Creating and Using Triggers

When to Use Triggers ............................................. 15-3
How to Create a Trigger .......................................... 15-4
  Assigning a Trigger Name ...................................... 15-5
  Specifying the Trigger Event ................................. 15-5
  Defining the Triggered Actions .............................. 15-6
  A Complete CREATE TRIGGER Statement .................... 15-7
Using Triggered Actions ........................................... 15-7
  Using BEFORE and AFTER Triggered Actions ............... 15-7
  Using FOR EACH ROW Triggered Actions ..................... 15-9
    Using the REFERENCING Clause ............................. 15-9
    Using the WHEN Condition .................................. 15-10
  Using SPL Routines as Triggered Actions .................. 15-11
    Passing Data to a SPL Routine ............................. 15-11
    Using SPL Procedure Language ............................. 15-12
    Updating Nontriggering Columns with Data from an SPL Routine ........................................ 15-12
Tracing Triggered Actions ........................................ 15-13
Generating Error Messages ...................................... 15-14
  Applying a Fixed Error Message ............................. 15-14
  Generating a Variable Error Message ....................... 15-16
Summary ................................................................... 15-17
An SQL trigger is a mechanism that resides in the database. It is available to any user who has permission to use it. It specifies that when a particular action, an insert, a delete, or an update, occurs on a particular table, the database server should automatically perform one or more additional actions. The additional actions can be INSERT, DELETE, UPDATE, EXECUTE PROCEDURE, or EXECUTE FUNCTION statements.

This chapter describes the purpose of each component of the CREATE TRIGGER statement, illustrates some uses for triggers, and describes the advantages of using an SPL routine as a triggered action.

---

**When to Use Triggers**

Because a trigger resides in the database and anyone who has the required privilege can use it, a trigger lets you write a set of SQL statements that multiple applications can use. It lets you avoid redundant code when multiple programs need to perform the same database operation.

You can use triggers to perform the following actions as well as others that are not found in this list:

- Create an audit trail of activity in the database. For example, you can track updates to the orders table by updating corroborating information to an audit table.
- Implement a business rule. For example, you can determine when an order exceeds a customer’s credit limit and display a message to that effect.
How to Create a Trigger

- Derive additional data that is not available within a table or within the database. For example, when an update occurs to the quantity column of the items table, you can calculate the corresponding adjustment to the total_price column.
- Enforce referential integrity. When you delete a customer, for example, you can use a trigger to delete corresponding rows (that is, rows that have the same customer number) in the orders table.

How to Create a Trigger

You use the CREATE TRIGGER statement to create a trigger. The CREATE TRIGGER statement is a data definition statement that associates SQL statements with a precipitating action on a table. When the precipitating action occurs, the associated SQL statements, which are stored in the database, are triggered. Figure 15-1 illustrates the relationship of the precipitating action, or trigger event, to the triggered action.

The CREATE TRIGGER statement consists of clauses that perform the following actions:

- Assign a trigger name.
- Specify the trigger event, that is, the table and the type of action that initiate the trigger.
- Define the SQL actions that are triggered.
Assigning a Trigger Name

You can create a trigger using DB-Access, the SQL Editor, or one of the SQL APIs. This section describes the CREATE TRIGGER statement as you would enter it using the interactive Query-language option in DB-Access. In an SQL API, you simply precede the statement with the symbol or keywords that identify it as an embedded statement.

### Assigning a Trigger Name

The trigger name identifies the trigger. It follows the words CREATE TRIGGER in the statement. It can be up to 18 characters long, beginning with a letter and consisting of letters, the digits 0 to 9, and the underscore. In the following example, the portion of the CREATE TRIGGER statement that is shown assigns the name `upqty` to the trigger:

```
CREATE TRIGGER upqty -- assign trigger name
```

### Specifying the Trigger Event

The trigger event is the type of statement that activates the trigger. When a statement of this type is performed on the table, the database server executes the SQL statements that make up the triggered action. The trigger event can be an INSERT, DELETE, or UPDATE statement. When you define an UPDATE trigger event, you can name one or more columns in the table to activate the trigger. If you do not name any columns, then an update of any column in the table activates the trigger. You can create only one INSERT and one DELETE trigger per table, but you can create multiple UPDATE triggers as long as the triggering columns are mutually exclusive.

In the following excerpt of a CREATE TRIGGER statement, the trigger event is defined as an update of the `quantity` column in the `items` table:

```
CREATE TRIGGER upqty
UPDATE OF quantity ON items -- an UPDATE trigger event
```
Defining the Triggered Actions

This portion of the statement identifies the table on which you create the trigger. If the trigger event is an insert or delete, only the type of statement and the table name are required, as the following example shows:

```
CREATE TRIGGER ins_qty
    INSERT ON items  -- an INSERT trigger event
```

**Defining the Triggered Actions**

The *triggered actions* are the SQL statements that are performed when the trigger event occurs. The triggered actions can consist of INSERT, DELETE, UPDATE, EXECUTE PROCEDURE, or EXECUTE FUNCTION statements. In addition to specifying what actions are to be performed, however, you must also specify *when* they are to be performed in relation to the triggering statement. You have the following choices:

- Before the triggering statement executes
- After the triggering statement executes
- For each row that is affected by the triggering statement

A single trigger can define actions for each of these times.

You define a triggered action by specifying when it occurs and then providing the SQL statement or statements to execute. You specify when the action is to occur with the keywords BEFORE, AFTER, or FOR EACH ROW. The triggered actions follow, enclosed in parentheses. The following triggered action definition specifies that the SPL routine `upd_items_p1` is to be executed before the triggering statement:

```
BEFORE(EXECUTE PROCEDURE upd_items_p1)  -- a BEFORE action
```
A Complete CREATE TRIGGER Statement

If you combine the trigger-name clause, the trigger-event clause, and the triggered-action clause, you have a complete CREATE TRIGGER statement. The following CREATE TRIGGER statement is the result of combining the components of the statement from the preceding examples. This trigger executes the SPL routine `upd_items_p1` whenever the `quantity` column of the `items` table is updated.

```sql
CREATE TRIGGER upqty
UPDATE OF quantity ON items
BEFORE(EXECUTE PROCEDURE upd_items_p1)
```

If a database object in the trigger definition, such as the SPL routine `upd_items_p1` in this example, does not exist when the database server processes the CREATE TRIGGER statement, it returns an error.

Using Triggered Actions

To use triggers effectively, you need to understand the relationship between the triggering statement and the resulting triggered actions. You define this relationship when you specify the time that the triggered action occurs; that is, BEFORE, AFTER, or FOR EACH ROW.

Using BEFORE and AFTER Triggered Actions

Triggered actions that occur before or after the trigger event execute only once. A BEFORE triggered action executes before the triggering statement, that is, before the occurrence of the trigger event. An AFTER triggered action executes after the action of the triggering statement is complete. BEFORE and AFTER triggered actions execute even if the triggering statement does not process any rows.
Among other uses, you can use BEFORE and AFTER triggered actions to determine the effect of the triggering statement. For example, before you update the \texttt{quantity} column in the \texttt{items} table, you could call the SPL routine \texttt{upd\_items\_p1}, as the following example shows, to calculate the total quantity on order for all items in the table. The routine stores the total in a global variable called \texttt{old\_qty}.

```sql
CREATE PROCEDURE upd_items_p1()
DEFINE GLOBAL old_qty INT DEFAULT 0;
LET old_qty = (SELECT SUM(quantity) FROM items);
END PROCEDURE;
```

After the triggering update completes, you can calculate the total again to see how much it has changed. The following SPL routine, \texttt{upd\_items\_p2}, calculates the total of \texttt{quantity} again and stores the result in the local variable \texttt{new\_qty}. Then it compares \texttt{new\_qty} to the global variable \texttt{old\_qty} to see if the total quantity for all orders has increased by more than 50 percent. If so, the routine uses the RAISE EXCEPTION statement to simulate an SQL error.

```sql
CREATE PROCEDURE upd_items_p2()
DEFINE GLOBAL old_qty INT DEFAULT 0;
DEFINE new_qty INT;
LET new_qty = (SELECT SUM(quantity) FROM items);
IF new_qty > old_qty * 1.50 THEN
    RAISE EXCEPTION -746, 0, 'Not allowed - rule violation';
END IF
END PROCEDURE;
```

The following trigger calls \texttt{upd\_items\_p1} and \texttt{upd\_items\_p2} to prevent an extraordinary update on the \texttt{quantity} column of the \texttt{items} table:

```sql
CREATE TRIGGER up_items
UPDATE OF quantity ON items
BEFORE(EXECUTE PROCEDURE upd_items_p1())
AFTER(EXECUTE PROCEDURE upd_items_p2());
```

If an update raises the total quantity on order for all items by more than 50 percent, the RAISE EXCEPTION statement in \texttt{upd\_items\_p2} terminates the trigger with an error. When a trigger fails in INFORMIX-Universal Server and the database has logging, the database server rolls back the changes made by both the triggering statement and the triggered actions. For more information on what happens when a trigger fails, see CREATE TRIGGER in the \textit{Informix Guide to SQL: Syntax}. 

Using FOR EACH ROW Triggered Actions

A FOR EACH ROW triggered action executes once for each row that the triggering statement affects. For example, if the triggering statement has the following syntax, a FOR EACH ROW triggered action executes once for each row in the items table in which the manu_code column has a value of ‘KAR’:

```sql
UPDATE items SET quantity = quantity * 2 WHERE manu_code = 'KAR'
```

If the triggering statement does not process any rows, a FOR EACH ROW triggered action does not execute.

Using the REFERENCING Clause

When you create a FOR EACH ROW triggered action, you must usually indicate in the triggered action statements whether you are referring to the value of a column before or after the effect of the triggering statement. For example, imagine that you want to track updates to the quantity column of the items table. To do this, you create the following table to record the activity:

```sql
CREATE TABLE log_record
(item_num SMALLINT,
 ord_num INTEGER,
 username CHARACTER(8),
 update_time DATETIME YEAR TO MINUTE,
 old_qty SMALLINT,
 new_qty SMALLINT);
```

To supply values for the old_qty and new_qty columns in this table, you must be able to refer to the old and new values of quantity in the items table; that is, the values before and after the effect of the triggering statement. The REFERENCING clause enables you to do this.

The REFERENCING clause lets you create two prefixes that you can combine with a column name, one to reference the old value of the column and one to reference its new value. These prefixes are called correlation names. You can create one or both correlation names, depending on your requirements. You indicate which one you are creating with the keywords OLD and NEW. The following REFERENCING clause creates the correlation names pre_upd and post_upd to refer to the old and new values in a row:

```sql
REFERENCING OLD AS pre_upd NEW AS post_upd
```
Using FOR EACH ROW Triggered Actions

The following triggered action creates a row in log_record when quantity is updated in a row of the items table. The INSERT statement refers to the old values of the item_num and order_num columns and to both the old and new values of the quantity column.

```
FOR EACH ROW(INSERT INTO log_record
    VALUES (pre_upd.item_num, pre_upd.order_num, USER, CURRENT,
            pre_upd.quantity, post_upd.quantity));
```

The correlation names defined in the REFERENCING clause apply to all rows affected by the triggering statement.

**Important:** If you refer to a column name in the triggering table without using a correlation name, the database server makes no special effort to search for the column in the definition of the triggering table. You must always use a correlation name with a column name in SQL statements within a FOR EACH ROW triggered action, unless the statement is valid independent of the triggered action. For more information, see CREATE TRIGGER in the “Informix Guide to SQL: Syntax.”

**Using the WHEN Condition**

As an option, you can precede a triggered action with a WHEN clause to make the action dependent on the outcome of a test. The WHEN clause consists of the keyword WHEN followed by the condition statement given in parentheses. In the CREATE TRIGGER statement, the WHEN clause follows the keywords BEFORE, AFTER, or FOR EACH ROW and precedes the triggered-action list.

When a WHEN condition is present, if it evaluates to true, the triggered actions execute in the order in which they appear. If the WHEN condition evaluates to false or unknown, the actions in the triggered-action-action list do not execute. If the trigger specifies FOR EACH ROW, the condition is evaluated for each row also.
Using SPL Routines as Triggered Actions

In the following trigger example, the triggered action executes only if the condition in the WHEN clause is true; that is, if the post-update unit price is greater than two times the pre-update unit price:

```sql
CREATE TRIGGER up_price
UPDATE OF unit_price ON stock
REFERENCING OLD AS pre NEW AS post
FOR EACH ROW WHEN(post.unit_price > pre.unit_price * 2)
    (INSERT INTO warn_tab VALUES(pre.stock_num, pre.order_num,
                              pre.unit_price, post.unit_price, CURRENT))
```

For more information on the WHEN condition, see CREATE TRIGGER in the Informix Guide to SQL: Syntax.

Using SPL Routines as Triggered Actions

Probably the most powerful feature of triggers is the ability to call an SPL routine as a triggered action. The EXECUTE FUNCTION statement, which calls an SPL routine, lets you pass data from the triggering table to the SPL routine and also to update the triggering table with data that the SPL routine returns. SPL also lets you define variables, assign data to them, make comparisons, and use procedural statements to accomplish complex tasks within a triggered action.

**Passing Data to a SPL Routine**

You can pass data to an SPL routine in the argument list of an EXECUTE PROCEDURE or EXECUTE FUNCTION statement. Figure 15-2 shows an EXECUTE FUNCTION statement that passes values from the `quantity` and `total_price` columns of the `items` table to the SPL routine `calc_totpr`:

```sql
CREATE TRIGGER upd_totpr
UPDATE OF quantity ON items
REFERENCING OLD AS pre_upd NEW AS post_upd
FOR EACH ROW EXECUTE FUNCTION calc_totpr(pre_upd.quantity,
                                        post_upd.quantity, pre_upd.total_price) INTO total_price
```

Passing data to an SPL routine lets you use the data in the operations that the routine performs.
Using SPL Routines as Triggered Actions

Using SPL Procedure Language

Figure 15-3 shows the SPL routine calc_totpr, which is executed in the trigger example shown in Figure 15-2. SPL is used in the calc_totpr routine to calculate the change that needs to be made to the total_price column when quantity is updated in the items table.

```
CREATE FUNCTION calc_totpr(old_qty SMALLINT, new_qty SMALLINT,
   total MONEY(8)) RETURNING MONEY(8);
DEFINE u_price LIKE items.total_price;
DEFINE n_total LIKE items.total_price;
LET u_price = total / old_qty;
LET n_total = new_qty * u_price;
RETURN n_total;
END FUNCTION;
```

The calc_totpr routine receives both the old and new values of quantity and the old value of total_price and performs the following operations:

- Divides the old total price by the old quantity to derive the unit price.
- Multiplies the unit price by the new quantity to obtain the new total price.
- Returns the new total price.

The EXECUTE FUNCTION statement in the trigger example of Figure 15-2 calls the calc_totpr routine. In this example, SPL allows the trigger to derive data (from the calc_totpr routine) that is not directly available from the triggering table.

Updating Nontriggering Columns with Data from an SPL Routine

Within a triggered action, the INTO clause of the EXECUTE FUNCTION statement lets you update nontriggering columns in the triggering table. The EXECUTE FUNCTION statement in the following example calls the calc_totpr SPL routine that contains an INTO clause, which references the column total_price:

```
FOR EACH ROW(EXECUTE FUNCTION calc_totpr(pre_upd.quantity,
   post_upd.quantity, pre_upd.total_price) INTO total_price);
```

The value that is updated into total_price is returned by the RETURN statement at the conclusion of the SPL routine. The total_price column is updated for each row that the triggering statement affects.
Tracing Triggered Actions

If a triggered action does not behave as you expect, place it in an SPL routine, and use the SPL TRACE statement to monitor its operation. Before starting the trace, you must direct the output to a file with the SET DEBUG FILE TO statement. The following example shows TRACE statements that have been added to the SPL routine `items_pct`. The SET DEBUG FILE TO statement directs the trace output to the file `/usr/mydir/trig.trace`. The TRACE ON statement begins tracing the statements and variables within the routine.

```sql
CREATE PROCEDURE items_pct(mac CHAR(3))
DEFINE tp MONEY;
DEFINE mc_tot MONEY;
DEFINE pct DECIMAL;
SET DEBUG FILE TO '/usr/mydir/trig.trace';
TRACE 'begin trace';
TRACE ON;
LET tp = (SELECT SUM(total_price) FROM items);
LET mc_tot = (SELECT SUM(total_price) FROM items
WHERE manu_code = mac);
LET pct = mc_tot / tp;
IF pct > .10 THEN
   RAISE EXCEPTION -745;
END IF
TRACE OFF;
END PROCEDURE;

CREATE TRIGGER items_ins
INSERT ON items
REFERENCING NEW AS post_ins
FOR EACH ROW(EXECUTE PROCEDURE items_pct (post_ins.manu_code));
```

The following example shows sample trace output from the `items_pct` routine as it appears in the file `/usr/mydir/trig.trace`. The output reveals the values of routine variables, routine arguments, return values, and error codes.

```
trace expression :begin trace
trace on
expression:
   (select (sum total_price)
   from items)
evaluates to $18280.77 ;
let tp = $18280.77
expression:
   (select (sum total_price)
   from items
   where (= manu_code, mac))
evaluates to $3008.00 ;
let mc_tot = $3008.00
expression:// mc_tot, tp)
```
Generating Error Messages

When a trigger fails because of an SQL statement, the database server returns the SQL error number that applies to the specific cause of the failure.

When the triggered action is an SPL routine, you can generate error messages for other error conditions by using one of two reserved error numbers. The first one is error number -745, which has a generalized and fixed error message. The second one is error number -746, which allows you to supply the message text, up to a maximum of 71 characters.

Applying a Fixed Error Message

You can apply error number -745 to any trigger failure that is not an SQL error. The following fixed message is for this error:

-745 Trigger execution has failed.

For more information on using the TRACE statement to diagnose logic errors in SPL routine, see Chapter 14, “Creating and Using SPL Routines.”
Applying a Fixed Error Message

You can apply this message with the RAISE EXCEPTION statement in SPL. The following example generates error -745 if `new_qty` is greater than `old_qty` multiplied by 1.50:

```sql
CREATE PROCEDURE upd_items_p2()
DEFINE GLOBAL old_qty INT DEFAULT 0;
DEFINE new_qty INT;
LET new_qty = (SELECT SUM(quantity) FROM items);
IF new_qty > old_qty * 1.50 THEN
  RAISE EXCEPTION -745;
END IF
END PROCEDURE
```

If you are using DB-Access, the text of the message for error -745 displays on the bottom of the screen, as Figure 15-4 shows:

If you trigger the erring routine through an SQL statement in your SQL API, the database server sets the SQL error status variable to -745 and returns it to your program. To display the text of the message, follow the routine that your Informix application development tool provides for retrieving the text of an SQL error message.
Generating a Variable Error Message

Error number -746 allows you to provide the text of the error message. Like the preceding example, the following one also generates an error if `new_qty` is greater than `old_qty` multiplied by 1.50. However, in this case the error number is -746, and the message text `Too many items for Mfr.` is supplied as the third argument in the RAISE EXCEPTION statement. For more information on the syntax and use of this statement, see the RAISE EXCEPTION statement in Chapter 14, "Creating and Using SPL Routines."

```sql
CREATE PROCEDURE upd_items_p2()
    DEFINE GLOBAL old_qty INT DEFAULT 0;
    DEFINE new_qty INT;
    LET new_qty = (SELECT SUM(quantity) FROM items);
    IF new_qty > old_qty * 1.50 THEN
        RAISE EXCEPTION -746, 0, 'Too many items for Mfr.';
    END IF
END PROCEDURE;
```

If you use DB-Access to submit the triggering statement, and if `new_qty` is greater than `old_qty`, you will get the result that Figure 15-5 shows.

```
INSERT INTO items VALUES( 2, 1001, 2,'HRO', 1, 126.00);
```

![Figure 15-5 Error Number -746 with User-Specified Message Text](image)
If you invoke the trigger through an SQL statement in an SQL API, the database server sets `sqlcode` to -746 and returns the message text in the `sqlerrm` field of the SQL Communications Area (SQLCA). See the manual for your SQL API for in-depth information about using the SQLCA.

### Summary

To introduce triggers, this chapter covers the following topics:

- The purpose of each component of the `CREATE TRIGGER` statement
- How to create BEFORE and AFTER triggered actions and how to use them to determine the impact of the triggering statement
- How to create a FOR EACH ROW triggered action and how to use the `REFERENCING` clause to refer to the values of columns both before and after the action of the triggering statement
- The advantages of using SPL routines as triggered actions
- How to trace triggered actions if they are behaving unexpectedly
- How to generate two types of error messages within a triggered action