Using Triggers to Maintain Database Integrity

By Jeff Ash

Most applications that manipulate and query data use some type of validation to assure data integrity. Currently, whether using WRITE RECORD operations or SQL INSERT and UPDATE statements in an interpreter session, all BBj® programs perform this validation in the application before writing the values to disk. However, what happens when an external ODBC or JDBC application such as a Web application performs write operations to that same data? The validation does not occur unless the ODBC or JDBC application duplicates that same validation routine. With BBj 6.0, developers can move the validation code to the table or file level, so that it occurs regardless of the method used to provide access to the table (e.g. WRITE RECORD, REMOVE, READ RECORD, or using SQL).

What is a Trigger?

A trigger consists of a block of BBj code that executes when a particular type of operation occurs on a particular data table. In other database management systems, a trigger belongs to a particular table in the database defined in the data dictionary. Since many BBj applications still use direct table or file access calls such as WRITE RECORD and READ RECORD that do not require a Data Dictionary definition, triggers belong to individual physical files. This means that a WRITE RECORD operation on a file and an SQL UPDATE or INSERT statement on a table defined in the Data Dictionary both cause write triggers to fire.

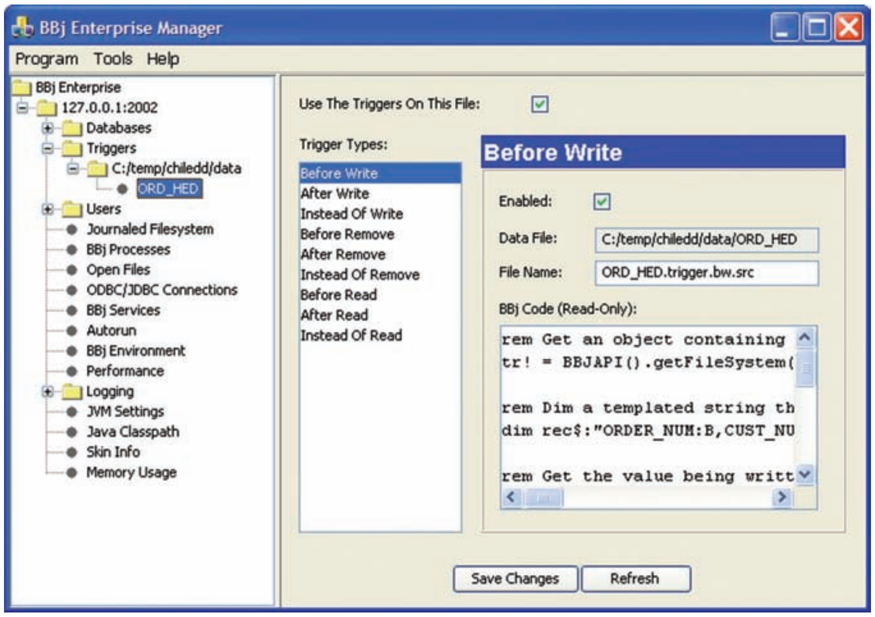
Real-World Example

A real-world trigger example should help illustrate trigger concepts and spark some ideas for other uses in your own applications. This example uses the Chile Company database included in the BBj installation. Suppose the Chile Company application creates a new order and then writes the order header information to the **`ord_hed`** file. Before adding the order, we want to check the validity of the customer number. If the customer number is not valid, the write operation should fail. This check should take place regardless of whether the operation originates from the BBj application or through a Web application using ODBC or JDBC.

The following steps walk through the process of adding a trigger to the **`ord_hed`** data file as shown in **Figure 1**:

1. Log in to the BBj Enterprise Manager.
2. Right click on the **Triggers** folder and select "Mount Directory of Files."
3. Navigate to the **`[bbj_install_directory]/demos/chiledd/data`** directory and click the [Select] button.
4. With the **Triggers** folder expanded, right click on the newly mounted directory name and select "Add Trigger."
5. Locate the **`ORD_HED`** file and select it from the file chooser.
6. With the newly mounted folder expanded, select the **`ORD_HED`** node in the Enterprise Manager tree.
7. Ensure the "Use The Triggers On This File" checkbox is checked.
8. Select "Before Write" in the Trigger Types list.
9. Ensure the "Enabled" checkbox is checked.
10. Click the [Save Changes] button.
11. Use the BASIS IDE or another text editor to save the code sample in **Figure 2** (available for download) listed below in a file named **`ORD_HED.trigger.bw.src`** and located in the same directory as the **`ORD_HED`** data file:

continued...



Use The Triggers On This File: ✓
Enabled: ✓
Data File: C:/temp/chiledd/data/ORD_HED
File Name: ORD_HED.trigger.bw.src

```
rem Get an object containing
tr! = BBJAPI().getFileSystem(

rem Dim a templated string th
dim rec$:"ORDER_NUM:B,CUST_NU

rem Get the value being writt
```

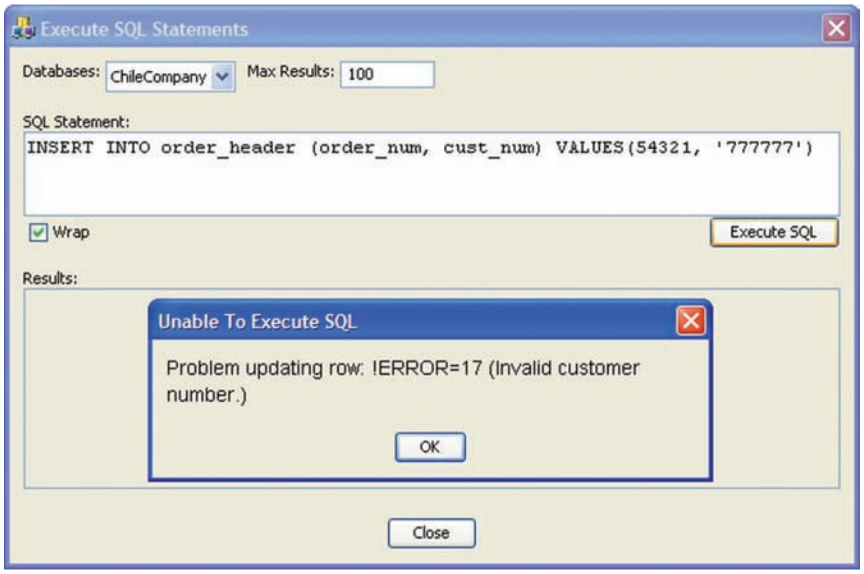
Figure 1. Trigger dialog box

Jeff Ash
Software Engineer

```
rem Get an object containing the trigger information.
tr! = BBJAPI().getFileSystem().getTriggerData()

rem Dim a templated string that matches the layout for ORD_HED
r$ = "ORDER_NUM:B,CUST_NUM:C(6),ORDER_DATE:N(7),SHIP_DATE:N(7),SHIP_ZONE:C(2),"
r$ = r$ + "SHIP_METHOD:C(5),COMMENT:C(30),SALESPERSON:C(3),MDSE_TOTAL:N(12),"
r$ = r$ + "TAX_TOTAL:N(12),FRGHT_TOTAL:N(12)"
dim rec$:r$

rem Get the value being written to disk
rec$ = tr!.getWriteBuffer()

rem See if the CUST_NUM is valid by ensuring it's in the customer data file
custNum$ = rec.cust_num$
chan = sqlunt
sqlopen(chan)"ChileCompany"
sqlprep(chan)"select count(cust_num) REC_COUNT from customer where cust_num = ?"
sqlexec(chan)custNum$
dim result$:sqltmpl(chan)
result$ = sqlfetch(chan)
sqlclose(chan)

rem If the CUST_NUM doesn't exist, throw an error
if result.rec_count = 0 then throw "Invalid customer number.",17
```

Figure 2. Code sample **`ORD_HED.trigger.bw.src`**

To test the new trigger, run the following UPDATE statement from a BBj program using the SQL verbs or from an ODBC or JDBC program such as Microsoft Query (**Figure 3**). Notice how the trigger causes the update to fail with our custom error message when using an invalid customer number.



```
INSERT INTO order_header (order_num, cust_num) VALUES(54321, '777777')
```

Problem updating row: !ERROR=17 (Invalid customer number.)

Figure 3. Custom error from trigger

Other Uses for Triggers

There are a number of other uses for triggers. Consider read and write triggers to encrypt/decrypt file data automatically. Another use might be a write trigger that logs information to a table that tracks who performed write operations on a file or a trigger that inserts data into a table in another database to keep the two databases in sync.

Summary

Most applications perform data validation inside the application itself. However, in BBj 6.0, developers can move data validation to a trigger attached to the data file. Using triggers makes this validation code available to everyone who accesses the data file, regardless of whether it is from a BBj application or a third party ODBC or JDBC application. When the code is at the file level, it provides a mechanism for much more consistent and reliable validation of data and therefore provides a solid mechanism for maintaining data integrity in your BASIS DBMS.

Download the sample program referenced in this article from
www.basis.com/advantage/mag_v10n1/triggers.zip