



## **Understanding Query Transformation in DB2 for z/OS**

***Daniel L Luksetich***  
***DanL Database Consulting***  
***danl@db2expert.com***  
***WWW.DB2EXPERT.COM***

Dan Luksetich is a senior DB2 DBA consultant. He works as a DBA, application architect, presenter, author, and teacher. Dan has been in the information technology business for over 30 years, and has worked with DB2 for over 25 years. He has been a COBOL and BAL programmer, DB2 system programmer, DB2 DBA, and DB2 application architect. His experience includes major implementations on z/OS, AIX, i Series, and Linux environments.

Dan's experience includes:

Application design and architecture

Database administration

Complex SQL

SQL tuning

DB2 performance audits

Replication

Disaster recovery

Stored procedures, UDFs, and triggers

Dan works everyday on some of the largest and most complex DB2 implementations in the world. He is a certified DB2 DBA, system administrator, and application developer, and has work on the teams that have developed the DB2 for z/OS certification exams. He is the author of several DB2 related articles as well as co-author of the DB2 9 for z/OS Certification Guide and the DB2 10 for z/OS Certification Guide.



IDUG DB2 North America Tech Conference  
Philadelphia, Pennsylvania | May 2015

#IDUGDB2

## International DB2 Users Group

- INDEPENDENT User Group Founded in 1988
- Web site – [IDUG.ORG](http://IDUG.ORG)
- Annual Conferences
  - North American in May
  - EMEA in November
  - Australasia in September
  - India



## IDUG Content Committee

- Dedicated to supporting the IDUG and DB2 user community by providing valuable free content
  - IDUG web site
  - Conferences
- Web Site
  - DB2 news blog
  - DB2 beginner's blog
  - IDUG content blog
  - IDUG content file library
  - Videos
    - IDUG Tech Talk Channel
    - YouTube

## 5 Key Bullet Points



- Objective 1: Learn what is query transformation.
- Objective 2: Learn the difference between query transformation and optimization.
- Objective 3: Demonstrate the effects of query transformation on specific query constructs.
- Objective 4: Learn how Query transformation is an important performance feature.
- Objective 5: Learn how to defeat some of the negative impacts of query transformation and take advantage of query transformation for performance.

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

4

Dan Luksetich is a data scientist and senior DB2 DBA consultant. He works as a DBA, application architect, presenter, author, and teacher. Dan has been in the information technology business for over 29 years, and has worked with DB2 for over 24 years. He has been a COBOL and BAL programmer, DB2 system programmer, DB2 DBA, and DB2 application architect. His experience includes major implementations on z/OS, AIX, i Series, Windows, and Linux environments.

Dan's experience includes application design and architecture, business analytics, SQL consulting and education, database administration, complex SQL, SQL tuning, DB2 performance audits, replication, disaster recovery, stored procedures, UDFs, and triggers. Dan works everyday on some of the largest and most complex DB2 implementations in the world. He is a certified DB2 DBA for DB2 for z/OS and LUW, system administrator for z/OS, and application developer, and has worked on the teams that have developed the DB2 for z/OS certification exams. He is the author of several DB2 related articles as well as co-author of the DB2 9 for z/OS Certification Guide and the DB2 10 for z/OS Certification Guide.

Dan is an IBM Gold Consultant, IBM Champion for Data Management, and the chairman of the International DB2 Users Group Content Committee.

***Disclaimer – Please Read the Following***

- The information contained in this presentation is based on techniques, algorithms, and documentation published by the several authors and companies, and in addition is the result of research. It is therefore subject to change at any time without notice or warning.
- The information contained in this presentation has not been submitted to any formal tests or review and is distributed on an “As is” basis without any warranty, either expressed or implied.
- The use of this information or the implementation of any of these techniques is a client responsibility and depends on the client’s ability to evaluate and integrate them into the client’s operational environment.
- While each item may have been reviewed for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere.
- Clients attempting to adapt these techniques to their own environments do so at their own risks.
- Foils, handouts, and additional materials distributed as part of this presentation or seminar should be reviewed in their entirety.

No animals were harmed during testing

## Agenda



- Introduction to query transformation
- Get a hint as to what query transformation does
- Learn how to view rewritten queries
- Understand why DB2 rewrites queries
- Look at specific rewritten queries
- Learn how to influence query transformation

## *What is query transformation?*



- Process that happens before optimization
- Part of optimization (in my opinion)
- Different programming teams within DB2 development
  - QST (query transformation)/parser
  - APS (Access Path Selection – internally referred to as optimizer)
  - Post-optimizer/parallelism
  - ASLGEN
  - Structure Gen
  - Runtime
- Query transformation concepts
  - Parse query
  - Look up DB2 objects
  - Read statistics
  - Rewrite query for submission to optimizer
  - Transform queries to special DB2 objects
    - Materialized query table
    - Temporal table
    - View

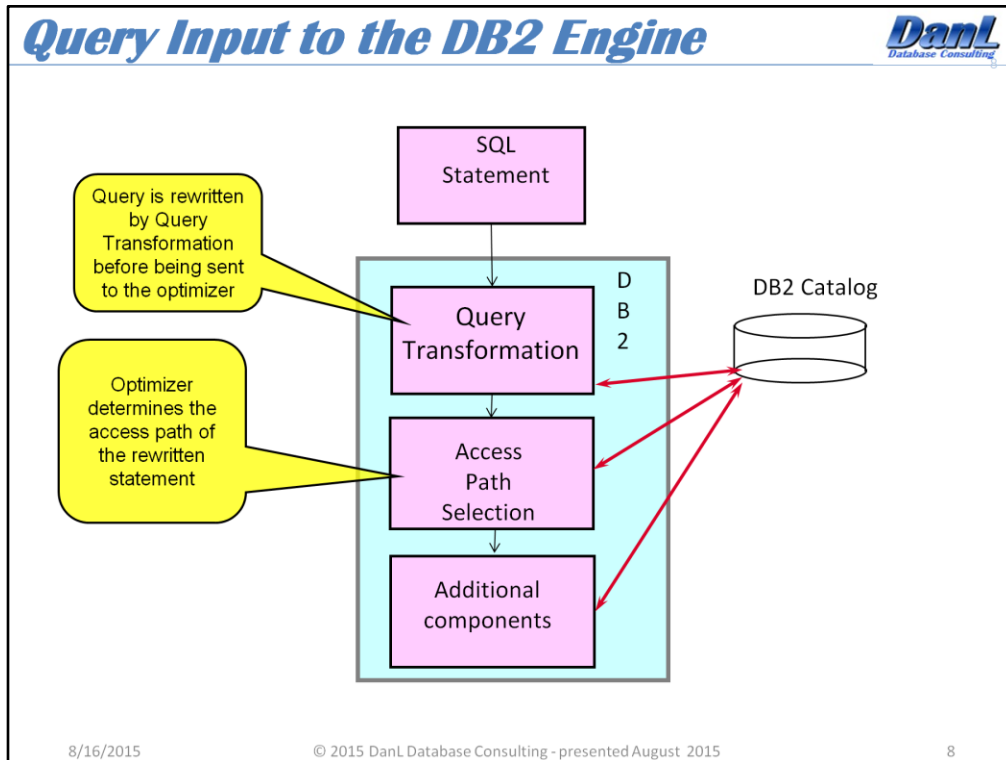
8/16/2015

© 2015 DanL Database Consulting - presented August 2015

7

When most people talk about the processing of DB2 SQL statements they speak of the DB2 optimizer. What many don't realize is that before the query passes through the optimizer it first passes through something called query transformation. This component of the DB2 engine is critical to performance, and a vital part of query processing. In this presentation we are only going to talk about the impact of query transformation on SQL statements that read data. However, the query transformation component of DB2 does so much more.

It is important to understand query transformation because it is a separate piece of code from the optimizer, owned by a separate development team within DB2 development, and the first component to manage your queries as you send them in to be processed.



As a query is submitted to DB2, either dynamically at execution time or statically during the bind process, it passes through the query transformation component of the DB2 engine. This component is going to look at certain information about the objects being accessed in the query, and make certain decisions as to whether or not the query is optimal as written. If not, the query will be rewritten to present the optimizer with more optimization choices than what was originally submitted. In addition, query transformation is looking at specific DB2 objects and rules, and will rewrite a query such that proper optimization is possible. In other words it fixes your poorly written queries!



## *What Does Query Transformation Do?*

- Accepts and parses the input query
- Reads the DB2 catalog
- Determines if query should be rewritten
  - Query structure
  - Table and index definitions
  - Special structures
    - Temporal table
    - Materialized query table
- Query rewritten according to internal rules
- Labels predicates as stage1 or stage 2
- And much much much more!!

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

9

You do not have much control over the query rewrite performed by query transformation. We will, however, discuss some important issues and changes you can make to SQL statements to properly work with query transformation.

There are many things that query transformation does. In this presentation we are only touching the surface of the things that happen during query transformation. A thorough review of the DB2 performance guide and the DB2 10 performance redbook can help determine all that happens during query rewrite. Most importantly, you should be looking at rewritten queries within your organization to see for yourself what is happening.

## Why Have Query Transformation?



- Because they (IBM) know more than you! ☺
- Special objects require it (MQT, Temporal, etc.)
- SQL is Orthogonal
  - One statement can be written many ways
  - There is an optimal way to construct a statement
    - depending upon the underlying data
    - DB2 catalog can be used to determine this optimal construction
- Many tools are used to generate SQL
  - Generated generically for many different DBMS
  - Transformation can make the SQL specific to DB2 without change to the original query

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

10

Back in the early days of DB2 many people were talking about efficient SQL coding techniques. Back then you had to code statements a certain way in order to get them to perform properly. While that is still true today it is much less so. The SQL language is very orthogonal. This means that a statement that serves a logical purpose can be physically written in a myriad different ways. There still can be an optimal way to write a statement, but fewer and fewer people know how to do this, and many SQL statements are generated by tools. Query transformation has evolved into a very valuable component that can translate a query that serves a logical purpose into its high performing physical counterpart.

## *How Do You View Rewritten Queries?*



- You must use the DB2 EXPLAIN facility to populate EXPLAIN tables
- EXPLAIN tables that hold transformation information must be created
  - DSN\_PREDICAT\_TABLE
  - DSN\_QUERY\_TABLE
- Use a tool to view the rewritten query
  - IBM Data Studio
  - Other query tuning tool from your favorite vendor
  - SQL (with significant skill)

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

11

Until recently in the history of DB2 it was impossible to see the rewritten query. There existed for some time a set of “hidden” EXPLAIN tables that contained detailed information about query optimization. These tables were finally exposed to the public with DB2 version 8. To the best of my knowledge there are two tables that hold information about your rewritten queries.

The DSN\_PREDICAT\_TABLE contains information about predicates in a query. This includes predicates that are generated during query rewrite. Complex OR’s may be rewritten to In-lists, predicates against one table could be transitively closed to another, and more. All of these predicates are listed in this table. While it is difficult to read, and IBM discourages reading this table without the use of a tool, I do have an advanced EXPLAIN table query of my own that looks at this data.

The DSN\_QUERY\_TABLE is another EXPLAIN table that is used by EXPLAIN tools to look at both the original and rewritten query. The information is stored as XML documents, and not at all easy to read with the human eye.

## Using SQL to View Rewritten Queries



- You can query certain explain tables to view all or part of a rewritten query
- DSN\_PREDICAT\_TABLE
  - Contains a row for every predicate of an explained query
  - Shows generated predicates as well as original predicates
  - Does not show the complete rewritten statement
  - Example EXPLAIN table query at db2expert.com
- DSN\_QUERY\_TABLE
  - Contains entire original query and entire rewritten query
  - Also contains table and column information
  - All in the form of XML documents
  - Must have patience and skill to read this, or programming skills
  - Not populated for static binds with EXPLAIN

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

12

Here is the current version of my advanced EXPLAIN query:

```
WITH MAXTIME (COLLID, PROGNAME, QUERYNO, EXPLAIN_TIME) AS (SELECT COLLID, PROGNAME, QUERYNO, MAX(EXPLAIN_TIME)
FROM PLAN_TABLE WHERE COLLID = 'DSNESPCS' AND PROGNAME = 'DSNESM68' AND QUERYNO = 1
GROUP BY COLLID, PROGNAME, QUERYNO)

SELECT SUBSTR(P.PROGNAME, 1, 8) AS PROGNAME ,SUBSTR(DIGITS(P.QUERYNO), 6) CONCAT '-' CONCAT SUBSTR(DIGITS(P.QBLOCKNO), 4)
CONCAT '-' CONCAT SUBSTR(DIGITS(P.PLANNO), 4) AS QQP ,SUBSTR(CHAR(P.METHOD), 1, 3) AS MTH
,SUBSTR(CHAR(F.PREDNO), 1, 3) AS P_NO,SUBSTR(CHAR(P.MERGE_JOIN_COLS), 1, 3) AS MIC
,SUBSTR(P.CREATOR, 1, 8) AS TBCREATOR ,SUBSTR(P.TNAME, 1, 18) AS TBNAME,SUBSTR(P.CORRELATION_NAME, 1, 8) AS CORR_NM
,DEC(D.ONECOMPROWS, 10, 1) AS ROWS_POST_FILTER ,P.ACCESSSTYPE AS ATYP ,SUBSTR(P.ACCESSNAME, 1, 15) AS A_NM
,P.INDEXONLY AS IXO ,CHAR(P.MIXOPSEQ) AS MIX,CHAR(P.MATCHCOLS) MCOL,F.STAGE AS STAGE ,DEC(E.FILTER_FACTOR, 11, 10) AS FF
,E.BOOLEAN_TERM AS BT,SUBSTR(E.TEXT, 1, 30) AS PRED_TEXT30,P.SORTN_JOIN CONCAT P.SORTC_UNIQ CONCAT
P.SORTC_JOIN CONCAT P.SORTC_ORDERBY CONCAT P.SORTC_GROUPBY AS NI_CUIJOG ,P.PREFETCH AS PF,P.COLUMN_FN_EVAL AS CFE
,P.PAGE_RANGE AS PGRNG,P.JOIN_TYPE AS JT,P.QBLOCK_TYPE AS QB_TYP,P.PARENT_QBLOCKNO AS P_QB,P.TABLE_TYPE AS TB_TYP
,P.EXPLAIN_TIME AS B_TM FROM PLAN_TABLE P INNER JOIN MAXTIME M ON M.COLLID = P.COLLID
AND M.PROGNAME = P.PROGNAME AND M.QUERYNO = P.QUERYNO AND M.EXPLAIN_TIME = P.EXPLAIN_TIME
LEFT JOIN DSN_FILTER_TABLE F ON M.COLLID = F.COLLID AND M.PROGNAME = F.PROGNAME AND M.QUERYNO = F.QUERYNO AND P.QBLOCKNO = F.QBLOCKNO AND
P.PLANNO = F.PLANNO AND M.EXPLAIN_TIME = F.EXPLAIN_TIME
and P.ACCESSSTYPE Not In ('MX', 'MI', 'MU')
LEFT JOIN DSN_PREDICAT_TABLE E ON F.PROGNAME = E.PROGNAME AND F.QUERYNO = E.QUERYNO AND F.QBLOCKNO = E.QBLOCKNO
AND F.PREDNO = E.PREDNO AND M.EXPLAIN_TIME = E.EXPLAIN_TIME
LEFT JOIN TABLE (SELECT MIN(X.ONECOMPROWS) AS ONECOMPROWS
FROM DSN_DETCOST_TABLE X WHERE M.PROGNAME = X.PROGNAME AND M.QUERYNO = X.QUERYNO
AND P.QBLOCKNO = X.QBLOCKNO AND P.PLANNO = X.PLANNO AND M.EXPLAIN_TIME = X.EXPLAIN_TIME) AS D
ON 1=1 ORDER BY PROGNAME, B_TM, QQP, MIX, F.PREDNO;
```

## *DSN\_QUERY\_TABLE*



- Contains information about EXPLAINED statements
- Populated by EXPLAIN statement execution
- Includes the “before” and “after” query text
- As XML. Not easy for a person to read

```
SELECT QUERYNO, SEQNO, NODE_DATA
FROM   DSN_QUERY_TABLE
WHERE  QUERYNO = 111
AND    QUERY_STAGE = 'AFTER'
ORDER BY SEQNO
```

## DSN\_QUERY\_TABLE Example



- This is the before and after for the most simple query
  - SELECT 1 FROM SYSIBM.SYSDUMMY1

### Before

```
<QUERY><SUBQUERY QBNO='1' PRUNED='N'>(<SEL-CLAU>SELECT <SEL-LIST
STAR='N'><LIST-ITEM TYPE='EXPR'><LIT VALUE='>1</LIT></LIST-ITEM> </SEL-
LIST></SEL-CLAU><FROM-CLAU> FROM <TAB-REF TNO='1'
TYPE='T'>SYSIBM.SYSDUMMY1</TAB-REF> </FROM-CLAU>)</SUBQUERY></QUERY>
```

### After

```
<QUERY><SUBQUERY QBNO='1' PRUNED='N'>(<SEL-CLAU>SELECT <SEL-LIST
STAR='N'><LIST-ITEM TYPE='EXPR'><LIT VALUE='>1</LIT></LIST-ITEM> </SEL-
LIST></SEL-CLAU><FROM-CLAU> FROM <TAB-REF TNO='1'
TYPE='T'>SYSIBM.SYSDUMMY1</TAB-REF> </FROM-CLAU>)</SUBQUERY></QUERY>
```

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

14

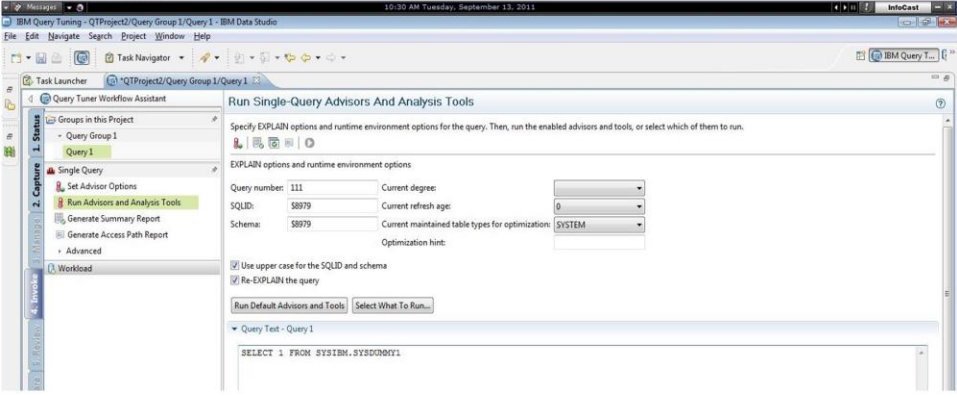
You can see here that the rewritten query is not easy to read with the human eye. This particular example is simply from an EXPLAIN of this query

```
SELECT 1 FROM SYSIBM.SYSDUMMY1
```

The rewrite actually did nothing to the original query. However, it is still difficult to read unless you parse through the XML. That's why it's better to use a tool.

## IBM Data Studio – Query Tuning DanL Database Consulting

- The query tuner feature of Data Studio allows you to input SQL statements from many different sources
- This example shows a simple statement entered as text



The screenshot displays the IBM Data Studio Query Tuner interface. The main window is titled "Run Single-Query Advisors And Analysis Tools". It features a sidebar on the left with a "Task Launcher" and a "Query Tuner Workflow Assistant" pane. The main area contains a form for specifying EXPLAIN options and runtime environment options. The form includes fields for "Query number" (111), "Current degree" (1), "SQLID" (58979), "Current refresh age" (0), "Scheme" (58979), and "Current maintained table types for optimization" (SYSTEM). There are also checkboxes for "Use upper case for the SQLID and schema" and "Re-EXPLAIN the query". A "Run Default Advisors and Tools" button is visible. Below the form, a "Query Test - Query 1" section shows the SQL statement: `SELECT 1 FROM SYSIBM.SYSDUMMY1`.

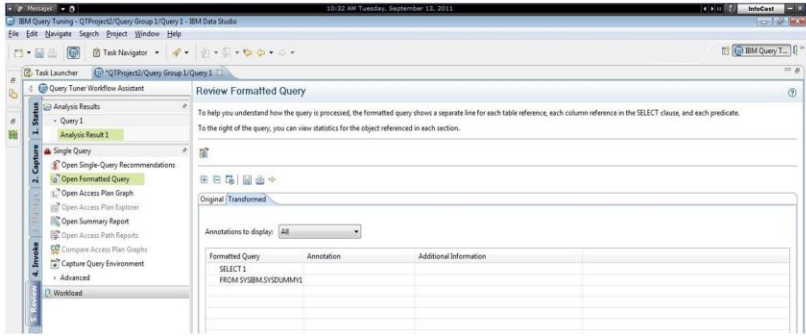
8/16/2015 © 2015 DanL Database Consulting - presented August 2015 15

IBM Data Studio is available as a free download from IBM. This tool is what was used to look at all of the rewritten queries in this presentation. This tool runs on a computer running windows or Linux and can be used to tune queries in DB2 for z/OS and DB2 for LUW. You need a remote connection to your database in order to run the tool.

This screen snap shows a simple SQL statement being input as text to Data Studio query tuning. There are many sources you can draw on to input SQL statements into the tool.

## IBM Data Studio

- The query tuner feature allows you to input SQL statements from many different sources
- This example our simple statement transformed
  - In this case it looks the same as our original simple query
  - All examples in this presentation were cut from Data Studio



8/16/2015
© 2015 DanL Database Consulting - presented August 2015
16

Once a statement has been EXPLAINed using the tool, you can then choose to view the access path, the rewritten query, or addition query information or recommendations.

This screen snap shows the transformed query. To view this I simply hit the appropriate tab in the product.



### *Some Examples of Query Rewrite Concepts*



- Subqueries
- Views
- Nested table expressions
- CASTing data types
- Transitive closure
- Union distribution
- Temporal queries
- QST Enhancements in DB2 11 for z/OS
  
- This is only the tip of the iceberg as to query rewrite!

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

17

As mentioned before the query transformation component of DB2 does a lot. We can only cover so much in a short period of time. So, I'm focusing on some of what I feel are significant features. Your results may vary, and I strongly recommend experimentation as the best guide to how DB2 rewrites queries.

## *Predicate Transitive Closure*



- Transitive closure mathematical property
  - If  $A=B$  and  $B=C$  then  $A=C$
- Property can be applied to DB2 predicates
  - If  $A.EMPNO = '000010'$
  - And  $A.EMPNO = B.EMPNO$
  - Then  $B.EMPNO = '000010'$
- DB2 can apply predicate transitive closure
  - Subqueries
  - Joins
- Gives the optimizer potentially more index choices
- Also gives the optimizer more choices for table access sequence

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

18

Remember that one of the goals of query transformation is to supply the optimizer with appropriate access path choices. One of those choices is table access sequence for multi-table queries. When DB2 is bringing data together from multiple tables the most efficient access is to access the table that retrieves fewer rows first, and then access the next table, and so on.

Sometimes tables are joined by common columns, and those columns are also referenced in local predicates. DB2 can use the transitive property to create redundant predicates during query rewrite. This allows for local predicates to be propagated across tables, and thus giving the optimizer more choices for potential index access, stronger filtering, and more efficient table access sequence.

## Transformation of Subqueries



- A subquery can be rewritten three different ways
  - Correlated
  - Non-Correlated
  - As a join
- When is Which Better?
  - Non-Correlated subquery
    - Subquery result is small compared to outer query
    - Subquery result is not excessively large
    - Outer query utilizes matching index access for predicate containing subquery
    - There is no supporting index for the subquery
    - DB2 can rewrite correlated subqueries into non-correlated subqueries

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

19

The result of any subquery can be achieved by coding that subquery, the equivalent alternate subquery, or a join.

Which is best? I don't know. It all depends upon multiple things:

- The available indexes on the tables in the query
- Whether or not a join will introduce duplicates
- The size of the tables involved
- The size of the result sets of the portions of the queries (local table access)

It takes some skill to identify these things and code a subquery appropriately. Fortunately, query transformation will make a lot of these considerations for you and rewrite the query you've written no matter how you write it (except joins).

## *Transformation of Subqueries*



- When is Which Better (continued)
  - Correlated subquery
    - Subquery result is potentially large compared to outer query
    - Outer query has no supporting index for predicate containing subquery
    - DB2 can rewrite non-correlated subqueries as correlated subqueries
    - Subquery has supporting index
    - Subquery contains duplicates (can “early out” on duplicates)
  - Join
    - Good matching index potential for outer query and subquery
    - Subquery does not introduce duplicate rows
    - DB2 can automatically transform both non-correlated and correlated subqueries to joins
      - Transformation of non-correlated into join may be a combination of query transformation and query optimization

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

20

Joins are preferred. This is due to the fact that the table access sequence is more flexible, the index choices more dynamic, and transitive closure more apparent. In other words, a subquery is likely to be transformed into a join if it is determined that there are good indexes to support first access to either table. Sometimes a sort will be introduced to avoid duplicates in the result set.

## Non-Correlated to Correlated



- This example shows a non-correlated subquery transformed into a correlated (and non-correlated) subquery with predicate transitive closure applied (“bubble up”)

### Before

```
SELECT EMPNO, LASTNAME
FROM EMP AS E
WHERE EMPNO IN
(SELECT MGRNO
FROM DEPT AS D
WHERE MGRNO IN ('000010', '000020'))
```

### After

```
SELECT E.EMPNO, E.LASTNAME
FROM EMP AS E
WHERE ( E.EMPNO IN ( '000010', '000020' )
AND E.EMPNO IN (SELECT D.MGRNO
FROM DEPT AS D
WHERE ( D.MGRNO IN ( '000010', '000020' )
AND E.EMPNO IN ( '000010', '000020' )
AND E.EMPNO = D.MGRNO)))
```

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

21

This very cool example shows a non-correlated subquery rewritten as a correlated subquery. In addition, DB2 was able to use transitive closure to add a redundant predicate against the employee table.

In this particular case DB2 is trying to accommodate what can be a rather inefficient subquery. The original query has to access the department table first, however there is a local predicate on the MGRNO column. This is the same column that is used on the right side of the predicate against the EMP table in the outer portion of the query. DB2 can apply transitive closure to create a local predicate against the EMP table EMPNO column. Since the EMPNO column is a unique primary key on the EMP table it is probably more efficient to access that table first. So DB2 rewrites the non-correlated query, which would have accessed the DEPT table first, into a correlated subquery, which will access the EMP table first.

## Correlated to Join



- This example shows a correlated subquery transformed into a join (your results may vary)

### Before

```
SELECT EMPNO, LASTNAME
FROM EMP AS E
WHERE EXISTS
(SELECT 1
 FROM DEPT AS D
 WHERE D.MGRNO = E.EMPNO)
```

### After

```
SELECT E.EMPNO
       , E.LASTNAME
FROM DEPT AS D
       , EMP AS E
WHERE D.MGRNO = E.EMPNO
```

This particular example shows a non-correlated subquery transformed into a join. This is probably due to the fact that there are no local predicates. DB2 then wants to have the ability to access either table first in the table access sequence and a join will enable this.

What is not shown here is that the join will have to contain a sort to eliminate duplicates. Most likely the DEPT table is access first, which is the opposite of the original query.

## *When are Subqueries Not Transformed?*



- DB2 Can Transform Subqueries Efficiently
  - Correlated into Non-Correlated
  - Non-Correlated into Correlated
  - Either into a Join
- Sparse Index can be Created to Support the Join
- Dependent Upon Catalog Statistics
- DB2 Will Attempt to Make the Same Performance Choices You can Make Manually
- User Does Not Have Control
  - Or DB2 Cannot Transform a Subquery Due to Limitations
- APS can also transform subqueries when QST does not

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

23

DB2 can transform subqueries many different ways, and typically you do not have control over this. It attempts to make the best decision based upon table and indexes designs, as well as statistics.

There are 2 types of subquery transformations - 1) QST rewrite from subquery to join, which only happens in limited situations (for example, non-correlated subquery is unique, or simple correlated unique or duplicate), and 2) by APS (with help from QST). If QST does the rewrite to join, then APS cannot undo this. If APS sees a subquery (QST didn't rewrite), then this is where it may cost the correlated and non-correlated form.

## *When are Subqueries Not Transformed*



- In Some Cases DB2 Cannot Correlate or De-Correlate Subqueries or Transform to Join
  - Set (Aggregate) Functions in the Subquery Can Change Value
  - Grouping Can Change the result
  - Some Correlated Range Predicates
  - Some Not Equal Correlated Predicates
  - Some INSERT, UPDATE, DELETE, or SELECT FOR UPDATE Statements with Subqueries Don't Qualify
- In These Situations You Should Code for Performance Manually
  - Coding Efficient Subquery Techniques will Still Apply
  - Only way to know is to test

There are many situations in which subqueries are not transformed. If you want to defeat subquery transformation you may want to try some of this. I recommend against doing that, but there are always exceptions.



## *Merge of Views and Table Expressions*



- Query Transformation has the ability to merge table expressions and views into the statement that references them
- This allows DB2 to avoid materialization of the table expression or view
  - Most times merge is better than materialization
  - Data read once rather than multiple times
- Rarely merge can be a detriment to performance
  - Especially CPU
  - When multiple functions executed
  - Table references distributed

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

25

DB2 query transformation aggressively merges nested table expressions and views referenced in a query into the text of the query. This is because it relies on the basic idea that merging an expression into a query is more efficient than materializing the view or table expression.

## Simple Merge of a Table Expression

- This example shows a table expression merged into the outer statement that references it

**Before**

```
SELECT *
FROM
  (SELECT DEPTNO, DEPTNAME, MGRNO
   FROM DEPT
   WHERE DEPTNO = 'A00') AS TAB1
```

**After**

```
SELECT DEPT.DEPTNO
       , DEPT.DEPTNAME
       , DEPT.MGRNO
FROM DEPT
WHERE DEPT.DEPTNO = 'A00'
```

8/16/2015 © 2015 DanL Database Consulting - presented August 2015 26

In this simple example a table expression that reads the DEPT sample table is referenced in an outer SELECT. DB2 query transformation determines that there is nothing in the table expression (aggregate function, non-deterministic function or expression, DISTINCT, and more) that would force it to have to materialize the expression. Therefore, the table expression is merged with the outer statement and sent on to optimization.

## Sort may Avoid a Merge



- This example shows a table expression with an embedded ORDER BY clause does not get merged
- Due to a sort the table expression will be materialized

### Before

```
SELECT *  
FROM  
(SELECT DEPTNO, DEPTNAME, MGRNO  
FROM DEPT  
ORDER BY DEPTNAME) AS TAB1
```

### After

```
SELECT *  
FROM TABLE (SELECT DEPT.DEPTNO  
              , DEPT.DEPTNAME  
              , DEPT.MGRNO  
            FROM DEPT  
            ORDER BY DEPT.DEPTNAME ASC)  
AS TAB1 ( DEPTNO  
         , DEPTNAME  
         , MGRNO)
```

In this example there is an ORDER BY in the table expression which results in a sort of the data because no index exists to support sort avoidance. Thus a merge cannot be avoided and the table expression will be materialized.

## ORDER OF Clause Effect on Rewrite



- The ORDER OF clause in the original query causes an additional ORDER BY to be generated during rewrite
- This query will sort twice

### Before

```
SELECT *
FROM
(SELECT DEPTNO, DEPTNAME, MGRNO
FROM DEPT
ORDER BY DEPTNAME) AS TAB1
ORDER BY ORDER OF TAB1
```

### After

```
SELECT *
FROM TABLE (SELECT DEPT.DEPTNO
              , DEPT.DEPTNAME
              , DEPT.MGRNO
              FROM DEPT
              ORDER BY DEPT.DEPTNAME ASC)
AS TAB1( DEPTNO, DEPTNAME, MGRNO)
ORDER BY TAB1.DEPTNAME ASC
```

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

28

Here is the same query as the previous slide, but I've added an ORDER BY ORDER OF clause as part of the final query. Query transformation will take the ORDER of clause and the ORDER BY statement that it references, and add another ORDER BY clause. This results in this statement ordering twice instead of once.

It is important to not that an ORDER OF clause is not useful for avoiding a sort, and will probably introduce additional sorts.

## Effect of Merge on Repeat References to Expressions



- Repeat references to a merged expression invoke that expression multiple times
- DB2 11 for z/OS offers an alternative to these repeated references called “expression sharing”
  - Enabled via hidden subsystem installation parameter (zPARM) QUERY\_REWRITE\_DEGREE (use ultra extreme caution in a sandbox)
- Use of RAND() in table expression can defeat this
  - BUT, will introduce materialization

### Before

```
SELECT MAX (PART) , MIN (PART) , AVG (PART)
FROM
  (SELECT LENGTH (RTRIM (LASTNAME)) AS PART
   FROM EMP) AS TAB1
```

### After

```
SELECT MAX ( LENGTH ( RTRIM ( EMP.LASTNAME ) ) )
       , MIN ( LENGTH ( RTRIM ( EMP.LASTNAME ) ) )
       , AVG ( LENGTH ( RTRIM ( EMP.LASTNAME ) ) )
FROM EMP
```

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

29

This is an example of a situation in which merge may not be such a good thing. With the speed of processing improvements to sorts and workfile management sorts and materialization are not as expensive as they were in previous releases of DB2. When a view or table expression is merged with the outer statement that references it that merge is literal. Everything that is referenced in the table expression is merged with whatever references it in the outer statement. In this example there exists a reference to a couple of built-in DB2 functions. Since no rule for materialization was found in the table expression DB2 determines it can be merged to the outer statement. The outer statement has multiple references to the result of the expression in the referenced nested table expression. That particular expression, the execution of the two DB2 built-in functions, is then repeated for each reference to the result in the outer statement during rewrite. What ultimately happens in this case is that a series of functions that are coded once in the original statement are executed three times per row in the rewritten statement. In the case of complex embedded expressions with many references to generated data you may want to consider forcing materialization of the nested table expression. I typically do this by adding a RAND() function within the table expression. DB2 11 for z/OS has introduced a potential performance improvement to repeated executions of expressions in merged nested table expressions with the introduction of a feature called “expression sharing”. This feature is currently activated via a hidden installation parameter, QUERY\_REWRITE\_DEGREE, which is set to 1 by default. Setting the value to 2 will enable expression sharing and enable the reuse of expression results in the outer portion of the query. This would, in effect, get all of the advantages of merge without the overhead of repeated expression execution. Keep in mind that this is a hidden installation parameter that applies to the entire subsystem and should be tested with extreme caution and under the guidance of IBM to see if it will help in your environment.

## *Joins, Transitive Closure, and View Merge*



- DB2 can CAST data types for unmatched types in predicates
- DB2 can use transitive closure to generate predicates and offer more index choices and table sequence for joins
  - DB2 can avoid processing redundant predicates that it generates
    - It does NOT avoid processing your redundant predicates!
- Views are merged into references much like table expressions

## Table Join on Mismatched Data Types

- DB2 will generate a CAST expression to match the data types
- This can influence table access sequence

### Before

```
SELECT E.LASTNAME
FROM   EMP E
INNER JOIN
      DEPT D
ON     E.LASTNAME = D.DEPTNAME
```

### After

```
SELECT E.LASTNAME
FROM DEPT AS D
, EMP AS E
WHERE CAST( E.LASTNAME AS VARCHAR(36) ) = D.DEPTNAME
```

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

31

DB2 query transformation has the ability to now match data types for mismatched columns. This can happen for host variables as well as for joined columns.

In this particular example we are joining two tables together by columns that do not match data types. Query transformation will introduce a CAST expression to cast the data types to a compatibility. This also can influence the table access sequence, as the optimizer will want to access the table with the expression on the join column first in order to use an index to the table without the expression on its join column. Your results may vary and the query transformation component may as well take indexes into consideration when determining which column to CAST.

## Table Join on Mismatched Data Types

- You can choose to cast yourself and DB2 will not redundantly cast
- This (meaning you) can influence table access sequence
  - Be careful when casting as you can possibly truncate

### Before

```
SELECT E.LASTNAME
FROM   EMP E
INNER JOIN
      DEPT D
ON     E.LASTNAME = CAST(D.DEPTNAME AS VARCHAR(15))
```

### After

```
SELECT E.LASTNAME
FROM   EMP E
INNER JOIN
      DEPT D
ON     E.LASTNAME = CAST(D.DEPTNAME AS VARCHAR(15))
```

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

32

You can defeat the introduction of CAST expressions for mismatched data types in query transformation by introducing your own CAST expression to do the conversion. In this way you do two things:

1. You defeat the introduction of a CAST expression by query transformation
2. You influence the table access sequence by forcing access to the table on which the CAST expression has been coded first.

This technique is a strong influence on both query transformation and the optimizer.



## Simple View Merge



- The text of a view is merged with the referencing statement
- This is very similar to table expression merge

### View Definition

```
CREATE VIEW VEMP AS
SELECT ALL EMPNO , FIRSTNAME , MIDINIT , LASTNAME , WORKDEPT
FROM EMP ;
```

### Before

```
SELECT LASTNAME
FROM VEMP
```

### After

```
SELECT EMP.LASTNAME
FROM EMP
```

In much a similar way to the merging of nested table expressions, views are literally merged into the text of the statement that references them.

This simple example demonstrates the merging of a view on the EMP table into the statement that references it. The existence of the view is not even apparent in the rewritten query.

## Join to View with Transitive Closure Applied



- For an inner join DB2 will use transitive closure to apply a common predicate to both tables
- Allows for either table to be chosen first in access path with before join filtering

### Before

```
SELECT D.DEPTNAME, E.LASTNAME
FROM   DEPT D
INNER JOIN
      VEMP E
ON D.DEPTNO = E.WORKDEPT
AND D.DEPTNO = 'A00'
```

### After

```
SELECT D.DEPTNAME
      , EMP.LASTNAME
FROM DEPT AS D
      , EMP
WHERE ( D.DEPTNO = 'A00'
      AND EMP.WORKDEPT = 'A00'
      AND D.DEPTNO = EMP.WORKDEPT)
```

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

34

This example demonstrates two things; view merge and predicate transitive closure. Here the original query joins the department table to the VEMP view on the department number column. Query transformation first merges the view into the referencing statement, and then predicate transitive closure is applied. Since there is a local predicate on the department table, `D.DEPTNO = 'A00'`, and a join predicate joining that column to the employee table, `D.DEPTNO=E.WORKDEPT` (via the view), DB2 query transformation is able to introduce a redundant predicate against the employee table. This redundant predicate, `AND EMP.WORKDEPT='A00'`, allows the optimizer to pick with table is access first depending upon table size and available indexes. More access path choices means more potential query performance.

Predicate transitive closure in this case is possible due to the fact that this is an inner join.

## Join to View with Transitive Closure Applied During Join

- For an outer join can apply transitive closure to apply a common predicate to both tables, but only as a during join predicate
- DB2 11 can also push down the predicate into a table expression
- Could possibly be used to influence table access sequence

### Before

```
SELECT D.DEPTNAME, E.LASTNAME
FROM   DEPT D
LEFT JOIN
VEMP E
ON D.DEPTNO = E.WORKDEPT
WHERE D.DEPTNO = 'A00'
```

### After

```
SELECT D.DEPTNAME
      , EMP.LASTNAME
FROM (DEPT AS D
      LEFT OUTER JOIN EMP
      ON (EMP.WORKDEPT = 'A00'
          AND D.DEPTNO = EMP.WORKDEPT)
      WHERE D.DEPTNO = 'A00' )
```

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

35

In the case of an outer join DB2 cannot apply transitive closure for the D.DEPTNO predicate. This is because if that predicate was applied to the EMP table it would eliminate the chance of null values for the EMP table in the join and that eliminates the query as an outer join, which can change the query result. Since query accuracy is paramount then transitive closure cannot be applied.

I sometimes convert inner joins to outer joins if the result will not be impacted and I need to influence the table access sequence and reduce the chance for predicate transitive closure to be applied. This is when I'm doing specific performance tuning and want DB2 to pick the outer table (table on left side of a left outer join for example) first in the table join sequence.

## Join with Transitive Closure Not Applied



- For an inner join DB2 cannot use transitive closure to apply a common predicate to both tables for subqueries
- This works for IN and LIKE
  - IN-Lists available for transitive closure in DB2 10 and 11

### Before

```
SELECT D.DEPTNAME, E.LASTNAME
FROM   DEPT D
INNER JOIN
      EMP E
ON D.DEPTNO = E.WORKDEPT
WHERE D.DEPTNO IN
      (SELECT 'A00' FROM SYSIBM.SYSDUMMY1)
```

### After

```
SELECT D.DEPTNAME
      , E.LASTNAME
FROM DEPT AS D
      , EMP AS E
WHERE ( D.DEPTNO IN ( SELECT 'A00'
                      FROM SYSIBM.SYSDUMMY1 )
      AND D.DEPTNO = E.WORKDEPT )
```

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

36

Transitive closure is not available to LIKE, IN, and subqueries in DB2 9 for z/OS. For DB2 10 and 11 In-lists are available for transitive closure.

I sometimes introduce a local predicate that is not available for transitive closure in order to avoid transitive closure and influence table access sequence, especially for inner joins.

**Join and Predicate Distribution across a UNION ALL**

- When a view or table expression contains a UNION ALL
  - Joins are distributed across the UNION ALL
  - Predicates are distributed across the UNION ALL (Also for UNION)
- What does this mean for query performance
  - Materialization can be eliminated
  - Index access can be achieved on both sides of a UNION ALL
  - There may be redundant table access

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

37

When you access a table expression or view that contains a UNION ALL you must be very careful. It is a balance between materialization, merge, and redundant table access. Every query is different and so these types of queries should always be examined in great detail to see exactly what DB2 query transformation and optimization is doing with these queries, and possibly changing them to improve the performance.

## Join Distribution Across a UNION ALL

- How many times do you think the EMP table is accessed?
- How many times is it really accessed?
- Materialization of the view is avoided

### Before

```
SELECT A.EMPNO, X.EMPNO, X.NAME
FROM   EMP A
INNER JOIN
  (SELECT B.MGRNO, B.DEPTNAME
   FROM   DEPT B
   UNION ALL
   SELECT C.RESPEMP, C.PROJNAME
   FROM   PROJ C)
AS X(EMPNO, NAME)
ON A.EMPNO = X.EMPNO
```

### After

```
( SELECT A.EMPNO, B.MGRNO, B.DEPTNAME
  FROM DEPT AS B, DSN8910.EMP AS A
  WHERE A.EMPNO = B.MGRNO
  UNION ALL
  SELECT A.EMPNO, C.RESPEMP, CAST( C.PROJNAME AS VARCHAR(36) )
  FROM PROJ AS C
       , EMP AS A
  WHERE A.EMPNO = C.RESPEMP)
```

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

38

In this example the EMP table is joined to two other tables in a referenced nested table expression. How many times is the EMP table accessed in this statement? Well, if you just looked at the original query you would think once, but the rewritten query shows that the table expression has been merged with the outer statement that references it. This means that materialization of the table expression was avoided, but the access to the EMP table was distributed across both query blocks of the UNION. Thus the EMP table is actually accessed twice in the query execution!

## Join and Predicate Distribution Across a UNION ALL



- Predicate distribution can aid in index access
- Predicate distribution can fuel query block pruning

### Before

```
SELECT A.EMPNO, X.EMPNO, X.NAME
FROM   EMP A
INNER JOIN
(SELECT B.MGRNO, B.DEPTNAME
 FROM   DEPT B
 WHERE  B.MGRNO = '000010'
 UNION ALL
 SELECT C.RESPEMP, C.PROJNAME
 FROM   PROJ C
 WHERE  C.RESPEMP = '000020')
AS X(EMPNO, NAME)
ON A.EMPNO = X.EMPNO
WHERE X.EMPNO = '000010'
```

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

39

In addition to joins being distributed across query blocks of a UNION in a table expression or view, predicates can also be distributed as well.

## Join and Predicate Distribution Across a UNION ALL



- Predicate distribution can aid in index access
- Predicate distribution can fuel query block pruning

After

```
( SELECT A.EMPNO
   , B.MGRNO
   , B.DEPTNAME
 FROM DEPT AS B
   , EMP AS A
 WHERE ( A.EMPNO = '000010'
        AND B.MGRNO = '000010'
        AND B.MGRNO = '000010'
        AND A.EMPNO = B.MGRNO)

 UNION ALL

 SELECT A.EMPNO
   , C.RESPEMP
   , CAST( C.PROJNAME AS VARCHAR(36) )
 FROM EMP AS A
   , PROJ AS C
 WHERE ( A.EMPNO = '000020'
        AND A.EMPNO = '000010'
        AND C.RESPEMP = '000010'
        AND A.EMPNO = C.RESPEMP
        AND C.RESPEMP = '000020'))
```

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

40

In this specific example the original query had a join from the EMP table to a table expression containing a UNION of two tables. There was also a local predicate coded against the EMP table that happened to also be the join column. DB2 query transformation is able to utilize join distribution, predicate distribution, and predicate transitive closure in this query. This gives the optimizer even more chances to pick the correct table access sequence for each of the joins on either side of the UNION. In addition, the optimizer can also apply a concept called query block pruning to eliminate any query block that has an impossible set of predicates. In this particular example the second block of the UNION has an impossible compound predicate (A.EMPNO = '000020' and A.EMPNO = '000010') and thus the query block is never executed.

This kind of query transformation and optimization is very useful for temporal queries and data warehouse queries. Although, you do have to be aware of repetitive table access, especially when joining to multiple table expressions or view.



## *Temporal Queries and Query Transformation*



- System-period temporal table definitions contain two tables
  - Base table
  - History table
- A query against the base table can read either base alone or both base and history table
- Query transformation determines which tables will be read
  - Dependent upon system-period specification of query
  - Query rewritten accordingly

8/16/2015

© 2015 DanL Database Consulting - presented August 2015

41

Temporal table design is something people have been doing for years, but has now been automated to a certain extent in DB2 10 for z/OS. This automation takes what would normally be weeks or months of coding in applications and moves that functionality directly into the database engine, which can be implemented in a matter of minutes!

## *New Directions for Query Transformation*



- DB2 10 introduced some major changes
- More correlation and de-correlation of queries
  - Subqueries
  - Correlated nested table expressions
- Expanded predicate transitive closure
  - IN-lists now available for transitive closure
- In-lists can be (sort of) transformed to joins
  - In-memory table built containing values
  - In-memory table joined to searched table

## DB2 11 and Query Transformation



- Improvements to System-Period Temporal Access
  - CURRENT TEMPORAL SYSTEM\_TIME special register picks from two possible access paths
    - Non-temporal access to base table only
    - Time travel access using UNION ALL
- Rewriting of Common Stage 2 to Stage 1 Predicates
  - YEAR(<date column>) op value (this is stage 2 non-indexable)
    - <date column> op DATE(value) (this is stage 1 indexable)
    - <date column> BETWEEN value AND value (this is stage 1 indexable)
  - DATE(<timestamp column>) op value (this is stage 2 non-indexable)
    - <timestamp column> op TIMESTAMP(value) (this is stage 1 indexable)
  - Value BETWEEN < column> AND < column> (this is stage 2 non-indexable)
    - < column> >= value AND < column> <= value (this is stage 1 indexable)

See IDUG DB2 11 Whitepaper for Details

## DB2 11 and Query Transformation



- Changes directed at common formats of generated queries. No user action really needed
  - Improvements to some IN and OR Predicate processing for improved index matching
  - Improved predicate pushdown of some materialized views and table expressions
- Pruning of always true and always false predicates
  - Great for those pesky “optional search” queries

### Before

```
SELECT LASTNAME
FROM EMP WHERE (EMPNO = '000010' AND 1=2)
OR (BIRTHDATE = '1954-01-01' AND 1=1)
```

### After

See IDUG DB2 11 Whitepaper  
for Details

```
SELECT LASTNAME
FROM EMP
WHERE BIRTHDATE = '1954-01-01'
```

## Summary



- Query Transformation is a little talked about and highly misunderstood component of query processing
- Knowledge of Query Transformation is important for advanced query tuning
- There are tools available now to see rewritten queries
- Eventually Query Transformation will put us out of work?



## **Understanding Query Transformation in DB2 for z/OS**

***Daniel L Luksetich  
DanL Database Consulting  
danl@db2expert.com  
WWW.DB2EXPERT.COM***

DanL Database Consulting

- Long and short term consulting
- Experience on multiple platforms (z/OS, AIX, UNIX, Windows, Linux)
- DB2 SQL Education (multi-platform)
- DB2 application and database design
- DB2 performance audits