

SELECT - SQL

Dieses Kapitel behandelt folgende Themen:

- Funktion
- Syntax-Beschreibung
- Join-Abfragen
- SELECT – Cursor-orientierte Auswahl

Funktion

Gemäß der Standard-SQL-Funktionalität unterstützt Natural sowohl das cursor-orientierte SELECT, mit dem eine beliebige Anzahl von Reihen gelesen werden kann, als auch das nicht cursor-orientierte Singleton SELECT, das maximal eine Reihe liest.

Mit dem Konstrukt `SELECT . . . END-SELECT` verwendet Natural die gleiche Datenbankschleifen-Verarbeitung wie beim `FIND`-Statement.

Syntax-Beschreibung

Zwei verschiedene Strukturen sind möglich:

Syntax 1 – Cursor-orientierte Auswahl

Common Set-Syntax

<code>SELECT selection INTO</code>	{ <i>parameter</i> , ...
	{ <code>VIEW {view-name [correlation-name]}</code> ,
	... }
[{ <code>UNION</code>	[<code>[ALL] [(SELECT selection table-expression())</code>]
{ <code>EXCEPT</code>	}
[<code>INTERSECT</code>]]
<code>ORDER BY</code>	{ <i>integer</i>
	{ <i>column-reference</i>
	<i>expression</i> } [<code>ASC</code>
	<code>DESC</code>]
<i>statement ...</i>	
{ <code>END-SELECT (structured mode</code>	}
<i>only)</i>	
<code>LOOP (reporting mode only)</code>	

Extended Set-Syntax:

[WITH_CTE <i>common-table-expression</i> ,...]				
SELECT <i>selection</i> INTO	{	<i>parameter</i> ,...		
		VIEW { <i>view-name</i> [<i>correlation-name</i>]},...		
[[UNION			[ALL] [(SELECT <i>selection table-expression</i>)]]
[EXCEPT				
[INTERSECT				
ORDER BY	{	<i>integer</i>	[[ASC]
		<i>column-reference</i>	DESC	
		<i>expression</i>		
		INPUT SEQUENCE		
[OPTIMIZE FOR <i>integer</i> ROWS]				
[WITH	{	CS		
		RR		
		UR		
		RS		
		RS KEEP UPDATE LOCKS		
		RR KEEP UPDATE LOCKS		
QUERYNO <i>integer</i>				
[FETCH FIRST	[{ 1		{ ROW	ONLY
	{ <i>integer</i>		ROWS	
[WITH HOLD]				
[WITH RETURN]				
[WITH	{	ASENSITIVE SCROLL	[: <i>scroll_hv</i> [GIVING [:] <i>sqlcode</i>]
		INSENSITIVE SCROLL		
		SENSITIVE STATIC SCROLL		
		SENSITIVE DYNAMIC SCROLL		
[WITH ROWSET POSITIONING	{	[:] <i>row_hv</i>	ROWS	ROWS_RETURNED
FOR	{	<i>integer</i>		[:] <i>ret_row</i>
[IF-NO-RECORDS-FOUND-clause]				
<i>statement</i> ...				
{		END-SELECT (<i>structured mode only</i>)		
{		LOOP (<i>reporting mode only</i>)		

Syntax-Elementbeschreibung – Syntax 1:

SELECT <i>selection</i>	<p>Das cursor-orientierte SELECT-Statement dient (wie das FIND-Statement) dazu, ausgehend von einem Suchkriterium Reihen (Rows) von einer oder mehreren Datenbanktabellen auszuwählen. Darüber hinaus wird die Cursor-Verwaltung von Natural automatisch erledigt und muss daher nicht mehr im Anwendungsprogramm kodiert werden.</p> <p>Weitere Informationen siehe <i>SELECT-Cursor-Oriented</i> weiter unten.</p>
INTO	<p>In der INTO-Klausel geben Sie die Zielfelder im Programm an, die mit dem Ergebnis der Abfrage gefüllt werden sollen.</p> <p>Weitere Informationen und Beispiele siehe <i>INTO-Klausel</i> weiter unten.</p>
VIEW	<p>Wenn in der INTO-Klausel ein oder mehrere Views referenziert werden, muss die Anzahl der in der <i>selection</i> gemachten Angaben der Anzahl der in dem/den View(s) definierten Felder entsprechen (hierbei werden Gruppenfelder, redefinierte Felder und Indikatorfelder nicht mitgezählt).</p> <p>Weitere Informationen und Beispiele siehe <i>VIEW-Klausel</i> weiter unten.</p>
<i>table-expression</i>	<p>Die <i>table-expression</i> besteht aus einer FROM-Klausel und außerdem einer optionalen WHERE-Klausel.</p> <p>Weitere Informationen und Beispiele siehe <i>table-expression</i> weiter unten.</p>
UNION	<p>UNION vereinigt die Ergebnisse von zwei oder mehr <i>select-expressions</i> miteinander. Weitere Informationen und ein Beispiel siehe <i>Abfrage mit UNION</i> weiter unten.</p>
ORDER BY	<p>Die ORDER BY-Klausel sortiert das Ergebnis der Abfrage in einer bestimmten Reihenfolge.</p> <p>Weitere Informationen und Beispiele siehe <i>ORDER BY-Klausel</i> weiter unten.</p>
IF NO RECORDS FOUND	<p>Mit der IF NO RECORDS FOUND-Klausel können Sie eine Schleifen-Verarbeitung angeben, die ausgeführt werden soll für den Fall, dass kein Datensatz die im vorangegangenen SELECT-Statement angegebenen Selektionskriterien erfüllt.</p> <p>Weitere Informationen siehe <i>IF NO RECORDS FOUND-Klausel</i> weiter unten.</p>
END-SELECT	<p>Das für Natural reservierte Schlüsselwort END-SELECT muss zum Beenden des SELECT-Statements benutzt werden.</p>

Die folgenden Sytax-Elemente gehören zum SQL Extended Set:

WITH_CTE <i>common-table-expression,...</i>	WITH_CTE <i>common-table-expression</i>: Mit dieser Option können Sie eine Ergebnistabelle definieren, die in einer FROM-Klausel eines nachfolgenden SELECT-Statements referenziert werden kann. Nach dem Schlüsselwort WITH_CTE können mehrere <i>common-table-expressions</i> angegeben werden. Jeder dieser Ausdrücke kann in der FROM-Klausel einer nachfolgenden <i>common-table-expression</i> referenziert werden. Weitere Informationen siehe <i>SELECT- Cursor-Oriented</i> .
OPTIMIZE FOR	OPTIMIZE FOR-Klausel: Diese Klausel gilt nur für DB2-Datenbanken. Bei anderen Datenbanken verursacht sie einen Laufzeitfehler.
WITH CS/RS/UR/...	WITH CS/RS/UR/...-Klausel: Diese Klausel gilt nur für DB2-Datenbanken. Bei anderen Datenbanken verursacht sie einen Laufzeitfehler.
QUERYNO	QUERYNO-Klausel: Diese Klausel wird zurzeit nicht unterstützt und wird ignoriert.
FETCH FIRST	FETCH FIRST-Klausel: Diese Klausel gilt nur für DB2-Datenbanken. Bei anderen Datenbanken verursacht sie einen Laufzeitfehler.
WITH HOLD	WITH HOLD-Klausel: Diese Klausel wird zurzeit nicht unterstützt. Falls sie verwendet wird, erzeugt sie einen Compiler-Fehler.
WITH RETURN	WITH RETURN-Klausel: Diese Klausel wird zurzeit nicht unterstützt. Falls sie verwendet wird, erzeugt sie einen Compiler-Fehler.

<p>WITH ... SCROLL</p>	<p>WITH ... SCROLL-Klausel:</p> <p>Beliebig positionierbare RDBMS-Cursor werden mit dieser Klausel aktiviert. Beliebig positionierbare Cursor können entweder <code>ASENSITIVE</code>, <code>INSENSITIVE</code>, <code>SENSITIVE STATIC</code> oder <code>SENSITIVE DYNAMIC</code> sein.</p> <p>Anmerkung: Nicht alle SQL-Datenbanksysteme unterstützen alle Optionen.</p> <ul style="list-style-type: none"> ● Mit <code>WITH ASENSITIVE SCROLL</code> wird angegeben, dass der Cursor entweder <code>INSENSITIVE</code> oder <code>SENSITIVE DYNAMIC</code> ist. Dies wird von der Datenbank zum Zeitpunkt der Öffnung des Cursors ermittelt, und zwar in Abhängigkeit von der Fähigkeit des Cursors, nur Lesezugriff zu gewährleisten. Wenn über den Cursor nur gelesen werden kann, wird der Cursor <code>INSENSITIVE</code>. Wenn der Cursor Lese- und Schreibzugriff garantiert, wird er <code>SENSITIVE DYNAMIC</code>. Dies wird nicht bei DB2-Datenbanken unterstützt. ● Mit <code>WITH INSENSITIVE SCROLL</code> wird angegeben, dass der Cursor insensitiv ist für Änderungen, Löschungen und auch für Einfügungen, die auf der Datenbanktabelle ausgeführt werden, nachdem er geändert worden ist. <code>Positioned Updates</code> und <code>Deletes</code> sind nicht zulässig beim <code>INSENSITIVE SCROLL</code>-Cursor. Dies wird unterstützt bei Oracle-, Adabas D-, Informix-, MS SQL Server ODBC- und DB2-Datenbanken. ● Mit <code>WITH SENSITIVE STATIC</code> wird angegeben, dass der Cursor sensitiv ist für Änderungen und Löschungen, die gegen die Datenbanktabelle ausgeführt werden, aber nicht für Einfügungen, nachdem der Cursor geöffnet worden ist. <code>Positioned Updates</code> und <code>Deletes</code> sind für <code>SENSITIVE STATIC SCROLL</code>-Cursor zulässig. Dies wird unterstützt bei Adabas D-, MS SQL Server ODBC- und DB2-Datenbanken. ● Mit <code>WITH SENSITIVE DYNAMIC</code> wird angegeben, dass der Cursor sensitiv ist für Aktualisierungen, Löschungen und Einfügungen gegen die Basistabelle, nachdem der Cursor geöffnet wurde. <code>Positioned Updates</code> und <code>Deletes</code> sind für <code>SENSITIVE DYNAMIC SCROLL</code>-Cursor zulässig. Dies wird unterstützt bei Adabas D-, MS SQL Server ODBC- und DB2-Datenbanken. <p>Beliebig positionierbare Cursor ermöglichen es der Anwendung, eine beliebige Reihe im Result Set jederzeit zu positionieren, solange der Cursor offen ist.</p> <p>Beliebig positionierbare Cursor werden bei Sybase-Datenbanken überhaupt nicht unterstützt. Beliebig positionierbare Cursor werden beim MS SQL Server DBLIB Interface nicht unterstützt, sondern nur beim MS SQL Server ODBC Interface.</p> <p>Die Positionierung wird durchgeführt in Abhängigkeit vom Inhalt von <code>scroll_hv</code>. Der Inhalt wird jedesmal ausgewertet, wenn ein <code>FETCH</code> auf der Datenbank ausgeführt wird.</p> <p>Weitere Informationen siehe <i>SELECT – Cursororientierte Auswahl</i>.</p>
-------------------------------	---

Syntax 2 – Nicht cursor-orientierte Auswahl

Common Set-Syntax

```

SELECT SINGLE

    selection INTO      {
                        parameter ,...
                        VIEW {view-name [correlation-name ]}, ... } table-expression

    [IF-NO-RECORDS-FOUND-clause]
    statement...

    {
      END-SELECT (structured mode only)
      LOOP (reporting mode only)
    }

```

Extended Set-Syntax

```

SELECT SINGLE

    selection INTO      {
                        parameter ,...
                        VIEW {view-name [correlation-name ]}, ... } table-expression

    [
      WITH      {
                CS
                RR
                UR
              }
    ]

    [IF-NO-RECORDS-FOUND-clause]
    statement...

    {
      END-SELECT (structured mode only)
      LOOP (reporting mode only)
    }

```

Syntax-Elementbeschreibung – Syntax 2:

SELECT SINGLE	SELECT SINGLE unterstützt die Funktionalität eines keine Cursor verwendenden Singleton SELECT, das maximal eine Reihe liest. Es kann nicht von einem Positioned UPDATE- bzw. DELETE-Statement referenziert werden.
INTO	In der INTO-Klausel geben Sie die Zielfelder im Programm an, die mit dem Ergebnis der Abfrage gefüllt werden sollen. Weitere Informationen und Beispiele siehe <i>INTO-Klausel</i> weiter unten.
VIEW	Wenn in der INTO-Klausel ein oder mehrere Views referenziert werden, muss die Anzahl der in der <i>selection</i> gemachten Angaben der Anzahl der in dem/den View(s) definierten Felder entsprechen (hierbei werden Gruppenfelder, redefinierte Felder und Indikatorfelder nicht mitgezählt). Weitere Informationen und Beispiele siehe <i>VIEW-Klausel</i> weiter unten.
<i>table-expression</i>	Die <i>table-expression</i> besteht aus einer FROM-Klausel und einer optionalen WHERE-Klausel. Weitere Informationen und Beispiele siehe Abschnitt <i>table-expression</i> weiter unten.
WITH CS/RR/UR	WITH CS/RR/UR-Klausel: Diese Klausel gehört zum SQL Extended Set. Diese Klausel gilt nur für DB2-Datenbanken. Bei anderen Datenbanken verursacht sie einen Laufzeitfehler.
IF NO RECORDS FOUND	In der IF NO RECORDS FOUND-Klausel können Sie eine Verarbeitung angeben, die ausgeführt werden soll für den Fall, dass kein Datensatz die im vorangegangenen SELECT-Statement angegebenen Selektionskriterien erfüllt. Weitere Informationen, siehe <i>IF NO RECORDS FOUND-Klausel</i> weiter unten.
END-SELECT	Das für Natural reservierte Schlüsselwort END-SELECT muss zum Beenden des SELECT-Statements benutzt werden.

INTO-Klausel

<pre> INTO { parameter ,... { VIEW {view-name [correlation-name]},... } </pre>

In der INTO-Klausel geben Sie die Zielfelder im Programm an, die mit dem Ergebnis der Abfrage gefüllt werden sollen. Sie können in der INTO-Klausel entweder einzelne *parameters* oder ganze im DEFINE DATA-Statement definierte Views angeben.

Die Zielfelder können aus einer einzigen Tabelle kommen bzw. bei einer Join-Operation (vgl. Abschnitt *Join-Abfragen*) auch aus mehreren.

Anmerkung:

In der Standard-SQL-Syntax wird die INTO-Klausel nur in nicht cursor-orientierten Singleton SELECT-Operationen verwendet, bei denen eine einzelne Reihe gelesen werden soll. Natural erlaubt die INTO-Klausel jedoch sowohl in cursor-orientierten als auch in nicht cursor-orientierten SELECT-Operationen.

Die *selection* kann auch aus nur einem Stern (*) bestehen. In einer standardmäßigen *selection-expression* steht dieser für eine Liste aller Spaltennamen der in der FROM-Klausel angegebenen Tabelle(n). Im Natural-SELECT-Statement hat der Ausdruck `SELECT *` jedoch eine andere Bedeutung: Alle in der INTO-Klausel gemachten Angaben werden auch in der *selection* verwendet, ohne dass sie dort explizit angegeben werden müssen. Ihre Namen müssen vorhandenen Datenbank-Spaltennamen entsprechen.

Beispiele:

Beispiel 1:

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
   02 NAME
   02 AGE
END-DEFINE
...
SELECT *
   INTO NAME, AGE
```

Beispiel 2:

```
...
SELECT *
   INTO VIEW PERS
```

Obige Beispiele sind mit den folgenden gleichwertig:

Beispiel 3:

```
...
SELECT NAME, AGE
   INTO NAME, AGE
```

Beispiel 4:

```
...
SELECT NAME, AGE
   INTO VIEW PERS
```

VIEW-Klausel:

VIEW {*view-name* [*correlation-name*]}, ...

Wenn in der INTO-Klausel ein oder mehrere Views referenziert werden, muss die Anzahl der in der *selection* gemachten Angaben der Anzahl der in dem/den View(s) definierten Felder entsprechen (hierbei werden Gruppenfelder, redefinierte Felder und Indikatorfelder nicht mitgezählt).

Anmerkung:

Die Natural-Zielfelder wie auch die Tabellenspalten müssen im Natural-DDM definiert sein (wobei die Namen allerdings unterschiedlich sein dürfen, da die Zuweisung entsprechend ihrer Reihenfolge erfolgt).

Beispiel einer INTO-Klausel mit View:


```

DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 NAME
  02 AGE
END-DEFINE
...
SELECT FIRSTNAME, AGE
  INTO VIEW PERS
  FROM SQL-PERSONNEL
...

```

Die Zielfelder NAME und AGE, die Teil eines Natural-Views sind, erhalten den Inhalt der Datenbankspalten FIRSTNAME und AGE.

<i>parameter</i>	<p>Wenn Sie einzelne <i>parameter</i> als Zielfelder angeben, müssen sie in Anzahl und Format mit den in der entsprechenden <i>selection</i> angegebenen <i>columns</i> bzw. <i>scalar-expressions</i> übereinstimmen, wie oben beschrieben. Siehe <i>scalar-expressions</i>.</p> <p>Beispiel:</p> <pre> DEFINE DATA LOCAL 01 #NAME (A20) 01 #AGE (I2) END-DEFINE ... SELECT NAME, AGE INTO #NAME, #AGE FROM SQL-PERSONNEL ... </pre> <p>Die Zielfelder #NAME und #AGE, die Natural-Programmvariablen sind, erhalten den Inhalt der Datenbankspalten NAME und AGE.</p>
<i>correlation-name</i>	<p>Wenn die VIEW-Klausel in einem SELECT * verwendet wird, in dem mehrere Tabellen mit JOIN verknüpft werden, sind <i>correlation-names</i> erforderlich, falls der angegebene View Felder enthält, die Spalten referenzieren, welche in mehreren dieser Tabellen vorkommen. Um zu bestimmen, von welcher Spalte ausgewählt werden soll, werden bei der Generierung der Auswahlliste alle diese Spalten mit dem angegebenen <i>correlation-name</i> qualifiziert. Der einem View zugewiesene <i>correlation-name</i> muss einem der <i>correlation-names</i> entsprechen, mit denen die verknüpften Tabellen qualifiziert werden. Siehe auch <i>Join-Abfragen</i>.</p> <p>Beispiel:</p> <pre> DEFINE DATA LOCAL 01 PERS VIEW OF SQL-PERSONNEL 02 NAME 02 FIRST-NAME 02 AGE END-DEFINE ... SELECT * INTO VIEW PERS A FROM SQL-PERSONNEL A, SQL-PERSONNEL B ... </pre>

table-expression

Die *table-expression* besteht aus einer FROM-Klausel und einer optionalen WHERE-Klausel. Die GROUP BY- und HAVING-Klauseln sind nicht erlaubt.

Beispiel 1:

```
DEFINE DATA LOCAL
01 #NAME      (A20)
01 #FIRSTNAME (A15)
01 #AGE       (I2)
...
END-DEFINE
...
SELECT NAME, FIRSTNAME, AGE
      INTO #NAME, #FIRSTNAME, #AGE
      FROM SQL-PERSONNEL
      WHERE NAME IS NOT NULL
            AND AGE > 20
...
      DISPLAY #NAME #FIRSTNAME #AGE
END-SELECT
...
END
```

Beispiel 2:

```
DEFINE DATA LOCAL
01 #COUNT   (I4)
...
END-DEFINE
...
SELECT SINGLE COUNT(*) INTO #COUNT FROM SQL-PERSONNEL
...
```

Weitere Informationen siehe *selection* und *table-expression*.

Abfrage mit UNION

Anmerkung:

Im Folgenden wird der Begriff SELECT-Statement verwendet im Sinne einer vollständigen *query-expression*, die aus mehreren mit UNION verknüpften *select-expressions* besteht.

UNION vereinigt die Ergebnisse von zwei oder mehr *select-expressions* miteinander. Die in den einzelnen *select-expressions* angegebenen Spalten müssen UNION-kompatibel sein, d.h. in Anzahl, Typ und Format zueinander passen.

Redundante doppelte Reihen werden immer aus dem Ergebnis einer UNION eliminiert, es sei denn, der UNION-Operator enthält ausdrücklich ein ALL. Allerdings ist es bei UNION nicht möglich, DISTINCT explizit als Alternative zu ALL anzugeben.

Beispiel:

```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 NAME
  02 AGE
  02 ADDRESS (1:6)
```

```

END-DEFINE
...
SELECT NAME, AGE, ADDRESS
  INTO VIEW PERS
  FROM SQL-PERSONNEL
  WHERE AGE > 55
UNION ALL
SELECT NAME, AGE, ADDRESS
  FROM SQL-EMPLOYEES
  WHERE PERSNR < 100
ORDER BY NAME
...
END-SELECT
...

```

Grundsätzlich ist die Anzahl der *select-expressions*, die mit UNION verknüpft werden können, beliebig.

Nur die erste *select-expression* darf eine INTO-Klausel enthalten.

Wird eine ORDER BY-Klausel verwendet, muss sie nach der letzten *selection-expression* angegeben werden. Die zu sortierenden Spalten müssen durch die Spaltennummern identifiziert werden, nicht durch die Spaltennamen.

ORDER BY-Klausel

$\text{ORDER BY } \left\{ \left\{ \begin{array}{c} \textit{integer} \\ \textit{column-reference} \end{array} \right\} \left[\begin{array}{c} \text{ASC} \\ \text{DESC} \end{array} \right] \right\} , \dots$

Die ORDER BY-Klausel sortiert das Ergebnis der Abfrage in einer bestimmten Reihenfolge.

Jede ORDER BY-Klausel muss eine Spalte der Ergebnistabelle spezifizieren. In den meisten ORDER BY-Klauseln wird die Spalte entweder durch eine *column-reference* (also den optional qualifizierten Spaltennamen) oder durch die Spaltennummer identifiziert.

In einer Abfrage mit UNION muss eine Spalte durch die Spaltennummer identifiziert werden. Die Spaltennummer ist die Ordinalzahl, die die Position einer Spalte (von links nach rechts) innerhalb der *selection* angibt, also eine Ganzzahl (*Integer*). Dadurch ist es möglich, ein Ergebnis auf der Grundlage einer berechneten Spalte, die keinen Namen hat, zu sortieren.

<i>expression</i>	Gibt einen Ausdruck mit Operatoren an (d.h. nicht nur einen <i>column-name</i> oder <i>integer</i>).
INPUT SEQUENCE	Gibt an, dass die Ergebnistabelle die Eingabe-Reihenfolge der Reihen reflektiert, die in der VALUES-Klausel eines INSERT-Statements angegeben sind. Eine Sortierung der INPUT SEQUENCE kann nur angegeben werden, wenn ein INSERT-Statement in einer FROM-Klausel vorhanden ist.

Beispiel:

```

DEFINE DATA LOCAL
1 #NAME          (A20)
1 #YEARS-TO-WORK (I2)
END-DEFINE
...
SELECT NAME , 65 - AGE
      INTO #NAME, #YEARS-TO-WORK
      FROM SQL-PERSONNEL
      ORDER BY 2
      ...

```

Als Sortierreihenfolge können Sie entweder aufsteigend (ASC = Ascending) oder absteigend (DESC = Descending) angeben. Standardmäßig gilt ASC.

Beispiel:

```

DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
1 NAME
1 AGE
1 ADDRESS (1:6)
END-DEFINE
...
SELECT NAME, AGE, ADDRESS
      INTO VIEW PERS
      FROM SQL-PERSONNEL
      WHERE AGE = 55
      ORDER BY NAME DESC
      ...

```

Weitere Informationen siehe *integer*-Werte und *column-reference*.

IF NO RECORDS FOUND-Klausel:

Anmerkung:

Diese Klausel ist eigentlich nicht Bestandteil von Natural SQL; sie stellt eine Natural-Funktion dar, die für SQL-Schleifenverarbeitung zur Verfügung gestellt wird.

Structured Mode-Syntax

```

IF NO [RECORDS] [FOUND]
{
  ENTER
  statement...
}
END- NOREC

```

Reporting Mode-Syntax

```

IF NO [RECORDS] [FOUND]
{
  ENTER
  statement
  DO statement... DOEND
}

```

In der `IF NO RECORDS FOUND`-Klausel können Sie eine Schleifenverarbeitung angeben, die ausgeführt werden soll für den Fall, dass kein Datensatz die im vorangegangenen `SELECT`-Statement angegebenen Selektionskriterien erfüllt.

Wenn kein Datensatz die angegebenen Selektionskriterien erfüllt, dann löst die `IF NO RECORDS FOUND`-Klausel eine Verarbeitungsschleife aus, die einmal mit einem leeren Datensatz durchlaufen wird. Falls Sie dies nicht wünschen, geben Sie in der `IF NO RECORDS FOUND`-Klausel das Statement `ESCAPE BOTTOM` an.

Enthält die `IF NO RECORDS FOUND`-Klausel ein oder mehrere Statements, werden diese ausgeführt, unmittelbar bevor die Schleife durchlaufen wird. Sollen vor Durchlaufen der Schleife keine weiteren Statements ausgeführt werden, muss die `IF NO RECORDS FOUND`-Klausel das Schlüsselwort `ENTER` enthalten.

Anmerkung:

Falls das Ergebnis-Set des `SELECT`-Statements aus einer einzelnen Reihe von `NULL`-Werten besteht, wird die `IF NO RECORDS FOUND`-Klausel nicht ausgeführt. Dies kann der Fall sein, wenn die *selection*-Liste nur aus einer der *aggregate-functions* `SUM`, `AVG`, `MIN` oder `MAX` auf Spalten besteht, und der Set, mit dem diese *aggregate-functions* operieren, leer ist. Bei obengenannter Verwendung dieser *aggregate-functions* sollten Sie daher keine `IF NO RECORDS FOUND`-Klausel benutzen, sondern stattdessen die Werte der betreffenden Null-Indikator-Felder abfragen.

Datenbankwerte

Natural setzt alle Datenbankfelder, die die in der aktuellen Verarbeitungsschleife angegebene Datei referenzieren, auf Leerwerte, es sei denn, eines der in der `IF NO RECORDS FOUND`-Klausel angegebenen Statements weist den Feldern andere Werte zu.

Auswertung von Systemfunktionen

Natural-Systemfunktionen werden einmal für den leeren Datensatz ausgewertet, der für die aus der `IF NO RECORDS FOUND`-Klausel resultierende Verarbeitung erstellt wurde.

Join-Abfragen

Ein Join ist eine Abfrage, bei der Daten von mehr als einer Tabelle gelesen werden. Alle betroffenen Tabellen müssen in der `FROM`-Klausel angegeben werden.

Beispiel:

```
DEFINE DATA LOCAL
1 #NAME      (A20)
1 #MONEY     (I4)
END-DEFINE
...
SELECT NAME, ACCOUNT
  INTO #NAME, #MONEY
  FROM  SQL-PERSONNEL P, SQL-FINANCE F
  WHERE P.PERSNR = F.PERSNR
        AND F.ACCOUNT > 10000
  ...
```

Ein Join bildet immer zunächst das kartesische Produkt der in der FROM-Klausel angegebenen Tabellen und eliminiert später von diesem kartesischen Produkt alle Reihen, die die in der WHERE-Klausel angegebene Join-Bedingung nicht erfüllen.

Bei längeren Tabellennamen können Sie sich durch Verwendung von *Correlation-names* Schreibarbeit sparen. Wird in einer Join-Abfrage eine Tabelle mit sich selbst verknüpft, ist die Angabe von *correlation-names* erforderlich, um die beiden nötigen Referenzen auf dieselbe Tabelle voneinander zu unterscheiden

SELECT – Cursor-orientierte Auswahl

Wie das Natural FIND-Statement wird das cursor-orientierte SELECT-Statement benutzt, um mittels einer Suchbedingung eine Untermenge von Reihen (Datensätzen) von einer oder mehreren DB2-Tabelle/n auszuwählen. Da eine Datenbankschleife initiiert wird, muss die Schleife durch ein LOOP- (Reporting Mode) oder END-SELECT-Statement geschlossen werden. Bei dieser Statement-Struktur verwendet Natural dieselbe Schleifenverarbeitung wie beim FIND-Statement.

Außerdem ist vom Anwendungsprogramm keine Cursor-Verwaltung erforderlich; sie wird automatisch von Natural durchgeführt.

Im Folgenden finden Sie Informationen zu:

- WITH_CTE common-table-expression,...
- OPTIMIZE FOR integer ROWS
- WITH - Isolation Level
- FETCH FIRST
- WITH INSENSITIVE/SENSITIVE

WITH_CTE *common-table-expression*,...

Mit dieser Klausel können Sie Ergebnistabellen definieren, die in jeder FROM-Klausel des nachfolgenden SELECT-Statements referenziert werden können.

Das Natural-spezifische Schlüsselwort WITH_CTE entspricht dem SQL-Schlüsselwort WITH. WITH_CTE wird durch den Natural-Compiler in das SQL-Schlüsselwort WITH umgesetzt.

Jeder *common-table-expression* muss folgender Syntax entsprechen:

```
[common-table-expression-name [(column-name,...)] AS (fullselect) ]
```

Syntax-Beschreibung:

<i>common-table-expression-name</i>	Muss ein nicht qualifizierter SQL-Identifizier sein und muss sich von allen anderen, im selben Statement angegebenen <i>common-table-expression-names</i> unterscheiden. Jeder <i>common-table-expression-name</i> kann in der FROM-Klausel eines darauffolgenden <i>common-table-expression-name</i> oder in der FROM-Klausel des folgenden SELECT-Statements angegeben werden.
<i>column-name</i>	Muss ein nicht qualifizierter SQL-Identifizier sein und muss innerhalb eines <i>common-table-expression-name</i> eindeutig sein.
AS (<i>fullselect</i>)	Die Anzahl der <i>column-names</i> muss gleich der Anzahl der Spalten des <i>fullselect</i> sein.

Ein *common-table-expression* kann verwendet werden

- anstelle einer View, wenn man das Erstellen einer View vermeiden möchte;
- wenn dieselbe Ergebnistabelle in einem *fullselect* gemeinsam genutzt werden muss ;
- wenn das Ergebnis durch Rekursion abgeleitet werden muss.

Rekursive Abfragen sind in Anwendungen wie zum Beispiel Stücklisten von Nutzen.

Beispiel:

```
WITH_CTE
RPL (PART,SUBPART,QUANTITY) AS
( SELECT ROOT.PART,ROOT.SUBPART,ROOT.QUANTITY
  FROM HGK-PARTLIST ROOT
  WHERE ROOT.PART = '01'
  UNION ALL
  SELECT CHILD.PART,CHILD.SUBPART,CHILD.QUANTITY
    FROM RPL PARENT, HGK-PARTLIST CHILD
    WHERE PARENT.SUBPART = CHILD.PART
  )
SELECT DISTINCT PART,SUBPART,QUANTITY
  INTO VIEW V1
  FROM RPL
  ORDER BY PART,SUBPART,QUANTITY
END-SELECT
```

OPTIMIZE FOR *integer* ROWS

[OPTIMIZE FOR *integer* ROWS]

Die OPTIMIZE FOR *integer* ROWS-Klausel wird verwendet, um DB2 im Voraus über die Anzahl (Ganzzahl) von Reihen zu informieren, die von der Ergebnistabelle eingelesen werden sollen. Ohne diese Klausel geht DB2 davon aus, dass alle Reihen der Ergebnistabelle eingelesen werden sollen und führt dementsprechend eine Optimierung durch.

Diese optionale Klausel ist nützlich, wenn Sie wissen, wie viele Reihen wahrscheinlich ausgewählt werden, weil eine Optimierung von Integer-Reihen die Verarbeitungszeit verbessern kann, wenn die Anzahl der tatsächlich ausgewählten Reihen nicht den Ganzzahlwert überschreitet (der im Bereich von 0 bis 2147483647 liegen kann).

Beispiel:

```
SELECT name INTO
#name FROM table WHERE AGE = 2 OPTIMIZE FOR 100 ROWS
```

WITH - Isolation Level

WITH	{	CS	}
		RR	}
		RR KEEP UPDATE LOCK	}
		RS	}
		RS KEEP UPDATE LOCKS	}
		UR	}

Diese WITH-Klausel ermöglicht es Ihnen, einen expliziten Isolation Level anzugeben, mit dem das Statement ausgeführt werden soll. Die folgenden Optionen stehen zur Verfügung:

Option	Bedeutung
CS	Cursor-Stabilität
RR	Wiederholbarer Lesevorgang
RS	Lese-Stabilität
RS KEEP UPDATE LOCKS	Nur gültig, wenn eine FOR UPDATE OF-Klausel angegeben wird. Lese-Stabilität und Beibehaltung von Aktualisierungssperren.
RR KEEP UPDATE LOCKS	Nur gültig, wenn eine FOR UPDATE OF-Klausel angegeben wird. Wiederholbarer Lesevorgang und Beibehaltung von Aktualisierungssperren.
UR	Freier Lesevorgang

WITH UR kann nur bei einem SELECT-Statement angegeben werden, und wenn es für die Tabelle nur eine Lesezugriffsberechtigung gibt. Der standardmäßige Isolation Level wird durch die Trennung des Pakets oder Plans festgelegt, in den das Statement eingebunden ist. Der standardmäßige Isolation Level ist auch abhängig davon, ob für die Ergebnistabelle nur eine Lesezugriffsberechtigung besteht oder nicht. Um den standardmäßigen Isolation Level zu ermitteln, greifen Sie auf die IBM-Literatur zurück.

Anmerkung:

Diese Option funktioniert auch für eine Nicht-Cursor-Auswahl.

FETCH FIRST

```
[ FETCH FIRST { 1 } { ROWS } ONLY
                { integer } { ROW } ]
```

Die `FETCH FIRST`-Klausel begrenzt die Anzahl der über `FETCH` abzurufenden Reihen. Es verbessert die Verarbeitungszeit der Abfragen mit möglicherweise großen Result-Sets, wenn nur eine beschränkte Anzahl von Reihen erforderlich ist.

WITH INSENSITIVE/SENSITIVE

```
[ WITH { ASENSITIVE SCROLL
          INSENSITIVE SCROLL
          SENSITIVE STATIC SCROLL
          SENSITIVE DYNAMIC SCROLL } [:] scroll_hv [GIVING [:] sqlcode] ]
```

Natural unterstützt Scrollable Cursor von SQL über die Klauseln `WITH ASENSITIVE SCROLL`, `WITH SENSITIVE STATIC SCROLL` und `SENSITIVE DYNAMIC SCROLL`. Scrollable Cursor ermöglichen es Natural-Anwendungen, eine Reihe in einem Result Set beliebig zu positionieren. Mit Non-Scrollable Cursors können die Daten nur sequentiell vom Anfang zum Ende gelesen werden.

`ASENSITIVE` Scrollable Cursors sind entweder `INSENSITIVE` – wenn der Cursor `READ-ONLY` ist – oder `SENSITIVE DYNAMIC` – wenn der Cursor nicht `READ-ONLY` ist.

`INSENSITIVE` und `SENSITIVE STATIC` Scrollable Cursors benutzen Zwischenergebnis-Tabellen, und für sie ist deshalb eine TEMP-Datenbank in DB2 erforderlich (siehe die betreffende DB2-Literatur von IBM).

`INSENSITIVE SCROLL` bezieht sich auf einen Cursor, der bei Positioned `UPDATE`- oder Positioned `DELETE`-Operationen nicht benutzt werden kann. Außerdem reflektiert ein einmal geöffneter `INSENSITIVE SCROLL`-Cursor keine `UPDATES`, `DELETES` oder `INSERTS` gegen die Basistabelle, nachdem der Cursor geöffnet wurde.

`SENSITIVE STATIC SCROLL` bezieht sich auf einen Cursor, der für Positioned `UPDATE`- oder Positioned `DELETE`-Operationen benutzt werden kann. Außerdem reflektiert ein `SENSITIVE STATIC SCROLL`-Cursor `UPDATES`, `DELETES` der Basistabellen-Reihen. Der Cursor reflektiert keine `INSERT`-Operationen.

`SENSITIVE DYNAMIC` Scrollable Cursors reflektieren `UPDATES`, `DELETES` und `INSERTS` gegen die Basistabelle, während der Cursor geöffnet ist.

Nachfolgend finden Sie Informationen zu:

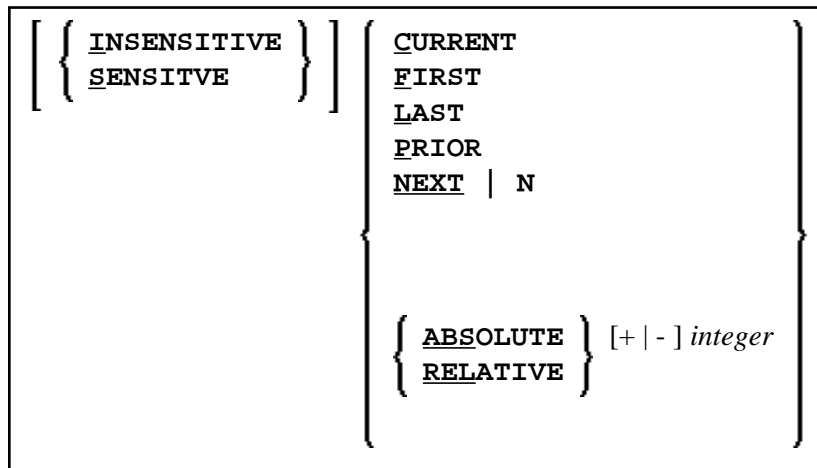
- `scroll_hv`
- `scroll_hv` - Optionen
- `GIVING [:] sqlcode`

scroll_hv

Die Variable *scroll_hv* muss alphanumerisch sein.

Die Variable *scroll_hv* gibt an, welche Reihe der Ergebnistabelle während einer Ausführung der Datenbank-Verarbeitungsschleife abgerufen wird.

Der Inhalt von *scroll_hv* wird jedesmal ausgewertet, wenn der Datenbank-Verarbeitungsschleifenzyklus ausgeführt wird.

***scroll_hv* - Optionen**

Option	Erläuterung
CURRENT	Ruft die aktuelle Reihe (erneut) ab.
FIRST	Ruft die erste Reihe ab.
LAST	Ruft die letzte Reihe ab.
NEXT	Ruft die Reihe nach der aktuellen Reihe ab. Dies ist die Voreinstellung.
PRIOR	Ruft die Reihe vor der aktuellen Reihe ab.
+/- <i>integer</i>	Gilt nur in Verbindung mit ABSOLUTE oder RELATIVE. Gibt die Position der abzurufenden Reihe ABSOLUTE oder RELATIVE an. Geben Sie ein Plus- (+) oder Minus-Zeichen (-) und dahinter eine Ganzzahl ein. Die Voreinstellung ist ein Plus-Zeichen (+).
ABSOLUTE	Gilt nur in Verbindung mit +/- <i>integer</i> . Benutzt <i>integer</i> als die absolute Position innerhalb des Result Set, von dem aus die Reihe abgerufen wird.
RELATIVE	Gilt nur in Verbindung mit +/- <i>integer</i> . Benutzt <i>integer</i> als Position <i>relativ</i> zur aktuellen Position innerhalb des Result Set, von dem aus die Reihe abgerufen wird.

Bei bestimmten RDBMS-Systemen gelten einige Einschränkungen:

- DB2 unterstützt nicht das Schlüsselwort CURRENT.
- In einer SELECT FOR UPDATE-Schleife unterstützt DB2 nur NEXT als Positionierungsoption.
- MS SQL Server (ODBC-Interface) unterstützt nicht das Schlüsselwort CURRENT.
- Adabas D unterstützt keine RELATIVE-Positionierung.

GIVING [:] *sqlcode*

Die Angabe von GIVING [:] *sqlcode* ist optional. Falls angegeben, muss die Natural-Variable [:] *sqlcode* das Format I4 haben. Die Werte für diese Variable werden von DB2 SQLCODE der zugrundeliegenden FETCH-Operation zurückgegeben. Dadurch wird es der Anwendung ermöglicht, auf verschiedene, bei geöffnetem "scrollable cursor" vorgefundene Status zu reagieren. Die wichtigsten, von SQLCODE angezeigten Status-Codes sind in der folgenden Tabelle aufgeführt

SQLCODE	Erläuterung
0	FETCH-Operation erfolgreich, Daten zurückgegeben, außer bei einem FETCH mit der Option BEFORE oder AFTER.
+100	Reihe nicht gefunden, Cursor noch geöffnet, keine Daten zurückgegeben.
-1	Allgemeiner Fehler bei Fetch-Operation auf eine Reihe.

Wenn Sie `GIVING [:] sqlcode` angeben, muss die Anwendung auf die verschiedenen Status reagieren. Wenn ein SQLCODE +100 fünfmal hintereinander ohne Terminal I/O eingegeben wird, gibt die NDB-Laufzeit den Natural-Fehler NAT3296 aus, um Anwendungsschleifen zu verhindern. Die Anwendung kann die Verarbeitungsschleife durch Ausführen eines `ESCAPE`-Statements beenden.

Wenn Sie `GIVING [:] sqlcode` nicht angeben, außer bei SQLCODE 0 und SQLCODE +100, erzeugt jeder SQLCODE den Natural-Fehler NAT3700, und die Verarbeitungsschleife wird beendet. Mit SQLCODE +100 (Reihe nicht gefunden) wird die Verarbeitungsschleife beendet.

Siehe auch das Beispielprogramm `DEM2SCR` in der Natural-Systembibliothek `SYSDB2`.