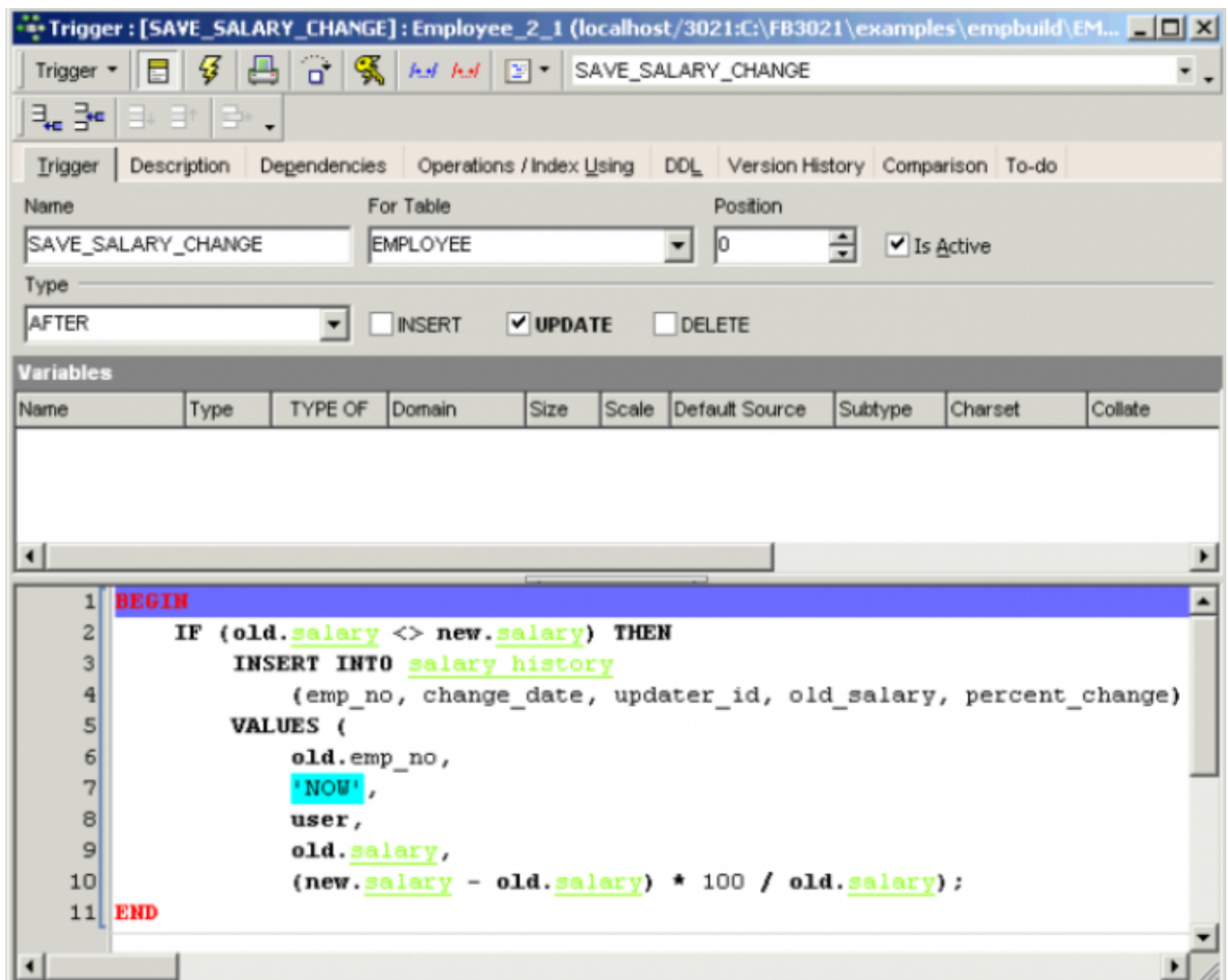# Trigger

A trigger is an independent series of commands stored as a self-contained program (SQL script) in the database. Triggers are executed automatically in the database when certain events occur. For example, it is possible to check before an insert, whether a primary key already exists or not, and if necessary allocate a value by a generator. These events are database-, table- or row-based.

Triggers are the so-called database police force, as they are vital for database integrity and security by enforcing the rules programmed by the database developer. They can include one or more execute commands. They can also be used as an alarm (= event alerter) that sends an event of a certain name to the Firebird/InterBase® *Event Manager*.

Triggers take no input parameters and do not return values.

A trigger is never called directly. Instead, when an application or user attempts to INSERT, UPDATE or DELETE a row in a table, any triggers associated with that table and operation automatically execute, or fire. Triggers defined for UPDATE on non-updatable views fire even if no update occurs.

The sequence in which triggers are specified is determined by the term TRIGGER POSITION, and different trigger types can be specified (see below).

They can be created, edited and deleted using the IBExpert DB Explorer right-click menu, from the Table Editor or Field Editor, or directly in the IBExpert SQL Editor.

Since Firebird 1.5 universal triggers (which can be used simultaneously for insert and/or update and/or delete) are available and Firebird 2.1 introduced database triggers (see below for further information). Firebird 2.1 also supports alternative syntax for the CREATE TRIGGER statement that complies with SQL2003. Please refer to the SQL2003 compliance for CREATE TRIGGER chapter in the Firebird 2.1 Release Notes for details.

Before Firebird 1.5, a trigger containin PLAN statement would be rejected by the compiler. Since Firebird 1.5 a valid plan can be included and will be used.

An example of a trigger:

```
CREATE TRIGGER TEST_TRIG FOR TEST
ACTIVE BEFORE INSERT POSITION 0
AS
begin
   if (new.id is null) then
       new.id=gen_id (GLOB_ID,1);
end
```

Several triggers can be created for one event. The POSITION parameter determines the sequence in which the triggers are executed.

Triggers are almost identical to stored procedures, the main difference being the way they are called. Triggers are called automatically when a change to a row in a table occurs, or certain database actions occur. Most of what is said about stored procedures applies to triggers as well, and they share the same language, PSQL. PSQL is a complete programming language for stored procedures and triggers. It includes:

- SQL data manipulation statements: INSERT, UPDATE, DELETE and singleton SELECT.
- SQL operators and expressions, including generators and UDFs that are linked with the calling application.
- Powerful extensions to SQL, including assignment statements, control-flow statements, context variables, event-posting statements, exceptions, and error-handling statements.

A Summary of PSQL commands can be found in the Stored procedure and trigger language chapter.

back to top of page

Database triggers

Database triggers were implemented in Firebird 2.1. These are user-defined PSQL modules that can be defined to fire in various connection-level and transaction-level events. This allows you to, for example, set up a protocol relatively quickly and easily.
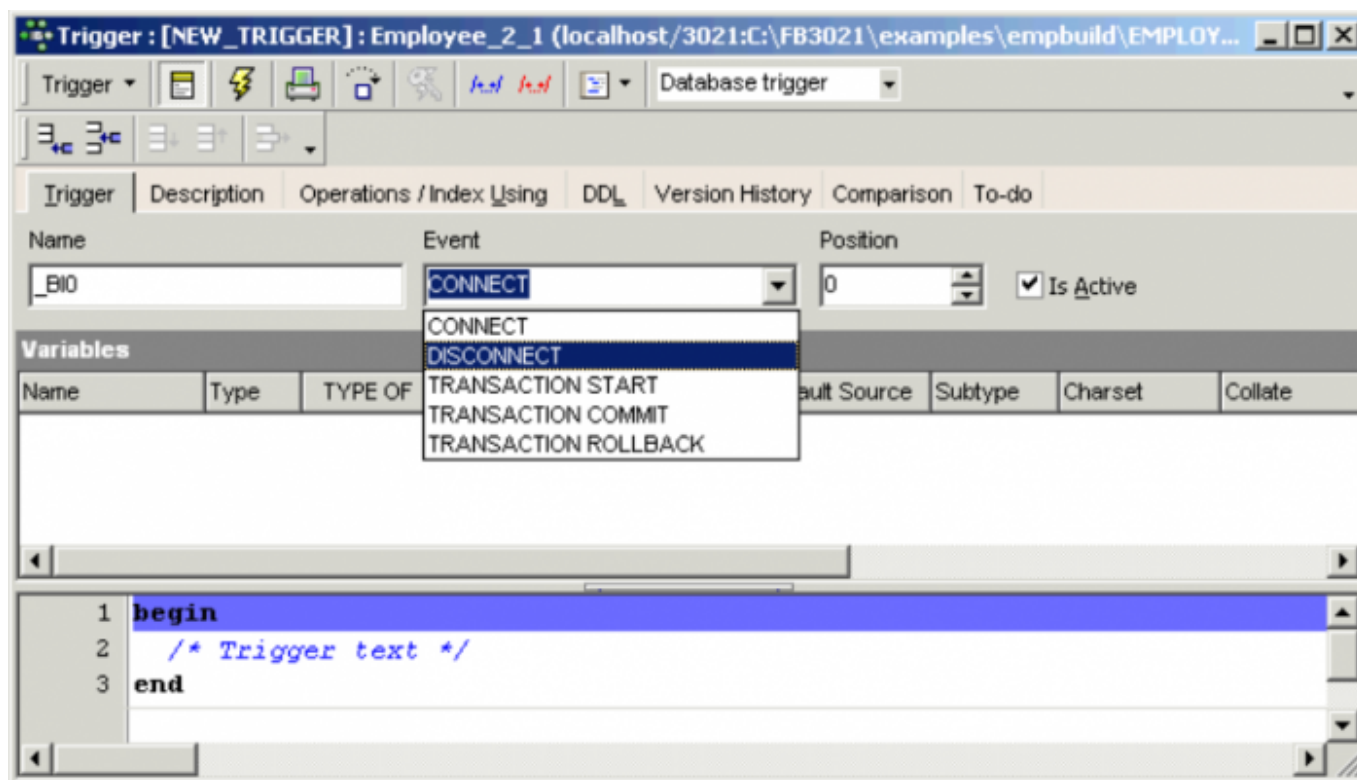
Database trigger types

Database-wide triggers can be fired on the following database trigger types:

CONNECT	The database connection is established, a transaction begins, triggers are fired - uncaught exceptions rollback the transaction, disconnect the attachment and are returned to the client. Finally the transaction is committed.
DISCONNECT	A transaction is started, triggers are fired - uncaught exceptions rollback the transaction, disconnect the attachment and are stopped. The transaction is committed and the attachment disconnected.
TRANSACTION START	Triggers are fired in the newly-created user transaction - uncaught exceptions are returned to the client and the transaction is rolled back.
TRANSACTION COMMIT	Triggers are fired in the committing transaction - uncaught exceptions rollback the trigger's savepoint, the commit command is aborted and an exception is returned to the client. For two-phase transactions the triggers are fired in PREPARE and not in COMMIT.
TRANSACTION ROLLBACK	Triggers are fired in the rolling-back transaction - changes made will be rolled back together with the transaction, and exceptions are stopped.

Only the SYSDBA or the database owner can:

- define database triggers
- switch them of for a new connection by:
  - new isc_dpb_no_db_triggers tag
  - new -no_dbtriggers switch in utilities

In IBExpert database triggers can be created, edited and deleted in the same way as table-bound triggers (see New trigger for details). Simply switch to *Database trigger* in the toolbar, to access the options specific to database triggers:

Specify who is allowed to access your application, or raise an exception when certain unwanted

applications attempt to access your database. Database triggers are also a really nice feature for protocols, enabling you for example to create your own login mapping with IP addresses an so on.

An example of a database trigger (source *Firebird 2.1* What's New, by Vladyslav Khorsum):

Example of an ON CONNECT trigger

```
isql temp.fdb -user SYSDBA -pass masterkey
Database: temp.fdb, User: SYSDBA
SQL> SET TERM ^ ;
SQL> CREATE EXCEPTION EX_CONNECT 'Forbidden !' ^
SQL> CREATE OR ALTER TRIGGER TRG_CONN ON CONNECT
CON> AS
CON> BEGIN
CON> IF (<bad user>)
CON> THEN EXCEPTION EX_CONNECT USER || ' not allowed !';
CON> END ^
SQL> EXIT ^

isql temp.fdb -user BAD_USER -pass ...
Statement failed, SQLCODE = -836
exception 217
-EX_CONNECT
-BAD_USER not allowed !
-At trigger 'TRG_CONN' line: 5, col: 3
Use CONNECT or CREATE DATABASE to specify a database
SQL> EXIT;
```

If you encounter problems with an ON CONNECT trigger, so that noone can connect to the database any more, use the -no_dbtriggers switch in the utilities:

```
isql temp.fdb -user SYSDBA -pass masterkey
-nodbtriggers Database: temp.fdb, User: SYSDBA
SQL> ALTER TRIGGER TRG_CONN INACTIVE;
SQL> EXIT;
```

Database triggers can be quickly and easily defined in IBExpert's Trigger Editor (see below).

See also:

Firebird 2.1 Release Notes: Database triggers

back to top of page

Table triggers

Table trigger types

Trigger types refer to the trigger status (ACTIVE or INACTIVE), the trigger position (BEFORE or AFTER) and the operation type (INSERT, UPDATE or DELETE).

They are specified following the definition of the table or view name, and before the trigger body.

ACTIVE or INACTIVE

ACTIVE or INACTIVE is specified at the time a trigger is created. ACTIVE is the default if neither of these keywords is specified. An inactive trigger does not execute.

BEFORE or AFTER

A trigger needs to be defined to fire either BEFORE or AFTER an operation. A BEFORE INSERT trigger fires before a new row is actually inserted into the table; an AFTER INSERT trigger fires after the row has been inserted.

BEFORE triggers are generally used for two purposes:

1. They can be used to determine whether the operation should proceed, i.e. certain parameters can be tested to determine whether the row should be inserted, updated or deleted or not. If not, an exception can be raised and the transaction rolled back.
2. BEFORE triggers can also be used to determine whether there are linked rows that might be affected by the operation. For example, a trigger might be used to automatically reassign sales before deleting a sales employee.

AFTER triggers are generally used to update columns in linked tables that depend on the row being inserted, updated or deleted for their values. For example, the PERCENT_CHANGE column in the SALARY_HISTORY table is maintained using an AFTER UPDATE trigger on the EMPLOYEE table.

To summarize: Use BEFORE until all data manipulation operations have been completed. The EMPLOYEE database trigger SET_CUST_NO is an example of a BEFORE INSERT, as a new customer number is generated before the data set has been inserted.

When manipulation of the table data should have been concluded before checking or altering other data, then use an AFTER trigger. The EMPLOYEE database trigger SAVE_SALARY_CHANGE is an example of AFTER UPDATE trigger, as the changes to the data have already been completed, before the trigger fires.

INSERT, UPDATE, DELETE

A trigger must be defined to fire on one of the keywords INSERT, UPDATE or DELETE.

1. An INSERT trigger fires before or after a row is inserted into the table.
2. An UPDATE trigger fires when a row is modified in the table.
3. A DELETE trigger fires when a row is deleted from the table.

If the same trigger needs to fire on more than one operation, a universal trigger needs to be defined. Before Firebird 1.5 triggers were restricted to either insert or update or delete actions, but now only one trigger needs to be created for all of these. For example:

```
AS
   BEGIN
      if (new.bez<>'')
      then new.bez=upper(new.bez);
   END
```

The '' UPPER applies to INSERT and UPDATE operations.

Please note that special characters, such as German umlauts, are not recognized and altered to upper case, as the character is treated technically as a special character, and not an alphabetical letter.

For further information regarding NEW variables, please refer to NEW and OLD context variables.

NEW and OLD context variables

In triggers (but not in stored procedures), Firebird/InterBase® provides two context variables that maintain information about the row being inserted, updated or deleted:

1. OLD.columnName refers to the current or previous values in a row being updated or deleted. It is not relevant for INSERT triggers.
2. NEW.columnName refers to the new values in a row being inserted or updated. It is not relevant for DELETE triggers.

Using the OLD. and NEW. values you can easily create history records, calculate the amount or percentage of change in a numeric value, find records in another table that match either the OLD. or NEW. value or do pretty well anything else you can think of. Please note that NEW. variables can be modified in a BEFORE trigger; since the introduction of Firebird 2.0 it is not so easy to alter them in an AFTER trigger. OLD. variables cannot be modified.

It is possible to read to or write from these trigger variables.

New to Firebird 2.0: Restrictions on assignment to context variables in triggers

- Assignments to the OLD context variables are now prohibited for every kind of trigger.
- Assignments to NEW context variables in AFTER-triggers are also prohibited.

Tip: If you receive an unexpected error Cannot update a read-only column then violation of one of these restrictions will be the source of the exception.

back to top of page

DDL triggers

DDL triggers are new to Firebird 3.0. They can be written to execute when database objects are modified or deleted. The purpose of a DDL trigger is to enable restrictions to be placed on users who attempt to create, alter or drop a DDL object. You can read more about DDL triggers in the Firebird 3.0.0 Alpha 1 Release Notes (excerpt from the initial version published 23 July 2013).
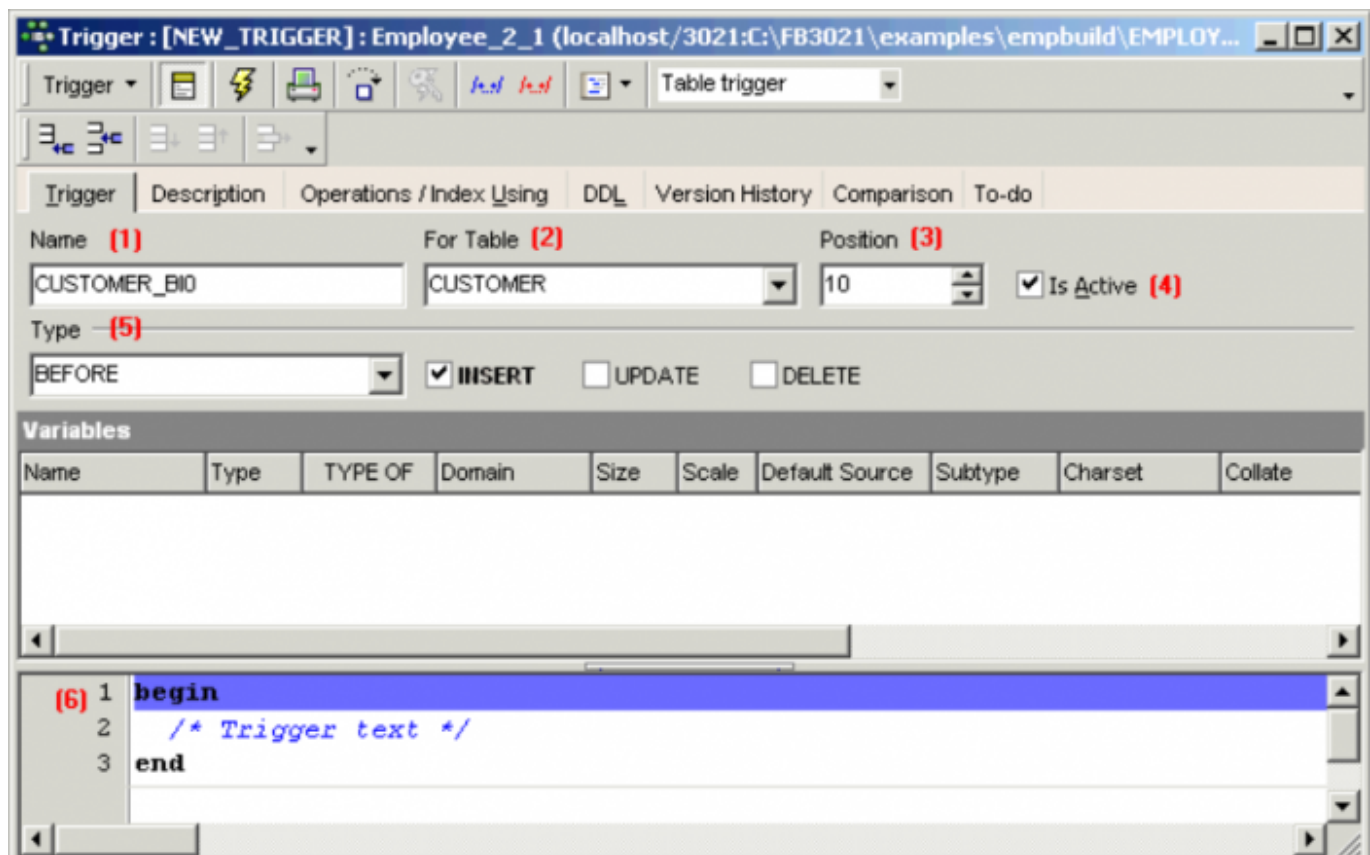
IBExpert supports DDL triggers since version 2014.01.01.

back to top of page

New trigger

There are numerous ways to create a trigger in IBExpert.

1. Using the IBExpert Database menu item, New Trigger or the respective icon on the New Database Object toolbar.
2. From the DB Explorer by right-clicking on the highlighted trigger branch of the relevant connected database (or key combination [Ctrl + N]).

Both these options open the Trigger Editor:

The Trigger Editor's first page allows the following to be specified simply and quickly, with the aid of pull-down lists, provided the lazy mode has been switched on:

- (1) **Name:** the trigger name can be altered as wished, if you do not wish to keep the default name. As with all database objects it is important to make rule about , which will aid you and other developers in the years to come to easily recognize objects, where they belong and their relationship to other objects. The illustration above depicts a BEFORE INSERT trigger. The name is composed of the table name, BI is the abbreviation for Before Insert and 10 denotes the specified position.
- (2) **For Table:** select the table or view name from the drop-down list.
- (3) **Position:** 255 positions are allowed per table, (starting at 0, up to 254). Several triggers on a table can also have the same firing position if it is irrelevant which one is fired first. As the positions do not have to be consecutive numbers it is wise to develop a convention, beginning let's say with 50, and numbering in 10 or 20 intervals. That way, you can insert and position new triggers at any time, without having to alter all your existing triggers to adjust the firing position. It's extremely important to layer the execution order of your triggers for logical reasons. For example, The before insert logging trigger on a table needs to know the data set's primary key, so the before insert primary key trigger needs to be fired first.
- (4) **Is Active:** check the box active/inactive as appropiate.
- (5) **Type:** specify trigger type as BEFORE or AFTER, and check the action(s) INSERT, UPATE and/or DELETE as wished. Checking all three manipulation options automatically generates a universal trigger.
- (6) **Trigger body:** The trigger body can be completed in the SQL window.

3. A trigger can also be created in the Table Editor or View Editor, on the *Triggers* page by selecting the desired BEFORE/AFTER operation and using the mouse right-click menu item *New Trigger*. This opens the *New Trigger Editor* shown above.
4. Or in the Field Editor on the Autoincrement page. For example, a trigger text for a new generator can be simply and quickly created using the *Edit Field / Autoinc*, *Create Generator* and then *Create Trigger*.

For those preferring direct SQL input, the CREATE TRIGGER statement has the following syntax:

```
CREATE TRIGGER <trigger_name>
FOR <table_name>
<keywords_for_trigger_type>
AS
<local_variable_declarations>
BEGIN
<body_of_trigger>
END
```

The trigger name needs to be unique within the database, and follow the Firebird/InterBase® naming conventions used for columns, tables, views and procedures.

Triggers can only be defined for a single database, table or updatable view. Triggers that should apply to multiple tables need to be called using a stored procedure. This can be done simply by creating a stored procedure which refers to the trigger. Please refer to the Using procedures to create and drop triggers chapter in the Firebird Development using IBExpert documentation.

Triggers fire when a row-based operation takes place on the named table or view.

back to top of page

Local variable declarations

Triggers use the same extensions to SQL that Firebird/InterBase® provides for stored procedures. Therefore, the following statements are also valid for triggers:

- DECLARE VARIABLE
- BEGIN ... END
- SELECT ... INTO : variable_list
- Variable = Expression
- /* comments */
- EXECUTE PROCEDURE
- FOR select DO ...
- IF condition THEN ... ELSE ...
- WHILE condition DO ...

As with stored procedures, the CREATE TRIGGER statement includes SQL statements that are conceptually nested inside this statement. In order for Firebird/InterBase® to correctly parse and interpret a trigger, the database software needs a way to terminate the CREATE TRIGGER that is different from the way the statements inside the CREATE TRIGGER are terminated. This can be done using the SET TERM statement.

Don't forget to finally compile the new trigger using the respective toolbar icon or [F9], and, if desired, autogrant privileges, again using the respective toolbar icon or key combination [Ctrl + F8].

As a trigger is associated with a table, the table owner and any user granted privileges to the table automatically have rights to execute associated triggers.

Triggers can be granted privileges on tables, just as users or procedures can be granted privileges. Use the Autogrant Privileges icon or the GRANT statement: here instead of using TO USERNAME, use TO TRIGGER trigger_name. Triggers' privileges can be revoked similarly using REVOKE.

When a user performs an action that fires a trigger, the trigger will have privileges to perform its actions if one of the following conditions is true:

- The trigger has privileges for the action.
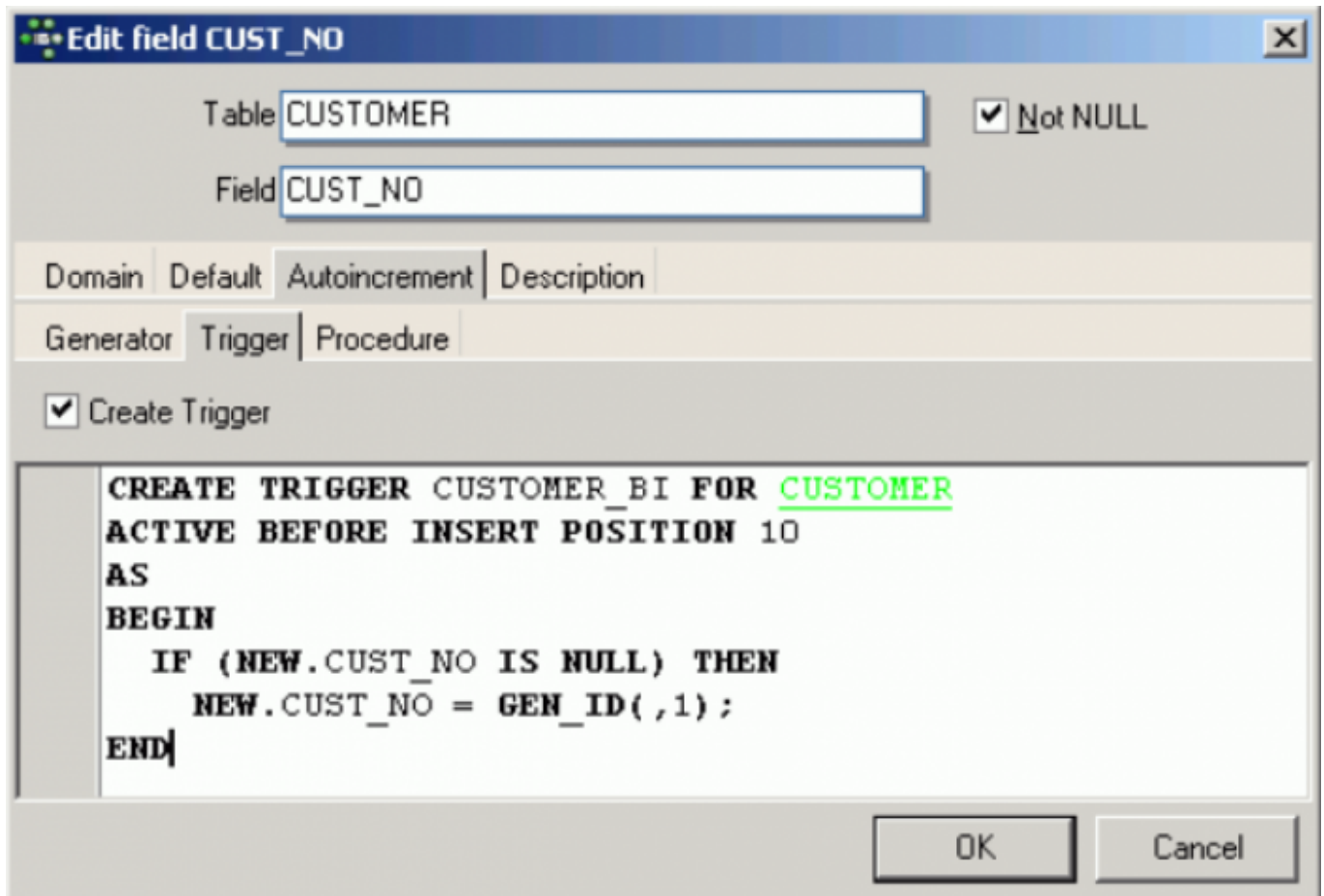- The user has privileges for the action.

See also:

Data Control Language - DCL

back to top of page

Create a trigger for a generator

Generally a generator is used to determine unique identification numbers for primary keys. A BEFORE INSERT trigger can be defined for this to generate a new ID, increasing the current value using the GEN_ID() function, and automatically entering it in the respective table field.

The above illustrates the Field Editor, started from the Table Editor.

See also:

Firebird Generator Guide: Creating unique row IDs

back to top of page

Create a trigger for a view

It is possible to create a trigger for a view directly in the View Editor on the Trigger page. This is particularly interesting for read-only views. For example, BEFORE INSERT, insert into Table1 new_fields and table2 new_data for fields. BEFORE UPDATES and BEFORE DELETE triggers should also be added, in order to distribute the data manipulation made in the view into the respective base tables.

back to top of page

Trigger Editor

The Trigger Editor can be started using the IBExpert Database menu item, *New Trigger*; from the DB Explorer, using the right mouse-click menu or double-clicking on an existing trigger, or alternatively directly from the View or Triggers page.
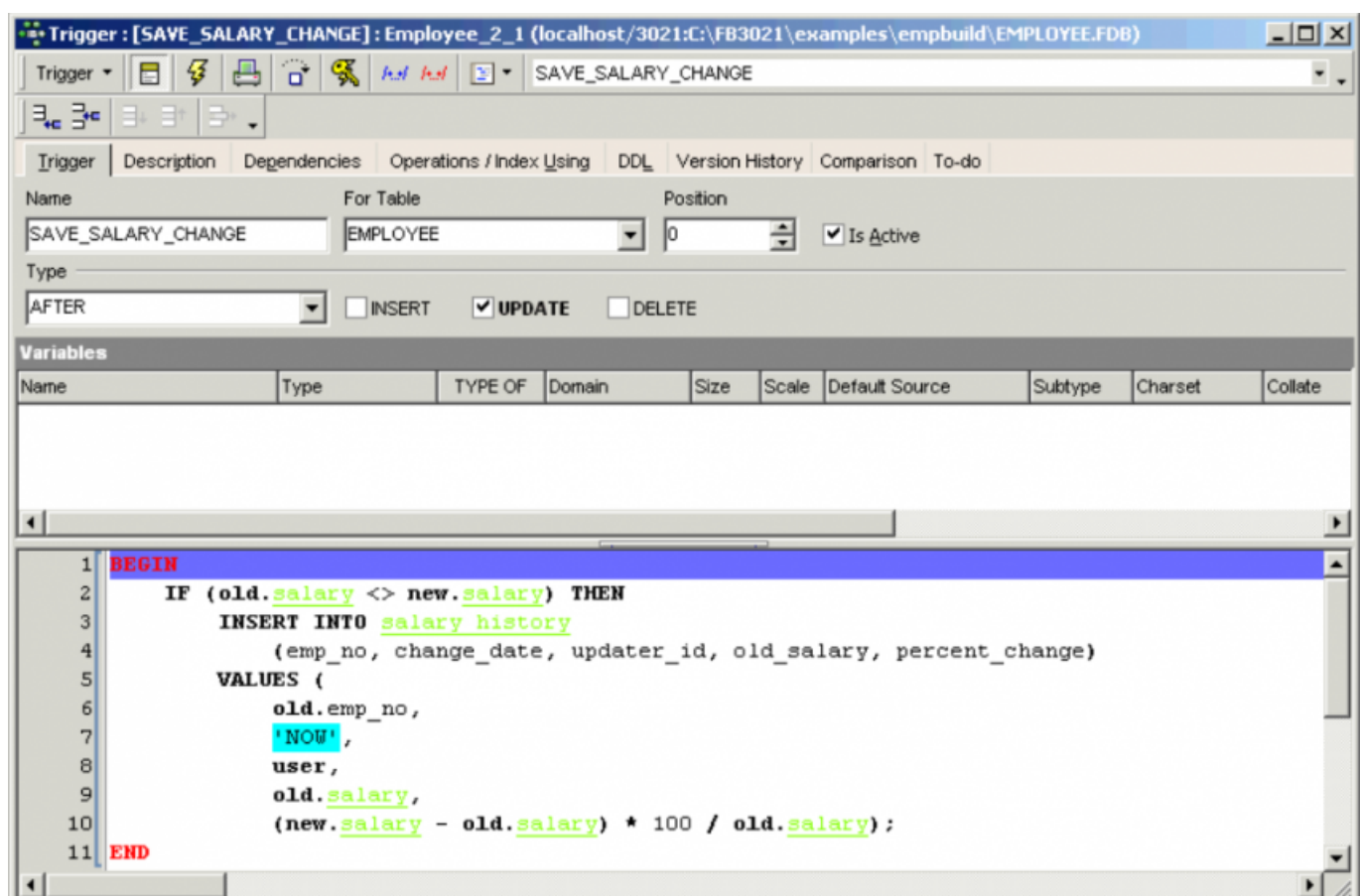
Please refer to New Trigger when creating a trigger for the first time.

The Trigger Editor has its own toolbar (see Trigger Editor toolbar) and offers the following options:

- Trigger
- Description
- Dependencies
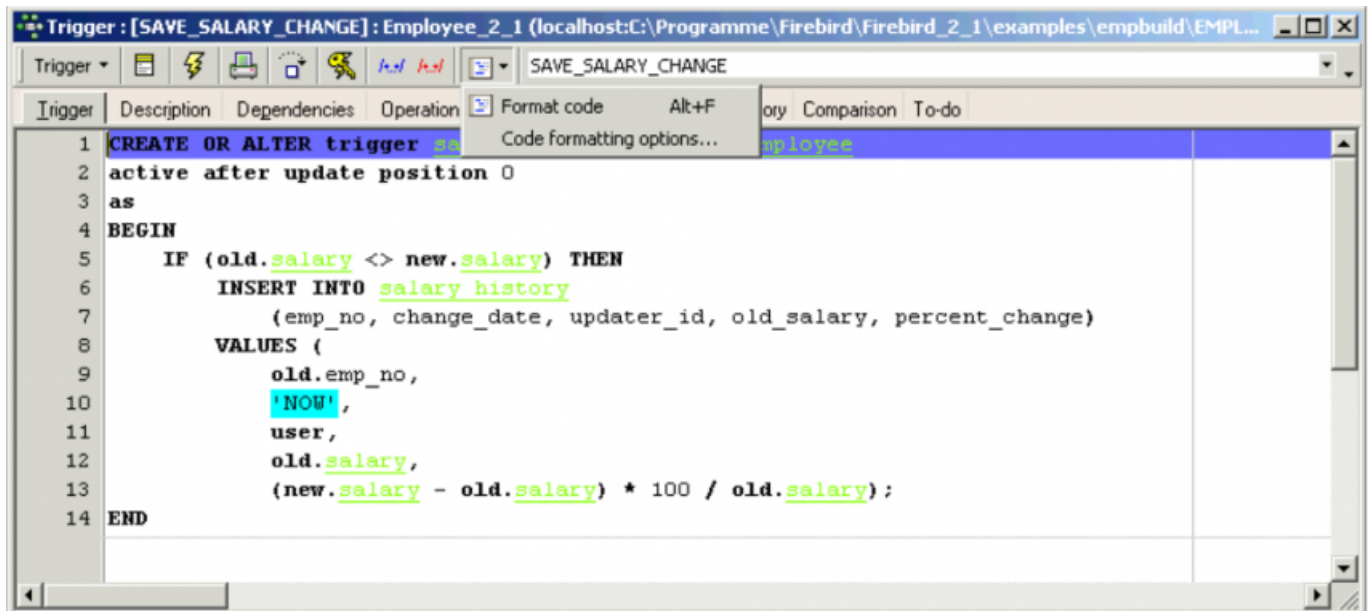- Operations/Index Using
- DDL
- Version History
- Comparison
- To-Do

The drop-down menu on the left of the toolbar also offers the options, Drop trigger and Edit trigger table [table_name].

Trigger page The Trigger Editor's first page allows the trigger name, table or view name, position, active/inactive, and trigger type to be specified simply and quickly, with the aid of pull-down lists, provided the lazy mode has been switched on:

If you are not able to view the Variables grid in the center of the window, check the Variables in grid option found in the IBExpert Options menu item, Object Editor Options on the Triggers editor page.

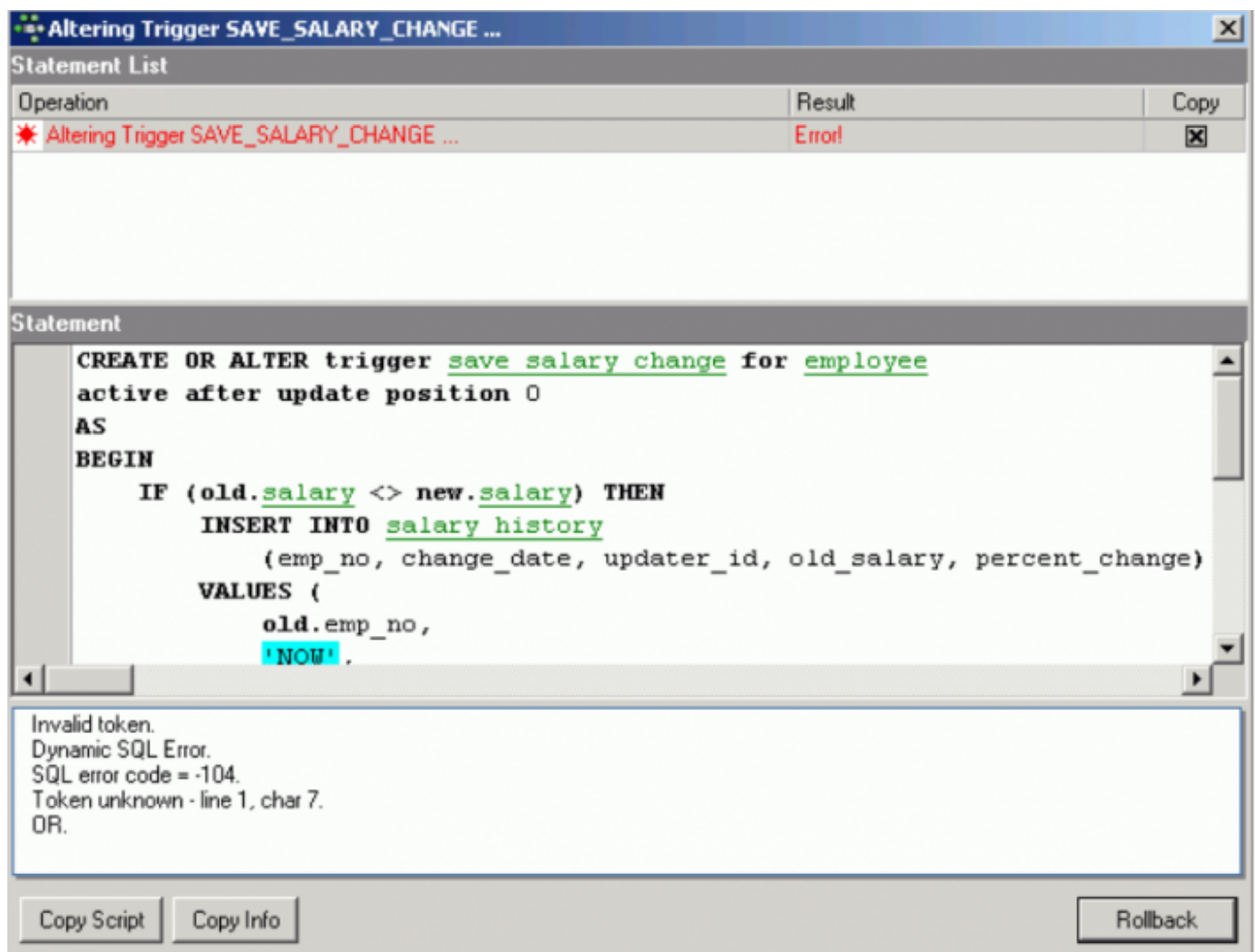If lazy mode switched off, all information needs to be specified in the SQL window:

The SQL window provides a template for both standard (for the whole trigger) and lazy mode, where the trigger body can be input. These templates can be altered if wished, using the IBExpert menu item Options / General Templates / New Trigger.

The Code Formatter enables you to format the source code of views, triggers and stored procedures. Code formatting options ... allows you to customize a range of specifications for all or for individual statements. Please refer to the IBExpert Options menu item, Code formatting options ... for further information.

As with all SQL input windows, the SQL Editor Menu can be called using the right mouse button. The keyboard shortcuts available in the SQL Editor are also available here. These options may be used to perform a number of actions, for example:

  * Comment or Uncomment code using the right-click context-sensitive menu.
  * Indent a marked block of code with [Ctrl + Shift + I] and move back with [Ctrl + Shift + U].

When the trigger or trigger alterations are complete, it can be compiled using the respective icon or [Ctrl + F9]. If errors are found, click YES when the Compile Anyway query appears, to produce an SQL error script (below the trigger text), to detect the error source.

If the problem is more complicated, the options *Copy Script* or *Copy Info* can be used before finally rolling back the trigger.

The Trigger Editor also has its own *Debug Trigger* icon. For more information regarding this, please refer to Debug Procedure or Trigger.

back to top of page

Description

Please refer to Table Editor / Description.

Dependencies

Please refer to Table Editor / Dependencies.

Operations/Index Using

Please refer to Procedure Editor / Operations / Index Using.

DDL

Please refer to Table Editor / DDL.

Version History

Please refer to View Editor / Version History.

Comparison

Please refer to Table Editor / Comparison.

To-do

Please refer to Table Editor / To-do.

[back to top of page](#)

Comment Trigger Body/Uncomment Trigger Body

In certain situations it may be necessary to disable certain commands or parts of trigger code. It is possible to do this temporarily, without it being necessary to delete these commands. Simply select the rows concerned in the [SQL Editor](#), and select either the editor toolbar icons:

the right mouse button menu item, *Comment Selected*, or key combination [Ctrl + Alt + .]. This alters command rows to comments. The commented text can be reinstated as SQL text by using *Uncomment Trigger Body* icon (above), the right mouse button menu item *Uncomment Selected*, or [Ctrl+ Alt + ,].

It is also possible to use the *Quick comment feature* available in all IBExpert code editors. Using the [Ctrl] + [Alt] + [.] shortcut (or select the right-click menu item, *Comment selected*), you can quickly comment the current selection of code or selected block. And use the right-click menu item, *Uncomment selected* or [Ctrl] + [Alt] + [,] shortcut to unselect.

It can not only be used to add comments and documentary notes to more complex stored procedures and triggers; but also to factor out selected parts of code during the testing phase, or even for customer applications, where certain features are not currently needed but may be required at a future date. The code can be reinstated by simply uncommenting as and when required.
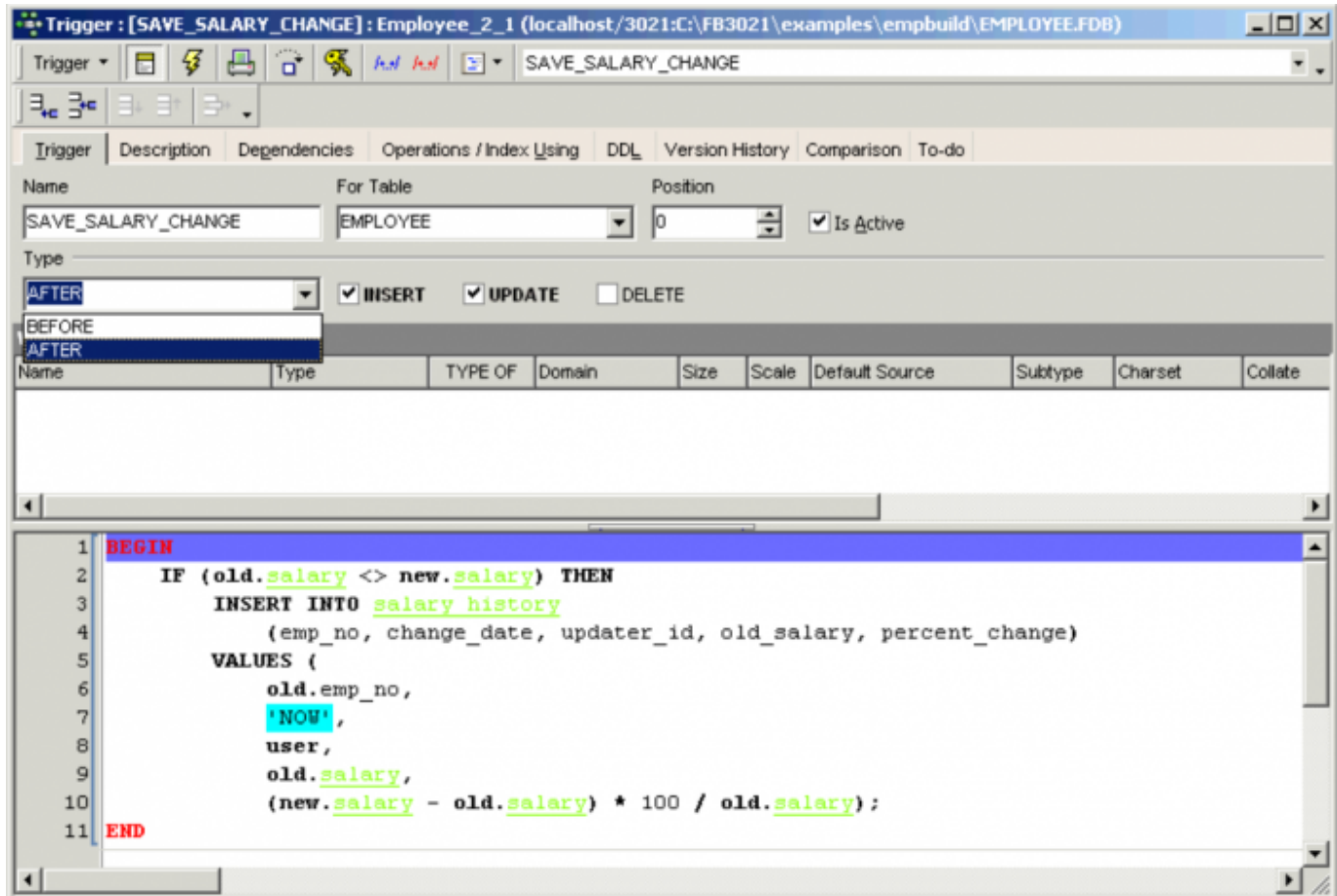
[back to top of page](#)

Edit trigger/alter trigger

Both the trigger header and the trigger body may be altered.

The trigger header may be activated or deactivated, or its position changed (in relation to other triggers).

If the trigger body needs to be altered, there is no need to make any alterations to the header, unless you wish to of course! Although in this case, it would probably make more sense to drop the trigger and create a new one. Any amendments to the trigger body override the original contents.

Triggers can easily be altered in the DB Explorer's [Trigger Editor](#), opened either by double-clicking on the trigger name, or right-clicking and selecting *Edit Trigger* [Ctrl + O]. The header information can be changed as wished using the drop-down lists to alter position, active/non-active and type:

(Image shows lazy mode.) The body text may be altered in the SQL panel as wished.

Finally the revised trigger needs to be compiled and committed, for the alterations to become effective.

Note: To alter a trigger defined automatically by a CHECK constraint on a table, use ALTER TABLE to change the constraint definition.

The SQL syntax for alterations to the trigger header is as follows:

```
ALTER TRIGGER name
[ACTIVE | INACTIVE]
[{BEFORE | AFTER} {DELETE | INSERT | UPDATE}]
[POSITION number]
```

where n is the new position number. Or to alter the trigger body:

```
ALTER TRIGGER <trigger_name>
AS
BEGIN
    <new_trigger_body>
END
```

If any of the arguments to ALTER TRIGGER are omitted, then they default to their current values, that is the value specified by CREATE TRIGGER, or the last ALTER TRIGGER.

A trigger can be altered by its creator, the SYSDBA user, and any users with operating system root

privileges.

Note: Each time you use CREATE, ALTER or DROP TRIGGER, InterBase® increments the metadata change counter of the associated table. Once that counter reaches 255, no more metadata changes are possible on the table (you can still work with the data though). A backup-restore cycle is needed to reset the counter and perform metadata operations again.

This obligatory cleanup after many metadata changes is in itself a useful feature, however it also means that users who regularly use ALTER TRIGGER to deactivate triggers during e.g. bulk import operations are forced to backup and restore much more often then needed. Since changes to triggers don't imply structural changes to the table itself, Firebird (since version 1.0) does not increment the table change counter when CREATE, ALTER or DROP TRIGGER is used. One thing has remained though: once the counter is at 255, you can no longer create, alter or drop triggers for that table.

A new syntax for changing triggers, or creating them if they do not already exist, was introduced in Firebird 2.0. Please refer to CREATE OR ALTER TRIGGER for further information.

back to top of page

Recreate trigger

New to Firebird 2.0: The DDL statement RECREATE TRIGGER is now available in DDL. Semantics are the same as for other RECREATE statements.
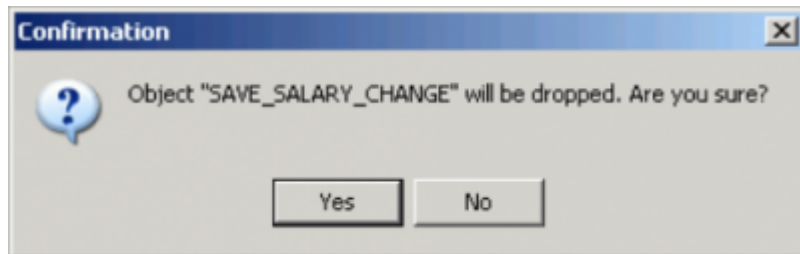
See also:

RECREATE TRIGGER

Drop trigger/delete trigger

Dropping or deleting a trigger removes a user-defined trigger definition from the database. System-defined triggers, such as those created for CHECK constraints, cannot be dropped. Use the IBExpert Table Editor or the ALTER TABLE command to drop the CHECK clause that defines the trigger.

A trigger can only be dropped if other users are not performing any changes to any tables which may relate to the specified trigger, at the time of deletion. In IBExpert, a trigger can be dropped from the DB Explorer by selecting the trigger to be deleted and using the right-click menu item *Drop Trigger* or [Ctrl + Del] or, if the trigger is already opened in the Trigger Editor, use the Trigger Editor main menü item, (opened by clicking *Trigger* in the top left-hand corner), *Drop Trigger*.

IBExpert asks for confirmation

before finally dropping.

For those preferring to use SQL, the syntax is as follows:

```
DROP TRIGGER <trigger_name>
```

An alternative solution to dropping triggers is to alter them to the INACTIVE status. That way they are left in the database, but disabled from firing, just in case they might be needed after all at a later date.

A trigger can be dropped by its creator, the SYSDBA user, or any user with operating system root privileges.

From:
http://ibexpert.com/docu/ - **IBExpert**

Permanent link:
http://ibexpert.com/docu/doku.php?id=01-documentation:01-13-miscellaneous:glossary:trigger

Last update: **2023/08/21 17:45**