IBMiSqlScripts

Processing SQL scripts

User guide

Contents

Introduction

This application was motivated by the fact that the popular utility Query/400 (later called Query for i) is unable to display and print all characters coded in Unicode character sets UCS-2 (CCSID 13488), UTF-16 (CCSID 1200), or UTF-8 (CCSID 1208).

This application enables creating, saving and running scripts of SQL statements for the IBM DB2 for i. Script is defined here as a text of SQL statements written in a text file with the suffix `.sql` or `.SQL`. One or more SQL statements delimited by a semicolon can be written in the script. The SQL statements may be of any kind (DDL or DML) and can contain parameters designated by question marks. Most of the time the script will be a query, i. e. a single SELECT statement. That means that the creator of the script must know the SQL language at least at the level of the SELECT statement.

Scripts are usually amended by specially structured comment lines which enable flexible formatting of the script result when displayed on the screen or printed on paper.

Programs are written in Java language and require version Java SE 8 or higher. They cooperate with IBM i Toolbox for Java (or JTOpen framework). The programs were created and tested Mac OS X, macOS, Windows 7, 10, 11 with remote Internet connection to system IBM i.

The application is not to be installed, it is ready to use (see Using the application below).

Note: Scripts for SQL routines or triggers written in SQL PL language cannot be processed.

Using the application

The application is not installed. It is delivered as a directory containing the following directories and files.

Directories

- *helpfiles* - contains user guides in Czech and English languages,
- *icons* - contains icons for buttons in the editor,
- *logfiles* - contains text files *err.txt* and *out.txt* to which messages are redirected from files System.err and System.out (console).
- *paramfiles* - contains file *Parameters.txt* with the application parameters,
- *printfiles* - contains text files, which are result of script runs,
- *scriptfiles* - contains skript files whose names end with `.sql`,
- *workfiles* - contains the text file *Print.txt*, result of the most recent script run.

Note: Files *err.txt* and *out.txt* serve to find the cause of an error in program.

Program files

- File *jt400-21.0.4.jar* - contains subset of Java classes from the framework IBM i Toolbox for Java.
- File ***Q_Menu.jar*** - contains Java application classes of programs and starts the application for the application administrator and the creator of scripts.
- File ***Q_MenuUser.jar*** - contains Java application classes of programs and starts the application for the application user.

Location and running the application

The application directory can be placed anywhere and possibly renamed. Shortcuts (aliases) can be created from files ***Q_Menu.jar*** (programs for the administrator) ***Q_MenuUser.jar*** (programs for the user) and placed somewhere.

The application starts by double click on the shortcuts (aliases) or on the original `.jar` files; the application menu will show.

The same copy of the application works in systems macOS and Windows.

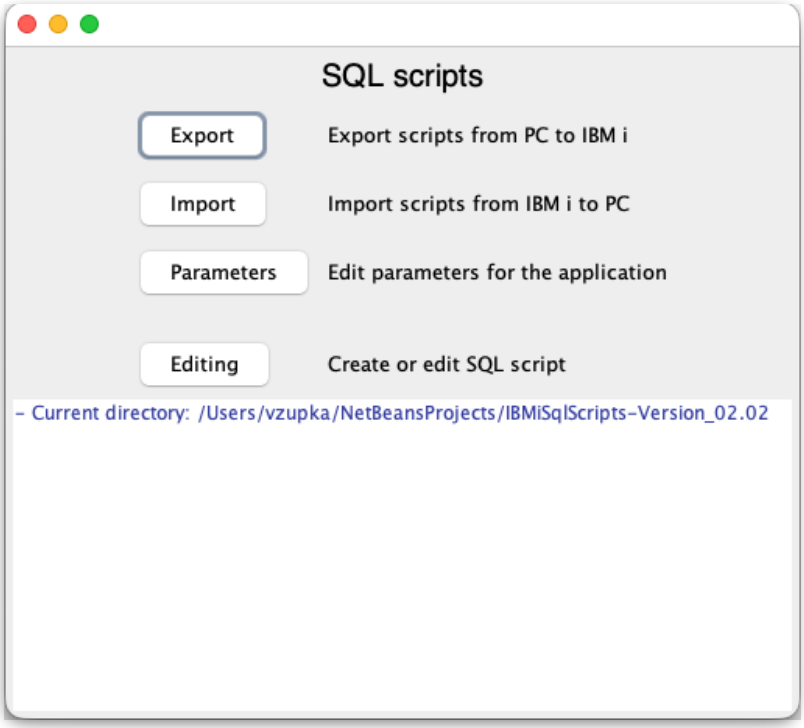
Installing a new version

Replace two `.jar` files *Q_Menu.jar* a *Q_MenuUser.jar* by their new versions. Replace also uuser documentation in the directory *helpfiles* by the new files *IBMiSqlScriptsUserDocEng.pdf* and *IBMiSqlScriptsUserDocCz.pdf*.

Important: Direcrory *scriptfiles* contanis scripts created by application administrator. It is possible to save them into another directory or rewrite them by scripts from another directory. Scripts may be saved more reliably into an IFS directory, which the administrator creates for that purpose. One or more such directories can be created and used for *export* from or *import* to the directory *scriptfiles*, *(*see below*)*.

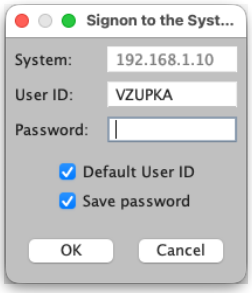
Screens for the application administrator

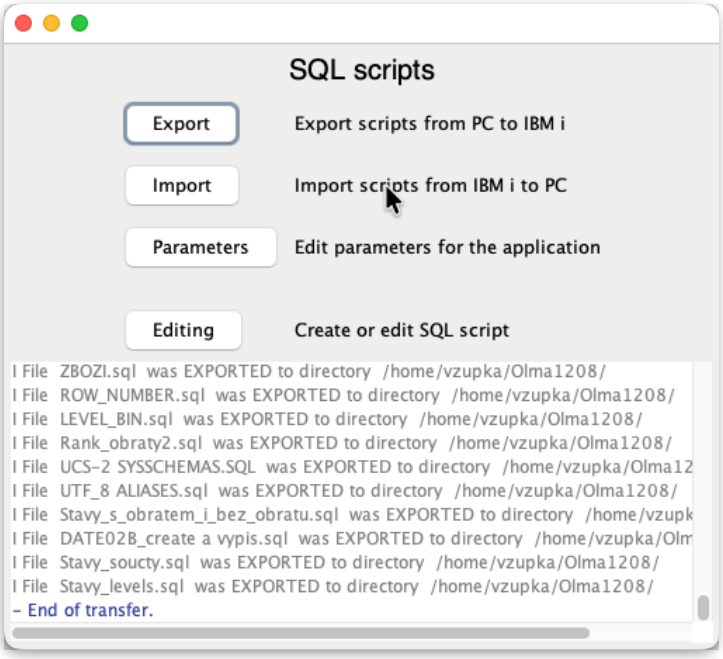
The first screen of the application is a menu of a set of functions.



Export

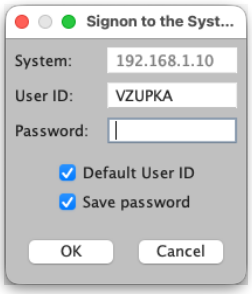
This function invokes the signon dialog to access system IBM i (if the user is not already signed on). Then *all scripts* are transferred from the local directory *scriptfiles* to the IFS directory defined in application parameters. A protocol about transfer is displayed.

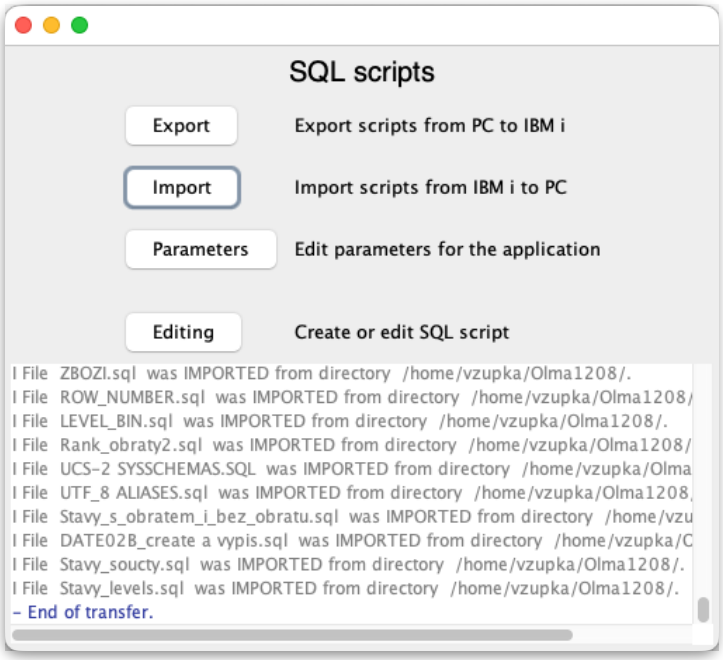




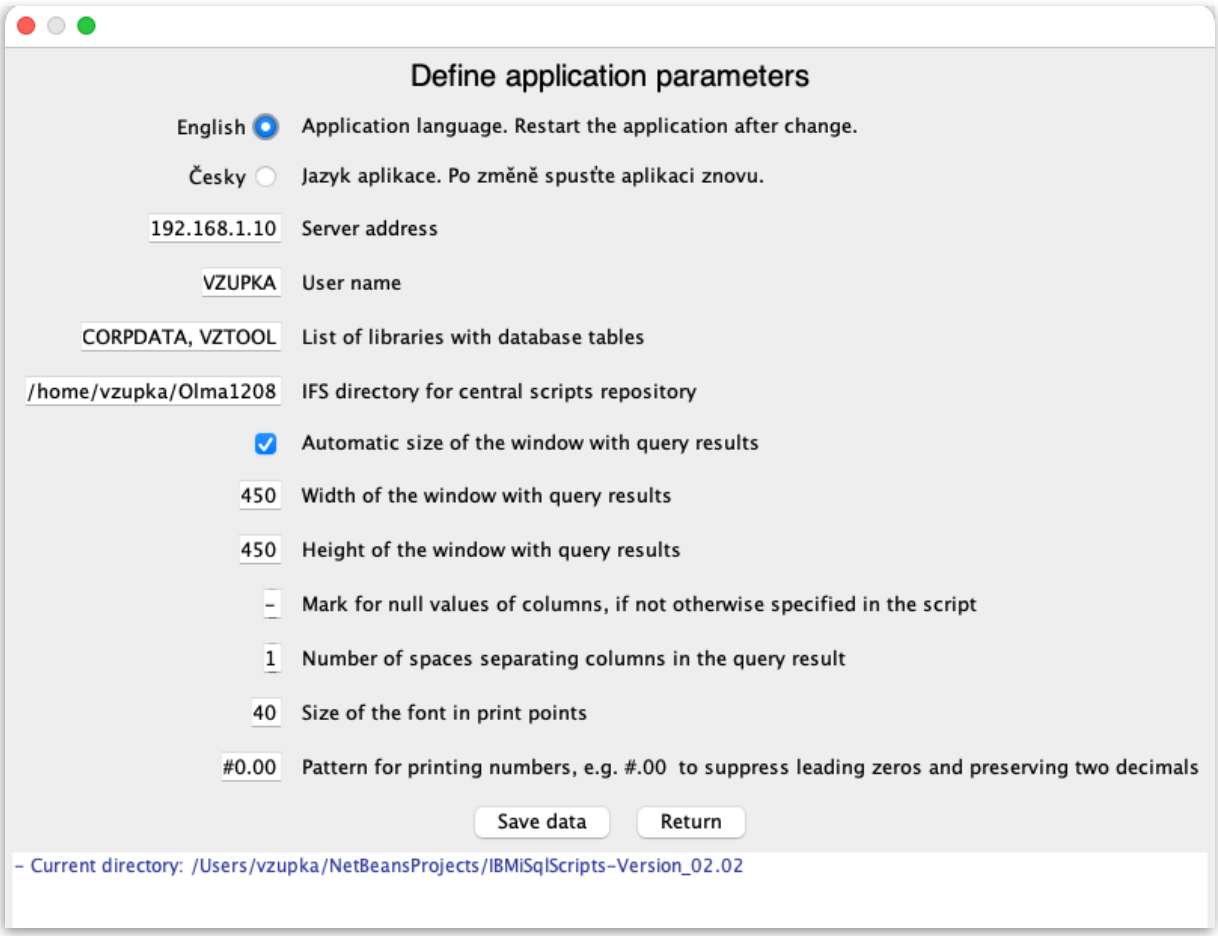
Import

This function invokes the signon dialog to access system IBM i (if the user is not already signed on). Then *all scripts* are transferred from the IFS directory (defined in application parameters) to the local directory *scriptfiles*. A protocol about transfer is displayed.

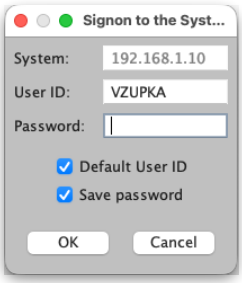




Parameters



All users may enter a profile name for access to the system IBM i. This name will be used in the *Signon to the System* dialog as a default.



Application language

The application can be run in US English (en_US) or Czech (cs_CZ) localization. Localization concerns titles, messages, button texts, and formatting of decimal numbers in the script result, and data and time in the result header. The option is fully applied only after restarting the application.

Server address

Enter single IP address in dot or domain form.

User name

The administrator enters a profile name with authorization to write files to the IFS directory. If manipulation SQL statements (other than SELECT) are to be run, the corresponding authorization is necessary.

List of libraries with database tables

The application uses naming convention "*system*" for SQL statement processing. The administrator enters one or several library names separated by a blank or a comma.

At the time of connecting the database to the program a library list is created that replaces the library list of the database server job. The libraries are searched in the sequence when the SQL statement is processed. If the first entry is *LIBL, the specified list is added to the library list of the database server job.

Note: A library (schema) name qualifying an SQL object in the script proper takes precedence, of course. The separator of the qualifying library can be either a *dot* (**.**) or a forward *slash* (**/**). As soon as such a statement is used, subsequent running statements search objects in this library (schema) only. If they do not find them there, the system reports an error message. To remedy this situation you can:

- Write statement SET SCHEMA DEFAULT at the end of the script; it activates the original library list.
- Create a script of a single statement SET SCHEMA DEFAULT and run it before running the statement that did not find the required objects.

IFS directory for central scripts repository

The IFS directory is used as a central repository to *export* and *import* of *scripts* to/from the IBM i server. If no path to a directory is entered these functions cannot be used. Path to the directory must begin and may or may not end with a forward slash /. The application ensures correctnes of the path ending.

The directory must be created beforehand by the CL command CRTDIR (MKDIR, MD), e. g.

```
CRTDIR DIR('/home/vzupka/OlmaOSX')
```

Script texts will be written into the directory in UTF-8 encoding. This encoding can be defined for all the files by the CL command

```
CHGATR OBJ('/home/vzupka/OlmaOSX') ATR(*CCSID) VALUE(1208)
```

Browsing of script texts will then be easier in 5250 emulation using CL command

```
WRKLNK '/home/VZUPKA/Olma1208'
```

Size of the window with the script (query) results

If the checkbox "Automatic size of the window with query results" is selected the result window accommodates to the dimensions of the displayed results. Otherwise, the window will display in the dimensions entered in the fields "Width of the window with query results" and "Height of the window with query results" with sliders, if necessary.

Note: If the input value is not an integer, it is changed to 0.

Mark for null column values

The text (mark) entered in this field (or empty text) will be displayed (and printed) in the query result for all NULL column values.

Note: A text for null column values can also be defined in each individual script using a comment definition line (see below).

Number of spaces separating columns in the query result

This entry designates number of spaces that are appended to each column in header lines and data lines in the query result text. Can also be 0. If the input value is not an integer, it is changed to 0.

Note: The number of spaces can also be defined in each individual script using a comment definition line (see below).

Size of the font in print points

This entry is the number of points, where one point has the size 1/72 of inch. It is applied in printing on paper. If the input value is not an integer, it is changed to 0.

Note: The font size can also be defined in each individual script using a comment definition line (see below).

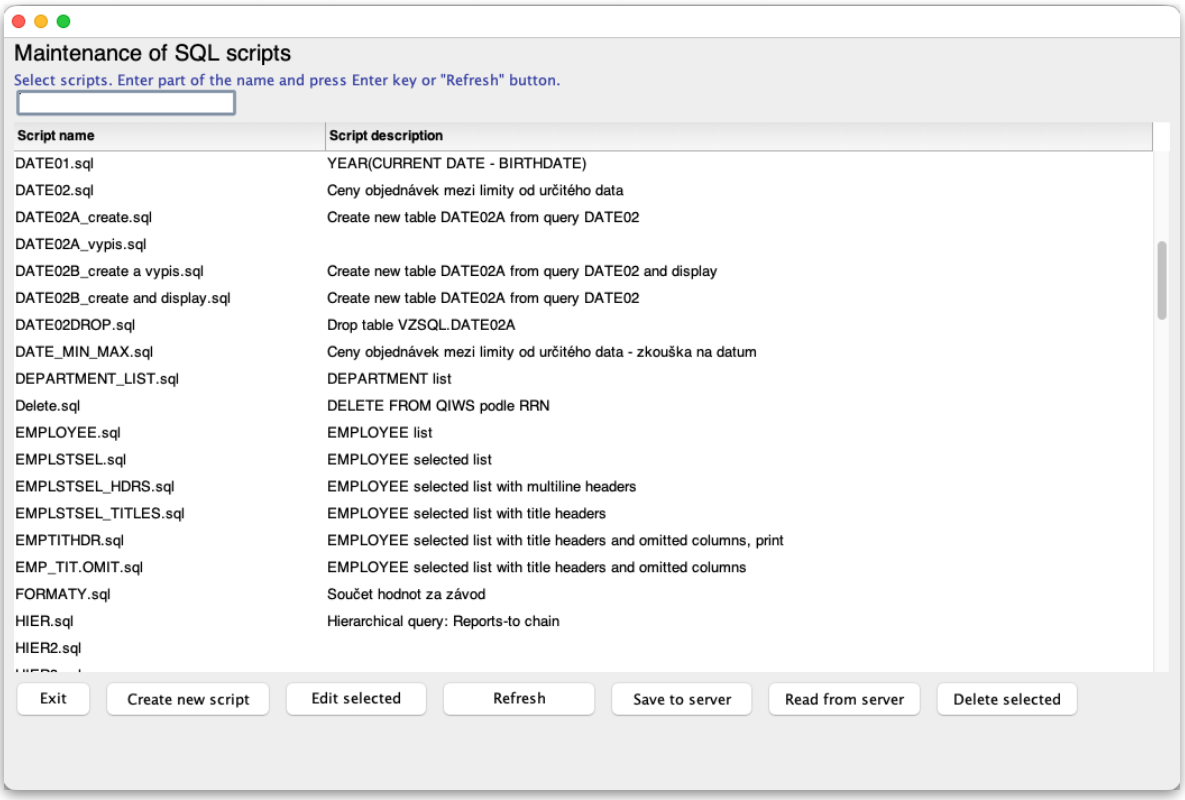
Pattern for printing numbers

This entry is the pattern (mask) for printing numbers which defines the output picture (format) of decimal numbers in all scripts. A common pattern can be #.00 that suppresses leading zeros and keeps two decimal positions including trailing zeros. If an empty value is entered in this field, the patterns defined for individual columns are applied, or the rule for standard output of numbers will hold (see below).

Note: The pattern can also be defined for individual columns in each script using a comment definition line (see below). The individual pattern always takes precedence.

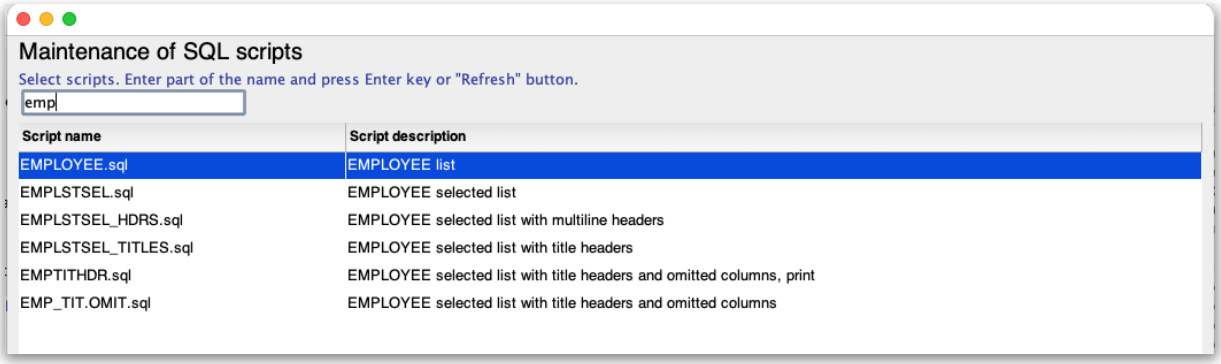
Editing

A table with the list of scripts is displayed which can be empty initially. If it is non-empty, the whole list is displayed when the input field is empty.



The user can shorten the (non-empty) list entering a *search pattern* in the text field and pressing *Enter* key or *Refresh* button.

The search pattern is *any text* to search in all script names of *scriptfiles* directory. For example, text *emp* limits the list to script names containing this text. The case of letters is ignored.

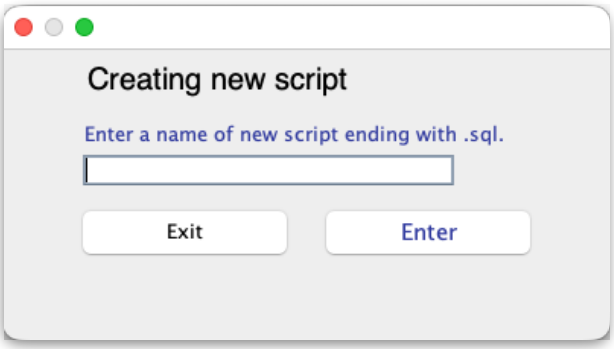


Here, names started with EMP weree chosen.

A series of buttons is at the bottom of the window: *Exit*, *Create new script*, *Edit selected*, *Refresh*, *Save to server*, *Read from server*, *Delete selected*. Description follows.

Create new script

The button invokes a window with an input field, where the user writes a name ended with `.sql` or `.SQL` and confirms with the *button Enter* or the *key Enter*.



If a script with this name already exists, a message is reported.

Note: The script name must not be empty and may contain only ASCII characters and must end with `.sql` *or* `.SQL`.

Refresh

This button clears the message (if present) and cancels any selection of a script. If the input field contains a text, only scripts that contain this text (no matter of letter size) will show in the list. If the field is empty, all scripts are shown.

Save to server

This button invokes the signon dialog to access system IBM i (if the user is not already signed on). Then the *selected script* is transferred from the local directory *scriptfiles* to the IFS directory (defined in application parameters). A message about the transfer is displayed.

Read from server

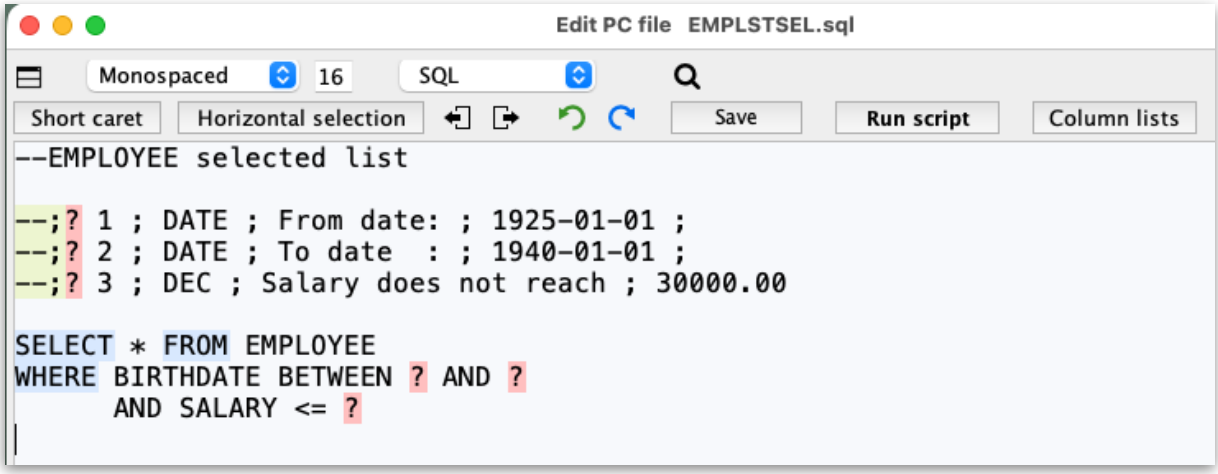
This button invokes the signon dialog to access system IBM i (if the user is not already signed on). Then the *selected script* is transferred from the IFS directory (defined in application parameters) to the local directory *scriptfiles*. A message about the transfer is displayed.

Delete selected

The selected script is deleted from the local directory *scriptfiles* (without any confirmation).

Edit selected

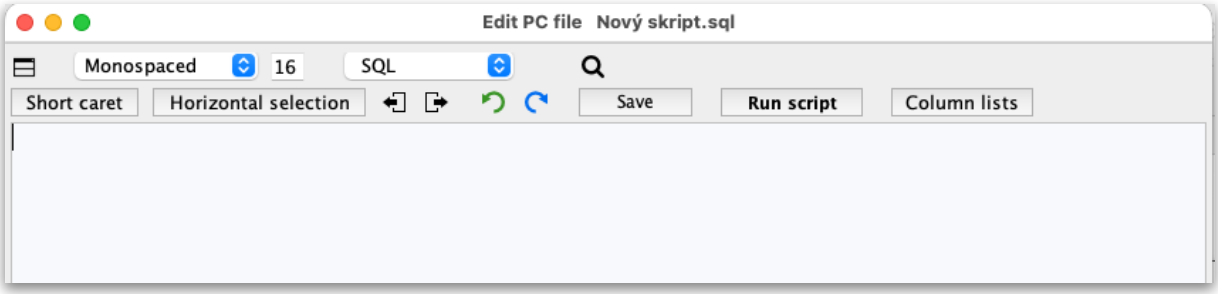
The selected script (e. g. EMPLSTSEL) is displayed in a new window and the user can enter or edit the text of the script and then press *Save* button or keyboard schortcut Ctrl S



Note: *Cmd* key is used instead of Ctrl in macOS.

The script is saved to its file in the directory *scriptfiles*.

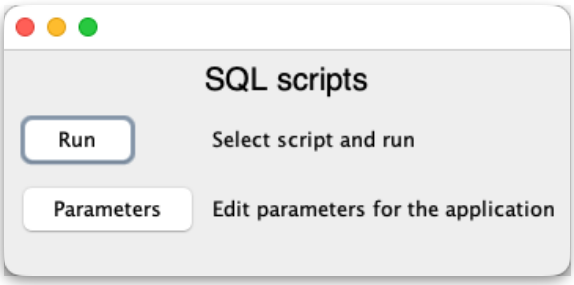
If the script is new, the editor contains an empty area.



The user can enter the whole script or copy/paste it from another source (see chapter Editing scripts).

Screens for the user

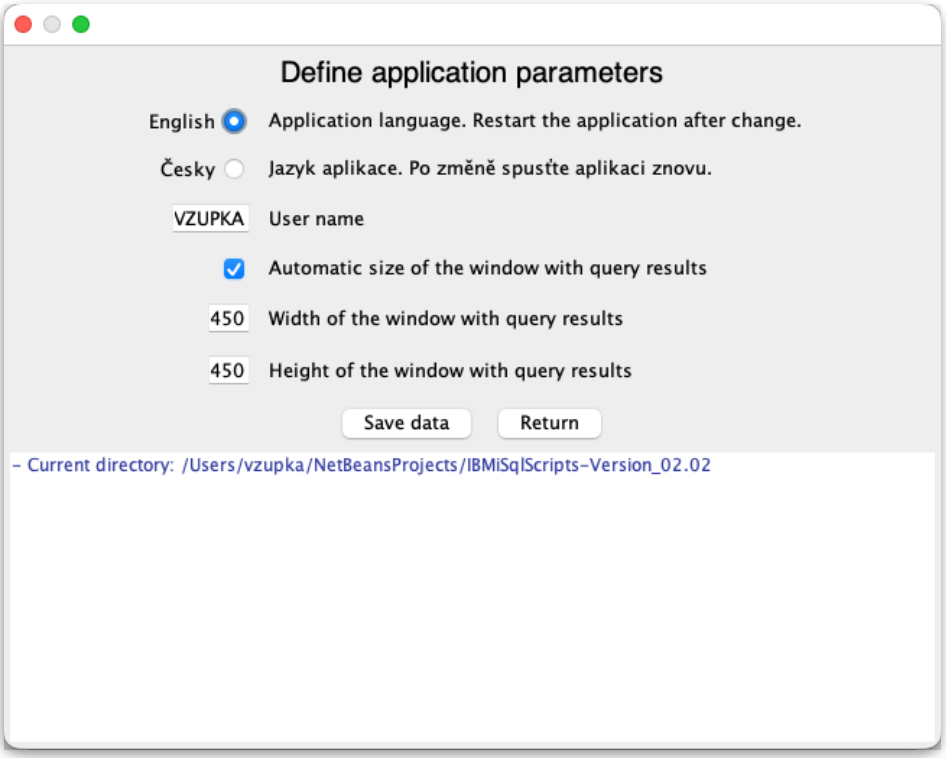
This main screen is shortened version of the one designed for the application administrator.



Run

This function is described below in chapter *Run scripts*.

Parameters



This function is a simplified version of the one designated for the application administrator (see above). It offers only those parameters the ordinary user makes use of. The parameters have the same meaning as described above.

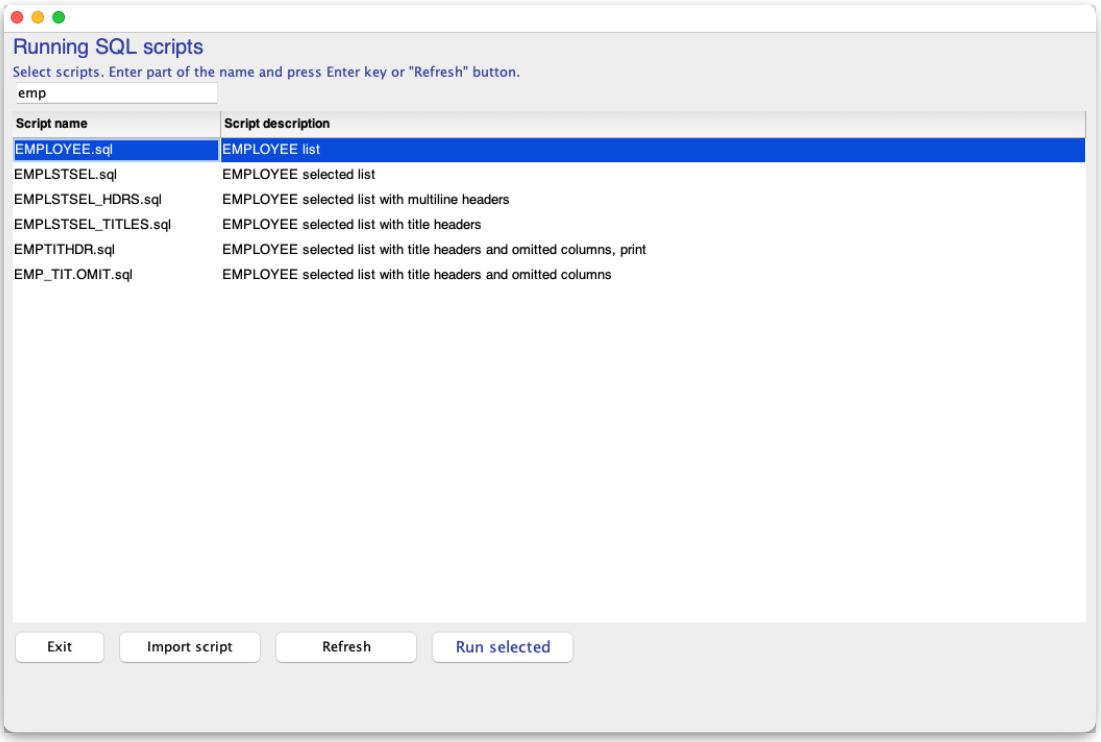
Running scripts

In this chapter, running scripts is illustrated in examples. The scripts introduced here are assumed to have been previously created by the administrator. The user can't see the text of the script, only its description and the result.

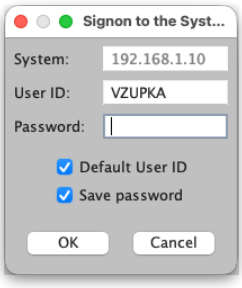
Press button *Run* in the application main menu and the window with the title *Running SQL scripts* is displayed containing list of scripts. The list is empty initially. In order to get entries in the list, scripts must be created. Usually, the application administrator creates them using the Edit function.

Query without variable parameters

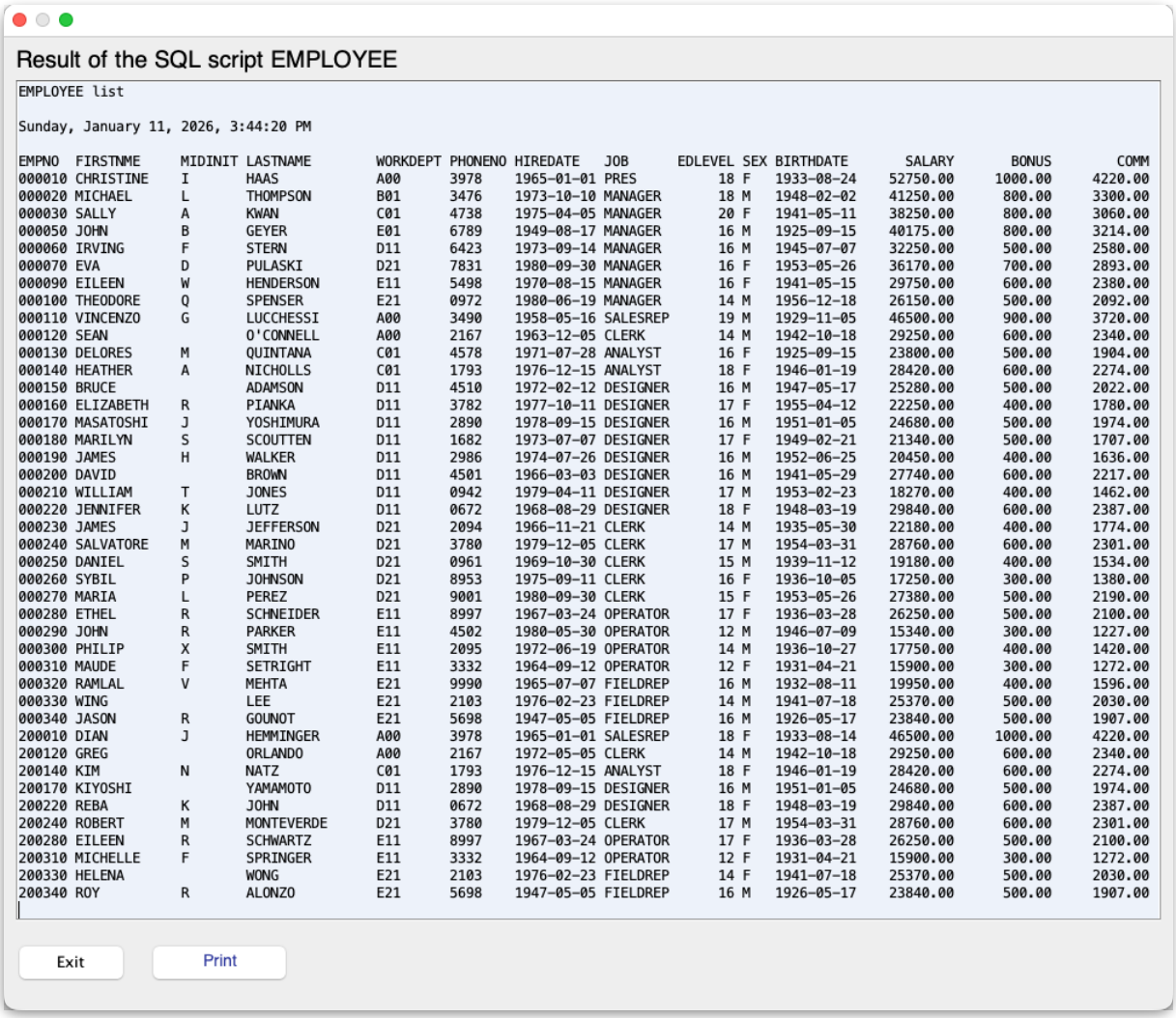
Select the row with the script name “EMPLOYEE.sql” and run the following query using button *Run selected*. The script is as follows.



If the server was not connected, the Signon window will show, where password must be filled.



The result of the query is displayed :



The result is also written to the file *Print.txt* in directory *workfiles* and also to the file `EMPLOYEE.sql` in directory *printfiles*. Names of columns from the table EMPLOYEE are used as headers.

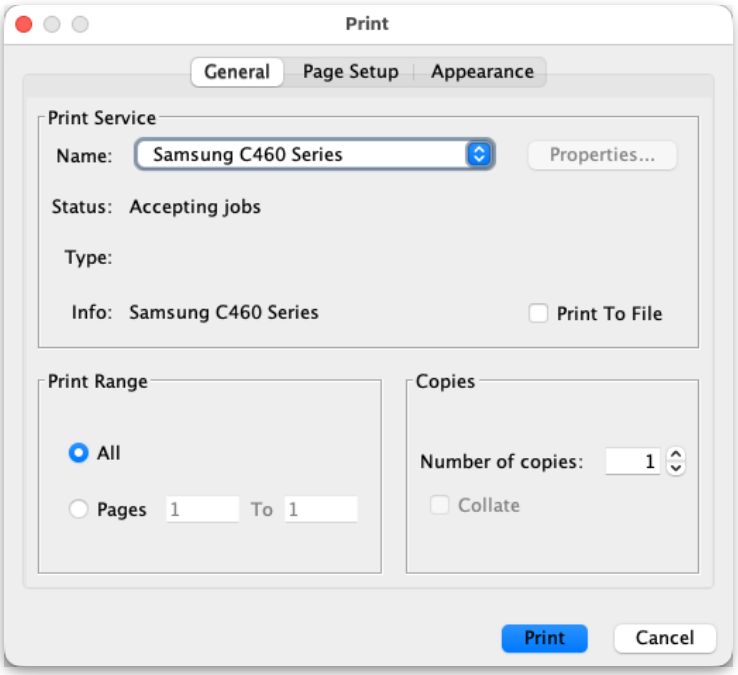
The script, which the user does not see, is a simple one as follows:

```
--EMPLOYEE list
SELECT * FROM EMPLOYEE
```

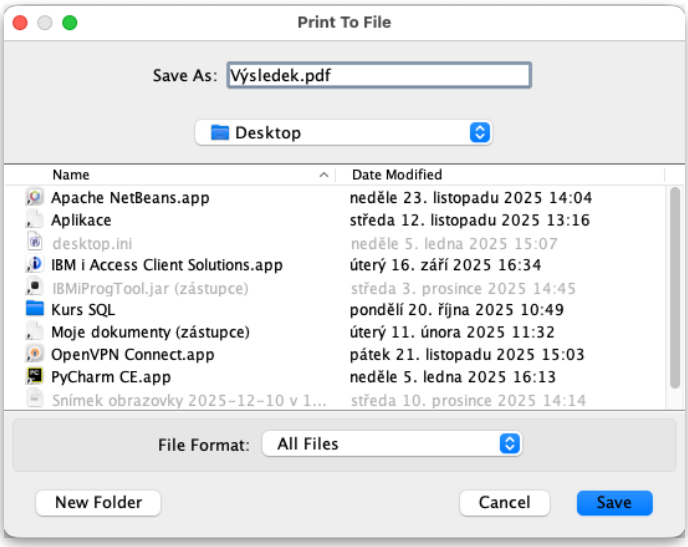
Now, you may be happy with the result and exit from the list, or print the result on a printer using the button *Print*.

Printing the script result

To perform the printing, a dialog like this is displayed.



If you check *Print to file*, and after that press button *Print*, menu of directories is shown.

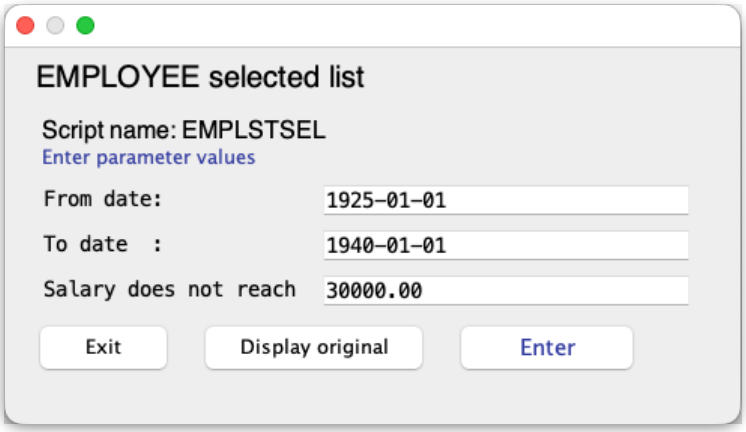


Select a directory, for example `Desktop`, where you can save the result, maybe as file `Výsledek.pdf`.

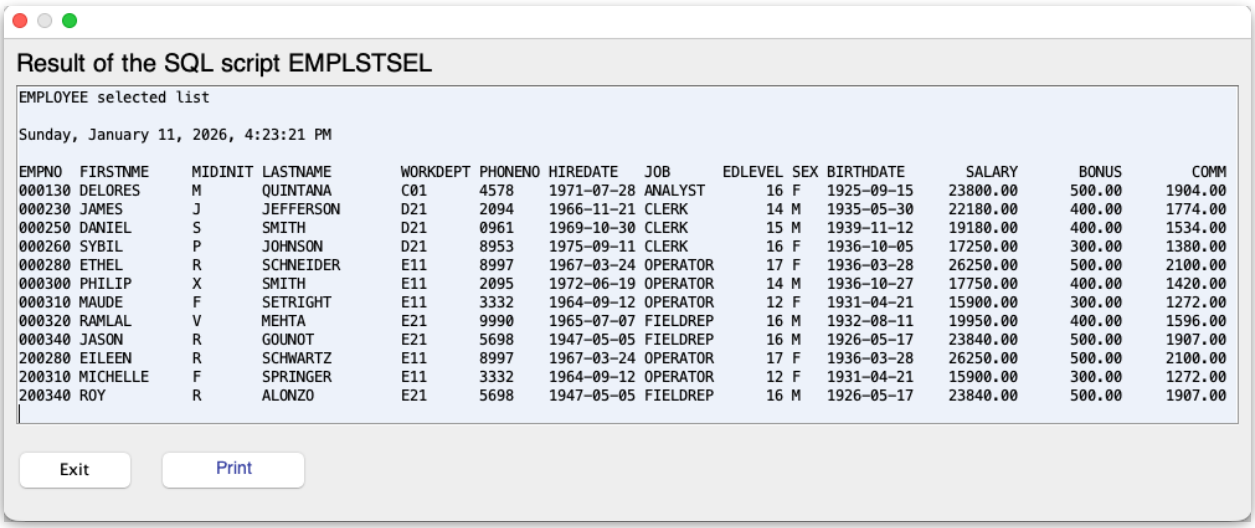
Query with variable parameters

An SQL statement can contain parameters depicted by question marks.Select the row with the script name `EMPLSTSEL.sql` and run the query script below using the button *Run selected*.

The following dialog window is displayed. Three input fields contain default values of parameters, that can be changed.



Keeping the default values and pressing *Enter* button shows the following window with the



query result.

The user may, like in the previous example, be happy with the result and exit the query, or print the result on the printer.

This script contains three question marks in the SELECT statement that define places for variable values.

```
--EMPLOYEE selected list

--;? 1 ; DATE ; From date: ; 1925-01-01 ;
--;? 2 ; DATE ; To date  : ; 1940-01-01 ;
--;? 3 ; DEC ; Salary does not reach ; 30000.00

SELECT * FROM EMPLOYEE
WHERE BIRTHDATE BETWEEN ? AND ?
      AND SALARY <= ?
```

Three comment lines begin with four-character symbol **`--;?`** in the first position and contain four entries delimited by semicolon. They enable building a prompt window for the user to enter variable values in places of question marks.

The *first* entry is a number that must exactly correspond to the sequence of the corresponding question mark. Here, number 1 stands for the first, number 2 for the second, and number 3 for the third question mark.

The *second* entry is the data type of the SQL column (here DATE and DEC). It serves for a preliminary check of the entered value and for documentation. The entry may be empty, but the delimiting semicolon must be present.

The *third* entry is a explaining text for the value to be entered, here it is `From date:`, etc. This entry may be empty but with the delimiting semicolon.

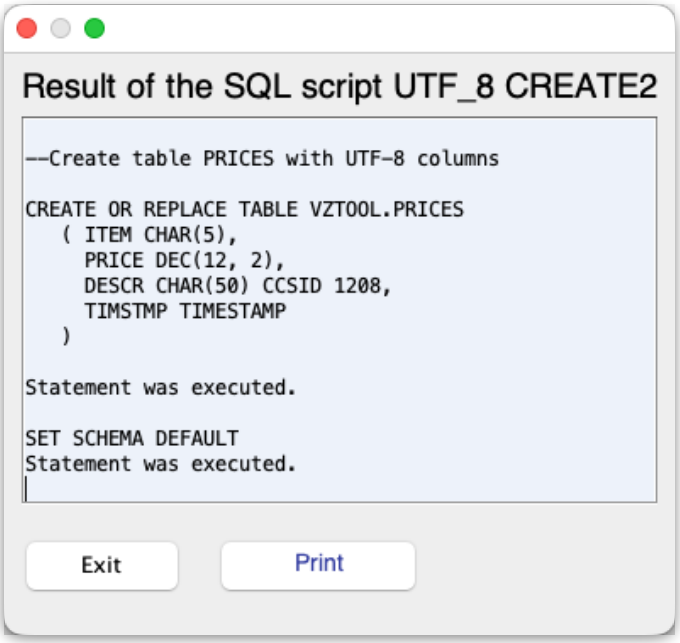
The *fourth* entry is a default value of the parameter. It must correspond to the data type, here it is the date `1925-01-01` in ISO format, etc. This entry may or may not end with a semicolon.

Non-query script

As an illustration, the non-query script UTF_8 CREATE2.sql in the schema (library) VZTOOL is used:

```
--Create table PRICES with UTF-8 columns
CREATE OR REPLACE TABLE VZTOOL.PRICES
   ( ITEM CHAR(5),
     PRICE DEC(12, 2),
     DESCR CHAR(50) CCSID 1208
   );
SET SCHEMA DEFAULT
```

Every statement but the last is ended by semicolon.



Rules for writing scripts

A script is composed of one or more SQL statements delimited by a semicolon. The last statement ends with the last text line (not a semicolon). The script can contain comments – simple and bracketed.

The s*imple comment* begins with two consecutive hyphens (`--`) and ends with end of line. Simple comments are used in this application for special parameters for the program that evaluates the parameter values before or after the script run.

The *bracketed comment* begins with `/*` and ends with `*/` like in different programming languages. It can be placed in the script where the space is required by the statement. It is not used for parameter specification.

The *first line of the script* beginning in the position 1 of the line with the two dash characters (`--`) is considered *script description*. The text after the two dashes is displayed in the list of scripts as the script description.

The important part of scripts are *special comment lines* beginning in the position 1 of the line with special 4-character symbols. They represent the following parameters:

`--;?` variable parameter values (question marks in the SQL statement),
`--;t` title header,
`--;H` header line for columns in the query result,
`--;T` vertical formatting of total (group) lines and symbol for null column values,
`--;D` pattern (mask) for output of decimal numbers (DECIMAL and NUMERIC types),
`--;O` list of columns to be omitted from the result output,
`--;L` group level, separation text and grouping column name,
`--;S` summarized column name amd list of summary types (S, A, M, m, C),
`--;s` list of leading texts for summary lines (replacing standard SUM, AVG, MAX, MIN, COUNT),
`--;P` print parameters.

These four-character symbols were selected so that any characters might follow (except for the semicolon that divides the values).

Definition of variable parameter values (parameter markers)

Question marks called *parameter markers* in the SQL statement text serve as placeholders for variable values. Values are assigned to the SQL statement before it is performed. Special comment definition lines serve to this purpose. Number and sequence of the definition lines determine the number and sequence of the input fields in a prompt window. The definition comment line begins with **`--;?`** in position 1 and defines

- serial number of the parameter (marker) in the SQL statement,
- data type of the parameter,
- explanation test of parameter meaning,
- default value of the parameter.

```
--;? sequence-number; [data-type]; [explanation]; [default-value][;]
```

Number of definition lines must be the same as the number of question marks. Each definition line contains exactly four entries delimited by a semicolon. Leading and trailing spaces in the entries are ignored.

Note: If the script contains more than one SQL statement *only one* SQL statement can make use of parameter markers.

Entries in the definition line

Sequence number

The first entry is a mandatory sequence number which must exactly match the sequence of the corresponding question mark in the SQL statement. Sequence number 1 applies for the first question mark, 2 for the second question mark etc. The sequence of input fields in the window is determined by the sequence of definition lines in the script, not by this number.

Data type

The second entry is optional and designates one of the following data types.

DEC
DECIMAL
NUMERIC
INT
INTEGER
BIGINT
DATE
TIME
TIMESTAMP

The listed types serve for preliminary check of the value entered in the input field of the prompt window, and also for documentation. If one of these types is entered the check is performed *before the SQL statement is run*. If this entry is not present or if another text is entered, the check of the type is done only *at the time of performing of the SQL statement*. This entry may be empty but the semicolon must be present.

Explanation text

The third entry is an optional explanation text to the parameter value, e. g. `From date:`. This entry can also be empty but it must be ended by a semicolon. Leading or trailing spaces do not matter.

Default value

The fourth entry is a default value corresponding to the data type, e. g. *1925-01-01*. This entry can also be empty and may and may not be ended by a semicolon.

Entries of types DEC, DECIMAL, NUMERIC may contain digits, decimal point and a minus sign before the number.

Entries of types INT, INTEGER, BIGINT may contain digits and a minus sign before the number.

Entries of type DATE must be entered in ISO format, YYYY-MM-DD, e. g. 2014-02-15.

Entries of type TIME must be entered in ISO format, HH:MM:SS, e. g. 19:31:05.

Entries of type TIMESTAMP must be entered in the ISO format,

YYYY-MM-DD HH:MM:SS.MMMMMM, e. g. 2000-04-05 23:59:59.999999

The length of the timestamp is 26 characters. A space is between date and time. Leading zeros can be omitted from the months, days, hours, minutes, seconds. Trailing zeros in

microseconds may be partly or completely omitted. The form ending with 24:00:00.000000 is allowed.

Note: If the column type is BINARY or VARBINARY the default value should be entered in hexadecimal characters (0123456789abcdefABCDEF). Two characters correspond to a byte.

Definition lines can be entered in arbitrary sequence; their sequence numbers are important.

Examples of definitions:

```
--;?01 ; ; Price from: ; 5.50;
--;? 02;DEC; Price to:; 23000
--;? 03; DATE; From date:;2014-01-29;
--;? 04;  ; Binary data - two bytes: ; 0F9C;
```

Entering parameter values when running scripts

When the prompting window for entering SQL query parameters is displayed both default data from definitions and data entered to input fields by the user are checked. The data must correspond to the type *entered* in --;? definition comment line. This check is done before the SQL statement is started.

- DEC, DECIMAL, NUMERIC type is checked for the number format. The number must contain only digits, decimal separator and minus or plus sign. Decimal separator, if present, must be a dot (`.`). The sign must be on the left.
- DATE type is checked against format ISO.
- TIME type is checked against format ISO.
- TIMESTAMP type is checked against format ISO.

If the type is missing or a different text is entered in the specification line, a failure (if any) is reported at the time the SQL statement is performed. The SQL message will be apparent in the window that shows after performing the script.

Note 1: If the column type is BINARY or VARBINARY the value should be entered in hexadecimal characters (0123456789abcdefABCDEF). Two hexadecimal characters correspond to a byte. The value is not checked for correctness. Invalid character is translated to 0.

Note 2: Query results of the BINARY or VARBINARY columns are also shown in the hexadecimal form.

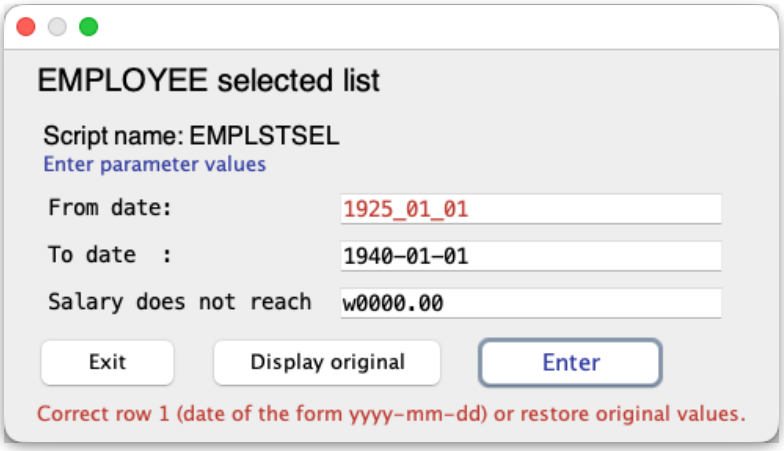
Example of entering parameter data

The following script defines three parameters (question marks).

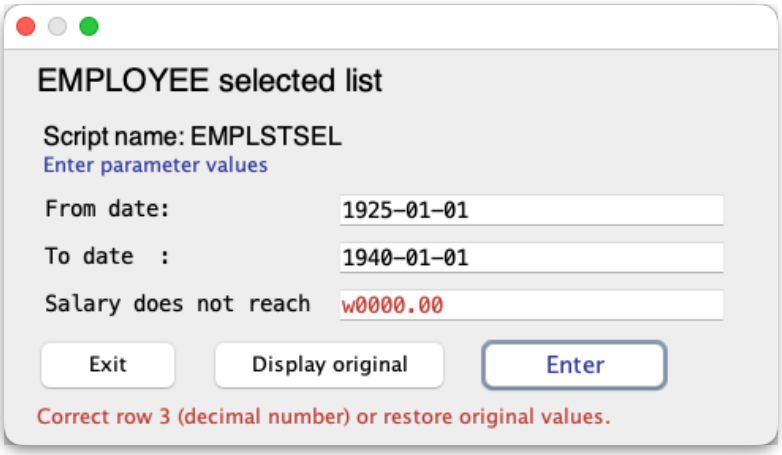
```
--EMPLOYEE selected list

--;? 1 ; DATE ; From date: ; 1925-01-01 ;
--;? 2 ; DATE ; To date  : ; 1940-01-01 ;
--;? 3 ; DEC ; Salary does not reach ; 30000.00

SELECT * FROM EMPLOYEE
WHERE BIRTHDATE BETWEEN ? AND ?
      AND SALARY <= ?
```

After starting the script a prompt dialog is invoked which allows changing the default parameter values and/or running the SQL statement (button Enter).

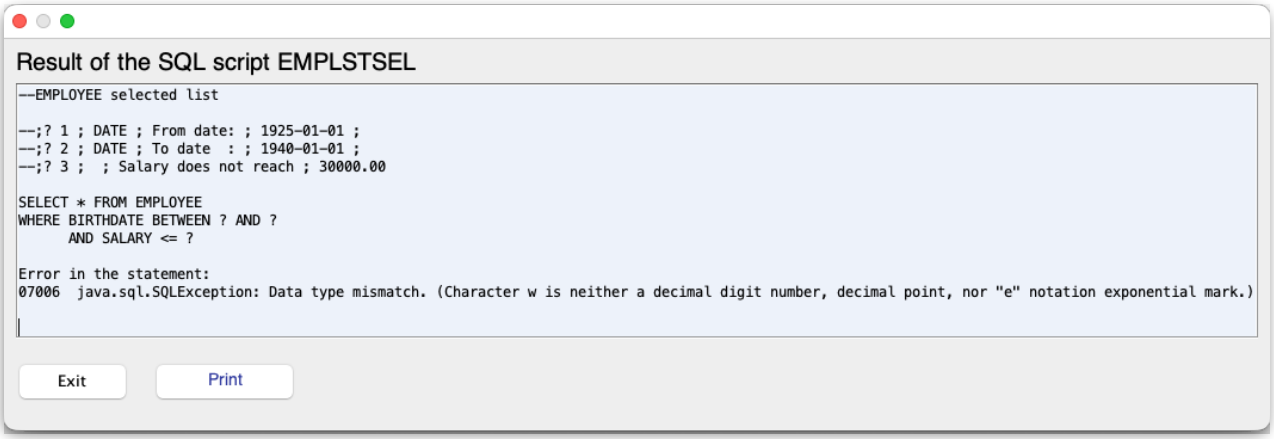
If we change the hyphen in the first parameter to underscore and the number in the third parameter to w0000.00, a message is displayed for the error in the *first* parameter.



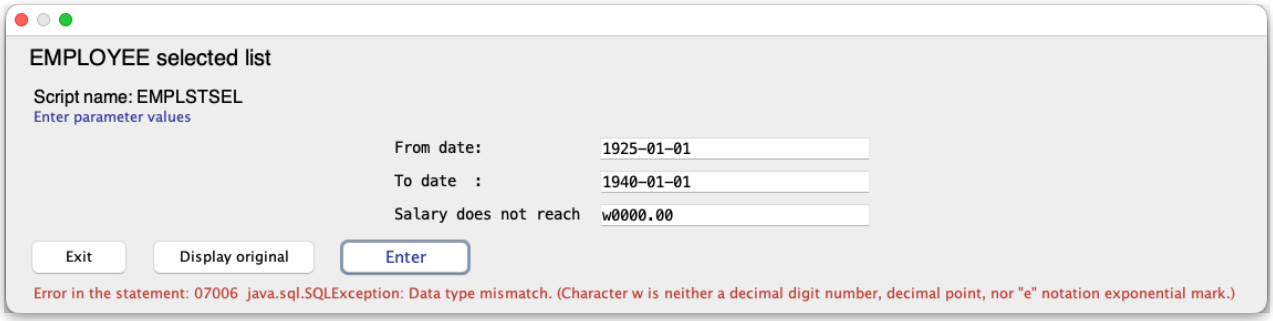
After correcting the first parameter, error in the *third* parameter is reported.



If we omit the type DEC in the third parameter definition and make the same error, the script will run and the result shows a copy of the script and the SQL error message.



After exiting the prompt dialog is redisplayed with the same error message.



Definition of title headers

Any number of title header lines can be specified for a query script. They are applied in the sequence as written in the script. The title header line specification begins with the symbol **`--;t`** in the position 1 of the specification line after which any text may follow.

The title header line can contain symbols of variable names that have the following form:

`&column`

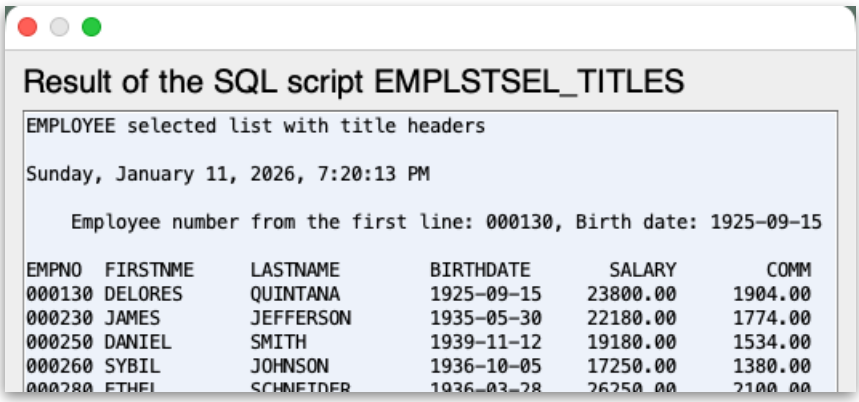
where *column* is the column name followed by a space or end of line. The variable is replaced by the column value from the *first line* of the result set even if the column is specified as omitted (see below).

Title headers are written out before column headers and are printed on the first page only.

Script EMPLSTSEL_TITLES.sql

```
--EMPLOYEE selected list with title headers

--;t    Employee number from the first line: &EMPNO , Birth date: &BIRTHDATE
--;t

--;? 1 ; DATE ; From date: ; 1925-01-01 ;
--;? 2 ; DATE ; To date  : ; 1940-01-01 ;
--;? 3 ; DEC  ; Salary does not reach ; 30000.00

SELECT EMPNO,
   FIRSTNME,
   LASTNAME,
   BIRTHDATE,
   SALARY,
   COMM
FROM EMPLOYEE
   WHERE BIRTHDATE BETWEEN ? AND ?
         AND SALARY <= ?
```

produces title headers with values of columns EMPNO and BIRTHDATE from the first row of the query result:



Definition of column headers

User headers

Any number of header definition lines can be defined for a query statement (SELECT). The definition lines are applied in the sequence as written in the script. A header definition line begins with the symbol **`--;H`** after which texts (column titles) delimited by semicolons are entered.

```
--;H [ [text-1] [; text-2] [; ...] [; text-n] [;] ]
```

Texts are taken including leading and trailing spaces. Multiline headers can be aligned or width of displayed columns can be increased this way.

Script EMPLSTSEL_HDRS.sql

```
--EMPLOYEE selected list with multiline headers

--;HEmployee;First       ;Mid. ;Last          ;Work;Phone ;Hire      ;Job       ;Ed.;
Sex;Birth      ;  Salary;Bonus;Commission
--;Hnumber  ;name        ;init.;name          ;dept;number;date      ;          ;lv.;
;date       ;        ;      ;
--;H--------;-----------;-----;--------------;----;------;---------;---------;---;
---;----------;--------;-----;----------
--;H

--;? 1 ; DATE ; From date: ; 1925-01-01 ;
--;? 2 ; DATE ; To date  : ; 1940-01-01 ;
--;? 3 ; DEC  ; Salary does not reach ; 30000.00

SELECT * FROM EMPLOYEE
WHERE BIRTHDATE BETWEEN ? AND ?
      AND SALARY <= ?
```

produces four column headers in the query result, where the last header is empty:

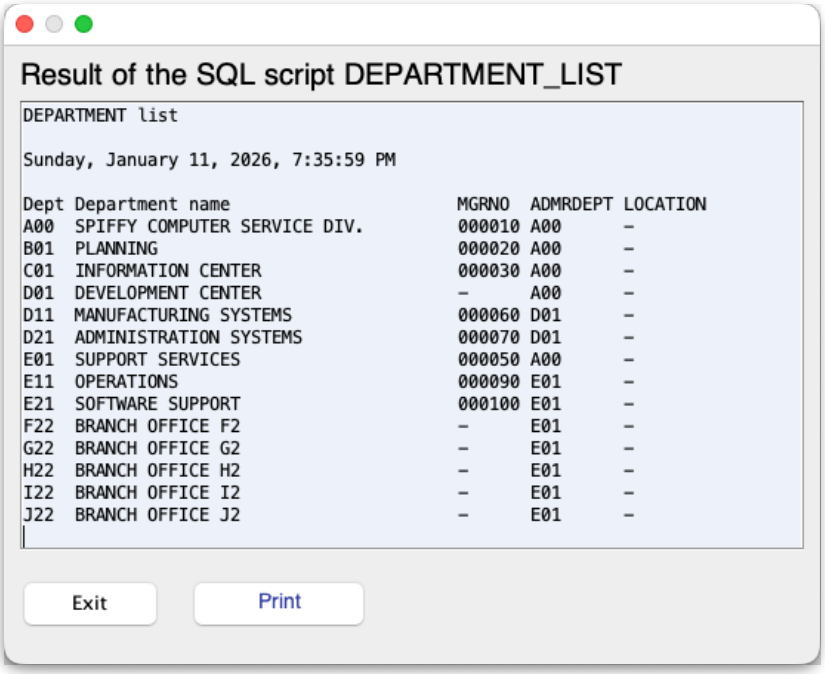
Result of the SQL script EMPLSTSEL_HDRS

```
EMPLOYEE selected list with multiline headers

Sunday, January 11, 2026, 7:28:11 PM

Employee First       Mid.  Last          Work Phone  Hire       Job          Ed. Sex Birth          Salary       Bonus  Commission
number   name        init. name          dept number date                    lv.     date
-------- ----------- ----- ------------- ---- ------ ---------  ---------    --- --- ----------   --------     -----  ----------

000130   DELORES     M     QUINTANA      C01  4578   1971-07-28 ANALYST       16 F   1925-09-15   23800.00     500.00     1904.00
000230   JAMES       J     JEFFERSON     D21  2094   1966-11-21 CLERK         14 M   1935-05-30   22180.00     400.00     1774.00
000250   DANIEL      S     SMITH         D21  0961   1969-10-30 CLERK         15 M   1939-11-12   19180.00     400.00     1534.00
000260   SYBIL       P     JOHNSON       D21  8953   1975-09-11 CLERK         16 F   1936-10-05   17250.00     300.00     1380.00
000280   ETHEL       R     SCHNEIDER     E11  8997   1967-03-24 OPERATOR      17 F   1936-03-28   26250.00     500.00     2100.00
```

Standard headers

If there is not any header definition line, the *standard column headers* from the SELECT statement are displayed. The headers are the column names defined in the table resulting from the query. The standard column header may be the original but also renamed column name (simple or enclosed in quotes). All headers in the script DEPARTMENT_LIST.sql are standard.

```
--DEPARTMENT list
select DEPTNO AS "Dept",
  DEPTNAME AS "Department name",
  MGRNO,
  ADMRDEPT,
  LOCATION
from DEPARTMENT
```



Definition of vertical and horizontal division

Only one comment definition line for groups should be specified. If there are more lines, only the first is accepted. The definition begins in the first position of the line by four-character symbol **`--;T`** and is followed by entries delimited by a semicolon. They define

- number of empty lines *before* a line or a group of lines *n* or `B`*n*,
- number of empty lines after a line or a group of lines *n* or `A`*n*,
- symbol replacing the null value of the column,
- number of spaces delimiting columns in the output line `S`*n* - from 0,
- one or more column names for suppressing their duplicated values.

```
--;T [ [spaces-before] [; spaces-after] [; null-mark]
       [; spaces-between-columns] [; column-name] ... ] [;]
```

Parameter *spaces-before* has the form **Bn** or **n**, where n is an integer. Parameter *spaces-after* has the form **An** or **n**. Parameter *spaces-between-columns* has the form **Sn** or **n**. The first two parameters may be empty. If one of them is empty or invalid, *zero* is taken. If the parameter *null-mark* or *spaces-between-columns* is empty or invalid, the corresponding value from the application parameters is taken.

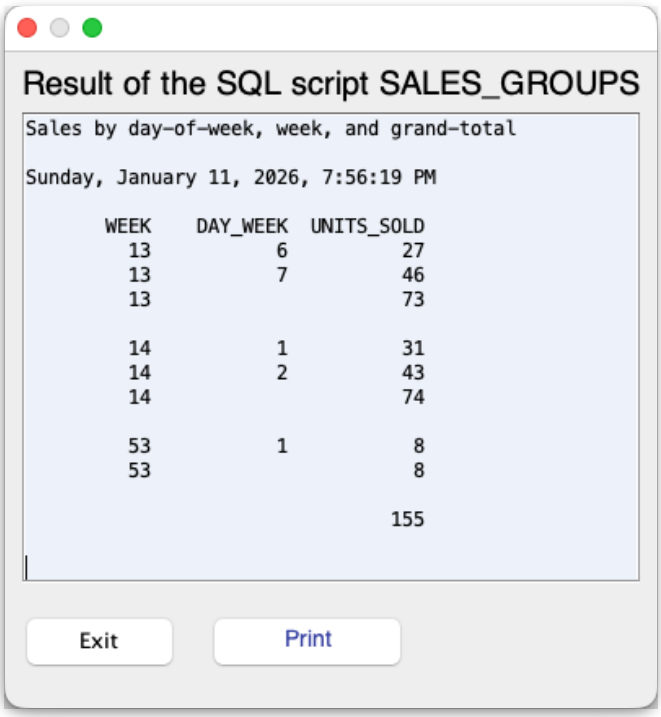
The fifth and following parameters specifies the *column name*, that is checked if its value has changed in a group of contiguous rows. The column value will be printed on the first line of such a group while the next lines will contain spaces.

Note: Empty lines are inserted in the query result before or after the lines or line groups resulting from group levels (see below). Empty lines are also inserted before or after individual lines containing any columns with empty (NULL) values when no group levels are not defined.

Script SALES_GROUPS.sql contains the clause – function GROUP BY ROLLUP () that groups columns DAY_WEEK and WEEK.

```
--Sales by day-of-week, week, and grand-total

--;T B0 ; A1 ; -

SELECT WEEK(SALES_DATE) AS WEEK,
  DAYOFWEEK(SALES_DATE) AS DAY_WEEK,
  SUM(SALES) AS UNITS_SOLD
FROM SALES
GROUP BY ROLLUP( WEEK(SALES_DATE), DAYOFWEEK(SALES_DATE) )
ORDER BY WEEK, DAY_WEEK
```

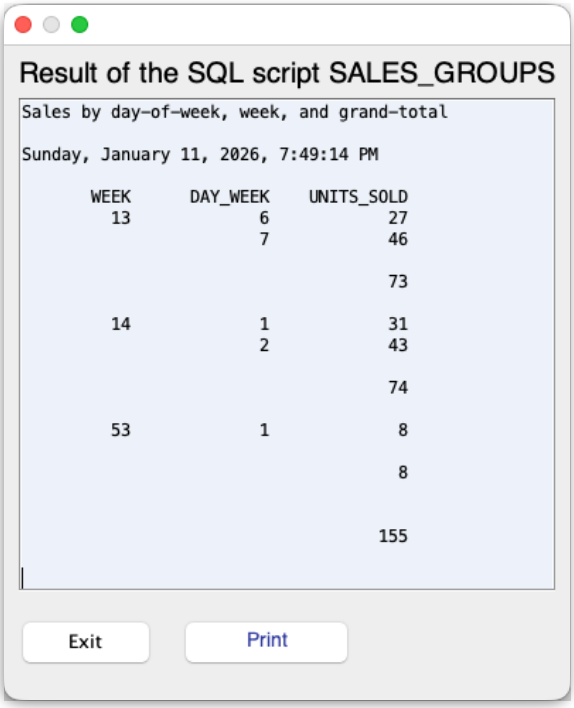
This script produces *one* empty line *after* the data line, while the symbol for the null column value is a *hyphen* (character **-**).



If we change the definition to specify *one* empty line *before* the data line, *one* empty line *after* the data line, *space* for null values, *three column separating spaces*, and column WEEK to *blank out repeated values*:

```
--;T B1 ; A1 ; ; S3 ; week;
```

the result will look like this:



Formatting numbers for output

Decimal numbers of the type DEC, DECIMAL, NUMERIC are displayed and printed using a format. A format is either *standard* for a given localization or prescribed by a *pattern (mask)*.

Note: Patterns do not apply to integer types (INT, INTEGER, BIGINT).

Symbols for creating patterns

Characters comprising a patern are shown in the following table along with their placement and meaning.

Symbol	Location	Localized?	Meaning
0	Number	Yes	Digit
#	Number	Yes	Digit, zero shows as absent
.	Number	Yes	Decimal separator or monetary decimal separator
-	Number	Yes	Minus sign
,	Number	Yes	Grouping separator
%	Prefix or suffix	Yes	Multiply by 100 and show as percentage
¤	Prefix or suffix	No	Currency sign, replaced by currency symbol. If doubled, replaced by international currency symbol. If present in a pattern, the monetary decimal separator is used instead of the decimal separator.
‰	Prefix or suffix	Yes	Multiply by 1000 and show as per mille value
'	Prefix or suffix	No	Used to quote special characters in a prefix or suffix, for example, "'#'#" formats 123 to "#123". To create a single quote itself, use two in a row: "# o''clock".

Warning: Placing 0 between sequence of # and vice versa is an error.

Examples (US localization)

Number	Pattern (mask)	Output	Comment
123456789.50	#,####.00	1,2345,6789.50	Separates groups of 4 digits in the whole part
1234567.50	,###.00 ¤	1,234,567.50 $	Separates groups of 3 digits in the whole part
1234567.50	#.00 ¤¤	1234567,50 USD	Adds the international currency symbol for dollar
1234567.50	#.00 USD	1234567.50 USD	Adds constant “USD” after the number.
1234567.50	¤#.00	$1234567.50	Floating currency symbol
1234567.50	#0**.00	1234567.50**	Asterisks are added after the number with no space
-1234567.55	#0.0	-1234567.6	Rounds to 1 decimal position (to even)
-0.04	#.00	-.04	Suppresses zero before decimal point
5.55	00000.000	00005.550	Does not suppress leading zeros and adds a 0 digit at the end

Number	Pattern (mask)	Output	Comment
0.56	#.00 %	56 %	Expressing the fraction as percentage
0.56	#.00 %	%56	Floating percet sign on the left
0.007	#.0	7,0 ‰	Expressing the fraction as per mille
12345	-#.00	-12345	Floating minus sign on the left. Does not invert positive number to negative.
-12345	-#.00	--12345	If the number is negative adds one more minus sign.

Defining patterns

A pattern may be defined in application Parameters, or in comment definition lines in the script, or both methods may be combined. The pattern defined in Parameters applies for all numeric columns of the script for which no individual definition is defined in a comment definition line.

A *pattern definition line* begins in the first position with the symbol **`--;D`** which is followed by two entries: a pattern and a column name.

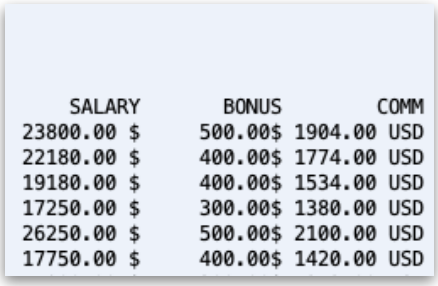
Pattern definition has the following format.

```
--;D pattern ; column-name [;]
```

Example of currency symbols

```
--EMPLOYEE selected list with currency symbols

--;? 1 ; DATE ; From date: ; 1925-01-01 ;
--;? 2 ; DATE ; To date  : ; 1940-01-01 ;
--;? 3 ;      ; Salary does not reach ; 30000.00

--;D #.00 $ ; SALARY
--;D #.00¤ ; bonus
--;D #.00 ¤¤; Comm

SELECT * FROM EMPLOYEE
WHERE BIRTHDATE BETWEEN ? AND ?
      AND SALARY <= ?
```

This script defines three patterns for three columns and produces three differently edited currency values.



```
    SALARY       BONUS        COMM
 23800.00 $     500.00$ 1904.00 USD
 22180.00 $     400.00$ 1774.00 USD
 19180.00 $     400.00$ 1534.00 USD
 17250.00 $     300.00$ 1380.00 USD
 26250.00 $     500.00$ 2100.00 USD
 17750.00 $     400.00$ 1420.00 USD
```

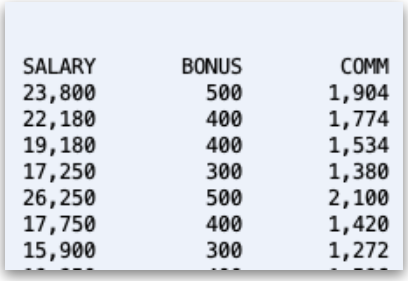
Formatting numbers without pattern

If no pattern is defined neither in application parameters nor in the definition lines, the decimal number is formatted according to *standard rules* in a given locality. For USA these rules apply:

- Whole part of the number is divided into comma separated groups of 3 digits.
- Decimal fraction lacks trailing zeros.
- Zero decimal fraction causes suppression of the decimal point.

Example of standard localization of decimal numbers

If the pattern in the application Parameters is *empty* and no individual pattern definitions are defined in comment definition lines, numbers in the the output of the script EMPLSTSEL in US standard localization will look like this:



SALARY	BONUS	COMM
23,800	500	1,904
22,180	400	1,774
19,180	400	1,534
17,250	300	1,380
26,250	500	2,100
17,750	400	1,420
15,900	300	1,272

Decimal numbers show without fractions and are divided into comma separated groups of 3 digits. Other standard rules apply to percentages, currency symbols etc.

Omitting columns from output

The definition line begins in the position 1 with the symbol **--;O** which is followed by column names delimited by semicolon. Any number of definition lines may be specified.

The definition has the following format.

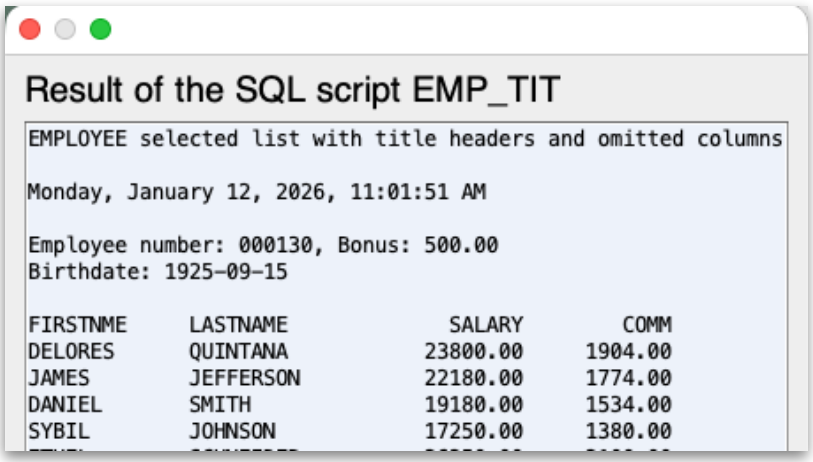
```
--;O [ column-name ; column-name ; ... [;] ]
```

Columns specified in this specification are omitted from the display and print which concerns both values and column headers.

Example with omitted columns and title headers

Script EMP_TIT.OMIT.sql specifies three title headers, two omitted columns along with some question mark parameteres. There can be more rows with **--;O** .

```
--EMPLOYEE selected list with title headers and omitted columns

--;? 1 ; DATE ; From date: ; 1925-01-01 ;
--;? 2 ; DATE ; To date  : ; 1940-01-01 ;
--;? 3 ; DEC ; Salary does not reach ; 30000.00

--;tEmployee number: &EMPNO , Bonus: &BONUS
--;tBirthdate: &BIRTHDATE
--;t

--;O EMPNO ; MIDINIT;
--;O BONUS; BIRTHDATE

SELECT EMPNO, FIRSTNME, MIDINIT, LASTNAME,
       BIRTHDATE, SALARY, BONUS, COMM
   FROM EMPLOYEE
   WHERE BIRTHDATE BETWEEN ? AND ?
         AND SALARY <= ?
```



Omitted columns are not included in the output but their values from the first row are visible in title headers. Editing of BONUS is now performed according to the specification in application parameters which is #.00.

Note: There is no necessity to bind title headers to omitted columns. Both can be used independently.

Print parameters

Only one definition line may be specified. If more are specified, the first one is taken, others are ignored. The definition line begins in the position 1 with the symbol **`--;P`** which is followed by entries delimited by semicolon that define

- paper size `A4`, `A3`, or `LETTER` (not tested on paper),
- font size of the form `FS`*n*, where *n* is number of print points, e. g. `fs9`,
- page orientation `PORTRAIT` or `P`, or `LANDSCAPE` or `L`,
- left margin of the form `LM`*n*, where *n* is number of milimeters from the edge of paper printable area,
- right margin of the form `RM`*n*,
- top margin of the form `TM`*n*,
- bottom margin of the form `BM`*n*,

The definition has the following format.

```
--;P  paper-size; FSn; orientation; LMn; RMn ; TMn ;BMn [;]
```

The line may or may not be ended by semicolon. Individual entries may be omitted but number of semicolons before the first non-empty entry must be retained.

- If paper size is empty or invalid `A4` is taken.
- If font size is empty or non-numeric the value from the application parameters is taken.
- If orientation is empty or invalid `PORTRAIT` is taken.
- If a margin is empty or invalid, `10` is taken (10 mm).

Summarization of query result

The result of a query (the result set) can be summarized according to group levels (level of summarization). Number of levels is not explicitly limited. The group summarization is defined by the set of three parametric definitions:

`--;L` group level, its leading text and the name of its group column,
`--;S` name of the summarized column and a list of summarization types (S, A, M, m, C),
`--;s` list of leading texts for summary lines (instead of standard SUM, AVG, MAX, MIN, COUNT),

Number of group levels is defined by the number of **`--;L`** lines. Number of summarized columns is defined by the number of **`--;S`** lines. At most one **`--;s`** line can be specified. Lines `--;S` and `--;s` have sense only when some `--;L` definitions are specified.

Definitions of group levels

The definition line begins in the position 1 with the symbol **`--;L`** which is followed by entries delimited by semicolon.

The definition has the following format.

```
--;L [level]; [group-leading-text]; [level-column]; [NP] [;]
```

The first entry *level* is either 0 or any text, even empty. Usually 1, 2, etc. are chosen.

The second entry *group-leading-text* is any text containing optional *variables* of the form `&column` followed by a space or end of line. The variable is replaced by the column value from the *last line* of the group even if the column is specified as omitted in the parameter `--;O`.

The third entry *level-column* represents the column defining the group of given level. This entry has no meaning in level 0.

The fourth entry NP (New Page) is applied when printing on paper. If the group with this entry changes, the first row of the new group is printed on the next page.

Entry of `--;L0` defines the last level called final total or grand total. `L0` specification line is optional but if present it must be the first (or the only) one. If `L0` is not present the last level is not processed. The *level* entry in other ("lower") group levels will usually be the serial number of the level but it can be any text. For example, the following specification lines define two group levels L2 and L1, and the last level L0.

```
--;L0 ;*** GRAND TOTAL ***
--;L1 ;=== &SALES_PERSON  ===          ; SALES_PERSON
--;L2 ;--- &SALES_PERSON  --- &REGION ; REGION
```

Hierarchy of levels is determined by the sequence of specification lines, not by the number at the letter L. The last line defines the lowest level.

Definitions of summarized columns

In order to summarize (accumulate), colum names with their summarizing type(s) must be specified along with some group levels (L0, L1, ...).

One or more definition lines `--;S` is needed to summarize column values. The definition lines need not be specially ordered. Each definition line begins in the position 1 with the symbol **`--;S`** which is followed by entries delimited by semicolon. The entries define a

column name and the type of summarization of its values. The definition has the following format.

```
--;S [summarized-column; [; ind] [; ind] [; ind] [; ind] [; ind] [;]]
```

where *ind* is one of the symbols – indicators S, A, M, m, C. They represent summarization types *sum*, *average*, *maximum*, *minimum* and *count*. These indicators need not be entered in any specific order and they need not be specified at all. Summary lines in the query result are ordered in this specific order: S - sum, A - average, M - maximum, m - minimum, C - count).

If empty or invalid value is in one of indicator's positions, it is assumed not being specified.

If no specification line `--;S` is present, no level processing is done even if specifications `--;L` are present and vice versa.

All five types of summarization can be done for columns of types DECIMAL, NUMERIC, INTEGER, SMALLINT a BIGINT. Only maximum, minimum and count can be used for types CHAR, VARCHAR, VARGRAPHIC, DATE, TIME, and TIMESTAMP.

Null values are excluded from summarization.

Note: If, by mistake, sum (S) or average (A) is specified for *non-numeric* column, no value is computed and printed but the line with the summary type indication (see below) is printed (even if no *numeric* column is summarized).

For example, the specification

```
--;S SALES_DATE ; M ; m
--;S SALES ; S ; A ; M ; m ; C
```

define two summarized columns, where cloumn SALES_DATE is evaluated for maximum (M) and minimum (m), and column SALES is evaluated for sum (S), average (A), Maximum (M), minimum (m), and count (C).

Summary type indications

Standard summary type indications are SUM for sum, AVG for average, MAX for maximum, MIN for minimum, and COUNT for count. These indications can be replaced by different texts using specification line `--;s`.

The definition line begins in the position 1 with the symbol **`--;s`** which is followed by entries delimited by semicolon.

The definition has the following format.

```
--;s [ [sum-text]; [avg-text]; [max-text]; [min-text]; [count-text] [;] ]
```

where the texts from corresponding positions replace standard indications. If the text is missing in a position, standard indication for the position is taken.

For example, the specification line

```
--;s Sum of sales: ; Average: ; Maximum: ; Minimum: ;  Number of days
```

replaces all standard indications.

Example

The following script specifies user headers (`--;H`), vertical structure (`--;T`), three group levels (`--;L`) over columns REGION, SALES_PERSON and final total, and one summarized column SALES with all possible summary types. Specification `--;s` changes standard summary indications.

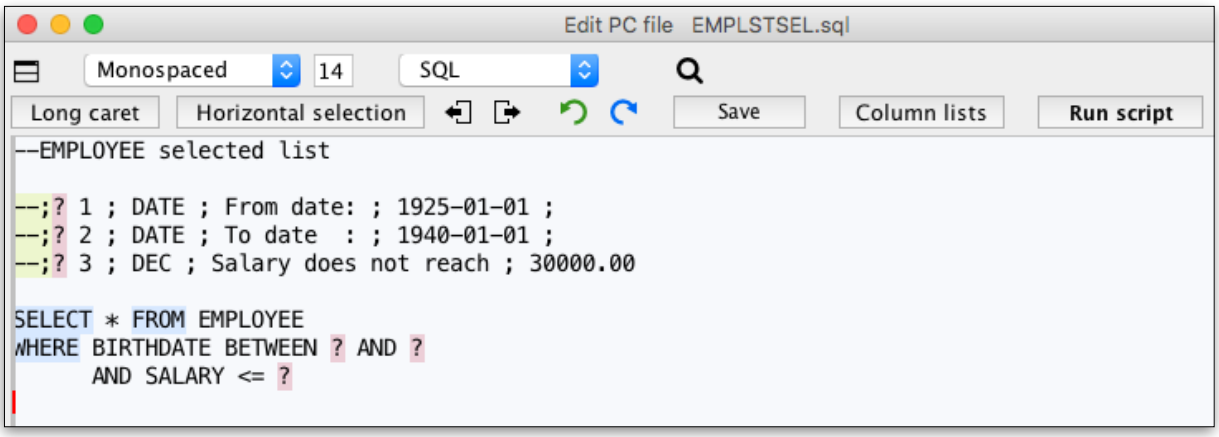
```
--Sales summarized by person and region

--;HSales date;Sales person;Region;Sales
--;H----------;------------;------;-----
--;H

--;T 1; 1; null ;  ; SALES_PERSON; REGION

--;L0 ;*** GRAND TOTAL ***;
--;L1 ;=== &SALES_PERSON  ===         ; SALES_PERSON
--;L2 ;--- &SALES_PERSON  --- &REGION ; REGION

--;S SALES ; S ; A ; M ; m ; C

--;s Sum of sales: ; Average: ; Maximum: ; Minimum: ;  Work shifts:

SELECT  SALES_DATE , SALES_PERSON , REGION , SALES
FROM SALES
ORDER BY
    SALES_PERSON , REGION , SALES_DATE
```

The script produces the following result.

```
Sales date   Sales person      Region                  Sales
----------   ------------      ------                  -----

1996-03-29   GOUNOT            Manitoba                    7
1996-03-30                                                 1
1996-04-01                                                 7

--- GOUNOT --- Manitoba
                               Sum of sales:              15
                                    Average:               5
                                    Maximum:               7
                                    Minimum:               1
                                Work shifts:               3

1996-04-01                      Ontario-North              1

--- GOUNOT --- Ontario-North
                               Sum of sales:               1
                                    Average:               1
                                    Maximum:               1
                                    Minimum:               1
                                Work shifts:               1

1996-03-29                      Ontario-South              3
1996-03-30                                                 2
1996-03-31                                                 2
1996-04-01                                                 3


... etc.


1995-12-31                      Ontario-South              1
1996-03-29                                                 3
```

```
1996-03-30                                          1
1996-04-01                                          3

--- LUCCHESSI --- Ontario-South
                              Sum of sales:         8
                                   Average:         2
                                   Maximum:         3
                                   Minimum:         1
                               Work shifts:         4

1996-03-29                     Quebec                1
1996-03-30                                           2

--- LUCCHESSI --- Quebec
                              Sum of sales:         3
                                   Average:         1
                                   Maximum:         2
                                   Minimum:         1
                               Work shifts:         2

=== LUCCHESSI ===
                              Sum of sales:        14
                                   Average:         1
                                   Maximum:         3
                                   Minimum:         1
                               Work shifts:         9

*** GRAND TOTAL ***
                              Sum of sales:       155
                                   Average:         3
                                   Maximum:        18
                                   Minimum:         1
                               Work shifts:        40
```

Editing scripts



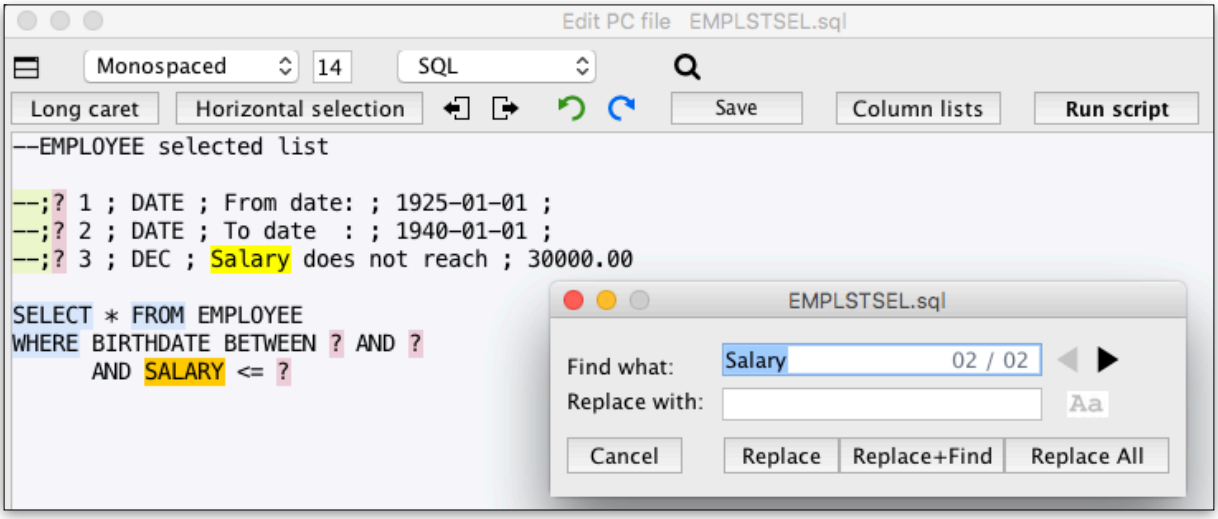
- *Split/unsplit* - toggle split/unsplit editor text vertically by a movable horizontal line into upper and lower area. A copy of the editor text is in the lower area. All text changes made in one area are automatically projected in the other area.
- *Lucida Console etc.* – choose a font and its size from the combo box.
- *SQL* – choose from the combo box to highlight SQL statement keywords and special comments or choose **NONE* not to highlight anything.
- *Find text* – invokes a window to find text. Shortcut Ctrl F may also be used.
- *Long caret/Short caret* – defines the pointer in text as a long vertical line or a short vertical line (standard).
- *Horizontal/Vertical selection* – defines method of selecting text. Horizontal selection is a common method in PC editors. Vertical selection selects a rectangular area in the text.
- *Shift selection* – button shifts selected text one position to the left, button shifts selected text to the right. Keyboard shortcuts Ctrl ← and Ctrl → may also be used.
- *Undo* – remove changes. Keyboard shortcut Ctrl Z may also be used.
- *Redo* – restore changes. Keyboard shortcut Ctrl Y may also be used.
- *Save* – save changes. Keyboard shortcut Ctrl S may also be used.
- *Column lists* – invokes a window where after selecting a specific schema and a table (view) the colums are listed. The column list may be modified and edited (adding commas), and copied to the editor area.
- *Run script* – runs the SQL script.

ESC key escapes editing (without saving) and removes the window.

Note: *Cmd* key is used instead of Ctrl in macOS.

Finding text

Clicking on the magnifying glass 🔍 button or pressing keyboard shortcut Ctrl F (Cmd F in macOS) invokes a window to find text.



- *Find what* – enter a text pattern to be found in the editor area. Numbers in the field show sequence number and number of matches.
- ◀ ▶ *Arrow buttons* – find preceding or following matches in the file. An arrow gets dark on clicking on it and indicates direction of searching and replacing. The opposite arrow gets gray. The same function is provided by keyboard shortcuts Ctrl ↑(previous) and Ctrl ↓ (next).
- *Replace with* – enter a text replacement.
- Aa/**Aa** *toggle button* – when light gray, the search is case insensitive, when black, the found text must exactly match the the pattern.
- *Replace* – replace the text just found by the replacement
- *Replace+Find* – replace the matching text just found by the replacement and find the *next* matching text.
- *Replace All* – replace *all* matching texts by the replacement text.

Note: *Cmd* key is used instead of Ctrl in macOS.

Shifting selected text

Left and Right buttons shift the selected text left or right *by one position*. The keyboard shortcuts Ctrl ← and Ctrl → do the same functions. Results of shifts may be undone or redone.

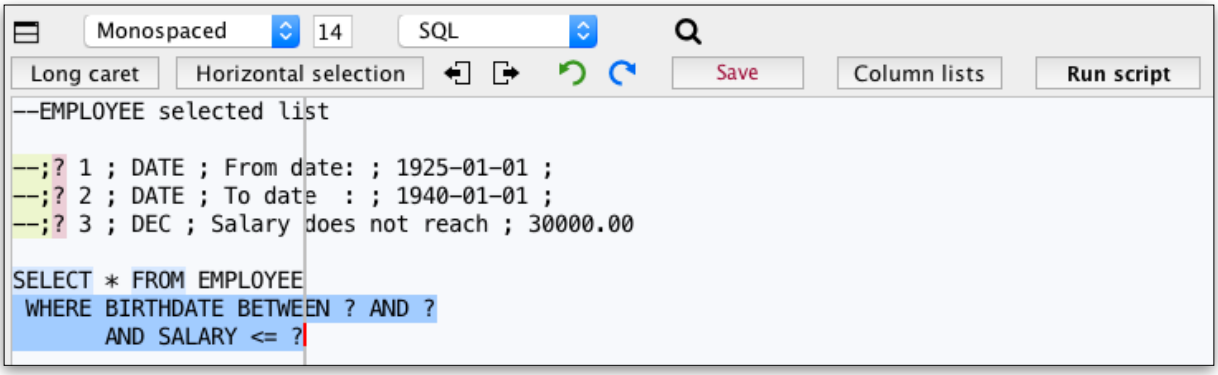
Note: *Cmd* key is used instead of Ctrl in macOS.

Horizontal selection

The selection shifts *right* by one position.

The selection shifts *left* along with the rest of the lines if it begins at the start of a line and if all lines of the selection contain at least one space at the start.

In the followin picture, the selection was shifted right.

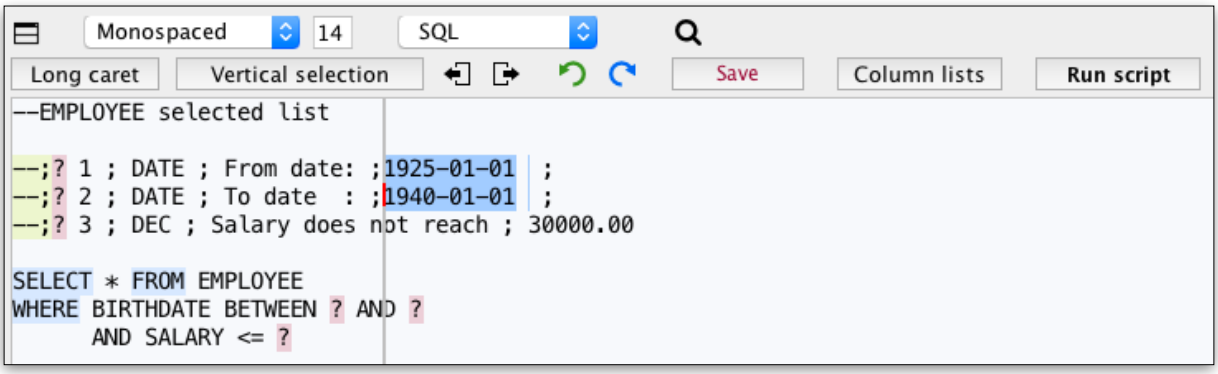


Vertical selection

The selected *rectangle* shifts *left* if there remains at least one column of spaces in the unselected part on the left of the rectangle. The rectangle *overwrites* the columns on the left and leaves a column of spaces behind (on the right).

The selected *rectangle* shifts *right* along with the rest of the lines and leaves a column of spaces behind (on the left).

In the following picture, the rectangle was shifted left.



Copy, cut and paste selected text

Common command shortcuts Ctrl **C**, Ctrl **X**, Ctrl **V** are used to copy, cut and paste selected text.

Copy and Cut operations store the selected text in the *operating system clipboard*.

Paste operation reads data from the clipboard and inserts it to the desired place. This may be in the editor area or elsewhere in PC.

Results of these operations may be undone or redone.

Note: *Cmd* key is used instead of Ctrl in macOS.

Horizontal selection

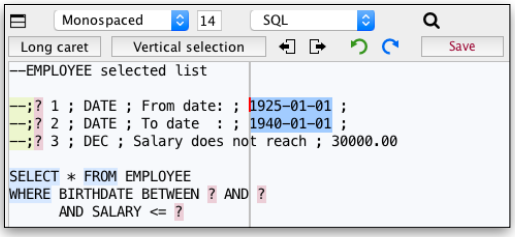
All these operations work as usual in PC.

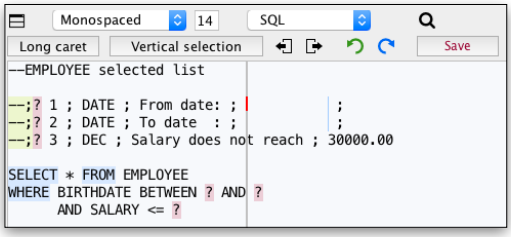
Vertical selection

Copy operation copies the selection into an internal area and also into the system clipboard.

Cut operation copies the selection into an internal area and also into the system clipboard, then *clears* (puts spaces in) *the rectangle area*.

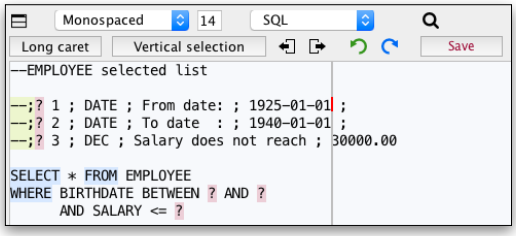
The rectangle from the left picture was Cut:



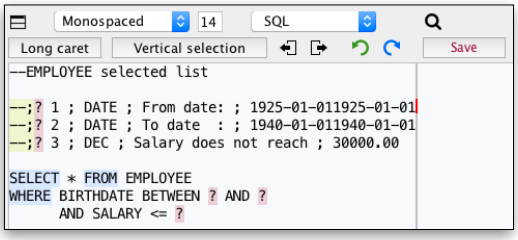


Note the *caret* position, it stands at the *beginning* of the cut rectangle area. The blue line at right denotes the right edge of the rectangle. If desired, a following *paste operation* inserts the erased data back into its original positions.

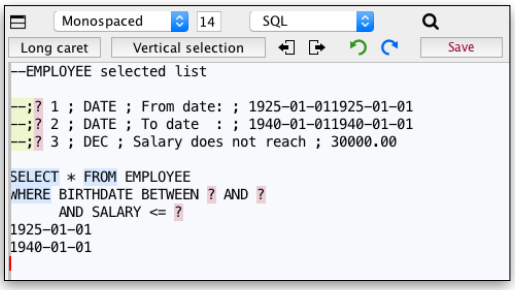
Paste operation replaces the editor area right and down from the *caret position*. It may be the area originally copied or cut:



This time the *caret* stands at the *right top* edge of the rectangle just pasted. An immediate following paste operation inserts data on the right of the caret:



If the editor area is shorter than the inserted rectangle, extra empty lines are appended and the pasted data is inserted in the extra lines:



Column lists

This function makes editing scripts easier by retrieving column lists from IBM i objects.

Button *Column lists* invokes a window, where the user selects a schema and a table (CORPDATA and EMPLOYEE in the pictures).

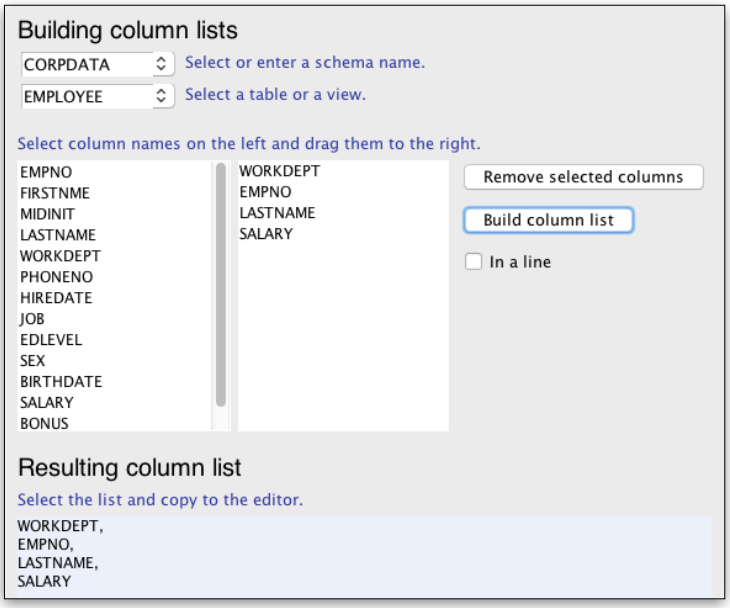
This action fills the *left frame* with the list of column names taken from the table (EMPLOYEE).

The lower part of the window – "Resulting column list" – now contains the column names with commas between and with each name on a separate line. This list, as a whole, or its selected parts, can be selected and directly *copied* or *dragged* to the editor area.

Removing selected columns

Selected names from the right frame are removed using button *Remove selected columns*.

Building the column list



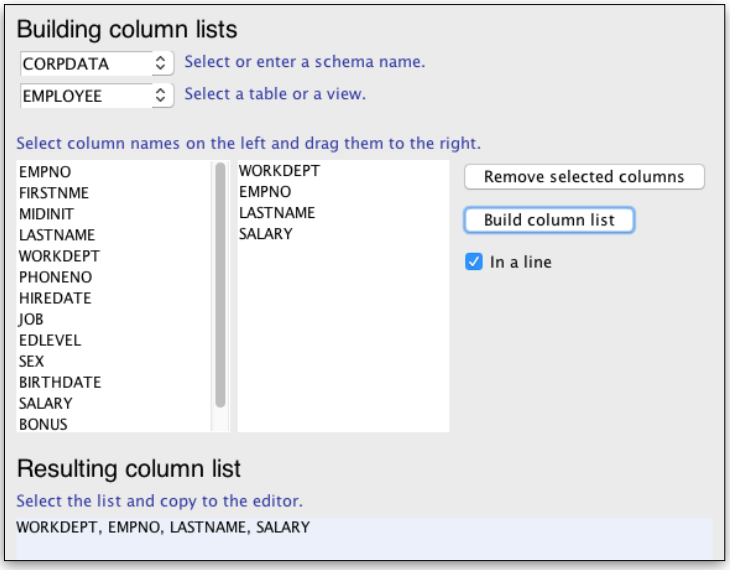
Names can be draged from the left frame and dropped to the *right frame*.

Button *Build column list* produces an edited column list in the frame *Resulting column list* from the names contained in the *right frame*. The list is arranged in separate lines if the checkbox *In a line* is unchecked.

This list, as a whole, or its selected parts, can be selected and directly *copied* or *dragged* to the editor area.

Building the column list in a line

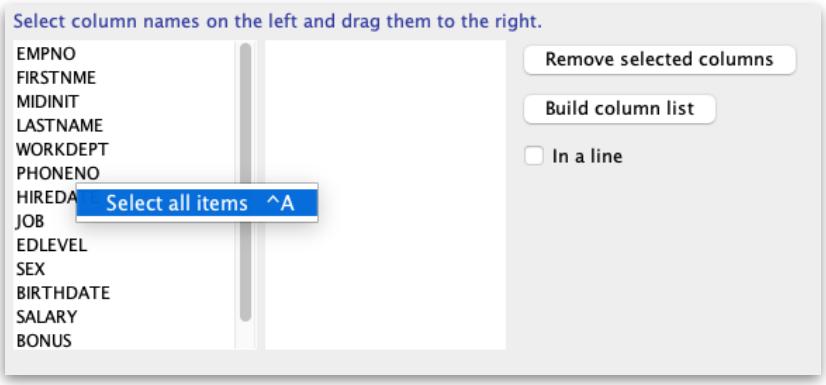
Check box *In a line* builds the edited list with all names in one line.



This list, as a whole, or its selected parts, can be selected and directly *copied* or *dragged* to the editor area.

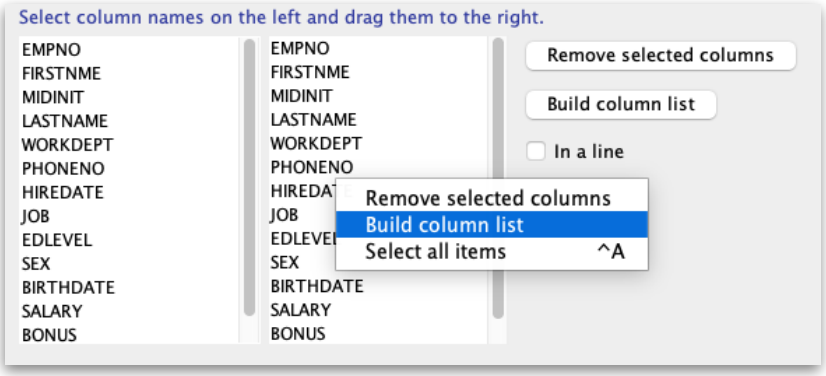
Selection of all names in the left and right frame

- Right click on the frame and select the command *Select all items* in the popup menu.
- Press the keyboard shortcut Ctrl A (Cmd A in macOS).



Popup menu on the right frame

- *Remove selected columns*. Acts the same as the button of the same name.
- *Build column list*. Acts the same as the button of the same name.
- *Select all items*. Acts the same as the keyboard shortcut Ctrl A (Cmd A in macOS).



Processing members of physical and logical files

An example will illustrate how to work with data members of physical and logical files. In Logical file can also be processed with all logical members as a whole.

Files are created using DDS (Data Description Specifications) because tables created by SQL statements CREATE TABLE and CREATE VIEW do not enable to add or remove data members.

Creation of physical and logical file with members

Create physical file PRICES.

```
 ****************************************
 *   File PRICES - Item prices
 ****************************************
A                                     UNIQUE
A          R PRICESR
 *   Item number
A            ITEMNBR        5A
 *   Unit price
A            UNITPR         9P 2
 *   Item description
A            DESCR         50A
 *   Key - Item number
A          K ITEMNBR
```

Create logical file PRICESL.

```
 ************************************************
 *   File PRICESL
 *   Logical file
 ************************************************
A          R PRICESR                  PFILE(PRICES)
A            ITEMNBR
A            UNITPR
A          K ITEMNBR
```

Define maximum number of members in file PRICES.

```
CHGPF      FILE(VZTOOL/PRICES) MAXMBRS(12)
```

Add three members to physical file PRICES.

```
ADDPFM     FILE(VZTOOL/PRICES) MBR(PRICES_01)
ADDPFM     FILE(VZTOOL/PRICES) MBR(PRICES_02)
ADDPFM     FILE(VZTOOL/PRICES) MBR(PRICES_03)
```

Define maximum number of members in logical file PRICESL.

```
CHGLF  FILE(PRICESL) MAXMBRS(12)
```

Add three members also to logical file PRICESL.

```
ADDLFM FILE(VZTOOL/PRICESL) MBR(PRICESL_01) DTAMBRS((VZTOOL/PRICES(PRICES_01)))
ADDLFM FILE(VZTOOL/PRICESL) MBR(PRICESL_02) DTAMBRS((VZTOOL/PRICES(PRICES_02)))
ADDLFM FILE(VZTOOL/PRICESL) MBR(PRICESL_03) DTAMBRS((VZTOOL/PRICES(PRICES_03)))
```

Script to create alias objects and insert data in members

Script PRICES_ALIASES.sql drops alias objects (if they exixt) and creates new ones, for each member of physical and logical file, named like the members.

```
/* Remove alias objects for the physical and logical file. */
DROP ALIAS VZTOOL/PRICES_01 ;
DROP ALIAS VZTOOL/PRICES_02 ;
DROP ALIAS VZTOOL/PRICES_03 ;
DROP ALIAS VZTOOL.PRICESL_01 ;
DROP ALIAS VZTOOL.PRICESL_02 ;
DROP ALIAS VZTOOL.PRICESL_03 ;

/* Create alias objects for members of physical file PRICES. */
CREATE ALIAS VZTOOL.PRICES_01 FOR VZTOOL.PRICES(PRICES_01) ;
CREATE ALIAS VZTOOL.PRICES_02 FOR VZTOOL.PRICES(PRICES_02) ;
CREATE ALIAS VZTOOL.PRICES_03 FOR VZTOOL.PRICES(PRICES_03) ;

/* Create alias objects for members of logical file PRICESL. */
CREATE ALIAS VZTOOL.PRICESL_01 FOR VZTOOL.PRICESL(PRICESL_01) ;
CREATE ALIAS VZTOOL.PRICESL_02 FOR VZTOOL.PRICESL(PRICESL_02) ;
CREATE ALIAS VZTOOL.PRICESL_03 FOR VZTOOL.PRICESL(PRICESL_03) ;

/* Delet records from physical file PRICES. */
DELETE FROM PRICES_01 ;
DELETE FROM PRICES_02 ;
DELETE FROM PRICES_03 ;

/* Insert records to two members of physical file PRICES. */
INSERT INTO PRICES_01 values ('00001', 8.99, 'Chocolate cakes') ;
INSERT INTO PRICES_01 values ('00002', 459.00, 'Tooth paste Kalodont') ;
INSERT INTO PRICES_01 values ('00003', 1.25, 'Washing line') ;

INSERT INTO PRICES_02 values ('00004', 10.50, 'Men''s socks black') ;
INSERT INTO PRICES_02 values ('00005', 120.00, 'T-shirt white') ;
INSERT INTO PRICES_02 values ('00006', 10.55, 'Men''s socks white, new')
```

ALIAS objects are created, that have type *FILE and attribute DDMF (Distributed Data Management File). DDMF file object enables accessing remote location from local location. It is an object of architecture SNA, APPC. For us, both locations are identical and the object serves as a medium to access data file member.

```
Opt  Object      Type        Attribute   Text
 _   PRICES      *FILE       PF-DTA      PRICES
 _   PRICES_01   *FILE       DDMF
 _   PRICES_02   *FILE       DDMF
 _   PRICES_03   *FILE       DDMF
 _   PRICESL     *FILE       LF          PRICESL
 _   PRICESL_01  *FILE       DDMF
 _   PRICESL_02  *FILE       DDMF
 _   PRICESL_03  *FILE       DDMF
```

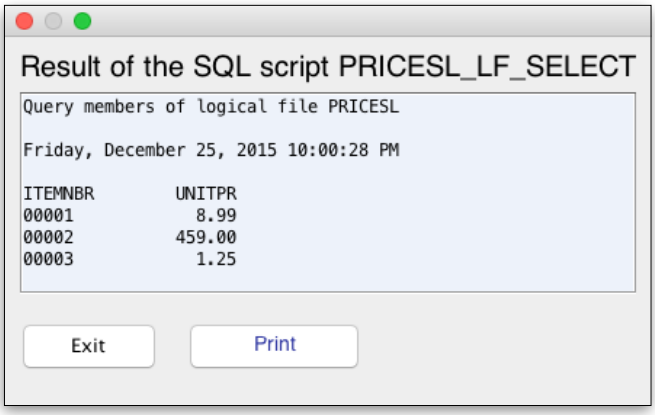
Query data members of the logical file

Queries are specified in the following script to check result of preceding script.

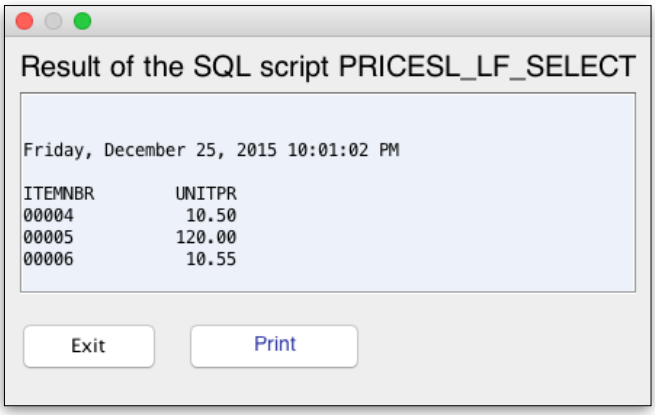
```
--Query members of logical file PRICESL

SELECT * FROM PRICESL_01;
SELECT * FROM PRICESL_02;
SELECT * FROM PRICESL_03;
SELECT * FROM PRICESL
```

The following windows are shown with contents of the logical members.

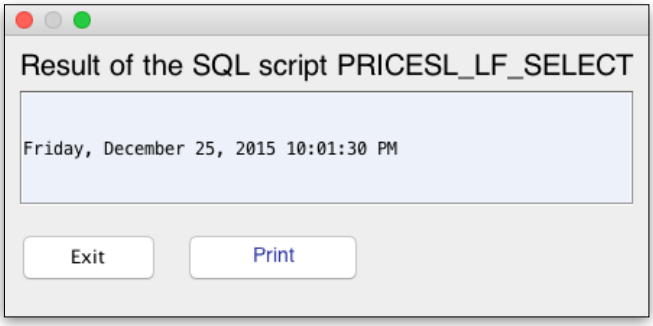
Member PRICESL_01:



Member PRICESL_02:



Member PRICESL_03 (empty):



All members together (note that the standard member PRICESL was not removed):