Craig S. Mullins

Return to Home Page

February / March 2008

The Resource for Users of IBM zSeries & S/390 Systems

z/JOURNAL

zData Perspectives

by Craig S. Mullins

Much Ado About DB2 Locking.

Of the many new features and functionality offered by DB2 9 for z/OS, some of the more intriguing ones deal with locks and locking.

One of the intriguing DB2 9 locking features is the ability for a transaction to skip over rows that are locked. This can be accomplished by means of the SKIP LOCKED DATA option within your SQL statement(s). SKIP LOCKED DATA can be specified in SELECT, SELECT INTO, and PREPARE, as well as searched UPDATE and DELETE statements. You can also use the SKIP LOCKED DATA option with the UNLOAD utility.

Of course, if a program skips over locked data then that data is not accessed and the program will not have it available. When this option is used DB2 will just skip over any locked data instead of waiting for it to be unlocked. The benefit, of course, is improved performance because you will not incur any lock wait time. But it comes at the cost of not accessing the locked data at all. This means that you should only utilize this clause when your program can tolerate skipping over some data.

The SKIP LOCKED DATA option is compatible with cursor stability (CS) isolation and read stability (RS) isolation. But it cannot be used with uncommitted read (UR) or repeatable read (RR) isolation levels. DB2 will simply ignore the SKIP LOCKED DATA clause under UR and RR isolation levels. Let's look at an example. Suppose we have a table with 4 rows like this:

KEY	**FNAME**	**LNAME**
1	JOE	MAMA
2	KIM	PORTANT
3	JOE	PATERNO
4	DON	KNOTTS

Assume row level locking and an UPDATE that changes FNAME to JIM WHERE FNAME = 'JOE', and it is hanging out there without a COMMIT. Next, we run:

```
SELECT COUNT (*)
FROM   TABLE
WHERE  FNAME >= 'AAA'
SKIP LOCKED DATA;
```

The count returned would be 2 because DB2 skips the two locked rows. Of course, if the locks are released the count would be 4 again.

Another DB2 9 improvement is optimistic locking support. With optimistic locking, locks are obtained immediately before a read and then released. Update locks are obtained immediately before an update operation and held until the end of the transaction. Optimistic locking uses the RID and a row change timestamp to test whether data has been changed by another transaction since the last read operation. A row change timestamp, new DB2 9, is an automatically generated timestamp column, defined as follows:

```
NOT NULL [GENERATED ALWAYS | BY DEFAULT]
FOR EACH ROW ON UPDATE
AS ROW CHANGE TIMESTAMP
```

The syntax is similar to that used for other automatically generated DB2 values, such as sequences. For tables having a row change timestamp column, DB2 will automatically generate the timestamp value for each row when the row is inserted, and modify the timestamp for each row when any column in that row is updated. You can use this new column as a condition for making an UPDATE, by specifying it in your WHERE clause. For example, assume we create a CUSTOMER table with a row change timestamp column. If we want to find all of the customer rows that were changed in the past week we could run the following query:

```
SELECT CUSTNO, CUST_NAME
FROM   CUSTOMER
WHERE  ROW CHANGE TIMESTAMP FOR CUSTOMER <=
       CURRENT TIMESTAMP
AND    ROW CHANGE TIMESTAMP FOR CUSTOMER >=
       CURRENT TIMESTAMP - 7 DAYS;
```

Programs that use updateable static scrollable cursors can use optimistic locking as long as the package is bound specifying ISOLATION(CS). If you have this situation, DB2 will deploy optimistic locking to reduce the duration of locks between consecutive FETCH operations and between FETCH operations and subsequent positioned UPDATE or DELETE operations.

Without optimistic locking, the lock taken at the first FETCH is held until the next FETCH. With optimistic locking, when the application requests a FETCH to position the cursor on a row, DB2 locks that row, executes the FETCH and releases the lock. So, when you move to DB2 9 evaluate your applications looking for programs that could take advantage of optimistic locking and consider adding the ROW CHANGE TIMESTAMP to appropriate tables.

From zJournal, Feb / Mar 2008

.

Home.