# SELECT - SQL

Dieses Kapitel behandelt folgende Themen:

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

Siehe auch die folgenden Abschnitte in der *Database Management System Interfaces*-Dokumentation:

- *SELECT SINGLE - Non-Cursor-Oriented* im Teil *Natural for DB2*.
- *SELECT* im Teil *Natural for SQL/DS*.

---

## 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.

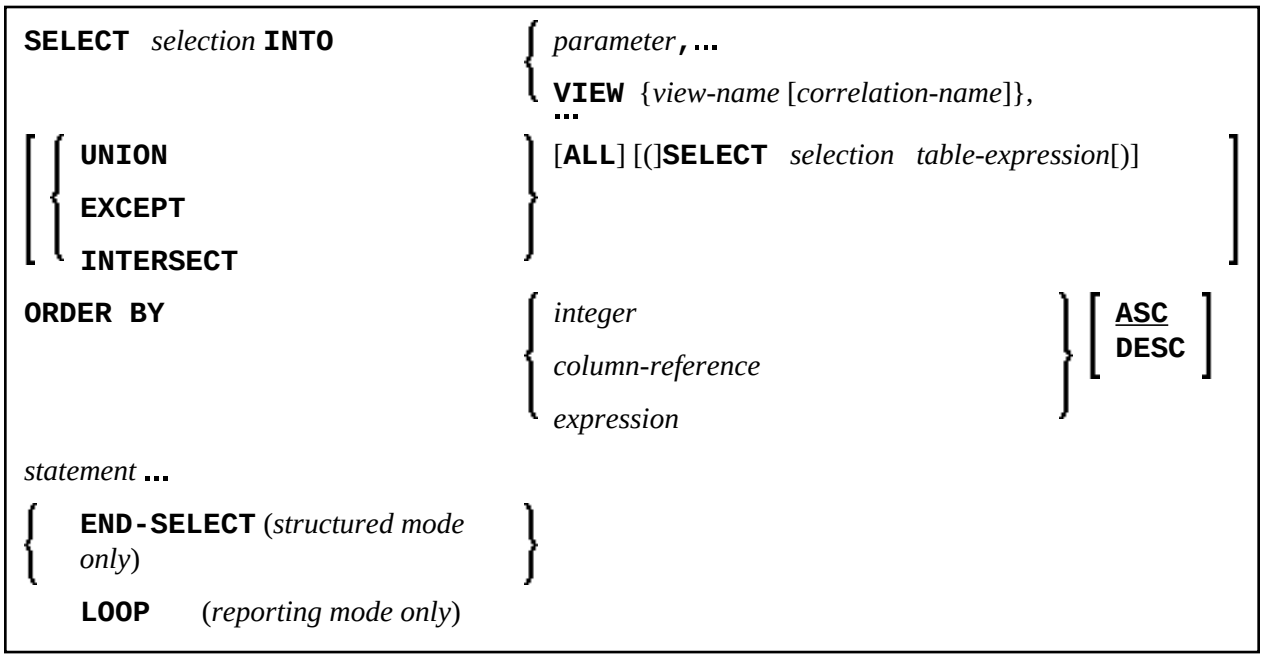
Mit dem Konstrukt `SELECT ... END–SELECT` verwendet Natural die gleiche Datenbankschleifen-Verarbeitung wie beim `FIND`-Statement.

## Syntax-Beschreibung

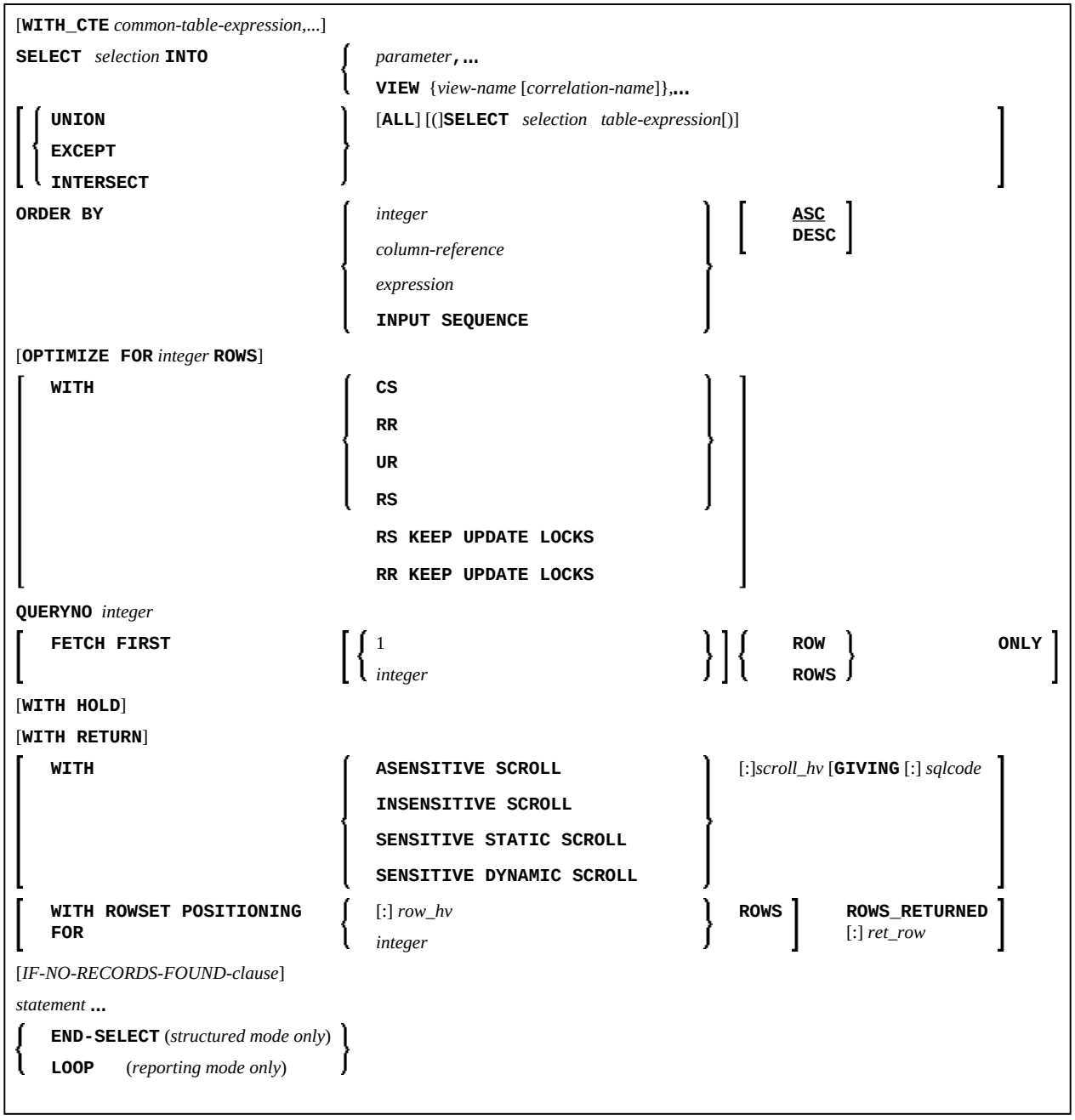
Zwei verschiedene Strukturen sind möglich:

### Syntax 1 – Cursor-orientierte Auswahl

Common Set-Syntax




Extended Set-Syntax:




Syntax-Elementbeschreibung – Syntax 1:

| | |
|---|---|
| **SELECT *selection*** | 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.<br><br>Weitere Informationen siehe *SELECT-Cursor-Oriented* weiter unten. |
| **INTO** | In der `INTO`-Klausel geben Sie die Zielfelder im Programm an, die mit dem Ergebnis der Abfrage gefüllt werden sollen.<br><br>Weitere Informationen und Beispiele siehe *INTO-Klausel* weiter unten. |
| **VIEW** | 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).<br><br>Weitere Informationen und Beispiele siehe *VIEW-Klausel* weiter unten. |
| ***table-expression*** | Die *table-expression* besteht aus einer `FROM`-Klausel und außerdem einer optionalen `WHERE`-Klausel.<br><br>Weitere Informationen und Beispiele siehe *table-expression* weiter unten. |
| **UNION** | `UNION` vereinigt die Ergebnisse von zwei oder mehr *select-expressions* miteinander. Weitere Informationen und ein Beispiel siehe *Abfrage mit UNION* weiter unten. |
| **ORDER BY** | Die `ORDER BY`-Klausel sortiert das Ergebnis der Abfrage in einer bestimmten Reihenfolge.<br><br>Weitere Informationen und Beispiele siehe *ORDER BY-Klausel* weiter unten. |
| **IF NO RECORDS FOUND** | 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.<br><br>Weitere Informationen siehe *IF NO RECORDS FOUND-Klausel* weiter unten. |
| **END-SELECT** | Das für Natural reservierte Schlüsselwort `END-SELECT` muss zum Beenden des `SELECT`-Statements benutzt werden. |

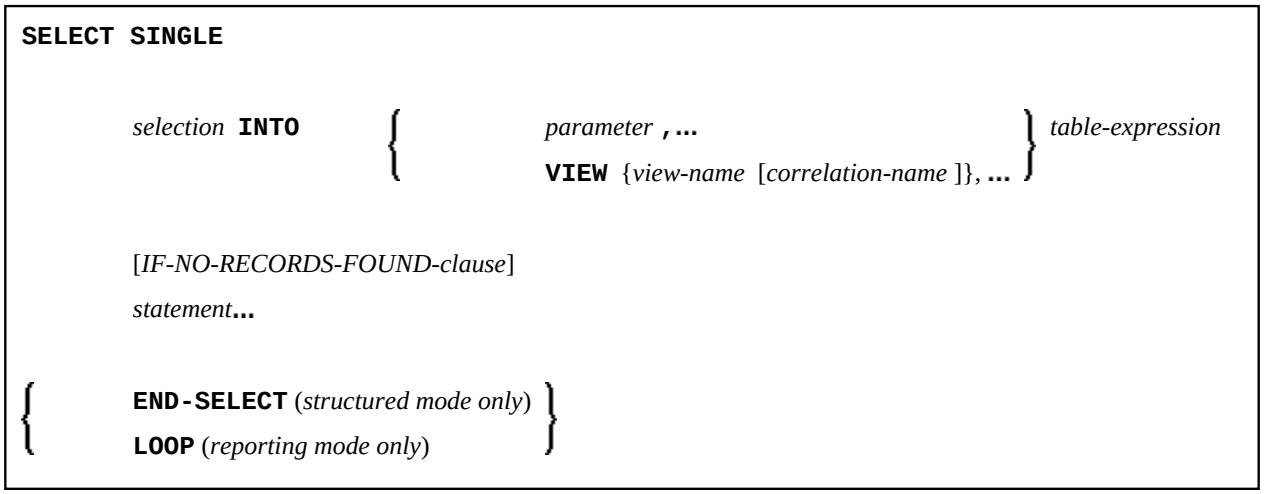
Die folgenden Sytax-Elemente gehören zum SQL Extended Set:

| | |
|---|---|
| **WITH_CTE *common-table-expression,...*** | **WITH_CTE *common-table-expression*:**<br><br>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 *common-table-expressions* angegeben werden. Jeder dieser Ausdrücke kann in der `FROM`-Klausel einer nachfolgenden *common-table-expression* referenziert werden.<br><br>Weitere Informationen siehe *SELECT- Cursor-Oriented*. |
| **OPTIMIZE FOR** | **OPTIMIZE FOR-Klausel:**<br><br>Weitere Informationen finden Sie im Teil *Natural for DB2* der *Database Management System Interfaces*-Dokumentation. |
| **WITH CS/RS/UR/...** | **WITH CS/RS/UR/...-Klausel:**<br><br>Diese Option ermöglicht es Ihnen, einen expliziten Isolation Level anzugeben, mit dem das Statement ausgeführt werden soll. Weitere Informationen zu dieser Klausel siehe *Statement and System Variables* im Teil *Natural for DB2* der *Database Management System Interfaces*-Dokumentation. |
| **QUERYNO** | **QUERYNO-Klausel:**<br><br>Die `QUERYNO`-Klausel gibt die Anzahl an, die für dieses SQL-Statement in `EXPLAIN`-Ausgaben und Ablaufverfolgungssätzen benutzt werden soll. Weitere Informationen zu dieser Klausel finden Sie im Abschnitt *Statement and System Variables* im Teil *Natural for DB2* der *Database Management System Interfaces*-Dokumentation. |
| **FETCH FIRST** | **FETCH FIRST-Klausel:**<br><br>Die `FETCH FIRST`-Klausel beschränkt die Anzahl der Reihen, die mit `FETCH` eingelesen werden können. Weitere Informationen zu dieser Klausel finden Sie im Abschnitt *Statement and System Variables* im Teil *Natural for DB2* der *Database Management System Interfaces*-Dokumentation. |
| **WITH HOLD** | **WITH HOLD-Klausel:**<br><br>Weitere Informationen finden Sie im entsprechenden Abschnitt im Teil *Natural for DB2* der *Database Management System Interfaces*-Dokumentation. |
| **WITH RETURN** | **WITH RETURN-Klausel:**<br><br>Weitere Informationen finden Sie im entsprechenden Abschnitt im Teil *Natural for DB2* der *Database Management System Interfaces*-Dokumentation. |

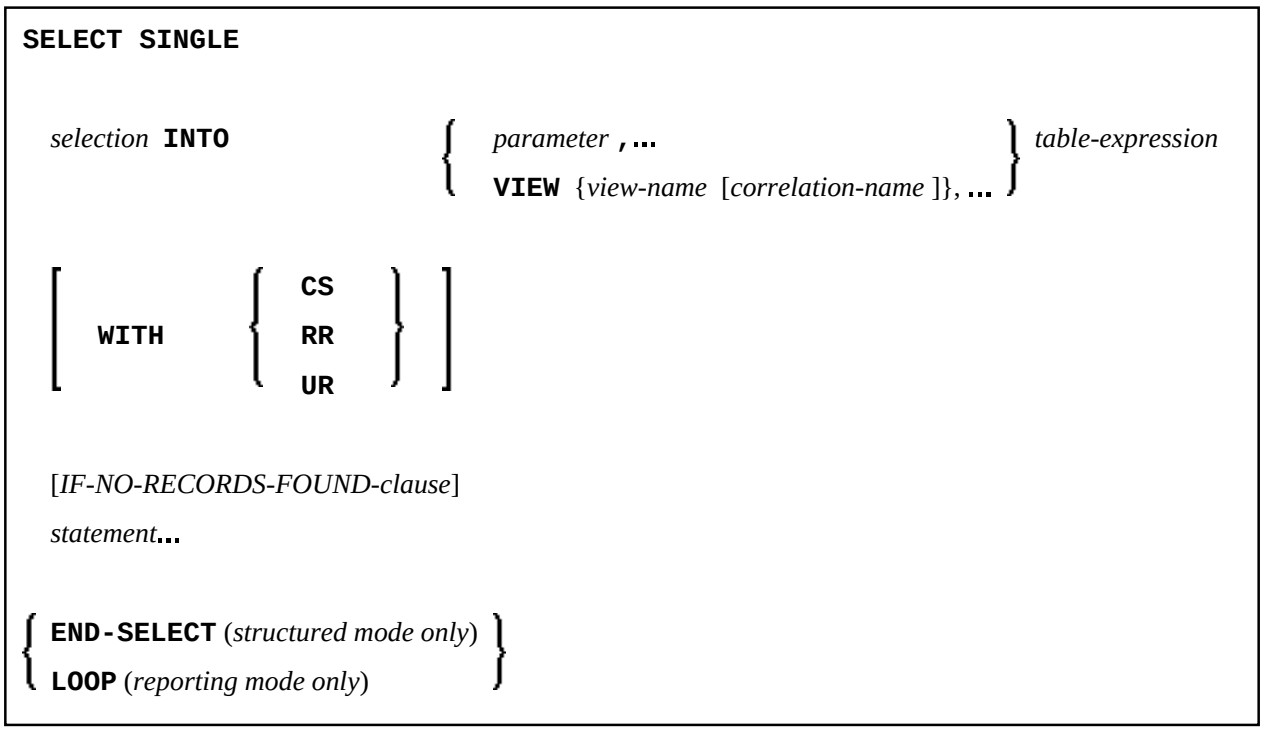
| | |
|---|---|
| **WITH ... SCROLL** | **WITH ... SCROLL-Klausel:**<br><br>Beliebig positionierbare DB2-Cursor werden mit dieser Klausel aktiviert. Beliebig positionierbare Cursor können entweder `ASENSITIVE`, `INSENSITIVE`, `SENSITIVE STATIC` oder `SENSITIVE DYNAMIC` sein.<br><br>● Mit `WITH ASENSITIVE SCROLL` wird angegeben, dass der Cursor entweder `INSENSITIVE` oder `SENSITIVE DYNAMIC` ist. Dies wird von DB2 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 `INSENSITIVE`. Wenn der Cursor Lese- und Schreibzugriff garantiert, wird er `SENSITIVE DYNAMIC`.<br><br>● Mit `WITH INSENSITIVE SCROLL` 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. Positioned Updates und Deletes sind nicht zulässig beim `INSENSITIVE SCROLL`-Cursor.<br><br>● Mit `WITH SENSITIVE STATIC` 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. Positioned Updates und Deletes sind für `SENSITIVE STATIC SCROLL`-Cursor zulässig.<br><br>● Mit `WITH SENSITIVE DYNAMIC` wird angegeben, dass der Cursor sensitiv ist für Aktualisierungen, Löschungen und Einfügungen gegen die Basistabelle, nachdem der Cursor geöffnet wurde. Positioned Updates und Deletes sind für `SENSITIVE DYNAMIC SCROLL`-Cursor zulässig.<br><br>Beliebig positionierbare Cursor ermöglichen es der Anwendung, eine beliebige Reihe im Result Set jederzeit zu positionieren, solange der Cursor offen ist.<br><br>Die Positionierung wird durchgeführt in Abhängigkeit vom Inhalt von *scroll_hv*. Der Inhalt wird jedesmal ausgewertet, wenn ein `FETCH` auf DB2 ausgeführt wird.<br><br>Weitere Informationen siehe *SELECT – Cursororientierte Auswahl*. |

## Syntax 2 – Nicht cursor-orientierte Auswahl

Common Set-Syntax



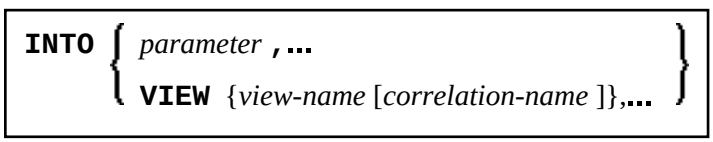

Extended Set-Syntax




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. Weitere Informationen finden Sie im Abschnitt *SELECT SINGLE - Non-Cursor-Oriented* im Teil *Natural for DB2* der *Database Management System Interfaces*-Dokumentation. |
| **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 *INTO-Klausel* weiter unten. |
| **VIEW** | 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). Weitere Informationen und Beispiele siehe *VIEW-Klausel* weiter unten. |
| ***table-expression*** | Die *table-expression* besteht aus einer `FROM`-Klausel und einer optionalen `WHERE`-Klausel. Weitere Informationen und Beispiele siehe Abschnitt *table-expression* weiter unten. |
| **WITH CS/RR/UR** | **WITH CS/RR/UR-Klausel:**<br><br>Diese Klausel gehört zum SQL Extended Set.<br><br>Mit dieser Klausel können Sie einen expliziten Isolation Level angeben, auf dem das Statement ausgeführt werden soll. Weitere Informationen finden Sie im Abschnitt *Statement and System Variables* im Teil *Natural for DB2* der *Database Management System Interfaces*-Dokumentation. |
| **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 *IF NO RECORDS FOUND-Klausel* weiter unten. |
| **END-SELECT** | Das für Natural reservierte Schlüsselwort `END-SELECT` muss zum Beenden des `SELECT`-Statements benutzt werden. |

## INTO-Klausel




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.

| | |
|---|---|
| ***parameter*** | Wenn Sie einzelne *parameter* als Zielfelder angeben, müssen sie in Anzahl und Format mit den in der entsprechenden *selection* angegebenen *columns* bzw. *scalar-expressions* übereinstimmen, wie oben beschrieben. Siehe *scalar-expressions*.<br><br>Beispiel:<br><br>`DEFINE DATA LOCAL`<br>`01 #NAME (A20)`<br>`01 #AGE (I2)`<br>`END-DEFINE`<br>`...`<br>`SELECT NAME, AGE`<br>`INTO #NAME, #AGE`<br>`FROM SQL-PERSONNEL`<br>`...`<br><br>Die Zielfelder #NAME und #AGE, die Natural-Programmvariablen sind, erhalten den Inhalt der Datenbankspalten NAME und AGE. |
| ***correlation-name*** | Wenn die VIEW-Klausel in einem SELECT * verwendet wird, in dem mehrere Tabellen mit JOIN verknüpft werden, sind *correlation-names* 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 *correlation-name* qualifiziert. Der einem View zugewiesene *correlation-name* muss einem der *correlation-names* entsprechen, mit denen die verknüpften Tabellen qualifiziert werden. Siehe auch *Join-Abfragen*.<br><br>Beispiel:<br><br>`DEFINE DATA LOCAL`<br>`01 PERS VIEW OF SQL-PERSONNEL`<br>`02 NAME`<br>`02 FIRST-NAME`<br>`02 AGE`<br>`END-DEFINE`<br>`...`<br>`SELECT *`<br>`INTO VIEW PERS A`<br>`FROM SQL-PERSONNEL A, SQL-PERSONNEL B`<br>`...` |

## *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:

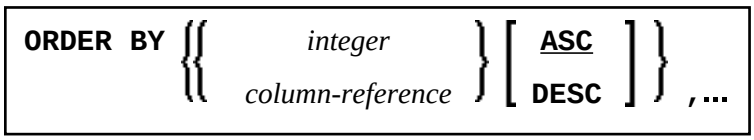
```
DEFINE DATA LOCAL
01 PERS VIEW OF SQL-PERSONNEL
  02 NAME
  02 AGE
  02 ADDRESS (1:6)
END-DEFINE
...
SELECT NAME, AGE, ADDRESS
  INTO VIEW PERS
  FROM SQL-PERSONNEL
  WHERE AGE > 55
UNION ALL
SELECT NAME, AGE, ADDRESS
  FROM SQL-EMPLOYEES
  WHERE PERSNR < 100
ORDER BY NAME
...
END-SELECT
...
```

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




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.

| | |
|---|---|
| *expression* | Gibt einen Ausdruck mit Operatoren an (d.h. nicht nur einen *column-name* oder *integer*). |
| 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:

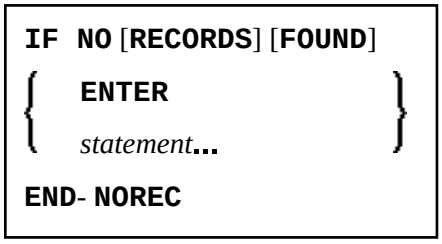
```
DEFINE DATA LOCAL
1 PERS VIEW OF SQL-PERSONNEL
1 NAME
1 AGE
1 ADDRESS   (1:6)
END-DEFINE
...
SELECT NAME, AGE, ADDRESS
  INTO VIEW PERS
  FROM SQL-PERSONNEL
  WHERE AGE = 55
  ORDER BY NAME DESC
  ...
```

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

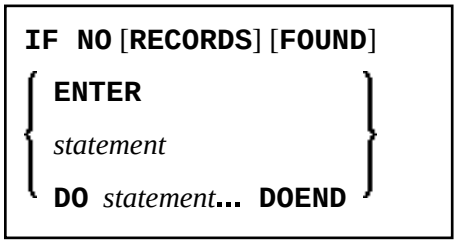
## IF NO RECORDS FOUND-Klausel:

**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




### Reporting Mode-Syntax




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
- QUERYNO
- FETCH FIRST
- WITH HOLD
- WITH RETURN
- 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:

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

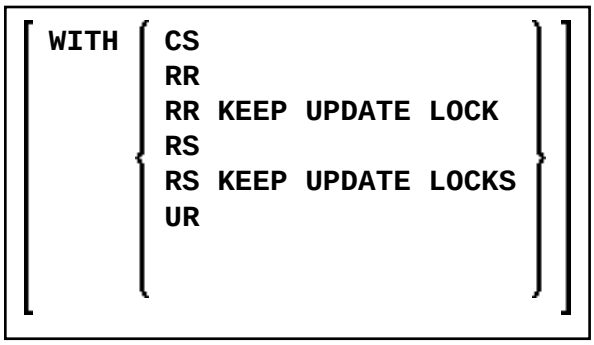
Die OPTIMIZE FOR *integer* ROWS-Klausel wird verwendet, um DB2 im Voraus über die Anzahl (Ganzzahl) von Reihen zu informieren, die von der Ergebnistabelle eingelesen werden sollen. Ohne diese Klausel geht DB2 davon aus, dass alle Reihen der Ergebnistabelle eingelesen werden sollen und führt dementsprechend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




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.<br><br>Lese-Stabilität und Beibehaltung von Aktualisierungssperren. |
| RR KEEP UPDATE LOCKS | Nur gütlig, wenn eine FOR UPDATE OF-Klausel angegeben wird.<br><br>Wiederholbarer Lesevorgang und Beibehaltung von Aktualisierungssperren. |
| UR | Freier Lesevorgang |

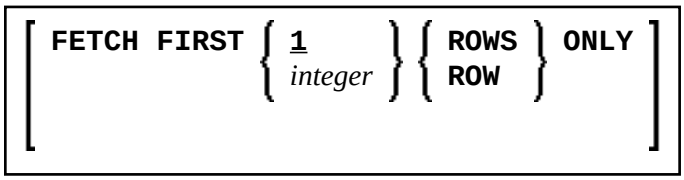
WITH UR kann nur bei einem SELECT-Statement angegeben werden, und wenn es für die Tabelle nur eine Lesezugriffsberechtigung gibt. Der standardmäßige Isolation Level wird durch die Trennung des Pakets oder Plans festgelegt, in den das Statement eingebunden ist. Der standardmäßige Isolation Level ist auch 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.

## QUERYNO


Die QUERNO-Klausel gibt die für dieses SQL-Statement zu benutzende Anzahl bei der EXPLAIN-Ausgabe und der Ablaufverfolgung von Datensätzen an. Die Anzahl wird als QUERYNO-Spalte in der PLAN_TABLE für die Reihen benutzt, die Informationen zu diesem Statement enthalten.

## FETCH FIRST




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 HOLD


Die WITH HOLD-Klausel wird benutzt, um zu verhindern, dass der Cursor durch eine Commit-Operation innerhalb von Datenbankschleifen geschlossen wird. Wenn WITH HOLD angegeben wird, schreibt eine Commit-Operation alle Änderungen der aktuellen logischen Arbeitseinheit weg, gibt aber nur Sperren frei, die nicht zur Verwaltung des Cursors erforderlich sind. Diese optionale Klausel ist hauptsächlich nützlich bei der Stapelverarbeitung, sie wird ignoriert im pseudo-konversationalen CICS-Modus und in meldungsgesteuerten IMS-Programmen.

Beispiel:

```
SELECT name INTO #name FROM table
WHERE AGE = 2 WITH HOLD
```

## WITH RETURN


Die WITH RETURN-Klausel dient zum Erstellen von Result Sets (Ergebnismengen). Deshalb gilt diese Klausel nur für Programme, die als eine Natural Stored Procedure eingesetzt werden. Wenn die WITH RETURN-Klausel in einem SELECT-Statement angegeben wird, bleibt der zugrundeliegende Cursor offen, wenn die damit verbundene Verarbeitungsschleife verlassen wird, außer wenn die Verarbeitungsschleife alle Reihen des Result Set selbst gelesen hätte. Während der ersten Ausführung der Verarbeitungsschleife wird nur der Cursor geöffnet. Die erste Reihe wird noch nicht abgerufen. Dadurch wird es dem Natural-Programm ermöglicht, einen vollständigen Result Set an den Aufrufer der Stored Procedure zurückzugeben. Es bleibt dem Natural-Programmierer überlassen, wie viele Reihen von der Natural Stored Procedure abgearbeitet werden und wie viele nicht abgearbeitete Reihen des Result Set an den Aufrufer der Stored Procedure zurückgegeben werden. Wenn Sie Reihen der SELECT-Operation in der Natural Stored Procedure verarbeiten möchten, müssen Sie folgende Codezeilen eingeben, um zu verhindern, dass die erste leere Reihe in der Verarbeitungsschleife verarbeitet wird.

```
IF *counter =1 ESCAPE TOP
END-IF
```

Wenn Sie sich entschließen, die Verarbeitung der Reihen zu beenden, müssen Sie folgende Codezeilen in der Verarbeitungsschleife eingeben:

```
If condition ESCAPE BOTTOM
END-IF
```

Wenn das Programm alle Reihen des Result Set liest, wird der Cursor geschlossen, und es wird kein Result Set für dieses `SELECT WITH RETURN` an den Aufrufer der Stored Procedure zurückgegeben.

Die folgenden Programme sind Beispiele zum Einlesen vollständiger Result Sets (Beispiel 1) und unvollständiger Result Sets (Beispiel 2).

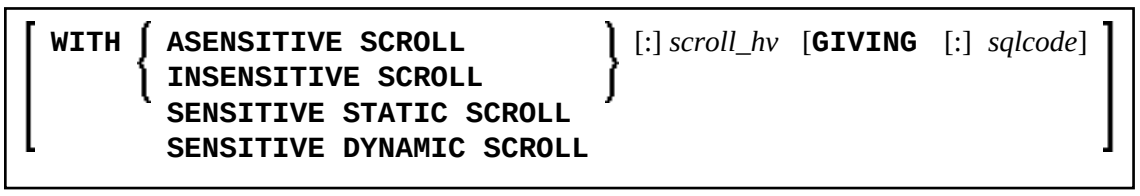
### Beispiel 1:

```
DEFINE DATA LOCAL
. . .
END DEFINE
*
*  Return all rows of the result set
*
SELECT * INTO VIEW V2
                    FROM SYSIBM-SYSROUTINES
                    WHERE RESULT_SETS > 0
                    WITH RETURN
ESCAPE BOTTOM
END-SELECT
END
```

### Beispiel 2:

```
DEFINE DATA LOCAL
. . .
END DEFINE
*
*  Read the first two rows and return the rest as result set
*
SELECT * INTO VIEW V2
                    FROM SYSIBM-SYSROUTINES
                    WHERE RESULT_SETS > 0
                    WITH   RETURN
WRITE PROCEDURE *COUNTER
IF *COUNTER = 1 ESCAPE TOP END-IF
IF *COUNTER = 3 ESCAPE BOTTOM END-IF
END-SELECT
END
```

## WITH INSENSITIVE/SENSITIVE




Natural for DB2 unterstützt Scrollable Cursor von DB2 über die Klauseln `WITH ASENSITIVE SCROLL`, `WITH SENSITIVE STATIC SCROLL` und `SENSITVE DYNAMIC SCROLL`. Scrollable Cursor ermöglichen es Natural for DB2 -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 – Sensitivitätsangabe
- scroll_hv - Optionen
- GIVING [:] sqlcode

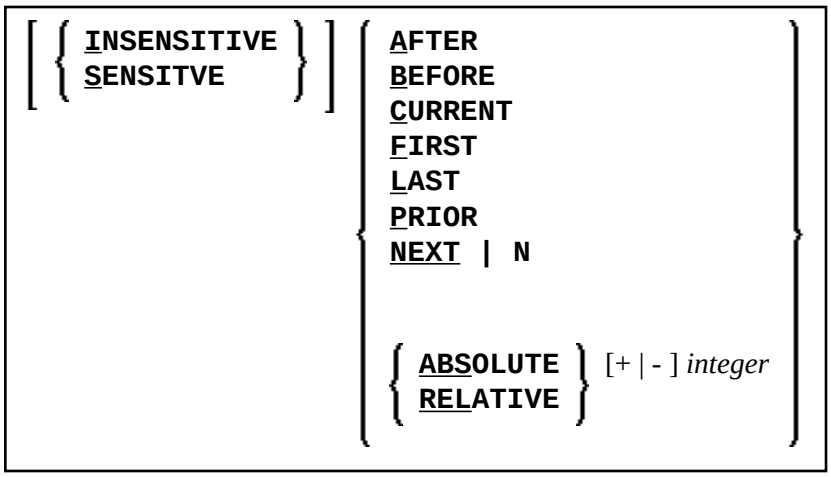
### *scroll_hv*

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.

Außerdem gibt sie während einer FETCH-Operation die Sensitivität von UPDATEs oder DELETEs gegen die Basistabelle an.

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




### *scroll_hv* – Sensitivitätsangabe

Die Angabe der Sensitivität `INSENSITIVE` oder `SENSITIVE` ist optional.

Wenn sie bei einem `FETCH` gegen einen `INSENSITIVE SCROLL`-Cursor weggelassen wird, ist die Voreinstellung `INSENSITIVE`.

Wenn sie bei einem `FETCH` gegen einen `SENSITIVE STATIC/DYNAMIC SCROLL`-Cursor weggelassen wird, ist die Voreinstellung `SENSITIVE`.

Die Sensitivität gibt an, ob die Reihen in der Basistabelle überprüft werden oder nicht, wenn Sie eine `FETCH`-Operation für einen Scrollable Cursor ausführen.

Wenn die entsprechende Basistabellen-Spalte für die `WHERE`-Klausel zulässig ist und nicht gelöscht wurde, gibt ein `SENSITIVE FETCH` die Reihe der Basistabelle zurück.

Wenn die entsprechende Basistabellen-Spalte nicht für die `WHERE`-Klausel zulässig ist oder nicht gelöscht wurde, gibt `SENSITIVE FETCH` ein `UPDATE`-Hole- oder einen `DELETE`-Hole-Status (SQLCODE +222) zurück. Ein `INSENSITIVE FETCH` überprüft nicht die betreffende Basistabellen-Spalte.

### *scroll_hv* - Optionen

Im Folgenden finden Sie eine Erläuterung der Optionen, die zur Bestimmung der abzurufenden Reihe(n), der Position, ab der der Abholvorgang gestartet wird, und/oder der Richtung, in der geblättert werden soll, dienen:

| Option | Erläuterung |
|---|---|
| AFTER | Positioniert nach der letzten Reihe. Es wird keine Reihe abgerufen. |
| BEFORE | Positioniert vor der ersten Reihe. Es wird keine Reihe abgerufen. |
| 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. |
| *+/- integer* | Gilt nur in Verbindung mit `ABSOLUTE` oder `RELATIVE`.<br><br>Gibt die Position der abzurufenden Reihe `ABSOLUTE` oder `RELATIVE` an.<br><br>Geben Sie ein Plus- (+) oder Minus-Zeichen (–) und dahinter eine Ganzzahl ein.<br><br>Die Voreinstellung ist ein Plus-Zeichen (+). |
| ABSOLUTE | Gilt nur in Verbindung mit +/– *integer*.<br><br>Benutzt *integer* als die absolute Position innerhalb des Result Set, von dem aus die Reihe abgerufen wird.<br><br>Weitere Einzelheiten hinsichtlich auf positiver und negativer Positionsziffern entnehmen Sie der *DB2 SQL Reference*-Dokumentation von IBM. |
| RELATIVE | Gilt nur in Verbindung mit +/– *integer*.<br><br>Benutzt *integer* als Position *relativ* zur aktuellen Position innerhalb des Result Set, von dem aus die Reihe abgerufen wird.<br><br>Weitere Einzelheiten hinsichtlich auf positiver und negativer Positionsziffern entnehmen Sie der *DB2 SQL Reference*-Dokumentation von IBM. |

### 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. |
| +222 | UPDATE/DELETE-Hole, Cursor noch geöffnet, keine Daten zurückgegeben. Die entsprechende Reihe der Basistabelle wurde aktualisiert oder gelöscht, so dass die Reihe nicht mehr für die WHERE-Klausel zulässig ist. |
| +231 | Fetch-Operation mit der Option CURRENT, wobei der Cursor aber nicht in einer Reihe positioniert ist, keine Daten zurückgegeben. Diese Situation entsteht, wenn das vorangegangene FETCH den SQLCODE +100 zurückgegeben hat. |

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.