Basic Syntactical Items

This chapter describes basic syntactical items, which are not explained any further within the individual SQL statement descriptions.

This chapter covers the following topics:

- Constants
- Names
- Parameters
- Natural Formats and SQL Data Types

Constants

The constants used in the syntactical descriptions of the Natural SQL statements are:

- *constant*
- *integer*

These items are described below.

constant	The item *constant* always refers to a Natural constant.
integer	The item *integer* always represents an integer constant.

Note:
If the character for decimal point notation (session parameter DC) is set to a comma (,), any specified numeric constant must not be followed directly by a comma, but must be separated from it by a blank character; otherwise an error or wrong results occur.

Invalid Syntax:	**Valid Syntax:**
`VALUES (1,'A')  leads to a syntax error` `VALUES (1,2,3)  leads to wrong results`	`VALUES (1 ,'A')` `VALUES (1 ,2 ,3)`

Names

The names used in the syntactical descriptions of the Natural SQL statements are:

- *authorization-identifier*
- *ddm-name*
- *view-name*

- *column-name*
- *table-name*
- *correlation-name*

These items are described below.

authorization-identifier	The item *authorization-identifier*, which is also called creator name, is used to qualify database tables and views. See also below.
ddm-name	The item *ddm-name* always refers to the name of a Natural data definition module (DDM) as created with the Natural utility SYSDDM.
view-name	The item *view-name* always refers to the name of a Natural view as defined in the DEFINE DATA statement.
column-name	The item *column-name* always refers to the name of a physical database column.

table-name	Syntax:
	
	[*location-name*.][*authorization-identifier*.]*ddm-name*
	
	The item *table-name* in this section is used to reference both SQL base tables and SQL viewed tables.
	
	ddm-name
	
	A Natural data definition module (DDM) must have been created for a table to be used. The name of such a DDM must be the same as the corresponding database table name or view name.
	
	location-name
	
	This optional item specifies the location of the table to be accessed.
	
	authorization-identifier
	
	There are two ways of specifying the *authorization-identifier* of a database table or view.
	
	One way corresponds to the standard SQL syntax, in which the *authorization-identifier* is separated from the table name by a period. Using this form, the name of the DDM must be the same as the name of the database table without the *authorization-identifier*.
	
	Example:
	
	`DEFINE DATA LOCAL`
	`01 PERS VIEW OF PERSONNEL`
	`  02 NAME`
	`  02 AGE`
	`END-DEFINE`
	`SELECT *`
	`  INTO VIEW PERS`
	`  FROM SQL.PERSONNEL`
	`...`
	
	Alternatively, you can define the *authorization-identifier* as part of the DDM name. The DDM name then consists of the *authorization-identifier* and the database table name separated by a hyphen (-). The hyphen between the *authorization-identifier* and the table name is converted internally into a period.
	
	Note:
	This form of DDM name can also be used with a FIND or READ statement, because it conforms to the DDM naming conventions applicable to these statements.
	
	Example:
	
	`DEFINE DATA LOCAL`
	`01 PERS VIEW OF SQL-PERSONNEL`
	`  02 NAME`
	`  02 AGE`
	`END-DEFINE`
	`SELECT *`
	`  INTO VIEW PERS`
	`  FROM SQL-PERSONNEL`
	`...`
	
	If the *authorization-identifier* has been specified neither explicitly nor within the DDM name, it is determined by the SQL database system.
	
	In addition to being used in SELECT statements, table names can also be specified in DELETE, INSERT and UPDATE statements.
	
	Examples:
	
	`...`
	`DELETE FROM SQL.PERSONNEL`
	`  WHERE AGE IS NULL`
	`  ...`
	
	`...`
	`INSERT INTO SQL.PERSONNEL (NAME,AGE)`
	`  VALUES ('ADKINSON',35)`
	`...`
	
	`...`
	`UPDATE SQL.PERSONNEL`
	`SET SALARY = SALARY * 1.1`
	`WHERE AGE > 30`
	`...`

correlation-name	The item *correlation-name* represents an alias name for a *table-name*. It can be used to qualify column names; it also serves to implicitly qualify fields in a Natural view when used with the INTO clause of the SELECT statement.
	
	Example:
	
	`DEFINE DATA LOCAL`
	`01 PERS-NAME (A20)`
	`01 EMPL-NAME (A20)`
	`01 AGE (I2)`
	`END-DEFINE`
	`...`
	`SELECT X.NAME , Y.NAME , X.AGE`
	`INTO PERS-NAME , EMPL-NAME , AGE`
	`FROM SQL-PERSONNEL X , SQL-EMPLOYEES Y`
	`WHERE X.AGE = Y.AGE`
	`END-SELECT`
	`...`
	
	Although in most cases the use of *correlation-names* is not necessary, they may help to make the statement clearer.

Parameters

parameter

[:] *host-variable* [**INDICATOR** [:] *host-variable*] [**LINDICATOR** [:] *host-variable*]

The syntax items are described below:

See also *Natural Formats and SQL Data Types*.	
host-variable	A *host-variable* is a Natural user-defined variable (no system variable) which is referenced in an SQL statement. It can be either an individual field or defined as part of a Natural view. When defined as a receiving field (for example, in the INTO clause), a *host-variable* identifies a variable to which a value is assigned by the database system. When defined as a sending field (for example, in the WHERE clause), a *host-variable* specifies a value to be passed from the program to the database system. See also *Natural Formats and SQL Data Types*.

[:]	**Colon:**
	
	To comply with SQL standards, a host-variable can also be prefixed by a colon (:). When used with flexible SQL, host-variables must be qualified by colons.
	
	Example:
	
	`SELECT NAME INTO :#NAME FROM PERSONNEL`
	`WHERE AGE = :VALUE`
	
	The colon is always required if the variable name is identical to an SQL reserved word. In a context in which either a host-variable or a column can be referenced, the use of a name without a colon is interpreted as a reference to a column.

INDICATOR	**INDICATOR Clause:**
	
	The INDICATOR clause is an optional feature to distinguish between a "null" value (that is, no value at all) and the actual values 0 or "blank".
	
	When specified with a receiving *host-variable* (target field), the INDICATOR *host-variable* (null indicator field) serves to find out whether a column to be retrieved is "null".
	
	Example:
	
	`DEFINE DATA LOCAL`
	`1 NAME (A20)`
	`1 NAMEIND (I2)`
	`END-DEFINE`
	`SELECT *`
	`INTO NAME INDICATOR NAMEIND`
	`...`
	
	In this example, NAME represents the receiving *host-variable* and NAMEIND the null indicator field.
	
	If a null indicator field has been specified and the column to be retrieved is null, the value of the null indicator field is negative and the target field is set to 0 or "blank" depending on its data type. Otherwise, the value of the null indicator field is greater than or equal to 0.
	
	When specified with a sending *host-variable* (source field), the null indicator field is used to designate a null value for this field.
	
	Example:
	
	`DEFINE DATA LOCAL`
	`1 NAME (A20)`
	`1 NAMEIND (I2)`
	`UPDATE ...`
	`SET NAME = :NAME INDICATOR :NAMEIND`
	`WHERE ...`
	
	In this example, :NAME represents the sending host-variable and :NAMEIND the null indicator field. By entering a negative value as input for the null indicator field, a null value is assigned to a database column.
	
	An INDICATOR *host-variable* is of format/length I2.

LINDICATOR	**LINDICATOR Clause:**
	
	The LINDICATOR clause is an optional feature which is used to support columns of varying lengths, for example, VARCHAR or LONG VARCHAR type.
	
	When specified with a receiving *host-variable* (target field), the LINDICATOR *host-variable* (length indicator field) contains the number of characters actually returned by the database into the target field. The target field is always padded with blanks.
	
	If the VARCHAR or LONG VARCHAR column contains more characters than fit in the target field, the length indicator field is set to the length actually returned (that is, the length of the target field) and the null indicator field (if specified) is set to the total length of this column.
	
	Example
	
	`DEFINE DATA LOCAL`
	`1 ADDRESSLIND (I2)`
	`1 ADDRESS (A50/1:6)`
	`END-DEFINE`
	`SELECT *`
	`  INTO :ADDRESS(*) LINDICATOR :ADDRESSLIND`
	`  ...`
	
	In this example, :ADDRESS(*) represents the target field which receives the first 300 bytes (if available) of the addressed VARCHAR or LONG VARCHAR column, and :ADDRESSLIND represents the length indicator field which contains the number of characters actually returned.
	
	When specified with a sending *host-variable* (source field), the length indicator field specifies the number of characters of the source field which are to be passed to the database.
	
	Example:
	
	`DEFINE DATA LOCAL`
	`1 NAMELIND (I2)`
	`1 NAME (A20)`
	`1 AGE (I2)`
	`END-DEFINE`
	`MOVE 4 TO NAMELIND`
	`MOVE 'ABC%' TO NAME`
	`SELECT AGE`
	`  INTO :AGE`
	`WHERE NAME LIKE :NAME LINDICATOR :NAMELIND`
	`  ...`
	
	A LINDICATOR *host-variable* is of format/length I2 or I4. For performance reasons, it should be specified immediately before the corresponding target or source field; otherwise, the field is copied to the temporary storage at runtime.
	
	If the LINDICATOR field is defined as an I2 field, the SQL data type VARCHAR is used for sending or receiving the corresponding column. If the LINDICATOR *host-variable* is specified as I4, a large object data type (CLOB/BLOB) is used.
	
	If the field is defined as DYNAMIC, the column is read in an internal loop up to its real length. The LINDICATOR field and *LENGTH are set to this length. In case of a fixed length field, the column is read up to the defined length. In both cases, the field is written up to the value defined in the LINDICATOR field.
	
	Let a fixed length field be defined with a LINDICATOR field specified as I2. If the VARCHAR column contains more characters than fit into this fixed length field, the length indicator field is set to the length actually returned and the null indicator field (if specified) is set to the total length of this column (retrieval). This is not possible for fixed length fields >= 32 KB (length does not fit into null indicator field).

Natural Formats and SQL Data Types

The Natural format of a host-variable is converted to an SQL data type according to the following table:

Natural Format/Length	SQL Data Type
A*n*	CHAR (*n*)
B2	SMALLINT
B4	INT
B*n*; *n* not equal to 2 or 4	CHAR (*n*)
F4	REAL
F8	DOUBLE PRECISION
I2	SMALLINT
I4	INT
N*nn.m*	NUMERIC (*nn*+*m*,*m*)
P*nn.m*	NUMERIC (*nn*+*m*,*m*)
T	TIME
D	DATE
G*n*; for view fields only	GRAPHIC (*n*)

Natural does not check whether the converted SQL data type is compatible to the database column. Except for fields of format N, no data conversion is done.

In addition, the following extensions to standard Natural formats are available with Natural SQL:

- A one-dimensional array of format A can be used to support alphanumeric columns longer than 253 bytes. This array must be defined beginning with index 1 and can only be referenced by using an asterisk (*) as the index. The corresponding SQL data type is CHAR (*n*), where *n* is the total number of bytes in the array.
- A special host variable indicated by the keyword LINDICATOR can be used to support variable-length columns. The corresponding SQL data type is VARCHAR (*n*); see also the LINDICATOR clause.
- The Natural formats date (D) and time (T) can be used with Entire Access and will be converted into the corresponding database-dependent formats (see the Entire Access documentation for details)

A sending field specified as one-dimensional array without a LINDICATOR field is converted into the SQL data type VARCHAR. The length is the total number of bytes in the array, not taking into account trailing blanks.