

Dictionary

of commands and parameters

of SQLfast languages

Last updated on December 2022

This document is also available as an integrated help document that can be opened through menu **Help** > **SQLfast references** > **SQLfast commands** at the **Expert** level.

Part 1 - SQLfast statements summary

- **--**: comment line
- **\$var\$**: variable/value substitution operator; replaces variable name with the value of the variable
- **svar\$**: alternative variable/value substitution operator; replaces variable name with the value of the variable in which single quotes (*apostrophes*) have been doubled
- **abortDB**: kills the current transaction
- **ask/askData**: opens a data entry dialog box
- **askText**: opens a text entry dialog box
- **askCombo**: opens a composite dialog box
- **b64encode α To β** : encodes a character string in *standard base64*; α and β are either **Var** or **File**, depending on the nature of the source and target containers
- **b64encode α To β -u**: encodes a character string in *url-safe base64*

- **b64decodeToß**: decodes the value of a variable in *base64* (*standard* or *url-safe*)
- **BfileToVariable**: store the contents of a binary file into a variable
- **closeDB**: closes the current database
- **closeHelp**: closes the help window
- **closeTextFile**: closes a text file
- **commitDB**: commits the current transaction
- **compute**: computes an expression and stores its value in the target variable
- **continue**: no operation
- **copyFile**: copies a file
- **createDB**: creates a new empty database
- **createOrReplaceDB**: creates a new empty database or replaces an existing one
- **createDictionary**: creates the dictionary of the current database
- **deleteDB**: deletes a database
- **deleteDictionary**: deletes the dictionary of the current database
- **deleteFile**: deletes a file
- **dirNames**: writes in a text file the names of sub-directories in a directory
- **dirNamesAll**: recursively writes in a text file the names of sub-directories in a directory
- **dirFileName**: writes in a text file the names of files and sub-directories in a directory
- **dirFileNamesAll**: recursively writes in a text file the names of files and sub-directories in a directory
- **displayHelp**: opens a help window and displays the contents of a help file
- **dropVariable**: deletes a user variable
- **dropAllVariables**: deletes all user variables
- **error**: triggers a user defined error
- **execSQL**: executes an SQLfast script
- **execApp**: executes a Python program or a function within a Python program. Same as **function**.
- **execProg**: executes an external program.
- **exit**: exits from a loop
- **extract**: gets data from the database
- **fastExec**: executes a pure SQL script. Statements are not checked.
- **fileNames**: writes in a text file the names of files in a directory
- **fileNamesAll**: recursively writes in a text file the names of files in a directory

-
- **fileToVariable**: stores in a variable the contents of a file or a line from a text file
 - **for, for-endfor**: *range, SQL, file* or *list* loop
 - **function**: executes a Python program or function. Same as **execApp**.
 - **generic**: executes a non standard SQL query
 - **goto**: continues the execution at a named point of the script
 - **hashFile**: compute the hash value of a file
 - **HEXdecode α To β** : decodes a *hexadecimal* character string into a byte string
 - **HEXencode α To β** : encodes a byte string into a *hexadecimal* character string; α and β are either **Var** or **File**, depending on the nature of the source and target containers
 - **if, if-endif, if-else-endif**: alternative statement
 - **import**: loads the data from an external file in a table
 - **insertLOB**: SQL-like query to insert a large object in an existing row.
 - **label**: defines a named point in the script
 - **next**: forces the next iteration in a **for** or **while** loop
 - **nextTextLine**: reads next line in a text file
 - **onError**: defines the reaction of the interpreter when an error occurs
 - **openDB**: opens a database
 - **openTextFile**: opens a text file
 - **output**: enable/disable writing on the current output channel
 - **outputOpen, outputAppend**: specifies the current output channel and its writing mode
 - **parameter commitmode**: defines the default commit mode: autocommit or transaction
 - **parameter delimiters**: defines the variable/value substitution operator delimiters
 - **parameter label-separator**: defines the symbol separating two successive labels in dialog boxes
 - **parameter select-align**: tells whether column values are aligned in **select** statements
 - **parameter styleXXX**: defines various style aspects in dialog boxes
 - **parameter colorXXX**: defines colors in dialog boxes
 - **pause**: same as **showMessage**
 - **print**: same as **write** but always writes in the output window
 - **renameFile**: renames a file
 - **replace**: replaces in a variable value all instances of a substring with another one

- **return**: terminates the execution of the current (sub-)script
- **selectFile**: opens a file selection dialog box
- **selectDirectory**: opens a directory selection dialog box
- **selectList**: opens a simple list box
- **selectList2**: opens a dual list box
- **selectMany**: opens a checkbox dialog box
- **selectOne**: opens a radio button dialog box
- **selectList**: opens a simple list box
- **selectList2**: opens a dual list box
- **set**: assigns a value to a variable
- **set-n**: same as set with variable/value substitution disabled
- **showMessage**: displays a message in a dialog box; same as **pause**
- **showData**: displays data in a dialog box similar to that of **askData**
- **showText**: displays a text in a dialog box similar to that of **askText**
- **showPicture**: displays a picture from a gif file
- **showDrawing**: displays a still or animated SQLdraw drawing
- **shutDown**: terminates the execution of all scripts and stops the SQLfast interpreter
- **start-timer**: resets the timer
- **stop**: terminates the execution of the current main script (and of all its sub-scripts)
- **trace**: enable/disable statement execution tracing
- **substitution**: enables/disables variable/value substitution
- **variableToFile**: stores the value of a variable in a file
- **wait**: suspends script execution for a fixed or random time duration
- **webfileToFile**: download a file from the web
- **webpageToVariable**: transfer the HTML code of a web page into a variable
- **while, while-endwhile**: condition-driven loop
- **write**: write a character string on the current output channel
- **writeVariables**: writes the values of all user variables on the current output channel
- **writeSystemVariables**: write the values of all system variables on the current output channel
- **writeParameters**: writes the values of all parameters on the current output channel
- **SQL queries**:
 - **create**: SQL create query

- **alter**: SQL alter query
- **drop**: SQL drop query
- **insert**: SQL insert query
- **update**: SQL update query
- **delete**: SQL delete query
- **select**: SQL select query
- **with select**: SQL CTE-based select query
- **with insert**: SQL CTE-based insert query
- **with update**: SQL CTE-based update query
- **with delete**: SQL CTE-based delete query

Part 2 - SQLfast statements by category

SQL statements

Standard SQL queries

1. **create**: SQL create query
2. **alter**: SQL alter query
3. **drop**: SQL drop query
4. **select**: SQL select query
5. **insert**: SQL insert query
6. **update**: SQL update query
7. **delete**: SQL delete query
8. **with select**: SQL CTE-based select query
9. **with insert**: SQL CTE-based insert query
10. **with update**: SQL CTE-based update query
11. **with delete**: SQL CTE-based delete query

Pseudo-SQL statements

Additional, non standard, statements related to database management

1. **createDB**: creates a new empty database
2. **createOrReplaceDB**: creates a new empty database or replaces an existing one
3. **deleteDB**: deletes a database
4. **openDB**: opens a database
5. **closeDB**: closes the current database
6. **abortDB**: kills the current transaction
7. **commitDB**: commits the current transaction
8. **parameter commitmode**: defines the default commit mode: autocommit or transaction
9. **generic**: executes a non standard SQL query
10. **createDictionary**: creates the dictionary of the current database
11. **deleteDictionary**: deletes the dictionary of the current database
12. **extract**: gets data from the database
13. **insertLOB**: SQL-like query to insert a large object in an existing row.
14. **import**: loads the data from an external file in a table

15. **fastExec**: executes a pure SQL script. Statements are not checked.

Variable assignment statements

Statements that create a variable or affect its contents.

1. **set**: assigns a value to a variable
2. **set-n**: same as set with variable/value substitution disabled
3. **compute**: computes an expression and stores its value in the target variable
4. **extract**: gets data from the database
5. **for**, **for-endfor**: *range*, *SQL*, *file* or *list* loop
6. **replace**: replaces in a variable value all instances of a substring with another one
7. **nextTextLine**: reads next line in a text file
8. **fileToVariable**: stores in a variable the contents of a file or a line from a text file
9. **BfileToVariable**: store the contents of a binary file into a variable
10. **webpageToVariable**: transfer the HTML code of a web page into a variable
11. **ask/askData**: opens a data entry dialog box
12. **askText**: opens a text entry dialog box
13. **askCombo**: opens a composite dialog box
14. **selectMany**: opens a checkbox dialog box
15. **selectOne**: opens a radio button dialog box
16. **HEXdecode α To β** : decodes a *hexadecimal* character string into a byte string
17. **HEXencode α To β** : encodes a byte string into a *hexadecimal* character string; α and β are either **Var** or **File**, depending on the nature of the source and target containers
18. **b64encode α To β** : encodes a character string in *standard base64*; α and β are either **Var** or **File**, depending on the nature of the source and target containers
19. **b64encode α To β -u**: encodes a character string in *url-safe base64*
20. **b64decode α To β** : decodes the value of a variable in *base64 (standard or url-safe)*
21. **hashFile**: compute the hash value of a file
22. **execApp**: executes a Python program or a function within a Python program. Same as **function**.
23. **function**: executes a Python program or function. Same as **execApp**.

24. **outputOpen, outputAppend**: specifies the current output channel and its writing mode
25. **start-timer**: resets the timer
26. **dropVariable**: deletes a user variable
27. **dropAllVariables**: deletes all user variables

User interface

Statements that allow a script to interact with the user.

1. **ask/askData**: opens a data entry dialog box
2. **askText**: opens a text entry dialog box
3. **selectList**: opens a simple list box
4. **selectList2**: opens a dual list box
5. **selectOne**: opens a radio button dialog box
6. **selectMany**: opens a checkbox dialog box
7. **showMessage**: displays a message in a dialog box; same as **pause**
8. **showData**: displays data in a dialog box similar to that of **askData**
9. **showText**: displays a text in a dialog box similar to that of **askText**
10. **showPicture**: displays a picture from a gif file
11. **showDrawing**: displays a still or animated SQLdraw drawing
12. **askCombo**: opens a composite dialog box
13. **displayHelp**: opens a help window and displays the contents of a help file
14. **closeHelp**: closes the help window
15. **outputOpen, outputAppend**: specifies the current output channel and its writing mode
16. **parameter styleXXX**: defines various style aspects in dialog boxes
17. **parameter colorXXX**: defines colors in dialog boxes
18. **write**: write lines in the output channel
19. **print**: same as **write** but always writes in the output window

Directories and files

Statements that manipulate a file (or files) as a whole.

1. **selectFile**: opens a file selection dialog box
2. **selectDirectory**: opens a directory selection dialog box

3. **copyFile**: copies a file
4. **deleteFile**: deletes a file
5. **renameFile**: renames a file
6. **fileNames**: writes in a text file the names of files in a directory
7. **fileNamesAll**: recursively writes in a text file the names of files in a directory
8. **dirNames**: writes in a text file the names of sub-directories in a directory
9. **dirNamesAll**: recursively writes in a text file the names of sub-directories in a directory
10. **dirFileName**: writes in a text file the names of files and sub-directories in a directory
11. **dirFileNamesAll**: recursively writes in a text file the names of files and sub-directories in a directory

File contents manipulation

Statements that create, modify, read, process or display the contents of a file.

1. **HEXdecode α To β** : decodes a *hexadecimal* character string into a byte string
2. **HEXencode α To β** : encodes a byte string into a *hexadecimal* character string; α and β are either **Var** or **File**, depending on the nature of the source and target containers
3. **b64encode α To β** : encodes a character string in *standard base64*; α and β are either **Var** or **File**, depending on the nature of the source and target containers
4. **b64encode α To β -u**: encodes a character string in *url-safe base64*
5. **b64decode α To β** : decodes the value of a variable in *base64 (standard or url-safe)*
6. **hashFile**: compute the hash value of a file
7. **fileToVariable**: stores in a variable the contents of a file or a line from a text file
8. **variableToFile**: stores the value of a variable in a file
9. **webfileToFile**: download a file from the web
10. **BfileToVariable**: store the contents of a binary file into a variable
11. **openTextFile**: opens a text file
12. **nextTextLine**: reads next line in a text file
13. **closeTextFile**: closes a text file
14. **displayHelp**: opens a help window and displays the contents of a help file

15. **for**, **for-endfor**: *range*, *SQL* or *file* loop
16. **outputOpen**, **outputAppend**: specifies the current output channel and its writing mode
17. **showPicture**: displays a picture from a gif file
18. **showDrawing**: displays a still or animated SQLdraw drawing
19. **import**: loads data from an external file in a table

Control statements

Statements that may change the sequential execution of the statements of a script.

1. **return**: terminates the execution of the current (sub-)script
2. **stop**: terminates the execution of the current main script (and of all its sub-scripts)
3. **shutDown**: terminates the execution of all scripts and stops the SQLfast interpreter
4. **wait**: suspends script execution for a fixed or random time duration
5. **trace**: enable/disable statement execution tracing
6. **pause**: same as **showMessage**
7. **continue**: no operation
8. **execSQL**: executes an SQLfast script
9. **execApp**: executes a Python program or a function within a Python program. Same as **function**.
10. **function**: executes a Python program or function. Same as **execApp**.
11. **execProg**: executes an external program.
12. **fastExec**: executes a pure SQL script. Statements are not checked.
13. **if**, **if-endif**, **if-else-endif**: alternative statement
14. **for**, **for-endfor**: *range*, *SQL*, *file* or *list* loop
15. **while**, **while-endwhile**: condition-driven loop
16. **next**: forces the next iteration in a **for** or **while** loop
17. **exit**: exits from a loop
18. **goto**: continues the execution at a named point of the script
19. **label**: defines a named point in the script
20. **onError**: defines the reaction of the interpreter when an error occurs
21. **error**: triggers a user defined error

Part 3 - SQLfast statement description

symbols

|

Element separator in the label list of dialog boxes. The right side widget will be placed *under* the last one. Also used to locate elementary dialog boxes within in composite boxes.

||

Element separator in the label list of dialog boxes. The right side widget will be placed *at the right* of the last one. Also used to locate elementary dialog boxes within in composite boxes.

|||

Element separator in the composite boxes. The right side widget will be placed *under* the last one but as far as possible to the left.

@

Specifies a string control character (newline, tabulation, spaces, etc.)

/

Specifies control parameters in the message field of dialog boxes.

In an elementary dialog box (**askText**, **showText**, **showPicture**, **show-drawing**), specifies size, boldface and line wrapping.

In a composite box (**askCombo**), specifies the size of an elementary box.

[]

In a **for** statement, specifies a *range sequence*, a *sequential text file* or an *SQL sequence*.

In any dialog box, specifies the *message field*.

In an **askCombo** composite box, specifies an *elementary box*.

In an **askData** dialog box, specifies the list of predefined values.

()

In **if** and **while** statements, specifies a condition.

In **askData** dialog boxes, specifies a list of predefined constants.

{ }

Creates a title in a dialog box.

Protects an argument that may include commas in **execApp**, **function** and **execProg** statements.

—

The underscore symbol (**_**) plays no particular role in SQLfast. However, some components such as the script loader (SQLfastLoader.py) may generate labels and variables in translated scripts. To avoid conflicts with user defined variables and labels, their name is prefixed with an underscore character. Therefore, it is recommended to avoid variables names and labels beginning with an underscore character.

\$var\$

Substitution operator that replaces the name of variable **var** by its value. This expression may appear *almost* anywhere in *almost* all statements. For each statement, the resolution process is performed recursively until no further substitution is possible. Default delimiters **\$** and **\$** can be changed by statement **parameter delimiters** or by parameter **delimiters** in file SQLfast.ini. In this pair of symbols, only the first one is mandatory. Substitution can be disabled by statement **substitution** or in statement **set-n**.

\$vars\$

Same behavior as **\$var\$** except that the value is modified such that each single quote is replaced by two of them. This operator is evaluated only once, when all **\$var\$** operators have been resolved. Designed to simplify the use of character strings in SQL expressions, in particular in expressions and conditions as well as in pure SQL queries.

-- Loading data

Comment line. "--" as first non blank characters makes SQLfast engine ignore this line.

A

abortDB;

Cancels the data modifications carried out in the database since the last **commitDB** or **abortDB**, or since the start of the script. Relevant in transaction commit mode but useless in autocommit mode.

```
alter table CUSTOMER add column STATUS text;
```

Modifies some aspects of the structure of a table of the database. Note that the SQLite version of this query is limited to adding a column and renaming a table. Updates the local dictionary if any.

```
ask X,Y,Z = [Enter coordinates] Width | Depth | Height;
ask-n X,Y,Z = etc.;
ask-u X,Y,Z = etc.;
```

In the *new* mode (**ask** or **ask-n**), opens a dialog box, lets the user enter a series of values and assigns them to the target variables. In the *update* mode (**ask-u**), opens a dialog box, displays the values of the target variables and lets the user change them if desired.

The dialog box comprises a message field and a series of labelled entry fields. The character string enclosed between symbols "[" and "]" appears in the message field. The contents of the message field appears in boldface if parameter **/b** is specified in the [] block. Parameter **/x** specifies the width of the entry fields.

The entry field labels are specified in the label list at the right of the message, separated by symbols "|" (that can be changed if symbol | appears in a label). Separator "||" starts a new field column.

A label enclosed between { and } defines a **title**, that is, a mere label, with no associated field.

A list of predefined values can be associated to an entry field label. This list can be specified as a list of constants (Category[(B1,B2,C1,C2)]) or as the result of an SQL query (Category[select distinct Cat from CUSTOMER]). The user can select a predefined value from the list or type a value in the entry field. It is possible to force the user to select a predefined value from the list through symbol "!", as in Category[!(B1,B2,C1,C2)] and Category[!select distinct Cat from CUSTOMER].

The elements of the list of constants can (but need not) be quoted ('B1'). If an element includes a quote, the latter must be doubled and the element must be quoted ('O' 'Hara'). Similarly, if an element includes a comma, it must be quoted ('E.', 'F.', 'Codd').

The message, the labels and the titles may comprise string control characters (@).

Returns in system variable DIALOGbutton the name of the button that closed the dialogue (e.g., OK or Cancel).

```
askCombo [/bComplete the form]
    [ask-u A,B,C=[Identification] Name|Surname|Address]
    | [selectOne D=[Status] A1|A2|A3|B|CC]
    || [/y2showText txt= [/x72/y18 Description]];
```

Assembles several elementary boxes (`ask`, `selectOne`, `selectMany`, `askText`, `selectList`, `selectList2`, `showData`, `showText`, `showPicture`) to form a single dialog box. Each individual elementary box can be in *update* or *new* mode. A global message is supplied by the first `[]` block.

Size parameters `/x` and `/y` can be specified for each elementary box. They define the number of horizontal and vertical frames used by the box. Default values are `/x1/y1`.

Box separators `|` and `||` have the same effect as in multiwidget elementary boxes (e.g., `ask`). Separator `|||` acts as separator `|` then moves the box at the far left.

Returns in system variable `DIALOGbutton` the name of the button that closed the dialogue (e.g., OK or Cancel).

```
askData X,Y,Z = [Enter coordinates] Width | Depth | Height;
askData-n X,Y,Z = etc.;
askData-u X,Y,Z = etc.;
```

Same as `ask`.

```
askDelete DETAIL;
```

Lets the user select rows and interactively delete some of them. *Not implemented yet.*

```
askInsert DETAIL;
```

Lets the user interactively insert new rows. *Not implemented yet.*

```
askText CV;
askText CV = [/x64/y12 Fill Johnson's CV];
askText-n CV = [/bFill Johnson's CV];
askText-u CV = [Fill Johnson's CV];
```

In the *new* mode (`askText` or `askText-n`), lets the user enter a text of arbitrary length and assigns it to the target variable. In the *update* mode (`askText-u`), displays the contents of the target variable and lets the user update it if desired. The new text version is then returned in the target variable. The dialog box comprises an optional message field, the contents of which is specified by the character string enclosed between symbols "[" and "]". This `[]` block can also include display parameters: `/b` (message in bold-face), `/y` (height of the text field) and `/x` (width of the text field).

Returns in system variable `DIALOGbutton` the name of the button that closed the dialogue (e.g., OK or Cancel).

```
askUpdate PRODUCT;
```

Lets the user select rows and interactively update their column values. *Not implemented yet.* Use a loop instead.

B

```
b64encode $\alpha$ To $\beta$  target = source;
```

Encodes in *standard base64* the content of variable or file **source** and stores it in variable or file **target**. Keyword α (= **Var** or **File**) specifies the nature of **source** and β the nature of **target**. For example, statement **b64encodeVarToFile** target.txt = source encodes in *standard base64* and stores in file target.txt the value of variable source.

base64 encoding transforms an arbitrary byte string (containing for example accented characters, utf-8 or cp1252 codes, images, binary code of programs) into a string of printable characters using a reduced alphabet made up of uppercase letters A-Z, lowercase letters a-z, digits 0-9 + additional characters '+' and '/', totalling 64 distinct characters. Each sequence of three bytes (*8-bit entities*) of the source is interpreted as a sequence of four *sextets* (*6-bit entities*), each denoting one of the 64 characters of the reduced alphabet. So, a source of **N** arbitrary bytes is transformed into a string of about $4/3 \times N$ printable characters.

```
b64encode $\alpha$ To $\beta$ -u target = source;
```

Same as **b64encode α To β** using the *url-safe base64*. In this reduced alphabet, the two additional characters are '-' and '_' instead of '+' and '/'. This alternative alphabet generally is used to avoid character strings that could erroneously be confused with file paths and url's, in which characters '+' and '/' have a special meaning.

```
b64decode $\alpha$ To $\beta$  target = source;
```

Decodes the content of variable or file **source**, coded in *base64* (*standard* and *url-safe*), and stores it in variable or file **target**. For example, statement **b64decodeFileToFile** picture.jpg = source.txt stores in clear in file picture.jpg the *base64*-encoded contents of file source.txt.

```
BfileToVariable pet = MyDog.jpg;
```

Loads the content of binary file MyDog.jpg into target variable pet. Variable FILEdiag is set to OK if the file exists and is not empty, and to NONE otherwise. If no data has been read from the file, the target variable keeps its former value. Since the value of pet is (or may be) binary, it cannot be used in a variable/value substitution operator. It can be used in limited situations such in insertLOB and showData statements.

C

closeDB;

Closes the currently opened database. An implicit `abortDB` is executed before closing. The dictionary, if any, is deleted.

closeHelp;

Closes the help window.

closeTextFile inputText.txt;

Closes text file inputText.txt.

commitDB;

Confirms the data modifications carried out since the last `commitDB` or `abortDB`, or since the start of the script, and makes them permanent. Note that, in many DBMS (such as SQLite), DDL queries automatically execute a `commitDB`. Relevant in transaction commit mode but useless in autocommit mode.

compute N = \$N\$ + \$increment\$;

Evaluates the right hand side expression and assigns the result to the target variable. After variable substitution, the resulting expression comprises constants only and must comply with the SQL syntax of *select list* elements. SQLfast provides a large set of more than sixty operators and functions.

continue;

The execution of this statement has no effect. It can be used as a *null* statement where a statement is required but no action is needed.

copyFile ORDERS.db,ORDERS.bak;

copyFile-e ORDERS.db,ORDERS.bak;

copyFile-r ORDERS.db,ORDERS.bak;

Creates a copy of source file ORDERS.db with name ORDERS.bak. The source file must exist and must be closed. However, if suffix `-e` is specified, does nothing if the source file does not exist. The target file must not exist. However, if suffix `-r` is specified, the target file is replaced if it already exists. These suffixes can be combined in `-er` or `-re`.

createDB ORDERS.db;

Creates a new empty database and opens it. This database may not exist. Condition `fileExists` can be used to test this fact. Creates the local dictionary if parameter **Dictionary** is **on**.

An *in memory* database is created if the database name is `InMemory`. Such a database is stored in RAM and is lost when it is closed.

createDictionary

Creates (or recreates if already exists) a local data dictionary for the current database. From now on, the dictionary is updated after each DDL statement execution. The dictionary includes six tables: `SYS_TABLE`, `SYS_COLUMN`, `SYS_KEY`, `SYS_KEY_COMP`, `SYS_INDEX`, `SYS_INDEX_COMP`. The last two tables are created for technical reason and may be ignored. Synthetic view `SYS_INTER_TABLE` represents the directed graph defined by foreign keys.

createOrReplaceDB `ORDERS.db`;

Creates or replaces a database and opens it. If a database with this name already exists, it is replaced by an empty database. Creates the local dictionary if parameter **Dictionary** is **on**.

create index ...; **create index if not exists ...;**

Creates (conditionally with the `if not exists` clause) an index for a table of the current database. Updates the local dictionary if any.

create table `CUST (CNum ... );` **create table if not exists** `CUST (CNum ... );` **create temporary table** `CUST (CNum ... );`

Creates (conditionally with the `if not exists` clause) a new table in the current database. Updates the local dictionary if any.

create trigger `Trg_CUST_INSERT ...;` **create trigger if not exists** `Trg_CUST_INSERT ...;` **create temporary trigger** `Trg_CUST_INSERT ...;`

Creates (conditionally with the `if not exists` clause) a trigger in the current database. Updates the local dictionary if any.

create view `LONDON_CUST as select ...;` **create view if not exists** `LONDON_CUST as select ...;` **create temporary view** `LONDON_CUST as select ...;`

Creates (conditionally with the `if not exists` clause) a view in the current database. Updates the local dictionary if any.

create view `LONDON_CUST as with ...;` **create view if not exists** `LONDON_CUST as with ...;` **create temporary view** `LONDON_CUST as with ...;`

Same as `create view ... as select` but defined as a CTE, possibly recursive. Updates the local dictionary if any.

D

date()

Temporal function that can be used in statement **compute** and in conditions. See other temporal functions in **SQLfast expression** section.

datetime()

Temporal function that can be used in statement **compute** and in conditions. See other temporal functions in **SQLfast expression** section.

delete from DETAIL **where** ProdID = 'MDF-60';

Deletes selected rows from a table.

deleteDB ORDERS.db;

Deletes a database.

deleteDictionary

Deletes the current local data dictionary, if any. Does not close the current database.

deleteFile firstFile.txt;

deleteFile-e firstFile.txt;

Deletes file firstFile.txt. This file must exist and must be closed. However, deleteFile-e, does nothing if the file does not exist.

dirNames dir.txt = D:/SQLfast/Scripts;

dirNames-s dir.txt = D:/SQLfast/Scripts;

dirNames-r dir.txt = D:/SQLfast/Scripts;

Writes in text file dir.txt the sorted list of the full names (full path included) of the direct subdirectories in root directory D:/SQLfast/Scripts. If suffix **-s** is specified, then only short local names are provided (path excluded). Root directory name is not included in the text file, unless suffix **-r** is specified. When a directory name includes invalid characters, the latter are replaced by character **_** and the name is prefixed with character *****.

dirNamesAll dir.txt = D:/SQLfast/Scripts;

dirNamesAll-s dir.txt = D:/SQLfast/Scripts;

dirNamesAll-r dir.txt = D:/SQLfast/Scripts;

Recursive form of dirNames. Writes in text file dir.txt the sorted list of the full names (full path included) of the direct and indirect subdirectories in root directory D:/SQLfast/Scripts. If suffix **-s** is specified, then only short local names are provided (path excluded). Directories are listed in depth-first order. Root directory name is not included in the text file, unless

suffix **-r** is specified. When a directory name includes invalid characters, the latter are replaced by character **_** and the name is prefixed with character *****.

```
dirFileNames dir.txt = D:/SQLfast/Scripts,ext;  
dirFileNames-s dir.txt = D:/SQLfast/Scripts,ext;  
dirFileNames-r dir.txt = D:/SQLfast/Scripts,ext;
```

Same as **dirNames** + **fileNames**. Provides a sorted list of directory and file names. Parameter **ext** is optional.

```
dirFileNamesAll dir.txt = D:/SQLfast/Scripts,ext;  
dirFileNamesAll-s dir.txt = D:/SQLfast/Scripts,ext;  
dirFileNamesAll-r dir.txt = D:/SQLfast/Scripts,ext;
```

Recursive form of **dirFileNames**. Same as **dirNamesAll** + **fileName-sAll**. Provides a sorted list of directory and file names. Parameter **ext** is optional.

```
displayHelp User_Manual.help;
```

Opens a help window and display the contents of the specified file. Only one help window can be opened at a time. If the file name is relative, it is local to help folder (default: **SQLfastHelp_EN** or **SQLfastHelp_FR**).

```
drop index ...;
```

Deletes an index of the current database.

```
drop table CUST;
```

Deletes a table in the current database. This table may not be referenced by other tables nor be used in SQL views (though some DBMS provide for less strict conditions). Updates the local dictionary if any.

```
drop view LONDON_CUST;
```

Deletes a view in the current database. Updates the local dictionary if any.

```
dropVariable result;
```

Deletes user variable **result**. From now on, this variable is unknown. System variables cannot be deleted.

```
dropAllVariables;
```

Deletes all user variables.

E

error Sorry, a system error occurred;

Triggers a script-defined error. The message is available in system variable `EXTENDEDdiag`. The execution stops or continues depending on the `onError` status. This statement is mainly used to control the execution of high-level statements when they are translated by the precompiler.

execApp loadScript = **dbaseToSQLConverter** \$profile\$;

Suspends the execution of the current script and starts the execution of a Python application or function. When the application execution stops, current script execution resumes.

Same as:

execApp loadScript = **dbaseToSQLConverter:Main** \$profile\$.

Executes default function `Main(p, ...)` of Python module `dbaseToSQLConverter.py`, with the arguments specified at the right side (`$profile$`). The function returns one (or several) values, that are assigned to SQLfast target variables (loadScript).

Target variables and arguments are comma-separated. If an argument includes, or may include, commas (example: 35, Main Street), it must be enclosed between { and }. *Important*: leading and trailing spaces are significant: in argument list "A, B, C ", the arguments are "A", " B" and "C ".

execApp is a synonym of **function**.

execProg "Foxit Reader.exe" Manual.pdf;

Starts an external program ("Foxit Reader.exe") with the specified input file (Manual.pdf) and continues the execution of the current script. The current script does not suspend its execution. There is no automatic synchronization mechanism as for **execSQL** and **execApp**. If needed, the file name can be enclosed into { and }.

execSQL Produce-Invoice.sql;

Suspends the execution of the current script and starts the execution of secondary script Produce-Invoice.sql. When the secondary script stops, current script execution resumes. The current script is the *calling* script while the secondary script is the *called* script. A script can call itself, either directly, or indirectly (recursive call allowed).

All the variables are global, be they created by the main script or any secondary script. Similarly an opened database is global. Once it has been opened, in some script, it is available to all scripts, main or secondary, unless specified otherwise by parameter `autocloseDB`.

The name of the script to execute is relative to the current *script directory*, unless an absolute path is specified. The script directory path is specified by parameter `scriptdirectory` in initialization file `SQLfast.ini`.

The number of simultaneous calls can be limited to a maximum through parameter `maxProcLevel`. The calling level of the current script is given by system variable `procLevel`.

exit;

In the body of a **for** or **while** statement, forces the loop to terminate.

```
extract N,A,C = select Name,Address,City from CUSTOMER
                where CustID = '$ID$';
```

Retrieves the first row resulting from the execution of the select statement and assigns its column values to the target variables. System variable `SQLdiag` indicates whether a row was found and returned (value OK) or not (value NONE). If no row has been extracted, the variables are set to an empty string.

```
extract N = select CustID from CUSTOMER
            where Cat = 'B1' #2;
```

Retrieves the second row resulting from the execution of the select statement and assigns its column values to the target variables. System variable `SQLdiag` indicates whether this row was found and returned (value OK) or not (value NONE). If no row has been extracted, the variables are set to an empty string. Such a select query is called *indexed select*.

```
extract N,A,C = with ...;
```

Same as `extract ... = select ...` in which the row is defined through a **with** query, possibly recursive

```
export [select NCLI, Name from CUSTOMER] to CUST.rtf;
```

The result set of the `select` query is written in the external file in the format specified by the file extension (here `rtf`). This function generally provides a more precise result than the pure `select` statement. (*not implemented yet*)

F

```
fastExec source.sql,$first$,$last$,$step$;
fastExec-c source.sql,$first$,$last$,$step$;
fastExec-t source.sql,$first$,$last$,$step$;
fastExec-ct source.sql,$first$,$last$,$step$;
```

Fast execution of the SQL statements of script `source.sql`. The statements are not validated nor prepared (no variable/value substitution). Only SQL queries

are allowed. The script is not preloaded before execution as is the case for normal SQLfast scripts, so that the size of scripts is only limited by disk space. The script file typically comprises DDL queries + a large number of **insert**, **update** and **delete** queries, plus blank lines and comments. For other statements and for more complex scripts, use **execSQL** instead.

Parameters **first** and **last** specify the range of line numbers that will be executed (useful for executing a subset of a long script). Parameter **step** specifies the number of lines after which a checkpoint is executed. At each checkpoint, a `commitDB` is executed if suffix **-c** is specified and a message comprising the current line number and the current execution time is written to the output channel if suffix **-t** is specified. Suffixes **-c** and **-t** can be combined in **-ct** or **-tc**.

When a syntax or file error is detected, the execution stops and `FILEdiag` is set to `ERROR` or `NONE`.

When an SQL execution error is detected, the execution stops and `SQLdiag` is set to `ERROR`.

In both cases, system variable `EXTENDEDdiag` provides details on the error.

Example: `fastExec E:/Load.sql,1,100000000,100000;`

```
fileNames fil.txt = D:/SQLfast/Scripts;
fileNames fil.txt = D:/SQLfast/Scripts, jpg;
fileNames-s fil.txt = D:/SQLfast/Scripts, jpg;
```

Writes in text file `fil.txt` the sorted list of the full names (full path included) of the files in directory `D:/SQLfast/Scripts`. If extension parameter `jpg` is provided, only the files with this extension are considered. If suffix **-s** is specified, then only short local names are provided (path excluded). When a file name includes invalid characters, the latter are replaced by character `_` and the name is prefixed with character `*`.

```
fileNamesAll fil.txt = D:/SQLfast/Scripts, jpg;
fileNamesAll-s fil.txt = D:/SQLfast/Scripts, jpg;
```

Recursive form of `fileNames`. Writes in text file `fil.txt` the sorted list of the full names (full path included) of the files in directory `D:/SQLfast/Scripts` as well as those in its direct and indirect subdirectories. If extension parameter `jpg` is specified, only the files with this extension are considered. If suffix **-s** is specified, then only short local names are provided (path excluded). Directories are visited in depth-first order. When a file name includes invalid characters, the latter are replaced by character `_` and the name is prefixed with character `*`.

```
fileToVariable comment = report.txt;
```

Loads the contents of file `report.txt` into target variable `comment`. Variable `FILEdiag` is set to `OK` if the file exists and is not empty, and to `NONE` other-

wise. If no data has been read from the file, the target variable keeps its former value. This statement expect to load *text files*. However, binary file can also be loaded, but the status of the variable will not be '*binary*'.

```
fileToVariable rule = Source-rules.txt #10;
```

Loads the 10th line of text file Source-rules.txt into target variable rule. Variable FILEdiag is set to OK if the file exists and this line has been found, and to NONE otherwise. If no data has been read from the file, the target variable keeps its former value.

```
for I = [2,$NC$] set quantity-$I$ = 0;
for I = [2,$NC$,$Step$] write $I$;
```

*Elementary **range for** statement*, in which the body comprises one statement only. For the successive integer numbers in interval [2, \$NC\$], assigns them to variable I and executes the body statement (set, write). If present, parameter \$Step\$ specifies the increment of the sequence; in this case, the three components can be integer or real numbers.

```
for I = [2,$NC$];
for I = [2,$NC$,$Step$];
    <body>
endfor;
```

*Block **range for** statement*, in which the body may comprises several statements. For the successive integer numbers in interval [2, \$NC\$], assigns them to variable I and executes the statements of the loop body (<body>). If present, parameter \$Step\$ specifies the increment of the sequence; in this case, the three components can be integer or real numbers.

```
for N,C = [select Name, City from CUSTOMER
           where CustID = $I$] write $N$ - $C$;
```

*Elementary **SQL for** statement*, in which the body comprise one statement only. For the successive rows returned by the SQL select statement, assigns the values of the specified columns to the target variables then executes the body statement (write). If the query returns no row, the statement is not executed. System variable SQLdiag returns OK or NONE according to the number of rows returned and processed.

```
for I,A = [select CustID,Account from CUSTOMER
           where City = '$C$'];
    compute TA = $TA$ + $A$;
    write Customer ID: $I$;
endfor;
```

*Block **SQL for** statement*, in which the body may comprise several statements. For the successive rows returned by the SQL select statement,

assigns the values of the specified columns to the variables then executes the body statements until the `endfor` statement. If the query returns no row, the statement is not executed. System variable `SQLdiag` returns OK or NONE according to the number of rows returned and processed.

```
for I,A = [select CustID,Account from CUSTOMER
          order by City,CustID #[2,$K$]];
```

Indexed SQL for statement (elementary or block). Returns the rows of the result set the ranks of which fall in the interval specified in the `#` clause. Single value rank (`#8`) is not allowed.

```
for I,A = [with ...];
```

Same as `for .. [select ...]` in which the sequence is defined as a **with** query, possibly recursive

```
for L = [file inputText.txt] write $L$;
```

Elementary file for statement, in which the body comprises one statement only. For the successive lines in text file `inputText.txt`, assigns it to target variable `L` then executes the body statement (`write`). A line is any, possibly empty, character string followed by a *new line* or *end of file* character in the file. The latter character is not part of the returned line.

```
for L = [file inputText.txt];
  write Line: $L$;
endfor;
```

Block file for statement, in which the body may comprise several statements. For the successive lines in text file `inputText.txt`, assigns it to target variable `L` then executes the body statements (`write`). A line is any, possibly empty, character string followed by a *new line* or *end of file* character in the file. The latter character is not part of the returned line.

```
for L = [items one;two;three] write $L$;
for L = [item(,) one,two,three] write $L$;
```

Elementary list for statement, in which the body comprises one statement only. For the successive elements of SQLfast list `one;two;three`, assigns it to target variable `L` then executes the body statement (`write`). Keyword `items` (or `itemS`) indicates that the list has been built with the default separator. If another separator (`<sep>`) has been used, it is specified by the form `item(<sep>)`.

```

for L = [items one;two;three];
for L = [item(,) one,two,three];
    write Item: $L$;
endfor;

```

*Block **list for** statement*, in which the body may comprise several statements. For the successive elements of SQLfast list `one;two;three` (using the standard list separator) assigns it to target variable `L` then executes the body statement (`write`). Keyword `items` (or `itemS`) indicates that the list has been built with the default separator. If another separator (`<sep>`) has been used, it is specified by the form `item(<sep>)`.

```

function num1,num2 = LNum:Decompose 123.456, 2;

```

Suspends the execution of the current script and starts the execution of function `Decompose(n1,n2)` of Python library `LNum.py` with character string arguments `123.456` and `2`.

Target variables and arguments are comma-separated. If an argument includes, or may include, commas (example: `35, Main Street`), it must be enclosed between `{` and `}`:

```

function add = LStr:Frame {$address$},10,r;

```

`function` is a synonym of `execApp`.

G

```

generic PRAGMA foreign_keys = OFF;

```

Submits a non-standard statement to the DBMS. The SQLfast engine does not check the validity of the statement.

```

goto Next-Customer;

```

Script execution continues at the statement that follows the referenced label.

H

```

hashFile h = SQLfast-tutorial.pdf;

```

Stores in variable `h` the hash value of file `SQLfast-tutorial.pdf`.

```

HEXdecodeTo target = source;

```

Decodes the contents of variable or file `source`, coded in *hexadecimal*, and stores it in variable or file `target`. For example, statement `b64decodeFileToFile picture.jpg = picture.txt` stores in clear in file `picture.jpg` the *hexadecimal*-encoded contents of text file `picture.txt`.

HEXencode α To β target = source;

Encodes into a *hexadecimal* character string the contents of variable or file **source** and stores it in variable or file **target**. Keyword α (= **Var** or **File**) specifies the nature of **source** and β the nature of **target**. For example, statement **HEXencodeVarToFile** target.txt = source encodes in *hexadecimal* and stores in file target.txt the value of variable source.

hexadecimal encoding transforms an arbitrary byte string (containing for example accented characters, utf-8 or cp1252 codes, images, binary code of programs) into a string of characters from the reduced 16-symbol alphabet 0-9A-F. Each half-byte (4 bits) of the source byte string is coded by one of these characters. A source of **N** arbitrary bytes is transformed into a string of $2 \times \mathbf{N}$ printable characters.

I

if (\$A\$ = \$OK\$) **delete from** PRODUIT **where** QonHand < 0;

Elementary if statement, in which the body comprises one statement only. If the condition evaluates to *true*, executes the following statement (**delete**). The syntax of the condition is that of SQL.

if (\$A\$ = \$OK\$);
delete from PRODUIT **where** QonHand < 0;
endif;

Single branch block if-endif statement, in which the body may comprise several statements. If the condition evaluates to *true*, executes the statements that follow, until statement **endif**. If it evaluates to *false*, the execution continues at the first statement that follows the **endif** statement. If-endif blocks can be embedded.

if (\$QonHand\$ <= 0);
execSQL Restock.sql
else;
write Stock level of \$NP\$ sufficient
endif;

Two branch block if-else-endif statement. If the condition evaluates to *true*, executes the statements that follow, until statement **else**. Otherwise, executes the statements between statements **else** and **endif**.

import CUSTOMER **from** LONDON-1.csv;
import CUSTOMER **from** LONDON-1.dbf;

Loads the data of an external file into a database table. The format of the input data is specified by the file extension.

a) comma-separated values source files (extension 'csv')

The process of **csv** import is controlled by three system variables which are initialized in **SQLfast.ini** file and can be modified dynamically:

- **csv-separator**: the symbol used as value separator,
- **csv-header**: whether the source file includes a first record that specifies the column names ('0' = No, '1' = Yes),
- **csv-maxerrornbr**: the maximum number of errors beyond which the loading process is aborted and rolled back.
- **csv-mute**: hide the error messages.

If the target table already exists, the data are appended to its current contents. The header, if any, is ignored. The interpretation of an *empty string* in the source file depends on the properties of the target column. If the column is *nullable*, a null value is inserted, otherwise an empty string is inserted.

If the target table does not exist, the source file must include a header. A new table is created comprising columns with the name specified by the header and with type `varchar(256)`. The data of the source file are then inserted in this table. Each column is first considered not null, until a missing value is extracted, in which case the column becomes *nullable*. There is no attempt to guess the data types of columns.

If no more than `csv-maxerrornbr` errors have been detected during the operation, the loading is considered successful. Otherwise, it is aborted and all the insertions are rolled back. The number of data errors (e.g., not enough or too many values) and insert errors (e.g., uniqueness, value or foreign key errors) is provided by system variable **ImportErrors**.

In case of success (no more than `csv-maxerrornbr` errors), the insertions are committed at Level 0 but not at Level 1, in which case a `commitDB` or an `abortDB` must be executed explicitly by the script.

The errors are reported in log file **LONDON-1.csv.log**, located in the folder of the source file.

Empty lines are ignored. A records which comprise empty values only is skipped. If a record includes fewer values than expected, empty values are added. If a record includes too many values, it is truncated. These conditions do not raise an error.

The flow of error messages can be disabled through system variable `csv-mute = 1`.

b) dBase source files (extension 'dbf')

The process of **dbf** import is controlled by a system variable which is initialized in **SQLfast.ini** file and can be modified dynamically:

- **incoding_dbf**: the character coding scheme of the source file (typically **cp850** for DOS/Windows files).

If the target table already exists, the data are appended to its current contents. The interpretation of an *empty string* in the source file depends on the properties of the target column. If the column is *nullable*, a null value is inserted, otherwise an empty string is inserted.

If the target table does not exist, a new table is created comprising columns with the name and type specified in the source file. The data of the source file are then inserted in this table. Each column is first considered not null, until a missing value is extracted, in which case the column become *nullable*. There is no attempt to *guess* the uniqueness constraints.

Deleted records (delete-byte = '*') and corrupt records are skipped. The number of data errors (corrupt data) and insert errors (e.g., uniqueness, value or foreign key errors) is provided by system variable **ImportErrors**.

The insertions are committed at Level 0 but not at Level 1, in which case a `commitDB` or an `abortDB` must be executed explicitly by the script.

The errors are reported in log file LONDON-1.dbf.log, located in the folder of the source file.

a) other formats

To be developed.

```
insert into DETAIL values (30189,'$label$', $qty$);
insert into DETAIL (OrdID, ProdID, Qord)
      values (30189, '$label$', $qty$);
```

Inserts a new row in a table.

```
insertLOB var = IMAGE from ALBUM where ImID = 123;
```

Pseudo-SQL statement used to store the contents of variable `var`, considered to be a *large object* (text, image, sound, video, etc.) in column `IMAGE` of table `ALBUM` in all the rows that satisfy the **where** clause.

Column `IMAGE` must be a BLOB. If the value of `var` is a *binary* value, `var` must be of type *binary*, that is, it has been loaded through, e.g., a **BfileToVariable** statement. If this value is a pure text, `var` may have been assigned by any way, such as **BfileToVariable**, **fileToVariable** or `set`.

J

K

L

label Next-Customer;

Defines a labelled point in the script. Labels starting with character `_` should be avoided, as all the labels created by the preprocessor are prefixed by this symbol.

M

N

next;

In the body of a **for** or **while** statement, forces the loop to execute the next iteration.

nextTextLine line = inputText.txt;

Read the next line in opened text file `inputText.txt` and stores it in target variable `line`. If a line has been extracted, `FILEdiag` contains OK, otherwise, it contains NONE and the target variable is empty

O

onError stop|continue;

Specifies the way the SQLfast engine reacts to script errors.

- **on-error** stop: an implicit `abortDB` is executed, an error message is written on the console and the script stops. This is the default behavior.
- **on-error** continue: the current operation is aborted, an error message is written on the console and the script continues its execution. Must be used with caution, since the behavior of the script may become unpredictable.

openDB ORDERS.db;

Opens a database. Script error raised if the database does not exist. Creates the local dictionary if parameter **Dictionary** is on.

openTextFile inputText.txt;

Opens text file `inputText.txt`. The file is made up of a sequence of lines each followed by a *new line* or *end of file* characters.

output on | off;

Enables or disables writing on the output channel. The default setting is output on.

outputOpen INVOICES.xml;

The current output channel is closed. File INVOICES.xml is opened and becomes the current output channel. If a file with this name already exists, its contents is lost. If the file name is relative, the file is created in system folder **Output**. It can be relocated, for instance in folder **Files**, through expression \$fileDirectory\$/INVOICES.xml.

outputOpen standard;

The current output channel is closed. The standard output file (a text file with the same name as the main script and extension *.lst) is opened and becomes the current output channel. Its former contents is lost. The file is created in system folder **Output**.

outputOpen console;

The current output channel is closed. The console is used again as the current output channel. The previous contents of the console is preserved. Note that *closing the console* is a fictitious operation. In the current version of SQLfast, this channel is replaced by window.

outputOpen window;

The current output channel is closed. An empty window is opened and becomes the current output channel. Its former contents is lost. (*currently disabled*)

outputOpen result.var;

The current output channel is closed. Variable result is now used as the current output channel. Its former contents is lost. Note that *closing a variable* is a fictitious operation.

outputAppend ERROR.err;

The current output channel is closed. File ERROR.err is opened and becomes the current output channel. If this file already exists, its contents is preserved. If the file name is relative, the file is created in system folder **Output**. It can be relocated, for instance in folder **Files**, through expression \$fileDirectory\$/INVOICES.xml.

outputAppend standard;

The current output channel is closed. The standard output file is opened and becomes the current output channel. Its former contents is preserved. The file is created in system folder **Output**.

outputAppend console;

Same as outputOpen. In the current version of SQLfast, this channel is replaced by window.

outputAppend window;

The current output channel is closed. The window is opened and becomes the current output channel. Its former contents, if any, is preserved. (*currently disabled*)

outputAppend result.var;

The current output channel is closed. Variable `result` is now used as the current output channel. Its former contents is preserved. If the file name is relative, the file is created in system folder **Output**. It can be relocated, for instance in folder **Files**, through expression `$fileDirectory$/INVOICES.xml`.

P

parameter commitmode = autocommit | transaction | default;

Defines the commit mode for further database opening statements (`createDB`, `createOrReplaceDB`, `openDB`). When a database is opened in `autocommit` mode, a **commit** is implicitly executed after each successful SQL modification statement. When opened in `transaction` mode, explicit **commit** statements must be used to commit modifications. The default commit mode resets the commit mode to the default value set in the `SQLfast.ini` file.

parameter delimiters = [];

The current symbols used as variable delimiters in variable substitution operator are replaced by new symbols (or character strings). At least the first symbol must be specified.

parameter label-separator = /;

Specifies the current label separator in **ask**, **selectOne**, **selectMany** statements. Similarly, specifies the current block separator in **askCombo** statement. Default is |.

parameter select-align = N;

Parameter `select-align` specifies whether the forthcoming `select` statements will write their result set in a tabular format (Y) or not (N). In particular, it must be set to N for formats that provide their own rules to define data alignment, such as RTF, LaTeX or CVS.

parameter styleXXX = flat | relief;

where XXX = `MainFrame`, `GeneralMessage`, `BoxFrame`, `BoxMessage`, `Text`, `Entry`, `Check`, `Radio`, `Button`. Defines the aspect of the components of dialog boxes: (1) main frame and general message of composite boxes, (2) frame and message of elementary boxes, (3) elementary widgets.

parameter colorXXX = #RRGGBB | color_name;

where XXX = MainFrame, GeneralMessage, BoxFrame, BoxMessage, Text, Entry, Check, Radio, Button. Defines the background color of the components of dialog boxes (same as styleXXX). #RRGGBB is the RGB hexadecimal definition of the color. color_name is the tkinter name of the color.

parameter entryWidth = 25;

Specifies the number of characters of data entry fields. Fields with predefined values are 7 character shorter.

pause Update completed.;

Execution stops, an information box pops up, showing a message (optional) and waiting for user clicking on the Continue button. Then execution resumes. Statement showMessage is a synonym of pause.

print Processing completed for customer \$CustName\$;

Same as write. Always writes in the output window, whatever the current output channel.

Q

R

renameFile D:/Rep/Customer.lst,export.sql;

renameFile-e D:/Rep/Customer.lst,export.sql;

renameFile-r D:/Rep/Customer.lst,export.sql;

Renames source file D:/Rep/Customer.lst to target name D:/Rep/export.sql. This file must be closed and must exist. However, if suffix **-e** is specified, does nothing if the source file does not exist. The target name must be a local name, without path. If it has a path, the latter is ignored (replaced by the source path). No file with the target file name may exist. However, if suffix **-r** is specified, the target file is replaced if it already exists. These suffixes can be combined in **-er** or **-re**.

replace pureText = \n,@n;

Replaces in the value of variable pureText all instances of string \n by an instance of string @n. Can be used in lieu of function Replace if the source string raises problems when inserted through substitution operator (it includes '\n' for example). Can be used to clean the contents of a Windows *.txt file, where newline control characters (chr(13)+chr(10)) are not recognized as '\n' by Python.

return;

Stops the execution of the current script. The calling script, if any, resumes its execution. Equivalent to `stop` if the current script is the main script.

S

selectDirectory source = D:/SQLfast/Texts;

Lets the user select a directory name in a directory selection dialog box. The selected name is stored in variable `source`, the starting directory is D:/SQLfast/Texts.

Returns an empty file name if button Cancel has been clicked on. System variable `DIALOGbutton` returns the name of the button used to quit the dialog box.

selectFile source = D:/SQLfast/Texts,Text files,txt;

Lets the user select a file name in a file selection dialog box. The selected name is stored in variable `source`, the current directory is D:/SQLfast/Texts and the file extension is `txt`. The file type descriptor is `Text files`.

Returns an empty file name if no name has been selected or if button Cancel has been clicked on. System variable `DIALOGbutton` returns the name of the button used to quit the dialog box.

```
selectList list = [Select cities] [(London,Paris,Madrid)];
selectList list = [Select cities]
                  [select distinct City from CUSTOMER];
selectList-u list = etc.;
```

Lets the user select one or several items in the list box filled with either a constant list or the result of an SQL query. The result is returned as a constant list, stored in variable `list`. In the *update* mode (**selectMany-u**), the values in variable `list` are preselected in the list box.

```
selectList2 list = etc.;
selectList2-u list = etc.;
```

Same as **selectList**, but the dialog box is displayed in two list boxes, the left side one contains the items not yet selected and the right side one the items already selected. In the *update* mode (**selectMany2-u**), the values in variable `list` are preselected in both list boxes

```

selectMany C,O,D,P = [Select tables to backup:]
                        CUSTOMER | CUSTORDER | DETAIL | PRODUCT;
selectMany-n C,O,D,P = etc.;
selectMany-u C,O,D,P = etc.;

```

In the *new* mode (**selectMany** or **selectMany-n**), lets the user check some items and returns **0** (checked) or **1** (not checked) in the target variables. In the *update* mode (**selectMany-u**), displays a list of items, some of them being possibly checked, and lets the user change them if desired. The dialog box comprises an optional message field and a series of labelled check buttons. The character string enclosed between symbols "[" and "]" appears in the message field and the button labels are specified in the label list at its right, separated by symbols "|". Returns **1** if a button is checked and **0** otherwise. An empty field name starts a new field column.

The rules concerning titles, spatial layout and control characters are those of the **ask** statement.

Also returns in system variable **DIALOGbutton** the name of the button that closed the dialogue (OK or Cancel).

```

selectOne P = [CUSTOMER table.@nSelect a function:]
                New customer | Modify customer | Delete customer;
selectOne-n P = etc.;
selectOne-u P = etc.;

```

In the *new* mode (**selectOne** or **selectOne-n**), lets the user check one item and returns its sequence number in the target variable. In the *update* mode (**selectOne-u**), displays a list of items, one of them being possibly checked, and lets the user change it if desired. The dialog box comprises a message field and a series of labelled radio buttons. The character string enclosed between symbols "[" and "]" appears in the message field and the button labels are specified in the label list at its right, separated by symbols "|". Returns **0** if no button is checked. An empty field name starts a new field column.

The rules concerning titles, spatial layout and control characters are those of the **ask** statement.

Also returns in system variable **DIALOGbutton** the name of the button that closed the dialogue (OK or Cancel).

```

set message = Customer $cus$ not found. Try again ...;

```

Assigns to the target variable **message** the character string at the right of the "=" sign, from the first non space character to the last character before the termination symbol. Variable substitution is performed before assignment. The syntax allows the simplified form **set** X = ; to assign to X an empty string.

```
set nC = $N$;
```

Assigns to variable nC the value of variable N. *Beware:* set A = B does not assign to variable A the **value** of variable B, but the **name** of variable B, that is, letter B.

```
set var1,var2,var3 = exp1,exp2,exp3;
```

Multi-variables variant of the set statement. Equivalent to **set** var1 = val1; **set** var2 = val2; etc. If a value includes, or may include, commas, it must be enclosed between { and }. These curly brackets are not part of the value. The absence of value (or spaces) assigns an empty string to the corresponding variable (ex., **set** A,B,C = , , ; clears variables A, B, C).

```
set-n nC = The value is $N$;
```

Assigns to variable nC the character string following assignment symbol =. Suffix **-n** disables variable substitution for this statement. **set-n** can also be used for multi-variables assignments.

```
showMessage Data loading is completed;
```

Displays a message in a simple box. Execution stops, an information box pops up, showing a message and waiting for user clicking on any button. Then execution resumes. Statement pause is a synonym of showMessage. System variable DIALOGbutton is set.

```
showData X,Y = [Enter coordinates] Width | Height;
```

Suspends the execution of the script and displays the values of variables X, Y in a box similar to that of the ask statement. When the user click on any button, the script execution resumes. This statement is a simplified version of the ask-u statement that does not ask the user to enter or to change the data displayed. System variable DIALOGbutton is set.

```
showText descr = [Current version of the description];
```

Suspends the execution of the script and displays the value of variable descr in a box similar to that of the askText statement. When the user click on any button, the script execution resumes. This statement is a simplified version of the askText-u statement that does not ask the user to enter or to change the text displayed. System variable DIALOGbutton is set.

```
showPicture pic = [Nice picture, isn't it?];
```

Suspends the execution of the script and displays the bitmap picture contained in variable or file pic. These sources must contain a **gif**, **jpg** or **jpeg** picture. When the user clicks on any button, the script execution resumes. System variable DIALOGbutton is set.

showDrawing sales.draw = [Evolution of car sales in UK];

Suspends the execution of the script and displays the drawing defined by file (or variable) sales.draw. The source must contain an SQLdraw script. The drawing (still or animated) is rendered by the SQLdraw engine. If the drawing is animated (i.e., it comprises wait or delete commands), button Play appears. When clicked on, the engine animates the drawing. When the user click on the Continue button, the script execution resumes. Button Print generates an **eps** file from the drawing. System variable DIALOGbutton is set.

select CustID, Name, City **from** CUSTOMER **where** CAT = 'B2';

This SQL query is executed and its result set is written on the current output channel. The data are written according to a standard tabular format. This format is defined by a set of variables and parameters the value of which can be modified to define other output formats. Predefined variables/parameters settings are available for CSV, Key-value, RTF, LaTeX, HTML, XML, Tuple, Jason and SQL format.

select CustID **from** CUSTOMER **where** City = 'Poitiers' #5;

select CustID **from** CUSTOMER **where** City = 'Poitiers' #[5,10];

Same as **select** but limited to the rows of the result set with the specified rank. This form is called *indexed select*.

shutDown;

Terminates all active scripts as well as the execution of the SQLfast interpreter. Mainly use in conjunction with the autoexec parameter.

start-timer;

Resets the timer to 0.

stop;

Stops the execution of the current main script. Can be executed by a secondary script.

substitution on | off;

Enables or disables variable substitution for all the forthcoming statements.

T

time()

Temporal function that can be used in statement **compute** and in conditions.

trace on/off

Enable/disable statement execution tracing.

U

update PRODUCT **set** QonHand = 55 **where** ProdID = 'MDF-60';

Changes the value of a column (or several columns) in selected rows.

V

variableToFile newRules = Source-rules.txt;

Store the contents of variable rules into file Source-rules.txt. Inverse of **fileToVariable**.

W

wait 500;
wait 100,900;

Suspends the script execution the specified number of milliseconds. With one parameter, the waiting time is the value of this parameter. With two parameters, the waiting time is a random number in the range specified by the parameters. The parameters are integer numbers.

webfileToFile D:/Images/cat_21.jpg
= http://www.lolcat.org/35.jpg;
webfileToFile D:/Images = http://www.lolcat.org/35.jpg;

Downloads a file from the web with the specified url (http://www.lolcat.org/35.jpg). The file is stored locally with the specified file name (D:/Images/cat_21.jpg). If a directory name is specified instead (D:/Images or D:/Images/), the file keeps the name of its web source (35.jpg). Sets system variables WEBdiag, FILEdiag and EXTENDEDdiag.

webpageToVariable page = http://www.lolcat.org/home.html;

Transfers into a variable (page) the HTML code of the web page the url of which is specified (http://www.lolcat.org/home.html). Sets system variables WEBdiag and EXTENDEDdiag.

while ('\$Answer\$' <> '') **askInsert** DETAIL;

Elementary while statement, in which the body comprises one statement only. The following statement (askInsert) is executed iteratively while condition ('\$Answer\$' <> '') evaluates to *true*.

```

while ('$SQLdiag$' = 'OK');
    ask C,P,Q = [Enter an order detail] Order |
                Product | Quantity;
    insert into DETAIL values ($C$, '$P$', $Q$);
endwhile;

```

Block while statement, in which the body may comprise several statements. The statements between the while and endwhile statements are executed iteratively while the condition evaluates to true.

```
while (True);
```

Executes the body of the loop forever. To stop these iterations, use **exit** statements. Constant True can only be used in this statement.

```
with T1 as (SFWquery1), T2 as (SFWquery2), ... SFWquery;
    select query based on Common Table Expressions (CTE). A CTE ( $T_i$  as
    (SFWqueryi)) is a local view the scope of which is limited to the with
    query. Can be used to simplify complex queries and to make them more
    readable.
```

```
with T1 as (SFWquery1), T2 as (SFWquery2), ... IDUquery;
    insert, delete or update query based on CTE.
```

```
with recursive T1 as (SFWquery1), T2 as (SFWquery2), ...
SFWquery;
```

Recursive select query based on CTE.

```
with recursive T1 as (SFWquery1), T2 as (SFWquery2), ...
IDUquery;
```

Recursive insert, delete or update query based on CTE.

```
write Processing completed for customer $CustName$;
```

The character string is written on the current output channel. Command write can be suffixed by "-a", "-b" or a combination of both, to write additional blank line before (b) and/or after (a) the character string. The argument string may include control characters (@).

```
writeVariables;
```

Writes the current value of user variables on the current output channel in *Key-value* format.

```
writeSystemVariables;
```

Writes the current value of system variables on the current output channel in *Key-value* format.

writeParameters;

Writes the current state of the parameters on the current output channel in *Key-value* format.

Part 4 - SQLfast expressions

The following operators and functions can be used in the right side part of **compute** statements and in **if** and **while** conditions. They can also be used in SQL expression, but for some of them, this is only valid for SQLite.

Some of these functions are native SQL functions and operators while others are coded as *DBMS user-defined functions*. The specs and code of the latter are available in source modules `SQLiteUDFlib.py` for SQL expressions and `SQLfastUDFlib.py` for SQLfast expressions. These modules are located in SQLfast *system directory*.

In versions of SQLfast compiled for Windows by **py2exe**, it is quite easy to add new function. Let us suppose we want to add function **comp**:

- we write the code of a new function called **SQLcomp** in file **SQLfastUDFlib.py** (and in `SQLiteUDFlib.py` if desired)
- we add in function **createFunctions** of the same file(s) a new line **connect.create_function** to link the new function to the SQLfast evaluation engine; we specify three parameters: the external name **comp**, through which it will be called, the number of arguments and the internal name **SQLcomp**
- we compile module **SQLfastUDFlib.py** into **SQLfastUDFlib.pyc**, for example through a main Python program that comprises the single instruction "**import SQLfastUDFlib**"
- we insert compiled module **SQLfastUDFlib.pyc** in archive **library.zip** located in the SQLfast system directory.

From now on, SQLfast scripts can use function **comp**.

Numeric expressions

- standard operators: +, -, *, /, unary -, % (= modulo)
- `abs(n)`: absolute value of *n*. *n* is a number or a numeric string.
- `round(n)`: floating-point integral number nearest to *n*. *n* is a number or a numeric string.
- `round(n,p)`: floating-point value *n* rounded to *p* digits to the right of the decimal point. *n* is a number or a numeric string.
- `truncate(n)`: integer which is the integral part of *n*. *n* is a number or a numeric string.
- `int(n)`: the nearest integer of *n*. *n* is a number or a numeric string.
- `random()`: pseudo random integer number in range [-9223372036854775808, 9223372036854775807].
- `random_i(n1,n2)`: pseudo random integer number in range [*n1*, *n2*]
- `random_f(n1,n2)`: pseudo random floating-point number in range [*n1*, *n2*]

-
- `randomblob(n, form)`: pseudo-random string of `n` bytes in format `form` (0 = binary, 1 = hexadecimal, 2 = base64).
 - `power(n, e)`: floating point value of `n` power `p`. `n` is a number or a numeric string.
 - `sqrt(n)`: floating point square root of number `n`. `n` is a number or a numeric string.
 - `sign2(n)`: sign of `n` (two values: -1, +1). `n` is a number or a numeric string.
 - `sign3(n)`: sign of `n` (three values: -1, 0, +1). `n` is a number or a numeric string.
 - `isNumeric(s)`: whether (0, 1) string `s` represents a decimal number; `s` can also be a number
 - `isNotNumeric(s)`: opposite of `isNumeric(s)`
 - `isInteger(s)`: whether (0, 1) string `s` represents an integer value (note: **12.0** is considered non integer); `s` can also be a number
 - `isNotInteger(s)`: opposite of `isInteger(s)`

Character string expressions

- concatenation operator: `s1 || s2`
- `substr(s, i, n)`: the part of string `s` from position `i` (the first one is **1**) with maximum length `n`. If `i` is negative, the position is counted from the end of `s`. If `n` is negative, the `abs(n)` characters preceding the `i`th are returned.
- `substr(s, i)`: the part of string `s` from position `i` until the last character of `s`. If `i` is negative, the position is counted from the end of `s`.
- `substr2(s, i1, i2)`: the part of string `s` from position `i1` to position `i2`.
- `upper(s)`: upper case conversion of `s`
- `instr(s, a)`: position (in range[1,length(s)]) in `s` of the first occurrence of substring `a`; 0 if none; `s` is a string or a number
- `countInst(s, a)`: number of non overlapping instances of `a` in `s`; `s` is a string or a number
- `length(s)`: number of character in `s`; `s` is a string or a number
- `like(p, s)`: same as SQL `like` predicate
- `like(p, s, e)`: same as SQL `like` predicate where `e` is the escape character
- `lower(s)`: lower case conversion of `s`
- `ltrim(s)`: string `s` with left spaces removed
- `ltrim(s, c)`: string `s` with left characters `c` removed
- `rtrim(s)`: string `s` with right spaces removed
- `rtrim(s, c)`: string `s` with right characters `c` removed
- `trim(s)`: string `s` with left and right spaces removed

- `trim(s, c)`: string `s` where the leading and trailing characters `c` have been removed.
- `deaccent(s)`: string `s` where all accents have been removed
- `simplify(s)`: same as `lower(deaccent(s))`
- `shrink(s)`: same as `simplify(s)` + remove symbols such as space, -, _, ', etc.
- `reduce(s, c)`: string `s` in which all the sequences of several instances of each character in `c` have been reduced to one instance.
- `reduceAll(s)`: string `s` in which all the sequences of several instances of the same character have been reduced to one instance.
- `replace(v, s, r)`: string `v` where all (disjoint) instances of `s` have been replaced by `r`; all the arguments are strings or numbers.
- `hex(s)`: hexadecimal utf-8 code point string of the characters of `s`
- `doubleQuote(s)`: string `s` in which all internal single quotes have been doubled; if used in SQLfast statements, better to express `s` into *double-quotes*!
- `quote(s)`: string `s` surrounded by single-quotes and in which all internal single quotes have been doubled; if used in SQLfast statements, better to express `s` into *double-quotes*!
- `repeat(s, n)`: string built as `n` times substring `s`
- `startsWith(s, h)`: whether string `s` starts with string `h`; all strings start with the empty string (")
- `endsWith(s, n)`: whether string `s` ends with string `e`; all strings end with "
- `frame(s, n, a, p)`: string (or number) `s` framed in a `n`-long string; `a` is the alignment ('<' = left, '>' = right, '^' = center); unused positions are padded with characters `p`; `s` is truncated (right, left, both) if its length is larger than `n`.
- `thousandSep(s, sep)`: the integer part of numeric string `s` is formatted by inserting symbol `sep` between 3-digit blocks. `s` is a string or a number.
- `b64encode(s, ext)`: string `s` encoded in *base64*; parameter `ext` denotes either the standard encoding ('std') or the url-safe encoding('url'); this function is a reduced version of statement **b64encodeαToβ** in which the source is a character string.
- `b64decode(s)`: *base64* string `s` decoded into its original text; this function is a reduced version of statement **b64decodeαToβ**.
- `char2hex(s)`: string `s` encoded in *hexadecimal*; this function is a reduced version of statement **HEXencodeαToβ**.
- `hex2char(s)`: *hexadecimal* string `s` decoded into its original text; this function is a reduced version of statement **HEXdecodeαToβ**.
- `utf8Compliant(s)`: whether string `s` may be an **utf8** byte string.

- `latinCompliant(s)`: whether string `s` may be a **Latin** (e.g., cp1252) byte string.
- `hexCompliant(s)`: whether string `s` may be an **hexadecimal** character string.
- `base64Compliant(s)`: whether string `s` may be an **base64** character string.
- `hash(s)`: returns a string of 64 hexadecimal characters resulting from the secure hashing function `sha256`.
- `encrypt(s, k)`: encrypts source data `s` with key `k` then codes it into base64. Null values and empty strings are encrypted as empty strings.
- `decrypt(c, k)`: decrypts *base64* coded encrypted value `c` with key `k`.

Logical expressions

- comparison operators: `=`, `<`, `>`, `<>`, `<=`, `>=`
- logical operators: `and`, `or`, `not`
- interval: `between`, `not between`
- string patterns: `like`, `not like`
- `&`: bit string **and** operator
- `|`: bit string **or** operator
- `~`: bit string **not** operator
- `<<`: bit string left shift operator
- `>>`: bit string right shift operator

Temporal expressions

SQLfast follows the ISO format for date, time and datetime:

- *date*: `YYY-MM-DD`
 - *time*: `HH:MM:SS` and `HH:MM:SS.mmm`
 - *datetime*: `YYY-MM-DD HH:MM:SS` and `YYY-MM-DD HH:MM:SS.mmm`
-
- time registers: `current_time`, `current_date`, `current_timestamp`
 - `time(t, m*)`: see *SQLfast manual, Chapter 10*; see also <https://docs.python.org/2/library/time.html?highlight=time#module-time>
 - `date(t, m*)`: *ditto*
 - `datetime(t, m*)`: *ditto*
 - `strftime(f, t, m*)`: *ditto*
 - `julianday(t, m*)`: *ditto*. Converts *date*, *time* and *datetime* values into a julian day, allowing them to be added and subtracted.

- `jd(t)`: simplified form of `julianday` for dates.
- `addToDate(d,n)`: adds `n` days to date `d`.
- `nextDate(d)`: adds 1 day to date `d`.
- `prevDate(d)`: subtracts 1 day from date `d`.
- `current_timestamp_full()`: string value of current date and time down to milliseconds in format `YYYY-MM-DD hh:mm:ss.mmm`
- `current_date_string()`: string value of current date in format `YYYY-MM-DD`
- `current_time_string()`: string value of current time in format `hh:mm:ss.mmm`
- `time_string(t)`: string value of time `t` (in seconds) in format `hh:mm:ss.mmm`
- `clock(t)`: current time in the day in format `hh:mm:ss`
- `year(d)`: if `d` is the string expression of a date or datetime, returns the year part as an integer
- `month(d)`: if `d` is the string expression of a date or datetime, returns the month part as an integer
- `day_in_month(d)`: if `d` is the string expression of a date or datetime, returns the day part as an integer
- `hour(t)`: if `t` is the string expression of a time or datetime, returns the hour part as an integer
- `minute(t)`: if `t` is the string expression of a time or datetime, returns the minute part as an integer
- `second(t)`: if `t` is the string expression of a time or datetime, returns the second part as an integer
- `millisecond(t)`: if `t` is the string expression of a time or datetime, returns the millisecond part as an integer
- `clock(t)`: if `t` is the string expression of a date, time or datetime, returns the `hh:mm:ss` part; allows missing parts, replaced by `00`
- `isDate(s)`: whether (0,1) string `s` represents a date or a datetime value in one of the six standard patterns (`YYYY-MM-DD`, `YY-MM-DD`, `MM-DD-YYYY`, `MM-DD-YY`, `DD-MM-YYYY`, `DD-MM-YY`); other separators are allowed: `'/'` and `'.'`
- `isNotDate(s)`: opposite of `isDate(s)`
- `datePattern(d)`: one of the six standard patterns with which string `d` matches. In case of mismatch, indicates its kind (string: not a string; length: invalid length; pattern: no pattern found).
- `normDate(d,span)`: the normalized form of date `d` (**YYYY-MM-DD**). `d` must be in one of the standard patterns. Returns `' '` in case of mismatch. Parameter

span is used for 2-digit years (YY). It indicates whether the year field is to be interpreted as a *historical* value (56 means 56 DC, or 0056) or as a *present* value (56 means 1956). Its values are 'H' or 'P', case insensitive.

- `durationSeconds(t1,t2)`: number of seconds and milliseconds between time `t1` and time `t2`. The arguments are date or timestamp values.
- `durationDays(t1,t2)`: number of days (rounded to closest integer) between date `t1` and date `t2`. The arguments are date or timestamp values. Clearer but slower than the difference of `julianday` values.
- `tooverlap(s1,e1,s2,e2)`: checks whether closed-open intervals `[s1,e1)` and `[s2,e2)` overlap.

File/web functions

- `fileExists(p)`: whether string `p` is the full name of an existing file
- `dirExists(p)`: whether string `p` is the full name of an existing directory
- `urlExists(p)`: whether a website with url `p` exists
- `normalizePath(p)`: `p` where '\' replaced with '/'; spaces removed
- `lastPathNode(p)`: last node of path `p` (leading/trailing '/' excluded)
- `pathPrefix(p)`: path `p` without its last node (trailing '/' removed)
- `nodeExtension(p)`: the extension of the last node of path `p`, if any
- `pureNodeName(p)`: the last node of path `p`, minus the extension, if any
- `sendMail(Host,Login,Pw,From,To,Cc,Form,Subj,Body,Attach)`: sends a mail. `Host` is the address of the SMTP server and (`Login`, `Pw`) are logging parameters. `From`, `To` and `Cc` are mail addresses of the recipient(s), of the sender(s) and of cc(s). `Form` is the format of the message ('plain' or 'html'), `Subj` is the subject, `Body` is the body and `Attach` is the list of the addresses of the attached files.
- `hashFile(f)`: returns the hash value computed on the file with path `f`.

General functions

- `typeof(v)`: type of value `v`
- `max(v1,v2,...)`: the highest value among list `v1,v2,...`; valid with (all) numeric or (all) string arguments
- `min(v1,v2,...)`: the lowest value among list `v1,v2,...`; valid with (all) numeric or (all) string arguments
- `case when c1 then v1 when c2 then v2 ... else vn end`: same as SQL case-end function
- `cast(v as t)`: the conversion of value `v` into type `t`
- `head(s,sep)`: the first word of character string `s`, leading spaces discarded. Returns '' if none. End of word: followed by a space character or end of string

if `sep = ''`, otherwise by a character in `sep` or end of string. `s` may be empty. If `s` starts with a character in `sep`, returns this character.

- `tail(s, sep)`: character string `s` minus its head leading spaces discarded. The tail of `s` starts at the first non-`sep` (and non-space) character that follows its head. *Exception*: if the head is a character in `sep`, the tail starts with the next (non-space) character in `sep`.
- `group_concat2(v, dis, o, dir, sep)`: replacement for `group_concat` aggregate function. Values `v` are concatenated, separated by `sep`. Parameter `dis` (0, 1) specified the `distinct` modifier. Parameter `o` defines the order key and `dir` specifies the order direction (1 = ascending, 2 = descending). Only available in SQL queries. This form is equivalent to MySQL `group_concat`.
- `printMessage(s)`: write text `s` in the output window. Symbols `@n` and `@t` are converted into *end-of-line* and *tab* respectively. Return False. Can be used in SQL queries only. Mainly used in trigger body to write execution traces.
- `writeTrace(index, time, proc, level, nature, raw, prep, cond)`: If `cond` is True, insert into table `TRACING.db.TRACES` a row formed with the arguments. Can only be used to trace the execution of SQL triggers.
- `writeFile(f, s)`: create text file `f` with name and write text `s` in it. Symbols `@n` and `@t` are converted into *end-of-line* and *tab* respectively. Can be used in SQL queries only. Returns False. Mainly used in trigger body to create text files.

List functions

An SQLfast **list** is a couple of character strings (**S,sep**). **S** is made up of items separated by instances of **sep**. An item is a character string. An item can (but need not) be quoted. If an item includes instances of **sep**, it must be quoted. If an item includes quotes, the latter must be doubled and the item itself must be quoted. The default separator is defined by system variable `list-separator` (typically `' ; '`). If **sep** is empty, **S** is considered a character list.

Examples of valid lists with default separator (ignore enclosing double-quotes):

```
"A;B;C;D"
"A; 'B' ;C; 'D' "
"A; ;B;C;D; "
" "
" ; "
" ; ; "
"A; 'O' 'Hara' ;B"
"A; 'B;C;D"
```

with empty separator:

```
"1234"
```

-
- `item(s, i, sep)`: returns the *i*th item of *s*; if *i* is negative, the position is considered backward (-1 denotes the last element)
 - `itemLen(s, sep)`: returns the number of items in *s*
 - `itemIndex(s, e, sep)`: returns the position of first item *e*. Returns 0 if *e* is absent.
 - `itemSort(s, dir, dis, sep)`: returns this list where the items have been sorted in ascending (*dir* = 0) or descending order (*dir* = 1). Duplicate items have been removed if *dis* = 1, and preserved if *dis* = 0. The ordering is alphanumeric: '12' > '112'.
 - `itemNSort(s, dir, dis, sep)`: same as `itemSort` with numeric items: '12' < '112'.
 - `itemSlice(s, i1, i2, sep)`: returns the sublist comprising the items with rank *i1* to *i2*. If both *i1* and *i2* are negative, with *i1* ≤ *i2*, the ranks are considered backward (-2, -1 denote the last two items).
 - `itemDelete(s, e, sep)`: returns *s* in which all the items that are instances of *e* have been removed.
 - `itemJoin(s1, s2, sep1, sep2, sep3)`: *s1* and *s2* are built with separator *sep1*. Returns list *s12* (with separator *sep3*), each element of which is the concatenation of the elements of same rank in *s1* and *s2* with separator *sep2*.
 - `itemUnion(s1, s2, sep)`: returns the *union* of *s1* and *s2*, regardless their order and ignoring duplicates. Same as SQL set *union*.
 - `itemUnionAll(s1, s2, sep)`: returns the *union* of *s1* and *s2*, regardless their order but considering duplicates. Same as SQL multiset *union all*.
 - `itemExcept(s1, s2, sep)`: returns the *difference* of *s1* and *s2*, regardless their order and ignoring duplicates. Same as SQL set *except*.
 - `itemExceptAll(s1, s2, sep)`: returns the *difference* of *s1* and *s2*, regardless their order but considering duplicates. Same as SQL multiset *except all*.
 - `itemInter(s1, s2, sep)`: returns the *intersection* of *s1* and *s2*, regardless their order and ignoring duplicates. Same as SQL set *intersect*.
 - `itemInterAll(s1, s2, sep)`: returns the *intersection* of *s1* and *s2*, regardless their order but considering duplicates. Same as SQL multiset *intersect all*.
 - `itemEqual(s1, s2, sep)`: tells whether *s1* and *s2* include the same items, regardless their order and ignoring duplicates (set equality).
 - `itemEqualAll(s1, s2, sep)`: tells whether *s1* and *s2* include the same items, regardless their order but considering duplicates (multiset equality).
 - `itemInclude(s1, s2, sep)`: tells whether *s1* includes all the items of *s2*, regardless their order and ignoring duplicates (set inclusion).
 - `itemIncludeAll(s1, s2, sep)`: tells whether *s1* includes all the items of *s2*, regardless their order but considering duplicates (multiset inclusion).

- `itemTranslate(s,p,r,sep)`: returns string `s` in which the items of `p` are replaced by the corresponding items of `r` (similar to the SQL `translate` function with lists instead of characters).
- `itemS(s,i)`: same as `item(s,i,sep)` with the default separator.
- `itemLenS(s)`: same as `itemLen(s,sep)` with the default separator.
- `itemIndexS(s,e)`: same as `itemIndex(s,e,sep)` with the default separator.
- `itemSortS(s,dir,dis)`: same as `itemSort(s,dir,dis,sep)` with the default separator.
- `itemNSortS(s,dir,dis)`: same as `itemNSort(s,dir,dis,sep)` with the default separator.
- `itemSliceS(s,i1,i2)`: same as `itemSlice(s,i1,i2,sep)` with the default separator.
- `itemDelete(s,e)`: same as `itemDelete(s,e,sep)` with the default separator.
- `itemUnionS(s1,s2)`: same as `itemUnion(s,e,sep)` with the default separator.
- `itemUnionAllS(s1,s2)`: same as `itemUnionAll(s,e,sep)` with the default separator.
- `itemExceptS(s1,s2)`: same as `itemExcept(s,e,sep)` with the default separator.
- `itemExceptAllS(s1,s2)`: same as `itemExceptAll(s,e,sep)` with the default separator.
- `itemInterS(s1,s2)`: same as `itemInter(s,e,sep)` with the default separator.
- `itemInterAllS(s1,s2)`: same as `itemInterAll(s,e,sep)` with the default separator.
- `itemEqualS(s1,s2)`: same as `itemEqual(s,e,sep)` with the default separator.
- `itemEqualAllS(s1,s2)`: same as `itemEqualAll(s,e,sep)` with the default separator.
- `itemIncludeS(s1,s2)`: same as `itemInclude(s,e,sep)` with the default separator.
- `itemIncludeAllS(s1,s2)`: same as `itemIncludeAll(s,e,sep)` with the default separator.
- `itemTranslateS(s,p,r)`: same as `itemTranslate(s,p,r)` with the default separator.

SQLfast function libraries

SQLfast libraries are user-defined Python modules comprising specific functions that can be called through **execApp** or **function** statements. Most generally, they form complex processing units, possibly including database and user interface actions, and they are able to return several results. Some of them are provided in the standard SQLfast distribution.

Please note that the simplest functions from these libraries are progressively migrated in the SQLfastUDFlib.py and SQLlitetUDFlib.py modules while others have been translated into plain SQLfast instructions. Therefore, some functions may be provided by more than one source. This, of course, may change in the future.

The specifications and the code can be found in the modules that are included in the SQLfast distribution.

LStr library

String manipulation functions. Most of them have already been converted into UDF functions.

- Replace s,s1,s2:
- Upper s
- Lower s
- Capitalize s
- UnFrench s
- French s
- DeAccent s
- ExtractString1 s,pos,lng
- ExtractString2 s,pos1,pos2
- ExtractLeft s,n
- ExtractRight s,n
- RemoveRight s,n
- RemoveLeft s,n
- InString s,arg,pos
- NumToChar i
- CharToNum s
- Length s
- SQLunquote s
- SQLquote s
- SQLquote2 s

Printed 12/12/22

- SQLquote3 *s*
- KeyValue *s*
- NextSection *text, pos, arg1, arg2, arg3*: searches string *text* from position *pos* (first character is at position 0) for the smallest fragment (called a *section*) between substrings *arg1* and *arg3* (those included). Embedded subfragments between *arg2* and *arg3* are preserved. Returns the section found and the position of the first character following *arg3*. *pos* = -1 if no section has been found.
- ExtractTaggedString *text, arg1, arg2*: returns the first fragment of string *text* between *arg1* and *arg2* (those excluded). Returns an empty string if no such fragment has been found.
- PrettyPrint *query, linebreak*: returns the input SQL select query formatted in a clear and structured layout. Line breaks are specified by parameter *linebreak*, allowing value *eol* (hard new-line according to the platform) or any arbitrary character string. In case of problem, returns the input query.
- *etc.*

LStr: cryptographic functions

- hash *S, form*: returns the hash value of byte string *S*. Uses the sha256 algorithm. If *form* = 1, delivers the hash value in hexadecimal format. If *form* = 2, delivers it in base64.
- miningHash *S, prefix, hashNbr, form*: mines byte string *S* by computing the first hash the binary value of which starts with the 0-bitstring specified by argument *prefix*. E.g., *prefix* = '00000' means that the hash must start with at least 5 zero-bits. The hashes are computed for the successive values of *nonce* from 0 to *hashNbr*.

Returns *hash*, the hash value found, *nonce*, the nonce of this hash (as a numeric character string) and *Nhash* the number of hashes computed. If *nonce* = '' and *Nhash* > 0, then no valid hash has been found. In this case, it is necessary to increase the value of *hashNbr*.

The format of hash is defined by argument *form*, as for hash function.

The execution time of this function may be large. On a typical personal computer, the function computes about 50,000 hashes per second for string *S* of 3,600 bytes. Experimentally, a valid hash for this byte string with a prefix of 20 zero-bits is found in 3 to 30 seconds. In real life, a valid hash with a prefix of 64 zero-bits is computed in about 10 minutes by a cluster of several thousands processors.

- generateSYMkey *n*: generates a random key of *n* bytes and returns it in base64 format. Is particularly used to create a key for symmetric encryption.
- encryptSYM *message, symKey*: *symmetric encryption*: encrypts byte string *message* with key *symkey* and returns it converted in base64.

- `decryptSYM crypto,symKey`: *symmetric decryption*: decrypts base64 character string `crypto` with key `symkey` and returns it as a byte string.
- `generateRSAkeys n`: *RSA asymmetric encryption*: generates public and private, a pair of keys for asymmetric encryption. The public key is approximately `n` bits long. The keys are converted in the PEM format, with their header footer and end-of-lines removed.
- `encryptRSA message,pubkey`: *RSA asymmetric encryption*: encrypts byte string `message` with public key `pubkey` and returns it converted in base64. Note that, when encrypting a transaction in a blockchain, this key is called secret key.
- `decryptRSA crypto,privkey`: *RSA asymmetric encryption*: decrypts base64 character string `crypto` with private key `privkey` and returns it as a byte string. Note that, when decrypting a transaction in a blockchain, this key is called public key.

LFile library

File and directory manipulation functions. Most of its functions have been moved to SQLfast statements or integrated into UDF functions.

- `splitPath p`: converts and decomposes path `p` into its significant fragments: (1) full (absolute) name, (2) prefix (full name minus last node), (3) local directory name (last node if directory path, otherwise ''), (4) local file name (last node if file path, otherwise ''), (5) extension if file path, otherwise ''.
- If path `p` denotes a **drive root** (e.g., 'F: / '), then both (3) and (4) are empty.
- If `p` denotes **no object**, tries to guess these components from the syntax of `p`. In case of empty or invalid path, returns a series of five empty strings.
- `infoFile p`: If `p` denotes **an existing file**, returns (1) its size in bytes, (2) its creation time, (3) its last update time and (4) its last access time. Otherwise returns four empty string.

LWeb library (currently disabled [January 2019])

Web-based manipulation functions. Allows files and web pages downloading and sending messages through SMTP protocol.

- `WebFileToFile url,timeout`: downloads the web file with address `url` and stores it in a local file. Also available as SQLfast statement `webfileToFile`.
- `WebPageToVariable p`: stores the HTML code of the web page with address `url` in a variable. Also available as SQLfast statement `webpageToVariable`.

- MailConnectSMTPserver *host, log, pw*: opens a connection with SMTP server **host**, through login **log** and password **pw**.
- MailSend *from, to, cc, for, subj, body, attach*: sends a mail through the SMTP connection currently opened. The mail is specified by its sender **from**, the list of its recipients **to**, the list of its recipients **cc**, its format **for** (*plain* or *html*), its subject **subj**, the name of the file that contains its body **body** and the addresses of the files attached **attach**.
- MailCloseSMTPserver: closes the connection currently opened.
- MailSendImmediate *host, log, pw, from, to, cc, for, subj, body, attach*: combination of MailConnectSMTPserver, followed by MailSend and followed by MailCloseSMTPserver. Does not use the connection currently opened but creates, uses and closes its own private connection. Also available as UDF function mailSend that can be used in SQL triggers.
- disableCertificate: disable certificate verification. To use with MUCH caution.

LTemp library

Complex operators for temporal database processing (see Section 10).

- project {<project-query>}: temporal projection of a temporal table.
- reduce {<reduce-query>}: reduction (coalescing) of a temporal table.
- valuedInterval {<interval-query>}: generates the valued stable intervals of a temporal table.
- valuedIntervalE {<interval-query>}: same as valuedInterval with more general expressions (allows table aliases).
- normalize {<normalize-query>}: complete a temporal table by replacing missing intervals with *null* states.

LGeo library

Geometric object manipulation functions. These functions are mainly used by the the application chapter of the tutorial devoted to *Geographic information systems*. They will be released and documented when this chapter is available.

- ConvertPolylineToPolygon *st*
- DistancePointPoint *st1, st2*
- LengthOfSegment *st*
- DistancePointLine *st1, st2*
- DistancePointSegment *st1, st2*
- AreaOfPolygon *st*

- LengthOfPolyline st
- PerimeterOfPolygon st
- Colinear st1,st2,st3
- PointOnSegment st1,st2
- PointInPolygon st1,st2
- ShapeFileLoad shapeFileName
- ShapesNumber shapes
- ShapesBbox shapes
- ShapesType shapes
- ShapesExtract shapes,shNum
- ShapesNormalize shapeList,minX,minY,maxX,maxY
- ShapesToDrawFile shapeList,fName
- ShapeMorphing shapes,st1,st2
- stringToList S
- listToString L
- *etc.*

Part 5 - System variables

The system variables are predefined variables that control and describe the behavior of scripts and of the SQLfast system. Many of them are set in initialization file **SQLfast.ini**. Except for the *Start parameters*, that are hidden, most of them can be read by scripts and even modified, to better control the way SQLfast and scripts behave.

Start parameters

These variables define the general behavior of SQLfast. They are set by initialization file SQLfast.ini. They cannot be read nor modified by scripts.

userLevel

Specifies the SQLfast interface at start time. Values 0 (Basic), 1 (Expert). Default value **0**.

welcome

Indicates whether the welcome windows is displayed at start time. Default value **1**.

commitmode

Specifies the default commit mode for database opening statements (`createDB`, `createOrReplaceDB`, `openDB`). Values: `transaction` (commit must be explicitly executed), `autocommit` (each modification query is automatically committed). Can be modified through the `parameter` statement.

autoclosedb

Indicates whether an opened database must be automatically closed when (1) the current SQLfast session terminates (`session`), (2) when the current main script terminates (`mainscript`) or (3) when the current script, main or secondary, terminates (`script`). The default value is `mainscript`.

autoexec

If not empty, specifies the name of a script that is executed automatically when SQLfast interpreter is started. To hide the SQLfast environment, this script can include a `shutDown` statement.

DBMS

Name of the DBMS used by SQLfast.

DBMSversion

Current version of the DBMS used by SQLfast.

dictionary

Indicates whether a local data dictionary must be created when a database is opened or created (0 = no, 1 = yes). Default 0. This dictionary can also be dynamically built through statement **createDictionary**.

maxProcLevel

Indicates the maximum number of simultaneous active processes. In recursive scripts, can be used to define the maximum depth of recursive calls. Default 101.

maxIVV

Indicates the maximum number of SQLfast variables.

outputtype

Specifies the default output channel. Values: standard, console, window. Default window.

outputmode

Specifies the default output mode for statements that write on the output channel. Values: write, append. Default append.

traceActive

Activates/disables tracing (1, 0).

traceSQL

Limits traces to SQL queries and other database statements (0 = no, 1 = yes).

traceAppend

Adds future traces to those already recorded (0 = replace, 1 = append).

loaderDebug

Prints in the output window the result of the precompilation of complex statements (0 = no trace; 1 = compiled script; 2 = compiled script + store it in an executable script file; 3 = source script, compiled script, label dictionary).

Default directories

SQLfast uses standard directories in which its resources are stored. Their names are specified by the following system variables. They can be read and modified by script.

generalDirectory

Defines the general directory in which the other directories (script, output, file, database, help) normally are created. The name is either a full name or keyword **_system**, in which case the general directory is that of the SQLfast.exe application. Default value **_system**.

dbDirectory

Defines the directory where the databases files are stored. The name is either a full name or a relative name. If a relative name is given, the database directory is located in the general directory. Default value **Databases**.

scriptDirectory

Defines the directory where the scripts are stored. The name is either a full name or a relative name. If a relative name is given, the script directory is located in the general directory. Default value **Scripts**.

outputDirectory

Defines the directory where the output files are stored. The name is either a full name or a relative name. If a relative name is given, the script directory is located in the general directory. Default value **Output**.

fileDirectory

Defines the directory where the external files (other than output files) are stored. The name is either a full name or a relative name. If a relative name is given, the file directory is located in the general directory. Default value **Files**.

helpDirectory

Defines the directory where the help and tutorial documents are stored. The name is either a full name or a relative name. If a relative name is given, the help directory is located in the general directory. There are two default help directories: SQLfastHelp_EN and SQLfastHelp_FR. The first one contains the English documents while the second one contains their French translation.

Character encoding schemes

These variables specify the code used to represent characters in various parts of SQLfast. These variables can be read and modified by scripts.

`coding_db`

Indicates the character encoding standard of the DBMS.

`incoding_script`

Indicates the character encoding standard of SQLfast scripts.

`incoding_file`

Indicates the character encoding standard of text input files. Must be the first line of SQLfast.ini, since it defines the way this file will be decoded.

`outcoding_file`

Indicates the character encoding standard of text output files.

`incoding_dbf`

Indicates the character encoding standard of dBase input files. This variable can be updated by the script.

`coding_dialog`

Indicates the character encoding standard in dialog boxes.

Size parameters of the main interfaces

These variables specifies the default size of the main window, the output window and the help window. They cannot be read nor modified by scripts.

`SwinY`

Default number of lines of the script area of the main window.

`OwinY`

Default number of lines of the output area.

`OwinX`

Default width (number of characters) of the output area.

`HwinY`

Default number of lines of the help window.

`HwinX`

Default width (number of characters) of the help window.

tutoImageMaxSize

In tutorials, maximum size of pictures (in pixels). If larger, pictures will be reduced.

charSize

Size of the characters in script and output windows.

Parameters of Help windows

The color rendering of Help and Tutorial documents can be very different depending on the screen settings. These system variables can be used to adapt these colors to the reader preferences.

colorHelp

Background color of the Help and Tutorial documents (default grey96).

colorHelpScript

In Help and Tutorial documents, background color of the embedded scripts.

Parameters of the Data windows

These variables specifies the characteristics of the **Data windows** created by the **Show Data** function. They can be read and modified by scripts.

maxTableNbr

In *Show data* function, maximum number of tables displayed before table selection.

WmaxRowWidth

In *Show data* function, maximum width of data windows.

WmaxRowNbr

In *Show data* function, maximum height of data windows.

maxRowNbr

In *Show data* function, maximum number of rows extracted in each data window.

maxShowDataWidth

Specifies the maximum width of columns in data windows.

showDataBg

Specifies the background color and pattern of Data windows. The background appears as alternate white and color strips.

- if color = #FFFFFF: uniform white
- if no specification: alternate white and 'boldBlue' strips
- if other color: alternate white and color strips. Color suggestions: #FDF5E6, #FFF5EE, #FFEB CD, #F0F0F0, #FAFAD2, pale green, pink, etc.

Data display parameters

These variables specify how resultsets from **select** queries must appear on the output channel. They can be read and modified by scripts.

dispNull

Indicates how SQL **null** values will appear on output channels.

eolInGUI

Value that will replace end-of-line characters (eol) in text extracted from the database and displayed in some GUI's. Indeed, eol's scramble the layout in the following displays:

- result set of **select** queries in the output window
- the data window created through the *Show Data* menu item and button
- the form displayed when double-clicking of a row in the data window.

When the substitute value is **eol**, then no replacement is performed.

maxSelectWidth

Specifies the maximum width of columns in the result set of **select** statements.

select-align

Indicates whether the result of **select** statements must be aligned (i.e. displayed in a tabular form) on the output channel or not. Must be modified through the **parameter** statement. Values: Y, N, default.

select0, ..., select4

Source variables of string control characters @1 to @4. These variables can be updated by the script.

selectA0, selectAi

Initial value and increment of string control characters **@A**. These variables can be updated by the script.

selectIntro, selectClose, bar11, ..., bar21, ..., bar31, ..., head1, ..., data1, ...

Formatting variables for the **select** SQL statement. These variables can be updated by the script.

Parameters of the dialog boxes

Through these variables, various aspects of dialog boxes can be defined. Unless indicated otherwise, they can be read and modified by scripts.

entryWidth1

In *askData* and *showData* boxes, number of characters of data entry fields. Valued from parameter **entryWidth**. Can also be modified through the `parameter` statement.

entryWidth2

In *askData* boxes, number of characters of data entry fields with predefined value list. Valued from parameter **entryWidth**. Can also be modified through the `parameter` statement.

buttonYES

Contains the label of the *positive* button of dialog boxes (default **OK**). This variable can be updated by the script. If empty, the positive button will not appear in next boxes.

buttonNO

Contains the label of the *negative* button of dialog boxes (default **Cancel**). This variable can be updated by the script. If empty, the negative button will not appear in next boxes.

label-separator

Specifies the symbol that will be used to separate labels in *ask*, *selectOne*, *selectMany* and *askCombo* statements. Must be modified through the `parameter` statement.

Widget styles

styleMainFrame

Specifies the style of the general frame of composite boxes (askCombo): flat or relief.

styleGeneralMessage

Specifies the style of the general message field of composite boxes (askCombo): flat or relief.

styleBoxFrame

Specifies the style of the frame of elementary boxes (ask, showData, selectOne, etc.): flat or relief.

styleBoxMessage

Specifies the style of the message field of elementary boxes: flat or relief.

styleInfoBox

Specifies the style of the message boxes (pause, showMessage): flat or relief.

styleEntry

Specifies the style of the data entry fields in elementary boxes (ask, showData): flat or relief.

styleText

Specifies the style of the text, picture and drawing fields in elementary boxes (askText, showText, showPicture, showDrawing): flat or relief.

styleCheck

Specifies the style of the check buttons in elementary boxes (selectMany): flat or relief.

styleRadio

Specifies the style of the radio buttons in elementary boxes (selectOne): flat or relief.

styleButton

Specifies the style of the control buttons (OK, Cancel, etc.): flat or relief.

Widget colors

colorMainFrame

Specifies the background color of the general frame in composite boxes (`askCombo`). The color is specified either by its RGB composition (`#RRGGBB`) or by its symbolic name (`yellow`).

colorGeneralMessage

Specifies the background color of the general message field in composite boxes (`askCombo`).

colorBoxFrame

Specifies the background color of the frame of elementary boxes (`ask`, `showData`, `selectOne`, etc.).

colorBoxMessage

Specifies the background color of the message field of elementary boxes (`ask`, `showData`, `selectOne`, etc.).

colorInfoBox

Specifies the background color of the message boxes (`pause`, `showMessage`).

colorEntry = `bgE, labE, titE`

Specifies the colors of the frame, labels and titles of data entry boxes (`ask`).

- **bgE**: background color of the frame
- **labE**: color of the labels
- **titE**: color of titles

colorEntryField = `bgF, txtF`

Specifies the colors of the data entry fields (`ask`).

- **bgF**: background color of the field
- **txtF**: color of the value

colorEntryDisplay

Specifies the background color of the data display fields (`showData`).

colorTextDisplay

Specifies the background color of the text, picture and drawing fields (`showText`, `showPicture`, `showDrawing`).

colorTextUpdate

Specifies the background color of the text, picture and drawing fields (`askText`).

colorCheck

Specifies the background color of the check buttons (`selectMany`) .

colorRadio

Specifies the background color of the radio buttons (`selectOne`).

colorButton

Specifies the background color of the control buttons (OK, Cancel, etc.).

Parameters of info boxes (`showMessage`)**infoFont**

System variable that specifies the font and the size in which the text of the message appears (e.g., `verdana, 9`).

infoWeight

System variable that indicates whether the text appears in normal (0) or bold-face (1) weight.

infoWrap

System variable that indicates whether the text is wrapped (1) or not (0).

Parameters of text boxes**stdTextX**

Specifies the default width (in characters) of the text box of the **askText** and **showText** statements. This parameter is also a variable that can be set by the script.

stdTextY

Specifies the default height (in lines) of the text box of the **askText** and **showText** statements. This parameter is also a variable that can be set by the script.

Parameters of picture box in showPicture

maxPictureX

Specifies the maximum width of the picture box of the **showPicture** statement. This parameter is also a variable that can be set by the script.

maxPictureY

Specifies the maximum height of the picture box of the **showPicture** statement. This parameter is also a variable that can be set by the script.

Parameters of drawing box in showDrawing

defaultCanWidth

In *showDrawing* statement, default *drawing canvas* width.

Note: The *drawing canvas* is the space in which lines and shapes are drawn. All or a part in the drawing canvas appears in the *drawing window*. This parameter is also a variable that can be set by the script

defaultCanHeight

In *showDrawing* statement, default *drawing canvas* height. This parameter is also a variable that can be set by the script

minCanWidth

In *showDrawing* statement, minimum *drawing canvas* width. This parameter is also a variable that can be set by the script

minCanHeight

In *showDrawing* statement, minimum *drawing canvas* height. This parameter is also a variable that can be set by the script

defaultCanColor

In *showDrawing* statement, default *drawing canvas* color. This parameter is also a variable that can be set by the script

minWinWidth

In *showDrawing* statement, minimum width of the *drawing window*. This parameter is also a variable that can be set by the script

defaultWinWidth

In *showDrawing* statement, default width of the *drawing window*. This parameter is also a variable that can be set by the script

maxWinWidth

In *showDrawing* statement, maximum width of the *drawing window*. This parameter is also a variable that can be set by the script

minWinHeight

In *showDrawing* statement, minimum height of the *drawing window*. This parameter is also a variable that can be set by the script

defaultWinHeight

In *showDrawing* statement, default height of the *drawing window*. This parameter is also a variable that can be set by the script

maxWinHeight

In *showDrawing* statement, maximum height of the *drawing window*. This parameter is also a variable that can be set by the script

Default external media file processors

These variables specify the default software that must be called to process the main types of media files. They can be read and modified by scripts.

TEXTprocessor

Specifies the default foreign program to display and process pure text files. The path expression is automatically adapted to the platform (e.g., '/' replaced by '\' on a Windows platform). This parameter is also a variable that can be set by the script.

WSprocessor

Specifies the default foreign program to display and process worksheet/spreadsheet files. This parameter is also a variable that can be set by the script. The path expression is automatically adapted to the platform.

DOCprocessor

Specifies the default foreign program to display and process document files. This parameter is also a variable that can be set by the script. The path expression is automatically adapted to the platform

IMAGEprocessor

Specifies the default foreign program to display (and process) image files. This parameter is also a variable that can be set by the script. The path expression is automatically adapted to the platform

PDFprocessor

Specifies the default foreign program to display *pdf* files. This parameter is also a variable that can be set by the script. The path expression is automatically adapted to the platform

HTMLprocessor

Specifies the default foreign program to display web pages and other similar materials. This parameter is also a variable that can be set by the script. The path expression is automatically adapted to the platform

VIDEOprocessor

Specifies the default foreign program to play (and process) video files. This parameter is also a variable that can be set by the script. The path expression is automatically adapted to the platform

SOUNDprocessor

Specifies the default foreign program to play (and process) audio files. This parameter is also a variable that can be set by the script. The path expression is automatically adapted to the platform

SQLfast lists**listSeparator**

Specifies the default character string used to separate the elements of SQLfast lists.

itemSpeed

Indicates whether validation and conversion preprocessing must be performed (0) or disabled (1).

Constant lists**constSeparator**

Specifies the default character string used to separate the elements of constant lists in `askData`, `selectList` and `selectList2` dialog boxes. Same for the result/source list of `selectList` and `selectList2` (*new* and *update* modes).

Dynamic columns

dynaColumns

Indicates whether scripts may include references to dynamic columns. Values: 0 (do not process dynamic column references), 1 (process dynamic column references).

dynaMarker

Specifies the prefix character used to mark dynamic column references.

Import/Export parameters

These variables control the import of external data. They can be read and modified by scripts.

csv-delimiter

Specifies the value separator when reading and writing csv data.

csv-header

Specifies whether the first line of the csv file provides a list of column names.

csv-maxerrornbr

Specifies the maximum number of data and insert errors beyond which the loading is aborted and insertions are rolled back.

csv-mute

Enable (0), disable (1) import error messages in the output channel.

Execution diagnosis registers

These system variables provide information on the current state of the execution of scripts. They can be read by scripts. Though this is not recommended, they can also be updated by scripts (except for `procLevel`).

dbName

Local name of the current database. Set when a database is created, opened, closed, deleted. This variable can be updated by the script.

dbNameFull

Full name of the current database (absolute path + local name). Set when a database is created, opened, closed, deleted. This variable can be updated by the script.

scriptName

Local name of the current script. Set when a script is started and closed. This variable can be updated by the script.

scriptNameFull

Full name of the current script (absolute path + local name). Set when a script is started and closed. This variable can be updated by the script.

mainScriptName

Local name of the main script. Set when a script is started *from the main window* and closed. This variable can be updated by the script.

mainScriptNameFull

Full name of the main script (absolute path + local name). Set when a script is started *from the main window* and closed. This variable can be updated by the script.

Error

Indicates whether the last statement executed successfully (0) or not (1).

SQLdiag

System variable containing the DBMS-independent diagnosis of the last SQL query execution. States whether the execution was successful ('OK') or not (others). This variable can be updated by the script. Values:

- OK: the requested object has been found
- NONE: the requested object has not been found
- DB: error related to the DB as a whole: DB already opened, DB not opened, DB already exists, etc.
- SYNTAX: syntax error
- NULL: not null constraint violation
- VALUE: invalid value
- ID: uniqueness constraint violation (primary and unique keys)
- FK: referential constraint violation (foreign key)
- SCHEMA: reference to unknown table or column
- SYSTEM: file system error
- ERROR: other errors.

EXTENDEDdiag

For DB-related statements, DBMS dependent error message. To examine when SQLdiag is different from OK, meaningless otherwise. Other statements can set this variable as well. This variable can be updated by the script.

FILEdiag

Diagnosis after a file related operation. Values: OK, NONE, ERROR. This variable can be updated by the script.

WEBdiag

Diagnosis after a web related operation. Values: SYNTAX, OK, NONE, ERROR, DECODE. This variable can be updated by the script.

ImportErrors

Indicates the number of critical errors detected in the last `import` process.

DIALOGbutton

Indicates which button was clicked on when the last dialog box was closed. Standard values: OK, Cancel, etc. This variable can be updated by the script.

procLevel

Indicates the level number of the current process. The *main script* is started from the SQLfast graphical interface (buttons Run, Run script, Select & Run, Run next) or as autoexec process. The process of main script has a level of 0. The other processes are assigned the level of their parent process + 1. The level is limited by parameter `maxProcLevel` set in file `SQLfast.ini`. Cannot be updated by the script.

Environment and time system registers

Can be read but cannot be updated by scripts.

date

Indicates the current date. Format: `YYY-MM-DD`. Cannot be updated by the script.

time

Indicates the current time in the day. Format: `HH:MM:SS.mmm`. Cannot be updated by the script.

clock

Indicates the current time in the day. Format: `HH:MM:SS`. Cannot be updated by the script.

dateclock

Indicates the current date + clock. Format: `YYY-MM-DD HH:MM:SS`. Cannot be updated by the script.

timer

Indicates the time spent since the last time the timer was reset. Format: `S...SS.mmm`. Can be updated by the script through statement `start-timer`.

current_date, current_time, current_timestamp

Temporal system variables that can be used in statement `compute` and in conditions.

platform

Identification of the operating system on which the interpreter is running. Set at start time. This variable cannot be updated by the script. Values (as by 2015) include the following keywords: `win32`, `win64`, `linux`, `darwin` (Mac OS), etc. (value provided by Python `os.platform` system function).

sysPath

Path of the SQLfast software. Set at start time. This variable cannot be updated by the script (value provided by Python `os.path` system function).

Miscellaneous

delimiters

Indicates the left and right delimiters used to denote the substitution operator. The right side delimiter is optional. Cannot be read and must be modified through the `parameter` statement.

timeout

Number of seconds statements `webpageToVariable` and `webfileToFile` will wait for the answer of a website before considering this site is unavailable. This variable can be read and updated by the script.

Part 6 - Parameter settings

The behavior of some aspects of the SQLfast processor is defined by a set of system parameters and variables. Their values are set in file SQLfast.ini. When this file is missing, or when it does not initialize some parameters, default values are assigned at start time. These parameters form a subset of the system variables described in Section 5.

The current state of the parameters and system variables can be written on the output channel with SQLfast statement **writeParameters**;

The following is a typical contents of **SQLfast.ini** file

```
-- -----
-- Initial standard parameters for SQLfast --
-- -----

--NOTES:
-- Parameter names are case insensitive (userlevel = USERLEVEL =
userLevel)

-- "incoding_file" MUST BE THE FIRST parameter setting of this
file.
-- It defines the encoding scheme according to which the rest of
-- the file have to be decoded.

incoding_file = cp1252

-- -----
-- Start parameters --
-- -----
-- User level (0,1) --
userLevel = 0
-- Display welcome window? --
welcome = 1
-- At Level 1, when to commit (autocommit,transaction) --
commitmode = transaction
-- At Level 1, when a DB is automatically closed
-- (script,mainscript,session) --
autoclosedb = mainscript
-- At Level 1, autoexec script file --
--autoexec = autoexec.sql
-- Internal DB manager --
DBMS = SQLite
-- Data dictionary creation on open DB
dictionary = 0
-- Maximum number of active processes
maxProcLevel = 101
-- Maximum number of variables --
maxIVV = 100000
-- Output channel --
outputtype = window
outputmode = write
-- tracing parameters --
traceActive = 0
traceSQL = 0
```

Printed 12/12/22

```

    traceAppend = 0
    loaderDebug = 0

-- -----
-- Default directories (must be defined in this order) --
-- -----
-- symbolics: _system, _general, _database, _script, _output, _file
generalDirectory = _system
dbDirectory      = Databases
scriptDirectory  = Scripts
outputDirectory  = Output
fileDirectory    = Files
helpDirectory    = SQLfastHelp_EN

-- -----
-- Character encoding schemes
-- -----
coding_db        = utf8
incoding_script  = cp1252
-- incoding_file   = cp1252
outcoding_file   = cp1252
coding_dialog    = utf8
incoding_dbf     = cp850

-- -----
-- Size parameters of the Main interfaces
-- -----
-- Initial size of Script window
SwinY = 12
-- Initial size of Output window
OwinY = 20
OwinX = 88
-- Initial size of Help window + Max picture size
HwinY = 34
HwinX = 106
tutoImageMaxSize = 980
-- Character size
charSize = 9

-- -----
-- Parameters of Help window
-- -----
-- Background color of the Help window
colorHelp = grey96
-- Background color of the embedded scripts
colorHelpScript = light yellow

-- -----
-- Parameters of the Data window
-- -----
-- Maximum number of tables before selection
maxTableNbr = 6
-- Maximum width of table windows
WmaxRowWidth = 255
-- Maximum height of table windows
WmaxRowNbr = 36
-- Maximum number of rows extracted
maxRowNbr = 99999
-- Maximum width of a column --
maxShowDataWidth = 64
-- Data windows: uniform background = 'none'
-- showDataBg = #FDF5E6

```

```

-- alternate colors = #FDF5E6,#FFF5EE,#FFEBCD,F0F0F0,#FAFAD2,
-- pale green,pink,etc.

-- -----
-- Data display parameters
-- -----
-- How to display nulls on output channel --
dispNull = --
-- Replacing "eol" in GUI select, showdata,
-- and row entry/display forms --
eolInGUI = @@n
-- Maximum width of a column --
maxSelectWidth = 64
-- Whether values of the select-list must be aligned or not --
select-align = Y

-- -----
-- Parameters of the dialog boxes
-- -----
-- Width of askData fields --
entryWidth = 30
-- Names of the two standard buttons --
buttonNO = Cancel
buttonYES = OK
-- Symbol used to separate labels in right part of dialog boxes --
label-separator = |
-- Widget styles --
styleMainFrame      = flat
styleGeneralMessage = flat
styleBoxFrame       = flat
styleBoxMessage     = flat
styleInfoBox        = flat
styleEntry          = flat
styleText           = flat
styleCheck          = flat
styleRadio          = flat
styleButton         = flat
-- Widget colors --
colorMainFrame      = grey90
colorGeneralMessage = grey85
colorBoxFrame       = grey90
colorBoxMessage     = grey85,black
colorInfoBox        = light yellow,black
colorEntry          = grey90,black,black
colorEntryField     = white,blue
colorEntryDisplay   = light yellow,black
colorTextUpdate     = white
colorTextDisplay    = light yellow
colorCheck          = grey90,black,black
colorRadio          = grey90,black,black
colorButton         = grey85,black

-- -----
-- Parameters of info boxes
-- -----
-- Default font --
infoFont = verdana,9
-- Default weight --
infoWeight = 1
-- Default wrapping --
infoWrap = 1

```

```

-- -----
-- Parameters of text boxes
-- -----
-- Default text box width --
stdTextX = 64
-- Default text box height --
stdTextY = 20

-- -----
-- Parameters of picture box in showPicture
-- -----
-- Default maximum box width --
maxPictureX = 800
-- Default maximum box height --
maxPictureY = 600

-- -----
-- Parameters of drawing box in showDrawing
-- -----
-- Minimum canvas width --
minCanWidth = 100
-- Minimum canvas height --
minCanHeight = 60
-- Default canvas width --
defaultCanWidth = 800
-- Default canvas height --
defaultCanHeight = 600
-- Default canvas background color --
defaultCanColor = white

-- Minimum window width --
minWinWidth = 100
-- Minimum window height --
minWinHeight = 60
-- Maximum window width --
maxWinWidth = 1000
-- Maximum window height --
maxWinHeight = 900
-- Default window width --
defaultWinWidth = 800
-- Default window height --
defaultWinHeight = 600

-- -----
-- Default external media file processors
-- -----
TEXTprocessor = "C:/WINDOWS/notepad.exe"
WSprocessor = "C:/Program Files (x86)/Microsoft Office/
Office10/EXCEL.EXE"
DOCprocessor = "C:/Program Files (x86)/Microsoft Office/
Office10/WINWORD.EXE"
IMAGEprocessor = "C:/Program Files/IrfanViews/i_view64.exe"
PDFprocessor = "C:/Program Files (x86)/Foxit Software/Foxit
Reader/Foxit Reader.exe"
HTMLprocessor = "C:/Program Files/Mozilla Firefox/firefox.exe"
VIDEOprocessor = "C:/Program Files (x86)/VideoLAN/VLC/vlc.exe"
SOUNDprocessor = "C:/Program Files (x86)/VideoLAN/VLC/vlc.exe"

-- -----
-- SQLfast lists
-- -----
-- default value separator in SQLfast lists --

```

```

    listSeparator = ;
-- Enable/disable argument validation and conversion
    itemSpeed = 0
-- constant separator in dialog box and for lists --
-- (to be changed with MUCH caution) --
    constSeparator = ,

-- -----
-- Dynamic columns
-- -----
-- May scripts include dynamic column references? --
    dynaColumns = 0
-- Default dynamic column marker --
    dynaMarker = !

-- -----
-- Import/Export parameters
-- -----
-- value separator
-- presence of a header record
-- max number of import errors before abort
-- disable messages in the output window
    csv-separator = ;
    csv-header = 1
    csv-maxerrornbr = 10
    csv-mute = 0

-- -----
-- Miscellaneous
-- -----
-- name/value substitution operator
-- (same for sql-oriented operator)
-- timeout in web-based operations
    delimiters = $ $
    delimiters2 = & &
    timeout = 30

```

Part 7 - Control characters

String control characters

Codes that can appear in the character strings to display: message, titles and labels in dialog boxes, text in message box, argument of **write** statement, all the row formatting variables (bars, header, data, selectIntro, selectClose) of the result of **select** statement.

- @n: insert a new line
- @s: inserts one space character (equivalent to @S1)
- @Sn: inserts n space characters
- @t: inserts a horizontal tabulation
- @h: in the result of **select** statement, inserts the column name
- @v: in the result of **select** statement, inserts the column value
- @A: inserts the numeric value of auto-incremented counter **A**. The initial value and increment of **A** are specified by system variables **selectA0** and **selectAi**. The counter is reset before the analysis of each data row.
- @0 to @4: insert the values of system variables **select0**, **select1**, ..., **select4**.
- @@: no operation. Just replaced by @.

Dialog box control characters

Codes that can appear in components of dialog boxes.

- /b: in the message field block, the message must appear in boldface
- /x25: in the message field block, width of the display area (characters for texts, in pixels for pictures and drawings). In askData, width of the entry fields. In heterogeneous composite boxes, horizontal size (in frames) of an elementary box.
- /y6: in the message field block, height of the display area (lines for texts, in pixels for pictures and drawings). In heterogeneous composite boxes, vertical size (in frames) of an elementary box.
- /w0: in the message field block of an askText or showText statement, line wrapping for text in display area (w0 for no wrapping, w1 for wrapping)
- !: in a predefined value expression, states that the user must choose an item in the drop-down list.

Part 8 - SQLdraw commands

The SQLdraw commands control the behaviour of the SQLfast vector graphic box (statement `showDrawing`). They are stored in a `*.draw` text file. They allow the creation of the drawing area, the display of shapes, their deletion, the display of messages and specifying a delay.

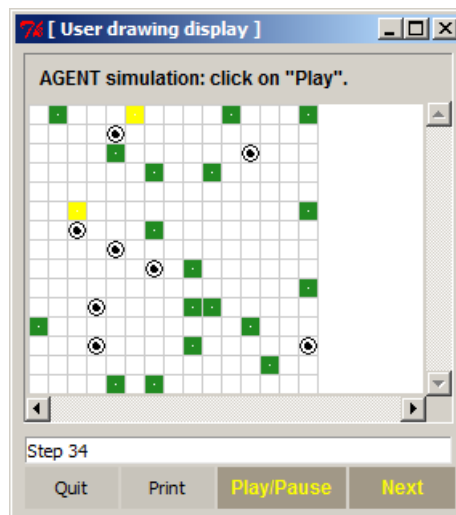


Figure A0.1 - Example of contents of the SQLfast graphic box (from an *Agent modeling* simulation)

A shape is a named basic graphic form: image, polyline, polygon, series of points, rectangle, circle. Once displayed, a shape can be deleted. A *shape sequence* is a suite of shapes that are displayed in the drawing area and that cannot be interrupted through the Pause button. A sequence is terminated by a `step` command or by the end of the file. When the SQLdraw box opens, it shows the first sequence. The next sequences can be displayed through buttons Next (only the next sequence) or Play (all the next sequences).

A color can be specified by its RGB components (`#RRGGBB`, where RR, GG, BB are hexa bytes) or by the poetic symbolic name of one of the 754 standard colors of Tkinter (see file Tkinter colors in RGB.xls).

```
area 600,480,"white"
```

Specifies the drawing area: width, height and background color

image "photo1",grid.gif

Specifies a *.gif file that will be loaded in the drawing area. *Currently disabled.*

polyline "Lines-08",1,"grey"

Header of a polyline (series of connected points): label, line width (in points), line color. Followed by a list of unlabelled points.

polyline "Lines-08",1,"grey","yellow"

Header of a polygon (series of connected points where the starting point is the ending point): label, line width (in points), line color, fill color. Followed by a list of unlabelled points.

points "Cities",1,"black","black",15

Header of a series of unconnected points of the same type: line width, line color, fill color, point type. Followed by a list of labelled points.

rectangle "crop",x1,y1,x2,y2,x3,y3,lineW,lineCol,fillCol

Draw rectangle x1,y1,x2,y2,x3,y3, with specified line width, line color and fill color (empty if none).

circle "round1",x,y,r,lineW,lineCol,fillCol

Draw a circle with center x,y and radius r and with specified line width, line color and fill color (empty if none).

wait 100

Delay of 100 ms.

step

Specifies a new shape sequence.

delete name

delete last: erase the last shape from the graphical area.

delete all: erase all the shapes from the graphical area.

delete Cities: erase the shape named Cities from the graphical area.

451,984

Unlabelled point. Origin 0,0 is the top left corner.

451,984,"London"

Labelled point. Origin 0,0 is the top left corner.

display message

Write a message in the message field of the graphic box.

Part 9 - SQLtuto commands

SQLtuto is a text formatting set of commands that is used to develop help documents and tutorials.

SQLtuto documents are pure texts that are rendered by the **Help** functions of the graphical interface (**Help** button in level 0 interface and **Help** menu in level 1 interface). SQLfast **displayHelp** statement can also be used to display application dependent help documents.

All SQLtuto commands (but one, `.stop:`) appear as the prefix of a paragraph. A command is enclosed between a dot (.) and a semi-colon (;). The size of the characters displayed in the help window is specified by command **charsize**. In the following, the metrics of the various formats are specified for `charsize = 1`.

Formating commands

In the following patterns, term `<text>` represents a character string terminated by an *end-of-line* symbol and `<header>` a simple character string.

`<text>`

simple text; as if prefixed by `.text :`

`.text0:<text>`

simple text; `<text>`: verdana 9, no space before

`.text:<text>`

simple text; `<text>`: verdana 9, 1 mm before

`.bold:<text>`

bold text; `<text>`: verdana 9 bold, 1 mm before

`.italic:<text>`

italic text; `<text>`: verdana 9 italic, 1 mm before

`.bolditalic:<text>`

bold italic text; `<text>`: verdana 9 bold, italic, 1 mm before

.center:<text>

centered text; <text>: verdana 9, centered, 1 mm before

.title1:<text>

main title; <text>: centered; helvetica 12, bold, 2 mm before, 2 mm after

.title2:<text>

level 2 title; <text>: left aligned, helvetica 11, bold, 2 mm before, 1 mm after

.title3:<text>

level 3 title; <text>: left aligned, helvetica 10, bold, 2 mm before, 1 mm after

.title4:<text>

level 4 title; <text>: left aligned, helvetica 10, bold, italic, 2 mm before, 1 mm after

.

A single dot creates a small blank line (courier 6). Size not modified by **.charSize:**. Used to create small space between paragraphs.

.code:<text>

code; <text>: left margin 6 mm, courier 9

.titlecode:<text>

code title; <text>: courier 9, bold; 2 mm before

.indent α :<text>, where α in [1,4]

indented text; <text>: verdana 9, left margin $3x\alpha$ mm, 2 mm before

.item α :<text>, where α in [1,4]

itemized text; <text>: verdana 9, dash, left margin $3x\alpha$ mm, 2 mm before

.item α :<header>:<text>, where α in [1,4]

itemized text with header; <header> bold, optional; <text>: verdana 9, dash, left margin $3x\alpha$ mm, 2 mm before

.item α c:<header>:<text>, where α in [1,4]

itemized text with header; <header>: courier 9 bold, optional; <text>: verdana 9, dash, left margin $3x\alpha$ mm, 2 mm before

Special commands

--<text>

Comment; text <text> is not displayed.

.image:<address>

Displays the image contained in file <address>; the default directory for images is <hdir>/Images, where <hdir> is the help directory.

.tutoImageMaxSize:n

To reduce the images that are too large, a max size of n pixels can be specified (default is stated by system variable `tutoImageMaxSize`). The image will be forced to be inscribed in an nxn square.

.figure:<text>

Displays <text> prefixed by 'Figure - '.

.charSize:<integer>

<integer> is a number from 0 to 3 specifying the character sizes. The default value is 1, which corresponds to the sizes indicated in the description above (that is, 9, 10, 11 and 12 points).

Other values: 0 = minus one point (8, 9, 10, 11); 2 = plus 1 point (10, 11, 12, 13); 3 = plus 2 points (11, 12, 13, 14).

The size specified by this command is global for the whole document. It is best to place it at the beginning of the document.

.stop:

Terminates the display; the following paragraphs are ignored

.link:<address>:<text>

Displays text <text> in blue characters; <address> is the address of another help document that will replace the current one when <text> is clicked on.

.script:<scriptId>:<execLabel>

Starts an embedded script. Generates the script Id and the execution button. See detail in Chapter 26 of the SQLfast manual.

.endscript:<scriptRef>:<caption>

Closes the current embedded script and generates the legend.

.scriptMaxHeight:n

Specifies the maximum of lines of the body window of embedded scripts.

.loadscripts:

Creates in the Scripts directory a script file for each embedded scripts the Id of which has suffix *.sql.

Part 10 - The LTemp temporal language

The SQLfast library **LTemp** provides a collection of functions the goal of which is to simplify and optimize the processing of temporal data. Each function requires a single argument which is a query expressed in a pseudo-SQL language (see case studies Temporal databases 1 and 2 for more detail). It returns a numeric code indicating how the execution was carried out.

Temporal projection

```
function status = LTemp:project {<project-query>};
<project-query> ::= select CITY,PROJECT
                    from    H_EMPLOYEE
                    where   Salary > 6000
                    into H_CITY_PRO
```

Computes the **generalized projection** of table (or view) H_EMPLOYEE on its columns CITY, PROJECT and stores the result in table H_CITY_PRO. If condition where is specified, only the rows that satisfy this condition are processed. If table H_CITY_PRO does not exist it is created.

Temporal reduction (coalescing)

```
function status = LTemp:reduce {<reduce-query>};
<reduce-query> ::= select CITY,PROJECT from H_EMPLOYEE
```

Reduces (coalesces) the consecutive value-equivalent states in the source table itself. Same effect as LTemp:project where the target table is the source table.

Generating valued stable intervals

```
function status = LTemp:valuedInterval {<interval-query>};
<interval-query> ::= select PROJECT,CITY,SALARY
                    from    H_EMPLOYEE
                    where   SALARY > 6000
                    group by PROJECT,CITY
```

```
into VALUED_STABLE_INTERVAL
```

This operator is used as the first step of **temporal aggregate** functions. It computes the largest intervals in source table `H_EMPLOYEE` (among the rows that satisfy condition `where`) in which the values of `PROJECT, CITY` are constant. Then, it associates with each of these intervals the values of `PROJECT, CITY, SALARY` that hold in the source table in this interval. From this result, that has been stored in table `VALUED_STABLE_INTERVAL`, various temporal aggregate functions can be applied on `SALARY` for each `PROJECT, CITY` group. Table `VALUED_STABLE_INTERVAL` is created if it does not exist.

When the `group by` clause is absent, the intervals are computed differently. An interval is considered stable throughout table `H_EMPLOYEE` if the values of `PROJECT, CITY, SALARY` (not only of `PROJECT, CITY` as in the former formula) are constant in this interval.

Temporal normalization

```
function status = LTemp:normalize {<normalize-query>};
<normalize-query> ::= update H_EMPLOYEE wrt H_PROJECT
                    on PROJECT = TITLE
                    into NH_EMPLOYEE
```

If, for each value of `PROJECT`, table `H_EMPLOYEE` is incomplete and includes *gaps*, aka *missing intervals*, this operator replaces them by fictitious states comprising the value of `PROJECT + null` values. So, the history of each value of `PROJECT` is now complete. If `NH_EMPLOYEE` is specified, the completed data are stored in table `NH_EMPLOYEE`. Otherwise, the new states are inserted in table `H_EMPLOYEE` itself. If table `NH_EMPLOYEE` does not exist, it is created.

If comparison table `H_PROJECT` is specified, column `TITLE` must also be specified. When comparing the history of a value of `PROJECT` in table `H_EMPLOYEE` with the history of the same value of `TITLE` in `H_PROJECT`, if the latter history either starts earlier or ends later, then one or two fictitious states are added to the history of `H_EMPLOYEE` in order to adjust both histories. If this way, the history of `H_EMPLOYEE` does not start after that of `H_PROJECT` and does not end before that of `H_PROJECT`, and this, for each value of `PROJECT`. This function is mainly used in the preparation of the arguments of *outer joins*.

If table `H_EMPLOYEE` is empty, a *null* state is generated for each value of `TITLE`. Its interval is the life span of this value in `H_PROJECT`.

