DBMaker

Trigger User's Guide

P-E9999-Trigger User's Guide

Version: 01.00

Document No: 43/DBM43-T04212006-01-TRIG

Author: Production Team
Syscom Computer Engineering CO.

Print Date: April 21, 2006

凌羣電腦
THE SYSCOM GROUP

Table of Content

1. Introduction

Welcome to the DBMaster Trigger user's Guide. This guide discusses the details about the Trigger and guides the user how to use it. Including how to create Trigger, modify Trigger and drop Trigger and so on. It provides integrated instances to help user understand.

1.1. Preface of DBMaster Trigger

Triggers are a very useful and powerful feature of the DBMaster database server. Triggers automatically execute predefined commands in response to specific events, regardless of which user or application program generated them. Triggers allow a database to be customized in ways that may not be possible with standard SQL commands. The database can consistently control complex or unconventional database operations without requiring any action on the part of users or application programs. Use triggers to:

- Implement business rules
- Create an audit trail for database activities
- Derive additional values from existing data
- Replicate data across multiple tables
- Perform security authorization procedures
- Control data integrity
- Define unconventional integrity constraints

Exercise restraint when using triggers to avoid forming complex interdependencies within the database that may be difficult to follow and change. Using triggers can also implement the desired functionality that cannot be implemented by using standard SQL commands or integrity constraints.

1.2. Additional Resources

DBMaker provides a complete set of DBMS manuals in addition to this one. For more detailed information on a particular subject, consult one of the books listed below:

- For an introduction to DBMaker's capabilities and functions, referring to the *DBMaker Tutorial*.
- For more information on designing, administering, and maintaining a DBMaker database, referring to the *Database Administrator's Guide*.
- For more information on database server management, referring to the *JServer Manager User's Guide*.

- For more information on configuring DBMaker, referring to the *JConfiguration Tool Reference*.
- For more information on the native ODBC API, referring to the *ODBC Programmer's Guide*.
- For more information on the dmSQL interface tool, referring to the *dmSQL User's Guide*.
- For more information on the SQL language used in dmSQL, referring to the *SQL Command and Function Reference*.
- For more information on the ESQL/C programming, referring to the *ESQL/C User's Guide*.
- For more information on error and warning messages, referring to the *Error and Message Reference*.
- For more information on the DBMaker COBOL Interface, referring to the *DCI User's Guide*.
- For more information on the Stored Procedure, referring to the *Stored Procedure User's Guide.*
- For more information on the Lock, referring to the *Lock User's Guide.*
- For more information on the DBMaker bundle-version, referring to the *DBMaker bundle Instruction*.
- For more details on the performance of the database, referring to the *Performance Tuning Guide.*

1.3. Document Conventions

This book uses a standard set of typographical conventions for clarity and ease of use. The NOTE, Example, and CommandLine conventions also have a second setting used with indentation.

CONVENTION	DESCRIPTION
Italics	Italics indicate the massages returned from the system. It always means error or warning. It is also used to indicate a short procedure in the normal paragraph text or the specification name of the DBMaster.
Boldface	Boldface indicates filenames, database names, table names, column names, user names, and other database schema objects. It is also used to emphasize menu commands in procedural steps.
KEYWORDS	All keywords used by the SQL language appear in uppercase when used in normal paragraph text.
NOTE	Contains important information or it reminds you to pay attention.

CONVENTION	DESCRIPTION
➲ Example	Examples are given to clarify descriptions, and commonly include text, as it will appear on the screen. Other forms of this convention include Prototype and Syntax.

Table 1-1 Document Conventions

2. Overview

This chapter tells the basic and necessary knowledge before user creating a trigger. It contains trigger components and the trigger operation.

2.1. Trigger components

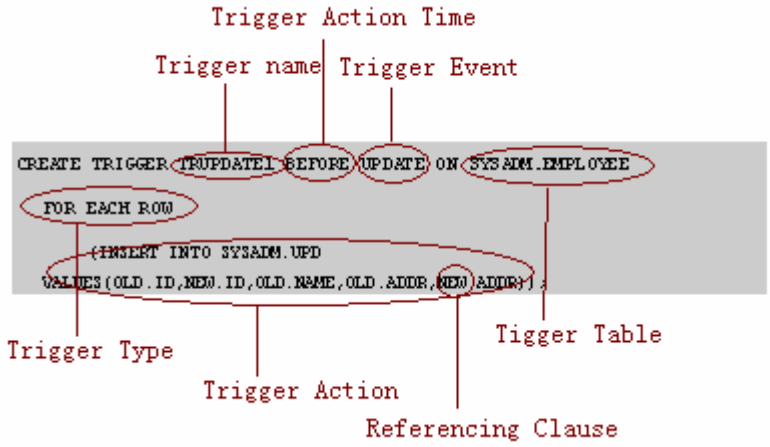
DBMaster stores trigger definitions in the system catalog.

Every DBMaster trigger has six main components:

- Trigger Name—a name that uniquely identifies the trigger
- Trigger Action Time—the time relative to when a trigger will be fired
- Trigger Event—a specific situation that occurs in the database in response to some user action, such as inserting data into a table
- Trigger Table—the name of the table that the trigger executes on
- Trigger Action—an SQL statement or stored procedure that is executed when the trigger event occurs
- Trigger Type—the type of trigger

Each of these components must be present in all triggers. In addition, there is an optional component, the REFERENCING clause.

The picture bellow can help user understand these components.



Picture 2-1 Trigger components

2.1.1. TRIGGER NAME

The trigger name uniquely identifies a trigger. Trigger names have a maximum length of 32 characters and may contain letters, numbers, the underscore character, and the symbols # and $. The first character cannot contain a number, and the name cannot contain spaces.

➲Example:

The followings are valid Trigger names:

```
trigger1; table1-t1; example#; example$;
```

The followings are invalid Trigger names:

```
1trigger; table1 t1; examplesfortrigger-mytriggername1;
```

2.1.2. TRIGGER ACTION TIME

The trigger action time specifies whether it should fire before or after the SQL statement that activates it. The trigger action time is specified by the BEFORE and AFTER time keywords. The BEFORE keyword instructs the trigger to fire before the SQL statement. The AFTER keyword instructs the trigger to fire after the SQL statement. Only one trigger time can be specified for each trigger.

2.1.3. TRIGGER EVENT

The trigger event is the database operation that causes a trigger to operate, or fire. The trigger event may be an INSERT, UPDATE, or DELETE statement that operates on the trigger table. There can be only one trigger event for each trigger statement. However, multiple trigger events can be used to activate multiple triggers.

2.1.4. TRIGGER TABLE

The trigger event operates on the associated trigger table. The trigger table must be a base table; it can not be a temporary table, view, or synonym. A trigger may only have one trigger table.

2.1.5. TRIGGER ACTION

A trigger action is the command that a trigger executes when it fires. The trigger action may be an INSERT, UPDATE, DELETE, or EXECUTE PROCEDURE statement. A trigger can only have a single trigger action.

2.1.6. TRIGGER TYPE

The trigger type specifies how many times the trigger will fire for each trigger event. There are two kinds of trigger type: row triggers and statement triggers. The FOR EACH ROW option specifies a row trigger, which fires a trigger action once for each row modified by the trigger event. The FOR EACH STATEMENT option specifies a statement trigger, which fires a trigger action once for each trigger event.

2.1.7. REFERENCING CLAUSE

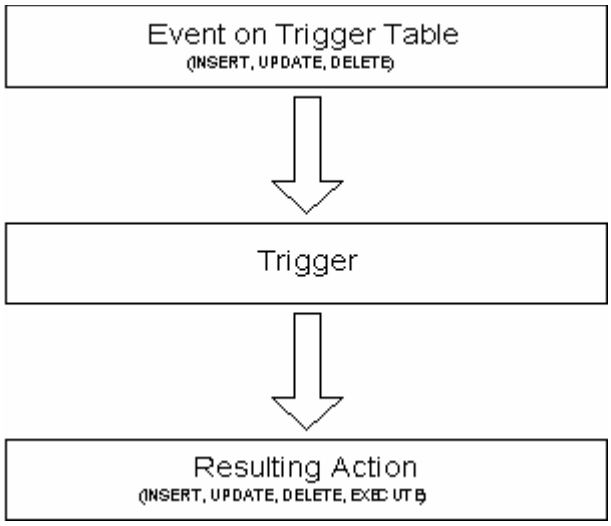
The REFERENCING clause defines correlating names for the old and new values in a column. This is primarily used when the default OLD and NEW names cannot be used because of a name conflict with a table.

2.2. Trigger Operation

DBMaster checks to see if a trigger should be fired and will execute the defined triggers whenever a user or an application program causes a trigger event. Firing triggers within a database ensures that DBMaster handles data consistently across all applications. So this guarantees that when a specific event occurs, a related action is also performed.

Users can create triggers to implement domain, column, referential, and unconventional integrity constraints. However, these can also be done by declarative integrity control.

Triggers do not have an owner, but are associated with a table.



Picture 2-2 Trigger event and action

3. Creating Triggers

The CREATE TRIGGER command creates a new trigger associated with a specific table. Only a user with privilege on the trigger table can execute the command. The user must also have the necessary object privileges for all objects referenced in the trigger definition in order to successfully create a trigger.

DBMaster also provides the JDBATOOL for user to create Triggers.

3.1. Creating Trigger syntax

Before starting to create a Trigger, the user should know some basic requirements and have the privilege.

3.1.1. BASIC REQUIREMENTS

All of the CREATE TRIGGER statements must contain at least the following:

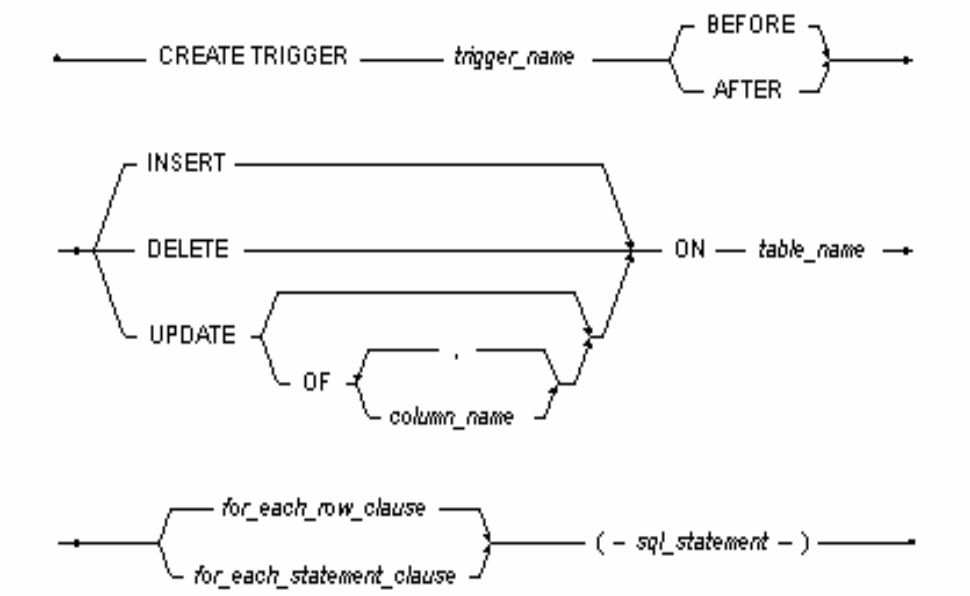
- A trigger name
- The trigger action time (before or after)
- The trigger event (insert or delete or update)
- The trigger table
- The trigger type (row or statement)
- The trigger action

3.1.2. SECURITY PRIVILEGES

All SQL statements in the trigger action operate with the same privileges as the owner of the trigger table, not with the privileges of the user executing the trigger event. If the trigger exists, any user executing the trigger event will result in the trigger firing.

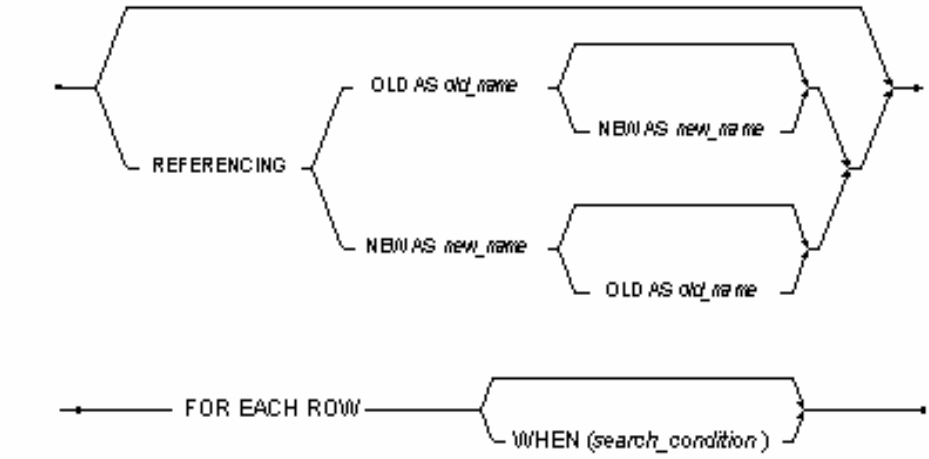
3.1.3. CREATE TRIGGER SYNTAX

The CREATING TRIGGER SYNTAX is as follows:



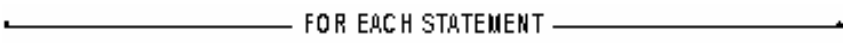
Picture 3-1 Create Trigger Syntax

FOR EACH ROW clause syntax:



Picture 3-2 for each row clause

FOR EACH STATEMENT clause syntax:



Picture 3-3 for each statement clause

➲ Example:

To create a trigger named **TRINSERT2**, this trigger fired when insert on the table **EMPLOYEE**, and when it fires, the new data will inserted into the table **NEWEMPLOYEE**. (Information about the tables **EMPLOYEE** and **NEWEMPLOYEE**, please refer the **Appendix**)

```
CREATE TRIGGER TRINSERT2 AFTER INSERT ON SYSADM.EMPLOYEE

        FOR EACH ROW
```

```
                (INSERT INTO SYSADM.NEWEMPLOYEE
                        VALUES (NEW.ID,NEW.NAME,NEW.ADDR));
```

3.2. Specifying the Trigger Action Time

As it mentioned previously, the trigger action time is specified by the BEFORE and AFTER time keywords. You can use the trigger time and trigger type in combination to create four triggers for each table for the same event (INSERT, DELETE, or UPDATE). For each event the BEFORE/FOR EACH ROW, AFTER/FOR EACH ROW, BEFORE/FOR EACH STATEMENT, and AFTER/FOR EACH STATEMENT combinations are possible.

A BEFORE/FOR EACH STATEMENT trigger executes once and only once before the triggering statement is performed. That is before the occurrence of the trigger event.

An AFTER/FOR EACH STATEMENT trigger executes once and only once after the triggering statement is complete.

NOTE: the BEFORE and AFTER FOR EACH STATEMENT triggers are executed even if the triggering statement does not process any rows. More details about FOR EACH STATEMENT type please refer the chapter 3.3 *Specifying the Trigger Type.*

3.2.1. ABOUT THE INSERT TRIGGER EVENT

The following examples show how to create triggers that fire before or after INSERT trigger events. We give one example for each trigger action time. The details of the reference tables please refer the **Appendix**.

➲ Example:

1. To create trigger for BEFORE INSERT Trigger event on table **EMPLOYEE**:

```
CREATE TRIGGER TRINSERT1 BEFORE INSERT ON SYSADM.EMPLOYEE
         FOR EACH ROW
                (INSERT INTO SYSADM.SALARY
                        VALUES (NEW.ID,NEW.NAME));
```

   Once the **TRINSERT1** was created successfully, if there were a new data was inserted into the table **EMPLOYEE**, the table **SALARY** will be added the new employee's id and name.

2. To create trigger for AFTER INSERT Trigger event on table EMPLOYEE:

```
CREATE TRIGGER TRINSERT2 AFTER INSERT ON SYSADM.EMPLOYEE
        FOR EACH ROW
                (INSERT INTO SYSADM.NEWEMPLOYEE
                        VALUES (NEW.ID,NEW.NAME,NEW.ADDR));
```

   Also, once the **TRINSERT2** was created successfully, if there were a new data was inserted into the table **EMPLOYEE**, the table **NEWEMPLOYEE** will be added the same new data too.

When create trigger on INSERT event, user must pay attention to the usage of the correlation name, it is not allowed to use the OLD correlation name in trigger of INSERT event. Because the old tuples don't exist when insert. The following example will cause error: *ERROR (6194): [DBMaker] cannot allow old correlation name in statement trigger or row trigger of insert trigger event.*

```
CREATE TRIGGER TRINSERT1 BEFORE INSERT ON SYSADM.EMPLOYEE
        FOR EACH ROW
              (INSERT INTO SYSADM.SALARY
                     VALUES (OLD.ID,OLD.NAME,NEW.ADDR))
```

3.2.2. ABOUT THE DELETE TRIGGER EVENT

The following examples show how to create triggers that fire before or after DELETE trigger events. We give one example for each trigger action time respectively. The details of the reference tables please refer the **Appendix**.

➲ Example:

3. To create trigger for BEFORE DELETE trigger event on table EMPLOYEE:

```
CREATE TRIGGER TRDELETE1 BEFORE DELETE ON SYSADM.EMPLOYEE
        FOR EACH ROW
              (DELETE FROM SYSADM.SALARY
                                  WHERE ID = OLD.ID);
```

Once the **TRDELETE1** was created successfully, if there were data was deleted from the table **EMPLOYEE**, the data will be deleted from table **SALARY** either.

4. To create trigger for AFTER DELETE trigger event on table EMPLOYEE:

```
CREATE TRIGGER TRDELETE2 AFTER DELETE ON SYSADM.EMPLOYEE
        FOR EACH ROW
              (INSERT INTO SYSADM.DIMISSORY
                     VALUES(OLD.ID,OLD.NAME,OLD.ADDR))
```

Once the **TRDELETE2** was created successfully, if there were data was deleted from the table **EMPLOYEE**, then the data will inserted into the table **DIMISSORY**.

When create trigger on DELETE event, user must pay attention to the usage of the correlation name, it is not allowed to use the NEW correlation name in trigger of DELETE event. Because the new tuples don't exist when delete. The following example will cause error: *ERROR (6195): [DBMaker] cannot allow a new correlation name in statement trigger or row trigger of delete trigger event.*

```
CREATE TRIGGER TRDELETE1 BEFORE DELETE ON SYSADM.EMPLOYEE
        FOR EACH ROW
              (INSERT INTO SYSADM.DEL
```

```
                    VALUES (NEW.ID,NEW.NAME,NEW.ADDR)
```

3.2.3. ABOUT THE UPDATE TRIGGER EVENT

The situation is different for UPDATE events. Two types of UPDATE triggers can be created: UPDATE <table> triggers, or UPDATE OF <column> triggers. An UPDATE <table> trigger fires whenever the table is updated. An UPDATE OF <column> trigger fires when specific columns are updated. Either one UPDATE <table> trigger or multiple UPDATE OF <column> triggers can be created on a single table. I.e. it is not allowed to create UPDATE <table> trigger and UPDATE OF <column> triggers simultaneously on a single table with the same action time. But if you want to, please use different action time. UPDATE OF <column> triggers may contain multiple columns, but columns between all UPDATE OF <column> triggers in a table must be mutually exclusive.

The details of the reference tables please refer the **Appendix**.

➲ Example: BEFORE UPDATE

5. To create trigger for BEFORE UPDATE trigger event on table **EMPLOYEE**:

```
CREATE TRIGGER TRUPDATE1 BEFORE UPDATE ON SYSADM.EMPLOYEE
        FOR EACH ROW
                (UPDATE SYSADM.SALARY
                        SET ID=NEW.ID,NAME=NEW.NAME WHERE ID=OLD.ID)
```

   Once **TRUPDATE1** was created successfully, if a row was updated in the **EMPLOYEE**, the row that has the same id in the **SALARY** will be updated too.

6. To create a column trigger event on the **addr** column of table **EMPLOYEE** BEFORE UPDATE the **addr**:

```
CREATE TRIGGER TRUPDATE2 BEFORE UPDATE OF addr ON SYSADM.EMPLOYEE
        FOR EACH ROW
                (INSERT INTO SYSADM.UPD
                        VALUES(OLD.ID,NEW.ID,OLD.NAME,OLD.ADDR,NEW.ADDR));
```

   Once **TRUPDATE2** was created successfully, when and only when update the **addr** column in the **EMPLOYEE**, the values will be inserted into the table **UPD**.

NOTE: it is not allowed to create the UPDATE <table> trigger and UPDATE OF <column> triggers simultaneously within a table with the same action time. So user shouldn't create the above two examples on one table simultaneously, if did, it will return an error message: *ERROR (6199): [DBMaker] cannot create update-of-column trigger if update trigger exists or vice versa.*

Also, after create the **example6**; it will be fail if user attempts to create another column trigger whose column contains the **addr**, for example:

```
CREATE TRIGGER TRUPDATE3 BEFORE UPDATE OF addr, id ON SYSADM.EMPLOYEE
         FOR EACH ROW
                (INSERT INTO SYSADM.UPD
                        VALUES (OLD.ID,NEW.ID,OLD.NAME,OLD,ADDR,NEW.ADDR));
```

The error massage retuned: *ERROR (6198): [DBMaker] illegal column found in update of column list already existed in other trigger.*

Of course, user can create other column triggers on the different columns, for example on **id**:

7. To create a column trigger event on the **id** of table EMPLOYEE BEFORE UPDATE the **id** :

```
CREATE TRIGGER TRUPDATE4 BEFORE UPDATE OF id ON SYSADM.EMPLOYEE
        FOR EACH ROW
                (INSERT INTO SYSADM.UPD
                        VALUES (OLD.ID,NEW.ID,OLD.NAME,OLD.ADDR,NEW.ADDR));
```

Once **TRUPDATE4** was created successfully, when and only when update the **id** column in the **EMPLOYEE**, the values will be inserted into the table **UPD**.

➲ Example: ATFER UPDATE

You can modify the action time to AFTER in **example6** or **example7** so that it can be create with the **example5** on the same table. As followings:

8. Modifying **example6**: To create a column trigger event on the **addr** of table **EMPLOYEE** AFTER UPDATE the **addr**:

```
CREATE TRIGGER TRUPDATE2 AFTER UPDATE OF addr ON SYSADM.EMPLOYEE
        FOR EACH ROW
                (INSERT INTO SYSADM.UPD
                        VALUES(OLD.ID,NEW.ID,OLD.NAME,OLD.ADDR,NEW.ADDR));
```

Once **TRUPDATE2** was created successfully, when and only when after updating the **addr** column in the **EMPLOYEE**, the values will be inserted into the table **UPD**.

9. Modifying **example7**: To create a column trigger event on the **addr**, **id** of table **EMPLOYEE** AFTER UPDATE the **addr** and **id.**

```
CREATE TRIGGER TRUPDATE3 AFTER UPDATE OF addr,id ON SYSADM.EMPLOYEE
         FOR EACH ROW
                (INSERT INTO SYSADM.UPD
                        VALUES (OLD.ID,NEW.ID,OLD.NAME,OLD.ADDR,NEW.ADDR));
```

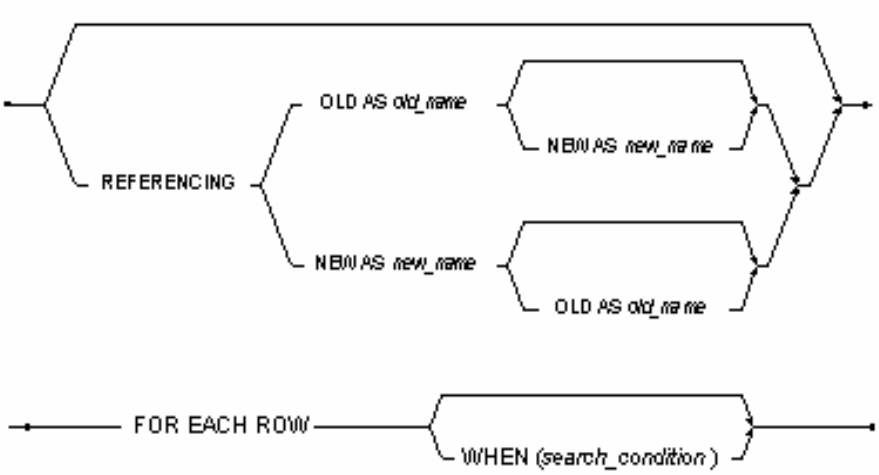
Once **TRUPDATE3** was created successfully, when and only when both the **addr** and **id** were updated, the values will be inserted into **UPD**.

3.3. Specifying the Trigger Type

The user can choose the Trigger type: FOR EACH ROW or FOR EACH STATEMENT. They will be described in detail in the following parts.

3.3.1. ABOUT THE FOR EACH ROW

The FOR EACH ROW clause syntax is as bellow:



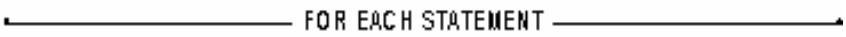
Picture 3-4 for each row clause

The FOR EACH ROW clause specifies that a trigger will fire once for each row that the trigger event modifies. If the trigger event does not modify any rows, the trigger will not fire. The OLD and NEW keywords are used to identify which values from the trigger table are to be used in the trigger action. The OLD keyword indicates that trigger table values from before the trigger event are used in the trigger action. The NEW keyword indicates that trigger table values from after the trigger event are used in the trigger action.

All the examples we give above are use the FOR EACH ROW trigger type. This type is flexible because it will fire for each row. If there are several rows were triggered it would be fire several times.

3.3.2. ABOUT THE FOR EACH STATEMENT

FOR EACH STATEMENT clause syntax:



Picture 3-5 for each statement clause

The FOR EACH STATEMENT clause specifies that a trigger will fire once and only once for each trigger event. Even if the trigger event statement does not process any rows, the trigger will fire. And it will be cause error if use the OLD or NEW keyword in the trigger action clause.

➲ Example:

10. To create a trigger in FOR EACH STATEMENT type:

```
CREATE TRIGGER TRINSERT3 BEFORE INSERT ON SYSADM.EMPLOYEE
       FOR EACH STATEMENT
              (INSERT INTO SYASAM.SALARY
                           VALUES (100,'temporary'.10))
```

Once **TRINSERT3** was created successfully, if there were data was inserted into the **EMPLOYEE**, the tuple (100,'temporary',10) will be insert into table **SALARY** then.

3.4. Using the Referencing Clause

In row triggers, the <sql_statement> (or action body) should indicate whether the column values used are from before or after the trigger event. For example, to log the old ID and new ID when updating the ID of EMPLOYEE, use the keywords OLD and NEW as shown in **example5**.

However, in some rare cases the table's name may be the NEW or OLD, or the table may contain columns with the names NEW or OLD. If this is the case, use the referencing clause to define correlation names. The reference clause allows for the creation of two prefixes that can be used with a column name: one to reference the old value of the column, and one to reference the new value. These prefixes are called correlation names. Use the keywords OLD and NEW to indicate the correlation names.

➲ Example:

11. To create a trigger for update event on table **NEW**:

```
CREATE TRIGGER TRUPDATE5 AFTER UPDATE OF ADDR ON SYSADM.NEW
    REFERENCING OLD as pre NEW as post
        FOR EACH ROW
             (INSERT INTO SYSADM.UPD
                    VALUES(pre.ID,post.ID,pre.NAME,pre.ADDR,post.ADDR);
```

Once **TRUPDATE5** was created successfully, if the **addr** column of the table **NEW** was updated, the values will be inserted into table **UPD**.

The details of the reference tables please refer the **Appendix**.

In this example, the triggering table name is **NEW**, so the correlation names pre and post are used in the action body. Referencing clauses are only valid for row triggers, and are not allowed in statement triggers. If a trigger event is INSERT, there is no old value for the newly inserted record, so the old value is not available. Similarly, if the trigger event is DELETE, there is no new value for the deleted record, so the new value is not available. For an UPDATE event trigger, both old and new values are available.

3.5. Using the WHEN Condition

A WHEN condition clause may precede a FOR EACH ROW triggered action to make the action execution dependent on the result of a Boolean expression. The when clause consists of a keyword WHEN followed by the conditional statement in parentheses follows the action time and precedes the triggered action body, and the WHEN clause is not allowed in the definition of a statement trigger, it is only allowed in row trigger.

➲ Example:

12. Modifying the **example3** to create a trigger using the WHEN condition:

```
CREATE TRIGGER TRWHEN BEFORE DELETE ON SYSADM.EMPLOYEE
        FOR EACH ROW
                WHEN (OLD.ID>0)
```

```
                    (DELETE FROM SYSADM.SALARY WHERE ID= OLD.ID);
```

The WHEN condition WHEN (OLD.ID>0) constrains that only the data whose **id** value is bigger than 0 could fire the trigger.

3.6. Specifying the Trigger Action

The trigger action is the SQL statement that is performed when the trigger event occurs. The trigger action can be an INSERT, DELETE, UPDATE, or EXECUTE PROCEDURE statement. No other statements are allowed. Stored procedures cannot contain COMMIT, ROLLBACK or SAVEPOINT transaction control statements. Triggers can specify only a single trigger action, which must be enclosed in parentheses.

➲ Example:

13. To create trigger specifying the Trigger Action is INSERT, take **example1** for example:

```
CREATE TRIGGER TRINSERT1 BEFORE INSERT ON SYSADM.EMPLOYEE
        FOR EACH ROW
              (INSERT INTO SYSADM.SALARY
                    VALUES (NEW.ID,NEW.NAME));
```

14. To create trigger specifying the Trigger Action is UPDATE, take **example5** for example:

```
CREATE TRIGGER TRUPDATE1 BEFORE UPDATE ON SYSADM.EMPLOYEE
       FOR EACH ROW
              (UPDATE SYSADM.SALARY
                    SET ID=NEW.ID,NAME=NEW.NAME WHERE ID=OLD.ID)
```

15. To create trigger specifying the Trigger Action is DELETE, take **example3** for example:

```
CREATE TRIGGER TRDELETE1 BEFORE DELETE ON SYSADM.EMPLOYEE
        FOR EACH ROW
              (DELETE FROM SYSADM.SALARY
                                  WHERE ID = OLD.ID);
```

16. About specifying the Trigger Action is Stored Procedures, we will give example in the chapter 6.1 *Stored Procedures in Action Body.*

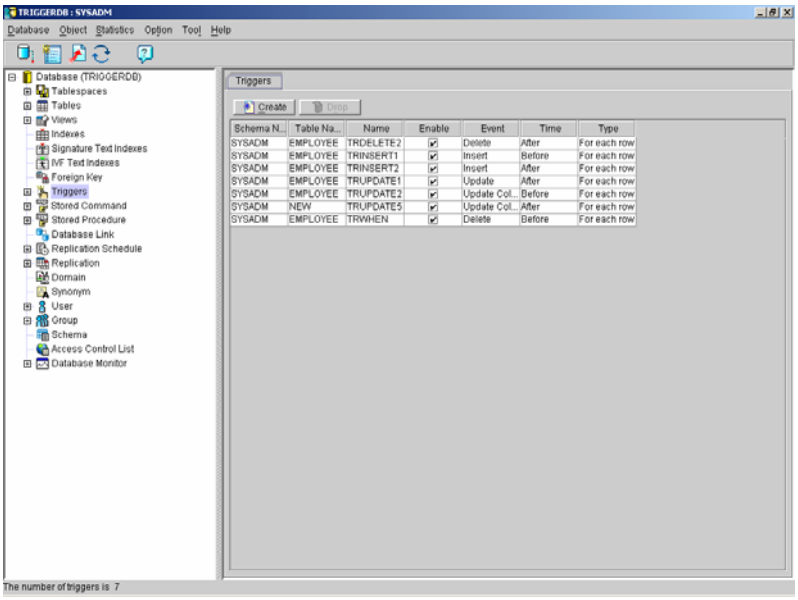
3.7. Using JDBATOOL to Create Trigger

Beside the above method to create trigger, DBMaster also provides the GUI—JDBATOOL to guide the user to create a trigger.

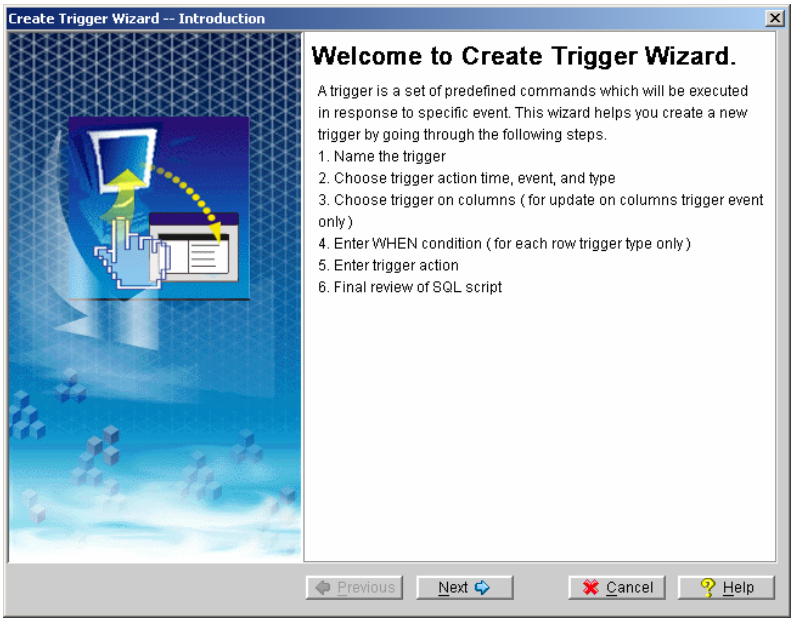
3.7.1. ASSIGNING A TRIGGER NAME AND TABLE

The following figures show how to assign a trigger name and indicate the table in which the trigger is created.

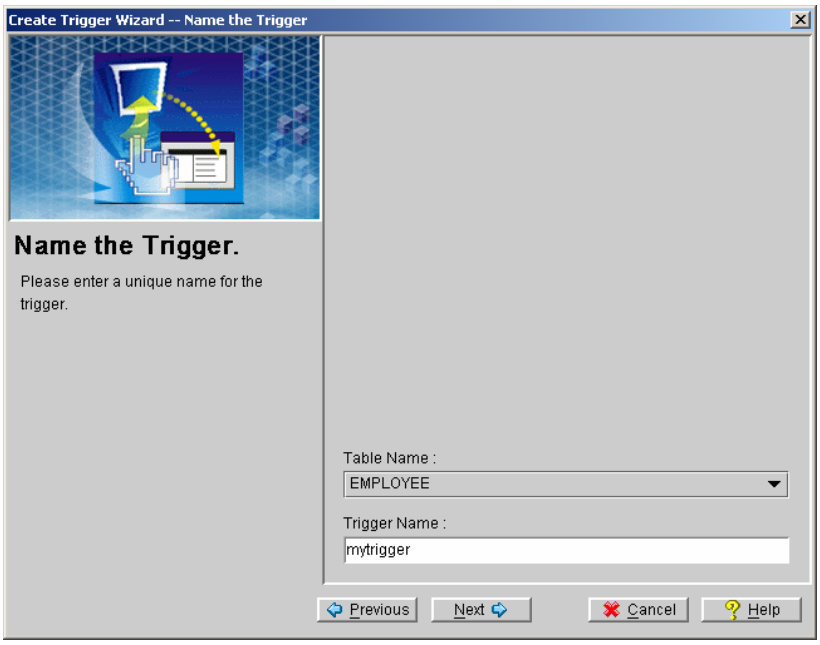
1. Click the **Triggers** object in the tree. The **Triggers** page is displayed.



2. Click **Create**. The Introduction window of the **Create Trigger Wizard** is displayed.

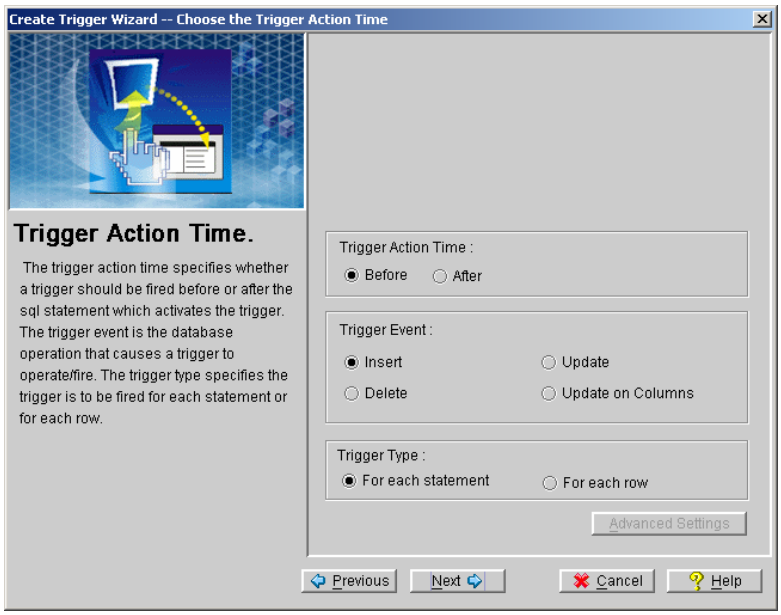


3. Click **Next**. The **Name the Trigger** window is displayed.
4. Select the table name in which the Trigger will be based on from the **Table Name** menu.
5. Enter the name of the trigger in the **Trigger Name** field.
6. Click **Next**. The **Choose the Trigger Action Time** window will open.



3.7.2. SPECIFYING TRIGGER ACTION SETTINGS

1. From the **Name the Trigger** window of the **Create Trigger Wizard**, click **Next**. The **Choose the Trigger Action Time** window will open.



2. Select one of the following trigger action time options from the **Trigger Action Time**:
   - To set the trigger to fire before the SQL statement, click the **Before** option button.
   - To set the trigger to fire after the SQL statement, click the **After** option button.
3. Select one of the following trigger events from the **Trigger Event** field:
   - To choose the INSERT command as the trigger event. Choose the **Insert** option.
   - To choose the UPDATE command as the trigger event .Choose the **Update** option.

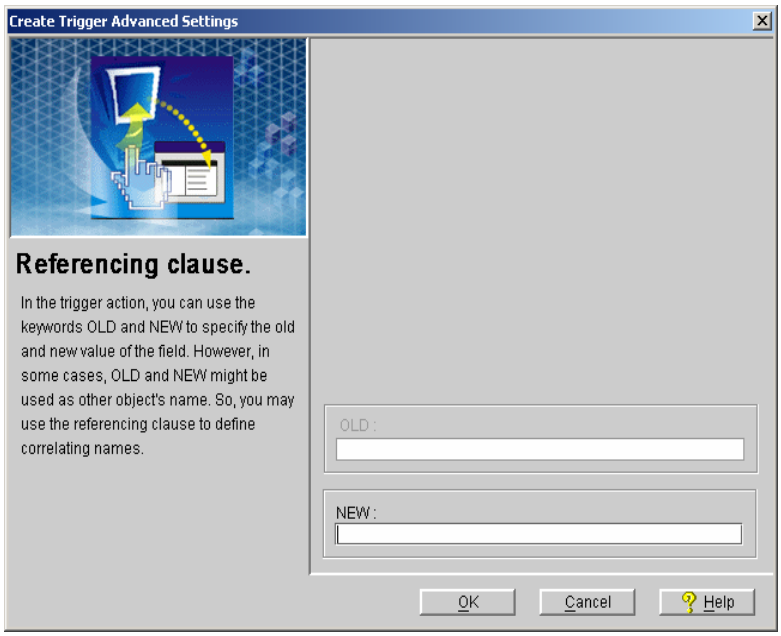
- To choose the DELETE command as the trigger event .Choose the **Delete** option.
- To choose the UPDATE COLUMN command as the trigger event. Choose the **Update on Columns** option.

4. Select one of the following trigger type options from the **Trigger Type** field.
   - Selecting the **For each row** option button sets the trigger statement to execute on each row modified by the trigger event.
   - Selecting the **For each statement** option button sets the trigger statement to execute upon each instance of the trigger event.
5. If the **For each statement** option button is selected, clicking **Next** will open the **Trigger Action** window. If the **For each row** option button is selected. Clicking **Advanced Settings** button will open the **Referencing clause** window.

3.7.3. INDICATING THE REFERENCING CLAUSE

Once you have indicated trigger options and if you selected the FOR EACH ROW as the trigger type, you can then specify the referencing clause of the trigger. The REFERENCING clause defines correlation names for the old and new values of a column. This is primarily used when you cannot use the default OLD and NEW names because a column or the table has the same name.

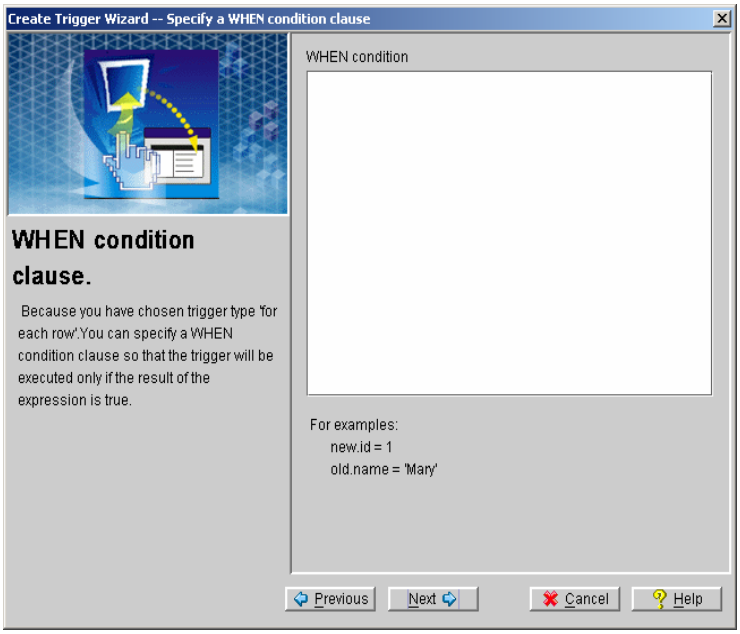
1. From the **Choose the Trigger Action Time** window of the Create Trigger Wizard click **Advanced Settings**. The **Referencing clause** window is displayed.
2. Enter a substitute name to refer to the old value in the **OLD** field.
3. Enter a substitute name to refer to the new value in the **NEW** field.
4. Click **OK**; return the **Choose the Trigger Action Time** window.
5. Click **Next**, the Specify the WHEN condition clause will open.



3.7.4. ENTERING THE WHEN CONDITION CLAUSE

Once the settings for the trigger action are complete, and you choose the for each row trigger to create, you may specify a WHEN clause to place constraints on the actions that will cause the trigger to fire. A WHEN statement must follow proper SQL syntax.

1. From the **Choose the Trigger Action Time window** of the Create Trigger Wizard (if For each row has been specified), click **Next**. The **Specify a WHEN condition clause** window is opened.



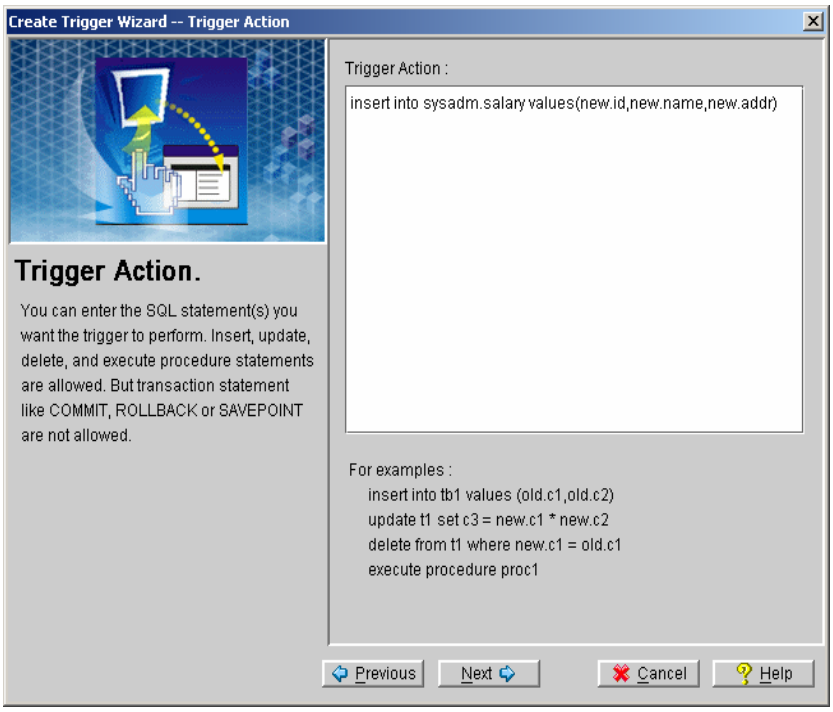
2. Enter the WHEN condition and click **Next**. The **Entering the statements for the trigger action** window will open.

3.7.5. ENTERING SQL STATEMENTS FOR THE TRIGGER ACTION

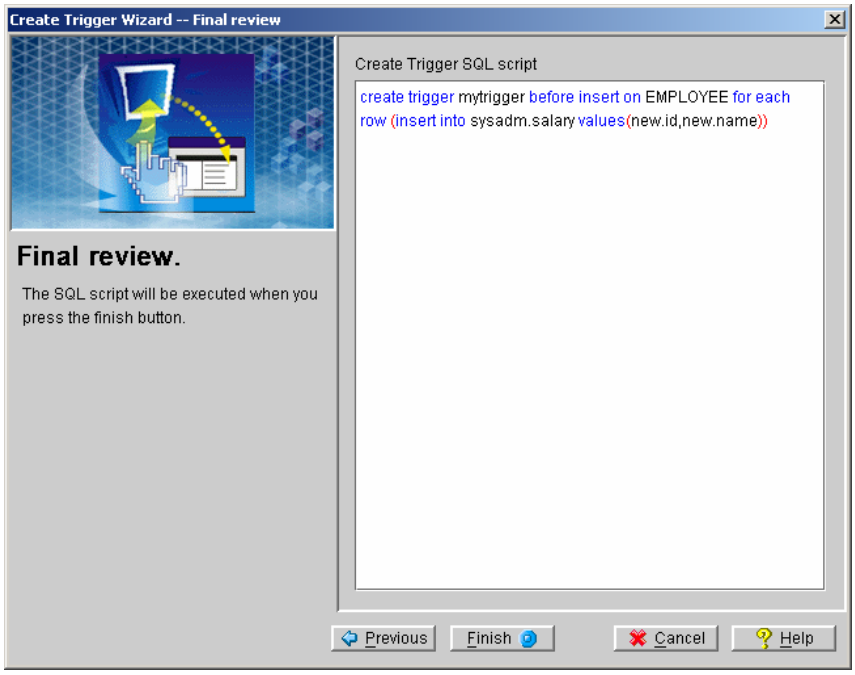
Once the trigger event and its constraints have been defined, the trigger action will be set. The trigger action is the command or set of commands that the trigger will carry out on all data that meet the constraints when the trigger is fired.

1. From the **Specify a WHEN condition clause**, click **Next**. The **Trigger Action** window will open.

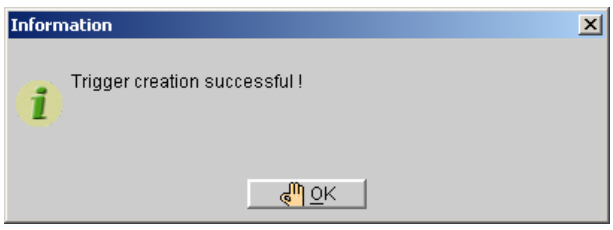
2. Enter the SQL statements that are to be performed by the trigger.



3. Click **Next**. The **Final review** window will open.



4. Review the final SQL script. Click **Finish** if no more changes are to be made. A message saying the trigger is created successfully is displayed.

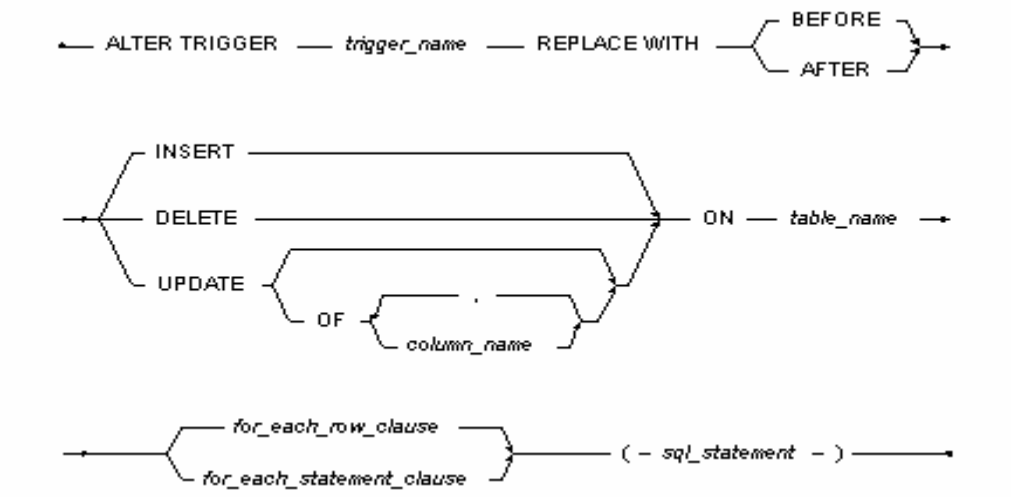


4. Modifying A Trigger

A trigger cannot be modified, but its definition can be replaced. When you want to modify a trigger definition, use the ALTER TRIGGER statement or the JDBATOOL. User can modify the trigger event, trigger action or trigger type.

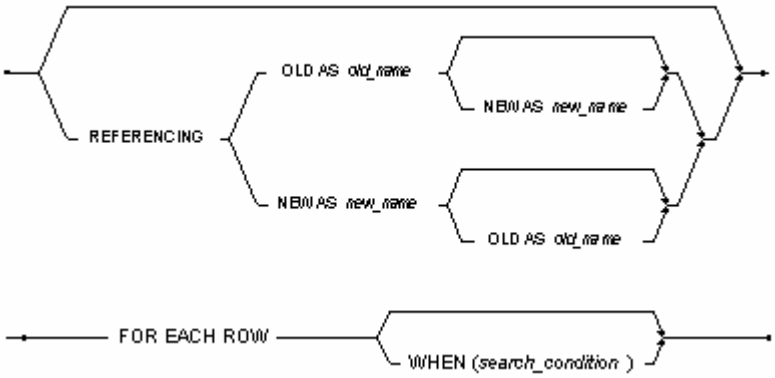
4.1. Using dmsql syntax

The ALTER TRIGGGER Syntax is as bellow:



Picture 4-1 Alter trigger syntax

FOR EACH ROW clause syntax:



Picture 4-2 FOR EACH ROW clause syntax

FOR EACH STATEMENT clause syntax:

Picture 4-3 FOR EACH STATEMENT clause syntax

To replace a trigger action, use the statement ALTER TRIGGER *trigger_name* REPLACE WITH.

➲ Example:

17. To change the Trigger Action for the **TRDELETE1** created in **Example 3**:

```
ALTER TRIGGER TRDELETE1 REPLACE WITH BEFORE DELETE ON SYSADM.EMPLOYEE
        FOR EACH ROW
                (DELETE FROM SYSADM.SALARY
                        WHERE NAME=OLD.NAME);
```

Now we use the **NAME** instead of the **ID** as the where condition.

18. To add another condition to the Trigger Action.

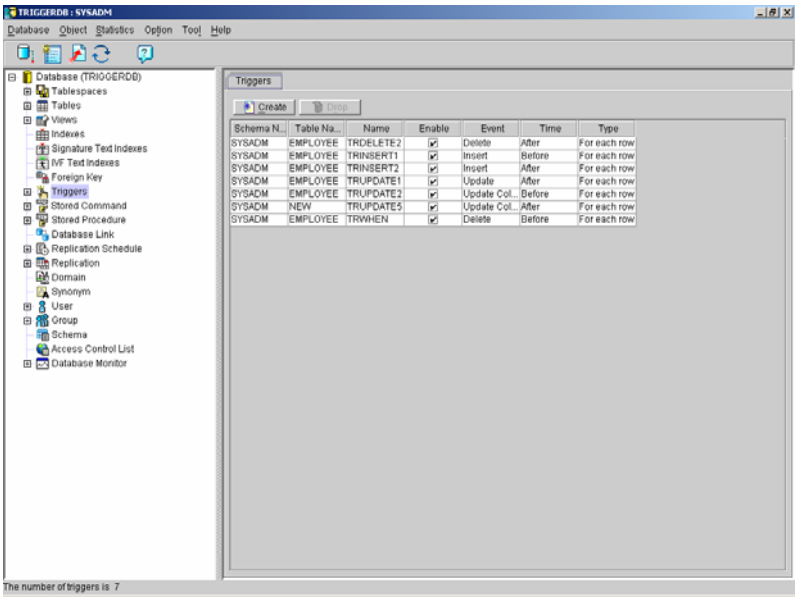
```
ALTER TRIGGER TRDELETE1 REPLACE WITH BEFORE DELETE ON SYSADM.EMPLOYEE
        FOR EACH ROW
                (DELETE FROM SYSADM.SALARY
                        WHERE NAME=OLD.NAME AND ID>15);
```

Now we add another condition **ID**>15.

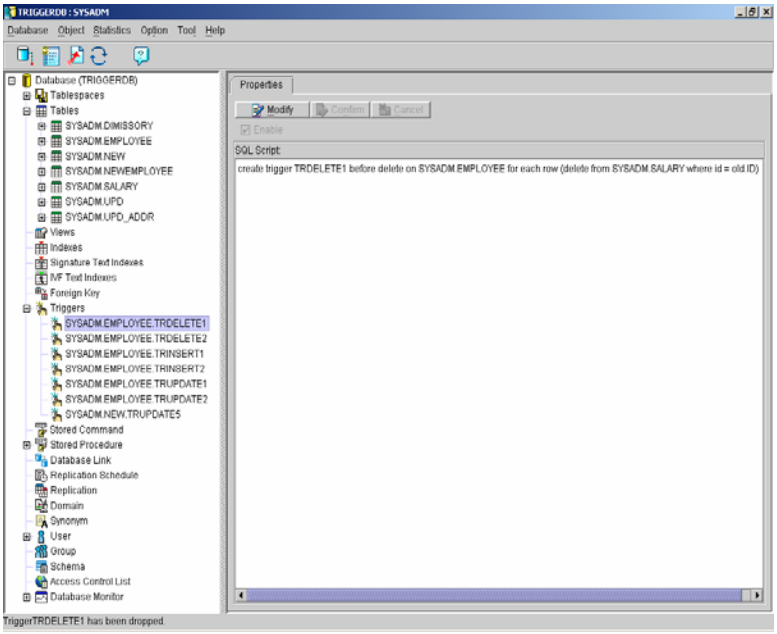
4.2. Using JDBATOOL

The same to the creating trigger, you can also use JDBATOOL to replace the trigger. The following figures guide you how to do it.

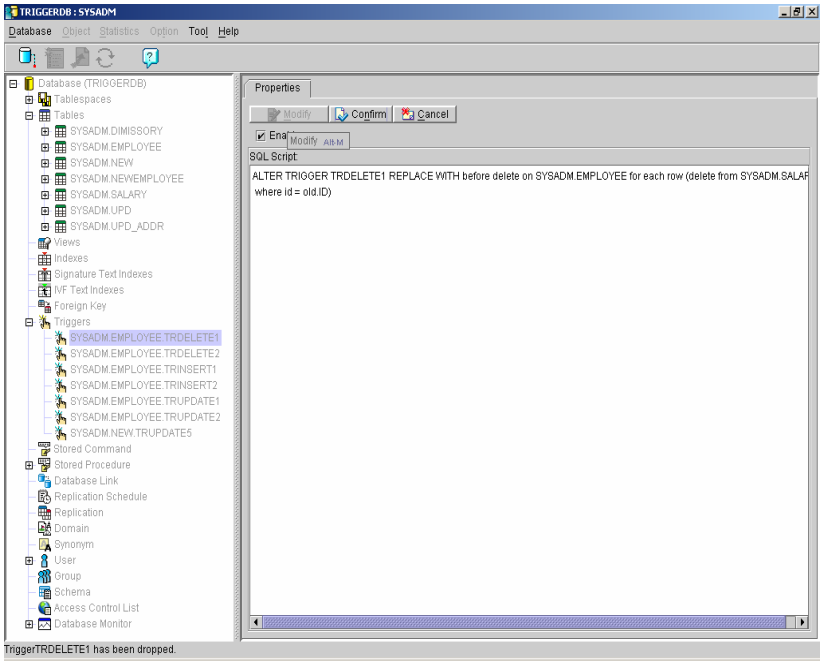
1. Click the object **Triggers** in the tree. All the triggers in the database will be displayed on the **Triggers** page.



2. Double click the trigger to be modified, or expand the Triggers node in the tree and select a trigger from the tree. The **Properties** page will appear.



3. Click the **Modify** button.



4. To disable the trigger, remove the check mark next to **Enable**.

5. To make changes to the SQL script, click in the appropriate place in the **SQL Script** field and edit the statement.

6. Click the **Confirm** button. The modified trigger will be displayed.

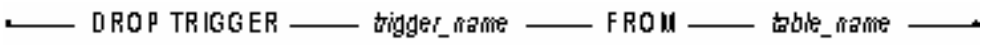
5. Dropping A Trigger

If you will not use a trigger any longer, you can drop it.

5.1. Using dmsql syntax

The DROP TRIGGER statement can be used to delete a trigger from the database. To drop a trigger, specify the name of the trigger to delete, and the associated table.

The DROP TRIGGER Syntax is as bellow:



Picture 5-1 DORP TIRGGER Syntax

➲ Example:

19. To drop a trigger:

```
DROP TRIGGER TRINSERT1 FROM EMPLOYEE;
```

Deleting a table will cause triggers referencing the table to be deleted. When a table schema is altered, DBMaster will try to execute the trigger according to the new table definition when the next time the trigger is executed. If the specified column in a trigger event or action is dropped, the trigger execution and statement will fail. The only solution is to drop the trigger or modify the trigger definition according to the new table schema.

20. The follow trigger is the trigger created in **Example 3**:

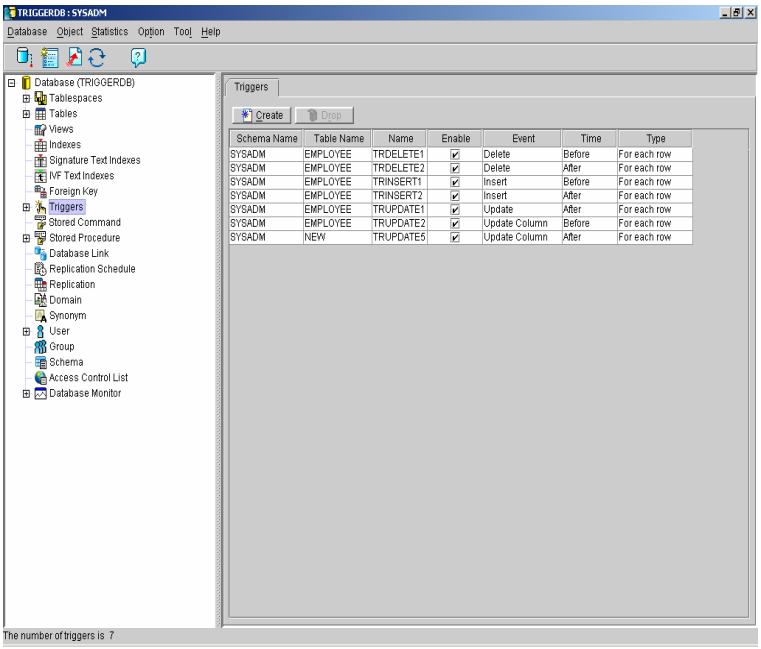
```
CREATE TRIGGER TRDELETE1 BEFORE DELETE ON SYSADM.EMPLOYEE
        FOR EACH ROW
              (DELETE FROM SYSADM.SALARY
                                    WHERE ID = OLD.ID);
```

If the column **ID** in table **SALARY** is dropped or the type is modified, an execution error will occur when the triggering statement (delete on **EMPLOYEE**) is performed causing the DBMS to attempt to fire trigger **TRDELETE1**.

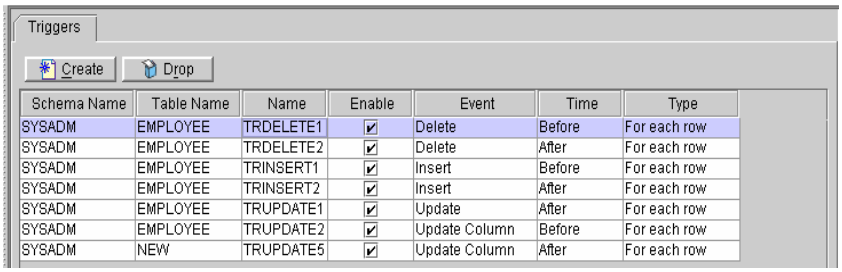
5.2. Using JDBATOOL

JDBATOOL allows user to drop the trigger too. The following figures guide you how to do it.

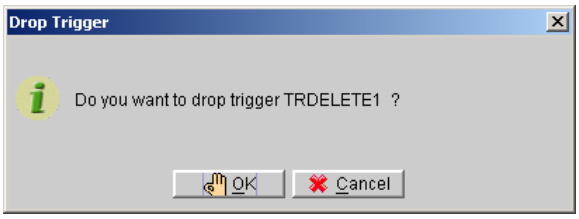
1. Click the object **Triggers** in the tree. All the triggers in the database will be displayed.



2. Select the trigger that is to be dropped by clicking on it.



3. Click **Drop**. A confirmation window will open to confirm if the trigger is to be dropped.



4. Click **OK**.

6. Using Triggers

When using triggers, there are more features for trigger you should know, such as how to use a stored procedure in the action body, what the order of the multiple trigger execute and so on. And they will be discussed in this chapter.

6.1. Stored Procedures in Action Body

One of the most powerful features of trigger is the ability to use a stored procedure as a trigger action. The Execute Procedure statement calls a stored procedure, enabling you to pass data from the triggering table to the stored procedure and then execute the procedure.

➲ Example:

21. To create a trigger and use the EXECUTE PROCEDURE statement: we have created a trigger named **TRINSERT2** previously in **example2** to insert data to table **NEWEMPLOYEE**, and now we implement the same function by executing Stored Procedure.

    First, we create a Stored Procedure (more information about stored procedure, please referring the *Stored Procedure User's Guide* of DBMaster):

```
EXEC SQL CREATE PROCEDURE SYSADM.PROC1(INT ID,CHAR(20) NAME, CHAR(20) ADDR);
       {
       EXEC SQL BEGIN CODE SECTION;
       EXEC SQL INSERT INTO NEWEMPLOYEE VALUES (:ID,:NAME,:ADDR);
       EXEC SQL END CODE SECTION;
       }
```

    Then, create a trigger and use EXECUTE PRODEDURE statement:

```
CREATE TRIGGER TRSP1 AFTER INSERT ON SYSADM.EMPLOYEE
       FOR EACH ROW
              (EXECUTE PROCEDURE
                     PROC1(NEW.ID,NEW.NAME,NEW.ADDR));
```

    NOTE: before you create this trigger, please drop the trigger **TRINSET2** first. Otherwise, DBMaster will return the error information: *ERROR (6573): [DBMaker] trigger of this type already exists*

Users can pass values to a stored procedure in the argument list. If the stored procedure call is part of the action for a row trigger, users can use the old and new correlation values to pass the column values it. If the stored procedure is part of an action statement trigger, users can only pass constants to the stored procedure. Within a trigger action, you can update non-triggering columns in the triggering table, with or without a stored procedure. A stored procedure fired by a trigger cannot contain transaction control statements, like BEGIN WORK, COMMIT WORK, ROLLBACK WORK, SAVEPOINT, or DDL statements. The stored procedure as a trigger action cannot be a cursory procedure that returns more than one row. Please see the *Stored Procedure User's Guide* for details about stored procedure.

NOTE: the Stored Procedure which be executed in the trigger should not with returned value, some keywords such as RETURNS STATUS are not allowed to appear.

6.2. Trigger Execution Order

If there are multiple triggers on one table, the column numbers in the triggering columns determine the order of trigger execution. The trigger execution begins with the trigger with the smallest triggering column number and proceeds in order to the highest number.

➲ Example:

22. To create trigger **TRUPDATE6** on table **EMPLOYEE:**

```
CREATE TRIGGER TRUPDATE6 BEFORE UPDATE OF addr ON SYSADM.EMPLOYEE
        FOR ECHA ROW
                (INSERT INTO SYSADM.UPD
                        VALUES (OLD.ID,OLD,ID,OLD.NAME,OLD.ADDR,OLD.ADDR))
```

23. To create trigger **TRUPDATE7** on table **EMPLOYEE**:

```
CREATE TRIGGER TRUPDATE7 BEFORE UPDATE OF id ON SYSADM.EMPLOYEE
        FOR EACH ROW
                (INSERT INTO SYSADM.UPD
                        VALUES (NEW.ID,NEW.ID,NEW.NAME,NEW.ADDR,NEW.ADDR))
```

Now, if operate the SQL statement *UPDATE EMPLOYEE SET ID=ID+1, ADDR='ADDR'* will fire two triggers – **TRUPDATE6** and **TRUPDATE7**. The **TRUPDATE7**, which has a lower triggering column number than **TRUPDATE6** will be executed first(**id** at the front of the **addr** in the table **EMPLOYEE**). So when look the table **UPD** we will find that the new record is on the front of the old record.

6.3. Security and Triggers

First, the user must have permission to run the trigger event; otherwise, the user cannot trigger the event. However, the user does not have to have permission to run the triggered action because the SQL statements in the triggered action operate under the domain privilege of the trigger owner. Once a trigger is created successfully, the trigger creator has privilege to execute the triggered action. Any one else who can issue the triggering statement can also fire the trigger.

For example, user **B** can update on both tables **T1** and **T2**, and user **A** can update **T1** only, but not **T2**. Now user **B** creates a trigger on update **T1**, and the action updates **T2**. When user **A** updates **T1**, the triggered action (UPDATE **T2**) is executed successfully since the triggered action is running under the domain privilege of user **B**. This security rule simplifies execution and eliminates the requirement for the user to have more privileges to execute the triggered action.

6.4. Cursors and Triggers

Update or Delete statements within a cursor act differently than a single update or delete statement. The entire trigger will be executed with each update or delete with the WHERE CURRENT OF clause.

For example, if four rows are changed with a cursor, the BEFORE/FOR EACH STATEMENT, BEFORE/FOR EACH ROW, AFTER/FOR EACH STATEMENT and AFTER/FOR EACH ROW triggers will be executed four times - once for each row.

6.5. Cascading Triggers

Executing one trigger may cause another trigger to also be executed. You can use cascading triggers to enforce referential integrity. DBMaster supports a maximum of 64 cascading triggers.

➲ Example:

Look the following two triggers, when you delete a data from the table **EMPLOYEE**, the data will be deleted from the table **SALARY** as we designed. But when there is data was deleted from the **SALARY**, we can delete the same data from the table **NEWEMPLOYEE** by firing another trigger.

24. First, the **TRDELETE1** was already created previously in **example3**:

```
CREATE TRIGGER TRDELETE1 BEFORE DELETE ON SYSADM.EMPLOYEE
        FOR EACH ROW
              (DELETE FROM SYSADM.SALARY
                                    WHERE ID = OLD.ID);
```

25. Then, create trigger **TRCAS** for AFTER DELETE on the table **SALARY**:

```
CREATE TRIGGER TRCAS AFTER DELETE ON SYSADM.SALARY
        FOR EACH ROW
              (DELETE FROM SYSADM.NEWEMPLOYEE
                     WHERE ID=OLD.ID)
```

7.Enabling and Disabling Triggers

When a trigger is created, the trigger is in ENABLED mode, which means the triggered action executes when the trigger event occurs.

Sometimes users may need to disable a trigger:

- When users have to load a large amount of data, disabling the triggers temporarily will speed up the loading operation.
- When the objects referenced in a trigger are unavailable.

➲ Example:

26. To disable the trigger **TRUPDATE1** on table **EMPLOYEE**:

```
dmSQL>AlTER TRIGGER TRUPDATE1 ON EMPLOYEE DISABLE;
```

27. To enable the trigger **TRUPDATE1** on table **EMPLOYEE**:

```
dmSQL>AlTER TRIGGER TRUPDATE1 ON EMPLOYEE ENABLE;
```

8.Creating Trigger Privileges

To create a trigger for a table, a user must be the table owner or DBA. The trigger creator must have privileges on all objects referenced in the CREATE TRIGGER statement to be sure successfully create a trigger.

In DBMaster, a trigger has no owner; it is associated with a table. The table owner and DBA have all privileges associated with a trigger. They can create, drop, or alter the triggers. The SQL statements in the trigger action operate under the domain privileges of the trigger owner, instead of the domain privileges of the user executing the trigger event.

Appendix Tables used in the Examples

The following table shows all the tables that used in the previous samples, including the detail information of tables. It can help user to understand the sample easily.

Table name	Table definition	Table description
EMPLOYEE	create table SYSADM.EMPLOYEE (ID SERIAL(1), NAME VARCHAR(20) DEFAULT NULL , ADDR VARCHAR(20) DEFAULT NULL) IN DEFTABLESPACE LOCK MODE PAGE FILLFACTOR 80;	Basic table contains the basic information of the employees;
SALARY	create table SYSADM.SALARY (ID SERIAL(1), NAME VARCHAR(20) default null , SALARY FLOAT default null) in DEFTABLESPACE LOCK MODE PAGE FILLFACTOR 80 ;	Table that stores the salary information of the employees.
NEWEMPLOYEE	create table SYSADM. NEWEMPLOYEE (ID SERIAL(1) , NAME VARCHAR(20) , ADDR VARCHAR(20)) IN DEFTABLESPACE LOCK MODE PAGE FILLFACTOR 80;	Used to store the data that was inserted into the table EMPLOYEE, that is the latest employee.
DIMISSORY	create table SYSADM. DIMISSORY (ID SERIAL(1), NAME VARCHAR(20) default null , ADDR VARCHAR(20) default null)	Used to store the date that was deleted from the table EMPLOYEE, that is the dimissory employee.

Table name	Table definition	Table description
	in DEFTABLESPACE LOCK MODE PAGE FILLFACTOR 80 ;	
UPD	create table SYSADM.UPD (OLDID INTEGER default null , NEWID INTEGER default null , NAME VARCHAR(20) default null , OLDADDR VARCHAR(20) default null , NEWADDR VARCHAR(20) default null) in DEFTABLESPACE lock mode page fillfactor 80 ;	Used to stored the data that was updated in table EMPLOYEE
NEW	create table SYSADM.NEW (ID SERIAL(1), NAME VARCHAR(20) DEFAULT NULL , ADDR VARCHAR(20) DEFAULT NULL) IN DEFTABLESPACE LOCK MODE PAGE FILLFACTOR 80;	Basic table

Table A-1 Tables used in the Examples