THE FUTURE OF DB2 SECURITY: PART II

By Craig S. Mullins

Part I of this two-part series (July 1992) highlighted the extensions to DB2 security provided by IBM in the latest release of DB2 (V2R3). It also presented two specific problem areas and solutions surrounding the current implementation of DB2 security. The following section continues to outline problems that still exist with DB2 security. In addition, a solution is proposed for each problem that is addressed. I have made every effort to conform to the specifications of RMV2 when formulating these proposed solutions.

Problem #3: No Means of Canceling DB2 Threads

Outside of bringing down DB2, the only way to cancel a DB2 thread is to cancel the allied agent job that initiated the thread. This is not always possible or the best solution.

Solution #3: Provide a Means To Terminate Threads

DB2 should provide the ability to terminate threads in much the same way that it can terminate utilities. This would enable authorized users to react more quickly to potentially problematic situations, canceling offending programs by terminating their thread. It also places the controls for access to the data base via application programs more firmly under the control of the DBMS.

The syntax of the command should permit single threads to be canceled in one of two methods. The first method, MODE(ABORT), should unconditionally cancel the thread and roll back all update activity to the last COMMIT point. The second method, MODE(COMMIT), should delay the termination of the thread until the next COMMIT statement is issued by the thread.

A hierarchy of thread termination should exist such that subsequent termination of the same thread using MODE(ABORT) will override prior MODE(COMMIT) terminations. This will enable a user to immediately terminate a thread after waiting what may be deemed too long for the next commit point within a program. This is particularly important for TSO threads, which can sometimes get swapped out for an inordinate amount of time (due to a downgrade in the TSO performance classes).

Figure 1 is the proposed syntax of the terminate thread command.

The connection name is generally the name of the plan associated with the thread to be terminated. The corrid, or correlation identifier, is either the user ID or the jobname associated with the thread to be terminated. If the combination of connection name and corrid is not unique, then all threads associated with that combination of connection name and correlation ID should be terminated as directed by the -TERM THREAD command.

An additional system privilege will be needed to control who can terminate threads. The proposed syntax follows:

```
GRANT TERMTHRD TO ——— authid ——— [WITH GRANT OPTION]
                  — PUBLIC —
```

TERMTHRD authority should be included as a component of the SYSOPR group level authority.

Problem #4: Complexity of Security Implementation

Often times the DB2 security administrator will get requests asking for the security of one user to be duplicated for another user. An example of this type of request is when a new programmer is added to a project and needs the same privileges as the other programmers on that project. To satisfy this type of security request, the administrator must utilize one of three methods:[1]

- analyze the DB2 catalog to determine what security is in place and replicate the DCL needed to grant the proper security;
- maintain PDS members containing the security needed for each project, and maintain these PDS members as the security needs of the applications change; and

Figure 1: Proposed Syntax of the Terminate Thread Command

```
-TERM THREAD (connection-name)  CORRID (corrid)  MODE ( ——— ABORT ——— )
                                                      — COMMIT —
```

Figure 2: Example of the GRANT LIKE Statement

```
GRANT  TABLE  SECURITY LIKE source table name TO target table name
                          and
GRANT  USER  SECURITY LIKE source authid TO target authid
```

- use a vendor tool (if available) specifically written to accomplish this task.

Also, requests may be received that ask for the security of one table to be used as a model for the security of a new table. For example:

A new table is being added to the billing system that contains billing history information. The same users that access normal billing information also need to access the billing history information. Therefore, the security from the table containing the normal billing information needs to be cloned for the history table.

Solution #4: Simplify Security With Powerful Security Cloning Statements

Security administrators should have the power to issue statements allowing GRANT LIKE and REVOKE everything statements. The statements need to be powerful enough to work for both user IDs and tables.

The GRANT LIKE statement would specify a source and a target user ID. All security implemented for the source ID would be cloned for the target ID. See Figure 2.

Note that the source and target table names (or authids) should never be allowed to be identical.

Additionally, the REVOKE everything statement should be added. This statement will allow a security administrator to remove all privileges from an authid or table. This will be particularly useful for:

- removing the security from a user when s/he moves from department to department or out of the company;
- moving a table from one environment to another where the same security will be needed (e.g., copying the table from one production DB2 subsystem to another).

Proposed syntax:

```
REVOKE TABLE SECURITY FROM table name
and
REVOKE USER SECURITY FROM authid
```

Both the GRANT LIKE and REVOKE everything statements should only be authorized for personnel granted SYSADM or SYSCTRL privileges.

Problem #5: Complex Administration Due to Cascading Revokes

Anyone even remotely familiar with DB2's security features knows all about the horrors of cascading revokes. The DB2 revoke statement is used to remove privileges from a user. A cascading revoke is an implicit, "behind the scenes" revocation of security. In other words, revoking an authorization from user 1 also causes user 2 to lose her/his authority. See Figure 3 for an example of cascading revokes.

Figure 3: Cascading Revokes

User1 grants authority to User2.

USER1 —grant—> USER2

User2 grants authority to user3, who in turn grants authority to user4.

USER2 —grant—> USER3 —grant—> USER4

User1 revokes user2's authority.

USER1 <—revoke— USER2

This causes user3's authority to be revoked because user1 no longer has the privilege to grant to others. This is a cascading revoke. The same fate befalls user4, because user3 will no longer have the authority either.

USER2 <—cascade— USER3 <—cascade— USER4

Cascading revokes can cause major headaches for DB2 security administrators. How can a cascading revoke cause problems? Well, let's take a quick look at the overall situation. The WITH GRANT OPTION clause of the GRANT statement enables users to grant privileges to other users enabling them to also grant the privileges. This can build into a large hierarchy of security where users pass on privileges from one to another.

Consider the example shown in Figure 4. Say Bob represents a SYSADM, responsible for granting all security within your DB2 subsystem. Assume that he distributes his security administration responsibilities by granting privileges WITH GRANT OPTION to people in each department (e.g., Ron and Dianne) who grant privileges and further delegate security administration roles. Quickly a very large hierarchy of security administration responsibility has been built, such as the one in Figure 4.

Then Bob quits. How do we remove him from the system? If we simply revoke his SYSADM authority, then everything that he granted will also be eliminated because of cascading revokes. The only way to safely remove Bob from the system is to do a complete analysis of the DB2 catalog security tables (listed in Figure 5) before revoking his SYSADM privilege. After revoking this privilege, we must piece together the information from the DB2 catalog, regrant all authority removed by the cascading revokes, cross our fingers and hope we didn't miss anything.

This is an unacceptable means of controlling DB2 security. Admittedly, there are several products from DB2 add-on tool vendors that address and control these problems, but optimally, IBM should rectify this inadequate DB2 security mechanism.

Solution #5: Provide Security Integrity Features

IBM should implement Security Integrity for DB2 and model it in much the same fashion as Referential Integrity (RI). Each revoke statement should have a mandatory revoke rule attached to it. The revoke rule will tell the DBMS what action to take for the dependent privileges in the DB2 catalog when any given privilege is revoked.

For those familiar with DB2's Referential Integrity delete rule, the revoke rule will be similar. There should be three revoke rules, one of which must be selected for each revoke requested. These are:

- CASCADE;
- RESTRICT; and
- SET AUTHID.

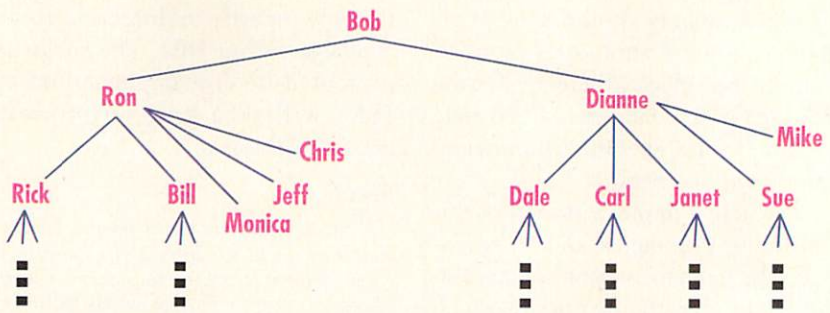
The CASCADE option will act exactly like the current DB2 revoke statement, and therefore should be the default if no rule is explicitly specified. Refer again to Figure 3 for an explanation of how the CASCADE option will function.

The RESTRICT option, on the other hand, will prohibit any revoke from occurring that would cause a cascading revoke. This will effectively restrict the revocation of security "behind the scenes."

Finally, the SET AUTHID revoke rule will permit the revoke to occur, but will not cause any cascading revokes. The revoke will occur, but instead of cascading any dependent privileges, the GRANTOR column in any row of a DB2 catalog table that would have been deleted due to cascading revoke will be set to the authorization ID of the user issuing the revoke. So, using the example in Figure 3, if SET AUTHID was specified, USER1 revokes from USER2, but USER3 keeps her/his authority, with the DB2 catalog being updated to indicate that USER1 granted the authority, instead of USER2. The security for USER4 will be completely unaffected.

Using the SET AUTHID rule leaves the security structure intact except for the specific authorization being explicitly revoked. This is often necessary when a DBADM or SYSADM (or anyone having the WITH GRANT OPTION for that matter) moves on. It is customary to remove the specific authid of the person no longer with the company, but to retain her/his work. The current method for accomplishing this is to analyze the DB2 catalog, rebuild all grant statements in the hierarchy before revoking, revoke the privilege(s) and reapply the grants. This is senseless. The solution presented in this section will eliminate the requirement for this difficult maneuvering.

Figure 4: Example of Hierarchic Security



Further, I think that this is a feature that should not only be added to DB2, but to the relational model as well.

Problem #6: Non-Recoverable Data Due to Cascading Drops

DB2 is great at providing a level of abstraction between its data base objects and the physical implementation of those objects. For example, when a tablespace is created using storage groups, DB2 will automatically allocate the underlying VSAM liner data set for that tablespace. Conversely, when a storage group defined tablespace is dropped, DB2 will automatically delete the underlying data set. This is good because it eliminates the tedious job of using IDCAMS to define and delete VSAM data sets for DB2 objects. However, once a tablespace is dropped, all of the data is gone. If the tablespace was dropped by mistake, there is no mechanism for reactivating it.

Solution #6: Provide Delayed Dropping and Reactivation

According to the relational model, a mechanism should exist to delay the deletion of data when a data base object is dropped. Upon execution of the DROP TABLESPACE statement, a grace period for deleting the underlying data set(s) should be permitted. This grace period should be specified as a duration in terms of calendar days, and should be explicitly stated in the CREATE TABLE SPACE DDL.

It also may be prudent to implement something of this nature for the DROP TABLE statement. This would allow the archival of table data for a specified number of days when the DROP TABLE statement is issued. DROP TABLE does not delete data sets, but it does make the

Figure 5: DB2 Catalog Security Tables

Table	Description of the Table
SYSCOLAUTH	Contains the UPDATE privileges held by DB2 users on specific table or view columns.
SYSDBAUTH	Contains data base privileges held by DB2 users.
SYSPACKAUTH	Contains package privileges held by DB2 users.
SYSPKSYSTEM	Contains the systems enabled (CICS, IMS, batch, etc.) for DB2 packages.
SYSPLANAUTH	Contains plan privileges held by DB2 users.
SYSPLSYSTEM	Contains the systems (CICS, IMS, batch, etc.) enabled for DB2 plans.
SYSRESAUTH	Contains resource privileges held by DB2 users.
SYSTABAUTH	Contains table privileges held by DB2 users.
SYSUSERAUTH	Contains system privileges held by DB2 users.

data inaccessible to data base users.

A command to reactivate data in dropped objects should also exist. This command should re-establish the dropped object, thereby allowing the data in the data sets (which were retained for a specified number of days) to be accessible.

The actual implementation of this capability is complex and far-reaching. The implementation of such features will alter the manner in which DB2 stores and accesses data. For example, DB2 will need to be augmented with the following abilities:

- The ability to distinguish between active data for available DB2 objects and inactive data for dropped DB2 objects. This can be accomplished via an additional column in the DB2 catalog table SYSIBM.SYSTABLES; for example, ACTIVE with Y indicating an active table and N indicating an inactive table.
- If inactive data exists for a dropped table, the creation of a table with the same name should not be allowed unless the data is either reactivated for that table or deleted. This can be accomplished by extending the CREATE TABLE syntax to include a clause specifying either DROP DATA to delete the retained data or RETAIN DATA to reactivate the data for the new table.
- All DB2 catalog entries should be retained until the data is dropped. This will enable the table (and all related DB2 objects) to be easily recreated.

Finally, a REACTIVATE statement should be provided that requires only the object name; for example, REACTIVATE TABLE SPACE tsname or REACTIVATE TABLE creator.tbname. This would enable tablespaces and tables to be reactivated without issuing full DDL.

Synopsis

DB2 V2.3 adds new and needed security features to DB2's arsenal of authorization tools, yet there is still a long way to go. The six problems addressed in these articles need to be corrected for a proper and user-friendly security architecture to be available within DB2. The solutions presented here, if implemented by IBM, will go a long way toward achieving this goal.

Footnote

[1] *If your organization has implemented secondary authorization routines, then this type of request is easier. However, secondary authorization routines require an exit to an external security package or table of secondary authids. Although most organizations use secondary authids, there are still a fair number of organizations that do not.*

Craig S. Mullins is the author of the recently released DB2 Developer's Guide *published by Prentice-Hall Computer Publishing.*

Was this article of value to you? If so, please let us know by circling Reader Service No. 39.

MANAGEMENT

Boost Competitiveness

With Software Change and Configuration Management

By Tom Burton

Business changes affect organizations that produce, revise and maintain software applications. Buffeted by the volume and complexity of software changes, software developers must balance quality, requirements and deadlines. It is no wonder then that obstacles seem insurmountable at times.

What is needed is a way to automate software changes to minimize disruption and to ensure quality. Fortunately, controlling software changes is feasible and well worth the effort. The solution is software configuration management (SCM).

What Is Software Configuration Management?

If you ask 10 people this question, you may get 10 different answers. Theoretically, SCM is defined as the process of identifying, organizing and managing modifications to software. However, like other computer terms, the definition today is not the same as it was a few years ago.

SCM began in the 1970s. As organizations wrote larger applications and used third-generation languages (3GLs), the need for managing changes to software source code increased. Solutions involved change control and library management. Products to support these activities were introduced. Many of these products—like SCCS, CMS, CA-PANVALET and The CA-LIBRARIAN—are still used today.

As software inventories grew in the 80s, the volume of software changes rose, requiring the software development and maintenance life cycle to be controlled from design to production. Audit requirements also emerged to control and manage changes.

SCM solutions took the form of