SQL/R

Report Generator
for HP ELOQUENCE

Supplement to Revision A.01.50

mse

Published Editions:

A.01.00 - 1992
A.01.36 - July 1995,Supplement
A.01.50 - August 1996, Supplement

Preface

This supplement to the **SQL/R** manual contains a complete description of additional functionality that has been implemented into **SQL/R** since the publication of the manual.

This supplement is on the **SQL/R** release A.01.36. HP-UX release 9.x is a prerequisite to this version.

In order to use **SQL/R** with HP-UX 10.x, you need **SQL/R** release A.01.40 or above, which will be released in August 1995.

Topics

Typographical Conventions

Unless otherweise noted, this manual uses the following symbolic conventions:

`Computer Font`	Computer font indicates commands, keywords, options, literals, source codes, system outputs and path names.
[ ]	The symbol [ ] indicates a key on a computer keyboard or an area or "button" on screen that can be activated by your mouse. For example, [CTRL] indicates the Control key and [Continue] is an on screen button.
[CTRL]-[char]	The symbol [CTRL]-[char] indicates a control character. For example [CTRL]-[Y] means you have to simultaneous press the Control key and the `Y` key on the keyboard.
italics	Within syntax statements, a word in italics represents a formal parameter or argument that you have to replace with an actual value. In the following example, you must substitute *filename* by the name of the file to be printed: `lp` *filename*
[ ]	Within syntax statements, brackets enclose optional elements. In the following example, brackets around `[-d`*dev*`]` indicate that the parameter and its delimiter are optional: `lp [-d`*dev*`]` *filename*
{ }	Within syntax statements, braces indicate that you must choose one of the listed items. In the following example, the braces around {`-c`|`-x`|`-v`} indicate, that you must choose one othe the arguments: `tar` {`-c`|`-x`|`-v`}

Additional Reading

The following additional documentation is referred to in this manual:

SQL/R Manual

The **SQL/R** manual contains a detailed description of the **SQL/R** syntax and commands.

HP-UX (online) Documentation

References of the form `services`(4) refer to the given topic or item (here `services`) contained in the indicated section (here 4) of the HP-UX-reference manual. It is also possible to obtain this documentation on-line using the command `man`, whereby in the case of `services`(4) the user should enter the following statement:

```
man 4 services
```

Contents

1

Installation

The installation procedure has been changed for the SQL/R product starting with release A.01.40. The installation procedure as described in the printed manual no longer applies. The new installation procedure is described in this document for HP-UX 9.x and HP-UX 10.x.

Major changes of SQL/R release A.01.50 include:

- **SQL/R** A.01.50 can only be installed an run on the HP-UX operating system release 9.x or later.
- **SQL/R** no longer needs its own installation or update tools, as it is now installed and updated by the usual operatin system tools.
- **SQL/R** uses a new licence scheme. Instead of "branding" the executable files with a system id, a licence file named **`licence`** is now present at

HP-UX 10.x	/etc/opt/sqlr
HP-UX 9.x	/opt/sqlr/etc

  The licence file is a plain text file which contains all licences which apply to the **SQL/R** product. The `/opt/sqlr/etc/chklic` utility may be used to check th licence file.
- **SQL/R A.01.50** has a different file structure than recent **SQL/R** releases in order to be HP-UX 10.x compliant. This new file structure is also provided on HP-UX 9.x.
- Optional products are no longer installed seperately. All files related to the **SQL/R** product will be installed automatically. This includes **SQL/R** Windows Client and **SQL/R** ODBC. Please note, that you need an apropriate licence entry in the license file in order to use optional extensions.

Please refer to `/opt/sqlr/newconfig/ReleaseNotes` for more details.

1.1 Installation overview

In order to use **SQL/R**, you need a license key. You should either have received a temporary license key along with the **SQL/R** software, or you can request your license key from your distributor or Marxmeier Software Entwicklung GmbH using the included fax form.

If this is an update, please check that **SQL/R** is not currently active. If you have **SQL/R** Windows Client or QUERY ODBC, please check that the daemons are no longer active.

Please note: You need about 10 MBytes of available disc space for installation in the file system, which contain `/tmp` and the `/opt` directories.

If this is an update of **SQL/R** from a revision before A.01.40, you may want to delete old software which is located at `/usr/sqlr`. This must be performed manually.

Please check for references to the former **SQL/R** location in your PATH variable. You might want to check `/etc/profile` and `.profile` in the users home directories.

If this is a new installation or an update to a **SQL/R** version before A.01.40, you must add your license key to the license file.

1.2 Installation on HP-UX 9.x

As “superuser”, follow the steps below.

1. Login as root
2. Insert the **SQL/R** DDS tape into your tape drive and extract the software from tape to temporary location.

   ```
   cd /tmp
   tar xv /dev/rmt/0m SQLR.updt
   ```

   where */dev/rmt/0m* is your DDS tape device file.
3. To install SQL/R run “update (1m)” by typing:

   ```
   /etc/update
   ```

   - Select “Change Source or Destination ->”.
   - Select “From Tape Device to Local Sytem..”.
   - Tab to the “Source” field and enter the following commands:

     ```
     /tmp/SQLR.updt
     ```

   - Press **F4** Done.
   - Select “Select/View Partitions and Filesets...”.
   - Activate the “SQL/R” partition and additional either the “SQLR-E” (for English localisation) or the “SQLR-E” (for German localisation) partition.
   - Select **F4** “Start Loading”
   - Type “Y” to start “Start loading filesets now?”.
4. Check `/tmp/update.log` to make sure installation completed with no error.
5. You may want to include **SQL/R** in your PATH. To archieve this for all users, you may edit the `/etc/profile`. Append the following line:

   ```
   PATH=$PATH:/opt/sqlr/bin
   ```

   To include **SQL/R** in the path of an individual user, you may add the line to the `.profile` in the home directory.
6. Add the license key to the file `licence`.

1.3 Installation on HP-UX 10.x

As “superuser”, follow the steps below to install the **SQL/R** software:

1. Login as root
2. Insert the **SQL/R** DDS tape into your tape drive and extract the software from tape to temporary location.

   ```
   cd /tmp
   tar xv /dev/rmt/0m SQLR.updt
   ```

   where */dev/rmt/0m* is your DDS tape device file.
3. To install SQL/R run “swinstall” by typing:

   ```
   /usr/sbin/swinstall -s /tmp/SQLR.sd
   ```

   where */tmp/SQLR.sd* is the path to the file extracted from the DDS tape.

   - In the Software Selection Window, highlight to select the “SQL/R-E” product (for Englisch localisation) or the “SQLR-G” product (for German localisation).
   - Then coose the “mark for install” item from the Actions Menu. The “Marked?” column will be automatically be set to “Yes”.
   - Select the “Install (analysis...)” item from the Actions Menu. When the analysis is finished with no error, (`Status: Ready`, click O.K.
   - Click **YES** in the Confirmation window to begin the actual installation.
4. When the installation is completed, a dialog is displayed to notify you that the install task is completed. You may exit then.
5. You may want to include **SQL/R** in your PATH. To archieve this for all users, you may edit the file `/etc/PATH`. Append the following:

   ```
   :/opt/sqlr/bin
   ```

   To include **SQL/R** in the path of an individual user, you may add the line below to the `.profile` in the home directory:

   ```
   PATH=$PATH:/opt/sqlr/bin
   ```
6. Add the license key to the file `licence`.

1.4 Delete the old SQL/R version

If this is an update of **SQL/R** from a revision before A.01.40, you may want to delete old software which is located at `/usr/sqlr`. This must be performed manually.

Warning: The procedure below will remove all files stored in the tree under `/usr/sqlr`! If you installed your own files you should move them to a save location first.

Deleting the old **SQL/R** software may acomplished by the following commands:

1. Login as root.
2. Change into the directory /usr
   ```
   cd /usr
   ```
3. Remove the directory `sqlr`
   ```
   rm -rf sqlr
   ```
4. Change into the directory /usr/bin:
   ```
   cd /usr/bin
   ```
5. Remove the following files:
   ```
   rm sqlr sqlred sqlrexec
   ```
6. Change into the directory /usr/lib/nls:
   ```
   cd /usr/lib/nls
   ```
7. Remove the old message files of SQL/R:
   ```
   rm /C/sqlr.cat
   rm /german/sqlr.cat
   ```

1.5 The SQL/R license key

In order to use **SQL/R**, you need a license key. You should either have received a temporary license key along with the **SQL/R** software, or you can request your license key from your distributor or Marxmeier Software Entwicklung GmbH using the included fax form.

To install the ODBC license key, perform the following steps:

1. Login as root.
2. Add the license key to your license file using your favorite editor program. The location of the license file depends on the revision of the HP-UX operating system:

   HP-UX 9.x `/opt/sqlr/etc/licence`
   HP-UX 10.x `/etc/opt/sqlr/licence`

   You can use the `/opt/sqlr/etc/chklic` utility program to check your license file.

Please note: The license key must by typed exactly as on the license sheet.

1.6 Additional information

Please refer to directory `/opt/sqlr/newconfig/ReleaseNotes`, for further documentation about fixed problems or enhancement documentation.

Directory `/opt/sqlr/newconfig/ReleaseNotes` contains the following files:

README.g	Release Notes in German language
README.e	Release Notes in English language
README.srv	SQL/R Windows Client Server Side Release Notes
README.odbc	SQL/R ODBC Server Side Release Notes
INSTALL	How to install or update SQL/R
A.01.35	SQL/R A.01.35 release notes
A.01.40	SQL/R A.01.40 release notes
A.01.41	SQL/R A.01.41 release notes
A.01.42	SQL/R A.01.42 release notes
A.01.50	SQL/R A.01.50 release notes

Files with the extention “g” are in german language, files with extention “e” ore no extention are in english language.

1.7 New product structure

With the release of HP-UX 10.0, Hewlett Packard has introduced a new file system layout paradigm, modelled after SVR4 and OSF. The model provides many benefits, such as separating the operation system from applications ans aligning HP with an industry-accepted file system layout. **SQL/R** since release A.01.40 follows the HP-UX 10.0 file system conventions and provides a similar file structure with HP-UX 9.x and HP-UX 10.x.

```
/opt/sqlr/                    Base directory
  |
  |-- etc/                    Utility programs
  |                           HP-UX 9.x: Application specific
  |                                      configuration files
  |
  |--- bin/                   Executable programs
  |
  |--- lib/                   Libraries
  |      |
  |      |-- nls/
  |            |
  |            |-- C/         Default message catalog
  |            |-- german/    German message catalog
  |
  |--- newconfig/
  |      |
  |      |-- ReleaseNotes/    Release Notes
  |      |
  |      |-- etc/             Example licence file
  |      |
  |      |-- startup/         HP-UX 10.0 startup/shutdown scripts
  |
  |
  |--- share/
  |      |
  |      |--- map/            Character set mappings
  |      |
  |      |--- sample/         Example scripts (from manual)
  |      |                    Linked to a language dependend directory
  |      |--- sample.e/       - English example scripts
  |      |--- sample.g/       - German example scripts
  |      |
  |      |--- db/             Example database
  |      |                    Linked to a language dependend directory
  |      |--- db.e/           - English example database
  |      |--- db.g/           - German example database
  |      |
  |      |--- example/        Example programs
```

The following directories are only present with HP-UX 10.x:

```
/etc/opt/sqlr/                Application specific
                              configuration files
```

```
/var/opt/sqlr/              Application specific temporary
                            files

/etc/rc.config.d/           Startup configuration files

/sbin/
  |
  | --- init.d/             Startup and shutdown scripts
  |
  | --- rc1.d/              Startup and shutdown link files
  | --- rc2.d/              for script sequencing
```

2

New Functions

This chapter covers a detailed description of new **SQL/R** functions. Some functions having related or similar activities are described together as "Family".

Every entry consists of the following parts:

- a short description of the function
- syntax definition showing how to use the function
- a detailed description of the activity of the function
- return value if any
- example showing how to use the function

Functions are used to modify existing fields or define new fields and expressions. Because of this, all functions begin with the character “@” in order to avoid possible conflicts with existing field names.

Functions fall into the following categories:

- **Conversion of data types**
  This functions convert data from one data type to the other. For example, strings can be converted into numeric values or date formats and vice versa.

@CHAR	@NUM	@VALUE
@DATETOCHAR	@STRING	
@DATEVALUE	@TIMEVALUE	

- **Manipulation of Character Strings**

@LEFT	@POS	@SUBSTR
@LENGTH	@RIGHT	@TRIM
@LOWER	@RPT	@UPPER

- **numeric Functions**

@ABS	@FRACT	@MOD
@DIV	@INT	@ROUND

- **Date and Time Functions**

`@DATE`	`@MINUTE`	`@SECONDS`
`@DATETOCHAR`	`@MINUTES`	`@TIME`
`@DATEVALUE`	`@MONTH`	`@TIMEVALUE`
`@DAY`	`@MONTHBEG`	`@WEEKBEG`
`@DAYS`	`@NOW`	`@WEEKDAY`
`@DIFFTIME`	`@QUARTER`	`@WEEKS`
`@HOUR`	`@QUARTERBEG`	`@YEAR`
`@HOURS`	`@SECOND`	`@YEARBEG`

Some of the functions given here are part of **SQL/R** (cf. chapters 6.7 pp.), though without the leading “@” character. Because of compatibility, the old versions of such functions are still supported. For the sake of uniformity they should however no longer be used in new scripts.

2.1 Conversion of Data Types

2.1.1 @CHAR

Activity

Conversion of a numeric value into a character

Syntax

`@CHAR( value )`

Description

`@CHAR` returns the character that corresponds to the argument *value*.

To obtain a meaningful result, the *value* should be in the range of 0 and 255. In cases where *value* is not in this interval, the value will be ANDed bitwise with 255, so that the function always returns a corresponding character.

Return Value

The function returns the character corresponding to the argument *value*.

Example

```
@CHAR( 49 ) and @CHAR( -207 ) return both "1"
@CHAR( 65 ) and @CHAR(  321 ) return both "A"
```

2.1.2 @DATETOCHAR

Activity

Conversion of a date-value into a string.

Syntax

`@DATETOCHAR( Date, Format )`

Description

`@DATETOCHAR` converts the date-value given by the argument *Date* into a string and returns this. The format for the string output is described in the argument *Format*. A list of permitted formats is given in appendix B of the **SQL/R** manual. The resulting string depends on your working environment (`NUMERIC`).

Return Value

The generated string is returned.

Example

The constant `@NOW` used in the example below is the date-value of the current day (today). It is explained later in this manual. The following results are obtained in accordance with the given formats.

```
@DATETOCHAR( @NOW, "%c") returns:
      "Mon May 22 1995, 17:58:29"
@DATETOCHAR( @NOW, "Today is the %d.%m.%y") returns:
      "Today is the 22.05.95"
```

2.1.3 @DATEVALUE

Activity

Converts a string or string expression into a date-value.

Syntax

`@DATEVALUE( String )`

Description

`@DATEVALUE` converts the string given by the argument *String* into the internal date format (the number of seconds since 01/01/1970). For a list of the supported date formats, please refer to the example below. Please note that constant date-values (e.g. "@01/01/95") do not have to be converted since this is done internally by **SQL/R**.

Return Value

The function returns the corresponding date-value. The return value is NULL if *String* does not define a correct date (e.g "Today is the 07/22/93").

Example

The following calls of the `@DATEVALUE` function all return the date-value `743292000`:

```
@DATEVALUE( "07/22/93" )
@DATEVALUE( "22.07.93" )
@DATEVALUE( "930722"   )
```

The example below retrieves all records that have a transaction date between 01/01/95 and today's date.

```
SELECT ... WHERE @DATEVALUE("01/01/95") <= trdate <= @NOW;
```

The same instruction using `BETWEEN ... AND ...` would result in an error message since the results of a function is not considered a constant value.

2.1.4 @NUM

Activity

Returns the numeric value corresponding to the ASCII code of the first character of the string argument.

Syntax

`@NUM( String )`

Description

`@NUM` returns the numeric value of the first character of the string indicated by the argument *String*. This can be a normal text as well as a string consisting of individual characters. Please note that the backslash character ("\") within a string is a control character which removes the special meaning of the following character. Therefore, "\" is a syntax error, because the second "will be treated as a printable character, not as the end of the quoted string. (see Example)

Return Value

The function returns the determined value (code).

Example

```
@NUM( "A" )          returns the value 65
@NUM( "\A" )         returns the value  65
@NUM( "Example" )    returns the value 66

@NUM( "\" )          Wrong
@NUM( "\\" )         OK
@NUM( """ )          Wrong
@NUM( "\"" )         OK
@NUM( '"' )          Correct and simpler
```

2.1.5 @STRING

Activity

Conversion of a numeric value into a string.

Syntax

`@STRING( format, number )`

Description

`@STRING` converts *number* into a string. The output format is defined by the argument *format*. The result depends on the working environment (`NUMERIC`).

The argument *format* is a string that has the following syntax:

`% [Flags] [Width] [.Precision] Type`

Compoment	Meaning
`%`	Each conversion specification is introduced by the character %. If it is not present, then the rest of the instruction is interpreted as a normal text.
`[Flags]`	Zero or more flags, which modify the meaning of the conversion specification.
`[Width]`	An optional string of decimal digits to specify a minimum field width in bytes. If the converted value has fewer characters than the field width, it is be padded to the field width (according to Flags). If the field width is preceded by a zero, the string is right adjusted with zero-padding on the left
`[.Precision]`	The precision gives the number of digits to appear after the radix character for the "f" conversion and the maximum number of significant digits for the "g" conversion. The precision takes the form of a period followed by a decimal digit string; a null digit string is treated as zero.
`[Type]`	A conversion character that indicates the type of conversion to be applied. This is the last character of the formating instruction. If it is missing, then the function returns a wrong result. The only permitted types are "f" and "g".

[Flags]	Meaning
`negative sign (-)`	Left justified output, in which blanks are added if necessary. The default setup gives right justified output with leading zeros or blanks.
`Positive sign (+)`	Always include a sign character. By default, the output has a sign only if it is a negative numeric value.
`Empty space ( )`	Positive values begin with blanks and negative values with negative signs.
`Hash sign (#)`	Always include a decimal point in the output. Additionally, if the output is of the type *g*, then trailing zeros are not suppressed.

[Width]	Meaning
`n`	Output of at least `n` characters. The output is right justified by default, so leading blanks are added if necessary. If the `Flag -` (left justify) is given, trailing spaces are appended instead.
`0n`	Output at least `n` characters. Leading zeros are added if necessary.

Type	[.Precision]	Meaning
f		The argument is converted to decimal notation in the style [-]dddrddd, where r is the radix character. The number of digits after the radix character is equal to the precision specification. The last digit is determined by rounding if necessary.
	`none`	If the precision is missing, six digits are output.
	`.0` or `.`	If the precision is explicitly zero, no radix character appears. The output contains only the whole number part without a decimal point and decimal places.
	`.n`	The output has `n` decimal places.
g		The argument is converted to decimal notation in the style [-]dddrddd, or [-]drddde[+/-]dd where r is the radix character and e is the exponent. Precision specifies the number of significant digits. Trailing zeros are removed from the fractional part of the result; a radix character appears only if it is followed by a digit. The last digit is determined by rounding if necessary.
	`none`	If the precision is missing, up to six significant decimal digits are output.
	`.0` or `.`	The output is formatted in exponential form.
	`.n`	The output has up to `n` significant digits. If `n` is smaller than the number of digits before the decimal, then the result is formatted in exponential form.

While the type **f** specifies the number of decimal places, the output format **g** specifies the number of significant digits.

Return Value

The function returns the generated string.

Example

```
SET LOCALE "NUMERIC=C";
@STRING( "%06.3g", 1.2345) returns "001.23"

SET LOCALE "NUMERIC=german";
```

```
@STRING( "%06.3f", 1.2345) returns "01,235"
@STRING( "%06.3g", 1.2345) returns "001,23"
@STRING( "%+6.3f", 1.2345) returns "+1,235"
@STRING( "%+6.3g", 1.2345) returns " +1,23"
@STRING( "%-6.3f", 1.2345) returns "1,235 "
@STRING( "%-6.3g", 1.2345) returns "1,23  "
```

2.1.6 @TIMEVALUE

Activity

Converts a string value into a time-value

Syntax

`@TIMEVALUE( String )`

Description

`@TIMEVALUE` converts the string given by the argument *String* into the corresponding time-value, i.e the number of seconds since midnight. For a list of supported time formats, please refer to the examples below. Please note that constant time-values (e.g. "@12:30:15") do not have to be converted explecitly since the conversion is done by **SQL/R** internally.

Return Value

The function returns the corresponding time-value. The return value is NULL if *String* does not define a correct time (for example "It is now 15:10:20").

Example

Both the following calls of `@TIMEVALUE` return the value `54302`:

```
@TIMEVALUE( "15:05:02" )
@TIMEVALUE( "150502"   )
```

The calls of `@TIMEVALUE` without the value for the seconds given in the argument return the corresponding value `54300`:

```
@TIMEVALUE( "15:05" )
@TIMEVALUE( "1505" )
```

2.1.7 @VALUE

Activity

Conversion of a string or string expression into a number.

Syntax

`@VALUE( String )`

Description

`@VALUE` converts the string given by the argument *String* into the corresponding numeric value. The decimal point can be either a "." or a character that is defined by the working environment (`NUMERIC`).

Please note that the conversion always stopps as soon as an invalid character is encountered in the string.

Return Value

The function returns the numeric value which is always of type double. This is true even when the value is an integer number.

Example

```
SET LOCALE "NUMERIC=C";
@VALUE( "12.34" ) returns 12.34
@VALUE( "12,34" ) returns 12.00
```

In the second case, the comma (",") is not permitted and the conversion is stopped.

```
SET LOCALE "NUMERIC=german";
@VALUE( "12.34" ) returns 12,34
@VALUE( "12,34" ) returns 12,34
```

Please be aware that the return value is of type `double`.

```
FIELD xx = IF (..., @VALUE( "100" ), 10);
```

This instruction returns an error message because of different data types on possible values returned of the `IF`. The right formulation of this instruction is as follows

```
FIELD xx = IF (..., @VALUE( "100" ), 10.0);
```

By specifying a decimal point, the constant "10" is explicitly recognized as a value of type `double`.

2.2 Manipulation of Character Strings

2.2.1 @LEFT, @RIGHT, @SUBSTR

Activity

This function can be used to extract and copy parts of existing strings.

Syntax

```
@LEFT( String, number )
@RIGHT( String, number )
@SUBSTR( String, Start, number )
```

Description

The functions `@LEFT`, `@RIGHT` and `@SUBSTR` each exracts *number* characters from a string defined by the argument *String* and returns it.

If *number* is longer than the length of the string designated by *String* then the whole string is returned without filling the remaining character positions with spaces.

The difference between the three functions is the position where each of them starts extracting the required character string. The function `@LEFT` extract the required characters from the beginning of the string, `@RIGHT` from the end and `@SUBSTR` starts its extraction from the position given by the argument *Start*, whereby the first character in the string has position 0.

Return Value

The function delivers the extracted string character. It returns `NULL` if either the argument *number* or *Start* or both have invalid values (e.g smaller than 0).

Example

```
@LEFT( "1234567890",   5)    returns "12345"
@LEFT( "123",          5)    returns "123"

@RIGHT( "1234567890",  5)    returns "67890"
@RIGHT( "123",         5)    returns "123"

@SUBSTR( "1234567890", 4, 3) returns "567"
@SUBSTR( "1234567890", 7, 5) returns "890"
```

2.2.2 @LENGTH

Activity

Determines the length of a string.

Syntax

```
@LENGTH( String )
```

Description

The function `@LENGTH` determines the number of characters (length) of the string indicated by the argument *String*.

Return Value

The function returns the determined length.

Example

```
@LENGTH( "1234567890" ) returns the number 10
```

2.2.3 @LOWER, @UPPER

Activity

Conversion of lower case to upper case (small letters to capital letters) and upper case to lower case characters respectively.

Syntax

```
@LOWER( String )
@UPPER( String )
```

Description

The function `@LOWER` converts the characters contained in the argument *String* to capital letters, while the function `@UPPER` does the opposite, i.e converts capital letters contained in *String* to small letters.

The conversion depends on the selected working environment (`LANG` and `CTYPE`). This is important, if European letters (like ä, ö, ü, Ä, Ö, Ü) are being converted.

Return Value

The function returns the converted string of characters.

Example

```
@UPPER( "test" ) returns "TEST"
@LOWER( "TEST" ) returns "test"

SET LOCALE "CTYPE=C";
SELECT @UPPER( "äöüß" ), @LOWER( "ÄÖÜß" );

gives the following output:  "äöüß   ÄÖÜß"

SET LOCALE "CTYPE=german";
SELECT @UPPER( "äöüß" ), @LOWER( "ÄÖÜß" );

gives the following output:  "ÄÖÜß   äöüß"
```

2.2.4 @POS

Activity

Searches for the occurrence of a string within another string.

Syntax

`@POS( String1, String2 )`

Description

The function `@POS` returns the position of the first occurrence of the string indicated by the argument *String2* in the string given by the argument *String1*.

Return Value

The function returns the position of the beginning of *String2*. If *String2* is not contained in *String1* then the return value is 0.

Example

```
@POS( "1234512345", "34" ) returns position 3
@POS( "1234512345", "XX" ) returns the value 0
```

2.2.5 @RPT

Activity

Constructs a new string by repeatedly appending a given string.

Syntax

`@RPT( String, number )`

Descripion

The function `@RPT` returns a string value constructed by appending the string argument *String number* times.

Return Value

The function returns either the constructed string or NULL if the value of the argument *number* is not permitted.

Example

```
@RPT( "ABC",  3 ) returns the string "ABCABCABC"
@RPT( "-",   10 ) returns the string "----------"
@RPT( "ABC",  0 ) returns an empty string ""
@RPT( "ABC", -1 ) returns NULL
```

2.2.6 @TRIM

Activity

Deletes leading and trailing spaces from a string.

Syntax

`@TRIM( String )`

Description

The function `@TRIM` deletes all the leading and trailing spaces from the character string *String*.

Return Value

The function returns the modified string.

Example

```
@TRIM( "   A B C  " ) returns "A B C"
```

2.3 Numeric Functions

2.3.1 @ABS

Activity

Determines the absolute value of a number.

Syntax

`@ABS( number )`

Description

The function `@ABS` computes the absolute value of the number given by the argument *number*. The return type is double, so `@ABS` will inherit the default output format of the double format having two decimal places. The output will be rounded if necessary. The `FIELD ... DISPLAY AS` statement may be used to specify a different output format.

Return Value

The function returns the absolute value of the given number. The return type is **double**.

Example

```
FIELD v1 = -1.236;
FIELD v2 = @ABS( v1 );
FIELD v3 = v2 * 100.0;
FIELD v4 = @ABS( v1 ) DISPLAY AS DOUBLE(8,4);

SELECT v1, v2, v3, v4;

this instruction returns the following result:
      V1     V2      V3         V4
   -1,236   1,24   123,6     1,2360
```

2.3.2 @DIV

Activity

Determines the whole number part of the result of the division of a number by another number.

Syntax

`@DIV( number, Divisor )`

Description

The function `@DIV` divides the number given by the argument *number* by the *Divisor* and returns the whole number part of the quotient.

Return Value

The function returns the whole number portion of the quotient. The data type of the result is the same as that of the argument *number*.

Example

```
@DIV(   20, 3 ) returns the  value 6  of type int
@DIV( 20.0, 3 ) returns the value 6.00 of type double
```

2.3.3 @FRACT

Activity

Computes the difference of a number from the next smaller whole number.

Syntax

`@FRACT( number )`

Description

The function `@FRACT` determines the difference between the number given by the argument *number* and the next whole number smaller than *number*. Please note that if *number* is negative, then the result obtained from this function differs from that of HP Eloquence.

Return Value

The function returns the computed difference, which is **always positive**.

Example

```
@FRACT(  1.48 ) returns  1.48 -   1.00  = 0.48
@FRACT(  1.78 ) returns  1.78 -   1.00  = 0.78
@FRACT( -1.48 ) returns -1.48 - (-2.00) = 0.52
@FRACT( -1.78 ) returns -1.78 - (-2.00) = 0.22
```

2.3.4 @INT

Activity

Returns the largest integer not greater than *number*.

Syntax

`@INT( number )`

Desciption

The function `@INT` returns the largest integer not greater than *number*.

Return Value

The function returns determined value.

Example

```
@INT(  1.48 ) returns  1.00,   @INT( -1.48 ) returns -2.00
@INT(  1.78 ) returns  1.00,   @INT( -1.78 ) returns -2.00
```

2.3.5 @MOD

Activity

Determines the remainder from the division of two numbers.

Syntax

`@MOD( number, Divisor )`

Description

The function `@MOD` divides the number given by the argument *number* by *Divisor* and returns the remainder. The remainder is defined as (*number* - *Divisor* * *quotient*), where the *quotient* is a whole number. Please note that if *number* and *Divisor* are negative, then the result obtained from this function differs from that of HP Eloquence.

Return Value

The function returns the remainder from the division. The data type of the return value depends on the type of the argument *number* and on the computed value. If *number* is of type `int` and the remainder is a whole number, then the result is of type `int` otherweise it is of type `double`.

Example

```
@MOD(  20 ,  3.5 ) returns  2.5 of type double
@MOD( -20 ,  3   ) returns -2   of type int
@MOD(  20., -3   ) returns  2.0 of type double
@MOD( -20., -3.5 ) returns -2.5 of type double
```

2.3.6 @ROUND

Activity

Rounds a number to a given accuracy

Syntax

`@ROUND( number, accuracy )`

Description

The function `@ROUND` rounds the number given by the argument *number* according to the accuracy prescribed by the argument *accuracy*. Accuracy is specified in the means of power of ten. So an accuracy value of -2 would specify an accuracy of 10^{-2} or two decimal places.

Return Value

The function returns the rounded number, which is always of type `double`.

Example

```
@ROUND(  1.494,  0) is  1.00,  @ROUND(  1.5,  0) is   2.00
@ROUND( -1.494,  0) is -1.00,  @ROUND( -1.5,  0) is  -2.00
@ROUND(  1.494, -2) is  1.49,  @ROUND(  1.5, -1) is   1.50
@ROUND( -1.495, -2) is -1.50,  @ROUND( -1.5, -1) is  -1.50
@ROUND(  4.995,  1) is  0.00,  @ROUND(  5.0,  1) is  10.00
@ROUND( -4.995,  1) is  0.00,  @ROUND( -5.0,  1) is -10.00
```

2.4 Date and Time Functions

2.4.1 @DATE

Activity

Computes the date from the given values of the year, month and day.

Syntax

`@DATE( year, month, day )`

Description

The function `@DATE` uses the arguments *year*, *month* and *day* to determine a corresponding date-value (i.e the number of seconds since 01/01/1970). The `DISPLAY AS DATE` rule may be used to specify the output format.

Return Value

The function returns the computed date-value of type `long`.

Example

```
FIELD f_date = @DATE( 95, 5, 20 )
                   DISPLAY AS DATE("%d.%m.%y");
SELECT f_date, @DATE( 95, 5, 20 ), @20.05.95;

Both instructions return the following result:
     F_DATE     @DATE(95,5,20)     20.05.95
     20.05.95        800920800    800920800
```

2.4.2 @TIME

Activity

Computes the time-value from values given for hour, minute and second.

Syntax

`@TIME( hour, minute, second )`

Description

The function `@TIME` uses the arguments *hour*, *minute* and *second* to determine a corresponding time-value (i.e the number of seconds since midnight). The `DISPLAY AS TIME` rule may be used to specify the output format.

Return Value

The function returns the computed time-value.

Example

```
FIELD f_time = @TIME( 15, 02, 02 ) DISPLAY AS TIME(8);
SELECT f_time, @TIME( 15, 05, 02 ), @15:05:02, @150502;

Both instructions return the following result:
     F_TIME     @TIME(15,05,02)   15:05:02     150502
   15:02:02              54302      54302      54302
```

2.4.3 @DIFFTIME

Activity

Determines the difference between two time-values in seconds

Syntax

`@DIFFTIME( date1, date2 )`

Description

The function `@DIFFTIME` computes the number of seconds between the two dates *Date1* and *Date2*.

Return Value

The function returns the computed difference, where this is negative if *Date2* is later than *Date1*.

Example

```
Date 1 = @NOW             = 802983720
Date 2 = @DATE(93, 7, 22) = 743292000

The call of  @DIFFTIME with these arguments returns
the following result:
   @DIFFTIME( @NOW, @DATE(93, 7, 22) ) =  59691720
   @DIFFTIME( @DATE(93, 7, 22), @NOW ) = -59691720
```

2.4.4 @DAY, @MONTH, @YEAR, @QUARTER, @WEEKDAY

Activity

Determine the desired value from a given date-value.

Syntax

```
@DAY( Date )
@MONTH( Date )
@YEAR( Date )
@QUARTER( Date )
@WEEKDAY( Date )
```

Description

Each of these functions takes a date-value (the number of seconds since 01/01/1970) as the actual parameter for the argument *Date*. The function `@DAY` computes the number of days of the month, `@MONTH` computes the number of months of the year and `@YEAR` the year from this date-value. The function `@QUARTER` computes the number of corresponding quarter year, a value between and including 1 and 4. The function `@WEEKDAY` determines the day of the week, with 0 as Sunday, 1 as Monday etc. Its return value is therefore one of the numbers 0 to 6.

Return Value

Each function returns the value computed from the date-value.

Example

```
Let @NOW = 24.08.1993

Then the functions return the following results:

    @DAY(     @NOW ) = 24
    @MONTH(   @NOW ) =  8
    @YEAR(    @NOW ) = 93
    @QUARTER( @NOW ) =  3
    @WEEKDAY( @NOW ) =  2  = Tuesday
```

2.4.5 @HOUR, @MINUTE, @SECOND

Activity

Compute the desired value from a given date- or time value.

Syntax

```
@HOUR( Date )
@MINUTE( Date )
@SECOND( Date )
```

Description

Each of these functions takes a date- or time-value (the number of seconds since 01/01/1970 or the number of seconds since midnight), as the actual parameter for the argument *Date*. The function `@HOUR` computes the number of hours, `@MINUTE` the number of minutes within the hour and `@SECOND` the number seconds within the minute from the argument *Date*.

These functions also operate with time-value as argument. The working environment can however lead to time differences depending on the local time zone.

Return Value

Each function delivers the value computed from argument *Date*.

Example

```
Let @NOW = 08/24/1993, 20:26:31

Then the functions return the following results:

  @HOUR(   @NOW )  = 20
  @MINUTE( @NOW )  = 26
  @SECOND( @NOW )  = 31
```

2.4.6 @WEEKBEG, @MONTHBEG, @QUARTERBEG, @YEARBEG

Activity

Determine the required date-value from a given date argument.

Syntax

```
@WEEKBEG( Date )
@MONTHBEG( Date )
@QUARTERBEG( Date )
@YEARBEG( Date )
```

Description

Each of these functions takes a date-value (the number of seconds since 01/01/1970) as the actual parameter for the argument *Date*. The function `@WEEKBEG` computes the date-value of the first day in the week (Monday) from the given argument *Date*, `@MONTHBEG` determines the corresponding first day of the month, `@QUARTERBEG` determines the date-value for the first day of the quarter year and `@YEARBEG` determines the corresponding date-value for the first day of the year.

Return Value

Each of these functions computes the required date and returns the corresponding date-value of type `long`.

Example

```
Let @NOW = 24.08.1993

SET DATE    = "%d.%m.%y";
FIELD w_beg = @WEEKBEG(    @NOW);
FIELD m_beg = @MONTHBEG(   @NOW);
FIELD q_beg = @QUARTERBEG( @NOW);
FIELD y_beg = @YEARBEG(    @NOW);

SELECT w_beg, m_beg, q_beg, y_beg;

Gives the following results:

   W_BEG     M_BEG     Q_BEG     Y_BEG
   23.08.93  01.08.93  01.07.93  01.01.93
```

2.4.7 @WEEKS, @DAYS, @HOURS, @MINUTES, @SECONDS

Activity

Use a given number or expression to compute the corresponding number of seconds.

Syntax

```
@WEEKS( number )
@DAYS( number )
@HOURS( number )
@MINUTES( number )
@SECONDS( number )
```

Description

Dates and Times are processed internally as the number of seconds since 01/01/1970. This is the standard date and time format for Unix. This format makes it just as easy to calculate with dates and times as with any other numbers. All functions compute the number of seconds from the product of the constants prescribed to the function and the value that is given by the argument *number*.

The constants prescribed to each function are as in the following list:

Function	Constant Factor	
@SECONDS( )	1 Second	
@MINUTES( )	60 Seconds	(60 * 1)
@HOURS( )	3600 Seconds	(60 * 60)
@DAYS( )	86400 Seconds	(24 * 3600)
@WEEKS( )	604800 Seconds	(7 * 86400)

Return Value

The functions return the computed number of seconds.

Example

```
Let @NOW = 24.08.1993

SET DATE = "%d.%m.%y";
FIELD tomorrow      = @NOW + DAYS(1);
FIELD two_weeks_ago = @NOW - @WEEKS(2);

SELECT tomorrow, two_weeks_ago;

Leads to the the following results:

   TOMORROW    TWO_WEEKS_AGO
   25.08.93    10.08.93
```

3

New SQL/R functionality

3.1 Comments

Two different styles of comment are possible with **SQL/R**. The first style is for a one line comment while the second style changes a whole paragraph of your script into a comment.

3.1.1 Comment Line

Any line which begins with the hash character "#" is interpreted as a comment (which **SQL/R** will ignore during execution).

For example:

```
# Some comment
Instructions
...
# More comments
Instructions
...
```

Incidentally, the hash character "#" has another, little-known use which can make your **SQL/R** scripts executable without having to start `sqlrexec` first. In HP-UX, an executable script whose first line begins with `#!` followed by the full pathname of another program, will execute that program before proceeding to the second line. When such a script is to be executed, the specified program is executed first, passing the file name and the argument to the called program.

For example:

Place the following line first in an **SQL/R** script and make the script executable, and you may start the **SQL/R** script like any shell script.

```
#!/usr/bin/sqlrexec -n
```

Make it executable by the HP-UX command:

```
chmod +x xxx.sql
```

Starting this file with:

```
./xxx.sql
```

is equivalent to:

```
/usr/bin/sqlrexec -n ./xxx.sql
```

3.1.2 Comment Paragraph

All characters enclosed within braces ("{" and "}") are interpreted as comments. This makes it possible to declare a whole paragraph of the script as a comment.

However two limitations have to be noted when using braces:

- The last character of a line may not be a semicolon. This is a limitation of the current syntax parser which considers each line with a trailing semicolon to be the last line of an executable statement.
- If a comment paragraph follows an instruction, then it may not begin on the same line as the semicolon that marks the end of the instruction. **SQL/R** interprets the end of an instruction as the end of the line and ignores the rest of the line. In this case, the opening brace marking the beginning of a comment paragraph will be ignored. That will eventually lead to a syntax error because **SQL/R** will interpret your comment as an instruction.

To improve human readability, we recommend that comment paragraphs (those enclosed by braces) be given their own lines. Indent each comment line for an even better appearance.

For example:

```
Instructions

  {
    First line of comment text
    ...
    Last line of comment text
  }

Instructions
```

3.2 Using Environment Variables

You may use environment variables in **SQL/R** scripts. An environment variable is recognized through its name and the leading character “$”. This provides a powerful method of using variable information (for example: a data base path) with a multiple use script file. Using environment variables in your scripts may make it easier to reuse **SQL/R** scripts. Please note the **SQL/R** scanner always treats environment variables as type string. You may need to convert the value of an environment variable to a different data type within your script.

Example

```
OPEN DATABASE "$HOME/db";
```

This instruction opens the data base "db", in the directory defined by the environment variable `HOME`. For example `HOME = /usr/sqlr`.

The following instruction is a case where the data type must be converted in order to use the environment variable.

Example

```
SELECT ... FROM ... WHERE comno = @VALUE( $COMNO );
```

3.3 The Constant NULL

The constant `NULL` makes it possible to compare values against the `NULL` value. Where `NULL` in this context does not mean zero value or an empty string but the fact that a field value is undefined. In this case, the field always contains the value `NULL`.

Example

```
FIELD value = IF (field value = NULL, 0.0, field value);
...
SELECT ..., MAX( value ), ...
```

All `NULL` values of field `value` are replaced by a zero value. (This is because arithmetic operations return a `NULL` result if any of the input values have a `NULL` value.)

You may not compare a field against a NULL value in the WHERE condition of a SELECT statement. (The internal algorithm which evaluates the WHERE condition will not allow it.) Instead, you must use the clause IS [NOT] NULL to achieve your required result.

Example

```
SELECT MAX( field value) FROM ...
WHERE field value IS NOT NULL;
```

The following example *does not work*:

```
SELECT MAX( field value) FROM ...
WHERE field value <> NULL;
```

3.4 The Constant @NOW

The constant @NOW returns the date-value of the present date (and time) of day. Its return value is of type long and is the number of seconds since 01/01/1970. You may use the @NOW constant in your script to reference the date the script was run without having to modify the script every time.

Example

```
SELECT ..., date, ... FROM ...
WHERE @YEARBEG( @NOW ) <= date AND date <= @NOW;
```

This selects all records in which the field "date" occurs between Jan 1st of the current year and the present date.

3.5 New SET command clauses

3.5.1 SET ECHO

Syntax

`SET ECHO = [ON | OFF]`

Description

The instruction `SET ECHO` corresponds to the `sqlrexec` commandline option "-e".

When `ECHO = ON` is set, every **SQL/R** command is traced to `stderr` before it is executed.

3.5.2 SET COLSEP

Syntax

`SET COLSEP = "Character"`

Description

The command `SET COLSEP` is used to define the character used by **SQL/R** to separate individual columns in a result row. The blank (space) character is the default column separator.

If output is redirected to an `ASCII` file (using the `SET OUTPUT` command), then the column separator becomes the field separator instead of the default "," character.

Alternate definition of column or field separators may be needed when **SQL/R** results will be imported by other applications which have limitations on their import file formats.

3.5.3 SET ROWSEP

Syntax

`SET ROWSEP = "Character"`

description

The `SET ROWSEP` command defines which character will be used to separate individual result rows and `BREAK` result lines (please refer to `REPORT SELECT ... CALCULATE`). The default separator character is a hyphen ("-").

The definition of a line separator has no effect if the output is redirected to an `ASCII` or a `DIF` file.

3.5.4 SET NULL

Syntax

```
SET NULL = "String" | "Character"
```

Description

The command `SET NULL` defines how to output `NULL` values. If the argument is a single character, it is repeated for the field width. The default output for a NULL value is the character string "NULL".

The command `SET NULL` has no effect when output is directed to an `ASCII` or a `DIF` file. In this case the zero value or an empty string is used, depending upon the field data type.

Example

```
CREATE VIEW CustomerOrdersView PATH Customers
  TO Orders WHERE Custno = Custno;
SELECT Orderno, Custno FROM CustomerOrdersView;
```

If there is no entry in the table `Orders` for a particular customer, all fields in the view `CustomerOrdersView` which are defined by the table `Orders` (for example Orders.Orderno) are undefined and will have the `NULL` value.

3.5.5 SET OVERFLOW

Syntax

```
SET OVERFLOW = "String" | "Character"
```

Description

The command `SET OVERFLOW` defines the output when a numeric value requires more space (characters) than was defined for the particular column. This definition only concerns output and does not affect the actual result.

If the argument is a single character, it is repeated for the field width. The default output on an overflow condition is the asterisk ("*") which repeats for the column width.

The command `SET OVERFLOW` has no effect when output is directed to an `ASCII` or `DIF` file.

Example

```
SET OVERFLOW = "#";
FIELD val    = 1234567890;
FIELD val1   = val DISPLAY AS LONG(10);
FIELD val2   = val DISPLAY AS LONG( 6);

SELECT val, val1, val2;

This instruction results in:
          VAL        VAL1        VAL2
   1234567890   1234567890   ######

This instruction results accordingly to:
   SET OVERFLOW = "TOO SHORT";

          VAL        VAL1        VAL2
   1234567890   1234567890   TOO SH
```

4

Enhancement of SQL/R statements

4.1 The REPORT SELECT - statement

Syntax

```
REPORT
      SELECT ...
USING LINEAR LIST;
```

Description

The `REPORT SELECT` statement was extended by the option `USING LINEAR LIST`. Exactly one set of results is displayed on each page, when this rule is used, whereby all column names and their corresponding result values are displayed together.

The use of this extension is especially recommended for the diagnosis of data base and during the development phase of **SQL/R** scripts.

Example

```
OPEN DATABASE "/usr/sqlr/db/db";

REPORT
   SELECT * FROM CUSTOMER
USING LINEAR LIST;
```

The following list of outputs results from this instruction, where such a page is generated for each result row.

```
CUSTNO    = "11006"
MATCHCODE = "ABB"
NAME1     = "ABB-Tecnoservice"
NAME2     = ""
NAME3     = "Postfach 605"
STREET    = ""
ZIPCITY   = "USA-LYONS, ILLINOIS 60534"
```

```
PHONE     = "KYLLBURG 065632019"
TURNOVER  = 1457.67
SALESAREA = "U"
```

4.2 The FIELD - statement

Syntax

```
FIELD ...  DISPLAY AS ...
      [ GROUP OPTION = {ON | OFF} ]
      [ CURRENCY OPTION = {ON | OFF} ]
      [ NULL = {"string" | "char"} ]
      [ OVERFLOW = {"string" | "char"} ]
      [ SUPPRESS REPEATING VALUES [UNTIL GROUP BREAK] ]
```

Description

Extension	Description
GROUP OPTION	This option is only valid for numeric fields. If GROUP OPTION = ON is set, then the output values are grouped. The grouping character and its position depends on the working environment (e.g "12.000,00" (German locale) or "12,000.00" (American locale)). The setting MONETARY defines the working environement for numbers of type MONEY while NUMERIC defines the working environment for all other numeric types. GROUP OPTION = ON is the default setting for values of type MONEY and GROUP OPTION = OFF is the default setting for all other numeric field types.
CURRENCY OPTION	This option is only valid for fields of type MONEY. If CURRENCY OPTION = ON is set, then the curency symbol is output along with the value. The currency symbol and its position depends on the working environment. (e.g "1.000,00 DM" (German locale) or "$ 1,000.00" (American locale)). Both the currency symbol and its formating are defined by MONETARY. The default setting is CURRENCY OPTION = OFF.

Extension	Description
NULL	This option defines how NULL values should be output for this field.
OVERFLOW	This option defines how a field overflow should be indicated for this field. A field overflow occurs, if the output of a numeric field contents requires more space (in terms of characters) than is defined for this particular field.
SUPPRESS ...	This option makes it possible to suppress the output of repeated values of a column so that blanks are given as output instead of the repeated field content. The normal output is continued when either the content of the field changes or the rule `UNTIL GROUP BREAK` os preset and a change of group is necessary. Such a change of group can be forced using the instruction `CALCULATE ... BREAK` (cf. chapter 6.18 pp.).

Example

The following example shows some modifications which may be applied for the example `man34` of the **SQL/R** manual (please refer to chapter 5.3).

This modifications have the following effects:

- The repeated output of order numbers `orderrnr` and order status `status` is suppressed.
- The total sum `sum` is always given at the output with grouping, and currency symbol. In case a NULL value occurs, then the the text "N/A" is output. If the predefined output field is too small, then the character "#" is the output.
- the text "N/A" is also printed if the fields `qty` and `price` are NULL.

```
FIELD orderno DISPLAY AS (10)
              SUPPRESS REPEATING VALUES
              UNTIL GROUP BREAK;

FIELD status = orderstat VALUES ARE ...
              DISPLAY AS LEFT(18)
              SUPPRESS REPEATING VALUES
              UNTIL GROUP BREAK;

FIELD qty DISPLAY AS DOUBLE(6, 0)
              NULL = "N/A";

FIELD price DISPLAY AS DOUBLE(8, 2)
              NULL = "N/A";

FIELD amount = (qty * price / icnt)
              DISPLAY AS MONEY(10, 2)
              GROUP OPTION = ON
              CURRENCY OPTION = ON
              NULL = "N/A"
              OVERFLOW = "#";
```

4.3 The WHERE - Condition

Syntax

`SELECT ... WHERE` *expression* **IS [NOT] NULL**

Description

The result of `CREATE VIEW` statement is a logical view (table), of the database consisting of a number of records linked through the corresponding fields and conditions.

If an individual record does not satisfy the link condition (as specified with by `CREATE VIEW`) its value will be NULL (undefined).

If the `IS NULL` or `IS NOT NULL` condition is present in the `WHERE` clause of a select statement, it possible to select or exclude result rows having NULL values.

Please note that the use of the rule `IS NULL` slightly prolongs the execution time.

Example

```
CREATE VIEW tmp PATH customers
    TO orders WHERE custno = customers.custno

SELECT custno
FROM tmp
WHERE orders.custno IS NULL;
```

This selects all customers who do not have a pending order.

```
...
WHERE orders.custno IS NOT NULL;
```

This selects all customer who have a pending order.

4.4 The CREATE VIEW - statement

Syntax

CREATE VIEW *view_name* PATH *source*
  TO *dest_1* WHERE *dest_1.keywordfield* = **<expression_1>**
  AND *dest_2* WHERE *dest_2.keywordfield* = **<expression_2>**

Description

The `WHERE` clause of the `CREATE VIEW` statement was extended to make it possible to specify an expression as the relating value. Up to **SQL/R** version A.01.31, it was only possible to use a individual data fields for the definition of the desired relation.

Please note that a field from a table that is used as a link field must either be a search item or must have an associated index.

Example

In the following example, for every record from table `Parts`, a suitable record from the table `text` is located, where the field `textnr` must match the leading four characters of the field `parts`.

```
CREATE VIEW tmp PATH parts
    TO text WHERE text.textid = @LEFT(parts.partno, 4);
```

Alternatively, a FIELD statement can be used instead of the concrete expression in the `CREATE VIEW` statement:

```
FIELD f_textid = @LEFT(parts.partno, 4);

CREATE VIEW tmp PATH parts
    TO text WHERE text.textid = f_textid;
```

A

Short Reference

A.1 SQL/R Commands

New entries are typeset in boldface.

CREATE VIEW view_name PATH $table\_spec$ $path\_group$
[DESCRIBE AS "description"] ;

$table\_spec$ = [table_alias_name =] table_name

$path\_group$ = TO $path\_element$ [AND $path\_element$ [AND ...]] [TO ...]

$path\_element$ = { ($path\_element$ $path\_group$) | $table\_spec$ WHERE [table_name.] field = **expression** }

DEFINE ["]macro_name["] AS "macro definition"
[DESCRIBE AS "description"] ;

EXIT ;

FIELD { alias = expression | field_name }
[VALUES ARE ([{"string" | num} =] "string" [, ...])]
[DISPLAY AS [[LEFT | CENTER | RIGHT] $format$] [fmt_option ...]]
[DESCRIBE AS "description"] ;

$format$ = {
(width)
(width, decimal places)
`INT` (width)
`LONG` (width)
`FLOAT` (width, decimal places)
`DOUBLE` (width, decimal places)
`FIXED` (width, decimal places)
`MONEY` (width [, decimal places])
`DATE` [("date_format", width)]
`TIME` [(width)]
}

fmt_option = [
GROUP OPTION = {`ON` | `OFF`}
CURRENCY OPTION = {`ON` | `OFF`}
NULL = {"char" | "text"}
OVERFLOW = {"char" | "text"}
SUPPRESS REPEATING VALUES [**UNTIL GROUP BREAK**]
]

`HELP` [{ identifier | "string" }] `;`

`REPORT SELECT` select_stmt
[`CALCULATE` $field\_calc$ [, ...]]
[`INTO` { `TERMINAL` | `PRINTER` | [`ASCII` | `EXPORT` | `DIF`] `FILE` ["]filename["] | `NULL` }]
[$report\_fmt$ | `USING` report_form | **USING LINEAR LIST**] `;`

$$field\_calc = \left[\left\{ \begin{array}{l} \texttt{SUM} \\ \texttt{AVG} \\ \texttt{MIN} \\ \texttt{MAX} \\ \texttt{COUNT} \end{array} \right\} \text{(field\_ref [, ...]) ["row label"]} \right]$$

$$\texttt{BREAK ON} \left\{ \begin{array}{l} \text{(field\_ref [, ...])} \\ \texttt{REPORT} \end{array} \right\} \left[\begin{array}{l} \texttt{SKIP [n]} \\ \texttt{PAGE [n]} \end{array} \right]$$

report_fmt = [`TITLE AS` "report title"]

[`DATE AS` { `TODAY` | "date string" }]

[`LENGTH` = num]

[`WIDTH` = num]

[`RUN`] file_name [("arg" [, "arg"] . . .)] ;

`SELECT` [`ALL` | `DISTINCT`]

{* | expression ["alternate_heading"] [, . . .] }

[`FROM` view_name]

[`WHERE` cond_expression]

[`GROUP BY` field_ref [, . . .] [`HAVING` cond_expression]]

[`ORDER BY` field_ref [`ASC` | `DESC`] [, field_ref [`ASC` | `DESC`]] . . .] ;

`SET LOCALE` "*category*=language[@modifier]" ;

$$category = \left\{ \begin{array}{l} \texttt{ALL} \\ \texttt{COLLATE} \\ \texttt{CTYPE} \\ \texttt{MONETARY} \\ \texttt{NUMERIC} \\ \texttt{TIME} \end{array} \right\}$$

```
               ⎧ TERMINAL                                      ⎫
               ⎪ PRINTER                                       ⎪
               ⎪ ⎡ ASCII  ⎤                                    ⎪
SET OUTPUT =   ⎨ ⎢ EXPORT ⎥ FILE ["]filename["]                ⎬ ;
               ⎪ ⎣ DIF    ⎦                                    ⎪
               ⎩ NULL                                          ⎭
```

```
    ⎧ DATE     = "date_fmt"            ⎫
    ⎪ ECHO     = [ON | OFF]            ⎪
    ⎪ COLSEP   = "char"                ⎪
    ⎪ LENGTH   = lines                 ⎪
SET ⎨ NULL     = ["char" | "string"]   ⎬ ;
    ⎪ OVERFOW  = ["char" | "string"]   ⎪
    ⎪ PRINTER  = "device"              ⎪
    ⎪ PROMPT   = "prompt string"       ⎪
    ⎪ ROWSEP   = "char"                ⎪
    ⎩ WIDTH    = columns               ⎭
```

```
     ⎧ DATE                        ⎫
     ⎪ FIELD { * | field_name }    ⎪
     ⎪ LENGTH                      ⎪
     ⎪ LOCALE                      ⎪
SHOW ⎨ MACRO { * | "macro_name" }  ⎬ ;
     ⎪ OUTPUT                      ⎪
     ⎪ PRINTER                     ⎪
     ⎪ VIEW { * | view_name }      ⎪
     ⎩ WIDTH                       ⎭
```

A.2 Constants

Constant	Meaning
`NULL`	represents a non defined empty field
`@date`	represents the date-value that corresponds to the argument *date*
`@time`	represents the time-value that corresponds to the argument *time*
`@NOW`	represents the date-value of the actual day

A.3 Functions

The arguments of all the listed functions can be constants, individual fields or formulas as well as expressions.

Function Nname	**(Argument**	**)**	**Function Name**	**(Argument**	**)**
`@ABS`	(number	)	`@NUM`	(string	)
`@CHAR`	(number	)	`@POS`	(string1, string2	)
`@DATE`	(year, month, day	)	`@QUARTER`	(date	)
`@DATETOCHAR`	(date, format	)	`@QUARTERBEG`	(date	)
`@DATEVALUE`	(date	)	`@RIGHT`	(string, length	)
`@DAY`	(date	)	`@ROUND`	(number, precision	)
`@DAYS`	(number	)	`@RPT`	(string, count	)
`@DIFFTIME`	(date1, date2	)	`@SECOND`	(date	)
`@DIV`	(number, divisor	)	`@SECONDS`	(number	)
`@FRACT`	(number	)	`@STRING`	(format, number	)
`@HOUR`	(date	)	`@SUBSTR`	(string, start, length	)
`@HOURS`	(number	)	`@TIME`	(hour, minute, second	)
`@INT`	(number	)	`@TIMEVALUE`	(string	)
`@LEFT`	(string, length	)	`@TRIM`	(string	)
`@LENGTH`	(string	)	`@UPPER`	(string	)
`@LOWER`	(string	)	`@VALUE`	(string	)
`@MINUTE`	(date	)	`@WEEKBEG`	(date	)
`@MINUTES`	(number	)	`@WEEKDAY`	(date	)
`@MOD`	(number, divisor	)	`@WEEKS`	(number	)
`@MONTH`	(date	)	`@YEAR`	(date	)
`@MONTHBEG`	(date	)	`@YEARBEG`	(date	)

B

Character Set Mapping

Like HP Eloquence, **SQL/R** uses the **Roman8** character set internally. It is however possible to map data to a terminal dependent character set in the **SQL/R** editor and by using the program `tmap`.

B.1 How does this work ?

Using the environment variable **`TERM`**, a file with the extension **`.map`** and which corresponds to the terminal type is searched for in the directory **`/usr/sqlr/map`**. If this file is found, the character set is modified according to the definition in the file.

If the environment variable **`SQLR_MAP`** is defined, then the directory specified by the `SQLR_MAP` is searched instead.

For example:

If the environment variable `TERM` has the value **`70060`**, then the file `/usr/sqlr/map/`**`70060`**`.map` is used to map the character set.

B.2 SQL/R Editor

The **SQL/R** editor automatically uses the procedure described above to map all inputs and outputs to the terminal character set. If `sqlrexec` is started from the editor, then its terminal output is mapped as well.

Note

If files are created using any other editor than the **SQL/R** editor on a terminal which does not have the **Roman8** character set and national characters (e.g. umlaut vowels) are used, then the file must be translated to the **Roman8** character set. This may be done by using `iconv` that is part of HP-UX. Please refer to `iconv`(1) for details.

B.3 tmap

tmap is an utility program that is available with **SQL/R** to map the character set during terminal output. It also uses the procedure described above to carry out the mapping.

Usage:

```
/usr/sqlr/tmap [file]
```

The source of data can either be the standard input (`stdin`) or a file.

For example:

```
/usr/sqlr/tmap README
```

Displays the file README in the terminal character set.

```
/usr/bin/sqlrexec -tn xxx.sql | /usr/sqlr/tmap
```

Converts the output of sqlrexec into the character set of the used terminal. (The argument **-t** must be given in order to keep the prompt during the page break).

B.4 iconv

The HP-UX program **iconv** can be used to translate files from one character set to another.

For example:

```
iconv -f roman8 -t iso8859_1 README
```

Translates the file README from the **Roman8** character set to the **ISO 8859/1** character set during the output. The output is displayed on the stdout.

Further information on `iconv`(1) can be obtained using

```
man 1 iconv
```

B.5 sqlrexec

The output from `sqlrexec` is in **Roman8** character set. If the output is to a device with a different character set, then the character set must be mapped accordingly, (for example using `iconv` or `tmap`).

For example

```
sqlrexec -tn xxx.sql | /usr/sqlr/tmap
```

Converts the output of sqlrexec into the character set of the used terminal. (The argument **`-t`** must be given in order to keep the prompt during the page break).

```
sqlrexec -n xxx.sql | tr -d "\014" |
 iconv -f roman8 -t iso8859_1 | more
```

Converts the output of sqlrexec from **Roman8** into the **ISO 8859/1** character set. The output is piped through `more`. (The page break is removed here using the `tr` command).

B.6 Supporting Other Terminal Names

In order to support terminal devices using the **ISO 8859/1** character set, link the file **`iso8859_1.map`** to the terminal dependent mapping filename.

For example:

```
cd /usr/sqlr/map
ln iso8859_1.map hp70060.map
```

C

Using the terminal printer (lprint)

C.1 Using lprint

This is a description of **lprint** in the version of 06/23/93. **lprint** makes it possible to print to a local printer (attached to your terminal). It works similar as the HP Eloquence PRINTER IS 10 statement.

Usage:

```
lprint [-r] [file]
```

You may either specify a filename to lprint or pass the data into stdin. The **-r** flag specifies raw printing mode. Normally lprint will expand a newline character (`\n`) into a `\r\n` sequence. You may suppress this behaviour by giving the **-r** option.

For example:

```
ll | lprint
lprint -r sample.DATA
```

C.2 How to configure lprint to different terminals

Terminal control is terminal type dependent. Normally, all required control sequences should be defined properly by the HP Eloquence terminfo descriptions.

1. `lprint` uses the `TERM` environment variable to identify the terminal type. A corresponding (compiled) terminfo description is required.

   - If the `TERMINFO` environment variable is undefined:
     The following directories are searched for a terminfo description (in this order):

     ```
     /usr/eloquence/terminfo
     /usr/lib/terminfo
     ```

- If the `TERMINFO` environment variable is defined (eg. /usr/uap/terminfo):

  The directories defined by the `TERMINFO` variable are searched first (eg. /usr/uap/terminfo). If no terminfo description could be found, the following directories are searched for a terminfo description:

  ```
  /usr/eloquence/terminfo
  /usr/lib/terminfo
  ```

2. If the LONG terminal name starts with "`hp` ", hp terminal printer protocol (S/U/F response after output) will be assumed, else no protocol will be assumed.

   For example:

   ```
   70092|70092a|70092A|hp 7009x/239x series,
                       ^^^
                       This is a HP Terminal

   70060|70060 Terminal(vt320; 7 bit),
         ^^^
         This is no HP Terminal

   c1003|c1003a|1003|1003a|700-41,
                           ^^^
                           This is no HP Terminal
   ```

3. `lprint` needs the following control sequence(s) (from terminfo) to activate the printer:

   Either

   ```
   prtr_non    (activate printer for n characters)
   ```

   or

   ```
   prtr_on     (activate printer)
   prtr_off    (switch printer off)
   ```

   The following keywords are used in the terminfo description:

Name	Keyword	700/92	vt320	c1003
prtr_non	mc5p	mc5p=\E&p%p1%dW		
prtr_on	mc5	mc5=\E&p13C	mc5=\E[5i	mc5=^R
prtr_off	mc4	mc4=\E&p11C	mc4=\E[4i	mc4=^T

Index