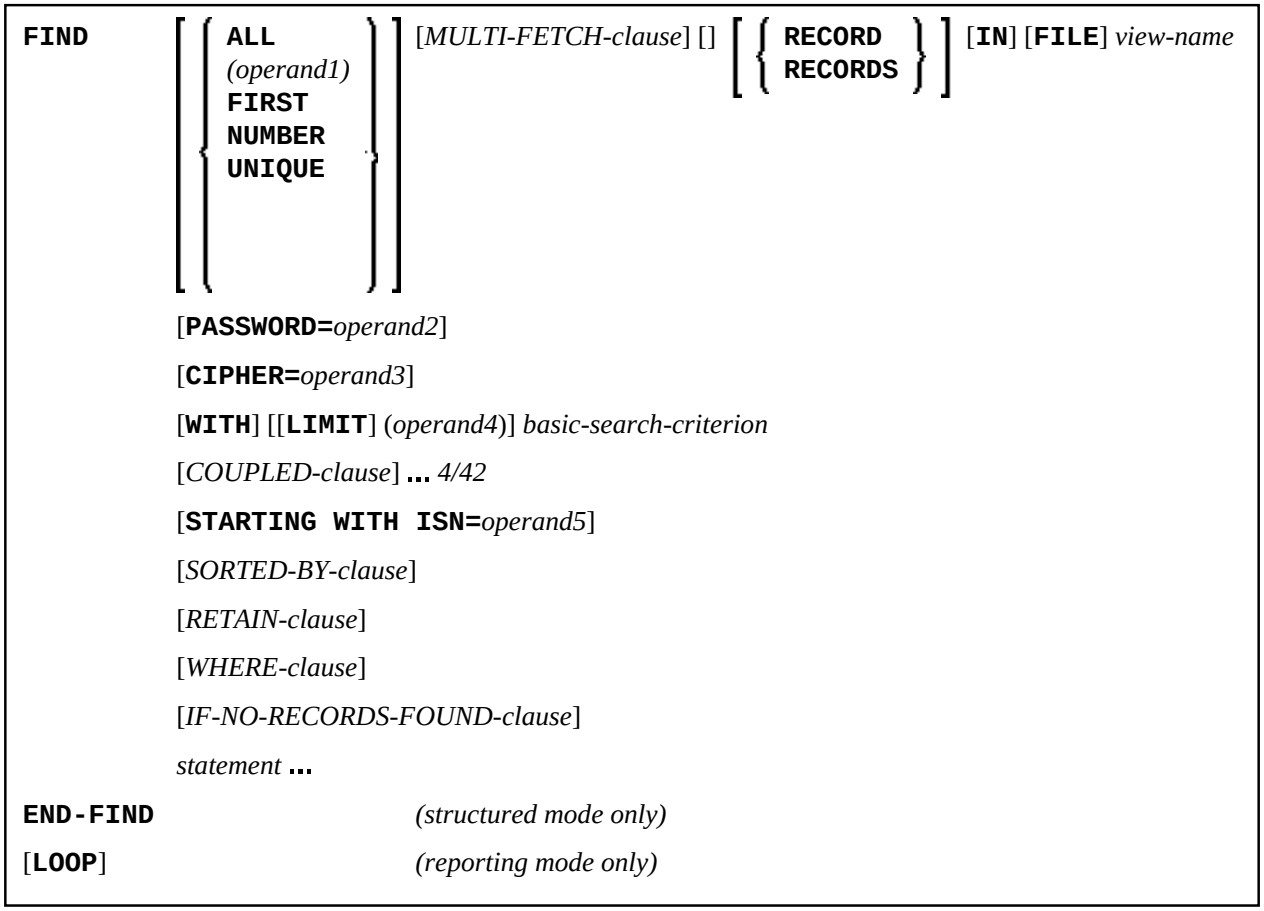
FIND



Dieses Kapitel behandelt folgende Themen:

- Funktion
- Einschränkungen
- Syntax-Beschreibung
- 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 | DELETE | END TRANSACTION | FIND | GET | GET SAME | GET TRANSACTION | HISTOGRAM | LIMIT | PASSW | PERFORM BREAK PROCESSING | READ | RETRY | STORE | UPDATE

Gehört zur Funktionsgruppe: *Datenbankzugriffe und Datenbankänderungen*

Funktion

Das Statement FIND dient dazu, Datensätze von der Datenbank auszuwählen, und zwar anhand eines Suchkriteriums, d.h. des Wertes eines Schlüsselfeldes (Deskriptors).

Mit dem FIND-Statement wird eine Verarbeitungsschleife initiiert, die für jeden gefundenen Datensatz durchlaufen wird. Innerhalb der FIND-Schleife kann jedes Feld eines gefundenen Datensatzes referenziert werden, und zwar ohne dass hierzu ein zusätzliches READ-Statement erforderlich wäre.

Siehe auch *FIND Statement* im *Leitfaden zur Programmierung*.

Datenbank-spezifische Anmerkungen

SQL	FIND FIRST sowie die PASSWORD-, CIPHER-, COUPLED- und RETAIN-Klauseln sind nicht erlaubt. FIND UNIQUE ist nicht erlaubt. (Ausnahme: Auf Großrechnern kann FIND UNIQUE für Primärschlüssel verwendet werden; allerdings ist dies nur aus Kompatibilitätsgründen erlaubt und sollte nicht benutzt werden.) Die SORTED BY-Klausel entspricht der SQL-Klausel ORDER BY. Das Suchkriterium (Basic-Search-Criterion) für eine SQL-Datenbank-Tabelle kann genauso angegeben werden wie für eine Adabas-Datei. Der Begriff *Datensatz* (Record) ist in diesem Zusammenhang dem SQL-Begriff *Reihe* (Row) gleichzusetzen.
XML	FIND FIRST sowie die PASSWORD-, CIPHER-, COUPLED- und RETAIN-Klauseln sind nicht erlaubt. FIND UNIQUE ist nicht erlaubt. Das Suchkriterium (Basic-Search-Criterion) für eine XML-Datenbank kann genauso angegeben werden wie für eine Adabas-Datei. Der Begriff *Datensatz* (Record) ist in diesem Zusammenhang dem SQL-Begriff *XML-Objekt* (XML Object) gleichzusetzen.

Systemvariablen beim FIND-Statement

Die Natural-Systemvariablen *ISN, *NUMBER und *COUNTER werden automatisch für jedes FIND-Statement erzeugt. Wird eine Systemvariable außerhalb der aktuellen Verarbeitungsschleife oder über ein FIND FIRST-, FIND NUMBER- oder FIND UNIQUE-Statement referenziert, muss mittels Statement-Label oder Sourcecode-Zeilennummer referenziert werden. Alle drei Systemvariablen haben Format/Länge P10; diese(s) Format/Länge kann nicht geändert werden.

***ISN**	Adabas	Bei Adabas-Datenbanken enthält `*ISN` die Adabas-ISN (Interne Satz-Nummer) des gerade verarbeiteten Datensatzes. `*ISN` kann nicht bei `FIND NUMBER` verwendet werden.
	Tamino	`*ISN` enthält die XML-Objekt-ID.
	SQL	`*ISN` ist nicht verfügbar.
	Entire System Server	`*ISN` ist nicht verfügbar.
***NUMBER**	Adabas	`*NUMBER` enthält die Anzahl der Datensätze, die das in der `WITH`-Klausel angegebene primäre Suchkriterium erfüllt haben.
	Tamino	Siehe **NUMBER für SQL Databases* in the *Systemvariablen*-Dokumentation.
	Entire System Server	`*NUMBER` ist nicht verfügbar.
***COUNTER**	`*COUNTER` enthält die Anzahl, wie oft die Verarbeitungsschleife durchlaufen worden ist.	

Siehe auch *Beispiel 13 - Sytemvariablen mit dem FIND-Statement benutzen*.

Mehrere FIND-Statements

Es ist möglich, mehrere `FIND`-Schleifen ineinander zu verschachteln. Hierbei wird eine jeweils innere Schleife für jeden Datensatz, der mit der jeweils äußeren Schleife ausgewählt wurde, durchlaufen. Siehe auch *Beispiel 14 – Mehrere FIND-Statements*.

Einschränkungen

Bei Entire System Server sind `FIND NUMBER` und `FIND UNIQUE` sowie die Klauseln `PASSWORD`, `CIPHER`, `COUPLED` und `RETAIN` nicht zulässig.

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
operand5	C	S					N	P	I		B*						ja	nein

* Format B von *operand1*, *operand4* und *operand5* kann nur mit einer Länge von kleiner oder gleich 4 benutzt werden.

Syntax-Element-Beschreibung:

ALL/*operand1*	**Begrenzung der Verarbeitung:**
	
	Sie können die Anzahl der ausgewählten Datensätze, die in der `FIND`-Schleife verarbeitet werden sollen, auf eine bestimmte Zahl begrenzen. Diese Zahl (*operand1*) geben Sie in Klammern unmittelbar nach dem Schlüsselwort `FIND` an, und zwar entweder als numerische Konstante (im Bereich von `0` bis `4294967295`) oder in Form einer numerischen Variablen.
	
	Andernfalls werden alle gefundenen Sätze weiterverarbeitet, was Sie zusätzlich durch das Schlüsselwort `ALL` hervorheben können.
	
	Geben Sie mit *operand1* ein Limit an, so gilt dieses Limit für die initiierte `FIND`-Schleife, wobei allerdings Datensätze, die aufgrund einer `WHERE`-Klausel abgelehnt werden, nicht mitgezählt werden.
	
	`FIND (5) IN EMPLOYEES WITH ...`
	
	`MOVE 10 TO #CNT(N2)`
	`FIND (#CNT) EMPLOYEES WITH ...`
	
	Das angegebene Limit hat für dieses Statement Vorrang vor einem mit einem `LIMIT`-Statement gesetzten Limit.
	
	Ist mit dem `LT`-Parameter ein kleineres Limit gesetzt, so gilt das `LT`-Limit.
	
	Anmerkungen:
	
	1. Wenn Sie eine vierstellige Anzahl von Datensätzen verarbeiten 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* hat keinen Einfluss auf die Größe eines ISN-Set, der mittels einer `RETAIN`-Klausel erzeugt wird. *operand1* wird zu Beginn des ersten `FIND`-Schleifendurchlaufs ausgewertet. Wird der Wert von *operand1* innerhalb der `FIND`-Schleife geändert, hat dies keine Auswirkungen auf die Anzahl der verarbeiteten Datensätze.

FIND FIRST | FIND NUMBER | FIND UNIQUE	**Suchoptionen:** Diese Optionen dienen dazu • nur den ersten der gefundenen Datensätze auszuwählen (*FIND FIRST*), • die Anzahl der gefundenen Datensätze zu ermitteln (*FIND NUMBER*), • bzw. sicherzustellen, dass nur ein Datensatz das Suchkriterium erfüllt (*FIND UNIQUE*). Näheres hierzu finden Sie in den entsprechenden Abschnitten weiter unten.
MULTI-FETCH-clause	**Multi-Fetch-Klausel:** Bei Adabas-Datenbanken: Im Standard-Modus liest Natural nicht mehrere Datensätze auf einmal bei einem einzigen Datenbankaufruf, sondern jeweils nur einen. Diese stabile Betriebsart kann aber einige Zeit in Anspruch nehmen, wenn eine größere Anzahl von Datenbank- Sätzen verarbeitet wird. Um die Leistung dieser Programme zu verbessern, bietet Natural die `MULTI-FETCH`-Klausel, die es ermöglicht, mehr als einen Datensatz pro Datenbankzugriff zu lesen. Weitere Informationen, siehe *MULTI-FETCH-Klausel* weiter unten.
view-name	**View-Angabe:** Der Name eines Views, der entweder in einem `DEFINE DATA`-Statement-Block oder in einer separaten 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=*operand2*	**PASSWORD-Klausel:** Die PASSWORD-Klausel gilt nur für Zugriffe auf Adabas-Datenbanken. Mit Entire System Server ist diese Klausel nicht erlaubt. Die PASSWORD-Klausel dient dazu, ein Passwort (*operand2*) anzugeben, um auf Daten einer passwortgeschützten Adabas-Datei zugreifen zu können. Wollen Sie auf eine passwortgeschützte Datei zugreifen, sollten Sie sich bezüglich des Passwortes mit Ihrem Datenbank-Security-Administrator in Verbindung setzen. Wenn das Passwort als Konstante angegeben wird, sollte die PASSWORD-Klausel ganz am Anfang einer Sourcecode-Zeile stehen und es sollte sich zwischen dem Schlüsselwort PASSWORD und dem Gleichheitszeichen kein Leerzeichen befinden; dadurch ist gewährleistet, dass das Passwort im Sourcecode nicht sichtbar ist. Im TP-Modus können Sie die PASSWORD-Klausel unsichtbar machen, indem Sie dann zuerst das Terminalkommando %* eingeben, bevor Sie das Passwort eintippen. Wenn Sie die PASSWORD-Klausel weglassen, gilt das mit dem PASSW-Statement angegebene Passwort. Während der Ausführung einer Verarbeitungsschleife kann das Passwort nicht geändert werden. Siehe auch *Beispiel 1 – PASSWORD-Klausel*.

CIPHER=*operand3*	**CIPHER-Klausel:** Die CIPHER-Klausel gilt nur für Zugriffe auf Adabas-Dateien. Mit Entire System Server ist diese Klausel nicht erlaubt. Die `CIPHER`-Klausel dient dazu, einen Chiffrierschlüssel (*operand3*) anzugeben, um in chiffrierter Form gespeicherte Daten von Adabas-Dateien in entschlüsselter Form zu erhalten. Wollen Sie auf eine chiffrierte Datei zugreifen, sollten Sie sich bezüglich des Chiffrierschlüssels mit Ihrem Datenbank-Security-Administrator in Verbindung setzen. Der Chiffrierschlüssel kann als numerische Konstante (8 Stellen lang) oder als Inhalt einer Benutzervariablen (Format/Länge N8) angegeben werden. Wird der Chiffrierschlüssel als Konstante angegeben, sollte die `CIPHER`-Klausel ganz am Anfang einer Sourcecode-Zeile stehen; dadurch ist gewährleistet, dass der Chiffrierschlüssel im Sourcecode nicht sichtbar ist. Im TP-Modus können Sie die `CIPHER`-Klausel unsichtbar machen, indem Sie zuerst das Terminalkommando `%*` eingeben, bevor Sie den Chiffrierschlüssel eintippen. Während der Ausführung der `FIND`-Verarbeitungsschleife kann der Chiffrierschlüssel nicht geändert werden. Siehe auch *Beispiel 2 – CIPHER-Klausel*.

WITH LIMIT ***operand4*** ***basic-search-criterion***	**WITH-Klausel:** Die `WITH`-Klausel ist unbedingt erforderlich. Mit ihr wird das Suchkriterium (basic-search-criterion) in Form des Wertes eines als Schlüsselfeld (Deskriptor) definierten Datenbankfeldes angegeben. Siehe *Suchkriterien für Adabas-Dateien*. Datenbank-spezifischer Hinweis: Sie können in der `WITH`-Klausel einen Deskriptor, einen Subdeskriptor, einen Superdeskriptor, einen Hyperdeskriptor oder einen phonetischen Deskriptor angeben. Auf Großrechnern kann auch ein Nicht-Deskriptor (d.h. ein Feld, das im DDM mit `N` markiert ist) angegeben werden. Die Anzahl der Datensätze, die anhand des in der `WITH`-Klausel definierten Suchkriteriums ausgewählt werden sollen, können Sie begrenzen, indem Sie das Schlüsselwort `LIMIT` und dahinter in Klammern eine Zahl oder eine Benutzervariable angeben (*operand4*). Übersteigt die Anzahl der ausgewählten Datensätze diese Zahl, wird das Programm mit einer entsprechenden Fehlermeldung abgebrochen. **Anmerkung:** Wenn das Limit eine vierstellige Zahl sein soll, geben Sie diese mit einer vorangestellten Null an: (`0`*nnnn*); denn Natural interpretiert jede vierstellige Zahl in Klammern als Zeilennummer-Referenzierung auf ein Statement.
COUPLED-clause	**COUPLED-Klausel:** Diese Klausel gilt nur für Such-Zugriffe auf Adabas-Dateien. Siehe *COUPLED-Klausel*.
STARTING WITH ISN=*operand5*	**STARTING WITH-Klausel:** Diese Klausel kann zum Repositionieren innerhalb einer `FIND`-Schleife, deren Verarbeitung unterbrochen wurde, benutzt werden. Siehe *STARTING WITH-Klausel*.
SORTED-BY-clause	**SORTED BY-Klausel:** Diese Klausel dient dazu, die ausgewählten Datensätze in der Reihenfolge der Werte eines oder mehrerer (maximal 3) Deskriptoren zu sortieren. Siehe *SORTED BY-Klausel*.

RETAIN-clause	**RETAIN-Klausel:** Mit dieser Klausel ist es möglich, das Ergebnis einer ausgedehnten Suche in einer großen Datei für die weitere Verarbeitung zurückzustellen. Siehe *RETAIN-Klausel*.
WHERE-clause	**WHERE-Klausel:** Diese Klausel dient dazu, ein zusätzliches Selektionskriterium (*logical-condition*) anzugeben. Siehe *WHERE-Klausel*.
IF-NO-RECORDS-FOUND-clause	**IF NO RECORDS FOUND-Klausel:** In dieser Klausel können Sie eine Schleife angeben, die mit einem `FIND`-Statement ausgeführt werden soll für den Fall, dass kein Datensatz die in der `WITH`- und `WHERE`-Klausel des `FIND`-Statements angegebenen Selektionskriterien erfüllt. Siehe *IF NO RECORDS FOUND-Klausel*.
END-FIND	Das für Natural reservierte Schlüsselwort `END-FIND` muss zum Beenden des `FIND`-Statements verwendet werden.

FIND FIRST

Das `FIND FIRST`-Statement dient dazu, den ersten Datensatz, der die `WITH`- und `WHERE`-Selektionskriterien erfüllt, auszuwählen und zu verarbeiten.

Bei Adabas-Dateien wird derjenige der ausgewählten Datensätze verarbeitet, der die niedrigste Adabas-ISN (Interne Satznummer) hat.

`FIND FIRST` initiiert *keine* Verarbeitungsschleife.

Einschränkungen bei FIND FIRST

- `FIND FIRST` darf nur im Reporting Mode verwendet werden.
- `FIND FIRST` ist beim Zugriff auf SQL-Datenbanken nicht möglich.
- Ein `FIND FIRST`-Statement darf keine `IF NO RECORDS FOUND`-Klausel enthalten.

Systemvariablen bei FIND FIRST

Beim FIND FIRST-Statement stehen folgende Natural-Systemvariablen zur Verfügung:

***ISN**	Die Systemvariable `*ISN` enthält die Adabas-ISN (Interne Satznummer) des ausgewählten Datensatzes. `*ISN` enthält den Wert Null (0), wenn kein Datensatz die `WITH`- und `WHERE`-Selektionskriterien erfüllt. Mit Entire System Server kann `*ISN` nicht verwendet werden.
***NUMBER**	Die Systemvariable `*NUMBER` enthält die Anzahl der Datensätze, die vor Auswertung des `WHERE`-Kriteriums das in der `WITH`-Klausel angegebene primäre Selektionskriterium erfüllt haben. `*NUMBER` enthält den Wert Null (0), wenn kein Datensatz das `WITH`-Kriterium erfüllt. Mit Entire System Server kann `*NUMBER` nicht verwendet werden.
***COUNTER**	Die Systemvariable `*COUNTER` enthält `1`, wenn ein Datensatz gefunden wurde, sie enthält `0`, wenn kein Datensatz gefunden wurde.

Beispiel für `FIND FIRST` siehe Programm `FNDFIR` (Reporting Mode).

FIND NUMBER

Mit dem Statement `FIND NUMBER` können Sie ermitteln, wieviele Datensätze die `WITH`- und `WHERE`-Kriterien erfüllen. `FIND NUMBER` löst keine Verarbeitungsschleife aus *und liest auch keine Datensätze von der Datenbank*.

Anmerkung:
Eine `WHERE`-Klausel kann hierbei einen beträchtlichen Verarbeitungsmehraufwand verursachen.

Einschränkungen bei FIND NUMBER

- Ein `FIND NUMBER`-Statement darf keine `SORTED BY`- oder `IF NO RECORDS FOUND`-Klausel enthalten.
- Im Structured Mode darf keine `WHERE`-Klausel benutzt werden.
- Mit Entire System Server kann `FIND NUMBER` nicht verwendet werden.

Systemvariablen bei FIND NUMBER

Bei `FIND NUMBER` stehen folgende Natural-Systemvariablen zur Verfügung:

***NUMBER**	Die Systemvariable `*NUMBER` enthält die Anzahl der Datensätze, die das in der `WITH`-Klausel angegebene primäre Selektionskriterium erfüllt haben.
***COUNTER**	Die Systemvariable `*COUNTER` enthält die Anzahl der Datensätze, die die Selektionskriterien der `WHERE`-Klausel erfüllt haben. `*COUNTER` steht nur zur Verfügung, wenn das `FIND NUMBER`-Statement eine `WHERE`-Klausel enthält.

Beispiel für `FIND NUMBER` siehe das Programm `FNDNUM` (Reporting Mode).

FIND UNIQUE

Dieses Statement gewährleistet, dass nur ein einziger Datensatz die Selektionskriterien erfüllt. FIND UNIQUE initiiert keine Verarbeitungsschleife. Enthält das FIND UNIQUE-Statement eine WHERE-Klausel, wird zur Auswertung dieser Klausel eine automatische interne Verarbeitungsschleife durchlaufen.

Wird kein oder mehr als ein Datensatz gefunden, wird eine entsprechende Fehlermeldung ausgegeben; dieser Fall kann mit einem ON ERROR-Statement abgefangen werden.

Einschränkungen bei FIND UNIQUE

- FIND UNIQUE darf nur im Reporting Mode verwendet werden.
- FIND UNIQUE ist beim Einsatz von Entire System Server nicht möglich.
- Bei SQL-Datenbanken ist FIND UNIQUE nicht erlaubt. (Ausnahme: auf Großrechnern kann FIND UNIQUE für Primärschlüssel verwendet werden; allerdings ist dies nur aus Kompatibilitätsgründen erlaubt und sollte nicht benutzt werden.)
- Ein FIND UNIQUE-Statement darf keine SORTED BY- oder IF NO RECORDS FOUND-Klausel enthalten.

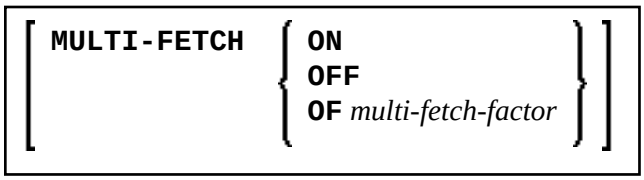
Systemvariablen bei FIND UNIQUE

***ISN**	Die Systemvariable *ISN enthält die eindeutige ISN-Nummer des Datensatzes, der selbst wiederum eindeutig sein muss.
***NUMBER**	Die Systemvariable *NUMBER enthält bei einer gültigen Ausführung des FIND UNIQUE-Statements immer eine 1. *NUMBER kann einen anderen positiven Wert (=0 oder >=2) enthalten, wenn ein Fehler aufgetreten ist. Diese Fehlerbedingung kann vom ON ERROR-Statement benutzt werden. *NUMBER ist nicht zulässig, wenn die WHERE-Klausel fehlt.
***COUNTER**	Die Systemvariable *COUNTER enthält die Anzahl der Datensätze nach Auswertung des WHERE-Kriteriums. *COUNTER ist nicht zulässig, wenn die WHERE-Klausel fehlt.

Beispiel für FIND UNIQUE siehe Programm FNDUNQ (Reporting Mode).

MULTI-FETCH-Klausel

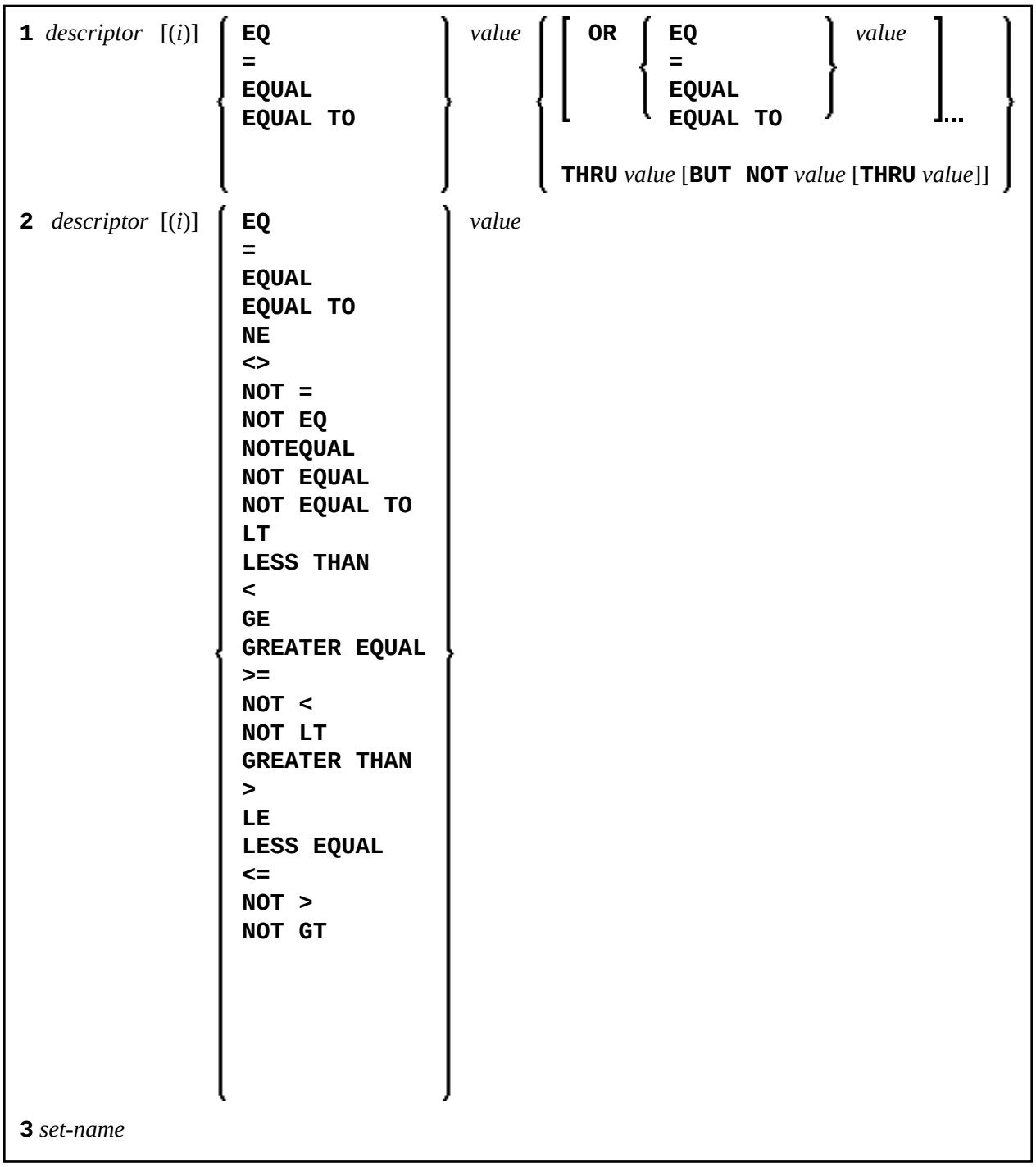
Anmerkung:
Diese Klausel kann nur bei Adabas-Datenbanken benutzt werden.



Anmerkung:
[MULTI-FETCH OF *multi-fetch-factor*] wird bei den Datenbanktypen ADA und ADA2 nicht ausgewertet. Es erfolgt die Standardverarbeitung (siehe Profilparameter MFSET. Beim Datenbanktyp ADA2 wird die MULTI-FETCH-Klausel komplett ignoriert; siehe *Database Management System Assignments* in der *Configuration Utility*-Dokumentation.

Weitere Informationen siehe *Multi-Fetch-Klausel* (Adabas) im *Leitfaden zur Programmierung*.

Suchkriterium bei Adabas-Dateien (*basic-search-criterion*)



Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate													Referenzierung erlaubt	Dynam. Definition
descriptor		S	A			A	U	N	P	I	F	B	D	T	L				nein	nein
value	C	S				A	U	N	P	I	F	B	D	T	L				ja	nein
set-name	C	S				A													nein	nein

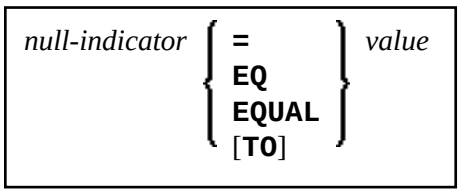
Syntax-Element-Beschreibung:

descriptor	**Deskriptor-Feld:** Es kann ein Adabas-Deskriptor, -Subdeskriptor, -Superdeskriptor, -Hyperdeskriptor oder phonetischer Deskriptor angegeben werden. Es kann auch ein im DDM als Nicht-Deskriptor markiertes Feld angegeben werden.
(*i*)	**Index:** Ein Deskriptorfeld, das Teil einer Periodengruppe ist, kann mit oder ohne Index angegeben werden. Wird kein Index angegeben, so wird ein Datensatz ausgewählt, wenn der Suchwert in einer beliebigen Ausprägung gefunden wird. Wird ein Index angegeben, so wird ein Datensatz nur ausgewählt, wenn der Suchwert in der im Index angegebenen Ausprägung gefunden wird. Der Index muss als Konstante angegeben werden; es darf kein Indexbereich angegeben werden. Ist das angegebene Deskriptorfeld ein multiples Feld, darf kein Index angegeben werden. Ein Datensatz wird ausgewählt unabhängig davon, in welcher Ausprägung des Feldes der Suchwert gefunden wird.
value	**Suchwert:** Der Suchwert, den das Deskriptorfeld haben soll. Die Formate von Suchwert und Deskriptorfeld müssen kompatibel sein.
set-name	**Set-Name:** Identifiziert einen Set von Datensätzen, die mit einem `FIND`-Statement ausgewählt wurden, das eine `RETAIN`-Klausel enthielt. Der referenzierte Set muss von derselben physischen Adabas-Datei ausgewählt worden sein. *set-name* kann entweder als Textkonstante (bis zu 32 Zeichen lang) oder in Form einer alphanumerischen Variablen angegeben werden. Mit Entire System Server kann *set-name* nicht angegeben werden.

Siehe auch:

- *Beispiel 3 – Basis-Suchkriterium in WITH-Klausel*
- *Beispiel 4 – Basis-Suchkriterium mit multiplem Feld*

Suchkriterium mit Null-Indikator (*basic-search-criterion*)



Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
null-indicator		S							I								nein	nein
value	C	S					N	P	I	F	B						ja	nein

Syntax-Element-Beschreibung:

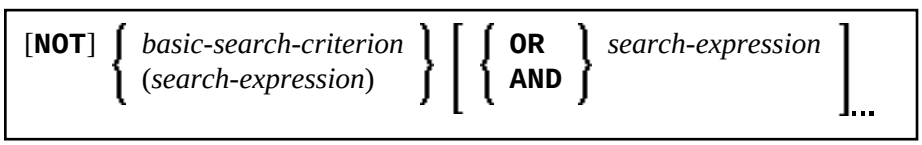
null-indicator	Der Null-Indikator.	
value	Möglicher Wert:	
	`-1`	Das entsprechende Feld enthält keinen Wert.
	`0`	Das entsprechende Feld enthält einen Wert.

Verknüpfen von Suchkriterien (für Adabas-Dateien)

Es ist möglich, mehrere Suchkriterien mit den Boole'schen Operatoren `AND`, `OR` und `NOT` miteinander zu verknüpfen. Mit Klammern kann außerdem die Reihenfolge der Kriterienauswertung gesteuert werden. Die Auswertung verknüpfter Suchkriterien geschieht in folgender Reihenfolge:

1. ( ): Klammern
2. `NOT`: Negation (nur für *basic-search-criterion* der Form [2]).
3. `AND`: Und-Verknüpfung
4. `OR`: Oder-Verknüpfung

Suchkriterien können mit logischen Operatoren verknüpft werden, um einen komplexen Suchausdruck (*search-expression*) zu bilden. Eine solcher komplexer Suchausdruck hat folgende Syntax:



Siehe auch *Beispiel 5 - Mehrere Beispiele für komplexe Suchausdrücke in WITH-Klausel* .

Such-Schlüsselfelder (Deskriptoren) für Adabas-Dateien

Adabas-Benutzer können bei der Suche nach Datensätzen als Suchschlüssel Datenbankfelder verwenden, die als Deskriptoren definiert sind.

Subdeskriptoren, Superdeskriptoren, Hyperdeskriptoren und phonetische Deskriptoren

Bei Adabas-Datenbanken können für die Konstruktion von Suchkriterien Subdeskriptoren, Superdeskriptoren, Hyperdeskriptoren und phonetische Deskriptoren verwendet werden.

- Ein Subdeskriptor ist ein Schlüsselfeld, das Teil eines Feldes ist.
- Ein Superdeskriptor ist ein Schlüsselfeld, das aus mehreren Feldern oder Teilfeldern besteht.
- Ein Hyperdeskriptor ist ein Schlüsselfeld, das durch einen benutzerdefinierten Algorithmus gebildet wird.
- Ein phonetischer Deskriptor ermöglicht die Suche nach einem Feldwert (z.B. der Name einer Person) anhand des Klanges eines Wertes. Bei der Suche mit einem phonetischen Deskriptor werden alle Werte gefunden, die so ähnlich klingen wie der Suchwert.

Bei welcher Datei welche Felder als Deskriptoren, Sub-, Super-, Hyper- und phonetische Deskriptoren verwendet werden können, ist in dem betreffenden DDM definiert.

Werte für Subdeskriptoren, Superdeskriptoren, phonetische Deskriptoren

Die mit Subdeskriptoren, Superdeskriptoren und phonetischen Deskriptoren angegebenen Suchwerte müssen mit dem jeweiligen internen Format kompatibel sein: ein Subdeskriptor hat dasselbe interne Format wie das Feld, von dem er ein Teil ist; ein phonetischer Deskriptor hat immer alphanumerisches Format; das interne Format eines Superdeskriptors ist binär, falls alle Felder, aus denen er sich zusammensetzt, numerisches Format haben; andernfalls ist sein internes Format alphanumerisch.

Werte für Sub- und Superdeskriptoren können wie folgt angegeben werden:

- als numerische oder hexadezimale Konstanten; hat ein Superdeskriptor binäres Format (vgl. oben), muss eine numerische oder hexadezimale Konstante als Wert angegeben werden;
- als Werte von Benutzervariablen, die mit einem `REDEFINE`-Statement redefiniert wurden, um die Teile auszuwählen, die den Sub- bzw. Superdeskriptorwert darstellen.

Deskriptoren aus Datenbank-Arrays

Ein Deskriptor, der Teil eines Datenbank-Arrays ist, kann ebenfalls als Suchfeld verwendet werden. Bei Adabas-Dateien kann dies ein multiples Feld sein oder ein Feld, das in einer Periodengruppe enthalten ist.

Ein Deskriptorfeld, das Teil einer Periodengruppe ist, kann entweder mit oder ohne Index angegeben werden. Wird kein Index angegeben, so wird ein Datensatz ausgewählt, wenn der Suchwert in irgendeiner Ausprägung gefunden wird. Wird ein Index angegeben, so wird ein Datensatz nur ausgewählt, wenn der Suchwert in der im Index angegebenen Ausprägung gefunden wird. Der Index muss als Konstante angegeben werden. Es darf kein Indexbereich angegeben werden.

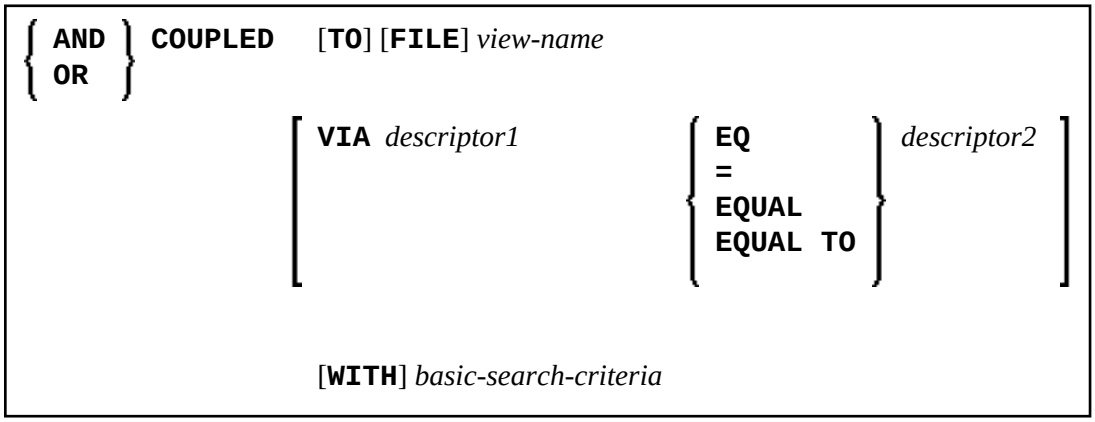
Ist das angegebene Deskriptorfeld ein multiples Feld, darf kein Index angegeben werden. Ein Datensatz wird ausgewählt unabhängig davon, in welcher Ausprägung des Feldes der Suchwert gefunden wird.

Siehe auch *Beispiel 5 - Mehrere Beispiele für komplexe Suchausdrücke in WITH-Klausel*.

COUPLED-Klausel

Diese Klausel gilt nur für Zugriffe auf Adabas-Dateien.

Mit Entire System Server darf diese Klausel nicht verwendet werden.



Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate											Referenzierung erlaubt	Dynam. Definition
descriptor1		S	A			A	N	P			B						nein	nein
descriptor2		S	A			A	N	P			B						nein	nein

Anmerkung:
Ohne `VIA`-Klausel kann die `COUPLED`-Klausel bis zu viermal angegeben werden, mit `VIA`-Klausel bis zu 42-mal.

Adabas bietet die Möglichkeit, Dateien miteinander zu koppeln. Dadurch ist es mit der `COUPLED`-Klausel möglich, im Suchkriterium eines einzigen `FIND`-Statements Deskriptoren von verschiedenen Dateien anzugeben.

Zwei verschiedene `COUPLED`-Klauseln desselben `FIND`-Statements dürfen nicht dieselbe Adabas-Datei verwenden.

Ein *set-name* (siehe *RETAIN-Klausel*) darf nicht im Suchkriterium (*basic-search-criteria*) angegeben werden.

Datenbankfelder der in der `COUPLED`-Klausel angegebenen Datei können anschließend im Programm nicht referenziert werden, wenn auf diese Datei kein separater `FIND`- oder `READ`-Zugriff erfolgt.

Anmerkung:
Wenn die `COUPLED`-Klausel verwendet wird, kann die Haupt-`WITH`-Klausel gegebenenfalls weggelassen werden. Wenn die Haupt-`WITH`-Klausel weggelassen wird, dürfen die Schlüsselwörter `AND`/`OR` in der

COUPLED-Klausel nicht angegeben werden.

Physisches Koppeln ohne VIA-Klausel

Die in der COUPLED-Klausel ohne VIA verwendeten Dateien müssen mit der entsprechenden Adabas-Utility physisch gekoppelte Adabas-Dateien sein (wie in der Adabas-Dokumentation beschrieben).

Siehe auch *Beispiel 7 – Physisch gekoppelte Dateien benutzen*.

Die Referenzierung von NAME im DISPLAY-Statement ist gültig, da dieses Feld in der Datei EMPLOYEES (Angestellte) enthalten ist; eine Referenzierung von MAKE (Fabrikat) hingegen wäre nicht gültig, da MAKE in der Datei VEHICLES (Fahrzeuge) enthalten ist, welche in der COUPLED-Klausel angegeben wurde.

In diesem Beispiel werden Datensätze nur gefunden, wenn die beiden Dateien EMPLOYEES und VEHICLES physisch gekoppelt sind.

Logisches Koppeln mit VIA-Klausel

Die Option VIA *descriptor1 = descriptor2* erlaubt es Ihnen, in einer Suchabfrage mehrere Adabas-Dateien logisch miteinander zu koppeln, dabei ist:

- *descriptor1* ein Feld aus dem ersten View.
- *descriptor2* ein Feld aus dem zweiten View.

Die beiden Dateien brauchen nicht physisch in Adabas gekoppelt zu sein. Diese COUPLED-Option nutzt die ab Adabas Version 5 gebotene Möglichkeit des Soft Coupling aus, die in der Adabas-Dokumentation beschrieben ist.

Siehe auch *Beispiel 8 – VIA-Klausel*.

STARTING WITH-Klausel

Diese Klausel gilt nur für Adabas-Datenbanken.

Sie können diese Klausel benutzen, um als *operand5* eine Adabas-ISN (Internal Sequence Number), die als ein Startwert für die Auswahl von Datensätzen benutzt werden soll.

Diese Klausel kann zum Repositionieren innerhalb einer FIND-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.

Anmerkung:
Als tatsächlicher Startwert wird nicht der Wert von *operand5*, sondern der nächsthöhere Wert genommen.

Beispiel:

Siehe Programm FNDSISN in Library SYSEXSYN.

SORTED BY-Klausel

Diese Klausel gilt nur für Zugriffe auf Adabas-, Tamino- und SQL-Datenbanken.

Mit Entire System Server darf diese Klausel nicht verwendet werden.

SORTED [**BY**] *descriptor* **...** *3* [**DESCENDING**]

Die SORTED BY-Klausel dient dazu, die ausgewählten Datensätze in der Reihenfolge der Werte eines oder mehrerer (maximal 3) Deskriptoren zu sortieren. Diese Deskriptoren brauchen nicht mit den als Suchkriterium verwendeten Deskriptoren identisch zu sein.

Normalerweise wird in *aufsteigender* Reihenfolge der Werte sortiert; wünschen Sie eine *absteigende* Sortierfolge, geben Sie das Schlüsselwort DESCENDING an. Der Sortiervorgang verwendet die Adabas-Invertierten-Listen, es werden hierbei keine Datensätze gelesen.

Anmerkung:
Diese Klausel kann beträchtliche zusätzliche Verarbeitungszeit beanspruchen, falls das zur Steuerung der Sortierfolge verwendete Deskriptorfeld viele Werte enthält. Das liegt daran, dass die gesamte Wertliste abgesucht werden muss, bis alle ausgewählten Datensätze lokalisiert worden sind. Wollen Sie eine große Zahl von Datensätzen sortieren, sollten Sie daher stattdessen das Statement SORT verwenden.

Bei der Verwendung einer SORTED BY-Klausel gelten die Adabas-Sortierlimits (siehe ADARUN-Parameter LS in der Adabas-Dokumentation).

In einer SORTED BY-Klausel darf kein Deskriptor, der Teil einer Periodengruppe ist, angegeben werden; ein multiples Feld (ohne Index) darf angegeben werden.

Auf Großrechnern dürfen in einer SORTED BY-Klausel keine Nicht-Deskriptoren angegeben werden.

Eine SORTED BY-Klausel darf nicht zusammen mit einer RETAIN-Klausel verwendet werden.

Siehe auch *Beispiel 9 – SORTED BY-Klausel*.

Hinweis zur gemeinsamen Verwendung der Klauseln STARTING WITH und SORTED BY

Wenn sowohl die STARTING WITH- als auch die SORTED BY-Klausel im selben FIND-Statement verwendet werden und die darunterliegende Datenbank Adabas ist, ist Folgendes zu beachten:

Bei Adabas für Großrechner

Bei Adabas für Großrechner wird das FIND-Statement in den folgenden Schritten ausgeführt:

1. Alle Datensätze, auf die das Suchkriterium zutriff, werden gesammelt und in die ISN-Reihenfolge gebracht.

2. Die Datensätze werden nach dem in der SORTED BY-Klausel angegeben Deskriptor sortiert.

3. Der Datensatz, dessen ISN-Wert in der STARTING WITH-Klausel angegeben ist, wird in die nach Deskriptor sortierte Datensatzliste platziert.

4. Die auf Datensätze, die nach dem unter Schritt 3 gefundenen Datensatz kommen, werden in der FIND-Schleife zurückgegeben.

Bei Adabas für OpenSystems

Bei Adabas für OpenSystems (UNIX, Windows, OpenVMS) wird dasselbe Statement wie folgt ausgeführt:

1. Alle Datensätze, auf die das Suchkriterium zutrifft, werden gesammelt und in die ISN-Reihenfolge gebracht.

2. Der Datensatz, dessen ISN-Wert in der STARTING WITH-Klausel angegeben ist, wird in die nach ISN sortierte Datensatzliste platziert.

3. Die Datensätze, die nach dem unter Schritt 2 gefundenen Datensatz kommen, werden nach dem in der SORTED BY-Klausel angegebenen Deskriptor sortiert und in der FIND-Schleife zurückgegeben.

Beispiel:

Wenn man das folgende Programm mit Adabas Version 8 für Großrechner und Adabas Version 6.1 für UNIX/OpenVMS/Windows ausführt,

```
DEFINE DATA LOCAL
1 V1 VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
1 #ISN (I4)
END-DEFINE
FORMAT NL=5 SG=OFF PS=43 AL=15
*
PRINT   'FIND' (I)
FIND V1 WITH NAME = 'B' THRU 'BALBIN'
  RETAIN AS 'SET1'
  IF *COUNTER = 4 THEN
    #ISN := *ISN
  END-IF
  DISPLAY *ISN V1
END-FIND
*
PRINT / 'FIND .. SORTED BY NAME' (I)
FIND V1 WITH 'SET1'
  SORTED BY NAME
  DISPLAY *ISN V1
END-FIND
*
PRINT / 'FIND .. STARTING WITH ISN = ' (I) #ISN (AD=I)
FIND V1 WITH 'SET1'
  STARTING WITH ISN = #ISN
  DISPLAY *ISN V1
END-FIND
*
PRINT / 'FIND .. STARTING WITH ISN = ' (I) #ISN (AD=I)
```

```
       ' .. SORTED BY NAME' (I)
FIND V1 WITH 'SET1'
  STARTING WITH ISN = #ISN
  SORTED BY NAME
  DISPLAY *ISN V1
END-FIND
END
```

erhält man folgendes Ergebnis:

Ergebnis bei Natural für Großrechner (Adabas Version 8)

```
ISN        NAME          FIRST-NAME          CITY
----- --------------- --------------- ---------------

FIND V1 WITH NAME = 'B' THRU 'BALBIN'
   12 BAILLET         PATRICK         LYS LEZ LANNOY
   58 BAGAZJA         MARJAN          MONTHERME
  351 BAECKER         JOHANNES        FRANKFURT
  355 BAECKER         KARL            SINDELFINGEN
  370 BACHMANN        HANS            MUENCHEN
  490 BALBIN          ENRIQUE         BARCELONA
  650 BAKER           SYLVIA          OAK BROOK
  913 BAKER           PAULINE         DERBY

FIND .. SORTED BY NAME
  370 BACHMANN        HANS            MUENCHEN
  351 BAECKER         JOHANNES        FRANKFURT
  355 BAECKER         KARL            SINDELFINGEN
   58 BAGAZJA         MARJAN          MONTHERME
   12 BAILLET         PATRICK         LYS LEZ LANNOY
  650 BAKER           SYLVIA          OAK BROOK
  913 BAKER           PAULINE         DERBY
  490 BALBIN          ENRIQUE         BARCELONA

FIND .. STARTING WITH ISN = 355
  370 BACHMANN        HANS            MUENCHEN
  490 BALBIN          ENRIQUE         BARCELONA
  650 BAKER           SYLVIA          OAK BROOK
  913 BAKER           PAULINE         DERBY

FIND .. STARTING WITH ISN = 355 .. SORTED BY NAME
   58 BAGAZJA         MARJAN          MONTHERME
   12 BAILLET         PATRICK         LYS LEZ LANNOY
  650 BAKER           SYLVIA          OAK BROOK
  913 BAKER           PAULINE         DERBY
  490 BALBIN          ENRIQUE         BARCELONA
```

Ergebnis bei Natural für OpenSystems (Adabas Version 6.1)

```
ISN        NAME          FIRST-NAME          CITY
----- --------------- --------------- ---------------

FIND V1 WITH NAME = 'B' THRU 'BALBIN'
   12 BAILLET         PATRICK         LYS LEZ LANNOY
   58 BAGAZJA         MARJAN          MONTHERME
  351 BAECKER         JOHANNES        FRANKFURT
  355 BAECKER         KARL            SINDELFINGEN
  370 BACHMANN        HANS            MUENCHEN
  490 BALBIN          ENRIQUE         BARCELONA
  650 BAKER           SYLVIA          OAK BROOK
  913 BAKER           PAULINE         DERBY

FIND .. SORTED BY NAME
  370 BACHMANN        HANS            MUENCHEN
  351 BAECKER         JOHANNES        FRANKFURT
  355 BAECKER         KARL            SINDELFINGEN
   58 BAGAZJA         MARJAN          MONTHERME
   12 BAILLET         PATRICK         LYS LEZ LANNOY
  650 BAKER           SYLVIA          OAK BROOK
  913 BAKER           PAULINE         DERBY
  490 BALBIN          ENRIQUE         BARCELONA

FIND .. STARTING WITH ISN = 355
  370 BACHMANN        HANS            MUENCHEN
  490 BALBIN          ENRIQUE         BARCELONA
  650 BAKER           SYLVIA          OAK BROOK
  913 BAKER           PAULINE         DERBY

FIND .. STARTING WITH ISN = 355 .. SORTED BY NAME
  370 BACHMANN        HANS            MUENCHEN
  650 BAKER           SYLVIA          OAK BROOK
  913 BAKER           PAULINE         DERBY
  490 BALBIN          ENRIQUE         BARCELONA
```

Ein `FIND`-Statement mit höchstens einer dieser Optionen (`SORTED BY` oder `STARTING WITH ISN`) liefert immer dieselben Datensätze in derselben Reihenfolge, egal unter welchem System das Statement ausgeführt wird. Werden jedoch die beiden Klauseln zusammen benutzt, ist das Ergebnis davon abhängig, auf welcher Plattform das Datenbank-Statement von Adabas bedient wird.

Wenn Sie beabsichtigen, ein solches Natural-Programm auf mehreren Plattformen einzusetzen, sollten Sie deshalb die Kombination der Klauseln `SORTED BY` und `STARTING WITH ISN` im selben `FIND`-Statement vermeiden.

RETAIN-Klausel

Diese Klausel gilt nur für Zugriffe auf Adabas-Datenbanken.

Mit Entire System Server darf diese Klausel nicht verwendet werden.

RETAIN AS *operand6*

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition
operand6	C	S				A										ja	nein

Syntax-Element-Beschreibung:

RETAIN AS	**Suchergebnis für weitere Verarbeitung bereit stellen:** Mit der `RETAIN`-Klausel ist es möglich, das Ergebnis einer ausgedehnten Suche in einer großen Datei für die weitere Verarbeitung zurückzustellen. Die ausgewählten Datensätze werden als ISN-Set in die Adabas-Arbeitsdatei gestellt. Dieser Set kann in nachfolgenden `FIND`-Statements als Suchkriterium (*basic-search-criterion*) angegeben werden, um aus dem Set eine gezieltere Auswahl von Datensätzen für die weitere Verarbeitung zu treffen. Der Set ist dateispezifisch und kann nur von einem anderen `FIND`-Statement verwendet werden, das dieselbe Datei verarbeitet. Der Set kann von einem beliebigen anderen Natural-Programm referenziert werden.
operand6	**Set-Name:** Der Set-Name identifiziert den ISN-Set. Der Name kann als alphanumerische Konstante oder in Form einer alphanumerischen Benutzervariablen angegeben werden. Es wird nicht geprüft, ob ein Set-Name bereits vergeben ist; wird ein Set-Name zweimal vergeben, überschreibt der neue Set den alten.

Siehe auch *Beispiel 10 - RETAIN-Klausel*.

Freigabe von Sets

Die Anzahl der Sets, die zurückgestellt werden können, ist nicht begrenzt; die Anzahl der ISNs pro Set auch nicht. Die zu einer gegebenen Zeit benötigte Mindestanzahl von ISN-Sets sollte definiert werden. Nicht länger benötigte Sets sollten mit einem `RELEASE SETS`-Statement aus der Arbeitsdatei gelöscht werden.

Wenn sie nicht von einem `RELEASE`-Statement freigegeben werden, bleiben erstellte ISN-Sets während der gesamten Natural-Session erhalten und werden nicht automatisch gelöscht, außer bei einem Library-Wechsel. Ein mit einem Programm erstellter Set kann von einem anderen Programm referenziert und verarbeitet oder unter Angabe zusätzlicher Suchkriterien weiter selektiert werden.

Zugriffe anderer Benutzer

Die Datensätze, deren ISNs in einem ISN-Set enthalten sind, sind nicht vor Zugriff/Veränderung durch andere Benutzer geschützt. Bevor Sie Datensätze aus dem Set verarbeiten, empfiehlt es sich daher sicherzustellen, dass das ursprüngliche Selektionskriterium, mit dem der Set erstellt wurde, immer noch auf die ausgewählten Datensätze zutrifft.

Dies geschieht mit einem neuen FIND-Statement, bei dem man den Set-Namen in der WITH-Klausel als primäres Suchkriterium angibt und in einer WHERE-Klausel das ursprüngliche Primärsuchkriterium (d.h. das Suchkriterium aus der WITH-Klausel des FIND-Statements, mittels dessen der Set erstellt wurde).

Einschränkung

Eine RETAIN-Klausel darf nicht zusammen mit einer SORTED BY-Klausel verwendet werden.

WHERE-Klausel

```
WHERE logical-condition
```

Die WHERE-Klausel dient dazu, ein zusätzliches Selektionskriterium (*logical-condition*) anzugeben, das ausgewertet wird, *nachdem* ein über die WITH-Klausel ausgewählter Datensatz gelesen wurde und *bevor* ein ausgewählter Datensatz weiter verarbeitet wird (einschließlich AT BREAK-Auswertung).

Die für logische Bedingungen gültige Syntax ist im Abschnitt *Logische Bedingungen* im *Leitfaden zur Programmierung* erklärt.

Ist für das FIND-Statement die Anzahl der zu verarbeitenden Datensätze durch ein Limit begrenzt, so werden Datensätze, die aufgrund einer WHERE-Klausel *nicht* weiterverarbeitet werden, bei der Ermittlung des Limits nicht mitgezählt. Sie werden allerdings bei der Ermittlung eines auf allgemeinerer Ebene gesetzten Limits (Session-Parameter LT, GLOBALS-Kommando oder LIMIT-Statement) mitgezählt.

Siehe auch *Beispiel 11 - WHERE-Klausel*.

IF NO RECORDS FOUND-Klausel

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 Verarbeitung angeben, die ausgeführt werden soll für den Fall, dass kein Datensatz die in der WITH- und WHERE-Klausel des FIND-Statements angegebenen Selektionskriterien erfüllt.

Ist dies der Fall, 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.

Siehe auch *Beispiel 12 - IF NO RECORDS FOUND-Klausel*.

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.

Einschränkung

Bei FIND FIRST, FIND NUMBER und FIND UNIQUE kann diese Klausel nicht verwendet werden.

Beispiele

- Beispiel 1 — PASSWORD-Klausel
- Beispiel 2 — CIPHER-Klausel
- Beispiel 3 — Basis-Suchkriterium in WITH-Klausel
- Beispiel 4 — Basis-Suchkriterium mit multiplem Feld
- Beispiel 5 — Mehrere Beispiele für komplexe Suchausdrücke in WITH-Klausel
- Beispiel 6 — Mehrere Beispiele für die Benutzung von Datenbank-Arrays
- Beispiel 7 — Physisch gekoppelte Dateien benutzen
- Beispiel 8 — VIA-Klausel
- Beispiel 9 — SORTED BY-Klausel
- Beispiel 10 — RETAIN-Klausel
- Beispiel 11 — WHERE-Klausel
- Beispiel 12 — IF NO RECORDS FOUND-Klausel

- Beispiel 13 — Systemvariablen mit dem FIND-Statement benutzen
- Beispiel 14 — Mehrere FIND-Statements

Siehe auch das Beispiel für FIND NUMBER: Programm FNDNUM.

Beispiel 1 — PASSWORD-Klausel

```
** Example 'FNDPWD': FIND (with PASSWORD clause)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
*
1 #PASSWORD (A8)
END-DEFINE
*
INPUT 'ENTER PASSWORD FOR EMPLOYEE FILE:' #PASSWORD (AD=N)
LIMIT 2
*
FIND EMPLOY-VIEW PASSWORD = #PASSWORD
                 WITH NAME = 'SMITH'
  DISPLAY NOTITLE NAME PERSONNEL-ID
END-FIND
*
END
```

Ausgabe des Programms FNDPWD:

```
ENTER PASSWORD FOR EMPLOYEE FILE:
```

Beispiel 2 — CIPHER-Klausel

```
** Example 'FNDCIP': FIND (with PASSWORD/CIPHER clause)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
*
1 #PASSWORD (A8)
1 #CIPHER   (N8)
END-DEFINE
*
LIMIT 2
INPUT 'ENTER PASSWORD FOR EMPLOYEE FILE:   ' #PASSWORD (AD=N)
    / 'ENTER CIPHER KEY FOR EMPLOYEE FILE: ' #CIPHER   (AD=N)
*
FIND EMPLOY-VIEW PASSWORD = #PASSWORD
                 CIPHER   = #CIPHER
                 WITH NAME = 'SMITH'
  DISPLAY NOTITLE NAME PERSONNEL-ID
END-FIND
*
END
```

Ausgabe des Programms FNDCIP:

```
ENTER PASSWORD FOR EMPLOYEE FILE:
ENTER CIPHER KEY FOR EMPLOYEE FILE:
```

Beispiel 3 — Basis-Suchkriterium in WITH-Klausel

```
FIND STAFF WITH NAME = 'SMITH'
FIND STAFF WITH CITY NE 'BOSTON'
FIND STAFF WITH BIRTH = 610803
FIND STAFF WITH BIRTH = 610803 THRU 610811
FIND STAFF WITH NAME = 'O HARA' OR = 'JONES' OR = 'JACKSON'
FIND STAFF WITH PERSONNEL-ID = 100082 THRU 100100
                               BUT NOT 100087 THRU 100095
```

Beispiel 4 — Basis-Suchkriterium mit multiplem Feld

Wenn der im Basis-Suchkriterium benutzte Deskriptor ein multiples Feld ist, können grundsätzlich vier verschiedene Arten von Ergebnissen erzielt werden (das Feld MU-FIELD in den folgenden Beispielen wird als multiples Feld angenommen):

```
FIND XYZ-VIEW WITH MU-FIELD = 'A'
```

Dieses Statement gibt Datensätze zurück, bei denen *wenigstens* eine Ausprägung von MU–FIELD den Wert A hat.

```
FIND XYZ-VIEW WITH MU-FIELD NOT EQUAL 'A'
```

Dieses Statement gibt Datensätze zurück, bei denen *wenigstens* eine Ausprägung von MU–FIELD *nicht* den Wert A hat.

```
FIND XYZ-VIEW WITH NOT MU-FIELD NOT EQUAL 'A'
```

Dieses Statement gibt Datensätze zurück, bei denen *jede* Ausprägung von MU–FIELD den Wert A hat.

```
FIND XYZ-VIEW WITH NOT MU-FIELD = 'A'
```

Dieses Statement gibt Datensätze zurück, bei denen *keine* der Ausprägungen von MU–FIELD den Wert A hat.

Beispiel 5 — Mehrere Beispiele für komplexe Suchausdrücke in WITH-Klausel

```
FIND STAFF WITH BIRTH LT 19770101 AND DEPT = 'DEPT06'

FIND STAFF WITH JOB-TITLE = 'CLERK TYPIST'
                    AND (BIRTH GT 19560101 OR LANG = 'SPANISH')

FIND STAFF WITH JOB-TITLE = 'CLERK TYPIST'
                    AND NOT (BIRTH GT 19560101 OR LANG = 'SPANISH')

FIND STAFF WITH DEPT = 'ABC' THRU 'DEF'
                    AND CITY = 'WASHINGTON' OR = 'LOS ANGELES'
                    AND BIRTH GT 19360101
```

```
FIND CARS WITH MAKE = 'VOLKSWAGEN'
                  AND COLOR = 'RED' OR = 'BLUE' OR = 'BLACK'
```

Beispiel 6 — Mehrere Beispiele für die Benutzung von Datenbank-Arrays

In den folgenden Beispielen wird davon ausgegangen, dass das Feld SALARY (Gehalt) ein in einer Periodengruppe enthaltener Deskriptor und das Feld LANG (Sprache) ein multiples Feld ist.

```
FIND EMPLOYEES WITH SALARY LT 20000
```

Führt zu einer Suche aller Ausprägungen von SALARY.

```
FIND EMPLOYEES WITH SALARY (1) LT 20000
```

Führt zu einer Suche nur nach der ersten Ausprägung.

```
FIND EMPLOYEES WITH SALARY (1:4) LT 20000    /* invalid
```

Eine Bereichsangabe muss nicht für ein als Suchkriterium benutztes Feld innerhalb einer Periodengruppe vorgenommen werden.

```
FIND EMPLOYEES WITH LANG = 'FRENCH'
```

Führt zu einer Suche aller Werte von LANG.

```
FIND EMPLOYEES WITH LANG (1) = 'FRENCH'        /* invalid
```

Ein Index darf für ein als Suchkriterium benutztes multiples Feld nicht angegeben werden.

Beispiel 7 — Physisch gekoppelte Dateien benutzen

```
** Example 'FNDCPL': FIND (using coupled files)
** NOTE: Adabas files must be physically coupled when using the
**       COUPLED clause without the VIA clause.
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 MAKE
END-DEFINE
*
FIND EMPLOY-VIEW WITH CITY = 'FRANKFURT'
     AND COUPLED TO
     VEHIC-VIEW WITH MAKE = 'VW'
  DISPLAY NOTITLE NAME
END-FIND
*
END
```

Beispiel 8 — VIA-Klausel

```
** Example 'FNDVIA': FIND (with VIA clause)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
```

```
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
END-DEFINE
*
FIND EMPLOY-VIEW WITH NAME = 'ADKINSON'
     AND COUPLED TO VEHIC-VIEW
     VIA PERSONNEL-ID = PERSONNEL-ID WITH MAKE = 'VOLVO'
  DISPLAY PERSONNEL-ID NAME FIRST-NAME
END-FIND
*
END
```

Ausgabe des Programms FNDVIA:

```
Page      1                                                  05-01-17  13:18:22

PERSONNEL         NAME              FIRST-NAME
   ID
--------- -------------------- --------------------

20011000  ADKINSON             BOB
```

Beispiel 9 — SORTED BY-Klausel

```
** Example 'FNDSOR': FIND (with SORTED BY clause)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 FIRST-NAME
  2 PERSONNEL-ID
END-DEFINE
*
LIMIT 10
FIND EMPLOY-VIEW WITH CITY = 'FRANKFURT'
                 SORTED BY NAME PERSONNEL-ID

  DISPLAY NOTITLE NAME (IS=ON) FIRST-NAME PERSONNEL-ID
END-FIND
*
END
```

Ausgabe des Programms FNDSOR:

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

BAECKER              JOHANNES             11500345
BECKER               HERMANN              11100311
BERGMANN             HANS                 11100301
BLAU                 SARAH                11100305
BLOEMER              JOHANNES             11200312
DIEDRICHS            HUBERT               11600301
DOLLINGER            MARGA                11500322
FALTER               CLAUDIA              11300311
                     HEIDE                11400311
FREI                 REINHILD             11500301
```

Beispiel 10 — RETAIN-Klausel

```
** Example 'RELEX1': FIND (with RETAIN clause and RELEASE statement)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 BIRTH
  2 NAME
*
1 #BIRTH (D)
END-DEFINE
*
MOVE EDITED '19400101' TO #BIRTH (EM=YYYYMMDD)
*
FIND NUMBER EMPLOY-VIEW WITH BIRTH GT #BIRTH
     RETAIN AS 'AGESET1'
IF *NUMBER = 0
  STOP
END-IF
*
FIND EMPLOY-VIEW WITH 'AGESET1' AND CITY = 'NEW YORK'
  DISPLAY NOTITLE NAME CITY BIRTH (EM=YYYY-MM-DD)
END-FIND
*
RELEASE SET 'AGESET1'
*
END
```

Ausgabe des Programms RELEX1:

```
        NAME                 CITY            DATE
                                               OF
                                             BIRTH
-------------------- -------------------- ----------

RUBIN                NEW YORK             1945-10-27
WALLACE              NEW YORK             1945-08-04
```

Beispiel 11 — WHERE-Klausel

```
** Example 'FNDWHE': FIND (with WHERE clause)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 JOB-TITLE
  2 CITY
END-DEFINE
*
FIND EMPLOY-VIEW WITH CITY = 'PARIS'
                 WHERE JOB-TITLE = 'INGENIEUR COMMERCIAL'
  DISPLAY NOTITLE
          CITY JOB-TITLE PERSONNEL-ID NAME
END-FIND
*
END
```

Ausgabe des Programms FNDWHE:

```
        CITY                  CURRENT          PERSONNEL         NAME
                             POSITION             ID
-------------------- ------------------------ --------- --------------------

PARIS                INGENIEUR COMMERCIAL      50007300  CAHN
PARIS                INGENIEUR COMMERCIAL      50006500  MAZUY
PARIS                INGENIEUR COMMERCIAL      50004700  FAURIE
PARIS                INGENIEUR COMMERCIAL      50004400  VALLY
PARIS                INGENIEUR COMMERCIAL      50002800  BRETON
PARIS                INGENIEUR COMMERCIAL      50001000  GIGLEUX
PARIS                INGENIEUR COMMERCIAL      50000400  KORAB-BRZOZOWSKI
```

Beispiel 12 — IF NO RECORDS FOUND-Klausel

```
** Example 'FNDIFN': FIND (using IF NO RECORDS FOUND)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 15
EMP. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
  /*
  VEH. FIND VEHIC-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (EMP.)

    IF NO RECORDS FOUND
      MOVE '*** NO CAR ***' TO MAKE
    END-NOREC
    /*
    DISPLAY NOTITLE
            NAME (EMP.) (IS=ON)
            FIRST-NAME (EMP.) (IS=ON)
            MAKE (VEH.)
  END-FIND
  /*
END-READ
END
```

Ausgabe des Programms FNDIFN:

```
        NAME              FIRST-NAME              MAKE
-------------------- -------------------- --------------------

JONES                VIRGINIA             CHRYSLER
                     MARSHA               CHRYSLER
                                          CHRYSLER
                     ROBERT               GENERAL MOTORS
                     LILLY                FORD
                                          MG
                     EDWARD               GENERAL MOTORS
                     MARTHA               GENERAL MOTORS
                     LAUREL               GENERAL MOTORS
                     KEVIN                DATSUN
```

```
                    GREGORY             FORD
JOPER               MANFRED             *** NO CAR ***
JOUSSELIN           DANIEL              RENAULT
JUBE                GABRIEL             *** NO CAR ***
JUNG                ERNST               *** NO CAR ***
JUNKIN              JEREMY              *** NO CAR ***
KAISER              REINER              *** NO CAR ***
```

Beispiel 13 — Systemvariablen mit dem FIND-Statement benutzen

```
** Example 'FNDVAR': FIND (using *ISN, *NUMBER, *COUNTER)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 CITY
END-DEFINE
*
LIMIT 3
FIND EMPLOY-VIEW WITH CITY = 'MADRID'
  DISPLAY NOTITLE PERSONNEL-ID NAME
                  *ISN *NUMBER *COUNTER
END-FIND
*
END
```

Ausgabe des Programms FNDVAR:

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

60000114  DE JUAN                     400          41           1
60000136  DE LA MADRID                401          41           2
60000209  PINERO                      405          41           3
```

Beispiel 14 — Mehrere FIND-Statements

In dem folgenden Beispiel werden zuerst alle Mitarbeiter mit Namen SMITH in der Datei EMPLOYEES (Angestellte) ausgewählt. Dann wird die PERSONNEL–ID (Personalnummer) aus der Datei EMPLOYEES als Suchschlüssel für einen Zugriff auf die Datei VEHICLES (Fahrzeuge) benutzt.

```
** Example 'FNDMUL': FIND (with multiple files)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
*
LIMIT 15
EMP. FIND EMPLOY-VIEW WITH NAME = 'SMITH'
  /*
  VEH. FIND VEHIC-VIEW WITH PERSONNEL-ID = EMP.PERSONNEL-ID
    IF NO RECORDS FOUND
```

```
      MOVE '*** NO CAR ***' TO MAKE
    END-NOREC
    DISPLAY NOTITLE
            EMP.NAME (IS=ON)
            EMP.FIRST-NAME (IS=ON)
            VEH.MAKE
  END-FIND
END-FIND
END
```

Ausgabe des Programms FNDMUL:

Der sich ergebende Report zeigt NAME und FIRST–NAME (Vorname) aus der Datei EMPLOYEES für alle Mitarbeiter mit Namen SMITH sowie MAKE (Fabrikat) jedes Autos aus der Datei VEHICLES dieser Mitarbeiter an.

```
        NAME              FIRST-NAME              MAKE
-------------------- -------------------- --------------------

SMITH                GERHARD              ROVER
                     SEYMOUR              *** NO CAR ***
                     MATILDA              FORD
                     ANN                  *** NO CAR ***
                     TONI                 TOYOTA
                     MARTIN               *** NO CAR ***
                     THOMAS               FORD
                     SUNNY                *** NO CAR ***
                     MARK                 FORD
                     LOUISE               CHRYSLER
                     MAXWELL              MERCEDES-BENZ
                                          MERCEDES-BENZ
                     ELSA                 CHRYSLER
                     CHARLY               CHRYSLER
                     LEE                  *** NO CAR ***
                     FRANK                FORD
```