Multiple event triggers support on IBM DB2 for i

Sarah Mackenzie
Xing Xing Shen
John Broich

June 12, 2013

This article explains how to use multiple event triggers to have a single SQL source trigger handle multiple events for a table or view. The support is intended to make it easier for IT shops that use trigger programs to manage and maintain those triggers. And, the syntax and semantics of the enhanced SQL support for triggers match the support on the other IBM® DB2® platforms to provide compatibility and portability across databases.

Overview

For customers who use SQL triggers to help manage their business, there is a new IBM DB2 for i enhancement that makes it easier to manage and maintain those triggers. IBM i 7.1 Technology Refresh 6 delivers support for multiple event triggers, which allows a single SQL trigger to handle a combination of trigger events (INSERT, UPDATE, and DELETE). With the new support, it is now possible to have only one source SQL trigger that needs to be updated to accommodate future changes to your database and business requirements.

The support introduces trigger predicates and new keywords that can be used to control the logic flow in the trigger based on the event that caused the trigger to be invoked. The new predicates, INSERTING, UPDATING, and DELETING, are self-explanatory and refer to the event that caused the trigger to be fired. There is also a new SQL syntax added to the CREATE TRIGGER statement to enable multiple events to be defined for the trigger. That syntax allows for the specification of the events that can be handled by the trigger.

Usage of the SQL multiple event trigger support requires that DB2 PTF Group SF99701 level 23 be applied on your system (see SF99701 Level 22 enhancements).

Multiple event trigger examples

Here are a couple of examples that show the usage of the trigger predicates and the new syntax of the CREATE TRIGGER statement.

In the first example, assume that you have a table named *employee* to track employee information. This table contains two columns, among others, named *edlevel* and *salary*. Also, assume that the

table currently has two BEFORE trigger programs defined for it, one for INSERT and the other for UPDATE that can adjust or set the *salary* column based on the *edlevel*.

Here is the BEFORE INSERT trigger:

Listing 1. BEFORE INSERT trigger insert_set_salary

```
CREATE OR REPLACE TRIGGER insert_set_salary
  NO CASCADE
  BEFORE INSERT ON employee
  REFERENCING NEW AS N FOR EACH ROW
  -- Give starting salary based on education level.
  BEGIN
    SET n.salary = CASE n.edlevel WHEN 18 THEN 50000
                                  WHEN 16 THEN 40000
                                  ELSE 25000
    END;
  END
```

Here is the BEFORE UPDATE trigger:

Listing 2. BEFORE UPDATE trigger update_set_salary

```
CREATE OR REPLACE TRIGGER update_set_salary
  NO CASCADE
  BEFORE UPDATE ON employee
REFERENCING NEW AS n OLD AS o
  FOR EACH ROW
  WHEN( n.edlevel > o.edlevel )
  -- Give 10% raise to existing employees with new education level.
  BEGIN
    SET N.SALARY = N.SALARY * 1.1;
  END
```

To replace those triggers with a single SQL trigger that gets fired for either an UPDATE or INSERT statement to the *employee* table, use the trigger predicates in the body of the trigger and the new syntax on the CREATE TRIGGER statement to indicate that multiple events are to be handled in the trigger.

To indicate the events that are to be handled in the trigger, the CREATE TRIGGER statement syntax allows for the specification of the events handled by the trigger. These events are separated by the OR keyword. In the multiple event trigger (shown in Listing 3), the SQL trigger is defined to be fired before both, update and insert, events.

The multiple event trigger is shown below. Note that in the first example of a multiple event trigger, the new syntax is shown in bold format.

Listing 3. Multiple event trigger set_salary

```
CREATE OR REPLACE TRIGGER SET_SALARY
  NO CASCADE
  BEFORE UPDATE OR INSERT ON employee
  REFERENCING NEW AS n OLD AS o
  FOR EACH ROW
  WHEN( o.edlevel IS NULL OR n.edlevel > o.edlevel )
  BEGIN
    -- Give 10% raise to existing employees with new education level.
    IF UPDATING THEN SET n.salary = n.salary * 1.1;
    -- Give starting salary based on education level.
    ELSEIF INSERTING THEN
      SET n.salary = CASE n.edlevel WHEN 18 THEN 50000
                                    WHEN 16 THEN 40000
                                    ELSE 25000
      END;
    END IF;
  END
```

In this example, note that there is a case where the transition variables passed to the trigger will not be relevant (the old row image for an INSERT statement, as there is no 'old' row). For cases where the row image passed in the transition variables is not relevant (the other case is the 'new' row for a DELETE), each transition variable in the irrelevant row contains the NULL value.

Here is another example where all three events (INSERT, UPDATE, and DELETE) are handled in a single SQL trigger.

Column *prod_count* in table *company_stats* contains the current number of different products manufactured by this company. In the following example, the multiple event trigger:

- Increments the number of products each time a new product is developed.
- Decrements the number of products each time the company discontinues selling a product.
- Returns an error when the price of a product changes (increase or decrease) by more than 10 percent.

Listing 4. Multiple event trigger prod_maintain

```
CREATE TRIGGER prod_maintain
AFTER INSERT OR DELETE OR UPDATE OF prod_price ON PRODUCTS
 REFERENCING NEW AS n OLD AS o FOR EACH ROW
 BEGIN
    IF INSERTING
THEN UPDATE products SET prod_count = prod_count + 1;
 END IF;
 IF DELETING
 THEN UPDATE products SET prod_count = prod_count - 1;
 END IF;
    IF UPDATING AND
       (n.prod_price > 1.1 * o.prod_price OR
        n.prod_price < 0.9 * o. prod_price)
THEN SIGNAL SQLSTATE '75000'
 SET MESSAGE_TEXT = 'Price change > 10%, attention!';
 END IF;
  END
```

For this example, a distinct logic is ran for each of the events that the trigger program handles. For an INSERT and DELETE, the corresponding update of *prod_count* in table *products* occurs; for an UPDATE, an error condition is signaled by the trigger if the price change exceeds a threshold.

One thing to note about the trigger predicates (INSERTING, DELETING, UPDATING) is that they can be used in control statements (similar to the prior examples) or within any SQL statement where a predicate (such as SELECT or UPDATE) can be specified, provided the statement containing the predicate is inside an SQL trigger. In the next example, the trigger event predicates are used in an INSERT statement to generate a row for the *product_history* table.

Listing 5. Table product_history

```
CREATE TABLE product_history(prod_id INT,
                             posttime TIMESTAMP,
                             change_type CHAR(1))
```

Listing 6. Multiple event trigger set_prod_hist

```
CREATE TRIGGER set_prod_hist
  AFTER INSERT OR UPDATE OR DELETE ON products
  REFERENCING NEW AS n OLD AS o
  FOR EACH ROW MODE DB2SQL
  BEGIN
    INSERT INTO product_history VALUES (
      CASE
         WHEN INSERTING OR UPDATING
           THEN n.prod_id
         WHEN DELETING
           THEN o.prod_id
      END,
      CURRENT TIMESTAMP,
      CASE
        WHEN INSERTING
          THEN 'I'
        WHEN UPDATING
          THEN 'U'
        WHEN DELETING
          THEN 'D'
      END
    );
  END
```

For this trigger, the same routine logic is used by all three trigger events. The trigger event predicates are mostly used to determine whether the product ID needs to be read from the previous or next row value and a second time to set the type of the update.

The next example shows how an SQL trigger might be used by a grocery store to track additions, deletions, and changes to a price look-up (PLU) code tracking table when there are changes to the produce that the store sells (the PLU is located on a small sticker on fruits and vegetables sold individually in grocery stores). In this simple example, a different procedure is called based upon the event that fired the trigger. The business logic to handle those events is inside the respective procedure.

Listing 7. Multiple event trigger plu_track

```
CREATE TRIGGER plu_track
BEFORE INSERT OR UPDATE OR DELETE ON prod_plu
  REFERENCING NEW AS n OLD AS o FOR EACH ROW
  BEGIN
    IF INSERTING THEN
      CALL prod_plu_ins(n.plu);
    ELSEIF DELETING THEN
      CALL prod_plu_del(o.plu);
    ELSE
      CALL prod_plu_upd(n.plu);
    END IF;
  END
```

Determining the trigger type

The *qsys2.systriggers* catalog view was changed to reflect the new multiple event trigger support. The existing column *event_manipulation* contains a new value, 'MULTI', if the trigger is a multiple event trigger. In addition, three new CHAR(1) columns, *eventinsert*, *eventupdate*, and *eventdelete* contain a 'Y' if the trigger program was created with a corresponding trigger event predicate.

When DB2 for i PTF Group SF99701 Level 23 is installed, the *qsys2.systriggers* catalog view will have contents as described in the following table:

Table 1. *systriggers* values for trigger programs

Operations specified in event clause of trigger	Value of the EVENT_MANIPULATION column	Value of the EVENTINSERT column	Value of the EVENTUPDATE column	Value of the EVENTDELETE column
INSERT	INSERT	Y	N	N
UPDATE	UPDATE	N	Y	N
DELETE	DELETE	N	N	Y
INSERT, UPDATE	MULTI	Y	Y	N
INSERT, DELETE	MULTI	Y	N	Y
UPDATE, DELETE	MULTI	N	Y	Y
INSERT, UPDATE, DELETE	MULTI	Y	Y	Y

In the set_salary trigger example from above, when two single event triggers are created a separate record for each trigger is written to the *qsys2.systriggers* catalog view.

Figure 1. Single event triggers in the *qsys2.systriggers* catalog view

TRIGGER_NAME	EVENT_MANIPULATION	EVENTUPDATE	EVENTINSERT	EVENTDELETE
INSERT_SET_SALARY	INSERT	N	Y	N
UPDATE_SET_SALARY	UPDATE	Y	N	N

When the two single event triggers are combined into one multiple event trigger, only one record is written to the *qsys2.systriggers* catalog view.

Figure 2. Multiple event triggers in the *qsys2.systriggers* catalog view

TRIGGER_NAME	EVENT_MANIPULATION	EVENTUPDATE	EVENTINSERT	EVENTDELETE
SET_SALARY	MULTI	Y	Y	N

Coexistence with native I/O and existing CL commands

As you would expect with the integrated database on IBM i, the existing Control Language (CL) commands can be used with the program object created for a SQL multiple event trigger. These commands have common sense behavior when working with these programs. For example, the remove physical file trigger (RMVPFTRG) command can be used to remove a multiple event trigger program from a file, but must remove all events that are handled by that trigger program. For this reason, the TRGEVENT(*ALL) parameter must be used when removing such a trigger program.

Considerations for multiple event triggers

Although the trigger program can handle multiple events, all the events must be defined to be fired at the same time. The SQL trigger must be defined as either BEFORE or AFTER for all of the events.

As part of this support, a loosely related trigger program restriction was removed. Before this support, an SQL BEFORE trigger could not contain an SQL statement that did any committable work, such as an INSERT, CREATE TABLE, and so on. With this new support, this restriction is removed and the before trigger program can contain these statements.

Without the new support, an attempt to create an SQL BEFORE trigger that included one of these statements would have caused a SQLSTATE of 42987 to be returned on the CREATE TRIGGER statement.

Also, note that prior to this new support, there was also a QAQQINI option, SQL_MODIFIES_SQL_DATA, which allowed the trigger to contain SQL statements to perform committable work. Because the restriction that this option provided a work around for was removed with the new support, the QAQQINI option will be ignored but tolerated when set to '*NO' , as it is no longer relevant.

New and old transition variables are always passed to a multiple event trigger, but for some operations they are not relevant, and therefore, contain the null value. For example, in the set_salary trigger, transition variables representing the old row "O" would be NULL in INSERTING operations.

The new trigger predicates can only be used in a SQL trigger, and they cannot be referenced in an SQL function or procedure or any other SQL statement outside of a trigger. However, that trigger program does not need to be a multiple event trigger program to use the predicates; it can be a trigger that only handles a single event.

Summary

Multiple event triggers provide a means to have a single SQL source trigger handle multiple events for a table or view. The support is intended to make it easier for IT shops that use triggers to

manage and maintain those triggers. Additionally, the syntax and semantics of the enhanced SQL support for triggers match the support on the other DB2 platforms to provide compatibility and portability across databases. The multiple event SQL trigger support is available beginning with the IBM i 7.1 TR6 release. So, if it a good fit for your DB2 for i database environment, give it a try.

Resources:

- This link takes you to the update SQL programming guide description of the support for multiple event triggers. V7R1 SQL Programming Guide for multiple event triggers
- This link takes you to the CREATE TRIGGER statement syntax in the SQL reference showing the new syntax for the statement to support multiple event triggers. V7R1 CREATE TRIGGER Statement