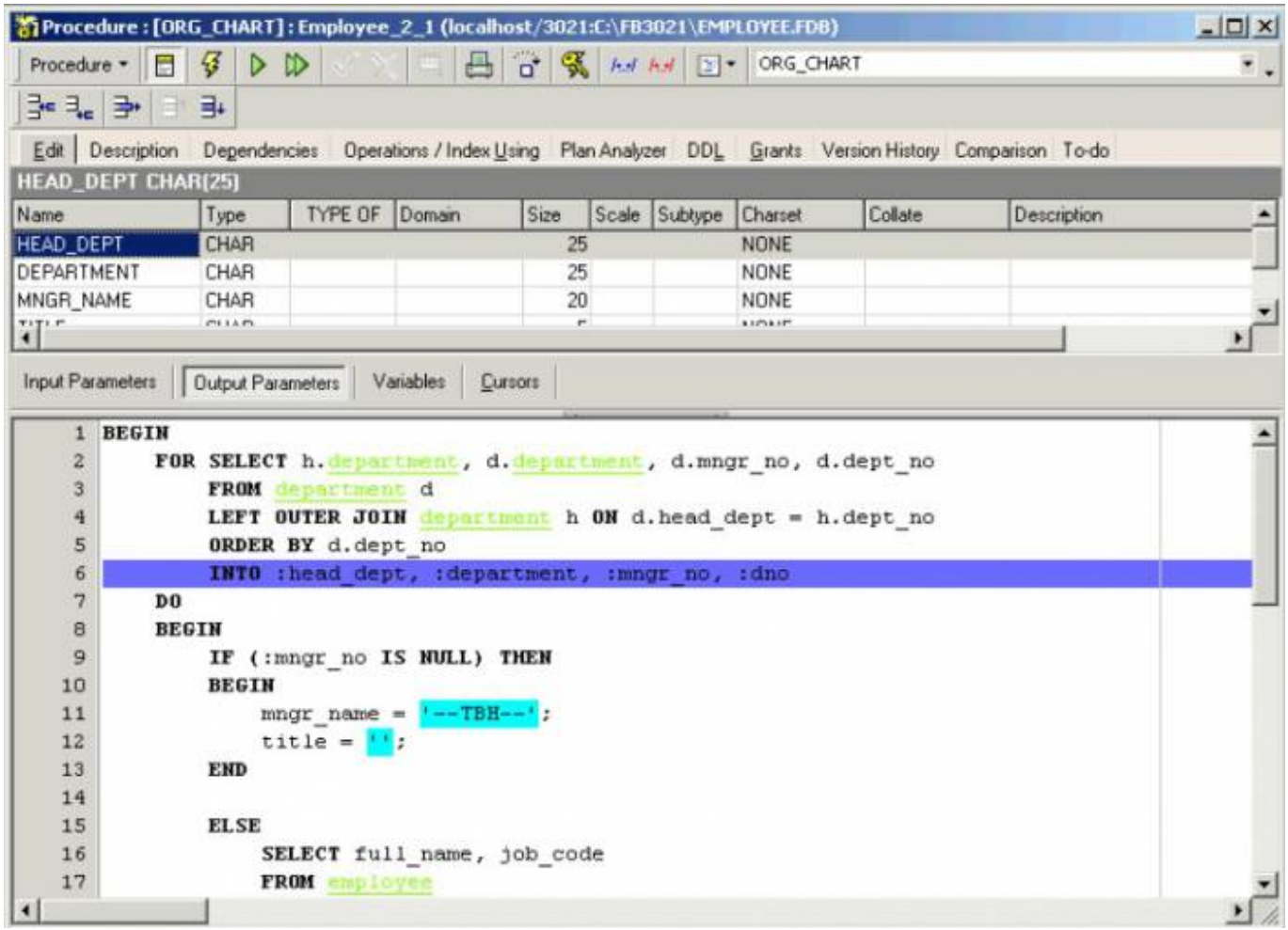
Stored Procedure

A stored procedure is a series of commands (also known as routines) stored as a self-contained program in the database as part of the database's metadata, and can be called by multiple client applications. They are pre-compiled, so they don't need to be sent over the network and parsed every time, they are just executed. They can be started by the EXECUTE PROCEDURE command with specification of the procedure name and a list of parameters. Procedures can take parameters and - like SELECTs - give back their data in the form of a table.

It is similar to a trigger, but is not automatically executed or bound to a specific table.



It is written in Firebird/InterBase® procedure and trigger language, also known as 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.
- Extensions to SQL, including assignment statements, control-flow statements, context variables (for triggers), event-posting statements, exceptions, and error-handling statements.

A Summary of PSQL commands can be found in the Stored procedure and trigger language chapter. Program execution occurs on the server.

Currently the maximum size of a stored procedure or trigger in Firebird and InterBase® is 48 KB of

BLR (the size of the byte code language compiled from stored procedure or trigger language and not the source code itself, which may include comments). However, as this comprises well over 1,000 lines of code, it is wiser to split any procedures of this size into smaller ones anyway, as this will improve not just the readibility and ease of maintenance but also, more often than not, the efficiency.

Each stored procedure is a stand-alone module of code that can be executed interactively or as part of a SELECT statement, from another stored procedure or from another application environment.

They can be invoked directly from applications, or can be substituted for a table or view in a SELECT statement; they can receive input parameters and return values to applications.

With the client/server database concept, it is important that the database is not just used to store data, but is actively involved in the data query and data manipulation processes. As the database must also be able to guarantee data integrity, it is important that the database can also handle more complex operations than just simple comparisons. Firebird/InterBase® uses stored procedures as the programming environment for integrating active processes in the database.

The stored procedure language is a language created to run in a database. For this reason its range is limited to database operations and necessary functions.

Stored procedures provide SQL enhancements that support variables, comments, declarative statements, conditional testing and looping as programming elements. They have full access to SQL DML statements allowing a multitude of command types; they cannot however execute DDL statements, i.e. a stored procedure cannot create a table.

Stored procedures offer the following advantages when implementing applications:

1. Reduction of network traffic by off-loading application processes from the client to the server. This is particularly important for remote users using slower modem connections. And for this reason of course, they are fast.
2. Splitting up of complex tasks into smaller and more logical modules. Stored procedures can be invoked by each other. Stored procedures allow a library of standardized database routines to be constructed, that can be called in different ways.
3. They're reusable. Rather than recreate a statement on the client each time it's needed, it's better to store it in the database. They can be shared by numerous applications using a single database. Alterations to the underlying data definitions only need to be implemented in the stored procedure and not in the individual applications themselves. Readability is enhanced, and redundancy, maintenance, and documentation are greatly reduced.
4. Full access to SQL and the database's metadata. This allows certain environments to perform extended operations on the database that might not be possible from another application language. The language even offers functions that are not available in SQL, e.g. IF…WHEN…ELSE, DECLARE VARIABLE, SUSPEND, etc.
5. Enhanced security: if database operations such as INSERT, ALTER or DROP can only be performed on a table by stored procedures, the user has no privileges to access the table directly. The only right the user has is to execute the stored procedure.
6. As stored procedures are part of Firebird or InterBase®, it is irrelevant which front end is subsequently used, be it Delphi, PHP or other.

There are no disadvantages to using stored procedures. There are however, two limitations. Firstly, any variable information must be able to be passed to the stored procedure as parameters or the information must be placed in a table that the stored procedure can access. Secondly, the procedure

and trigger language may be too limited for complex calculations. Stored procedures should be used under the following circumstances:

1. If an operation can be carried out completely on the server with no necessity to obtain information from the user while the operation is in process. When invoking a stored procedure these input parameters can be incorporated in the stored procedure.
2. If an operation requires a large quantity of data to be processed, whose transfer across the network to the client application would cost an enormous amount of time.
3. If the operation must be performed periodically or frequently.
4. If the operation is performed in the same manner by a number of different processes, or processes within the application, or by different applications.

The stored procedure must contain all statements necessary for the database connection, creation or alteration of the stored procedure, and finally the disconnection from the database.

All SQL scripts can be incorporated into a stored procedure and up to ten SQLs incorporated in one single procedure, as well as the additional functions already mentioned, making stored procedures considerably quicker and more flexible than SQL.

Stored procedures can often be used as an alternative to views (being more flexible and offering more control) as the ORDER BY instruction cannot be used in a view (the data sets are displayed as determined by the optimizer, which is not always intelligent!). In such a case, a stored procedure should be used.

Stored procedures are almost identical to triggers, the only exception being the way they are called: triggers are called automatically when a change to a row in a table occurs. Most of what is said about stored procedures applies to triggers as well.

back to top of page

Executing stored procedures

Firebird/InterBase® stored procedures are divided into two groups with respect to how they are called. Select procedures return result values through output parameters, because they can be used in place of a table name in an SQL SELECT statement. A select procedure must be defined to return one or more values, or an error will result. Executable procedures can be called by an application directly, using the EXECUTE PROCEDURE statement. An executable procedure need not return values to the calling program. To be able to call a procedure, the user must have EXECUTE rights (see Grant Manager). In IBExpert the template already includes this statement for you (refer to the illustration in the SET TERM chapter below).

The simplest way to execute a stored procedure is to use the EXECUTE PROCEDURE statement. This statement can be used in one of the following ways:

- From within another stored procedure.
- From within a trigger.
- From an application.

When a procedure is executed from within an Firebird/InterBase® application, such as another procedure or a trigger, it has the following syntax:

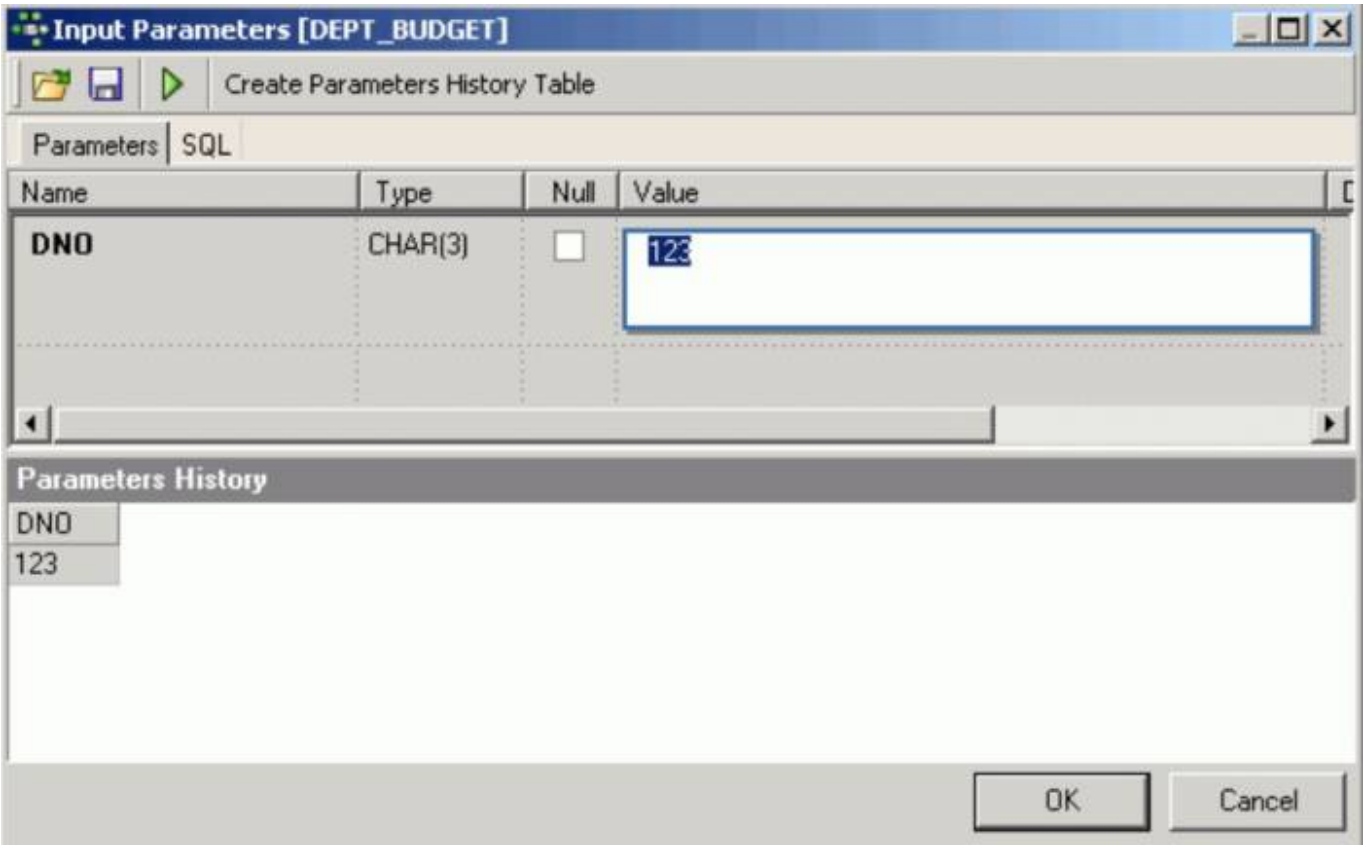
```
EXECUTE PROCEDURE
<procedure_name>
<input_parameter_list>
RETURNING_VALUES
<parameter_list>
```

If the procedure requires input variables, or if it is to return output variables, the relevant parameters need to be specified. In each case, <parameter_list> is a list of parameters, separated by commas (see stored procedure parameters for further information).

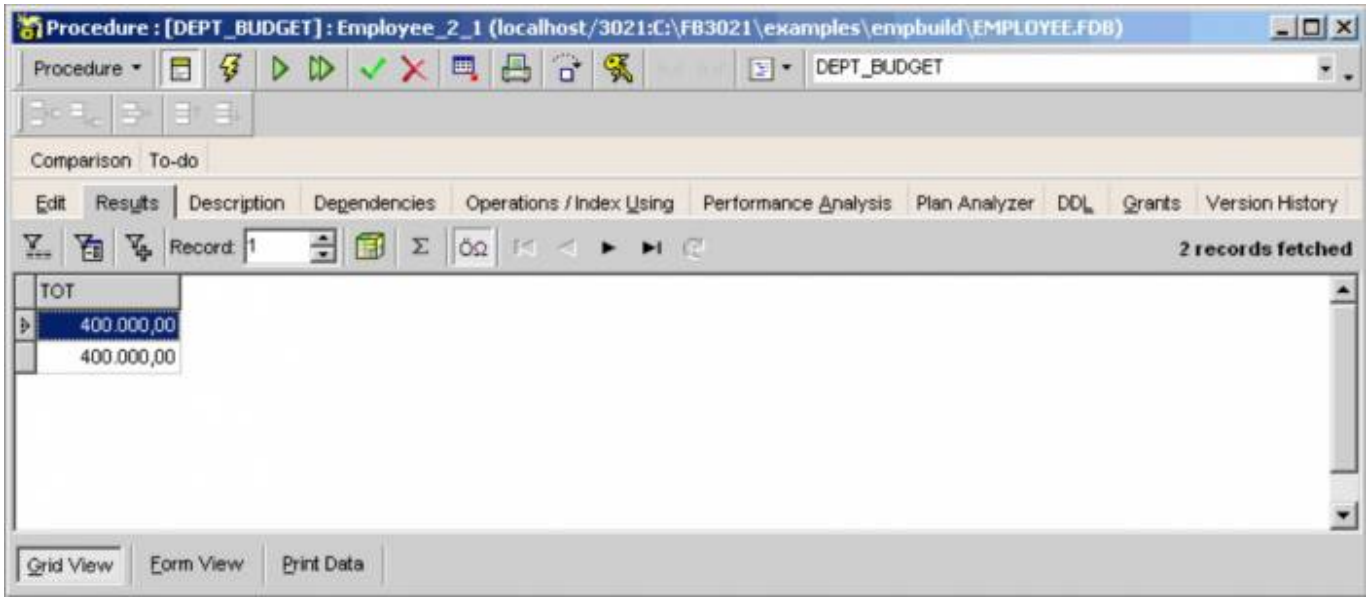
Each time a stored procedure calls another procedure, the call is said to be nested because it occurs in the context of a previous and still active call to the first procedure.

Stored procedures can be nested up to 1,000 levels deep. This limitation helps to prevent infinite loops that can occur when a recursive procedure provides no absolute terminating condition. Nested procedure calls may be restricted to fewer than 1,000 levels by memory and stack limitations of the server.

When using IBExpert's Procedure Editor to execute a procedure, IBExpert tells you whether input parameters need to be entered:



before displaying the return values (= output or results) on the *Results* page:



back to top of page

Select procedures

It is possible to use a stored procedure in place of the table reference in a SELECT statement. This type of procedure is known as a select procedure.

When a stored procedure is used in place of a table, the procedure should return multiple columns or rows, i.e. it assigns values to output parameters and uses SUSPEND to return these values. This allows the SELECT statement to filter the results further by different criteria.

SUSPEND is used to suspend execution of the procedure and return the contents of the output variables back to the calling statement. If the stored procedure returns multiple rows, the SUSPEND statement needs to be used inside a FOR SELECT ... DO loop to return the rows one at a time.

Non-select procedures

Execute or non-select procedures perform an action and do not return any results.

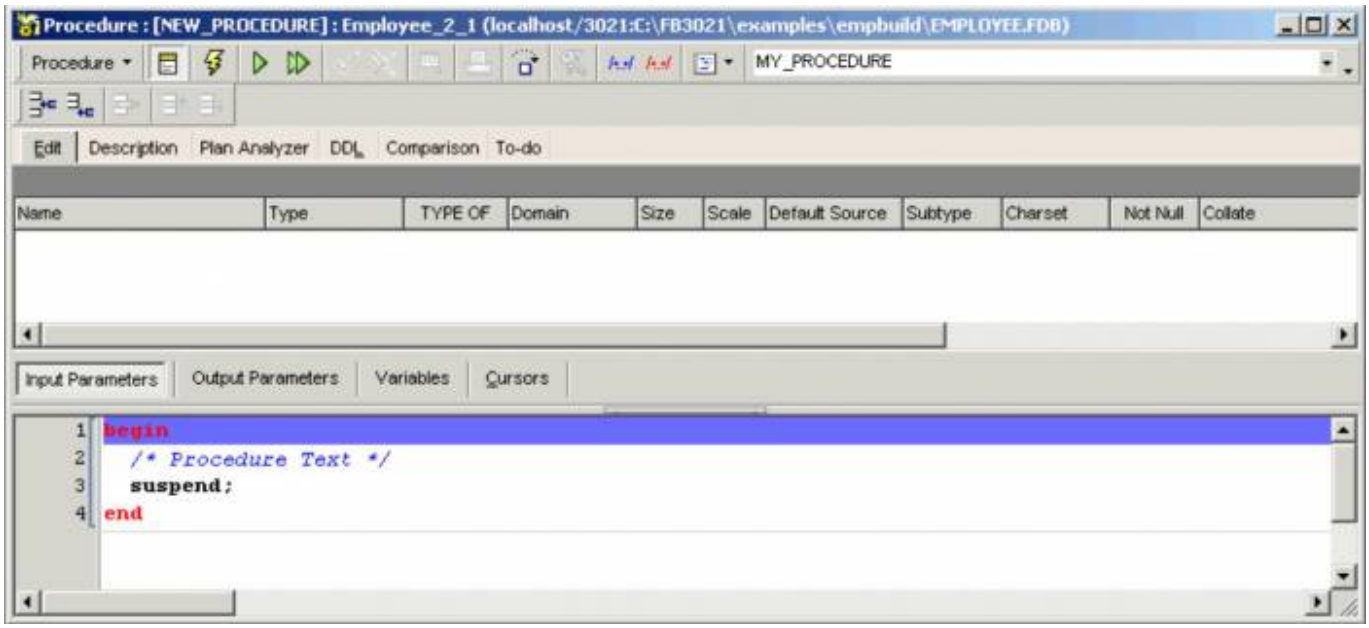
back to top of page

New procedure

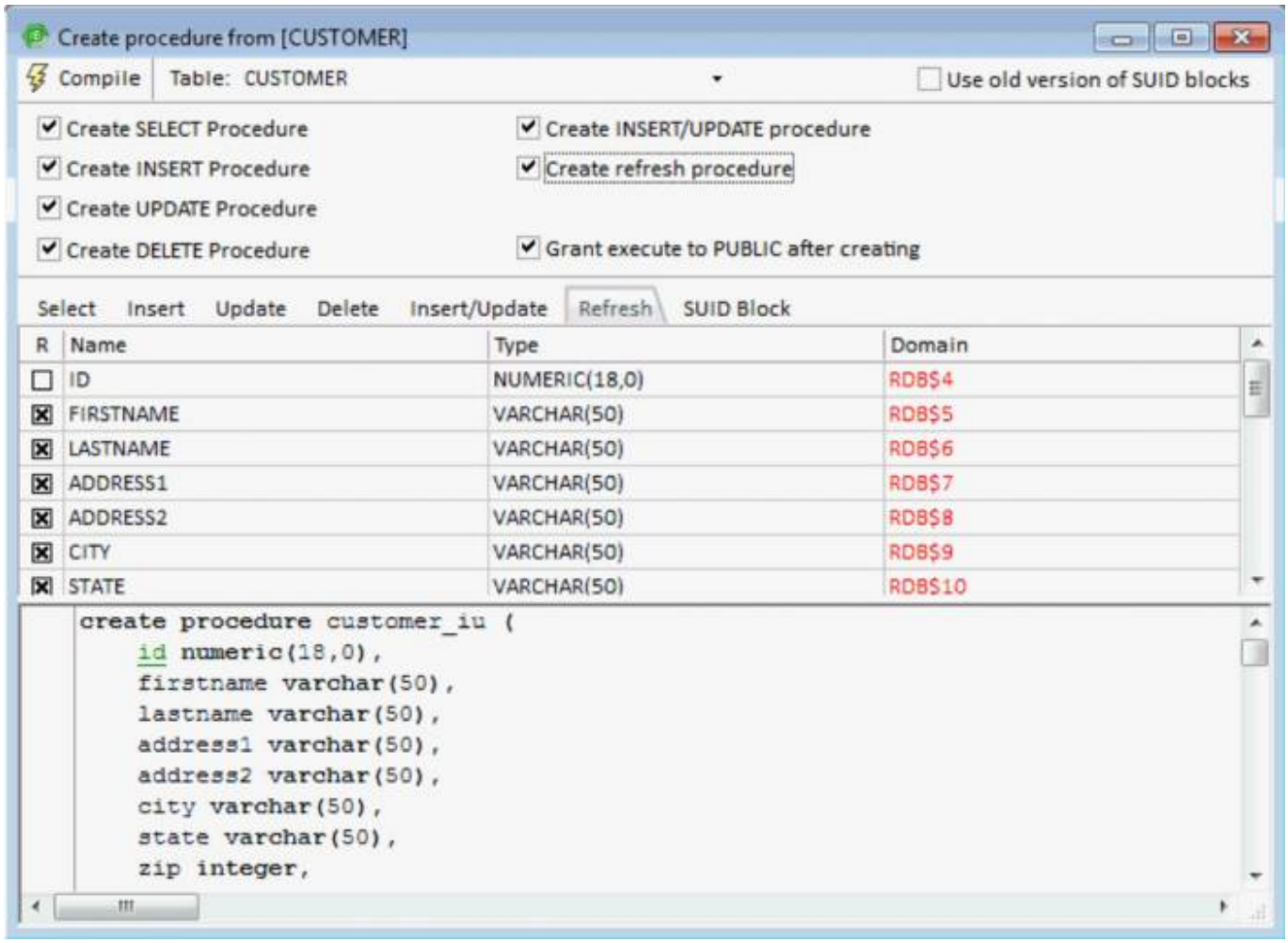
There are numerous ways to approach creating a new stored procedure:

1. Using the IBExpert menu item *Database* / *New Procedure* or using the *New Procedure* icon on the New Database Object toolbar to start the Procedure Editor.

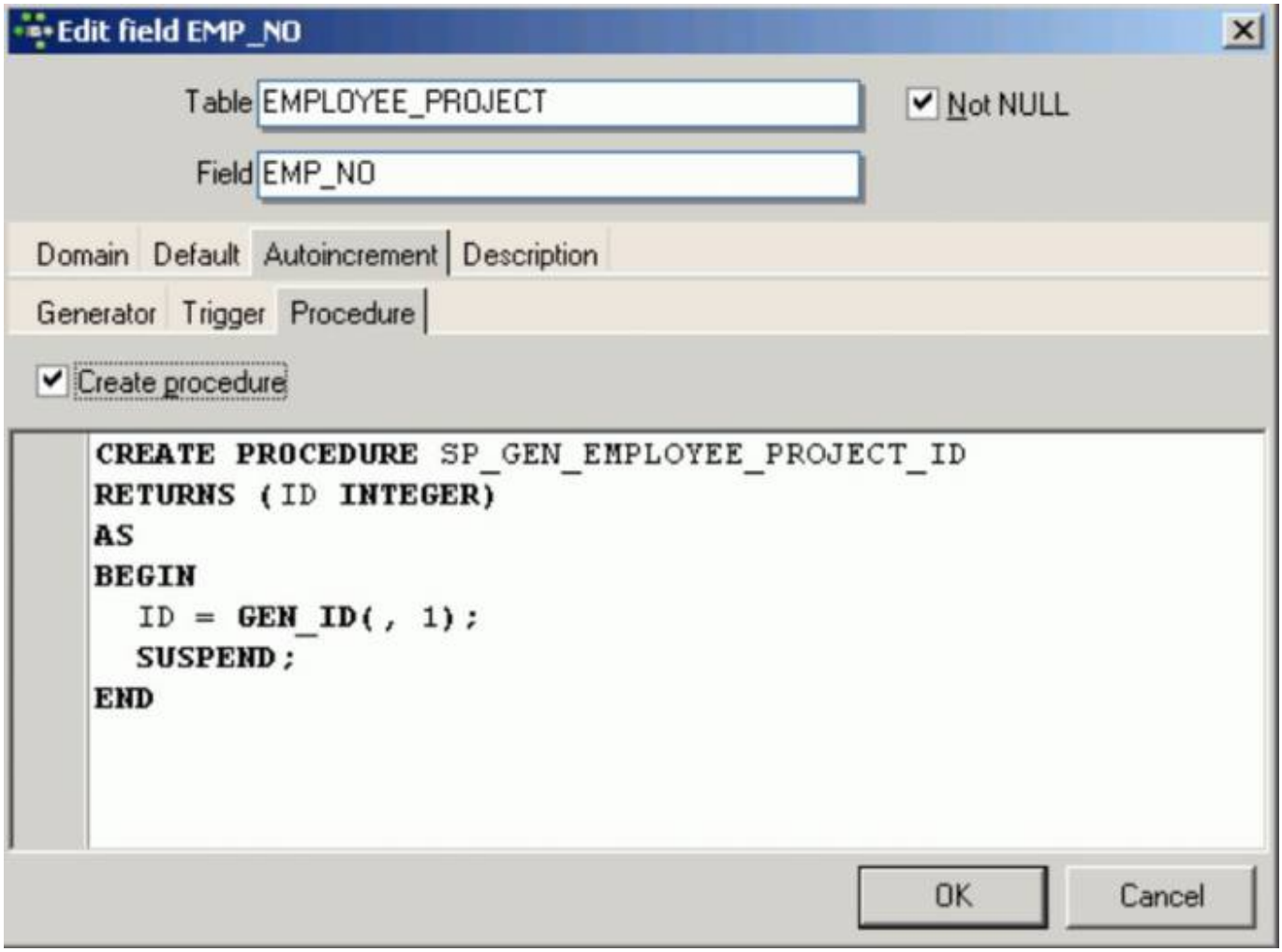
2. From the DB Explorer by right-clicking on the highlighted procedure branch of the relevant connected database (or key combination [Ctrl + N]) which also starts the Procedure Editor.



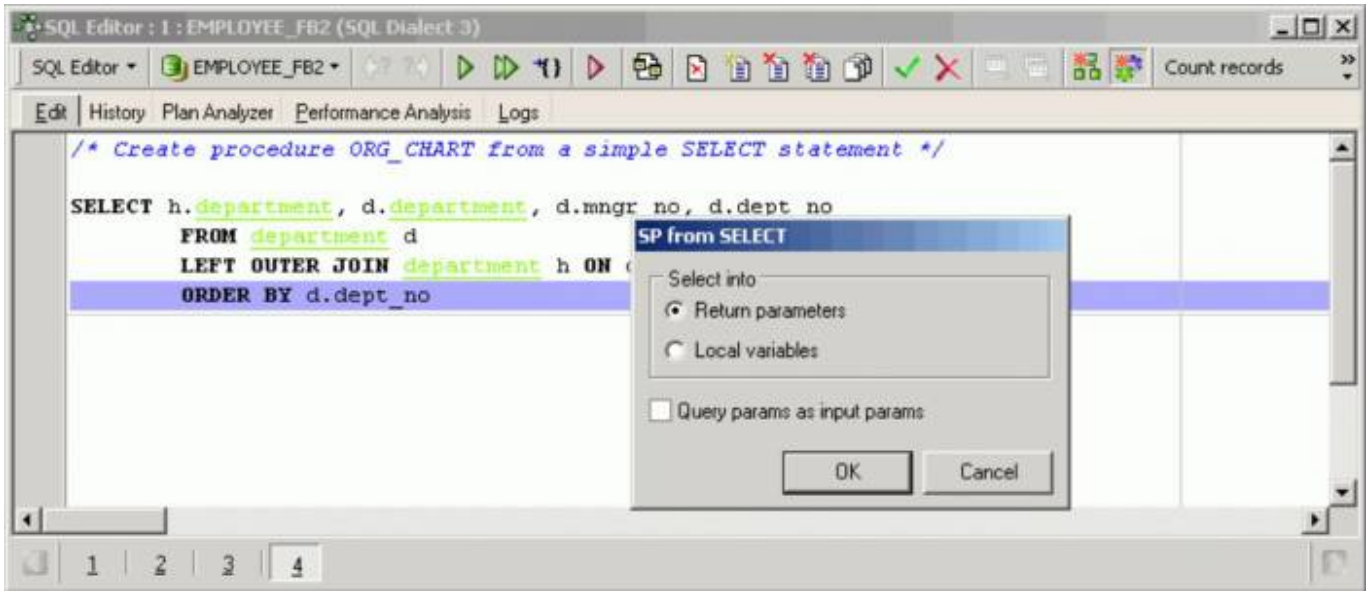
3. A stored procedure can also be created directly from a selected table in the DB Explorer, using the right-click pop-up menu item Create SIUD procedures.



4. Or created directly from the Field Editor.



5. Or created in the IBExpert SQL Editor, and then saved as a stored procedure. When an SQL script has been successfully committed, and the results are as wished, the script can be integrated into a stored procedure using the *Stored Procedure* button. The stored procedure script appears, and simply needs to be named and completed.



The CREATE PROCEDURE statement has the following syntax:

```
CREATE PROCEDURE <Procedure_Name>
```

```
<Input_Parameter_List>
RETURNS
<Return_Parameter_List>
AS
<Local_Variable_Declarations>
BEGIN
<Procedure_Body>
END
```

The CREATE and RETURNS statements (if there is a return statement) comprise the stored procedure's header. Everything following the AS keyword is the procedure's body. There can also be statements between the AS and BEGIN keywords that are also considered part of the body. These statements declare local variables for the stored procedure, and are detailed in the chapter, Stored Procedure Language.

Please refer to the Firebird Release Notes relevant for the Firebird version you are using:

- Firebird 4.0 Release Notes (12 May 2020 - Document v.0400-33 - for Firebird 4.0 Beta 2 Release) - Chapter 11 - Procedural SQL (PSQL)
- Firebird 3.0.5 Release Notes (4 January 2020 - Document v.0305-03 - for Firebird 3.0.5 Release) - Chapter 10 - Procedural SQL (PSQL)
- Firebird 2.5.3 Release Notes - Procedural SQL (PSQL)
- Firebird 2.1.6 Release Notes - Procedural SQL (PSQL)
- Firebird 2.0.7 Release Notes - Stored Procedure Language (PSQL)

Further information explaining the necessary components can be found under Procedure Editor, started using the first two menu options (i.e. IBExpert Database menu and DB Explorer right mouse button menu).

The Procedure Editor has its own toolbar (see Procedure Editor toolbar). To the right of the toolbar, the new procedure name can be specified. The procedure name follows the naming convention for any Firebird/InterBase® object and must be unique. The *Lazy Mode* icon can be used to switch the lazy mode on and off as wished:

The *New Procedure* Editor has five pages:

1. Edit
2. Description
3. Plan Analyzer
4. DDL
5. Comparison

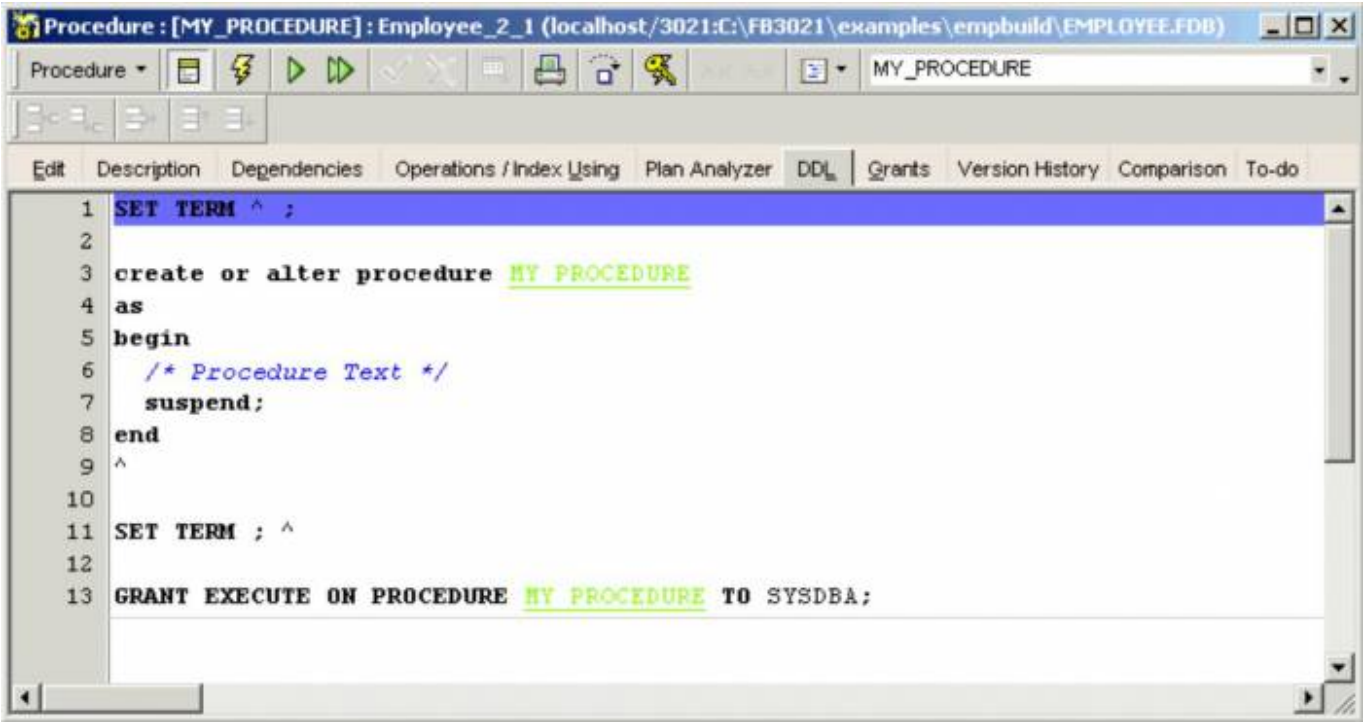
described under Procedure Editor. A new procedure is created on the Procedure Editor / Edit page.

back to top of page

SET TERM

Every command in a Firebird/InterBase® script must be terminated by a semicolon, including the procedure itself. To distinguish the semicolons in the procedure from the terminating semicolon, another temporary terminator is needed for the end of the procedure. SET TERM replaces the terminator semicolon with a user-defined character. After the procedure itself is terminated by this new terminator, the terminator symbol is set back to the semicolon.

When using the IBExpert Procedure Editor, the procedure templates already include this code, so you don't have to worry about it. If you open the New Procedure Editor and take a peek at the DDL page, you will see how much code has already be generated by IBExpert, although you haven't even started to define your procedure:



Even SUSPEND and the GRANT EXECUTE statement have been included.

For those who wish to view the syntax and an example of how to use this when coding by hand, please refer to SET TERM terminator.

back to top of page

Stored procedure parameters (input and output/returns)

Input parameters are a list of variables (=values) that are passed into the procedure from the client application. These variables can be used within the procedure to modify its behavior.

The return parameter (or output parameter) list represents values that the procedure can pass back to the client application, such as the result of a calculation. Each list is in the following format:

```
ParameterName1 ParameterType,
```

```
ParameterName2 ParameterType,
...
ParameterNameN ParameterType
```

ParameterType is any valid Firebird/InterBase® data type except blob, domain and arrays of data types. It is even possible to copy/paste parameter values to/from the *Input parameters* form.

Local variables / DECLARE VARIABLE statement

Local variables can be defined within the procedure body. Local variables of any Firebird/InterBase® type can be declared within a stored procedure. As with any other structured programming environment, these variables only exist while the procedure is running, and their scope is local to the procedure. They are invisible outside the procedure and are destroyed when the procedure finishes. There are no global variables available with stored procedures and triggers. If values need to be shared by two or more procedures, they should either be passed as parameters or stored in a table.

Local variables are declared immediately after the AS clause, using the DECLARE VARIABLE statement. For example the variable ANY_SALES is declared in the EMPLOYEE database's DELETE_EMPLOYEE procedure:

```
DECLARE VARIABLE ANY_SALES INTEGER;
```

Each variable must be declared in its own DECLARE VARIABLE statement, as each statement can declare only one variable.

back to top of page

Procedure body

The procedure body consists of a compound statement, which can be any number of Firebird/InterBase® procedure and trigger language statements. The procedure body starts with a BEGIN statement, followed by any local variable declarations and their data types, and ends with an END statement.

BEGIN and END must also be used to surround any block of statements that logically belong together, such as the statements within a loop.

BEGIN and END do not need terminating characters, except for the final END within the procedure.

Comment Procedure Body/Uncomment Procedure Body

In certain situations it may be necessary to disable certain commands or parts of SQL text. This can be easily done 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 the *Uncomment Procedure* icon (above), the right mouse button menu item *Uncomment Selected*, or [Ctrl+ Alt + ,]. This feature also comments/uncomments DECLARE VARIABLE/CURSOR sections to remove dependencies from objects in DECLARE CURSOR selects.

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.

back to top of page

Lazy mode

```
tutorial available
```

Using lazy mode, the programmer does not have to worry about which input and output parameters need to be considered. The user can switch between lazy mode and classic mode using the

icon in the Procedure Editor and Trigger Editor.

Lazy mode can be deactivated altogether by changing the default Editor Mode, found in the IBExpert Options menu items, Object Editor Options... / Procedures Editor and Object Editor Options... / Triggers Editor, from *Lazy* to Standard.

It is possible to select domains as a data type for input/output parameters and variables. In this case IBExpert copies information from the domain definition to the native data type of the parameter/variable. You can even drag 'n' drop a domain from the Database Explorer.

Parameters grid in Lazy mode: for domain-based parameters it is possible (since IBExpert version 2021.02.09) to copy the domain comment to parameter comment. To enable this put following the function call into the “After IBExpert starts” event block:

```
  ibec_SetGlobalVar('IBE$PRMFM_COPY_DOMAIN_COMMENT', TRUE);
```

To enable for a specified database only:

```
  ibec_SetGlobalVar('IBE$PRMFM_COPY_DOMAIN_COMMENT.123', TRUE);
```

where 123 is an unique identifier of the registered database.

The SEGMENT SIZE can also be specified for blob parameters and variables whilst working in lazy mode.

back to top of page

Stored Procedure Editor

The Procedure Editor can be started using the Database / New Procedure menu item; from the DB Explorer, using the right mouse-click menu or double-clicking on an existing procedure.

Please refer to New Procedure when creating a stored procedure for the first time.

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

1. Edit
2. Result
3. Description
4. Dependencies
5. Operations/Index Using
6. Performance Analysis
7. Plan Analyzer
8. DDL
9. Grants
10. Version History
11. Comparison
12. To-Do

At the time of writing, the maximum size of a stored procedure is limited in Firebird and InterBase® to 64K.

Edit

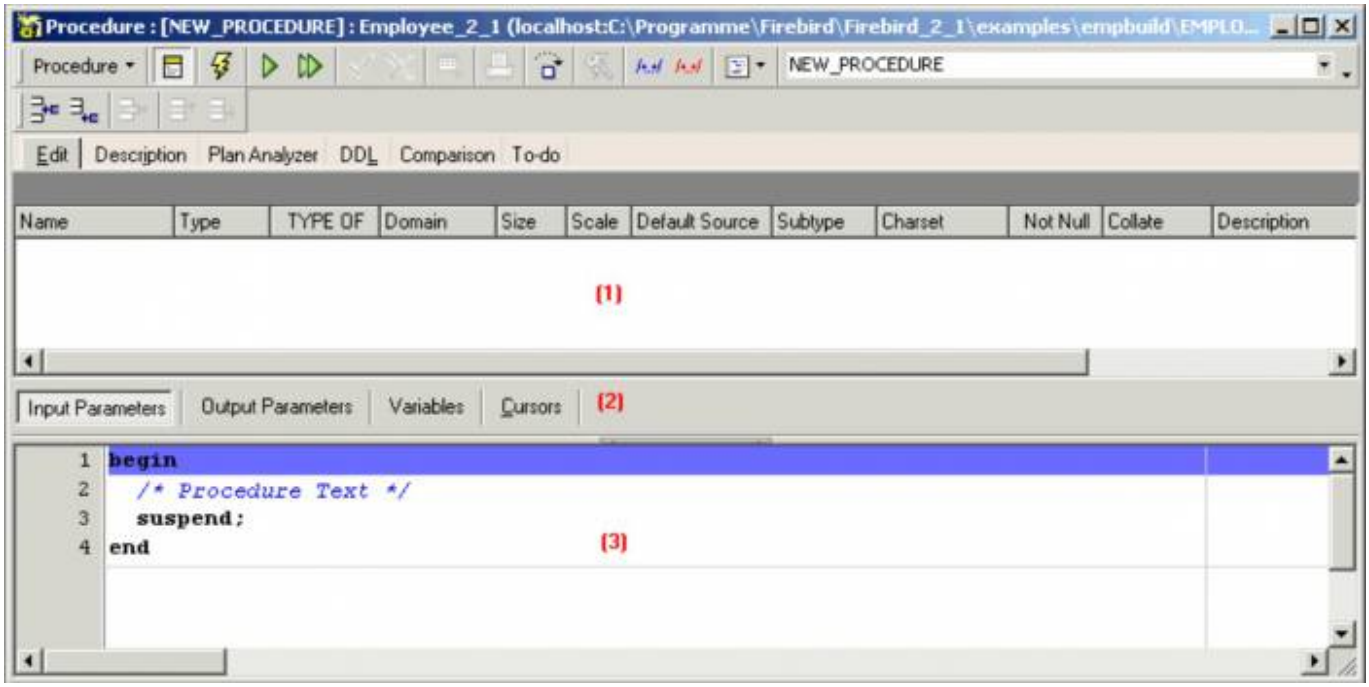
The CREATE PROCEDURE statement has the following syntax:

```
CREATE PROCEDURE <Procedure_Name>
<Input_Parameter_List>
RETURNS
<Return_Parameter_List>
AS
<Local_Variable_Declarations>
BEGIN
<Procedure_Body>
END
```

A stored procedure comprises the following components:

1. input parameters
2. output parameters (returns)
3. variables
4. procedure body
5. comments (optional)

If the lazy mode is switched off, the *Edit* page offers a single SQL input area, with the procedure syntax already displayed. If the lazy mode is switched on, the *Edit* page consists of three areas:



(1) The field grid, where new parameters can be specified.

(2) In the middle are three buttons specifying the parameter type, i.e. input parameters, output parameters and variables. It is possible to drag 'n' drop parameters/variables from the field grid onto the corresponding button to move them. For example, click the *Output Parameters* button, drag a named variable from the field grid onto the *Variable* button. Click the Variable button to view the new variable in the field grid.

(3) Below this is the SQL panel for direct code input. Again the procedure syntax is already displayed to help the user.

As with all Editors, it is possible to format the code text, such as:

- *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]

Please refer to Localizing Form for further keyboard shortcuts.

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.

For those who do not wish to use the basic syntax template, or wish to add certain statements themselves to create their own standard, this can be done using the IBExpert menu item Options / General Templates, and clicking on either the Standard Mode or Lazy Mode under New Procedure.

You can disable the display of procedure code size in the *Edit* tab title in Options / Object Editor Options / Procedure Editor, by deactivating the *Always display procedure DDL* size option.

As with all SQL input windows, the SQL Editor Menu can be called using the right mouse button.

The basic parameters of the stored procedure are set here as SQL text for creating the procedure. A parameter can have any Firebird/InterBase® data type except blob or array. The input parameters are set in brackets after the procedure name, the output parameters are set in brackets after the RETURNS statement, and the procedure body written in InterBase® procedure and trigger language, bracketed by BEGIN and END statements.

New parameters can be quickly and easily specified, by clicking the respective button (i.e. *input*, *output* or *variables*), and inserting field information using the respective icon or right-click menu, in the same manner as creating a new table.

Local variables of any Firebird/InterBase® type can be declared within a stored procedure (please refer to local variables), after the AS keyword and before the BEGIN (which marks the begin of the procedure body).

Alternatively, the required information can be entered directly in the editor's input panel and field names can be simply dragged from the DB Explorer or SQL Assistant into the procedure script. The code insight can be used to save time wasted searching for correct names, and to prevent any possible spelling errors. A right mouse-click within this area produces the SQL Editor menu.

The input parameters are set with their types in brackets after the procedure name. By checking the Code Parameter option under Options / Editor Options / Code Insight, a list of the necessary parameters automatically appears. Output parameters are specified in the same way after RETURNS. The operations to be performed by the procedure are described after the BEGIN statement. Please refer to Stored Procedure and Trigger Language for further details.

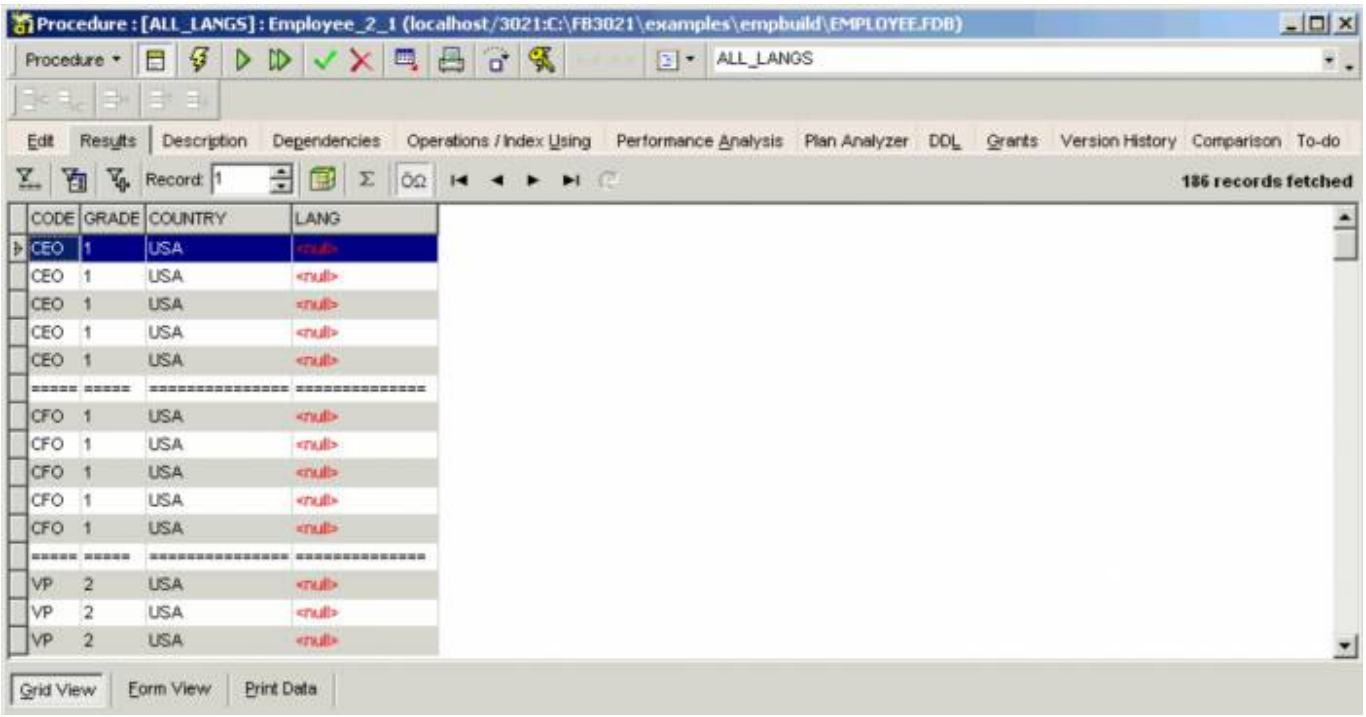
After inputting the required information, the stored procedure can be executed using [F9] or the relevant icon. The statement window appears, where the resulting SQL statement can be viewed before committing. If necessary the code can subsequently be debugged using the debugging icon or [Shift + Ctrl + D]. (Please refer to Debug Procedure for more details.)

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

back to top of page

Results

The *Results* page appears following execution of the procedure, and displays all data sets fetched:



Please refer to SQL Editor / Results for details.

Description

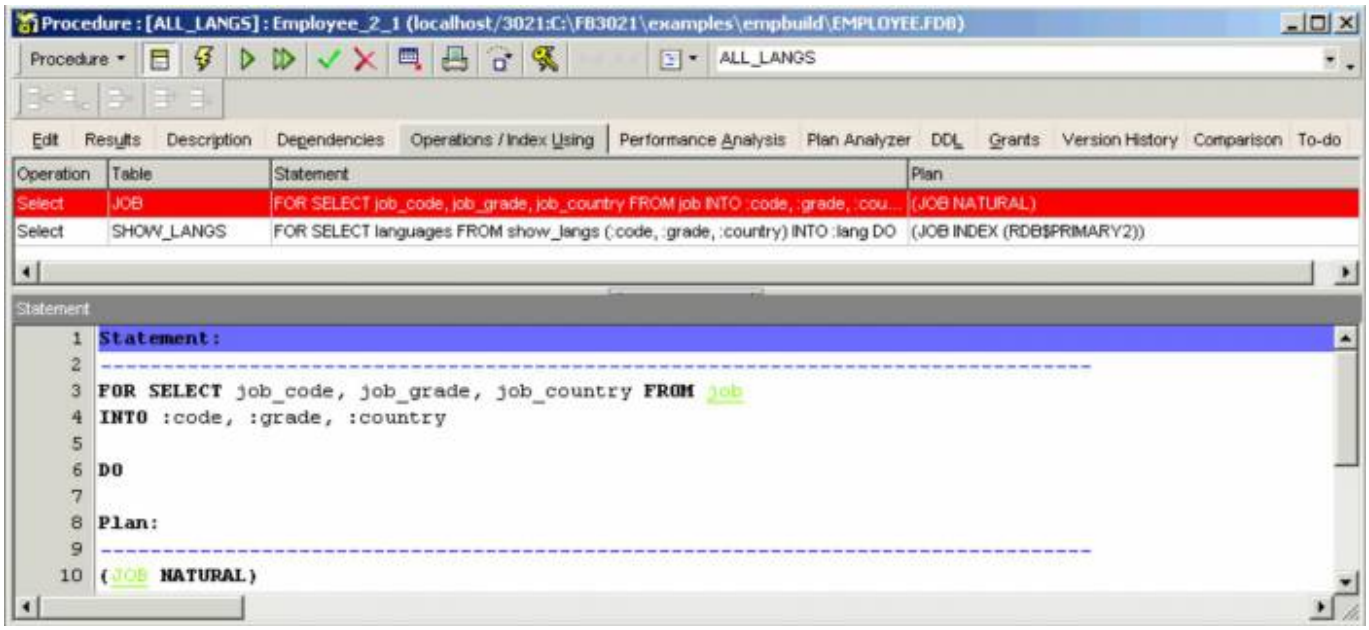
Please refer to Table Editor / Description.

Dependencies

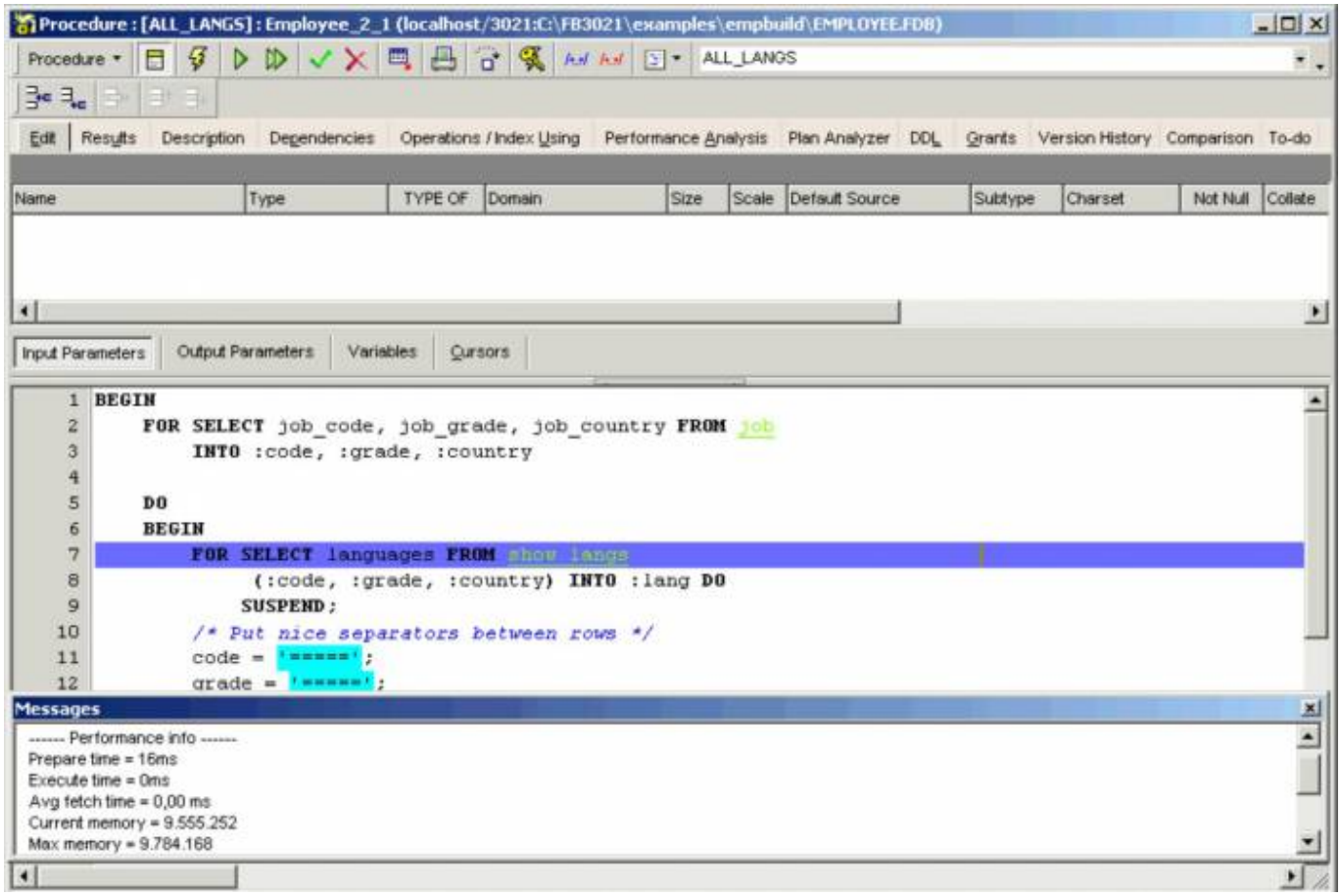
See Table Editor / Dependencies.

Operations/Index Using

This page dissects the procedure into single operations, and examines them to see whether they use a plan (i.e. index) or not. The ALL_LANGS procedure in the sample EMPLOYEE database displays red-marked entries, which indicates a plan@NATURAL (i.e. no indices are used). When an operation is selected, the statement for this operation is displayed in the lower window:



By double-clicking on a selected operation, the SQL panel appears, highlighting the SQL statements for this operation, enabling further analysis and amendments. For example, should perhaps the ORDER BY be altered, or perhaps a different JOIN?

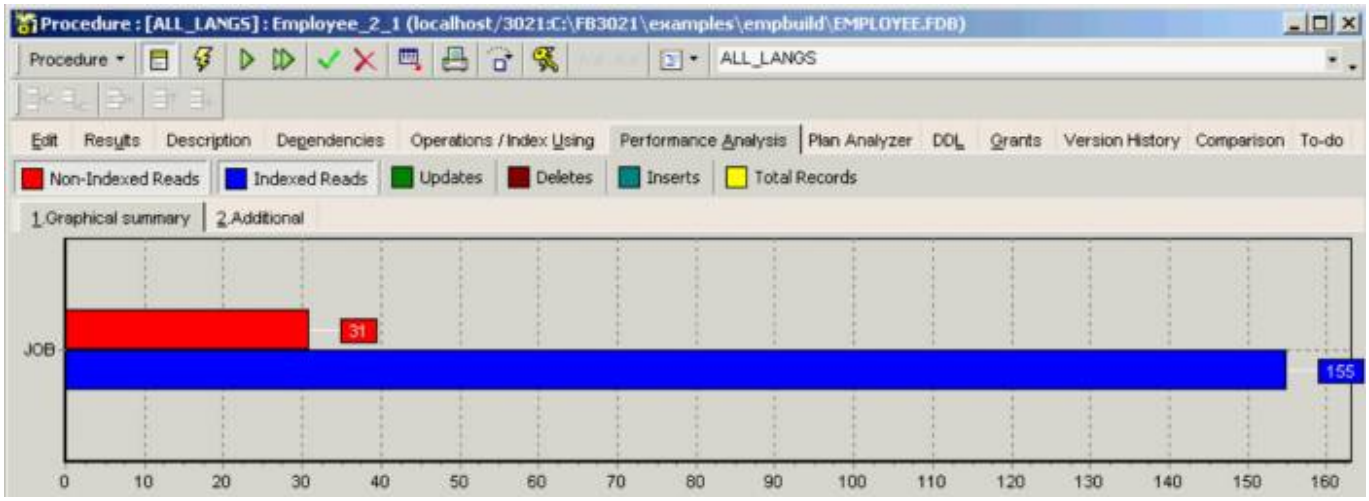


Input and output parameters and variable fields can be displayed by clicking on the buttons in the center of the editor. Alterations may be made directly in the SQL window and subsequently executed and committed.

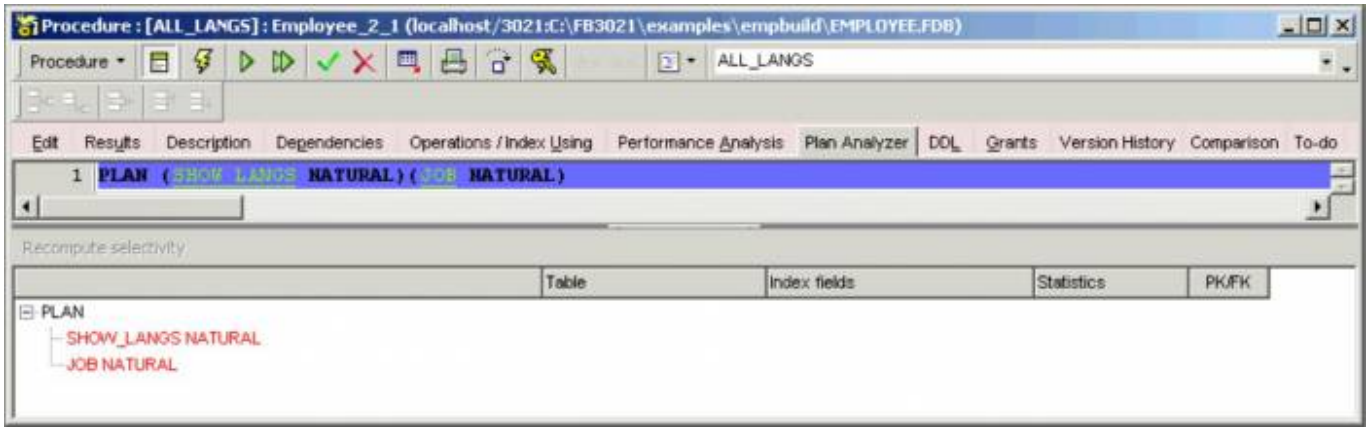
back to top of page

Performance Analysis

This page only appears once a procedure has been executed. Please refer to SQL Editor / Performance Analysis for details.



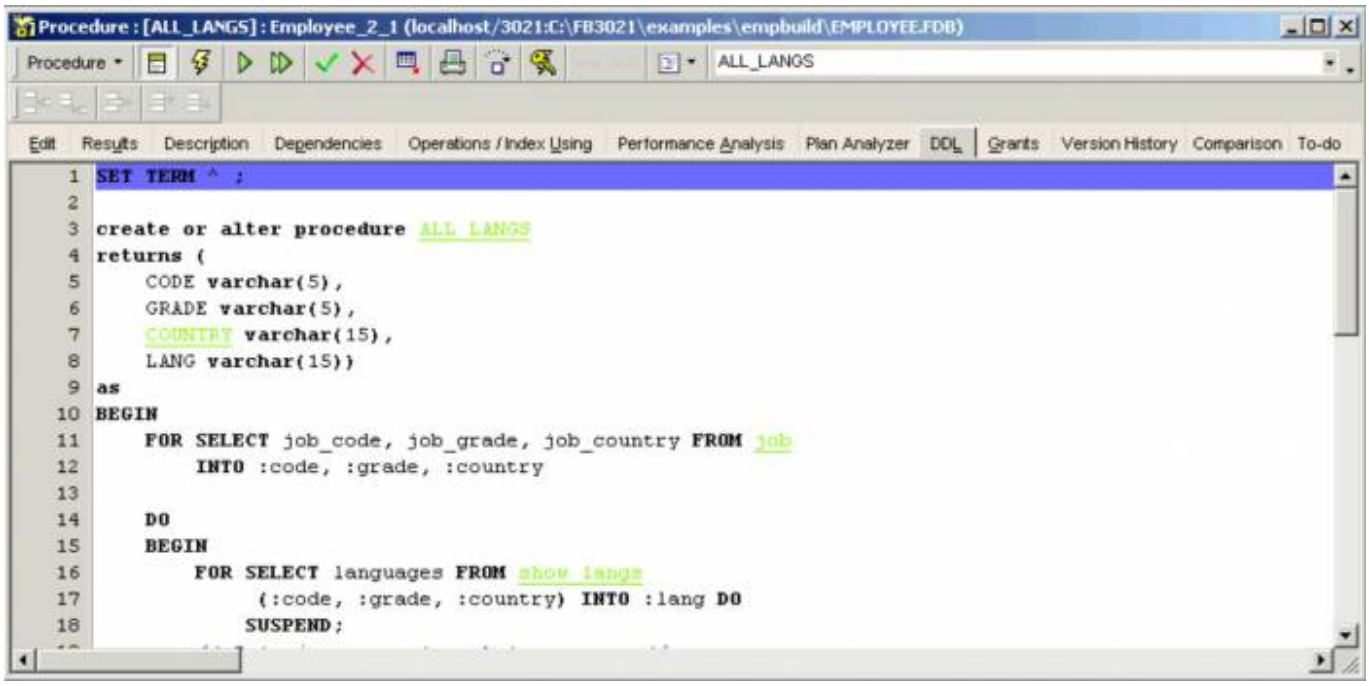
Plan Analyzer



Please refer to SQL Editor / Plan Analyzer.

DDL

The DDL page includes the CREATE PROCEDURE statement, stored procedure and parameter descriptions and GRANT statements.



Grants

Please refer to Table Editor / Grants and autogrant privileges.

Version History

Please refer to View / Version History.

Comparison

Please refer to Table Editor / Comparison.

To-Do

Please refer to Table Editor / To-Do.

back to top of page

Debug procedure, trigger, function (IBExpert Debugger)

A stored procedure, trigger or function can be simply and quickly debugged in IBExpert (this feature is unfortunately not included in the free IBExpert Personal Edition) using the *Debug Procedure* icon or [F8]:

IBExpert simulates running the procedure or trigger on the database server by interpreting the procedure and running the commands one at a time. It offers a number of useful functionalities, such as *breakpoints*, *step into*, *trace* or *run to cursor*, you can watch certain parameters, analyze the performance and indices used, and you can even change values on the fly. It even offers full UTF8/Unicode. If you have Delphi experience you will easily find your way around the Debugger as key strokes etc. are the same.

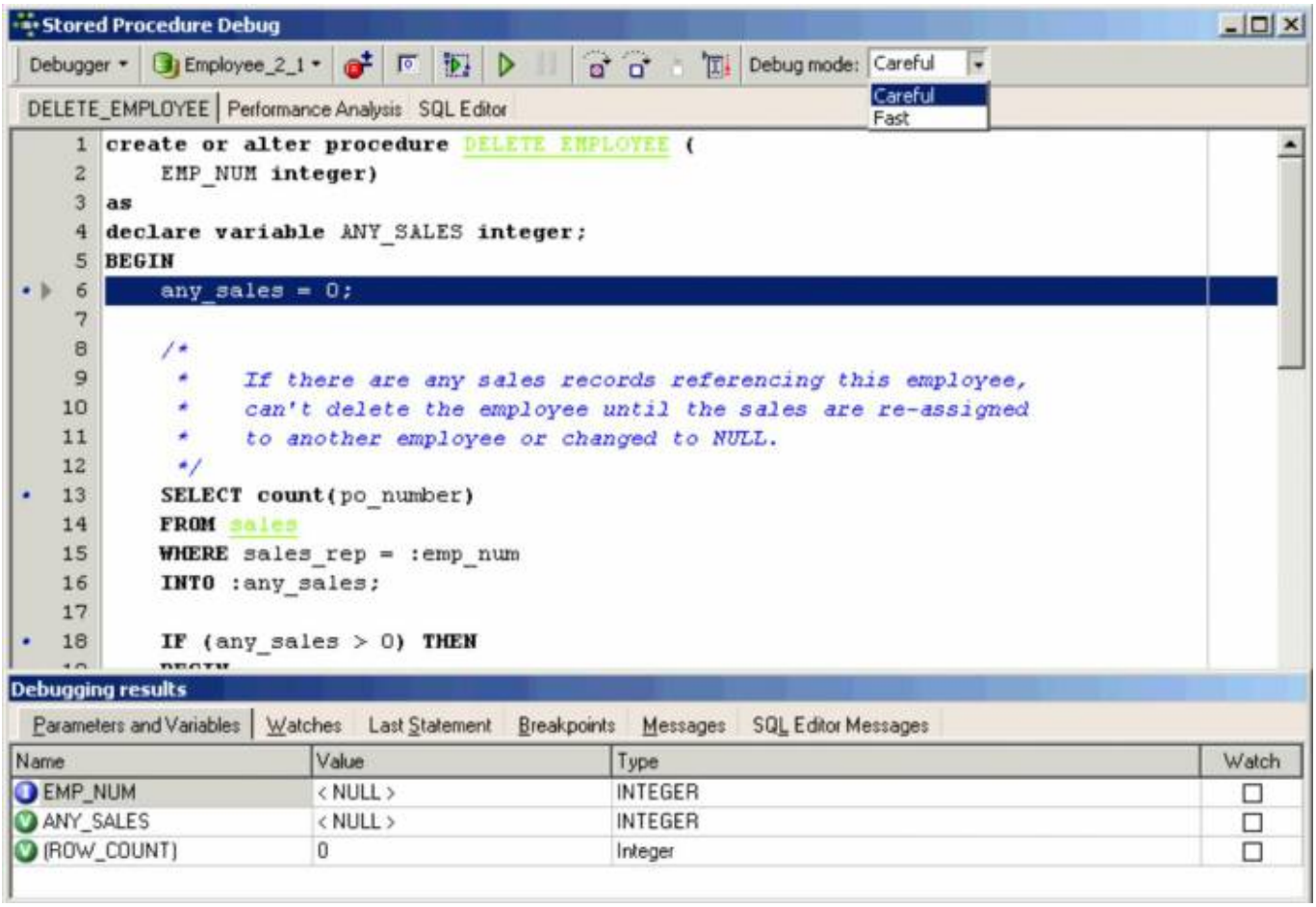
By the way, if you are experiencing problems with calling or opening a stored procedure from IBExpert, please refer to the IBExpert FAQ page. Here you will find a number of tips that generally solve the majority of problems.

A particularly interesting feature is the *Collect Statistics* mode described below. An example of this using the IBExpert Demo DB1 procedures INITALL and DELETEALL can be found here.

It is possible to debug procedures/triggers/functions that contain subroutines. To debug such PSQL objects IBExpert will create a temporary debug package with subroutines declared in the PSQL object source. Normally that package will be deleted when the debugger window is closed.

IBExpert also supports Firebird 3.0 scroll cursors in SP/Trigger editors and Debugger. Simply open the procedure or trigger in the Procedure Editor or Trigger Editor by double-clicking on the procedure/trigger name in the DB Explorer and click the *Debug* icon on the Procedure or Trigger Editor toolbar (or [Shift + Ctrl + D]) to start the *Debugger* window.

The Debug Procedure/Trigger Editor comprises 3 pages, the Debug page (described here), Performance Analysis and the SQL Editor.



The debugging options *Careful* and Fast: in the default debug mode, *Careful*, a corresponding SELECT

statement is composed and executed on the server side. The *Fast* mode executes certain statements, such as simple assignments and Boolean expressions of IF/WHILE statements, on the client side if possible. The Fast mode should be used for example, if you need to repeatedly execute a loop, which contains statements that can be calculated on the client side, as this will greatly reduce total execution time. Select the preferred option using the drop-down list in the top right-hand corner before starting the debug process.

Using the debugging option *Collect Statistics*, you can now run the debugger through procedures line by line and analyze the performance, allowing you to quickly and easily detect performance leaks in complex procedures. Please refer to the example illustrated in Debugger Collect Statistics mode.

The upper half of this dialog displays the SQL text. The object name (if applicable) is displayed in the Windows bar. The lower area displays a number of pages.

Parameters and Variables

The parameters are listed in a grid. The circular symbols to the left of the name indicate whether the parameters are input (I) or output (O). Variables logically have the key (V). Further information displayed here includes the parameter value, scope and data type. The *Watch* boxes can be checked, to specify which variables should be observed.

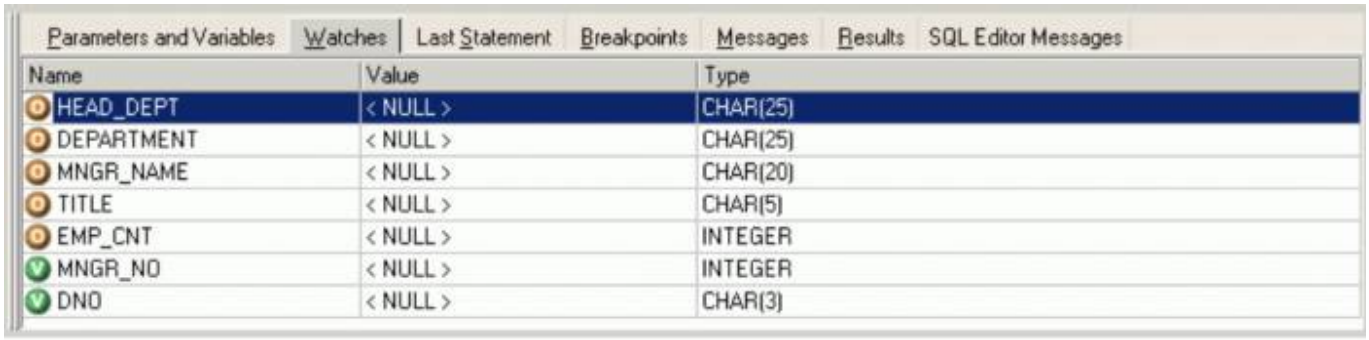
The variable contents can be viewed in the Value column or by directly by holding the mouse over the variable name in the code itself.

It is possible to initialize parameters/variables using values of any data grid. Just drag and drop a cell value from any data grid onto the corresponding node in the parameters/variables list to initialize the variable with the value of the data cell. It is also possible to initialize multiple variables/parameters by holding the [Ctrl] key when dropping. In this case IBExpert searches for the corresponding parameter/variable (by name) for each field in the data record, and if the parameter/variable is found it will be initialized with the value of the field with the same name.

Universal triggers which use the context variables INSERTING/UPDATING/DELETING can also be debugged here. The Debugger interprets these variables as regular input parameters with a BOOLEAN data type and they are FALSE by default.

back to top of page

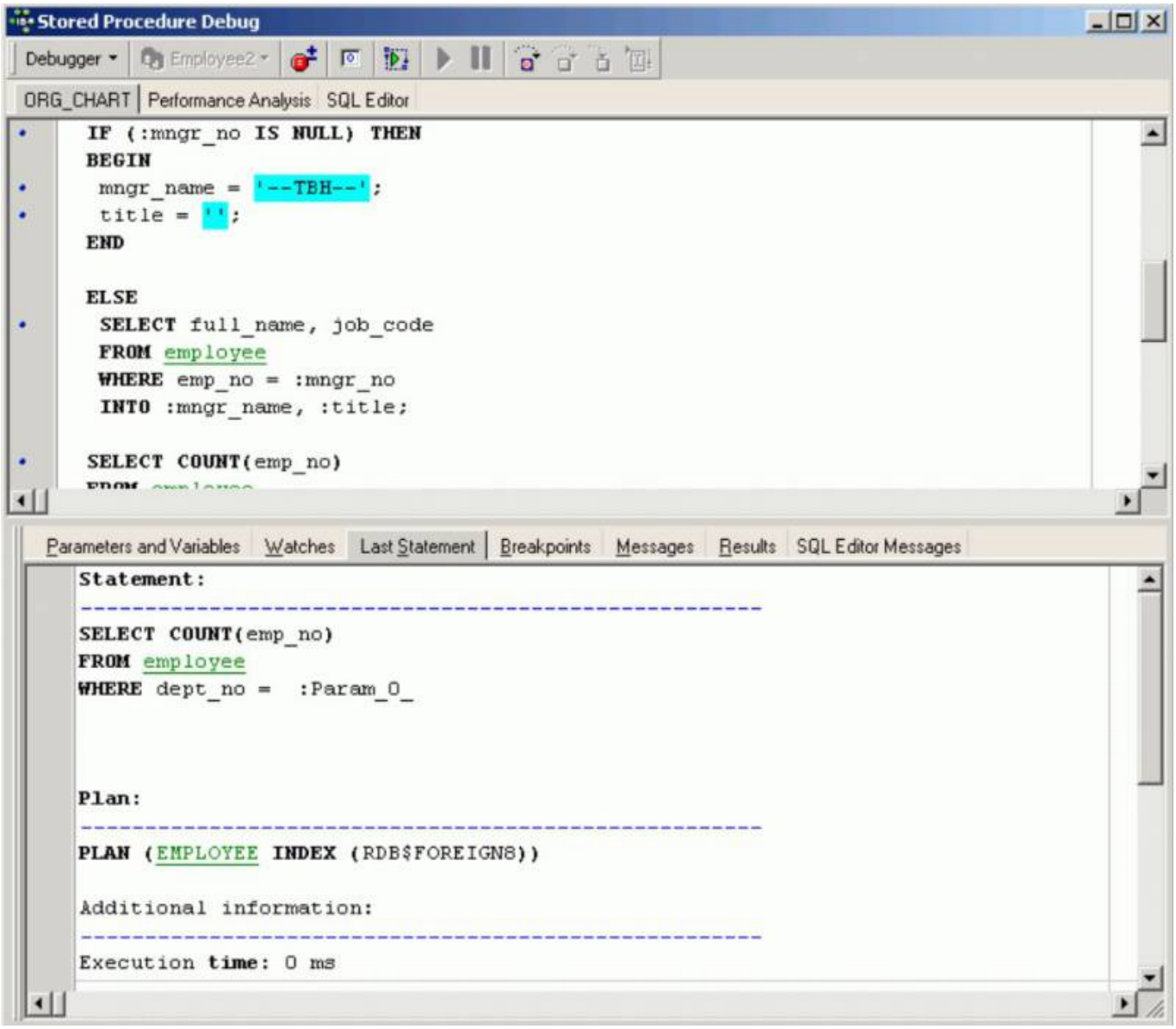
Watches



The *Watches* page displays those parameters and variables that have been checked for particular observation in the previous window.

Last Statement

Following execution, the last internal statement is displayed here, along with additional information such as execution time:



back to top of page

Breakpoints

This page displays the positions where breakpoints have been specified, using the respective icon in the Debug Procedure toolbar, the [F5] key, or by clicking on the blue points in the SQL left margin.

When the procedure is executed (using the respective icon or [F9]), it always stops automatically at these breakpoints. The procedure can thus be executed step by step, either using [F8] (or the

respective toolbar icon) to continue execution step by step (not including the next sublevel), or [F7] (or the respective toolbar icon) to continue step by step including the next sublevel.

Alternatively, if you have a procedure or trigger containing cursors, you can of course use the *Run to Cursor* icon, or [F4], to execute a part of a stored procedure or trigger up to the location of the cursor in the code editor.

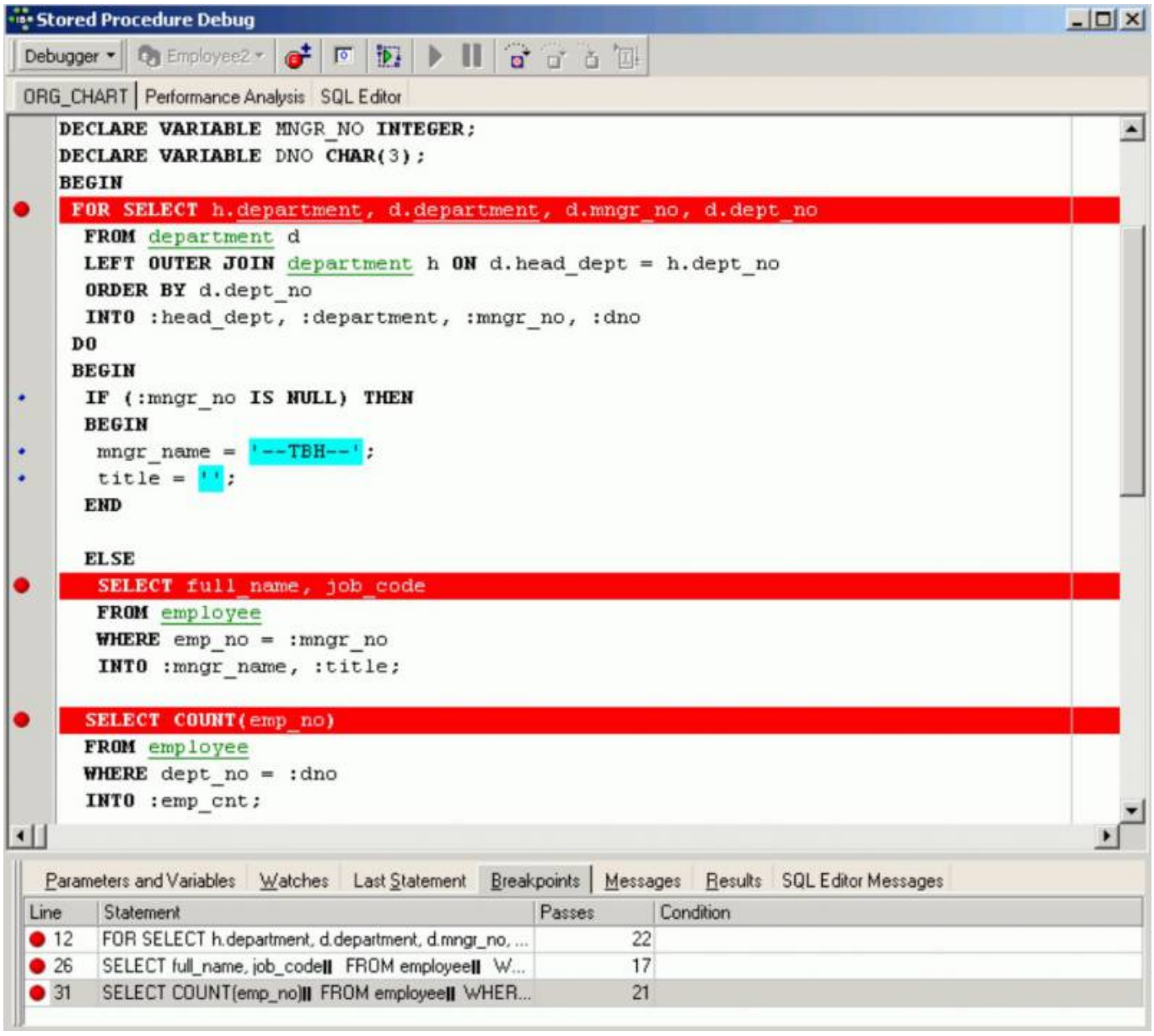
It is also possible to define breakpoints using comments. To define a breakpoint simply write a special comment line:

– IBE_BREAKPOINT

or

/* IBE_BREAKPOINT */

before the statement where the debug process should be paused.



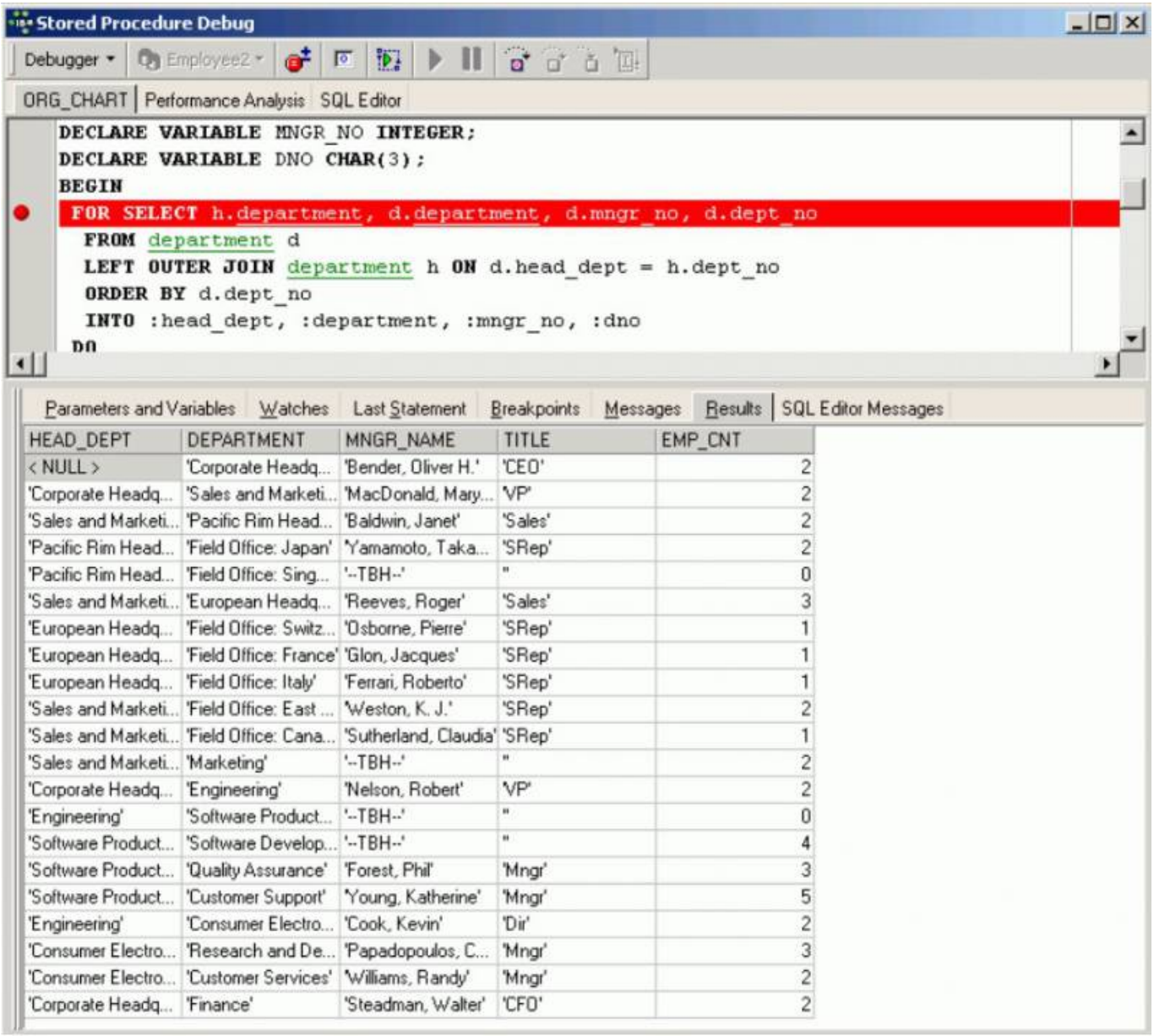
back to top of page

Messages

These indicate the sort of error that has occurred and where, by highlighting the relevant SQL row.

Results

This page only appears if there are output parameters in the procedure.



HEAD_DEPT	DEPARTMENT	MNGR_NAME	TITLE	EMP_CNT
< NULL >	'Corporate Headq...	'Bender, Oliver H.'	'CEO'	2
'Corporate Headq...	'Sales and Marketi...	'MacDonald, Mary...	'VP'	2
'Sales and Marketi...	'Pacific Rim Head...	'Baldwin, Janet'	'Sales'	2
'Pacific Rim Head...	'Field Office: Japan'	'Yamamoto, Taka...	'SRep'	2
'Pacific Rim Head...	'Field Office: Sing...	'--TBH--'	''	0
'Sales and Marketi...	'European Headq...	'Reeves, Roger'	'Sales'	3
'European Headq...	'Field Office: Switz...	'Osborne, Pierre'	'SRep'	1
'European Headq...	'Field Office: France'	'Glon, Jacques'	'SRep'	1
'European Headq...	'Field Office: Italy'	'Ferrari, Roberto'	'SRep'	1
'Sales and Marketi...	'Field Office: East ...	'Weston, K. J.'	'SRep'	2
'Sales and Marketi...	'Field Office: Cana...	'Sutherland, Claudia'	'SRep'	1
'Sales and Marketi...	'Marketing'	'--TBH--'	''	2
'Corporate Headq...	'Engineering'	'Nelson, Robert'	'VP'	2
'Engineering'	'Software Product...	'--TBH--'	''	0
'Software Product...	'Software Develop...	'--TBH--'	''	4
'Software Product...	'Quality Assurance'	'Forest, Phil'	'Mngr'	3
'Software Product...	'Customer Support'	'Young, Katherine'	'Mngr'	5
'Engineering'	'Consumer Electro...	'Cook, Kevin'	'Dir'	2
'Consumer Electro...	'Research and De...	'Papadopoulos, C...	'Mngr'	3
'Consumer Electro...	'Customer Services'	'Williams, Randy'	'Mngr'	2
'Corporate Headq...	'Finance'	'Steadman, Walter'	'CFO'	2

back to top of page

SQL Editor Messages

These are displayed here when applicable.

When debugging a procedure, first take a look at the values of the parameters and then use [F8] to go through the procedure step by step ([F9] executes fully). After each step, all variable values can be

seen. Don't forget to work with breakpoints [F5].

back to top of page

Edit procedure/alter procedure

Procedures can be altered directly in the Procedure Editor, started by double-clicking directly on the procedure name in the DB Explorer. Alternatively use the DB Explorer's right mouse-click menu item *Edit Procedure* or key combination [Ctrl + O].

ALTER PROCEDURE has exactly the same syntax as CREATE PROCEDURE. In fact, when procedures are altered the original procedure definition is replaced. It may seem that ALTER PROCEDURE is therefore not necessary, as a procedure could be dropped and then recreated to carry out any changes. However this will not work if the procedure to be changed is called by another procedure. If procedure A calls procedure B, procedure B cannot be dropped because procedure A depends on its existence.

The SQL syntax for this command is:

```
ALTER PROCEDURE <procedure_name>
<revised_input_parameter_list>
RETURNS
<revised_return_parameter_list>
AS
<local_variable_declarations>
BEGIN
<procedure_body>
END
```

The complete procedure header and body must be included in the ALTER PROCEDURE statement. The syntax is exactly the same as CREATE PROCEDURE, except CREATE is replaced by ALTER.

Important: Be careful about changing the type, number, and order of input and output parameters to a procedure, since existing application code may assume the procedure has its original format.

Procedures in use are not altered until they are no longer in use.

ALTER PROCEDURE changes take effect when they are committed. Changes are then reflected in all applications that use the procedure without recompiling or relinking.

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

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

back to top of page

Recreate procedure

Implemented in Firebird 2.0, this DDL statement RECREATE PROCEDURE is now available in DDL. Semantics are the same as for other RECREATE statements.

See also:

RECREATE PROCEDURE

Drop procedure/delete procedure

A procedure may only be dropped, if it is not being used at the time of deletion. Also it may not be dropped if it is used by other procedures, triggers, views or SELECTs, until this dependency is removed.

The Procedure Editor / Dependencies page displays which database objects use this procedure, and which objects this procedure uses. Most database objects can be dropped directly on the Dependencies page or the Dependencies Viewer by using the right-click menu on the selected object, and choosing the menu item Drop Object or [Ctrl + Del].

To drop a procedure use the DB Explorer right mouse-click menu item *Drop Procedure*... (or [Ctrl + Del]) or, if the procedure is already opened in the Procedure Editor, use the Procedure Editor main menü item, (opened by clicking *Procedure* in the top left-hand corner), *Drop Procedure*.

IBExpert asks for confirmation:

before finally dropping the procedure. Once dropped, it cannot be retrieved; the procedure has to be recreated, if a mistake has been made!

Using SQL the syntax is:

```
DROP PROCEDURE <procedure_name>;
```

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

From:
https://www.ibexpert.com/docu/ - **IBExpert**

Permanent link:
https://www.ibexpert.com/docu/doku.php?id=02-ibexpert:02-03-database-objects:stored-procedure

Last update: **2023/08/18 06:47**