Db2Night Show

Favorite Db2 Performance and Optimization Features V9-V12

Tony Andrews *tandrews@themisinc.com*
Twitter @tonyandrews12

Follow @ThemisTraining

Themis
Leaders in IT Education

Agenda

- Does the optimizer actually rewrite some coded predicates. At times, yes!
- Does the optimizer actually rewrites the query. At times, yes!
- Learn many of the optimization features from V9 through V12
- What is transitive closure? What is safe query optimization? What are filter factors? What is global query optimization? What is safe query optimization?   SQL Pagination? Other questions?

V9 – Correlate/De-Correlate

```
SELECT E.EMPNO,  E.LASTNAME, E.SALARY
FROM EMP  E
WHERE E.EMPNO IN
      (SELECT EP.EMPNO
        FROM EMPPROJACT  EP
        WHERE EP.PROJNO = 'IF2000' )
```

QBK	METH	TNAME	ATYPE	MCOLS	ACCESS NAME	IX ONLY	SCU	SCJ	SCO	SCG	PRF	COLFN	QB TYP
1	0	DSNWFQB(02)	R	0							S		SELECT
1	1	EMP	I	1	XEMP1	N							SELECT
2	0	EMPPROJACT	I	1	XEMPPROJACT1	Y							NCORSUB
2	3						Y		Y				NCORSUB

```
EMPNO   LASTNAME          SALARY
------  -----------       ------------
000030  KWAN              38250.00
000140  NICHOLLS          28420.00

DSNE610I NUMBER OF ROWS DISPLAYED IS 2
```

V9 – Optimizer will sometime rewrite a correlated subquery to a non-correlated subquery and vice versa.

Notes: This query stayed as non-correlated.
        Subquery put entries into a workfile (DSNWFQB1) ➔ dataset workfile specific to query block 1

V9 – Correlate/De-Correlate EXISTS

```
SELECT E.EMPNO,  E.LASTNAME, E.SALARY
FROM EMP  E
WHERE EXISTS
    (SELECT 1
     FROM EMPPROJACT   EP
     WHERE E.EMPNO = EP.EMPNO
        AND EP.PROJNO = 'IF2000' )
```

Correlated Subquery gets the same results.

QBK	METH	TNAME	ATYPE	MCOLS	ACCESS NAME	IX ONLY	SCU	SCJ	SCO	SCG	PRF	COLFN	QB TYP
1	0	DSNWFQB(02)	R	0							S		SELECT
1	1	EMP	I	1	XEMP1	N							SELECT
2	0	EMPPROJACT	I	1	XEMPPROJACT1	Y							NCORSUB
2	3						Y		Y				NCORSUB

```
EMPNO   LASTNAME          SALARY
------  -----------       ------------
000030  KWAN              38250.00
000140  NICHOLLS          28420.00

DSNE610I NUMBER OF ROWS DISPLAYED IS 2
```

Optimizer Query Rewrite

The other way to eliminate duplicates is by coding the SQL using a Correlated Subquery with the Exists clause.

Note: Optimizer rewrote the correlated exists logic to a non-correlated subquery (exactly like the previous page)

Predicate Generation Through Transitive Closure

<u>The Premise</u>

If A must equal B

And A must be RED,

Then B must also be RED.

Predicate Generation Through Transitive Closure Cont'd

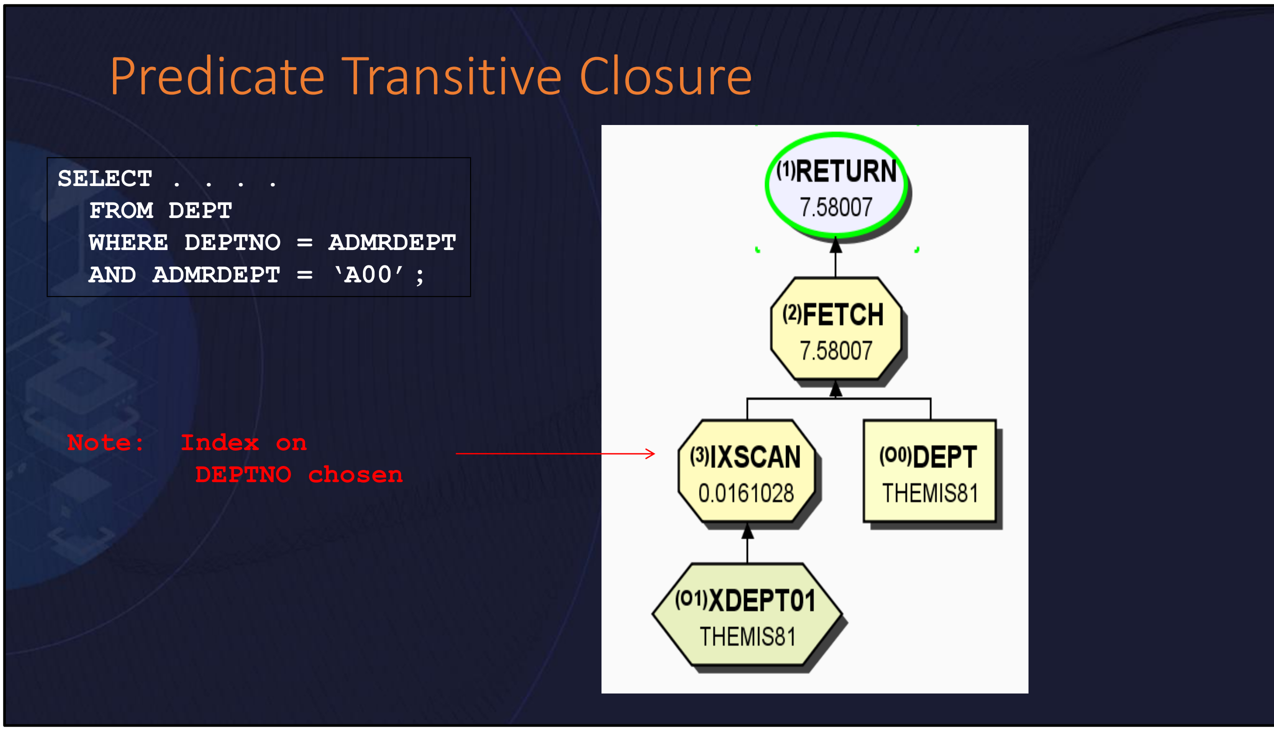
Single Table Db2 Generated Predicate

Index XDEPT1 on DEPTNO
Index XDEPT3 on ADMRDEPT

```
SELECT . . . .
  FROM DEPT
  WHERE DEPTNO = ADMRDEPT
  AND ADMRDEPT = 'A00' ;
```

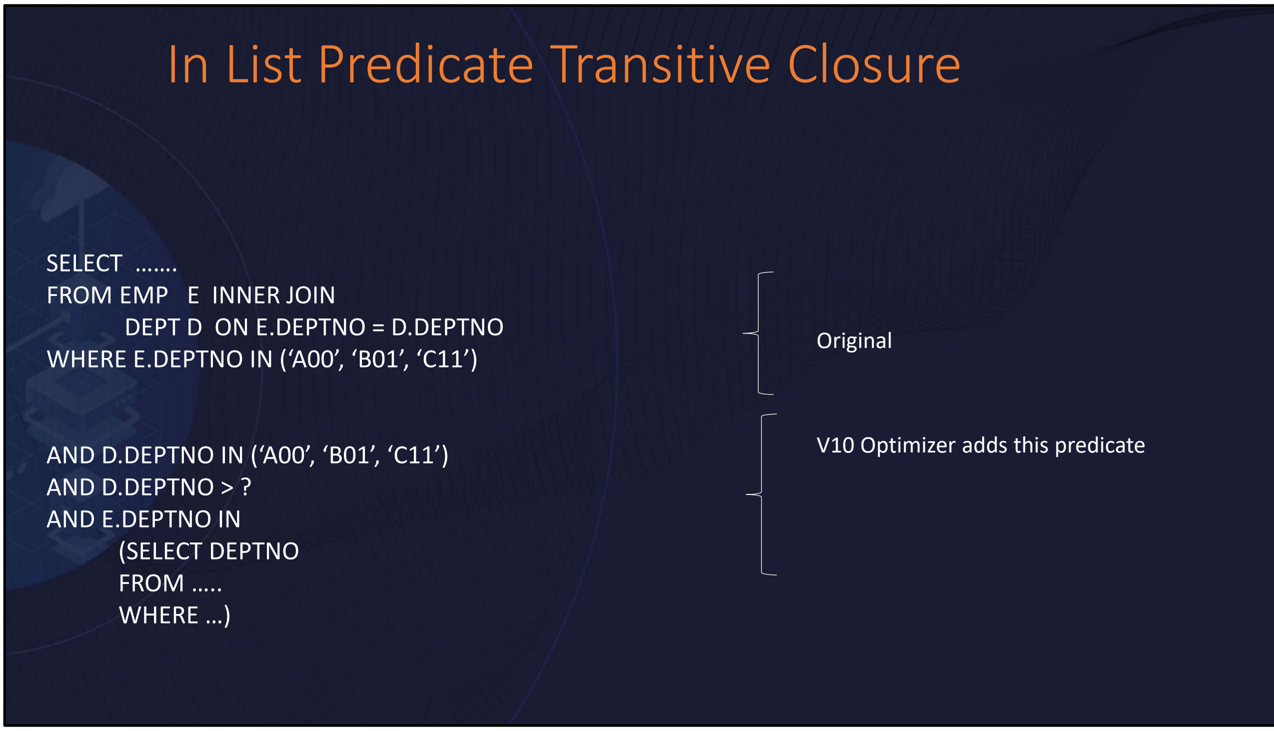
```
SELECT . . . .
 FROM DEPT
 WHERE DEPTNO = ADMRDEPT
 AND ADMRDEPT = 'A00'
 AND DEPTNO = 'A00' ;
```

XDEPT1 index chosen !



Visual Explain will show all predicates used by the optimizer, both those included in the query and those generated by predicate transitive closure.

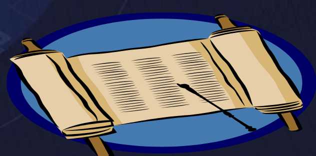
Why does the optimizer do the predicate transitive closure? It provides the optimizer a possible other index to evaluate.



All predicates will get transitive closure except the LIKE predicate. So developers need to code it themselves.

Single Matching Indexing With ‘Or’ Predicates

- Scrolling Performance Issues
- AKA Pagination
- Complex ‘Or’ Predicate logic

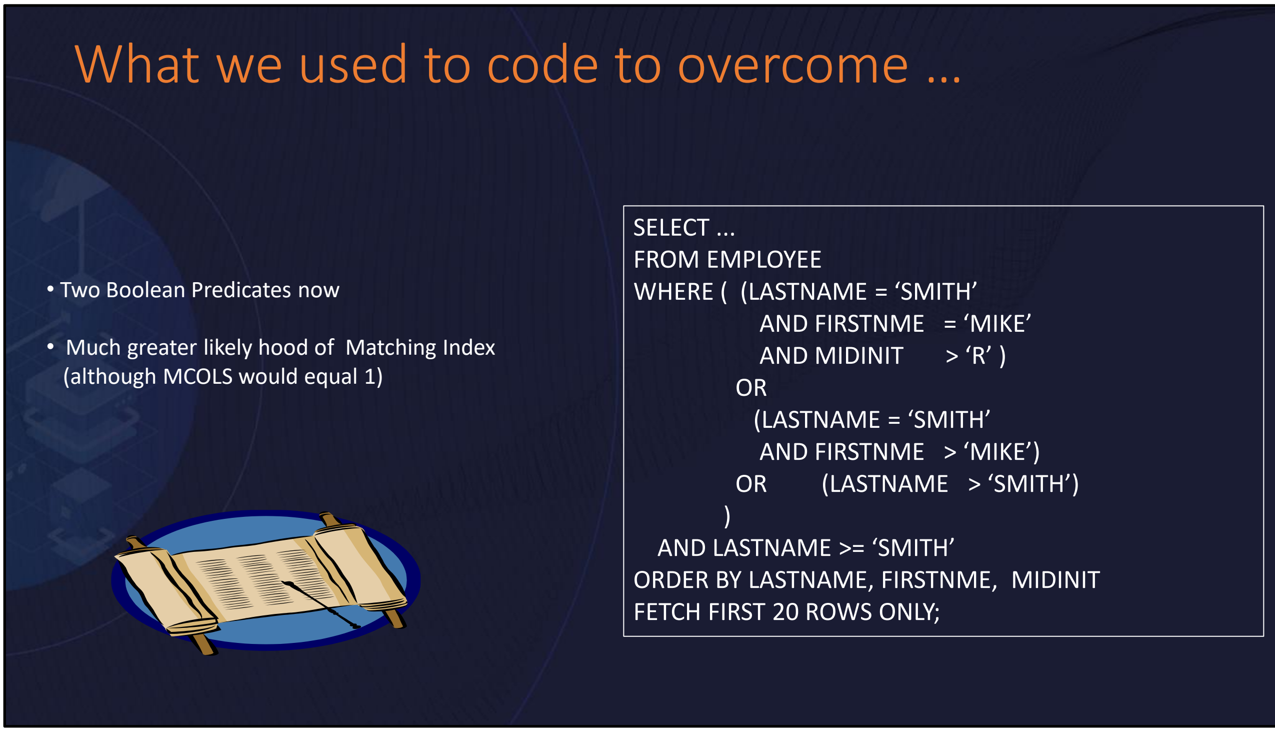


```
SELECT ...
FROM EMPLOYEE
WHERE (LASTNAME = ‘SMITH’
      AND FIRSTNME   = ‘MIKE’
      AND MIDINIT      > ‘R’ )
OR
             (LASTNAME = ‘SMITH’
        AND FIRSTNME   > ‘MIKE’)
OR        (LASTNAME   > ‘SMITH’)
ORDER BY LASTNAME, FIRSTNME,
                   MIDINIT
FETCH FIRST 20 ROWS ONLY;
```

Prior to V10, a way to get better performance from these is to rewrite them as Boolean type predicates. See below.

A better rewrite for older versions: Predicates AND’d together (Boolean) are typically more efficient then predicates OR’d together.

```
SELECT ...
FROM EMPLOYEE
WHERE (  (LASTNAME = ‘SMITH’
               AND FIRSTNME   = ‘MIKE’
               AND MIDINIT      > ‘R’ )
            OR
              (LASTNAME = ‘SMITH’
               AND FIRSTNME   > ‘MIKE’)
            OR        (LASTNAME   > ‘SMITH’)
          )
   AND LASTNAME >= ‘SMITH’
ORDER BY LASTNAME, FIRSTNME,  MIDINIT
FETCH FIRST 20 ROWS ONLY;
```



Prior to V10, a way to get better performance from these is to rewrite them as Boolean type predicates. See below.

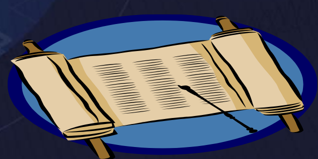
A better rewrite for older versions: Predicates AND’d together (Boolean) are typically more efficient then predicates OR’d together.

```
SELECT …
FROM EMPLOYEE
WHERE (  (LASTNAME = ‘SMITH’
           AND FIRSTNME   = ‘MIKE’
           AND MIDINIT      > ‘R’ )
         OR
          (LASTNAME = ‘SMITH’
           AND FIRSTNME   > ‘MIKE’)
         OR       (LASTNAME   > ‘SMITH’)
        )
   AND LASTNAME >= ‘SMITH’
ORDER BY LASTNAME, FIRSTNME,  MIDINIT
FETCH FIRST 20 ROWS ONLY;
```

Single Matching Indexing With 'Or' Predicates

The Process

- Process the first predicate until 20 rows are met, or end of data
- Process the second predicate only if needed
- And so on .....



```
SELECT ...
FROM EMPLOYEE
WHERE (LASTNAME = 'SMITH'
      AND FIRSTNME   = 'MIKE'
      AND MIDINIT      > 'R' )
OR
            (LASTNAME = 'SMITH'
        AND FIRSTNME   > 'MIKE')
OR        (LASTNAME   > 'SMITH')
ORDER BY LASTNAME, FIRSTNME,
                  MIDINIT
FETCH FIRST 20 ROWS ONLY;
```

Prior to V10: Most likely multi index processing
V10: Matching single index access

Single Matching Indexing With 'Or' Predicates

This new processing method is shown on the PLAN_TABLE as ACCESSTYPE = 'NR' with a line for each predicate involved.

QBLOCKNO	PLANNO	METHOD	CREATOR	TNAME	ACCESSTYPE	MATCHCOLS
1	1	0	THEMIS81	EMP	NR	3
1	1	0	THEMIS81	EMP	NR	2
1	1	0	THEMIS81	EMP	NR	1

NR: Range List Index Scan

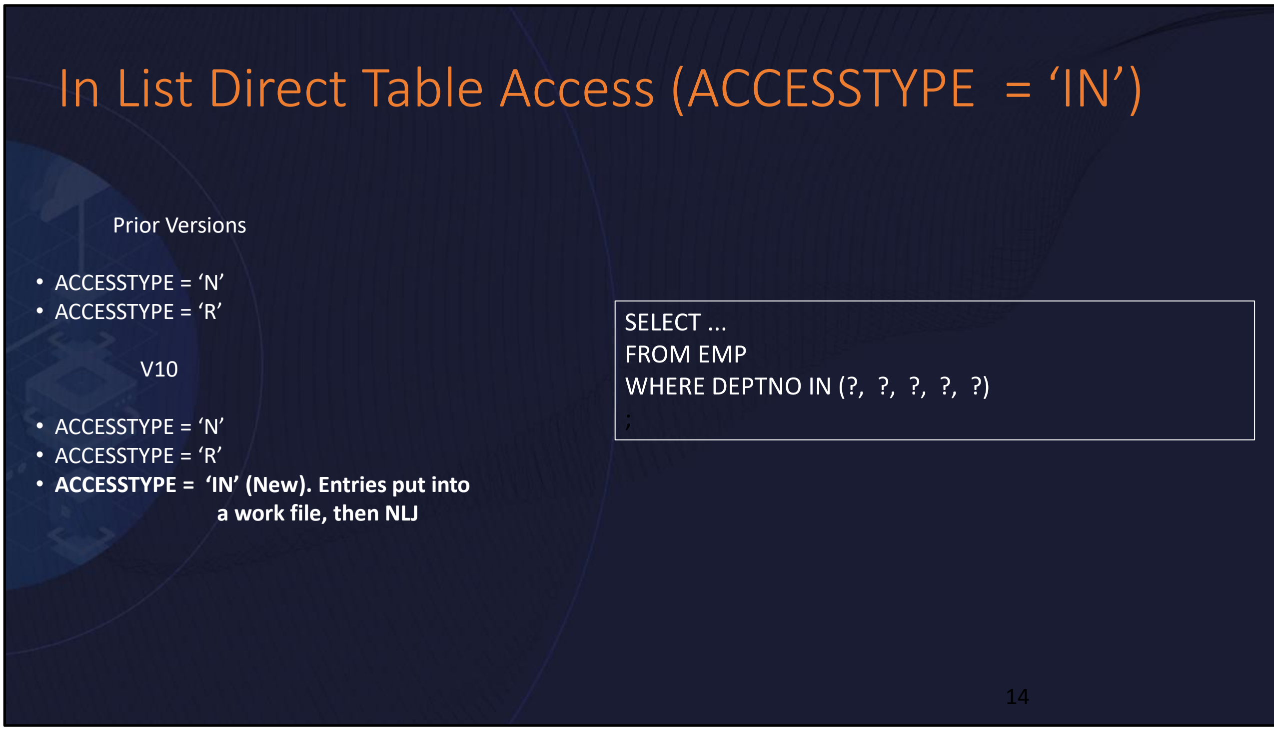
Begins first with the 3 matching columns, then the 2, then the one in order to fulfill the result set. The order in the PLAN_TABLE will be the sequence it has coded in the SQL. This allows the customer to match the PLAN_TABLE to the SQL. The actual order of execution will be dependent on the literal values used at runtime.

The DB2 implementation of range-list will re-order the OR conditions at runtime based upon the literal values. Thus, it is not possible for BIND/PREPARE to know the order in which these will be executed.

Available with Db2 12 is data-dependent pagination, which uses row value expressions in a basic predicate. This enables a query o access part of a Db2 result table based on a logical key value:

With the additional comparison operators supported with row-value-expression comparisons, application developers can choose to simplify their SQL and potentially make their applications more readable. AND… the optimization will most likely take on the the 'NR' processing if an ' Optimize for n Rows' or 'Fetch first n Rows' is coded. And… if all the 'OR' predicates map to the same index.

'NR': What is important to understand is how many MATCHCOLS for each 'OR' predicate. The order in the PLAN_TABLE will be the sequence the predicates were coded in the SQL. This allows us to easily match the PLAN_TABLE to the SQL. The actual order of execution will be dependent on the literal values used at runtime.



With 'In' predicates that match indexes, Db2 may choose to process via ACCESSTYPE 'N', 'R', or now 'IN'. This all depends on the filter factor for the predicates and how many rows Db2 thinks the predicate will affect.

ACCESSTYPE 'N': An IN-list scan can be thought of as a series of matching index scans with the values in the IN predicate being used for each successive equal matching index scan. There will be a matching index scan for each value in the list.
ACCESSTYPE 'R': If Db2 thinks that the values in the list will affect a high percentage of rows in the table (or a high percentage of pages in the table), then it will choose a tablespace scan for processing.

ACCESSTYPE 'IN': This was new in V10. Db2 will load the list values into an IN-Memory table and use that table as the composite table for a nested loop join process. Also shows table type = 'I'. For example:

QBLOCKNO	PLANNO	METHOD	CREATOR	TNAME	ATYPE	MCOLS	TABLE_TYPE
1	1	0	ODYTA	DSNIN001	IN	0	I
1	2	1	THEMIS81	EMP	I	1	T

The naming convention for the in-memory tables is as follows:
DSNIN indicates that it relates to IN-list.
The number after DSNIN (001) represents the predicate number.
A number in parenthesis represents the query block number.

In List Direct Table Access (ACCESSTYPE = ‘IN’)

QBLOCKNO	PLANNO	METHOD	CREATOR	TNAME	ATYPE	MCOLS	TABLE_TYPE
1	1	0	ODYTA	DSNIN001	IN	0	I
1	2	1	THEMIS81	EMP	I	1	T

The naming convention for the in-memory tables is as follows:

- DSNIN indicates that it relates to IN-list.
- The number after DSNIN (001) represents the predicate number.
- A number in parenthesis represents the query block number.

With ‘In’ predicates that match indexes, Db2 may choose to process via ACCESSTYPE ‘N’, ‘R’, or now ‘IN’. This all depends on the filter factor for the predicates and how many rows Db2 thinks the predicate will affect.

ACCESSTYPE ‘N’: An IN-list scan can be thought of as a series of matching index scans with the values in the IN predicate being used for each successive equal matching index scan. There will be a matching index scan for each value in the list.

ACCESSTYPE ‘R’: If Db2 thinks that the values in the list will affect a high percentage of rows in the table (or a high percentage of pages in the table), then it will choose a tablespace scan for processing.

ACCESSTYPE ‘IN’: This is new in V10. Db2 will load the list values into an IN-Memory table and use that table as the composite table for a nested loop join process. Also shows table type = ‘I’.

In List Direct Table Access (ACCESSTYPE = ‘IN’)

Biggest advantage is with an ‘In’ predicate on the first 2 columns of an index.

- V9 uses only first predicate
- V10 combines both predicates

```
SELECT ...
FROM EMP
WHERE LASTNAME IN (?,  ?,  ?,  ?)
      AND FIRSTNME IN  (?, ?, ?)
```

16

The biggest advantage of this is now even with 2 IN=list predicates, Db2 can make use of in-memory work files providing tremendous benefits. This would be when the IN-list predicates represent two or more leading columns of an index. Db2 10 can combine the IN-list predicates to reduce the number of index getpages and list prefetch operations. Db2 9 and prior versions could do a 1 column match with multiple In-list predicates, and the other predicate would show as a screening predicate.

In List Direct Table Access (ACCESSTYPE = 'IN')

Explain prior to V10: Note matching columns of 1.

QBLOCKNO	PLANNO	METHOD	CREATOR	TNAME	ATYPE	MCOLS	TABLE_TYPE
1	1	0	THEMIS81	EMP	N	1	T

V10 Explain: Note matching columns of 2.

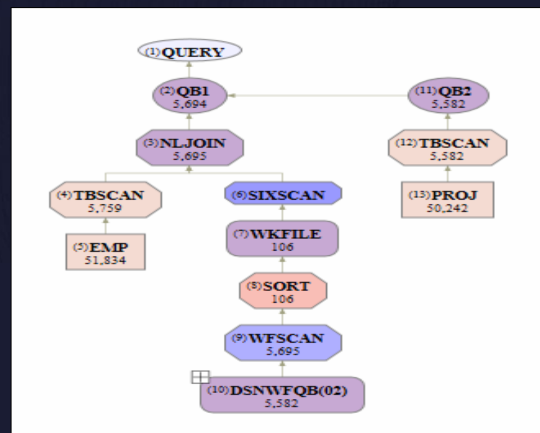
QBLOCKNO	PLANNO	METHOD	CREATOR	TNAME	ATYPE	MCOLS	TABLE_TYPE
1	1	0	ODYTA	DSNIN002	IN	0	I
1	2	1	ODYTA	DSNIN003	IN	0	I
1	3	1	THEMIS81	EMP	I	2	T

Prior to V10: 1 matching column

V10: Work file loaded for each IN LIST. Then 2 matching columns based on different combinations.

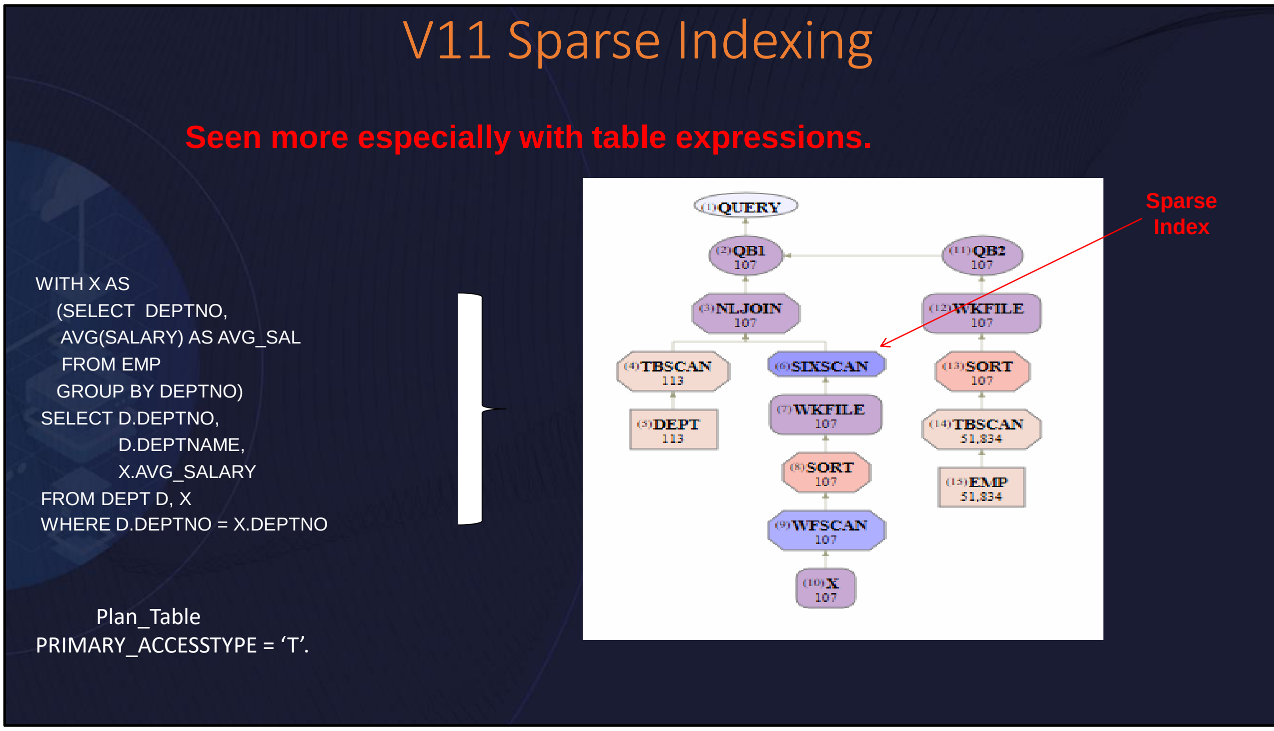
Visual Explain – Sparse Index ‘In List’

```
SELECT * FROM EMP E
   WHERE  E.JOB = 'DESIGNER'
     AND  E.DEPTNO IN
        (SELECT P.DEPTNO
         FROM   PROJ P
         WHERE  P.MAJPROJ = 'MA2100');
```



This is a visual explain from the same query shown previously. As you can see, the visual explain shows a little more detail on the explain information by showing that it builds what’s called a sparse index for the subquery work file and checks the data in the file via a nested loop process. This sparse index is built in memory.

File built with data is called DSNWFQB(02) which is a dataset from QB2.



V11 sparse index processing is similar to hash joining on other platforms (Db2 LUW, SQL Server, Oracle). This is usually a good thing that the optimizer chooses. The index is built with hashed values in memory (called In-Memory-Data-Cache). Could overflow to a work file if the entries in the sparse index are too many that overflows the MXDTCACH setting.

The Sparse index gets built at runtime, with the hash matching join being faster than index lookups on the inner table of the nested loop join. Especially if the join has enough rows from the outer to inner to "pay back" the build / cost of the sparse index/hash.

This helps especially with table expressions that get ‘materialized’. Always look in the explain to see if an index was built.

Safe Query Optimization

Uncertainties in this query:

- What are the host variable values ?
- How to generate a good filter factor without known values ?
- How to generate good filter factor for range predicates ?

```
SELECT ...
FROM EMPLOYEE
WHERE DEPTNO IN (?,  ?,  ?,  ?,  ?)
     AND EMPNO > ?
     AND LASTNAME BETWEEN ? AND ?
;
```

20

It used to be so many optimization steps were based on predicates filter factors:
- Which index
- Order of tables
- Join Type

But now when none of the predicates shows a significant filter factor, the optimizer may choose based on what it thinks is a safer and better predictable predicate.

```
For example:  WHERE BIRTHDATE < :HV-BDATE           Filter Factor = 18.2%
              AND DEPTNO IN                         Filter Factor = 22.6%
                         (SELECT DEPTNO
                          FROM DEPT
                          WHERE DEPTNAME LIKE'D%')
```

In this example, older versions of Db2 would have chosen the index on BIRTHDATE, but because the filter factor are close, V110 may take the safer route in using the DEPTNO index because of the uncertainty of a range predicate.

It is often difficult to pick the most optimal access path based on:
- Range Predicates
- Unpredictable RID pool environment
- Use of host variables not knowing the values at runtime
- Non uniform distribution of values for a column

One way to help these situations was to let Db2 know of the values being processed by either binding with 'Reopt', coding dynamic SQL, or hard coding values.

Index Include Columns

Features

- Add non-key columns to unique indexes
- Helps to gain 'Index Only' access
- Columns are not part of the unique constraint

```
CREATE UNIQUE INDEX XEMP4 ON EMP (COLA, COLB, COLC)
 INCLUDE (COLF, COLG)

or

ALTER EMPX1
INCLUDE LASTNAME
;
```

21

Prior to V10: Any unique constraint required a unique index for enforcement. But only those columns defining the constraint can be used in the index. A table can have an arbitrary number of unique constraints, with at most one unique constraint defined as a primary key.

Prior to V10, may have seen something like the following on a table because the EMPNO column was a unique constraint due to it being the primary key. So another index was created with additional column(s) to improve certain queries.

Index1 = EMPNO
Index2 = EMPNO, DEPTNO

Following are the steps needed in order to get column(s) included as part of an existing index.

Alter Index with the include clause. This puts the index in page set rebuild pending stage (PSRBD).

```
ALTER XEMP1 INCLUDE (LASTNAME)
;
COMMIT
;
```

Rebuild the index, or Reorg the tablespace.
Execute Runstats
Perform any necessary rebinds
Run an explain for verification

Notes: Included column only allowed for unique indexes.
Included columns not allowed with indexes on expressions.
Indexes with an already existing included column cannot have further unique columns added via the ALTER.

SYSIBM.SYSKEYS shows the included columns.

V10 Currently Committed Data – Only committed data is returned

-911 SQLCODE

Handles locking issues due to:
- Insert locks from another process
- Delete locks from another process

Bind Option: CONCURRENTACCESSRESOLUTION
Prepare Option: USE CURRENTLY COMMITTED

What happens when a delete lock in encountered ?
Reader will still get that row if it fits its ‘Where‘ logic.

What happens when an insert lock is encountered:
Reader will not get that row.

This new feature is supported only as a bind parameter or in a dynamic prepare statement, and allows access to data that was last committed before a lock would take place. It works with read processes only, and only when locks take place due to another process executing inserts or deletes. If another process is taking locks due update executions, the read processes will be locked as normal.

Using *currently committed* , only committed data is returned, as was the case previously, but now read processes do not wait for writers or deleters to release locks. Instead, readers return data that is based on the currently committed version; that is, data prior to the start of the write or delete operation.

Using *currently committed* , only committed data is returned, as was the case previously, but now read processes do not wait for writers or deleters to release locks. Instead, readers return data that is based on the currently committed version; that is, data prior to the start of the write operation.

More Stage 1 Predicates

New Stage 1 predicates:

WHERE value BETWEEN COL1 AND COL2
WHERE SUBSTR(COLX, 1, n) = value ➔ From Pos 1 only
WHERE DATE(TS_COL) = value
WHERE YEAR(DT_COL) = value

Db2 11 rewrites some of the more common stage 2 local predicates, including the following predicates, to an indexable form:

Db2 9 for z/OS delivered the ability to create an index on an expression, which required the developer or DBA to identify the candidate queries and create the targeted indexes. The Db2 11 predicate rewrites allow optimal performance without needing to intervene for better performance.

Note: Db2 will only rewrite if there is no index on expression that matches.

Example1:

WHERE SUBSTR(LASTNAME,1,3) = :hv   is a stage 2 non indexabe predicate

V11, this becomes:

WHERE LASTNAME = (exp)   is a stage 1 indexable (exp is a Db2 computed value for boundaries of column)

. Example: SUBSTR(LASTNAME,1,3) =‘AND’ becomes LASTNAME BETWEEN ‘AND……’ and ‘ANDzzzzzzz’

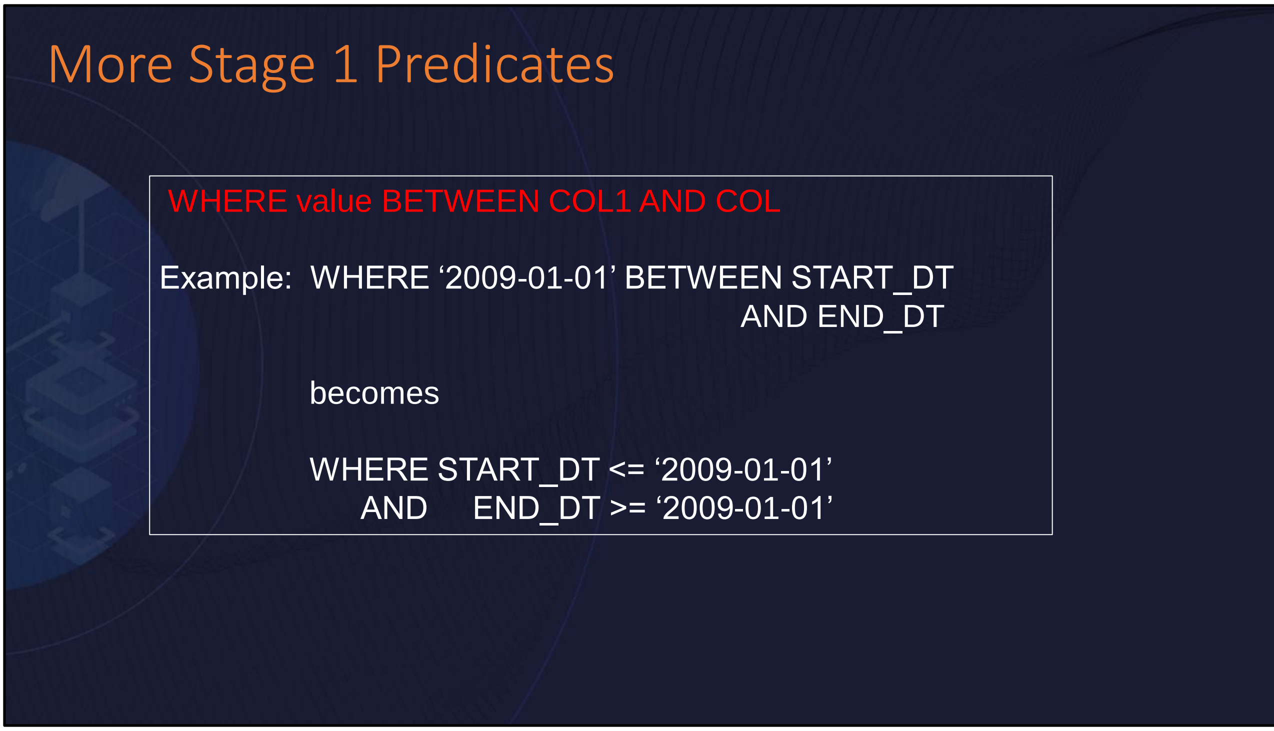
Example2:

WHERE SUBSTR(LASTNAME,1,3) <= :hv   is a stage 2 non indexabe predicate

V11, this becomes:

WHERE LASTNAME <= (exp)   is a stage 1 indexable (exp is a Db2 computed value for boundaries of column)

. Example: SUBSTR(LASTNAME,1,3) <=‘AND’ becomes LASTNAME <= ‘C1D5C4FFFFFFFFFFFFFFFFFF’



Db2 11 optimizer is now starting to do what developers had to do all these years, and that is to take many of the stage 2, non indexable predicates and rewrite them more efficiently. This is a simple standard rewrite the optimizer now takes care of.

Db2 9 for z/OS delivered the ability to create an index on an expression, which required the developer or DBA to identify the candidate queries and create the targeted indexes. The Db2 11 predicate rewrites allow optimal performance without

More Stage 1 Predicates

WHERE DATE(TS_COL) = value

Example: WHERE DATE(ORDER_TS) = ‘2009-01-01’

becomes

WHERE ORDER_TS BETWEEN
‘2009-01-01-00.00.00.000000’ AND
‘2009-01-01-24.00.00.000000’

Db2 11 optimizer is now starting to do what developers had to do all these years, and that is to take many of the stage 2, non indexable predicates and rewrite them more efficiently. This is a simple standard rewrite the optimizer now takes care of.

Db2 9 for z/OS delivered the ability to create an index on an expression, which required the developer or DBA to identify the candidate queries and create the targeted indexes. The Db2 11 predicate rewrites allow optimal performance without

More Stage 1 Predicates

WHERE YEAR(HIREDATE) = value

Example: WHERE YEAR(HIREDATE) = 2009

becomes

WHERE HIREDATE BETWEEN
‘2009-01-01’ AND ‘2009-12-31’

Case Logic

Case logic in predicates

Local predicate

```
SELECT *
FROM EMP
WHERE EDLEVEL  =
   CASE ?
    WHEN ‘HS’  THEN 12
    WHEN ‘CO’  THEN 14
    WHEN ‘GR’  THEN 16
  ELSE 00 END;
```

Join Predicate

```
SELECT E.EMPNO E.LASTNAME
 FROM EMP E,  DEPT D
 WHERE E.DEPTNO =
      CASE ?
       WHEN ‘D’ THEN D.DEPTNO
       WHEN ‘A’ THEN D.ADMRDEPT
      END
```

CASE expressions are also enhanced to support indexability as shown More common, complex resolution of code values to their business value are being included in a view or table expression to be used within a query, rather than using a code table or dimension table for this purpose. When used in predicates, Db2 V11 can now use these expressions as indexable, rather than stage 2 predicates as in previous releases.

A CASE expression must be able to be evaluated before.

Predicate Pushdown

Db2 11 supports pushdowns into table expressions

OR predicates

```
SELECT E.EMPNO, E.SALARY,
            TEMP.NUM_EMPS
FROM EMP E ,
         (SELECT DEPTNO,
          COUNT(*) AS NUM_EMPS
         FROM EMP
         GROUP BY DEPTNO) AS TEMP
WHERE E.DEPTNO  = TEMP.DEPTNO
AND (TEMP.DEPTNO LIKE 'C%' OR
         TEMP.DEPTNO LIKE 'D%');
```

Db2 V11 takes the outside predicates and pushes them inside the table expression. The idea is to apply the predicate before any materialization takes place.

Note1: If EMP E table has any local predicates, then the TEMP local predicate does not get pushed

‘OR COLUMN IS NULL’ Predicates

- Optimizer may now choose single index access
- V11 is able to choose multi index access
- May transform to ‘In List’ table and nested loop join

Rewrite V11:
MGRNO IN (?, ?, NULL)

```
SELECT ...
FROM DEPT
WHERE MGRNO = ? or   MGRNO IS NULL

OR

SELECT ...
FROM DEPT
WHERE MGRNO IN (?, ?) or   MGRNO IS NULL

OR

SELECT ...
FROM DEPT
WHERE MGRNO > ?  or   MGRNO IS NULL
```

‘OR COLUMN IS NULL’ Predicates

Db2 V11 may now choose single index access for these predicates by rewriting the predicates as follows:

WHERE MGRNO IN (?, ?, NULL)

OR the query may transform into an ‘In List’ table and nested loop join that was introduced in V10. This has now been expanded to handle the ‘OR IS NULL’ condition.

The predicate MGRNO > ? or MGRNO IS NULL would at best get milti-index access in previous version. V11 can now handle this logic with single index access.

IN Predicate with OR Predicate

- Prior to V11 Db2 would not choose multi index access with an IN predicate
- V11 optimizer may now choose multi index access

```
SELECT ...
FROM EMP
WHERE LASTNAME = ?
      OR EMPNO IN (?, ?)
```

Db2 V10 most likely would chose a table space scan with these predicates ‘Ord‘’ together due to the IN predicate. If the IN predicate was an EQUAL predicate, then it may choose multi index access,

Db2 V11 will now take into consideration multi index access when ‘Oring’ and IN predicate with another predicate.

Early Out Join Processing

One to many relationship between EMP and EMPPROJACT tables

- Now works like a correlated subquery where each inner table probe will stop after first match is found

```
SELECT DISTINCT E.EMPNO, E.LASTNAME
FROM EMP  E  INNER JOIN
         EMPPROJACT EPA
  ON  E.EMPNO  =  EPA.EMPNO
```

Previously version of Db2, this was only available when the optimizer actually transformed an EXISTS subquery to a join.

Duplicates from T2 used to be removed by DISTINCT, in V11 each inner table probe will stop after 1st match is found.

Right Joins ALWAYS Rewritten as LEFT Joins

These 2 queries are logically equivalent

```
SELECT D.DEPTNO,  D.DEPTNAME,
           D.MGRNO,   E.FIRSTNME, E.LASTNAME
 FROM DEPT D LEFT JOIN EMP E
     ON D.MGRNO = E.EMPNO
```

```
SELECT D.DEPTNO,  D.DEPTNAME,
           D.MGRNO,   E.FIRSTNME, E.LASTNAME
 FROM EMP E  RIGHT JOIN DEPT D
     ON D.MGRNO = E.EMPNO
```

Whatever table is to the left of LEFT JOIN is what I call the driver table and that is where the processing starts, The other table is called the NULL SUPPLYING TABLE as it will sends nulls back for any column referenced if there is not a match on values being joined.

Whatever table is to the right of RIGHT JOIN is the driver table, and the other the NULL SUPPLYING TABLE.   So these two queries are logically equivalent and will return the exact same rows.

The optimizer will take EVERY right join and rewrite it as a left join. Logically there is never a need to code right join over left join, and the optimizer takes this into account. There is a JOIN_TYPE column in the PLAN_TABLE that will show a ‘L for both queries.

Left Join to Inner Join Rewrite

The first query will gets rewritten as an INNER JOIN

```
SELECT D.DEPTNO,  D.DEPTNAME,
            D.MGRNO,   E.FIRSTNME, E.LASTNAME
 FROM DEPT D LEFT JOIN EMP E
      ON D.MGRNO = E.EMPNO
WHERE E.JOB = ‘ANALYST’
```

```
SELECT D.DEPTNO,  D.DEPTNAME,
            D.MGRNO,   E.FIRSTNME, E.LASTNAME
 FROM DEPT D  INNER JOIN EMP E
      ON D.MGRNO = E.EMPNO
WHERE E.JOB = ‘ANALYST’
```

Once you apply predicate logic on the NULL SUPPLYING TABLE, you are automatically cancelling out any exception values that are not found on that table, thus making it a straight inner join. Once the optimizer sees OUTER JOIN and a predicate on the NULL SUPPLYING TABLE, it gets rewritten as an inner join. JOIN_TYPE = ‘ ‘ in the PLAN_TABLE.

The material in this presentation is further developed in the following Themis courses:

DB3052 – Db2 for z/OS Database Performance Tuning

SQ1010 – Dealing With Complex Queries
Cross Platform SQL
DB1037 – Advanced Query Tuning With IBM Data Studio
on z/OS
DB1032 – Db2 for z/OS Optimization Performance
and Tuning
DB1006 – Db2 LUW Query Tuning With IBM Data Studio

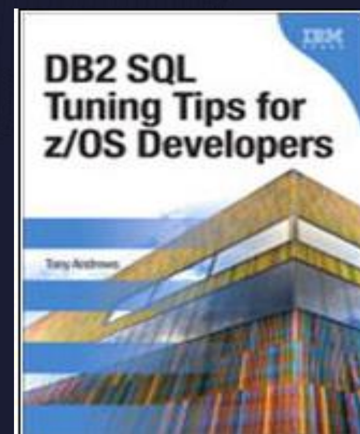
Links to these courses may be found at: www.themisinc.com

Tony's Email: tandrews@themisinc.com

Education. Check out www.amazon.com

Finally! A book of DB2 SQL tuning tips for developers, specifically designed to improve performance.

DB2 SQL developers now have a handy reference guide with tuning tips to improve performance in queries, programs and applications.



Education. Check Out
www.themisinc.com

- On-site and Public
- Instructor -led
- Hands-on
- Customization
- Experience
- Over 30 DB2 courses
- Over 400 IT courses

Themis inc.

US 1-800-756-3000
Intl. 1-908-233-8900

Speaker: Tony Andrews
Company: Themis Inc.
Email Address:
tandrews@themisinc.com

Thank you for attending! I hope you learned something new today!

Thank you Db2Night Show!