

**SIEMENS**  
**NIXDORF**



SINIX

# INFORMIX V4.1

Ergänzungsband für  
SQL-Sprachbeschreibung  
SQL-Nachschlagen  
ESQL/COBOL  
ESQL/C  
C-ISAM  
Interaktiver Debugger



## Sie haben

uns zu diesem Handbuch etwas mitzuteilen?  
Schicken Sie uns bitte Ihre Anregungen unter  
Angabe der Bestellnummer dieses Handbuches.

Siemens Nixdorf Informationssysteme AG  
Manualredaktion STM QM 2  
Otto-Hahn-Ring 6  
W-8000 München 83

Fax: (0 89) 6 36-4 04 43

email im EUnet:  
man@sieqm2.uucp

## Sie haben

uns zu diesem Handbuch etwas mitzuteilen?  
Schicken Sie uns bitte Ihre Anregungen unter  
Angabe der Bestellnummer dieses Handbuches.

Siemens Nixdorf Informationssysteme AG  
Manualredaktion STM QM 2  
Otto-Hahn-Ring 6  
W-8000 München 83

Fax: (0 89) 6 36-4 04 43

email im EUnet:  
man@sieqm2.uucp

# INFORMIX (SINIX)

Ergänzungsband

Einleitung

SQL-Sprachbeschreibung

SQL-Nachschlagen

ESQL/COBOL

ESQL/C

C-ISAM

Interaktiver Debugger

Anhang

Verzeichnisse

## Wollen Sie mehr wissen....

. . . über dieses Produkt  
. . . oder ein anderes Thema der Informationstechnik?

Unsere Training Center stehen für Sie bereit.  
Besuchen Sie uns in Berlin, Essen, Frankfurt oder Hamburg,  
in Hannover, Mainz, München, Stuttgart, Wien oder Zürich.

Auskunft und Informationsmaterial erhalten Sie über:

München (089) 636-2009  
oder schreiben Sie an:

Siemens Nixdorf Training Center  
Postfach 83 09 51, W-8000 München 83

Copyright © Siemens Nixdorf Informationssysteme AG, 1992. Alle Rechte vorbehalten.  
Basis: INFORMIX-SQL, INFORMIX-ESQL/COBOL, INFORMIX-ESQL/C, INFORMIX-C-ISAM  
Copyright © INFORMIX Software Inc. 1986/87.  
INFORMIX ist ein eingetragenes Warenzeichen der INFORMIX Software Inc.

Weitergabe sowie Vervielfältigung oder Übersetzung dieser Unterlage, auch von Teilen, durch Drucken, Kopieren oder ähnliche Verfahren, Verwertung und Mitteilung ihres Inhaltes nicht gestattet, soweit nicht ausdrücklich zugestanden. Zuwiderhandlungen verpflichten zu Schadenersatz.  
Liefermöglichkeiten und technische Änderungen vorbehalten.

---

# Inhalt

<b>1</b>	<b>Einleitung</b>			<b>1</b>
	Aufbau dieses Ergänzungsbandes			2
	Darstellungsmittel			2
<b>2</b>	<b>INFORMIX SQL Sprachbeschreibung V4.1</b>			<b>3</b>
2.1	Änderungen im Überblick			3
2.2	Beschreibung der Änderungen für beide INFORMIX-Systeme (INFORMIX-SE und INFORMIX-ONLINE)			4
	Allgemeiner Hinweis zu Datenbankanwendungen			4
	Ergänzung zur SQL-Syntax für einfache Namen			4
	Änderung der Reservierung von Schlüsselwörtern			5
	USER-Funktion			5
	Erweiterung der SELECT-Anweisung			5
	Satzzeiger			6
	Erfolgskontrolle bei Programmeinbettung			6
	Änderungen unter INFORMIX-ONLINE			8
	Datenbanken unter INFORMIX-ONLINE ändern			8
	Transaktionssicherung			8
	Effektivere Überprüfung von Eindeutigkeitsbedingungen (Constraint) für Tabellenspalten			9
	Blobdaten in der LOAD- und UNLOAD-Anweisung			10
2.3	Fehlerbehebungen			11
	DECIMAL	Abschnitt 4.2.2	S. 4-14f	11
	FLOAT, DOUBLE PRECISION	Abschnitt 4.2.2	S. 4-19	11
	CLOSE DATABASE-Anweisung	Kapitel 6	S. 6-36	11
	CREATE TABLE-Anweisung	Kapitel 6	S. 6-64ff	11
	COMMIT WORK-Anweisung	Kapitel 6,	S. 6-38	12
	PREPARE-Anweisung	Kapitel 6,	S. 6-156	12
	EXECUTE-Anweisung	Kapitel 6,	S. 6-107	12

<b>3</b>	<b>INFORMIX-SQL-Nachschlagen V4.1</b>	<b>13</b>
	Kompatibilität	13
3.1	Änderungen im Überblick	14
3.2	Beschreibung der Änderungen	16
	Arbeiten mit ONLINE-Datentypen auf Menüebene	16
	ONLINE-Datentypen definieren	16
	ONLINE-Datentypen abfragen	18
	Änderungen für Formate	19
	Neues Attribut - INVISIBLE	19
	BLOB-Daten anzeigen	21
	Optionales Suffix bei PERFORM	22
	C-Anschluß für PERFORM	22
	Änderungen für ACE	25
	Neue Anweisung im OUTPUT-Abschnitt	25
	Sortierung im SELECT-Abschnitt	26
	Veränderte Reihenfolge im FORMAT-Abschnitt	26
	Erweiterung der PRINT-Anweisung	27
	ONLINE-Datentypen im REPORT	27
	Paralleles Sortieren (nur für INFORMIX-ONLINE)	28
	Optionales Suffix bei ACE	28
	C-Anschluß für ACE	29
	Übersetzen des Listenprogramms	32
	C-Anschluß	33
	C-Programmstruktur	33
	Eingabeparameter	36
	Rückgabewerte	38
	Spezielle Bibliotheksfunktionen von PERFORM	39
	PF_GETTYPE	40
	PF_GETVAL	41
	PF_PUTVAL	43
	PF_NXFIELD	45
	PF_MSG	46
	Der Übersetzungs-, Binde- und Ablaufprozeß	47
	Beispiele für C-Programme in ACE und PERFORM	48
	ctools.h	61
3.3	Fehlerbehebungen	64
<b>4</b>	<b>INFORMIX-ESQL/COBOL V4.1</b>	<b>67</b>
4.1	Änderungen im Überblick	67
4.2	Beschreibung der Änderungen	68
4.2.1	Änderungen zum Bereich Erfolgskontrolle	68
	Die Variablen SQLERRD[1] und SQLERRD[4] des SQLCA-Satzes	68
	SQLNOTFOUND für eine ANSI-Datenbank	68
	Benutzung der SQLCODE-Variable	69

4.3	Fehlerbehebungen	70
	Zu 2.3 Vereinbarung der Hostvariablen S.2-7	70
	Zu 2.3.5 SMALLFLOAT und FLOAT S.2-15	70
	Zu 2.3.6 DECIMAL S.2-16	70
	Zu 2.6.3 Beispiele S.2-35	70
	Zu ECO-MSGFehlermeldungsnummern in Text umsetzen, Kapitel 4	70
	Zu 3.3 INFORMIX-ESQL/COBOL und LEVEL II COBOL	71
<b>5</b>	<b>INFORMIX-ESQL/C V4.1</b>	<b>73</b>
5.1	Änderungen im Überblick	73
5.2	Beschreibung der Änderungen	74
5.2.1	Änderungen zum Bereich Erfolgskontrolle	74
	Die Variablen SQLERRD[0] und SQLERRD[3] der sqlca-Struktur	74
	SQLNOTFOUND für eine ANSI-Datenbank	75
	Einbindung der Include-Datei sqlca.h	76
	Benutzung der SQLCODE-Variablen	77
5.2.2	Bibliotheksfunktionen (siehe ESQL/C-Handbuch, Kapitel 4)	78
	sqldetach Vaterprozeß von Sohnprozeß trennen	78
5.3	Fehlerbehebungen	79
	Indikatorvariable zur Erkennung von Informationsverlust	79
	Die sqlerrd [2]- Variable aus der sqlca-Struktur	79
<b>6</b>	<b>C-ISAM V4.1</b>	<b>81</b>
6.1	Änderungen im Überblick	81
	Umstellung auf C-ISAM Version 4.1	82
6.2	Beschreibung der Änderungen	83
	Programmierung mit Datensätzen variabler Länge	83
	Dateien mit Datensätzen variabler Länge erzeugen und verwenden	83
	Index definieren	85
	Dateiverwaltung bei Datensätzen variabler Länge	87
	Geänderte Funktionen	91
	ISADDINDEX	91
	ISBUILD	93
	ISINDEXINFO	96
	ISOPEN	99
	ISREAD	101
	ISREWCURR	105
	ISREWREC	106
	ISREWRITE	107
	ISWRCURR	109
	ISWRITE	110
	Die include-Datei isam.h (Anhang C)	112
	Fehlercodes (Anhang D)	118
	Dateiformate (Anhang E)	125
	Formate von Indexdateien	125



	Formate von Datendateien	131
	Formate von AUDIT-Dateien	131
	Formate von Transaktionsprotokoll-Dateien	132
<b>7</b>	<b>INFORMIX Interactive Debugger Version 4.1</b>	<b>137</b>
7.1	Änderungen im Überblick	137
7.2	Beschreibung der Änderungen	139
	BREAK	139
	CALL	139
	CLEANUP	140
	DUMP	140
	LET	141
	PRINT	141
	TRACE	142
	VARIABLE	143
	WHERE	143
7.3	Fehlerbehebungen	144
<b>A</b>	<b>Anhang</b>	<b>147</b>
A.1	Umgebungsvariablen	147
	Umgebungsvariable DBFORMAT	148
	Umgebungsvariable INFORMIXDIR	150
	Umgebungsvariable INFORMIXTERM	151
	Umgebungsvariable PSORT_NPROCS (nur ONLINE)	151
	Umgebungsvariable PSORT_DBTEMP (nur ONLINE)	151
	Umgebungsvariable SACEISQL (nur ONLINE)	152
A.2	Dienstprogramme	153
	Funktionserweiterung der Schalter für bcheck	154
A.3	Fehlermeldungen ausgeben (nur ONLINE)	157
	ASCII-Dateien	157
	Postscript-Dateien	159
A.4	Reservierte Wörter	160
	<b>Literatur</b>	<b>169</b>
	<b>Stichwörter</b>	<b>173</b>

---

## Einleitung

Der vorliegende Ergänzungsband enthält die Änderungen der Version 4.1 gegenüber der Version 4.0. Für folgende Produkte finden Sie die Beschreibung in diesem Band:

- SQL-Sprachbeschreibung (Kapitel 2)
- SQL-Nachschlagen (Kapitel 3)
- ESQL/Cobol (Kapitel 4)
- ESQL/C (Kapitel 5)
- C-ISAM (Kapitel 6)
- Interactive Debugger (Kapitel 7)

Um mit diesen INFORMIX-Komponenten zu arbeiten, benötigen Sie die jeweilige Beschreibung für die Version V4.0. Eine Auflistung dieser Handbücher finden Sie am Ende dieses Buches in der Literaturliste.

Die folgenden Handbücher der Version INFORMIX V4.1 werden in einer Neuausgabe herausgegeben:

- ONLINE Administratorhandbuch
- NET/STAR Netzkomponente für INFORMIX
- 4GL-Nachschlagen
- Fehlermeldungen für alle INFORMIX-Produkte
- NLS National Language Support

### Aufbau dieses Ergänzungsbandes

Dieser Ergänzungsband enthält die Änderungen für die zuerst genannten INFORMIX-Produkte. In jedem Kapitel sind die Änderungen eines Produkts vollständig beschrieben. Beispielsweise beschreibt Kapitel 2 die Neuerungen für die SQL-Sprachbeschreibung, Kapitel 3 die für SQL-Nachschnitten usw.

Die einzelnen Kapitel beginnen jeweils mit einem Überblick über die Änderungen für das spezielle Produkt. Daran schließt sich eine vollständige Beschreibung der funktionalen Änderungen an. Die inhaltliche Einteilung dieser Beschreibung orientiert sich an den Handbüchern für die Version V4.0.

Im letzten Abschnitt eines jeden Kapitels sind die Fehler berichtigt, die sich in der Beschreibung der INFORMIX Version V4.0 angefundener haben.

Der Anhang dieses Buches beschreibt die neuen und geänderten Umgebungsvariablen und das erweiterte Dienstprogramm *bcheck*. Des weiteren können Sie hier nachlesen, wie Sie bei ONLINE-Systemen Fehlermeldungen am Bildschirm oder Drucker ausgeben können.

### Darstellungsmittel

Die Darstellungsmittel für Syntax und Beispiele entsprechen denen in den Handbüchern für die INFORMIX-Version 4.0. Einzige Neuerung ist die Verwendung der folgenden Zeichen



für Warnungen.



für Hinweise und weiterführende Informationen

---

# INFORMIX SQL Sprachbeschreibung V4.1

## Änderungen im Überblick

Folgende Änderungen sind im Handbuch SQL-Sprachbeschreibung[3] beschrieben und beziehen sich auf SQL-Sprachelemente. Im folgenden erhalten Sie einen Überblick über den gesamten Änderungsumfang der INFORMIX-Version 4.1.

Für die **SELECT-Anweisung** ergeben sich folgende Änderungen:

- Sie können UNION und UNION ALL in derselben SELECT-Anweisung benutzen.
- Sie können UNION ALL und SELECT DISTINCT in derselben ben SELECT-Anweisung benutzen.
- Sie können ORDER BY und SELECT DISTINCT in derselben SELECT-Anweisung benutzen.
- Bei SELECT-Verknüpfungen können Sie mehrere ORDER BY Sortierungen angeben.

Für **Satzzeiger** ergeben sich folgende Änderungen:

- Bei einer ANSI-Datenbank können Sie die Klausel FOR UPDATE bei der Vereinbarung des Nicht-Scrollsatzzeigers weglassen und trotzdem den Satzzeiger in der UPDATE- bzw. DELETE-Anweisung benutzen.
- Bei einer ANSI-Datenbank erhalten Sie einen Fehler, wenn Sie einen geschlossen Satzzeiger mit der CLOSE-Anweisung erneut schließen.

Änderungen für **Namen**

- Wenn Sie für Eigner- oder login-Namen in einer ANSI-Datenbank keine Anführungsstriche angeben, konvertiert das System alle Buchstaben in Großbuchstaben. Diese Konversion dient dazu, den ANSI-Standard zu erfüllen. Die Schreibweise in der login\_Tabelle muß mit der login-Angabe im Namen übereinstimmen
- Nahezu 200 Wörter sind in der Version 4.1 nicht länger reserviert und können als Namen benutzt werden. Allerdings kann es dabei zu Mehrdeutigkeiten kommen. Es empfiehlt sich daher, die Reservierungen weiterhin soweit als möglich zu beachten.

# Beschreibung der Änderungen für beide INFORMIX-Systeme (INFORMIX-SE und INFORMIX-ONLINE)

### Allgemeiner Hinweis zu Datenbankanwendungen

**Referenz in der Sprachbeschreibung:**    **Abschnitt 2.1**

Vorhandene 4.0-Datenbankanwendungen, die ANSI-Datenbanken verwenden, müssen in der Version 4.1 angepaßt werden, da sie in der Regel nicht ablaufen werden.

### Ergänzung zur SQL-Syntax für einfache Namen

**Referenz in der Sprachbeschreibung:**    **Abschnitt 3.2.1**

Wird ein Name ohne Anführungsstriche in einer SQL-Anweisung benutzt, die auf einer ANSI-Datenbank ausgeführt wird, wandelt INFORMIX automatisch alle Buchstaben in Großbuchstaben um. Eine Ausnahme bilden *informix* und *public*, die immer kleingeschrieben werden. Der ANSI-Standard schreibt diese Änderung vor. Wenn die angegebene Schreibweise in Kleinbuchstaben oder gemischten Buchstaben erhalten bleiben soll, müssen Sie den Namen in Anführungsstriche setzen. Bei jedem folgenden Zugriff auf diesen Namen, müssen Sie die Anführungsstriche angeben.

### Beispiel

Ein Benutzer mit der Kennung **james** erzeugt eine Tabelle **kundenhinweis** in einer ANSI-Datenbank :

```
$CREATE TABLE "james".kundenhinweis ( kd_nr integer, hinweis  
char (240));
```

Auf diese Tabelle kann man jetzt z.B. folgendermaßen zugreifen:

```
$SELECT * from "james".kundenhinweis;
```

## Änderung der Reservierung von Schlüsselwörtern

**Referenz in der Sprachbeschreibung:**    **Abschnitt 3.1 und 3.2.1**

Schlüsselwörter sind weitgehend nicht mehr reserviert. Sie sind als Namen zugelassen; es kann dabei allerdings zu Uneindeutigkeit kommen. Außerdem schreibt der ANSI-Standard weitgehend die Beachtung der Reservierung vor. Es empfiehlt sich daher, die Schlüsselwörter als reservierte Wörter zu betrachten. Im Anhang A.4 finden Sie die Liste der Schlüsselwörter; aus dieser Liste ist ersichtlich, welche Wörter reserviert sind, welche Wörter vom ANSI-Standard als reserviert betrachtet werden und welche Wörter nicht mehr reserviert sind.

## USER-Funktion

**Referenz in der Sprachbeschreibung:**    **Kapitel 5, USER-Funktion**

Name des aktuellen Benutzers als Zeichenkette. Wenn Sie auf einer Nicht-ANSI-Datenbank arbeiten, gibt USER den aktuellen Benutzernamen unkonvertiert zurück. Wenn Sie auf einer ANSI-Datenbank arbeiten, konvertiert USER alle Buchstaben des Namens in Großbuchstaben. Ausnahmen sind *informix* und *public*, die immer in Kleinbuchstaben erscheinen.

## Erweiterung der SELECT-Anweisung

**Referenz in der Sprachbeschreibung:**    **Kapitel 6, S. 6-201 ff**

Die UNION-Klausel verbindet zwei SELECT-Anweisungen. Die Ergebnistabelle enthält alle Sätze, die in der ersten und zweiten Ergebnistabelle vorkommen. Sie können mehr als zwei Ergebnistabellen verbinden, wenn Sie die UNION-Klausel mehrmals verwenden, dabei dürfen sowohl UNION als auch UNION ALL in einer Verbindungskette vorkommen.

Außerdem dürfen UNION ALL und SELECT DISTINCT in derselben SELECT-Anweisung vorkommen.

Ab dieser Version dürfen die Klauseln ORDER BY und INTO TEMP in derselben SELECT-Anweisung stehen.

### Satzzeiger

**Referenz in der Sprachbeschreibung:    Abschnitt 2.13.3**

Die Klausel FOR UPDATE ist bei ANSI-Datenbanken zum Ändern und Löschen von Sätzen überflüssig.

Bei einer ANSI-Datenbank erhalten Sie einen Fehler, wenn Sie einen geschlossenen Satzzeiger mit der CLOSE-Anweisung erneut schließen.

### Erfolgskontrolle bei Programmeinbettung

**Referenz in der Sprachbeschreibung:    Abschnitt 2.13.5 und Kapitel 6**

Kann eine der folgenden Anweisungen beim Zugriff auf eine ANSI-Datenbank keinen Satz finden, so wird die Variable *sqlcode* aus der *sqlca*-Struktur auf 100 gesetzt statt bisher auf 0. Betroffen sind folgende Anweisungen:

- DELETE
- INSERT INTO ...SELECT
- SELECT INTO TEMP
- UPDATE

Genauere Beschreibung finden Sie in den Handbüchern für die Programmeinbettung.

	Positivfall	Fehlerfall (siehe unten)
INFORMIX-4GL	<p>status = 0 SQLCA.SQLERRD[3] ist die Anzahl der bearbeiteten Sätze.</p> <p>bei ANSI-Datenbank zusätzlich: status = 100 , wenn kein Satz zum Bearbeiten gefunden wurde.</p>	<p>status &lt; 0 SQLCA.SQLERRD[3] ist die Anzahl der erfolgreich bearbeiteten Sätze Der Rest der Sätze ist nicht gelöscht.</p>
INFORMIX-ESQL/C	<p>sqlca.sqlcode = 0 sqlca.sqlerrd[2] ist die Anzahl der bearbeiteten Sätze.</p> <p>bei ANSI-Datenbank zusätzlich: sqlca.sqlcode=SQLNOTFOUND wenn kein Satz zum Bearbeiten gefunden wurde. SQLNOTFOUND ist mit 100 in sqlca.h definiert.</p>	<p>sqlca.sqlcode &lt; 0 sqlca.sqlerrd[2] ist die Anzahl der mit DELETE Sätze. Der Rest der Sätze ist nicht gelöscht.</p>
INFORMIX-ESQL/COBOL	<p>SQLCODE OF SQLCA = 0 SQLERRD[3] ist die Anzahl der bearbeiteten Sätze.</p> <p>bei ANSI-Datenbank zusätzlich: SQLCODE OF SQLCA = SQLNOTFOUND wenn kein Satz zum Bearbeiten gefunden wurde. SQLNOTFOUND ist mit 100 im SQLCA-Satz definiert.</p>	<p>SQLCODE OF SQLCA &lt; 0 SQLERRD[3] ist die Anzahl der erfolgreich bearbeiteten Sätze Der Rest der Sätze ist nicht gelöscht.</p>

Zur Fehlersituation sei noch einmal darauf hingewiesen, daß die Variable *sqlerrd* immer den Wert 0 hat, wenn Sie mit Transaktionssicherung, insbesondere einer ANS\_Datenbank, arbeiten.



## Änderungen unter INFORMIX-ONLINE

### Datenbanken unter INFORMIX-ONLINE ändern

**Referenz in der Sprachbeschreibung:    Abschnitt 2.10.2 und Kapitel 6**

Bei INFORMIX-ONLINE haben Sie zusätzlich die Möglichkeit, Daten aus externen Tabellen abzufragen, wobei die Datenbank in einem anderen INFORMIX-System auf demselben oder auf einem anderen Rechner liegen kann.

Sie können außerdem verschiedene Datenbanken in einer Transaktion verändern (UPDATE), wenn diese Datenbanken alle unter einem ONLINE-System stehen. Dabei brauchen Sie nicht die aktuelle Datenbank schließen und auch nicht die fremden Datenbanken öffnen.

### Transaktionssicherung

**Referenz in der Sprachbeschreibung:    Abschnitt 2.10.2 und Kapitel 6**

Das Konzept der Transaktionssicherung ermöglicht die Einhaltung und Wiederherstellung von konsistenten Zuständen in einer Datenbank oder wenn Sie mit INFORMIX-ONLINE arbeiten, in mehreren Datenbanken. Wollen Sie in einer Transaktion mehrere Datenbanken bearbeiten, müssen diese alle unter dem gleichen INFORMIX-ONLINE-System stehen.

## Effektivere Überprüfung von Eindeutigkeitsbedingungen (Constraint) für Tabellenspalten

### Referenz in der Sprachbeschreibung: Abschnitt 2.7

Vor der Version 4.1 wurde ein Constraint bei jedem Satz überprüft, auf den bei der Ausführung einer UPDATE- oder INSERT-Anweisung zugegriffen wurde. Ab der Version 4.1 wird dieses Verfahren geändert. Die Eindeutigkeitsbedingung wird erst nach allen Änderungen einer Anweisung auf einer Tabelle überprüft. Da nur die gesamte Wirkung auf die Tabelle auf Nichteinhaltung des Constraint überprüft wird und zwischenzeitlich erreichte Werte ignoriert werden, sind jetzt einige Anweisungen gültig, die vorher gegen den Constraint verstoßen haben. Online überprüft die Wirkung von UPDATE- und INSERT-Anweisungen sowohl in expliziten als auch in impliziten Transaktionen. Wird ein Constraint verletzt, gibt INFORMIX-ONLINE eine Fehlermeldung zurück und setzt alle Sätze, die von der verletzenden Anweisung betroffen sind, auf ihren ursprünglichen Wert.

Tritt eine Constraint-Verletzung auf und es gibt keine "Zurücksetzen im Fehlerfall"-Fehlerbehandlungsroutine, werden alle Anweisungen, die erfolgreich ausgeführt wurden, in die Datenbank übernommen. Nur die Anweisung, die gegen Constraint verstößt wird nicht ausgeführt.

Das folgende Beispiel zeigt die Veränderung des Überprüfungsverfahrens. Das Beispiel erzeugt eine Tabelle *east\_ship* mit zwei Spalten, *code* und *name* und zwei Sätzen. Eine Eindeutigkeitsbedingung wird der Spalte *code* vergeben. Während die UPDATE-Anweisung ausgeführt wird, erhält der Spaltenwert von *code* den Wert 2; dieser Wert verstößt zwischenzeitlich gegen die Eindeutigkeitsbedingung. Nach der vollständigen Ausführung der UPDATE-Anweisung ist die Eindeutigkeit nicht verletzt. Die Anweisung kann erfolgreich abgeschlossen werden.

---

```
CREATE TABLE east_ship (code INTEGER UNIQUE, name CHAR (25));
INSERT INTO east_ship VALUES (1, "JS Delivery");
INSERT INTO east_ship VALUES (2, "Express Products");
UPDATE east_ship SET code = code + 1
```

---

**Blobdaten in der LOAD- und UNLOAD-Anweisung****Referenz in der Sprachbeschreibung: Kapitel 6,  
LOAD- und UNLOAD-Anweisung**

Wollen Sie unter ONLINE mit der LOAD- bzw. UNLOAD-Anweisung Blobdaten einfügen bzw. ausgeben, so dürfen Sie unter DELIMITER folgende Zeichen nicht angeben:

- die Ziffern 0 - 9
- die Buchstaben A - F (Groß- und Kleinbuchstaben)
- Leerzeichen \_
- Tabulatorzeichen
- den Gegenschrägstrich \

Neben dem vordefinierten | können Sie als Trennzeichen die Buchstaben G - Z, Komma und Punkt vereinbaren. Dabei sind alle Regeln wie in Kapitel 6 beschrieben zu beachten.

Für die UNLOAD-Anweisung gilt:

Byte-Daten werden im hexadezimalen Dump-Format ohne Leerzeichen und Neue-Zeile-Zeichen ausgegeben. Das entladene Byte-Datum kann demnach sehr lang sein und ist möglicherweise nicht ausdrückbar oder editierbar. Bei VARCHAR werden abschließende Leerzeichen nicht mitausgegeben.

Für die LOAD-Anweisung gilt:

VARCHAR- und TEXT-Daten können führende und abschließende Leerzeichen enthalten, BYTE-Daten nicht. Sie können in der LOAD-Anweisung Daten angeben, die länger als die zugehörige Spalte sind. Die überzähligen Zeichen werden nicht beachtet.

## Fehlerbehebungen

Im Handbuch INFORMIX-SQL Sprachbeschreibung V4.0 sind einige Fehler aufgetreten, die im folgenden berichtigt werden sollen.

Über jedem neuen Absatz finden Sie jeweils den zugehörigen Abschnitt aus dem Manual und die Seitenangabe.

### **DECIMAL    Abschnitt 4.2.2 S. 4-14f**

Zur Beschreibung der Angabe  $n$  in DECIMAL muß es heißen: Wenn Sie mehr als  $m$  Nachkommastellen angeben, rundet INFORMIX automatisch auf  $m$  Nachkommastellen.

Der Wertebereich für DECIMAL-Festpunkt-Spalten muß lauten: Eine DECIMAL-Festpunkt-Spalte kann Festpunktzahlen enthalten, deren Betrag im Bereich von  $0.5 * 10^{-n}$  bis  $10^{m-n} - 10^{-n}$  liegt.

Wenn Sie eine Realzahl angeben, die betragsmäßig kleiner als  $0.5 * 10^{-n}$  ist, wird diese auf 0 gerundet.

### **FLOAT, DOUBLE PRECISION    Abschnitt 4.2.2 S. 4-19**

DOUBLE PRECISION ist ein Synonym für FLOAT. Allerdings ist die Angabe der Genauigkeit bei DOUBLE PRECISION in dieser Version nicht möglich. FLOAT-Datenwerte können derzeit nur dann in einem abdruckbaren Format entladen werden ( z.B. dbexport), wenn sie innerhalb des Wertebereichs für DECIMAL liegen.

### **CLOSE DATABASE-Anweisung    Kapitel 6 S. 6-36**

Im ersten Abschnitt der Beschreibung steht irrtümlicherweise, daß ein erneutes Schließen einer Datenbank ignoriert wird. Sie erhalten aber eine Fehlermeldung.

### **CREATE TABLE-Anweisung    Kapitel 6 S. 6-64ff**

Die Beschreibung der temporären Tabelle muß um folgenden Sachverhalt ergänzt werden: Sie können keine Namen für Constraints in einer temporären Tabelle vergeben.

**COMMIT WORK-Anweisung    Kapitel 6, S. 6-38**

Im ersten Abschnitt der Anweisungsbeschreibung muß es heißen: COMMIT WORK führt zu einem Fehler, wenn keine Transaktion eröffnet ist.

**PREPARE-Anweisung    Kapitel 6, S. 6-156**

Unter *anweisungsbezeichner* ist folgendes zu ergänzen: Der Name muß innerhalb der Datei eindeutig sein. Auch wenn Sie eine FREE-Anweisung angeben, darf dieser Name nicht in einer folgenden PREPARE-Anweisung erneut verwendet werden.

Unter *anweisung* ist folgendes zu ergänzen: Der Anweisungsstring in der PREPARE-Anweisung darf maximal 255 Zeichen lang sein.

Zu Multistatement-PREPARE:

Es ist möglich SELECT-Anweisungen mit INTO TEMP zu mehreren in einer PREPARE-Anweisung vorzubereiten. Sonst müssen SELECT-Anweisungen einzeln mit PREPARE vorbereitet werden.

**EXECUTE-Anweisung    Kapitel 6, S. 6-107**

Die Beschreibung der EXECUTE-Anweisung muß folgendermaßen erweitert werden: Es ist sinnvoll, eine EXECUTE-Anweisung auf ihren Erfolg zu überprüfen. Folgende Fehlersituation kann INFORMIX nicht abfangen: Eine Transaktion wird als Multistatement-PREPARE vorbereitet. In der Transaktion befindet sich eine fehlerhafte Anweisung, z.B. Erzeugen einer existierenden Tabelle. Der Fehler tritt erst bei der EXECUTE-Anweisung auf. Alle Anweisungen, die vor dem Fehler stehen, werden ausgeführt. Mit dem Auftreten des Fehlers wird die EXECUTE-Anweisung abgebrochen, aber die angefangene Transaktion wird nicht zurückgesetzt. Im weiteren Verlauf wird der mit EXECUTE nicht mehr ausgeführte Transaktionsschluß vermißt. So läßt sich z.B. die Datenbank weder schließen noch löschen. Es ist daher notwendig, die EXECUTE-Anweisung auf ihren Erfolg zu prüfen und abhängig von der Prüfung eine COMMIT WORK- oder ROLLBACK-Anweisung einzusetzen.

---

# INFORMIX-SQL-Nachschlagen V4.1

Dieses Kapitel enthält die Änderungen von INFORMIX-SQL-Nachschlagen V4.1. Es beginnt mit einem Überblick über die Änderungen, so daß Sie spezielle Punkte leicht finden können. Die Reihenfolge der einzelnen Abschnitte entspricht der Reihenfolge der Kapitel der Version 4.0.

Danach folgt die vollständige Beschreibung der Änderungen. Im letzten Abschnitt dieses Kapitels sind die Fehler berichtigt, die sich im Handbuch INFORMIX-SQL Nachschlagen V4.0 angefundnen haben.


## Kompatibilität

Wenn Sie sowohl Produkte der Version 4.0 als auch der Version 4.1 einsetzen, wird die neue Funktionalität nicht unterstützt.

Wenn Sie Format- oder Listenprogramme verwenden wollen, die Sie mit früheren Versionen von INFORMIX-SQL erstellt haben, müssen Sie folgende Dateien neu kompilieren:

- alle Dateien mit der Endung **.per**
- alle Dateien mit der Endung **.ace**

Wenn Sie Programme, die Sie mit ESQL/C-Versionen kleiner 4.1 erstellt haben, in Ihre Format- und Listenprogramme einbeziehen wollen, müssen Sie die entsprechenden Dateien mit den Endungen **.ec** oder **.c** neu übersetzen.

 Die Versionsnummern von INFORMIX-SQL und ESQL/C müssen übereinstimmen.

## Änderungen im Überblick

Folgende Änderungen betreffen das Handbuch INFORMIX-SQL Nachschlagen. Sie werden im zweiten Kapitel dieses Abschnitts beschrieben. Hier sehen Sie einen Überblick über diese Änderungen:

### **Kapitel 2 - SQL anwenden**

Die Verwendung von ONLINE-Datentypen auf Menüebene wurde überarbeitet.

### **zu Kapitel 3 und 4 - Formbuild und Perform**

- Für Formate wird das Attribut *INVISIBLE* zur Verfügung gestellt.
- Die Endungen *.per* und *.frm* sind optional.
- C-Anschluß besteht die Möglichkeit, C-Funktionen im Formatprogramm aufzurufen.

### **zu Kapitel 5 - ACE**

- Für den OUTPUT-Abschnitt wird die *TOP OF PAGE*-Anweisung eingeführt, die einen Seitenvorschub auslöst.
- Die Reihenfolge der Kontrollblöcke *PAGE HEADER* und *ON EVERY ROW* hat sich geändert.
- Mit der PRINT-Anweisung kann auch der NULL-Wert ausgedruckt werden.
- Bei ONLINE-Backends können Sie parallel sortieren.
- Die Endungen *.ace* und *.arc* sind optional.
- Über den C-Anschluß besteht die Möglichkeit, C-Funktionen im Listenprogramm aufzurufen.

**zu Kapitel 7 - Systemumgebung von INFORMIX**

Folgende Umgebungsvariablen kommen neu zur Funktionalität von INFORMIX-Nachschlagen dazu oder wurden geändert:

- DBFORMAT                      numerische Werte bei Ein- und Ausgaben formatieren
- INFORMIXDIR                 Dateiverzeichnis der INFORMIX-Produkte
- INFORMIXTERM                Bildschirmsteuerung wählen
- PSORT\_NPROCS                Menge der Prozessoren für paralleles  
  (nur ONLINE)                 Sortieren
- PSORT\_DBTEMP                temporäres Dateiverzeichnis für paral-  
  (nur ONLINE)                 lele Sortierung
- SACEISQL (nur ONLINE)     Isolationsstufe für Listenprogramme  
  (nur ONLINE)                 setzen

Die Beschreibung der neuen Umgebungsvariablen finden Sie im Anhang dieses Buches. Sie finden weiterhin im Anhang eine überarbeitete und geänderte Beschreibung des Dienstprogramms *bcheck* und eine Anleitung, wie Sie Fehlermeldungen am Bildschirm oder Drucker ausgeben können, wenn Sie mit einer ONLINE-Datenbank arbeiten.

**C-Anschluß für ACE und Perform**

Sowohl in Formaten als auch in Listen können Sie ab der Version V4.1 C-Funktionen einbinden.



## Beschreibung der Änderungen

### Arbeiten mit ONLINE-Datentypen auf Menüebene

Wenn Sie auf Menüebene eine Tabelle erstellen, können Sie vordefinierte Standard-Datentypen auswählen. Wenn Sie mit ONLINE arbeiten, können Sie ab der Version 4.1 im Menü TYP mit der Menüfunktion VARIABLE LENGTH die Datentypen VARCHAR, TEXT und BYTE auswählen.

### ONLINE-Datentypen definieren

Wählen Sie im Menü TABELLE ERSTELLEN die Menüfunktion *Neu*, um eine neue Tabelle anzulegen oder *Modifizieren*, um eine bereits existierende Tabelle zu ändern. Nachdem Sie die Datenbank ausgewählt und Tabellen- und Spaltennamen angegeben haben, werden Sie im Menü TYP aufgefordert, den Datentyp, den die Spalte erhalten soll auszuwählen. Das Menü TYP ist im folgenden dargestellt, *tablename* steht für den Tabellennamen, *dbname* für den Namen der Datenbank:

```

NEU:TYP tablename: Char Numeric Serial Date dateTime Interval ...
Erlaubt jede Kombination von Buchstaben, Zahlen und Symbolen
-----dbname-----

```

Fortsetzung des Menüs TYP:

```

NEU:TYP tablename: ... VARIABLE LENGTH
Zeigt das VARIABLE LENGTH-Menue fuer Spalten mit variabler Laenge
-----dbname-----

```

Im Untermenü *VARIABLE LENGTH* können Sie dann die *ONLINE*-Datentypen auswählen:

```

NEU:VARIABLE LENGTH: Varchar Text Byte
Variable Length-Daten - enthalten maximal 255 Zeichen
-----1-----dbname-----

```

Wenn Sie *VARCHAR* ausgewählt haben, müssen Sie noch die maximale Länge dieser Spalte bestimmen.

```

NEU:MAXIMUM LENGTH>>
Geben Sie die maximale Laenge von 1 bis 255 ein. Weiter mit RETURN
-----1-----dbname-----

```

Eine *VARCHAR*-Spalte hat eine Ober- und eine Untergrenze. Auch die Untergrenze können Sie festlegen (weitere Informationen über *ONLINE*-Datentypen stehen in der *SQL*-Sprachbeschreibung[3] in Kapitel 4).

```

NEU:MINIMUM LENGTH>>
Geben Sie die zu reservierende Platzmenge von 0 bis max. Laenge an
-----1-----dbname-----

```

Wenn Sie im Menü *VARIABLE LENGTH* einen *BLOB*-Datentyp ausgewählt haben, müssen Sie angeben, wo die Daten gespeichert sind. Das Menü *BLOBSpace* bietet Ihnen folgende Möglichkeiten:

```

NEU:BLOBSpace tablename: Tabelle BLOBSpace-Name
Spaltendaten werden in derselben Tabelle gespeichert wie andere Spalten
-----1-----dbname-----

```

Wenn Sie die Option *Tabelle* auswählen, speichert ONLINE die BLOB-Daten in demselben *dbspace* wie die Daten der anderen Spalten (Näheres über die ONLINE-Datenverwaltung erfahren Sie im ONLINE-Handbuch[5] in Kapitel 2).

Nach der Auswahl von *BLOBspace-Name*, fordert ONLINE Sie auf, den Namen des *BLOBspace* anzugeben:

```

NEU:BLOBSpace NAME>>
Geben Sie den BLOBSpace Namen an
_____1_____dbname_____
    
```

Hier können Sie den Namen jedes beliebigen, existierenden blobspace angeben.

### ONLINE-Datentypen abfragen

Der interaktive Editor von INFORMIX zeigt die Ergebnisse einer Abfrage unterschiedlich an. Dies hängt ab vom jeweiligen Datentyp der abgefragten Spalte. Die Werte der Spalten mit Standard-Datentypen werden vollständig angezeigt, ebenso wie die Werte von VARCHAR-Spalten.

Wenn Sie eine TEXT-Spalte auswählen, gibt INFORMIX den Inhalt der Spalte am Bildschirm aus. Sollte der Inhalt länger als eine Bildschirmseite, sein, können Sie ihn mit dem Menüpunkt *Naechster* durchblättern. Haben Sie eine BYTE-Spalte in Ihre Abfrage miteinbezogen, wird nicht der Inhalt der Spalte ausgegeben, sondern lediglich die Meldung `<BYTE value>` angezeigt.

## Änderungen für Formate

Dieser Abschnitt beschreibt die Änderungen, die sich in der Version V4.1 für Formatprogramme ergeben (siehe auch Kapitel 3 und 4). Die in Klammern angegebenen Kapitel- und Seitenangaben beziehen sich auf das Handbuch INFORMIX-SQL-Nachschlagen[2].

### Neues Attribut - INVISIBLE

Mit INFORMIX-SQL können Sie jetzt *INVISIBLE* als Attribut (siehe auch Seite 3-36) für ein Bildschirmfeld benutzen. Wenn Sie für ein Bildschirmfeld in Ihrem Formatprogramm *INVISIBLE* definiert haben, werden die Zeichen, die Sie eingeben, nicht angezeigt. Sie können während der Eingabe den Cursor in diesem Feld nicht bewegen.

Als zweites Attribut sollten Sie *AUTONEXT* angeben, da nur bei dieser Kombination ein Piepton signalisiert, wann die Grenze des Feldes erreicht ist. Sie gelangen dann automatisch in das nächste Feld.

Ohne *AUTONEXT* ertönt kein Signal, wenn zu viele Zeichen eingegeben werden. Die Eingabe wird rechts abgeschnitten.

INFORMIX ignoriert alle Farbattribute in Verbindung mit *INVISIBLE*. Nur das Attribut *COLOR=REVERSE* ist möglich.

Auch wenn Sie zu einem *INVISIBLE*-Feld das Attribut *PICTURE* definieren, zeigt INFORMIX die Werte nicht an.

Benutzen können Sie das Attribut *INVISIBLE* wie jedes andere Attribut im *ATTRIBUTES*-Abschnitt eines Formatprogramms, wie das folgende Beispiel zeigt.

### Beispiel

Nachdem Sie ein Standardformat erzeugt haben, hier für die Tabelle *kunden* der Datenbank *versand*, können Sie mit der Menüoption *Modifizieren* das Format nach Ihren Vorstellungen abändern.

```

DATABASE stores
SCREEN SIZE 24 BY 80
{
kunden_nr           [f000      ]
vorname             [f001        ]
nachname            [f002        ]
firma               [f003          ]
adresse1            [f004          ]
adresse2            [f005          ]
ort                 [f006          ]
bundesland          [a0]
plz                  [f007      ]
telefon             [f008          ]
}
END

```

```

TABLES
customer
ATTRIBUTES
f000 = kunden.kunden_nr, INVISIBLE;
f001 = kunden.vorname;
f002 = kunden.nachname;
f003 = kunden.firma;
f004 = kunden.adresse1;
f005 = kunden.adresse2;
f006 = kunden.ort;
a0 = kunden.bundesland;
f007 = kunden.plz;
f008 = kunden.telefon;
END

```

Die Spalte *kunden\_nr* ist durch das *INVISIBLE*-Attribut als nicht sichtbar definiert, und die Werte werden bei einer Abfrage nicht angezeigt.

PERFORM: Suchen Vorw. Rueckw. BLOB Neuaufnahmen Korrigieren ...	
Sucht in der aktiven Datebanktabelle. *1:kunden Tabelle*	
kunden_nr	[ ]
vorname	[Johannes ]
nachname	[Partellmann ]
firma	[Olympia Sport ]
adresse1	[Spinosastr. 10 ]
adresse2	[ ]
ort	[Ulm ]
bundesland	[BY]
plz	[7900]
telefon	[0731/6011232 ]
26 Saetze gefunden	

Mit der Menüoption *PRINT* können Sie sich die Werte einer *INVISIBLE*-Spalte folgendermaßen anzeigen lassen:

- ▷ Lassen Sie sich den Bildschirm mit dem *INVISIBLE*-Wert anzeigen, den Sie sehen möchten.
- ▷ Wählen Sie im *PERFORM*-Menü die Menüoption *PRINT*
- ▷ Geben Sie der Datei einen Namen (Drücken Sie RETURN, um den Standardwert zu wählen).

- ▷ Wählen Sie zwischen *Anhängen* und *Erstellen*.  
Mit der Option *Anhängen* wird die Ausgabe an den Inhalt der Datei angehängt, mit *Erstellen* wird der Inhalt überschrieben.
- ▷ Wählen Sie zwischen *Aktueller Liste* und *Bildschirm*.  
Mit *Aktueller Liste* werden alle Informationen der Abfrage in die Ausgabedatei geschrieben. Mit der Option *Bildschirm* bestimmen Sie, daß nur der Inhalt des Bildschirms ausgegeben wird.
- ▷ Wählen Sie *ASCII-Format* im Gegensatz zum *Bildschirm-Format*.  
Mit dem *ASCII-Format* werden alle Werte ausgegeben, auch die als *INVISIBLE* definierten.

### Beispiel

Die Ausgabe des vorangegangenen Beispiels sieht dann folgendermaßen aus.

```
*perform.out* 1 linie, 88 zeichen
116|Johann|Partellmann|Olympia Sport|Spinosastr.10|U1m|BY|7900|0713
/601123|
```

Die Kundennummer wird mit den anderen Werten ausgegeben.

### BLOB-Daten anzeigen

Mit dem Menüpunkt *Suchen* im *PERFORM*-Menü (siehe auch Seite 4-17) können Sie sich den Inhalt von TEXT-Spalten am Bildschirm ausgeben lassen. Der Inhalt von BYTE-Spalten wird nur dann ausgegeben, wenn Sie im ATTRIBUTES-Abschnitt Ihres Format-Programms für *PROGRAM* einen aufrufbaren Editor angegeben haben (z.B. PROGRAM=vi).

Sie können sich die BLOB-Daten nur anschauen, nicht ändern.

```
PERFORM: Suchen Vorw. Rueckw. View Neuaufnahmen Korrigieren ...
BLOB-Inhalte mit Editorkommandos anzeigen *1:kunden Tabelle*
```

Nach der Auswahl der Menüoption *View* setzt INFORMIX die Schreibmarke auf das erste TEXT-Feld oder auch BYTE-Feld (wenn Sie das Attribut *PROGRAM* angegeben haben).

Wenn Sie sich den Inhalt der BLOB-Spalte anschauen möchten, geben Sie ein Ausrufezeichen (!) ein. Mit den Tasten RETURN, TAB oder **↓** gehen Sie zum nächsten BLOB-Feld, das Sie sich anschauen können. Mit **↑** erreichen Sie das vorherige BLOB-Feld. Mit **ESC** gelangen Sie zurück ins Menü *PERFORM*.

### Optionales Suffix bei PERFORM

Die Namen für den Aufruf von Formatdateien auf Betriebssystemebene sind ab der Version 4.1 wahlfrei. Die folgenden zwei Möglichkeiten führen zum gleichen Ergebnis. Sie können Ihre Formatdatei, wie gewohnt, weiterhin mit dem Suffix **.per** aufrufen:

```
sformbuild format.per
```

Sie können die Datei aber auch ohne, oder mit einem frei gewählten Suffix aufrufen.

```
sformbuild format
```

Die von INFORMIX-SQL automatisch erstellten Formate behalten die Standardendung. Die Endungen der Dateien, die übersetzte Bildschirmformate enthalten -Endung **.frm** - können Sie ebenfalls entweder frei bestimmen oder weglassen.

### C-Anschluß für PERFORM

Dieser Abschnitt behandelt die Schnittstelle zwischen Routinen der Programmiersprache C und PERFORM. Sie können die Funktionen nur dann benutzen, wenn Sie zusätzlich INFORMIX-ESQL/C installiert haben.

PERFORM kann Formate ohne zusätzliche Änderung bearbeiten. Es könnte jedoch einmal vorkommen, daß Sie eine nicht vorhandene Funktion hinzufügen wollen. PERFORM könnte z.B. C-Funktionen zur Prüfung der Gültigkeit von Daten, zur Aufzeichnung des Datums, der Zeit und des Benutzernamens oder zur Überarbeitung und Aktualisierung der Datenbank aufrufen.

Das allgemeine Format eines PERFORM-Formatprogramms besteht aus fünf Abschnitten:

```
DATABASE-Abschnitt  
SCREEN-Abschnitt  
TABLES-Abschnitt  
ATTRIBUTES-Abschnitt  
[INSTRUCTIONS-Abschnitt]
```

C-Funktionen können Sie in Kontrollblöcken des INSTRUCTIONS-Abschnitts im Formatprogramm aufrufen.

### Aufruf der C-Funktion

Im Kontrollblock können überall dort C-Funktionen stehen, wo Sie einen Ausdruck verwenden können. Eine C-Funktion kann aber auch als Anweisung allein stehen. Mehr über Kontrollblöcke und Ausdrücke bei FORMBUILD lesen Sie im Kapitel 3 des Handbuches *INFORMIX-SQL Nachschlagen V4.0*. Die Aufrufsyntax für eine benutzerdefinierte Funktion lautet wie folgt:

```
[CALL] benfunkt([ausdruck,...])
```

#### CALL

Diese Angabe ist wahlfrei. Liefert die C-Funktion keinen Wert, muß CALL verwendet werden. Fehlt CALL, muß *benfunkt* einen Wert als Ergebnis liefern.

#### *benfunkt*

ist der Name, mit dem die C-Funktion im PERFORM-Programm angesprochen wird.

#### *ausdruck*

kann maximal 10 Ausdrücke beinhalten. Ein Ausdruck ist definiert als:

- ein Feldbezeichner
- eine Konstante
- eine Mengenfunktion
- eine C-Funktion
- die Funktion TODAY
- die Funktion CURRENT
- jede beliebige Kombination der vorhergehenden Ausdrücke, die durch die Verwendung der arithmetischen Operatoren +,-,\* und / gebildet wird.

Wenn Sie mit ONLINE arbeiten, können Sie zwar VARCHAR-Spalten an eine C-Funktion übergeben, jedoch keine Werte dieses Datentyps von einer C-Funktion zurück erhalten.

### Beispiele

```
after editadd of proj_nr
  let f001 = userfunc(f002)

before editupdate of zahldatum
  if boolfunc(f003) then
    let f004 = 15
  else
    let f004 = 10

after add update remove of kunde
  call userfunc()
```



### ON BEGINNING und ON ENDING

*ON BEGINNING* und *ON ENDING* sind zwei neue Kontrollblöcke im INSTRUCTIONS-Abschnitt, die Sie zusammen mit Aufrufen von C-Funktionen verwenden können.

Bei *ON BEGINNING* wird unmittelbar nach dem Aufruf von PERFORM die angegebene C-Funktion aufgerufen und abgearbeitet. Mit einer solchen C-Funktion können Sie z.B. Befehle abarbeiten, ein spezielles Paßwort anfordern oder eine temporäre Arbeitsdatei zur Speicherung eines Stapels von Transaktionssätzen anlegen.

Mit dem Kontrollblock *ON ENDING* wird unmittelbar nach dem Befehl EXIT eine C-Funktion aufgerufen. Hier können Sie z.B. Berechnungen durchführen, die die Änderungen zusammenfassen. Sie können die neu aufgenommenen Sätze zusammenstellen und drucken oder Arbeitsdateien löschen.

Im INSTRUCTIONS-Abschnitt können Sie mehrere ON BEGINNING- und/oder ON ENDING-Kontrollblöcke verwenden. Es darf jedoch nur eine CALL-Anweisung in jedem Kontrollblock enthalten sein.

#### Beispiel

```
ON BEGINNING
    CALL benfunkt(ausdruck,...)
ON ENDING
    CALL benfunkt(ausdruck,...)
```

### Übersetzen des Formatprogramms

Mit FORMBUILD übersetzen Sie ein Formatprogramm, unabhängig davon, ob es C-Funktionsaufrufe enthält oder nicht. Wenn *prodat* Ihr Formatprogramm inclusive C-Funktionen ist, geben Sie folgenden Befehl ein:

```
sformbl prodat
```

Die Standard-Endung *.per* ist auch hier optional. Für weitere Informationen siehe Abschnitt *C-Anschluß* in diesem Kapitel.

## Änderungen für ACE

Dieser Abschnitt beschreibt die Änderungen, die sich in der Version V4.1 für Listen ergeben (siehe auch Kapitel 5). Die in Klammern angegebenen Seitenangaben beziehen sich auf das Handbuch INFORMIX-SQL-Nachschlagen[2].

### Neue Anweisung im OUTPUT-Abschnitt

Im OUTPUT-Abschnitt (siehe auch Seite 5-18) legen Sie das Papierformat und das Ausgabemedium für die Liste fest.

Mit der Anweisung *TOP OF PAGE* veranlassen Sie den Drucker an bestimmten Stellen des Listenprogramms zu einem Seitenvorschub. Die *TOP OF PAGE*-Anweisung gilt nur innerhalb eines REPORTS und hat folgende Syntax:

```
TOP OF PAGE "char-zeichenfolge"
```

#### char-zeichenfolge

besteht aus ein oder zwei alphanumerischen Zeichen, die Ihren Drucker zu einem Seitenvorschub veranlassen. Bei den meisten Druckern ist dies die Zeichenfolge  $\wedge L$ , der Ascii-Code für einen Seitenvorschub (sehen Sie dazu auch in die Gebrauchsanweisung für Ihren Drucker).

Ist das erste Zeichen der Zeichenfolge ein  $\wedge$ , versteht ACE die Zeichenfolge als Control-Sequenz. Ohne  $\wedge$  wird das erste Zeichen als druckerspezifischer Wert für einen Seitenumbruch interpretiert.

ACE setzt den Wert für einen Seitenvorschub immer dann ein, wenn im Listenprogramm eine neue Seite erforderlich ist. Ein Seitenvorschub ist dann nötig, wenn

- das Seitenende erreicht ist
- eine *SKIP TO TOP OF PAGE*-Anweisung ausgeführt wird
- eine *SKIP n LINES*-Anweisung mehr Zeilen beansprucht, als auf der aktuellen Seite vorhanden sind
- eine *NEED*-Anweisung mehr Zeilen beansprucht, als auf der aktuellen Seite vorhanden sind

Wenn Sie keine *SKIP TO TOP OF PAGE*-Klausel angegeben oder einen *PAGE TRAILER* definiert haben, wird die *TOP OF PAGE*-Klausel nicht beachtet.

Wenn Sie zugleich eine *TOP OF PAGE*- und eine *BOTTOM MARGIN*-Anweisung definiert haben, wird *BOTTOM MARGIN* ignoriert. Fehlt eine *TOP OF PAGE*-Klausel, füllt ACE nach *SKIP TO TOP OF PAGE* die Seite standardmäßig mit Leerzeilen, bis das Seitenende erreicht ist.

## Beispiel

```

REPORT kunden_liste (r_kunde)

  DEFINE r_kunde RECORD LIKE kunde.*

  OUTPUT
    TOP OF PAGE " "
    REPORT TO "liste"

  SELECT * FROM kunde INTO kunde.*

  FORMAT EVERY ROW
END REPORT

```

## Sortierung im SELECT-Abschnitt

Im SELECT-Abschnitt (siehe auch Seite 5-21) wählen Sie die Spalten aus einer oder mehrerer Tabellen, die im Listenprogramm bearbeitet und ausgegeben werden sollen. Dabei können Sie mehrere oder verschachtelte SELECT-Anweisungen angeben. Die Beschränkung, daß nur die letzte SELECT-Anweisung eine ORDER BY-Sortierung enthalten darf, ist aufgehoben.

## Veränderte Reihenfolge im FORMAT-Abschnitt

Im FORMAT-Abschnitt (siehe auch Seite 5-28) wird für jede neue Seite im REPORT der *PAGE HEADER*-Kontrollblock durchlaufen, für jeden gelesenen Satz wird *ON EVERY ROW* ausgeführt.

Bisher hat ACE bei einer neuen Seite zuerst die *PAGE HEADER*-Anweisung und dann die *ON EVERY ROW*-Anweisung abgearbeitet. In der Version 4.1 führt ACE die *PAGEHEADER*-Anweisung erst aus, wenn die erste *PRINT*-, *SKIP*- oder *NEED*-Anweisung erreicht wird.

Das hat den Vorteil, daß Spalten, die Sie mit einem Ordnungsbegriff (z.B. *BEFORE GROUP OF*) sortieren, im *PAGE HEADER* und bei *ON EVERY ROW* mit denselben Sätzen versorgt werden. So können Sie mit dem *PAGE HEADER* auf Werte des aktuellen Datensatzes zugreifen.

Das bedeutet jedoch, daß die Variablen, denen Sie neue Werte im *PAGE HEADER* zuweisen, erst ab der nächsten erreichten *PRINT*-, *SKIP*- oder *NEED*-Anweisung bei *ON EVERY ROW* mit diesen neuen Werten initialisiert sind.

Sollten Sie mit REPORTS arbeiten, bei denen in der *PAGE HEADER*-Anweisung Variablen verwendet werden, die das Verhalten der *ON EVERY ROW*-Anweisung kontrollieren, kann das im Listenausdruck zu veränderten Ergebnissen führen.

### Erweiterung der PRINT-Anweisung

Sie können mit der *PRINT*-Anweisung (siehe auch Seite 5-38) NULL-Zeichen in Ihrem Listenprogramm ausgeben. Dies geschieht mit der ASCII-Funktion, der Sie den Wert 0 zuordnen. Die ASCII-Funktion gibt nur in Verbindung mit der *PRINT*-Anweisung ein NULL-Zeichen aus, ohne PRINT gibt sie ein Leerzeichen aus.

#### Beispiel

```
ON EVERY ROW PRINT ASCII 0;
```

### ONLINE-Datentypen im REPORT

Sie können VARCHAR- und TEXT-Spalten in einem Listenprogramm bearbeiten, jedoch keine BYTE-Spalten (siehe auch Seite 5-13).

In Ausdrücken verhalten sich VARCHAR-Spalten und -Variablen wie CHARACTER-Spalten und -Variablen. Wenn Sie im REPORT eine VARCHAR-Spalte definieren, dürfen Sie die Untergrenze nicht angeben. Die Obergrenze müssen Sie angeben.

Sie können die Obergrenze der Variablen aber auch kleiner angeben, als die entsprechende Spalte, wenn Sie nur einen Teil der Spalte ausgeben möchten.

#### Beispiel

```
DEFINE bemerkung VARCHAR(120)
```

Eine TEXT-Spalte dürfen Sie nicht angeben in einem

- arithmetischen Ausdruck
- bool'schen Ausdruck
- Mengenfunktion
- *BEFORE GROUP OF*-Anweisung
- *AFTER GROUP OF*-Anweisung

In der *PRINT*-Anweisung können Sie die TEXT-Spalte angeben. Die *PRINT*-Anweisung verhält sich in diesem Fall wie die *PRINT FILE*-Anweisung, die die TEXT-Spalte als zu druckende Datei behandelt.

### Paralleles Sortieren (nur für INFORMIX-ONLINE)

INFORMIX stellt für das ONLINE-Backend eine verbesserte Sortiermöglichkeit zur Verfügung. *Psort* kann mehrere Prozessoren benutzen, um verschiedene Sortierprozesse zu starten und zu steuern.

ONLINE benutzt *Psort*

- bei Abfragen (*ORDER BY*-Sortierung in einer *SELECT*-Abfrage)
- beim Löschen von doppelten Sätzen (*UNIQUE*- oder *DISTINCT*-Anweisung)

ONLINE sortiert nur parallel, wenn dadurch eine bessere Performanz, also Zeitersparnis, gewährleistet ist. Bei kleineren Satzmengen oder bei einem existierenden Index über die zu sortierende Spalte wird *Psort* nicht aktiviert.

*Psort* läßt mehrere Sortierläufe im Speicher ablaufen und schreibt die Ergebnisse auf die Platte. Anschließend werden die verschiedenen Ergebnisse zu einem einzigen Ergebnis zusammengefaßt. ONLINE errechnet die benötigte Anzahl von Prozessen auf Grund des Umfangs der Abfrage und der Anzahl der Prozessoren im System.

Mit den Umgebungsvariablen *PSORT\_NPROCS* und *PSORT\_DBTEMP* können Sie festlegen, wie viele Prozesse gleichzeitig gestartet werden können und in welchem Dateiverzeichnis die Zwischenergebnisse der Sortierläufe abgelegt werden sollen (siehe auch im Anhang dieses Handbuchs bei *Umgebungsvariablen*).

### Optionales Suffix bei ACE

Die Namen für den Aufruf von Listenprogrammen auf Betriebssystemebene sind ab Version 4.1 wahlfrei. Die folgenden zwei Möglichkeiten führen zum gleichen Ergebnis. Sie können Ihr Listenprogramm, wie gewohnt, weiterhin mit dem Suffix **.ace** aufrufen:

```
saceprep liste.ace
```

Sie können die Datei aber auch ohne, oder mit einem frei gewählten Suffix aufrufen.

```
saceprep liste
```

Die von INFORMIX-SQL automatisch erstellten Listen erhalten weiterhin die Endung **.ace**.

Die Endung der Dateien, die übersetzte Listenprogramme enthalten, mit der Endung **.arc**, können Sie ebenfalls frei bestimmen.

## C-Anschluß für ACE

Der Abschnitt beschreibt, wie Sie C-Funktionen für ACE vom Listenprogramm aus aufrufen können. Um die Einbindung von C-Funktionen zu ermöglichen, muß INFORMIX-ESQL/C auf Ihrem Rechner installiert sein.

ACE kann Ihre Datenbanklisten ohne zusätzliche Änderung bearbeiten, es könnte jedoch einmal vorkommen, daß Sie eine nicht bereitgestellte Funktion hinzufügen wollen. Eine von ACE aufgerufene C-Funktion könnte beispielsweise über die in einer Liste enthaltenen Daten statistische Berechnungen durchführen und die Ergebnisse in die Liste aufnehmen.

Ein Listenprogramm besteht aus mindestens drei und maximal sechs Abschnitten. Die Reihenfolge müssen Sie einhalten. Näheres zu Syntax und Bedeutung können Sie in INFORMIX-SQL-Nachschlagen[2] V4.0 Kapitel 5 nachlesen.

```
DATABASE-Abschnitt
[DEFINE-Abschnitt]
[INPUT-Abschnitt]
[OUTPUT-Abschnitt]
{
  SELECT-Abschnitt
  READ-Abschnitt
}
FORMAT-Abschnitt
```

Den Funktionsnamen vereinbaren Sie im DEFINE-Abschnitt, aufgerufen wird die Funktion dann im FORMAT-Abschnitt. Mit ACEPREP übersetzen Sie danach das Listenprogramm.

### Vereinbaren von C-Funktionen

Sie vereinbaren eine C-Funktion im DEFINE-Abschnitt des Listenprogramms.

```
DEFINE
  FUNCTION benfunkt
END
```

**DEFINE**

leitet den DEFINE-Abschnitt ein.

**FUNCTION**

ist das einleitende Schlüsselwort, um eine Funktion zu vereinbaren. Sie können auch mehrere Funktionen vereinbaren, die jeweils wieder mit *FUNCTION* eingeleitet werden.

*benfunkt*

ist der Name, mit dem die C-Funktion im Programm angesprochen wird. *benfunkt* muß den Syntaxregeln eines ACE-Bezeichners entsprechen. Nach dem Funktionsnamen dürfen Sie keine Klammern angeben!

**END**

beendet den DEFINE-Abschnitt.



Neben der FUNCTION-Anweisung können Sie weiterhin PARAM-, VARIABLE- und ASCII-Deklarationen vereinbaren.

**Aufruf von C-Funktionen**

Bei der Ortsangabe im FORMAT-Abschnitt können Sie einen oder mehrere Kontrollblöcke angeben, die festlegen, wann ACE eine bestimmte Anweisung ausführt.

```

ortsangabe := [
PAGE HEADER
PAGE TRAILER
FIRST PAGE HEADER
ON EVERY ROW
ON LAST ROW
BEFORE GROUP OF
AFTER GROUP OF
]

```

Jeder Ortsangabe folgen eine oder mehrere Anweisungen, die entsprechend der Ortsangabe ausgeführt werden. Hier können Sie auch eine oder mehrere C-Funktionen aufrufen. Näheres zu Kontrollblöcken und Anweisungen lesen Sie bitte in Kapitel 5 des Handbuchs INFORMIX-SQL-Nachschlagen[2] nach.

```
[CALL] benfunkt([ausdruck,...])
```

**CALL**

Diese Angabe ist wahlfrei. Liefert die C-Funktion keinen Wert, muß CALL verwendet werden. Fehlt CALL, muß *benfunkt* einen Wert als Ergebnis liefern.

Eine C-Funktion kann in einem Ausdruck überall dort vorkommen, wo eine Konstante verwendet werden kann. In diesem Fall dürfen Sie CALL nicht angeben und die C-Funktion muß einen Wert als Ergebnis liefern.

*benfunkt*

ist der Name einer C-Funktion, die vorher im DEFINE-Abschnitt vereinbart worden ist.

*ausdruck*

kann maximal 10 Ausdrücke beinhalten, die durch Kommata getrennt sind. Ein Ausdruck kann eine einfache numerische oder alphanumerische Konstante sein, eine komplexe Reihe von Spaltennamen, ACE-Variablen, ACE-Parametern, ACE-Funktionen (wie etwa Mengen- und Datumsfunktionen), eine unter Hochkommata gestellte Zeichenkette oder eine Verknüpfung von arithmetischen und logischen Operatoren und Schlüsselwörtern.

Wenn Sie mit ONLINE arbeiten, können Sie zwar VARCHAR-Spalten an eine C-Funktion übergeben, jedoch keine Werte dieses Datentyps von einer C-Funktion zurück erhalten.

## Beispiele

Dieser Kontrollblock ruft die C-Funktion *stat* auf, die Statistiken über die Daten jener Sätze berechnet, für die die Auftragsnummer = *auftrags\_nr* ist.

```
after group of auftrags_nr
  call stat(auftrags_nr)
```

Dieser Kontrollblock gibt die Auftragsnummer und einen Wert aus, der den Gesamtpreis jedes Auftrags mit der Zeitspanne, in der der Auftrag ausständig gewesen ist, in Beziehung setzt. Er ruft eine C-Funktion auf, die den Logarithmus berechnet.

```
on every row
  print auftrags_nr,
    logarithm((total of gesamtpreis)/(today - auftragsdatum))
```

Dieser Kontrollblock wurde ACE-Beispiel 1 am Ende dieses Kapitels entnommen. Er gibt das Systemdatum und die Systemzeit am oberen Rand der ersten Seite der Liste aus. Die Funktion *to\_unix* übergibt ihr Argument (eine Zeichenkette) an das Betriebssystem SINIX.

```
first page header
  call to_unix("date")
```



### Übersetzen des Listenprogramms

Mit ACEPREP übersetzen Sie ein Listenprogramm, gleichgültig ob es C-Funktionsaufrufe enthält oder nicht. Wenn *prodat* die Datei ist, die ein Listenprogramm einschließlich C-Funktionen enthält, geben Sie folgenden Befehl ein:

```
saceprep prodat
```

Auch hier ist die Standard-Endung *.ace* optional. Lesen Sie dazu bitte mehr in Kapitel 5 des Handbuchs *INFORMIX-SQL-Nachschlagen*[2].

## C-Anschluß

In den vorhergehenden Abschnitten haben Sie erfahren, wo Sie bei ACE und PERFORM C-Funktionen verwenden können und wie Sie diese vereinbaren. Dieser Abschnitt beschreibt, wie diese C-Funktionen aufgebaut sein müssen, damit Sie sie in einem Format- oder Listenprogramm verwenden können. Er zeigt auch, wie Sie Werte an C-Funktionen übergeben können und von diesen zurückgeliefert werden.

Wenn Sie innerhalb eines Listen- oder Formatprogramms C-Funktionen verwenden wollen, müssen Sie ein C-Programm schreiben, das die entsprechenden Include-Dateien und Strukturvereinbarungen sowie Ihre Funktionen enthält. Übersetzen Sie anschließend Ihr Programm und fügen Sie ihm die entsprechenden Bibliotheken hinzu.

C-Funktionen können die Bibliotheksroutinen benutzen, die im ESQL/C-Handbuch[8] beschrieben sind. Sie können INFORMIX-ESQL/C-Anweisungen enthalten und mathematische Funktionen oder andere C-Funktionen aufrufen. PERFORM kann in C-Funktionen zusätzlich die in diesem Kapitel beschriebenen Sonderfunktionen verwenden. Durch die Möglichkeit, benutzerdefinierte C-Funktionen einzubinden, erhöht sich die Leistungsfähigkeit und Flexibilität von ACE und PERFORM.

### C-Programmstruktur

Um ein Listen- oder Formatprogramm zu erzeugen, das Ihre Funktionen einbezieht, müssen Sie ein C-Programm schreiben, das die entsprechenden Vereinbarungen enthält. Ihr Programm kann eine oder mehrere Funktionen beinhalten, und Sie können andere Funktionen zur internen Verwendung in Ihrem Programm definieren.

#### Beispiel

Das folgende Beispiel zeigt die allgemeine Struktur eines solchen C-Programms, das zwei benutzerdefinierte Funktionen beinhaltet:

```
$include ctools.h
/* Weitere includes möglich wie benötigt */

valueptr funct1();
valueptr funct2();

struct ufunc userfuncs[] =
{
  "myfunct1", funct1,
  "myfunct2", funct2,
  0,0
};

/* Weitere globale Vereinbarungen hinzufügen */
```

```

valueptr funct1()
{
    :
    :
    /* funct1 erhält keine Argumente und
       liefert eine Zeichenkette */
    :
    :
    strreturn(s, len);
}

valueptr funct2(arg1, arg2)
valueptr arg1, arg2;
{
    :
    :
    /* funct2 erhält zwei Argumente
       und liefert kein Ergebnis */
    :
    :
}

```

Der Aufbau des C-Programms ist im folgenden erläutert.

Am Beginn Ihres C-Programms muß folgende Zeile stehen:

```
$include ctools.h
```

Die Include-Datei *ctools.h* enthält die benötigte Konstantendefinition und ist am Ende dieses Kapitels aufgeführt.

*ctools.h* fügt automatisch die Include-Dateien *value.h*, *datetime.h* und *sqltypes.h* in Ihr C-Programm ein.

Je nach Bedarf können Sie auch andere Dateien, wie etwa *math.h* oder *stdio.h*, einfügen. Wenn Sie INFORMIX-ESQL/C verwenden, wird *sqlca.h* automatisch eingebunden und Sie können weitere Include-Dateien miteinbeziehen.

Bevor Sie *ufunc* initialisieren, müssen Sie Ihre Funktionen vereinbaren. *ctools.h* enthält die Definition der Struktur *value* und Zeiger auf diese Struktur.

```

typedef struct value *valueptr;
typedef struct value *acevalue;
typedef struct value *perfvalue;

```

Die beiden letzten Zeiger sollen die Kompatibilität mit früheren Versionen von INFORMIX gewährleisten. Alle Ihre Funktionen müssen vom Typ *valueptr* sein. Sind *funct1()* und *funct2(arg1,arg2)* Ihre Funktionen, müssen Sie diese als nächstes vereinbaren.

```
valueptr funct1();
valueptr funct2();
```

Führen Sie in Ihrem Programm die Strukturvereinbarung und -initialisierung für *userfuncs[]* als nächsten Schritt durch. Diese Strukturen sind erforderlich, damit ACE und PERFORM Ihre Funktionen zur Laufzeit aufrufen können.

```
struct ufunc userfuncs[] =
{
    "myfunct1", funct1,
    "myfunct2", funct2,
    0,0
};
```

Die in Anführungsstriche gestellten Zeichenketten "*myfunct1*" und "*myfunct2*" müssen namensidentisch mit den im ACE- bzw. PERFORM-Programm verwendeten Funktionen sein. *funct1* und *funct2*, die "*myfunct1*" bzw. "*myfunct2*" entsprechen, sind Zeiger auf die im C-Programm definierten Funktionen. Beachten Sie, daß die hier definierten C-Funktionen nicht die gleichen Namen haben müssen wie in Ihrem ACE- bzw. PERFORM-Programm. Das Array *userfuncs* soll die Verbindung zwischen diesen beiden Namen herstellen. Die beiden Nullen am Ende des Arrays sind als Endezeichen erforderlich.

Den letzten Teil des C-Programms bilden Ihre Anweisungen. Wie bereits vorher erwähnt, müssen alle in ACE oder PERFORM aufgerufenen Funktionen so vereinbart werden, daß sie als Ergebnis einen Zeiger auf eine Struktur vom Typ *value* liefern. Zusätzlich müssen alle Argumente Ihrer Funktionen vom Typ *valueptr* sein.

Der Aufruf von *strreturn* in der Definition von *funct1* ist ein Beispiel für die Verwendung eines von mehreren Makros, die als Ergebnis Werte vom Typ *valueptr* liefern. Diese und andere Konvertierungsfunktionen werden im nächsten Abschnitt beschrieben.

## Eingabeparameter

Die Include-Datei *ctools.h* ermöglicht es, Werte von ACE und PERFORM an eine C-Funktion zu übergeben. Bei dieser Übergabe müssen die Datentypen der übergebenen Werte mit den Datentypen der Parameter C-Funktion übereinstimmen. Um diese Übereinstimmung zu erreichen, gibt es in einem C-Programm zwei Möglichkeiten:

- Prüfung des Datentyps
- Datentyp-Konvertierung

### Prüfung des Datentyps

Mit den folgenden Definitionen können Sie den Typ der Daten prüfen, die an die C-Funktion übergeben werden. Wenn beispielsweise der Parameter *arg* an die C-Funktion übergeben wird, können Sie mit folgenden Definitionen den Datentyp von *arg* feststellen und den Wert ausgeben.

Definition	Rückgabe
<code>arg-&gt;v_charp</code>	Zeiger auf string
<code>arg-&gt;v_len</code>	Länge von string
<code>arg-&gt;v_int</code>	INTEGER-Wert
<code>arg-&gt;v_long</code>	LONG-Wert
<code>arg-&gt;v_float</code>	FLOAT-Wert
<code>arg-&gt;v_double</code>	DOUBLE-Wert
<code>arg-&gt;v_decimal</code>	DECIMAL-, MONEY-, DATETIME-, oder INTERVAL-Wert
<code>arg-&gt;v_type</code>	Datentyp
<code>arg-&gt;v_ind</code>	Indikatorvariabel auf NULL
<code>arg-&gt;v_prec</code>	DATETIME bzw. INTERVAL-Datumskomponente

Der Datentyp von *arg* kann durch den Vergleich von `arg->v_type` mit den in *ctools.h* definierten Integer-Konstanten bestimmt werden.

v_type	SQL-Typ	C-Typ
SQLCHAR	CHAR	{ char string fixchar
SQLSMINT	SMALLINT	short
SQLINT	INTEGER	long
SQLFLOAT	FLOAT	double
SQLSMFLOAT	SMALLFLOAT	float
SQLDECIMAL	DECIMAL	dec_t
SQLSERIAL	SERIAL	long
SQLDATE	DATE	long
SQLMONEY	MONEY	dec_t
SQLDTIME	DATETIME	dtime_t
SQLINTERVAL	INTERVAL	intrvl_t

Hat **arg->v\_type** den Wert SQLCHAR, dann ist der Zeiger auf die Zeichenkette in **arg->v\_charp** und die Zeichenketten-Länge in **'arg->v\_len** verfügbar. Die Zeichenkette endet nicht mit einem Nullbyte.

**arg->v\_ind** ist negativ, wenn der Wert von *arg* NULL ist. Ansonsten ist **arg->v\_ind** gleich 0.

Wenn Sie mit ONLINE arbeiten, können Sie mit der zusätzlichen Definition für **arg->v\_type** prüfen, ob es sich bei dem übergebenen Parameter um den Datentyp VARCHAR handelt.

<b>v_type</b>	<b>SQL-Typ</b>	<b>C-Typ</b>
SQLVCHAR	VARCHAR	char

Werte vom Datentyp TEXT oder BYTE können Sie nicht an C-Funktionen übergeben.

### Datentyp-Konvertierung

Die Include-Datei *ctools.h* bietet eine Alternative zur Bestimmung des Parametertyps, der von ACE oder PERFORM übergeben wird. Die unten angeführten Funktionen konvertieren einen Parameter, der als Zeiger auf eine Struktur vom Typ **value** übergeben wurde, in einen C-Datentyp Ihrer Wahl.

<b>Funktion</b>	<b>Ergebnis</b>
toint	int
tolong	long
tofloat	double
todouble	double
todate	long
todecimal	dec_t
todatetime	dtime_t
tointerval	intrvl_t

Diese Funktionen erfordern einen Zeiger auf eine Struktur vom Typ *value* und liefern als Ergebnis einen Wert des angegebenen Typs. *todecimal*, *todatetime* und *tointerval* brauchen jeweils ein zweites Argument:

Funktion	zweites Argument
<i>todecimal</i>	Zeiger auf Struktur <i>dec_t</i>
<i>todatetime</i>	Zeiger auf Struktur <i>dtime_t</i>
<i>tointerval</i>	Zeiger auf Struktur <i>intrvl_t</i>

Ist die Datentyp-Konvertierung nicht erfolgreich, wird der globalen Integer-Variablen *toerrno* ein negativer Wert zugewiesen; *toerrno* hat den Wert 0, wenn die Konvertierung erfolgreich ist.

## Rückgabewerte

Wenn Ihre Funktion einen Wert an ACE oder PERFORM zurückliefern soll, muß der Wert in eine Struktur vom Typ *value* eingetragen und ein Zeiger auf diese Struktur als Ergebnis geliefert werden. Um dies für Sie durchzuführen, enthält *ctools.h* folgende Makros:

Makro	Ergebnis
<i>intreturn(i)</i>	integer i
<i>lngreturn(l)</i>	long l
<i>floreturn(f)</i>	float f
<i>dubreturn(d)</i>	double d
<i>strreturn(s,c)</i>	string s mit Länge c (short)
<i>decreturn(d)</i>	decimal d (Typ <i>dec_t</i> )
<i>dtimereturn(d)</i>	datetime d (Typ <i>dtime_t</i> )
<i>invreturn(i)</i>	interval i (Typ <i>intrvl_t</i> )

Verwenden Sie das entsprechende Makro auch dann, wenn Sie nur eine Fehlerbedingung als Ergebnis liefern wollen. Verwenden Sie nicht einfach die *return*-Anweisung.

Da *strreturn(s,c)* einen Zeiger auf die Zeichenkette *s* liefert, achten Sie darauf, daß Sie *s* als eine Variable der Speicherklasse *static* oder *extern* definieren.

Sie können folgendes Makro aus der Include-Datei *ctools.h* verwenden, um VARCHAR-Werte zurückzugeben:

Makro	Ergebnis
<i>vcharreturn(s,c)</i>	string s mit Länge c (short)

Sie können keine Werte vom Datentyp TEXT oder BYTE zurückgeben.

## Spezielle Bibliotheksfunktionen von PERFORM

Mit fünf C-Funktionen können Sie PERFORM-Bildschirme innerhalb von C-Funktionen steuern:

pf_gettype	bestimmt Typ und Länge eines Bildschirmfeldes.
pf_getval	liest einen Wert von einem Bildschirmfeld ein.
pf_putval	weist einem Bildschirmfeld einen Wert zu.
pf_nxfield	setzt den Cursor auf ein bestimmtes Feld.
pf_msg	schreibt eine Meldung am unteren Rand des Bildschirms.

Diese Funktionen werden auf den folgenden Seiten ausführlich beschrieben. Ihr Returnwert ist 0, wenn die Funktion erfolgreich ist, ansonsten liefert sie einen Fehlercode ungleich 0.



## PF\_GETTYPE

*pf\_gettype* liefert als Ergebnis den SQL-Datentyp und die Länge des Bildschirmfeldes für einen angegebenen Feldbezeichner.

```
pf_gettype(tagname, type, len)
char *tagname;
short *type, *len;
```

### *tagname*

ist eine Zeichenkette. Sie enthält den Feldbezeichner, der ein Bildschirmfeld spezifiziert.

### *type*

ist ein Zeiger auf eine short integer-Zahl, die den Datentyp des Bildschirmfeldes *tagname* beschreibt.

Die folgende Aufstellung der möglichen Werte von *type* ist in *ctools.h* definiert:

<b>type</b>	<b>SQL-Typ</b>
SQLCHAR	CHAR
SQLSMINT	SMALLINT
SQLINT	INTEGER
SQLFLOAT	FLOAT
SQLSMFLOAT	SMALLFLOAT
SQLDECIMAL	DECIMAL
SQLSERIAL	SERIAL
SQLDATE	DATE
SQLMONEY	MONEY
SQLDTIME	DATETIME
SQLINTERVAL	INTERVAL
SQLVCHAR	VARCHAR
SQLTEXT	TEXT
SQLBYTES	BYTES

### *len*

ist ein Zeiger auf eine short integer-Zahl. Er enthält die Länge des Bildschirmfeldes *tagname* am PERFORM-Bildschirm.

### Rückgabewert

- 0 Die Funktion war erfolgreich; das Bildschirmfeld wurde gefunden.
- 3759 Dieser Feldbezeichner ist im Format nicht vorhanden.

## PF\_GETVAL

*pf\_getval* liefert den in einem Bildschirmfeld enthaltenen Wert und falls es sich um ein Feld vom Typ CHAR handelt - dessen Länge.


```
pf_getval(tagname, retvalue, valtype, vallen)
char *tagname, *retvalue;
short valtype, vallen;
```

### *tagname*

ist eine Zeichenkette. Sie enthält den Feldbezeichner, der ein Bildschirmfeld spezifiziert.

### *retvalue*

ist ein Zeiger auf eine Variable vom Typ 'string', 'short', 'long', 'float', 'double', 'decimal', 'datetime' oder 'interval', die *pf\_getval* liefern kann.

 *retvalue* muß ein Zeiger auf die Variable sein, die den Wert enthält. Es ist ein gebräuchlicher Programmierfehler, die Variable selbst zu benutzen. Das führt zu einem Systemfehler zur Laufzeit, der nicht vom Compiler erkannt wird.

Der Datentyp von *retvalue* wird durch den Wert des Parameters *valtype* bestimmt. *valtype* muß nicht genau mit dem Datentyp des Bildschirmfeldes übereinstimmen, doch sollten beide entweder numerische Felder oder vom Typ CHAR sein, sodaß PERFORM die richtige Datentyp-Konvertierung durchführen kann.

### *valtype*

ist eine Zahl von Typ 'short integer'. Sie gibt den Typ des Wertes an, auf den *retvalue* zeigen sollte.

Für *valtype* haben Sie folgende Auswahlmöglichkeiten:

<b>valtype</b>	<b>SQL-Typ</b>
CCHARTYPE	CHAR
CFIXCHARTYPE	
CSTRINGTYPE	
CINTTYPE	INTEGER
CSHORTTYPE	SMALLINT
CLONGTYPE	INTEGER, DATE, SERIAL
CFLOATTYPE	SMALLFLOAT
CDOUBLETYPE	FLOAT
CDECIMALTYPE	DECIMAL, MONEY
CDATETIME	DATETIME
CINTERVAL	INTERVAL

Ist *valtype* ein numerisches Feld und das Bildschirmfeld von Typ CHAR, so wird - soweit möglich - eine Datentyp-Konvertierung durchgeführt. Kann die Konvertierung nicht erfolgreich abgeschlossen werden, zeigt *retvalue* auf 0.

Ist *valtype* vom Typ CHAR und das Bildschirmfeld ein numerisches Feld, wird zu einer Zeichenkette konvertiert. Entspricht die Zeichenkette nicht der von *vallen* spezifizierten Länge, enthält *retvalue* die gekürzte Zeichenkette mit einem Nullbyte am Ende.

Wenn Sie mit ONLINE arbeiten, können Sie zusätzlich folgendes für *valtype* angeben:

<b>valtype</b>	<b>SQL-Typ</b>
CVCHARTYPE	VARCHAR
CLOCATORTYPE	SQLTEXT
	SQLBYTES

#### *vallen*

ist eine Zahl von Typ 'short integer', die die Länge der Zeichenkette (plus 1 für das Nullbyte am Ende) spezifiziert, die in *retvalue* als Ergebnis geliefert wird. Dies gilt jedoch nur, wenn *valtype* vom Typ CCHARTYPE ist. Für jeden anderen Wert von *valtype* wird *vallen* ignoriert.

Bei VARCHAR-Werten muß in *vallen* die Obergrenze angegeben werden, die angibt, wieviele Bytes der Puffer aufnehmen kann. Bei TEXT und BYTE zeigt *retvalue* auf eine loc\_t-Struktur. PERFORM kopiert seine interne Locator-Struktur auf Ihre Struktur.

#### **Rückgabewert**

0	Die Funktion war erfolgreich; das Bildschirmfeld wurde gefunden.
3700	Der Benutzer ist nicht berechtigt, das Feld zu lesen.
3759	Dieser Feldbezeichner ist im Format nicht vorhanden.

## PF\_PUTVAL


*pf\_putval* legt einen Wert in einem spezifizierten Feld des PERFORM-Bildschirmes ab. Der Benutzer muß die Berechtigung haben, Daten in das gewünschte Bestimmungsfeld einzutragen oder dieses zu aktualisieren.

```
pf_putval(pvalue, valtype, tagname)
    char *pvalue;
    short valtype;
    char *tagname;
```

### *pvalue*

ist ein Zeiger auf eine Variable vom Typ STRING, SHORT, INTEGER, LONG, FLOAT, DOUBLE, DECIMAL, DATETIME oder INTERVAL. Der Wert wird in das durch *tagname* bestimmte Bildschirmfeld eingetragen.

Der Datentyp von *pvalue* wird durch den Wert des Parameters *valtype* bestimmt.

 *pvalue* muß ein Zeiger auf die Variable sein, die den Wert enthält. Es ist ein gebräuchlicher Programmierfehler, die Variable selbst zu benutzen. Das führt zu einem Systemfehler zur Laufzeit, der nicht vom Compiler erkannt wird.

Bei TEXT und BYTE muß *pvalue* auf eine loc\_t-Struktur zeigen. PERFORM erwartet, daß die loc\_t-Struktur genau die gleiche Information enthält, wie die loc\_t-Struktur, die zu *tagname* gehört. Sie müssen also zuerst *pf\_getval* aufrufen, um eine Kopie zu erhalten. Danach dürfen Sie nichts mehr ändern. Sie können die loc\_t-Struktur verwenden, um den aktuellen BLOB-Wert zu ändern, welchen PERFORM in einer temporären Datei gespeichert hat.

### *valtype*

ist eine Zahl von Typ SHORT INTEGER. Sie gibt den Typ des Wertes an, auf den *pvalue* zeigt.

Für *valtype* gibt es folgende Auswahlmöglichkeiten:

<b>valtype</b>	<b>SQL-Typ</b>
CCHARTYPE } CFIXCHARTYPE } CSTRINGTYPE }	CHAR
CINTTYPE	INTEGER
CSHORTTYPE	SMALLINT
CLONGTYPE	INTEGER, DATE
CFLOATTYPE	SMALLFLOAT
CDOUBLETTYPE	FLOAT
CDECIMALTYPE	DECIMAL, MONEY
CDATETIME	DATETIME
CINTERVAL	INTERVAL

Ist *valtype* einer der CHAR-Typen und das Bildschirmfeld ein numerisches Feld, so wird - soweit möglich - eine Datentyp-Konvertierung durchgeführt. Kann die Konvertierung nicht erfolgreich abgeschlossen werden, wird der Wert 0 in das Bildschirmfeld eingetragen.

Ist der spezifizierte Typ ein numerisches Feld und das Bildschirmfeld vom Typ CHAR, findet eine Konvertierung in eine Zeichenkette statt. Paßt die Zeichenkette nicht in das Bildschirmfeld, wird die Zeichenkette rechts gekürzt.

Paßt ein Zahlenwert nicht in ein numerisches Bildschirmfeld, wird das Feld mit Sternchen gefüllt.

Wenn Sie mit ONLINE arbeiten, können Sie für *valtype* zusätzlich angeben:

<b>valtype</b>	<b>SQL-Typ</b>
CVCHARTYPE	VARCHAR
CLOCATORTYPE	SQLTEXT
	SQLBYTES

Sie können mit VARCHAR-Werten arbeiten, als ob es alphanumerischen Werte wären.

#### *tagname*

ist eine Zeichenkette, die den Feldbezeichner enthält. Dieser Feldbezeichner spezifiziert jenes Bildschirmfeld, in dem sich die Daten befinden, auf die *pvalue* zeigt.

#### **Rückgabewert**

0	Die Funktion war erfolgreich; das Bildschirmfeld wurde gefunden.
3710	Der Benutzer ist nicht berechtigt, das Feld zu aktualisieren.
3720	Der Benutzer darf dem Feld nichts hinzufügen.
3756	Das Bildschirmfeld ist nicht in der aktuellen Tabelle enthalten.
3759	Dieser Feldbezeichner ist im Format nicht vorhanden.

## PF\_NXFIELD

*pf\_nxfield* steuert die Positionierung des Cursors am PERFORM-Bildschirm. Diese Funktion wird im Eingabemodus (entweder Aufnahme eines neuen Satzes oder Aktualisierung eines alten) benötigt.

```
pf_nxfield(tagname)  
char *tagname;
```

### *tagname*

ist eine Zeichenkette, die den Feldbezeichner für das Bildschirmfeld des PERFORM-Bildschirmes enthält. Der Cursor wird auf dieses Bildschirmfeld positioniert.

Hat *tagname* den Wert EXITNOW, bewirkt *pf\_nxfield* eine sofortige Beendigung des Einfüge- oder Änderungsvorgangs, und der Satz wird eingefügt bzw. geändert. Diese Option entspricht dem Drücken der ESCAPE-Taste zur Beendigung der Transaktion.

Während eines BEFORE EDITADD- oder BEFORE EDITUPDATE-Vorgangs einer Tabelle legt *pf\_nxfield* das erste zu bearbeitende Bildschirmfeld fest.

Während eines AFTER EDITADD- oder AFTER EDITUPDATE-Vorganges einer Tabelle setzt *pf\_nxfield* den Cursor auf das angegebene Bildschirmfeld *tagname* zur weiteren Bearbeitung und trägt den Satz nicht ein.

Vor einem BEFORE EDITADD/EDITUPDATE-Vorgang oder nach einem AFTER EDITADD/EDITUPDATE-Vorgang eines Feldes legt *pf\_nxfield* das nächste zu bearbeitende Feld fest.

Nach einem AFTER ADD- oder AFTER UPDATE-Vorgang ist *pf\_nxfield* wirkungslos, da der Satz bereits eingetragen worden ist.

### Rückgabewert

- |      |  |
|------|--|
| 0    | Die Funktion war erfolgreich; das Bildschirmfeld wurde gefunden. |
| 3710 | Der Benutzer ist nicht berechtigt, das Feld zu aktualisieren.    |
| 3720 | Der Benutzer darf dem Feld nichts hinzufügen.                    |
| 3755 | Das Bildschirmfeld ist ein 'display-only'-Feld.                  |
| 3756 | Das Bildschirmfeld ist nicht in der aktuellen Tabelle enthalten. |
| 3759 | Dieser Feldbezeichner ist im Format nicht vorhanden.             |

## PF\_MSG

*pf\_msg* gibt eine Meldung am unteren Rand des Bildschirms aus.

```
pf_msg(msgstr, reverseflag, bellflag)
      char *msgstr;
      short reverseflag, bellflag;
```

### *msgstr*

ist eine Zeichenkette, die die am unteren Rand des Bildschirms ausgegebene Meldung enthält.

Bei normaler Anzeige kann die Zeichenkette bis zu 80 Zeichen umfassen. Bei Inversdarstellung ist die Höchstzahl der Zeichen kleiner als 80, da die Steuerzeichen für die Inversdarstellung an einigen Bildschirmen eine oder mehrere Stellen benötigen.

### *reverseflag*

ist eine Zahl vom Typ 'short integer', die angibt, ob die Meldung in Inversdarstellung ausgegeben wird

(0 = normale Anzeige, 1 = invers).

### *bellflag*

ist eine Zahl vom Typ 'short integer', die angibt, ob die Meldung mit akustischem Signal ausgegeben wird (0 = ohne Signal, 1 = mit Signal).



Werden mehrere Aufrufe von *pf\_msg* gleichzeitig durchgeführt, da verschiedene Bedingungen zur selben Zeit erfüllt sind, ist nur die letzte ausgegebene Meldung für den Benutzer sichtbar.

## Der Übersetzungs-, Binde- und Ablaufprozeß

Nachdem Sie Ihre C-Funktionen geschrieben haben, müssen Sie sie übersetzen und die benötigten Bibliotheksfunktionen hinzufügen und einbinden, um eine eigene Version von *sacego* oder *sperform* zu generieren.

Nach der Übersetzung Ihrer eigenen Version von *sacego* oder *sperform* können Sie Listen oder Formate mit dem folgenden Befehl ablaufen lassen:

```
custprog progdat
```

*progdat*

ist der Name des Listen- oder Formatprogramms, das Sie mit Hilfe von ACEPREP oder FORMBUILD übersetzt haben.

INFORMIX bietet Ihnen zur Vereinfachung des Übersetzungs- und Bindeprozesses Shell-Skripts an. Sie können diese Shell-Skripts genauso verwenden wie das standardmäßige C-Übersetzungs- und Bindeprogramm *cc*. Die Namen spezieller ACE-, PERFORM-, INFORMIX-ESQL/C-Bibliotheken, Präprozessoren oder die Verzeichnisse der Include-Dateien dieser Programme brauchen Sie dafür nicht zu wissen. Das Shell-Skript können Sie folgendermaßen aufrufen:

```
[CACE | CPERF] cprogram.[c | ec] [...] -o custprog other-C-list
```

**CACE**

ist das Shell-Skript, das eine eigene Version von *sacego* generiert.

**CPERF**

ist das Shell-Skript, das eine eigene Version von *sperform* generiert.

*cprogram*

ist der Name des C-Programms, das C-Funktionen enthält, wie in den vorhergehenden Abschnitten beschrieben.

**.c**

ist das zu verwendende Suffix, falls *cprogram* keine INFORMIX-ESQL/C-Anweisungen enthält.

**.ec**

ist das zu verwendende Suffix, falls *cprogram* INFORMIX-ESQL/C-Anweisungen enthält (Siehe Kapitel 1 *Verwendung von SQL* im Handbuch ESQL/C[8]).

**-o**

spezifiziert den Namen der Ausgabedatei.



*custprog*

ist der Name Ihrer selbsterstellten Version von *sacego* oder *sperform*.

*other-C-list*

die verbleibenden Argumente, die Sie an den Standardcompiler *cc* übergeben wollen.



Sie können mehrere C-Programme gleichzeitig übersetzen.

## Beispiele für C-Programme in ACE und PERFORM

Dieser Abschnitt zeigt Beispiele von ACE- und PERFORM-Anwendungen. ACE-C-Funktionen können auch mit PERFORM verwendet werden. Diese Programmbeispiele werden mit der Beispieldatenbank geliefert. Sie müssen jedoch auch über INFORMIX-SQL verfügen, um diese Programme übersetzen und ausführen zu können.

### Beispiel 1 für ACE

Das folgende Quellprogramm ruft eine Benutzerfunktion zur Ausführung eines Systemkommandos auf. Der Name des Programms in der Beispieldatenbank lautet *a\_ex1.ace*.

```
database
  versand
end

define
  function to_unix
end

select * from kunde
end

format
  first page header
    call to_unix("date")
  skip 1 line
  on every row
    print kunden_nr, 3 spaces,
      vorname clipped, 1 space, nachname
end
```

Im folgenden ist die aufgerufene Benutzerfunktion *to\_unix.c* aufgelistet.

```

#include "ctools.h"
valueptr to_unix();
struct ufunc userfuncs[] =
{
  "to_unix", to_unix,
  0,0
};

valueptr to_unix(string)
valueptr string;
{
char savearea[80];

/*Byte schreiben von string nach savearea*/
bycopy(string->v_charp, savearea, string->v_len);

/*mit Null abschliessen*/
savearea[string->v_len]=0;

system(savearea);
}

```

### Beispiel 2 für ACE

Das folgende ACE-Programm ruft eine C-Funktion auf, die die Quadratwurzel einer Zahl vom Typ DECIMAL berechnet. Die C-Funktion verwendet einige Dezimalfunktionen, die in Kapitel 4 im Handbuch INFORMIX-ESQL/C[8] beschrieben sind.

```

database
  versand
end

define
  function decsqroot
end

select a.auftrags_nr, sum(gesamtpreis) t_cost
  from auftrag a, posten p
  where a.auftrags_nr = p.auftrags_nr
  group by a.auftrags_nr
end

format
  on every row
  print auftrags_nr, t_cost
  on last row
  skip 1 line
  print "Mittelwert aller Aufträge beträgt : ",
    (total of t_cost)/count
    using "$#####.##"
  print "Standardabweichung          : ",
    decsqroot((total of t_cost*t_cost)/count
      - ((total of t_cost)/count)**2)
    using "$#####.##"
end

```

Das Programm *a\_ex2.ace* der Beispieldatenbank berechnet den Mittelwert und die Standardabweichung der Gesamtkosten aller Aufträge der Datenbank *versand*. Im folgenden ist die Benutzerfunktion *decsqrt.c* aufgelistet.

```
#include "ctools.h"
#include <math.h>

valueptr squareroot();

struct ufunc userfuncs[] =
{
    "decsqroot", squareroot,
    0, 0
};

valueptr squareroot(pnum)
valueptr pnum;
{
    double dub;
    dec_t dec;

    /* decimal in double konvertieren */
    dectodbl(&pnum->v_decimal, &dub);

    dub = sqrt(dub);

    /* double in decimal konvertieren */
    deccvdbl(dub, &dec);

    /* decimal zurueckliefern */
    decreturn(dec);
}
```

### Beispiel 1 für PERFORM

Bei Datenbankanwendungen ist es oft sinnvoll, zusammen mit den eingegebenen Daten die Identifikation des Erfassers und die Zeit der Dateneingabe festzuhalten. Es ist nicht notwendig, daß der Erfasser diese Daten eingibt. Das Betriebssystem SINIX kann den Erfasser anhand der Benutzerkennung identifizieren und die Zeit liefern. Mit Hilfe der in diesem Kapitel beschriebenen Methoden können Sie ein C-Programm schreiben, das Ihnen diese Daten vom Betriebssystem liefert und am Bildschirm ausgibt. PERFORM fügt sie dem Satz hinzu, wenn dieser in die Tabelle eingetragen wird.

Mit dem folgenden Formatprogramm können neue Kunden in die Datenbank *versand* aufgenommen werden. Das Format ist in *p\_ex1.per* der Beispieldatenbank zu finden; *stamp.c* enthält die Funktion *stamptime*.

Gehen Sie bei diesem Beispiel davon aus, daß die Tabelle *kunde* zwei weitere Felder hat: *erfasser* und *erf\_zeit*, beide vom Typ CHAR(10). Das Format zeichnet automatisch die Benutzerkennung des Erfassers und den Zeitpunkt auf, zu dem der Kunde in die Datenbank aufgenommen wird.

Der Cursor bewegt sich von links oben über die Kundendaten nach unten. Dabei wird die Reihenfolge der im ATTRIBUTES-Abschnitt angeführten Felder eingehalten. Nach dem Feld 'Telefon' springt der Cursor zum Feld 'Kundenname'. Drückt der Erfasser zur Beendigung der Transaktion die ESCAPE-Taste, wird die C-Funktion *stamptime* aufgerufen.

```

database versand
screen
{
    *****
    *                               Kundenformular                               *
    *=====
    * Nummer      :[f000          ]                                           *
    * Kundenname  :[f001          ][f002          ]                             *
    * Firma       :[f003          ]                                           *
    * Anschrift   :[f004          ]                                           *
    *             :[f005          ]                                           *
    * Ort         :[f006          ] Bundesland:[a0] PLZ:[f007 ] *
    * Telefon     :[f008          ]                                           *
    *=====
    * Erfasser   :[f009          ] Zeitpunkt der Erfassung :[f010 ] *
    *=====
}

tables
    kunde

attributes
f000 = kunde.kunden_nr, noentry;
f001 = kunde.vorname;
f002 = kunde.nachname;
f003 = kunde.firma;
f004 = kunde.adresse1;
f005 = kunde.adresse2;
f006 = kunde.ort;
a0 = kunde.bundesland, default="BW", upshift, autonext;
f007 = kunde.plz, autonext;
f008 = kunde.telefon;
f009 = kunde.erfasser;
f010 = kunde.erf_zeit;

instructions

after editadd editupdate of telefon
    nextfield = f001

after editadd editupdate of kunde
    call stamptime()

end

```

Die Funktion *stamptime* wird vom Formatprogramm aufgerufen, wenn der Erfasser zur Beendigung der Transaktion die ESCAPE-Taste drückt. Zusätzlich zu der in diesem Kapitel bereits definierten Sonderfunktion *pf\_putval* verwendet *stamptime* die Systemfunktionen *time*, *localtime* und *getlogin*.

Die Benutzererkennung des Erfassers wird von der Zeichenketten-Funktion *getlogin* geliefert und im Bildschirmfeld *erfasser* ausgegeben.

Die Systemzeit wird in Stunden und Minuten zerlegt und anschließend in angepaßter Form als Variable von Typ 'string' das Bildschirmfeld *erf\_zeit* geschrieben. Danach nimmt PERFORM den Satz unter Verwendung der Daten des Bildschirms in die Tabelle *kunde* auf.

```
#include <stdio.h>
#include <time.h>
#include "ctools.h"

valueptr stamptime();

struct ufunc userfuncs[] =
    {
        "stamptime", stamptime,
        0,0
    };

valueptr stamptime()
{
    long seconds, time();
    char usertime[10], *getlogin();
    struct tm *timerec, *localtime();

    seconds = time((long *) 0);
    timerec = localtime(&seconds);

    pf_putval(getlogin(), CCHARTYPE, "f009");

    sprintf(usertime, "%02d:%02d",
            timerec->tm_hour, timerec->tm_min);
    pf_putval(usertime, CCHARTYPE, "f010");
}
```

## Beispiel 2 für PERFORM

Normalerweise können Sie am PERFORM-Bildschirm nur Daten eines einzigen Satzes eingeben. Mit den *DISPLAY-ONLY*-Feldern, die Dateneingabe erlauben, zusammen mit Kontrollblöcken und den hier beschriebenen C-Funktionsaufrufen können Sie einen *DISPLAY-ONLY*-Bildschirm erzeugen. In diesen Bildschirm können Sie Daten für mehrere Sätze gleichzeitig eingeben. Nach Eingabe der Daten und Drücken der ESCAPE-Taste holt Ihr C-Programm die Daten vom Bildschirm und schreibt die Sätze in die Tabellen.

In dem folgenden Auftragsformular können bis zu fünf Artikel mit Mengen- und Gesamtpreisangabe aufgelistet werden. Die Erweiterung auf mehr Artikel ist natürlich sowohl im Eingabeformat als auch im darauffolgenden C-Programm möglich. Bitte beachten Sie, daß alle Bildschirmfelder entweder *DISPLAY-ONLY*- oder *LOOKUP*-Felder sind. Letztere erhalten ihre Werte von jenen *DISPLAY-ONLY*-Feldern, die eine Dateneingabe erlauben.

Nach Aufruf des PERFORM-Programms wird zur Öffnung der Datenbank *versand* ein Funktionsaufruf durchgeführt.

Zur Verwendung des Formats müssen Sie eine Liste der Kundennummern (*kunden\_nr*) aller existierenden Kunden zur Verfügung haben. Nach Eingabe der Kundennummer in das Bildschirmfeld *f000* gibt PERFORM den Vor- und Zunamen sowie die Adresse des Kunden in den Feldern *f001* bis *f007* aus. Danach springt der Cursor zum Feld *f010* (*Auftragsdatum*), wo Sie mit der Eingabe der Auftragsbeschreibung beginnen können. Da für das Feld *Auftragsdatum* das aktuelle Datum *TODAY* vorgegeben ist, erscheint dieses Feld bereits gefüllt. Sie können jedoch ein anderes Datum eingeben.

Nach erfolgter Dateneingabe in die Felder *f011* und *f012* springt der Cursor zur Artikelliste und Sie geben die Lagernummer (*artikel\_nr*) und den Lieferantencode (*herstellercode*) des ersten Artikels ein. PERFORM gibt die Beschreibung des gewählten Lagerartikels und den Einzelpreis aus. Es ruft die Funktion *st\_desc* auf, um einen Suchvorgang in der Tabelle *artikel* durchzuführen, läßt den Cursor im Feld *q1* und wartet auf Ihre Mengeneingabe.

Nach erfolgter Mengeneingabe berechnet PERFORM den Gesamtpreis dieses Artikels und gibt die laufende Summe aller Artikel am unteren Rand des Bildschirms aus. Der Cursor springt danach zum ersten Feld und wartet auf die Eingabe der Lagernummer des zweiten Artikels.

Es wird so lange ein Artikel nach dem anderen bearbeitet, bis Sie die ESCAPE-Taste drücken und die Transaktion beenden. PERFORM trägt dann das Auftragsdatum in die Tabelle *auftrag* und jeden Artikel in die Tabelle *posten* ein. Es gibt auch die neue *Auftrnr* am Bildschirm aus.

Wenn Sie PERFORM verlassen, wird eine Funktion aufgerufen, die die Datenbank schließt.

```
database versand
```

```
screen
{
```

---

AUFTRAGSFOMULAR

---

```

Kundennummer:[f000      ] Ansprechpartner:[f001 ][f002      ]
      Firma:[f003      ]
Anschrift:[f004      ] [f005      ]
      Ort:[f006      ] Bundesland:[a0] PLZ:[f007 ]

```

---

```

Auftrnr:[f009      ] Auftrdatum:[f010      ] Kaufauftragsnr:[f011      ]
Lieferhinweis:[f012      ]

```

---

Artikelnr.	Code	Bezeichnung	Menge	Preis	Gesamt
[sn1 ]	[mc1]	[de1 ]	[q1 ]	[pr1 ]	[tp1 ]
[sn2 ]	[mc2]	[de2 ]	[q2 ]	[pr2 ]	[tp2 ]
[sn3 ]	[mc3]	[de3 ]	[q3 ]	[pr3 ]	[tp3 ]
[sn4 ]	[mc4]	[de4 ]	[q4 ]	[pr4 ]	[tp4 ]
[sn5 ]	[mc5]	[de5 ]	[q5 ]	[pr5 ]	[tp5 ]
Aufgelaufener Betrag inkl. MwSt und Zustellgeb.:					[rt ]

```

}
END
```

```
tables
```

```

kunde
auftrag
posten
artikel
```

```
attributes
```

```

f000 = displayonly allowing input type integer,
      lookup f001 = kunde.vorname,
             f002 = kunde.nachname,
             f003 = kunde.firma,
             f004 = kunde.adresse1,
             f005 = kunde.adresse2,
             f006 = kunde.ort,
             a0 = kunde.state,
             f007 = kunde.plz
             joining *kunde.kunden_nr;
f009 = displayonly type integer;
f010 = displayonly allowing input type date, default = today;
f011 = displayonly allowing input type char;
f012 = displayonly allowing input type char;
```

```
sn1 = displayonly allowing input type smallint;
mc1 = displayonly allowing input type char, upshift;
q1 = displayonly allowing input type smallint;
sn2 = displayonly allowing input type smallint;

mc2 = displayonly allowing input type char, upshift;
q2 = displayonly allowing input type smallint;
sn3 = displayonly allowing input type smallint;
mc3 = displayonly allowing input type char, upshift;
q3 = displayonly allowing input type smallint;
sn4 = displayonly allowing input type smallint;
mc4 = displayonly allowing input type char, upshift;
q4 = displayonly allowing input type smallint;
sn5 = displayonly allowing input type smallint;
mc5 = displayonly allowing input type char, upshift;
q5 = displayonly allowing input type smallint;

de1 = displayonly type char;
pr1 = displayonly type money, right;
tp1 = displayonly type money, right;
de2 = displayonly type char;
pr2 = displayonly type money, right;
tp2 = displayonly type money, right;
de3 = displayonly type char;
pr3 = displayonly type money, right;
tp3 = displayonly type money, right;
de4 = displayonly type char;
pr4 = displayonly type money, right;
tp4 = displayonly type money, right;
de5 = displayonly type char;
pr5 = displayonly type money, right;
tp5 = displayonly type money, right;
rt = displayonly type money, reverse, right;

instructions
after editadd of mc1
  if st_desc(1) then
    nextfield = q1
  else
    begin
      comments "Unzulaessige Artikelangaben"
      let sn1 = null
      let mc1 = null
      nextfield = sn1
    end
after editadd of mc2
  if st_desc(2) then
    nextfield = q2
  else
    begin
      comments "Unzulaessige Artikelangaben"
      let sn2 = null
      let mc2 = null
      nextfield = sn2
    end
```



```
after editadd of mc3
  if st_desc(3) then
    nextfield = q3
  else
    begin
      comments "Unzulaessige Artikelangaben"
      let sn3 = null
      let mc3 = null
      nextfield = sn3
    end

after editadd of mc4
  if st_desc(4) then
    nextfield = q4
  else
    begin
      comments "Unzulaessige Artikelangaben"
      let sn4 = null
      let mc4 = null
      nextfield = sn4
    end

after editadd of mc5
  if st_desc(5) then
    nextfield = q5
  else
    begin
      comments "Unzulaessige Artikelangaben"
      let sn5 = null
      let mc5 = null
      nextfield = sn5
    end

after editadd of q1
  let tp1 = q1 * pr1
  let rt = tp1 * 1.1
after editadd of q2
  let tp2 = q2 * pr2
  let rt = (tp1 + tp2) * 1.1
after editadd of q3
  let tp3 = q3 * pr3
  let rt = (tp1 + tp2 + tp3) * 1.1
after editadd of q4
  let tp4 = q4 * pr4
  let rt = (tp1 + tp2 + tp3 + tp4) * 1.1
after editadd of q5
  let tp5 = q5 * pr5
  let rt = (tp1 + tp2 + tp3 + tp4 + tp5) * 1.1
```

```

before editadd of displaytable
  let de1 = null
  let de2 = null
  let de3 = null
  let de4 = null
  let de5 = null
  let pr1 = null
  let pr2 = null
  let pr3 = null
  let pr4 = null
  let pr5 = null
  let tp1 = null
  let tp2 = null
  let tp3 = null
  let tp4 = null
  let tp5 = null
  let rt = null
after editadd of displaytable
  let f009 = add_order()
end

```

Beim Verlassen des Feldes *Code* eines jeden Artikels ruft PERFORM die Funktion *st\_desc* auf und beim Drücken der ESCAPE-Taste die Funktion *add\_order*. Im folgenden wird der Quellcode dieser Funktionen dargestellt. Neben den in diesem Kapitel definierten C-Funktionen verwendet das Programm einige Bibliotheksfunktionen von PERFORM sowie Funktionen, die in Kapitel 4 im Handbuch INFORMIX-ESQL/C[8] beschrieben sind.

Die Funktion *st\_desc* erhält die von Ihnen für *artikel\_nr* und *herstellercode* eingegebenen Werte und überprüft, ob einer davon NULL ist. Ist keiner der beiden Werte NULL, sucht sie *bezeichnung* und *preis* für den durch *artikel\_nr* und *herstellercode* definierten Lagerartikel, gibt sie auf dem Bildschirm aus und liefert als Ergebnis den Wert 1 (wahr). Ist entweder *artikel\_nr* oder *herstellercode* NULL oder gibt es für sie keinen entsprechenden Artikel in der Tabelle *artikel*, liefert *st\_desc* den Wert 0 (falsch).

Die Funktion *add\_order* prüft als erstes, ob Artikel eingegeben worden sind. Ist dies nicht der Fall, gibt sie eine Meldung aus, daß kein Auftrag eingetragen worden ist und endet. Ist zumindest ein Artikel eingegeben worden, sammelt *add\_order* die Daten über den Auftrag und startet eine Transaktion. Nach Aufnahme eines neuen Satzes in die Tabelle *auftrag* trägt *add\_order* in einer Schleife jeden Artikel des Auftrages in die Tabelle *posten* ein. Ein Satz wird in die Tabelle *posten* eingetragen, wenn *add\_order* feststellt, daß der Gesamtpreis für diesen Satz ungleich NULL ist. Nur wenn alle mit dem Auftrag verbundenen Datenbankänderungen erfolgreich durchgeführt sind, wird die Transaktion festgeschrieben und der Auftrag tatsächlich eingetragen. *add\_order* liefert als Ergebnis den Wert von *auftrags\_nr* an PERFORM zurück. PERFORM gibt den Wert am Bildschirm aus.

Das Format ist in *p\_ex2.per* der Beispieldatenbank zu finden. Die C-Funktionen sind hingegen in *mult\_item.ec* enthalten.

```
#include <ctools.h>
#include sqlca;

extern valueptr st_desc();
extern valueptr add_order();

struct ufunc userfuncs[] =
{
    "st_desc", st_desc,
    "add_order", add_order,
    0,0
};

char *sn[] = {
    "sn1",
    "sn2",
    "sn3",
    "sn4",
    "sn5"
};

char *mc[] = {
    "mc1",
    "mc2",
    "mc3",
    "mc4",
    "mc5"
};

char *de[] = {
    "de1",
    "de2",
    "de3",
    "de4",
    "de5"
};

char *q[] = {
    "q1",
    "q2",
    "q3",
    "q4",
    "q5"
};
```

```

char  *pr[] = {
        "pr1",
        "pr2",
        "pr3",
        "pr4",
        "pr5"
    };
char  *tp[] = {
        "tp1",
        "tp2",
        "tp3",
        "tp4",
        "tp5"
    };

valueptr st_desc(item)
valueptr item;
{
    $      int      s;
    $      char     m[4];
    $      char     d[16];
    $      dec_t    up;
    $      int      i;

    i = item->v_int - 1;
    pf_getval(sn[i], &s, CINTTYPE, 0);
    pf_getval(mc[i], m, CCHARTYPE, 4);
    if (risnull(CINTTYPE, &s) || risnull(CCHARTYPE, m))
        intreturn(0); /* Liefert 'falsch' zurueck, wenn ein
                       oder beide Felder NULL */

    $      select bezeichnung, preis into $d, $up from artikel
                where artikel_nr = $s and herstellercode = $m;
    if (sqlca.sqlcode == 0)
        {
            pf_putval(d, CCHARTYPE, de[i]);
            pf_putval(&up, CDECIMALTYPE, pr[i]);
            intreturn(1);
        }
    else
        intreturn(0);
}

valueptr add_order()
{
    $      long custno;
    $      long o_date;
    $      char ponum[11];
    $      char instr[41];
    $      long onum;
    $      int stno;
    $      char manc[4];
    $      int quan;
    $      dec_t total;
    $      int itno;
    $      int i;
    $      char errstr[80];

```

```

        /* Feststellen, ob Artikel im Formular eingegeben */
pf_getval("tp1", &total, CDECIMALTYPE, 0);
if (risnull(CDECIMALTYPE, &total))
{
    pf_msg("Keine Artikelangabe, kein Auftrag aufgenommen", 0, 1);
    pf_nxfield("sn1");
    lngreturn(0);
}
/* Daten holen fuer Satz fuer Tabelle orders */
pf_getval("f000", &custno, CLONGTYPE, 0);
pf_getval("f010", &o_date, CLONGTYPE, 0);
pf_getval("f011", ponum, CCHARTYPE,11);
pf_getval("f012", instr, CCHARTYPE,41);

/* Transaktion beginnen */
$ begin work;
/* Auftragsdaten in Tabelle auftrag aufnehmen */
$ insert into auftrag (auftrags_nr, kunden_nr, auftragsdatum,
    fremd_nr, lieferhinweis)
    values(0, $custno, $o_date, $ponum, $instr);
if (sqlca.sqlcode != 0)
{
    sprintf(errstr, "Fehlercode %d beim INSERT", sqlca.sqlcode);
    pf_msg(errstr,0,1);
$ rollback work;
    lngreturn(0);
}
onum = sqlca.sqlerrd[1]; /* SERIAL-Wert vergeben */

/* Daten holen fuer Satz fuer Tabelle posten */
for (i=0;i<5;i++)
{
    pf_getval(sn[i], &stno, CINTTYPE, 0);
    pf_getval(mc[i], manc, CCHARTYPE, 4);
    pf_getval(q[i], &quan, CINTTYPE, 0);
    pf_getval(tp[i], &total, CDECIMALTYPE, 0);
    if (! risnull(CDECIMALTYPE, &total))
    {
        itno = i + 1;
$ insert into posten values
        ($itno, $onum, $stno, $manc, $quan, $total);
        if (sqlca.sqlcode != 0)
        {
            sprintf(errstr, "Fehlercode %d beim UPDATE", sqlca.sqlcode);
            pf_msg(errstr, 0,1);
$ rollback work;
            lngreturn(0);
        }
    }
}
$ commit work;
lngreturn(onum);
}

```

## ctools.h

```
/*
 * This is the file which must be included in any C subroutine
 * source * file which is to be linked to libsace.a and libsperf.a. */
#include "value.h" #include "datetime.h" #include "sqltypes.h"

typedef struct value * valueptr; typedef struct value * acevalue; type-
def struct value * perfvalue;

extern struct value retstack;

#define intreturn(i)      {retstack.v_type=SQLSMINT;\ ret-
                          stack.v_int=(i);\ return(&retstack);}

#define lngreturn(i)     {retstack.v_type=SQLINT;\ ret-
                          stack.v_long=(i);\ return(&retstack);}

#ifndef NOFLOAT #define floreturn(d)
{retstack.v_type=SQLSMFLOAT;\
 retstack.v_float=(d);\
 return(&retstack);}

#define dubreturn(d)    {retstack.v_type=SQLFLOAT;\ ret-
                          stack.v_double=(d);\ return(&retstack);}
#endif /* NOFLOAT */

#define strreturn(s,c)  {retstack.v_type=SQLCHAR;\ ret-
                          stack.v_charp=(s);\ retstack.v_len=(c);\
                          return(&retstack);}

#define vcharreturn(s,c) {retstack.v_type=SQLVCHAR;\
                          retstack.v_charp=(s);\
                          retstack.v_len=(c);\
                          return(&retstack);}

#define decreturn(d)   {retstack.v_type=SQLDECIMAL;\ decco-
                          py(&d, &retstack.v_decimal);\ re-
                          turn(&retstack);}

#define dtimereturn(dt) {retstack.v_type=SQLDTIME;\
                          retstack.v_prec=(dt).dt_qual;\
                          deccopy(&(dt).dt_dec, &ret-
                          stack.v_decimal);\ return(&retstack);}

#define invreturn(inv)  {retstack.v_type=SQLINTERVAL;\ ret-
                          stack.v_prec=(inv).in_qual;\ decco-
                          py(&(inv).in_dec, &retstack.v_decimal);\
                          return(&retstack);}
```

```
extern int toint();      /* toint() takes a pointer to value struc-
                        *       ture
                        *       as an argument.
                        *       Returns the value (converted
                        *       to integer)
                        *       of the value structure (v_int).
                        */

extern long tolong();   /* tolong() takes a pointer to value struc-
                        *       ture
                        *       as an argument.
                        *       Returns the value (converted
                        *       to long)
                        *       of the value structure (v_long).
                        */

#ifdef NOFLOAT extern double todouble(); /* todouble() takes a
pointer to value structure
                        *       as an argument.
                        *       Returns the value (converted
                        *       to double)
                        *       of the value structure
                        *       (v_double).
                        */

#endif /* NOFLOAT */

extern long todate();   /* todate() takes a pointer to value struc-
                        *       ture
                        *       Returns the value (converted
                        *       to long)
                        *       of the value structure (v_long).
                        */

extern int todecimal(); /* todecimal() takes a pointer to a value
structure
                        *       as the first argument, and a
                        *       pointer to
                        *       a dec_t structure as a second
                        *       argument.
                        */

extern int todatetime(); /* todatetime(val, dttime, dtqual)
                        *       valueptr val;
                        *       dttime_t *dttime;
                        *       int dtqual;
                        *
                        *       input args: val, dtqual
                        *       output args: dttime */

extern int tointerval(); /* tointerval(val, intrvl, invqual)
                        *       valueptr val;
                        *       intrvl_t *intrvl;
                        *       int invqual;
                        *
                        *       input args: val, invqual
                        *       output args: intrvl */
```

```

#define intcon    toint
#define longcon   tolong

#ifdef NOFLOAT #define dubcon  todouble
#endif /* NOFLOAT */

struct ufunc
    {
        char *uf_id; struct value *(*uf_func)(); };

/*
 * The structure declaration for "userfuncs" must be put in the * user's data area. A
 * hypothetical case using the user C functions * called "userfunc1" and "userfunc2" is
 * shown below. * * valueptr
 * userfunc1(); These routines must be externed before
 * valueptr userfunc2(); the userfuncs structure is initialized
 *
 * struct ufunc userfuncs[] =
 *     {
 *         "userfunc1", userfunc1,
 *         "userfunc2", userfunc2,
 *
 *         ↑ Pointer to the user function.
 *
 *         ↑ The name of the user function
 *           as defined in the DEFINE state-
 *           ment of ACE.
 *
 *         0,0 ← Note that this array must be termina-
 *              ted by two zeros.
 *
 *     };
 *
 * These structures are required so that ACE can call the user sub-
 * routines * at run time. */

```



## Fehlerbehebungen

Im Handbuch INFORMIX-SQL-Nachschlagen V4.0 sind einige Fehler aufgetreten, die im folgenden berichtigt werden sollen. Anhand der Seitenzahl und der Kapitelnummer können Sie die zu verbessernden Stellen im Handbuch leicht finden.

### Seite 3-42

#### ATTRIBUTE FORMAT

formatiert Zahlen oder Datumsangaben. FORMAT darf nicht für die Datentypen DATETIME und INTERVAL verwendet werden.

#### "zahlenformat"

beschreibt die Lage des Kommas bei Bildschirm-Feldern, die Spalten des Datentyps DECIMAL, SMALLFLOAT oder FLOAT zugeordnet sind. Die einzelnen Stellen des Wertes werden durch das Zeichen # dargestellt. Ein Punkt (.) gibt die Lage des Kommas an. Wenn Sie keinen Punkt angeben für die Darstellung, wird der Wert als Ganzzahl interpretiert und die Nachkommastellen aufgerundet. Für die Darstellung von negativen Werten ist ein Minuszeichen (-) anzugeben. Ohne das Minuszeichen können negative Werte zwar eingegeben werden, die Anzeige im Format erfolgt jedoch ohne Minuszeichen. Das Minuszeichen geben Sie an der ersten Stelle der Formatdarstellung an.

#### Beispiel

```
FORMAT = "-###,###.##"
```

### Seite 3-56

Im INSTRUCTIONS-Abschnitt eines Formatprogramms zeigt die Anweisung COMPOSITES an, daß zwei oder mehrere Spalten zweier oder mehrerer Tabellen über einen COMPOSITES-Join miteinander verbunden sind.

Mit dem Voranstellen eines Sterns \* können Sie eine Tabelle zur dominanten Tabelle bestimmen. Das bedeutet, daß in die verbundenen Spalten pro Satz nur gleiche Werte eingetragen werden dürfen.

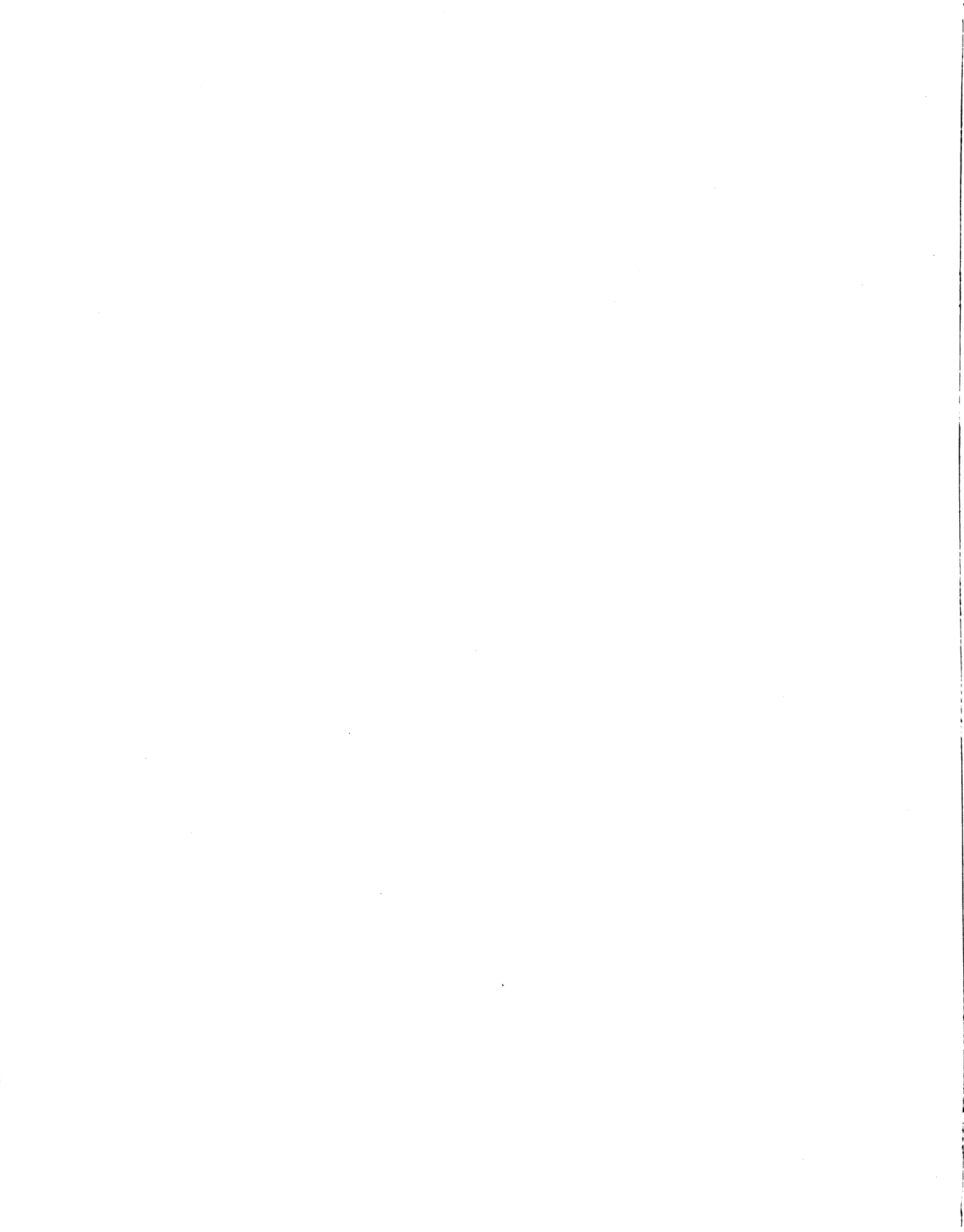


Dies gilt nicht für NULL-Werte.

NULL-Werte werden bei einer Überprüfung auf Gleichheit nicht als übereinstimmende Werte betrachtet und der Eintrag des Satzes in die Tabelle abgewiesen.

**Tip:** Tragen Sie statt eines NULL-Wertes (keine Eingabe) einen von Ihnen definierten Schlüsselwert (z.B. die Zahl 99) in die Join-Felder ein. Dieser Wert erfüllt ebenso die Gleichheit und kann später von Ihnen ebenso als "NULL-Wert" abgeprüft werden.

 Beachten Sie auch, daß Sie die Spalten, die Sie mit einem COMPOSITES-Join verbinden wollen, mit einem CLUSTER-Index indiziert haben.



---

# INFORMIX-ESQL/COBOL V4.1

## Änderungen im Überblick

Folgende Änderungen betreffen das ESQL/COBOL-Handbuch und werden im folgenden beschrieben. Die Änderungen im Überblick:

- Die Variablen SQLERRD[1] und SQLERRD[4] enthalten jetzt Werte, die zur Abfrage-Optimierung genutzt werden können.
- SQLCODE aus dem SQLCA-Satz wird auf 100 gesetzt, wenn Sie mit einer ANSI-Datenbank arbeiten und eine der folgende Anweisungen keinen Ergebnissatz hat:

INSERT INTO  
SELECT  
SELECT INTO TEMP  
DELETE  
UPDATE

- Sie können in dieser Version in reinen COBOL-Programmen, die zu ESQL/COBOL-Programmen dazugebunden werden, die Variable SQLCODE benutzen. Sie enthält die gleichen Werte, wie SQLCODE aus dem SQLCA-Satz. Auch in ESQL/COBOL-Programmen können Sie diese Variable benutzen.

Wenn Sie das Programm mit dem Schalter `-ansi` übersetzen oder die die Umgebungsvariable `DBANSIWARN` gesetzt haben, erhalten Sie Warnungen für jeden Bezeichner, der unter ANSI-Standard reserviert ist.

## Beschreibung der Änderungen

### Änderungen zum Bereich Erfolgskontrolle

#### Die Variablen SQLERRD[1] und SQLERRD[4] des SQLCA-Satzes

Der SQLCA-Satz ist im ESQL/COBOL-Handbuch unter 2.7.1 ausführlich beschrieben. Die Variablen SQLERRD[1] und SQLERRD[4] waren bisher nicht in Gebrauch. Sie enthalten jetzt folgende Werte:

SQLERRD[1] gibt die geschätzte Anzahl gefundener Sätze an.

SQLERRD[4] ist die gewichtete Summe von Plattenzugriffen und Anzahl bearbeiteter Sätze.

Wenn Sie die SET EXPLAIN-Anweisung benutzen, dann entspricht in sqlexplain.out SQLERRD[1] der Angabe 'geschätzte Anzahl Sätze' und SQLERRD[4] der Angabe 'geschätzte Kosten der SELECT-Anweisung aus der Anzahl Plattenzugriffe und der Anzahl der verarbeiteten Sätze'.

#### SQLNOTFOUND für eine ANSI-Datenbank

SQLCODE wird auf SQLNOTFOUND gesetzt, wenn Sie mit einer ANSI-Datenbank arbeiten und bei einer der folgenden Anweisungen auf keinen Satz zugegriffen werden konnte:

```
INSERT INTO tabellenname SELECT ...  
DELETE  
UPDATE  
SELECT INTO TEMP
```

Der Server gibt weiterhin SQLCODE = 0 zurück, wenn es sich um eine Nicht-ANSI-Datenbank handelt. So erfüllt ESQL/COBOL den ANSI-Standard und bleibt gleichzeitig kompatibel für Nicht-ANSI-Datenbanken.

### Beispiel

Im folgenden Beispiel werden in eine Tabelle nur Warenbestellungen aufgenommen, wenn sie mehr als 10000 mal bestellt wurden. Sollte es keine solche Bestellung geben, wird bei einer ANSI-Datenbank SQLNOTFOUND (100) zurückgegeben oder 0 bei einer Nicht-ANSI-Datenbank. \*\*\*\*\*Beispiel noch umschreiben \*\*\*\*\*

---

```
EXEC SQL INSERT INTO HOT-ITEMS
      SELECT DISTINCT STOCK-STOCK-NUM,
                     STOCK-MANU-CODE, DESCRIPTION
      FROM ITEMS, STOCK
      WHERE STOCK-STOCK-NUM = ITEMS-STCK-NUM
            AND STOCK-MANU-CODE = ITEMS-MANU-CODE
            AND QUANTITY > 10000;
END-EXEC.
```

---

### Benutzung der SQLCODE-Variable

Sie können ab dieser Version in reinen COBOL-Programmen, die zu ESQL/COBOL-Programmen dazugebunden werden, die Variable SQLCODE benutzen. Sie enthält die gleichen Werte, wie SQLCODE aus dem SQLCA-Satz. Auch in ESQL/COBOL-Programmen können Sie diese Variable benutzen.

## Fehlerbehebungen

Im Handbuch INFORMIX-ESQL/COBOL V4.0 sind einige Fehler aufgetreten, die im folgenden berichtigt werden sollen.

Über jedem neuen Absatz finden Sie jeweils die Seitenangabe und den zugehörigen Abschnitt, wo die fehlerhafte Stelle steht.

### Zu 2.3 Vereinbarung der Hostvariablen S.2-7

In der Tabelle über die entsprechenden Datentypen muß es in der ersten Zeile heißen:  
 CHAR (n)    PICTURE X(n)     $n \leq 32511$

### Zu 2.3.5 SMALLFLOAT und FLOAT S.2-15

Im letzten Satz des Abschnitts muß es heißen: Genaue COBOL-Entsprechungen gibt es nicht; nächstliegende COBOL-Datentypen sind PIC S9(m)V9(n) mit  $m+n \leq 7$  bzw.  $m+n \leq 14$  (siehe auch unter DECIMAL).

### Zu 2.3.6 DECIMAL S.2-16

In der Zuordnung COBOL-Datentyp zu SQL-Datentyp muß es heißen:

PIC S((m)V9(n)	—		SMALLFLAOT, wenn $m+n \leq 7$ DECIMAL (p) mit $p \leq m+n$ FLOAT, wenn $8 \leq m+n \leq 14$
----------------	---	---	---

### Zu 2.6.3 Beispiele S.2-35

In den Beispielen müssen die Unterstriche in den Namen durch Bindestriche ersetzt werden.

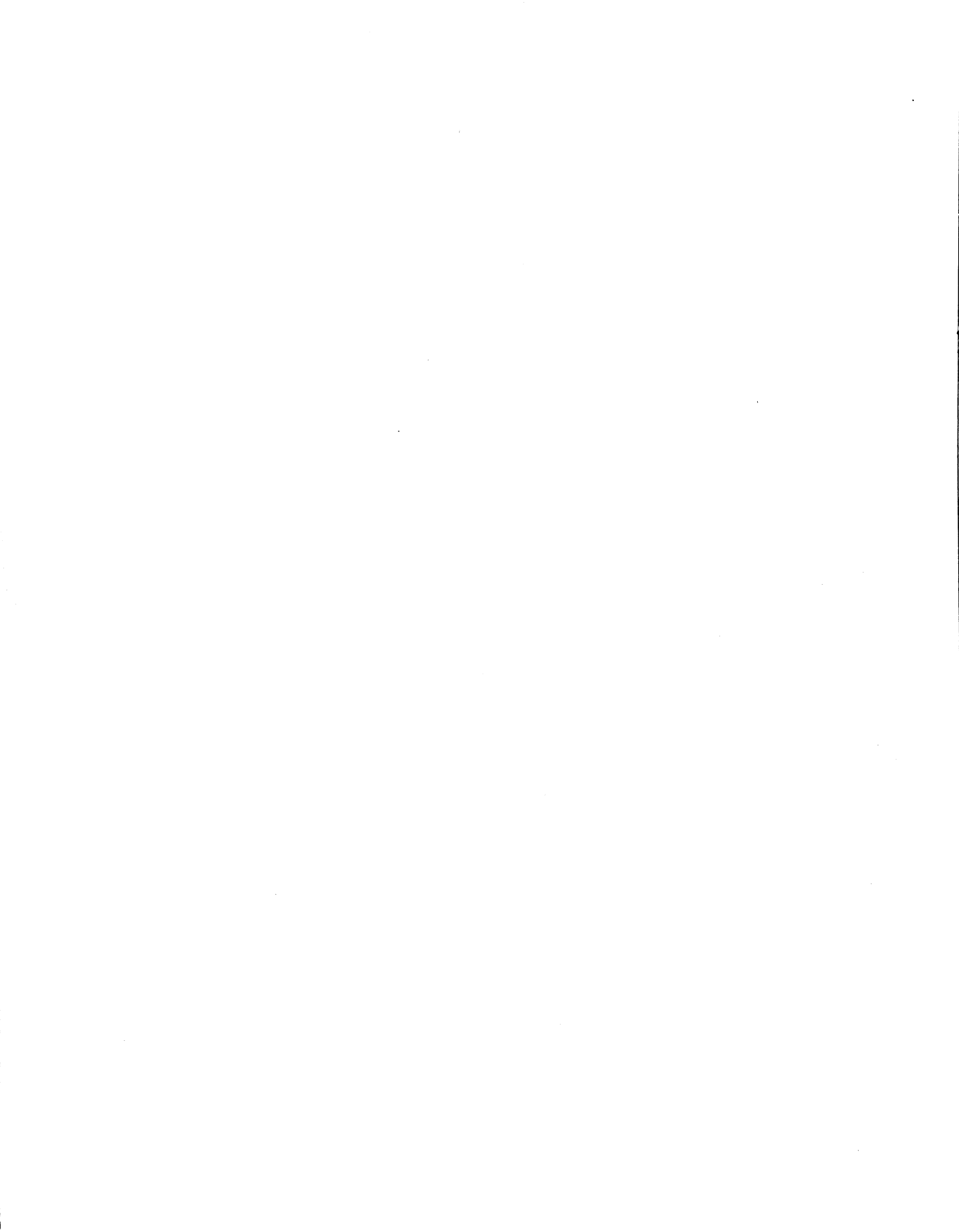
### Zu ECO-MSG Fehlermeldungsnummern in Text umsetzen, Kapitel 4

s-len muß als PIC S9(9) COMP-5 vereinbart werden.

**Zu 3.3 INFORMIX-ESQL/COBOL und LEVEL II COBOL**

Dieser Abschnitt entfällt. INFORMIX-ESQL/COBOL unterstützt nur den COBOL-Compiler COB85.





---

# INFORMIX-ESQL/C V4.1

## Änderungen im Überblick

Folgende Änderungen betreffen das ESQL/C-Handbuch und werden im nächsten Abschnitt beschrieben. Es folgt ein Überblick über diese Änderungen:

- Die Include-Datei `sqlca.h` wird jetzt automatisch zu einem ESQL/C-Programm dazugebunden.
- Die Variablen `sqlerrd[0]` und `sqlerrd[3]` enthalten jetzt Werte, die zur Abfrage-Optimierung genutzt werden können.
- Die Variable `sqlca.sqlcode` wird auf 100 gesetzt, wenn Sie mit einer ANSI-Datenbank arbeiten und eine der folgende Anweisungen keinen Ergebnissatz hat:

```
INSERT INTO...SELECT
SELECT INTO TEMP
DELETE
UPDATE
```

- Sie können jetzt in reinen C-Programmen, die zu ESQL/C-Programmen dazugebunden werden, die Variable `SQLCODE` benutzen. Sie enthält die gleichen Werte, wie `sqlca.sqlcode`. Auch in ESQL/C-Programmen können Sie diese Variable benutzen.
- Wenn Sie in ESQL/C-Programmen unter ANSI-Standard reservierte Bezeichner benutzen und das Programm mit dem Schalter `-ansi` übersetzen oder die Umgebungsvariable `DBANSIWARN` gesetzt haben, erhalten Sie Warnungen für jedes benutzte reservierte Wort.
- Es gibt eine neue Bibliotheksfunktion: `sqldetach`. Mit dieser Funktion kann ein Sohnprozeß von einem Vaterprozeß so abgetrennt werden, daß der Sohnprozeß seine eigene Datenbankverbindung haben kann.

## Beschreibung der Änderungen

### Änderungen zum Bereich Erfolgskontrolle

#### Die Variablen SQLERRD[0] und SQLERRD[3] der sqlca-Struktur

Die sqlca-Struktur ist im ESQL/C-Handbuch unter 2.7.1 ausführlich beschrieben. Die Variablen sqlerrd[0] und sqlerrd[3] waren bisher nicht in Gebrauch. Sie enthalten jetzt folgende Werte:

sqlerrd[0] gibt die geschätzte Anzahl gefundener Sätze an.

sqlerrd[3] ist die gewichtete Summe von Plattenzugriffen und Anzahl bearbeiteter Sätze.

Wenn Sie die SET EXPLAIN-Anweisung benutzen, dann entspricht in sqexplain.out sqlerrd[0] der Angabe 'geschätzte Anzahl Sätze' und sqlerrd[3] der Angabe 'geschätzte Kosten der SELECT-Anweisung aus der Anzahl Plattenzugriffe und der Anzahl der verarbeiteten Sätze'.

Folgender Ausschnitt aus einem ESQL/C-Beispielprogramm zeigt eine Anwendung für die neuen Werte. Das Programm wird nur fortgesetzt, wenn die Anzahl der geschätzten Ergebnissätze kleiner ist, als ein vorgegebenes Maximum:

---

```
$prepare prep1 from
    "select vorname, nachname into $vorname, $nachname
    from kunde where nachname > 'C'";
if (sqlca.sqlerrd[0] < MAXROWS)
{
    $declare democursor cursor for prep1; $open democursor;
    for (;;)
    {
        $fetch democursor; if (sqlca.sqlcode != 0) break; printf
        ("%s %s", vorname, nachname); }
    $close democursor; }
else
{
    printf("SELECT auf kunde name abgebrochen:");
    printf("zuviele Sätze (# der Sätze = %d)", sqlerrd[0]); }
```

---

### SQLNOTFOUND für eine ANSI-Datenbank

sqlcode = SQLNOTFOUND wird gesetzt, wenn Sie mit einer ANSI-Datenbank arbeiten und bei einer der folgenden Anweisungen auf keinen Satz zugegriffen werden konnte:

```
INSERT INTO tabellenname SELECT ...
DELETE
UPDATE
SELECT INTO TEMP
```

Bei einer Nicht-ANSI-Datenbank gibt der Server nach wie vor sqlcode = 0 zurück. So erfüllt ESQL/C den ANSI-Standard und bleibt gleichzeitig kompatibel für Nicht-ANSI-Datenbanken.

### Beispiel

Im folgenden Beispiel werden in eine Tabelle nur Warenbestellungen aufgenommen, wenn sie mehr als 10000 mal bestellt wurden. Sollte es keine solche Bestellung geben, wird bei einer ANSI-Datenbank SQLNOTFOUND (100) zurückgegeben oder 0 bei einer Nicht-ANSI-Datenbank.

---

```
$insert into hot_items
  select distinct stock.stock_num,
                stock.manu_code, description
  from items, stock
  where stock.stock_num = items.stock_num
        and stock.manu_code = items.manu_code
        and quantity > 10000;
```

---

## Einbindung der Include-Datei sqlca.h

Die Include-Datei sqlca.h ist automatisch in ein ESQL/C-Programm eingebunden. Sie benötigen für diese Datei keine Include-Anweisung mehr. Bestehende Programme, die diese Include-Anweisung enthalten, müssen nicht verändert werden.

Es folgt die Include-Datei sqlca.h, wie sie in der Version 4.1 gültig ist.

```

#ifndef SQLCA_INCL
#define SQLCA_INCL

struct sqlca_s
{
    long sqlcode;
    char sqlerrm[72]; /* error message parameters */
    char sqlerrp[8];
    long sqlerrd[6];
                    /* 0 - estimated number of rows returned */
                    /* 1 - serial value after insert or ISAM error code */
                    /* 2 - number of rows processed */
                    /* 3 - estimated cost */
                    /* 4 - offset of the error into the SQL statement */
                    /* 5 - rowid after insert */
    struct sqlcaw_s
    {
        char sqlwarn0; /* = W if any of sqlwarn[1-7] = W */
        char sqlwarn1; /* = W if any truncation occurred or
                        database has transactions */
        char sqlwarn2; /* = W if a null value returned or
                        ANSI database */
        char sqlwarn3; /* = W if no. in select list != no. in into list or
                        turbo backend */
        char sqlwarn4; /* = W if no where clause on prepared update, delete or
                        incompatible float format */
        char sqlwarn5; /* = W if non-ANSI statement */
        char sqlwarn6; /* reserved */
        char sqlwarn7; /* reserved */
    } sqlwarn;
};

/* NOTE: 4gl assumes that the sqlwarn structure can be defined as
 *      sqlwarn — an 8 character string, because single-char
 *      variables are not recognized in 4gl.
 *
 * If this structure should change, the code generated by 4gl compiler
 *      must also change
 */

```

```
#ifdef VMS
noshare
#endif /* VMS */
extern struct sqlca_s sqlca;

extern long SQLCODE;

#define SQLNOTFOUND 100

#endif /* SQLCA_INCL */
```

### Benutzung der SQLCODE-Variable

Sie können ab Version 4.1 in reinen C-Programmen, die an ESQL/C-Programme gebunden werden, die Variable SQLCODE zur Erfolgskontrolle benutzen. Sie enthält die gleichen Werte wie sqlca.sqlcode in ESQL/C-Programmen. SQLCODE muß als externe Variable vom Datentyp long vereinbart sein. Es gibt zwei Möglichkeiten diese Variable zu vereinbaren:

- Sie können SQLCODE als externe Variable in jedem Modul Ihres Programms wie folgt vereinbaren:  
`extern long SQLCODE;`
- Sie vereinbaren SQLCODE als globale Variable in einem Modul Ihres Programms und benutzen die externe Vereinbarung in den anderen Modulen:  
`/* Modul1 */ long SQLCODE;`  
`/* Modul2 */ extern long SQLCODE;`

Wollen Sie diese Variable als Hostvariable benutzen, müssen Sie der Vereinbarung ein \$-Zeichen voranstellen.

Auch in ESQL/C-Programmen kann diese Variable benutzt werden. Nach jeder SQL-Anweisung enthält SQLCODE den aktuellen Wert.

**Bibliotheksfunktionen (siehe ESQL/C-Handbuch, Kapitel 4)****sqldetach Vaterprozeß von Sohnprozeß trennen**

sqldetach trennt einen Sohnprozeß von einem Vaterprozeß, sodaß der Sohnprozeß seine eigene Datenbankverbindung haben kann.

---

```
sqldetach()
```

---

sqldetach sollte verwendet werden, wenn der Sohnprozeß den gleichen Code-Speicherplatz benutzt, wie der Vaterprozeß. Das ist der Fall, wenn kein exec()-Aufruf nach einem fork()-Aufruf erfolgt.

Diese Funktion kann nicht mit vfork() benutzt werden.

**Beispiel**

```
main()
{
    int pid;
    int status;

    $database newstore;

    pid = fork();
    if (pid == 0)
    {
        /* Sohnprozeß */
        sqldetach();
        $database sporty;
        ...
        $close database;
    }
    else
    {
        /* Vaterprozeß */
        ...
        while (wait (&status) != pid);
        ...
    }
}
```

## Fehlerbehebungen

Im Handbuch INFORMIX-ESQL/C V4.0 sind einige Fehler aufgetreten, die im folgenden berichtigt werden sollen.

Über jedem neuen Absatz finden Sie jeweils die Seitenangabe und den zugehörigen Abschnitt, wo die fehlerhafte Stelle steht.

### Indikatorvariable zur Erkennung von Informationsverlust

**Referenz im ESQL/C-Handbuch: Abschnitt 2.4.1 S. 2-36**

Der zweite Absatz unter dieser Überschrift muß heißen: ESQL/C weist der an die Hostvariable gebundenen Indikatorvariablen die gesamte Länge des Spalteninhalts zu. (Der nächste Teilsatz entfällt)

### Die sqlerrd [2]- Variable aus der sqlca-Struktur

**Referenz im ESQL/C-Handbuch: Abschnitt 2.7.1 S. 2-74**

sqlerrd[2] enthält nach INSERT-, UPDATE- und DELETE-Anweisungen die Anzahl der verarbeiteten Sätze.

### Funktionen decround und dectrunc

**Referenz im ESQL/C-Handbuch: Kapitel 4, S. 4-39, 4-49**

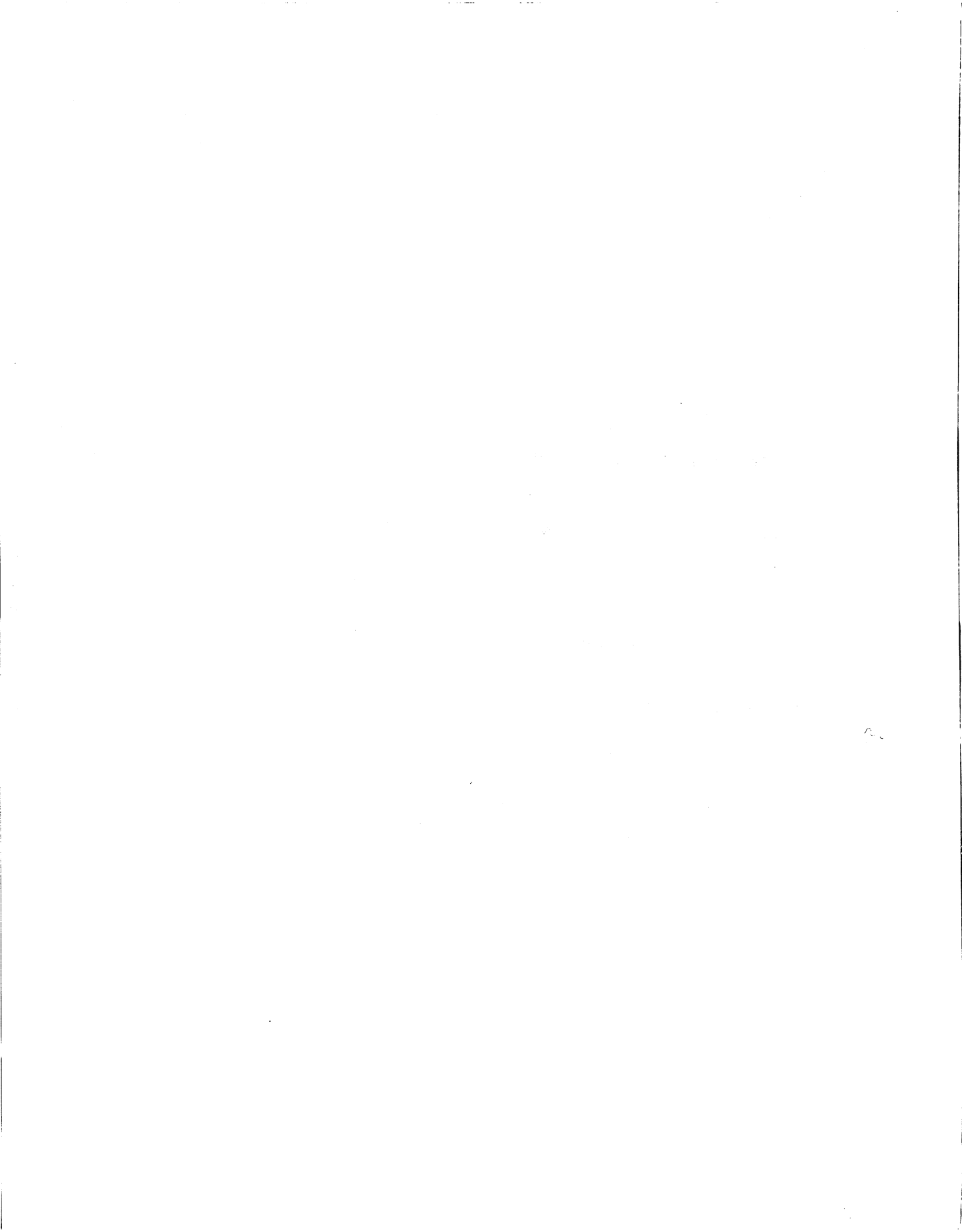
Der zweite Parameter für beide Funktionen *s* ist kein Zeiger auf ein INTEGER, sondern ein einfaches INTEGER. So daß die Syntax für die Funktionen folgendermaßen aussieht:

```
#include <decimal.h>
```

```
int decround(d,s)
    dec_t *d;
    int s;
```

```
int dectrunc(d,s)
    dec_t *d;
    int s;
```





## C-ISAM V4.1

Dieses Kapitel enthält die Neuerungen von C-ISAM V4.1 für die Version V4.0. Es beginnt mit einem Überblick der Änderungen, so daß Sie spezielle Punkte leicht finden können. Aus diesem Grund wurde die Kapiteleinteilung der Beschreibung der Version 4.0 übernommen.

Danach folgt die ausführliche Beschreibung der Änderungen. Die inhaltliche Einteilung der Beschreibung entspricht der Struktur des Handbuchs C-ISAM V4.0. Die Neuerungen sind in den direkten Zusammenhang gestellt.

Beachten Sie, daß die Bibliothek *libisam3.a* in C-ISAM V4.1 nicht mehr zur Verfügung steht.

### Änderungen im Überblick

In die Version 4.1 von *C-ISAM* wurden zwei Verbesserungen aufgenommen; es entspricht damit den Richtlinien aus dem *X/Open Portability Guide, Issue 4 (XPG4)*. Neu ist die Aufnahme von Datensätzen variabler Länge und einige Fehlercodes.

Zu Beginn wird beschrieben, wie Sie auf *C-ISAM* 4.1 umstellen und gleichzeitig die Datenintegrität sicherstellen.

#### **Kapitel 2 - Verwendung von C-ISAM**

Sie können ab Version 4.1 Dateien mit Datensätzen variabler Länge erzeugen und benutzen.

#### **Kapitel 9 - Aufrufformate und -Beschreibungen**

Folgende Funktionen wurden bei *C-ISAM* 4.1 geändert.

- ISADDINDEX
- ISBUILD
- ISINDEXINFO
- ISOPEN
- ISREAD
- ISREWCURR
- ISREWREC

- ISREWRITE
- ISWRCURR
- ISWRITE

Diese Funktionen sind vollständig beschrieben.

### Anhang

Folgende Änderungen ergeben sich für den Anhang von C-ISAM V4.0:

- "Die Datei *isam.h*" hat sich geändert.
- *isstat3* und *isstat4* sind neue Statusvariablen
- Die Formate für folgende Dateien haben sich geändert:
  - Indexdateien
  - Datendateien
  - AUDIT-Dateien
  - Transaktionsprotokoll-Dateien

### Umstellung auf C-ISAM Version 4.1

Informix hat das Format der Transaktionsprotokoll-Dateien geändert, damit auch Datensätze variabler Länge berücksichtigt werden können. Führen Sie vor der Installation von *C-ISAM* 4.1 folgende Schritte aus, um sicherzustellen, daß alle Transaktionen ggf. korrekt zurückgesetzt (ROLLBACK) bzw. nachgezogen (ROLLFORWARD) werden können:

- ▷ Sichern Sie die vorhandenen Datendateien, Transaktionsprotokoll-Dateien und AUDIT-Dateien, wenn keine Prozesse die Dateien momentan benutzen.
- ▷ Leeren oder löschen Sie die Transaktionsprotokoll-Dateien und AUDIT-Dateien, sobald die Sicherung erfolgt ist.
- ▷ Installieren Sie *C-ISAM* 4.1 unter Verwendung der Installationsanweisungen, die zusammen mit dem Produkt ausgeliefert werden.

Durch Ausführung dieser Schritte stellen Sie sicher, daß *C-ISAM* 4.1 eigene Transaktionsprotokoll-Dateien erzeugt und auf diese beim Zurücksetzen bzw. Nachfahren von Transaktionen zurückgreifen kann.

## Beschreibung der Änderungen

Die vorliegende Ergänzung beschreibt diese neuen *C-ISAM*-Möglichkeiten sowie deren Verwendung. Wenn Sie diese Ergänzung benutzen, sollten Sie mit den im *C-ISAM Programmierhandbuch* für Version 4.0 beschriebenen Informationen vertraut sein.

### Programmierung mit Datensätzen variabler Länge

Eine Datei kann entweder Datensätze mit variabler oder mit fester Länge enthalten. Bei Datensätzen mit variabler Länge kann ein Teil des Satzes von fester Länge sein. Der Teil mit variabler Länge befindet sich dann am Ende des Datensatzes im Anschluß an den Teil mit fester Länge. Um die Kompatibilität zu früheren *C-ISAM*-Versionen sicherzustellen, wird standardmäßig davon ausgegangen, daß ein Datensatz, der nicht ausdrücklich als Datensatz mit variabler Länge angegeben ist, feste Länge hat. Wie bei Datensätzen fester Länge müssen Sie eine C-Variable deklarieren, die die Daten aus dem Datensatz variabler Länge aufnimmt, während Sie diesen bearbeiten. Informationen über das Programmieren mit Datensätzen fester Länge finden Sie im *C-ISAM Programmierhandbuch*, Version 4.0.

Der Teil der Daten eines Datensatzes mit variabler Länge, deren Länge fest vorgegeben ist, wird in der Datendatei abgespeichert. Aus diesem Grund ist es wichtig, daß Sie die Indexdateien (*.idx*) *nicht löschen*. Wenn Sie die *.idx*-Dateien löschen, können die Dateien und die Daten mit variabler Länge darin nicht wiederhergestellt werden, es sei denn, Sie benutzen dazu eine Sicherungskopie.

Wenn der Indexteil der *.idx*-Dateien Fehler enthält, lassen Sie das Dienstprogramm *bcheck* ablaufen, ohne die *.idx*-Dateien zu löschen. Hierdurch bleiben die Daten variabler Länge erhalten. Sie finden ausführliche Informationen über die Wiederherstellung bei Datenverlust im Abschnitt *Dateiverwaltung bei Datensätzen variabler Länge* in der vorliegenden Ergänzung.

Die Möglichkeit, Datensätze variabler Länge zu benutzen, gibt es nur bei *C-ISAM*; sie ist bei keinem anderen Informix-Produkt, das *INFORMIX-SE* benutzt, verfügbar.

### Dateien mit Datensätzen variabler Länge erzeugen und verwenden

Sie erstellen und verwenden Dateien mit Datensätzen variabler Länge ähnlich wie Dateien mit Datensätzen fester Länge. Bei Datensätzen mit variabler Länge müssen Sie zusätzlich zu den üblichen Funktionsaufrufen die globale Variable *isreclen* verwenden.

### Datei erstellen

Sie erzeugen eine *C-ISAM*-Datei mit Datensätzen variabler Länge mit der Funktion *isbuild*.

1. Bevor Sie *isbuild* aufrufen, setzen Sie *isreclen* auf die kleinste Anzahl von Bytes, die in einem Datensatz variabler Länge vorkommen können. Hierdurch wird die Länge des Datensatzteils, der feste Länge hat, bestimmt. Die Gesamtlänge des Datensatzes kann zwischen 2 und 32 511 Byte liegen.
2. Sie rufen *isbuild* auf und geben ISVARLEN als Teil des Parameters *mode* an. Hierdurch bestimmen Sie, daß die Datei Datensätze variabler Länge enthält. Sie geben für den Parameter *satzlaenge* die maximale Datensatzlänge an, wobei sowohl der Anteil fester als auch der Anteil variabler Länge zusammengerechnet werden. Der kleinste Wert, den Sie in ISVARLEN benutzen können ist 1. Der kleinste Datensatz mit variabler Länge, den Sie benutzen können, enthält zwei Bytes; ein Byte für den Teil mit fester Länge und ein Byte für den Teil mit variabler Länge.

### Beispiel

Die beiden aufgeführten Anweisungen erstellen die *C-ISAM*-Datei *employee* mit Datensätzen, die maximal 1284 Byte lang sein können und mindestens 84 Byte lang sein müssen. Dabei beträgt der Anteil für die variable Länge bis zu 1200 Byte.

```
isreclen = 84;  
fd = isbuild("employee:", 1284, &key, ISINOUT + ISEXCLLOCK +  
            ISVARLEN);
```

Die Datei *employee* ist außerdem zur Ein- und Ausgabe geöffnet (ISINOUT) und ist exklusiv gesperrt (ISEXCLLOCK). Weitere Einzelheiten finden Sie in der ausführlichen Beschreibung von *isbuild* im Abschnitt *Geänderte Funktionen* in der vorliegenden Ergänzung.

### Datei öffnen

Wenn Sie eine Datei öffnen, die Datensätze variabler Länge benutzt, geben Sie im Parameter *mode* ISVARLEN an. Öffnen Sie eine Datei mit ISVARLEN, wird die globale Variable *isreclen* auf die maximale Länge des Datensatzes gesetzt. Wenn Sie bei Datensätzen variabler Länge ISVARLEN nicht angeben, versucht *C-ISAM*, die Datei wie eine Datei mit Datensätzen mit nur fester Länge zu öffnen. In der ausführlichen Beschreibung von *isopen* im Abschnitt *Geänderte Funktionen* der vorliegenden Ergänzung finden Sie weitere Informationen.

Wollen Sie eine Datei öffnen, ohne zu wissen, ob sie Datensätze variabler oder fester Länge enthält, so probieren Sie eine Möglichkeit. Ist dies nicht erfolgreich, öffnen Sie die Datei auf die andere Art.

### Beispiel

Im folgenden Programmausschnitt wird die Datei *employee* zuerst als Datei mit Datensätzen fester Länge geöffnet. Tritt bei diesem *isopen*-Aufruf ein Fehler auf, so wird *mode* so verändert, daß es auch ISVARLEN enthält. Dann wird *isopen* erneut aufgerufen.

```
varlen = FALSE;           /* Flag, das angibt, ob Datei VARLEN ist */
mode = ISINOUT + ISMANULOCK;
fd = isopen(employee, mode); /*Versuch, Datei als FIXLEN zu öffnen*/
if (fd < 0)
{
    mode += ISVARLEN;
    fd = isopen(employee, mode); /* Versuch, Datei als VARLEN zu öffnen */
    if (fd < 0)
    {
        printf ("Fehler bei isopen"); /* Öffnen hat nicht funktioniert */
        exit(-1);
    }
    varlen = TRUE;
    maxlen = isreclen;
}
```

### Datei schließen

Sie schließen Dateien mit Datensätzen variabler Länge genauso wie Dateien mit Datensätzen fester Länge. Dazu verwenden Sie *isclose*. Im *Programmierhandbuch* finden Sie eine Beschreibung von *isclose*.

### Index definieren

Sie können nur für den Teil eines Datensatzes mit fester Länge Indizes definieren. Wenn Sie bei Datensätzen mit variabler Länge für den Teil mit fester Länge Indizes definieren, gehen Sie genauso vor wie bei den standardmäßigen Datensätzen fester Länge. In Kapitel 3 des *Programmierhandbuchs* finden Sie eine ausführliche Beschreibung von Indizes für Dateien.

### Indexstrukturen bestimmen

Wie bei Dateien mit Datensätzen fester Länge benutzen Sie die Funktion *isindexinfo*, um herauszufinden, welche Indizes für eine Datei definiert sind, die Datensätze variabler Länge enthält. Ein Aufruf von *isindexinfo* setzt *isreclen* und liefert Informationen über die Indizes in einer Struktur vom Typ *dictinfo*, die aus vier Teilen besteht. Es handelt sich dabei um die folgenden vier Teile:

- di\_nkeys        Wenn die Datei Datensätze variabler Länge unterstützt, ist das höchstwertige Bit gesetzt. Die restlichen Bits zeigen an, wie viele Indizes für die Datei definiert sind, wie dies auch bei Datensätzen fester Länge der Fall ist.
- di\_reclsize    Dieses Feld enthält die maximale Datensatzlänge in Byte.
- di\_idxsize     Dieses Feld enthält die maximale Anzahl von Bytes in einem Indexknoten. (Knoten werden im Abschnitt "B+ Baum-Aufbau" im *Programmierhandbuch* beschrieben.)
- di\_nrecords    Diese Datei enthält die Anzahl von Datensätzen in der Datei.

Die Funktion *isindexinfo* setzt *isreclen* auf die Mindestgröße des Datensatzes in Byte.

### Datentypen bei Datensätzen variabler Länge

Da Sie für den variablen Teil eines Datensatzes keinen Index definieren können, müssen Sie den Datentyp oder die Länge von einzelnen Feldern in diesem Teil nicht angeben. Sie können mit Hilfe der Funktion *ld* bzw. *st* Daten von einem C-ISAM-Datensatz zu einer C-Variablen oder umgekehrt übertragen. Im *Programmierhandbuch* finden Sie weitere Informationen über die Routinen zur Umwandlung von Datentypen.

### Datensätze mit variabler Länge sperren

Sie sperren Dateien mit Datensätzen variabler Länge genau wie Dateien mit Datensätzen fester Länge. In Kapitel 5 des *Programmierhandbuchs* finden Sie weitere Informationen über Sperren.

### Transaktionen mit Datensätzen variabler Länge

Sie starten und beenden Transaktionen bei Datensätzen variabler Länge genauso wie bei Datensätzen fester Länge. Einige der Transaktionsprotokoll-Formate, die bei Datensätzen variabler Länge benutzt werden, unterscheiden sich gegenüber denjenigen, die bei Datensätzen fester Länge verwendet werden. Alle Transaktionsprotokoll-Formate, die in den Transaktionsprotokollen verwendet werden, sind im Abschnitt *Formate von Transaktionsprotokoll-Dateien* in dieser Ergänzung aufgeführt.

Wenn eine Transaktion mit einer Operation zum Aktualisieren die Länge eines Datensatzes variabler Länge verkürzt, so stellt ein Aufruf von *isrollback* die Daten des ursprünglichen Datensatzes wieder her, und zwar bringt es sie in den Zustand, den sie beim letzten Aufruf von *isbegin* hatten. Allerdings können die Daten an anderer Stelle gespeichert sein. Daher kann es sein, daß die Sicherungskopie einer Datei und die auf denselben logischen Zustand wiederhergestellte Datei nicht dasselbe binäre Speicherabbild besitzen, obwohl beide Dateien dieselben Benutzerdaten enthalten.

### AUDIT-Protokolle bei Datensätzen variabler Länge

Wie bei Dateien mit Datensätzen fester Länge können Sie auch bei Dateien mit Datensätzen variabler Länge AUDIT-Protokolle verwenden. Im Abschnitt "AUDIT-Protokoll" des *Programmierhandbuchs* finden Sie Informationen zur Implementierung von AUDIT-Protokollen.

Das Format der AUDIT-Datei für Datensätze variabler Länge enthält einen zusätzlichen Eintrag, der aus zwei Bytes besteht und die tatsächliche Länge des Datensatzes angibt. Im Abschnitt "Formate von AUDIT-Dateien" in der vorliegenden Ergänzung wird der Gesamtinhalt des Formats von AUDIT-Dateien dargestellt.

### Dateiverwaltung bei Datensätzen variabler Länge

Es ist wichtig, sowohl von Daten aus Datensätzen fester als auch variabler Länge stets aktuelle Sicherungskopien zu machen. Bei Dateien mit Datensätzen fester Länge können leicht Daten verloren gehen, wenn die *.dat*-Dateien aus irgendeinem Grund zerstört werden. Bei Dateien mit Datensätzen variabler Länge können leicht Daten verloren gehen, wenn entweder die *.dat* oder die *.idx*-Dateien aus irgendeinem Grund zerstört werden. Bei Datensätzen fester Länge können Sie eine Indexdatei neu erzeugen, wenn Sie die Schlüsselvereinbarungen und einige der Verzeichnisinformationen kennen. Bei Datensätzen variabler Länge können Sie den Indexteil der *.idx*-Dateien anhand derselben Informationen wieder erzeugen, aber Sie können nicht die Daten, die sich in den Indexdateien befanden wiederherstellen.

Wenn bei einer Datei mit Datensätzen variabler Länge Daten zerstört werden, können Sie mit Hilfe der Richtlinien aus den nächsten beiden Abschnitten eine saubere Datei erzeugen.



### Fehler in Datendateien

Sie benutzen eine Sicherungskopie der Daten- und Indexdateien und verwenden Sie dann die entsprechenden Transaktionsprotokolle oder AUDIT-Protokolle, um die *.dat*- und *.idx*-Dateien wiederherzustellen.

### Fehler in Indexdateien

Sie löschen die *.idx*-Dateien nicht. Mit Hilfe des Dienstprogramms *bcheck* beseitigen Sie Ungenauigkeiten in den Indexteilen der *.idx*-Dateien. Sind die Indexteile der *.idx*-Dateien fehlerhaft, können Sie die Indizes mit der Funktion *iscluster* wieder neu erstellen. Die Funktion *iscluster* öffnet die Datei, kopiert die Datensätze in eine neue Datei, und zwar in der durch die Parameter angegebenen Reihenfolge, erzeugt die Indizes neu, löscht die alte Datei und gibt der neuen Datei den Namen der alten Datei, aber eine neue Dateikennzahl. Die Funktion *iscluster* benutzt die Indizes in der alten Datei nicht, d.h. daß, sofern nur die Indexteile der *.idx*-Dateien fehlerhaft sind, die Funktion *iscluster* möglicherweise das Problem beseitigt.

Die Daten variabler Länge werden von *bcheck* oder *iscluster* nicht wiederhergestellt. Wenn Sie *iscluster* ablaufen lassen und es Fehler beim Lesen einiger Datensätze erzeugt, müssen Sie die Datenteile der Datei wiederherstellen. Hierzu können Sie ein Programm schreiben, das die Datensätze in der alten Datei in eine neue Datei einliest und alle fehlerhaften Datensätze mit einem Flag kennzeichnet. Dann können Sie mit einem anderen Programm diese Datensätze löschen und durch neue ersetzen. Wenn Sie die fehlerhaften Datensätze nicht ersetzen können, müssen Sie alle Dateien, d.h. sowohl die *.dat*- als auch die *.idx*-Dateien, aus einer Sicherungskopie wiederherstellen.

### Beispiel

Das folgende Programm liest die Datei *oldfile*, erzeugt eine neue Datei mit dem Namen *newfile* mit denselben Indizes wie die alte Datei, liest jeden einzelnen Datensatz aus der alten Datei und überträgt diese in die neue Datei. Kann ein Datensatz nicht gelesen werden, schreibt das Programm einen Platzhaltersatz in die neue Datei, damit die Reihenfolge der Datensätze gewährleistet ist. Sie können die Hilfsdatensätze mit einem anderen Programm löschen und entsprechende Aktionen ausführen.

```
#include <isam.h>
#include <stdio.h>

#define SUCCESS 0
#define SIZE 32511

char    dumrec[] = "Platzhaltersatz" ;
struct  keydesc key;
struct  dictinfo info ;
int     old_fd, new_fd;
```

```

/* Dieses Programm liest eine "alte" Datei mit Datensätzen variabler
 * Länge sequentiell durch und kopiert alle Datensätze in eine neue * Datei. Kann ein
 * Datensatz nicht gelesen werden, wird ein * Hilfsdatensatz eingefügt, der später analy-
 * siert werden kann. Die * beiden neuen und alten Dateinamen sind in diesem Pro-
 * gramm fest kodiert, * könnten aber genausogut zur Laufzeit angefordert werden. */

main()
{

int          minlen, maxlen, rr, ww;
char        record[SIZE];

printf("iserrno ist %d\n", iserrno);

/* alte Datei öffnen, um Dateikennzahl zu erhalten und
 * maximale Länge zu bestimmen */

old_fd = isopen ("isfile1", ISVARLEN + ISINPUT + ISEXCLLOCK);
printf("iserrno ist %d\n", iserrno);
maxlen = isreclen;
printf("alte Datei geöffnet mit fd = %d und maxlen = %d\n",
      old_fd, maxlen);

/* isindexinfo für den Primärschlüssel aufrufen, um
 * entsprechende Schlüsselvereinbarung und minimale (feste) * Länge zu erhalten */

isindexinfo(old_fd, &key, 1);
minlen = isreclen;
printf("Mit isindexinfo minlen = %d bestimmt\n ", minlen);

/* Aufbau der neuen Datei mit den Eigenschaften der alten Datei
 * Dabei sind Primärschlüssel, nicht aber die Sekundärschlüssel * eingeschlossen. */

new_fd = isbuild("newfile",maxlen,&key, ISVARLEN + ISINOUT +
                ISEXCLLOCK);
printf("Neue Datei erstellt mit new_fd = %d\n", new_fd);

/* Sekundärindizes zu neuer Datei hinzufügen */

addindex();

/* Zeiger vor den ersten Datensatz positionieren */

isstart(old_fd, &key, 0, record, ISFIRST);

/* Jeden einzelnen Datensatz der alten Datei lesen.
 * Falls beim Lesen Fehler auftreten, Hilfsdatensatz in die
 * neue Datei schreiben, damit die ursprüngliche Satznummerierung
 * beibehalten wird. Sind beim Lesen keine Fehler aufgetreten,
 * Datensatz in die neue Datei schreiben. Wenn eine Leseoperation
 * auf ein EOF oder einen anderen Fehler trifft bzw. wenn eine
 * Schreiboperation auf einen Fehler trifft, Abbruch.
 */

```

```

rr = SUCCESS;
ww = SUCCESS;

while (rr >= SUCCESS)
{
    rr = (isread(old_fd, record, ISNEXT + ISLOCK));
    printf("isread ausgeführt, rr = %d\n", rr );

    /*isreclen wurde von isread auf Bytezahl im Datensatz gesetzt */

    printf("iserrno = %d \n", iserrno);
    if (iserrno == EENDFILE) {printf("jetzt Abbruch\n"); break;}
    if (rr < SUCCESS ) ww = (iswrite (new_fd, dumrec));
    else      ww = iswrite(new_fd, record);
    if (ww < SUCCESS ) break;
}

if (iserrno == EENDFILE)
    printf ("isread ist am Ende der Datei angelangt.\n");
else if (ww < SUCCESS ) printf("Fehler bei iswrite\n");
iscleanup();
}

addindex()
{
    int cc, numkeys;

    cc = isindexinfo (old_fd, &info, 0);
    if (cc != SUCCESS) {printf ("Fehler %d bei isindexinfo", iserrno);
    exit(1);}
    numkeys = info.di_nkeys & 0x7fff;
    while(numkeys > 0)
    {
        isindexinfo(old_fd, &key, numkeys--);
        printf("isindexinfo wird ausgeführt mit numkeys = %d\n");
        isaddindex(new_fd,&key);
    }

    return;
}

```

Eine Beschreibung des Formats der Indexdateien finden Sie im Abschnitt *Formate von Indexdateien* am Ende dieses Kapitels.

## Geänderte Funktionen

Die folgenden Funktionen wurden in *C-ISAM* Version 4.1 geändert:

- **isaddindex**
- **isbuild**
- **isindexinfo**
- **isopen**
- **isread**
- **isrewcurr**
- **isrewrec**
- **isrewrite**
- **iswrcurr**
- **iswrite**

Im vorliegenden Abschnitt finden Sie eine ausführliche Beschreibung jeder einzelnen dieser Funktionen. Bitte benutzen Sie für Ihre Arbeit mit *C-ISAM* diese Beschreibungen und nicht diejenigen aus Version 4.0 des *C-ISAM Programmierhandbuches*.

### ISADDINDEX

#### Überblick

Mit *isaddindex* fügen Sie einen Index zu einer *C-ISAM*-Datei hinzu.

#### Syntax

```
isaddindex(isfd, keydesc)
    int isfd;
    struct keydesc *keydesc;
```

*isfd*

ist die Dateikennzahl, die von *isopen* oder *isbuild* zurückgegeben wird.

*keydesc*

ist ein Zeiger auf eine Schlüsselvereinbarungsstruktur.

#### Hinweise

1. Die *C-ISAM*-Datei muß für exklusiven Zugriff geöffnet sein.
2. Die Anzahl der Indizes, die Sie hinzufügen können, ist unbeschränkt.
3. Sie können nur für den Satzteil, der feste Länge hat, Indizes definieren. Wenn die durch *keydesc* angezeigte Zeichenposition die für die Datei definierte Mindestlänge eines Datensatzes überschreitet, tritt bei *isaddindex* ein Fehler auf. (Weitere Informationen finden Sie bei *isbuild* im Abschnitt *Geänderte Funktionen*.)

4. Die maximale Anzahl von Teilen, die Sie für einen Index definieren können, ist NPARTS.
5. Die Datei *isam.h* enthält die Definition von NPARTS. (Normalerweise ist NPARTS gleich 8.)
6. Die maximale Schlüssellänge ist MAXKEYSIZE. Die Datei *isam.h* enthält die Definition von MAXKEYSIZE. (Normalerweise beträgt der Wert 120 byte.)
7. Der Aufruf *isaddindex* kann innerhalb einer Transaktion nicht zurückgesetzt werden. Allerdings kann er wiederhergestellt werden.

### Rückgabewerte

- |    |   |
|----|---|
| -1 | Fehler; <i>iserrno</i> enthält den Fehlercode |
| 0  | erfolgreich                                   |

### Beispiel

```
#include <isam.h>
struct keydesc nkey;
.
.
.
nkey.k_flags = ISDUPS;
nkey.k_nparts = 2;
nkey.k_part[0].kp_start = 4;
nkey.k_part[0].kp_leng = 10;
nkey.k_part[0].kp_type = CHARTYPE;
nkey.k_part[1].kp_start = 24;
nkey.k_part[1].kp_leng = 1;
nkey.k_part[1].kp_type = CHARTYPE;
.
.
.
if ((fd=isopen("employee", ISEXCLLOCK+ISINOUT)) >= 0)
{
    if (isaddindex(fd, &nkey) < 0)
    {
        printf ("Fehler %d bei isaddindex", iserrno);
        exit (1);
    }
    .
    .
    .
}
```

## ISBUILD

### Überblick

Mit *isbuild* erzeugen Sie eine *C-ISAM*-Datei.

### Syntax

```
isbuild(dateiname, satzlaenge, keydesc, mode)
    char *dateiname;
    int satzlaenge;
    struct keydesc *keydesc;
    int mode;
```

#### dateiname

ist der Name der Datei ohne Dateinamenerweiterung.

#### satzlaenge

ist die Länge des Datensatzes in Byte. Soll der Datensatz einen Teil variabler Länge besitzen, so ist *satzlaenge* die maximale Länge des Datensatzes. *satzlaenge* ist eine Zahl zwischen 1 und 32 511 einschließlich.

#### keydesc

ist ein Zeiger auf eine Schlüsselvereinbarungsstruktur, die den Primärschlüssel definiert.

#### mode

ist eine Kombination eines Parameters für den Zugriffsmodus, eines Parameters für den Sperrmodus und wahlweise eines Parameters für die Länge oder für Protokolle. Sie geben *mode* an, indem Sie einen Parameter für den Zugriffsmodus zu einem Parameter für den Sperrmodus addieren. Es gibt die folgenden Parameter für den Zugriffsmodus:

ISINPUT        öffnet die Datei zur Eingabe.

ISOUTPUT      öffnet die Datei zur Ausgabe.

ISINOUT        öffnet die Datei zur Ein- und Ausgabe.

Es gibt die folgenden Parameter für den Sperrmodus:

ISEXCLLOCK    gibt eine exklusive Dateisperre an.

ISMANULOCK    gibt eine manuelle Datei- oder Satzsperrung an bzw. legt fest, daß keine Sperre vorhanden ist.

ISAUTOLOCK    gibt automatische Satzsperrungen an.

Außerdem können Sie die folgenden Parameter verwenden:

ISVARLEN	gibt an, daß der Datensatz einen Teil variabler Länge besitzt.
ISFIXLEN	gibt an, daß der Datensatz keinen Teil variabler Länge besitzt.
ISTRANS	ermöglicht es der Funktion <i>isrollback</i> , Änderungen an <i>C-ISAM</i> -Dateien innerhalb einer Transaktion rückgängig zu machen.
ISNOLOG	legt fest, daß dieser Aufruf und alle folgenden Aufrufe für diese Datei nicht mitprotokolliert werden.

### Hinweise

1. Wenn Sie ISFIXLEN oder ISVARLEN nicht benutzen, wird standardmäßig davon ausgegangen, daß der Datensatz feste Länge hat.
2. Wenn Sie ISVARLEN benutzen, müssen Sie der Variablen *isreclen* die Mindestzahl der Bytes im Datensatz zuweisen. Besitzt der Datensatz einen Teil fester Länge, enthält *isreclen* die Länge dieses Teils. Der Teil mit variabler Länge befindet sich am Ende des Datensatzes.
3. Die Funktion *isbuild* erzeugt zwei Betriebssystemdateien mit den Namen *dateiname.dat* und *dateiname.idx*. (Wenn Ihre *C-ISAM*-Version den Betriebssystemaufruf *fcntl()* nicht benutzt, wird außerdem eine dritte Datei mit dem Namen *dateiname.lok* erzeugt.) Diese Dateien werden zusammen als eine logische *C-ISAM*-Datei behandelt.
4. Der Parameter *dateiname* sollte eine mit Null abgeschlossene Zeichenkette sein, die mindestens vier Zeichen kürzer als der längste gültige Dateiname im Betriebssystem ist.
5. Die Funktion liefert eine ganzzahlige Dateikennzahl, die die Datei spezifiziert.
6. Die Datei bleibt geöffnet mit den Zugriffs- und Sperrmodi, die im Parameter *mode* festgelegt wurden.
7. Der Parameter *keydesc* gibt die Struktur des Primärindex an. Sie können *k\_nparts* = 0 setzen. Dies bedeutet, daß kein Primärschlüssel existiert und die Daten sequentiell in der Reihenfolge der Datensatznummern (d.h. in physikalischer Reihenfolge) verarbeitet werden.
8. Sie können später Indizes hinzufügen, indem Sie die Funktion *isaddindex* verwenden.

9. Wenn Sie vor dem Erstellen der neuen Datei ein Transaktionsprotokoll geöffnet haben und Sie die neue Datei im Fall eines Systemabsturzes wiederherstellen wollen, müssen Sie vor dem *isbuild*-Aufruf die Funktion *isbegin* aufrufen.
10. Die Funktion *isbuild* kann nicht zurückgesetzt werden.
11. Wenn Sie vor dem Erstellen der neuen Datei ein Transaktionsprotokoll geöffnet haben und Sie die neue Datei im Fall eines Systemabsturzes nicht wiederherstellen wollen, verwenden Sie den Parameter ISNOLOG. Dadurch verhindern Sie das Protokollieren eines *isbuild*-Aufrufs sowie weiterer C-ISAM-Aufrufe der Datei. In diesem Fall sollten Sie sicherstellen, daß alle weiteren *isopen*-Aufrufe für diese Datei ebenfalls ISNOLOG verwenden.

### Rückgabewerte

- 1 Fehler; *iserrno* enthält den Fehlercode
- >=0 Dateikennzahl

### Beispiel

```
#include <isam.h>
struct keydesc key;
.
.
.
key.k_flags = ISNODUPS;
key.k_nparts = 1;
key.k_part[0].kp_start = 0;
key.k_part[0].kp_leng = LONGSIZE;
key.k_part[0].kp_type = LONGTYPE;
.
.
.
if((fd=isbuild("employee",84,&key,ISINOUT+ISEXCLOCK))<0)
{
printf ("Fehler %d bei isbuild",iserrno);
exit (1);
}
/* entsprechender Aufruf für einen Datensatz variabler Länge */
/* isreclen zuerst auf feste Länge setzen */
isreclen = 84
if((fd=isbuild("v_employee", 1084, &key,
ISINOUT+ISEXCLOCK+ISVARLEN)) <0
{
printf ("Fehler %d bei isbuild",iserrno);
exit (1);
}
```



## ISINDEXINFO

### Überblick

Mit *isindexinfo* können Sie Informationen über die Struktur und Indizes einer *C-ISAM*-Datei abrufen.

### Syntax

```
isindexinfo(isfd, bereich, nummer)
    int isfd;
    struct keydesc *bereich;
    int nummer;
```

#### isfd

ist die Dateikennzahl, die von *isopen* oder *isbuild* zurückgegeben wird.

#### bereich

ist ein Zeiger auf eine Struktur. *bereich* kann aber auch ein Zeiger auf eine *dictinfo*-Struktur sein.

#### nummer

ist entweder eine Indexnummer oder Null.

### Hinweise

1. Wenn Sie Informationen über einen speziellen Index abrufen wollen, müssen Sie im Argument *nummer* die entsprechende Indexnummer angeben. Sie benutzen einen Zeiger auf eine *keydesc*-Struktur, in der die Informationen abgelegt werden.
2. Sie erhalten allgemeine Informationen, einschließlich der Anzahl der Indizes, der Indexknotengröße und Datensatzlänge, wenn Sie *isindexinfo* aufrufen, wobei *nummer* auf Null gesetzt ist und der *bereich* ein Zeiger auf eine Struktur vom Typ *dictinfo* ist.
3. Indizes sind numeriert, wobei bei 1 begonnen wird. Der Primärindex ist immer Index 1.
4. Da Indizes gewöhnlich hinzugefügt und auch gelöscht werden, kann sich die Nummer eines bestimmten Index ändern. Sie erreichen, daß alle Indizes mit einbezogen werden, indem Sie eine Schleife über die in *dictinfo* angezeigte Anzahl der Indizes bilden.

5. Wenn die Datei Datensätze mit variabler Länge enthält, speichert *isindexinfo* die Mindestsatzlänge (d.h. die Länge des Datensatzteils mit fester Länge) in der globalen Variablen *isreclen*.

Wenn die Datei Datensätze variabler Länge enthält, enthalten die Variablen *di\_nkeys* und *di\_recsize*, auf die *bereich* zeigt, außerdem Informationen, die sich speziell auf die Datensätze mit variabler Länge beziehen, und zwar wie folgt:

*di\_nkeys*            Wenn die Datei Datensätze variabler Länge unterstützt, ist das höchstwertige Bit gesetzt. Die restlichen Bits geben an, wie viele Indizes für die Datei definiert sind, wie dies auch bei Datensätzen fester Länge der Fall ist.

*di\_recsize*        Dieses Feld enthält die maximale Datensatzlänge in Bytes.

Im *Programmierhandbuch* finden Sie weitere Informationen über die Struktur *dictinfo*.

### Rückgabewerte

- 1        Fehler; *iserrno* enthält den Fehlercode
- 0        erfolgreich

### Beispiele

- Wenn Sie allgemeine Informationen über die *C-ISAM*-Datei abrufen wollen, rufen Sie *isindexinfo* folgendermaßen auf:

```
#include <isam.h>
struct dictinfo info;
:
:
:
fd = isopen ("employee", ISINPUT+ISEXCLLOCK);
isindexinfo (fd,&info,0);
printf ("\nDatensatzlänge in Bytes=%d",info.di_recsize);
printf ("\nAnzahl der Datensätze in der Datei=%d",
        info.di_nrecords);

isclose (fd);
exit (0);
```

- Wenn Sie Informationen über jeden einzelnen Index erhalten wollen, rufen Sie *isindexinfo* folgendermaßen auf:

```
#include <isam.h>
struct dictinfo info;
struct keydesc kdesc;
.
.
/* Anzahl der Schlüssel holen */
isindexinfo (fd,&info,0);
/* höchstwertiges Bit ausblenden, damit Anzahl der
 * für die Datei definierten Indizes übrigbleibt */

numkeys = info.di_nkeys & 0x7fff;
while (numkeys > 0)
{
/* Struktur holen und Indexnummer um eins verringern */
isindexinfo (fd,&kdesc,numkeys--);
.
.
}
}
```

**ISOPEN****Überblick**

Mit *isopen* öffnen Sie eine *C-ISAM*-Datei, damit sie verarbeitet werden kann.

**Syntax**

```
isopen(dateiname, mode)
    char *dateiname;
    int mode;
```

**dateiname**

ist der Name der Datei.

**mode**

ist eine Kombination eines Parameters für den Zugriffsmodus, eines Parameters für den Sperrmodus und wahlweise eines transaktionsbezogenen Parameters. Sie geben *mode* an, indem Sie einen Parameter für den Zugriffsmodus zu einem Parameter für den Sperrmodus addieren. Es gibt die folgenden Parameter für den Zugriffsmodus:

**ISINPUT** öffnet die Datei zur Eingabe (nur zum Lesen).

**ISOUTPUT** öffnet die Datei zur Ausgabe (nur zum Schreiben).

**ISINOUT** öffnet die Datei zur Ein- und Ausgabe.

Es gibt die folgenden Parameter für den Sperrmodus:

**ISEXCLLOCK** gibt eine exklusive Dateisperre an.

**ISMANULOCK** gibt eine manuelle Datei- oder Satzsperrung an bzw. legt fest, daß keine Sperre vorhanden ist.

**ISAUTOLOCK** gibt automatische Satzsperrungen an.

Außerdem können Sie die folgenden Parameter verwenden:

**ISVARLEN** gibt an, daß jeder Datensatz einen Teil von variabler Länge enthält. Wenn Sie die Datei mit *ISVARLEN* erstellt haben, so müssen Sie diese auch mit *ISVARLEN* öffnen.

**ISFIXLEN** gibt an, daß der Datensatz keinen Teil variabler Länge enthält.

**ISTRANS** ermöglicht es der Funktion *isrollback*, Änderungen an *C-ISAM*-Dateien innerhalb einer Transaktion rückgängig zu machen.

**ISNOLOG** legt fest, daß dieser Aufruf und alle folgenden Aufrufe für diese Datei nicht mitprotokolliert werden.



Wenn zu einem bestimmten Zeitpunkt Änderungen an einer *C-ISAM*-Datei vorgenommen werden, die aber nicht in der Protokolldatei aufgezeichnet werden, ist die Wiederherstellung der Daten unmöglich. Für eine *C-ISAM*-Datei müssen immer entweder alle Transaktionen oder keine Transaktion mitprotokolliert werden. Sollen Änderungen mitprotokolliert werden, rufen Sie *isbegin* auf, bevor Sie *isopen* aufrufen.

### Hinweise

1. Die Funktion liefert die Dateikennzahl, die Sie bei späteren Bearbeitungen der *C-ISAM*-Datei benutzen müssen.
2. Wenn Sie die Datei öffnen, erhalten Sie Zugriff über den Primärindex. Sollten Sie eine andere Reihenfolge benötigen, wählen Sie mit *isstart* einen anderen Index oder die Reihenfolge der Satznummern aus.
3. Der Parameter *dateiname* muß eine mit Null abgeschlossene Zeichenkette ohne Erweiterung enthalten. Dies ist der Dateiname der *C-ISAM*-Datei, die bearbeitet werden soll.
4. Wenn Sie den Parameter *ISVARLEN* bei dem Funktionsaufruf verwenden, wird die globale ganzzahlige Variable *isreclen* auf die maximale Satzlänge für die Datei gesetzt.
5. Wenn Sie *ISVARLEN* und *ISFIXLEN* nicht angeben, wird standardmäßig *ISFIXLEN* vorausgesetzt. Sofern Sie versuchen, eine Datei mit Datensätzen variabler Länge ohne *ISVARLEN* zu öffnen, tritt ein Fehler auf.

### Rückgabewerte

- 1 Fehler; *iserrno* enthält den Fehlercode
- >=0 Dateikennzahl

### Beispiel

```
fd_per = isopen("perform", ISINOUT+ISMANULOCK+ISTRANS);  
fd_per = isopen("employee", ISINOUT+ISEXCLLOCK)  
fd_per = isopen("v_employee", ISVARLEN+ISINOUT+ISEXCLLOCK)
```

**ISREAD****Überblick**

Mit *isread* lesen Sie Datensätze sequentiell oder an bestimmten Positionen, je nach Inhalt des Parameters *mode*.

**Syntax**

```
isread(isfd, satz, mode)
    int isfd;
    char *satz;
    int mode;
```

*isfd*

ist die Dateikennzahl, die von *isopen* oder *isbuild* zurückgegeben wird.

*satz*

ist ein Zeiger auf eine Zeichenkette, die den Suchwert enthält und in der der Datensatz abgelegt wird.

*mode*

ist einer der folgenden Parameter:

ISCURR	liest den aktuellen Datensatz.
ISFIRST	liest den ersten Datensatz.
ISLAST	liest den letzten Datensatz.
ISNEXT	liest den nächsten Datensatz.
ISPREV	liest den vorhergehenden Datensatz
ISEQUAL	liest den Datensatz, bei dem der entsprechende Wert mit dem Suchwert übereinstimmt.
ISGREAT	liest den ersten Datensatz, bei dem der entsprechende Wert größer als der Suchwert ist.
ISGTEQ	liest den ersten Datensatz, bei dem der entsprechende Wert größer oder gleich dem Suchwert ist.

Wahlweise können Sie einen oder mehrere der folgenden Sperrparameter an den Suchmodus anhängen:

ISLOCK	sperrt den Datensatz.
ISSKIPLOCK	setzt den Satzzeiger und <i>isrecnum</i> auf den gesperrten Datensatz; wenn <i>isread</i> auf einen gesperrten Datensatz trifft, können Sie nochmals <i>isread</i> aufrufen, und zwar mit dem Parameter ISNEXT, und so den gesperrten Datensatz überspringen.
ISWAIT	veranlaßt den Prozeß zu warten, bis eine Sperre auf einem Datensatz wieder freigegeben wird.
ISLCKW	entspricht ISLOCK+ISWAIT.

### Hinweise

1. Schreiben Sie den Suchwert im *satz* an die entsprechende Stelle für den Schlüssel.
2. Wenn die Suche erfolgreich ist, fügt *isread* die Daten aus dem gefundenen Datensatz in die entsprechenden Positionen von *satz* ein.
3. Der Datensatz wird zum aktuellen Datensatz für die Datei.
4. Die Funktion *isread* setzt die globale Variable *isrecnum* auf die Satznummer des Datensatzes, den sie liest. Enthält die Datei Datensätze mit variabler Länge, setzt *isread* die globale Variable *isreclen* auf die Anzahl der Bytes, die im Datensatzpuffer zurückgeliefert werden. (Der Inhalt des Puffers nach dem Wert von *isreclen* ist undefiniert.)
5. Sie können mit *isread* spezielle Datensätze unter Verwendung der Satznummer lesen. Sie rufen *isstart* mit einer *keydesc*-Struktur auf, die *k\_nparts*=0 enthält, so daß das Abrufen nach der physikalischen Reihenfolge geschieht. Darauf folgende Aufrufe von *isread* mit ISEQUAL als *mode* führen dazu, daß die Funktion in *isrecnum* nachschaut und die Satznummer liest.
6. Fügen Sie ISLOCK zu einem der Abrufparameter dazu, wenn Sie einen Datensatz sperren wollen. Der Sperrmodus ISMANULOCK muß beim Öffnen der Datei angegeben werden. Der Datensatz bleibt so lange gesperrt, bis Sie die Sperre mit *isrelease*, *iscommit* oder *isrollback* freigeben.
7. Wenn Sie nur einen Teil eines zusammengesetzten Index verwenden, sollten Sie den Parameter ISEQUAL nicht verwenden. Die Funktion *isread* findet nämlich keine genau übereinstimmenden Werte für einen partiellen Suchwert, wenn der Parameter ISEQUAL verwendet wird. Sie können *isstart* mit ISEQUAL aufrufen und dann *isread* mit ISCURR verwenden, um die erste Stelle zu finden, an der der Datensatz auftritt.

8. Wenn Sie *isread* mit ISCURRE oder ISNEXT verwenden, nachdem Sie einen Datensatz mit *iswrite* hinzugefügt haben, so liefert *isread* den Datensatz, den Sie soeben hinzugefügt haben.
9. Wenn Sie *isread* mit ISCURRE oder ISNEXT nach einem *isstart*-Aufruf verwenden, so liefert *isread* in beiden Fällen den Anfangsdatsatz.
10. Wenn Ihr *isread*-Aufruf mit dem Parameter ISCURRE, ISNEXT oder ISPREV auf einen gesperrten Datensatz trifft, ändert sich der Inhalt von *isrecnum* nicht im Vergleich zum letzten gültigen *isread*-Aufruf. Außerdem ist der aktuelle Datensatz immer noch der letzte gültige Datensatz, wie er von dem vorangehenden *isread* zurückgegeben wurde.

Wenn Sie gesperrte Datensätze überspringen wollen, verwenden Sie den Parameter ISSKIPLOCK. Ist ISSKIPLOCK angegeben, so enthält *isrecnum* die Satznummer des gesperrten Datensatzes, wenn *isread* auf einen gesperrten Datensatz trifft. Der gesperrte Datensatz wird dann zum aktuellen Datensatz. Durch nochmaliges Aufrufen von *isread*(ISNEXT) wird zum nächsten Datensatz gesprungen.

11. Wenn Ihr *isread*-Aufruf mit dem Parameter ISFIRST, ISLAST, ISEQUAL, ISGREAT, oder ISGTEQ auf einen gesperrten Datensatz trifft, wird *isrecnum* auf die Nummer des gesperrten Datensatzes gesetzt.
12. Wenn Sie die Bibliothek *libisam3.a* benutzen, um zu früheren C-ISAM-Programmen kompatibel zu sein (und wenn Sie keine Kompatibilität zu X/Open-Standards benötigen), können Sie zwei aufeinanderfolgende *isread*(ISNEXT)-Aufrufe anstoßen, um einen gesperrten Datensatz zu überspringen.
13. Sie können ISWAIT und ISLCKW nur verwenden, wenn Ihre C-ISAM-Version den Aufruf *fcntl()* zum Sperren von Datensätzen benutzt. Dies ist bei der vorliegenden C-ISAM-Version der Fall.
14. Wenn *isread* auf einen gesperrten Datensatz trifft und der Parameter ISSKIPLOCK nicht angegeben ist, geschieht folgendes:
  - Wird das Flag ISWAIT benutzt, wartet der Prozeß auf die Freigabe der Sperre.
  - Wird ISWAIT nicht benutzt, gibt der Prozeß den Wert 107 (ELOCKED) in *iserrno* zurück.
15. Sobald ein *isread*-Aufruf EENDFILE zurückgibt, ist die Position des aktuellen Datensatzes undefiniert. Rufen Sie *isread*(ISNEXT) nochmals auf, so wird der Code ENOCURR zurückgegeben.



**Rückgabewerte**

- 1 Fehler; *iserrno* enthält den Fehlercode
- 0 erfolgreich

**Beispiele**

- Der folgende Programmcode sucht den Datensatz mit dem Schlüsselwert 100 im Feld des Primärschlüssels:

```
/* 100 in die richtige Position im Datensatz bringen */
stlong(100L,&emprec[0]);

if (isread(fd,emprec,ISEQUAL)<0)
{
    if (iserrno == ENOREC) printf ("Datensatz nicht gefunden");
    .
    .
    .
}
```

- Der folgende Programmcode liest Datensatz 500:

```
pkey.k_nparts = 0; /* physikalische Reihenfolge wählen */
isrecnum = 500L; /* Satznummer auf ersten Datensatz setzen,
                 der verarbeitet werden soll */

cc = isstart(fd,&pkey,0,emprec,ISEQUAL);
if (cc >= 0)
    if (isread(fd,emprec,ISEQUAL)<0)
    {
        printf ("Lesefehler %d",iserrno);
        .
        .
        .
    }
```

## ISREWCURR

### Überblick

Mit *isrewcurr* ändern oder aktualisieren Sie Felder im aktuellen Datensatz.

### Syntax

```
isrewcurr(isfd, satz)
    int isfd;
    char *satz;
```

*isfd*

ist die Dateikennzahl, die von *isopen* oder *isbuild* zurückgegeben wird.

*satz*

enthält den vollständigen Datensatz einschließlich der geänderten Felder.

### Hinweise

1. Wenn Sie auf einen Datensatz mit variabler Länge *isrewcurr* anwenden, müssen Sie zuerst die globale Variable *isreclen* auf die tatsächliche Länge der Daten im Parameter *satz* setzen.
2. Wenn Sie ein Schlüsselfeld ändern, ändert *C-ISAM* ebenfalls den Indexeintrag.
3. Sie können den Wert des Primärschlüsselfeldes ändern.
4. Die Funktion setzt *isrecnum* auf die Satznummer des aktuellen Datensatzes. Die aktuelle Datensatzposition verändert sich nicht, d.h. *isrecnum* enthält die Satznummer des soeben geschriebenen Datensatzes.

### Rückgabewerte

-1	Fehler; <i>iserrno</i> enthält den Fehlercode
0	erfolgreich

### Beispiel

```
cc = isrewcurr(fd,emprec);
```

Wenn Sie einen Datensatz variabler Länge benutzen, könnten Sie den folgenden Aufruf verwenden. Ist die Mindestlänge des Datensatzes 84 byte, die maximale Länge 1084 byte und sind die Daten, die an die Funktion übergeben werden 923 byte lang, setzen Sie *isreclen* auf 923, bevor Sie *isrewcurr* aufrufen.

```
isreclen = 923;
cc = isrewcurr(fd, emprec);
```

## ISREWREC

### Überblick

Mit *isrewrec* ändern Sie einen Datensatz, der durch seine Satznummer spezifiziert wird.

### Syntax

```
isrewrec(isfd, satznummer, satz)
    int isfd;
    long satznummer;
    char *satz;
```

*isfd*

ist die Dateikennzahl, die von *isopen* oder *isbuild* zurückgegeben wird.

*satznummer*

ist die Satznummer.

*satz*

enthält den vollständigen Datensatz einschließlich geänderter Felder.

### Hinweise

1. Wenn Sie auf einen Datensatz mit variabler Länge *isrewrec* anwenden, müssen Sie zuerst die globale Variable *isreclen* auf die tatsächliche Länge der Daten im Parameter *satz* setzen.
2. Wenn Sie ein Schlüsselfeld ändern, ändert *C-ISAM* ebenfalls den Indexeintrag.
3. Sie können den Wert des Primärschlüsselfeldes ändern.
4. Die Funktion setzt *isrecnum* auf die Satznummer des Datensatzes.
5. Die Position des aktuellen Datensatzes wird nicht verändert.

### Rückgabewerte

- |    |   |
|----|---|
| -1 | Fehler; <i>iserrno</i> enthält den Fehlercode |
| 0  | erfolgreich                                   |

### Beispiel

Der folgende Aufruf schreibt Datensatz 404 zurück:

```
cc = isrewrec(fd,404L,emprec);
```

## ISREWRITE

### Überblick

Mit *isrewrite* schreiben Sie die nicht-primären Schlüsselfelder eines Datensatzes in eine *C-ISAM*-Datei zurück.

### Syntax

```
isrewrite(isfd, satz)
    int isfd;
    char *satz;
```

*isfd*

ist die Dateikennzahl, die von *isopen* oder *isbuild* zurückgegeben wird.

*satz*

enthält den vollständigen Datensatz einschließlich des Primärschlüssels und der geänderten Felder.

### Hinweise

1. Wenn Sie *isrewrite* auf einen Datensatz variabler Länge anwenden, müssen Sie zuerst die globale Variable *isrecLen* auf die tatsächliche Länge der Daten im Parameter *satz* setzen.
2. Der Primärschlüssel im *satz* spezifiziert den Datensatz, den Sie zurückschreiben wollen.
3. Der Primärindex muß eindeutig sein.
4. Sie können den Wert eines Primärschlüsselfeldes nicht ändern.
5. Sie können diese Funktion nicht verwenden bei Dateien, die mit *INFORMIX-4GL*, *INFORMIX-SQL* oder einer eingebetteten Sprache z.B. *INFORMIX-ESQL/C* erzeugt wurden, da die, *C-ISAM*-Dateien, die SQL-Datenbanken bilden, keine Primärindizes enthalten. Verwenden Sie stattdessen *isrewcurr* oder *isrewrec*.
6. Wenn Sie ein Schlüsselfeld in einem Index, der kein Primärindex ist, verändern, so ändert die Funktion auch den Index.
7. *C-ISAM* ändert die aktuelle Datensatzposition nicht.
8. Die Funktion setzt *isrecnum* auf die Satznummer des Datensatzes.

**Rückgabewerte**

- 1 Fehler; iserrno enthält den Fehlercode
- 0 erfolgreich

**Beispiel**

```
stchar("San Francisco",&emprec[64],20);/* zu änderndes Feld */  
cc = isrewrite(fd,emprec); /* Primärschlüssel kann  
nicht verändert werden */
```

**ISWRCURR****Überblick**

Mit *iswrcurr* schreiben Sie einen Datensatz und machen diesen zum aktuellen Datensatz.

**Syntax**

```
iswrcurr(isfd, satz)
    int isfd;
    char *satz;
```

**isfd**

ist die Dateikennzahl, die von *isopen* oder *isbuild* zurückgegeben wird.

**satz**

ist ein Zeiger auf den Datensatz, den Sie schreiben wollen.

**Hinweise**

1. Wenn Sie *iswrcurr* auf einen Datensatz variabler Länge anwenden, müssen Sie zuerst die globale Variable *isreclen* auf die tatsächliche Länge der Daten im Parameter *satz* setzen.
2. Jeder Index erhält einen Schlüssel für den Datensatz.
3. Die Funktion setzt *isrecnum* auf die Satznummer dieses Datensatzes.
4. Der Datensatz wird zum aktuellen Datensatz.

**Rückgabewerte**

-1	Fehler; <i>iserrno</i> enthält den Fehlercode
0	erfolgreich

**Beispiel**

```
stlong(101L,&emprec[0]);
.
.
.
if (iswrcurr(fd,emprec) < 0)
{
    printf ("Fehler %d bei iswrcurr",iserrno);
    .
    .
}
else /* Dieser Datensatz ist der aktuelle Datensatz. */
{
.
.
.
}
```

## ISWRITE

### Überblick

Mit *iswrite* schreiben Sie einen neuen Datensatz in eine *C-ISAM*-Datei.

### Syntax

```
iswrite(isfd, satz)  
    int isfd;  
    char *satz;
```

*isfd*

ist die Dateikennzahl, die von *isopen* oder *isbuild* zurückgegeben wird.

*satz*

ist ein Zeiger auf den Datensatz, den Sie schreiben wollen.

### Hinweise

1. Wenn Sie *iswrite* auf einen Datensatz variabler Länge anwenden, müssen Sie zuerst die globale Variable *isreclen* auf die tatsächliche Länge der Daten im Parameter *satz* setzen.
2. Jeder Index erhält einen Schlüssel für den Datensatz.
3. Die aktuelle Datensatzposition ändert sich nicht.
4. Die Funktion setzt *isrecnum* auf die Satznummer dieses Datensatzes.

### Rückgabewerte

- |    |   |
|----|---|
| -1 | Fehler; <i>iserrno</i> enthält den Fehlercode |
| 0  | erfolgreich                                   |

## Beispiel

```
stlong(100L,&emprec[0]);
.
.
.
if (iswrite(fd,emprec) < 0)
{
    printf ("Fehler %d bei iswrite",iserrno);
    .
    .
}
else /* aktuelle Datensatzposition unverändert */
{
    .
    .
    .
}
```



## Die include-Datei isam.h (Anhang C)

In der folgenden Abbildung sehen Sie den Inhalt der include-Datei *isam.h*. Diese Version der Datei enthält die notwendigen Definitionen für die Verwendung von Datensätzen variabler Länge.

```

#ifndef ISAM_INCL                /* avoid multiple include problems */
#define ISAM_INCL

#define CHARTYPE                0
#define DECIMALTYPE            0
#define CHARSIZE                1

#define INTTYPE 1
#define INTSIZE 2

#define LONGTYPE                2
#define LONGSIZE                4

#define DOUBLETYP              3
#ifndef NOFLOAT
#define DOUBLESIZE              (sizeof(double))
#endif /* NOFLOAT */

#ifndef NOFLOAT
#define FLOATTYP              4
#define FLOATSIZE              (sizeof(float))
#endif /* NOFLOAT */

#define USERCOLL(x)            ((x))

#define COLLATE1                0x10
#define COLLATE2                0x20
#define COLLATE3                0x30
#define COLLATE4                0x40
#define COLLATE5                0x50
#define COLLATE6                0x60
#define COLLATE7                0x70

#define MAXTYPE 5
#define ISDESC 0x80            /* add to make descending type */
#define TYPEMASK                0x7F    /* type mask */

#define BYTEMASK                0xFF    /* mask for one byte */
#define BYTESHFT                8      /* shift for one byte */

#ifndef ldint
#define ldint(p)                ((short)(((p)[0]<<BYTESHFT)+((p)[1]&BYTEMASK)))
#define stint(i,p)              ((p)[0]=(i)>>BYTESHFT,(p)[1]=(i))
#endif

#ifndef ldlong
long ldlong();
#endif

```

```

#ifndef NOFLOAT
#ifndef ldfloat
double ldfloat();
#endif
#ifndef lddb1
double lddb1();
#endif
double ldf1tnull();
double lddb1null();
#endif

#define ISFIRST 0      /* position to first record */
#define ISLAST 1      /* position to last record */
#define ISNEXT 2      /* position to next record */
#define ISPREV 3      /* position to previous record */
#define ISCURR 4      /* position to current record */
#define ISEQUAL 5     /* position to equal value */
#define ISGREAT 6     /* position to greater value */
#define ISGTEQ 7     /* position to >= value */

/* isread lock modes */
#define ISLOCK 0x100 /* record lock */
#define ISSKIPLOCK 0x200 /* skip record even if locked */
#define ISWAIT 0x400 /* wait for record lock */
#define ISLCKW 0x500 /* ISLOCK + ISWAIT */

/* isstart lock modes */
#define ISKEEPLOCK 0x800 /* keep rec lock in autolk mode */

/* isopen, isbuild lock modes */
#define ISAUTOLOCK 0x200 /* automatic record lock */
#define ISMANULOCK 0x400 /* manual record lock */
#define ISEXCLLOCK 0x800 /* exclusive isam file lock */

/* isopen, isbuild file types */
#define ISINPUT 0      /* open for input only */
#define ISOUTPUT1 1    /* open for output only */
#define ISINOUT 2     /* open for input and output */
#define ISTRANS 4     /* open for transaction proc */
#define ISNOLOG 8     /* no login for this file */
#define ISVARLEN 0x10 /* variable length records */
#define ISFIXLEN 0x0  /* (non-flag) fixed length records only */

/* audit trail mode parameters */
#define AUDSETNAME 0    /* set new audit trail name */
#define AUDGETNAME 1    /* get audit trail name */
#define AUDSTART 2     /* start audit trail */
#define AUDSTOP 3     /* stop audit trail */
#define AUDINFO 4     /* audit trail running? */

/*
 * Define MAXKEYSIZE 240 and NPARTS 16 for AF251
 */
#define MAXKEYSIZE 120 /* max number of bytes in key */
#define NPARTS 8      /* max number of key parts */

struct keypart
{

```

```

    short kp_start;          /* starting byte of key part */
    short kp_leng;          /* length in bytes */
    short kp_type;         /* type of key part */
};

struct keydesc
{
    short k_flags;          /* flags */
    short k_nparts;        /* number of parts in key */
    struct keypart
        k_part[NPARTS];    /* each key part */
    /* the following is for internal use only */
    short k_len;           /* length of whole key */
    long k_rootnode;       /* pointer to rootnode */
};
#define k_start    k_part[0].kp_start
#define k_leng     k_part[0].kp_leng
#define k_type     k_part[0].kp_type

#define ISNODUPS    000    /* no duplicates allowed */
#define ISDUPS     001    /* duplicates allowed */
#define DCOMPRESS  002    /* duplicate compression */
#define LCOMPRESS  004    /* leading compression */
#define TCOMPRESS  010    /* trailing compression */
#define COMPRESS   016    /* all compression */
#define ISCLUSTER  020    /* index is a cluster one */

struct dictinfo
{
    short di_nkeys;        /* number of keys defined (msb set for
        VARLEN) */
    short di_recsz;       /* (maximum) data record size */
    short di_idxsz;       /* index record size */
    long di_nrecords;     /* number of records in file */
};

#define EDUPL      100    /* duplicate record */
#define ENOTOPEN   101    /* file not open */
#define EBADARG    102    /* illegal argument */
#define EBADKEY    103    /* illegal key desc */
#define ETOOMANY   104    /* too many files open */
#define EBADFILE   105    /* bad isam file format */
#define ENOTEXCL   106    /* non-exclusive access */
#define ELOCKED    107    /* record locked */
#define EKEXISTS   108    /* key already exists */
#define EPRIMKEY   109    /* is primary key */
#define EENDFILE   110    /* end/begin of file */
#define ENOREC     111    /* no record found */
#define ENOCURR    112    /* no current record */
#define EFLOCKED   113    /* file locked */
#define EFNAME     114    /* file name too long */
#define ENOLOK     115    /* can't create lock file */
#define EBADMEM    116    /* can't alloc memory */
#define EBADCOLL   117    /* bad custom collating */
#define ELOGREAD   118    /* cannot read log rec */
#define EBADLOG    119    /* bad log record */
#define ELOGOPEN   120    /* cannot open log file */
#define ELOGWRIT   121    /* cannot write log rec */

```

```

#define ENOTRANS      122    /* no transaction */
#define ENOSHMEM     123    /* no shared memory */
#define ENOBEGIN     124    /* no begin work yet */
#define ENONFS       125    /* can't use nfs */
#define EBADROWID    126    /* reserved for future use */
#define ENOPRIM      127    /* no primary key */
#define ENOLOG       128    /* no logging */
#define EUSER        129    /* reserved for future use */
#define ENODBS       130    /* reserved for future use */
#define ENOFRREE     131    /* no free disk space */
#define EROWSIZE     132    /* row size too big */
#define EAUDIT       133    /* audit trail exists */
#define ENOLOCKS     134    /* no more locks */
#define ENOPARTN     135    /* reserved for future use */
#define ENOEXTN      136    /* reserved for future use */
#define EOVBCHUNK    137    /* reserved for future use */
#define EOVBDS       138    /* reserved for future use */
#define EOVLG        139    /* reserved for future use */
#define EGBLSECT     140    /* global section disallowing access-VMS */
#define EOVPARTN     141    /* reserved for future use */
#define EOVPAGE      142    /* reserved for future use */
#define EDEADLOK     143    /* reserved for future use */
#define EKLOCKED     144    /* reserved for future use */
#define ENOMIRROR    145    /* reserved for future use */
#define EDISKMODE    146    /* reserved for future use */
#define EARCHIVE     147    /* reserved for future use */
#define ENEMPT       148    /* reserved for future use */
#define EDEADDEM     149    /* reserved for future use */
#define EDEMO        150    /* demo limits have been exceeded */
#define EBADVCLLEN   151    /* reserved for future use */
#define EBADRMSG     152    /* reserved for future use */
#define ENOMANU      153    /* must be in ISMANULOCK mode */
#define EDEADTIME    154    /* reserved for future use */
#define EPMCHKBAD    155    /* reserved for future use */
#define EB_BUSY      160    /* reserved for future use */
#define EB_NOOPEN    161    /* reserved for future use */
#define EB_NOBS      162    /* reserved for future use */
#define EB_PAGE      163    /* reserved for future use */
#define EB_STAMP     164    /* reserved for future use */
#define EB_NOCOL     165    /* reserved for future use */
#define EB_FULL      166    /* reserved for future use */
#define EB_PSIZE     167    /* reserved for future use */
#define EB_ARCH      168    /* reserved for future use */
#define EB_CHKNLOG   169    /* reserved for future use */
/* Dismountable media blobs errors */
#define EB_SFULL     180    /* reserved for future use */
#define EB_DMENV     181    /* reserved for future use */
/* Shared Memory errors */
#define ES_PROCDEFS  21584   /* can't open config file */
#define ES_IILLVAL   21586   /* illegal config file value */
#define ES_ICONFIG   21595   /* bad config parameter */
#define ES_IILLUSRS  21596   /* illegal number of users */
#define ES_IILLCKS   21597   /* illegal number of locks */
#define ES_IILLFILE  21598   /* illegal number of files */
#define ES_IILLBUFF  21599   /* illegal number of buffs */
#define ES_SHMGET    25501   /* shmget error */
#define ES_SHMCTL    25502   /* shmctl error */
#define ES_SEMGET    25503   /* semget error */

```

```

#define ES_SEMCTL      25504   /* semctl error */

/*
 * For system call errors
 * iserrno = errno (system error code 1-99)
 * iserrrio = IO_call + IO_file
 * IO_call = what system call
 * IO_file = which file caused error
 */

#define IO_OPEN 0x10   /* open() */
#define IO_CREA 0x20   /* creat() */
#define IO_SEEK 0x30   /* lseek() */
#define IO_READ 0x40   /* read() */
#define IO_WRIT 0x50   /* write() */
#define IO_LOCK 0x60   /* locking() */
#define IO_IOCTL 0x70  /* ioctl() */

#define IO_IDX 0x01    /* index file */
#define IO_DAT 0x02    /* data file */
#define IO_AUD 0x03    /* audit file */
#define IO_LOK 0x04    /* lock file */
#define IO_SEM 0x05    /* semaphore file */

/*NOSHARE is needed as an attribute for global variables on VMS systems */
#ifdef VMS
#define NOSHARE noshare
#else /* VMS */
#define NOSHARE
#endif /* VMS */

NOSHARE extern int iserrno;           /* isam error return code */
NOSHARE extern int iserrrio;         /* system call error code */
NOSHARE extern long isrecnum;        /* record number of last call */
NOSHARE extern int isreclen;        /* actual record length, or */
                                     /* minimum (isbuild, isindexinfo) */
                                     /* or maximum (isopen) */
NOSHARE extern char isstat1;         /* cobol status characters */
NOSHARE extern char isstat2;
NOSHARE extern char isstat3;
NOSHARE extern char isstat4;
NOSHARE extern char *isversnumber;   /* C-ISAM version number */
NOSHARE extern char *iscopyright;    /* RDS copyright */
NOSHARE extern char *isserial;      /* C-ISAM software serial number */
NOSHARE extern int issingleuser;     /* set for single user access */
NOSHARE extern int is_nerr;          /* highest C-ISAM error code */
NOSHARE extern char *is_errlist[];   /* C-ISAM error messages */
/* error message usage:
 * if (iserrno >= 100 && iserrno < is_nerr)
 * printf("ISAM error %d: %s\n", iserrno, is_errlist[iserrno-100]);
 */
struct audhead
{
    char au_type[2];           /* audit record type aa,dd,rr,ww */
    char au_time[4];          /* audit date-time */
#ifdef VMS
    char au_procid[2];        /* process id number */
    char au_userid[2];        /* user id number */

```

```
#else /* VMS */
    char au_uic[4];           /* VMS user id number */
#endif /* VMS */
    char au_recnum[4];       /* record number */
    char au_reclen[2];      /* audit record length beyond header */
};
#define AUDHEADSIZE      14  /* num of bytes in audit header */
#define VAUDHEADSIZE     16  /* VARLEN num of bytes in audit header */
#endif /* ISAM_INCL */
```

## Fehlercodes (Anhang D)

Nach einem Aufruf von *C-ISAM* werden Statusinformationen in den vier Bytes *isstat1*, *isstat2*, *isstat3* und *isstat4* zurückgegeben. Diese Bytes werden hauptsächlich von COBOL-Programmen benutzt, die *C-ISAM*-Dateien verwenden. *isstat1* enthält Statusinformationen allgemeiner Art, z.B. ob ein *C-ISAM*-Aufruf erfolgreich war oder nicht; *isstat2* enthält genauere Informationen, deren Bedeutung mit dem Statuscode in *isstat1* zusammenhängt.

In der folgenden Tabelle sind die Werte von *isstat1* aufgeführt:

Wert von <i>isstat1</i>	Beschreibung
0	erfolgreiche Beendigung
1	Dateiende
2	ungültiger Schlüssel
3	Systemfehler
9	benutzerdefinierte Fehler

In der folgenden Tabelle sind die Werte von *isstat2* in Verbindung mit *isstat1* aufgeführt:

isstat1	isstat2	Bedeutung
0-9	0	Es gibt keine weiteren Informationen.
0	2	Mehrfach vorkommender Schlüsselwert gefunden.  Nach einer READ-Anweisung bedeutet dies, daß der Schlüsselwert des aktuellen Schlüssels mit dem Wert desselben Schlüssels im nächsten Datensatz übereinstimmt.  Nach einer WRITE- oder REWRITE-Anweisung bedeutet dies, daß der soeben geschriebene Datensatz einen mehrfach vorkommenden Schlüsselwert erzeugt hat, und zwar für mindestens einen anderen Datensatzschlüssel, für den mehrfach vorkommende Schlüsselwerte erlaubt sind.
2	1	Der Primärschlüsselwert wurde vom COBOL-Programm zwischen der erfolgreichen Ausführung einer READ-Anweisung und der Ausführung der nächsten REWRITE-Anweisung geändert.
	2	Sie haben versucht, einen Datensatz zu schreiben oder zurückzuschreiben, wobei ein mehrfach vorkommender Schlüssel in einer indizierten Datei entstehen würde.
	3	Kein Satz mit dem angegebenen Schlüssel vorhanden.
	4	Sie haben versucht, über die extern definierten Grenzen einer indizierten Datei hinauszuschreiben.
9		Der Wert des zweiten Statusbytes wird vom Benutzer definiert.



Die folgende Tabelle führt die Kombinationen von *isstat3* und *isstat4* auf:

<b>isstat3</b>	<b>isstat4</b>	<b>Bedeutung</b>
0	0	Erfolgreiche Beendigung; es gibt keine weiteren Informationen.
0	2	Erfolgreiche Beendigung; mehrfach vorkommender Schlüsselwert gefunden.  Nach einer READ-Anweisung bedeutet dies, daß der Schlüsselwert des aktuellen Schlüssels mit dem Wert desselben Schlüssels im nächsten Datensatz übereinstimmt. Nach einer WRITE- oder REWRITE-Anweisung bedeutet dies, daß der soeben geschriebene Datensatz einen mehrfach vorkommenden Schlüsselwert erzeugt hat, und zwar für mindestens einen anderen Datensatzschlüssel, für den mehrfach vorkommende Schlüsselwerte erlaubt sind.
1	0	Dateianfang oder -ende wurde ohne erfolgreiche Beendigung erreicht.
2	2	Sie haben versucht, einen Datensatz mit WRITE zu schreiben oder mit REWRITE zurückzuschreiben, wobei ein mehrfach vorkommender Schlüssel für einen Schlüssel entstehen würde, der dies nicht erlaubt.
2	3	Ein Datensatz mit dem angegebenen Schlüssel kann nicht gefunden werden.
3	5	Der in der Funktion <i>isopen()</i> angegebene Dateiname existiert nicht.
3	7	Der Parameter mode, der in der Function <i>isopen()</i> angegeben wurde, ist für die Datei nicht erlaubt.
3	9	Die festen Dateiattribute und der Parameter mode, der in der Funktion <i>isopen()</i> angegeben wurde, widersprechen einander.
4	2	Sie haben versucht, eine Datei zu schließen, die nicht geöffnet war.
4	3	Dieser Aufruf benötigt einen aktuellen Datensatz. Entweder es gibt keinen aktuellen Datensatz oder der aktuelle Datensatz wurde gelöscht.
4	4	Sie haben versucht, einen Datensatz mit WRITE zu schreiben oder mit REWRITE zurückzuschreiben, der größer oder kleiner als für die Datei erlaubt ist.

isstat3	isstat4	Bedeutung
4	6	Sie haben versucht mit READ und ISNEXT zu lesen, aber es gibt keinen gültigen nächsten Datensatz, da kein aktueller Datensatz definiert ist oder eine vorhergehende READ-Anweisung auf eine Endebedingung getroffen ist.
4	7	Sie haben versucht, auf eine Datei, die nicht im Modus ISINPUT oder ISINOUT geöffnet ist, eine READ-Anweisung oder <i>isstart()</i> anzuwenden.
4	8	Sie haben versucht, auf eine Datei, die nicht im Modus ISOUTPUT oder ISINOUT geöffnet ist, eine WRITE-Anweisung oder <i>iswrcurr()</i> anzuwenden.
4	9	Sie haben versucht, auf eine Datei, die nicht im Modus ISINOUT geöffnet ist, eine DELETE- oder REWRITE-Anweisung oder <i>isdelrec()</i> , <i>isdelcurr()</i> , <i>isrewrec()</i> oder <i>isrewcurr()</i> anzuwenden.
9		Benutzerdefinierte Fehler; der Wert von <i>isstat4</i> wird vom Benutzer definiert.

Die folgende Tabelle zeigt den Zusammenhang zwischen den *isstat*-Variablen und den *C-ISAM*-Fehlercodes. Bei anderen Fehlern, die *isstat3* und *isstat4* nicht unterstützen, ist *isstat3* gleich *isstat1* und *isstat4* gleich *isstat2*.

Name	Nummer	Beschreibung	isstat1	isstat2	isstat3	isstat4
EDUPL	100	Sie haben versucht, einen mehrfach vorkommenden Wert mit einer der folgenden Funktionen zu einem Index hinzuzufügen: <i>isaddindex</i> , <i>iswrite</i> , <i>isrewrite</i> oder <i>isrewcurr</i> .	2	2	2	2
ENOTOPEN	101	Sie haben versucht, eine <i>C-ISAM</i> -Datei zu bearbeiten, die zuvor nicht mit einem <i>isopen</i> -Aufruf geöffnet wurde.	9	0	4 4 4 4	2 7 8 9 0
EBADARG	102	Eines der Argumente des <i>C-ISAM</i> -Aufrufs liegt nicht innerhalb des Bereichs der für dieses Argument zulässigen Werte.	9	0	3 3 4 9	7 9 4
EBADKEY	103	Mindestens eines der Elemente, die die Schlüsselvereinbarung bilden, liegt außerhalb des Bereichs der für dieses Element zulässigen Werte.	9	0	9	0
ETOOMANY	104	Diese Anforderung würde die Maximalzahl der Dateien, die zu einem Zeitpunkt geöffnet sein dürfen, überschreiten.	9	0	9	0
EBADFILE	105	Das Format der <i>C-ISAM</i> -Datei wurde verfälscht.	9	0	9	0
ENOTEXCL	106	Um einen Index hinzuzufügen oder zu löschen, müssen Sie die Datei mit exklusivem Zugriff öffnen.	9		9	

Name	Nummer	Beschreibung	isstat1	isstat2	isstat3	isstat4
ELOCKED	107	Der/die von diesem Aufruf angeforderte Datensatz/Datei wurde von einem anderen Benutzer gesperrt. Auf ihn/sie kann daher nicht zugegriffen werden.	9		9	
EKEXIST S	108	Sie haben versucht, der <b>C-ISAM</b> -Datei einen Index hinzuzufügen, der zuvor bereits definiert wurde.	9		9	
EPRIMKEY	109	Sie haben versucht, den Primärschlüsselwert zu löschen. Der Primärschlüssel kann durch den Aufruf <b>isdelindex</b> nicht gelöscht werden.	9		9	
EENDFILE	110	Der Dateianfang oder das Dateiende wurde erreicht.	1	0	1 4	0 6
ENOREC	111	Es konnte kein Datensatz gefunden werden, der den geforderten Wert an der angegebenen Stelle enthält.	2	3	2	3
ENOCURR	112	Dieser Aufruf muß den aktuellen Datensatz bearbeiten. Der aktuelle Datensatz ist aber nicht definiert.	2	1	4 4	3 6
EFLOCKED	113	Die Datei wurde von einem anderen Benutzer exklusiv gesperrt.	9		9	
EFNAME	114	Der Dateiname ist zu lang.	9		9	0
ENOLOK	115	Die Lock-Datei kann nicht erzeugt werden.	9	0		
EBADMEM	116	Angemessener Speicherplatz kann nicht belegt werden.	9		9	
EBADCOLL	117	Fehlerhafte benutzer-spezifische Sortiersequenz	9	0		
ELOGREAD	118	Datensatz aus der Protokolldatei kann nicht gelesen werden.	9	0		

Name	Nummer	Beschreibung	isstat1	isstat2	isstat3	isstat4
EBADLOG	119	Format des Datensatzes aus der Transaktionsprotokoll-Datei kann nicht erkannt werden.	9	0		
ELOGOPEN	120	Die Transaktionsprotokoll-Datei kann nicht geöffnet werden.	9	0		
ELOGWRIT	121	In die Transaktionsprotokolldatei kann nicht geschrieben werden.	9	0		
ENOTRANS	122	Momentan läuft keine Transaktion ab.	9	0		
ENOBEGIN	124	Der Anfang der Transaktion ist nicht zu finden.	9	0		
ENONFS	125	Der Network File Server kann nicht benutzt werden.	9	0		
EBADROWID	126	Falsche Datensatznummer.	9	0		
ENOPRIM	127	Kein Primärschlüssel.	9	0		
ENOLOG	128	Keine Protokollierung.	9	0		
EUSER	129	Zu viele Benutzer.	9	0		
ENOFREE	131	Kein freier Platten-speicherplatz.	9	0		
EROWSIZE	132	Datensatz zu lang.	9	0		
EAUDIT	133	AUDIT-Protokoll vorhanden.	9	0		
ENOLOCKS	134	Keine Sperren mehr.	9	0		
EDEMO	150	Demo-Grenzen wurden überschritten.	9	0		
ENOMANU	153	Muß sich im Modus ISMANULOCK befinden.	9	0		

## Dateiformate (Anhang E)

*C-ISAM* benutzt die folgenden vier Dateiformate:

- Indexdatei-Formate
- Datendatei-Formate
- AUDIT-Datei-Formate
- Transaktionsprotokoll-Datei-Formate

In den folgenden Abschnitten werden die Formate dieser Knoten dargestellt. Die Beziehungen zwischen den Knoten werden im *C-ISAM Programmierhandbuch* in Kapitel 3 "Arbeiten mit Indizes" beschrieben. In diesem Abschnitt werden alle Datei-Formate dargestellt, nicht nur die Formate, die speziell für Datensätze variabler Länge gelten. Sie haben also beim Nachschlagen in diesem Abschnitt die Gewähr, vollständige Informationen zu erhalten.

### Formate von Indexdateien

*C-ISAM*-Indexdateien (*.idx*) enthalten die folgenden Knoten:

- Verzeichnisknoten
- Schlüsselvereinbarungsknoten
- B+ Knoten
- Knoten der Freispeicherliste
- AUDIT-Protokoll-Knoten
- Knoten des restlichen Speichers

Der Verzeichnisknoten besitzt zwei neue Felder, damit Datensätze variabler Länge unterstützt werden. Der Knoten des restlichen Speichers ist neu und wird nur für Datensätze variabler Länge benutzt. Die folgenden Tabellen beschreiben diese beiden Knoten sowie die anderen unveränderten Knoten.

## Format des Verzeichnisknotens

Byte-  
Offsets

0	2 Bytes - Validierung	Wert = FE53
2	1 Byte - Anzahl der reservierten Bytes am Anfang des Indexknotens	Wert = 2
3	1 Byte - Anzahl der reservierten Bytes am Ende des Indexknotens	Wert = 2
4	1 Byte - Anzahl der reservierten Bytes je Schlüsseleintrag (einschließlich Datensatznummer)	Wert = 4
5	1 Byte - reserviert	Wert = 4
6	2 Bytes - Knotenlänge in der Indexdatei - 1 (z.B. 511 oder 1023)	Wert = N
8	2 Bytes - Anzahl der Schlüssel	
10	2 Bytes - reserviert	
12	1 Byte - Dateiversionsnummer	
13	2 Bytes - Datensatzlänge (in Bytes)	
15	4 Bytes - Indexknotennummer der ersten Schlüsselvereinbarung	
19	6 Bytes - reserviert	
25	4 Bytes - Indexknotennummer der Liste mit freien Datensätzen	
29	4 Bytes - Indexknotennummer der Liste mit freien Indexknoten	
33	4 Bytes - Satznummer des letzten Datensatzes in der Datendatei	
37	4 Bytes - Indexknotennummer des letzten Knotens in der Indexdatei	
41	4 Bytes - Transaktionsnummer	
45	4 Bytes - eindeutige Kennung	
49	4 Bytes - Zeiger auf AUDIT-Protokoll-Information	
53	2 Bytes - maximale Datensatzlänge	
55	4 Bytes - Hash-Zeiger der Freigruppe 0	
59	4 Bytes - Hash-Zeiger der Freigruppe 1	
63	4 Bytes - Hash-Zeiger der Freigruppe 2	
67	4 Bytes - Hash-Zeiger der Freigruppe 3	
71	4 Bytes - Hash-Zeiger der Freigruppe 4	

Format des Schlüsselvereinbarungsknotens

Byte-Offsets			
0	2 Bytes	- Anzahl der in diesem Knoten benutzten Byte	
2	4 Bytes	- Indexknoten für Fortsetzung der Schlüsselvereinbarungen	
6	2 Bytes	- Länge der Vereinbarung	Wird für jeden Schlüssel wiederholt
8	4 Bytes	- Indexknotennummer der Wurzel	
12	1 Byte	- Komprimierungs-Flags	
13	2 Bytes	- Länge des Schlüsselteils 1 (höchstwertiges Bit = Duplikate)	Wird für jeden Schlüsselteil wiederholt
15	2 Bytes	- Position im Datensatz	
17	1 Byte	- Datentypparameter	
	.		
	.		
	.		
n-2	1 Byte	- Flag	Wert = FF
n-1	1 Byte	- Ende des Schlüsselvereinbarungsknotens	Wert = 7E



## Format des Knotens für den Restspeicher

Byte-  
Offsets

0	2 Bytes - reserviert
2	2 Bytes - konstante Zahl 7E26 hex
4	4 Bytes - Vorwärtszeiger in gestreuter (hashed) Freispeicherliste der Restspeicherseite
8	4 Bytes - Rückwärtszeiger in gestreuter (hashed) Freispeicherliste der Restspeicherseite
12	2 Bytes - freier Platz, der auf dieser Seite des Restspeichers verfügbar ist Wert = 4
14	2 Bytes - Offset zum freien Platz auf dieser Seite des Restspeichers
16	4 Bytes - Restzeiger auf nächsten Restspeicher, falls vorhanden
20	1 Byte - Flags
21	1 Byte - Anzahl der zugeordneten Slots
22	1 Byte - Hashgruppe für Benutzung der Freispeicherliste
23	versch. - Datenspeicherplatz
	versch. - freier Platz
	4 Bytes - Slot-Tabelle, niedrigster Eintrag
	4 Bytes - Einträge der Slot-Tabelle
	⋮
n-6	4 Bytes - Slot-Tabelle, höchster Eintrag
n-2	1 Byte - Typ:7C hex
n-1	1 Byte - reserviert

## Format des B+ Baum-Knotens

Byte- Offsets			
0	2 Bytes	- Anzahl der in diesem Knoten benutzten Byte	
2	1 Byte	- Anzahl der führenden Bytes (bei Komprimierung)	} Wird für jeden Schlüssel eintrag wieder- holt
3	1 Byte	- Anzahl der nachgestellten Leerzeichen (bei Komprimierung)	
4	K Bytes	- Schlüssel (kann komprimiert sein)	
4+K	2 Bytes	- für mehrfach vorkommenden Schlüssel (bei Komprimierung)	
6+K	4 Bytes	- Zeiger auf Datensatz (höchstwertiges Bit kann ein Flag für mehrfache Schlüsselwerte sein)	
	.		
	.		
	.		
N-2	1 Byte	- Indexbaumnummer (dies ist immer das zweitletzte Byte im Knoten)	
N-1	1 Byte	- Ebene im Baum (0 = Blattknoten) (dies ist immer das letzte Byte im Knoten)	

## Format des Knotens der Freispeicherliste

Byte-  
Offsets

0	2 Byte - Anzahl der in diesem Knoten benutzten Byte
2	4 Byte - Anzahl der führenden Byte (bei Komprimierung)
6	N-8 Byte - Anzahl der nachgestellten Leerzeichen (bei Komprimierung)
	.
	.
	.
N-2	1 Byte - FF bedeutet Freispeicherliste für Datendatei FE bedeutet Freispeicherliste für Indexdatei
N-1	1 Byte - Flag für das Ende des Knotens der Freispeicherliste <span style="float: right;">Wert = 7F</span>

## Format des AUDIT-Protokoll-Knotens

Byte-  
Offsets

0	2 Bytes - Anzahl der in diesem Knoten benutzten Byte
2	2 Bytes - Flags <span style="float: right;">0 = AUDIT-Protokoll eingeschaltet 1 = AUDIT-Protokoll ausgeschaltet</span>
4	64 Bytes - Pfadname des AUDIT-Protokolls
	.
	.
	.
N-2	1 Byte - Ende des AUDIT-Protokollknotens <span style="float: right;">Wert = 7D</span>

### Formate von Datendateien

Datendateien (*.dat*) enthalten nur Datensätze mit festgelegter Länge, ein Flag am Ende jedes Datensatzes und, falls die Daten auch einen Teil variabler Länge enthalten, zwei zusätzliche Felder, die die Länge und Position des variablen Teils beschreiben.

Ist das Flag gleich Null (ASCII-Null), so ist der Datensatz gelöscht. Die folgende Tabelle zeigt das Datendatei-Format.

Byte- Offsets	
0	Anfang des Datensatzes mit Länge R
R	1 Byte - Lösch-Flag
R+1	2 Bytes - Länge der gültigen Daten im Restspeicher- teil; kann kleiner als der bereitgestellte Speicher sein
R+3	4 Bytes - Das erste Byte ist die Slot-Nummer (wo der erste Teil des Restes gespeichert wird); die letzten drei Byte stellen die Knotennummer des Restspeichers dar

### Formate von AUDIT-Dateien

Die AUDIT-Datei enthält Datensätze, die aus einem Vorspann mit festgelegter Länge und einem Abbild des Datensatzes bestehen. Gehört die AUDIT-Datei zu einer Datei mit Datensätzen variabler Länge, enthält sie einen Eintrag, der aus zwei Bytes besteht und der die tatsächliche Länge des Datensatzes angibt. (Dieser Eintrag wird bei AUDIT-Dateien für Dateien mit Datensätzen fester Länge nicht benutzt.)

Die folgende Tabelle zeigt das Format von AUDIT-Dateien für Datensätze variabler Länge:

Byte-Offsets	
0	2 Bytes - Datensatztyp im AUDIT-Protokoll (aa, dd, rr oder ww)
2	4 Bytes - Zeit
6	2 Bytes - Prozeßnummer (pid)
8	2 Bytes - Benutzeridentifikator (uid)
10	4 Bytes - Datensatznummer aus der Datendatei
14	2 Bytes - tatsächliche Länge der Daten variabler Länge in Byte
16	R Bytes - Abbild des Datensatzes
R+16	

Wenn es sich um eine Operation handelt, die Daten zurückschreibt, werden sowohl das Before als auch das After Image in der AUDIT-Datei aufgezeichnet. Das Before Image wird zuerst als vom Typ *rr* aufgelistet; es folgt das After Image vom Typ *ww*. Beide Abbilder haben dieselbe Datensatznummer.

### Formate von Transaktionsprotokoll-Dateien

Datensätze aus der Transaktionsprotokoll-Datei enthalten einen Vorspann mit festgelegter Länge sowie andere Informationen, die vom Transaktionstyp abhängen. Die folgende Tabelle zeigt das Format des Vorspanns.

## Format des Vorspanns bei Transaktionsdatensätzen

Byte-Offsets	
0	2 Bytes - Länge des Protokoll-Datensatzes
2	2 Bytes - Transaktionstyp
6	2 Bytes - Transaktionsidentifikator
8	2 Bytes - Benutzeridentifikator (uid)
10	2 Bytes - Transaktionszeit
18	8 Bytes - reserviert
	.
	.
	.

## Beispiel

Das Format des Vorspanns einer Transaktionsprotokoll-Datei ist gleich für Datensätze fester wie variabler Länge. Allerdings wurden einige der Transaktionstypen so geändert, daß auch Datensätze variabler Länge berücksichtigt werden können. Das folgende Beispiel listet alle Transaktionstypen auf.

```

/* Definition des Datensatzvorspanns */
#define LG_LEN          0          /* aktuelle Datensatzlänge */
#define LG_TYPE        LG_LEN+INTSIZE /* Typ des Protokoll-Datensatzes */
#define LG_XID         LG_TYPE+2   /* Transaktionsidentifikator */
#define LG_USER        LG_XID+INTSIZE /* Benutzerkennung */
#define LG_TIME        LG_USER+2   /* Transaktionszeit */
#define LG_PREV        LG_TIME+LONGSIZE /* vorhergeh. Protokoll-Datensatz */
#define LG_PREVLEN     LG_PREV+LONGSIZE /* vorhergeh. Protokoll-Länge */

/* Datensatzdefinition für BEGIN, COMMIT und ROLLBACK WORK */
#define LG_TXSIZE      LG_PREVLEN+INTSIZE+INTSIZE /* Datensatzlänge */

/* Datensatzdefinition für das Erstellen einer Datei */
#define LG_FMODE       LG_PREVLEN+INTSIZE /* Aufbaumodus */
#define LG_RECLEN      LG_MODE+INTSIZE /* minimale Datensatzlänge */
#define LG_MAXLEN      LG_RECLEN+INTSIZE /* maximale Datensatzlänge oder null */
#define LG_KFLAGS      LG_MAXLEN+INTSIZE /* Schlüssel-Flag */
#define LG_NPARTS      LG_KFLAGS+INTSIZE /* Anzahl der Teile des Schlüssels */
#define LG_KLEN        LG_NPARTS+INTSIZE /* Schlüssellänge insgesamt */

/* Datensatzdefinition für das Löschen einer Datei */
#define LG_FNAME       LG_PREVLEN+INTSIZE /* Pfadname des Dateiverzeichnisses */

```

```

/* Datensatzdefinition für das Umbenennen einer Datei */
#define LG_OLEN      LG_PREVLEN+INTSIZE /* Länge des alten Dateinamens */
#define LG_NLEN      LG_OLEN+INTSIZE   /* Länge des neuen Dateinamens */
#define LG_ONAME     LG_NLEN+INTSIZE   /* alter Dateiname */

/* Datensatzdefinition für das Öffnen und Schließen einer Datei */
#define LG_ISFD      LG_PREVLEN+INTSIZE /* Dateikennzahl */
#define LG_VARLEN    LG_ISFD+INTSIZE   /* VARLEN-Flag der Datei */
#define LG_FPATH     LG_VARLEN+INTSIZE /* Pfadname d. Dateiverzeichnisses */

/* Index erzeugen und löschen */
#define LG_IFLAGS    LG_ISFD+INTSIZE   /* Schlüssel-Flags */
#define LG_INPARTS   LG_IFLAGS+INTSIZE /* Anzahl der Teile d. Schlüssels */
#define LG_IKLEN     LG_INPARTS+INTSIZE /* Schlüssellänge insgesamt */

/* eindeutige Kennung festsetzen */
#define LG_UNIQID    LG_ISFD+INTSIZE   /* neue eindeutige Kennung */

/* Datensatzdefinition für Before und After Images */
#define LG_RECNO     LG_ISFD+INTSIZE   /* Satznummer */
#define LG_IMGLLEN   LG_RECNO+LONGSIZE /* Länge des Datensatzabbilds */
#define LG_RECORD    LG_IMGLLEN+INTSIZE /* Satzdaten */

/* Abbild der Änderung (vorher (Before Image) und nachher (After Image) zusammen) */
#define LG_BEFLLEN   LG_RECNO+LONGSIZE /* Länge des Before Image */
#define LG_AFTLEN    LG_BEFLLEN+INTSIZE /* Länge des After Image */
#define LG_BUPDATE   LG_AFTLEN+INTSIZE /* Before Image für Änderung */
/* (gefolgt von After Image) */

/* Datensatz für einen Sicherungspunkt */
#define LG_SAVEPT    LG_PREVLEN+INTSIZE /* Nummer des Sicherungspunktes */

#define LG_SSIZE     LG_SAVEPT+INTSIZE  /* Satzlänge */

#define LG_PAGESIZE  4096                /* voreingestellte Protokoll-Puffergröße */

/* Typen der Protokoll-Datensätze */
#define LG_ERROR     0                    /* Fehler bei Lesen oder Schreiben des Protokolls */
#define LG_BEGWORK   1                    /* BEGIN WORK */
#define LG_COMWORK   2                    /* COMMIT WORK */
#define LG_ROLWORK   3                    /* ROLLBACK WORK */
#define LG_DELETE    4                    /* gelöschter Datensatz */
#define LG_INSERT    5                    /* neu eingefügter Datensatz */
#define LG_UPDATE    6                    /* geänderter Datensatz */
#define LG_VERSION   7                    /* Version */
#define LG_SVPOINT   8                    /* Sicherungspunkt */
#define LG_FOPEN     9                    /* Datei öffnen */
#define LG_FCLOSE    10                   /* Datei schließen */
#define LG_CKPOINT   11                   /* Checkpoint */
#define LG_BUILD     12                   /* neue Datei aufbauen */
#define LG_ERASE     13                   /* alte Datei löschen */
#define LG_RFORWARD  14                   /* ROLLFORWARD */
#define LG_CINDEX    15                   /* Index erzeugen */
#define LG_DINDEX    16                   /* Index löschen */
#define LG_EOF       17                   /* Ende der Protokolldatei */
#define LG_RENAME    18                   /* Datei umbenennen */
#define LG_SETUNIQID 19                   /* eindeutige Kennung festsetzen */
#define LG_UNIQUEID  20                   /* eindeutige Kennung holen */
#define LG_RBSVPT    21                   /* Zurücksetzen auf Sicherungspunkt */

#define TRUE         1
#define FALSE        0

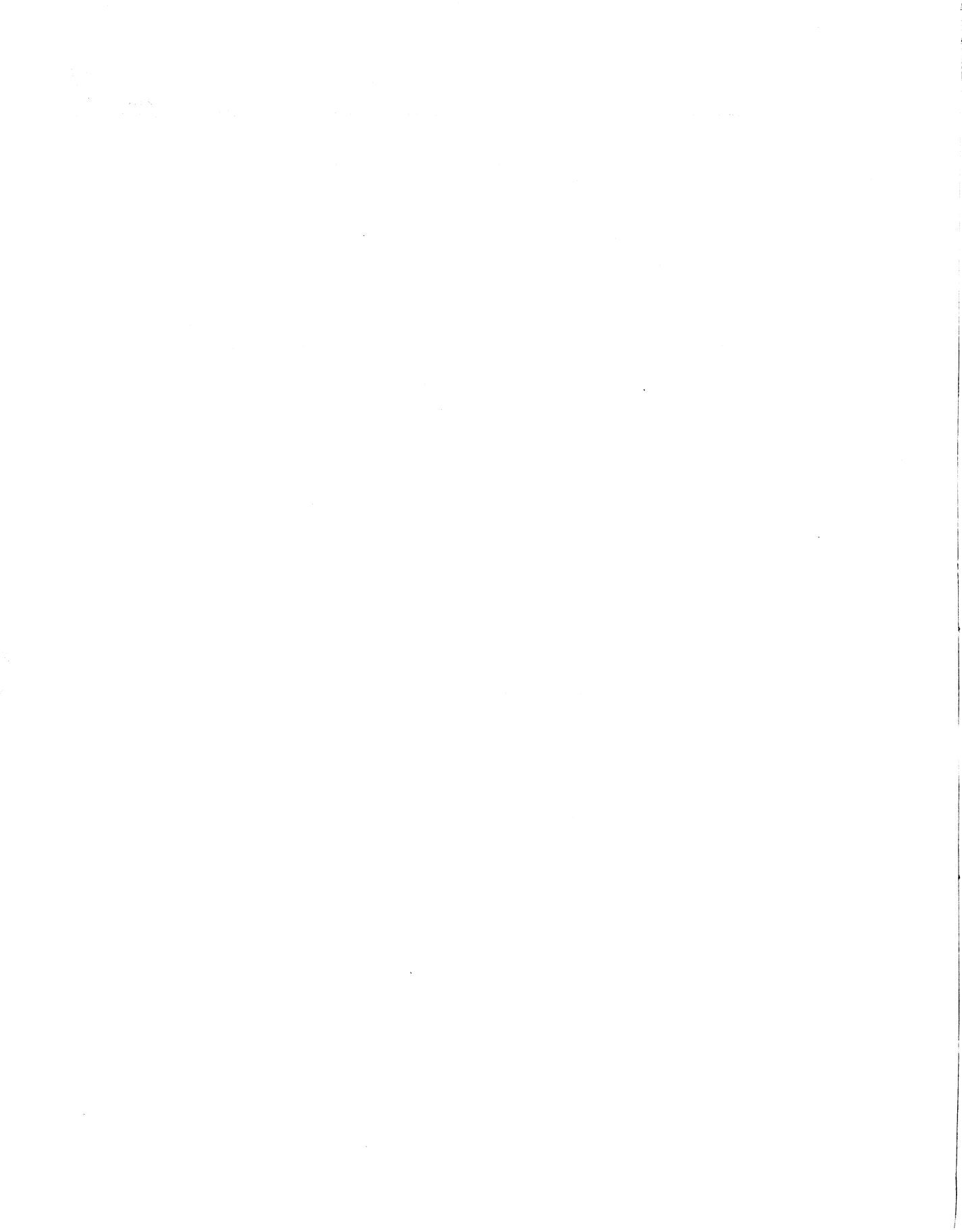
struct txlist
{

```

```
    int tx_xid;                /* Transaktionsidentifikator */
    struct xrloc *tx_nextrec; /* nächster Protokoll-Datensatz in Transaktion */
    struct txlist *tx_next;   /* nächste Transaktion */
};

struct xrloc
{
    int xr_logtype;          /* Typ von Protokoll-Datensätzen */
    int xr_size;            /* Größe von Protokoll-Datensätzen */
    long xr_loc;           /* Stelle in Protokolldatei */
    struct xrloc *xr_next; /* nächster Protokoll-Datensatz in der Transaktion*/
};
```





---

# INFORMIX Interactive Debugger Version 4.1

Dieses Kapitel enthält die Neuerungen für den interaktiven Debugger von INFORMIX V4.1.

Es beginnt mit einem Überblick über die Änderungen, so daß Sie spezielle Punkte leicht finden können. Die Reihenfolge der modifizierten Kommandos ist alphabetisch, wie in Kapitel 9 der Version 4.0.

Danach folgt die vollständige Beschreibung der Kommandoänderungen. Im letzten Abschnitt dieses Kapitels sind die Fehler, die sich in der Beschreibung des interaktiven Debugger angefundnen haben, berichtet.

Beschreibungen von Umgebungsvariablen und der Dienstprogramme, mit denen die meisten der hier beschriebenen INFORMIX-Produkte arbeiten, finden Sie im Anhang dieses Buches.

## Änderungen im Überblick

### Kapitel 9 - Kommandobeschreibung

Die folgenden Kommandos haben für die Arbeit mit ONLINE zusätzliche Funktionalität erhalten:

- BREAK
- CALL
- CLEANUP
- DUMP
- LET
- PRINT
- TRACE
- VARIABLE
- WHERE

### Anhang B - Umgebungsvariablen

Folgende Umgebungsvariablen kommen neu zur Funktionalität von INFORMIX dazu oder wurden geändert:

- DBFORMAT                      numerische Werte bei Ein- und Ausgaben formatieren
- INFORMIXDIR                 Dateiverzeichnis der INFORMIX-Produkte
- INFORMIXTERM                Bildschirmsteuerung wählen
- PSORT\_NPROCS                Menge der Prozessoren für paralleles  
  (nur ONLINE)                 Sortieren
- PSORT\_DBTEMP                temporäres Dateiverzeichnis für paral-  
  (nur ONLINE)                 lele Sortierung
- SACEISQL                     Isolationsstufe für Listenprogramme  
  (nur ONLINE)                 setzen

Die Beschreibung der neuen Umgebungsvariablen finden Sie im Anhang dieses Buches. Sie finden weiterhin im Anhang eine überarbeitete und geänderte Beschreibung des Dienstprogramms *bcheck* und eine Anleitung, wie Sie Fehlermeldungen am Bildschirm oder Drucker ausgeben können, wenn Sie mit einer ONLINE-Datenbank arbeiten.

## Beschreibung der Änderungen

Bestimmte Kommandos des interaktiven Debuggers haben eine zusätzliche Funktionalität, wenn Sie mit einem ONLINE-System arbeiten. Diese Kommandos werden im folgenden beschrieben.

### BREAK

Sie können auch Variablen der Datentypen VARCHAR, TEXT und BYTE mit einem BREAK-Kommando verwenden. Das BREAK-Kommando für TEXT- und BYTE-Variablen wird immer dann ausgeführt, wenn den Variablen ein neuer Wert zugewiesen wird. Bei Variablen anderer Datentypen wird im Gegensatz dazu BREAK nur dann ausgeführt, wenn sich der Wert der Variablen ändert.

In einer bedingten Anweisung (IF) innerhalb des BREAK-Kommandos dürfen Sie keine TEXT- oder BYTE-Variablen benutzen, es sei denn, Sie fragen diese Variablen auf den NULL-Wert ab.

#### Beispiel

```
BREAK ...  
  IF textvar IS NULL
```

### CALL

Sie können mit CALL auch Funktionen aufrufen, die Variablen der Datentypen VARCHAR, TEXT und BYTE als Parameter übergeben.

#### Beispiel

```
CALL myfunc(textvar)
```

## CLEANUP

Zusätzlich zu den bereits bestehenden Möglichkeiten, mit CLEANUP Variablen zu initialisieren und alle geöffneten Fenster und Formate zu schließen, setzt CLEANUP automatisch allen Speicher frei, der von Variablen der Datentypen TEXT und BYTE belegt ist. Wenn Sie eine TEXT- oder BYTE-Variable in einer Datei abgelegt haben (IN FILE), deren Name nicht vom Programm gewählt wurde, werden mit CLEANUP die temporären Dateien freigesetzt, die 4GL benutzt hat, um die Variable zu speichern. Nachdem CLEANUP ausgeführt ist, müssen Sie jede BLOB-Variable, die Sie wieder benutzen wollen, mit LOCATE initialisieren.

## DUMP

Wenn die aktuelle Funktion eine TEXT-Variable benutzt, wird nur die erste Zeile der Datei ausgegeben. Bei einer BYTE-Variable wird nur die Meldung `<byte value>` am Bildschirm angezeigt, und nicht der Inhalt der Variablen. Wenn die TEXT- oder BYTE-Variable den NULL-Wert enthält, wird statt der ersten Zeile oder der Meldung `<byte value>` die Meldung `(NULL)` am Bildschirm ausgegeben.

## Beispiel

In den folgenden Beispielen ist *varname* die Variable, deren Wert ausgegeben werden soll.

TEXT-Variablen, die entweder im Hauptspeicher oder in einer Datei gespeichert sind, werden folgendermaßen ausgegeben:

```
varname = "erste Zeile der Datei"
```

BYTE-Variablen, die entweder im Hauptspeicher oder in einer Datei gespeichert sind, werden folgendermaßen ausgegeben:

```
varname = <byte value>
```

TEXT- und BYTE-Variablen, die den Wert NULL enthalten, werden folgendermaßen ausgegeben:

```
varname = (NULL)
```

## LET

Mit dem Kommando LET können Sie einer VARCHAR-Variable Werte zuweisen. Der Datentyp VARCHAR kann auch in Ausdrücken innerhalb eines LET-Kommandos auftreten.

Dies gilt nicht für BLOB-Datentypen. Sie können TEXT- oder BYTE-Variablen mit LET nur NULL-Werte zuweisen. In Ausdrücken innerhalb des LET-Kommandos dürfen Sie TEXT und BYTE nicht verwenden.

Wenn Sie einer BLOB-Variablen einen NULL-Wert zuweisen, wird deswegen kein Speicherplatz freigegeben, sondern nur die Indikatorvariable geändert. Bevor Sie allerdings einer BLOB-Variablen mit LET den NULL-Wert zuweisen können, müssen Sie die BLOB-Variable mit LOCATE initialisieren.

### Beispiel

```
LET bytevar = NULL
```

## PRINT

Mit der folgenden Anweisung können Sie das aktuelle Datum und die Uhrzeit ausgeben lassen:

### Beispiel

```
PRINT CURRENT
```

Wenn Sie mit PRINT eine TEXT-Variable ausgeben lassen, wird der gesamte Inhalt der Variablen angezeigt. Ist der Wert der Variablen länger als eine Bildschirmseite, wird der Inhalt seitenweise ausgegeben. "Blättern" können Sie mit RETURN, da am Seitenende jeweils die Meldung `Press RETURN to continue` erscheint. Mit der Anweisung

```
PRINT var[i,j]
```

können Sie die Länge des Variablenwerts in der Ausgabe steuern. Dabei muß *var* eine Variable vom Datentyp TEXT sein. Für *i*, *j* können Sie INTEGER-Werte angeben, die Ausgabeanfang bzw.-ende der Variablen bestimmen. Ist die Ausgabe von *i* bis *j* länger als eine Bildschirmseite, können Sie mit RETURN "blättern". Mit der folgenden PROGRAM-Anweisung können Sie ein Programm, z.B. einen Editor, angeben, mit dem Sie TEXT und BYTE-Variablen bearbeiten können:

```
PRINT blobvar PROGRAM = "programmname"
```

Wenn Sie sich mit PRINT eine BYTE-Variable ohne PROGRAM-Anweisung anzeigen lassen, wird nur die Meldung `<byte value>` angezeigt und nicht der Inhalt der Variablen.

**TRACE**

Sie können auch die Änderung von Werten bei VARCHAR-, TEXT- und BYTE-Variablen verfolgen.

Wenn Sie für eine BLOB-Variable einen Ablaufverfolgungspunkt nach einer LOCATE-Anweisung gesetzt haben, erhalten Sie, je nach Fundort (Hauptspeicher oder Datei) folgende Meldungen:

```
varname LOCATED IN FILE "dateiname" oder varname LOCATED IN MEMORY
```

Bei Änderung des Wertes einer TEXT-Variablen sowohl im Hauptspeicher als auch in einer Datei erhalten Sie folgende Meldung:

```
varname SET TO "erste Zeile der Datei" IN <funktion, zeile, modul>
```

*erste Zeile der Datei* ist dabei die ausgegebene erste Zeile des neuen Wertes. Bei *<funktion, zeile, modul>* wird die Funktion, Zeilennummer und das Modul angegeben, in der die Variable gefunden wurde.

Bei Änderung des Wertes einer BYTE-Variablen sowohl im Hauptspeicher als auch in einer Datei erhalten Sie folgende Meldung:

```
varname SET TO <byte value> IN <funktion, zeile, modul>
```

*<byte value>* ist dabei die ausgegebene Meldung am Bildschirm. Bei *<funktion, zeile, modul>* wird die Funktion, Zeilennummer und das Modul angegeben, in der die Variable gefunden wurde.

Bei Änderung des Wertes einer TEXT- oder BYTE-Variablen, die den NULL-Wert enthält, erhalten Sie folgende Meldung:

```
varname = (NULL)
```

Bei Änderung des Wertes einer TEXT- oder BYTE-Variablen zum NULL-Wert erhalten Sie folgende Meldung:

```
varname CHANGED TO (NULL) IN <funktion, zeile, modul>
```

**VARIABLE**

Das Kommando VARIABLE zeigt BLOB-Variablen folgendermaßen an:

- BYTE- und TEXT-Variablen vor einer LOCATE- oder nach einer FREE-Anweisung:

```
varname TYPE {TEXT}
              {BYTE}
```

- BYTE- und TEXT-Variablen im Hauptspeicher nach einer LOCATE-Anweisung:

```
varname TYPE {TEXT} IN MEMORY
              {BYTE}
```

- BYTE- und TEXT-Variablen in einer Datei nach einer LOCATE-Anweisung:

```
varname TYPE {TEXT} IN FILE "dateiname"
              {BYTE}
```

**WHERE**

Mit dem Kommando WHERE können Sie sich auch TEXT- und BYTE-Argumente anzeigen lassen. Name und Wert der Argumente werden folgendermaßen angezeigt:

- ein BLOB-Argument, dessen Wert NULL ist

```
varname = (NULL)
```

*varname* ist hierbei der Name des Argumentes, (NULL) der ausgegebene Wert.

- ein TEXT-Argument

```
varname = "erste Zeile der Datei"
```

*erste Zeile der Datei* ist hierbei die erste Zeile des TEXT-Argumentes.

- ein BYTE-Argument

```
varname = <byte value>
```

*<byte value>* ist die aktuelle Meldung am Bildschirm für den Wert des BYTE-Argumentes.



## Fehlerbehebungen

Im Band INFORMIX Interactive Debugger V4.0 sind einige Fehler aufgetreten, die im folgenden berichtet werden sollen.

Anhand der Seitenzahl und der Kapitelnummer können Sie die zu verbessernden Stellen im Handbuch leicht finden.

### Kapitel 4, Seite 4-22

#### Erreichen des ersten Unterbrechungspunktes

Der Anweisungspunkt *Beantworten Sie die Abfrage* durch Drücken von RETURN ist falsch. Richtig muß es heißen:

Beantworten Sie die Abfrage durch Drücken von ESCAPE.

### Kapitel 7, Seite 7-18

#### Abbruchfehler 2

In Kapitel 7 soll anhand des Programms *cust\_order* gezeigt werden, wie Sie zwei typische Laufzeitfehler erkennen und beheben können. Der zweite Fehler wird auf Seite 7-18 beschrieben.


Allerdings tritt dieser Fehler nur dann auf, wenn die Tabelle *stock* mindestens 16 Sätze enthält. Die ausgelieferte Beispieldatenbank enthält jedoch genau 15 Sätze.

Wenn Sie also alle Schritte in Kapitel 7 durcharbeiten wollen, müssen Sie in die Tabelle *stock* einen Satz aufnehmen.

Wenn Sie das folgende 4GL-Programm compilieren und ablaufen lassen, wird an die Tabelle *stock* ein Satz angefügt.

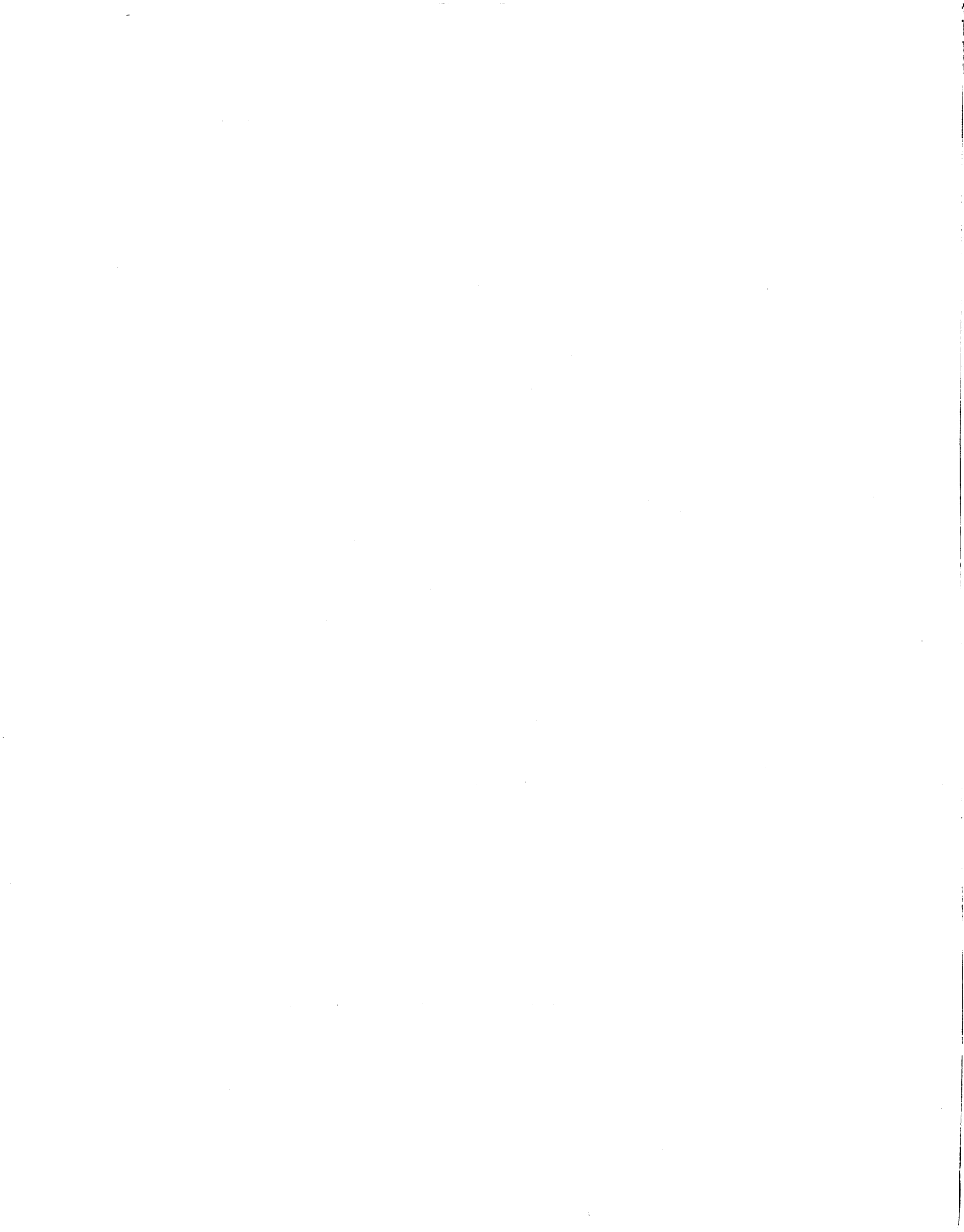
```
#####
#
# title: add_stock.4gl
#
# purpose: This programm adds a single row to the stock table.
#
# notes: You will recieve an error if the stock_number and
# manufacturer code duplicates an existing value. This means that
# you must edit the value in the stock_num and manu_code columns
# if you want to run this program more than one time.
#
#####
```

```
DATABASE stores
MAIN
  INSERT INTO stock VALUES
    (7, "SMT", "basketball", 800.00, "case", "24/case")
END MAIN
# end add_stock.4gl program
```

 Die Ausgaben des 4GL-Programms werden über Pipe an das Ausgabefenster des Debugger gereicht. Wenn im 4GL-Programm Shell-Prozeduren aufgerufen werden, kann die Ausgabe des Debuggers vom normalen Programmablauf abweichen.

### Beispiel

Die Shell-Prozedur ist selbst eine Pipekette und enthält als letzte Anweisung *pg* oder *more*. In diesem Fall erfolgt keine seitenweise sondern eine fließende Ausgabe.



---

## Anhang

- A.1 Umgebungsvariablen
- A.2 Dienstprogramme
- A.3 Fehlermeldungstexte (nur ONLINE) ausgeben
- A.4 Reservierte Wörter

### Umgebungsvariablen

Sie können die "Umgebung" beeinflussen, in der INFORMIX arbeitet. Dies geschieht über Variablen, deren Inhalt Sie individuell verändern können und die INFORMIX anschließend entsprechend auswertet. Die folgende Tabelle faßt alle von INFORMIX verwendeten Umgebungsvariablen kurz zusammen:

Umgebungsvariable	Bedeutung
DBANSIWARN	Überwachung des ANSI-Standards
DBDATE	Ein- und Ausgabeformat für das Datum
DBDELIMITER	Trennzeichen definieren
DBEDIT	Texteditor bestimmen
DBFORMAT	Eingabeformat für numerische Werte bestimmen
DBMENU	Standard-Benutzermenü vereinbaren
DBMONEY	Ausgabeformat für Geldbeträge
DBPATH	Pfade für Datenbanken und Dateien
DBPRINT	Ausgabeprogramm definieren
DBTEMP	Dateiverzeichnis für temporäre Dateien
DBLANG	Dateiverzeichnis für Meldungsdateien
INFORMIXDIR	Dateiverzeichnis für INFORMIX-Dateien
INFORMIXTERM	Bildschirmsteuerung termcap/terminfo
SQLEXEC	INFORMIX-Backend bestimmen
TBCONFIG	<i>tbconfig</i> -Datei für das INFORMIX-ONLINE-System

Ist eine Umgebungsvariable nicht gesetzt, so benutzt INFORMIX einen Standardwert. Die Umgebungsvariable DBFORMAT ist neu hinzugekommen, INFORMIXDIR und INFORMIXTERM sind verändert worden. Eine Beschreibung der neuen Umgebungsvariable und der Änderungen finden Sie im folgenden Abschnitt, eine ausführliche Beschreibung der restlichen Umgebungsvariablen finden Sie im Anhang des Handbuchs INFORMIX-SQL-Nachschlagen[2].

Für ONLINE-Datenbanken gibt es drei neue Umgebungsvariablen:

PSORT_NPROCS	Prozessoren für paralleles Sortieren angeben
PSORT_DBTEMP	temporäre Verzeichnisse für paralleles Sortieren wählen
SACEISQL	Isolationsstufe für Listenprogramme setzen

## Umgebungsvariable DBFORMAT

Mit DBFORMAT können Sie bestimmen, wie numerische Werte, die ein Anwender eingibt oder die von INFORMIX eingelesen und ausgegeben werden, ausgegeben werden. Es können alle numerischen Werte mit DBFORMAT formatiert werden. DBFORMAT überlagert die Definition von DBMONEY.

```
DBFORMAT='anfang:tausend:dezimal:ende'
```

### anfang

ist ein alphanumerischer Wert, der bis zu sieben Zeichen lang sein kann und das jeweilige, länderspezifische Währungszeichen enthält. INFORMIX wertet diese Angabe dann nur bei Bildschirmfeldern mit dem Datentyp MONEY aus.

Sie können *anfang* auch weglassen, den nachfolgenden Doppelpunkt müssen Sie schreiben.

### tausend

hier können ein oder mehrere Zeichen stehen, die INFORMIX als Tausendertrennzeichen verwendet (im Deutschen ist dies ein Punkt .). Folgende Zeichen dürfen Sie nicht benutzen:

- Zahlen
- <, >, |, :, ?, !, =, [, ]

Sie können das Tausendertrennzeichen auch weglassen, und statt dessen beim Setzen von DBFORMAT ein Leerzeichen angeben. Das Standardtrennzeichen ist dann ein Stern \*. Wenn Sie den Stern als Tausendertrennzeichen angeben, wird er nicht ausgegeben.

Das Tausendertrennzeichen wird nur dann am Bildschirm oder in einer Liste ausgegeben, wenn Sie es im dazugehörigen Programm mit den Anweisungen USING oder FORMAT definieren, ansonsten wird es unterdrückt (siehe auch Beispiel).

Wenn Sie mehrere Zeichen angegeben haben, die als Trenner benutzt werden können, nimmt INFORMIX das erste Zeichen, das der Anwender in ein Format als Trennzeichen eingibt.

#### dezimal

hier können ein oder mehrere Zeichen stehen, die INFORMIX als Dezimaltrennzeichen verwendet (im Deutschen ist dies ein Komma , ). Folgende Zeichen dürfen Sie nicht benutzen:

- Zahlen
- <, >, |, :, ?, !, =, [, ], \*
- jedes Zeichen, das als Tausendertrennzeichen vergeben ist

Das Dezimaltrennzeichen müssen Sie angeben.

Wenn Sie mehrere Zeichen angegeben haben, die als Trenner benutzt werden können, nimmt INFORMIX das erste Zeichen, das der Anwender in ein Format als Trennzeichen eingibt.

#### ende

ist ein alphanumerischer Wert, der bis zu sieben Zeichen lang sein kann und das jeweilige, länderspezifische Währungszeichen enthält. INFORMIX wertet diese Angabe nur bei Bildschirmfeldern mit dem Datentyp MONEY aus.

Sie können *ende* auch weglassen, den vorangehenden Doppelpunkt müssen Sie angeben.

Standardbelegung für DBFORMAT ist '...:.'

## Beispiel

Die Umgebungsvariable DBFORMAT ist folgendermaßen belegt:

```
DBFORMAT='DM:.,,:'
```

Wenn Sie für die MONEY-Spalte *stueckpreis* keine USING-Anweisung angeben, druckt ACE in einer Liste die Spaltenwerte von *stueckpreis* wie folgt:

posten_nr	hersteller	beschreibung	liefereinheit	stueckpreis
1	HRO	SKI-Handschuh	Box	DM250,00
1	HSK	SKI-Handschuh	Box	DM800,00
1	SMT	SKI-Handschuh	Box	DM450,00
4	HSK	Fussball	Col.	DM960,00
4	HRO	Fussball	Col.	DM480,00
.	.	.	.	.
9	ANZ	Volleyb. profi	solo	DM20,00
25	HSK	Golfschlaeger	set	DM1599,00

Wenn Sie in Ihrem Listenprogramm die Spalte *stueckpreis* mit einer USING-Anweisung formatieren, ändert sich die Ausgabe der Spalte wie folgt:

```
column 57, stueckpreis USING "DM###,###.##"
```

posten_nr	hersteller	beschreibung	liefereinheit	stueckpreis
1	HRO	SKI-Handschuh	Box	DM 250,00
1	HSK	SKI-Handschuh	Box	DM 800,00
1	SMT	SKI-Handschuh	Box	DM 450,00
4	HSK	Fussball	Col.	DM 960,00
4	HRO	Fussball	Col.	DM 480,00
.	.	.	.	.
9	ANZ	Volleyb. profi	solo	DM 20,00
25	HSK	Golfschlaeger	set	DM 1.599,00

## Umgebungsvariable INFORMIXDIR

Mit der Umgebungsvariable geben Sie an, in welchem Dateiverzeichnis sich die INFORMIX-Produkte befinden. Der Pfadname darf die Länge von 64 Zeichen nicht überschreiten.

Standardbelegung für INFORMIXDIR ist */usr/lib/informix* oder */opt/lib/informix*.

### Umgebungsvariable INFORMIXTERM

Mit der Umgebungsvariablen INFORMIXTERM geben Sie an, welche der Dateien *termcap* oder *terminfo* INFORMIX zur Bildschirmsteuerung verwendet. Sie können die von INFORMIX mitgelieferte *termcap*-Datei im Verzeichnis *\$INFORMIXDIR* benutzen, wobei dann die Belegung wie folgt lauten muß:

```
INFORMIXTERM=$INFORMIXDIR/termcap
```

Wenn Sie eine andere *termcap*- oder *terminfo*-Datei verwenden wollen, müssen Sie auf Betriebssystemebene die Umgebungsvariable **TERMCAP** bzw. **TERMINFO** belegen.

### Umgebungsvariable PSORT\_NPROCS (nur ONLINE)

Damit Sie mit *Psort* parallel sortieren können, müssen Sie die Umgebungsvariable *PSORT\_NPROCS* setzen. Mit *PSORT\_NPROCS* definieren Sie die Obergrenze der Prozesse, die gleichzeitig durch einen Suchlauf gestartet werden können. Wenn Sie *PSORT\_NPROCS* nicht belegt haben, können Sie nicht mit *Psort* arbeiten.

Wenn Sie *PSORT\_NPROCS* mit 0 besetzen, sucht sich *Psort* die benötigte Anzahl von Prozessen, soweit dies möglich ist.

Wenn Sie *PSORT\_NPROCS* mit einer Zahl größer 0 besetzen, steht diese Zahl für die maximale Anzahl verfügbarer Prozesse. ONLINE errechnet die Anzahl von Prozessen, die mit dieser Grenze möglich sind.

*Psort* können Sie optimieren, wenn Sie *PSORT\_NPROCS* mit dem Wert belegen, der der Zahl der System-Prozessoren entspricht.

Standardbelegung für *PSORT\_BPROCS* ist 3.

### Umgebungsvariable PSORT\_DBTEMP (nur ONLINE)

Mit der Umgebungsvariablen *PSORT\_DBTEMP* legen Sie fest, in welche Dateiverzeichnisse ONLINE die Zwischenergebnisse von *Psort* schreibt. ONLINE schreibt die Ergebnisse der Reihe nach in die aufgelisteten Dateiverzeichnisse. Um die höchste Performance zu erzielen, sollten Sie bei *PSORT\_DBTEMP* Dateiverzeichnisse festlegen, die auf unterschiedlichen Platten liegen. Idealerweise sollten auf den Platten keine weiteren temporären Dateien liegen.

Wenn Sie *PSORT\_DBTEMP* nicht gesetzt haben, benutzt ONLINE das Dateiverzeichnis, daß Sie mit der Umgebungsvariable *DBTEMP* bestimmt haben. Ist *DBTEMP* auch nicht belegt, benutzt ONLINE das Verzeichnis */tmp*.

Standardbelegung für *PSORT\_DBTEMP* ist */tmp*.



**Umgebungsvariable SACEISQL (nur ONLINE)**

Mit Hilfe der Umgebungsvariable *SACEISQL* können Sie bestimmen, unter welcher Isolationsstufe ein REPORT ablaufen soll. *SACEISQL* gilt nur für INFORMIX ONLINE und kann folgende Werte enthalten:

DIRTY READ  
COMMITTED READ  
CURSOR STABILITY  
REPEATABLE READ

Das *SACEGO*-Programm liest den Inhalt der Umgebungsvariablen zur Laufzeit und führt die Anweisungen dann unter der gewünschten Isolationsstufe durch. Ist *SACEISQL* nicht oder nur ungenügend definiert, arbeitet ONLINE unter der Standardeinstellung COMMITTED READ.

Standardbelegung für *SACEISQL* ist COMITTED READ.

## Dienstprogramme

Folgende Dienstprogramme, die auf Betriebssystemebene aufzurufen sind, stehen dem INFORMIX-Anwender bzw. INFORMIX-Datenbankverwalter zur Verfügung:

Dienstprogramm	Bedeutung	gültig für Backend
<i>bcheck</i>	Indexprüfprogramm	SE
<i>dbexport</i>	Datenbank exportieren	SE, ON 1)
<i>dbimport</i>	Datenbank importieren	SE, ON 1)
<i>dbload</i>	Daten aus Datei in Datenbank einlesen	SE, ON 1)
<i>dblog</i>	Transaktionsprotokoll-Dateien ausgeben	SE 1)
<i>dbschema</i>	Anweisungen für Datenbankaufbau erzeugen	SE, ON 1)
<i>tbcheck</i>	Indexprüfprogramm	ON 2)
<i>tbload</i>	Datenbank oder Tabelle binär laden	ON 2)
<i>tblog</i>	logische Protokolle ausgeben	ON 2)
<i>tbunload</i>	Datenbank oder Tabelle binär entladen	ON 2)
<i>tbparams</i>	logische oder physikalische Protokolle bearbeiten	ON 2)

SE = INFORMIX-SE

ON = INFORMIX-ONLINE

- 1) Die Beschreibung dieses Dienstprogramms finden Sie im Handbuch SQL-Nachschlagen[?].
- 2) Die Beschreibung dieses Dienstprogramms finden Sie im ONLINE-Handbuch [4].

## Funktionserweiterung der Schalter für bcheck

*bcheck* ist ein Dienstprogramm, das Indexdateien prüft oder auch repariert. Es kann nur für INFORMIX-SE-Datenbanken verwendet werden. Wenn Sie mit ONLINE arbeiten, benutzen Sie anstelle von *bcheck* das Dienstprogramm *tbcheck* (siehe INFORMIX-ONLINE-Handbuch [4]).

Das Programm *bcheck* vergleicht eine Indexdatei (*.idx*) mit einer Datendatei (*.dat*) und stellt fest, ob die beiden konsistent sind. Ist dies nicht der Fall, werden Sie von *bcheck* gefragt, ob Sie die fehlerhaften Indizes löschen und neu aufbauen wollen.

Sie können das Dienstprogramm *bcheck* sowohl bei Dateien mit Datensätzen fester Länge als auch bei Dateien mit Datensätzen variabler Länge benutzen. Die Syntax für die Verwendung von *bcheck* bei Datensätzen variabler Länge, wie hier gezeigt, stimmt mit derjenigen für Datensätze fester Länge überein. Nur die Option *-i* hat eine besondere Funktion bei Datensätzen variabler Länge.

Das Dienstprogramm *bcheck* repariert nicht den Teil der Indexdateien, in dem die Längen variabel sind.

```
bcheck - [ i | l | j | n | q | s ] filename
```

- i Nur die Indexdatei wird geprüft
- l Die Einträge in B+ Bäumen werden aufgelistet
- n Alle Fragen werden mit "nein" beantwortet
- j Alle Fragen werden mit "ja" beantwortet
- q Der Ausdruck der Programmüberschrift wird unterdrückt
- s Die Knotengröße der Indexdatei wird neu festgelegt. Mit dieser Option wird der Parameter NODESIZE auf den im Programmcode festgelegten Wert gesetzt. Diese Option ändert aber keine der Eigenschaften in den Indexschlüsseln.

Außer wenn Sie die Schalter *-n* oder *-j* benutzen, ist *bcheck* interaktiv und wartet auf eine Eingabe von Ihnen, wenn es einen Fehler gefunden hat.

Sie sollten den Schalter *-j* mit Vorsicht benutzen. Insbesondere sollten Sie *bcheck* nicht mit Schalter *-j* laufen lassen, wenn Sie die Dateien zum ersten Mal prüfen.

### Indizes überprüfen

Wenn Sie den Schalter *-i* bei Datensätzen fester Länge verwenden, werden die Indexinformationen in den Indexdateien auf Konsistenz mit den Datendateien geprüft.

Benutzen Sie den Schalter *-i* mit Datensätzen variabler Länge, wird der Gesamthalt der Indexdatei sowohl auf freien Platz als auch auf Konsistenz geprüft. Auch die Daten variabler Länge, die in der Indexdatei gespeichert sind, werden berücksichtigt. Das Dienstprogramm *bcheck* benutzt diese Informationen, falls die Indexdatei neu aufgebaut werden muß.

### Knoten- und Indexgrößen neu festlegen

Der *bcheck*-Schalter *-s* hat keine Wirkung auf Datensätze variabler Länge, die in den Indexdateien gespeichert sind. Mit *C-ISAM* können Sie die Größe der Schlüssel in einem Index neu festlegen, wenn Sie *iscluster* mit einer neuen *keydesc*-Struktur benutzen.

Wenn Sie mit *bcheck* eine Datei prüfen, die Felder unterschiedlicher Länge enthält, können zwei neue Meldung am Ende der Ausgabe erscheinen. Diese Meldungen informieren Sie über den Speicherplatz der variablen Felder, wie Sie im folgenden Beispiel erkennen können. Sie beantworten wieder alle Fragen mit "nein", indem Sie das Programm mit *-n* aufrufen:

```
bcheck -n kunde__100
```

```
BCHECK C-ISAM B-tree Pruefprogramm Version 4.10.U
Copyright (C) 1981-1990 Informix Software, Inc.
Software Serial Nummer RDS#B007653
```

```
C-ISAM Datei: kunde__100
```

```
Pruefen der Dateibesreibung und Datei-Groessen.
Knotengroesse der Index-Datei = 1024
Aktuelle Knotengroesse der C-ISAM Index-Datei = 1024
Pruefen der Datendatei Saetze
Pruefen der Indizes- und Schluesselbeschreibungen.
Index 1 = Schluessel Eindeutig: (0,4,2)
  1 benutzte Index-Knoten — 1 benutzte Index b-tree Ebenen
Pruefen der Freiplatzlisten fuer Datensaeetze und Index-Knoten
68 benutzte Index-Knoten, 3 frei — 96 benutzte Datensaeetze, 16 frei
64 benutzte Index-Seiten fuer Datensaeetze variabler Laenge im Speicher
15761 Bytes dieser Seiten sind frei, ein Durchschnitt von 246 Bytes pro seite
```

**Ressourcen aus nichtabrufbaren Dateien wiederherstellen**

Sofern Sie Datensätze variabler Länge benutzen und die Dateien gravierende Fehler enthalten, kann *bcheck* den beschädigten Indexteil der Dateien reparieren. Allerdings kann es keine beschädigten Datensätze reparieren. Da die Daten variabler Länge in den Indexdateien gespeichert werden, kann es vorkommen, daß Sie die Daten nicht abrufen können.

Um Indexdateien zu reparieren, die fehlerhafte Daten variabler Länge enthalten, müssen Sie die fehlerhaften Datensätze mit Ihrem eigenen *C-ISAM*-Programm löschen. Im Abschnitt *Dateiverwaltung bei Datensätzen variabler Länge* im Kapitel *C-ISAM* in diesem Handbuch finden Sie weitere Informationen über das Wiederherstellen von Daten aus fehlerhaften *.idx*-Dateien.

## Fehlermeldungen ausgeben (nur ONLINE)

Mit der Version 4.1 von INFORMIX-SQL werden auch Dateien mitausgeliefert, die den Text der Fehlermeldungen für alle INFORMIX-Produkte enthalten. Diese Dateien finden Sie im Dateiverzeichnis *\$INFORMIXDIR/msg*. Sie sind sowohl im ASCII- als auch im Postscript-Format vorhanden.

### ASCII-Dateien

Die ASCII-Dateien liegen im Dateiverzeichnis *\$INFORMIXDIR/msg/errmsg.txt* und können auf zwei Arten ausgegeben werden:

- Mit der Prozedur *finderr* können Sie einzelne oder eine Liste von Fehlermeldungen am Bildschirm ausgeben.
- Mit der Prozedur *rofferr* können Sie eine Fehlermeldungsdatei kopieren und in diese Kopie *troff*-Auszeichnungen einbringen.

Die Fehlermeldungen sind durchnummeriert von -1 bis -32100. Beim Aufruf der Prozeduren *finderr* oder *rofferr* können Sie das Minuszeichen weglassen.

### Fehlermeldungen am Bildschirm ausgeben

Die Prozedur *finderr* gibt einzelne Fehlermeldungen oder eine ganze Liste davon am Bildschirm aus. Sie können die Ausgabe der Fehlermeldungen auch in eine Datei oder auf einen Drucker umlenken. *finderr* rufen Sie folgendermaßen auf:

```
finderr meldungs_nr
```

```
meldungs_nr
```

ist die Nummer der Fehlermeldung, die Sie am Bildschirm ausgeben wollen. Sie können bis zu 16 Nummern angeben.

### Beispiele

Die folgenden Aufrufe von *finderr* geben beide die Fehlermeldung -359 am Bildschirm aus:

```
finderr -359
```

```
finderr 359
```

## Fehlermeldungen ausgeben

---

Der folgende Aufruf von *finderr* gibt mehrere Fehlermeldungen am Bildschirm aus:

```
finderr 233 107 113 134 143 14 154 | more
```

### Fehlermeldungen formatiert ausdrucken

Die Prozedur *rofferr* kopiert einen Teil der Fehlermeldungsdatei und formatiert diese Kopie mit *troff*-Auszeichnungen. Danach können Sie die Datei ausdrucken. *rofferr* rufen Sie folgendermaßen auf:

```
rofferr anfangs_nr [ende_nr]
```

*anfangs\_nr*

ist die Nummer der ersten Fehlermeldung, die aufbereitet und gedruckt werden soll. Diese Nummer müssen Sie angeben.

*ende\_nr*

ist die Nummer der letzten Fehlermeldung, die aufbereitet und gedruckt werden soll. Diese Angabe ist optional.

### Beispiele

Die folgenden Aufrufe von *rofferr* drucken beide die Fehlermeldung -359 aus:

```
rofferr -359 | nroff -man | lpr
```

```
rofferr 359 | nroff -man | lpr
```

Der folgende Aufruf von *rofferr* druckt die Fehlermeldungen von -1300 bis -4999 aus.

```
rofferr -1300 -4999 | nroff -man | lpr
```

## Postscript-Dateien

Die Postscript-Dateien liegen im Dateiverzeichnis  $\$INFORMIXDIR/msg$  und haben das Suffix *ps*.

Es handelt sich um sechs Dateien, die jeweils 50 Seiten Fehlermeldungstext enthalten. Zusätzlich können Sie noch eine Umschlag- und eine Titelseite ausdrucken. Im folgenden sind die Namen der Postscript-Dateien aufgelistet.

- cover.ps
- titlepage.ps
- errmsg1.ps
- errmsg2.ps
- errmsg3.ps
- errmsg4.ps
- errmsg5.ps
- errmsg6.ps



## Reservierte Wörter

Die folgende Liste enthält Wörter, die Schlüsselwörter für die INFORMIX-Produkte sind. Ab der Version 4.1 sind viele Schlüsselwörter nicht mehr reserviert. Aus der folgenden Liste können Sie erkennen, welche Wörter weiterhin reserviert sind (fett gedruckt), welche der ANSI-Standard als reserviert vorschreibt (mit Stern gekennzeichnet) und welche frei als Namen benutzt werden können. Werden die SQL-Anweisungen in eine Programmiersprache eingebettet, sind zusätzlich die reservierten Wörter der jeweiligen Programmiersprache zu beachten.

Bei der Benutzung von Schlüsselwörtern als Namen für Tabellen oder Spalten kann es zu ungewollten Ergebnissen oder Fehlern kommen, weil das System z.B. bei Funktionsnamen eher die Funktion ausführt als den Spalten- oder Tabellennamen zu interpretieren. Es ist daher empfehlenswert, Schlüsselwörter soweit wie möglich nicht als Namen zu verwenden.

Sollte es notwendig sein, Schlüsselwörter als Namen zu verwenden, so ist es oft sinnvoll, den Namen in der entsprechenden Anweisung voll zu qualifizieren und in der Form *tabellenname.spaltenname* anzugeben.

In dieser Version wird das Schlüsselwort AS eingeführt, mit dessen Hilfe eine Reihe Schlüsselwörter als Namen verwendet werden können. AS entspricht dem ANSI-Entwurf SQL2, nicht jedoch dem ANSI. Sie erhalten Warnungen, wenn Sie AS verwenden und die Umgebungsvariable DBANSIWARN ist gesetzt, bzw. wenn Sie den Schalter -ansi benutzen.

Sie müssen AS verwenden, wenn Sie folgende Namen als Spaltennamen verwenden wollen: AS FROM UNITS YEAR MONTH DAY HOUR MINUTE SECOND FRACTION

Ebenso müssen Sie AS verwenden, wenn Sie folgende Namen als Tabellennamen verwenden wollen: ORDER FOR GROUP HAVING INTO UNION WHERE CREATE GRANT WITH

Die Syntax für die Benutzung von AS ist folgendermaßen:

Für Spaltennamen:

spaltenname AS schlüsselwort FROM tabellenname

Für Tabellennamen:

SELECT spaltenauswahl FROM tabellennamen AS tabellen-alias

Beispiel:

Sie wollen Spalten, mit den Namen AS oder FROM in einer SELECT-Anweisung angeben: SELECT mycol AS as FROM mytab SELECT mycol AS from FROM mytab

Ihre Tabelle soll `order` heißen. Dann kann eine `SELECT`-Anweisung folgendermaßen aussehen: `SELECT * FROM mytab AS order`

Nicht jedes der im folgenden fettgedruckten Wörter ist in allen INFORMIX-Produkten reserviert. Aus Gründen der Portabilität sollten Sie jedoch die Reservierung in allen Programmen berücksichtigen, und z.B. 4GL-Schlüsselwörter nicht in ESQL/C-Programmen verwenden, obwohl diese im ESQL/C-Programm nicht zu einem Fehler führen.

## Reservierte Wörter

---

<b>abort</b>	byte
absolute	call
<b>accept</b>	<b>case</b>
<b>access</b>	char *
add	character *
<b>after</b>	check *
all *	<b>clear</b>
<b>allowing</b>	<b>clipped</b>
alter	close *
and *	cluster
ansi	<b>col</b>
any *	<b>color</b>
<b>array</b>	<b>colors</b>
as *	column
asc *	<b>columns</b>
<b>ascending</b>	<b>command</b>
<b>ascii</b>	<b>comment</b>
at	<b>comments</b>
<b>attribute</b>	commit *
<b>attributes</b>	committed
audit	<b>composites</b>
<b>auto</b>	<b>compress</b>
<b>autonext</b>	connect
authorization *	<b>constant</b>
<b>average</b>	constraint
avg *	<b>construct</b>
<b>background</b>	continue *
before	<b>convert</b>
begin *	count *
<b>beginning</b>	create *
<b>bell</b>	current *
between *	cursor *
<b>black</b>	<b>cyan</b>
<b>blanks</b>	database
<b>blink</b>	date
<b>blue</b>	datetime
<b>bold</b>	<b>date_type</b>
<b>border</b>	day
<b>bottom</b>	dba
buffered	<b>debug</b>
by *	dec *

---

<b>dec_t</b>	exec *
decimal *	execute
<b>decimal_type</b>	exists *
declare *	<b>exit</b>
<b>default</b>	<b>exitnow</b>
<b>defaults</b>	exits *
<b>defer</b>	explain
define	extend
delete *	extent
<b>delimiter</b>	<b>extern</b>
<b>delimiters</b>	<b>external</b>
desc *	<b>false</b>
<b>descending</b>	fetch *
describe	<b>field</b>
descriptor	file
<b>dim</b>	<b>finish</b>
dirty	first
<b>display</b>	<b>fixchar</b>
<b>displayonly</b>	float *
distinct *	flush
<b>do</b>	for *
<b>dominant</b>	<b>foreach</b>
double *	<b>form</b>
<b>down</b>	<b>format</b>
<b>downshift</b>	<b>formonly</b>
drop	found *
<b>dtime</b>	fraction
<b>dtime_t</b>	free
<b>eco-*</b>	from *
<b>editadd</b>	function
<b>editupdate</b>	<b>globals</b>
else	go *
end *	go to
end-exec	goto *
endif	grant *
<b>ending</b>	<b>green</b>
error	group *
escape *	having *
<b>every</b>	<b>header</b>
exclusive	<b>headings</b>

## Reservierte Wörter

---

<b>help</b>	<b>line</b>
hold	<b>lineno</b>
hour	<b>lines</b>
<b>identified</b>	<b>load</b>
<b>if</b>	<b>locate</b>
ifdef	locator
ifndef	lock
immediate	<b>loc_t</b>
in *	log
include	<b>long</b>
index	<b>long_float</b>
indicator *	<b>long_integer</b>
<b>infield</b>	<b>lookup</b>
<b>info</b>	<b>loop</b>
<b>initialize</b>	<b>magenta</b>
<b>input</b>	<b>main</b>
insert *	<b>margin</b>
<b>instructions</b>	<b>master</b>
int *	matches
integer *	max *
<b>interrupt</b>	mdy
<b>intersect</b>	<b>memory</b>
interval	<b>menu</b>
into *	<b>message</b>
<b>intrvl_t</b>	min *
<b>inverse</b>	<b>minus</b>
<b>invisible</b>	minute
is *	<b>mod</b>
<b>isam</b>	mode
isolation	modify
join	<b>module</b>
<b>joining</b>	money
key	month
<b>label</b>	<b>name</b>
last	<b>natural</b>
<b>left</b>	<b>need</b>
<b>len</b>	<b>new</b>
length	next
<b>let</b>	<b>nextfield</b>
like *	no

<b>nocr</b>	privileges *
<b>noentry</b>	program
<b>normal</b>	<b>prompt</b>
not *	public *
<b>not found</b>	put
<b>notfound</b>	<b>query</b>
<b>noupdate</b>	<b>queryclear</b>
<b>now</b>	<b>quit</b>
null *	raise
numeric *	<b>range</b>
of *	read
off	<b>readonly</b>
on *	real *
open *	<b>record</b>
option *	recover
<b>options</b>	<b>red</b>
or *	<b>register</b>
order *	relative
<b>otherwise</b>	<b>remove</b>
<b>out</b>	rename
outer	<b>repair</b>
<b>output</b>	repeatable
<b>package</b>	<b>report</b>
page	<b>required</b>
<b>pageno</b>	resource
<b>param</b>	<b>return</b>
<b>pause</b>	<b>returning</b>
<b>percent</b>	<b>reverse</b>
perform	revoke
<b>picture</b>	<b>right</b>
<b>pipe</b>	rollback *
<b>positive</b>	rollforward
power	row
precision *	rowid
prepare	<b>rows</b>
previous	<b>run</b>
<b>print</b>	<b>savepoint</b>
<b>printer</b>	schema *
prior	<b>screen</b>
<b>privilege</b>	scroll

## Reservierte Wörter

---

second	<b>status</b>
section *	<b>stdv</b>
select *	<b>step</b>
serial	stop
<b>serial_type</b>	<b>string</b>
set *	<b>struct</b>
share	<b>subtract</b>
<b>shift</b>	<b>subtype</b>
<b>short</b>	sum *
<b>short_float</b>	synonym
<b>short_integer</b>	<b>systables</b>
sitename	table *
size	<b>tables</b>
<b>skip</b>	temp
<b>sleep</b>	text
smallfloat	<b>then</b>
smallint *	<b>through</b>
some *	<b>thru</b>
<b>space</b>	<b>time</b>
<b>spaces</b>	<b>tiny_integer</b>
sql *	to *
<b>sql*</b>	today
<b>sqlca</b>	<b>top</b>
<b>sqlchar_type</b>	<b>total</b>
sqlcode *	<b>trailer</b>
<b>sqlda</b>	<b>trailing</b>
<b>sqldecimal_type</b>	<b>true</b>
<b>sqlerr</b>	<b>type</b>
sqlerror *	<b>typedef</b>
<b>sqlfloat_type</b>	undef
<b>sqlint_type</b>	<b>underline</b>
<b>sqlmoney_type</b>	union *
<b>sqlsmfloat_type</b>	unique *
<b>sqlsmint_type</b>	units
sqlwarning	<b>unload</b>
stability	unlock
start	<b>up</b>
<b>startlog</b>	update *
<b>static</b>	<b>upshift</b>
statistics	user *

using  
**validate**  
values \*  
varchar  
**variable**  
**vc\_t**  
**verify**  
view \*  
wait  
**waiting**  
**warning**  
weekday  
**when**  
whenever \*  
where \*  
**while**  
**white**  
**window**  
with \*  
**without**  
**wordwrap**  
work \*  
**wrap**  
year  
**yellow**  
**yes**  
**zerofill**





---

## Literatur

Die mit \* gekennzeichneten Titel sind nicht von Siemens herausgegeben.

- [1] Betriebssystem SINIX  
**INFORMIX-SQL V4.0**  
Datenbanksystem  
Kennenlernen
- [2] Betriebssystem SINIX  
**INFORMIX-SQL V4.0**  
Nachschlagen  
Benutzerhandbuch
- [3] Betriebssystem SINIX  
**INFORMIX V4.0**  
SQL-Sprachbeschreibung
- [4] Betriebssystem SINIX  
**Fehlermeldungen für  
INFORMIX-Produkte V4.1**
- [5] Betriebssystem SINIX  
**INFORMIX-ONLINE V4.1**  
Datenbank-Server  
Administrator-Handbuch
- [6] Betriebssystem SINIX  
**INFORMIX-NET/STAR V4.1**  
Netzkomponente für INFORMIX-SE  
Netzkomponente für INFORMIX-ONLINE
- [7] Betriebssystem SINIX  
**INFORMIX-ESQL/C V4.0**  
C-Schnittstelle für das  
Datenbanksystem INFORMIX
- [8] Betriebssystem SINIX  
**INFORMIX-ESQL/COBOL V4.0**  
COBOL-Schnittstelle für das  
Datenbanksystem INFORMIX

- [9] Betriebssystem SINIX  
**INFORMIX-4GL V4.1**  
Nachschlagen
- [10] Betriebssystem SINIX  
**INFORMIX-4GL V4.0**  
Kennenlernen
- [11] Betriebssystem SINIX  
**C-ISAM V4.0**  
Indexsequentielle Zugriffsmethode
- [12] Betriebssystem SINIX  
**Systemverwaltung**
- [13] Betriebssystem SINIX  
**Kommandos**
- [14] Betriebssystem SINIX  
**CES**  
C-Entwicklungssystem
- [15] Betriebssystem SINIX  
**Schnittstellen**
- [16] Betriebssystem SINIX  
**INFORMIX-NLS V4.1**  
Benutzerhandbuch

\* C.J. Date  
An Introduction to Database Systems  
Addison-Wesley  
1986

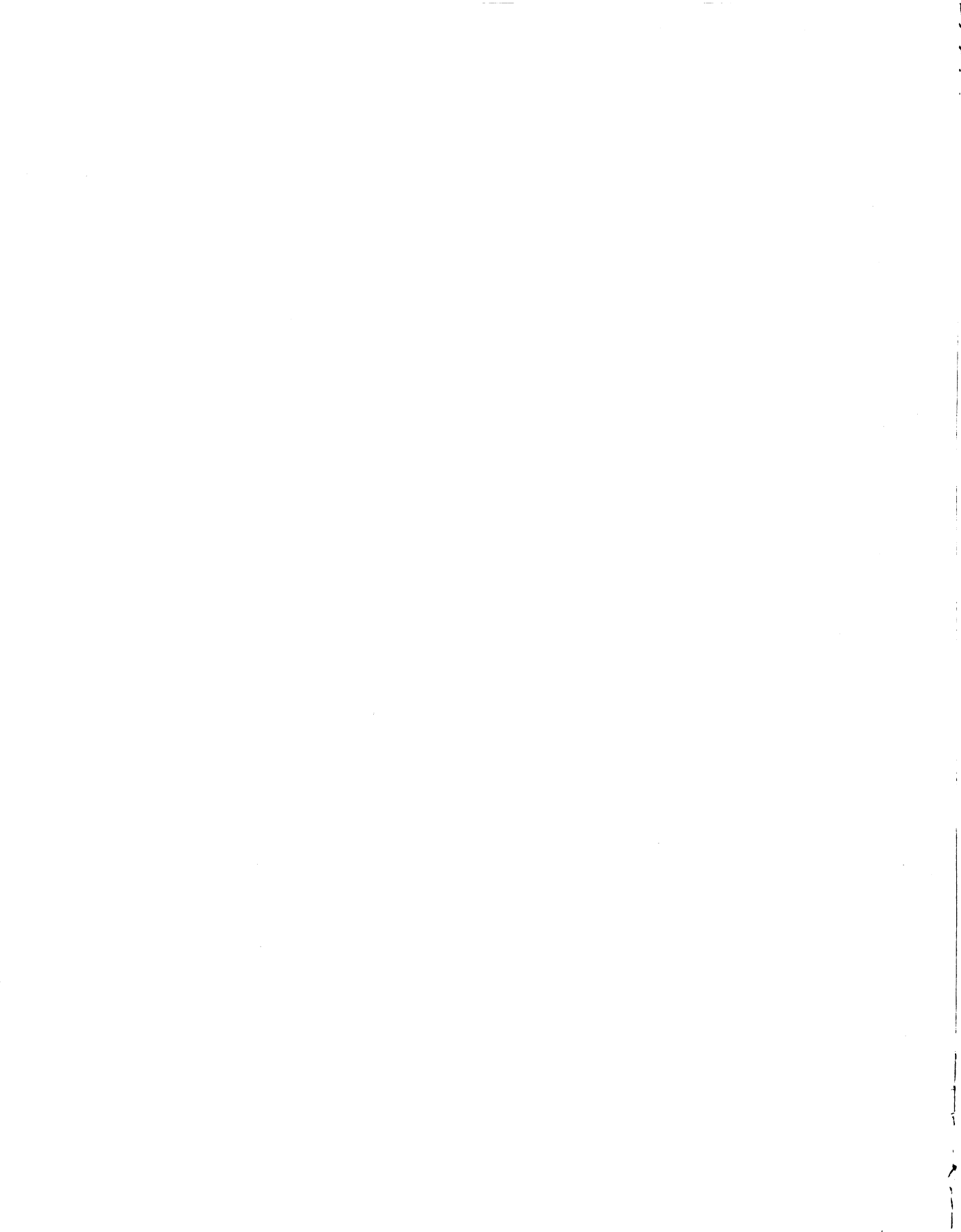
\* C.J. Date  
A Guide to The SQL Standard  
Addison-Wesley  
1990

Mit \* markierte Titel sind nicht von der Siemens Nixdorf Informationssysteme AG oder der Siemens AG herausgegeben.

### **Bestellen von Handbüchern**

Die aufgeführten Handbücher finden Sie mit ihren Bestellnummern im *Druckschriftenverzeichnis* der Siemens Nixdorf Informationssysteme AG. Dort ist auch der Bestellvorgang erklärt. Neu erschienene Titel finden Sie in den *Druckschriften-Neuerscheinungen*.

Beide Veröffentlichungen erhalten Sie regelmäßig, wenn Sie in den entsprechenden Verteiler aufgenommen sind. Wenden Sie sich bitte hierfür an eine Geschäftsstelle unseres Hauses.



---

## Stichwörter

### A

ACE, Seitenvorschub 25  
ASCII-Dateien, Fehlermeldungen 157  
Attribut, INVISIBLE 19

### B

bcheck, Dienstprogramm 154  
Betriebssystem, Dienstprogramme 153  
binär  
  entladen, tbunload 153  
  laden, tload 153  
BREAK, Debuggerkommando 139

### C

C-ISAM 81  
  Datendatei 83  
  Datensätze variabler Länge 81, 83  
  Datensicherheit 82  
  Fehlercodes 118  
  Indexdateien 83  
  isam.h 112  
  isstat1 118  
  isstat2 118  
  isstat3 118  
  isstat4 118  
  Kompatibilität 83  
  Umstellung auf neue Version 82  
CALL, Debuggerkommando 139  
CLEANUP, Debuggerkommando 140  
CLOSE-Anweisung 6  
Constraint 9

### D

- Datenbankanwendungen 4
- Datensätze variabler Länge
  - AUDIT-Protokoll 87
  - bcheck 88
  - bearbeiten 84
  - Dateiverwaltung 87
  - Datendatei 83
  - Datentyp 86
  - di\_idxsize 86
  - di\_nkeys 86
  - di\_nrecords 86
  - di\_recsz 86
  - erzeugen 83
  - Index definieren 85
  - Indexdateien 83
  - isbuild 84
  - iscluster 88
  - isreclen 84
  - ISVARLEN 84
  - Kompatibilität 83
  - Transaktionen 87
- DBFORMAT, neue Umgebungsvariable 148
- DELETE-Anweisung 6
- Dienstprogramme 153
  - bcheck 154
  - tbcheck 153
  - tbload 153
  - tblog 153
  - tbunload 153
- DUMP, Debuggerkommando 140

### E

- Eindeutigkeitsbedingung 9
- Erfolgskontrolle 6, 68

### F

- Fehlermeldungen ausgeben
  - am Bildschirm 157
  - am Drucker 158
  - Postscript 159
- Fehlermeldungen ausgeben (nur ONLINE) 157
- Fehlermeldungstexte, ASCII 157

**G**

Großbuchstaben 4

**I**

Indexprüfprogramm

tbcheck 153

bcheck 154

INFORMIXDIR, Umgebungsvariable 150

INFORMIXTERM, Umgebungsvariable 151

INSERT-Anweisung 6

INVISIBLE

Attribut 19

Werte anzeigen 20

ISADDINDEX 91

isfd 91

keydesc 91

isam.h 112

ISBUILD 93

ISAUTOLOCK 93

ISEXCLOCK 93

ISFIXLEN 94

ISINOUT 93

ISINPUT 93

ISMANULOCK 93

ISNOLOG 94

ISOUTPUT 93

ISTRANS 94

ISVARLEN 94

keydesc 93

mode 93

satzlaenge 93

ISINDEXINFO 96

bereich 96

di\_nkeys 97

di\_recszize 97

isfd 96

nummer 96

Isolationsstufe setzen, SACEISQL 152

ISOPEN 99

dateiname 99

ISAUTOLOCK 99

ISEXLLOCK 99

ISFIXLEN 99

ISINOUT 99



ISINPUT 99  
ISMANULOCK 99  
ISNOLOG 99  
ISOUTPUT 99  
ISTRANS 99  
ISVARLEN 99  
mode 99  
ISREAD 101  
ISCURR 101  
ISEQUAL 101  
isfd 101  
ISFIRST 101  
ISGREAT 101  
ISGTEQ 101  
ISLAST 101  
ISLCKW 102  
ISLOCK 102  
ISREAD 101  
ISSKIPLOCK 102  
ISWAIT 102  
satz 101  
ISREWCURR 105  
satz 105  
ISREWCURR,isfd 105  
ISREWREC 106  
isfd 106  
satz 106  
satznummer 106  
ISREWRITE 107  
satz 107  
ISREWRITE,isfd 107  
ISWCURR 109  
isfd 109  
satz 109  
ISWRITE 110  
isfd 110  
satz 110

**K**  
Kleinbuchstaben 4  
Kommando (Debugger)  
BREAK 139  
CALL 139  
CLEANUP 140

DUMP 140  
LET 141  
PRINT 141  
TRACE 142  
VARIABLE 143  
WHERE 143  
Kontrollblock  
  ON EVERY ROW 26  
  PAGE HEADER 26  
  Reihenfolge 26

**L**  
LET, Debuggerkommando 141  
LOAD-Anweisung 10  
logische Protokolle ausgeben, tblog 153

**N**  
Name 4

**O**  
ONLINE, paralleles Sortieren 28

**P**  
paralleles Sortieren, ONLINE 28  
Postscript-Dateien, Fehlermeldungstexte 159  
PRINT, Debuggerkommando 141  
Protokolle bearbeiten, tbparams 153  
PSORT\_DBTEMP, neue Umgebungsvariable 151  
PSORT\_NPROCS, neue Umgebungsvariable 151

**R**  
Reportanweisung, TOP OF PAGE 25  
Reservierte Wörter 5, 160

**S**  
SACEISQL, neue Umgebungsvariable 152  
SACEISQL, Isolationsstufe setzen 152  
Satzzeiger 6  
Schlüsselwörter 160  
Schlüsselwort 5  
SELECT/INTO TEMP 6  
SELECT/ORDER BY 5  
SELECT/UNION 5  
SQLCA-Satz 68  
SQLCA-Struktur 6  
sqlca-Struktur 74  
sqlca.h, Include-Datei 76

SQLCODE 6, 68  
sqlcode 75  
sqldetach, Bibliotheksfunktion 78  
SQLERRD 68, 74  
SQLNOTFOUND 75

### Suffix

.ace 28  
.arc 28  
.frm 22  
.per 22

### T

Tabellen, externe 8  
tbcheck, Dienstprogramm 153  
tblaod, Dienstprogramm 153  
tblog, Dienstprogramm 153  
tbparams, Dienstprogramm 153  
tbunload#, Dienstprogramm 153  
TOP OF PAGE, Reportanweisung 25  
TRACE, Debuggerkommando 142  
Transaktionssicherung 8  
Trennzeichen 10

### U

Umgebungsvariable 147  
  DBFORMAT 148  
  INFORMIXDIR 150  
  INFORMIXTERM 151  
  PSORT\_DBTEMP 151  
  PSORT\_NPROCS 151  
  SACEISQL 152  
Umgebungsvariablen (Debugger) 138  
UNLOAD-Anweisung 10  
UPDATE-Anweisung 6  
USER-Funktion 5

### V

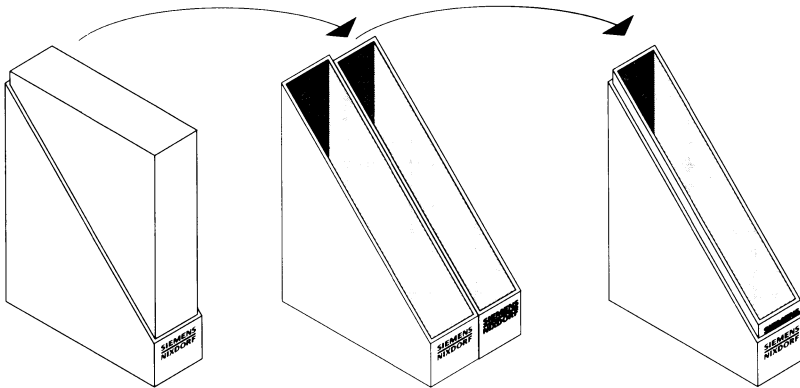
VARIABLE, Debuggerkommando 143

### W

WHERE, Debuggerkommando 143

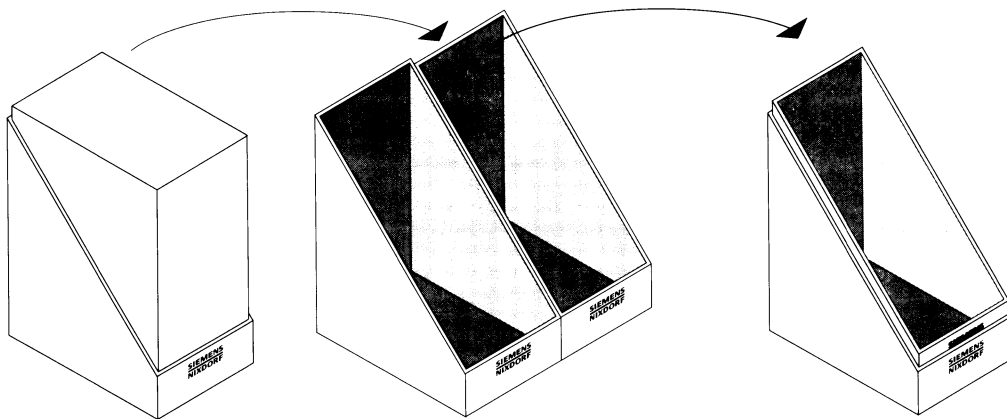
# Sammelboxen

Für Handbücher des vorliegenden Formates bieten wir zweiteilige Sammelboxen in zweierlei Größen an. Der Bestellvorgang entspricht dem für Handbücher.



Breite: ca. 5 cm

Bestellnummer: U3775-J-Z18-1



Breite: ca. 10 cm

Bestellnummer: U3776-J-Z18-1

960094 MC&D



9Y502875

Herausgegeben von/Published by  
Siemens Nixdorf Informationssysteme AG  
Postfach 21 60, W-4790 Paderborn  
Postfach 83 09 51, W-8000 München 83

Bestell-Nr./Order No. **U6202-J-Z145-1**  
Printed in the Federal Republic of Germany  
3200 AG 1926. (4000)