Top 25+ DB2 SQL Tuning Tips for Developers

Presented by Tony Andrews, Themis Inc.

tandrews@themisinc.com
Objectives

By the end of this presentation, developers should be able to:

- Understand what SQL Optimization is
- Understand Stage 1 versus Stage 2 predicates
- Sargeable versus Non-Sargeable predicates
- Residual predicates
- Understand what the DB2 Explain Tool does
- Understand what the Visual Explain can tell us
- Understand what makes queries run too long
- Learn what to do when they see a table scan taking place.
- Learn 10 steps to tuning a query
My Experience Shows

• That the majority of performance problems among applications are caused by poorly coded programs or improperly coded SQL.
• That poor performing SQL is responsible for as much as 80 percent of response-time issues.
• That most developers need educated in how to read and analyze the DB2 Explain output.
• There are many SQL developers in the IT industry, but I have found that less than 10% of them really know the performance issues involved with SQL programming or how to fix them.
My Experience Shows

• The largest performance gains are often obtained from tuning application code.
• That performance, no matter how good IT shops think it is, can always be better.
• That a little bit of training, shop coding standards, and program walkthroughs can totally change the culture of IT development departments, and minimize performance issues and incident reporting in production.
Developers should always program with performance in mind.

Programmers should always have two goals in mind when developing programs and applications for user groups.

- To get the correct results for requested data
- To get the results back as quickly as possible

So many times programmers today lose sight of the second goal. They either:

- Do not know what to do to get programs to run faster
- Blame other pieces of their environment (database, network, TCP/IP, workstation, operating system, etc.), or
- Think that the time spent processing the data was pretty good based on the amount of data.
The DB2 Optimizer

- Catalog Statistics
- Object Definitions
- SQL
- Access Path
- Access Path Hint
- RID Pool Sorts

Themis
Optimizer Predicate Categories

- Indexable or Non Indexable
- Stage 1 or Stage 2, Sargeable or Non-Sargeable
- Predicate type (Equal, In, Between, Range, Subq, etc.)
- Filter Factor
- When applied during join processing (Before, During, After)
Top 20+ Tuning Tips #1

1). Take out any / all Scalar functions coded on columns in predicates.

For example, this is the most common:

```
SELECT EMPNO, LASTNAME
FROM EMPLOYEE
WHERE YEAR(HIRE_DATE) = 2005
```

Should be coded as:

```
SELECT EMPNO, LASTNAME
FROM EMPLOYEE
WHERE HIRE_DATE BETWEEN '2005-01-01' and '2005-12-31'
```

V9 z/OS: Can now create indexes on SQL expressions.
LUW: Optimizer much more powerful in predicate rewrites
V8 LUW: Allows generated columns
Top 25+ Tuning Tips #1

Use the Visual Explain, notice the predicate being stage 2 and the index beginning with PRSTDATE was not chosen.
Top 25+ Tuning Tips #1

Notice the DB2 LUW Optimizer Rewrite

Database server transformed query for access plan

```sql
SELECT Q1.EMPNO AS "EMPNO", Q1.LASTNAME AS "LASTNAME", Q1.SALARY AS "SALARY", Q1.HIREDATE AS "HIREDATE"
FROM THEMIS82.EMP AS Q1
WHERE ('01/01/2009' <= Q1.HIREDATE) AND (Q1.HIREDATE <= '12/31/2009')
```
Top 25+ Tuning Tips #2

2). Take out any / all mathematics coded on columns in predicates.

For example:

```
SELECT EMPNO, LASTNAME               SELECT EMPN, LASTNAME
FROM EMPLOYEE                                   FROM EMPLOYEE
WHERE SALARY * 1.1 > 50000.00         WHERE HIREDATE − 7 DAYS > ?
```

Should be coded as:

```
SELECT EMPNO, LASTNAME               SELECT EMPN, LASTNAME
FROM EMPLOYEE                                   FROM EMPLOYEE
WHERE SALARY > 50000.00 / 1.1           WHERE HIREDATE > DATE(?)
                                          + 7 DAYS
```

V9: Can now create indexes on SQL expressions

LUW: Optimizer much more powerful in predicate rewrites
Top 25+ Tuning Tips #1

No DB2 LUW Optimizer Rewrite on this one

Predicate | Table Access | Join Operation

Database server transformed query for access plan

```
SELECT Q1.EMPNO AS "EMPNO", Q1.LASTNAME AS "LASTNAME", Q1.SALARY AS "SALARY",
       Q1.HIREDATE AS "HIREDATE"
FROM THEMIS82.EMP AS Q1
WHERE (+50000.00 < (Q1.SALARY * +1.1))
```
3). Stay away from ‘Distinct’ if at all possible.

If duplicates are to be eliminated from the result set, try:

- ‘Group By’ which looks to take advantage of any associated indexes to eliminate a sort for uniqueness.
- Rewriting the query using an ‘In’ or ‘Exists’ subquery. This will work if the table causing the duplicates (due to a one to many relationship) does not have data being returned as part of the result set.

Z/OS: Prior to V9, the keyword ‘Distinct’ most always involved a sort, now DB2 may take advantage of any index (unique or non unique) to avoid sorts.
3). Distincts can often be rewritten.

```
SELECT DISTINCT E.EMPNO, E.LASTNAME
FROM EMP E, EMPPROJACT EPA
WHERE E.EMNO = EPA.EMPNO
GROUP BY E.EMPNO, E.LASTNAME
```

```
SELECT E.EMPNO, E.LASTNAME
FROM EMP E
WHERE E.EMNPO IN
    (SELECT EPA.EMPNO
     FROM EMPPROJACT EPA)
```

```
SELECT E.EMPNO, E.LASTNAME
FROM EMP E
WHERE EXISTS
    (SELECT 1
     FROM EMPPROJACT EPA
     WHERE EPA.EMPNO = E.EMPNO)
```
4). Minimize the SQL requests to DB2.

This is huge in performance tuning of programs, especially batch programs because they tend to process more data. Every time an SQL call is sent to the database manager, there is overhead in sending the SQL statement to DB2, going from one address space in the operating system to the DB2 address space for SQL execution.

In general developers need to minimize:
- The number of time cursors are Opened/Closed
- The number of random SQL requests (noted as synchronized reads in DB2 monitors).

z V8: Multi Row Fetch, Update, and Inserting. Recursive SQL. Select from Insert.

z V9: ‘Upsert’ processing. Fetch First / Order By within subqueries.
5). z/OS Give prominence to Stage 1 over Stage 2 Predicates.

Always try to code predicates as Stage 1 and indexable. In general, Stage 2 predicates do not perform as well and consume extra CPU. See the IBM SQL Reference Guide to determine what predicates are Stage 1 vs. Stage 2 and make sure to go to the correct Version of DB2 when checking.

Recommendation: Use Visual Explain (V8), OSC (V9), Data Studio (V9).

IBM DB2 Manuals: Search on ==> Summary of Predicate Processing

5). LUW Give prominence to SARG versus RESIDUAL Predicates.

Most predicates in LUW are Sargeable at either the index level or the data file level. LUW has very few Residual predicates. Unlike z/Os which many more residual and/or Stage 2 predicates.

Recommendation: Use Data Studio to check predicates.
Stage 1 versus Stage 2 Predicates

- Stage 1 (DB2 Data Manager) is responsible for translating the data stored on pages into a result set of rows and columns. Predicates that are written in a fairly straightforward way can usually be evaluated by the Data Manager with relatively little expense.

- Stage 2 (Relational Data Services) handle more complex predicates, data transformations, and computations. These Stage 2 predicates are much more expensive for DB2 to resolve than Stage 1 due to additional processing and additional code path. Additionally, RDS cannot make effective use of indexes.
Use the Visual Explain in Optimization Service Center or Data Studio to see any stage 2 predicates. Note the filter factor information also.
## Top 25+ Tuning Tips #5 Cont...

5). **DB2 z/OS. Give prominence to Stage 1 over Stage 2 Predicates cont...**

<table>
<thead>
<tr>
<th>Predicate Type</th>
<th>Indexable?</th>
<th>Stage 1?</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>COL = value</td>
<td>Y</td>
<td>Y</td>
<td>16</td>
</tr>
<tr>
<td>COL = noncol expr</td>
<td>Y</td>
<td>Y</td>
<td>9,11,12,15</td>
</tr>
<tr>
<td>COL IS NULL</td>
<td>Y</td>
<td>Y</td>
<td>20,21</td>
</tr>
<tr>
<td>COL op value</td>
<td>Y</td>
<td>Y</td>
<td>13</td>
</tr>
<tr>
<td>COL op noncol expr</td>
<td>Y</td>
<td>Y</td>
<td>9,11,12,13</td>
</tr>
<tr>
<td>COL BETWEEN value1 AND value2</td>
<td>Y</td>
<td>Y</td>
<td>13</td>
</tr>
<tr>
<td>COL BETWEEN noncol expr1 AND noncol expr</td>
<td>Y</td>
<td>Y</td>
<td>9,11,12,13,15,23</td>
</tr>
<tr>
<td>value BETWEEN COL1 AND COL2</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>COL BETWEEN expression1 AND expression2</td>
<td>Y</td>
<td>Y</td>
<td>6,7,11,12,13,14</td>
</tr>
<tr>
<td>COL BETWEEN COL1 AND COL2</td>
<td>N</td>
<td>N</td>
<td>10</td>
</tr>
<tr>
<td>COL LIKE 'pattern'</td>
<td>Y</td>
<td>Y</td>
<td>5</td>
</tr>
<tr>
<td>COL IN (list)</td>
<td>Y</td>
<td>Y</td>
<td>17,18</td>
</tr>
<tr>
<td>COL &lt;&gt; value</td>
<td>N</td>
<td>Y</td>
<td>8,11</td>
</tr>
<tr>
<td>COL &lt;&gt; noncol expr</td>
<td>N</td>
<td>Y</td>
<td>8,11</td>
</tr>
<tr>
<td>COL IS NOT NULL</td>
<td>Y</td>
<td>Y</td>
<td>21</td>
</tr>
<tr>
<td>COL NOT BETWEEN value1 AND value2</td>
<td>N</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>
Top 25+ Tuning Tips #5 Cont…

Use the Visual Explain for guidance to Stage 1, Stage2, and Filter Factor information.
Top 25+ Tuning Tips #5

Watch out for compound predicates connected by ‘Or’

When predicates are connected by ‘OR’ (Non Boolean Term), the resulting compound predicate, the resulting compound predicate will be evaluated at the higher stage of the individual predicates. For example:

```sql
SELECT EMPNO, LASTNAME
FROM EMPLOYEE
WHERE LASTNAME = ‘ANDREWS’ OR YEAR(HIREDATE) = 1990
```
Top 25+ Tuning Tips #5

Watch out for compound predicates connected by ‘Or’
Top 25+ Tuning Tips #5

Same query connected by ‘And’

When predicates are connected by ‘OR’ (Non Boolean Term), the resulting compound predicate, the resulting compound predicate will be evaluated at the higher stage of the individual predicates. For example:

```sql
SELECT EMPNO, LASTNAME
FROM EMPLOYEE
WHERE LASTNAME = ‘ANDREWS’ AND YEAR(HIREDATE) = 1990
```
Top 25+ Tuning Tips #5

Overview of Diagram
Description of Selected Node
Displays information about the node that is highlighted in the diagram.

Properties
SQL Results
Access Plan Diagram

(1) QUERY
(2) QB1
(3) FETCH
(4) SORTRID
(5) IXSCAN
(6) EMP03
(7) EMP

Attributes
Name | Value
--- | ---
Input RIDs | 51834
Index Leaf Pages | 549
Matching Predicates | Filter Factor
THEMIS8.EMP.LASTNAME='Andrews' | 0.001

Description of Selected Attribute
Top 25+ Tuning Tips #5

LUW PREDICATES

SARGEABLE: Index or Data. Stage 1
RANGE DELIMITING: Matching Index predicates. Provide start and stop keys.
RESIDUAL: Stage 2
INDEX: Range-delimiting start and stop keys
DB2 LUW Sargeable Predicates

Range-delimiting predicates: Start and Stop index processing.

Index-SARGable predicates: Evaluated at the index level

Example: INDEX EMPX5: LASTNAME, FIRSTNAME, MIDINIT
WHERE clause: where lastname = :hv1 and midinit = :hv2

The first predicate (lastname = :hv1) is a range-delimiting predicate
second predicate (midinit = :hv2) is an index-SARGable predicate
6). Try to avoid filtering logic within application code.

It is always best to have all the filtering logic that is needed written as predicates in a SQL statement. Do not leave some predicates out and have the database manager bring in extra rows and then eliminate / bypass some of the rows through program logic checks. (Some people call this Stage 3 processing).

Deviate only when performance is an issue and all other efforts have not provided significant enough improvement in performance.

DB2 V8 introduced support for the manipulation of multiple rows on fetches, updates, and insert processing. Prior versions of DB2 would only allow for a program to process one row at a time during cursor processing. Now having the ability to fetch, update, or insert more than 1 row at a time reduces network traffic and other related costs associated with each call to DB2.

The recommendation is to start with 100 row fetches, inserts, or updates, and then test other numbers. It has been proven many times that this process reduces runtime on average of 35%. Consult the IBM DB2 manuals for further detail and coding examples.
8). Take advantage of Scalar Fullselects within the Select clause whenever possible. New as of V8:

Many times the output needed from SQL development requires a combination of detail and aggregate data together. There are typically a number of ways to code this with SQL, but with the Scalar Fullselect now part of DB2 V8, there is now another option that is very efficient as long as indexes are being used.

For Example: Individual Employee Report with Aggregate Department Averages

```sql
SELECT E1.EMPNO, E1.LASTNAME,
   E1.WORKDEPT, E1.SALARY, (SELECT AVG(E2.SALARY)
       FROM EMPLOYEE  E2
       WHERE E2.WORKDEPT = E1.WORKDEPT)
       AS DEPT_AVG_SAL
FROM EMPLOYEE E1
ORDER BY E1.WORKDEPT, E1.SALARY
```
Top 25+ Tuning Tips #8

8). Previous way #1.

For Example: Individual Employee Report with Aggregate Department Averages

WITH X AS
  (SELECT E2.WORKDEPT, DEC(ROUND(AVG(E2.SALARY),2),9,2) AS DEPT_AVG_SAL
   FROM EMPLOYEE E2
   GROUP BY E2.WORKDEPT)

SELECT E1.EMPNO, E1.LASTNAME, E1.WORKDEPT,
       E1.SALARY, X.DEPT_AVG_SAL
FROM EMP E1 INNER JOIN X ON E1.DEPTNO = X.DEPTNO
ORDER BY E1.WORKDEPT, E1.SALARY
9). Watch out for tablespace scans.

What do you do? If you as a developer see that a tablespace scan is occurring in your SQL execution, then go through the following checklist to help figure out why?

- The predicate(s) may be poorly coded in a non-indexable way.
- The predicates in the query do not match any available indexes on the table. Should have indexes even on small tables.
- The table could be small, and DB2 decides a tablespace scan may be faster than index processing.
- The catalog statistics say the table is small, or maybe there are no statistics on the table.
- The predicates are such that DB2 thinks the query is going to retrieve a large enough amount of data that would require a tablespace scan.
- The predicates are such that DB2 picks a non-clustered index, and the number of pages to retrieve is high enough based on total number of pages in the table to require a tablespace scan.
- The tablespace file or index files could physically be out of shape and need a REORG.
10). Only code the columns needed in the Select portion of the SQL statement.

Having extra columns can have an affect on:

- The optimizer choosing ‘Index Only’
- Expensiveness of any sorts
- Optimizer’s choice of join methods
11). Watch out for any data sorts.

Sorts can be expensive. At times an SQL query may execute multiple sorts in order to get the result set back as needed. Take a look at the DB2 explain tool to see if any sorting is taking place, then take a look at the SQL statement and determine if anything can be done to eliminate sorts. Data sorts are caused by:

- ‘Order By’
- ‘Group By’
- ‘Distinct’
- ‘Union’ versus ‘Union All’
- Join processing. Pay attention to the clustering order of data in tables.
- In List subqueries

** Sorts are as expensive as their size.
Top 25+ Tuning Tips #11
Top 25+ Tuning Tips #12

12). Try rewriting an ‘In’ subquery as an ‘Exists’ subquery or vice versa.

Each of these will produce the same results, but operate very differently. Typically one will perform better than the other depending on data distributions. For Example:

<table>
<thead>
<tr>
<th>Non Correlated Subquery</th>
<th>Can also be coded as:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT E.EMPNO, E.LASTNAME FROM EMPLOYEE E WHERE E.EMPNO IN (SELECT D.MGRNO FROM DEPARTMENT D WHERE D.DEPTNO LIKE 'D%')</td>
<td>SELECT E.EMPNO, E.LASTNAME FROM EMPLOYEE E WHERE EXISTS (SELECT 1 FROM DEPARTMENT D WHERE D.MGRNO = E.EMPNO AND D.DEPTNO LIKE 'D%')</td>
</tr>
</tbody>
</table>

z V9: Global Query Optimization. Optimizer now tries to determine how an access path of one query block may affect the others. This can be seen at times by DB2 rewriting an ‘Exists’ subquery into a join, or an ‘In’ subquery into an ‘Exists’ subquery. z V9 optimization calls this ‘Correlating’ and ‘De-correlating’.
Top 25+ Tuning Tips #12

12). ‘In’ versus ‘Exists’ Query Transformations. LUW transformed into a Join.

Original

```
SELECT THEMIS81.PROJ.PROJNO
, THEMIS81.PROJ.PROJNAME
FROM THEMIS81.PROJ
WHERE THEMIS81.PROJ.PROJNO IN (  
( SELECT THEMIS81.EMPPROJACT.PROJNO 
FROM THEMIS81.EMPPROJACT 
WHERE THEMIS81.EMPPROJACT.EMENDATE > '01/01/1983' 
)  
)
```

Transformed

```
SELECT DISTINCT Q2.PROJNO AS "PROJNO", Q2.PROJNAME AS "PROJNAME"
FROM THEMIS81.EMPPROJACT AS Q1, THEMIS81.PROJ AS Q2
WHERE (01/01/1983 < Q1.EMENDATE) AND (Q2.PROJNO = Q1.PROJNO)
```
12). ‘In’ versus ‘Exists’ Query Transformations. z/OS transformed into a Correlated Subquery.

Original

```
SELECT THEMIS81.PROJ.PROJNO
 , THEMIS81.PROJ.PROJNAME
 FROM THEMIS81.PROJ
 WHERE THEMIS81.PROJ.PROJNO IN ( 
   ( SELECT THEMIS81.EMPPROJACT.PROJNO 
     FROM THEMIS81.EMPPROJACT 
     WHERE THEMIS81.EMPPROJACT.EMENDATE > '01/01/1963' 
   ) 
 )
```

Transformed

```
SELECT THEMIS81.PROJ.PROJNO
 , THEMIS81.PROJ.PROJNAME
 FROM THEMIS81.PROJ
 WHERE THEMIS81.PROJ.PROJNO IN ( 
   SELECT THEMIS81.EMPPROJACT.PROJNO 
   FROM THEMIS81.EMPPROJACT
   WHERE ( THEMIS81.EMPPROJACT.EMENDATE > '1983-01-01' 
     AND THEMIS81.PROJ.PROJNO = THEMIS81.EMPPROJACT.PROJNO 
   )
 )
```
13). Make sure the data distribution statistics are current in the tables being processed.

This is done by executing the Runstats utility on each specific table and associated indexes. This utility loads up the system catalog tables with data distribution information that the optimizer looks for when selecting access paths. Some of the information that the Runstats utility can provide is:

- The size of the tables (# of rows)
- The cardinalities of columns
- The percentage of rows (frequency) for those uneven distribution of column values
- The physical characteristics of the data and index files
- Information by partition

Pay attention to the ‘Statstime’ column in the catalog tables as it will state when the last time Runstats has been executed on each table.

V8 – Volatile Tables. DB2 considers using Index access no matter the statistics.
Top 25+ Tuning Tips #13

13). Make sure the data distribution statistics are current in the tables being processed.
14. Basic runstats needed.

All tables in all environments should have the following statistics run:

- Cardinality statistics on all columns in all tables
- z Frequency Value statistics on any column(s) with uneven distribution
- LUW ‘With Distribution’
- Group statistics (distinct values over a group of columns) for any set of columns that are correlated

**V9:** Quantile Statistics (further breakdown of statistics). Helps with range predicates, between predicates, and the like predicate. Especially where there exists ‘Hot Spots’ of data distribution.
14). Basic runstats needed.
Top 25+ Tuning Tips #15

15. At times, use hard coding versus a host variable. Or REOPT.

For example: There exists a table with 1 million rows. In this table exists an index on the column Status_Code. After a typical Runstats utility is executed against this table, the optimizer will know that there are 3 different values for the status code. After a special Runstats that specifies frequency value statistics for that column, DB2 will know the following data distributions:

Status Code value ‘A’ contains 50% of the data
Status Code value ‘B’ contains 45% of the data
Status Code value ‘C’ contains 05% of the data.

A program contains the following SQL. The value is always ‘C’ that comes into the program:

```
SELECT COL1, COL2, COL3
FROM TABLE
WHERE STATUS_CD = :HOST-VARIABLE
```
16. Index Correlated Subqueries:

There are a couple of things to pay attention to when an SQL statement is processing using a Correlated subquery. Correlated subqueries can get executed many times in order to fulfill the SQL request. With this in mind, the subquery must be processed using an index to alleviate multiple tablespace scans. If the correlated subquery is getting executed hundreds of thousands or millions of times, then it is best to make sure the subquery gets executed using an index with Indexonly = ‘Yes’. This may require the altering of an already existing index.

For example:

```
SELECT E.EMPNO, E.LASTNAME
FROM EMPLOYEE E
WHERE EXISTS
  (SELECT 1
   FROM DEPARTMENT D
   WHERE D.MGRNO = E.EMPNO
   AND D.DEPTNO LIKE 'D%')
```
Top 25+ Tuning Tips #17

17. Avoid Discrepancies with Non Column Expressions.

Often times queries with expression can be non-indexable and/or Stage 2. When coding predicates containing expression, it’s best to:

1) Execute the expression prior to the SQL and put the answer into a variable that matches the columns definition.
2) Apply the appropriate scalar function to match the column definition.

For Example:

Where EDLEVEL = 123.45 * 12
should be coded as

Where EDLEVEL = SMALLINT(Round(123.45*12,0))

For Example:

Where LASTNAME like ‘000’ concat ‘%’
should be coded as

Where LASTNAME like ‘000%’
18. Subquery predicates involving Min and Max can often be a Stage 2 predicate.

When coding a predicate that involves a subquery selecting a Min or Max value, make sure you know if the column in the predicate is defined as nullable or not. This can make a difference between the predicate being a Stage 1 versus a Stage 2 predicate. *This is specific to the predicate operator being anything other than just the ‘=’ sign.*

For Example, If E.HIREDATE column is defined as ‘Not Null’ the SQL should be coded as:

```sql
SELECT E.EMPNO, E.LASTNAME
FROM EMPLOYEE E
WHERE E.HIREDATE <=
    (SELECT COALESCE(MAX(T2.HIREDATE), '9999-12-31')
     FROM TABLE2 T2
     WHERE ..... 
     AND ..... )
```
Top 25+ Tuning Tips #19

19. Make sure of the clustering order of data in your tablespaces.

Tables should be physically clustered in the order that they are typically processed by queries processing the most data. This ensures the least amount of ‘Getpages’ when processing.

1) Indexes specify the physical order.
2) Cluster = ‘YES’ or first index created

Long running queries with ‘List Prefetch’ and ‘Sorts’ in many join processes are good indicators that maybe a table is not in the correct physical order.
DB2 z V8 introduced the ability to now select what was just inserted with the same statement saving multiple calls to DB2. This again we call ‘Relational’ programming instead of ‘Procedural’ programming. The statement can retrieve the following information.

- Identity columns or sequence values that get automatically assigned by DB2
- User-defined defaults and expressions that are not known to the developer
- Columns modified by triggers that can vary from insert to insert depending on values
- ROWIDs, CURRENT TIMESTAMP that are assigned automatically

For example:

```sql
SELECT C1, C2, C5, C6, C8, C11, C12
FROM FINAL TABLE
 (INSERT (C2, C3, C4, C5, C6, C8, C10)
  VALUES  ('ABC', 123.12, 'A', 'DEF',
           50000.00, 'GHI', '2008-01-01')
)```
Top 25+ Tuning Tips #21

21. Checking for Non Existence:

When coding logic to determine what rows in a table do not exist in another table, there are a couple of common approaches. One approach is to code outer join logic and then check ‘Where D.MGRNO IS NULL’ from the other table, or coding ‘Not Exists’ logic. The following 2 examples both bring back employees that are not managers on the department table, yet the 2nd one is most often the more efficient. The DB2 Visual Explain tool shows by each predicate when filtering is accomplished.

Example 1:

```sql
SELECT E.EMPNO, E.LASTNAME
FROM EMPLOYEE E
  LEFT JOIN DEPARTMENT D
  ON D.MGRNO = E.EMPNO
WHERE D.MGRNO IS NULL
```

Example 2:

```sql
SELECT E.EMPNO, E.LASTNAME
FROM EMPLOYEE E
WHERE NOT EXISTS
  (SELECT 1
   FROM DEPARTMENT D
   WHERE D.MGRNO = E.EMPNO)
```
22. Stay away from Selecting a row from a table to help decide whether the logic in code should then execute an Update or an Insert.

This requires an extra call to DB2. V9: Use the new ‘Upsert’ SQL Merge statement

Example:

MERGE INTO EMPLOYEE E
    USING (VALUES ('000999', 'TONY', 'ANDREWS', 'A00') )
    AS NEWITEM (EMPNO, FIRSTNAME, LASTNAME, DEPARTMENT)
ON E.EMPNO = NEWITEM.EMPNO
WHEN MATCHED  THEN
    UPDATE SET FIRSTNAME = NEWITEM.FIRSTNAME,
        LASTNAME  = NEWITEM.LASTNAME
WHEN NOT MATCHED  THEN
    INSERT (EMPNO, FIRSTNAME, LASTNAME, DEPARTMENT)
VALUES (NEWITEM.EMPNO, NEWITEM.FIRSTNAME,
        NEWITEM.LASTNAME,  NEWITEM.DEPARTMENT)
Top 25+ Tuning Tips #23


What to do when the cursor is ‘Read Only’:

- Try defining a ‘Dynamic’ scrollable cursor. This at times allows for an ‘Order By’ in the cursor definition, while allowing a ‘Delete Where Current of Cursor’ statement within the processing.

- Fetch the ROWID (RID as of V9) for each row being processed in the ‘Read Only’ cursor, and execute all update or delete statements using the ROWID/RID value in place of the key fields for better performance.

What to do when the cursor is ‘Read Only’:

- Try defining a Scrollable cursor. This will materialize the cursor’s data into a workfile but then allows for cursor updating.
  DECLARE SENSITIVE, FETCH SENSITIVE

- Take advantage of the V8 ‘NOFOR’ precompile option. In static SQL, eliminates the need for the FOR UPDATE or FOR UPDATE OF clause in DECLARE CURSOR statements. When you use NOFOR, your program can make positioned updates to any columns that the program has DB2 authority to update.
24. Use Left Outer Joins over Right Outer Joins:

When coding outer join logic, it does not matter whether the developer codes a ‘Left Outer Join’ or a ‘Right Outer Join’ in order to get the logic correct, as long as they have the starting ‘Driver’ table coded correctly. There is no difference between a Left and Right outer join other than where the starting ‘Driver’ is coded. This is not really a tuning tip, but rather a tip to help all developers understand that left outer joins are more readable.

Developers in DB2 should only code ‘Left Outer Joins’. It is more straightforward because the starting ‘Driver’ table is always coded first, and all subsequent tables being joined to have ‘Left Outer Join’ coded beside them, making it more understandable and readable.
25. Predicate rewrites:

1) WHERE value1 BETWEEN COLA and COLB

   should be rewritten as

   WHERE value1 >= COLA and value1 <= COLB

2) WHERE YEAR(HIREDATE) = 2011

   should be rewritten as

   WHERE HIREDATE BETWEEN '2011-01-01' and '2011-12-31'
25. Predicate rewrites:

3) WHERE COLA NOT BETWEEN value 1 and value2

should be rewritten as

WHERE COLA < value1 or COLA > value2

4) WHERE HIREDATE + 1 MONTH > value1

should be rewritten as

WHERE HIREDATE > value1 – 1 MONTH
25. Predicate rewrites:

5) WHERE COLA NOT IN (value1, value2, value3)

should be rewritten as

** Try to identify a list of values that COLA can be equal to
** ‘Not In’ with a list is STAGE 1 but Non-Indexable

6) WHERE DECIMAL(a.COLX) = b.COLX

WHERE  a.COLA = INTEGER(b.COLA)

b.COLX is Decimal
a.CLA is Integer

** As of V7, numeric datatypes can be cast in these comparisons allowing for Stage 1 processing, however:

** If both columns are indexed, cast on the column with the larger filter factor.
25. Predicate rewrites:

7) WHERE SUBSTR(COLA,1,1) = 'A'

should be rewritten as

WHERE COLA LIKE 'A%'

WHERE COLA LIKE 'A%'
Top 25+ Tuning Tips #26
Predicate Evaluation Order

SELECT *
FROM EMP
WHERE SEX = 'F'
AND JOB = 'FIELDREP';

60% of the rows qualify

SELECT *
FROM EMP
WHERE JOB = 'FIELDREP'
AND SEX = 'F';

05% of the rows qualify

SELECT *
FROM EMP
WHERE SEX = 'F'
AND JOB = 'FIELDREP';

05% of the rows qualify

60% of the rows qualify
27. Do not Select a column in order to have it part of an ‘ORDER BY’.

Do not ‘ORDER BY’ columns you do not need.

SELECT EMPNO, LASTAME, SALARY, DEPTNO
FROM EMP
ORDER BY EMPNO, LASTNAME, SALARY

In this example, we only need to ‘ORDER BY EMPNO’ because it is a unique column.

SELECT EMPNO, LASTAME, SALARY, DEPTNO
FROM EMP
WHERE SALARY BETWEEN 50000.00 and 100000.00
FETCH FIRST 25 ROWS ONLY

If the optimizer knows exactly what you intend to retrieve it can make decisions based on that fact, and often times optimization will be different based on this known fact than if it was not coded, and the program just quit processing after the first 25.
29. Take advantage and promote ‘Index Only’ processing whenever possible.

```
SELECT LASTNAME, MIDINIT, FIRSTNME
    FROM EMP
WHERE LASTNAME LIKE 'A%'
ORDER BY LASTNAME, FIRSTNME

XEMP3 INDEX = (LASTNAME, FIRSTNME, MIDINIT)
```

No need for DB2 to leave the index file because everything it needs to process the result set is contained within the index file.
30. Multiple subqueries. Code most restrictive to least restrictive.

1) When there are multiple subqueries of the same type, always code in the order of most restrictive to least restrictive.

2) When a subquery contain both a Non correlated and Correlated subquery, DB2 will execute the Non Correlated first. Tune these by ensuring the Non Correlated is the most restrictive, or make them both the same type and order them accordingly.
31. Predicate Transitive Closure.

IF COLA = COLB and COLA = 123, then COLB MUST = 123

SELECT E.EMPNO, E.LASTNAME, E.DEPTNO, E.LOCATION
FROM EMP E, DEPT D
WHERE E.LOC = D.LOC
    AND E.LOC = 'NY'

DB2 OPTIMIZER ADDS

    AND D.LOC = 'NY'

BUT ... CURRENTLY DB2 Z DOES NOT APPLY TRANSITIVE CLOSURE FOR LIKE, IN, AND SUBQUERIES. SO DEVELOPERS SHOULD CODE THEIR OWN TRANSITIVE CLOSURE TO PROVIDE THE OPTIMIZER MORE INFORMATION.
Top 25+ Tuning Tips #31

31. DB2 LUW Predicate Transitive Closure.

**Input Query**

```
SELECT E.EMPNO, E.LASTNAME
FROM (THEMIS82.EMP AS E
     LEFT OUTER JOIN THEMIS82.DEPT AS D
     ON D.MGRNO = E.EMPNO)
WHERE E.EMPNO IN ('000010', '000040')
```

**Database server transformed query for access plan**

```
SELECT Q6.EMPNO AS "EMPNO", Q6.LASTNAME AS "LASTNAME"
FROM
(SELECT Q5.LASTNAME, Q5.EMPNO
  FROM THEMIS82.DEPT AS Q1 RIGHT OUTER JOIN
  (SELECT Q4.EMPNO, Q4.LASTNAME
   FROM THEMIS82.EMP AS Q4
   WHERE Q4.EMPNO IN ('000010', '000040')) AS Q5 ON (Q1.MGRNO = Q5.EMPNO)
 AND Q1.MGRNO IN ('000010', '000040')) AS Q6
```
Ten Steps to Tuning a Query

1) Check every predicate. Are they indexable and Stage 1?
2) Is there a ‘Distinct’? Is it needed? Can it be rewritten? Try ‘Group BY’.
3) Are there subqueries? Rewrite ‘In’ as ‘Exists’ and vice versa.
4) Check DB2 statistics on every table and every column involved.
5) Check the number of times every SQL statement is getting executed. Can the logic be changed to cut down the number of times requests are sent?
6) Check the DB2 Explain. Are there any table scans? Any sorts?
7) Check the DB2 VE. Are there any predicates with crazy Filter Factors?
8) Check the DB2 Explain. If there is a join, what is the order of tables chosen?
9) Are there any correlated subqueries? Can they be Index-Only?
10) Are there any columns in predicates with uneven distribution statistics? Should the value be hard coded?
11) Are there any range predicates. Could histogram statistics help?
12) Can any of the predicates be rewritten differently? Can the query be rewritten differently?
New DB2 V8 SQL Features

Following are some of the new SQL features in DB2 V8:

1) More Stage 1 predicates
2) Multi Row Fetch, Update, and Insert
3) Multiple Distincts
4) Expressions in the ‘Group By’
5) Common Table Expression
6) Dynamic Scrollable Cursors
7) Sequences versus Identity Columns
8) Materialized Query Tables (MQTs)
9) Recursive SQL
10) More efficient use of indexes. Forward and Backward scans
11) New XML functions and datatypes
12) New ‘Get Diagnostics’ for warning and error information
13) Select from an Insert statement
14) Scalar Fullselect within a ‘Select’, ‘Case’, Update, etc.
New DB2 V9 SQL Features

Following are some of the new SQL features in DB2 V9:

1) Set operations ‘Intersect’ and ‘Except’
2) Merge statement for ‘Upsert’ processing. Insert or Update
3) OLAP features for Ranking and Numbering of data
4) Native SQL Stored Procedures
5) ‘Instead of’ Triggers
6) New support and SQL for XML data
7) Optimization Service Center (OSC)
8) Distinct sort avoidance with non unique indexes
9) Indexing on Expressions
10) Statistics on Views
11) Skipped locked data
12) Truncate statement
New DB2 V9 SQL Features

Following are some of the new SQL features in DB2 V9:

13) Array host variables now supported for stored procedures
14) Timestamp auto update for inserts and Updates
15) Optimistic locking
16) New DECIMAL datatype
17) Select from Update or Delete getting old or new values
18) Fetch First, Order BY within subqueries
19) REOPT AUTO (Dynamic SQL)
20) Data Studio for Native Stored Procedures
Experience Shows

That a little knowledge goes a long way in IT development shops

“When you know something, say what you know. When you don't know something, say that you don't know. That is knowledge.”
Experience Shows

That standards, guidelines, and walkthroughs saves CPU costs and incident reporting in IT shops.

“There is always time for a 10 minute walkthrough”
Training and Consulting. Check Out www.themisinc.com

- On-site and Public
- Instructor-led
- Hands-on
- Customization
- Experience
- Over 25 DB2 courses
- Over 400 IT courses

US    1-800-756-3000
Intl.  1-908-233-8900
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.db2sqltuningtips.com
Thank You for allowing us at Themis to share some of our experience and knowledge!

- We hoped that you learned something new today
- We hope that you are a little more inspired to code with performance in mind

“I have noticed 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.”