What Is A Trigger?

Triggers are a powerful feature that can be provided with DBMS products. They enable nonbypassable, event-driven logic to be intrinsically intermingled with data.

By Craig S. Mullins

Many RDBMSes tout their ability to support triggers. But just what is a trigger? If you have never had the opportunity to utilize them, their power may at first elude you. However, once you have used a DBMS that supports triggers, living without them can be unthinkable.

The Basics

Triggers are event-driven, specialized procedures stored in the RDBMS. Each trigger is attached to a single, specified table. Triggers can be thought of as an advanced form of "rule" or "constraint" written using procedural logic. A trigger cannot be directly called or executed; it is automatically executed (or "fired") by the RDBMS as the result of an action — usually a data modification to the associated table.

Once a trigger is created, it is always executed when its "firing" event occurs (update, insert or delete). Triggers are optional; tables need not have triggers associated with them.

Triggers vs. Stored Procedures

Triggers are similar to stored procedures. Both consist of procedural logic stored at the database level. However, stored procedures are neither event-driven nor attached to a specific table. A stored procedure must be explicitly executed and can access many tables without being tied to any of them.

Why Use Triggers?

Triggers are useful for implementing procedural logic that must be executed on a regular basis due to a predefined event. By implementing triggers, scheduling problems can be eliminated because the trigger will be fired when the triggering event occurs. You need not remember to schedule or code an activity to perform the logic in the trigger. It happens automatically by virtue of being in the trigger.

Triggers can be implemented for many practical uses. Quite often, it is impossible to code business rules into the database using only DDL. For example, the DBMS may not support rules and/or constraints. Using triggers, a flexible environment for implementing business rules in the DBMS is established. This is important because having business rules in the database ensures everyone uses the same logic to accomplish the same process. Triggers are ideal for supporting business rules.

Triggers can be coded to access and/or modify other tables, print informational messages and specify complex restrictions. For example, consider a book publishing application. Triggers can be used to support the following scenarios.

- A business rule exists specifying that no more than two authors are permitted to write a single book title. In this case, a trigger can be coded to check that rows cannot be inserted that violate this requirement.
- A business rule allows only orders for books that have already been published or are being published within the next three weeks.
- The publisher wants to ensure that the royalty amount being paid is equal to the appropriate percentage of the profits. This is especially useful if the royalty amount is stored in the DBMS as redundant data.
- To curb costs, a business decision may be made that salaries are not to rise more than five percent at a time. A trigger can be coded to disallow updating the salary column more than five percent.

When Does A Trigger Fire?

Two options exist for when a trigger can fire: before the firing activity occurs or after the firing activity occurs. Unless you know how the triggers in your RDBMS function, properly functioning triggers cannot be coded.

Consider, for example, if the firing activity occurs before the trigger is fired; the update, insert or delete occurs first and, as a result, the trigger logic is executed. This is the method employed by Sybase SQL Server (Microsoft/Sybase, Inc., Emeryville, CA). In this case, the SQL data modification statement and the trigger code should be thought of as one and the same. If necessary, the trigger code can "roll back" the data modification.

However, what if the trigger is fired before the actual firing event occurs? In this situation, a rollback would not be required for the firing event code because it did not yet occur. However, a rollback may be required for any data modifications that occurred prior to this firing event within the same transaction.

Determining the procedural activity required when triggers are present can be a complicated task. It is important that all developers are schooled in the firing methods utilized for triggers by the DBMS product being used.

Do Triggers Always Fire?

Once created, do triggers always fire when the appropriate data modification occurs? This is usually the case. However, there may be exceptions in certain circumstances. For example, Sybase SQL Server does not fire insert triggers when the Bulk Copy (BCP) utility is used to load data. This may cause data integrity problems.

Additionally, it is possible triggers may fire whether any rows were actually affected or not. For example, consider the following SQL statement:

```
update titles
set price = price * .05
where type = "business"
```

It is possible that no business-type books currently exist in the titles table. However, if a trigger exists on the titles table, it would still be fired if this update were to be issued. In this situation, the trigger must be intelligent enough to determine that no rows were actually updated.

Types Of Triggers

In general, most DBMSes that support triggers provide three types:

- Insert — to be invoked when any row is inserted to the table
- Update — to be invoked when any column in the table is updated
- Delete — to be invoked when any row is deleted from the table.

Each table in the database can have up to three triggers, one each for insert, update and delete. It is usually possible (though not advisable) to create a single trigger that contains the firing logic for insert, update and delete. However, this is generally more difficult to administer, maintain and support.

Typically a single trigger can be used for:

- Any single action (insert or update or delete)
- Insert/update
- Insert/delete
- Update/delete
- Insert/update/delete.

Complex Triggers

The possibilities for triggers and the type of support they can provide are virtually endless. In addition to the basic insert, update and delete triggers, consider some of these other potential uses:

- Fired based on DDL (index created, table dropped, etc.)
- Fired based on time of day
- Fired based on system activity (tape received, user logs in, etc.)
- Fired based on storage constraints (database utilizing 90 percent of allocated disk space, etc.).

Before developing applications with a DBMS you may currently own (or before purchasing a new DBMS), be sure to analyze the trigger specifications to determine if these types of more complex triggers are supported.

Triggers Can Fire Other Triggers

A trigger is generally fired by an insert, update or delete. A trigger, however, can also contain within itself insert, update and delete logic. Therefore, a trigger is fired by a data modification but can also cause another data modification, thereby firing yet another trigger. When a trigger contains insert, update and/or delete logic, the trigger is said to be a nested trigger.

TABLE 1

Referential Integrity Rules

DELETE RESTRICT	If any rows exist in the dependent table, the primary key row in the parent table cannot be deleted.
DELETE CASCADE	If any rows exist in the dependent table, the primary key row in the parent table is deleted, and all dependent rows are also deleted.
DELETE NEUTRALIZE	If any rows exist in the dependent table the primary key row in the parent table is deleted, and the foreign key column(s) for all dependent rows are set to NULL as well.
UPDATE RESTRICT	If any rows exist in the dependent table, the primary key column(s) in the parent table cannot be updated.
UPDATE CASCADE	If any rows exist in the dependent table, the primary key column(s) in the parent table are updated, and all foreign key values in the dependent rows are updated to the same value.
UPDATE NEUTRALIZE	If any rows exist in the dependent table, the primary key row in the parent table is deleted, and all foreign key values in the dependent rows are updated to NULL as well.
INSERT RESTRICT	A foreign key value cannot be inserted into the dependent table unless a primary key value already exists in the parent table.
FK UPDATE RESTRICTION	A foreign key cannot be updated to a value that does not already exist as a primary key value in the parent table.
PENDANT DELETE	When the last foreign key value in the dependent table is deleted the primary key row in the parent table is also deleted.

Most DBMSes place a limit on the number of nested triggers that can be executed within a single firing event. If this were not done, it could be possible to have triggers firing triggers ad infinitum until all of the data was removed from an entire database. The limit for Sybase SQL Server is 16 levels of trigger nesting. If more than 16 levels of nesting occur, the transaction is aborted.

The ability to nest triggers provides an efficient method for implementing automatic data integrity. Because triggers generally cannot be bypassed, they provide an elegant solution to the enforced application of business rules. Use caution, however, to ensure the maximum trigger nesting level is not reached. Failure to heed this advice can cause an environment where certain types of updates cannot occur.

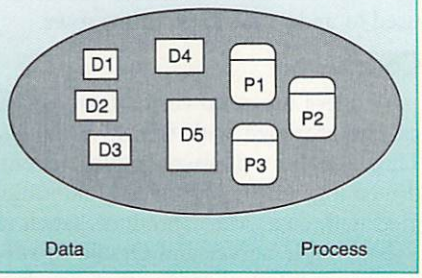
Primitive Encapsulation

Object-Oriented (OO) technology espouses the concept of encapsulation. In an ODBMS, an entity is defined in terms of both its state and its behavior. In other words, an object encapsulates both the data (state) and the valid procedures that can be performed on the data (behavior) (see Figure 1).

In the OO paradigm, messages are passed to objects invoking the encapsulated methods. Because each object contains its own operations, or methods, most of the procedural code is eliminated.

FIGURE 1

Encapsulation



EXAMPLE 1

Sample Trigger

```
create trigger title_upd
on titles for update
as

declare @num_row int
select @num_row = @@rowcount

if update (title_id)
begin

    if @num_row > 1
    begin
        print "Cannot update more than one PK row!"
        rollback transaction
        return
    end

    update titleauthor
        set titleauthor.titleid = NULL
    from titleauthor, inserted, deleted
    where titleauthor.titleid = deleted.title_id

end
return
```

Triggers can be thought of as a primitive form of encapsulation. Each trigger is fixed to a specific table. The trigger is, in essence, stored within the table as surely as the data is stored there.

Do not confuse the issue — there is much more to the concept of encapsulation than simply supporting triggers. The extent of this difference, however, is beyond the scope of this article.

Using Triggers To Implement Referential Integrity

One of the primary uses for triggers is to support Referential Integrity (RI). Although many DBMSes support a form of declarative RI, no current DBMS fully supports all possible referential constraints. See Table 1 for a list of these possibilities.

Triggers can be coded, in lieu of declarative RI, to support all of the RI rules. Of course, using triggers necessitates writing procedural code for each rule for each constraint, whereas declarative RI constraints are coded in the DDL used to create relational tables.

Consider the following trigger capabilities.

- A delete trigger on the parent table can be used to code:
  - Delete restrict
  - Delete cascade
  - Delete neutralize.
- An update trigger on the parent table can be used to code:
  - Update restrict
  - Update cascade
  - Update neutralize.
- An insert trigger on the dependent table can be used to code:
  - Insert restrict.
- An update trigger on the dependent table can be used to code the restriction that a foreign key cannot be updated to a nonprimary key value.
- A delete trigger can be coded on the dependent table to support a pendant RI constraint.

Referential Considerations

To use triggers to support RI rules, it

is sometimes necessary to know the values impacted by the action that fired the trigger. For example, consider the case where a trigger is fired because a row was deleted. The row, and all of its values, has already been deleted because the trigger is executed after its firing action occurs. If this is the case, however, how can you determine if referentially connected rows exist with those values?

One solution offered by Sybase SQL Server provides two specialized tables with each trigger:

- The INSERTED table
- The DELETED table.

Each trigger has one INSERTED table and one DELETED table available. These tables are accessible only from triggers. They provide access to the modified data by viewing information in the transaction log. The transaction log is a record of all data modification activity, automatically maintained by the DBMS.

When an INSERT occurs, the INSERTED table contains the rows just inserted into the table to which the trigger is attached. When a DELETE occurs, the DELETED table contains the rows just deleted from the table to which the trigger is attached. An UPDATE statement logically functions as a DELETE followed by an INSERT. Therefore, after an UPDATE, the INSERTED table contains the new values for the rows just updated in the table to which the trigger is attached. The DELETED table contains the old values for the updated rows.

Therefore, the trigger can use these specialized INSERTED and DELETED tables to query the affected data. Remember, too, that SQL data modification can happen a set-at-a-time. One DELETE or UPDATE statement can impact zero, one or many rows. This must be taken into account when coding triggers.

Coding A Trigger

Browse the sample trigger depicted in Example 1. This is an example of an SQL Server trigger. It is an update trigger, coded on the titles table. This trigger implements neutralizing updates. When the title_id column is updated, the foreign key values in the dependent table (titleauthor) are set to NULL.

Take special note of how the trigger checks to make sure that the title_id was actually modified. Also, notice how the INSERTED and DELETED tables are used to update the appropriate rows.

Conclusion

Triggers are a powerful feature that can be provided with DBMS products. They enable nonbypassable, event-driven logic to be intrinsically intermingled with data. Many products (such as Sybase SQL Server and Oracle 7) support triggers today; many others (such as DB2) will support them in the near future. It is a wise course of action to learn about triggers and what they can provide today, before it is too late!

ABOUT THE AUTHOR

Craig S. Mullins is a senior education specialist at Platinum technology, inc., where he develops courseware for Sybase SQL Server, the DB2 family and client/server. Platinum technology, inc., 1815 S. Meyers Rd., Oakbrook Terrace, IL 60181, (708) 620-5000. He can be reached through CompuServe, 70410,237.