Grundlegende Syntaxbestandteile

Dieses Kapitel behandelt grundlegende Syntaxbestandteile, die dann in den Beschreibungen der einzelnen Statements nicht mehr näher erläutert werden. Diese Teile sind:

- Konstanten
- Namen
- Parameter
- Natural-Formate und SQL-Datentypen

Konstanten

Die in den Syntaxbeschreibungen von Natural-SQL-Statements verwendeten Konstanten sind:

- *constant*
- *integer*

Diese Konstanten sind im Folgenden beschrieben.

constant	Das Element *constant* bezieht sich immer auf eine Natural-Konstante.
integer	Das Element *integer* bezieht sich immer auf eine Ganzzahl-Konstante.

Anmerkung:
Wenn das Dezimalzeichen mit dem (Session-Parameter DC) auf Komma (,) gesetzt ist, darf unmittelbar nach einer numerischen Konstanten kein Komma angegeben werden, sondern es muss ein Leerzeichen dazwischen stehen, weil es sonst zu einem Systemfehler kommt oder falschen Ergebnissen kommen kann.

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

Namen

Die in den Syntaxbeschreibungen von Natural-SQL-Statements verwendeten Namen sind:

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

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

Diese Elemente sind im Folgenden beschrieben.

authorization-identifier	Ein *authorization-identifier*, der auch "creator name" genannt wird, dient zur Qualifizierung von Datenbanktabellen und Views. Siehe auch weiter unten.
ddm-name	*ddm-name* ist jeweils der Name eines mit der Natural-Utility `SYSDDM` erzeugten Natural-DDMs.
view-name	*view-name* ist jeweils der Name eines im `DEFINE DATA`-Statement definierten Views.
column-name	*column-name* ist jeweils der Name einer physischen Datenbankspalte.

table-name	Syntax:
	
	authorization-identifier ddm-name
	
	Das Element *table-name* in diesem Kapitel dient zur Referenzierung von SQL-Basistabellen und SQL-Viewed-Tabellen. Für jede Tabelle muss ein entsprechendes Natural-DDM existieren. Der Name des DDMs muss mit dem Namen der entsprechenden physischen Datenbanktabelle bzw. des Views identisch sein.
	
	authorization-identifier
	
	Es gibt zwei Arten, den *authorization-identifier* einer Datenbanktabelle bzw. eines Views anzugeben.
	
	Die eine Art entspricht der Standard-SQL-Syntax: *authorization-identifier* und Tabellenname werden durch einen Punkt miteinander verbunden. Hierbei muss der DDM-Name dem Namen der physischen Datenbanktabelle (ohne *authorization-identifier*) entsprechen.
	
	Beispiel:
	
	`DEFINE DATA LOCAL`
	`01 PERS VIEW OF` **`PERSONNEL`**
	`  02 NAME`
	`  02 AGE`
	`END-DEFINE`
	`SELECT *`
	`  INTO VIEW PERS`
	`  FROM SQL.` **`PERSONNEL`**
	`...`
	
	Die andere Möglichkeit besteht darin, den *authorization-identifier* als Teil des DDM-Namens selbst zu definieren. Der DDM-Name besteht dann aus dem *authorization-identifier* gefolgt von einem Bindestrich (–) und gefolgt vom Namen der Datenbanktabelle. Intern wird der Bindestrich zwischen *authorization-identifier* und Tabellennamen in einen Punkt umgesetzt.
	
	Anmerkung:
	Diese Form des DDM-Namens kann auch in einem `FIND`- oder `READ`-Statement verwendet werden, da sie den für diese Statements geltenden DDM-Namenskonventionen entspricht.
	
	Beispiel:
	
	`DEFINE DATA LOCAL`
	`01 PERS VIEW OF` **`SQL-PERSONNEL`**
	`  02 NAME`
	`  02 AGE`
	`END-DEFINE`
	`SELECT *`
	`  INTO VIEW PERS`
	`  FROM` **`SQL-PERSONNEL`**
	`...`
	
	Wenn der *authorization-identifier* weder explizit noch als Teil des DDM-Namens angegeben wird, wird er vom betreffenden SQL-Datenbanksystem bestimmt.
	
	Table-names können nicht nur in `SELECT`-Statements verwendet werden, sondern auch in den Statements `DELETE`, `INSERT` und `UPDATE`.
	
	Beispiele:
	
	`...`
	`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	*correlation-name* ist ein Alias-Name für einen *table-name*. Er kann zur Qualifizierung von Spaltennamen verwendet werden. Außerdem dient er dazu, implizit Felder in einem Natural-View zu qualifizieren, der in der `INTO`-Klausel eines `SELECT`-Statements verwendet wird.
	
	Beispiel:
	
	`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`
	`...`
	
	Die Verwendung von *correlation-names* ist zwar in der Regel nicht nötig, kann aber helfen, die Lesbarkeit eines Statements zu erleichtern.

Parameter

parameter

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

Im Folgenden sind die Syntaxelemente beschrieben.

host-variable	Eine *host-variable* ist eine in einem SQL-Statement referenzierte Natural-Programmvariable (keine Systemvariable), die entweder ein eigenständiges Feld oder Teil eines Views sein kann. Wenn sie als empfangendes Feld (z.B. in einer `INTO`-Klausel) definiert wird, ist die *host-variable* ein Feld, das vom Datenbanksystem einen Wert erhält. Wenn sie als sendendes Feld (z.B. in einer `WHERE`-Klausel) definiert wird, ist die *host-variable* ein Feld, dessen Wert vom Programm an das Datenbanksystem übergeben wird. Siehe auch *Natural-Formate und SQL-Datentypen*.

[:]	**Doppelpunkt:**
	
	Gemäß den SQL-Standards kann einer *host-variable* ein Doppelpunkt (:) vorangestellt werden. Bei der Verwendung mit flexibler SQL muss ihr ein Doppelpunkt vorangestellt werden.
	
	Beispiel:
	
	`SELECT NAME INTO :#NAME FROM PERSONNEL`
	`WHERE AGE = :VALUE`
	
	Wenn ein Variablenname mit einem für SQL reservierten Wort identisch ist, ist der Doppelpunkt ebenfalls erforderlich. In Situationen, in denen entweder eine *host-variable* oder eine Spalte referenziert werden kann, wird ein Name ohne Doppelpunkt als Spaltenname interpretiert.

INDICATOR	**INDICATOR-Klausel**
	
	Diese Klausel ist optional und dient dazu, herauszufinden, ob eine zu lesende Spalte "Null" ist, d.h. keinen Wert enthält, oder tatsächlich den Wert Null (0) bzw. Leerzeichen enthält.
	
	Wenn sie mit einer empfangenden *host-variable* (Zielfeld) verwendet wird, dient die `INDICATOR` *`host-variable`* (Null-Indikatorfeld) dazu, herauszufinden, ob eine zu lesende Spalte "Null" ist.
	
	Beispiel:
	
	`DEFINE DATA LOCAL`
	`1 NAME (A20)`
	`1 NAMEIND (I2)`
	`END-DEFINE`
	`SELECT *`
	`INTO NAME INDICATOR NAMEIND`
	`...`
	
	In diesem Beispiel ist `NAME` die empfangende *host-variable* und `NAMEIND` das Null-Indikatorfeld.
	
	Ist ein Null-Indikatorfeld angegeben und die gelesene Spalte ist "Null", wird das Indikatorfeld auf einen negativen Wert und das Zielfeld je nach Datentyp auf Null (0) bzw. Leerzeichen gesetzt. Andernfalls ist der Wert des Null-Indikatorfeldes größer gleich Null (0).
	
	Wenn sie mit einer sendenden *host-variable* (Ausgangsfeld) verwendet wird, dient die `INDICATOR` *`host-variable`* dazu, dem Ausgangsfeld einen Nullwert zuzuweisen.
	
	Beispiel:
	
	`DEFINE DATA LOCAL`
	`1 NAME (A20)`
	`1 NAMEIND (I2)`
	`UPDATE ...`
	`SET NAME = :NAME INDICATOR :NAMEIND`
	`WHERE ...`
	
	In diesem Beispiel ist `:NAME` die sendende *host-variable* und `:NAMEIND` das Null-Indikatorfeld. Durch Eingabe eines negativen Wertes in das Null-Indikatorfeld wird der Datenbankspalte ein Nullwert zugewiesen.
	
	Eine `INDICATOR` *`host-variable`* hat Format/Länge I2.

LINDICATOR	**LINDICATOR-Klausel:**
	
	Diese Klausel ist optional und dient zur Unterstützung von Spalten des Typs VARCHAR oder LONG VARCHAR.
	
	Wenn sie mit einer empfangenden *host-variable* (Zielfeld) verwendet wird, enthält die LINDICATOR *host-variable* (Längen-Indikatorfeld) die Anzahl der tatsächlich von der Datenbank in das Zielfeld geschriebenen Zeichen. Das Zielfeld wird immer mit Leerzeichen aufgefüllt.
	
	Enthält die VARCHAR- bzw. LONG VARCHAR-Spalte mehr Zeichen als in das Zielfeld passen, wird im Längen-Indikatorfeld die Anzahl der tatsächlich gelesenen Zeichen ausgegeben und im Null-Indikatorfeld (falls angegeben) die tatsächliche Gesamtlänge der Spalte.
	
	Beispiel:
	
	`DEFINE DATA LOCAL`
	`1 ADDRESSLIND (I2)`
	`1 ADDRESS (A50/1:6)`
	`END-DEFINE`
	`SELECT *`
	`INTO :ADDRESS(*) LINDICATOR :ADDRESSLIND`
	`...`
	
	:ADDRESS(*) erhält die ersten 300 Bytes (falls vorhanden) der betreffenden VARCHAR- bzw. LONG VARCHAR-Spalte, und :ADDRESSLIND ist das Längen-Indikatorfeld, das die Anzahl der tatsächlich von der Datenbank gelesenen Zeichen enthält.
	
	Wenn sie mit einer sendenden *host-variable* (Ausgangsfeld) verwendet wird, gibt das Längen-Indikatorfeld an, wieviele Zeichen des Ausgangsfeldes an die Datenbank übergeben werden sollen.
	
	Beispiel:
	
	`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`
	`...`
	
	Eine LINDICATOR *host-variable* hat Format/Länge I2 oder I4. Um Verarbeitungszeit zu sparen, sollte sie unmittelbar vor dem betreffenden Ausgangs- bzw. Zielfeld angegeben werden; andernfalls würde sie zur Laufzeit in einen Zwischenspeicher kopiert.
	
	Wenn das LINDICATOR-Feld als I2-Feld definiert ist, wird der SQL-Datentyp VARCHAR zum Senden/Erhalten der betreffenden Spalte verwendet. Wird die LINDICATOR *host-variable* als I4 angegeben, wird ein großer Objektdatentyp (CLOB/BLOB) verwendet.
	
	Wenn das Feld als DYNAMIC (dynamisch) definiert wird, wird die Spalte in einer internen Schleife bis zu ihrer wirklichen Länge gelesen. Das LINDICATOR-Feld und *LENGTH werden auf diese Länge gesetzt. Bei Feldern fester Länge wird die Spalte bis zur definierten Länge gelesen. In beiden Fällen wird das Feld bis zum im LINDICATOR-Feld definierten Wert geschrieben.
	
	Ein Feld fester Länge soll zum Beispiel mit einem als I2 angegebenen LINDICATOR-Feld definiert werden. Wenn die VARCHAR-Spalte mehr Zeichen enthält als in dieses Feld fester Länge passen, wird das Längenindikatorfeld auf die tatsächlich zurückgegebene Länge gesetzt, und das Nullindikatorfeld (falls angegeben) wird auf die Gesamtlänge dieser Spalte (Lesen) gesetzt. Dies ist bei Feldern fester Länge >= 32 KB nicht möglich (die Länge ist größer gewählt als die Länge des Nullindikatorfeldes).

Natural-Formate und SQL-Datentypen

Das Natural-Format einer *host-variable* wird entsprechend der folgenden Tabelle in einen SQL-Datentyp umgesetzt:

Natural-Format/Länge	**SQL-Datentyp**
A*n*	CHAR (*n*)
B2	SMALLINT
B4	INT
B*n*; *n* ungleich 2 oder 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*; nur für Views	GRAPHIC (*n*)

Natural überprüft nicht, ob der SQL-Datentyp mit der Datenbankspalte kompatibel ist. Außer bei Feldern mit Format N wird keine Datenkonvertierung vorgenommen.

Bei Natural SQL gibt es zu den Standard-Natural-Formaten noch folgende Erweiterungen:

- Um alphanumerische Spalten zu unterstützen, die länger als 253 Bytes sind, kann ein eindimensionales Array vom Format A verwendet werden. Der Index dieses Arrays muss mit 1 anfangen und kann nur mit (*) referenziert werden. Der entsprechende SQL-Datentyp ist `CHAR (n)`, wobei *n* die Gesamtanzahl der Bytes des Arrays ist.

- Um Spalten mit variabler Länge zu unterstützen, kann eine *host-variable* mit Schlüsselwort LINDICATOR verwendet werden. Der entsprechende SQL-Datentyp ist `VARCHAR (n)`; vgl. `LINDICATOR`-Klausel.

- Die Natural-Formate Datum (D) und Zeit (T) können bei Natural for DB2 verwendet werden und werden in die entsprechenden datenbank-spezifischen Formate DB2 DATE und TIME umgesetzt (Näheres siehe *Natural for DB2*-Dokumentation).

Ein sendendes Feld, das als eindimensionales Array ohne `LINDICATOR`-Feld angegeben wird, wird in den SQL-Datentyp `VARCHAR` umgesetzt. Seine Länge ist die Gesamtanzahl der Bytes des Arrays ohne Berücksichtigung nachgestellter Leerzeichen.