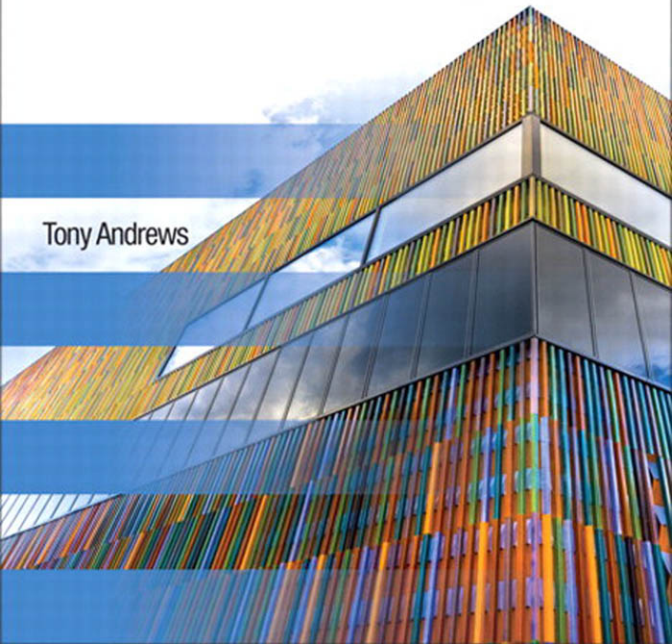
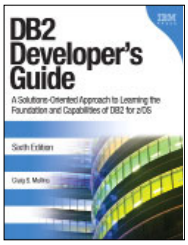
IBM PRESS

DB2 SQL Tuning Tips for z/OS Developers



Tony Andrews

Related Books of Interest



DB2 Developer's Guide

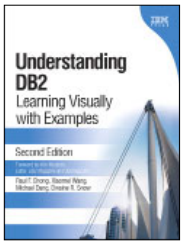
By Craig Mullins

ISBN: 0-13-283642-4

The field's #1 go-to source for on-the-job information on programming and administering DB2 on IBM z/OS mainframes.

Now, three-time IBM Information Champion Craig S. Mullins has thoroughly updated this classic for the newest versions of DB2 for z/OS: DB2 V9 andV10.

This Sixth Edition builds on the unique approach that has made previous editions so valuable. It brings together condensed, easy-to-read coverage of all essential topics: information otherwise scattered through dozens of IBM and third-party documents. Throughout, Mullins offers focused drill-down on the key details DB2 developers need to succeed, with expert, field-tested implementation advice and realistic examples.



Understanding DB2

Learning Visually with Examples, Second Edition

By Raul F. Chong, Xiaomei Wang, Michael Dang, and Dwaine R. Snow

ISBN: 0-13-300704-9

IBM® DB2® 9 and DB2 9.5 provide breakthrough capabilities for providing Information on Demand, implementing Web services and Service Oriented

Architecture, and streamlining information management. *Understanding DB2: Learning Visually with Examples, Second Edition*, is the easiest way to master the latest versions of DB2 and apply their full power to your business challenges.

Written by four IBM DB2 experts, this book introduces key concepts with dozens of examples drawn from the authors' experiences working with DB2 in enterprise environments. Thoroughly updated for DB2 9.5, it covers new innovations ranging from manageability to performance and XML support to API integration. Each concept is presented with easy-to-understand screenshots, diagrams, charts, and tables. This book is for everyone who works with DB2: database administrators, system administrators, developers, and consultants. With hundreds of well-designed review questions and answers, it will also help professionals prepare for the IBM DB2 Certification Exams 730, 731, or 736.

Listen to the author's podcast at:
ibmpressbooks.com/podcasts

Sign up for the monthly IBM Press newsletter at
ibmpressbooks/newsletters

Read Only cursor is when the cursor definition has an Order By statement. But with the advantage of the dynamic scrollable cursors, Delete Where Current of Cursor can still be executed when coded with an Order By statement. (See tuning tip #43, later in this chapter.)

Some locking issues when a cursor is specified with For Update Of, affect concurrency with the data involved. But if locking is not an issue, declaring your cursor to take advantage of Update Where Current of Cursor or Delete Where Current of Cursor will enable faster processing.

36. When Using Cursors, Use ROWSET Positioning and Fetching Using Multiple-Row Fetch, Multiple-Row Update, and Multiple-Row Insert

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. Having the ability to fetch, update, or insert more than one row at a time reduces network traffic and other related costs associated with each call to DB2.

When fetching data from a cursor, developers can code to fetch, say, 100 rows at a time with one FETCH statement by fetching the data into an array for each host variable. Note that 100 rows at a time seems to be the most efficient threshold; more or less than 100 has diminishing returns in terms of efficiency. The recommendation is to start with 100 row fetches, inserts, or updates and then test other numbers. Doing this can reduce runtime an average of 35%. Consult the IBM DB2 manuals for further detail and coding examples.

For local applications, using these multiple-row statements results in fewer database accesses. For distributed applications, using these multiple-row statements results in fewer network operations and a significant improvement in performance.

In order for distributed applications to take advantage of multi-row processing, some properties and settings need to be updated, depending on the type and version of IBM data server driver being used. (See tuning tips #46, #47, and #48, later in this chapter, for coding examples.)

37. Know the Locking Isolation Levels

DB2 offers four locking isolation levels: Repeatable Read (RR), Read Stability (RS), Cursor Stability (CS), and Uncommitted Read (UR). Each of these isolation levels allows the user and application to control the number and duration of read (Share) locks held within a unit of work. When you set the appropriate isolation level, based on a particular application's requirement, lock resources can be minimized, and the user/program concurrency can be increased. Take the following example:

```
SELECT LASTNAME, EMPNO
FROM EMP
WHERE LASTNAME LIKE 'S%'
WITH UR
```

With RR means that the same query can be executed multiple times within the same unit of work, and the results of the query will be identical every time (repeatable). A Share lock will be set and will stay on each row or page until the query or logical unit of work has completed. All accessed rows or pages are locked, even if they do not satisfy the predicate. For table scans, this would encompass each row/page within the table. For other queries not processing table scans, this would encompass any rows or pages that meet the predicate criteria of the SQL statement. In the example above, this would be all rows or pages containing last names that begin with S.

All Share locks with RR are held until a commit takes place. These share locks would effectively prevent updates, inserts, or deletes (X locks) from occurring on any of the rows/pages from any other process until a commit is executed.

> **NOTE** Most query tools on the market have their default isolation level set to RR, which is not good. This causes many problems in environments where users, analysts, developers, and others query the data often during the day. Many times users leave their workstations while a query running in the background is applying and holding locks on the data being retrieved. This is a common reason for many -911 SQLCODE errors.

With RS is very much like With RR, except that it will allow inserts from other users. It can at times lock more rows/pages because locks are taken and held on data, even when it goes to stage 2 processing to further check predicates. If there is a stage 2 predicate and the data does not fit the predicate criteria, the RS lock is still placed and held.

With CS sets a Share lock on each row or page processed, and the moment the cursor moves on to another row or page, it releases the lock. So at any one time, there is only one lock being held either on a row or page of data. This obviously allows good concurrency and some data integrity. Almost all batch COBOL programs in IT shops today are bound with the locking parameter CS. This is because as these programs execute cursor processing, they have no need to reread any data processed. The Share locks get freed up as the query moves through the cursor, and the query has data integrity as it processes each current row or page. This bind parameter, along with another bind parameter, Currentdata(No), provides an opportunity for avoiding locks altogether. With these two bind parameters together, DB2 can test whether a row or page has committed data on it, and if it has, DB2 will not have to obtain any lock.

With UR means that no Share locks are placed on any rows or pages processed by this query, and it does not matter if other processes have any locks on any of the data being retrieved. This can improve efficiency because it reduces overall processing time. But the one issue in using UR is that if some other process has applied updates to data being retrieved, UR will return the updated data from the buffer before the other process has

executed a commit. If for some reason the other process does a rollback of its updates, then this UR process has updated data that was never committed.

Even with the issue of possibly picking up non-committed data, there are definitely times when UR can be used:

1. **Data warehouse queries.** Typically no updates take place in these environments.
2. **OLAP/OLTP processing.** If most update processing is done in a nightly batch, then having all OLAP/OLTP processing with UR provides better performance and response time automatically due to the fact that no locking is taking place.
3. **Tables that rarely change.** Examples are codes and reference tables.
4. **Query tools.** When executing dynamic queries throughout the day, UR will not interfere with production processing by placing Share locks on data.

38. Know Null Processing

Developers, especially COBOL programmers, need to know and understand null processing. The COBOL language does not handle a null returned from DB2 as nicely as other languages. If a COBOL program receives a null from DB2 and the program is not set up to handle the null by using a null indicator or with VALUE, IFNULL, or COALESCE, then DB2 returns a -305 error. A null value by definition means an unknown value.

Example 1:

```
SELECT EMPNO,
SALARY + BONUS + COMM AS TOTAL_PAY
FROM EMP
```

If any of the columns Salary, Bonus, or Comm is null (or an unknown value), then the answer for Total_Pay comes back as null (or an unknown value). Example 1 should be coded as:

```
SELECT EMPNO,
       COALESCE (SALARY,0)  +
       COALESCE (BONUS, 0)  +
       COALESCE (COMM,  0)  AS TOTAL_PAY
FROM EMP
```

Example 2:

```
SELECT  AVG(SALARY)
FROM EMP
WHERE  WORKDEPT = 'XYZ'
```

If no rows are found, then the answer comes back null. Example 2 should be coded as follows. But make sure if setting the default value to 0, that a true average of 0 cannot happen:

```
SELECT
COALESCE(AVG(SALARY),0)
FROM EMP
WHERE  WORKDEPT = 'XYZ'
```

Example 3:

```
SELECT EMPNO, SALARY, BONUS, COMM
FROM EMP
WHERE BONUS <> COMM
```

If one of the columns has a value and the other is null, this does not make them unequal and part of the result set. If one of them has a value, and the other may not, and the developer wants them to be part of the result set due to not being equal, the query could be coded as follows:

```
SELECT EMPNO, SALARY, BONUS, COMM
FROM EMP
WHERE  COALESCE(BONUS,1) <>  COALESCE(COMM,0)
```

Note, however, that this predicate now becomes a stage 2 predicate. This works if both of the columns could contain null values.

Or as of V8, you could code it like this:

```
SELECT EMPNO, SALARY, BONUS, COMM
FROM EMP
WHERE BONUS IS DISTINCT FROM COMM
```

Note that this syntax is simpler and more straightforward but also stage 2. COBOL programmers need to know when nulls can be returned as part of the answer set, and they should try to eliminate them by coding the COALESCE or VALUE scalar functions. The -305 SQL error occurs when the answer is null and DB2 looks for null indicator to place the -1 value into. When using the VALUE, IFNULL, or COALESCE functions, a null indicator is not needed.

39. Always Program with Performance in Mind

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

- Get the correct results for requested data
- Get the results back as quickly as possible

Many times programmers 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 (the database, the network, TCP/IP, etc.), or think that the time spent processing the data was pretty

good, based on the amount of data. They need to always be sure their program is running as efficiently as possible. If they are not sure, then they should consult colleagues or, better yet, buy this book. The many tuning tips in this book will certainly give them a starting point. The one thing programmers should always know is how much data was processed based on the runtime. Did the program process 100,000 rows of data or 10 million rows of data? How many updates, inserts, deletes, selects, open cursors, commits, etc. should be noted as part of each output?

If a program has a performance issue, the first question is typically "How much data is being processed?" Every program should note somewhere the amount of data that was processed and log or display that information somewhere for others to fall back on.

40. Let SQL Do the Work

There is a divide between relational programming and procedural programming. Too many times developers do the following:

- Break up their joins or cursors. Joining tables outside DB2 is typically not smart or efficient SQL.
- Execute separate Exists logic.
- Execute separate Selects for code descriptions.
- Execute their own rounding logic.
- Execute their own truncating logic.
- Execute their own concatenation logic.
- Execute their own mathematics, string functions, etc. in their code.

Developers should take advantage of DB2 SQL scalar functions in order to get data exactly as needed for their output. Using scalar functions as part of the Select clause in SQL does not cause jobs to run long, but using them on columns in predicates can be a big performance issue. Developers should always code their queries in a query tool first, get the data back exactly as needed, and then move the query to their program. Most programs should simply be SQL fetches and code moves, with no other routines executed in between. Using functions can greatly reduce the code base, which means less code to execute and less code to test.

Always try to get what is needed into one SQL statement and then break it up if performance is an issue, even if it requires a 5- or 10-table join. Do not execute singleton Selects in order to check existence or to get code descriptions. Build this logic into the SQL statement.

41. Code with Lock Table

Developers often have programs that execute multiple SQL UPDATE, INSERT, and/or DELETE statements. Often when these statements execute, a DB2 lock takes place. Locks are a big part of overall performance and runtime, so the more locks that

take place, the longer processes execute. Typically DBAs have thresholds in place to keep developers from processing too many locks. They set a threshold option to escalate the locking to a tablespace lock and let the program continue, or they set a threshold to cause the program to abend.

If you code a Lock Table statement before processing, one exclusive lock is taken at the table level. Because this is a higher-level lock than a row or page, all individual locks on updates, inserts, or deletes are alleviated. This will make the program execute much faster, but the entire table is locked up by the program for the duration of its runtime.

For example, the Lock Table may help a program execute in 10 minutes instead of 30 to 40 minutes, due to having just one tablespace lock versus many individual row or page locks. But during those 10 minutes, everyone else is locked out of processing against that table, unless they use With UR.

Here's an example of using the Lock Table statement:

```
LOCK TABLE EMP IN EXCLUSIVE MODE
LOCK TABLE  EMP PART 3  IN EXCLUSIVE  MODE
```

Using the Lock Table statement is especially efficient when a program is going to set locks that affect 25% or more of the data in a table. Keep in mind that putting an exclusive lock on a table may help reduce runtime, but it keeps others from getting to the table. This may not be such a good idea if concurrency is an issue.

42. Consider OLTP Front-End Processing

When executing SQL statements out of front-end OLTP programs that may return multiple rows, consider trying the Optimize for n Rows statement to influence the optimizer to choose the best access path, based on the number. You should especially try this when the number of rows needed is significantly less than the total number of rows that may be returned. For example:

```
SELECT EMPNO, LASTNAME,  WORKDEPT, SALARY
FROM EMP
WHERE  WORKDEPT > ?
OPTIMIZE FOR 14 ROWS
```

If you know for sure that your screen will process only 100 rows, no matter how many get returned, then the SQL statement should specify Fetch First 100 Rows Only. (See tuning tip #65, later in this chapter.) For example:

```
SELECT EMPNO, LASTNAME,  WORKDEPT, SALARY
FROM EMP
WHERE  WORKDEPT > ?
FETCH FIRST 100 ROWS ONLY
```