Cross Platform SQL
New SQL

Tony Andrews, Application Tuning Consultant

Themis, Inc.

tandrews@themisinc.com

www.themisinc.com

Tony Andrews is currently a trainer, consultant, and technical advisor at Themis, Inc. He teaches courses on SQL, application programming, database design, and performance tuning. He also has more than 23 years experience in the development of IBM DB2 relational database applications. Most of this time, he has provided development and consulting services to Fortune 500 companies and government agencies. For the last 10 years, Tony has been splitting his time between performance and tuning consulting engagements and DB2/SQL training. His main focus is to teach today's developers the ways of RDMS application design, development, and SQL programming -- always with a special emphasis on improving performance. He is a current IBM Champion, and regular speaker at many user groups, IDUG NA, and IDUG EMEA.

Table of Contents

Currently Committed Data

WITH UR	V4.1
SKIP LOCKED DATA	V9
CURRENTLY COMITTED	V10

2

Currently Committed Data

WITH UR: This stands for 'Uncommitted Read', and applies to read-only SQL (Select, Fetch from a read-only cursor). It can be added at the end of the SQL statement or as a bind parameter. This tells DB2 to ignore any locks on data being accessed, but could return any modified and uncommitted data from other programs or processes that are in the middle of modifying data.

SKIP LOCKED DATA: This can be added to any Select, Update, or Delete statement and tells DB2 to ignore locks held on data being accessed and skip over any pages or rows that are currently locked by other programs or processes. Note that DB2 ignores SKIP LOCKED DATA when isolation repeatable read (RR) or uncommitted read (WITH UR) are used. You must use cursor stability (CS) or read stability (RS). DB2 also ignores SKIP LOCKED DATA when locking a table, partition, LOB, XML or table space. SKIP LOCKED DATA is used only when the transaction is using row or page-level locking.

V10 CURENTLY COMMITTED: This new feature is supported only as a bind parameter or in a dynamic prepare statement, and allows access to data that was last committed before a lock would take place. It works with read processes only, and only when locks take place due to another process executing inserts or deletes. If another process is taking locks due update executions, the read processes will be locked as normal.

Bind Option: CONCURRENTACCESSRESOLUTION

Prepare Option: USE CURRENTLY COMMITTED

Using *currently committed*, only committed data is returned, as was the case previously, but now read processes do not wait for writers or deleters to release locks. Instead, readers return data that is based on the currently committed version; that is, data prior to the start of the write operation.

Variable Inserts and Updates

What do you do when the input columns received for an update or insert change from execution to execution?

- Dynamic SQL?
- Static code for all possibilities?
- Have 1 update or insert that includes all columns each time?

3

Variable Inserts and Updates

Often times there are needs to insert or update table data for a subset of columns that could change from execution to execution. The programming difficulty here is trying to cover the possibility of any or all columns involved changing or not. Take for example a customer's address. If there is an address change, maybe the whole address could change or some pieces of it could change.

There are issues with each of these approaches.

Dynamic SQL has some runtime optimization overhead.

Static SQL statements for each possible combination can make the program code large, and grow it exponentially when new columns are added. This also does make for easy program maintenance.

Updating all columns each time has some wasted resources involved. This option also involves some extra processing for the program to figure out the original values.

V10 Extended Null Indicators

New for Inserts	New for Updates / Merge Updates
-5 = Use default value	-5 = Use default value
-7 = Use default value	-7 = No Update

Bind / Rebind **EXTENDEDINDICATOR (YES)**

Prepare **WITH EXTENDED INDICATORS**

4

V10 Extended NULL Indicators

V10 addresses this issue of supporting changes to a dynamic subset of columns on inserts and updates by introducing other null indicator values other than the 0 and -1. These new null indicator values tells DB2 that you do not have a value for a specific column, and how DB2 should handle the missing value (specific to an update or insert statement).

The current values of 0 and -1 will act the same. 0 means use the value in the associated host variable, and -1 means set the associated column to null.

The new values of -5 and -7 are specific to whether they are used in an update or an insert statement. For and insert statement, they both mean the same. For updates statements, they differ in telling DB2 to not do any update or to update using the default value.

As noted, to initiate the use of these extended null indicators, it will require the new parameter EXTENDEDINDICATOR(YES) for a package, and the WITH EXTENDED INDICATORS for dynamic Prepare statements.

NOTE: If the package is not bound with this parameter, or the Prepare does not include its statement, then any -5 or -7 in the update or delete will act as a -1 and set the associated column to null.

V10 Extended Null Indicators

Used with host variables for:

- Insert
- Update
- Merge, but not 'Select From Merge'

```
Move -5 to v-address2-ni
Insert into Emp
Values (:v-empno,
        :v-address2
        :v-address2-ni,
        …..
```

```
Move -7 to v-address2-ni
Update Emp
Set Address2 = :v-address2
                :v-address2-ni,
 Where Empno = :V-Empno
```

5

V10 Extended NULL Indicators

These new extended indicator values can only be used with host variables in the following situations:

Insert, Update. Merge (but not Select From Merge)

Examples:

MOVE -5 (or -7) TO V-ADDRESS2-NI *(no address2, use default)*

```
INSERT INTO EMP (EMPNO, ADDRESS, SALARY)
VALUE (:V-EMPNO,
       :V-ADDRESS2 :V-ADDRESS2-NI,
       :V-SALARY   :V-SALARY-IND)
```

MOVE 0 TO V-ADDRESS2-NI (*address2 changed*)

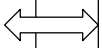
MOVE -7 TO V-SALARY-NI *(salary not changed, skip column)*

```
UPDATE EMP
SET ADDRESS2 = :V-ADDRESS2 :V-ADDRESS2-NI,
    SALARY   = :V-SALARY   :V-SALARY-NI
WHERE EMPNO = :EMPNO
```

Stored Procedure Result Set Example

PROC1

```
DECLARE  LOC1
 RESULT_SET_LOCATOR
 VARYING;

CALL S1.PROC2  (V1,V2,V3);

ASSOCIATE
 RESULT SET
 LOCATOR(LOC1) WITH
 PROCEDURE  S1. PROC2 ;

ALLOCATE C1 CURSOR
  FOR RESULT SET LOC1;

WHILE ..... DO
FETCH FROM C1…

CLOSE  C1 ;
```



PROC2

```
DECLARE SAL CURSOR
 WITH  RETURN
  FOR SELECT …
  FROM …
  WHERE …;
…
OPEN SAL;
…

END
```

Stored Procedure Result Set Example

V10 Returning Result Sets

```
DECLARE SAL CURSOR
  WITH  RETURN TO CALLER
   FOR SELECT …
   FROM …
   WHERE …;
…
OPEN SAL;
…

END
```

```
DECLARE SAL CURSOR
  WITH  RETURN TO CLIENT
   FOR SELECT …
   FROM …
   WHERE …;
…
OPEN SAL;
…

END
```

V10: ‘Return to Caller’ vs ‘Return to Client’

Prior to V10, a cursor defined as ‘With return’ is the default and short for ‘With Return To Caller’. This means that only the program that called the stored procedure will have access to the returned data and no other programs within a calling sequence.

As of DB2 V10, you can now code ‘With Return To Client’ which will give the calling (client) program access to returned data no matter how many levels deep in the calling sequence. *It is important to note that with this option, no other intermediate programs will have access to this data, only the calling (top level) program.*

This brings new functionality to V10, and keeps it n sync with DB2 LUW which currently has the ‘Return To Client’ option.

Without this option, stored procedures often loaded data into a declared temporary table where the first level program would then declare a cursor ‘With Return’ on the temporary table, giving the calling program its expected result set.

In IBM testing, it was shown that ‘With Return To Client’ outperformed the creating and loading of a Global Temporary Tables.

SELECT From UPDATE/DELETE/MERGE

```
SELECT HIREDATE,LASTNAME
  FROM FINAL TABLE
       (UPDATE EMP
           SET HIREDATE = CURRENT DATE
         WHERE EMPNO = '000340');
```

HIREDATE	LASTNAME
2009-03-15	GOUNOT

SELECT From UPDATE/DELETE/MERGE

The SELECT FROM INSERT feature was introduced in Version 8 to allow the insertion of a record and the retrieval of certain fields that were inserted. DB2 9 expands this feature to UPDATE, DELETE and MERGE. In this example, an update is being performed to set a date to the value of special register CURRENT DATE. If the application needs to know the value that was supplied, it may perform a select from the updated row(s) in a single SQL statement.

The primary task being accomplished here is an UPDATE. The SELECT allows the application to make fewer calls to DB2 than would otherwise be necessary.

The keyword FINAL TABLE should be used when selecting from an INSERT or MERGE. This will return the values as they exist AFTER the operation.

The keyword OLD TABLE should be used when selecting from a DELETE. This will return the values as they exist BEFORE the operation.

When selecting from an UPDATE, either FINAL TABLE or OLD TABLE may be used to view the column values before or after they update.

This Page Intentionally Left Blank