READ

```
{ READ   }  [{ ALL       }]  [MULTI-FETCH-clause] [RECORDS] [IN] [FILE] view-name
{ BROWSE }  [{ (operand1) }]
            [PASSWORD=operand2]
            [CIPHER=operand3]
            [WITH REPOSITION]
            [sequence/range-specification]
            [STARTING WITH ISN=operand4]
            [WHERE  logical-condition]
            statement ...
END-READ    (structured mode only)
[LOOP]      (reporting mode only)
```

Dieses Kapitel behandelt folgende Themen:

- Funktion
- Syntax-Beschreibung
- Bei READ verfügbare Systemvariablen
- Beispiele

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: ACCEPT/REJECT | AT BREAK | AT START OF DATA | AT END OF DATA | BACKOUT TRANSACTION | BEFORE BREAK PROCESSING | GET TRANSACTION DATA | DELETE | END TRANSACTION | FIND | HISTOGRAM | GET | GET SAME | LIMIT | PASSW | PERFORM BREAK PROCESSING | RETRY | STORE | UPDATE

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Das Statement READ dient dazu, Datensätze von der Datenbank zu lesen. Die Datensätze können in physischer Reihenfolge, in der Reihenfolge der Adabas-ISNs oder in der Reihenfolge der Werte eines Deskriptorfeldes gelesen werden.

Das READ-Statement initiiert eine Verarbeitungsschleife.

Siehe auch Abschnitt *READ-Statement* im *Leitfaden zur Programmierung*.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand1	C	S						N	P	I		B *						ja	nein
operand2	C	S				A												ja	nein
operand3	C	S						N										ja	nein
operand4	C	S						N	P	I		B *						ja	nein

* Format B von *operand1* and *operand4* kann mit einer Länge von kleiner gleich 4 benutzt werden.

Syntax-Element-Beschreibung:

operand1	**Anzahl der zu lesenden Datensätze:**
	
	Sie können die Anzahl der Datensätze, die mit dem `READ`-Statement gelesen werden sollen, durch Angabe von *operand1* (in Klammern hinter dem Schlüsselwort `READ`) in Form einer numerischen Konstanten (`0` bis `4294967295`) oder über eine Benutzervariable (in Klammern hinter dem Schlüsselwort READ) begrenzen. Zum Beispiel:
	
	`READ (5) IN EMPLOYEES ...`
	
	`MOVE 10 TO CNT(N2)`
	`READ (CNT) EMPLOYEES ...`
	
	Das angegebene Limit hat für dieses Statement Vorrang vor einem mit einem `LIMIT`-Statement gesetzten Limit.
	
	Ist mit dem Profil-/Session-Parameter `LT` ein kleineres Limit gesetzt, so gilt das `LT`-Limit.
	
	Anmerkungen:
	
	1. Wenn Sie eine vierstellige Anzahl von Sätzen lesen möchten, geben Sie diese mit einer vorangestellten Null an: (`0`*nnnn*); denn Natural interpretiert jede vierstellige Zahl in Klammern als Zeilennummer-Referenzierung auf ein Statement.
	2. *operand1* wird zu Beginn des ersten `READ`-Schleifendurchlaufs ausgewertet. Wird der Wert von *operand1* innerhalb der `READ`-Schleife geändert, hat dies keine Auswirkungen auf die Anzahl der gelesenen Datensätze.

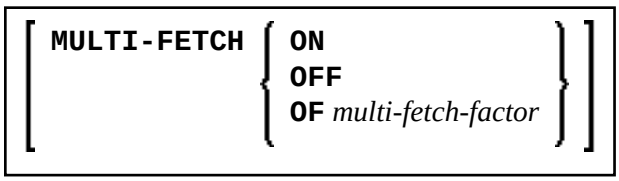
ALL	Um hervorzuheben, dass *alle* Datensätze gelesen werden sollen, können Sie optional das Schlüsselwort `ALL` angeben.
MULTI-FETCH-clause	Siehe *MULTI-FETCH-Klausel*.
view-name	**View-Name:** Als *view-name* geben Sie den Namen eines Views an, der entweder in einem `DEFINE DATA`-Statement oder in einer programmexternen Global oder Local Data Area definiert ist. Im Reporting Mode ist *view-name* der Name eines DDM, falls kein `DEFINE DATA LOCAL`-Statement benutzt wird.

PASSWORD **CIPHER**	**PASSWORD- und CIPHER-Klauseln** Diese Klauseln gelten nur für Zugriffe auf Adabas- oder VSAM-Datenbanken. Mit Entire System Server können sie nicht verwendet werden. Die `PASSWORD`-Klausel dient dazu, ein Passwort anzugeben, um auf Daten einer passwortgeschützten Datei zugreifen zu können. Die `CIPHER`-Klausel dient dazu, einen Cipher-Code (Chiffrierschlüssel) anzugeben, um in chiffrierter Form gespeicherte Daten in entschlüsselter Form zu erhalten. Weitere Informationen hierzu siehe Statements `FIND` und `PASSW`.
WITH REPOSITION	Diese Option macht das `READ`-Statement empfänglich für Repositionierungsereignisse. Siehe *WITH REPOSITION Option*.
sequence/range-specification	Diese Option gibt Lese-Reihefolge und -umfang an. Siehe *Lesereihenfolge und -umfang*.

STARTING WITH ISN=*operand4*	Diese Klausel gilt nur für Adabas- und VSAM-Datenbanken. **Zugriff auf Adabas** Diese Klausel kann in Verbindung mit einem READ-Statement in physischer oder logischer Reihenfolge (aufsteigend/absteigend) verwendet werden. Der angegebene Wert (*operand4*) steht für eine Adabas ISN und wird verwendet, um einen bestimmten Datensatz anzugeben, ab dem die READ-Leseschleife gestartet werden soll. **Logische Reihenfolge** Auch wenn das READ-Statement mit einem Gleichheitszeichen (=) versehen ist, gibt es nicht nur diejenigen Datensätze mit genau dem Startwert in dem betreffenden Deskriptorfeld zurück, sondern startet ab dem angegebenen Startwert einen logischen Suchlauf in aufsteigender oder absteigender Reihenfolge. Wenn einige Datensätze im Deskriptorfeld denselben Inhalt haben, werden sie in der Reihenfolge der ISNs sortiert zurückgegeben. Die Klausel STARTING WITH ISN ist so was wie ein "Selektionskriterium der zweiten Stufe", das nur gilt, wenn der Startwert mit dem Deskriptorwert für den ersten Datensatz übereinstimmt. Alle Datensätze mit einem Deskriptorwert, der mit dem Startwert identisch ist, und mit einer ISN, die *kleiner gleich* (*größer gleich* für ein absteigendes READ) der Start-ISN ist, werden von Adabas ignoriert. Der erste in der READ-Schleife zurückgegebene Datensatz ist entweder ● der erste Datensatz mit Deskriptor = Startwert und einer ISN *größer* (*kleiner* für ein absteigendes READ) als die Start-ISN ● oder wenn ein solcher Datensatz nicht vorhanden ist, der erste Datensatz mit einem Deskriptor *größer* (*kleiner* für ein absteigendes READ) als der Startwert. **Physische Reihenfolge** Die Datensätze werden in der Reihenfolge zurückgegeben, in der sie physisch gespeichert sind. Wenn eine STARTING WITH ISN-Klausel angegeben wird, ignoriert Adabas alle Datensätze, bis der Datensatz mit der ISN, die mit der Start-ISN identisch ist, erreicht ist. Der erste zurückgegebene Datensatz ist der nächste auf den Datensatz mit der Start-ISN folgende Datensatz. **Zugriff auf VSAM** Diese Klausel kann nur in physischer Reihenfolge verwendet werden. Der angegebene Wert (*operand4*) steht für eine VSAM RBA (relative Byte-Adresse von ESDS) oder RRN (relative Datensatznummer von RRDS), die als Startwert für die READ-Leseoperation verwendet werden soll. **Verwendung** Diese Klausel kann zum Repositionieren innerhalb einer READ-Schleife, deren Verarbeitung unterbrochen wurde, benutzt werden, um auf einfache Weise den nächsten Datensatz zu bestimmen, mit dem die Verarbeitung fortgesetzt werden soll. Dies ist besonders hilfreich, wenn der nächste Datensatz sich nicht eindeutig durch einen seiner Deskriptorwerte ermitteln lässt. Die Klause kann auch hilfreich sein in einer verteilten Client/Server-Anwendung, in der das Lesen der Datensätze von einem Server-Programm und die weitere Verarbeitung der Datensätze von einem Client-Programm durchgeführt werden, wobei die Datensätze nicht alle auf einmal, sondern stapelweise verarbeitet werden. Beispiel: Siehe Programm REASISND weiter unten.
WHERE *logical-condition*	Siehe *WHERE-Klausel*.
END-READ	Das für Natural reservierte Schlüsselwort END–READ muss zum Beenden des READ-Statements benutzt werden.

MULTI-FETCH Clause

Anmerkung:
Diese Klausel kann nur bei Adabas- oder DB2-Datenbanken benutzt werden.



Ausführliche Informationen siehe *Multi-Fetch-Klausel* (Adabas) im *Leitfaden zur Programmierung* oder *Multiple Row Processing* (SQL) im *Natural for DB2*-Teil der *Database Management System Interfaces*-Dokumentation.

WITH REPOSITION-Option

Anmerkung:
Diese Option ist nur beim Zugriff auf Adabas-, VSAM- oder DL/I-Datenbanken möglich.

Mit dieser Option können Sie innerhalb der aktiven `READ`-Schleife auf einen anderen Startwert für die zu lesenden Datensätze repositionieren. Die Verarbeitung des `READ`-Statements wird dann unter Verwendung des neuen Startwerts fortgesetzt.

Die Repositionierung kann auf zwei verschiedene Arten ausgelöst werden, wenn Sie ein `READ`-Statement in Zusammenhang mit der `WITH REPOSITION`-Option verwenden:

1. Wenn ein `ESCAPE TOP REPOSITION`-Statement ausgeführt wird, verzweigt Natural direkt zum Schleifenanfang und führt einen Neustart aus; d.h. dass die Datenbank im Einklang mit dem aktuellen Inhalt der Suchwert-Variable auf einen neuen Datensatz in der Datei repositioniert. Gleichzeitig wird der Schleifenzähler `*COUNTER` auf Null (0) zurückgesetzt.

2. Wenn eine `READ`-Schleife versucht, den nächsten Datensatz aus der Datenbank aufzurufen, und der Wert der Systemvariable `*COUNTER` Null (0) ist.

   Anmerkung:
   Wenn `*COUNTER` innerhalb der aktiven `READ`-Schleife auf Null gesetzt wird, wird die Verarbeitung des aktuellen Datensatzes fortgesetzt; es erfolgt keine sofortige Verzweigung zum Schleifenanfang.

Funktionstechnische Überlegungen

- Wenn das `READ`-Statement ein Schleifen-Limit hat (z.B. `READ (10) EMPLOYEES WITH REPOSITION ..`) und ein Neustart-Event ausgelöst wurde, arbeitet die Schleife 10 neue Datensätze ab, egal wieviele Datensätze bereits abgearbeitet worden sind, bis die Repositionierung erfolgt ist.

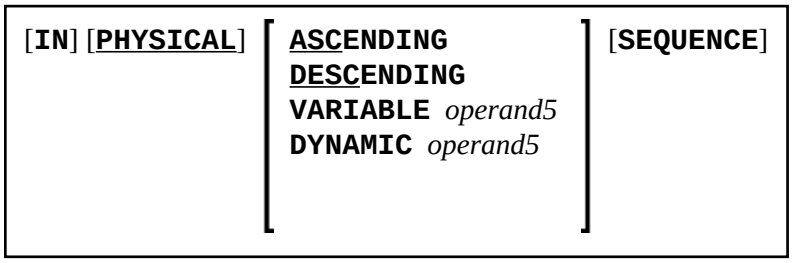
- Wenn ein `ESCAPE TOP REPOSITION`-Statement ausgeführt wird, die innerste Schleife aber keine Repositionierung ausführen kann (da das Schlüsselwort `WITH REPOSITION` nicht im `READ`-Statement gesetzt ist, oder es sich beim abgesetzten Schleifen-Statement nicht um ein `READ` handelt), wird ein entsprechender Laufzeitfehler ausgegeben.

- Da das ESCAPE TOP-Statement keine Referenzen erlaubt, können Sie nur einen Repositionierungs-Event initiieren, wenn die innerste Verarbeitungsschleife ein READ .. WITH REPOSITION-Statement ist.
- Ein Repositionierungs-Event löst weder die Ausführung des AT START OF DATA-Programmabschnittes aus, noch löst er die erneute Verarbeitung des Schleifenlimit-Operanden aus (wenn es sich um eine Variable handelt).
- Wenn der Suchwert nicht geändert wurde, repositioniert die Schleife auf denselben Datensatz wie beim ursprünglichen Schleifenanfang.

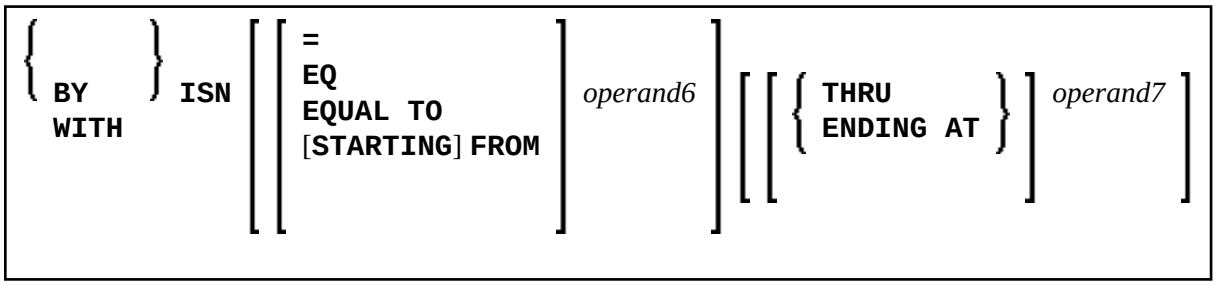
Lesereihenfolge und -umfang (*sequence/range-specification*)

Die folgenden Syntax-Optionen sind verfügbar, um Lese-Reihenfolge und/oder -Bereich anzugeben.

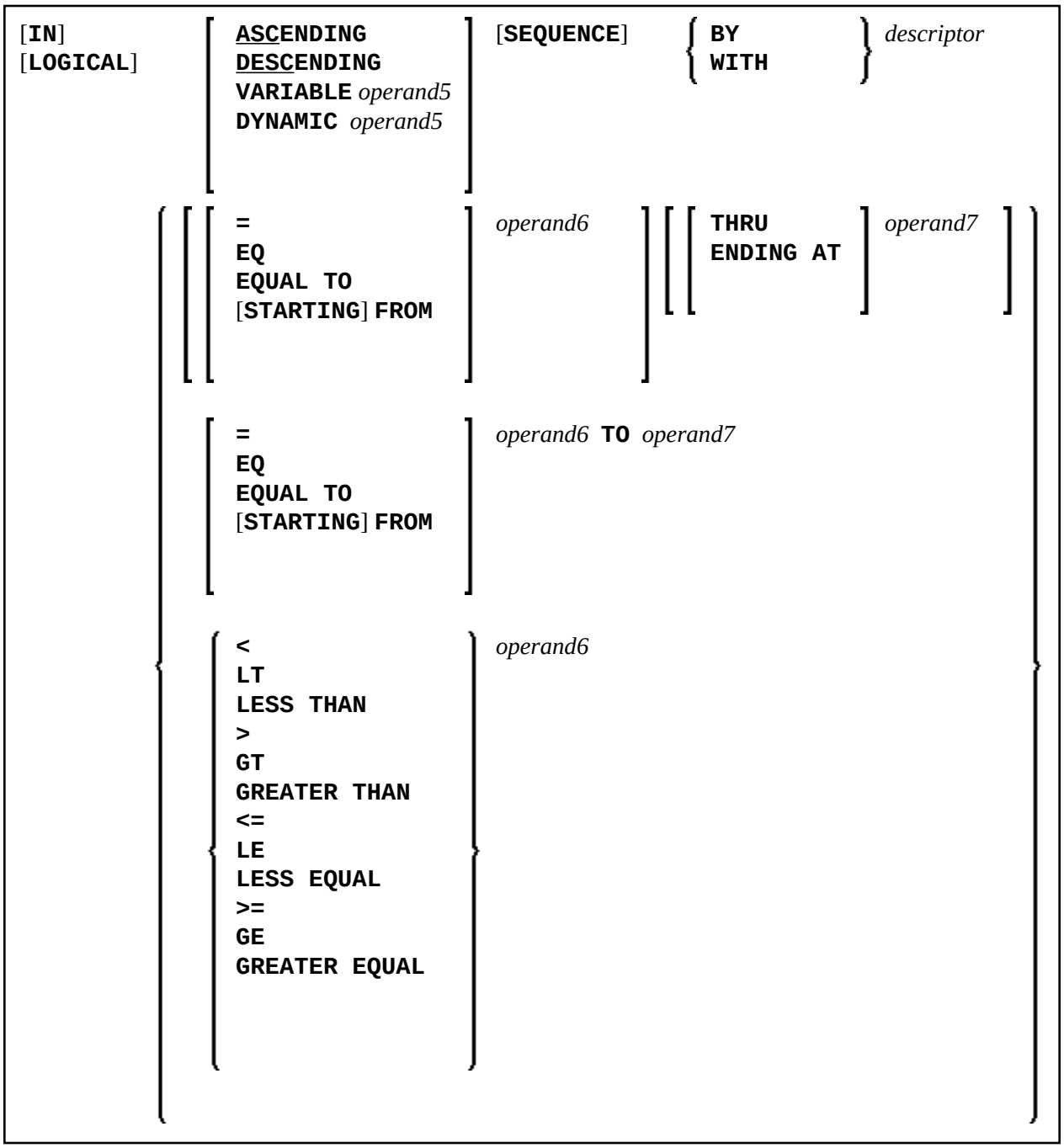
Syntax-Option 1:



Syntax-Option 2:



Syntax-Option 3:



Anmerkungen:

1. Die Syntax-Optionen 2 und 3 sind in Verbindung mit Entire System Server nicht verfügbar.
2. Im Diagram zu Syntax-Option 3 finden Sie Vergleichsoperanden, die ab Natural Version 4 für Großrechner verwendet werden können. Wenn diese Vergleichsoperanden zum Einsatz kommen, dürfen die Optionen `ENDING AT`, `THRU` und `TO` nicht benutzt werden. Diese Vergleichsoperanden gelten auch beim `HISTOGRAM`-Statement.

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate												Referenzierung erlaubt	Dynam. Definition
operand5		S				A												ja	nein
operand6	C	S				A		N	P	I	F	B *	D	T	L			ja	nein
operand7	C	S				A		N	P	I	F	B *	D	T	L			ja	nein

* Format B von *operand6* und *operand7* kann nur mit einer Länge von kleiner gleich 4 benutzt werden.

Syntax-Element-Beschreibung:

READ IN PHYSICAL SEQUENCE	`PHYSICAL SEQUENCE` bedeutet, dass die Datensätze in der physischen Reihenfolge, in der sie auf der Datenbank gespeichert sind, gelesen werden. **Anmerkung:** Bei VSAM-Datenbanken ist `READ PHYSICAL` nur beim Zugriff auf ESDS und RRDS möglich. Ist nichts anderes angegeben, gilt standardmäßig `IN PHYSICAL SEQUENCE`.
READ BY ISN	`READ BY ISN` ist nur bei Adabas- und VSAM-Datenbanken möglich, und zwar bei VSAM nur für ESDS und RRDS. `READ BY ISN` bedeutet, dass die Datensätze in der Reihenfolge der Adabas-ISNs (Interne Satznummern) bzw. VSAM-RBAs (relative Byte-Adressen von ESDS) bzw. -RRNs (relative Datensatznummern von RRDS) gelesen werden. **Anmerkung:** Bei XML-Datenbanken (z.B. Tamino) dient `READ BY ISN` zum Lesen von XML-Objekten entsprechend der Reihenfolge der Objekt-IDs.

READ IN LOGICAL SEQUENCE	`LOGICAL SEQUENCE` bedeutet, dass die Datensätze in der Reihenfolge der Werte eines bestimmten Deskriptorfeldes (Key) gelesen werden. Wenn Sie ein Deskriptorfeld angeben, werden die Datensätze in der Wertabfolge dieses Feldes gelesen. Als Feld können Sie einen Deskriptor, Subdeskriptor, Superdeskriptor oder Hyperdeskriptor verwenden, nicht aber einen phonetischen Deskriptor, einen Deskriptor innerhalb einer Periodengruppe oder einen Superdeskriptor, der ein Periodengruppenfeld enthält. Wenn Sie kein Deskriptorfeld angeben, wird der im verwendeten DDM eingetragene Standarddeskriptor (Feld `Default Sequence`) genommen. **Anmerkungen:** 1. Bei DL/I-Datenbanken: Der verwendete Deskriptor muss entweder das Schlüsselfeld eines Wurzelsegments oder ein Sekundärindexfeld sein. Ein verwendetes Sekundärindexfeld muss ebenfalls im `PROSEQ`-Parameter eines PCB angegeben werden; Natural benutzt diesen PCB und die entsprechende hierarchische Struktur, um die Daten zu lesen. Wenn Sie `READ LOGICAL` bei HDAM-Datenbanken verwenden, erhalten Sie keine sinnvollen Ergebnisse, da HDAM-Datenbanken zum Auffinden von Wurzelsegmenten eine Schlüsselumrechnungsroutine benutzen; daher sollten Sie für HDAM-Datenbanken `READ PHYSICAL` verwenden. 2. Bei VSAM-Datenbanken: `LOGICAL` ist nur möglich für KSDS mit definierten Primär- und Alternativschlüsseln bzw. ESDS mit Alternativschlüsseln. Wird ein Deskriptorfeld verwendet, das mit Nullwertunterdrückung definiert ist (gilt nur für Adabas), werden Datensätze, bei denen das Deskriptorfeld einen Nullwert enthält, nicht gelesen. Wird als Deskriptorfeld ein multiples Feld verwendet (gilt nur für Adabas), wird ein einzelner Datensatz mehrmals gelesen, wenn das Feld mehrere Werte enthält. Informationen zu `READ IN LOGICAL SEQUENCE` finden Sie auch im *Leitfaden zur Programmierung*; siehe *Statements für Datenbankzugriffe, READ-Statement*.

ASCENDING | DESCENDING | VARIABLE | DYNAMIC SEQUENCE	Diese Klausel gilt nur für Adabas-, XML-, VSAM- und SQL-Datenbanken. Bei einem `READ PHYSICAL`-Statement gilt sie nur für VSAM- und DB2-Datenbanken. Mit dieser Klausel können Sie bestimmen, ob die Datensätze in aufsteigender Reihenfolge oder in absteigender Reihenfolge gelesen werden sollen. ● Standardmäßig werden die Datensätze in aufsteigender Reihenfolge gelesen (was Sie mit dem Schlüsselwort `ASCENDING` auch ausdrücklich angeben können, aber nicht müssen). ● Wenn die Datensätze in absteigender Reihenfolge gelesen werden sollen, geben Sie das Schlüsselwort `DESCENDING` an. ● Wenn erst zur Laufzeit bestimmt werden soll, ob die Datensätze in aufsteigender oder absteigender Reihenfolge gelesen werden sollen, geben Sie das Schlüsselwort `VARIABLE` oder `DYNAMIC` gefolgt von einer Variablen (*operand5*) an. Der Wert von *operand5* zu Beginn der `READ`-Verarbeitungsschleife bestimmt dann die Reihenfolge. *operand5* muss Format/Länge A1 haben und kann den Wert `A` (für Ascending/aufsteigend) oder `D` (für Descending/absteigend) enthalten. ○ Wenn das Schlüsselwort `VARIABLE` benutzt wird, wird die Leserichtung (Wert von *operand5*) am Anfang der `READ`-Verarbeitungsschleife ausgewertet, und sie bleibt bestehen, bis die Schleife beendet wird, ganz gleich ob das Feld von *operand5* in der `READ`-Schleife geändert wird oder nicht. ○ Wenn das Schlüsselwort `DYNAMIC` benutzt wird, wird die Leserichtung (Wert von *operand5*) vor jedem Aufruf eines Datensatzes in der `READ`-Verarbeitungsschleife ausgewertet und kann von Datensatz zu Datensatz geändert werden. Dies ermöglicht es überall in der `READ`-Schleife, die Reihenfolge beim Durchblättern von aufsteigend in absteigend (und umgekehrt) zu ändern. **Anmerkungen:** 1. Bei Adabas-Datenbanken: Dynamische Lesereihenfolge setzt folgende Adabas-Version voraus: Adabas V7 (oder höher). 2. Bei XML-Datenbanken steht `DYNAMIC SEQUENCE` nicht zur Verfügung.

STARTING FROM ... ENDING AT/TO	Mit den Klauseln STARTING FROM und ENDING AT können Sie angeben, ab welchem Wert und bis zu welchem Wert des Deskriptorfeldes gelesen werden soll. Die STARTING FROM-Klausel (= oder EQ oder EQUAL TO oder [STARTING] FROM) legt den Startwert für die READ-Operation fest. Wenn ein Startwert angegeben wird, beginnt das Lesen mit dem angegebenen Wert. Wenn der Startwert in der Datei nicht vorhanden ist, wird der nächsthöhere (oder niedrigere bei einem absteigenden READ) Wert benutzt. Wenn kein höherer (oder niedrigerer für absteigendes READ) Wert vorhanden ist, wird die Schleife nicht durchlaufen. Um die Datensätze auf einen Endwert zu begrenzen, können Sie eine ENDING AT-Klausel mit den Bedingungen THRU, ENDING AT oder TO angeben, wobei dann bis einschließlich der angegebenen Werte gelesen wird. Immer wenn das READ-Deskriptorfeld den angegebenen Endwert überschreitet, wird die Schleife automatisch beendet. Obwohl die Basis-Funktionalität der Schlüsselwörter TO, THRU und ENDING AT sich jeweils ähnelt, so unterscheiden sie sich doch im Detail.
THRU/ENDING AT	Wenn THRU oder ENDING AT benutzt wird, wird nur der Startwert an die Datenbank übergeben, aber die Endwerte-Prüfung vom Natural-Laufzeitsystem durchgeführt, nachdem der Datensatz von der Datenbank zurückgegeben wird. Wenn die Lese-Richtung ASCENDING (aufsteigend) ist, müssen Sie den niedrigeren Wert als den Startwert und den höheren Wert als den Endwert angeben, da der Startwert den zuerst in der READ-Schleife zurückgegebenen Wert (und den Datensatz) darstellt. Wenn Sie aber einen rückwärtsgerichteten (DESCENDING) Lesevorgang starten, muss der höhere Wert im Startwert und der niedrigere Wert im Endwert erscheinen. Um das Ende des zu lesenden Wertebereichs zu bestimmen, liest Natural intern einen Datensatz über den Endwert hinaus ein. Wenn Sie die READ-Schleife verlassen haben, weil der Endwert erreicht war, denken Sie bitte daran, dass dieser letzte Datensatz wirklich nicht der letzte Datensatz innerhalb des erforderlichen Bereiches ist, sondern der erste Datensatz jenseits dieses Bereiches (außer in dem Fall, dass die Datei keinen weiteren Datensatz nach dem letzten Ergebnisdatensatz enthält). THRU / ENDING AT kann für alle Datenbanken benutzt werden, die die READ- oder HISTOGRAM-Statements unterstützen.

TO	Wenn das Schlüsselwort `TO` verwendet wird, werden sowohl der Startwert als auch der Endwert an die Datenbank übergeben, und Natural führt keine Prüfungen auf Wertebereiche hin aus. Wenn der Endwert überschritten wird, reagiert die Datenbank genauso wie wenn das Dateiende (End-of-File) erreicht wäre, und die Datenbank-Schleife wird verlassen. Da alle Bereichsprüfungen von der Datenbank vorgenommen werden, wird immer der niedrigere Wert (des Bereiches) im Startwert und der höhere Wert im Endwert angegeben, ungeachtet der Tatsache, ob Sie in aufsteigender (`ASCENDING`) oder absteigender (`DESCENDING`) Reihenfolge lesen. Die `TO`-Option steht nur zur Verfügung, wenn die zugrundeliegende Datenbank Adabas Version 7 (oder höher), DB2, VSAM oder DL/I ist.

Anmerkungen zu Funktionsunterschieden zwischen THRU/ENDING AT und TO

Die folgende Liste beschreibt die Funktionsunterschiede zwischen der Benutzung der Option `THRU/ENDING AT` und der Option `TO`.

THRU/ENDING AT	TO
Wenn die READ-Schleife beendet wird, weil der Endwert erreicht worden ist, enthält der View den ersten Datensatz, der außerhalb des Bereiches ist (Out-of-Range).	Wenn die READ-Schleife beendet wird, weil der Endwert erreicht worden ist, enthält der View den letzten Datensatz des angegebenen Bereiches.
Wenn eine Endwert-Variable im Verlauf einer READ-Schleife geändert wird, wird beim nächsten gelesenen Datensatz der neue Wert für eine Endwerte-Prüfung benutzt.	Die Endwert-Variable wird erst beim READ-Schleifenstart verarbeitet. Alle weiteren Änderungen im Verlauf der READ-Schleife haben keine Auswirkung.
Ein falsch angegebener Bereich (z.B. `READ .. ='B' THRU 'A'`) führt nicht zu einem Datenbank- Fehler, sondern dazu, dass einfach kein Datensatz zurückgegeben wird.	Ein falsch angegebener Bereich führt zu einem Datenbank-Fehler (z.B. Adabas RC=61), weil ein Wertebereich nicht in absteigender Reihenfolge angegeben werden darf.
Wenn ein `READ .. DESCENDING` mit Start- und Endwert benutzt wird, wird der Startwert zur Positionierung in der Datei benutzt, wohingegen der Endwert von Natural zum Abprüfen auf das Bereichsende (End-of-Range) hin benutzt wird. Deshalb ist der Startwert höher als der oder gleich dem Endwert.	Da beide Werte an die Datenbank übergeben werden, müssen sie in aufsteigender Reihenfolge erscheinen. Mit anderen Worten, der Startwert ist niedriger als der oder gleich dem Endwert, egal ob in aufsteigender oder in absteigender Reihenfolge gelesen wird.
Um auf einen Bereichsüberlauf hin abzuprüfen, muss der Deskriptorwert in dem zugrunde liegenden Datenbank-View erscheinen, d.h. er muss im Satzpuffer zurückgegeben werden.	Der Deskriptor ist für die zurückgegebenen Satzfelder nicht erforderlich.
Endwerte-Prüfungen auf Adabas-Mehrwertfelder (MU-Feld) oder Sub-/Super-/Hyperdeskritporen hin ist nicht möglich und führt zum Syntaxfehler NAT0160 bei der Kompilierung des Programms.	Sie können einen Endwert für MU-Felder und Sub-/Super-/Hyperdeskriptoren angeben.
Kann für alle Datenbanken benutzt werden.	Kann nur für Adabas Version 7 (oder höher), DB2, VSAM or DL/I benutzt werden.

WHERE-Klausel

WHERE *logical-condition*

Mit der WHERE-Klausel können Sie ein zusätzliches Selektionskriterium in Form einer logischen Bedingung (*logical-condition*) angeben. Diese wird ausgewertet, *nachdem* ein Wert gelesen wurde, aber *bevor* eine weitere Verarbeitung auf der Grundlage dieses Wertes (einschließlich AT BREAK-Verarbeitung) erfolgt.

Näheres zu logischen Bedingungen finden Sie unter *Logische Bedingungen* im *Leitfaden zur Programmierung*.

Ist über ein LIMIT-Statement oder eine Limit-Notation die Anzahl der zu lesenden Datensätze begrenzt, so werden bei einem READ-Statement, das eine WHERE-Klausel enthält, Datensätze, die aufgrund der WHERE-Bedingung nicht weiterverarbeitet werden, bei der Ermittlung des Limits nicht mitgezählt.

Bei READ verfügbare Systemvariablen

Die Natural-Systemvariablen `*ISN` und `*COUNTER` stehen mit dem `READ`-Statement zur Verfügung. Format/Länge dieser Systemvariablen ist P10.

Format und Länge können nicht geändert werden.

Die Systemvariablen werden folgendermaßen verwendet:

***ISN**	Die Systemvariable `*ISN` enthält die Adabas-ISN des gerade verarbeiteten Datensatzes. **Anmerkungen:** 1. Bei VSAM-Datenbanken enthält `*ISN` entweder die RRN (für RRDS) oder die RBA (für ESDS) das aktuellen Datensatzes. 2. Bei DL/I- und SQL-Datenbanken oder mit Entire System Server ist `*ISN` nicht verfügbar.
***COUNTER**	Die Systemvariable `*COUNTER` enthält die Anzahl, wie oft die Verarbeitungsschleife durchlaufen wurde.

Beispiele

- Beispiel 1 — READ-Statement
- Beispiel 2 — READ-Statement mit WITH REPOSITION
- Beispiel 3 — READ- und FIND-Statements miteinander kombiniert
- Beispiel 4 — READ-Statement mit DESCENDING-Option
- Beispiel 5 — READ-Statement mit VARIABLE-Option
- Beispiel 6 — READ-Statement mit DYNAMIC-Option
- Beispiel 7 — READ-Statement mit STARTING WITH ISN-Klausel

Beispiel 1 — READ-Statement

```
** Example 'REAEX1S': READ (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 3
*
WRITE 'READ IN PHYSICAL SEQUENCE'
```

```
READ EMPLOY-VIEW IN PHYSICAL SEQUENCE
  DISPLAY NOTITLE PERSONNEL-ID NAME *ISN *COUNTER
END-READ
*
WRITE / 'READ IN ISN SEQUENCE'
READ EMPLOY-VIEW BY ISN STARTING FROM 1 ENDING AT 3
  DISPLAY         PERSONNEL-ID NAME *ISN *COUNTER
END-READ
*
WRITE / 'READ IN NAME SEQUENCE'
READ EMPLOY-VIEW BY NAME
  DISPLAY         PERSONNEL-ID NAME *ISN *COUNTER
END-READ
*
WRITE / 'READ IN NAME SEQUENCE STARTING FROM ''M'''
READ EMPLOY-VIEW BY NAME STARTING FROM 'M'
  DISPLAY         PERSONNEL-ID NAME *ISN *COUNTER
END-READ
*
END
```

Ausgabe des Programms REAEX1S:

```
PERSONNEL         NAME            ISN         CNT
   ID
--------- -------------------- ----------- -----------

READ IN PHYSICAL SEQUENCE
50005800  ADAM                          1           1
50005600  MORENO                        2           2
50005500  BLOND                         3           3

READ IN ISN SEQUENCE
50005800  ADAM                          1           1
50005600  MORENO                        2           2
50005500  BLOND                         3           3

READ IN NAME SEQUENCE
60008339  ABELLAN                     478           1
30000231  ACHIESON                    878           2
50005800  ADAM                          1           3

READ IN NAME SEQUENCE STARTING FROM 'M'
30008125  MACDONALD                   923           1
20028700  MACKARNESS                  765           2
40000045  MADSEN                      508           3
```

Äquivalentes Reporting-Mode-Beispiel: REAEX1R.

Beispiel 2 — READ-Statement mit WITH REPOSITION

```
DEFINE DATA LOCAL
1 MYVIEW VIEW OF ...
  2 NAME
1 #STARTVAL (A20) INIT <'A'>
1 #ATTR     (C)
END-DEFINE
...
SET KEY PF3
...
READ MYVIEW WITH REPOSITION BY NAME = #STARTVAL
```

```
INPUT (IP=OFF AD=O) 'NAME:' NAME /
    'Enter new start value for repositioning:' #STARTVAL (AD=MT CV=#ATTR) /
    'Press PF3 to stop'
  IF *PF-KEY = 'PF3'
    THEN STOP
  END-IF
  IF #ATTR MODIFIED
    THEN ESCAPE TOP REPOSITION
  END-IF
END-READ
...

DEFINE DATA LOCAL
1 MYVIEW VIEW OF ...
  2 NAME
1 #STARTVAL (A20) INIT <'A'>
1 #ATTR     (C)
END-DEFINE
...
SET KEY PF3
...
READ MYVIEW WITH REPOSITION BY NAME = #STARTVAL
  INPUT (IP=OFF AD=O) 'NAME:' NAME /
    'Enter new start value for repositioning:' #STARTVAL (AD=MT CV=#ATTR) /
    'Press PF3 to stop'
  IF *PF-KEY = 'PF3'
    THEN STOP
  END-IF
  IF #ATTR MODIFIED
    THEN RESET *COUNTER
  END-IF
END-READ
...
```

Beispiel 3 — READ- und FIND-Statements miteinander kombiniert

Das folgende Beispiel verwendet zunächst ein READ-Statement, um Datensätze von der Datei EMPLOYEES (Mitarbeiter) in logischer Reihenfolge der Werte des Deskriptorfeldes NAME zu lesen. Dann werden mit einem FIND-Statement Datensätze von der Datei VEHICLES (Fahrzeuge) gelesen, wobei die Personalnummer (PERSONNEL–ID) der EMPLOYEES-Datei als Suchkriterium benutzt wird.

Der erzeugte Report zeigt den Namen und die Personalnummer aller von der EMPLOYEES-Datei gelesenen Personen sowie die Fabrikate (MAKE) der Autos (von der VEHICLES-Datei), die im Besitz dieser Personen sind. Besitzt eine Person mehrere Autos, werden entsprechend mehrere Zeilen pro Person ausgegeben.

```
** Example 'REAEX2': READ and FIND combination
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 FIRST-NAME
  2 NAME
  2 CITY
1 VEH-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 10
```

```
*
RD. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
  SUSPEND IDENTICAL SUPPRESS
  FD. FIND VEH-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (RD.)
    IF NO RECORDS FOUND
      ENTER
    END-NOREC
    DISPLAY NOTITLE (ES=OFF IS=ON ZP=ON AL=15)
            PERSONNEL-ID (RD.)
            FIRST-NAME (RD.)
            MAKE (FD.) (IS=OFF)

  END-FIND
END-READ
END
```

Ausgabe des Programms REAEX2:

```
   PERSONNEL      FIRST-NAME         MAKE
      ID
--------------- --------------- ---------------

20007500        VIRGINIA        CHRYSLER
20008400        MARSHA          CHRYSLER
                                CHRYSLER
20021100        ROBERT          GENERAL MOTORS
20000800        LILLY           FORD
                                MG
20001100        EDWARD          GENERAL MOTORS
20002000        MARTHA          GENERAL MOTORS
20003400        LAUREL          GENERAL MOTORS
30034045        KEVIN           DATSUN
30034233        GREGORY         FORD
11400319        MANFRED
```

Beispiel 4 — READ-Statement mit DESCENDING-Option

```
** Example 'READSCND': READ (with DESCENDING SEQUENCE)
************************************************************************
DEFINE DATA LOCAL
1 EMPL VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 BIRTH
END-DEFINE
*
READ (10) EMPL IN DESCENDING SEQUENCE BY NAME FROM 'ZZZ'
  DISPLAY *ISN NAME FIRST-NAME BIRTH (EM=YYYY-MM-DD)
END-READ
END
```

Beispiel 5 — READ-Statement mit VARIABLE-Option

```
** Example 'REAVSEQ':  READ  (with VARIABLE SEQUENCE)
************************************************************************
DEFINE DATA LOCAL
1 EMPL VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 BIRTH
*
```

```
1 #DIR        (A1)
1 #STARTVALUE (A20)
END-DEFINE
*
SET KEY PF7 PF8
*
INPUT 'Select READ direction'
   // 'Press' 08T 'PF7' (I)                  21T 'to read backward'
    /         08T 'PF8' (I) 'or' 'ENTER' (I) 21T 'to read forward'
*
IF *PF-KEY = 'PF7'
  MOVE 'D'   TO #DIR
  MOVE 'ZZZ' TO #STARTVALUE
ELSE
  MOVE 'A' TO #DIR
  MOVE 'A' TO #STARTVALUE
END-IF
*
READ (10) EMPL IN VARIABLE #DIR SEQUENCE
               BY NAME FROM #STARTVALUE
  DISPLAY *ISN NAME FIRST-NAME BIRTH (EM=YYYY-MM-DD)
END-READ
END
```

Beispiel 6 — READ-Statement mit DYNAMIC-Option

```
DEFINE DATA LOCAL
1 #DIRECTION (A1) INIT <'A'>   /* 'A' = ASCENDING
1 #EMPVIEW VIEW OF EMPLOYEES
2 NAME
...
END-DEFINE
...
READ #EMPVIEW IN DYNAMIC #DIRECTION SEQUENCE BY NAME = 'SMITH'
   INPUT (AD=O) NAME
       / 'Press PF7 to scroll in DESCENDING sequence'
       / 'Press PF8 to scroll in ASCENDING  sequence'
   ..
   IF *PF-KEY = 'PF7' THEN MOVE 'D' TO #DIRECTION END-IF
   IF *PF-KEY = 'PF8' THEN MOVE 'A' TO #DIRECTION END-IF
END-READ
...
```

Beispiel 7 — READ-Statement mit STARTING WITH ISN-Klausel

```
** Example 'REASISND': READ  (with STARTING WITH ISN)
************************************************************************
DEFINE DATA LOCAL
1 EMPL VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 BIRTH
*
1 #DIR      (A1)
1 #STARTVAL (A20)
1 #STARTISN (N8)
END-DEFINE
*
SET KEY PF3 PF7 PF8
*
MOVE 'ADKINSON' TO #STARTVAL
*
```

```
READ (9) EMPL BY NAME = #STARTVAL
  WRITE *ISN NAME FIRST-NAME BIRTH (EM=YYYY-MM-DD) *COUNTER
  IF *COUNTER = 5 THEN
    MOVE NAME TO #STARTVAL
    MOVE *ISN TO #STARTISN
  END-IF
END-READ
*
#DIR := 'A'
*
REPEAT
 READ EMPL IN VARIABLE #DIR  BY NAME = #STARTVAL
            STARTING WITH ISN = #STARTISN
    MOVE NAME TO #STARTVAL
    MOVE *ISN TO #STARTISN
    INPUT NO ERASE  (IP=OFF AD=O)
         15/01 *ISN  NAME  FIRST-NAME  BIRTH (EM=YYYY-MM-DD)
           //  'Direction:' #DIR
           //  'Press PF3 to stop'
           /   '      PF7 to go step back'
           /   '      PF8 to go step forward'
           /   '      ENTER to continue in that direction'
    /*
    IF *PF-KEY = 'PF7' AND #DIR = 'A'
      MOVE 'D' TO #DIR
      ESCAPE BOTTOM
    END-IF
    IF *PF-KEY = 'PF8' AND #DIR = 'D'
      MOVE 'A' TO #DIR
      ESCAPE BOTTOM
    END-IF
    IF *PF-KEY = 'PF3'
      STOP
    END-IF
  END-READ
  /*
  IF *COUNTER(0290) = 0
    STOP
  END-IF
END-REPEAT
END
```