Advanced SQL and the Power of Rewriting Queries

Tony Andrews

Themis Inc.

tandrews@themisinc.com

Often times there can 4,5,6 different ways to write an SQL query and get the same results back. What makes one better than any of the others, and is there any ones that are always better, or always worse? Sometimes rewriting 1 predicate in an SQL statement can cause optimization to change. This presentation breaks down many examples of query rewrites and how they can affect performance and optimization. Being strong in SQL is a great skill when it comes to performance and tuning of queries.".

Objectives

Presentation Objectives:

- Objective 1: To help the audience become stronger in SQL and understand the power of SQL rewrites.
- Objective 2: To help the audience become more educated in the area of performance and tuning from the application side.
- Objective 3: To help the audience understand how certain SQL statements operate within DB2.
- Objective 4: To help the audience understand the different areas of performance tuning when it comes to a program or query,
  Objective 5: To break down the many different predicate types and how they affect a query's optimization
- **Get empowered!**

Themis
Leaders in IT Education

This presentation is to show in detail one of the many ways to help get a query to run faster. As will be shown there are many factors that can have a impact on query performance, but the one that is very powerful and totally in the hands of the developers is the query rewrite.

Tuning Approaches

- **Change the SQL. Rewrite the query or predicates a different way**
- **Redesign the program flow**
- **Gather / Alter Statistics**
- **Change Physical Design**
- **System Tuning**

Themis
Leaders in IT Education

Improving SQL performance can be done in one of at least 5 ways. The first 2 ways are totally controlled by the developer. Changing the way a query is written to and keeping the same logic can often times send the optimizer down a different physical path in gathering the rows for the final result set. At times there may be 3, 4, 5, or 6 different ways to write a query and return the same result sets. The SQL must be written in a way that may be processed efficiently by the database. Redesigning program flow, and minimizing the number of time a program sends SQL statements to DB2 can have an impact.
An appropriate level of statistics about the data must be gathered to tell the optimizer about the nature of the data being accessed.
The way the physical objects are defined must be aligned with the types of queries that are to be performed. The 2 major areas here are indexing and the clustering order of data in a table .
Lastly, system tuning may be done to adjust the parameters under which the DB2 subsystem operates to effectively match the workload. Altering system parameters, tuning temporary space, and adjusting buffer pool sizes and thresholds are all examples of this type of tuning. An appropriately tuned system can affect an improvement in performance.

This presentation focuses on the first bullet, changing the SQL.

Review - DEPT Table

Partial data

DEPTNO	DEPTNAME	MGRNO
A00	SPIFFY COMPUTER SERVICE DIV.	000010
B01	PLANNING	000020
C01	INFORMATION CENTER	000030
D01	DEVELOPMENT CENTER	*<null>*
D11	MANUFACTURING SYSTEMS	000060
D21	ADMINISTRATION SYSTEMS	000070
E01	SUPPORT SERVICES	000050
E11	OPERATIONS	000090
E21	SOFTWARE SUPPORT	000100
F22	BRANCH OFFICE F2	*<null>*
G22	BRANCH OFFICE G2	*<null>*
H22	BRANCH OFFICE H2	*<null>*
I22	BRANCH OFFICE I2	*<null>*
J22	BRANCH OFFICE J2	*<null>*

A review of some of the data in the DB2 sample table DEPT.

Review - EMP Table

EMPNO	DEPTNO	LASTNAME
000010	A00	HAAS
000020	B01	THOMPSON
000030	C01	KWAN
000050	E01	GEYER
000060	D11	STERN
000070	D21	PULASKI
000090	E11	HENDERSON
000100	E21	SPENSER
000110	A00	LUCCHESI
000120	A00	O'CONNELL
000130	C01	QUINTANA
000140	C01	NICHOLLS
000150	D11	ADAMSON
000160	D11	PIANKA
000170	D11	YOSHIMURA
…..		

Partial data

A review of some of the data in the DB2 sample table EMP.

Review – EMPPROJACT Data

Partial data

EMPNO	PROJNO	ACTNO
000130	IF1000	90
000130	IF1000	100
000140	**IF1000**	**90**
000140	**IF2000**	**100**
000140	**IF2000**	**100**
000140	**IF2000**	**110**
000140	**IF2000**	**110**
000150	MA2112	60
000150	MA2112	180

Themis
Leaders in IT Education

A review of some of the data in the DB2 sample table EMPPROJACT. An employee will have a rows or rows in this table if they are currently working on a project (or projects). They could also have multiple rows if they are working multiple activities within a specific project. There is a 0-many relationship between the EMP table and this EMPPROJACT table.

Example 1

Provide a report of employees who work on Project 'IF2000'.
Show Employee Number, Last Name, and Salary.

Note: Some employees may not be working on projects
Some employees may be working on 1 project
Some employee may be working on multiple projects
Some employees may be working on multiple activities within same project

Tables Needed:

- EMP Employee table
- EMPPROJACT Employee/Project/Activity

Themis
Leaders in IT Education

This request for a report requires 2 tables. There is a 0-many relationship between the tables.

Example 1 – Solution 1

If we try a join, we get the following duplicate results:

Join Logic

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

EMPNO	LASTNAME	SALARY
000030	KWAN	38250.00
000140	NICHOLS	28420.00
000140	NICHOLS	28420.00
000140	NICHOLS	28420.00
000140	NICHOLS	28420.00

Duplicates because this employee works on multiple activities associated with project 'IF2000'

Themis
Leaders in IT Education

The first thought to fulfill this request would be to code and execute an SQL Join between the 2 tables, zeroing in on the employees that are a part of Project = 'IF2000'.

As can be seen, when a join is coded (and knowing the relationship between the tables) there is a potential for duplicates that show up due to the multiple activities some employees can be assigned for that project. This is where developers need to be aware and know the relationship between tables. I have seen many incident reports come about in production where a query was running for awhile and working only because it was lucky that there existed no duplicate in the data. But when more data was added, duplicates came about and the program/query then issued duplicate rows.

KNOW YOUR DATA and RELATIONSHIPS.

What can be done to eliminate the duplicates?

Example 1 – Solution 1

Need a Distinct or Group By to handle duplicates:

Join Logic

```
SELECT DISTINCT E.EMPNO, E.LASTNAME, E.SALARY
FROM  EMP                E,
      EMPPROJACT         EP
WHERE E.EMPNO = EP.EMPNO
  AND EP.PROJNO = 'IF2000'
```

Add a DISTINCT?

EMPNO	LASTNAME	SALARY
000030	KWAN	38250.00
000140	NICHOLS	28420.00

Themis
Leaders in IT Education

The easiest and first thought most developers think of is to add the Distinct word as part of the select, which will eliminate the duplicates.

In order for the RDBMS to eliminate duplicates via the Distinct, it may load the data into a Sort Workfile and execute a sort in order to get the data ordered. It then pulls out the unique values. Distinct does not always cause a sort to take place as DB2 often does sort avoidance. The only way to know for sure would be executing a DB2 Explain.

If there is a sort specific to the distinct, there is some overhead involved in this sort. A workfile must be allocated, then loaded, then sorted, and from there unique values are pulled for the final result set. Sorts are pretty fast in DB2, and my first questions when I see a sort occurring are: Can it be eliminated? How big is the sort? Sorts are expensive as their size.

Group By will also eliminate duplicates and works the same as a distinct.

Example 1 – Solution 2
Non Correlated Subquery

Non Correlated Subquery Logic

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

EMPNO	LASTNAME	SALARY
000030	KWAN	38250.00
000140	NICHOLS	28420.00

OR: Code a Non Correlated Subquery to eliminate duplicates

There are other ways to eliminate the duplicates needed for this example, both ways requiring a Subquery. These options are available because the there is no data needed to be retrieved from the EMPPROJACT table. This give us the option to take it out of the join, and move it into a subquery.

The first way is to code a Non Correlated Subquery, as shown on this page.

This query is checking to see if each EMPNO value is in the list of EMPNOs generated by the subquery. An EMPNO could be in the list multiple times, but the results will not show the EMPNO multiple times. Remember that the list of values coming out of a non-correlated subquery will have its results sorted in order to eliminate duplicates in the list, and to get the list in ascending order.

Db2 will sometimes take the values from the non correlated subquery, and instead of keeping the list in a an In-List, will put the values into a temporary table and then use it in a join to the outer table.

Example 1 – Solution 3
Correlated Subquery

Exists Correlated Subquery Logic

```
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' )
```

EMPNO	LASTNAME	SALARY
000030	KWAN	38250.00
000140	NICHOLS	28420.00

OR: Code a Correlated Subquery to eliminate duplicates

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

The logic here is as each EMPNO value is passed to the subquery for execution, the question "Does that join condition exist for the particular EMPNO value?" is processed. Even if it exists multiple times in the subquery, the value will still only get written out once.

Writing it this way eliminates any sorts that could be taking place in a jpin, but the subquery will get executed multiple times.

Which way is best ?

1). Using the Distinct
2). Writing a subquery using the 'In'
3). Writing a subquery using the 'Exists'

Of course the answer is 'It depends'.

Example 1 – Solution 4
Intersect Logic

Intersect Logic

```
WITH X AS
  (SELECT E.EMPNO
   FROM EMP E
   INTERSECT ALL
   SELECT EPA.EMPNO
   FROM EMPPROJACT EPA
   WHERE EPA.PROJNO = 'IF2000'
  )

SELECT E.EMPNO, E.LASTNAME, E.SALARY
FROM EMP E, X
WHERE E.EMPNO = X.EMPNO
ORDER BY 1
 ;
```

Intersect to get EMPNOs, then join to get LASTNAME and SALARY.

EMPNO	LASTNAME	SALARY
000030	KWAN	38250.00
000140	NICHOLS	28420.00

Themis
Leaders in IT Education

The SQL NTERSECT can also be used to find the EMPNOs that are in the EMP table and also in the EMPPROJACT table under project ‘IF2000’ . A join is then need back to the EMP table in order to get the LASTNAME and SALARY.

Example 1 – Solution 4 Count(*) logic

Count(*) Logic

```
SELECT E.EMPNO, E.LASTNAME, E.SALARY
FROM EMP E
WHERE 0 <
      (SELECT COUNT(*)
      FROM EMPPROJACT EPA
      WHERE EPA.EMPNO = E.EMPNO
          AND EPA.PROJNO = 'IF2000')
                                ;
```

Count Logic Typically not good.

EMPNO	LASTNAME	SALARY
000030	KWAN	38250.00
000140	NICHOLS	28420.00

Themis
Leaders in IT Education

This is pretty common logic for developers and analyst because it makes a little more straightforward sense. But this query is the most expensive in execution time and CPU and would not be a recommendation.

This is due to the fact that it must first count up all the rows that meet the criteria for each specific employee number, instead of stopping at the first occurrence (in the case of EXIST logic) or the one time building of an IN list. There is no need to always count the number of rows for each employee number working on 'IF20000' and comparing to 0. The counting of rows at times can have considerable overhead involved.

This logic seems to be found in much older code at companies and should be rewritten whenever found.

Example Differences

- Distinct. By coding the distinct, DB2 may sort the final result set to eliminate duplicates.
- Non Correlated Subquery. Subquery executes once and puts results either into a list or table to feed the outer query.
- Correlated Subquery. Subquery gets executed multiple times, executing for each unique EMPNO from the outer query.
- Count(*). Counts every row where each EMPNO value exists in the EMPPROJACT table and compares the count to 0. Very inefficient!
- INTERSECT ALL. First gets each EMPNO from top query that intersects (exists) in the EMPPROJACT table, Then join to EMP table for LASTNAME and SALARY.

Which one is best? It depends:

How big is the Distinct sort? The larger the size of sorts the more expensive the query and runtime. This is typically the worst. What does the join process look like? Are there indexes involved in the join? Which join method? Any join sorts?

Non Correlated Subquery. This is typically better because the sort is smaller than the Distinct sort. The sort is done on only one column for however many values come out of the subquery.

Correlated Subquery. This subquery will execute multiple times so it is imperative that the subquery uses a index when it gets processed for each value sent to it via the join. If the subquery can execute as 'Index Only' and there is matching index occurring, executing many times can be very fast and efficient.

So the answer is 'It Depends'. The nice thing about these is that we have options for duplicate data at times and they all execute very different within DB2. Depending on the data and physical design, one will usually run better than the others.

Example 2

Provide a list of employees that major in both 'MAT' and 'CSI'

Note: Each employee will have a row in the table for each major

EMPMAJOR

EMPNO	MAJOR
E1	MAT
E1	CSI
E2	MAT
E3	CSI
E4	ENG

Themis
Leaders in IT Education

This screen starts the second example. This is a typical table design where an ID / ACCT_NUM may have multiple rows each unique based on an ACCT type, status, code, ….

This table is not one of the DB2 sample tables but a good example of a common design. In this example, how do we find the employee numbers that have both a row with value of 'MAT' in the MAJOR column, and also a row with a value of 'CSI' in the MAJOR column?

Example 2 – Solution 1

PROBLEM: Find all employees who major in math (MAT) and computer science (CSI).

EMPMAJOR

EMPNO	MAJOR
E1	MAT
E1	CSI
E2	MAT
E3	CSI
E4	ENG

Group By / Having Logic:

```
SELECT EMPNO
FROM   EMPMAJOR
WHERE  MAJOR IN ('MAT', 'CSI')
GROUP BY EMPNO
HAVING COUNT(*) = 2;
```

Themis
Leaders in IT Education

The first solution would be to code a query with a Group BY and Having clause to see which EMPNO(s) have both (a count of 2) rows that have a ‘MAT’ and ‘CSI’.

Example 2 – Solution 2

PROBLEM: Find all employees who major in math (MAT) and computer science (CSI).

EMPMAJOR

EMPNO	MAJOR
E1	MAT
E1	CSI
E2	MAT
E3	CSI
E4	ENG

Quota Query Logic

```
SELECT DISTINCT EM1.EMPNO
FROM   EMPMAJOR AS EM1
WHERE 2 =
    (SELECT  COUNT(*)
     FROM    EMPMAJOR EM2
     WHERE  EM2.EMPNO = EM1.EMPNO
        AND  EM2.MAJOR IN  ('MAT', 'CSI') ;
```

Themis
Leaders in IT Education

The next solution would be to code what's called a 'Quota Query' where the number 2 is used in a correlated subquery.

Example 2 – Solution 3

PROBLEM: Find all employees who major in math (MAT) and computer science (CSI).

EMPMAJOR

EMPNO	MAJOR
E1	MAT
E1	CSI
E2	MAT
E3	CSI
E4	ENG

Self Join Logic:

```
SELECT EMPNO
FROM EMPMAJOR AS EMP1 JOIN
     EMPMAJOR AS EMP 2
  ON EMP1.EMPNO = EMP2.EMPNO
WHERE EMP1.MAJOR = 'MAT'
  AND EMP2.MAJOR = 'CSI';
```

Themis
Leaders in IT Education

And the last way to code logic for this would be using an SQL 'Self Join' where you join the EMP table to itself by EMPNO. And by joining this way we code where in one table there is a row with 'MAT' and self joining back but looking for a 'CSI' row.

Example 3 – Solution 1

PROBLEM: Find the youngest employee in each department.

HINT: Youngest employee would be the one with highest birthdate

```
SELECT E1.EMPNO, E1.LASTNAME
FROM EMP AS  E1
WHERE E1.BIRTHDATE  = (SELECT MAX(E2.BIRTHDATE)
                       FROM EMP E2
                       WHERE E2.DEPTNO = E1.DEPTNO)
```

Correlate Subquery by Deptno

This screen starts the third example. Each employee row in the EMP table contains the department that the employee works in and their birth date. This query is to find the youngest employee in each department.

Solution 1 would be coding a query with a correlated subquery.

Example 3 – Solution 2

PROBLEM: Find the youngest employee in each department.

```
SELECT E1.EMPNO, E1.LASTNAME
FROM EMP E1
WHERE (E1.DEPTNO,E1.BIRTHDATE)IN(SELECT E2.DEPTNO,
                                        MAX(E2.BIRTHDATE)
                                 FROM EMP E2
                                 GROUP BY E2.DEPTNO)
```

FULL SELECT

ROW-VALUE-EXPRESSION

Themis
Leaders in IT Education

This solution is using what is called a Row-Value expression in SQL where the 'IN' predicate contains 2 values for each entry in the in list.

The in-list would look like (deptno1 and max birthdate, deptno2 and max birthdate, …)

Example 4 – Solution 1

List all departments where a department average bonus is greater that its department average salary.

```
SELECT   D.DEPTNAME
        ,D.LOCATION
FROM DEPT D
WHERE   (SELECT AVG(BONUS) FROM EMP E1
         WHERE D.DEPTNO = E1.DEPTNO)
                  >
        (SELECT AVG(SALARY) FROM EMP E2
         WHERE D.DEPTNO = E2.DEPTNO)
```

Themis
Leaders in IT Education

Example 4 needs some aggregation (average bonus and average salary) calculations for its logic. This first solution is by coding correlated subqueries that are scalar fullselects to compare.

Example 4 – Solution 2

List all departments where a department average bonus is greater that its department average salary.

```
SELECT  D.DEPTNO, D.LOCATION, AVG(BONUS), AVG(SALARY)
FROM THEMIS81.DEPT AS   D   INNER JOIN
        THEMIS81.EMP   AS   E1   ON D.DEPTNO  =  E1.DEPTNO
GROUP BY D.DEPTNO, D.LOCATION
HAVING AVG(E1.BONUS)  >
          (SELECT AVG(SALARY)
           FROM THEMIS81.EMP E2
           WHERE E2.DEPTNO = D.DEPTNO)
```

This solution uses the Group By and Having SQL logic to obtain the results. Notice that the Having clause can also be correlated. This solution allows to also have the aggregated amounts as part of the output.

Example 4 – Solution 3

List all departments where a department average bonus is greater that its department average salary.

```
WITH X AS
   (SELECT  DEPTNO,  AVG(BONUS) AS AVG_BONUS,
                          AVG(SALARY) AS AVG_SALARY
    FROM EMP
   GROUP BY DEPTNO)
SELECT D.DEPTNO, D.DEPTNAME, X.AVG_BONUS, X.AVG_SALARY
FROM DEPT D, X
WHERE X.AVG_BONUS > X.AVG_SALARY
  AND X.DEPTNO = D.DEPTNO
```

Solution 3 uses a Common Table Expression (CTE) where the averages are calculated ahead of time and materialized into a temp table that can the be used in a direct join to the DEPT table. This solution allows to also have the aggregated amounts as part of the output.

These are great example of result sets that need both detail data from a table along with aggregated data in each line.

DB2 Explain

QUERYNO	QBLOCKNO	PROGNAME	PLANNO	METHOD	CREATOR	TNAME	ACCESSTYPE
1	1	DSNESM68	1	0	THEMIS81	DEPT	R
1	2	DSNESM68	1	0	THEMIS81	EMP	I
1	3	DSNESM68	1	0	THEMIS81	EMP	I
2	1	DSNESM68	1	0	THEMIS81	DEPT	I
2	1	DSNESM68	2	2	THEMIS81	EMP	R
2	2	DSNESM68	1	0	THEMIS81	EMP	I
3	1	DSNESM68	1	0	THEMIS81	X	R
3	1	DSNESM68	2	1	THEMIS81	DEPT	I
3	2	DSNESM68	1	0	THEMIS81	EMP	R
3	2	DSNESM68	2	3			

Notice the different access paths

Themis
Leaders in IT Education

This is an example of the DB2 Explain output for the access paths chosen for each of the previous 3 queries. Notice each one is very different than the others.

Notice the ‘X’ table. This is the CTE name given in solution 3. By seeing this table name in the DB2 Explain output tells us that the table will be materialized.

Optimizer Costing

```
SELECT QUERYNO,COST_CATEGORY,PROCMS,PROCSU,REASON
FROM DSN_STATEMNT_TABLE
ORDER BY QUERYNO
WITH UR
;
--------+---------+---------+---------+---------+---------+---------+---
  QUERYNO  COST_CATEGORY       PROCMS       PROCSU  REASON
--------+---------+---------+---------+---------+---------+---------+---
        1  A                        11           11
        2  B                   2671439      2606281  HAVING CLAUSE
        3  B                      5461         5328  TABLE CARDINALITY
```

This is the guesstimated runtime costs for the previous 3 queries. Notice big differences in costing numbers.

Example 5 – Solution 1

List employees that are not working on projects. This would be those EMP rows that do not have EMPNO values in the EMPPROJACT table.

```
SELECT E.EMPNO, E.LASTNAME
FROM EMP E
WHERE E.EMPNO NOT IN
        (SELECT EPA.EMPNO
        FROM EMPPROJACT EPA)
```

Themis
Leaders in IT Education

This is very typical programming logic for any relational database where you need to know the rows in 1 table where the primary key id not in another table: 3 ways to do this (NOT IN, NOT EXISTS, ANTI JOIN) .

This is an example of the 'NOT IN' logic.

Example 5 – Solution 2

List employees that are not working on projects. This would be those EMP rows that do not have EMPNO values in the EMPPROJACT table.

```
SELECT E.EMPNO, E.LASTNAME
FROM EMP E
WHERE NOT EXISTS
     (SELECT 1
      FROM EMPPROJACT EPA
      WHERE EPA.EMPNO = E.EMPNO)
```

Solution 2 shows the 'NOT EXISTS' correlated subquery way to code for the results.

Example 5 – Solution 3

List employees that are not working on projects. This would be those EMP rows that do not have EMPNO values in the EMPPROJACT table.

```
SELECT E.EMPNO, E.LASTNAME
FROM EMP E   LEFT JOIN
     EMPPROJACT EPA ON EPA.EMPNO = E.EMPNO
WHERE EPA.EMPNO IS NULL
```

Solution 3 is called the 'ANTI JOIN' where you code up an outer join, and then ask for the rows from the null supplying table where the joined column in null. This specifies that a particular join key was not found on the other table.

Example 5 – Solution 4

List employees that are not working on projects. This would be those EMP rows that do not have EMPNO values in the EMPPROJACT table.

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

Themis
Leaders in IT Education

One more solution using one of the newer SQL SET operation EXCEPT/EXCEPT ALL. What in the first output is an exception to what is in the second output.

Optimizer Costing

QUERYNO	COST_CATEGORY	PROCMS	PROCSU
1	A	5931	5083
2	A	27330	23419
3	A	17222	14758
4	A	35040	30026

Themis
Leaders in IT Education

And with different access paths comes different costing numbers.

Example 6: Max Date Row

Find the row with the most current date value when there exists multiple rows with same key value

Subquery

```
SELECT ...
  FROM TABLE T1
WHERE T1.PK = ?
  AND   T1.DATE =
     (SELECT MAX(T2.DATE)
      FROM TABLE T2
      WHERE T2.PK = T1.PK)
```

Solution 1

Themis
Leaders in IT Education

Another very common programming task with tables that contain multiple rows with the same key value, but are different based on a timestamp column.

Programming logic typically wants us to get the most current row (Max Timestamp column). This screen 2 different ways to code for this.

Works well with multiple join fields, or max number of other fields, or getting the min instead too… If you have a possibility of multiple max dates or timestamps containing same values, you can use a second field to narrow it down in the AND clause.

For example: Table EMP2 contains duplicate rows with an UPD_TSP column.

```
SELECT EMPNO, UPD_TSP
FROM EMP2  T1
WHERE T1.EMPNO = '000010'
   AND  T1.UPD_TSP =
        (SELECT MAX(T2.UPD_TSP)
         FROM EMP2 T2
         WHERE T2.EMPNO = T1.EMPNO)
;
SELECT T1.EMPNO, T1.UPD_TSP
FROM EMP2 T1 LEFT JOIN
       EMP2 T2 ON T1.UPD_TSP < T2.UPD_TSP
                AND T1.EMPNO = T2.EMPNO
WHERE T1.EMPNO = '000010'
  AND T2.UPD_TSP IS NULL
```

Example 6: Max Date Row

Find the row with the most current date value when there exists multiple rows with same key value

Self Join

```
SELECT ...
  FROM TABLE T1 LEFT JOIN
          TABLE T2 ON T1.DATE < T2.DATE
                  AND T1.PK = T2.PK
WHERE T1.PK = ?
    AND T2.DATE IS NULL
```

Solution 2

Themis
Leaders in IT Education

Another very common programming task with tables that contain multiple rows with the same key value, but are different based on a timestamp column.

Programming logic typically wants us to get the most current row (Max Timestamp column). This screen 2 different ways to code for this.

Works well with multiple join fields, or max number of other fields, or getting the min instead too… If you have a possibility of multiple max dates or timestamps containing same values, you can use a second field to narrow it down in the AND clause.

For example: Table EMP2 contains duplicate rows with an UPD_TSP column.

```
SELECT EMPNO, UPD_TSP
FROM EMP2  T1
WHERE T1.EMPNO = '000010'
   AND  T1.UPD_TSP =
        (SELECT MAX(T2.UPD_TSP)
         FROM EMP2 T2
         WHERE T2.EMPNO = T1.EMPNO)
;
SELECT T1.EMPNO, T1.UPD_TSP
FROM EMP2 T1 LEFT JOIN
       EMP2 T2 ON T1.UPD_TSP < T2.UPD_TSP
                AND T1.EMPNO = T2.EMPNO
WHERE T1.EMPNO = '000010'
  AND T2.UPD_TSP IS NULL
;
```

Example 7: Detail with Aggregation

Solution 1: Scalar Fullselect in the SELECT

```
SELECT E1.EMPNO, E1.LASTNAME, E1.DEPTNO, E1.SALARY,
        (SELECT DEC(AVG(E2.SALARY),7,2)
         FROM EMP E2
         WHERE E2.DEPTNO = E1.DEPTNO) AS AVG_SAL
FROM EMP E1
WHERE E1.DEPTNO < 'D01'
```

	EMPNO	LASTNAME	DEPTNO	SALARY	AVG_SAL
1	000120	O'CONNELL	A00	29250.00	45312.50
2	000110	LUCCHESI	A00	46500.00	45312.50
3	000011	HAAS	A00	52750.00	45312.50
4	000010	HAAS	A00	52750.00	45312.50
5	000020	THOMPSON	B01	41250.00	41250.00
6	000030	KWAN	C01	38250.00	30156.66
7	000140	NICHOLLS	C01	28420.00	30156.66
8	000130	QUINTANA	C01	23800.00	30156.66

Themis
Leaders in IT Education

Aggregate Partitioning: Many ways to get detail along with aggregation, even by department. This first example shows coding the Scalar Fullselect in the 'SELECT' portion of the query, and the second example shows getting the same result coding a nested table expression.

Another way is using the newer PARTITION BY clause:

```
SELECT DEPTNO, EMPNO, SALARY,
  AVG   (SALARY)
  OVER (PARTITION BY DEPTNO ) AS SUM_SAL
FROM EMP
```

Example 7: Detail with Aggregation

Solution 2: Table Expression

```
SELECT E.EMPNO, E.LASTNAME, E.DEPTNO, E.SALARY,
X.AVG_SAL
FROM EMP E,
   (SELECT DEPTNO, DEC(AVG(SALARY),7,2) AS AVG_SAL
     FROM EMP
     WHERE DEPTNO < 'D01'
     GROUP BY DEPTNO) AS X
WHERE E.DEPTNO < 'D01'
 AND E.DEPTNO = X.DEPTNO
```

	EMPNO	LASTNAME	DEPTNO	SALARY	AVG_SAL
1	000120	O'CONNELL	A00	29250.00	45312.50
2	000110	LUCCHESI	A00	46500.00	45312.50
3	000011	HAAS	A00	52750.00	45312.50
4	000010	HAAS	A00	52750.00	45312.50
5	000020	THOMPSON	B01	41250.00	41250.00
6	000030	KWAN	C01	38250.00	30156.66
7	000140	NICHOLLS	C01	28420.00	30156.66
8	000130	QUINTANA	C01	23800.00	30156.66

Aggregate Partitioning: Many ways to get detail along with aggregation, even by department. This first example shows coding the Scalar Fullselect in the 'SELECT' portion of the query, and the second example shows getting the same result coding a nested table expression.

Another way is using the newer PARTITION BY clause:

```
SELECT DEPTNO, EMPNO, SALARY,
  AVG   (SALARY)
  OVER (PARTITION BY DEPTNO ) AS SUM_SAL
FROM EMP
```

Example 7: Detail with Aggregation

Solution 3: Partition By

```
SELECT LASTNAME, DEPTNO, SALARY,
   DEC(AVG(SALARY) OVER (PARTITION BY DEPTNO), 7, 2)
                                                AS AVGSAL
FROM EMP
WHERE DEPTNO < 'D01'
```

	EMPNO	LASTNAME	DEPTNO	SALARY	AVG_SAL
1	000120	O'CONNELL	A00	29250.00	45312.50
2	000110	LUCCHESI	A00	46500.00	45312.50
3	000011	HAAS	A00	52750.00	45312.50
4	000010	HAAS	A00	52750.00	45312.50
5	000020	THOMPSON	B01	41250.00	41250.00
6	000030	KWAN	C01	38250.00	30156.66
7	000140	NICHOLLS	C01	28420.00	30156.66
8	000130	QUINTANA	C01	23800.00	30156.66

Themis
Leaders in IT Education

Aggregate Partitioning: Many ways to get detail along with aggregation, even by department. This first example shows coding the Scalar Fullselect in the 'SELECT' portion of the query, and the second example shows getting the same result coding a nested table expression.

Another way is using the newer PARTITION BY clause:

```
SELECT DEPTNO, EMPNO, SALARY,
  AVG   (SALARY)
  OVER (PARTITION BY DEPTNO ) AS SUM_SAL
FROM EMP
```

Example 7: Detail with Aggregation

Sometimes can be done with a join.
Different query than previous ones.

```
SELECT D.DEPTNO, D.DEPTNAME, AVG(SALARY) AS AVG_SAL
FROM DEPT D LEFT OUTER JOIN
        EMP E   ON D.DEPTNO = E.DEPTNO
GROUP BY D.DEPTNO, D.DEPTNAME
```

	DEPTNAME	LOCATION	MAXSAL
1	SPIFFY COMPUTER SERVICE DIV.		52750.00
2	PLANNING		41250.00
3	INFORMATION CENTER		38250.00
4	DEVELOPMENT CENTER		NULL
5	MANUFACTURING SYSTEMS		32250.00
6	ADMINISTRATION SYSTEMS		36170.00
7	SUPPORT SERVICES		40175.00
8	OPERATIONS		29750.00
9	SOFTWARE SUPPORT		26150.00
10	BRANCH OFFICE F2		NULL
11	BRANCH OFFICE G2		NULL
12	BRANCH OFFICE H2		NULL
13	BRANCH OFFICE I2		NULL
14	BRANCH OFFICE J2		NULL

Aggregate Partitioning: Many ways to get detail along with aggregation, even by department. This first example shows coding the Scalar Fullselect in the 'SELECT' portion of the query, and the second example shows getting the same result coding a nested table expression.

Another way is using the newer PARTITION BY clause:

```
SELECT DEPTNO, EMPNO, SALARY,
  AVG   (SALARY)
  OVER (PARTITION BY DEPTNO ) AS SUM_SAL
FROM EMP
```

Ex 8: Left Join – ON Clause Extensions

```
SELECT    D.DEPTNO, D.DEPTNAME,
          D.MGRNO, E.LASTNAME
FROM DEPT D LEFT OUTER JOIN EMP E
     ON       D.MGRNO = E.EMPNO
      AND     D.DEPTNO LIKE 'D%'
```

DEPTNO	DEPTNAME	MGRNO	LASTNAME
A00	SPIFFY COMPUTER SERVICE DIV.	000010	*<null>*
B01	PLANNING	000020	*<null>*
C01	INFORMATION CENTER	000030	*<null>*
E01	SUPPORT SERVICES	000050	*<null>*
D01	DEVELOPMENT CENTER	*<null>*	*<null>*
D11	MANUFACTURING SYSTEMS	000060	***STERN***
D21	ADMINISTRATION SYSTEMS	000070	***PULASKI***
E11	OPERATIONS	000090	*<null>*
E21	SOFTWARE SUPPORT	000100	*<null>*
F22	BRANCH OFFICE F2	*<null>*	*<null>*
G22	BRANCH OFFICE G2	*<null>*	*<null>*
H22	BRANCH OFFICE H2	*<null>*	*<null>*
I22	BRANCH OFFICE I2	*<null>*	*<null>*
J22	BRANCH OFFICE J2	*<null>*	*<null>*

By adding the predicate D.DEPTNO LIKE ‘D%’ to the ON clause only filters what is to be joined. It does not filter data from the final result set.

Note: Only predicates in a WHERE clause will filter rows from a result set. Predicates in an ON clause determines which rows should return data from the other table to be joined (non driver table

Sometimes in logic we want to ONLY go to another table to retrieve joined values based on certain condition(s). This is a good way to go about it.

Ex 8: Left Join – ON Clause Extensions

```
SELECT      D.DEPTNO, D.DEPTNAME, D.MGRNO,
            CASE
              WHEN D.DEPTNO LIKE 'D%' THEN  E.LASTNAME
              ELSE                          NULL
            END  AS LASTNAME
FROM DEPT D LEFT OUTER JOIN EMP E
         ON  D.MGRNO = E.EMPNO
```

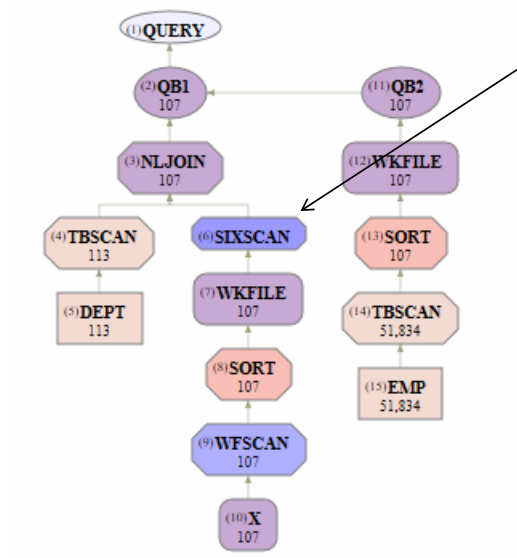
Themis
Leaders in IT Education

Many will use the CASE logic to decide whether to show some columns from the other table or not. In this example a join will take place for every department over to the employee table, and at the last minute decide whether to show the lastname or not.

V11 Sparse Indexing

Seen more especially with table expressions.

```
WITH X AS
   (SELECT  DEPTNO,
    AVG(SALARY) AS AVG_SAL
    FROM EMP
   GROUP BY DEPTNO)
SELECT D.DEPTNO,
          D.DEPTNAME,
         X.AVG_SALARY
FROM DEPT D, X
WHERE D.DEPTNO = X.DEPTNO
```



Themis
Leaders in IT Education

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.

Predicate Rewrites

Predicate Types:
- **Equal, Range, IN-List, Subquery, Not**

General Predicate Rules:
- **Predicates AND'd together are typically more efficient than predicates with OR logic**
- **Use constants whenever possible**
- **Write predicates to be indexable**
- **No functions/mathematics on columns in predicates**
- **Avoid NOT logic whenever possible**
- **Different predicate type generate different filter factors**
- **z/OS avoid stage 2 predicates**

NOTE: Some predicates are automatically rewritten by the optimizer. Check the Db2 explain output!

Even writing a predicate a different way may change the filter factor enough for the optimizer to change it optimization access path choice. Or rewriting a predicate to be indexable if not already!

Predicate Rewrites

For example: Predicates AND'd vs OR'd

```
SELECT EMPNO, LASTNAME, EDLEVEL, GENDER
FROM EMP
WHERE DEPTNO = 'A00'
            AND ( GENDER = 'F' OR EDLEVEL > 16 )
```

Or

```
SELECT EMPNO, LASTNAME, EDLEVEL, GENDER
FROM EMP
WHERE (DEPTNO = 'A00' AND GENDER = 'F' )
      OR (DEPTNO = 'A00' AND EDLEVEL > 16 )
```


Most likely these two queries will take different optimization paths. Which one is best? Well of course 'It Depends'.

Predicate Rewrites

For example: All of the following predicates will return the same result set. These are the only 'E' in the data.

```
SELECT EMPNO, LASTNAME
FROM EMP
WHERE DEPTNO IN ('E01', 'E11', 'E21')

WHERE DEPTNO LIKE 'E%'

WHERE DEPTNO BETWEEN 'E01' and 'E21'

WHERE DEPTNO IN
        (SELECT DEPTNO FROM DEPT
         WHERE DEPTNO LIKE 'E%')
```


Even writing a predicate a different way may change the filter factor enough for the optimizer to change it optimization access path choice. Different predicate types have different algorithms for generating filter factors. Sometimes what may seem like a simple change may change the optimization.

Thank you for allowing me to share some of my experience and knowledge today!

"I have seen that when the developers get educated, good SQL programming standards are in place, and program walkthroughs are executed correctly, incident reporting stays low, CPU costs do not get out of control, and most performance issues are found before promoting code to production."

Tony Andrews

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

- DB1032 – DB2 for z/OS Performance and Tuning
- DB1041 – DB2 z/OS Advanced SQL
- DB1037 – Advanced Query Tuning using IBM Data Studio
- SQ1010 – Cross Platform Advanced SQL
- DB1006 – DB2 LUW Advanced Query Tuning using IBM Data Studio

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

Tony's Email: tandrews@themisinc.com
Twitter: @tonyandrews12

Themis
Leaders in IT Education

Thank you for allowing me to share some of my experience and knowledge today!

Tony Andrews
tandrews@themisinc.com

- I hope that you learned something new today
- I hope that you are a little more inspired when it comes to SQL coding and performance tuning