VALENTINA SQL

Paradigma (www.paradigmasoft.com)

Acknowledgments:

**Andy Bachorski , Andy Fuchs, Bill Mounce, Brian Blood,
Craig A. Berry, David A. Bayly, Frank J. Schima, Guillermo Zaballa,
Hideaki Iimori, John Roberts, Lynn Fredricks, Paul Shaap, Robert Brenstein.**

Contents

Querying a database

Valentina uses SQL to search a database.

This is not a tutorial on using SQL. It contains only brief review of querying with SQL and the points where the SQL of Valentina differs from standard SQL. Valentina supports a subset of SQL, which allows you to perform queries of a database. Features of standard SQL, which are not described in this manual are not implemented.

Don't forget that SQL is case insensitive. The following is an example of the syntax of SQL as used in Valentina:

SELECT Columnlist
FROM [DISTINCT] Tablelist
[WHERE SearchCondition]
[ORDER BY SortCondition]

ColumnList = * | ** | FieldName | TableName.* | TableName.** | TableName.FieldName
• List of fields of BaseObjects separated by commas.
• Using the asterisk (*) in place of the fieldnames retrieves all fields in the table.
• If a column name has spaces in it, it must be surrounded by brackets, e.g., [first name] should be used for the field ‘first name’.
• If tablelist refers to multiple tables and there is ambiguity between names of fields then you can use the syntax TableName.FieldName to refer to a field. This syntax is called the full name of field.
• Each BaseObject of Valentina has an interanl field “RecID” which you can select to be shown in the cursor.

TableList = TableName | TableName alias
• List of tables (BaseObjects) separated by commas.
• If a tablename has spaces in it, it must be surrounded by brackets, e.g., [Product Groups]
• For TableName you can specify alias which can be used in the full name of field. Besides aliases are required for join of a Table(s) having self-recursion or ambiguous links.
• If you specify DISTINCT before TableList then on JOINValentina will do Object-Distinct, i.e. remove repeated pairs of records from Result table.

SearchCondition
• List of conditions that specify a subset of rows in the resulting table. See detailed description and examples below.

SortCondition
• List of fields on which resulting table (cursor) must be sorted. By default, the rows appear in ascending order. If you wish to use descending order, include the modifier DESC, i.e., “ORDER BY invoices.date DESC”

Statements SELECT and FROM are required. Statements WHERE and ORDER BY are optional. If statement WHERE is not specified then all records are selected.

Querying a single table

Assume that our database has only one table “Movie”.

MOVIE

Title	Director	price	sales
bad boy	Bill Adams	500'000	1'200'000
good boy	Sue Smith	356'000	950'000
bad girl	Sam Clark	653'000	860'000
good girl	Bob Smith	250'000	790'000
bad guys	Larry Fitch	640'000	1'800'000
good guys	Paul Cruz	750'000	1'100'000
dingo	Tom Snyder	265'000	930'000
black sea	Dan Roberts	420'000	685'000

The most simple SQL query which selects all fields and all rows of the table “Movie” is:

“select * from movie”.

The result of this query is the same table shown above

If you want to select some columns from the table you can write:

“select Title, Director from movie”

Title	Director
bad boy	Bill Adams
good boy	Sue Smith
bad girl	Sam Clark
good girl	Bob Smith
bad guys	Larry Fitch
good guys	Paul Cruz
dingo	Tom Snyder
black sea	Dan Roberts

You can select any fields in any order. The order of the fields in the SELECT statement affects the order of fields in the resulting table.

Note, a query may use one or several tables. The result of the query is ONE table, which contains selected fields and records which match specified search conditions. This table is called - **the Result Table**.

Now if you wish to select movies which costs more than 500’000 and sort the result by price you can use the query:

```
SELECT Title, price FROM movie
WHERE price >= 500000
ORDER BY price
```

Title	price
bad boy	500'000
bad guys	640'000
bad girl	653'000
good guys	750'000

Search expresions

In the WHERE statement you can can use the expressions shown below:

```
[NOT] FieldName { > | >= | < | <= | = | <> } value
IS [NOT] NULL
LIKE ‘StringValue’ [NO_CASE]
```

Any expression in brackets [ ] is optional.

- String, date and time values must be in the single quotas: ‘Brian’.
- You can use the operators AND and OR to build complex expressions as well as brackets ().

EXAMPLES:

```
WHERE price >= 500000 and price < 700’000
WHERE NOT(price >= 500000 and price < 700’000)
WHERE price <> 500000
WHERE price >= 500000 and sales < 1000000
WHERE sales is NULL
WHERE sales is not null
WHERE title = ‘bad boy’
WHERE title like ‘boy’
WHERE director like ‘PAUL’ no_case
WHERE bool_fld = 1
WHERE Date >= ‘1/1/99’ and Date < ‘1/2/99’
```

NOTES:

- For boolean fields you may use these values to make the comparison:
  (1/0) | (true/false) | (yes/no) | (on/off).

String Search

You can search for strings which “begins from ‘substr’ ” 2 ways:

1) WHERE left(name, 3) = ‘Bri’
This way uses the index of the string field to search. Actually Valentina don’t use second parameter of function left. It is always case sensitive.

2) Use RegEx search via LIKE:
WHERE name LIKE ‘Bri’ [no_case]
In this case the index is not used.

LIKE search

For string search on partial match is used SQL keyword LIKE. Valentina can do case insensitive LIKE-search if you specify no_case:
WHERE director like ‘PAUL’ no_case

For search on string which CONTAINS ‘StrValue’ use syntax:
WHERE str_fld LIKE ‘Comput’ no_case

For search on string which START WITH ‘StrValue’
WHERE str_fld LIKE ‘\AComput’ no_case

For search on string which ENDS WITH ‘StrValue’
WHERE str_fld LIKE ‘Comput\Z’ no_case

For search on string which EXACT match to ‘StrValue’
WHERE str_fld LIKE ‘\ANew York\Z’ no_case

RegEx search

Actually Valentina consider string for LIKE as a Regular Expression and perform search on it. The syntax of RegEx can be found in the folder “Syntax of RegEx”.

RegEx search never uses the index of a field because it needs to check all records. So time of search by RegEx is linear to the number of records in a Table.

RegEx search can be case insensitive For this you should use switch
(?i) to set case insensitive search ON for the folowing string
(?-i) to set case insensitive search OFF for the folowing string

Example:
“(?i)I AM CASE INSENSITIVE(?-I)I am Case Sensitive”

Querying two related tables

To establish a relationship between two tables, Valentina offers a special mechanism - fields of type **ObjectPtr**. We will call this type of relation an **ObjectPtr-link**, while the old RDBMS way is called a **RDB-link.**

A developer can choose either the traditional RDBMS or modern Valentina technique to establish a relation between tables, furthermore, he can mix both techniques to get the best solution (see “ValentinaKernel.pdf”, part “Relations between Tables” for more details).

Suppose we have 2 related tables.

MOVIE

	Title	Director	price	sales
1	bad boy	Bill Adams	500'000	1'200'000
2	good boy	Sue Smith	356'000	950'000
3	bad girl	Sam Clark	653'000	860'000
4	good girl	Bob Smith	250'000	790'000
5	bad guys	Larry Fitch	640'000	1'800'000
6	good guys	Paul Cruz	750'000	1'100'000
7	dingo	Tom Snyder	265'000	930'000
8	black sea	Dan Roberts	420'000	685'000

ACTORS

Name	Salary	Movie_ptr
Dandy	25000	3
First	32000	2
Second	45000	2
Last	33000	1
John	67500	5
Frank	55200	3
Susy	45000	3
Kat	32000	2
Bob	46000	4
...	...	...

The second table “Actors” is related to the first one with help of the field “Movie_ptr”. The value of this field stores the ID of the movie in which this actor plays. From these fragments of Tables we can see that the movie “Bad girl” (RecID=3) has actors Dandy, Frank and Suzy.

This is the standard way of relating tables for RDBMS - using an additional field of the table as a pointer to records in the related table.

Note, for this kind of relation ONE movie can be related with MANY actors. That is why such a relation is called as [ONE : MANY], [1:M].

Now let’s to see how we can query these 2 tables.
To see all the actors who play in the movie
“bad girl” we can write:

```
SELECT Title, Name
FROM Movie, Actors
WHERE title = ‘bad girl’ and movie.RecID = movie_ptr
ORDER BY name
```

Title	Name
bad girl	Dandy
bad girl	Frank
bad girl	Susy

• each BaseObject of Valentina has internal field with name “RecID” which you can use. (see “ValentinaKernel.pdf”, part “BaseObject” for more details)

As you can see we need an additional condition to relate the records in the 2 tables:

movie.RecID = actors.movie_ptr.

These kind of conditions we will call **LINK CONDITIONS**, while conditions like

title = ‘bad girl’ we will call **MATCH CONDITIONS**.

Standard SQL requires that you specify both the MATCH and LINK conditions in the query. Valentina is smart enough to resolve a query if you specify MATCH conditions only. Valentina requires LINK conditions only in complex cases when there is ambiguity in the way of resolving of query. So, the previous query for Valentina can looks like:

```
SELECT Title, Name FROM Movie, Actors
WHERE title = ‘bad girl’ ORDER BY name
```

The following query will select only one record { “bad girl”, Dandy }:

```
SELECT Title, Name FROM Movie, Actors
WHERE title = ‘bad girl’ and salary < 30000 ORDER BY name
```

Valentina Extensions of SQL

Field “RecID”

Each BaseObject contains an internal virtual (i.e. it does not take place on disk) field with the name “RecID”, which returns the value of the physical number for the current record (see “ValentinaKernel.pdf”, part “BaseObject” for more details).

Syntax “SELECT * ... ” does not select this field by default to be compatible with standard SQL. So you need to specify this field explicitly:

SELECT RecID, * FROM Person
SELECT Person.RecID, Task.RecID, Person.*, Task.Name FROM Person, Task

BaseObject Methods

Valentina support BaseObject Methods – a special kind of field which is calculated on the fly by the specified expression and does not take place on disk. To work correctly and easy with this feature Valentina expands the syntax of SQL and provides the following rules.

To be compatiable with SQL standard the folowing query select only real fields of BaseObjectA:

“SELECT * FROM BaseObjectA ...”

To select all real Fields and all Methods of a BaseObject Valentina uses the following syntax:

“SELECT ** FROM BaseObjectA ...”
“SELECT Person.Name, Task.** FROM Person, Task”

You can mix “SELECT * ” and explicit selection of needed Methods:

SELECT RecID, *, FullName, From Person

Expressions

You can build complex expressions using the operators and functions listed below. In an expression you can mix numeric and string constants with names of table fields. Table fields with a space in its name must be enclosed in brackets [the name]

In this part we will use the following variable names:
a, b - BOOLEAN values
x,y - DOUBLE values (real numbers)
xi,yi - INTEGER values
s, s1, str - STRING values.

Operators

The following table list supported operators in the priority order

()
OR
AND
= <> != > >= < <=
<< >>
+ -
* / %
NOT

(exp) **paranthesis, allow you change order of calculation.**

Example: (total + 500) * 2
Result: any

a OR b **logical OR operator**

Parameters: Boolean
Result Type: Boolean
Example: name = ‘’ OR name < ‘c’

a AND b **logical AND operator**

Parameters: Boolean
Result Type: Boolean
Example: name >= ‘c’ AND name < ‘d’

NOT a **logical NOT operator**

Parameters: Boolean
Result Type: Boolean
Example: NOT(name >= ‘c’ AND name < ‘d’)

a != b
a <> b **operator “not equal” (2 forms supported)**

Parameters:	Numbers or Strings	
Result Type:	Boolean	
Example:	‘aaa’ <> ‘’	-> true
	‘aaa’ != ‘’	-> true
	5 != 5	-> false

a = b **operator “equal”**

Parameters:	Numbers or Strings	
Result Type:	Boolean	
Example:	‘aaa’ = ‘’	-> false
	5 = 5	-> true

a > b **operator “greater than”**

Parameters:	Numbers or Strings	
Result Type:	Boolean	
Example:	‘aaa’ > ‘b’	-> false
	5 > 3	-> true

a >= b **operator “greater than or equal”**

Parameters:	Numbers or Strings	
Result Type:	Boolean	
Example:	‘aaa’ => ‘b’	-> false
	5 => 5	-> true

a < b **operator “less than”**

Parameters:	Numbers or Strings	
Result Type:	Boolean	
Example:	‘aaa’ < ‘b’	-> true
	5 < 5	-> false

a <= b **operator “less than or equal”**

Parameters:	Numbers or Strings	
Result Type:	Boolean	
Example:	‘aaa’ <= ‘b’	-> true
	5 <= 3	-> false

x + y **operator addition**

Parameters: DOUBLE or INTEGER
Result Type: DOUBLE or INTEGER, depending on x and y
Example: 2 + 5 -> 7
2.2 + 5 -> 7.2

x - y **operator subtraction**

Parameters: DOUBLE or INTEGER
Result Type: DOUBLE or INTEGER, depending on x and y
Example: 2 - 5 -> -3
2.2 - 5 -> -2.8

x * y **operator multiplication**

Parameters: DOUBLE or INTEGER
Result Type: DOUBLE or INTEGER, depending on x and y
Example: 2 * 5 -> 10
2.3 * 5 -> 11.5

x / y **operator division**

If value of y is 0 returns NULL.

Parameters: DOUBLE or INTEGER
Result Type: DOUBLE or INTEGER, depending on x and y
Example: (total + 500) * 2

xi % yi **operator module**

If value of y is 0 returns NULL.

Parameters: INTEGER
Result Type: INTEGER
Example: 5 % 2 -> 1

-x **negative operator.**

Parameters: DOUBLE or INTEGER
Result Type: DOUBLE or INTEGER, depending on x and y
Example: -(2+3) -> -5

xi << yi

Parameters:	INTEGER	
Result Type:	INTEGER	
Example:	1 << 3	-> 8

xi >> yi

Parameters:	INTEGER	
Result Type:	INTEGER	
Example:	8 >> 2	-> 2

Control flow functions

IF(exp1, exp2, exp3)

• If expr1 is TRUE (expr1 <> 0 and expr1 <> NULL) then IF() returns expr2, else it returns expr3. IF() returns a numeric or string value, depending on the context in which it is used.

Example: IF(5 > 2, ‘aaa’, ‘bbb’) -> ‘aaa’
IF(5 < 2, ‘aaa’, ‘bbb’) -> ‘bbb’
IF(fld IS NULL, ‘n/a’, fld)

IFNULL(exp1, exp2)

• If expr1 is not NULL, IFNULL() returns expr1, else it returns expr2. IFNULL() returns a numeric or string value, depending on the context in which it is used.

• IFNULL() is short form of IF(exp1 IS NULL, exp2, exp1)

Example: IFNULL(fld, ‘n/a’)

Mathematical functions

double ABS(double x)

• The function returns the absolute value for a given number

Example: abs(- 5.3) -> 5.3
abs(5.3) -> 5.3

integer SIGN(double x)

• The function returns +1 if a given number is positive
and it returns -1 if the number is negative.

Example: sign(- 5.3) -> -1
abs(5.3) -> +1

double SQRT(double x)

• The function returns the square root of any number.

Example: sqrt(9) -> 3

double SQUARE(double x)

• The function returns the square of any number.

Example: square(3) -> 9

double POW(double x, double y)

• The function performs the multiplication operation.

Example: pow(2, 4) -> 16

double EXP(double x)

• The function returns e^x (e to the xth power) for any number x.

Example: exp(3.9) -> 49.4024491055

double LOG(double x)

• The function returns the natural logarithm of any number x.
• If value is 0 or negative, log() returns NULL.

Example: log(3) -> 1.09861228867

double LOG10(double x)

• The function returns the base-10 logarithm of any number x.
• If value is 0 or negative, log10() returns NULL.

Example: log10(3) -> 0.047712125472
log10(1000) ->3

double CEILING(double x)

• The function rounds the supplied value to the next highest integer.

Example:	floor(1.1)	-> 2.0
	floor(-1.1)	-> -1.0

double FLOOR(double x)

• The function rounds any number down to the next lowest integer.

Example:	floor(1.1)	-> 1.0
	floor(-1.1)	-> -2.0

double ROUND(double x, [integer d])

• The function returns the argument X, rounded to a number with D decimals. If D is 0, the result will have no decimal point or fractional part.

Example:	round(1.2345678, 2)	-> 1.23
	round(1.2385678, 2)	-> 1.24
	round(123456.23, -2)	-> 123500

double TRUNCATE(double x, [integer d])

• The function returns the number X, truncated to d decimals.

Example:	truncate(1.2345678, 2)	-> 1.23
	truncate(1.2385678, 2)	-> 1.23

Trigonometric functions

All following functions work with radiant values.

double RADIANS(double x)

• The function converts degree values in radiant values.

Example: radians(180) -> 3.1415...

double SIN(double x)

• The function returns the sine for any number.

Example: sin(13.5) -> 0.803784

double COS(double x)

• The function returns the cosine for any number.

Example: cos(1) -> 0.540302

double TAN(double x)

• The function returns the tangent for any number.

Example: tan(1) -> 1.55741

double ASIN(double x)

• The function returns the principal value of the arc sine (sin-1), for any number in the range of -1 to +1.

Example: asin(0.25) -> 0.25268

double ACOS(double x)

• The function returns the principal value of the arc cosine (cos-1), for any number in the range of -1 to +1.

Example: acos(0.25) -> 1.31812

double ATAN(double x)

• The function returns the principal value of the arc tangent (tan-1) for any number.

Example: atan(0.25) -> 0.244979

double SINH(double x)

• The function returns the hyperbolic sine for any number.

Example: sinh(13.5) -> 364708.18498

double COSH(double x)

• The function returns the hyperbolic cosine for any number.

Example: cosh(1) -> 1.5430806348

double TANH(double x)

• The function returns the hyperbolic tangent for any number.

Example: tanh(1) -> 0.761594

String functions

integer LENGTH(string Str)

• The function computes the length of a given string

Example: length(‘tree’) -> 4

string CONCAT(string s1, string s2, ...)

• The function returns the concatenation of 2 or more strings.

Example: concat(‘Day’, ‘ ‘, ‘of birthday’) -> ‘Day of birthday’

string LEFT(string Str, integer Count)

•The function returns left part of Str up to Count characters.

Example: left(‘Valentina’,4) -> ‘Vale’

string RIGHT(string Str, integer Count)

•The function returns right part of Str up to Count characters.

Example: right(‘Valentina’,4) -> ‘tina’

string SUBSTR(string Str, integer Pos, [integer Count])

• The function returns a specified substring from a given string.
• If count is not specified then returns substr up to the end of Str.

Example: substr(‘Valentina’, 2, 4) -> ‘alen’

string LOCATE(string SubStr, string Str, [integer Pos])

• The function returns the position of the first occurrence of substring substr in string str, starting at position pos. Returns 0 if substr is not in str.

Example: locate(‘bar’, ‘foobarbar’) -> 4
locate(‘bar’, ‘foobarbar’, 5) -> 7

string INSERT(string Str, integer Pos, integer Count, string NewStr)

• The function inserts a specified string in another string at a given position.

Example: insert(‘Valentina’,2, 3, ‘fast’) -> ‘Vfasntina’
insert(‘Valentina’,9, 3, ‘fast’) -> ‘Valentinafas’

string REPLACE(string Str, string StrToReplace, string NewStr)

• The function returns the string Str with all occurrences of the StrToReplace replaced by the NewStr.

Example: replace(‘www.paradigmasoft.com’, ‘w’, ‘Vv’)
-> ‘VvVvVv.paradigmasoft.com’

string REVERSE(string Str)

• The function performs a string conversion. It reverse order of characters.

Example: reverse(‘Valentina’) -> ‘anitnelaV’

string LTRIM(string Str)

• The function returns the given string value without leading blanks.

Example: ltrim(‘ Valentina ‘) -> ‘Valentina ’

string RTRIM(string Str)

• The function returns the given string value without trailing blanks.

Example: rtrim(‘ Valentina ‘) -> ‘ Valentina’

string TRIM(string Str)

• The function returns the given string value without leading and trailing blanks.

Example: trim(‘ Valentina ‘) -> ‘Valentina’

string UPPER(string Str)

• The function performs a string conversion. All character are changed to their uppercase representation.

Example: upper(‘Valentina’) -> ‘VALENTINA’

string LOWER(string Str)

• The function performs a string conversion. All character are changed to their lowercase representation.

Example: lower(‘Valentina’) -> ‘valentina’

string SPACE(integer Count)

• The function returns a string consisting of a given number of spaces.

Example: space(8) -> ‘ ’

string REPEAT(string Str, integer Count)

• The function returns a string which contains Str repeated Count times..

Example: repeat(‘cool’, 3) ->’coolcoolcool’

string LPAD(string Str, integer UpToLength, string PadStr)

• The function returns the string Str, left-padded with the string PadStr until Str is UpToLength characters long.

Example: lpad(‘hi’, 5, ‘?’) -> ‘???hi’

string RPAD(string Str, integer UpToLength, string PadStr)

• The function returns the string Str, right-padded with the string PadStr until Str is UpToLength characters long.

Example: rpad(‘hi’, 5, ‘?’) -> ‘hi???’

integer STRSPN(string Str, string CharSet)

• Returns the length of the initial segment of a Str that contains only characters from a CharSet. The value is also a position of a first character in Str that is not in the Charset.

Example: strspn(‘aesop’, ‘ea’) -> 2

See also: STRCSPN(), HEAD(), TAIL()

integer STRCSPN(string Str, string CharSet)

• Returns the length of the initial segment of a Str that contains no characters from a CharSet. The value is also a position of a first character in Str that is in the Charset.

Example: strspn(‘aesop’, ‘pso’) -> 2

See also: STRSPN(), HEAD(), TAIL()

string HEAD(string Str, string CharSet)

• Returns the left substring of a Str that contains only characters from a CharSet.
• HEAD() allows using of operator NOT for Charset.

Example: HEAD(‘aesop’, ‘ea’) -> ‘ae’
HEAD(‘aesop’, not(‘spo’)) -> ‘ae’

See also: STRSPN(), STRCSPN(), TAIL()

string TAIL(string Str, string CharSet)

• Returns the right substring of a Str that contains characters after all initial charactres that are in CharSet.
• TAIL() allows using of operator NOT for Charset.
• TAIL() can be specified by rule:
concate(HEAD(s,set), TAIL(s,set)) -> returns original s

Example: HEAD(‘aesop’, ‘ea’) -> ‘sop’
HEAD(‘aesop’, not(‘spo’)) -> ‘sop’

See also: STRSPN(), STRCSPN(), TAIL()

Date / Time functions

int YEAR(date)

• Returns the year for date

Example: Year(‘23-01-1998’) -> 1998

int MONTH(date)

•Returns the month for date, in the range 1 to 12.

Example: Month(‘23-01-1998’) -> 01

int DAYOFMONTH(date)

• Returns the day of the month for date, in the range 1 to 31.

Example: DayOfMonth(‘23-01-1998’) -> 23

int DAYOFWEEK(date)

• Returns the weekday index for date (1 = Sunday, 2 = Monday, ... 7 = Saturday). These index values correspond to the ODBC standard.

Example: DayOfWeek(‘03-02-1998’) -> 3

int WEEKDAY(date)

• Returns the weekday index for date (0 = Monday, 1 = Tuesday, ... 6 = Sunday).

Example: WeekDay(‘03-02-1998’) -> 1

int DAYOFYEAR(date)

• Returns the day of the year for date, in the range 1 to 366.

Example: DayOfYear(‘03-02-1998’) -> 34

int WEEK(date, [first])

• With a single argument, returns the week for date, in the range 0 to 52, for locations where Sunday is the first day of the week. The two-argument form of WEEK() allows you to specify whether the week starts on Sunday or Monday. The week starts on Sunday if the second argument is 0, on Monday if the second argument is 1.

Example: Week('20-02-1998') -> 7
Week('20-02-1998' , 0) -> 7
Week('20-02-1998' , 1) -> 8

int QUARTER(date, [first])

• Returns the quarter of the year for date, in the range 1 to 4.

Example: QUARTER('01-04-1998') -> 1

int PERIOD_ADD(P, N)

• Adds N months to period P (in the format YYMM or YYYYMM). Returns a value in the format YYYYMM. Note that the period argument P is not a date value.

Example: PERIOD_ADD(9801, 2) -> 199803

int PERIOD_DIFF(P1, P2)

• Returns the number of months between periods P1 and P2. P1 and P2 should be in the format YYMM or YYYYMM. Note that the period arguments P1 and P2 are not date values.

Example: PERIOD_DIFF(9802, 199703) -> 11

int TO_DAYS(date)

• Given a date date, returns a daynumber (the number of days since year 0). TO_DAYS() is not intended for use with values that precede the advent of the Gregorian calendar (1582).

Example: TO_DAYS('07-10-1997') -> 729669

date FROM_DAYS(int N)

• Given a daynumber N, returns a DATE value.

Example: FROM_DAYS(729669) -> '07-10-1997'

int HOUR(time)

• Returns the hour for time, in the range 0 to 23.

Example: Hour(‘15:35:16’) -> 15

int MINUTE(time)

• Returns the minute for time, in the range 0 to 59.

Example: Minute(‘15:35:16’) -> 35

int SECOND(time)

• Returns the second for time, in the range 0 to 59.

Example: Minute(‘15:35:16’) -> 16

int SEC_TO_TIME(time)

• Returns the seconds argument, converted to hours, minutes and seconds, as a value in 'HH:MM:SS'

Example: SEC_TO_TIME(2378) -> ‘00:39:38’

int TIME_TO_SEC(time)

• Returns the time argument, converted to seconds.

Example: TIME_TO_SEC(‘00:39:38’) -> 2378