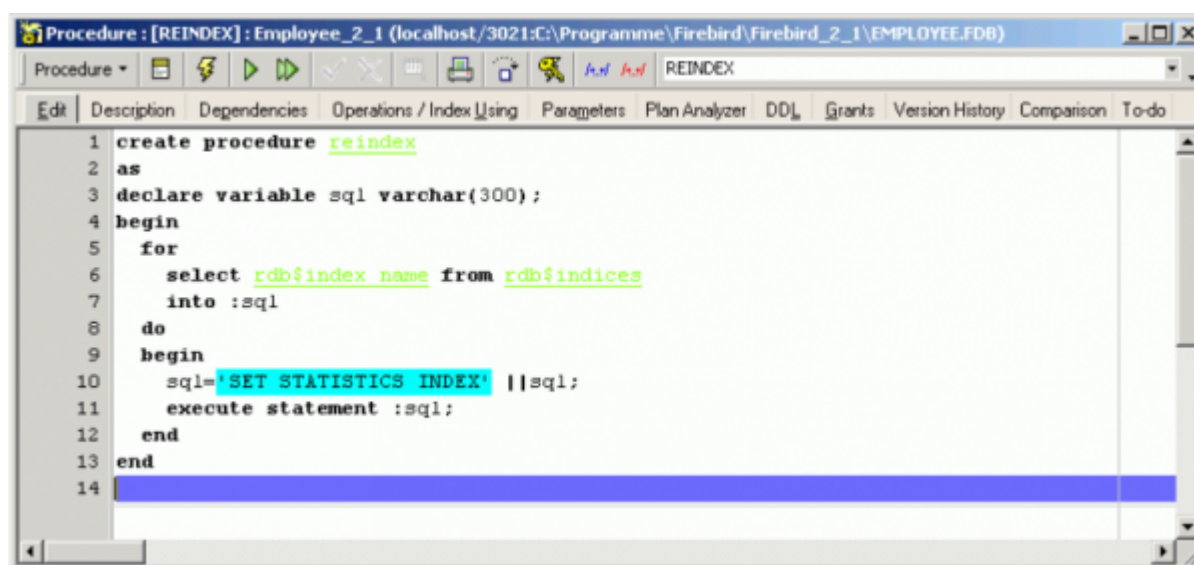


Firebird 2.0 Blocks

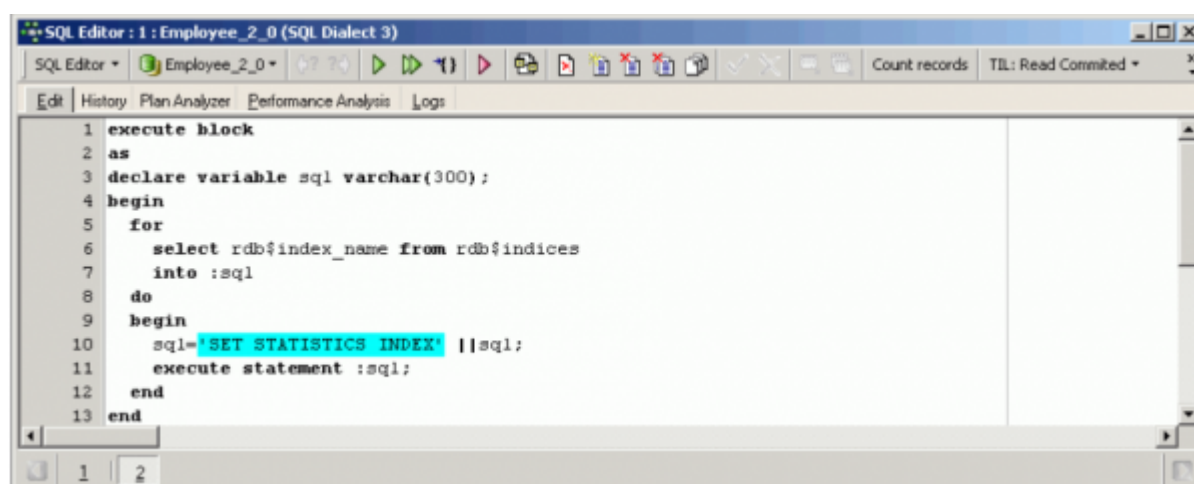
New to Firebird 2.0, Firebird's block implementation enables complex SQL operations in many application areas.

A block is a simple feature, using the new [EXECUTE BLOCK](#) syntax, which executes a block of [PSQL](#) code as if it were a stored procedure, optionally with [input and output parameters](#) and [variable](#) declarations. This allows the user to perform “on the fly” PSQL within a [DSQL](#) context. It performs a block of instructions on the server side, and can in fact be considered a virtual stored procedure.

To illustrate this, let's consider the following situation: you have a procedure, but you don't really want or need to store it in your database.:



You just want to create such a procedure on the fly and drop it afterwards. So make the following simple alterations:



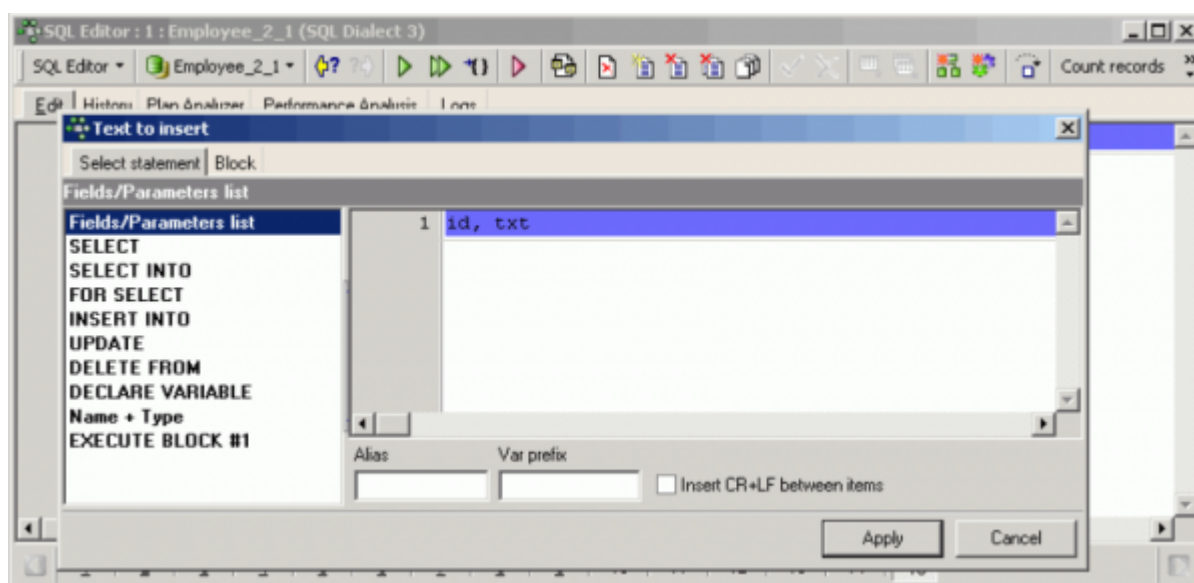
and it performs the same task, but as a dynamic block and not as a stored procedure.

The block transfers the source code from the client to the server, and executes it at the same speed as a stored procedure. The block is created and prepared when you start it, and deleted when you commit or roll back. The server will never use it again.

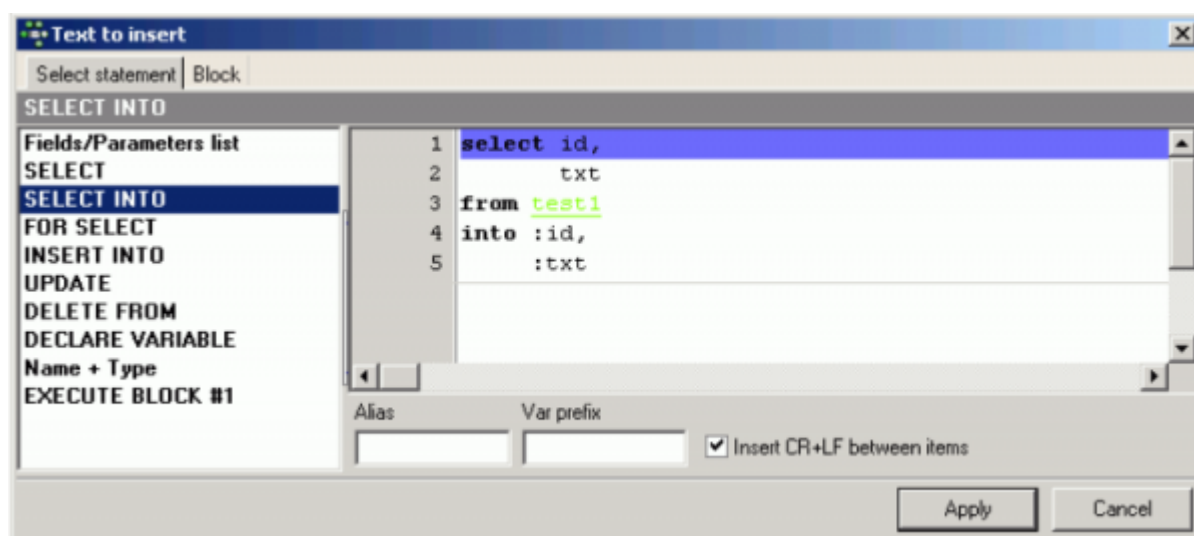
The major advantage of a block is when you are creating a variety of different but similar procedures from your client application, for example you have stored procedures for customer searching; in one stored procedure you are doing the customer search for the sales department, and in the other stored procedure you are doing the customer search for the invoice department. They have slightly different search criteria and want to see different columns in the result sets – this could be an interesting task, as the number of columns can be directly and dynamically created in a block.

EXECUTE BLOCK is not only a alternative to stored procedures; there are other uses, particularly for performance tasks.

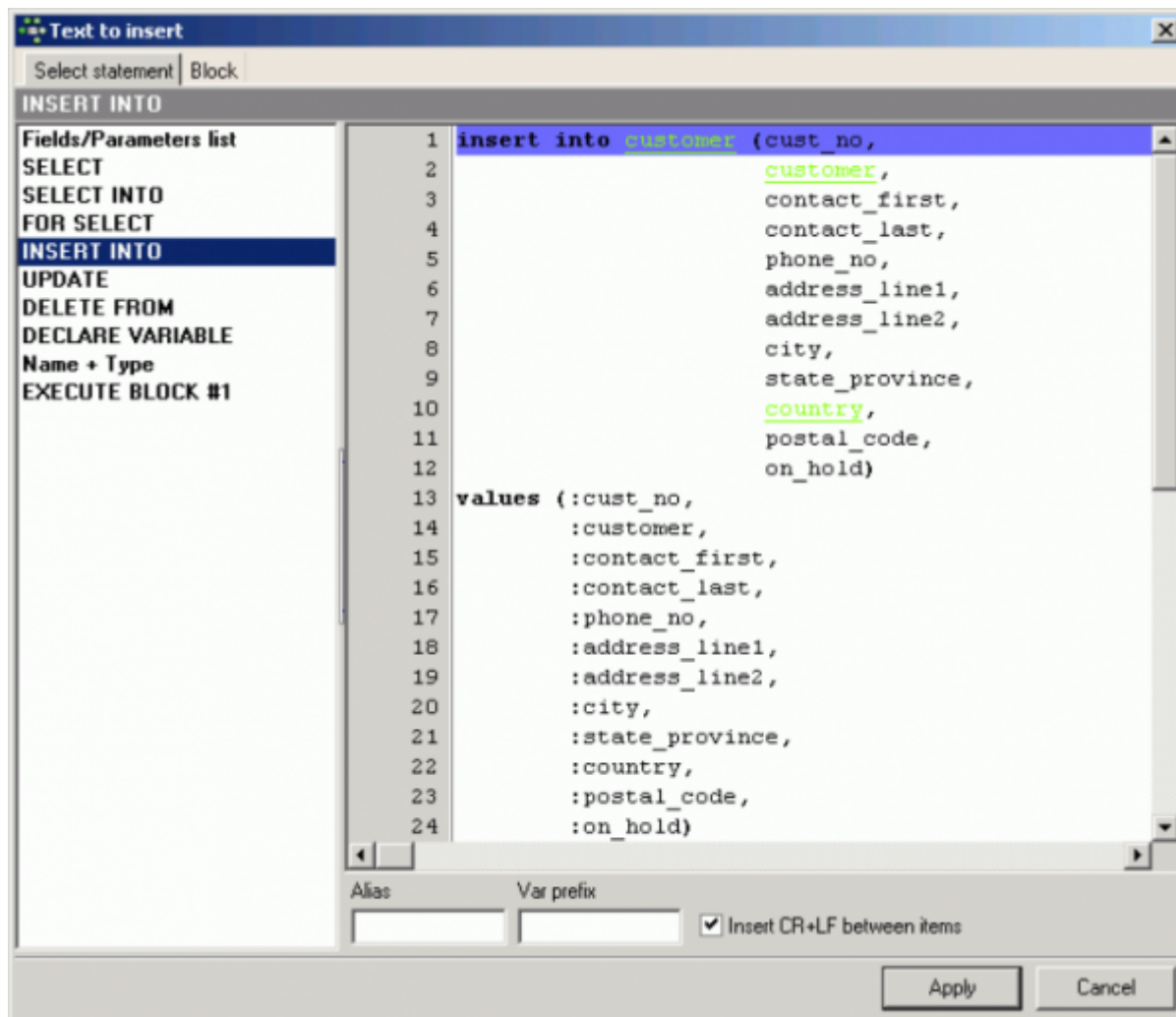
To illustrate this, for example, take a table TEST1, drag it from the [DB Explorer](#) into the [SQL Editor](#). The [Text to insert](#) window opens offering a range of options:



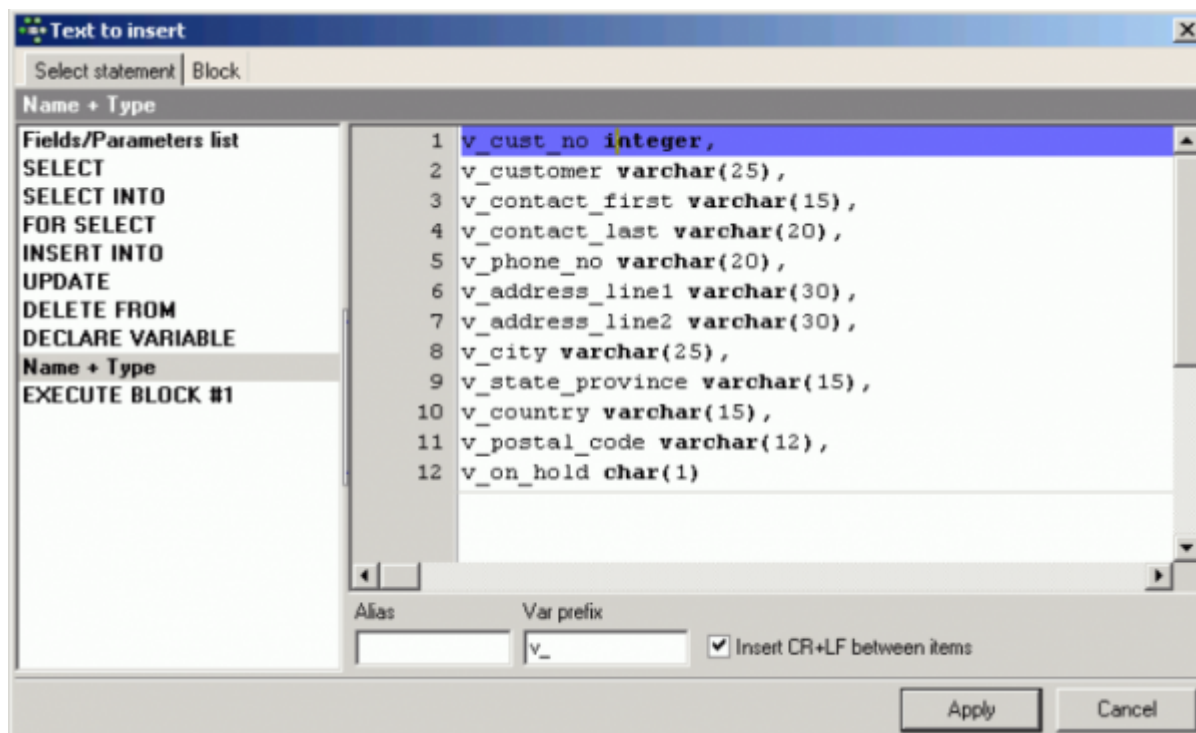
To prepare a `SELECT INTO` with carriage return and line feed, simply click on the `SELECT INTO` from the list on the left and check the *Insert CR+LF between items*. IBExpert then inserts the correct syntax:



In the case of this small table TEST1, this might not appear to be such an advantage, but if you take a look at a [table](#) with a larger number of [fields](#) (e.g. the `EMPLOYEE CUSTOMER` table), you will see how much it helps to have the field names and parameters already inserted into the standard syntax:

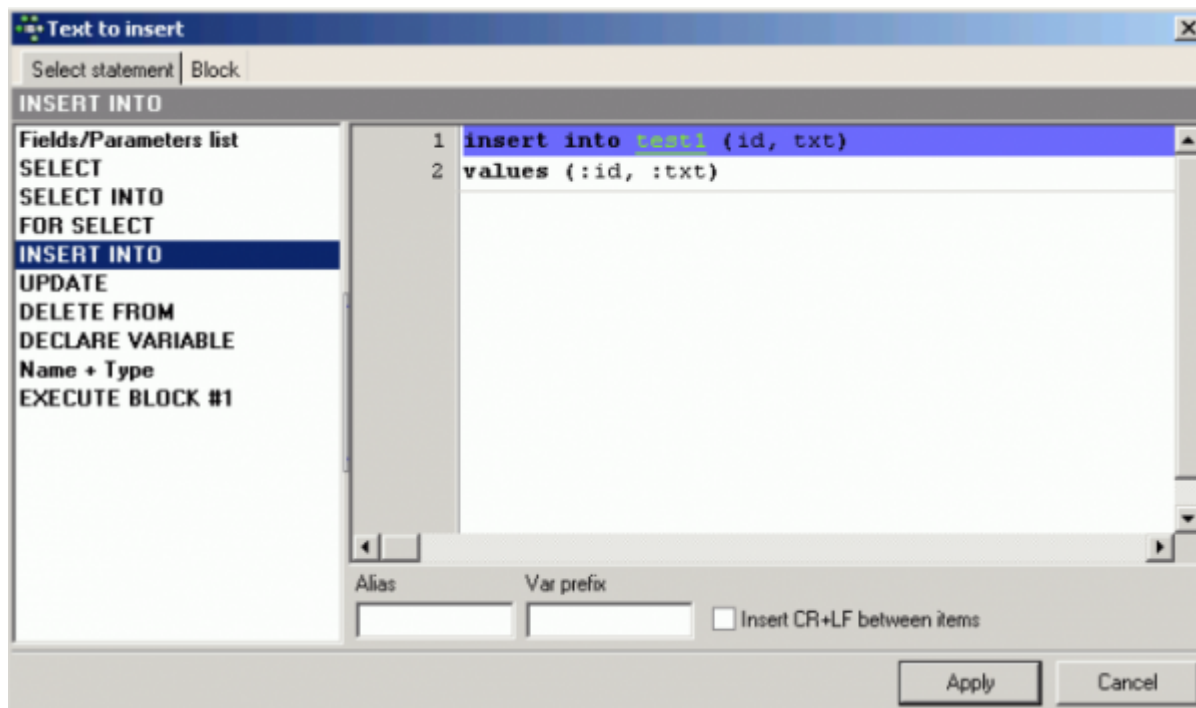


To ascertain the [datatype](#) definitions or to declare [variables](#), simply click on the *Name + Type* in the left-hand list. Variable prefixes can be inserted (for example: v_) in the field *Var prefix* below, to offer you an instant full list of variables for all [fields](#) in the table.

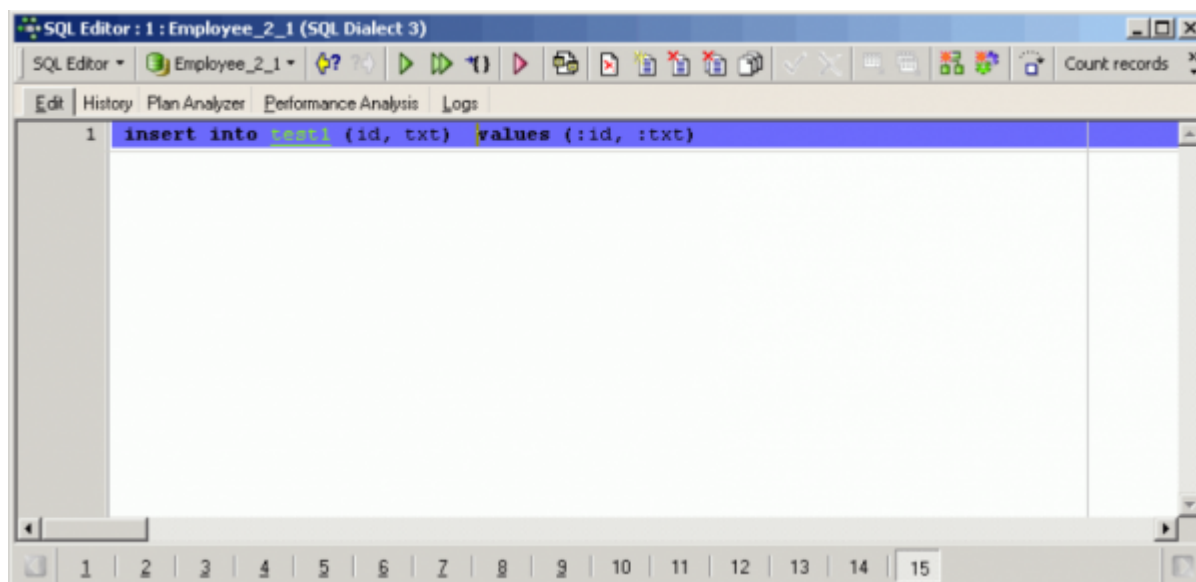


Firebird 2.1 also introduced the possibility to use [domains](#) for procedures, procedure [parameters](#) and so on. (Please refer to [Using domains in procedures](#) and the [Firebird 2.1 Release Notes](#) chapter, [Procedural SQL](#) for details and examples).

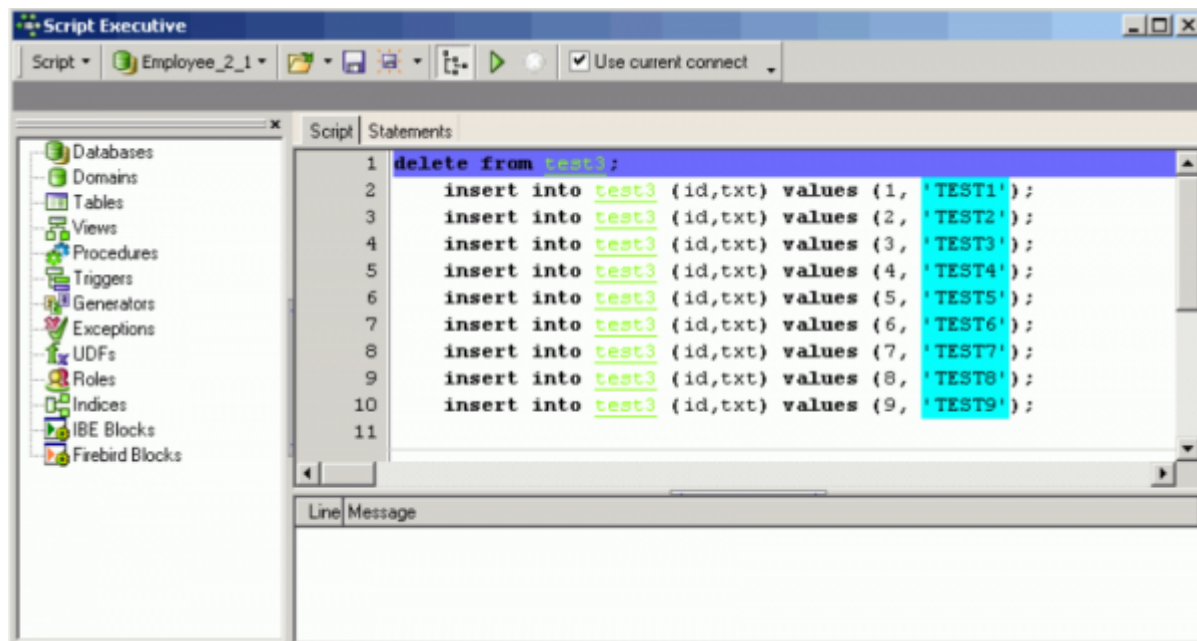
To continue with the implementation of the TEST1 table: an INSERT INTO statement is specified, without carriage return and line feed or a variable prefix:



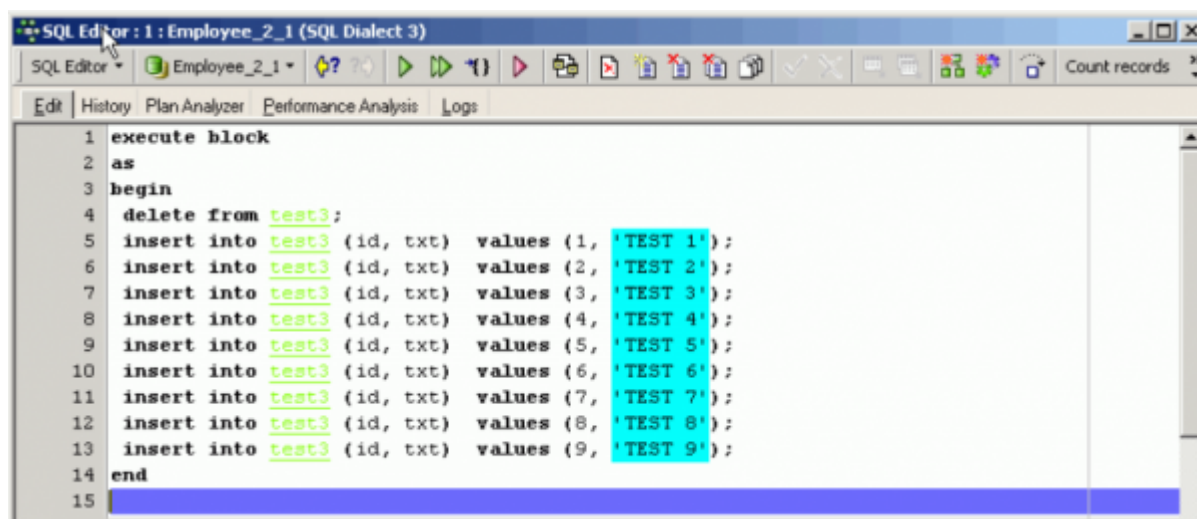
When it's ready simply apply and the INSERT INTO command is already formulated in the [SQL Editor](#) or [Script Executive](#):



Now to illustrate one of the main advantages of Firebird blocks, some operations are added, one by one:



Add the beginning and closing clauses, to turn these statements into a block:



The Firebird server now processes all operations in one go, and you can see that all operations have been sent as one package to the server.


```

TCP/IP Expert V1.0
File Mappings Screenlog Help
Status Logging

## 13:15:53:218 Channel 1; DATA (Server -> Client): Length= 32
00 00 00 09 00 00 00 01 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 01 00 00 00 00 00 00 00 00

## 13:15:53:218 Channel 1; DATA (Client -> Server): Length= 588
00 00 00 3E 00 00 00 00 00 00 44 00 00 00 01 > D
FF FF FF FF 00 00 00 03 00 00 02 09 65 78 65 63 exec
75 74 65 20 62 6C 6F 63 6B 0D 0A 61 73 0D 0A 62 ute block as b
65 67 69 6E 0D 0A 20 20 64 65 6C 65 74 65 20 66 eqin delete f
72 6F 6D 20 74 65 73 74 33 3B 0D 0A 20 20 69 6E rom test3; in
73 65 72 74 20 69 6E 74 6F 20 74 65 73 74 33 20 sert into test3
28 69 64 2C 20 74 78 74 29 20 76 61 6C 75 65 73 (id, txt) values
20 28 31 2C 20 27 54 45 53 54 31 27 29 3B 0D 0A (1, 'TEST1');
20 20 69 6E 73 65 72 74 20 69 6E 74 6F 20 74 65 insert into te
73 74 33 20 28 69 64 2C 20 74 78 74 29 20 76 61 st3 (id, txt) va
6C 75 65 73 20 28 32 2C 20 27 54 45 53 54 32 27 lues (2, 'TEST2'
29 3B 0D 0A 20 20 69 6E 73 65 72 74 20 69 6E 74 ); insert int
6F 20 74 65 73 74 33 20 28 69 64 2C 20 74 78 74 o test3 (id, txt
29 20 76 61 6C 75 65 73 20 28 33 2C 20 27 54 45 ) values (3, 'TE
53 54 33 27 29 3B 0D 0A 20 20 69 6E 73 65 72 74 ST3'); insert
20 69 6E 74 6F 20 74 65 73 74 33 20 28 69 64 2C into test3 (id,
20 74 78 74 29 20 76 61 6C 75 65 73 20 28 34 2C txt) values (4,
20 27 54 45 53 54 34 27 29 3B 0D 0A 20 20 69 6E 'TEST4'); in
73 65 72 74 20 69 6E 74 6F 20 74 65 73 74 33 20 sert into test3
28 69 64 2C 20 74 78 74 29 20 76 61 6C 75 65 73 (id, txt) values
20 28 35 2C 20 27 54 45 53 54 35 27 29 3B 0D 0A (5, 'TEST5');
20 20 69 6E 73 65 72 74 20 69 6E 74 6F 20 74 65 insert into te
73 74 33 20 28 69 64 2C 20 74 78 74 29 20 76 61 st3 (id, txt) va
6C 75 65 73 20 28 36 2C 20 27 54 45 53 54 36 27 lues (6, 'TEST6'
29 3B 0D 0A 20 20 69 6E 73 65 72 74 20 69 6E 74 ); insert int
6F 20 74 65 73 74 33 20 28 69 64 2C 20 74 78 74 o test3 (id, txt
29 20 76 61 6C 75 65 73 20 28 37 2C 20 27 54 45 ) values (7, 'TE
53 54 37 27 29 3B 0D 0A 20 20 69 6E 73 65 72 74 ST7'); insert
20 69 6E 74 6F 20 74 65 73 74 33 20 28 69 64 2C into test3 (id,
20 74 78 74 29 20 76 61 6C 75 65 73 20 28 38 2C txt) values (8,
20 27 54 45 53 54 38 27 29 3B 0D 0A 20 20 69 6E 'TEST8'); in
73 65 72 74 20 69 6E 74 6F 20 74 65 73 74 33 20 sert into test3
28 69 64 2C 20 74 78 74 29 20 76 61 6C 75 65 73 (id, txt) values
20 28 39 2C 20 27 54 45 53 54 39 27 29 3B 0D 0A (9, 'TEST9');
65 6E 64 0D 0A 00 00 00 00 00 00 19 15 04 07 09 end
0B 0C 0D 0E 10 11 12 13 08 05 07 09 0B 0C 0D 0E
10 11 12 13 08 00 00 00 FF FF 80 00

```

Especially when you need to insert or update a large amount of data, you can write your [application](#) in such a way as this, storing all the insert/update statements as a [TString](#) list or similar, writing [EXECUTE BLOCK](#) in front of it, concluding with an [END](#), and executing it as a single [statement](#).

Firebird 2.0 blocks can also be debugged directly in the [SQL Editor](#)SQL Editor (or alternatively in the [Block Editor](#)) using the [Block Debugger](#).

There is a limit to the amount of source code that can be transferred in a single package, it may not be larger than 32 Kb. In the case of larger data packets, it is necessary to split them into multiple packages, but this is usually still more efficient that sending each command individually.

Transactions cannot be controlled from inside a block because the block is always a part of your client transaction.

Blocks were implemented in Firebird 2.0. InterBase® 2007 introduced something similar but it does not have all the functionalities that Firebird has.

When you are working with IBExpert, you can use [IBEBlocks](#). Simply write [IBEBLOCK](#) instead of [BLOCK](#) and it still works!

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

Permanent link:
<http://ibexpert.com/docu/doku.php?id=01-documentation:01-06-white-papers:firebird-development-using-ibexpert:firebird2-blocks>

Last update: **2023/06/21 17:58**

