

New options for analyzing lock timeouts in DB2 9.5

Get to know a new DB2 9.5 feature that simplifies lock timeout analysis

Skill Level: Intermediate

[Dirk Fechner \(fechner@de.ibm.com\)](mailto:fechner@de.ibm.com)

IT Specialist
IBM

03 Apr 2008

The combination of the `db2pd` tool and the `db2cos` script can be used to gather information about the reason for a SQL0911N lock timeout error, as discussed in "[Analyzing lockwait situations in DB2 for Linux, UNIX, and Windows](#)" (developerWorks, Jul 2007). With DB2® 9.5, the options for lock timeout analysis have been significantly enhanced so that lock timeout analysis becomes even more simple. This article explores those new lock timeout reporting capabilities and examines the additional information that can be collected to determine the reason for a lock timeout occurrence.

Review of lock timeout analysis in DB2 9.1

The lock timeout analysis method using the `db2pd` tool and `db2cos` script consists of the following steps:

1. A special DB2 script — named `db2cos` — is adapted to execute a certain `db2pd` call each time the `db2cos` script is invoked. The `db2pd` call gathers information about locks, transactions, applications, and the statement cache and stores this information in a text file for analysis purposes.
2. To execute the `db2cos` script — and therefore the contained `db2pd`

command — automatically when a lock timeout occurs, lock timeout events are registered using the `db2pdcfg` command.

3. In the case of a lock timeout event, a DBA can examine the corresponding `db2pd` output generated by the automatic invocation of the `db2cos` script. This allows the DBA to determine the cause of the lock contention so he or she can take measures to avoid such situations in the future.

For a detailed description of those steps accompanied by a sample scenario, see the already mentioned article found in the [Resources](#) section.

Although the described method provides a lot of information that makes lock timeout analysis much simpler than in pre-Version 9 releases of DB2 for Linux, Unix, and Windows, it still has some shortcomings:

- Using this method requires manual adaptation of the `db2cos` script and registration of lock timeout events by calling `db2pdcfg`. Both steps are not complex but may be obstacles for novices.
- Interpreting the `db2pd` output to identify the applications and SQL statements involved in a lock timeout situation is not trivial and requires some exercise.
- In cases where lock timeouts are caused by transactions consisting of more than a single SQL command, the information collected by `db2pd` may not be enough to determine the SQL statement responsible for the problematic lock.

New lock timeout reporting capabilities in DB2 9.5

With its lock timeout reporting capabilities, DB2 9.5 introduces a new feature that makes lock timeout analysis child's play. To activate lock timeout reporting, just set the DB2 registry variable `DB2_CAPTURE_LOCKTIMEOUT` to `ON`, and restart your DB2 instance:

Listing 1. Activate lock timeout reporting in DB2 9.5

```
db2set DB2_CAPTURE_LOCKTIMEOUT=ON
db2stop
db2start
```

Yes, that's all. When `DB2_CAPTURE_LOCKTIMEOUT` is set to `ON`, DB2 automatically creates a report file for each lock timeout occurrence. The report file is written to the directory where the `DIAGPATH` database manager configuration (DBM CFG)

parameter points, to and contains information about, the date and time of the lock timeout, the problematic lock, the lock requester, and the lock owner.

So is the `db2cos` script obsolete in DB2 9.5? No, that's not the case. Usage of the `db2cos` script in combination with the `db2pd` tool has a broader range of use. That means the combination of those tools can still be used to capture information about any kind of DB2 event associated with a SQL code or a DB2 internal return code, not just lock timeouts.

Now concentrate on the new DB2 9.5 registry variable `DB2_CAPTURE_LOCKTIMEOUT` and walk through a sample lock timeout scenario using the DB2 `SAMPLE` database. If the `SAMPLE` database does not already exist, create it by calling the following command:

Listing 2. Creating the `SAMPLE` database

```
db2samp1
```

Lock timeouts can only occur when the database configuration (DB CFG) parameter `LOCKTIMEOUT` is set to a value other than -1. A value of -1 means that an application waits indefinitely for a required lock. In most cases, that is not the desired behavior, although -1 is the default for `LOCKTIMEOUT`. For this sample scenario, assume that `LOCKTIMEOUT` is changed to 10 (seconds — the value for `LOCKTIMEOUT` is given in seconds):

Listing 3. Changing the value for `LOCKTIMEOUT`

```
db2 "UPDATE DB CFG FOR SAMPLE USING LOCKTIMEOUT 10"
```

To force a lock timeout error, establish a first database connection and execute some SQL statements simulating a real transaction on the table `EMPLOYEE`:

Listing 4. First transaction giving each employee a two percent raise

```
db2 "CONNECT TO SAMPLE"  
db2 +c "UPDATE EMPLOYEE SET SALARY = SALARY * 1.02"  
db2 +c "SELECT LASTNAME, FIRSTNAME, SALARY FROM EMPLOYEE ORDER BY LASTNAME ASC"
```

The transaction so far consists of an `UPDATE` command giving each employee a two percent raise. Afterward, the new salary is queried with a `SELECT` statement. Note that auto-commit is disabled by specifying the `+c` option for the DB2 command line processor (CLP) calls. The `UPDATE` statement results in an exclusive (X) lock on each row in the `EMPLOYEE` table. Those X-locks persist until the transaction is

ended by a COMMIT or a ROLLBACK statement.

Without ending the transaction, establish a second database connection in a separate shell and start another transaction:

Listing 5. Second transaction giving each manager a 10 percent bonus (depending on the salary)

```
db2 "CONNECT TO SAMPLE"  
db2 +c "UPDATE EMPLOYEE SET BONUS = SALARY * 0.1 WHERE JOB = 'MANAGER'"
```

The intention of this transaction is to give each manager a 10 percent bonus depending on his or her current salary. As all rows in the EMPLOYEE table are X-locked by the first transaction, the second application goes into lock wait mode. After 10 seconds (remember the LOCKTIMEOUT setting) the expected lock timeout occurs.

So far nothing new. But as the DB2 registry variable DB2_CAPTURE_LOCKTIMEOUT is set to ON, lock timeout reporting is active and DB2 has automatically generated a lock timeout report in the DIAGPATH directory. Note that the default DIAGPATH changed in DB2 9.5 for Windows. If the DIAGPATH parameter is not set, the DIAGPATH points to the directory DB2INSTPROF\DB2INSTANCE where DB2INSTPROF is the location of the instance directory and DB2INSTANCE is the name of the DB2 instance. To determine the path stored in DB2INSTPROF, you can display the DB2 registry contents by executing `db2set -all`. The value of DB2INSTANCE is DB2 if you created the SAMPLE database in the default instance. The name of the report file is `db2locktimeout.dbpartition.agentid.timestamp` where `dbpartition` is always 0 for a single-partition database.

The lock timeout report generated by DB2 looks similar to the following:

Listing 6. Lock timeout report

```
LOCK TIMEOUT REPORT  
  
Date:                03/01/2008  
Time:                07:34:31  
Instance:           DB2  
Database:           SAMPLE  
Database Partition: 0  
  
Lock Information:  
  
Lock Name:          020006000400400100000000052  
Lock Type:          Row  
Lock Specifics:     Tablespace ID=2, Table ID=6, Row ID=x0400400100000000
```

```

Lock Requestor:
  System Auth ID:          FECHNER
  Application Handle:      [0-38]
  Application ID:          *LOCAL.DB2.080103063343
  Application Name:        db2bp.exe
  Requesting Agent ID:    5232
  Coordinator Agent ID:   5232
  Coordinator Partition:  0
  Lock timeout Value:     10000 milliseconds
  Lock mode requested:    ..U
  Application Status:     (SQLM_UOWEXEC)
  Current Operation:      (SQLM_EXECUTE_IMMEDIATE)
  Lock Escalation:        No

Context of Lock Request:
  Identification:         UOW ID (1); Activity ID (1)
  Activity Information:
    Package Schema:       (NULLID )
    Package Name:         (SQLC2G13NULLID )
    Package Version:      ( )
    Section Entry Number: 203
    SQL Type:              Dynamic
    Statement Type:       DML, Insert/Update/Delete
    Effective Isolation:  Cursor Stability
    Statement Unicode Flag: No
    Statement:            UPDATE EMPLOYEE SET BONUS = SALARY * 0.1
                        WHERE JOB = 'MANAGER'

Lock Owner (Representative):
  System Auth ID:          FECHNER
  Application Handle:      [0-33]
  Application ID:          *LOCAL.DB2.080103063332
  Application Name:        db2bp.exe
  Requesting Agent ID:    5488
  Coordinator Agent ID:   5488
  Coordinator Partition:  0
  Lock mode held:         ..X

List of Active SQL Statements:  Not available

List of Inactive SQL Statements from current UOW: Not available

```

The lock timeout report consists of four sections:

- The first section provides information about the date and time of the lock timeout occurrence and about the corresponding instance and database.
- The section *Lock Information* shows the lock that caused the lock timeout. Besides the internal lock name, the type of the lock (row or table lock) and the table information is shown. To determine the tablename, you have to query the catalog view `SYSCAT.TABLES` with the given tablespace ID and table ID:

Listing 7. Mapping a tablespace ID/table ID to a tablename

```

SELECT TABSCHEMA, TABNAME
FROM SYSCAT.TABLES
WHERE TBSPACEID = tbspaceid AND TABLEID = tableid

```

- The application that experienced the lock timeout is described in the section *Lock Requestor*. Some of the more interesting entries in this sections are the authentication ID used to connect to the database, the application name, the requested lock mode (and whether or not the lock is the result of a lock escalation), the isolation level of the statement that required the lock, and, last but not least, the SQL statement text itself.
- The last section *Lock Owner (Representative)* lists the application holding the problematic lock. As with the other application, you can see information about the authentication ID, the application name, and the lock mode. In some cases more than one application may hold a (share) lock that blocks the lock requestor. In those cases only one of the lock owners is shown in the lock timeout report. That's why the section has the addition *Representative*.

At the beginning of this article, three shortcomings were mentioned of the lock timeout analysis approach using `db2cos` with `db2pd`. The first one was usability. The `db2cos` with `db2pd` approach requires several steps to set up lock timeout monitoring. The new approach is much simpler, you just set `DB2_CAPTURE_LOCKTIMEOUT=ON` and that's it. The second shortcoming was complexity, because it requires some exercise to read a `db2pd` output and to correlate the different `db2pd` sections. With the new approach, DB2 generates a report file for you with all the necessary information in one place. But what about the last shortcoming mentioned: Not enough information about the SQL statements that were involved in the lock timeout situation? Until now you only know the SQL statement that was blocked by the existing lock, but you have no information about the statement that caused the lock. Concerning this point, the new lock timeout reporting functionality provides improvements too. Now see how it works.

Collect SQL statement history information

To get information about the SQL statements that were executed by the lock owner's application, create a deadlock event monitor with the `DETAILS HISTORY` option and activate the event monitor. For example, a suitable deadlock event monitor could be created and activated as follows:

Listing 8. Creating a deadlock event monitor with the `DETAILS HISTORY` option

```
db2 "CREATE EVENT MONITOR evmondeadlock FOR DEADLOCKS WITH DETAILS HISTORY  
    WRITE TO FILE 'path'"  
db2 "SET EVENT MONITOR evmondeadlock STATE 1"
```

You may ask, "Why do I need a deadlock event monitor for monitoring lock timeouts?" The answer is that lock timeout reporting builds partly on functionality delivered by deadlock event monitor code. When a deadlock event monitor is created with the `DETAILS HISTORY` option, DB2 keeps track of the SQL statements already executed in a transaction. In case of a deadlock or lock timeout, this information can be used to present a history of SQL statements that may be involved in the deadlock or lock timeout occurrence.

With the active deadlock event monitor, walk through the lock timeout scenario described above again. This time DB2 writes a lock timeout report, as shown in Listing 9:

Listing 9. Lock timeout report containing SQL statement history information

```

LOCK TIMEOUT REPORT

Date:                03/01/2008
Time:                15:10:13
Instance:            DB2
Database:            SAMPLE
Database Partition: 0

Lock Information:

Lock Name:           020006000400400100000000052
Lock Type:           Row
Lock Specifics:      Tablespace ID=2, Table ID=6, Row ID=x0400400100000000

Lock Requestor:
System Auth ID:      FECHNER
Application Handle:  [0-202]
Application ID:      *LOCAL.DB2.080103140934
Application Name:    db2bp.exe
Requesting Agent ID: 2356
Coordinator Agent ID: 2356
Coordinator Partition: 0
Lock timeout Value: 10000 milliseconds
Lock mode requested: ..U
Application Status:  (SQLM_UOWEXEC)
Current Operation:   (SQLM_EXECUTE_IMMEDIATE)
Lock Escalation:     No

Context of Lock Request:
Identification:      UOW ID (1); Activity ID (1)
Activity Information:
Package Schema:      (NULLID )
Package Name:        (SQLC2G13NULLID )
Package Version:     ( )
Section Entry Number: 203
SQL Type:            Dynamic
Statement Type:      DML, Insert/Update/Delete
Effective Isolation: Cursor Stability
Statement Unicode Flag: No
Statement:            UPDATE EMPLOYEE SET BONUS = SALARY * 0.1
                     WHERE JOB = 'MANAGER'

Lock Owner (Representative):
System Auth ID:      FECHNER

```

```

Application Handle:      [0-188]
Application ID:         *LOCAL.DB2.080103140511
Application Name:      db2bp.exe
Requesting Agent ID:   5488
Coordinator Agent ID:  5488
Coordinator Partition: 0
Lock mode held:       ..X

List of Active SQL Statements:  Not available

List of Inactive SQL Statements from current UOW:

Entry:                 #1
Identification:       UOW ID (6); Activity ID (2)
Package Schema:      (NULLID )
Package Name:        (SQLC2G13)
Package Version:     ( )
Section Entry Number: 201
SQL Type:            Dynamic
Statement Type:      DML, Select (blockable)
Effective Isolation: Cursor Stability
Statement Unicode Flag: No
Statement:           SELECT LASTNAME, FIRSTNME, SALARY FROM EMPLOYEE
                    ORDER BY LASTNAME ASC

Entry:                 #2
Identification:       UOW ID (6); Activity ID (1)
Package Schema:      (NULLID )
Package Name:        (SQLC2G13)
Package Version:     ( )
Section Entry Number: 203
SQL Type:            Dynamic
Statement Type:      DML, Insert/Update/Delete
Effective Isolation: Cursor Stability
Statement Unicode Flag: No
Statement:           UPDATE EMPLOYEE SET SALARY = SALARY * 1.02

```

The beginning of this lock timeout report is identical to what you have already seen. But this time the *Lock Owner* section contains additional, valuable information. Under the heading *List of Inactive SQL Statements from current UOW*, you see all the SQL statements that were executed in the lock owner's transaction prior to the occurrence of the lock timeout. From this list of SQL statements, you can identify the statement(s) that are responsible for the problematic lock. In this scenario, it is the UPDATE statement to increase the salary of each employee.

Note that this functionality is a major improvement to the db2cos with db2pd approach. With the db2cos with db2pd approach, you would only see the last statement executed by the lock owner's application — in this scenario the query on the EMPLOYEE table. But as the query did not cause the problematic X-lock, you would still not know which statement was responsible for the lock. With the new approach — DB2_CAPTURE_LOCKTIMEOUT with a deadlock event monitor — you have the history of all SQL statements executed in the lock owner's transaction, which makes it possible to identify the UPDATE as the relevant statement.

Hints for using lock timeout reporting

A deadlock event monitor with statement history capabilities affects all applications and increases the monitor heap usage by the DB2 database manager. So this kind of deadlock event monitor should be used with caution. You should always start by setting only `DB2_CAPTURE_LOCKTIMEOUT=ON`, and activate a deadlock event monitor with the `DETAILS HISTORY` option only if required.

When working with lock timeout reporting, you may notice that the number of lock timeout report files in the `DIAGPATH` continuously increases. DB2 does not delete those report files, so you as a DBA are responsible for deleting them or moving them to a different place so that there is always enough space on the `DIAGPATH`'s filesystem.

Even with the functionality of lock timeout reporting, it may not always be possible to determine the cause of lock timeouts easily. For example this may be the case when static SQL or DB2 internal locks are involved in lock timeout occurrences. The *Lock timeout reporting* chapter in the DB2 9.5 documentation provides a short list of those limitations (see the [Resources](#) section below). However, lock timeout reporting in DB2 9.5 is surely a functionality many DBAs waited for and will make analyzing lock timeouts much easier than ever before.

Summary

This article introduces the new lock timeout reporting capabilities of DB2 9.5. The new capabilities are compared to the approach described in "[Analyzing lockwait situations in DB2 for Linux, UNIX, and Windows](#)" to demonstrate the strengths of this new DB2 9.5 feature.

Resources

Learn

- "[Analyzing lockwait situations in DB2 for Linux, UNIX, and Windows](#)" (developerWorks, Jul 2007): Read about lockwait and lock timeout monitoring.
- The [Lock timeout reporting](#) chapter of the "IBM DB2 Database for Linux, UNIX, and Windows Information Center": Find limitations that should be considered when using the lock timeout reporting feature.
- "[DB2 9 Fundamentals exam 730 prep, Part 6: Data concurrency](#)" (developerWorks, Jul 2006): Get an introduction to DB2 data concurrency concepts as well as a nice overview of the different kinds of locks used by DB2.
- "[Understanding locking in DB2 Universal Database](#)" (developerWorks, Nov 2005): Discover the principles behind DB2's locking strategy.
- "[Improve concurrency with DB2 9.5 optimistic locking](#)" (developerWorks, Jan 2008): Find the new DB2 9.5 optimistic locking support that can help avoiding database concurrency problems.
- "[DB2 9.5 Information Center for Linux, UNIX, and Windows](#)" Access the complete DB2 9.5 documentation online in HTML format.
- [DB2 9 for Linux UNIX and Windows Support](#): Search for APARs, download fixpacks, get DB2 documentation in PDF format, and more
- Browse the [technology bookstore](#) for books on these and other technical topics.
- Visit the [developerWorks resource page for DB2 for Linux, UNIX, and Windows](#) to read articles and tutorials and connect to other resources to expand your DB2 skills.
- [developerWorks Information Management zone](#): Learn more about DB2. Find technical documentation, how-to articles, education, downloads, product information, and more.
- Stay current with [developerWorks technical events and webcasts](#).

Get products and technologies

- Download a free trial version of [DB2 Enterprise 9](#).
- Now you can use DB2 for free. Download [DB2 Express-C](#), a no-charge version of DB2 Express Edition for the community that offers the same core data features as DB2 Express Edition and provides a solid base to build and deploy applications.
- Download [IBM product evaluation versions](#) and get your hands on application development tools and middleware products from DB2®, Lotus®, Rational®,

Tivoli®, and WebSphere®.

Discuss

- Check out [developerWorks blogs](#) and get involved in the [developerWorks community](#).

About the author

Dirk Fechner

Dirk Fechner works as an IT specialist for IBM Software Group. His area of expertise is the administration of and application development with DB2 on distributed platforms. He has eight years of experience with DB2 and is an IBM Certified Advanced DBA and IBM Certified Application Developer. He currently supports administrators, developers, and end-users at Daimler on a wide variety of DB2 topics including administrative tasks, application development, and problem determination.