Information Management

IBM

Data Management

Optimizing Applications Using Extended Explain Tables

Jase Alpers
IBM Silicon Valley Lab

Disclaimer

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.

Agenda

- Primary cause of access path problems
- DSN_PREDICAT_TABLE structure
- Populating DSN_PREDICAT_TABLE
- Tuning opportunities

Primary Cause of Access Path Problems

- **Inaccurate selectivity estimates** due to ...
  - Lack of statistics
  - Difficult to estimate predicates
  - Host variables / parameter markers with …
    - Range predicates
    - Skewed columns

Access Path Instability

- Is the access path for your SQL obvious?
  - Is the optimizer able to apply the filtering early?
  - Are there indexes that support an efficient path?
  - Do statistics allow distinction between the choices?

- If no……
  - You're more likely to have performance regressions
    - From minor changes in statistics
    - At release or maintenance upgrade times

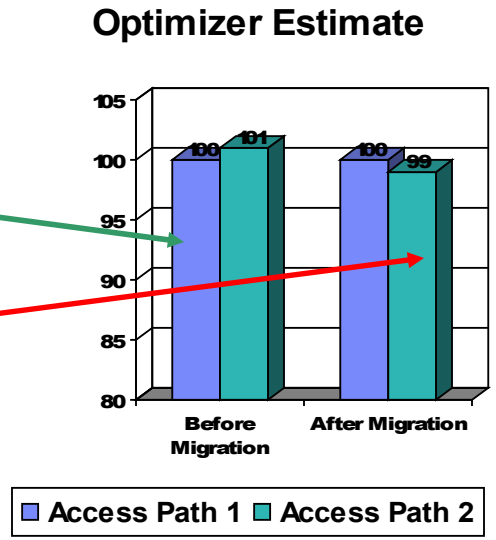
Why can performance fluctuate?

- When access path costs do not reflect reality
  - Eg. Actual performance of 2 access paths
    - Access path 1 = 100 msec
    - Access path 2 = 1 min

Best performing

600 X slower

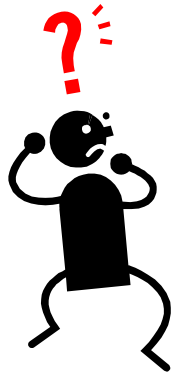
  - Before
    - Access path 1 wins
      - → Estimate 100msec vs 101msec
  - After migration/maint etc.
    - Access path 2 wins
      - → Estimate 100msec vs 99msec



The problem?

- After Rebind
  - Access path 2 cost improved from 101 to 99 msec
    - But reality is 1 min (approx 600X greater)

- Should the focus be on:
  - What caused the cost to change from 101 to 99?
  - OR, why the cost isn't closer to 1 min?



Primary Cause of Access Path Problems

- **Inaccurate selectivity estimates** due to ...
  - Lack of statistics
  - Difficult to estimate predicates
  - Host variables / parameter markers with …
    - Range predicates
    - Skewed columns

Agenda

- Primary cause of access path problems
- DSN_PREDICAT_TABLE structure
- Populating DSN_PREDICAT_TABLE
- Tuning opportunities

Example DSN_PREDICAT_TABLE content

- Query

```
SELECT SUM(L_EXTENDEDPRICE*L_DISCOUNT) AS REVENUE
  FROM LINEITEM
 WHERE L_SHIPDATE >= DATE('1994-01-01')
   AND L_SHIPDATE <  DATE('1994-01-01') + 1 YEAR
   AND L_DISCOUNT BETWEEN 0.06-0.01 AND 0.06+0.01
   AND L_QUANTITY < 24
```

- DSN_PREDICAT_TABLE Contents

	PREDNO	TYPE	LEFT_HAND_SIDE	LHS_TABNO	LHS_QBNO	RIGHT_HAND_SIDE	RHS_TABNO	RHS_QBNO
1_	1	AND		0	0		0	0
2_	2	RANGE	L_SHIPDATE	1	1	VALUE	0	0
3_	3	RANGE	L_SHIPDATE	1	1	VALUE	0	0
4_	4	BETWEEN	L_DISCOUNT	1	1	NONCOLEXP	0	0
5_	5	RANGE	L_QUANTITY	1	1	VALUE	0	0

	FILTER_FACTOR	TEXT
1_	1.000000000000000E+00	(((TPCD30G.LINEITEM.L_SHIPDATE>='1994-01-01' AND TPCD30G.LINEITEM.L_SHIPDATE<'19
2_	7.211780548095703E-01	TPCD30G.LINEITEM.L_SHIPDATE>='1994-01-01'
3_	4.305213093757629E-01	TPCD30G.LINEITEM.L_SHIPDATE<'1995-01-01'
4_	9.999996423721313E-02	TPCD30G.LINEITEM.L_DISCOUNT BETWEEN CAST((0.06-0.01) AS FLOAT(0,8)) AND CAST((
5_	4.799535274505615E-01	TPCD30G.LINEITEM.L_QUANTITY<24

DSN_PREDICAT_TABLE

- Contains information about each predicate
  - One row for each atomic predicate
  - One row for each AND/OR tree
- View into optimizer's selectivity estimates
  - Can be used to determine how likely it is that optimizer's estimates are incorrect

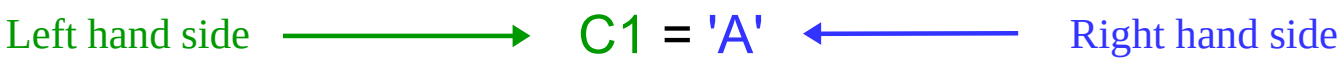
DSN_PREDICAT_TABLE Columns

- The predicate table correlates to the plan table
  - QUERYNO
  - QBLOCKNO
  - APPLNAME
  - PROGNAME
  - COLLID*
  - VERSION*
  - SECTNOI*
  - EXPLAIN_TIME (PLAN_TABLE.BIND_TIME)
- Unique identifier of a predicate within a query
  - PREDNO

*Added in DB2 10

DSN_PREDICAT_TABLE Columns

- Predicate Structure
  - Optimizer tracks the left hand side (LHS) and the right hand side (RHS) of a predicate



  - Optimizer may decide to swap sides from what was originally coded in the query

Column Name	Meaning
LHS_QBNO	Query block of LHS
LHS_TABNO	Table number of LHS
RHS_QBNO	Query block of RHS
RHS_TABNO	Table number of RHS

  - QBNO fields match PLAN_TABLE.QBLOCKNO
  - TABNO fields match PLAN_TABLE.TABNO

DSN_PREDICAT_TABLE Columns

- Predicate Structure

Column Name	Meaning
LEFT_HAND_SIDE	Column name or description of LHS
RIGHT_HAND_SIDE	Column name or description of RHS

  - Descriptions of LHS/RHS when not a column
    - VALUE
      - Literal value
    - COLEXP
      - Column expression
    - NONCOLEXP
      - Non-column expression
    - CORSUB
      - Correlated subquery
    - NONCORSUB
      - Non-correlated subquery

DSN_PREDICAT_TABLE Columns

- Predicate Attributes

Column Name	Meaning
TYPE	Predicate type – One of... • AND • OR • EQUAL • RANGE • BETWEEN • IN • LIKE • NOT LIKE • EXISTS • NOTEXIST • SUBQUERY • HAVING • OTHERS
CLAUSE	SQL clause where the predicate appears – One of... • WHERE • ON • HAVING • SELECT

DSN_PREDICAT_TABLE Columns

- Predicate Attributes

Column Name	Meaning
FILTER_FACTOR	Estimated selectivity
BOOLEAN_TERM	Whether a false value indicates the entire WHERE clause is false
SEARCHARG	Y – Indicates the predicate can be processed by DM (stage 1) N – Indicates the predicate must be handled by RDS (stage 2)
AFTER_JOIN	A – After join predicate D – During join predicate Blank - N/A
ADDED_PRED	Y – Predicate generated by predicate transitive closure
TEXT	Text of the predicate
MARKER	Y – Predicate has a host variable or parameter marker
PARENT_PNO	Parent predicate PREDNO
NEGATION	Y – The predicate is negated via NOT
LITERALS	The literal values used (separated by colons)

Example DSN_PREDICAT_TABLE content

- Query

```
SELECT SUM(L_EXTENDEDPRICE*L_DISCOUNT) AS REVENUE
  FROM LINEITEM
 WHERE L_SHIPDATE >= DATE('1994-01-01')
   AND L_SHIPDATE <  DATE('1994-01-01') + 1 YEAR
   AND L_DISCOUNT BETWEEN 0.06-0.01 AND 0.06+0.01
   AND L_QUANTITY < 24
```

- DSN_PREDICAT_TABLE Contents

	PREDNO	TYPE	LEFT_HAND_SIDE	LHS_TABNO	LHS_QBNO	RIGHT_HAND_SIDE	RHS_TABNO	RHS_QBNO
1_	1	AND		0	0		0	0
2_	2	RANGE	L_SHIPDATE	1	1	VALUE	0	0
3_	3	RANGE	L_SHIPDATE	1	1	VALUE	0	0
4_	4	BETWEEN	L_DISCOUNT	1	1	NONCOLEXP	0	0
5_	5	RANGE	L_QUANTITY	1	1	VALUE	0	0

	FILTER_FACTOR	TEXT
1_	1.000000000000000E+00	(((TPCD30G.LINEITEM.L_SHIPDATE>='1994-01-01' AND TPCD30G.LINEITEM.L_SHIPDATE<'19
2_	7.211780548095703E-01	TPCD30G.LINEITEM.L_SHIPDATE>='1994-01-01'
3_	4.305213093757629E-01	TPCD30G.LINEITEM.L_SHIPDATE<'1995-01-01'
4_	9.999996423721313E-02	TPCD30G.LINEITEM.L_DISCOUNT BETWEEN CAST((0.06-0.01) AS FLOAT(0,8)) AND CAST((
5_	4.799535274505615E-01	TPCD30G.LINEITEM.L_QUANTITY<24

Agenda

- Primary cause of access path problems
- DSN_PREDICAT_TABLE structure
- Populating DSN_PREDICAT_TABLE
- Tuning opportunities

Populating DSN_PREDICAT_TABLE

- All extended explain tables are populated during EXPLAIN
  - Table definitions must match the documented DDL
  - For most extended explain tables, no warnings or errors are generated. Problems result in no data.

- If no data is populated, most likely the DDL is incorrect.

Populating DSN_PREDICAT_TABLE

- Populate explain tables for an application
- Mine the data for application level actions
- Avoid tuning solely at the SQL level

Populating DSN_PREDICAT_TABLE

- Static SQL
  - Bind with EXPLAIN(YES) populates extended explain tables
  - Application use cases
    - REBIND a collection
    - REBIND with EXPLAIN(ONLY) (v10 only)
    - BIND into a new collection
      → Avoid impacting current access path choices

Populating DSN_PREDICAT_TABLE

- Dynamic SQL
  - Explain query populates extended explain tables
  - Application use case
    - Dynamic statement cache
      - → Needs an application or tool to populate explain tables
        - EXPLAIN STMTCACHE ALL
        - for each row in DSN_STATEMENT_CACHE_TABLE
          EXPLAIN <sql>
      - → Extended explain information is not saved in the cache so “EXPLAIN STMTCACHE STMTID xxx” will not populate extended explain tables

Agenda

- Primary cause of access path problems
- DSN_PREDICAT_TABLE structure
- Populating DSN_PREDICAT_TABLE
- Tuning opportunities

Tuning Opportunities

- Query rewrites
- Missing statistics 
- Identify potential indexes
- Queries at risk

Tuning Opportunities

- Query rewrites
  - SUBSTR(C1,1,1)='A'
    - Rewrite to C1 LIKE 'A%'
  - T1.C1 BETWEEN T2.C2 AND T2.C3
    - Rewrite to T1.C1 => T2.C2 AND T1.C1 <= T2.C3
  - DATE(TIMESTAMPCOL) = '2011-07-31'
    - Rewrite to
      TIMESTAMPCOL BETWEEN '2011-07-31-00.00.00.000000'
      AND '2011-07-31-23.59.59.999999'

Mining DSN_PREDICAT_TABLE – Query Rewrites

- Convert SUBSTR to LIKE
  - Stage 2 to stage 1 indexable

```
SELECT P.QUERYNO
      ,P.APPLNAME
      ,SUBSTR(P.PROGNAME,1,18) PROGNAME
      ,SUBSTR(P.TEXT,1,100) TEXT
FROM DSN_PREDICAT_TABLE P
WHERE P.LHS_TABNO = 0
  AND P.RHS_TABNO = 0
  AND P.LEFT_HAND_SIDE = 'COLEXP'
  AND P.RIGHT_HAND_SIDE = 'VALUE'
  AND P.TYPE = 'EQUAL'
  AND P.TEXT LIKE 'SUBSTR%'
  AND CASE
    WHEN POSSTR(P.TEXT,',')=0 THEN 0
    WHEN STRIP(SUBSTR(P.TEXT,POSSTR(P.TEXT,',')+1,1))='1' THEN 1
    ELSE 0
  END = 1
```

'Column expression = value' predicate

Equal predicate

SUBSTR starting at first character

*Note that this assumes the first comma is related to the SUBSTR function. Complex expressions may not be identified.

  - Rewrite SUBSTR(C1,1,1)='A' to C1 LIKE 'A%'
  - Simple application change that will improve performance even if no index is available/chosen
    - Stage 1 vs stage 2

Mining DSN_PREDICAT_TABLE – Query Rewrites

- Convert BETWEEN on columns to 2 range predicates
  - Partially stage 2 to stage 1 indexable

```
SELECT P.QUERYNO
      ,P.APPLNAME
      ,SUBSTR(P.PROGNAME,1,18) PROGNAME
      ,SUBSTR(P.TEXT,1,100) TEXT
FROM DSN_PREDICAT_TABLE P
WHERE P.TYPE = 'BETWEEN'
  AND P.RIGHT_HAND_SIDE = 'COLEXP'
```

Between predicate

RHS has a column

  - Rewrite C1 BETWEEN C2 AND C3 to C1 => C2 AND C1 <= C3
  - Simple application change that will improve performance even if no index is available/chosen
    - Stage 1 vs stage 2

Mining DSN_PREDICAT_TABLE – Query Rewrites

- Convert predicates with DATE(TIMESTAMPCOL) to range predicates
  - Stage 2 to stage 1 indexable

```
SELECT P.QUERYNO
      ,P.APPLNAME
      ,SUBSTR(P.PROGNAME,1,18) PROGNAME
      ,SUBSTR(P.TEXT,1,100) TEXT
FROM DSN_PREDICAT_TABLE P
WHERE P.LEFT_HAND_SIDE = 'COLEXP'
  AND P.TEXT LIKE 'DATE%'
```

Column expression on the LHS

DATE function

  - Rewrite DATE(TIMESTAMPCOL) = '2011-01-01' to TIMESTAMPCOL BETWEEN '2011-01-01-00.00.00.000000' AND '2011-01-01-23.59.59.999999'
  - Similar rewrites can be done for range predicates
  - Simple application change that will improve performance even if no index is available/chosen
    - Stage 1 vs stage 2

Tuning Opportunities

- Query rewrites
  - Queries with non-column expressions
    - 'Y' = ?
    - Push these into the application
  - Queries with trick predicates
    - Are these trick predicates still needed?
    - Are they solving a problem that only exists in an old version of DB2?
  - Queries with no stage 1 local predicates
  - Queries with cartesian joins

Mining DSN_PREDICAT_TABLE – Query Rewrites

- Identify queries with non-column expressions
  - 'Y' = ?

```
SELECT QUERYNO
      ,QBLOCKNO
      ,APPLNAME
      ,SUBSTR(PROGNAME,1,18) PROGNAME
      ,SUBSTR(TEXT,1,100) TEXT
FROM DSN_PREDICAT_TABLE
WHERE LEFT_HAND_SIDE IN ('VALUE','NONCOLEXP')
  AND RIGHT_HAND_SIDE IN ('VALUE','NONCOLEXP')
  AND LHS_TABNO = 0
  AND RHS_TABNO = 0
  AND LHS_QBNO = 0
  AND RHS_QBNO = 0
  AND CLAUSE <> 'HAVING'
  AND LITERALS = 'HV'
```

No columns referenced in the predicate

Predicate contains host variable or parameter marker

  - Push non-column expressions into the application

Mining DSN_PREDICAT_TABLE – Query Rewrites

- Identify queries with trick predicates
  - 0>1
  - T1.C1 = T1.C1

0>1 part

```
SELECT QUERYNO
      ,QBLOCKNO
      ,APPLNAME
      ,SUBSTR(PROGNAME,1,18) PROGNAME
      ,SUBSTR(TEXT,1,100) TEXT
FROM DSN_PREDICAT_TABLE
WHERE LEFT_HAND_SIDE IN ('VALUE','NONCOLEXP')
  AND RIGHT_HAND_SIDE IN ('VALUE','NONCOLEXP')
  AND LHS_TABNO = 0
  AND RHS_TABNO = 0
  AND LHS_QBNO = 0
  AND RHS_QBNO = 0
  AND CLAUSE <> 'HAVING'
  AND LITERALS <> 'HV'
UNION ALL
```

No columns referenced in the predicate

No host variable / parameter markers

T1.C1 = T1.C1 part

```
SELECT QUERYNO
      ,QBLOCKNO
      ,APPLNAME
      ,SUBSTR(PROGNAME,1,18) PROGNAME
      ,SUBSTR(TEXT,1,100) TEXT
FROM DSN_PREDICAT_TABLE
WHERE LHS_QBNO <> 0
  AND LHS_TABNO <> 0
  AND LHS_QBNO = RHS_QBNO
  AND LHS_TABNO = RHS_TABNO
  AND LEFT_HAND_SIDE = RIGHT_HAND_SIDE
```

Column exists on LHS

LHS and RHS match

- Are these trick predicates still needed?
- Were they added to address a limitation in a previous release of DB2?

Mining DSN_PREDICAT_TABLE – Query Rewrites

- Query blocks with no stage 1 local predicates

```
SELECT QUERYNO
      ,QBLOCKNO
      ,APPLNAME
      ,SUBSTR(PROGNAME,1,18)
FROM (
  SELECT QUERYNO, QBLOCKNO, APPLNAME, PROGNAME
        ,SUM(CASE WHEN SEARCHARG='Y' THEN 1
             ELSE 0 END) STAGE1_CNT
  FROM DSN_PREDICAT_TABLE
  GROUP BY QUERYNO, QBLOCKNO, APPLNAME, PROGNAME
) A
WHERE A.STAGE1_CNT = 0
```

No stage 1 predicates

  - This condition is legitimate, but should likely be rare
    - Are there opportunities to rewrite these queries so that stage 1 predicates exist?
      → Opportunity could be application specific
  - Mining query could be modified to join to SYSTABLES and limit to only query blocks containing a large table

Mining DSN_PREDICAT_TABLE – Query Rewrites

- Query blocks containing cartesian joins

QBs w/ >1 tbl →

All tables w/join predicates →

```
WITH MULT_TBL (QUERYNO, QBLOCKNO, APPLNAME
               ,PROGNAME, EXPLAIN_TIME) AS (
  SELECT QUERYNO, QBLOCKNO, APPLNAME
        ,PROGNAME, BIND_TIME
  FROM (
    SELECT DISTINCT QUERYNO, QBLOCKNO, APPLNAME
      ,QBLOCKNO, APPLNAME, PROGNAME, BIND_TIME, TABNO
    FROM PLAN_TABLE
    WHERE TABNO <> 0
  ) A
  GROUP BY QUERYNO, QBLOCKNO, APPLNAME
          ,PROGNAME, BIND_TIME
  HAVING COUNT(*)>1
),PREDS (QUERYNO, QBLOCKNO, APPLNAME, PROGNAME
        ,EXPLAIN_TIME, TABNO) AS (
   SELECT QUERYNO, QBLOCKNO, APPLNAME, PROGNAME
         ,EXPLAIN_TIME, RHS_TABNO
   FROM DSN_PREDICAT_TABLE
   WHERE LHS_TABNO <> 0
   UNION
   SELECT QUERYNO, QBLOCKNO, APPLNAME, PROGNAME
         ,EXPLAIN_TIME, LHS_TABNO
   FROM DSN_PREDICAT_TABLE
   WHERE RHS_TABNO <> 0
)
```

```
SELECT M.QUERYNO, M.QBLOCKNO, M.APPLNAME
      ,SUBSTR(M.PROGNAME,1,18) PROGNAME
      ,M.EXPLAIN_TIME
FROM MULT_TBL M
INNER JOIN PLAN_TABLE PL
  ON PL.QUERYNO = M.QUERYNO
 AND PL.QBLOCKNO = M.QBLOCKNO
 AND PL.APPLNAME = M.APPLNAME
 AND PL.PROGNAME = M.PROGNAME
 AND PL.BIND_TIME = M.EXPLAIN_TIME
LEFT OUTER JOIN PREDS P
  ON P.QUERYNO = M.QUERYNO
 AND P.QBLOCKNO = M.QBLOCKNO
 AND P.APPLNAME = M.APPLNAME
 AND P.PROGNAME = M.PROGNAME
 AND P.EXPLAIN_TIME = M.EXPLAIN_TIME
 AND P.TABNO = PL.TABNO
WHERE PL.TABNO <> 0
  AND P.QUERYNO IS NULL
```

No join predicates

- Identify queries with unexpected cartesian joins
- Mining query will miss join predicates on a column expression (i.e. T1.C1 = T2.C1 + 3)

Tuning Opportunities

- Missing statistics
  - Columns with no statistics
  - No frequency statistics on columns that are likely skewed
    - INACTIVE_IND = 'N'
    - ACTIVITY_CODE = 'A'
  - No histogram statistics for range predicates
    - EFF_DT > '2011-07-31'

Mining DSN_PREDICAT_TABLE – Missing Statistics

- Columns with no statistics

```
SELECT DISTINCT
       SUBSTR(C.TBCREATOR,1,18) TBCREATOR
      ,SUBSTR(C.TBNAME,1,18) TBNAME
      ,SUBSTR(C.NAME,1,18) NAME
FROM DSN_PREDICAT_TABLE P
INNER JOIN PLAN_TABLE PL
   ON PL.QUERYNO = P.QUERYNO
  AND PL.QBLOCKNO = P.QBLOCKNO
  AND PL.APPLNAME = P.APPLNAME
  AND PL.PROGNAME = P.PROGNAME
  AND PL.BIND_TIME = P.EXPLAIN_TIME
  AND PL.TABNO = P.LHS_TABNO
INNER JOIN SYSIBM.SYSCOLUMNS C
   ON C.TBCREATOR = PL.CREATOR
  AND C.TBNAME = PL.TNAME
  AND C.NAME = P.LEFT_HAND_SIDE
WHERE P.LHS_TABNO <> 0
  AND C.COLCARDF < 0
UNION
```

```
SELECT DISTINCT
       SUBSTR(C.TBCREATOR,1,18) TBCREATOR
      ,SUBSTR(C.TBNAME,1,18) TBNAME
      ,SUBSTR(C.NAME,1,18) NAME
FROM DSN_PREDICAT_TABLE P
INNER JOIN PLAN_TABLE PL
   ON PL.QUERYNO = P.QUERYNO
  AND PL.QBLOCKNO = P.QBLOCKNO
  AND PL.APPLNAME = P.APPLNAME
  AND PL.PROGNAME = P.PROGNAME
  AND PL.BIND_TIME = P.EXPLAIN_TIME
  AND PL.TABNO = P.RHS_TABNO
INNER JOIN SYSIBM.SYSCOLUMNS C
   ON C.TBCREATOR = PL.CREATOR
  AND C.TBNAME = PL.TNAME
  AND C.NAME = P.RIGHT_HAND_SIDE
WHERE P.RHS_TABNO <> 0
  AND C.COLCARDF < 0
ORDER BY 1,2,3
```

Identify columns on the LHS

Find columns with no statistics

Identify columns on the RHS

Union results from LHS and RHS of predicate

Mining DSN_PREDICAT_TABLE – Missing Statistics

- Missing frequency statistics

```
SELECT DISTINCT
       SUBSTR(C.TBCREATOR,1,18) TBCREATOR
      ,SUBSTR(C.TBNAME,1,18) TBNAME
      ,SUBSTR(C.NAME,1,18) NAME
      ,C.COLCARDF
FROM DSN_PREDICAT_TABLE P
INNER JOIN PLAN_TABLE PL
   ON PL.QUERYNO = P.QUERYNO
  AND PL.QBLOCKNO = P.QBLOCKNO
  AND PL.APPLNAME = P.APPLNAME
  AND PL.PROGNAME = P.PROGNAME
  AND PL.BIND_TIME = P.EXPLAIN_TIME
  AND PL.TABNO = P.LHS_TABNO
INNER JOIN SYSIBM.SYSCOLUMNS C
   ON C.TBCREATOR = PL.CREATOR
  AND C.TBNAME = PL.TNAME
  AND C.NAME = P.LEFT_HAND_SIDE
```

```
WHERE P.LHS_TABNO <> 0
  AND P.RIGHT_HAND_SIDE = 'VALUE'
  AND C.COLCARDF < 100
  AND C.COLCARDF <> -1
  AND (P.MARKER = 'N'
    OR C.NULLS = 'Y')
  AND NOT EXISTS (
    SELECT 1
    FROM SYSIBM.SYSCOLDIST CD
    WHERE CD.TBOWNER = C.TBCREATOR
      AND CD.TBNAME = C.TBNAME
      AND CD.NAME = C.NAME
      AND CD.TYPE = 'F'
      AND CD.NUMCOLUMNS = 1
  )
ORDER BY 1,2,3
```

'COL op value' predicate

Low cardinality column

Need a nullable column
Or predicate w/o hv

No frequency statistics
collected yet

- Why low cardinality columns?
  - More opportunity for data skew
  - Experience shows many low cardinality columns are skewed
- Why nullable column or predicate without host variable?
  - Predicates with a host variable can only use a null frequency statistic
  - But what about REOPT?

Mining DSN_PREDICAT_TABLE – Missing Statistics

- Missing histogram statistics for range predicates

```
SELECT DISTINCT
       SUBSTR(C.TBCREATOR,1,18) TBCREATOR
      ,SUBSTR(C.TBNAME,1,18) TBNAME
      ,SUBSTR(C.NAME,1,18) NAME
FROM DSN_PREDICAT_TABLE P
INNER JOIN PLAN_TABLE PL
   ON PL.QUERYNO = P.QUERYNO
  AND PL.QBLOCKNO = P.QBLOCKNO
  AND PL.APPLNAME = P.APPLNAME
  AND PL.PROGNAME = P.PROGNAME
  AND PL.BIND_TIME = P.EXPLAIN_TIME
  AND PL.TABNO = P.LHS_TABNO
INNER JOIN SYSIBM.SYSCOLUMNS C
   ON C.TBCREATOR = PL.CREATOR
  AND C.TBNAME = PL.TNAME
  AND C.NAME = P.LEFT_HAND_SIDE
```

```
WHERE P.LHS_TABNO <> 0
  AND P.RIGHT_HAND_SIDE = 'VALUE'
  AND P.TYPE IN ('RANGE',
      'BETWEEN','LIKE','NOT LIKE')
    AND (P.MARKER = 'N'
    OR C.NULLS = 'Y')
  AND NOT EXISTS (
    SELECT 1
    FROM SYSIBM.SYSCOLDIST CD
    WHERE CD.TBOWNER = C.TBCREATOR
      AND CD.TBNAME = C.TBNAME
      AND CD.NAME = C.NAME
      AND CD.TYPE = 'H'
      AND CD.NUMCOLUMNS = 1
  )
ORDER BY 1,2,3
```

'COL op value' predicate

Range, between or like predicate

Need a nullable column

Or predicate w/o hv

No histogram statistics collected yet

  - Again, what about REOPT?

Mining DSN_PREDICAT_TABLE – Missing Statistics

- Will these queries identify all missing statistics?
  - No, but they will identify common problems.
  - What else could be missing?
    - Correlation statistics (multi-column cardinality)
      - → Recommendation is to collect KEYCARD statistics on all indexes (default in DB2 10)
    - Histogram statistics for use outside range FF estimation
      - → Parallelism cutting
      - → Join FF estimation
    - Index statistics
      - → Easy to check – query SYSINDEXES
    - Multi-column frequency statistics

REOPT

- So, what about REOPT?
  - Optimizer cannot use frequency/histogram statistics (except NULL) with a host variable
  - This does not mean FF estimation with the host variable is accurate
    - Especially for range predicates with host variables
  - Determine if optimizer's estimated FF varies enough from the actual FF at runtime to require REOPT for a good, stable access path
    - If so, missing frequency/histogram statistics can be identified by...
      - → Removing the P.MARKER / C.NULLS predicates from earlier queries or...
      - → Adding a join to check if SYSPACKSTMT.STATUS = 'G'

REOPT

- So, what about REOPT?
  - Consider REOPT options and associated overhead
    - REOPT(ALWAYS)
      - → Likely not acceptable for transactional SQL
      - → Potentially a good choice for longer running SQL
    - REOPT(ONCE)
      - → Reduce the prepare impact
      - → First set of values could be unusual resulting in a sub-optimal cached path for future executions

Tuning Opportunities

- Identify potential indexes
  - Are there columns not indexed that...
    - Have high cardinality
    - Occur often in equal predicates
  - Are there predicates with a low filter factor on a column that is not indexed?

Mining DSN_PREDICAT_TABLE – Identify potential indexes

- Are there columns not indexed that...
  - Have high cardinality
  - Occur often in local equal predicates

```
SELECT SUBSTR(T.CREATOR,1,18) TBCREATOR
      ,SUBSTR(T.NAME,1,18) TBNAME
      ,SUBSTR(C.NAME,1,18) COLNAME
      ,C.COLCARDF / T.CARDF RATIO
      ,COUNT(*) AS CNT
FROM DSN_PREDICAT_TABLE P
INNER JOIN PLAN_TABLE PL
   ON PL.QUERYNO = P.QUERYNO
  AND PL.QBLOCKNO = P.QBLOCKNO
  AND PL.APPLNAME = P.APPLNAME
  AND PL.PROGNAME = P.PROGNAME
  AND PL.BIND_TIME = P.EXPLAIN_TIME
  AND PL.TABNO = P.LHS_TABNO
INNER JOIN SYSIBM.SYSCOLUMNS C
   ON C.TBCREATOR = PL.CREATOR
  AND C.TBNAME = PL.TNAME
  AND C.NAME = P.LEFT_HAND_SIDE
INNER JOIN SYSIBM.SYSTABLES T
   ON T.CREATOR = PL.CREATOR
  AND T.NAME = PL.TNAME
WHERE P.LHS_TABNO <> 0
  AND P.RHS_TABNO = 0
  AND P.TYPE = 'EQUAL'
  AND P.SEARCHARG = 'Y'
  AND P.NEGATION = 'N'
  AND C.COLCARDF / T.CARDF > 0.2
  AND T.CARDF > 100
  AND NOT EXISTS (
     SELECT 1
     FROM SYSIBM.SYSINDEXES IX
     INNER JOIN SYSIBM.SYSKEYS K
        ON K.IXCREATOR = IX.CREATOR
       AND K.IXNAME = IX.NAME
     WHERE IX.TBCREATOR = PL.CREATOR
       AND IX.TBNAME = PL.TNAME
       AND K.COLNAME = P.LEFT_HAND_SIDE
  )
GROUP BY T.CREATOR, T.NAME, C.NAME
        ,T.CARDF, C.COLCARDF
HAVING COUNT(*) > 2
ORDER BY CNT DESC
```

Local indexable equal predicate

High column cardinality compared to table cardinality

No index exists on this column

Occurs “often”

- What about join predicates?
  - Same logic, just need to check LHS and RHS and remove RIGHT_HAND_SIDE = 'VALUE' predicate

Mining DSN_PREDICAT_TABLE – Identify potential indexes

- What about cases where an index exists, but cannot be matching?
  - Modify the NOT EXISTS predicate

```
AND NOT EXISTS (
   SELECT 1
   FROM SYSIBM.SYSINDEXES IX
   INNER JOIN SYSIBM.SYSKEYS K
      ON K.IXCREATOR = IX.CREATOR
     AND K.IXNAME = IX.NAME
   INNER JOIN TABLE(
      SELECT COUNT(*) AS CNT
            ,SUM(CASE WHEN A.CNT = A.NUM_IN THEN 1
                      ELSE 0 END) AS NUM_IN
      FROM (
         SELECT COUNT(*) AS CNT
               ,SUM(CASE WHEN P2.TYPE='IN' THEN 1
                         ELSE 0 END) AS NUM_IN
         FROM DSN_PREDICAT_TABLE P2
             ,SYSIBM.SYSKEYS K2
         WHERE K2.IXCREATOR = K.IXCREATOR
           AND K2.IXNAME = K.IXNAME
           AND P2.QUERYNO = P.QUERYNO
           AND P2.QBLOCKNO = P.QBLOCKNO
           AND P2.APPLNAME = P.APPLNAME
           AND P2.PROGNAME = P.PROGNAME
           AND P2.EXPLAIN_TIME = P.EXPLAIN_TIME
           AND K2.COLSEQ < K.COLSEQ
           AND ((P2.LHS_TABNO=PL.TABNO AND
                 K2.COLNAME=P2.LEFT_HAND_SIDE)
             OR (P2.RHS_TABNO=PL.TABNO AND
                 K2.COLNAME=P2.RIGHT_HAND_SIDE)
           )
           AND P2.TYPE IN ('EQUAL','IN')
           AND P2.SEARCHARG = 'Y'
           AND P2.NEGATION = 'N'
         GROUP BY K2.COLNO
      ) A
   ) PREV_KEYS
     ON K.COLSEQ = PREV_KEYS.CNT - 1
   WHERE IX.TBCREATOR = PL.CREATOR
     AND IX.TBNAME = PL.TNAME
     AND K.COLNAME = P.LEFT_HAND_SIDE
     AND PREV_KEYS.NUM_IN <= 1
```

Look for indexable predicates for previous keys that don't stop matching

All previous keys can match

No more than 1 IN predicate exists

Mining DSN_PREDICAT_TABLE – Identify potential indexes

- Are there columns not indexed that have a good filter factor

```
SELECT SUBSTR(T.CREATOR,1,18) TBCREATOR
      ,SUBSTR(T.NAME,1,18) TBNAME
      ,SUBSTR(C.NAME,1,18) COLNAME
      ,P.FILTER_FACTOR * T.CARDF QUAL_ROWS
      ,COUNT(*) AS CNT
FROM DSN_PREDICAT_TABLE P
INNER JOIN PLAN_TABLE PL
   ON PL.QUERYNO = P.QUERYNO
  AND PL.QBLOCKNO = P.QBLOCKNO
  AND PL.APPLNAME = P.APPLNAME
  AND PL.PROGNAME = P.PROGNAME
  AND PL.BIND_TIME = P.EXPLAIN_TIME
  AND PL.TABNO = P.LHS_TABNO
INNER JOIN SYSIBM.SYSCOLUMNS C
   ON C.TBCREATOR = PL.CREATOR
  AND C.TBNAME = PL.TNAME
  AND C.NAME = P.LEFT_HAND_SIDE
INNER JOIN SYSIBM.SYSTABLES T
   ON T.CREATOR = PL.CREATOR
  AND T.NAME = PL.TNAME
WHERE P.LHS_TABNO <> 0
  AND P.RHS_TABNO = 0
  AND P.SEARCHARG = 'Y'
  AND P.NEGATION = 'N'
  AND P.FILTER_FACTOR * T.CARDF < 1000
  AND T.CARDF > 100
  AND NOT EXISTS (
     ...
  )
GROUP BY T.CREATOR, T.NAME, C.NAME
        ,T.CARDF, C.COLCARDF
HAVING COUNT(*) > 2
ORDER BY CNT DESC
```

Local indexable predicate

Less than 1000 rows expected to qualify

No matching index support

Occurs “often”

  - What if the filter factor is wrong?
    - Similar problem to comparing costs – estimates are only as good as inputs
      - → Are statistics up to date on this column?
      - → Is the filter factor a default?

Tuning Opportunities

- Queries at risk
  - Default filter factors
    - Range predicates with host variable / parameter marker
    - Column to column range predicates
    - Indexable range predicates with host variables / parameter markers
  - Data skew
    - Indexable predicates with host variables / parameter markers on skewed columns
  - Multiple risky index choices
    - More than one index choice on a range predicate with host variable / parameter marker
    - Some other combination of above

Mining DSN_PREDICAT_TABLE – Queries at risk

- Identify queries containing predicates with default filter factors
  - Range predicates with host variables / parameter markers

```
SELECT QUERYNO
      ,QBLOCKNO
      ,APPLNAME
      ,SUBSTR(PROGNAME,1,18) PROGNAME
      ,SUBSTR(LEFT_HAND_SIDE,1,18) COLNAME
      ,SUBSTR(TEXT,1,100) TEXT
FROM DSN_PREDICAT_TABLE
WHERE LHS_TABNO <> 0
  AND TYPE IN ('RANGE','BETWEEN','LIKE')
  AND RHS_TABNO = 0
  AND MARKER = 'Y'
```

Local range predicate

Contains host variable / parameter marker

  - Limited usefulness across an entire workload since this is a common pattern
  - Among problem queries, could be a good indicator for REOPT consideration

  - This doesn't mean optimizer cannot pick a good path for these queries
  - This does mean that optimizer is depending on a default guess
    - More likely to see access path problems

Mining DSN_PREDICAT_TABLE – Queries at risk

- Identify queries containing column to column range predicates
  - C1 < C2

```
SELECT QUERYNO
      ,QBLOCKNO
      ,APPLNAME
      ,SUBSTR(PROGNAME,1,18) PROGNAME
      ,SUBSTR(TEXT,1,100) TEXT
FROM DSN_PREDICAT_TABLE
WHERE LHS_TABNO <> 0
  AND RHS_TABNO <> 0
  AND TYPE IN ('RANGE','BETWEEN','LIKE')
```

Columns on both sides

Local range predicate

  - Histogram statistics on both sides can help
  - Design to avoid risky choices
    - Is there a clear choice access path?
    - Is there an index on this range predicate that might look attractive, but is not?

Mining DSN_PREDICAT_TABLE – Queries at risk

- Indexable local range predicates on indexed columns with default filter factors

```
SELECT DISTINCT
       P.QUERYNO
      ,P.QBLOCKNO
      ,P.APPLNAME
      ,SUBSTR(P.PROGNAME,1,18) PROGNAME
      ,SUBSTR(T.NAME,1,18) TBNAME
      ,SUBSTR(P.LEFT_HAND_SIDE,1,18) COLNAME
      ,SUBSTR(P.TEXT,1,100) TEXT
FROM DSN_PREDICAT_TABLE P
INNER JOIN PLAN_TABLE PL
   ON PL.QUERYNO = P.QUERYNO
  AND PL.QBLOCKNO = P.QBLOCKNO
  AND PL.APPLNAME = P.APPLNAME
  AND PL.PROGNAME = P.PROGNAME
  AND PL.BIND_TIME = P.EXPLAIN_TIME
  AND PL.TABNO = P.LHS_TABNO
INNER JOIN SYSIBM.SYSCOLUMNS C
   ON C.TBCREATOR = PL.CREATOR
  AND C.TBNAME = PL.TNAME
  AND C.NAME = P.LEFT_HAND_SIDE
INNER JOIN SYSIBM.SYSTABLES T
   ON T.CREATOR = PL.CREATOR
  AND T.NAME = PL.TNAME
INNER JOIN SYSIBM.SYSINDEXES IX
   ON T.CREATOR = IX.TBCREATOR
  AND T.NAME = IX.TBNAME
INNER JOIN SYSIBM.SYSKEYS K
   ON K.IXCREATOR = IX.CREATOR
  AND K.IXNAME = IX.NAME
  AND K.COLNAME = P.LEFT_HAND_SIDE
WHERE P.LHS_TABNO <> 0
  AND P.RHS_TABNO = 0
  AND P.SEARCHARG = 'Y'
  AND P.NEGATION  = 'N'
  AND P.MARKER = 'Y'
  AND P.TYPE IN ('RANGE','BETWEEN','LIKE')
```

Column is indexed

Indexable range predicate with a marker

  - More useful than local range predicates with markers
    - Cases where join order / join sequence are impacted by the incorrect filter factor are missed
  - There could be other, significantly better indexes such that this risky choice doesn't compete
  - Candidates for REOPT

Mining DSN_PREDICAT_TABLE – Queries at risk

- Indexable local range predicates with default filter factors on more than one index

```
SELECT QUERYNO
      ,QBLOCKNO
      ,APPLNAME
      ,SUBSTR(PROGNAME,1,18) PROGNAME
      ,COUNT(*)
FROM (
  SELECT DISTINCT
         P.QUERYNO
        ,P.QBLOCKNO
        ,P.APPLNAME
        ,P.PROGNAME
        ,IX.CREATOR
        ,IX.NAME
  FROM DSN_PREDICAT_TABLE P
  INNER JOIN PLAN_TABLE PL
     ON PL.QUERYNO = P.QUERYNO
    AND PL.QBLOCKNO = P.QBLOCKNO
    AND PL.APPLNAME = P.APPLNAME
    AND PL.PROGNAME = P.PROGNAME
    AND PL.BIND_TIME = P.EXPLAIN_TIME
    AND PL.TABNO = P.LHS_TABNO
  INNER JOIN SYSIBM.SYSCOLUMNS C
     ON C.TBCREATOR = PL.CREATOR
    AND C.TBNAME = PL.TNAME
    AND C.NAME = P.LEFT_HAND_SIDE
  INNER JOIN SYSIBM.SYSTABLES T
     ON T.CREATOR = PL.CREATOR
    AND T.NAME = PL.TNAME
  INNER JOIN SYSIBM.SYSINDEXES IX
     ON T.CREATOR = IX.TBCREATOR
    AND T.NAME = IX.TBNAME
  INNER JOIN SYSIBM.SYSKEYS K
     ON K.IXCREATOR = IX.CREATOR
    AND K.IXNAME = IX.NAME
    AND K.COLNAME = P.LEFT_HAND_SIDE
  WHERE P.LHS_TABNO <> 0
    AND P.RHS_TABNO = 0
    AND P.SEARCHARG = 'Y'
    AND P.NEGATION  = 'N'
    AND P.MARKER = 'Y'
    AND P.TYPE IN ('RANGE','BETWEEN','LIKE')
) A
GROUP BY QUERYNO, QBLOCKNO, APPLNAME, PROGNAME
HAVING COUNT(*)>1
```

Same conditions as before

More than one index

  - Similar considerations to previous example
    - More likely to cause query instability since there are multiple index choices competing for which optimizer doesn't have accurate filtering
  - Mining query could report cases in which multiple indexes filter on the same columns → Less risky
  - All are candidates for REOPT

Tuning Opportunities

- Queries at risk
  - Default filter factors
    - Range predicates with host variable / parameter marker
    - Column to column range predicates
    - Indexable range predicates with host variables / parameter markers
  - Data skew
    - Indexable predicates with host variables / parameter markers on skewed columns
  - Multiple risky index choices
    - More than one index choice on a range predicate with host variable / parameter marker
    - Some other combination of above

Conclusion – What did we discuss?

- Primary cause of access path problems
  - Inaccurate selectivity estimation
- Populating DSN_PREDICAT_TABLE
- Using DSN_PREDICAT_TABLE to identify query tuning opportunities
  - Missing statistics
  - Query rewrites
  - New index candidates
  - At risk queries

Optimize applications using extended explain tables

Jase Alpers

IBM

alpersj@us.ibm.com