Dec 2012 / January 2012

zData Perspectives

The Scoop on "Dirty Read"

by Craig S. Mullins

Application programmers should understand the potential for concurrency problems when accessing relational data. When one program attempts to read data that's in the process of being changed by another, the DBMS must forbid access until the modification is complete to ensure data integrity. Most DBMS products, including DB2, use a locking mechanism for all data items being changed. Therefore, when one task is updating data on a page, another task can't access data (i.e., read or update) on that same page until the data modification is complete and committed.

Programs that read DB2 data typically access numerous rows during their execution and are susceptible to concurrency problems. Since V4,

to modify another, or for any production, mission-critical work that can't tolerate data integrity problems.

Most DB2 applications aren't viable candidates for dirty reads, but there are a few situations where dirty reads can be beneficial. Examples include access to a reference, code, or look-up table (where the data is non-volatile), statistical processing on large amounts of data, analytical queries in data warehousing and Business Intelligence (BI) applications, or when a table (or set of tables) is used by a single user only (which is rare). Additionally, if the data being accessed is already inconsistent, little harm can be done using a dirty read to access the information.

Because of the data integrity issues associated with dirty reads, DBAs should keep track of the plans and packages that specify an isolation level of UR. This information can be found in the DB2

susceptible to concurrency problems. Since V4, DB2 has provided read-through locks, also known as "dirty read" or "uncommitted read," to help overcome concurrency problems. When using uncommitted reads, an application program can read data that has been changed, but not yet committed.

Dirty read capability is implemented using a new isolation level, UR, for uncommitted read. If the application program is using the UR isolation level, it will read data without taking locks. This lets the application program read data contained in the table as it's being manipulated. Consider the following sequence of events:

1. At 9 a.m., a transaction containing the following SQL to change a specific value is executed:

```
UPDATE EMP
  SET  FIRST_NAME = 'MICHELLE'
WHERE  EMPNO = 10020;
```

2. The transaction is long-running and continues to execute without issuing a COMMIT.
3. At 9:01 a.m., a second transaction attempts to SELECT the data that was changed, but not committed.

If the UR isolation level was specified for the second transaction, it would read the changed data even though it had yet to be committed. Because the program simply reads the data in whatever state it happens to be at that moment, it can execute faster than if it had to wait for locks to be taken and resources to be freed before processing.

However, the implications of reading uncommitted data must be carefully examined before being implemented, as several problems can occur. A dirty read can cause duplicate rows to be returned where none exist. Alternately, a dirty read can cause no rows to be returned when one (or more) actually exists.

So, when is it a good idea to implement dirty reads using the UR isolation level? The general rule is to avoid dirty reads whenever the results must be 100 percent accurate. For example, avoid UR if calculations must balance, data is being retrieved from one source

catalog. The following two queries can be used to find the applications using uncommitted reads.

Issue the following SQL for a listing of plans that were bound with ISOLATION(UR) or contain at least one statement specifying the WITH UR clause:

```
SELECT  DISTINCT S.PLNAME
FROM    SYSIBM.SYSPLAN P,
        SYSIBM.SYSSTMT S
WHERE   (P.NAME = S.PLNAME AND
         P.ISOLATION = 'U'
        )
        OR S.ISOLATION = 'U'
ORDER BY S.PLNAME;
```

Issue the following SQL for a listing of packages that were bound with ISOLATION(UR) or contain at least one statement specifying the WITH UR clause:

```
SELECT  DISTINCT P.COLLID, P.NAME, P.VERSION
FROM    SYSIBM.SYSPACKAGE   P,
        SYSIBM.SYSPACKSTMT  S
WHERE   (P.LOCATION = S.LOCATION AND
         P.LOCATION = ' '        AND
         P.COLLID = S.COLLID     AND
         P.NAME = S.NAME         AND
         P.VERSION = S.VERSION   AND
         P.ISOLATION = 'U'
        )
        OR S.ISOLATION = 'U'
ORDER BY S.COLLID, S.NAME, S.VERSION;
```

The dirty read capability can provide relief to concurrency problems and deliver faster performance in specific situations. Understand the implications of the UR isolation level and the "problems" it can cause before diving headlong into implementing it in your production applications.

From Enterprise Tech Journal, December 2012 / January 2013.

DB2PORTAL.com