1 — Data access
  - Stage 2 pushed down to DM
  - Stage 2
- Push down decision is made after the access path selection
  - Does not influence access path selection decisions (not cost-based)
  - RDS is still invoked to evaluate

Stage 2 predicate “pushed down” examples (1)

Given an index on CITY, ZIPCODE

```
EXPLAIN ALL SET QUERYNO = 1 FOR
 SELECT CUSTNO FROM CUSTOMERS
 WHERE CITY = ?
   AND MOD(ZIPCODE,5)=0;
```

DSN_FILTER_TABLE output

```
   +-----------------------------------------------+
   |  QUERYNO  |  PREDNO  |   STAGE    | PUSHDOWN |
   +-----------------------------------------------+
 1_|         1 |        2 | MATCHING   |          |
 2_|         1 |        3 | STAGE2     | I        | ← Eligible for IM
   +-----------------------------------------------+
```

Where is PREDNO=1? From DSN_PREDICAT_TABLE

```
   ------------------------------------------------------------------------------------
   | PREDNO | TYPE |  TEXT
   ------------------------------------------------------------------------------------
 1_|      1 | AND  | (SYSADM.CUSTOMERS.CITY=? AND MOD(SYSADM.CUSTOMERS.ZIPCODE,5)=0)
```

Stage 2 predicate “pushed down” examples (2)

```
EXPLAIN ALL SET QUERYNO = 2 FOR
 SELECT CUSTNO FROM CUSTOMERS
 WHERE CITY = ?
   AND( CASE WHEN ZIPCODE = 99999 THEN 1 ELSE ZIPCODE END) = 99999;

EXPLAIN ALL SET QUERYNO = 3 FOR
 SELECT CUSTNO
 FROM CUSTOMERS
 WHERE CITY = ?
 AND (MOD(ZIPCODE,5)=0 OR ZIPCODE IN (99998, 99999));
```

	QUERYNO	PREDNO	STAGE	PUSHDOWN	
3_	2	2	MATCHING		
4_	2	3	STAGE2		← Not Eligible (CASE expr)
5_	3	2	MATCHING		
6_	3	3	STAGE2		← Not Eligible (IN-list)

Stage 2 predicate “pushed down” – Predicate Tricks (3)

```
EXPLAIN ALL SET QUERYNO = 4 FOR
 SELECT CUSTNO
 FROM CUSTOMERS
 WHERE CITY = ?
 AND (ZIPCODE = 99999 OR 0=1);    ← documented trick

EXPLAIN ALL SET QUERYNO = 5 FOR
 SELECT CUSTNO
 FROM CUSTOMERS
 WHERE CITY = ?
 AND (ZIPCODE = 99999 OR 1=2);    ← Incorrect use of documented trick
```

	QUERYNO	PREDNO	STAGE	PUSHDOWN	
7_	4	2	MATCHING		
8_	**4**	**3**	**STAGE2**	**I**	**← Eligible for IM**
9_	5	2	MATCHING		
10_	**5**	**3**	**MATCHING**		**← ??????**

**** OR 1=2 has been pruned, leaving indexable predicate. OR 0=1 is NOT pruned.**

Stage 2 predicate “pushed down” – Predicate Tricks (4)

```
EXPLAIN ALL SET QUERYNO = 6 FOR
 SELECT CUSTNO
 FROM CUSTOMERS
 WHERE CITY = ?
 AND ZIPCODE = 99999+0;        ← Documented trick to disable index matching

EXPLAIN ALL SET QUERYNO = 7 FOR
 SELECT CUSTNO
 FROM CUSTOMERS
 WHERE CITY = ?
 AND ZIPCODE+0 = 99999;        ← Incorrect use of documented trick (stage 2 pred)
```

	QUERYNO	PREDNO	STAGE	PUSHDOWN	
11_	6	2	MATCHING		
12_	**6**	**3**	**STAGE1**		← applied as screening
13_	7	2	MATCHING		
14_	**7**	**3**	**STAGE2**	**I**	← Eligible for IM

Stage 2 predicates “pushed down” examples (5)

Given and index on CITY, ZIPCODE

```
EXPLAIN ALL SET QUERYNO = 8 FOR
 SELECT CUSTNO
 FROM CUSTOMERS
 WHERE CITY = ?
 AND (ZIPCODE+0 = 99999 OR ZIPCODE = 99998); ← All OR'd predicates are in index

EXPLAIN ALL SET QUERYNO = 9 FOR
 SELECT CUSTNO
 FROM CUSTOMERS
 WHERE CITY = ?
 AND (ZIPCODE+0 = 99999 OR CUSTNO = 1);      ← All OR'd predicates NOT in index
```

	QUERYNO	PREDNO	STAGE	PUSHDOWN	
15_	8	2	MATCHING		
16_	**8**	**3**	**STAGE2**	**I**	← **Eligible for IM**
17_	9	2	MATCHING		
18_	**9**	**3**	**STAGE2**	**D**	← **Eligible for DM (stage 1)**

Agenda

- Introduction
- IN-list and complex ORs
- Predicate simplification
- Stage 2 predicate pushdown
- View/Table expression merge
- Misc optimizations
- Summary

Merge expression on preserved side of Outer Join

- DB2 can merge view/table expression on preserved side of outer join
  - CASE, VALUE, COALESCE, NULLIF, IFNULL
    - Exception if merged predicate is stage 2

```
SELECT A.C1, B.C1, A.C2, B.C2
FROM T1, (SELECT COALESCE(C1,0) as C1, C2
          FROM   T2) A      <--table expression 'A' will be Merged
     LEFT OUTER JOIN
         (SELECT COALESCE(C1,0) as C1, C2
          FROM   T3) B      <-- B will be Materialized
     ON A.C2 = B.C2
WHERE T1.C2 = A.C2;
```

Merge single table view/ table expression with subquery

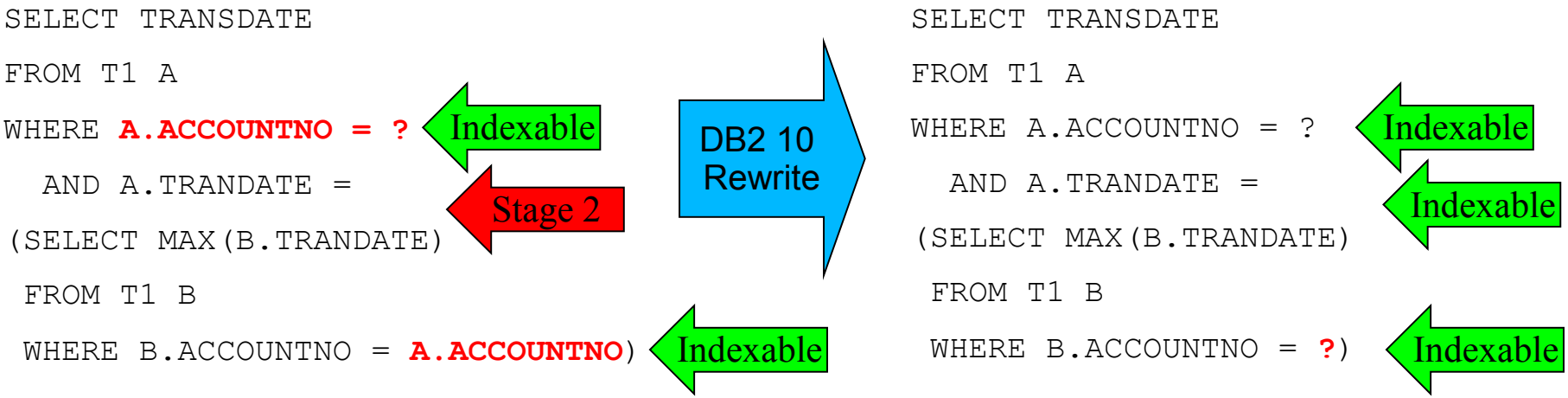
```
SELECT *
FROM T1 LEFT OUTER JOIN
      (SELECT * FROM T2
       WHERE T2.C1 = (SELECT MAX(T3.C1) FROM T3 ) ) TE <--subquery
ON T1.C1 = TE.C1;
```

```
SELECT *
FROM T1 LEFT OUTER JOIN T2                  <-- table expression is merged
ON T2.C1 = (SELECT MAX(T3.C1) FROM T3)   <-- subquery ON-predicate
AND T1.C1 = TT.C1;
```

- View/Table expression with subquery on NULL-supplied side
  - Merge into ON clause
- On preserved side
  - Merge into WHERE clause

Correlated to Non-correlated Rewrite

- DB2 10 can rewrite correlated to non-correlated
  - If correlation predicates are covered by local predicates in outer
    - Can result in additional index matching predicate
  - Only targets simple example shown



Agenda

- Introduction
- IN-list and complex ORs
- Predicate simplification
- Stage 2 predicate pushdown
- View/Table expression merge
- Misc optimizations
- Summary

Minimizing Optimizer Challenges – Safe Optimization

- Potential causes of sub-optimal plans
  - Insufficient statistics
  - Unknown literal values used for host variables or parameter markers

- Optimizer will evaluate the risk for each predicate

  - For example: WHERE BIRTHDATE < ?
    - Could qualify 0-100% of data depending on literal value used
  - As part of access path selection
    - Compare access paths with close cost and choose lowest risk plan

Minimizing impact of RID failure

- RID overflow can occur for
  - Concurrent queries each consuming shared RID pool
  - Single query requesting > 25% of table or hitting RID pool limit

- DB2 9 will fallback to tablespace scan*

- DB2 10 will continue by writing new RIDs to work file 
  - Work file usage may increase
    - Mitigate by increasing RID pool size (default increased in DB2 10).
    - MAXTEMPS_RID zparm for maximum WF usage for each RID list
  - Not supported for queries with column functions (MAX, MIN etc)
  - Runtime optimization (no REBIND needed)

* Hybrid join can incrementally process. Dynamic Index ANDing will use WF for failover.

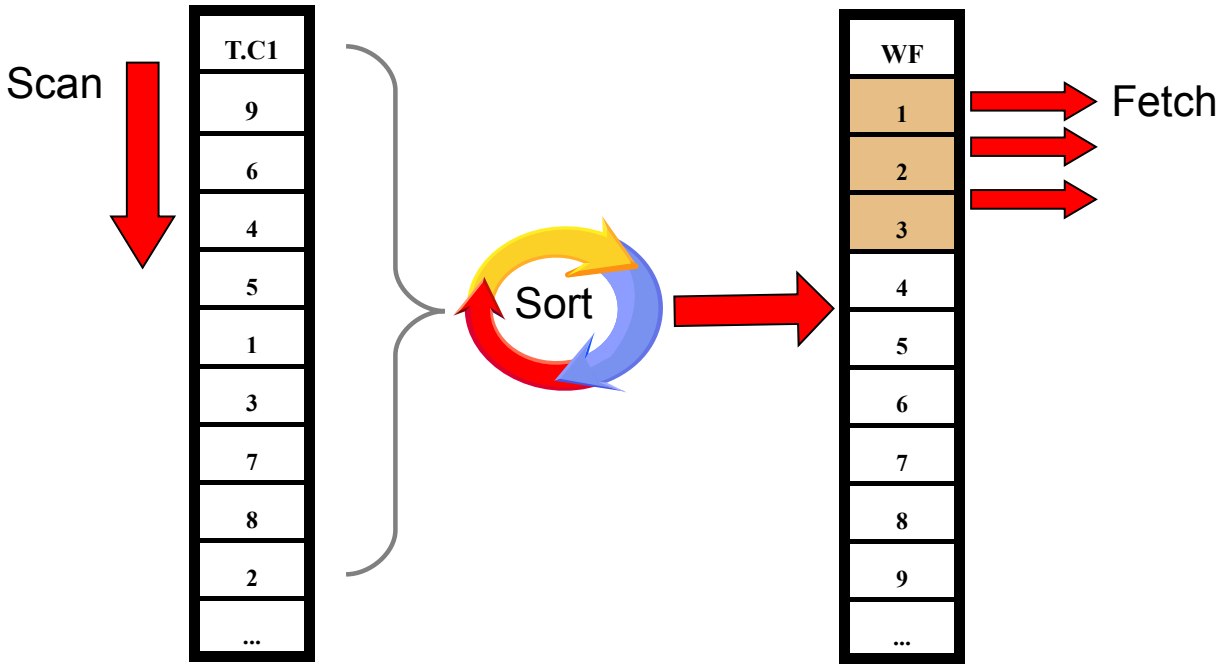
Sort performance enhancements

- Tournament tree sort is avoided for FETCH FIRST n ROWS ONLY
  - DB2 9 - only ORDER BY and (n * (sort key + data)) < 32K
  - DB2 10 - extended to GROUP BY (without HAVING), and to
    (n * (sort key + data)) < 128K
    if over 128K, each WF for each run contains only n rows
  - Instead of sorting, uses in-memory replacement technique
    - Demonstrated on next slides

Sort performance enhancements 2

- Sort is not avoided for FETCH FIRST n ROWS ONLY prior to DB2 10:

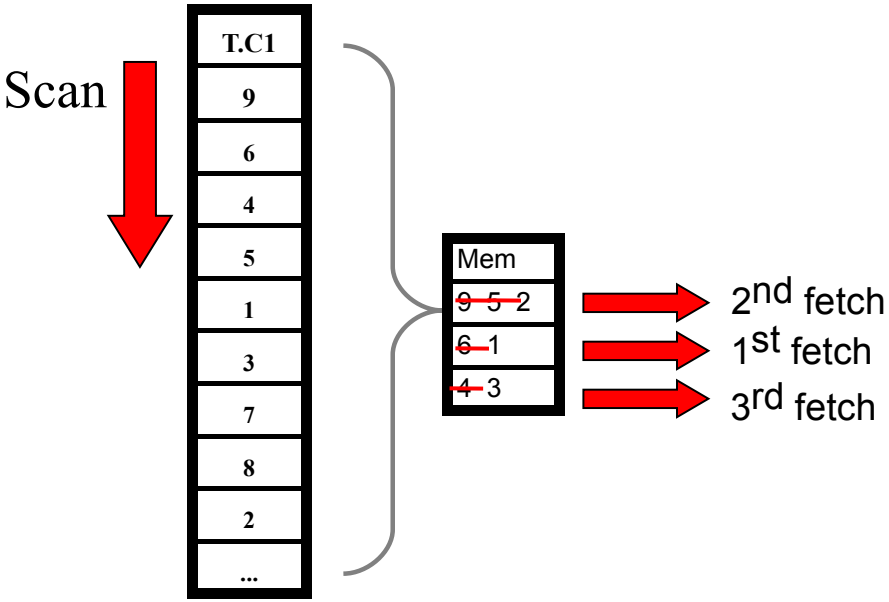
```
SELECT SUM(C1)
FROM   T1
GROUP BY C2
FETCH FIRST 3 ROWS ONLY;
```



Sort performance enhancements 3

- Sort avoidance for FETCH FIRST n ROWS ONLY in DB2 9 and later:

```
SELECT SUM(C1)
FROM   T1
GROUP BY C2
FETCH FIRST 3 ROWS ONLY;
```



Other Sort performance enhancements

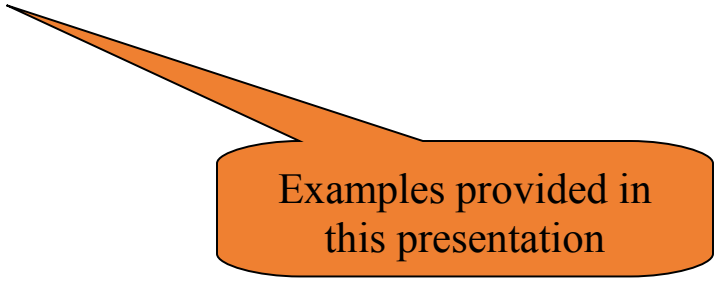
- In-memory work file for small sorts
  - extended to intermediate sorts (DB2 9 was top query sort only)
    - Up to 255 rows
  - block fetch for reading the in-memory Sort work file
    - top query sort only
    - (#rows in work file * (sort key + data)) < 1M
    - Significant CPU time reduction (due to avoiding RDS/DM trips)
- RID sort and RID intersection are done in-place, RID union is done with extra 2 RID blocks only
  - Minimizes RID pool storage usage (50% savings for sort/union)

Agenda

- Introduction
- IN-list and complex ORs
- Predicate simplification
- Stage 2 predicate pushdown
- View/Table expression merge
- Misc optimizations
- Summary

DB2 10 and Query Performance - Recap

- **Major focus of DB2 10 was "out-of-the-box" performance improvements**
  - Query performance has a large impact on application/system performance (and thus TCO)
    - Bigger focus on query performance than prior releases of DB2
  - 3-pronged approach for DB2 10 query performance
    - Plan management and query stabilization
    - Runtime optimizations (regardless of access path choice)
    - New optimizer choices



Acknowledgements and Disclaimers

The materials have been prepared by IBM or the session speakers and reflect their own views. They are provided for informational purposes only, and are neither intended to, nor shall have the effect of being, legal or other guidance or advice to any participant. 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. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this presentation or any other materials. 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 the applicable license agreement governing the use of IBM 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

Other company, product, or service names may be trademarks or service marks of others.

#SHAREorg

SHARE
Technology · Connections · Results

Do these DB2 10 for z/OS Optimizer Enhancments apply to me?

Andrei Lurie
IBM Silicon Valley Lab

February 4, 2013
Session Number 12739

SHARE in San Francisco 2013