Top 25+ SQL Tuning Tips for Developers

Tony Andrews

Themis Inc.

Session code: E04

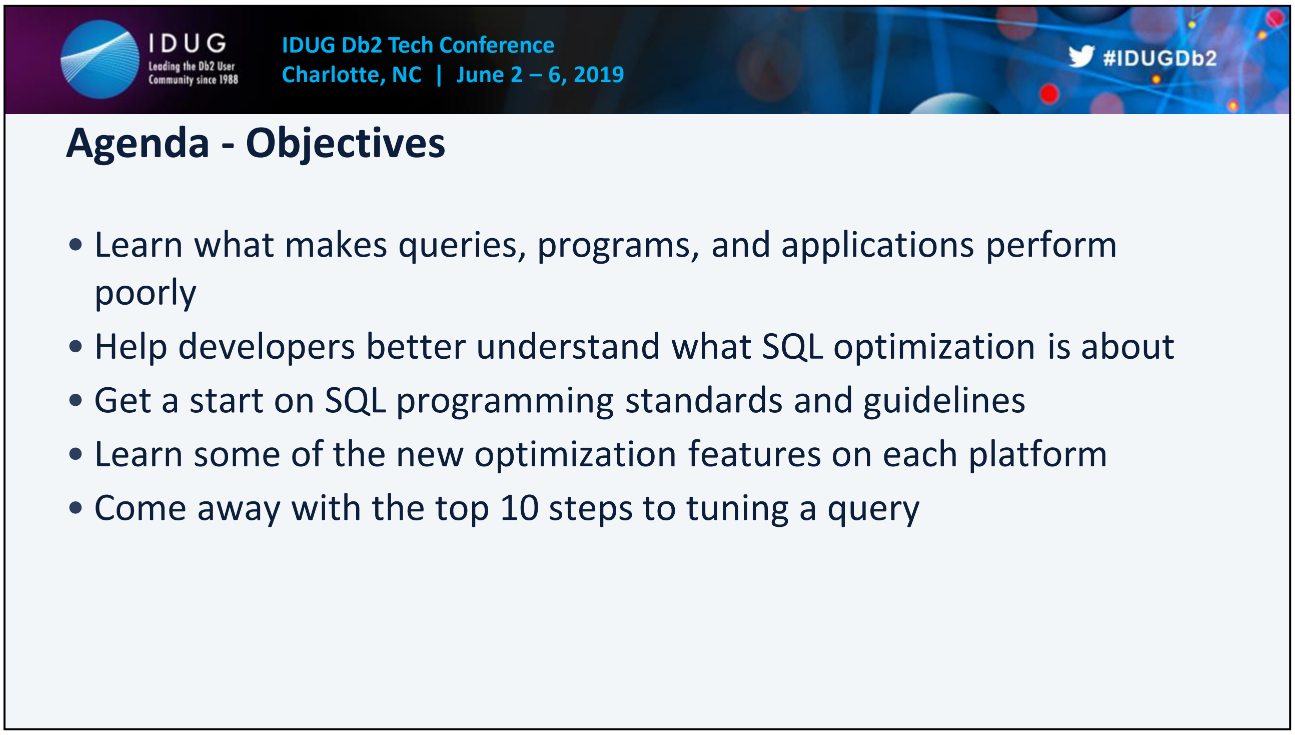
Monday 4:30

Cross Platform

IDUG

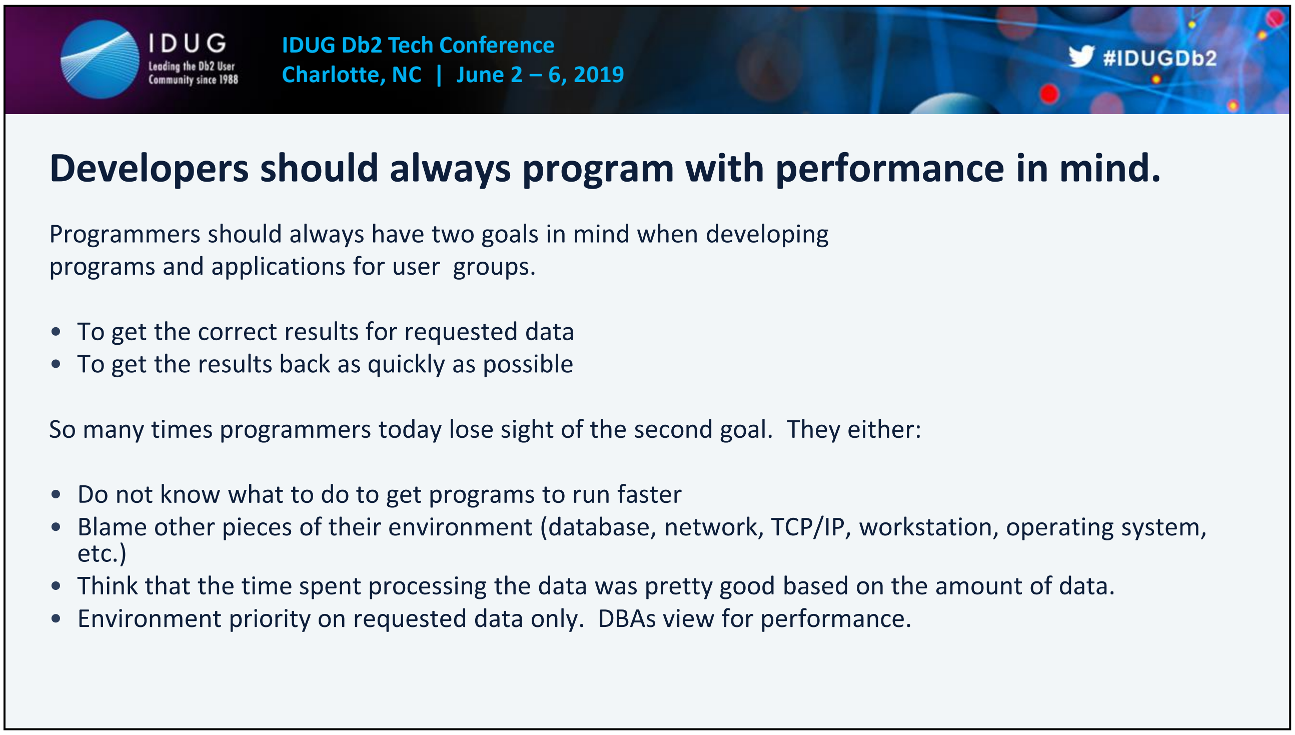
Leading the Db2 User Community since 1988

Most relational tuning experts agree that the majority of performance problems with applications that access a relational database are caused by poorly coded programs or improperly coded SQL. Industry experts agree that poorly performing SQL is responsible for many response-time issues. SQL developers should be informed of the many performance issues associated with the SQL language and the way they design their programs. It would be especially helpful if more developers were educated in how to read and analyze Db2 Explain output. This presentation was a hit at many RUGs years ago, and I thought it was time to reinvent it due to the many SQL and optimization changes in the past years. This is good for both z/OS and LUW platform developers. Attendees will leave with many tips, standards, and guidelines for good SQL programming 'Best Practices'.



Agenda - Objectives

- Learn what makes queries, programs, and applications perform poorly
- Help developers better understand what SQL optimization is about
- Get a start on SQL programming standards and guidelines
- Learn some of the new optimization features on each platform
- Come away with the top 10 steps to tuning a query



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:

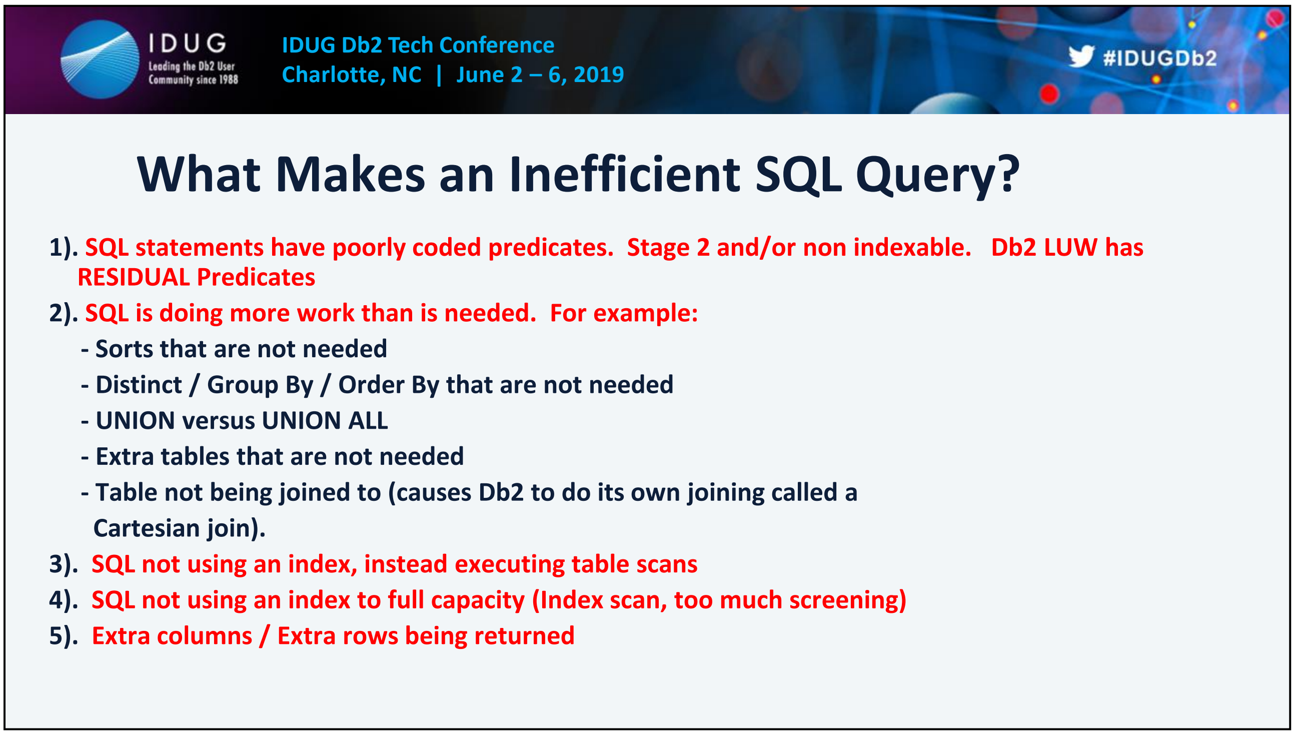
- Do not know what to do to get programs to run faster
- Blame other pieces of their environment (database, network, TCP/IP, workstation, operating system, etc.)
- Think that the time spent processing the data was pretty good based on the amount of data.
- Environment priority on requested data only. DBAs view for performance.

There has been great strides in optimizer code, hardware, memory …

But we still need some coding rules for the best efficiencies!!

we still need to know our data more than ever!

we still need to be strong in SQL programming!


IDUG Leading the Db2 User Community since 1988

IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

What Makes an Inefficient SQL Query?

1). SQL statements have poorly coded predicates. Stage 2 and/or non indexable. Db2 LUW has RESIDUAL Predicates

2). SQL is doing more work than is needed. For example:

- Sorts that are not needed
- Distinct / Group By / Order By that are not needed
- UNION versus UNION ALL
- Extra tables that are not needed
- Table not being joined to (causes Db2 to do its own joining called a Cartesian join).

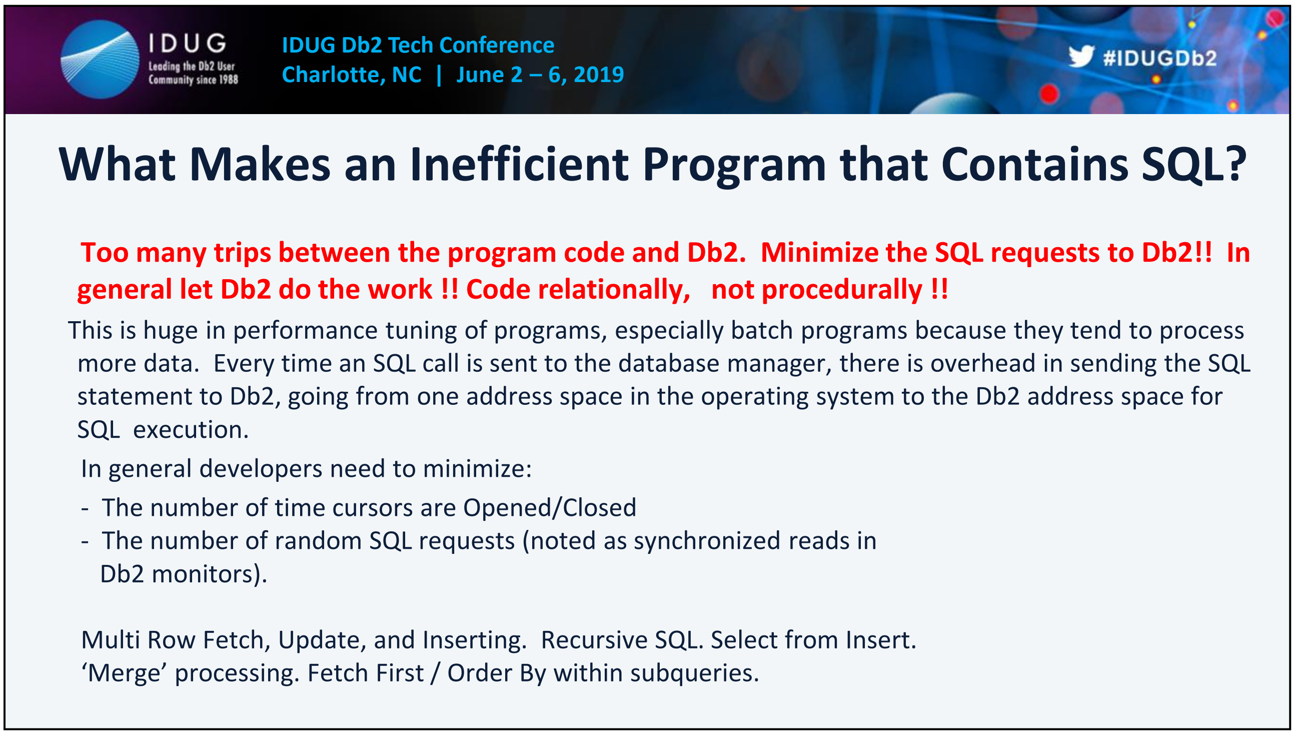
3). SQL not using an index, instead executing table scans

4). SQL not using an index to full capacity (Index scan, too much screening)

5). Extra columns / Extra rows being returned

Developers should review their own SQL code and make sure it is:

- Not doing more work than needed
- Not bring back more data (columns and/or rows) needed
- Not doing unneeded sorts



IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

What Makes an Inefficient Program that Contains SQL?

Too many trips between the program code and Db2. Minimize the SQL requests to Db2!! In general let Db2 do the work !! Code relationally, not procedurally !!

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:

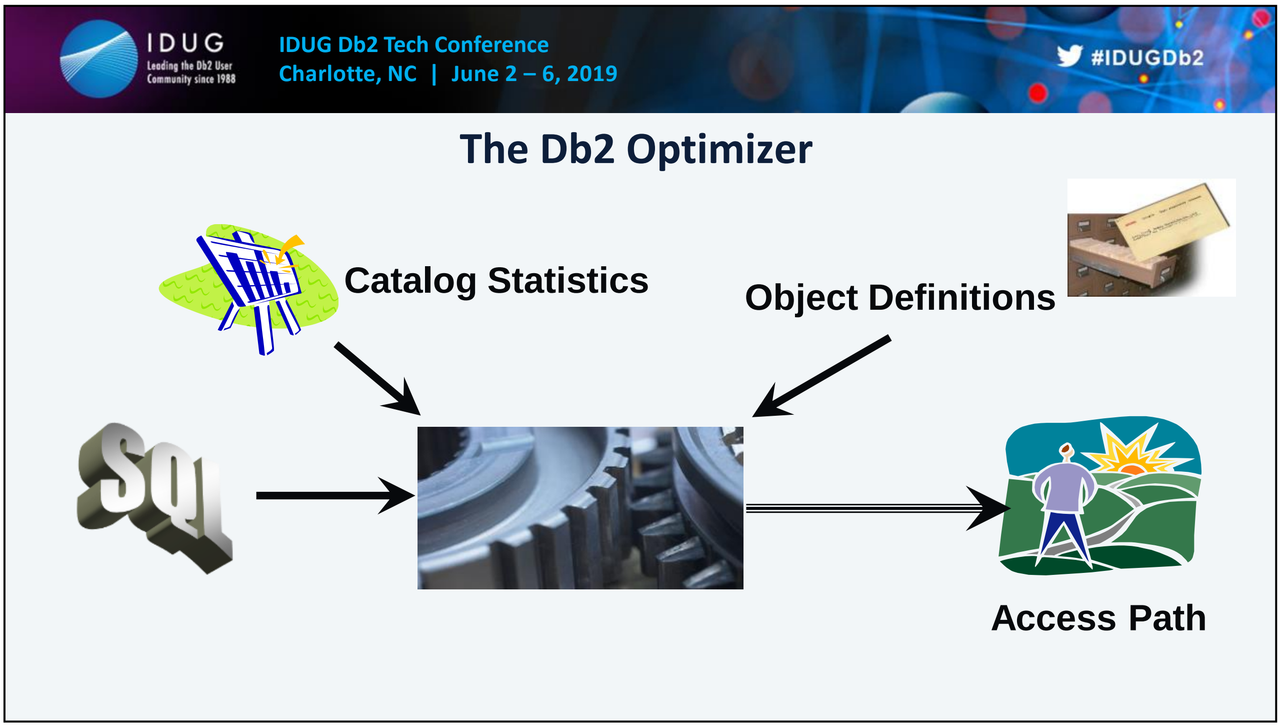
- The number of time cursors are Opened/Closed
- The number of random SQL requests (noted as synchronized reads in Db2 monitors).

Multi Row Fetch, Update, and Inserting. Recursive SQL. Select from Insert.
'Merge' processing. Fetch First / Order By within subqueries.

1) Minimize programming API (Application Programming Interface) to Db2. There is overhead involved in API calls, no matter how efficient the call may be.

2) Minimize the number of fetches by using multi row fetch. Take advantage of multi row deletes/inserts also.

3) Distributed Apps: Once in compatibility mode in V8 the blocks used for block fetching are built using the multi-row capability without any code change. This results in automatic savings for example distributed SQLJ applications.


IDUG
Leading the Db2 User Community since 1988
IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019
#IDUGDb2

The Db2 Optimizer

Through the Data Manipulation Language (DML) the user of a Db2 database supplies the “WHAT”; that is, the data that is needed from the database to satisfy the business requirements. Db2 then uses the information in the Db2 Catalog to resolve “WHERE” the data resides. The Db2 Optimizer is then responsible for determining the all important “HOW” to access the data most efficiently.

Ideally, the user of a relational database is not concerned with how the system accesses data. This is probably true for an end user of Db2, who writes SQL queries quickly for one-time or occasional use. It is less true for developers who write application pro-grams and transactions, some of which will be executed thou-sands of times a day. For these cases, some attention to Db2 access methods can significantly improve performance.

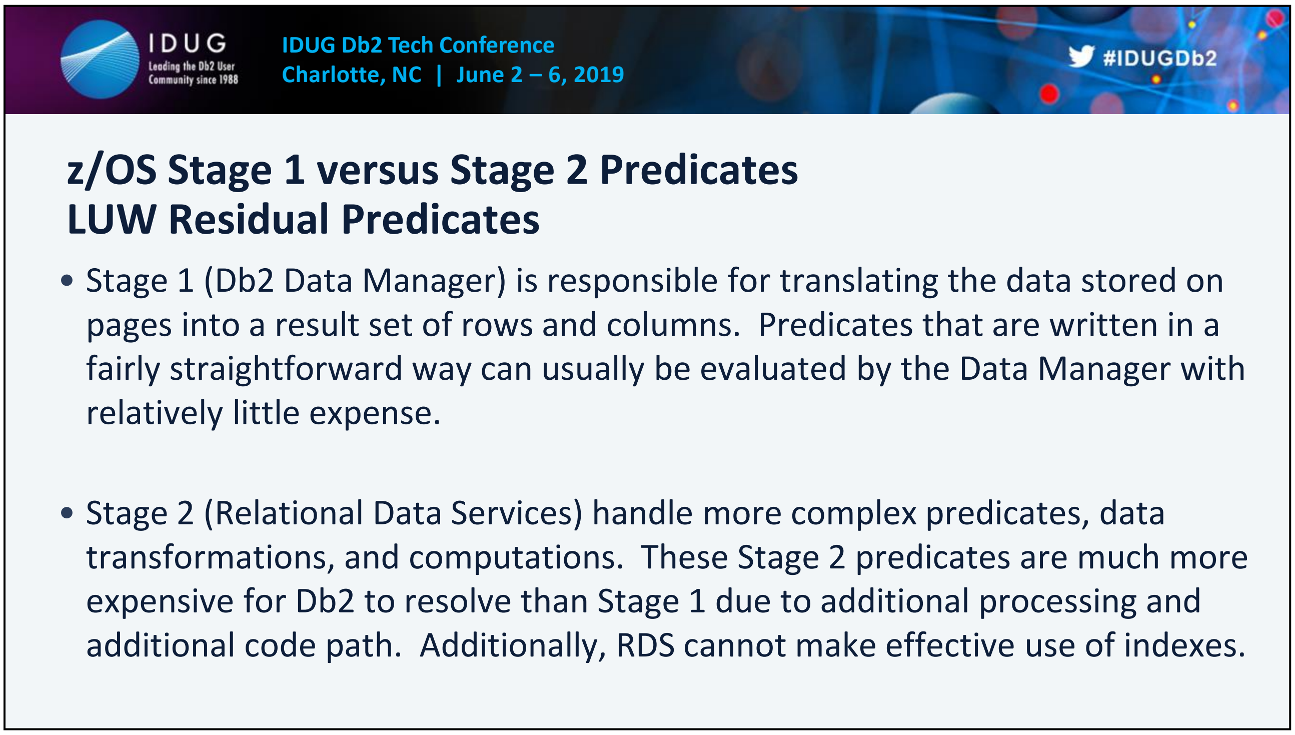
Db2’s access paths can be influenced in four ways:

- By rewriting a query in a more efficient form.
- By creating, altering, or dropping indexes.
- By updating the catalog statistics that Db2 uses to estimate access costs.
- By utilizing Optimizer Hints.

Watch out for different RID pool sizing from production and test environments. The row id (RID) pool is used for the RID sorts that accompany optimizer access path techniques such as list pre-fetch, hybrid join, and multi-index access. These pool sizes may vary from production environments to test environments with typically more RID pool sort area in production. This can at times cause the access path to be different in different environments

Other items affecting optimization:

- Buffer Pools.
- Rid Pools.



IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

z/OS Stage 1 versus Stage 2 Predicates
LUW Residual 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.

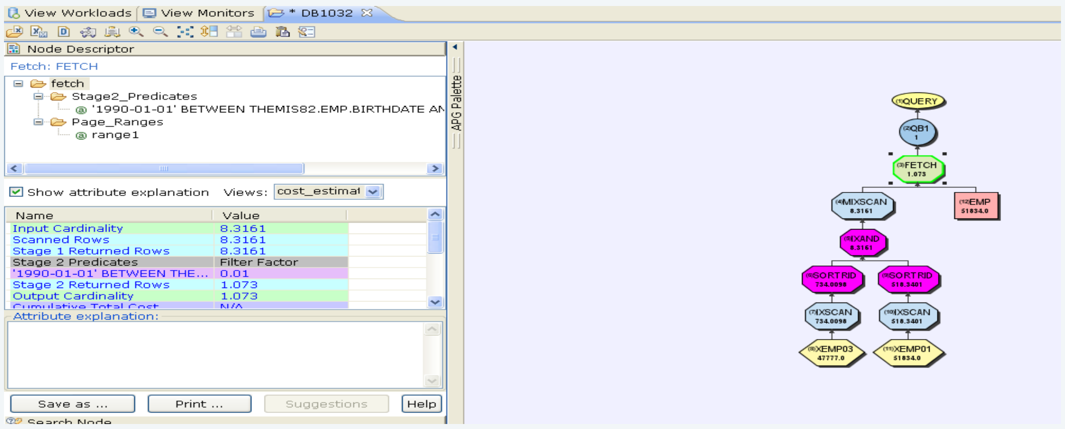
Stage 1 = Sargeable

Stage 2 = Non Sargeable. Predicate processing by this RDS are of Db2 is much more expensive
than the RDS Stage 1 area. Additional processing, additional code path, much more expensive then stage 1.

Indexable predicates evaluated first, Stage 1 predicates, next, and Stage 2 predicates last.

Use the Visual Explain in IBM Data Studio or query directly the DSN_PREDICAT_TABLE to see any stage 2 predicates. Note the filter factor information also.

Stage 2 Predicates



Fetch: FETCH

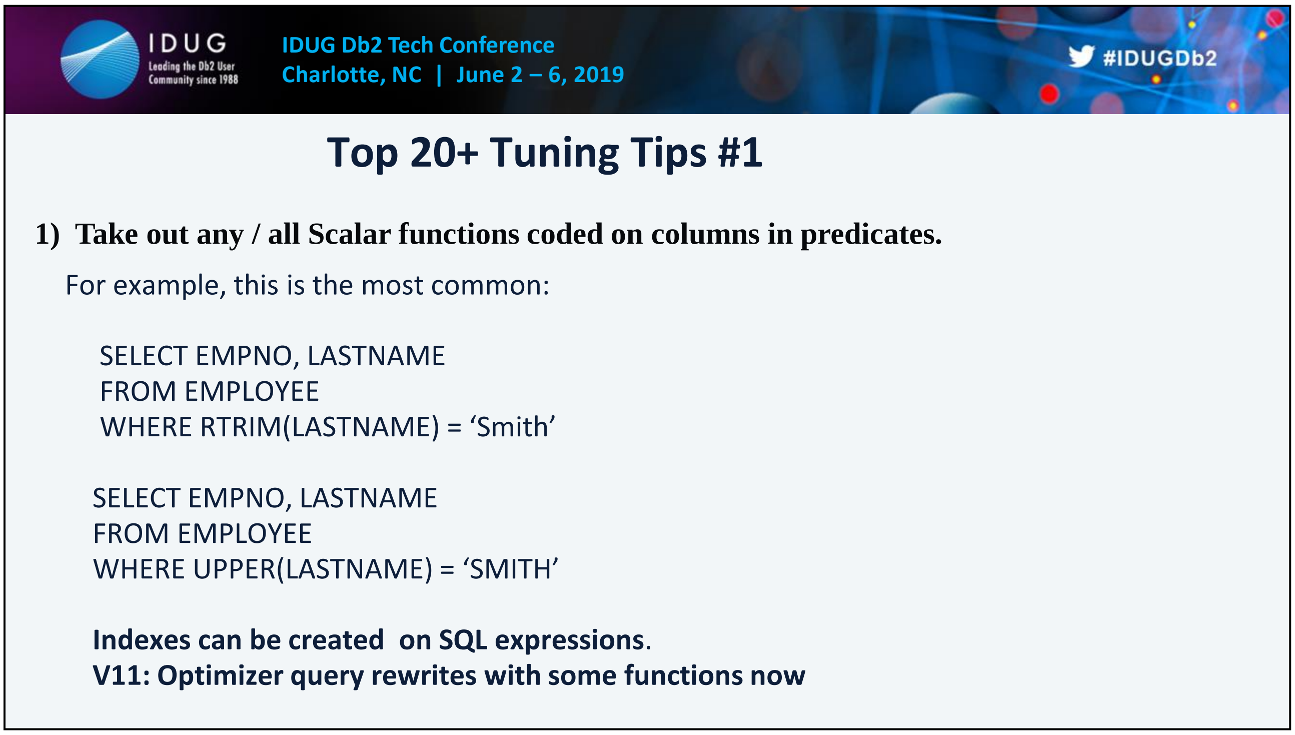
```
fetch
  Stage2_Predicates
    '1990-01-01' BETWEEN THEMIS82.EMP.BIRTHDATE AN
  Page_Ranges
    range1
```

☑ Show attribute explanation   Views: cost_estimat

Name	Value
Input Cardinality	8.3161
Scanned Rows	8.3161
Stage 1 Returned Rows	8.3161
Stage 2 Predicates	Filter Factor
'1990-01-01' BETWEEN THE...	0.01
Stage 2 Returned Rows	1.073
Output Cardinality	1.073
Cumulative Total Cost	N/A

Attribute explanation:

1) Click on the FETCH box to see any/all predicates not associated with the index chosen
2) Click on the IXSCAN boxes to see matching index and screening index predicate information



IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

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 RTRIM(LASTNAME) = 'Smith'

SELECT EMPNO, LASTNAME
FROM EMPLOYEE
WHERE UPPER(LASTNAME) = 'SMITH'
```

Indexes can be created on SQL expressions.
V11: Optimizer query rewrites with some functions now

Index on expressions.

1) If the expression contains a column reference and there exists an index on the expression, then the following become Stage 1 predicates.

T1.COL = T2 col expr, T1.COL <> T2 col expr, expression = value, expression <> value,

expression op value , expression op (subquery)

2) Create index empx3 on table employee (year(hiredate) asc)

Create index empx4 on table employee (upper(lastname, locale) asc)

for example (upper(lastname, 'UNI') asc) Unicode

for example (upper(lastname, 'En_US') asc) English USA

Note: When an index using Upper or Lower, the locale must be specified.

APAR (PK68295) will remove the locale requirement

Create index empx4 on table employee (soundex(lastname) asc)

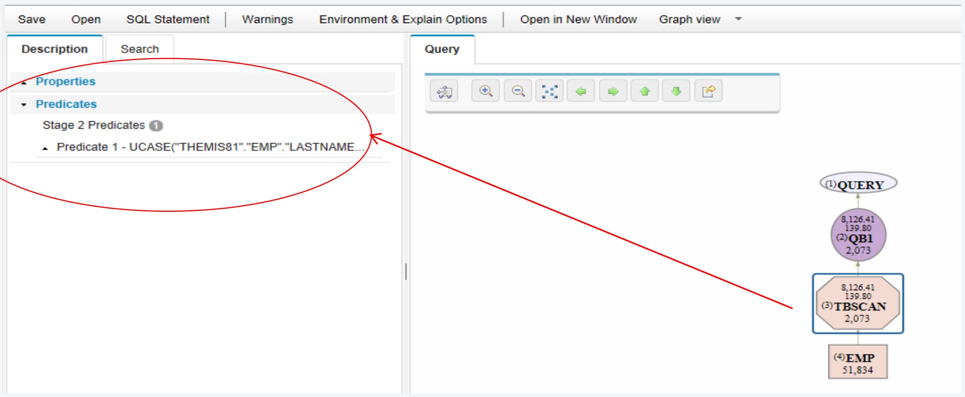
IDUG Leading the Db2 User Community since 1988

IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

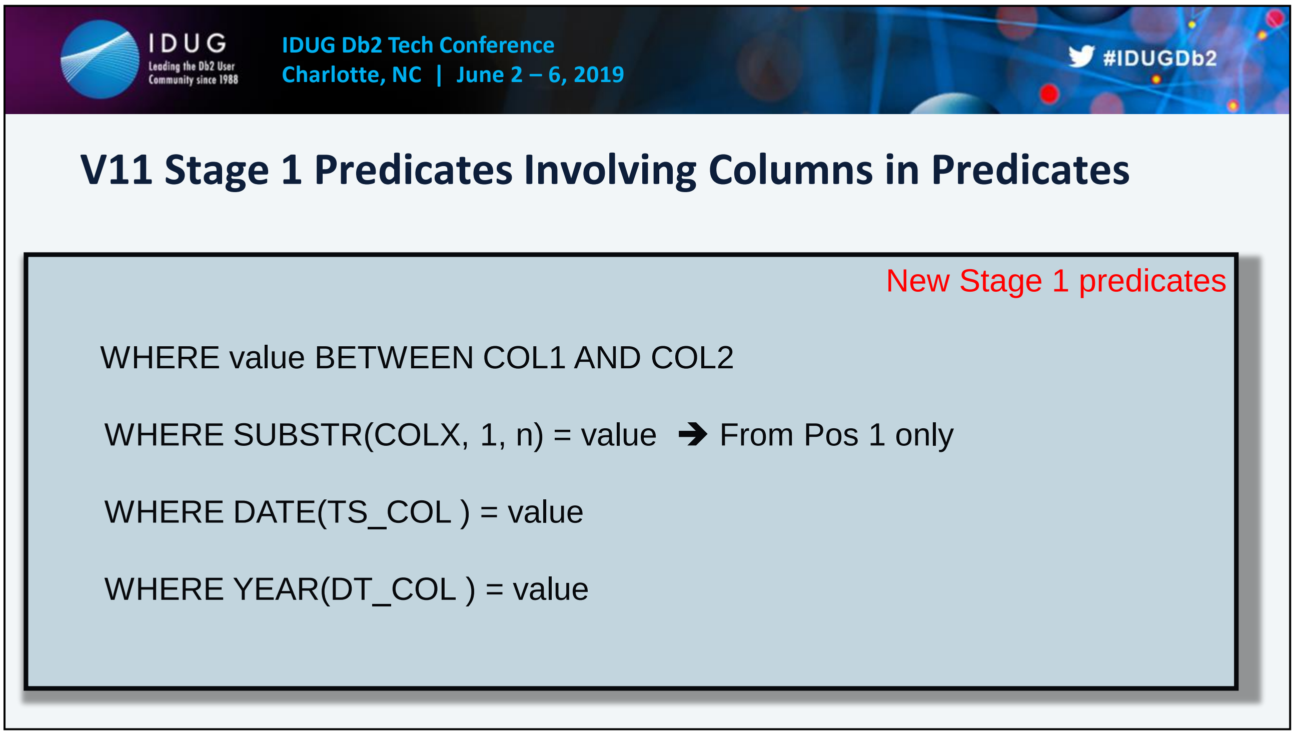
Top 25+ Tuning Tips #1

Use the Visual Explain, notice the predicate being stage 2 and the index beginning with LASTNAME was not chosen.



Stage 2 Predicates 1
Predicate 1 - UCASE("THEMIS81"."EMP"."LASTNAME...

This table contains an index (PRSTDATE, PRENDATE) and because of the YEAR function, Db2 did not choose to use the index, and the predicate is shown as Stage 2.


IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019
#IDUGDb2

V11 Stage 1 Predicates Involving Columns in Predicates

New Stage 1 predicates

WHERE value BETWEEN COL1 AND COL2

WHERE SUBSTR(COLX, 1, n) = value ➔ From Pos 1 only

WHERE DATE(TS_COL) = value

WHERE YEAR(DT_COL) = value

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

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

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

Example1:

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

V11, this becomes:

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

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

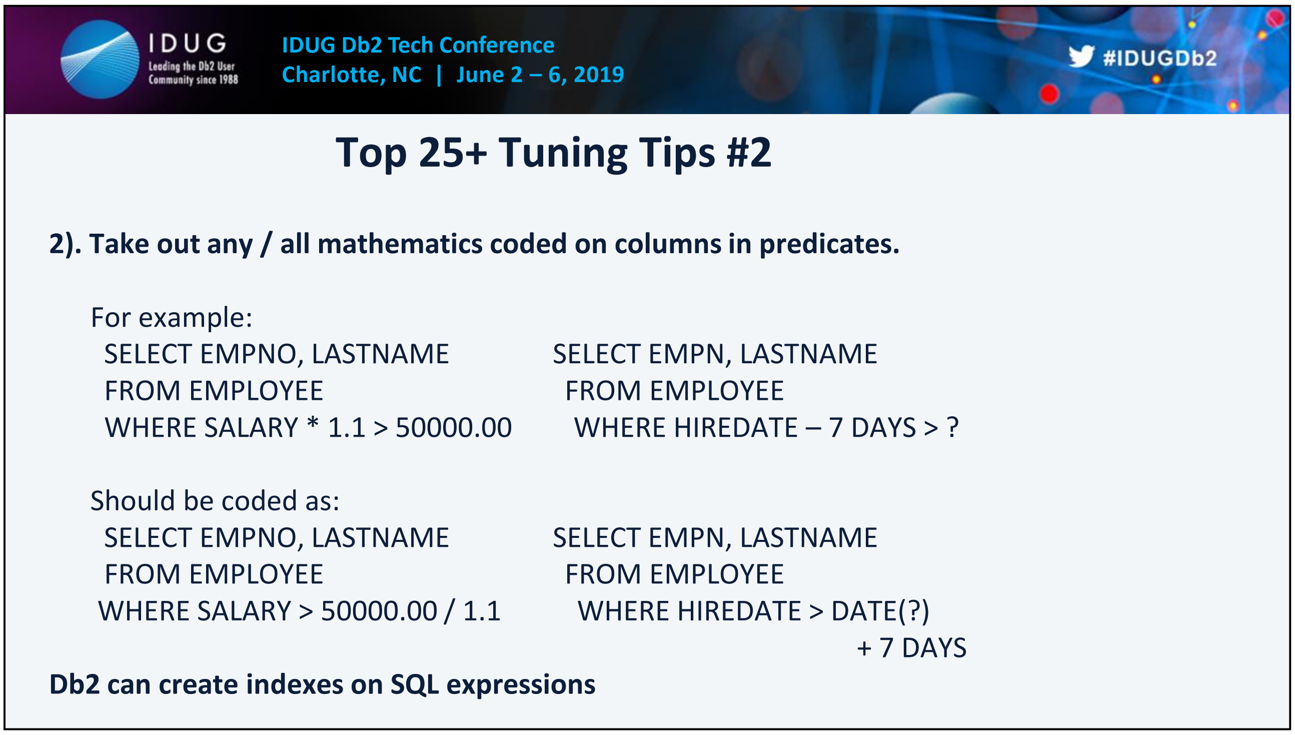
Example2:

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

V11, this becomes:

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

. Example: SUBSTR(LASTNAME,1,3) <=‘AND’ becomes LASTNAME <= ‘C1D5C4FFFFFFFFFFFFFFFFF’



IDUG Leading the Db2 User Community since 1988

IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

Top 25+ Tuning Tips #2

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

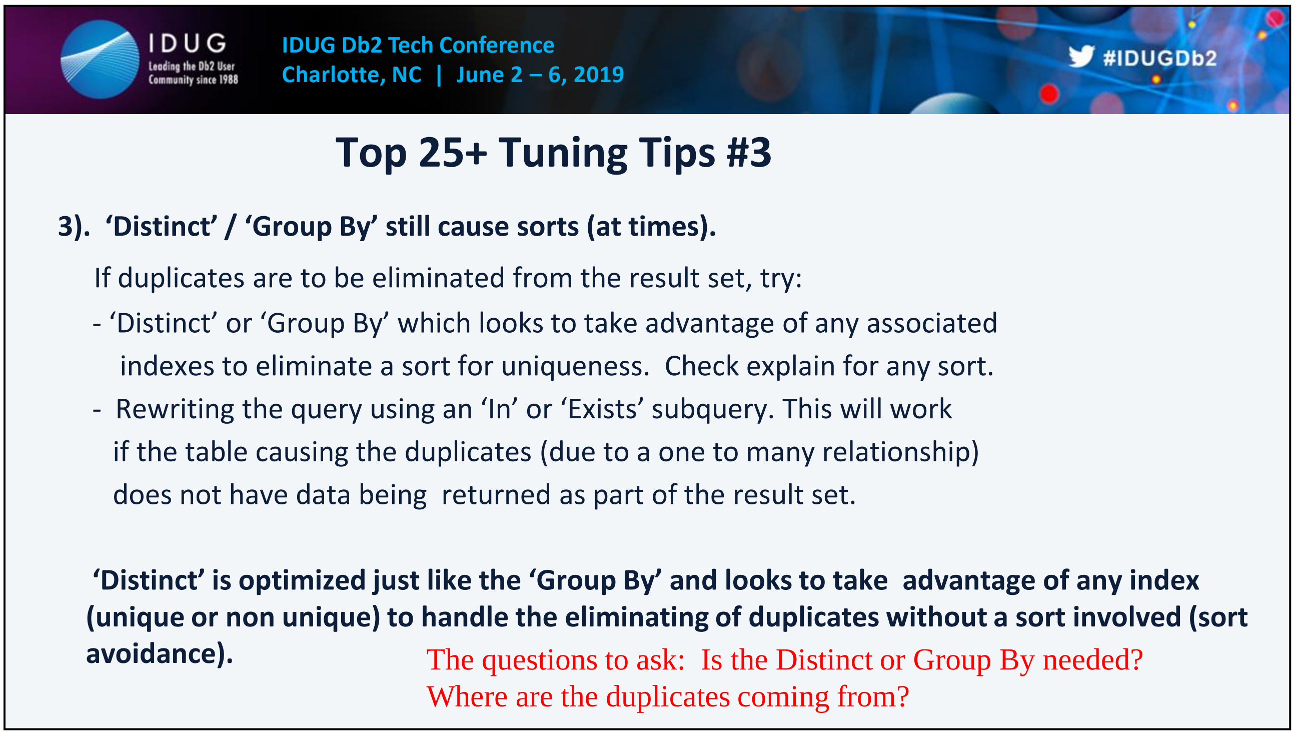
For example:

```
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
```

Db2 can create indexes on SQL expressions


IDUG Leading the Db2 User Community since 1988

IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

Top 25+ Tuning Tips #3

3). ‘Distinct’ / ‘Group By’ still cause sorts (at times).

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

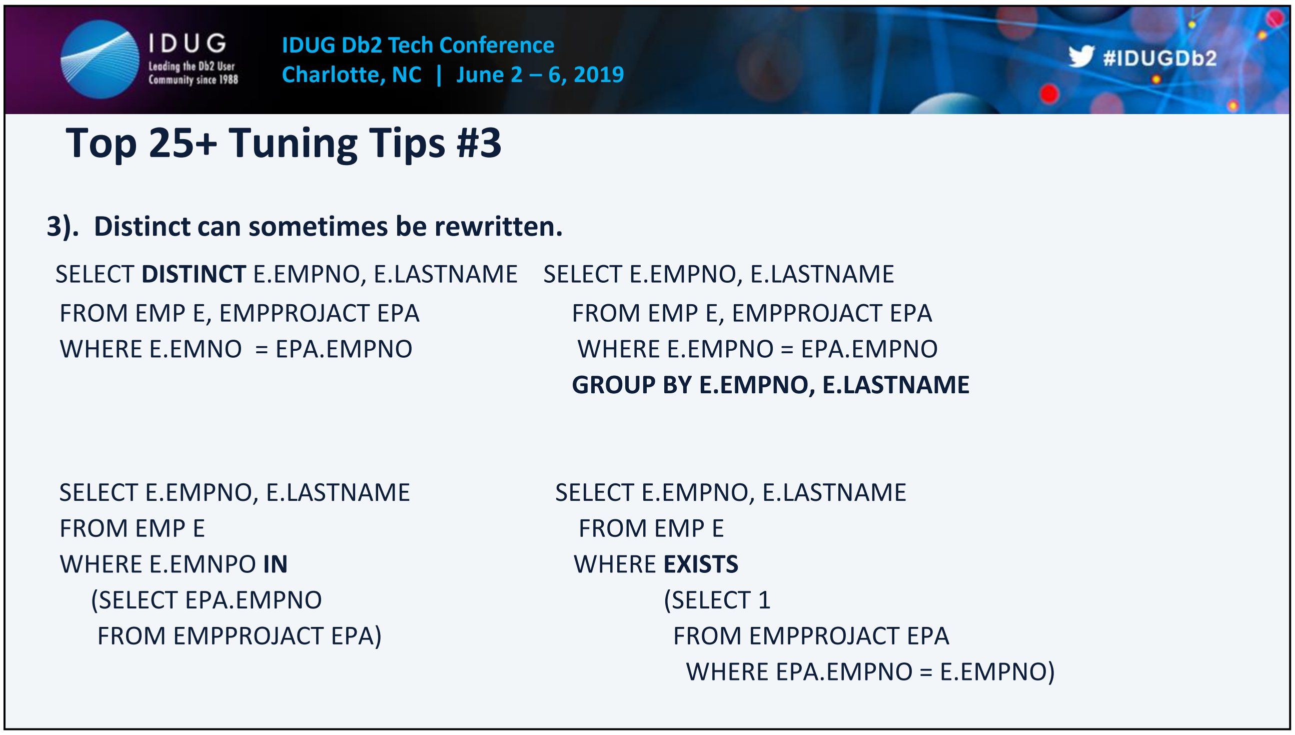
- ‘Distinct’ or ‘Group By’ which looks to take advantage of any associated indexes to eliminate a sort for uniqueness. Check explain for any sort.
- Rewriting the query using an ‘In’ or ‘Exists’ subquery. This will work if the table causing the duplicates (due to a one to many relationship) does not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

‘Distinct’ is optimized just like the ‘Group By’ and looks to take advantage of any index (unique or non unique) to handle the eliminating of duplicates without a sort involved (sort avoidance).

The questions to ask: Is the Distinct or Group By needed?
Where are the duplicates coming from?

1) There are sort enhancements for both ‘Distinct’ and ‘Group By’ with no column function. Was already available prior to V9 with ‘Group By’ and a column function. It now handles the duplicates more efficiently in the input phase, elimination a step 2 passing of data to a sort merge.

2) But developers should first ask… does the query need the Distinct or Group By?


IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019
#IDUGDb2

Top 25+ Tuning Tips #3

3). Distinct can sometimes be rewritten.

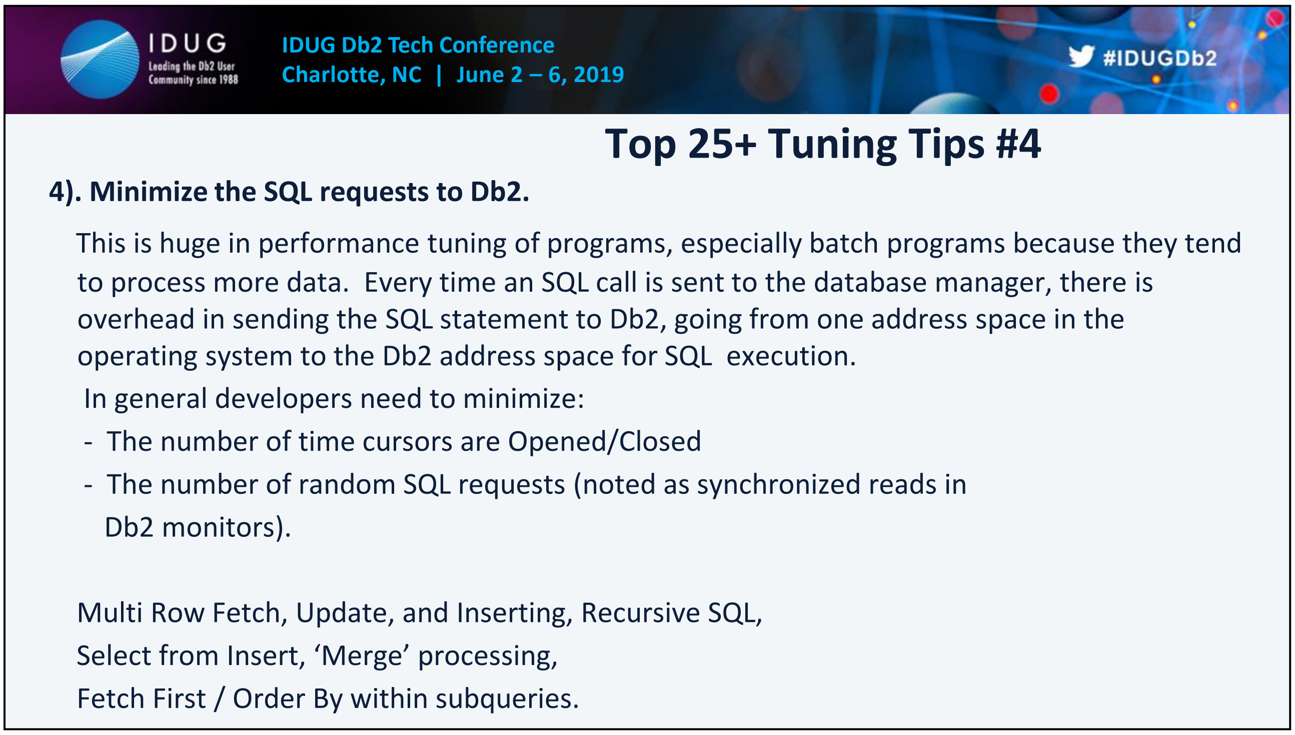
```
SELECT DISTINCT E.EMPNO, E.LASTNAME
 FROM EMP E, EMPPROJACT EPA
 WHERE E.EMNO  = EPA.EMPNO
```

```
SELECT E.EMPNO, E.LASTNAME
    FROM EMP E, EMPPROJACT EPA
     WHERE E.EMPNO = 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)
```

Often times, if one of the tables has no columns being selected from it, it can then be moved to a subquery.



IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

Top 25+ Tuning Tips #4

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:

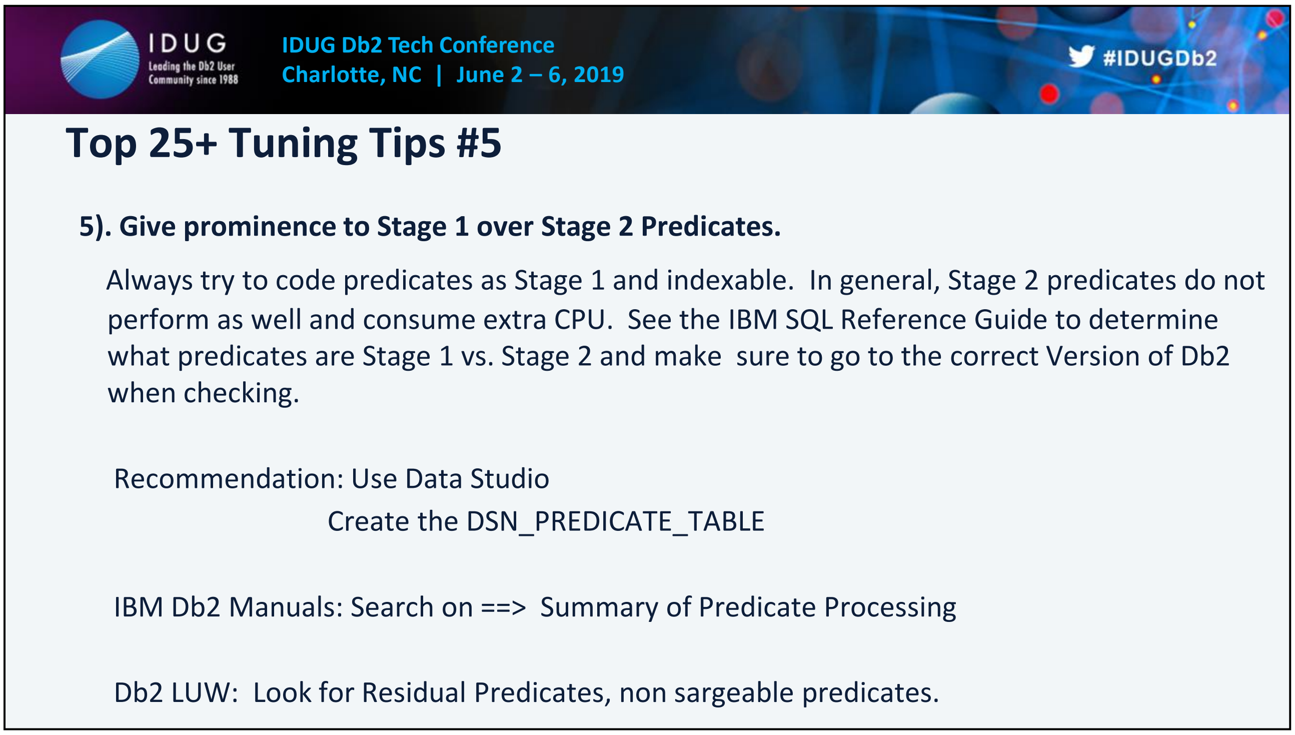
- The number of time cursors are Opened/Closed
- The number of random SQL requests (noted as synchronized reads in Db2 monitors).

Multi Row Fetch, Update, and Inserting, Recursive SQL,
Select from Insert, 'Merge' processing,
Fetch First / Order By within subqueries.

1) Minimize programming API (Application Programming Interface) to Db2. There is overhead involved in API calls, no matter how efficient the call may be.

2) Minimize the number of fetches by using multi row fetch. Take advantage of multi row deletes/inserts also.

3) Distributed Apps: Once in compatibility mode in V8 the blocks used for block fetching are built using the multi-row capability without any code change. This results in automatic savings for example distributed SQLJ applications.



Top 25+ Tuning Tips #5

5). 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 Data Studio
Create the DSN_PREDICATE_TABLE

IBM Db2 Manuals: Search on ==> Summary of Predicate Processing

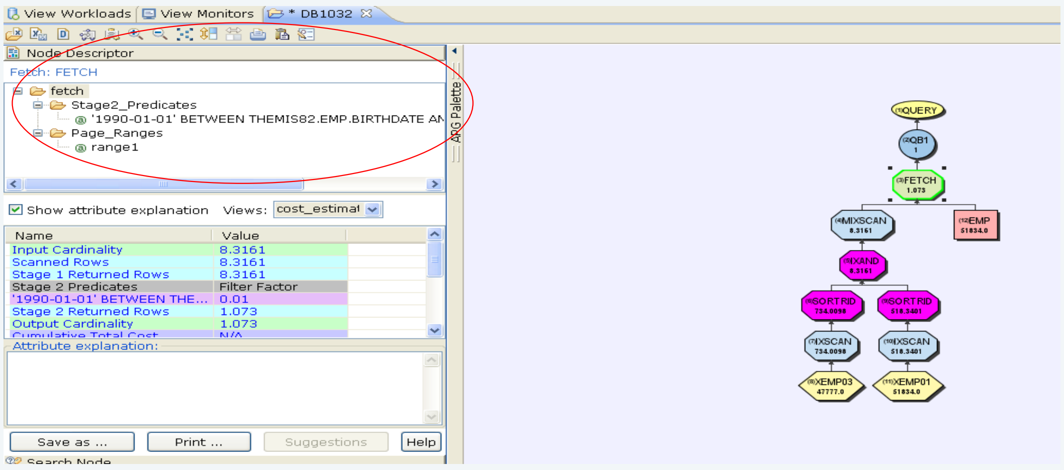
Db2 LUW: Look for Residual Predicates, non sargeable predicates.

The Visual Explain in IBM Data Studio shows a folder of any Stage 2 predicates involved in the query.

The stage 1 and stage 2 predicate information are loaded into the DSN_PREDICAT_TABLE when executing a Bind with Explain.

Top 25+ Tuning Tips #5 Cont…

Use the Visual Explain for guidance to Stage 1, Stage2, and Filter Factor information.



Fetch: FETCH

```
fetch
  Stage2_Predicates
    '1990-01-01' BETWEEN THEMIS82.EMP.BIRTHDATE AN
  Page_Ranges
    range1
```

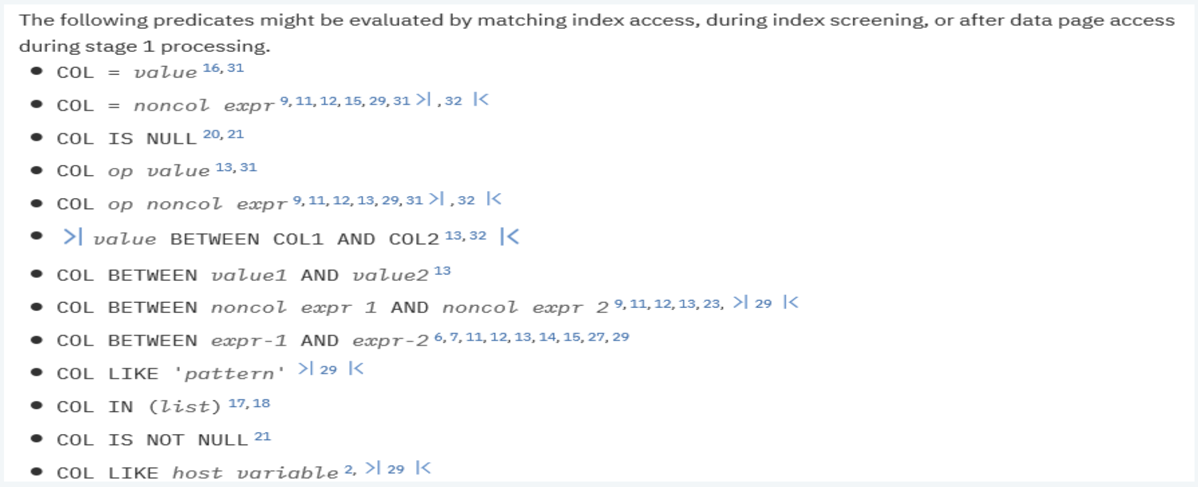
Show attribute explanation   Views: cost_estimat

Name	Value
Input Cardinality	8.3161
Scanned Rows	8.3161
Stage 1 Returned Rows	8.3161
Stage 2 Predicates	Filter Factor
'1990-01-01' BETWEEN THE…	0.01
Stage 2 Returned Rows	1.073
Output Cardinality	1.073
Cumulative Total Cost	N/A

Attribute explanation:

1) Click on the FETCH box to see any/all predicates not associated with the index chosen
2) Click on the IXSCAN boxes to see matching index and screening index predicate information

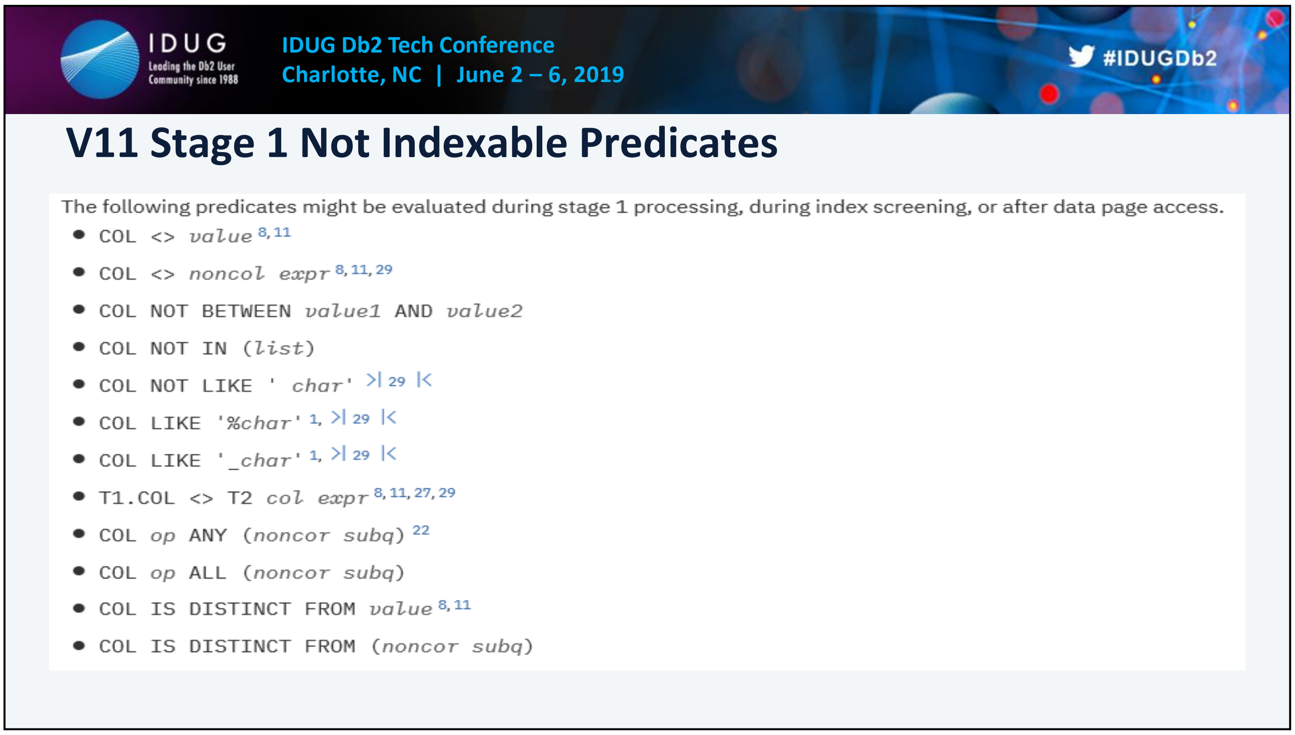
V11 Stage 1 Indexable Predicates


The following predicates might be evaluated by matching index access, during index screening, or after data page access during stage 1 processing.

- `COL = value`[16, 31]
- `COL = noncol expr`[9, 11, 12, 15, 29, 31 >| , 32 |<]
- `COL IS NULL`[20, 21]
- `COL op value`[13, 31]
- `COL op noncol expr`[9, 11, 12, 13, 29, 31 >| , 32 |<]
- >| `value BETWEEN COL1 AND COL2`[13, 32] |<
- `COL BETWEEN value1 AND value2`[13]
- `COL BETWEEN noncol expr 1 AND noncol expr 2`[9, 11, 12, 13, 23, >| 29 |<]
- `COL BETWEEN expr-1 AND expr-2`[6, 7, 11, 12, 13, 14, 15, 27, 29]
- `COL LIKE 'pattern'` >| [29] |<
- `COL IN (list)`[17, 18]
- `COL IS NOT NULL`[21]
- `COL LIKE host variable`[2, >| 29 |<]

Stage indexable predicates [31] This is a subset of many predicates from the manual.

Search on Db2 V11 summary of Predicate Processing



IDUG Leading the Db2 User Community since 1988

IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

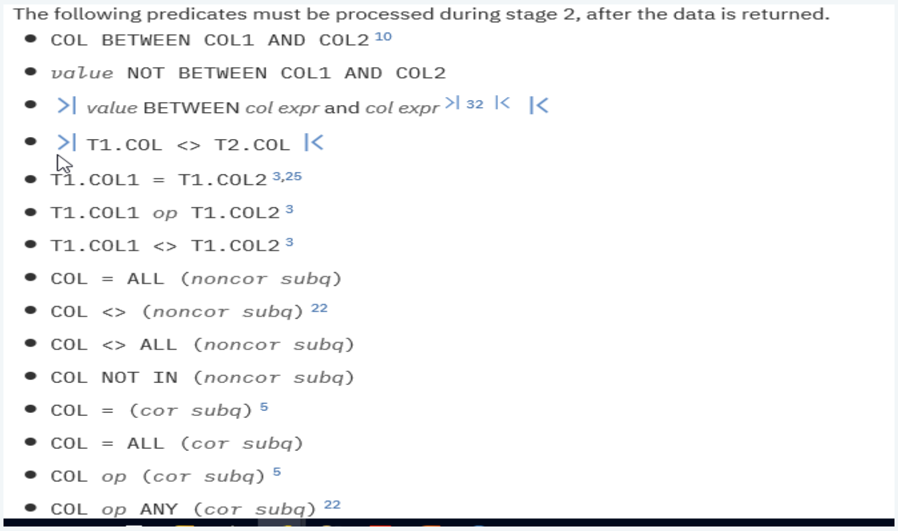
V11 Stage 1 Not Indexable Predicates

The following predicates might be evaluated during stage 1 processing, during index screening, or after data page access.

- COL <> *value* [8, 11]
- COL <> *noncol expr* [8, 11, 29]
- COL NOT BETWEEN *value1* AND *value2*
- COL NOT IN (*list*)
- COL NOT LIKE ' *char*' >| [29] |<
- COL LIKE '%*char*' [1], >| [29] |<
- COL LIKE '_*char*' [1], >| [29] |<
- T1.COL <> T2 *col expr* [8, 11, 27, 29]
- COL *op* ANY (*noncor subq*) [22]
- COL *op* ALL (*noncor subq*)
- COL IS DISTINCT FROM *value* [8, 11]
- COL IS DISTINCT FROM (*noncor subq*)

Stage 1 not indexable predicates [31] These predicates might be evaluated during stage 1 processing, during index screening, or after data page access.

V11 Stage 2 Predicates

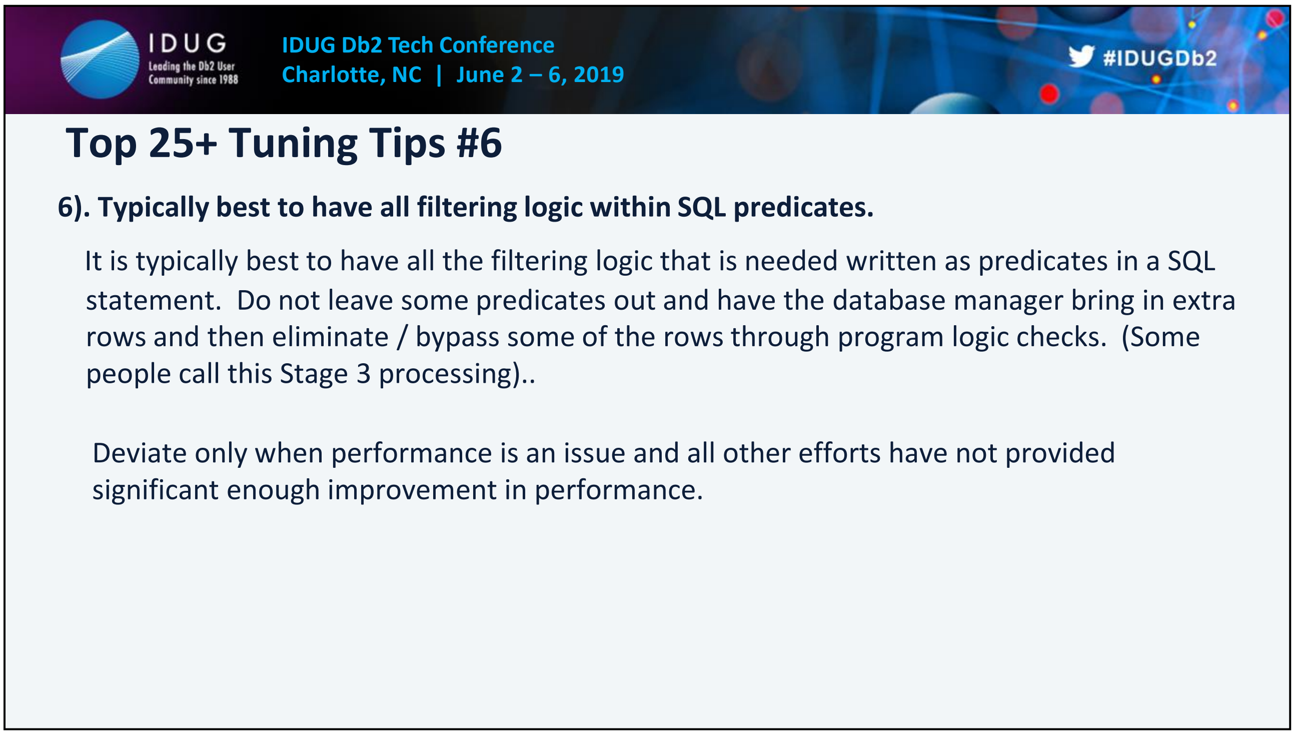


The following predicates must be processed during stage 2, after the data is returned.

- `COL BETWEEN COL1 AND COL2` [10]
- *value* `NOT BETWEEN COL1 AND COL2`
- >| *value* BETWEEN *col expr* and *col expr* >| [32] |< |<
- >| `T1.COL <> T2.COL` |<
- `T1.COL1 = T1.COL2` [3,25]
- `T1.COL1` *op* `T1.COL2` [3]
- `T1.COL1 <> T1.COL2` [3]
- `COL = ALL (`*noncor subq*`)`
- `COL <> (`*noncor subq*`)` [22]
- `COL <> ALL (`*noncor subq*`)`
- `COL NOT IN (`*noncor subq*`)`
- `COL = (`*cor subq*`)` [5]
- `COL = ALL (`*cor subq*`)`
- `COL` *op* `(`*cor subq*`)` [5]
- `COL` *op* `ANY (`*cor subq*`)` [22]

Stage 2 predicates

The predicates must be processed during stage 2, after the data is returned. This is a subset of the many S2 predicates.



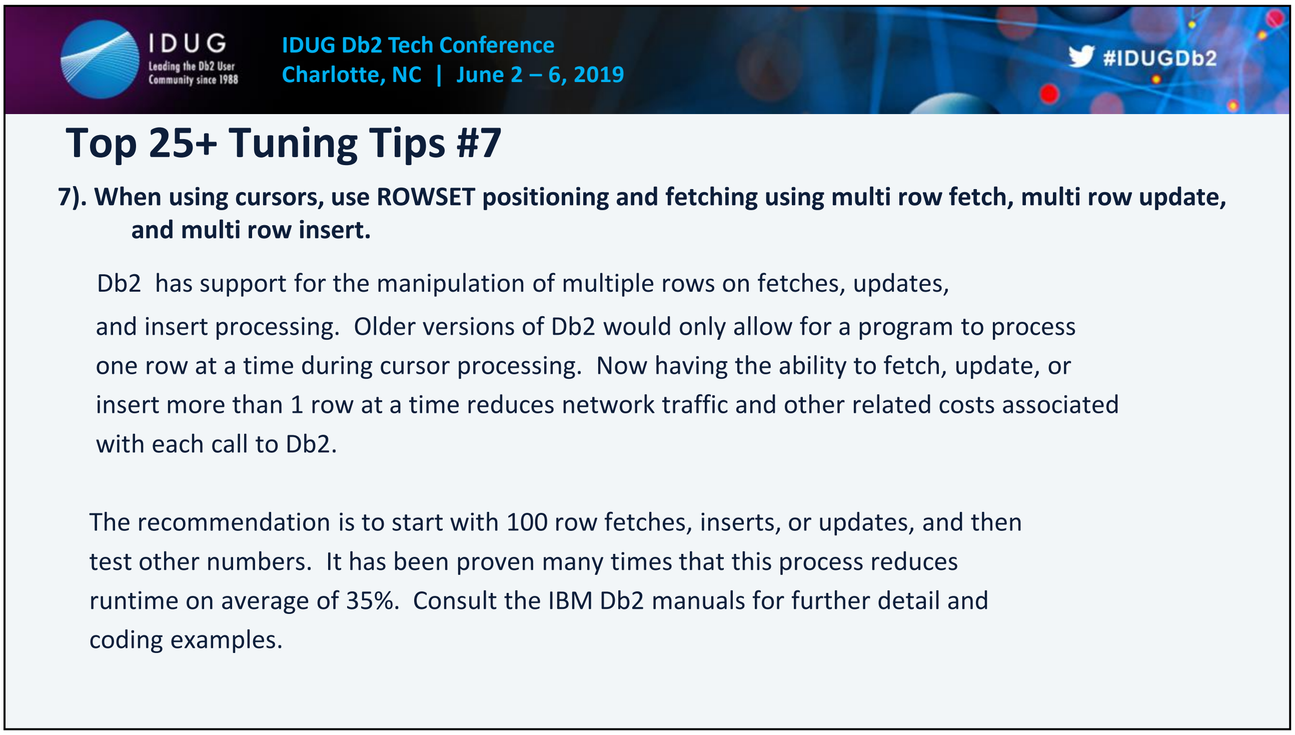
Top 25+ Tuning Tips #6

6). Typically best to have all filtering logic within SQL predicates.

It is typically 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.

Often times developers may execute a 'Fast Unload' to a mainframe file, and have a program read every record from the file and skip unwanted records. At times this can be very efficient especially if the program is going to Process a large percentage of rows from the table.


IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019
#IDUGDb2

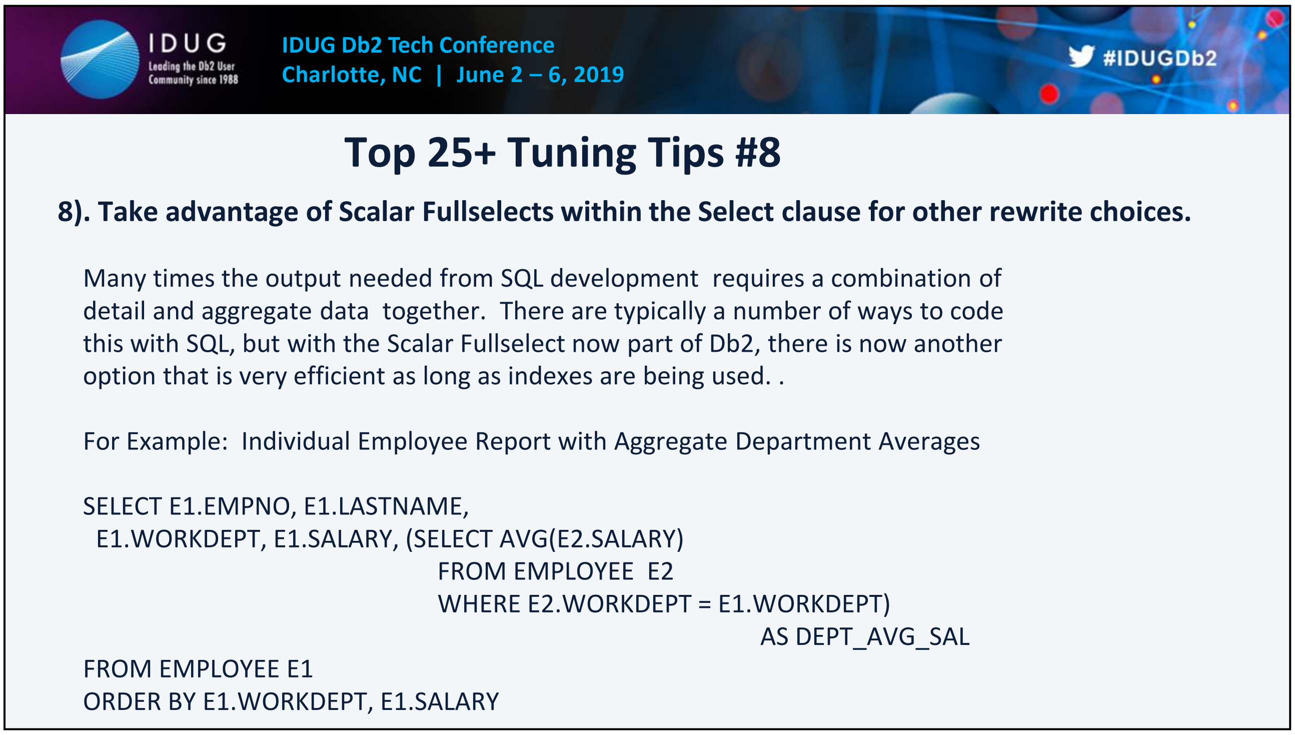
Top 25+ Tuning Tips #7

7). When using cursors, use ROWSET positioning and fetching using multi row fetch, multi row update, and multi row insert.

Db2 has support for the manipulation of multiple rows on fetches, updates, and insert processing. Older 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.

1) Check 'Get Diagnostics' only after receiving a SQLCODE <> 0
2) For multi-row fetch you get a +354 SQLCODE which says "one or more errors may have occurred."
3) For a non atomic multi-row insert you get a -253 if some of the rows fail, -254 if all fail
4) For an atomic multi-row insert you get the real SQLCODE of the first failure (since atomic means any failure backs out the entire insert), but you still need the diagnostics to determine which row actually tripped the error.
5) If you get multiple errors, they will be returned in reverse order. e.g. when inserting 5 rows, row no. 2 and row no. 4 had errors. GET DIAGNOSTICS shows three errors (not two!) - first is the generic, 2nd is error for row #4 and third is error for row #2.
6) If you perform a multi-row operation and receive an SQLCODE of 0 then you generally have no need for GET DIAGNOSTICS, which is VERY expensive. So, check SQLCODE first and only go to the DIAGNOSTICS when errors are encountered.
7) Multi Row for even a small number of rows returned can be beneficial. When searching for a "break-even" (runtime), don't forget to take into consideration the length of the data rows being fetched. Multi-row fetch will be especially beneficial (even for just a few rows) if the rows are short in length.
8) Seen best results with large amounts of data being returned from a cursor. Fetches of 100 rows each.



IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

Top 25+ Tuning Tips #8

8). Take advantage of Scalar Fullselects within the Select clause for other rewrite choices.

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, 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

```
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
```

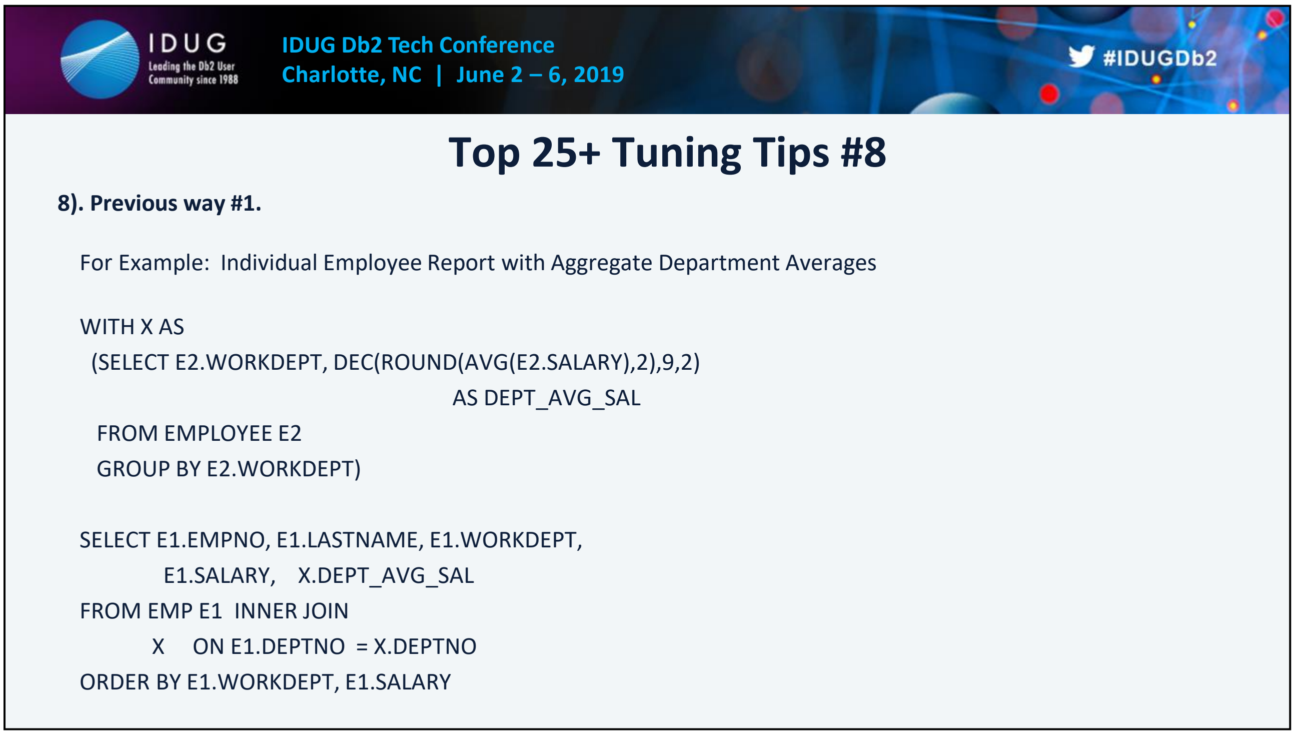
Not stating that this is the best option, but at times it may be. But it gives developers another way to code for certain results.

Prior Options:

```
1)  SELECT E1.EMPNO, E1.LASTNAME, E1.WORKDEPT, E1.SALARY, X.DEPT_AVG_SAL
    FROM EMPLOYEE E1  INNER JOIN
         (SELECT E2.DEPTNO, DEC(ROUND(AVG(E2.SALARY),2),9,2)
                                                   AS DEPT_AVG_SAL
          FROM EMP E2
          GROUP BY E2.WORKDEPT) AS X  ON E1.DEPTNO  = X.DEPTNO
    ORDER BY E1.WORKDEPT, E1.SALARY

2)  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

3) SELECT E1.EMPNO, E1.LASTNAME, E1.WORKDEPT, E1.SALARY,  AVG(E2.SALARY)  AS DEPT_AVG_SAL
   FROM EMP E1  INNER JOIN
        EMP E2  ON E1.DEPTNO  = E2.DEPTNO
   GROUP BY E1.EMPNO, E1.LASTNAME, E1.WORKDEPT, E1.SALARY
   ORDER BY E1.WORKDEPT, E1.SALARY
```


IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019
#IDUGDb2

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
```

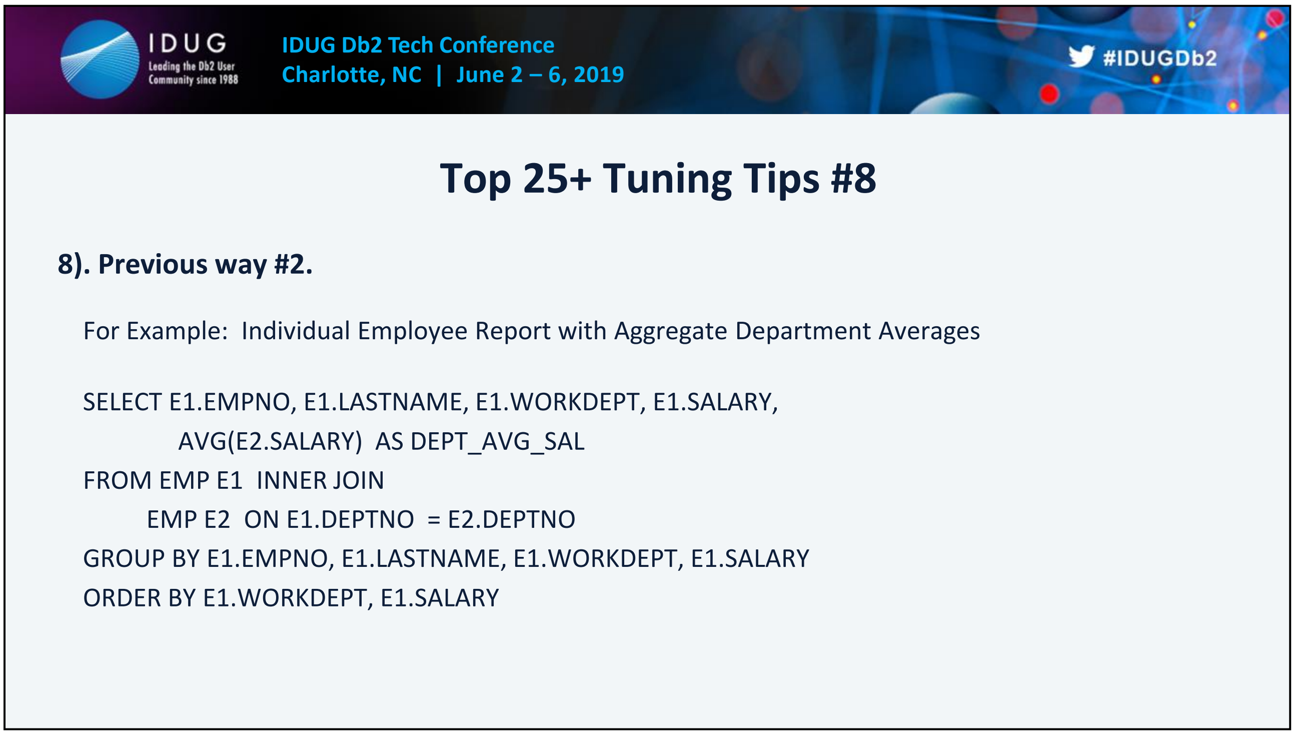
Prior Options:

```
-----------------
1)  SELECT E1.EMPNO, E1.LASTNAME, E1.WORKDEPT, E1.SALARY, X.DEPT_AVG_SAL
    FROM EMPLOYEE E1  INNER JOIN
         (SELECT E2.DEPTNO, DEC(ROUND(AVG(E2.SALARY),2),9,2)
                                                     AS DEPT_AVG_SAL
          FROM EMP E2
          GROUP BY E2.WORKDEPT) AS X  ON E1.DEPTNO  = X.DEPTNO
    ORDER BY E1.WORKDEPT, E1.SALARY

2)  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

3) SELECT E1.EMPNO, E1.LASTNAME, E1.WORKDEPT, E1.SALARY,  AVG(E2.SALARY)  AS DEPT_AVG_SAL
   FROM EMP E1  INNER JOIN
        EMP E2  ON E1.DEPTNO  = E2.DEPTNO
   GROUP BY E1.EMPNO, E1.LASTNAME, E1.WORKDEPT, E1.SALARY
   ORDER BY E1.WORKDEPT, E1.SALARY
```



IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

Top 25+ Tuning Tips #8

8). Previous way #2.

For Example: Individual Employee Report with Aggregate Department Averages

```
SELECT E1.EMPNO, E1.LASTNAME, E1.WORKDEPT, E1.SALARY,
            AVG(E2.SALARY)  AS DEPT_AVG_SAL
FROM EMP E1  INNER JOIN
        EMP E2  ON E1.DEPTNO  = E2.DEPTNO
GROUP BY E1.EMPNO, E1.LASTNAME, E1.WORKDEPT, E1.SALARY
ORDER BY E1.WORKDEPT, E1.SALARY
```

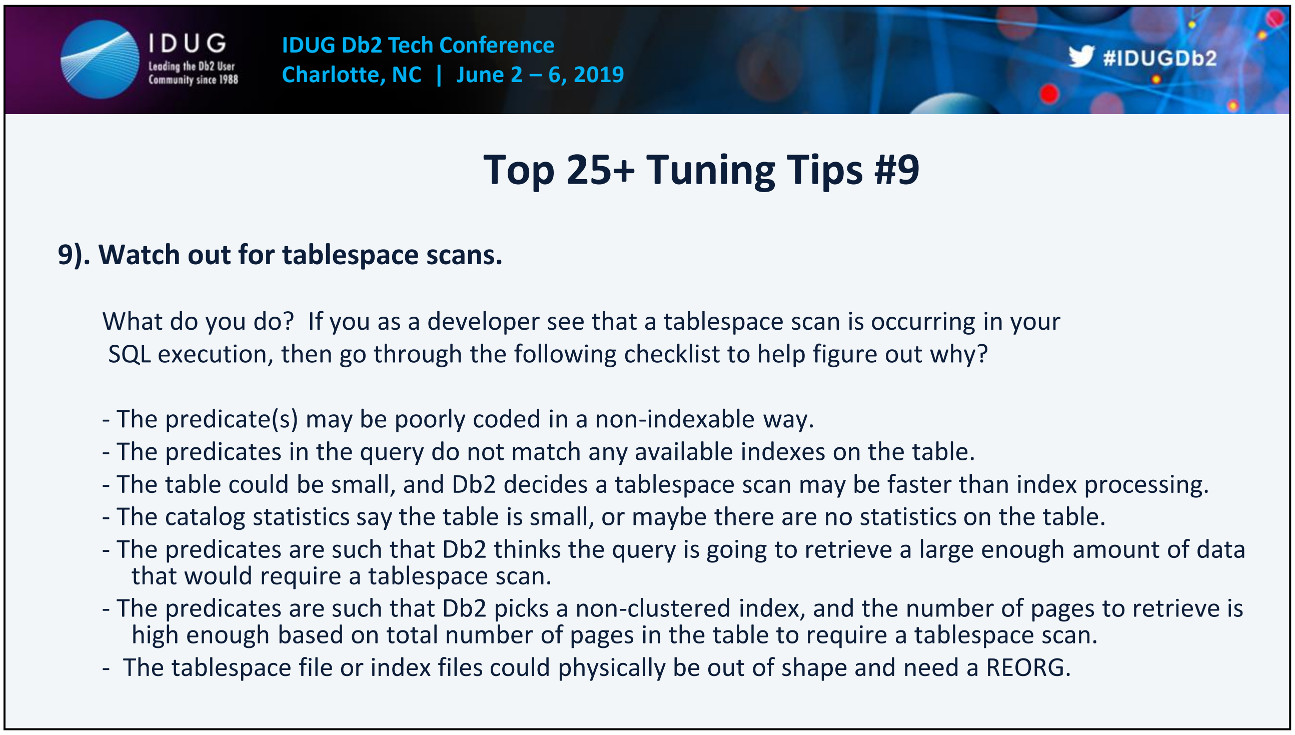
Prior Options:

```
1)   SELECT E1.EMPNO, E1.LASTNAME, E1.WORKDEPT, E1.SALARY, X.DEPT_AVG_SAL
     FROM EMPLOYEE E1  INNER JOIN
            (SELECT E2.DEPTNO, DEC(ROUND(AVG(E2.SALARY),2),9,2)
                                                          AS DEPT_AVG_SAL
             FROM EMP E2
             GROUP BY E2.WORKDEPT) AS X  ON E1.DEPTNO  = X.DEPTNO
     ORDER BY E1.WORKDEPT, E1.SALARY

2)   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

3) SELECT E1.EMPNO, E1.LASTNAME, E1.WORKDEPT, E1.SALARY,  AVG(E2.SALARY)  AS DEPT_AVG_SAL
   FROM EMP E1  INNER JOIN
          EMP E2  ON E1.DEPTNO  = E2.DEPTNO
   GROUP BY E1.EMPNO, E1.LASTNAME, E1.WORKDEPT, E1.SALARY
   ORDER BY E1.WORKDEPT, E1.SALARY
```


IDUG Leading the Db2 User Community since 1988

IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

Top 25+ Tuning Tips #9

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.
- 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.

Typically we do not want to see tablespace scans, but there re times when a tablespace scan will be more efficient than index processing:

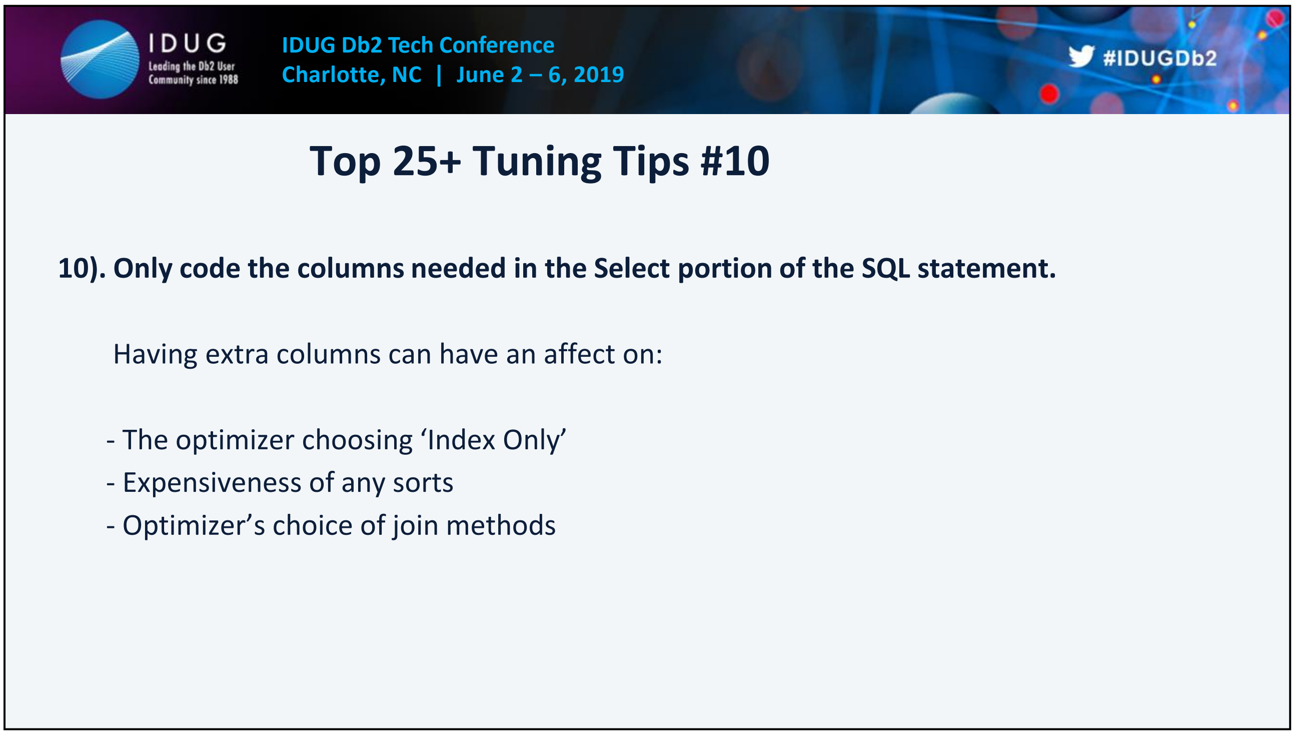
1) When retrieving a large amount of data from a table.

2) When processing using a non clustered index, and the number of pages hit in the table is high, whether many rows were returned or not.

Tablespace scans:

1) Will kick in ‘Sequential Prefetch’ and take advantage of asynchronous processing. V9 now has a larger prefetch quantity (From 32 to 64 for SQL processing, 128 pages for utilities).

2) Without non clustered index processing, there will be no list prefetch sort.



IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

Top 25+ Tuning Tips #10

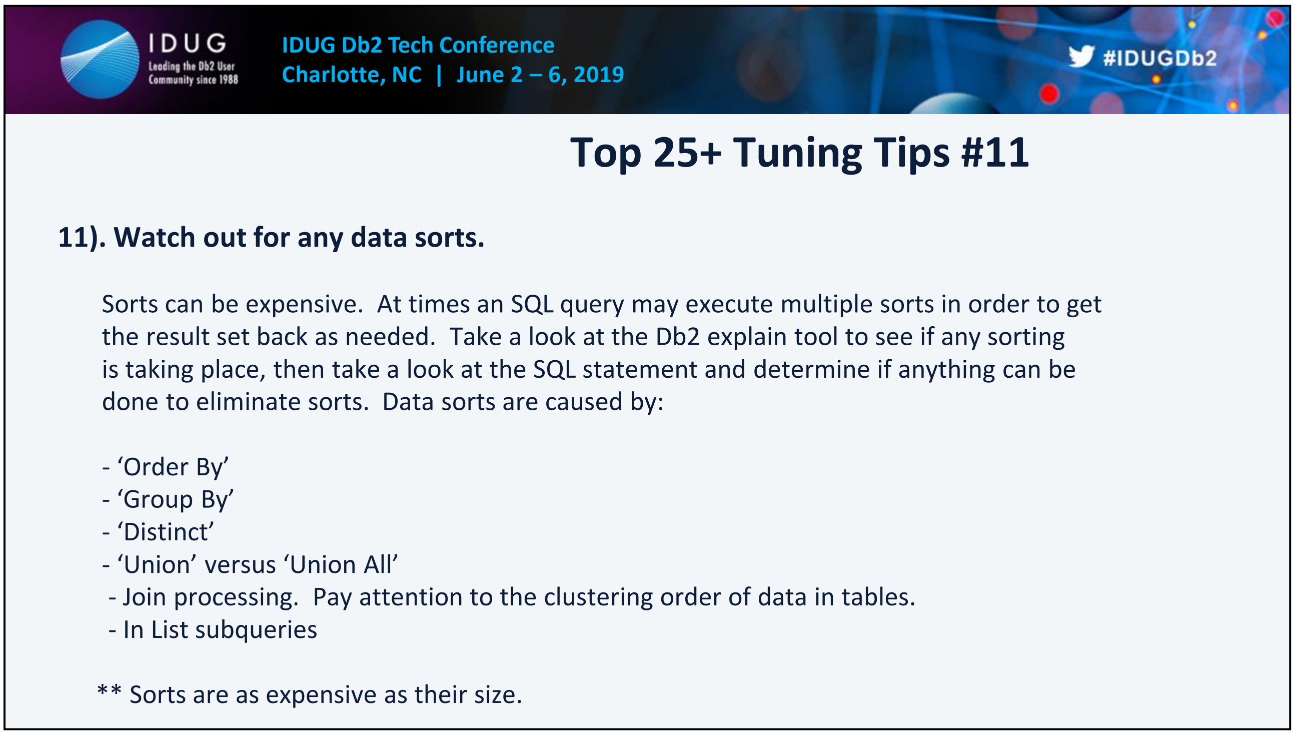
10). Only code the columns needed in the Select portion of the SQL statement.

Having extra columns can have an affect on:

- The optimizer choosing 'Index Only'
- Expensiveness of any sorts
- Optimizer's choice of join methods

For example: The optimizer may choose a Merge Scan join if:

- The qualifying rows of both new and composite tables are many
- The join predicate does not provide a significant amount of filtering
- Few columns are selected on the new table, meaning that when Db2 sorts the new table, the more efficient the sort


IDUG Leading the Db2 User Community since 1988

IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

Top 25+ Tuning Tips #11

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.

Many time developers have sorts carried over from copied code

Many times developers do not need to worry about duplicates

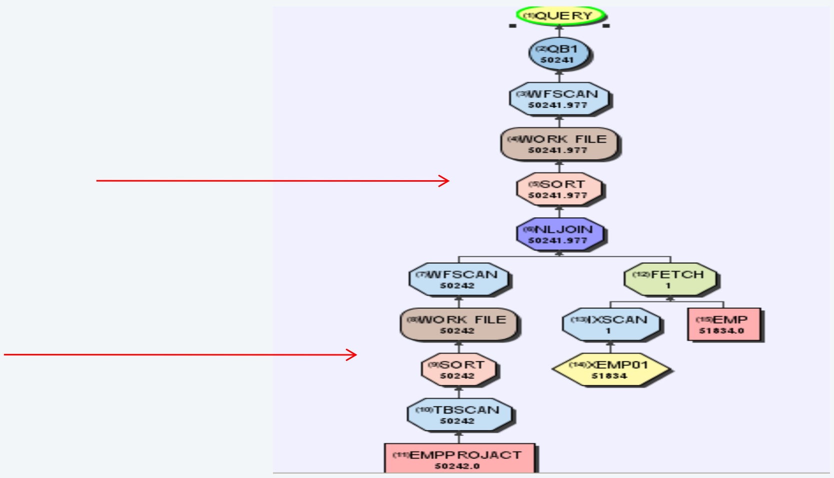
Many time developers can rewrite queries to alleviate sorts

IDUG Leading the Db2 User Community since 1988

IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

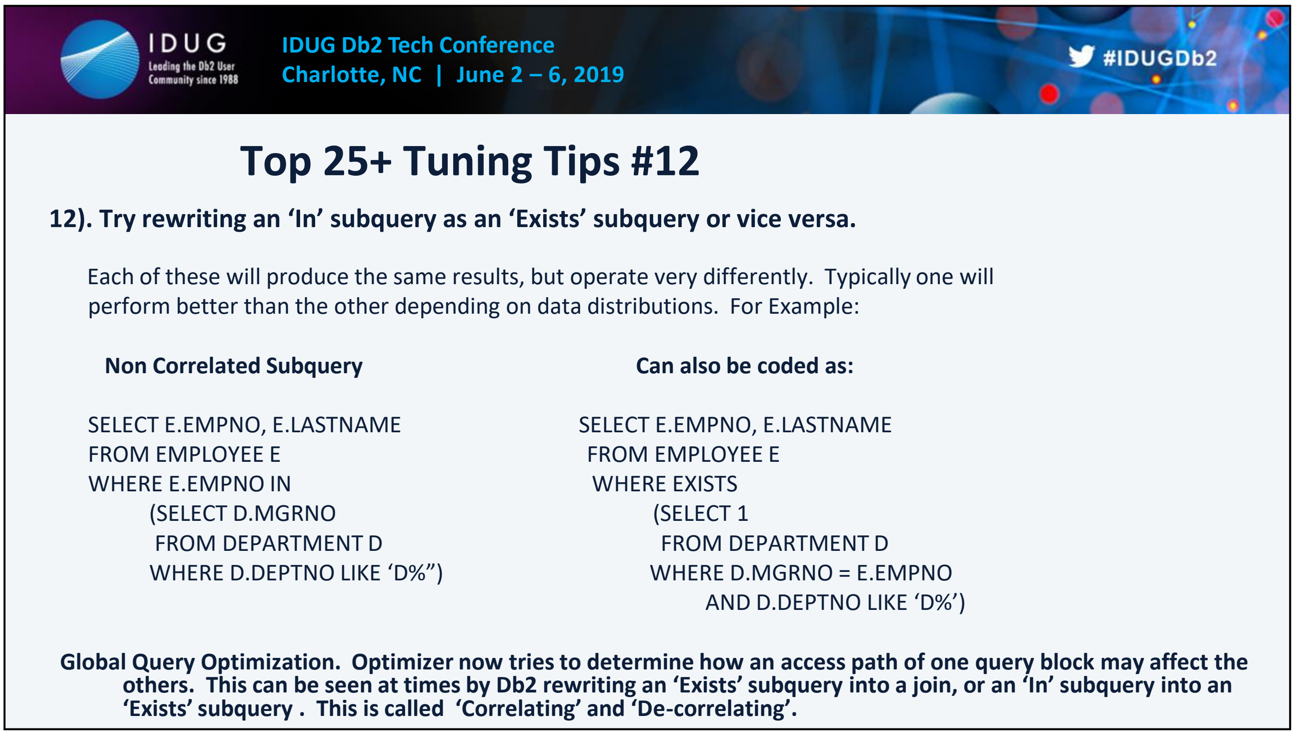
Top 25+ Tuning Tips #11



Many time developers have sorts carried over from copied code

Many times developers do not need to worry about duplicates

Many time developers can rewrite queries to alleviate sorts


IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019
#IDUGDb2

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:

Non Correlated Subquery

```
SELECT E.EMPNO, E.LASTNAME
FROM EMPLOYEE E
WHERE E.EMPNO IN
        (SELECT D.MGRNO
         FROM DEPARTMENT D
        WHERE D.DEPTNO LIKE 'D%")
```

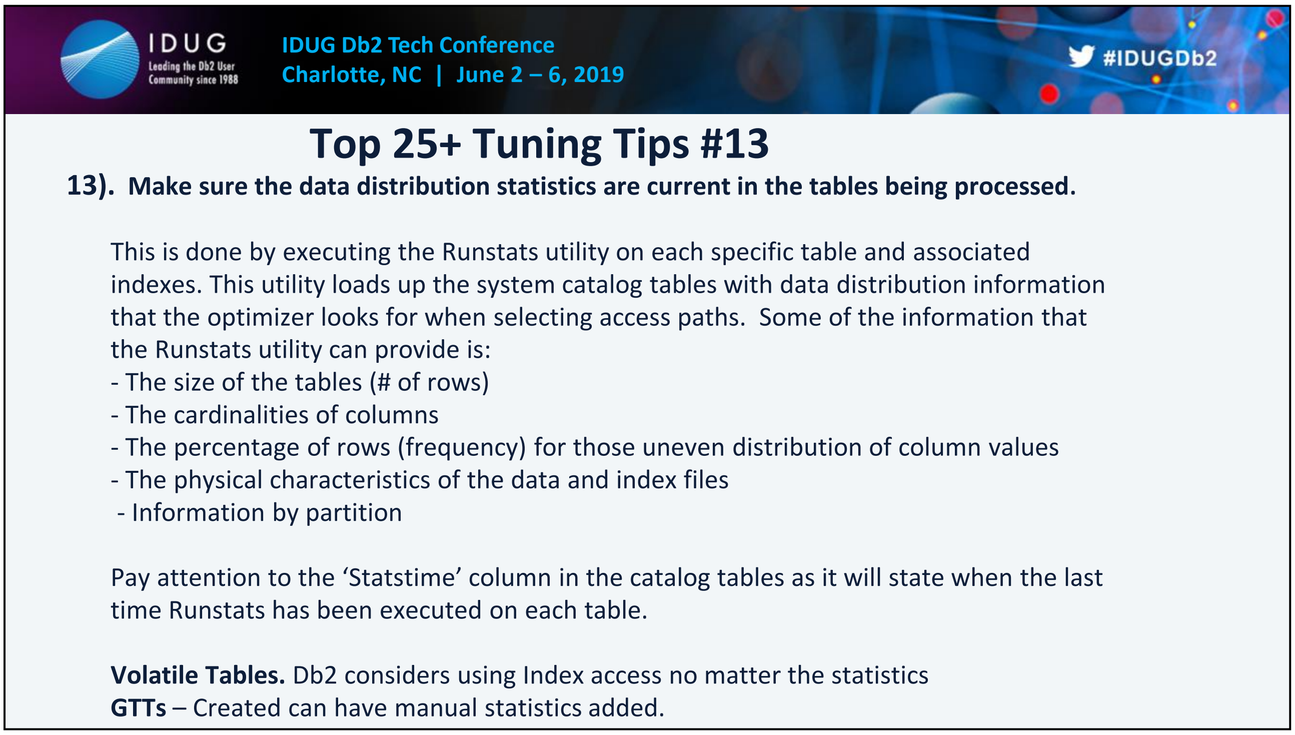
Can also be coded as:

```
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%')
```

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 . This is called 'Correlating' and 'De-correlating'.

This is better handled in V9 with Global Query Optimization.
Prior to V9, Db2 breaks this query into two parts: the subquery and the outer query.
Each of these parts is optimized independently. The access path for the subquery does not take into account the different ways in which the table in the outer query may be accessed and vice versa.
Global query optimization allows Db2 to optimize a query as a whole rather than as independent parts. This is accomplished by allowing Db2 to:
- Consider the effect of one query block on another
- Consider reordering query blocks

V9 – The optimizer takes into consideration correlated, non correlated, and join



IDUG Leading the Db2 User Community since 1988

IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

Top 25+ Tuning Tips #13

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.

Volatile Tables. Db2 considers using Index access no matter the statistics
GTTs – Created can have manual statistics added.

1) FREQVAL Statistics are important for any columns containing uneven distribution of data values. For example some tables may contain a status code column containing multiple values. If any of the values contains a high or low percentage of rows In the table, then it should have FREQVAL statistics run on that column.

2) Statistics are typically up to date in production, but many time are behind or even reset) in test environments.

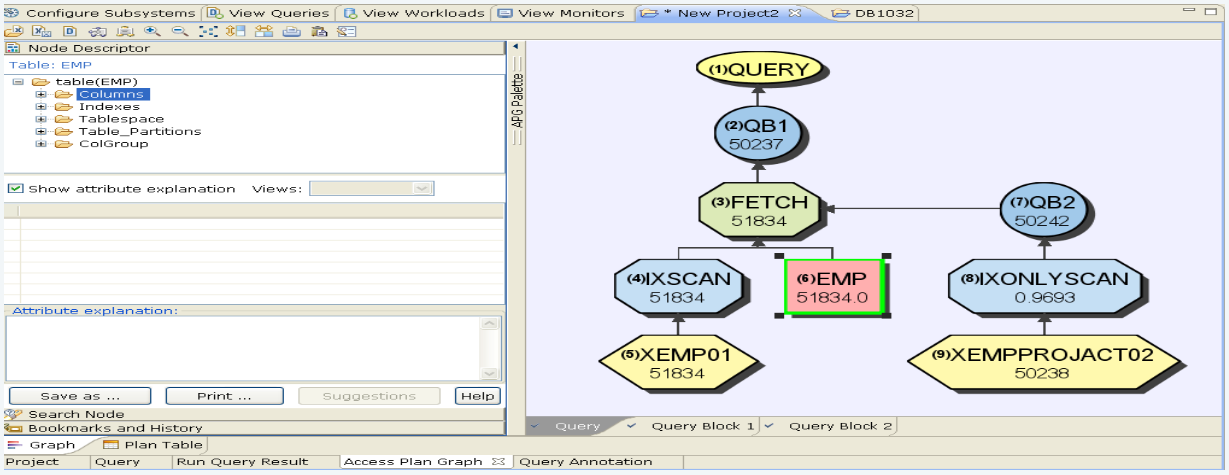
3) Volatile tables are always an issue. Statistics only reflect a point in time.

By declaring the table volatile, the optimizer will consider using index scan rather than table scan. The access plans that use declared volatile tables will not depend on the existing statistics for that table.

4) By creating a Global Temporary Table, manual statistics can then be added to the catalog tables for the average number of rows, average cardinalities, etc…

Top 25+ Tuning Tips #13

13). Make sure the data distribution statistics are current in the tables being processed.

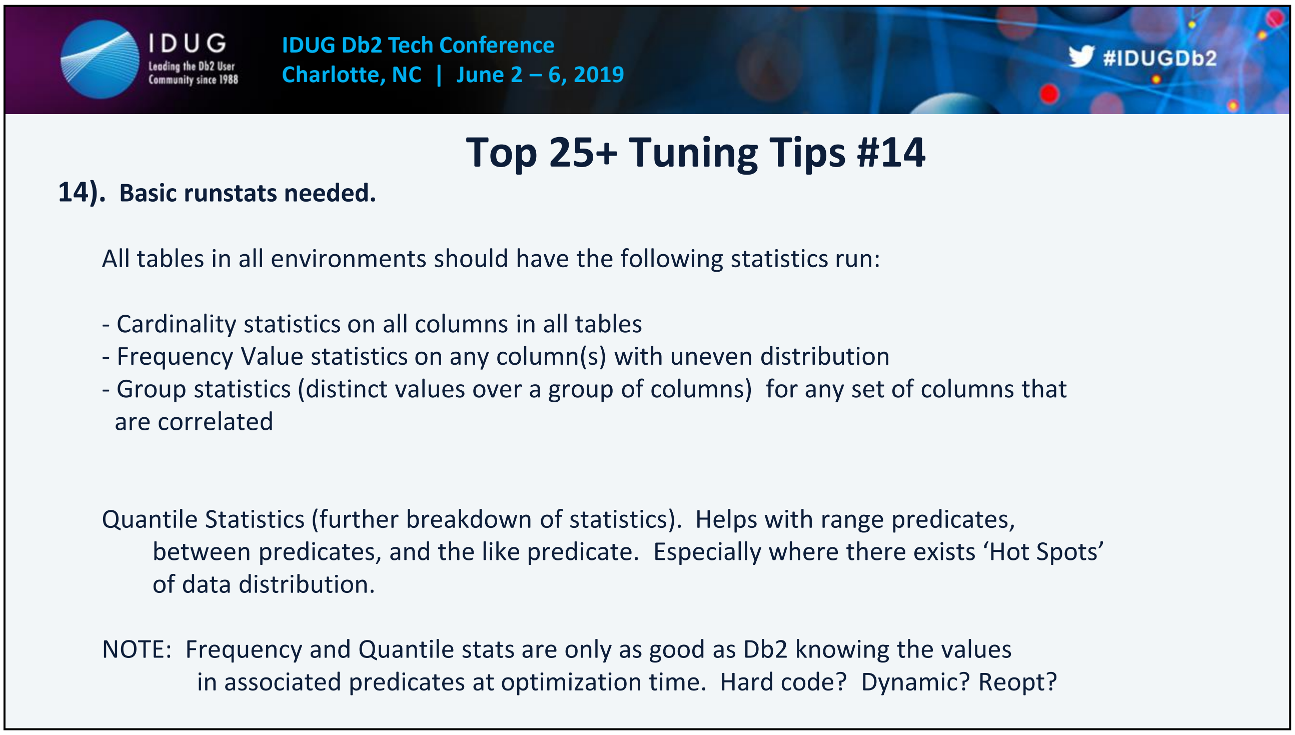


1) FREQVAL Statistics are important for any columns containing uneven distribution of data values. For example some tables may contain a status code column containing multiple values. If any of the values contains a high or low percentage of rows In the table, then it should have FREQVAL statistics run on that column.

2) Statistics are typically up to date in production, but many time are behind or even reset) in test environments.

3) Volatile tables are always an issue. Statistics only reflect a point in time.

By declaring the table volatile, the optimizer will consider using index scan rather than table scan. The access plans that use declared volatile tables will not depend on the existing statistics for that table.



IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

Top 25+ Tuning Tips #14

14). Basic runstats needed.

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

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

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.

NOTE: Frequency and Quantile stats are only as good as Db2 knowing the values in associated predicates at optimization time. Hard code? Dynamic? Reopt?

1) Correlate columns:

```
SELECT COUNT (DISTINCT CITY)   FROM TABLE1;    = 3
SELECT COUNT (DISTINCT STATE)  FROM TABLE1;    = 4      3*4 = 12

SELECT COUNT (*) FROM
  (SELECT DISTINCT CITY, STATE
   FROM TABLE1) AS X;                          =  4
```

*** Since 4 < 12, the columns CITY and STATE are said to be correlated.

2) Quantile Statistics

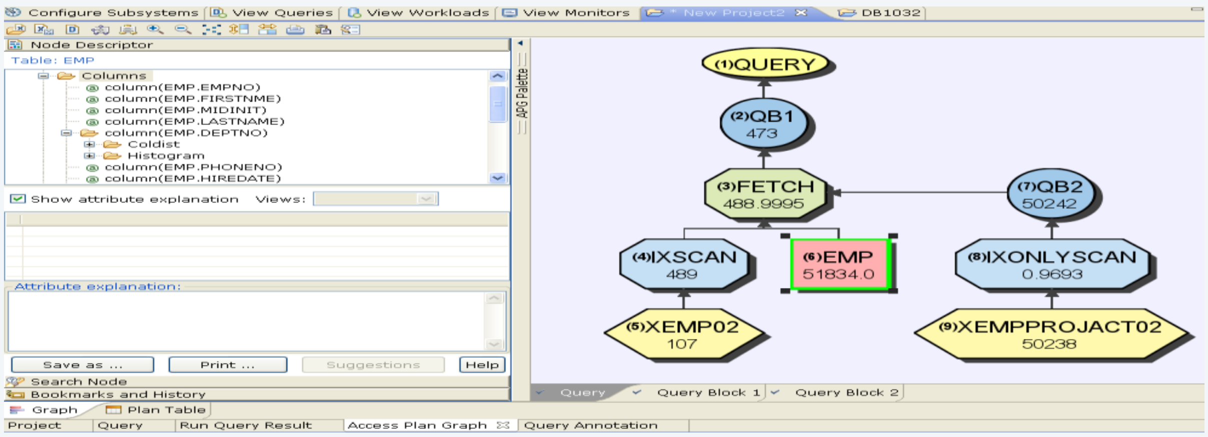
```
RUNSTATS INDEX("THEMIS81"."XEMP02"
HISTOGRAM NUMCOLS 1 NUMQUANTILES 20)
SHRLEVEL CHANGE REPORT YES
```

IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

Top 25+ Tuning Tips #14

14). Basic runstats needed.


Node Descriptor
Table: EMP
Columns
column(EMP.EMPNO)
column(EMP.FIRSTNME)
column(EMP.MIDINIT)
column(EMP.LASTNAME)
column(EMP.DEPTNO)
Coldist
Histogram
column(EMP.PHONENO)
column(EMP.HIREDATE)
Show attribute explanation Views:
Attribute explanation:

1) Correlate columns:

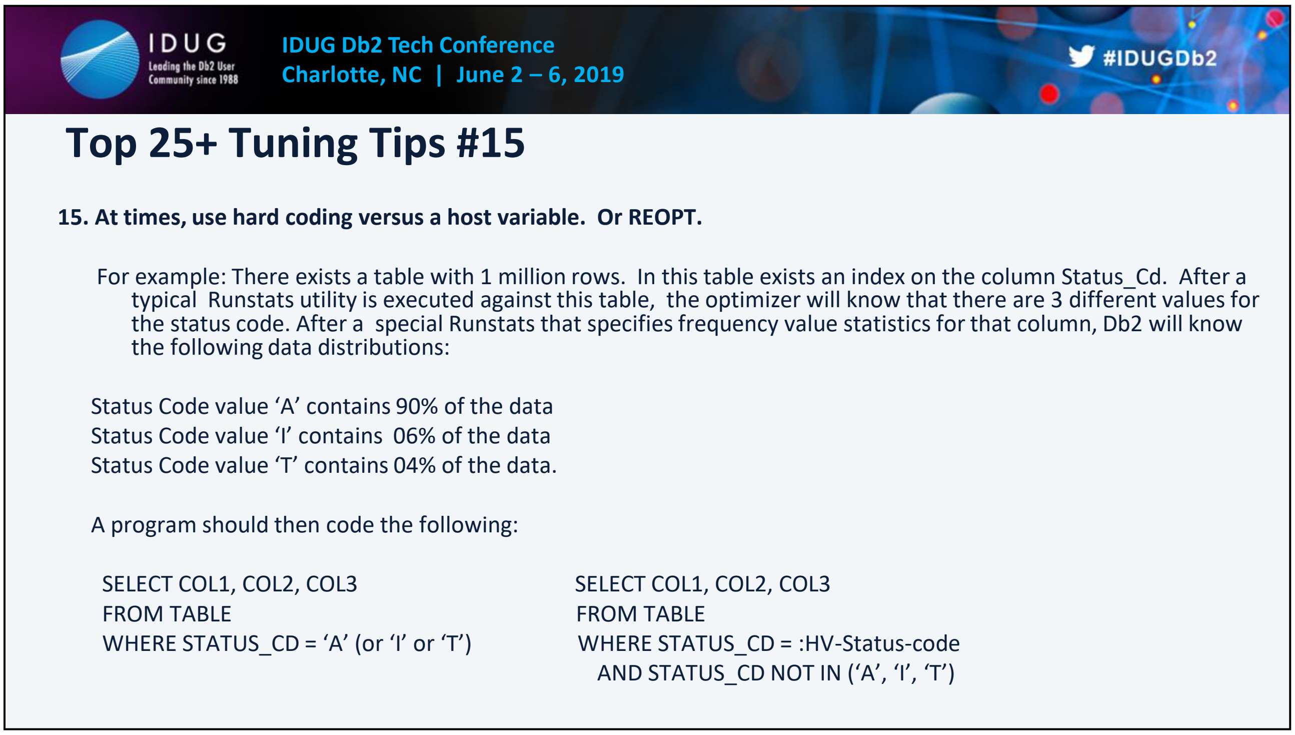
```
SELECT COUNT (DISTINCT CITY)    FROM TABLE1;    = 3
SELECT COUNT (DISTINCT STATE)  FROM TABLE1;    = 4       3*4 = 12

SELECT COUNT (*) FROM
  (SELECT DISTINCT CITY, STATE
   FROM TABLE1) AS X;                           =  4
```

*** Since 4 < 12, the columns CITY and STATE are said to be correlated.

2) Quantile Statistics

```
RUNSTATS INDEX("THEMIS81"."XEMP02"
HISTOGRAM NUMCOLS 1 NUMQUANTILES 20)
SHRLEVEL CHANGE REPORT YES
```



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_Cd. 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 90% of the data
Status Code value ‘I’ contains 06% of the data
Status Code value ‘T’ contains 04% of the data.

A program should then code the following:

```
SELECT COL1, COL2, COL3
FROM TABLE
WHERE STATUS_CD = ‘A’ (or ‘I’ or ‘T’)
```

```
SELECT COL1, COL2, COL3
FROM TABLE
WHERE STATUS_CD = :HV-Status-code
   AND STATUS_CD NOT IN (‘A’, ‘I’, ‘T’)
```

Static Bound Packages – REOPT(NONE)   REOPT(ALWAYS) or VARS

Dynamic Packages   - REOPT(ONCE)   REOPT(AUTO)

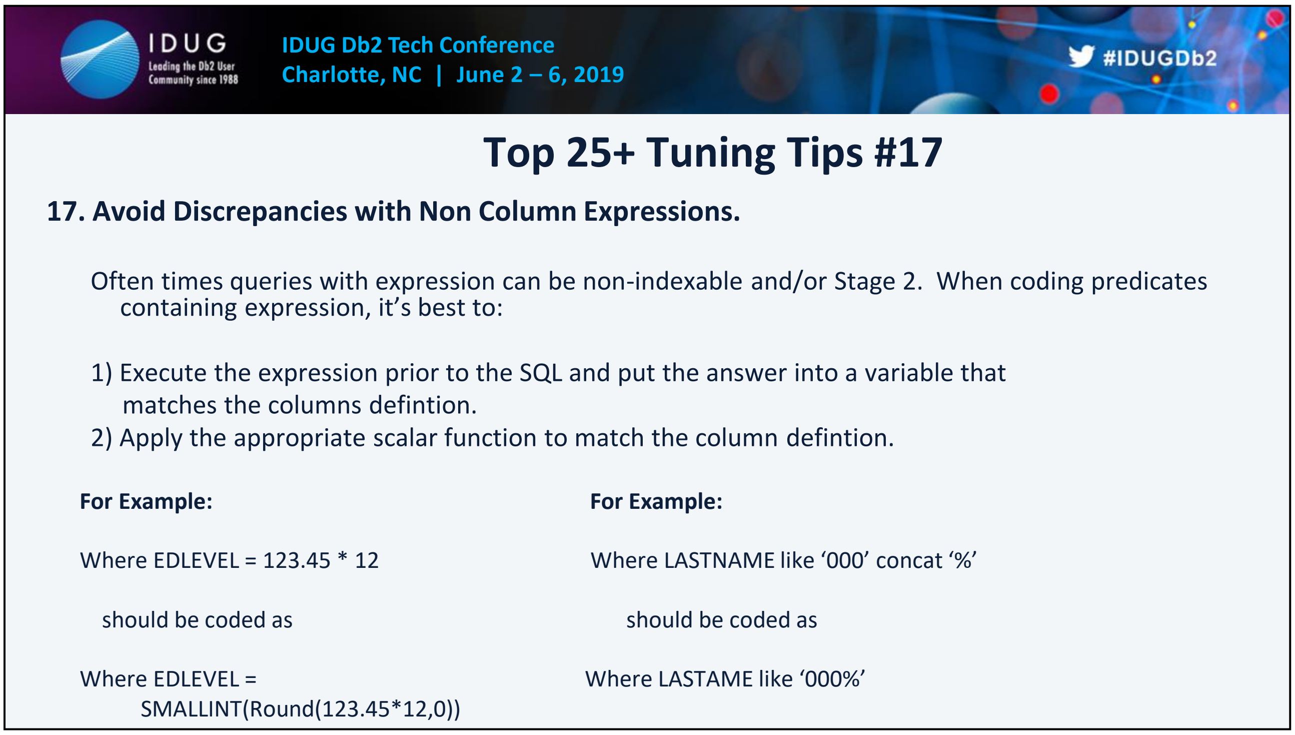
Top 25+ Tuning Tips #16

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%')
```


IDUG Leading the Db2 User Community since 1988

IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

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 defintion.
2) Apply the appropriate scalar function to match the column defintion.

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 '%'

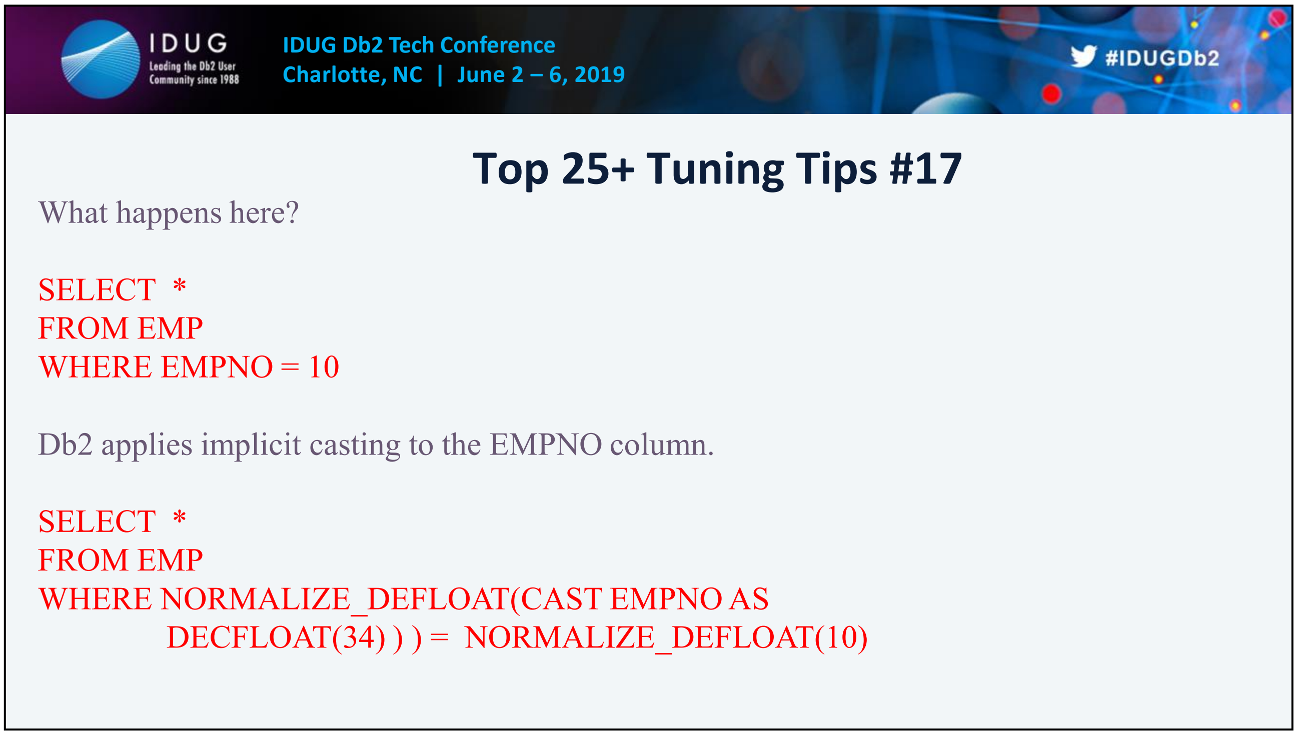
should be coded as

Where LASTAME like '000%'

Note: This comparison of a column to a host variable of a different definition has been improved a lot in Db2. Older versions if the data types and lengths of operands did not match, the predicate was automatically a Stage 2 predicate. As of V8 if the data types and operands do not match, but are within the same data type category (char, integers, decimals, etc.) the predicates can be processed as Stage 1 and some may be indexable. This was to help other languages like C and Java. C does not have a decimal data type, but needs to access data from Db2 with decimal columns. Java does not have fixed length character string data types, only variable character.

Still…. It's a good habit for developers to have variables that match the column definitions. Mostly due to any confusion of logic:

For example: SELECT * FROM EMP WHERE EDLEVEL = 12.25 EDLEVEL defined as a smallint? Will this return any rows?



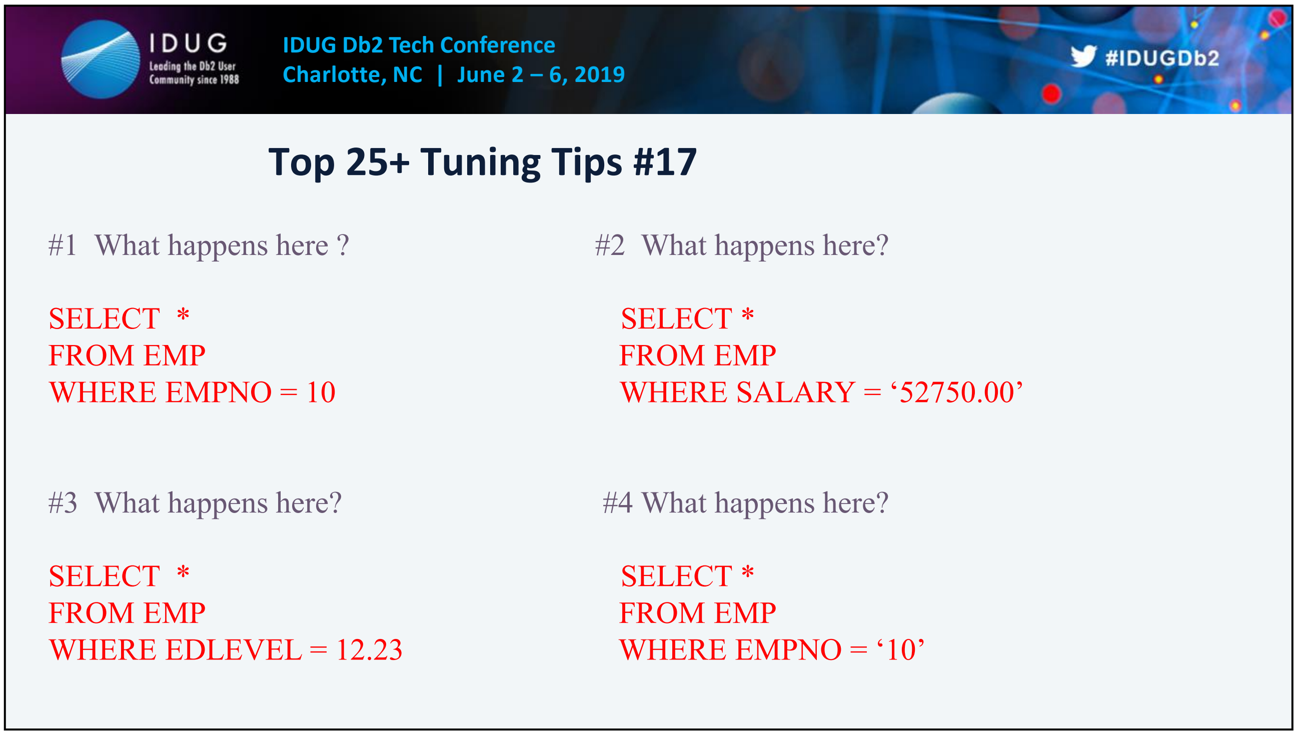
Top 25+ Tuning Tips #17

What happens here?

```
SELECT  *
FROM EMP
WHERE EMPNO = 10
```

Db2 applies implicit casting to the EMPNO column.

```
SELECT  *
FROM EMP
WHERE NORMALIZE_DEFLOAT(CAST EMPNO AS
              DECFLOAT(34) ) ) =  NORMALIZE_DEFLOAT(10)
```



IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

Top 25+ Tuning Tips #17

#1 What happens here ?

```
SELECT  *
FROM EMP
WHERE EMPNO = 10
```

#2 What happens here?

```
SELECT *
FROM EMP
WHERE SALARY = ‘52750.00’
```

#3 What happens here?

```
SELECT  *
FROM EMP
WHERE EDLEVEL = 12.23
```

#4 What happens here?

```
SELECT *
FROM EMP
WHERE EMPNO = ‘10’
```

Getting the correct data precision to match a column’s data type is more logic error than performance issues anymore. Db2 Covers incorrect precision in hard coded values and host variables better now as most all stage 1 and indexable?

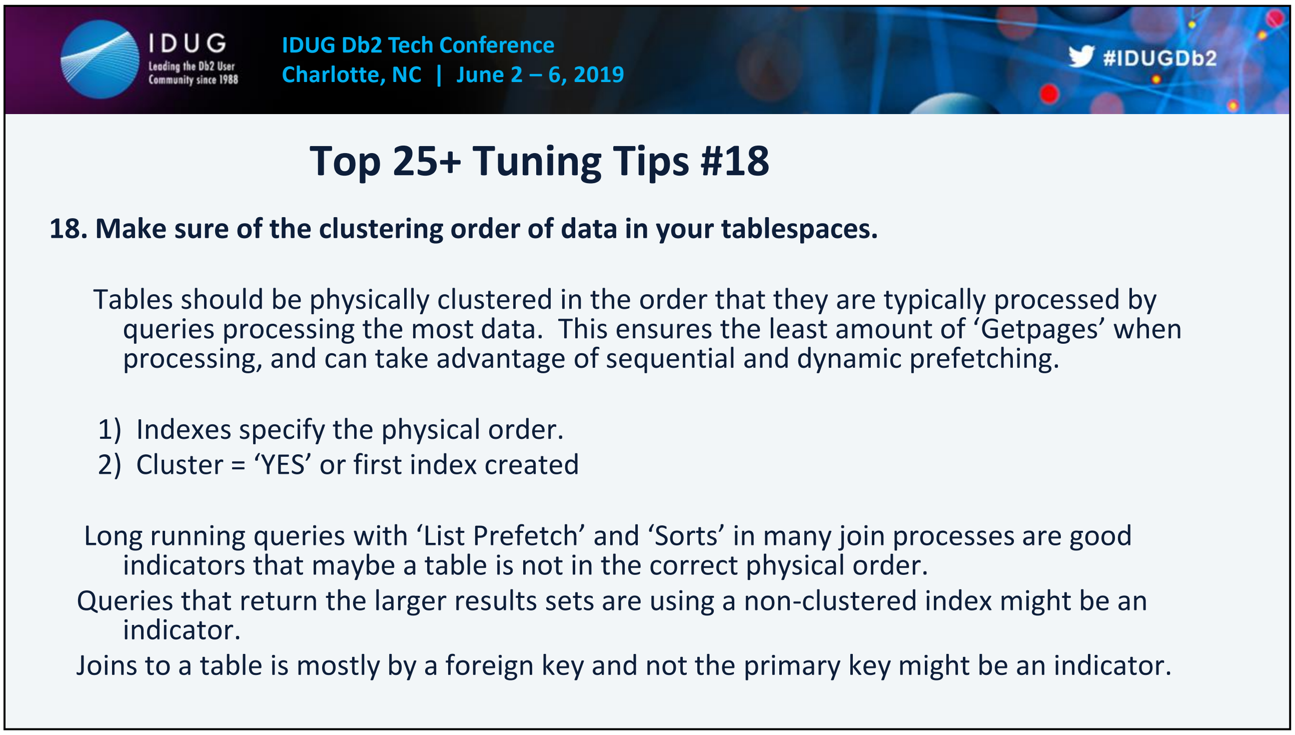
But do most developers know logically what happens when the comparison is of two different data types?

1) Implicit Casting takes place

2) Db2 will find those rows = 52750.00

3) No rows are found. Edlevel defined as SMALLINT.

4) Db2 looks for any EMPNO that equals ’10   ‘ (the number 10 followed by 4 spaces).



Top 25+ Tuning Tips #18

18. 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, and can take advantage of sequential and dynamic prefetching.

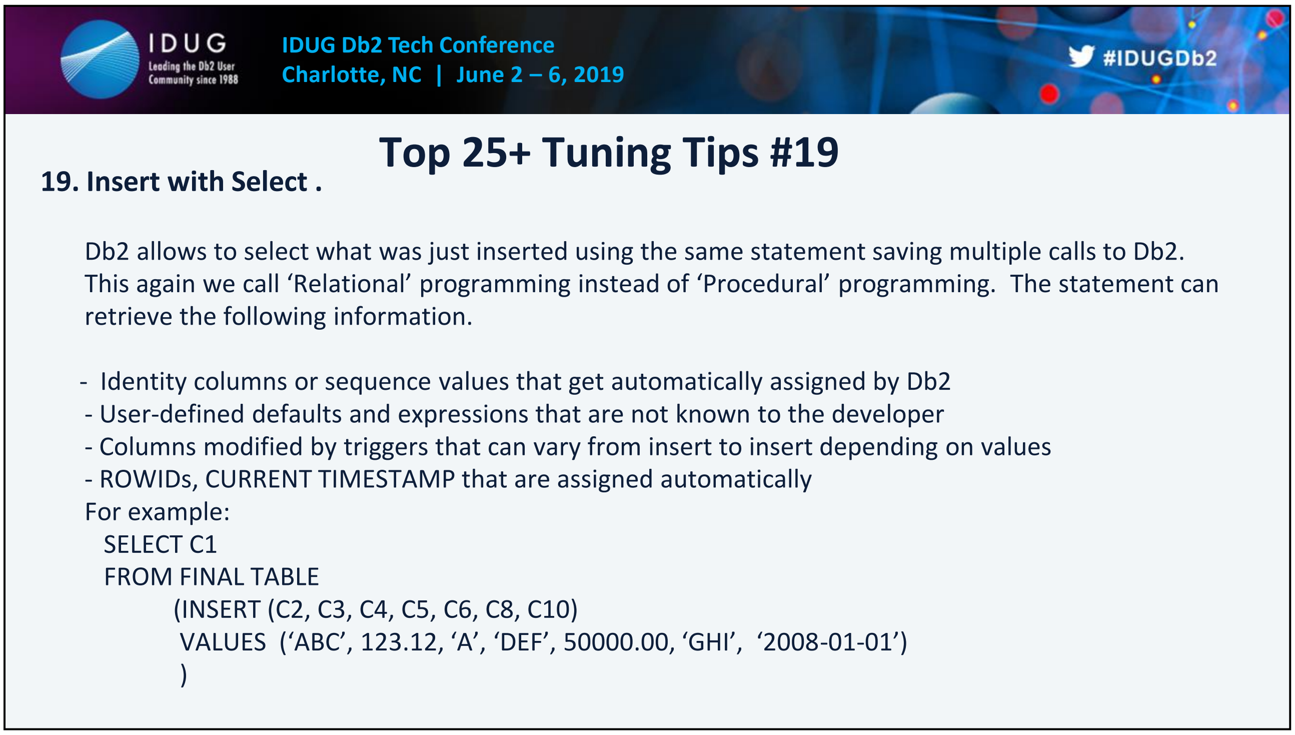
1) Indexes specify the physical order.
2) Cluster = 'YES' or first index created

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.

Queries that return the larger results sets are using a non-clustered index might be an indicator.

Joins to a table is mostly by a foreign key and not the primary key might be an indicator.

Getting the correct physical clustering can save sorts, and I/Os.



IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

Top 25+ Tuning Tips #19

19. Insert with Select .

Db2 allows to select what was just inserted using 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:

```
  SELECT C1
  FROM FINAL TABLE
           (INSERT (C2, C3, C4, C5, C6, C8, C10)
            VALUES  ('ABC', 123.12, 'A', 'DEF', 50000.00, 'GHI',  '2008-01-01')
            )
```

Good for Retrieving:

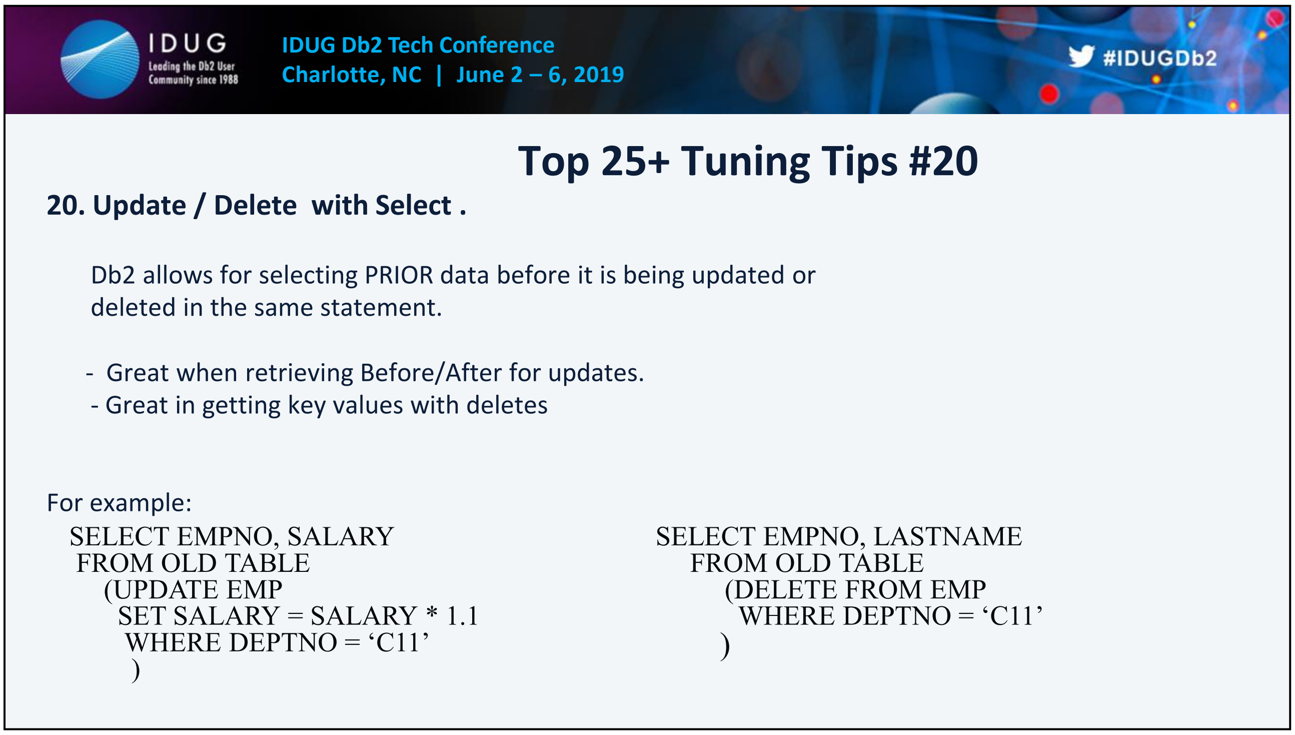
The value of an automatically generated column such as a ROWID or identity column

Any default values for columns

All values for an inserted row, without specifying individual column names

All values that are inserted by a multiple-row INSERT operation

Values that are changed by a BEFORE INSERT trigger


IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019
#IDUGDb2

Top 25+ Tuning Tips #20

20. Update / Delete with Select .

Db2 allows for selecting PRIOR data before it is being updated or deleted in the same statement.

- Great when retrieving Before/After for updates.
- Great in getting key values with deletes

For example:

```
SELECT EMPNO, SALARY
 FROM OLD TABLE
    (UPDATE EMP
      SET SALARY = SALARY * 1.1
       WHERE DEPTNO = ‘C11’
        )
```

```
SELECT EMPNO, LASTNAME
     FROM OLD TABLE
          (DELETE FROM EMP
            WHERE DEPTNO = ‘C11’
          )
```

Good for Retrieving:

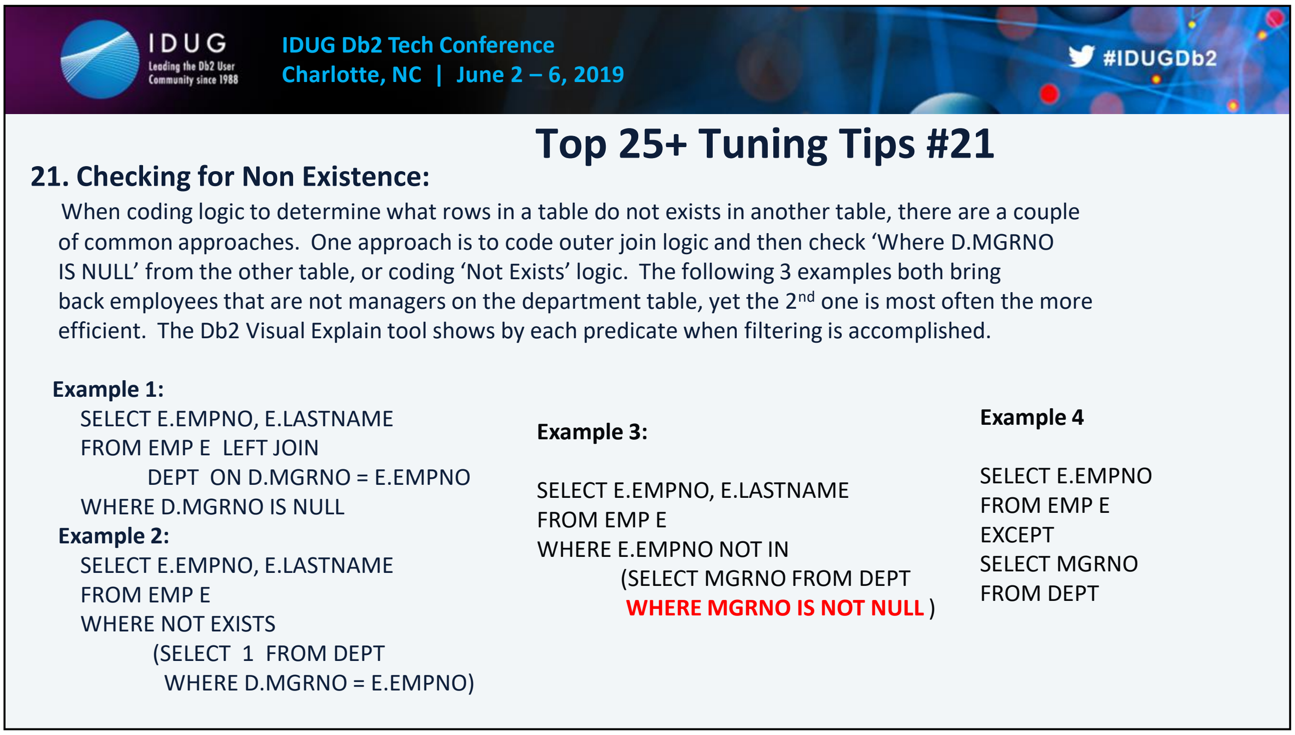
The value of an automatically generated column such as a ROWID or identity column

Any default values for columns

All values for an inserted row, without specifying individual column names

All values that are inserted by a multiple-row INSERT operation

Values that are changed by a BEFORE INSERT trigger



IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019
#IDUGDb2

Top 25+ Tuning Tips #21

21. Checking for Non Existence:

When coding logic to determine what rows in a table do not exists 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 3 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.

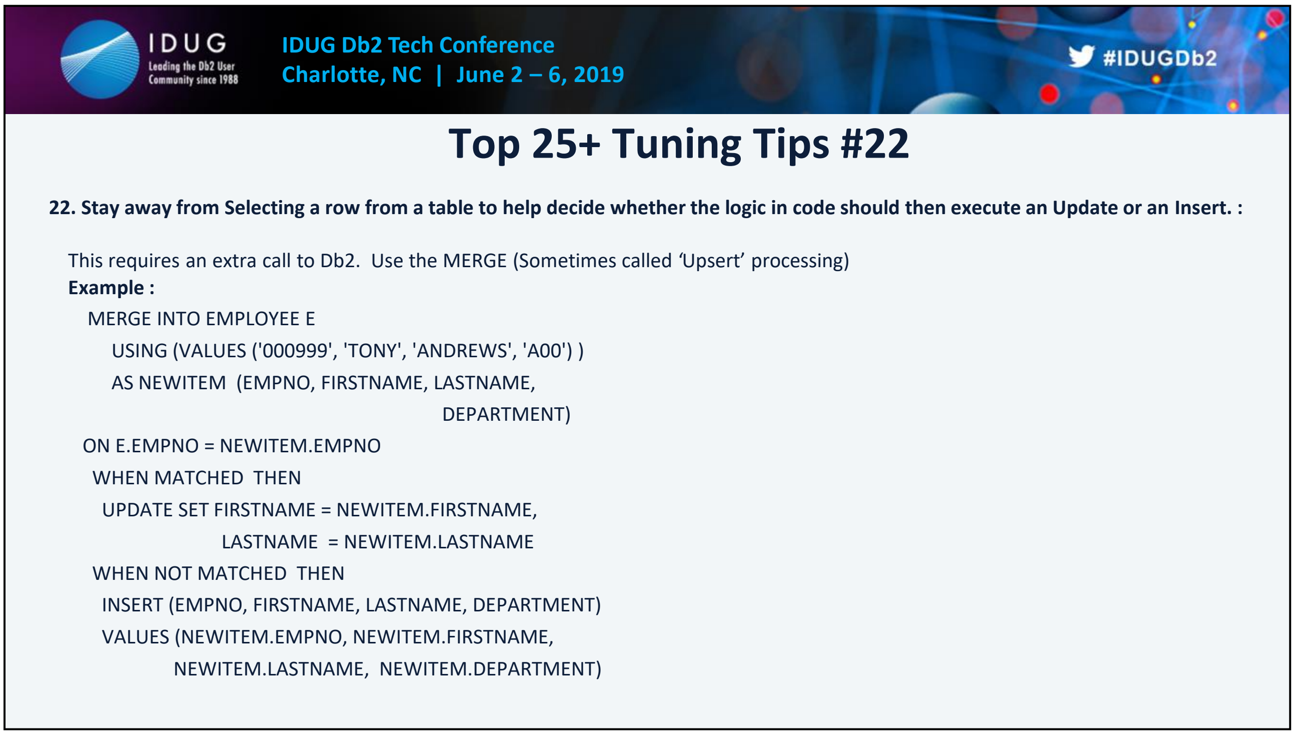
Example 1:
```
SELECT E.EMPNO, E.LASTNAME
FROM EMP E  LEFT JOIN
        DEPT  ON D.MGRNO = E.EMPNO
WHERE D.MGRNO IS NULL
```
Example 2:
```
SELECT E.EMPNO, E.LASTNAME
FROM EMP E
WHERE NOT EXISTS
          (SELECT  1  FROM DEPT
            WHERE D.MGRNO = E.EMPNO)
```
Example 3:
```
SELECT E.EMPNO, E.LASTNAME
FROM EMP E
WHERE E.EMPNO NOT IN
          (SELECT MGRNO FROM DEPT
           WHERE MGRNO IS NOT NULL )
```
Example 4
```
SELECT E.EMPNO
FROM EMP E
EXCEPT
SELECT MGRNO
FROM DEPT
```

4 ways to figure out 'Not Exist' logic

The 'Not In' logic is typically the worst performing, but if it is used and the column being selected in the subquery is a nullable column, then nulls need to be eliminated from the 'Not In' list, or Db2 returns nothing.

```
SELECT E.EMPNO, E.LASTNAME
FROM EMPLOYEE E
WHERE E.EMPNO NOT IN
          (SELECT MGRNO
           FROM DEPARTMENT
            WHERE MGRNO IS NOT NULL)
```

Also can use the SQL NOT EXCEPT logic.



IDUG Leading the Db2 User Community since 1988

IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

Top 25+ Tuning Tips #22

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. Use the MERGE (Sometimes called 'Upsert' processing)

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)
```

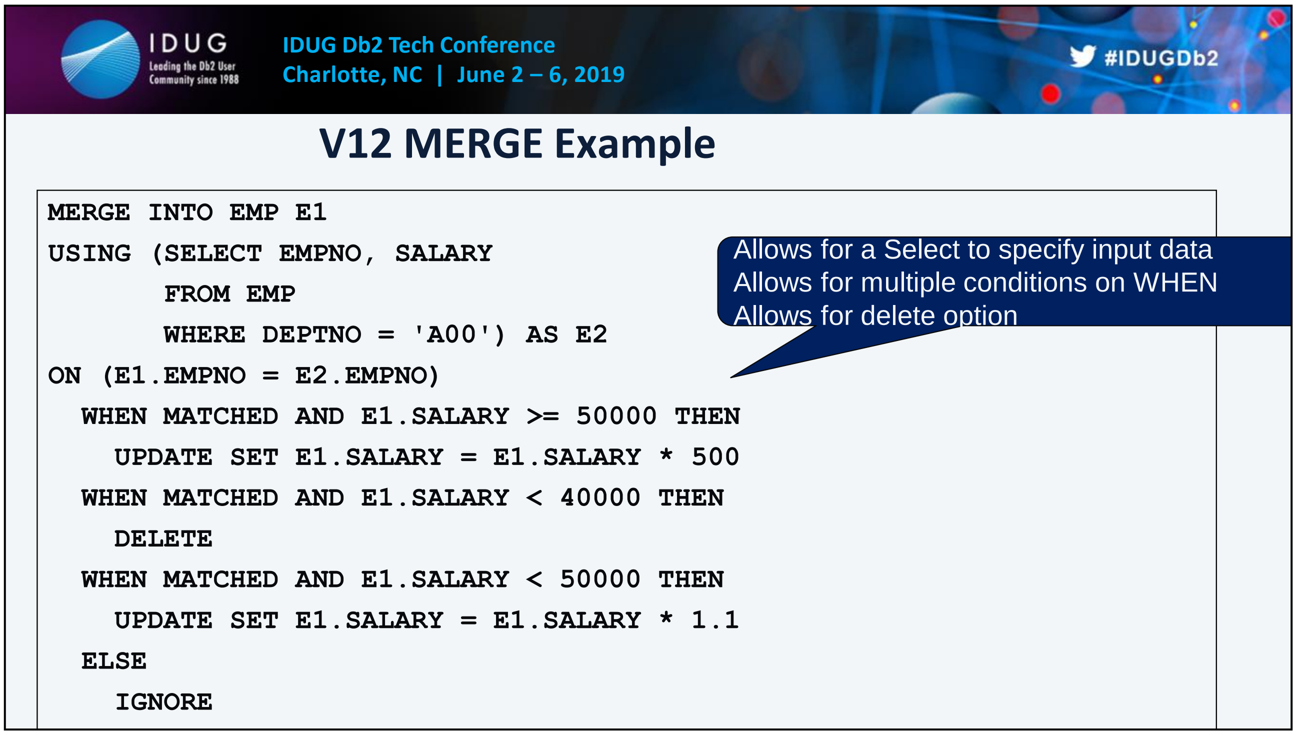
An SQL Merge statement was introduced to better handle this exact situation.

The merge statement specifies to Db2 what to do on a matched condition (execute an Update) or a non matched condition (execute an Insert), handling either condition within the same SQL statement. This is sometimes called an 'Upsert' statement

This can also be performed using Rowsets and Arrays. Testing results have often shown that MERGE is way more efficient than

- SELECT followed by INSERT or UPDATE and INSERT
- INSERT first (if duplicate SQLCODE -803), then UPDATE.

The size of the arrays has not shown much difference as it does in fetching. Executing multiple MERGE's with arrays of 100 or 1000 at a time versus fewer with executions with 10,000 had little differences if the number of executions was not dramatically different. Of course you need to do your own independent testing.

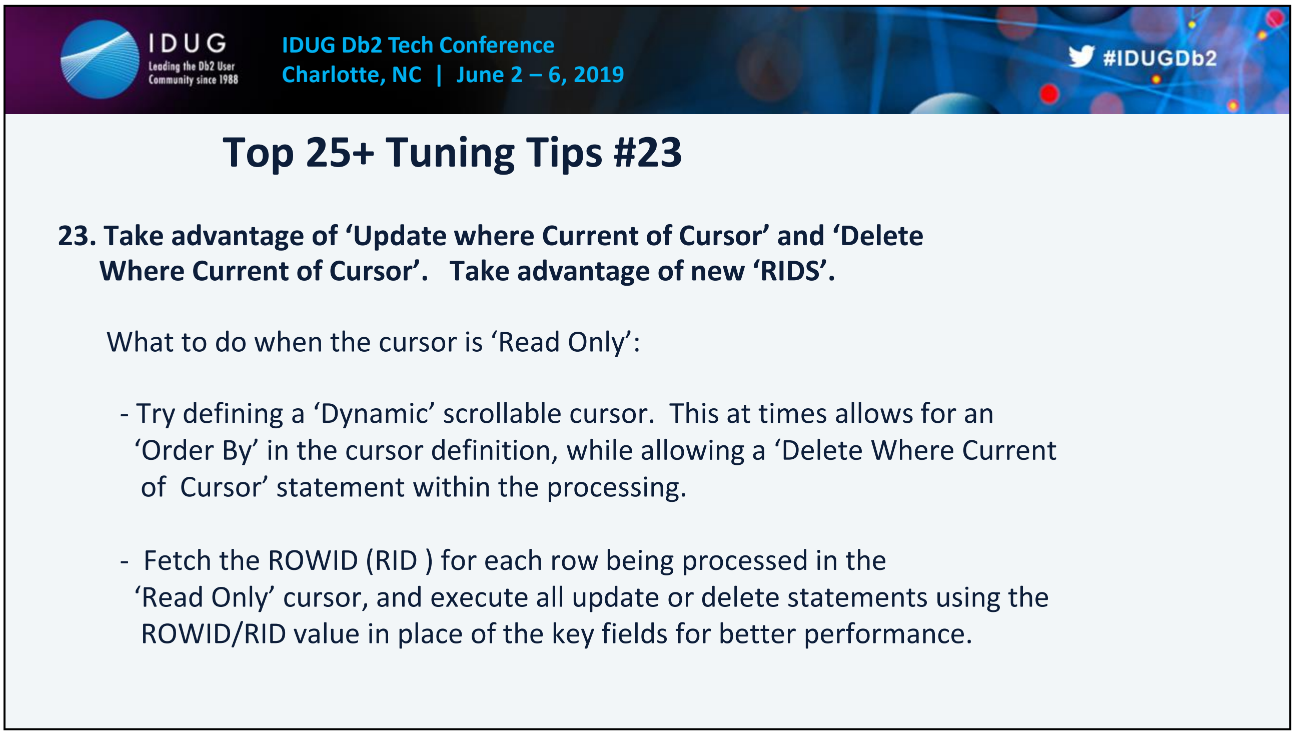

IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019
#IDUGDb2

V12 MERGE Example

```
MERGE INTO EMP E1
USING (SELECT EMPNO, SALARY
       FROM EMP
       WHERE DEPTNO = 'A00') AS E2
ON (E1.EMPNO = E2.EMPNO)
  WHEN MATCHED AND E1.SALARY >= 50000 THEN
    UPDATE SET E1.SALARY = E1.SALARY * 500
  WHEN MATCHED AND E1.SALARY < 40000 THEN
    DELETE
  WHEN MATCHED AND E1.SALARY < 50000 THEN
    UPDATE SET E1.SALARY = E1.SALARY * 1.1
  ELSE
    IGNORE
```

Allows for a Select to specify input data
Allows for multiple conditions on WHEN
Allows for delete option

When checking for multiple conditions, you must code a SELECT for the USING. VALUES does not work.


IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019
#IDUGDb2

Top 25+ Tuning Tips #23

23. Take advantage of 'Update where Current of Cursor' and 'Delete Where Current of Cursor'. Take advantage of new 'RIDS'.

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) 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.

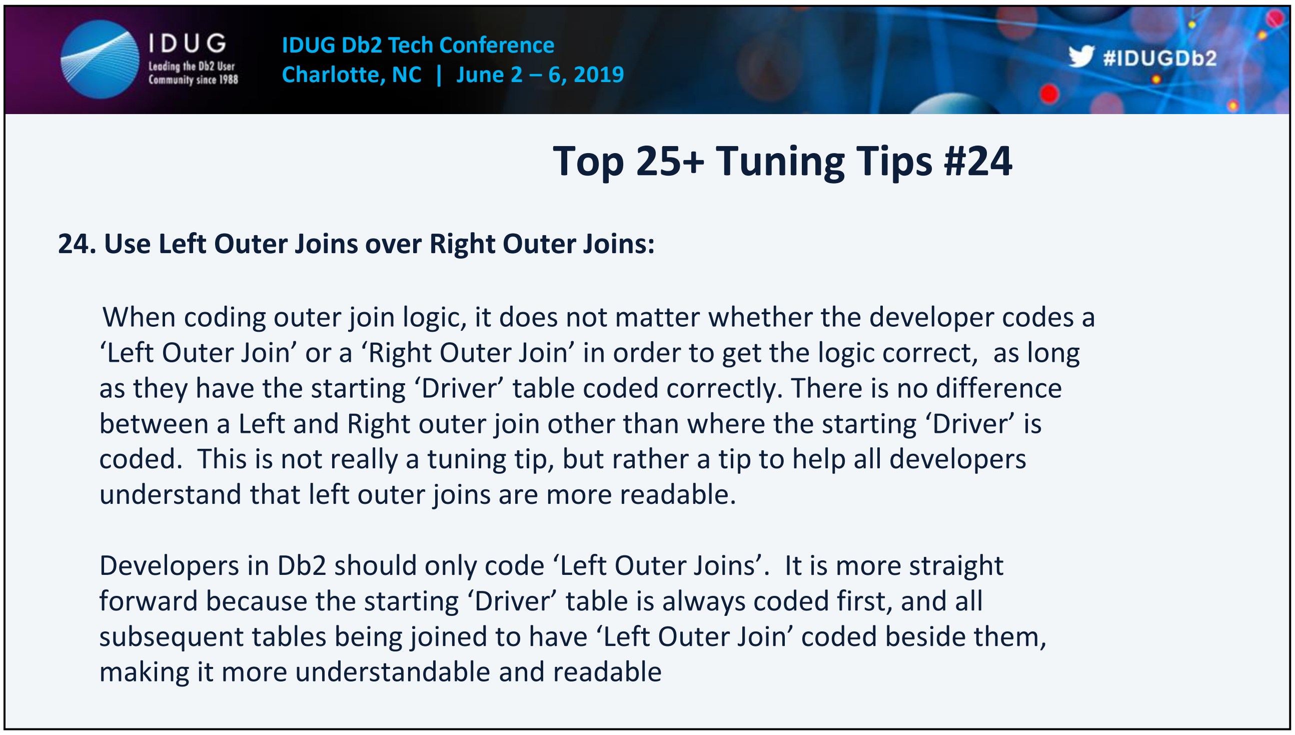
Allows the developer to have a cursor that might be 'Read Only' and still have the option to execute updates 'where current of cursor'

```
ROWID:  Select empno, lastname, RID(EMP)
            From EMP
            Where .....

             Update EMP
             Set Salary = Salary * 1.1
             Where Rid(EMP) = ……
```

The result of the RID function is BIGINIT.

ROWID Datatype = Internal(CHAR 17), External (Char40)



IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

#IDUGDb2

Top 25+ Tuning Tips #24

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 straight forward 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

Db2 optimization always converts right outer joins to left outer joins. See explain output.

Column = Join_Type in the Plan_Table

Top 25+ Tuning Tips #25

25. Predicate rewrites:

1) WHERE value1 BETWEEN COLA and COLB

should be rewritten as

WHERE value1 >= COLA and value 1 <= COLB

Top 25+ Tuning Tips #25

25. Predicate rewrites:

2) WHERE COLA NOT BETWEEN value 1 and value2

should be rewritten as

WHERE COLA < value1 or COLA > value2

3) WHERE HIREDATE + 1 MONTH > value1

should be rewritten as

WHERE HIREDATE > value1 – 1 MONTH

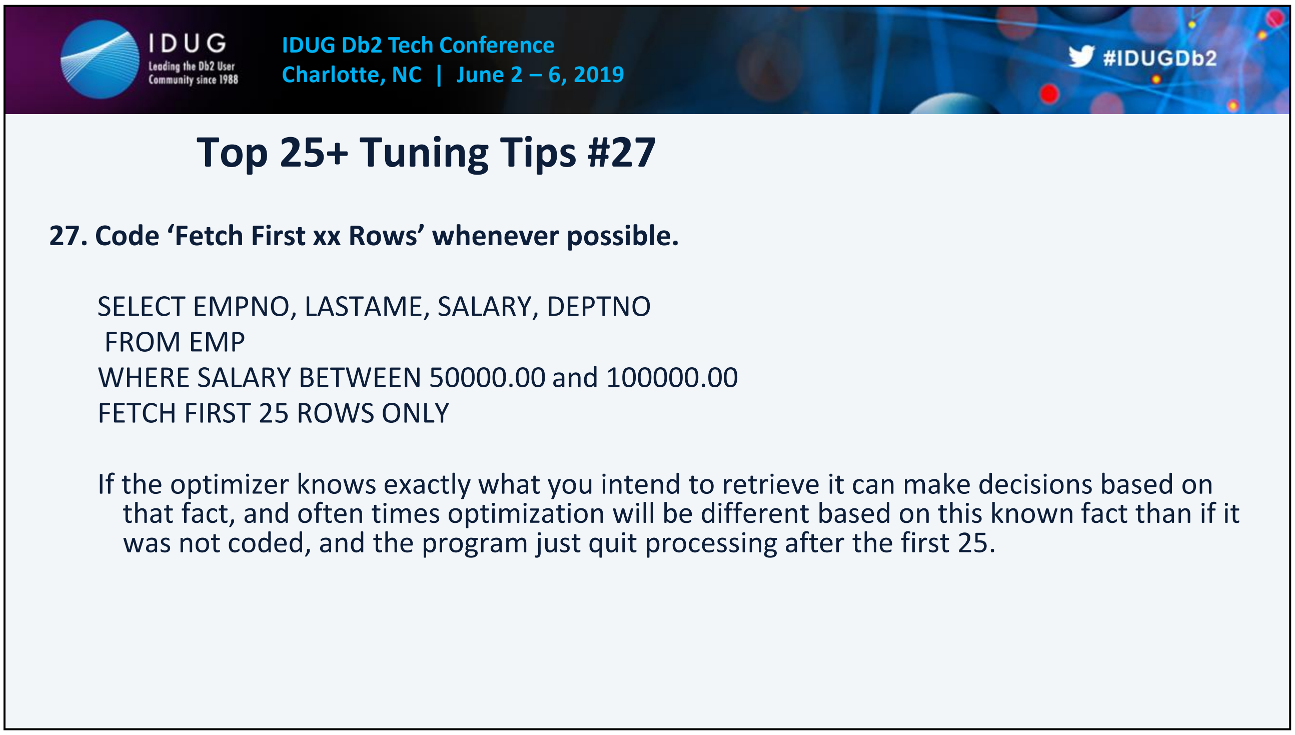
Top 25+ Tuning Tips #26

26. 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 'ORER BY EMPNO' because it is a unique column.


IDUG Db2 Tech Conference
Charlotte, NC | June 2 – 6, 2019

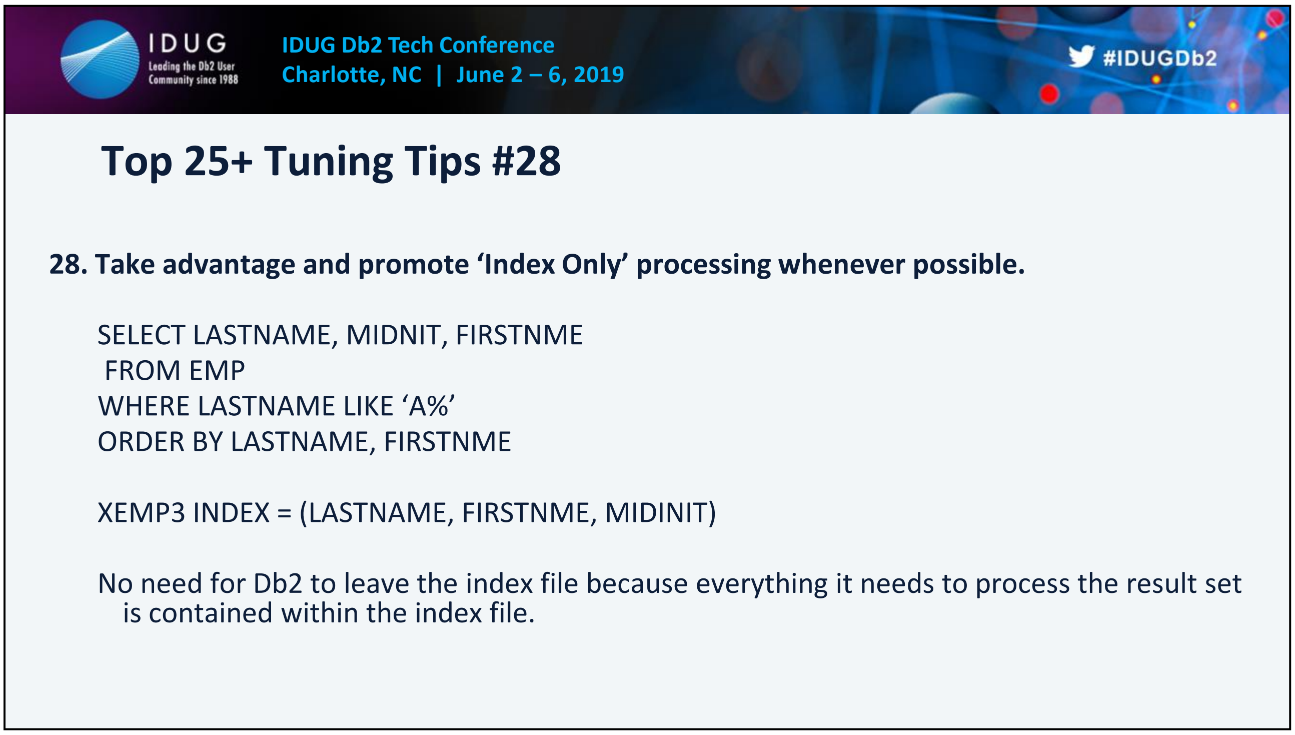
#IDUGDb2

Top 25+ Tuning Tips #27

27. Code 'Fetch First xx Rows' whenever possible.

```
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.



Top 25+ Tuning Tips #28

28. Take advantage and promote 'Index Only' processing whenever possible.

```
SELECT LASTNAME, MIDNIT, 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.

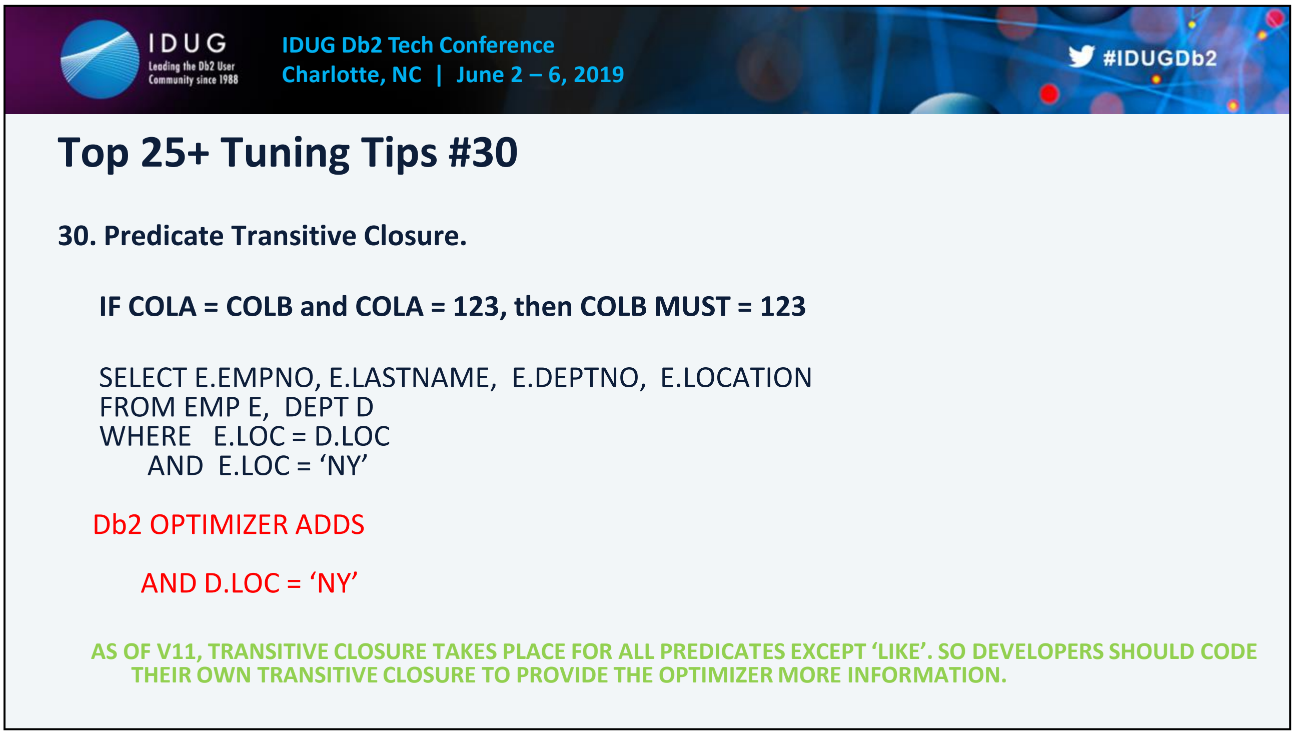
Often times, columns are added to an index specifically to get 'Index Only' processing.

Top 25+ Tuning Tips #29

29. 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.



Top 25+ Tuning Tips #30

30. 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’
```

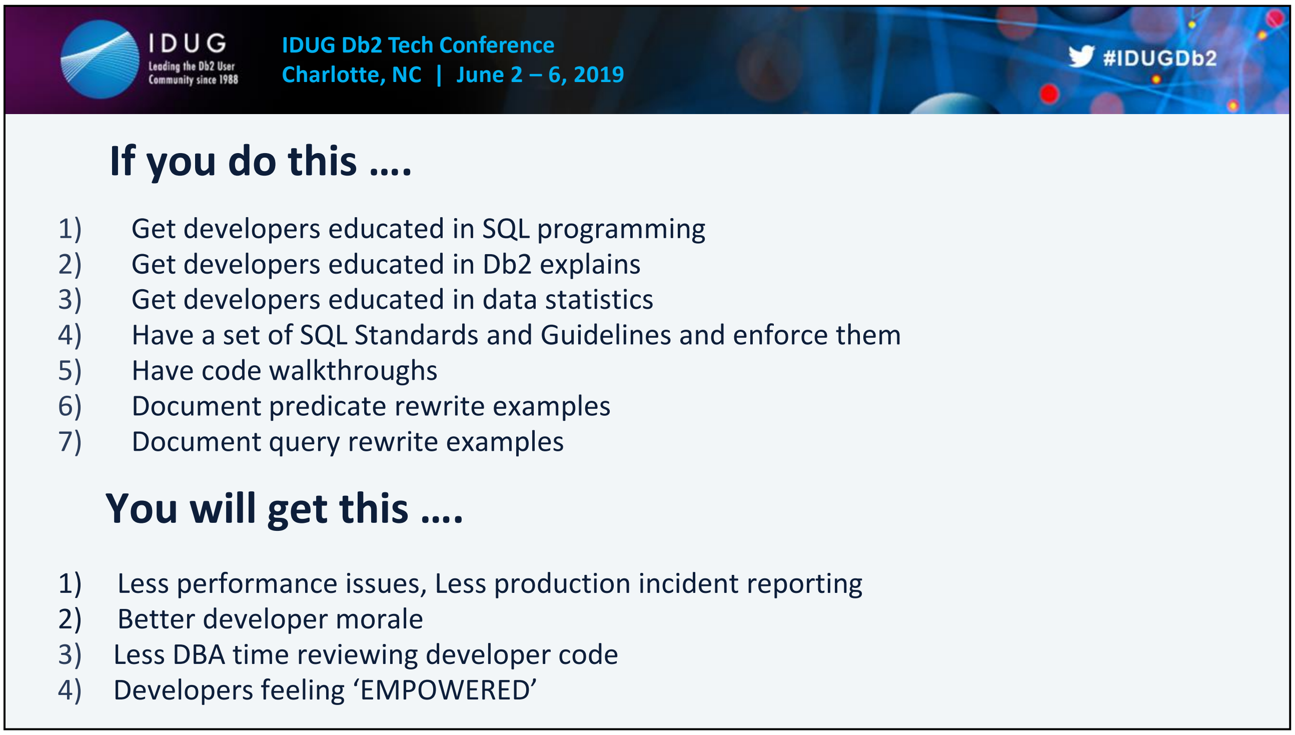
Db2 OPTIMIZER ADDS

```
AND D.LOC = ‘NY’
```

AS OF V11, TRANSITIVE CLOSURE TAKES PLACE FOR ALL PREDICATES EXCEPT ‘LIKE’. SO DEVELOPERS SHOULD CODE THEIR OWN TRANSITIVE CLOSURE TO PROVIDE THE OPTIMIZER MORE INFORMATION.

Top Steps to Tuning a Query

1) Check every predicate. Are they indexable and Stage 1?
2) Can any of the predicates be rewritten differently? Can the query be rewritten differently?
3) Is there a 'Distinct' or 'Group By'? Is it needed? Can it be rewritten?
4) Are there subqueries? Rewrite 'In' as 'Exists' and vice versa.
5) Check Db2 statistics on every table and every column involved.
6) 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?
7) Check the Db2 Explain. Are there any table scans? Any sorts?
8) Check the Db2 Explain. Index – What are the matching columns? Index Scan?
9) Are there any correlated subqueries? Can they be Index-Only? How often executed?
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? Then re-written dynamic



If you do this ….

1) Get developers educated in SQL programming
2) Get developers educated in Db2 explains
3) Get developers educated in data statistics
4) Have a set of SQL Standards and Guidelines and enforce them
5) Have code walkthroughs
6) Document predicate rewrite examples
7) Document query rewrite examples

You will get this ….

1) Less performance issues, Less production incident reporting
2) Better developer morale
3) Less DBA time reviewing developer code
4) Developers feeling 'EMPOWERED'

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

- 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

Tony Andrews
Themis Inc.
tandrews@themisinc.com

Session code: E04

IDUG
Leading the Db2 User Community since 1988

Please fill out your session evaluation before leaving!