SIEMENS
NIXDORF

SINIX

INFORMIX-ESQL/C V4.0

C-Schnittstelle für das Datenbanksystem
INFORMIX

Beschreibung

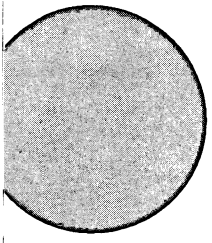
DOPPEL

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

Manualredaktion STM QM 2
Otto-Hahn-Ring 6
W-8000 München 83

Fax:
(089) 636-40443

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.

Manualredaktion STM QM 2
Otto-Hahn-Ring 6
W-8000 München 83

Fax:
(089) 636-40443

email im EUnet:
man@sieqm2.uucp

INFORMIX-ESQL/C (SINIX)

C-Schnittstelle für das Datenbanksystem INFORMIX

Beschreibung

Ausgabe Oktober 1990 (INFORMIX-ESQL/C V4.0)

... und Schulung?

Zu dem nachstehend beschriebenen Produkt, wie zu fast allen DV-Themen, bieten wir Kurse in unseren regionalen Training Centern an.

Zentrale Auskunft und Info-Material:

Telefon (089) 9275-3352
ab 2/91 6 36-4 89 99

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

Herausgegeben von
Siemens Nixdorf Informationssysteme AG

Vorwort

Dieses Handbuch beschreibt die Benutzung der Datenbanksprache SQL in C-Programmen.
Die Sprachelemente von C und SQL sind nicht das Thema dieses Handbuchs. Das Handbuch ist eine Schnittstellenbeschreibung und beschäftigt sich mit den Besonderheiten, die Sie beachten müssen, wenn Sie SQL-Anweisungen im C-Programm verwenden wollen.

An wen richtet sich das Handbuch?

Das Handbuch richtet sich an C-Programmierer, die mit dem Datenbanksystem INFORMIX und insbesondere mit SQL arbeiten.

Welche Vorkenntnisse benötigen Sie

Im Handbuch setzen wir voraus, daß Sie die Sprachen C und SQL kennen. Sie benötigen grundlegende Kenntnisse über relationale Datenbanken. Außerdem müssen Sie mit der Bedienung der INFORMIX-Produkte vertraut sein.

Was hat sich geändert?

Das Änderungsprotokoll auf den nachfolgenden Seiten gibt Ihnen einen Überblick über die fachlichen Änderungen gegenüber dem letzten Ausgabestand.

Eine Bitte an Sie

Schreiben Sie uns, wenn Sie mit dem Handbuch zurechtkommen. Wenn Sie zufrieden sind, sind wir auf dem richtigen Weg. Wenn Sie unzufrieden sind, teilen Sie uns bitte Ihre "Stolpersteine" mit, damit wir unsere Handbücher verbessern können.

Manualredaktion STM QM2
Otto-Hahn-Ring 6
W-8000 München 83

Änderungsprotokoll

Änderung des Vorgänger-Manuals
Stand Dezember 1987 (INFORMIX ESQL/C, Version V2.1)
durch die Neuausgabe vom
Oktober 1990 (INFORMIX ESQL/C, Version 4.0)

Im folgenden sind alle Neuerungen zu ESQL/C, Version 4.0 inhaltlich aufgelistet:

ESQL/C erfüllt den ANSI Level I SQL-Standard voll. Dem ANSI Level II SQL-Standard genügt ESQL/C teilweise.

ANSI Standard bedingte Änderungen:

Datentyp-Synonyme:	REAL	SMALLFLOAT
	DOUBLE PRECISION	FLOAT
	CHARACTER	CHAR
	NUMERIC	DECIMAL und DEC
	INTEGER	INT
weitere Synonyme:	UNIQUE	DISTINCT
NULL-Werte:	In einer GROUP BY-Klausel werden alle NULL-Werte als gleich betrachtet und bilden daher eine Gruppe.	
Kommentare	Sie können zwei Gedankenstriche (--) im ESQL/C-Programm angeben, um einen Kommentar zu schreiben.	
INDICATOR	Eine Indicatorvariable kann statt mit dem :-Zeichen mit dem Schlüsselwort INDICATOR an die Hostvariable gebunden werden.	
	Indikatorvariablen können jeden beliebigen Datentyp, außer DATETIME, INTERVAL und den Blobdatentypen haben.	

ANSI Datenbanken

INFORMIX-SE und INFORMIX-ONLINE unterstützen ANSI-Datenbanken.

Eine ANSI-Datenbank ist eine Datenbank mit folgenden Eigenschaften:

- Es ist automatisch Transaktionssicherung eingeschaltet. Die Transaktionssicherung kann nicht ausgeschaltet werden.
- Die voreingestellte Isolationsstufe bei INFORMIX-ONLINE ist Repeatable Read. Das führt dazu, daß sehr viele Sperren gesetzt werden, da auch für Leseoperationen Sperren benötigt werden.
- Fremde Datenbankobjekte müssen mit dem Namen des Eigentümers qualifiziert werden.
- Beim Erstellen einer Tabelle einer ANSI-Datenbank erhält nur der Eigentümer Zugriffsrechte auf die Tabelle. Alle Zugriffsrechte, auch für PUBLIC, müssen explizit mit GRANT vergeben werden.
- Bei Anweisungen, die den ANSI-Standard verletzen, werden Warnungen ausgegeben.

Sie können eine ANSI-Datenbank erstellen, indem Sie bei CREATE DATABASE die Klausel MODE ANSI angeben. Der Name der Datenbank enthält auch den owner, die Kennung des Benutzers, der die Datenbank erstellt.

Überprüfen auf ANSI-Kompatibilität

Sie können Ihr ESQL/C-Programm auf ANSI-Kompatibilität überprüfen, indem Sie die Umgebungsvariable DBANSIWARN benutzen oder beim Übersetzen die Option -ansi angeben. Außerdem zeigt splwarn in der sqlca-Struktur an, wenn Sie ein Sprachelement benutzen, das nicht dem Standard entspricht.

Neue Datentypen und neue reservierte Wörter

Es gibt für ESQL/C folgende neuen Datentypen:

DATETIME	Zeitpunkt
INTERVAL	Zeitspanne
BYTE	Blobdaten
TEXT	Blobdaten
VARCHAR	variable lange CHARACTER-Daten

UNIQUE CONSTRAINT ist ein neues reserviertes Wort. Es bewirkt, daß in einer Spalte oder in mehreren Spalten nur eindeutige Werte stehen dürfen.

Neuerungen zu Hostvariablen

Hostvariablen dürfen Strukturen sein. Hostvariablen können Arrays beliebigen Datentyps sein. Hostvariablen dürfen wie normale C-Variablen initialisiert werden. Typedef-Vereinbarungen gelten auch für Hostvariablen.

Neues Sortierverfahren und neuer Optimierer

Ein neues Sortierverfahren beschleunigt die Erzeugung von Indizes und die Durchführung von SELECTs mit ORDER BY- oder UNIQUE-Klauseln.

Ein neuer Optimierer beschleunigt die Abfrage langer Tabellen. Die neue Anweisung SET EXPLAIN ON schaltet die Ausgabe der Suchstrategie in eine Datei ein.

Neue Dienstprogramme

Die Dienstprogramme dbimport und dbexport importieren bzw. exportieren die Datenbank.

Neuer Präprozessor

Ein neuer Präprozessor ermöglicht bedingtes Übersetzen mit den Anweisungen: $include, $define, $undef, $ifdef, $ifndef, $else, $endif.

Nichtprotokollierte temporäre Tabellen

Mit der Angabe WITH NO LOG in der CREATE- oder SELECT-Anweisung können Sie die Transaktionssicherung für die temporäre Tabelle ausschalten.

Dynamisch formulierte SQL-Anweisungen

Mit der Anweisung PREPARE können Sie jetzt eine oder mehrere SQL-Anweisungen vorbereiten (multiple PREPARE). Mit der Anweisung FREE können Sie Speicherplatz freigeben, der für eine PREPARE-Anweisung angelegt wurde. Mit der Anweisung EXECUTE IMMEDIATE können Sie die Funktionen der Anweisungen PREPARE, EXECUTE und FREE in einem Schritt ausführen.

Neue Bibliotheksfunktionen

- Numerische Formatierfunktionen:
  rfmtlong, rfmtdouble, rfmtdec
- rgetmsg
- Rundungsfunktionen:
  decround, dectrunc
- SQL Datenbankserver-Kontrollfunktionen:
  sqlstart, sqlexit, sqlbreak

Groß- und Kleinschreibung

ESQL/C unterscheidet Klein- und Großschreibung.

Inhalt

1 Einführung

Dieses Kapitel gibt einen Überblick über die INFORMIX-Produkte. Es beschreibt, was ESQL/C ist und erläutert, welche metasprachlichen Darstellungsmittel in diesem Handbuch verwendet werden.

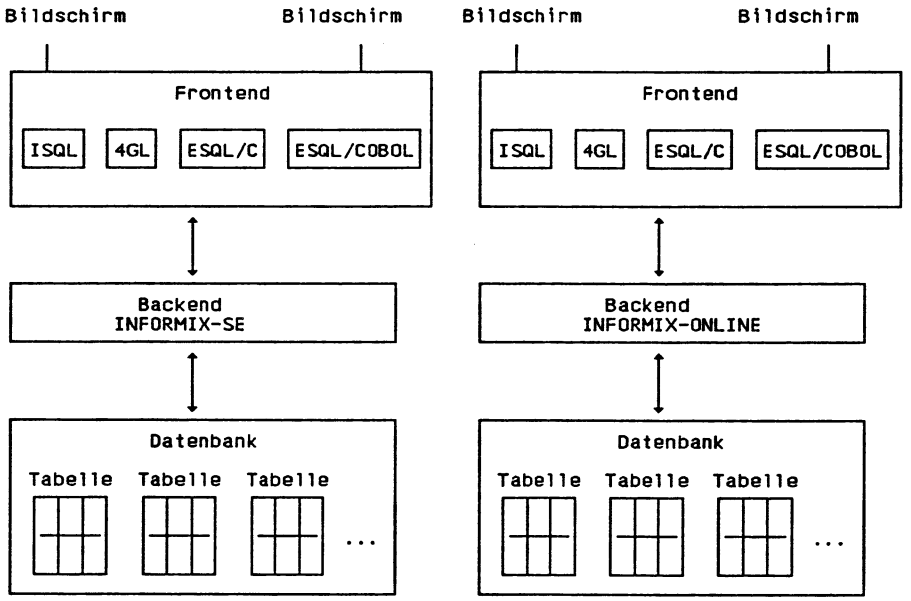
1.1 INFORMIX im Überblick

INFORMIX ist ein relationales Datenbanksystem. Ein Datenbanksystem dient dazu, den Benutzer bei der Organisation, Verwaltung und Manipulation großer Datenbestände zu unterstützen. Es übernimmt die Abspeicherung und Bereitstellung der Daten und erledigt Verwaltungsaufgaben. Dazu gehören insbesondere Integritätskontrollen und der Schutz der Daten vor unerlaubten Zugriffen.

Die Besonderheit eines relationalen Datenbanksystems besteht darin, daß der Benutzer die in der Datenbank gespeicherten Daten ausschließlich in Form von Tabellen sieht.

Ein INFORMIX-Datenbanksystem besteht schematisch aus folgenden drei Ebenen:

- Frontend
- Backend
- Datenbank.



1.1.1 Frontend

Das Frontend stellt die Benutzerschnittstellen bereit. INFORMIX stellt unterschiedliche Benutzerschnittstellen zur Verfügung. Es gibt Schnittstellen für den interaktiven Betrieb und die Programmeinbettung für die Programmiersprachen INFORMIX-4GL, ESQL/C und ESQL/COBOL.

1.1.2 Backend

Das Backend empfängt über ein Frontend die Anweisungen zur Bearbeitung der Datenbank. Es führt die Anweisungen aus und gibt die Ergebnisse an das jeweilige Frontend zurück, wo sie dem Benutzer zur Verfügung gestellt werden.

Das Backend liefert also die Dienste, die zur Ausführung der Datenbankoperationen erforderlich sind. Dazu gehören zum Beispiel die Bereitstellung einer Ablaufumgebung und des benötigten Speicherplatz, sowie Abspeichern und Bereitstellen der Daten.

Bei INFORMIX gibt es zwei unterschiedliche Backends:

- INFORMIX-SE (Standard Engine)
- INFORMIX-ONLINE.

INFORMIX-SE

INFORMIX-SE organisiert die Datenbank über das SINIX-Dateisystem. Es basiert auf C-ISAM, einer Bibliothek von C-Funktionen. Mit diesen Funktionen werden Dateien erzeugt und verwaltet, in denen die Daten der Datenbank abgespeichert sind.

INFORMIX-ONLINE

Das INFORMIX-ONLINE hat eine eigene Plattenverwaltung für seine Datenbanken, unabhängig vom SINIX-Dateisystem. Das hat den Vorteil, daß die Datenhaltung und -verwaltung auf die speziellen Anforderungen eines Datenbanksystems zugeschnitten und daher effizienter und sicherer sind.

Darüberhinaus bietet INFORMIX-ONLINE Möglichkeiten, die bei INFORMIX-SE nicht enthalten sind. Dazu gehören:

- Bessere Möglichkeiten für die Gewährleistung der Datenintegrität:
  Bei INFORMIX-ONLINE gibt es Isolationsstufen, die festlegen, ob beim Lesen fremde Sperren berücksichtigt und eigene gesetzt werden. Die Isolationsstufe ist einstellbar, so daß jeder Benutzer den Sicherheitsgrad bestimmen kann, der für seine Anweisungen erforderlich ist. Ausführliche Informationen dazu finden Sie im SQL-Handbuch [2], Kapitel2.
- Bessere Möglichkeiten für die Datensicherung:
  Einen Überblick über die einzelnen Möglichkeiten finden Sie im SQL-Handbuch [2], Kapitel 2, Abschnitt 2.12. Die ausführliche Beschreibung der Datensicherungsmethoden ist im ONLINE-Handbuch [6] beschrieben.
- Datentyp VARCHAR:
  Er ermöglicht es, Spalten zu definieren, die Zeichenketten unterschiedlicher Länge enthalten können. Die ausführliche Beschreibung finden Sie im SQL-Handbuch [2] Kapitel 4, Abschnitt 4.2.2.
- BLOB (**B**inary **L**arge **O**bjects)-Datentypen BYTE und TEXT:
  Sie ermöglichen es, Spalten zu definieren, die Byte-Folgen beliebiger Länge enthalten können. Die ausführliche Beschreibung finden Sie im SQL-Handbuch [2], Kapitel 4, Abschnitt 4.2.5.

- Zugriff auf externe Tabellen:
  Sie haben die Möglichkeit, auf Tabellen einer Datenbank zuzugreifen, die nicht die aktuelle Datenbank ist. Einzelheiten hierzu finden Sie im SQL-Handbuch [2], Kapitel 2, Abschnitt 2.1 und Abschnitt 2.2.
- Mehrere INFORMIX-ONLINE-Systeme auf demselben Rechner

Verträglichkeit von INFORMIX-SE und INFORMIX-ONLINE

Die beiden Backends können gleichzeitig auf demselben Rechner installiert sein.

Unter Berücksichtigung der genannten Unterschiede können Anwendungen mit jedem der beiden Backends laufen.

Datenbanken, die mit INFORMIX-SE erstellt wurden, müssen konvertiert werden, wenn sie mit INFORMIX-ONLINE verwendet werden sollen, und umgekehrt.
INFORMIX stellt entsprechende Dienstprogramme zur Konvertierung bereit.

INFORMIX-Netzprodukte

INFORMIX stellt für die Verwendung entfernter Datenbanken folgende Produkte zur Verfügung:

- INFORMIX-NET für INFORMIX-SE
- INFORMIX-STAR für INFORMIX-ONLINE.

Frontend und Backend können auf verschiedenen Rechnern laufen.

Außerdem haben Sie die Möglichkeit, eine entfernte Datenbank zu erstellen und beim Wechsel der aktuellen Datenbank eine entfernte Datenbank anzugeben.

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

1.1.3 Datenbank

In der Datenbank sind die Benutzer- und Verwaltungsdaten gespeichert. Die Einzelheiten, die Sie für die Verwendung von SQL über INFORMIX-Datenbanken wissen müssen, sind im SQL-Handbuch [2], Kapitel 2 beschrieben.

1.2 Was ist ESQL/C?

INFORMIX-ESQL/C (embedded SQL und Tools für C) ist ein Softwareprodukt, mit dem Sie die Datenbanksprache SQL in C-Programmen verwenden können.

ESQL/C ist für den C-Programmierer entwickelt worden, der Datenbankanwendungen in C realisieren will und für den fortgeschrittenen INFORMIX-Benutzer, der C-Funktionen im PERFORM und ACE benutzen will.

ESQL/C bietet C-Bibliotheksfunktionen und Include-Dateien, sowie einen Präprozessor, der für die Vorübersetzung des Quellprogramms zuständig ist.

Der ESQL/C-Präprozessor interpretiert die Hostvariablen-Vereinbarungen und die SQL-Anweisungen im ESQL/C-Quellprogramm. In Include-Dateien vordefinierte Datentypen unterstützen die SQL-Datentypen. Bibliotheksfunktionen ermöglichen Datentypkonversionen und weitere Dienste. Ein Anschluß an die INFORMIX-Produkte ACE und PERFORM erlaubt die Ausnutzung der Funktionen des Listengenerators und des Formatgenerators im ESQL/C-Programm.

ESQL/C können Sie mit beiden INFORMIX-Backends, INFORMIX-SE und INFORMIX-ONLINE betreiben. Das Handbuch berücksichtigt beide Backends.

Dieses Handbuch ist eine reine Schnittstellenbeschreibung.

Eine Beschreibung der Sprache C finden Sie z.B. bei Kerninhan/Ritchie, Programmieren in C [1].
Die Sprache INFORMIX SQL ist im zentralen SQL-Handbuch beschrieben [2]. Dieses Handbuch ist unbedingt erforderlich, um den vollen Funktionsumfang von ESQL/C zu erfassen.
Die Produkte ACE und PERFORM finden Sie in dem Handbuch INFORMIX-Nachschlagen [3].
Eine Einführung in relationale Datenbanken und INFORMIX beschreibt das Handbuch INFORMIX-Kennenlernen [4].
Die den Datenbanktabellen zugrundeliegende Datei- und Indexstruktur ist C-ISAM. Die Informationen über Index-sequentielle Zugriffsmethoden finden Sie im C-ISAM-Handbuch [5].
Informationen zum INFORMIX-ONLINE Backend finden Sie im ONLINE Administrator Handbuch [6].
Alle hier aufgeführten Titel sind im Literaturverzeichnis ausführlich aufgelistet.

1.3 Darstellungsmittel

In diesem Handbuch verwenden wir folgende Darstellungsmittel:

———	Syntaxdefinition
GROSS	SQL-Schlüsselwörter in Syntaxdefinitionen und im Fließtext
fett	In Syntaxdefinitionen konstante Teile, die keine SQL-Schlüsselwörter sind. Hervorhebungen im Fließtext.
[ ]	Optionale Angaben Die eckigen Klammern geben Sie nicht an.
{ | }	Alternative Angaben in Syntaxdefinitionen Die geschweiften Klammern geben Sie nicht an.
{ }	Zusammenfassung von Klauseln in Syntaxdefinitionen, die gemeinsam wiederholt werden können. Die geschweiften Klammern geben Sie nicht an.
...	In Syntaxdefinitionen bedeuten die Punkte, daß die vorausgehenden Angabe beliebig oft wiederholt werden kann. In Beispielen bedeuten die Punkte, daß die restlichen Teile für das Beispiel ohne Bedeutung sind. Die Punkte werden nicht angegeben.
,...	In Syntaxdefinitionen bedeutet diese Schreibweise, daß die vorausgehenden Angabe beliebig oft, durch Komma getrennt, wiederholt werden kann. Wird keine Wiederholung angegeben, fällt auch das Komma weg.

Verweise

Verweise werden wie folgt angegeben.

- Verweis auf einen Abschnitt in demselben Kapitel:
  siehe Abschnitt Abschnittsnr.
- Verweis auf einen Abschnitt in einem anderen Kapitel:
  siehe Kapitel Kapitelnummer, Abschnitt.
- Verweis auf ein anderes Handbuch:
  siehe Handbuchtitel [Literaturlistennummer] Bei der zentralen SQL-Beschreibung wird die Nummer nicht erwähnt.

2 SQL in ESQL/C

In diesem Kapitel erfahren Sie, wie Sie die Sprache SQL in Ihr ESQL/C-Quellprogramm einbauen können.

Die Sprache SQL ist im zentralen SQL-Handbuch beschrieben.

In diesem Kapitel werden keine Sprachelemente von SQL beschrieben, sondern nur die Besonderheiten, die Sie im ESQL/C-Quellprogramm berücksichtigen müssen.

Als erstes erläutern wir, welche formalen Schreibregeln zu beachten sind. Dann folgen einzelne Abschnitte über SQL-Themen, zu denen Sie Informationen brauchen, um ein richtiges ESQL/C-Quellprogramm formulieren zu können. Es werden nicht alle Themen aus den Konzepten in der SQL-Beschreibung behandelt. Alle Themen, die Sie hier nicht finden, sind in der SQL-Beschreibung auch für ESQL/C-Belange vollständig abgehandelt.

2.1 Schreibregeln für SQL-Anweisungen

Die SQL-Anweisungen beginnen im ESQL/C-Quellprogramm entweder mit EXEC SQL - das entspricht dem ANSI-Standard, oder mit dem $-Zeichen. Der Strichpunkt ; schließt die SQL-Anweisung ab.

Hostvariable, d.h. C-Variablen, die Sie in SQL-Anweisungen benutzen wollen, müssen in den SQL-Anweisungen mit dem :-Zeichen oder dem $-Zeichen geschrieben werden. : entspricht dem ANSI-Standard. Die Vereinbarung der Hostvariablen ist im Abschnitt 2.3 beschrieben.

```
EXEC SQL select kname into :kname from kunde;
$select kname into $kname from kunde;
```

Syntaxzeilen und Beispiele in diesem Manual sind in der Regel mit dem $-Zeichen geschrieben. Sie können anstelle des $-Zeichens immer EXEC SQL bzw. : angeben, wenn Ihr Programm dem ANSI-Standard entsprechen muß.

Der ESQL/C-Präprozessor unterscheidet Groß- und Kleinschreibung.

Kommentare im ESQL/C-Programm

Es gibt zwei Möglichkeiten, Kommentare in ESQL/C-Programme zu schreiben:

- Sie schließen, wie in C-Programmen, den Kommentar zwischen /* und */ ein.
- Sie schreiben vor den Kommentar zwei Spiegelstriche (--).

2.2 SQL-Anweisungen

Alle SQL-Anweisungen bis auf die WHENEVER-Anweisung sind im zentralen SQL-Handbuch [2] beschrieben. Aus der Beschreibung ist ersichtlich, für welche Datenbankserver und in welchem INFORMIX-Produkt jede Anweisung verwendet werden kann.

Im folgenden finden Sie eine Übersicht über alle SQL-Anweisungen. Sie ersehen daraus, welche Anweisungen nur im Dialog formuliert werden können, in einem ESQL/C-Programm also nicht verwendet werden können und welche Anweisungen nur eingebettet verwendet werden können, also speziell für die Verwendung im ESQL/C-Programm gedacht sind.

Die WHENEVER-Anweisung finden Sie im Abschnitt 2.7, Erfolgskontolle in diesem Kapitel beschrieben.

Übersicht über alle SQL-Anweisungen

Interaktiv und eingebettet

SQL-Anweisung	Bedeutung
ALTER INDEX	Indexdefinition ändern
ALTER TABLE	Tabellendefinition ändern
BEGIN WORK	Transaktion starten
CLOSE DATABASE	Datenbank schließen
COMMIT WORK	Transaktion beenden
CREATE AUDIT	Audit-Protokoll erzeugen
CREATE DATABASE	Datenbank erzeugen
CREATE INDEX	Index erzeugen
CREATE SYNONYM	Synonym definieren
CREATE TABLE	Tabelle erzeugen
CREATE VIEW	View erzeugen
DATABASE	Datenbank eröffnen
DELETE	Sätze löschen
DROP AUDIT	Audit-Protokoll löschen
DROP DATABASE	Datenbank löschen
DROP INDEX	Index löschen
DROP SYNONYM	Synonym löschen
DROP TABLE	Tabelle löschen
DROP VIEW	View löschen
GRANT	Zugriffsrechte vergeben
INSERT	Sätze einfügen
LOCK TABLE	Tabelle sperren
RECOVER TABLE	Tabelle aktualisieren
RENAME COLUMN	Spaltennamen ändern
RENAME TABLE	Tabellennamen ändern
REVOKE	Zugriffsrechte entziehen
ROLLBACK WORK	Transaktion rücksetzen
ROLLFORWARD DATABASE	Datenbanksicherung aktualisieren
SELECT	Daten abfragen
SET EXPLAIN	Suchstrategie ausgeben
SET ISOLATION	Isolationsstufe wählen
SET LOCK MODE	Warte-Modus definieren
SET LOG	Protokollierungsmodus ändern
START DATABASE	Transaktionssicherung einstellen
UNLOCK TABLE	Tabellensperre aufheben
UPDATE	Spaltenwerte ändern
UPDATE STATISTICS	Satzanzahl aktualisieren

Interaktiv

SQL-Anweisung	Bedeutung
CHECK TABLE	Tabelle prüfen
CREATE SCHEMA	Datenbankschema erzeugen
INFO	Tabellen-Information ausgeben
LOAD	Sätze aus Datei einfügen
OUTPUT	Sätze in Datei ausgeben
REPAIR TABLE	Index wiederherstellen
UNLOAD	Sätze in Datei ausgeben

Eingebettet

SQL-Anweisung	Bedeutung
CLOSE	Satzzeiger schließen
DECLARE	Satzzeiger vereinbaren
DESCRIBE	Anweisung analysieren
EXECUTE	dynamische Anweisung ausführen
EXECUTE IMMEDIATE (nur bei C)	dynamische Anweisung sofort ausführen
FETCH	Satzzeiger positionieren
FLUSH	Einfügepuffer leeren
FREE	Betriebsmittel freigeben
LOAD (nur bei 4GL)	Sätze aus Datei einfügen
OPEN	Satzzeiger öffnen
PREPARE	dynamische Anweisung vorbereiten
PUT	Satz in Einfügepuffer schreiben
UNLOAD (nur bei 4GL)	Sätze in Datei ausgeben

2.3 Hostvariablen

Die Hostvariablen sind C-Variablen, die Sie in SQL-Anweisungen benutzen können. Hostvariablen nehmen Werte aus der Datenbank an; sie werden in SQL- bzw. C-Anweisungen verarbeitet und dienen dazu, Werte aus dem ESQL/C-Programm in die Datenbank zurückzuschreiben.

Vereinbarung der Hostvariablen

Eine Hostvariable wird folgendermaßen vereinbart:

```
 $datentyp hvar_name;

oder

EXEC SQL BEGIN DECLARE SECTION
 datentyp hvar_name;
.
.
.
EXEC SQL END DECLARE SECTION
```

Die zweite Möglichkeit entspricht dem ANSI-Standard.

Hostvariablen müssen nach den Regeln der Programmiersprache C vereinbart werden. Sie haben einen C-Datentyp, der dem SQL-Datentyp entsprechen muß. Gibt es keinen zum SQL-Datentyp passenden C-Datentyp, so stellt ESQL/C vordefinierte Datentypen bereit, die Sie für die Hostvariable verwenden müssen.

Die folgende Übersicht zeigt, wie sich die Datentypen entsprechen und wann Sie ggf. den vordefinierten Datentyp verwenden müssen. Die C-Datentypen sind zur Unterscheidung von den SQL-Datentypen kleingeschrieben.

SQL-Datentyp	C-Datentyp	vordefinierter Datentyp
CHAR(n)	char array[n + 1];	string array[n + 1];
	char *;	fixchar array[n];
SMALLINT	short int;	
INTEGER	long int;	
SERIAL	long int;	
SMALLFLOAT	float;	
FLOAT	double;	
DATE	long int; char [];	
MONEY		dec_t oder struct decimal;
DECIMAL		dec_t oder struct decimal;
DATETIME		dtime_t oder struct dtime;
INTERVAL		intervl_t oder struct intrvl;
nur unter ONLINE:		
VARCHAR		varchar
BYTE		loc_t
TEXT		loc_t

Im folgenden sind unter den einzelnen Datentypen die Besonderheiten bei Vereinbarungen der Hostvariablen beschrieben, wobei besonders die vordefinierten Datentypen erklärt werden. Zu jedem Datentyp sind die zugehörigen Bibliotheksfunktionen aufgelistet mit dem Verweis, wo sie beschrieben sind.
Zu folgenden Datentypen gibt es keine weiteren Erläuterungen, da die Entsprechungen aus der obenstehenden Tabelle eindeutig sind:

SMALLINT	short int;
INTEGER	long int;
SERIAL	long int;
SMALLFLOAT	float;
FLOAT	double;

Der Datentyp MONEY entspricht dem Datentyp DECIMAL.
ESQL/C unterstützt die typdef-Vereinbarungen der Sprache C. Sie können diese Vereinbarung für Hostvariablen benutzen.

ESQL/C gibt eine Warnung aus, wenn Sie eine Hostvariable in zwei unterschiedlichen Programmeinheiten zweimal vereinbaren. Handelt es sich in beiden Fällen um lokale Variablen, können Sie die Warnung ignorieren. Handelt es sich um eine oder zwei globale Vereinbarungen, kann es zu Fehlern im Programm kommen. Die Warnung zeigt an, daß im folgenden Programm nur die zweite Vereinbarung gültig ist. Strukturen werden als doppelt vereinbart erkannt, wenn alle Strukturelemente in Typ und Länge übereinstimmen.

Damit alle Hostvariablen in einer Programmeinheit garantiert lokal vereinbart sind, können Sie statt der einfachen umschließenden Klammern das Symbolpaar ${ und $} angeben.

Der ANSI-Standard unterstützt die Symbolpaare nicht.

Initialisierung von Hostvariablen

Hostvariablen können wie C-Variablen bei der Vereinbarung initialisiert werden. Bei der Initialisierung mit Zeichenketten ist darauf zu achten, daß die Zeichenkette keine reservierten Wörter und nicht das ;-Zeichen enthält.
Der ESQL/C-Präprozessor prüft die Initialisierung nicht; der C-Compiler meldet erst ggfs. Fehler.

Bezug auf Hostvariablen

Verwenden Sie die Hostvariable in einer SQL-Anweisung, müssen Sie dem Variablennamen ein $-Zeichen oder ein :-Zeichen voranstellen. : entspricht dem ANSI-Standard.

Verwenden Sie die Hostvariable in einer C-Anweisung, so entfällt das Präfix.

Strukturen können in SQL-Anweisungen als Ganzes oder mit ihren Elementen benutzt werden. Führen Sie den Strukturnamen auf, so wird intern die Liste der Strukturkomponenten benutzt. Strukturen können geschachtelt sein.

ESQL/C erlaubt die Verwendung von Vektoren. In SQL-Anweisungen können Sie die Elemente der Vektoren aufführen. Außer bei Vektoren vom Datentyp CHAR, ist es nicht möglich den Vektornamen allein zu benutzen. Sie müssen immer gültige Indizes mitangeben.

Automatische Datentypkonvertierung

Ergeben sich Unterschiede zwischen dem Datentyp einer SQL-Spalte und einer Hostvariablen oder zwischen den Datentypen zweier Spalten (z.B. bei einem Vergleich), so versucht ESQL/C eine entsprechende Konvertierung.

Dies geschieht z.B. bei der Umwandlung von CHAR in einen numerischen Typ; dabei müssen die CHAR-Spalten mit Zahlen belegt sein. Wird ein CHAR-Wert mit einem numerischen Wert verglichen, so wird in ESQL/C der CHAR-Wert zuerst in einen numerischen Wert umgewandelt. Die Umwandlung eines numerischen Wertes in einen CHAR-Wert geschieht über einen String. Für sehr große oder sehr kleine Zahlenwerte wird in SQL die exponentielle Schreibweise verwendet.

Ist eine Umwandlung nicht möglich, z.B. weil sich daraus kein Sinn ergibt oder weil die Ergebnisvariable zu klein ist, um den umgewandelten Wert aufzunehmen, so liefert ESQL/C Ergebnisse, die der unten stehenden Tabelle entnommen werden können. Dabei steht N für einen numerischen Typ und C für einen Zeichen-Typ.

Umwandlung	Problem	Auswirkungen
C -> C	paßt nicht, weil die zugewiesene Zeichenkette zu lang ist.	Zeichenkette wird gekürzt. sqlca.sqlwarn.sqlwarn1 wird auf W gesetzt. Die Indikator-variable liefert die Länge der Originalzeichenkette.
N -> C	paßt nicht, weil die Zahl zu viele Ziffern hat	Zeichenkette wird mit Sternen (*) belegt. sqlca.sqlwarn.sqlwarn1 wird auf W gesetzt; der Indikator-variable wird ein positiver Integer-Wert zugewiesen.
C -> N	keine Zahl	Zahl ist undefiniert. sqlca.sqlcode ist negativ.
C -> N	Überlauf	Zahl ist undefiniert; sqlca.sqlcode ist negativ.
N -> N	Überlauf	Zahl ist undefiniert; sqlca.sqlcode ist negativ.

In ESQL/C haben Sie die Möglichkeit, diese Umwandlungen auch explizit vorzunehmen. Standardumwandlungen wie z.B. vom Typ int in den Typ FLOAT wird entweder der C-Compiler übernehmen oder Sie selbst formulieren eine entsprechende Cast-Operation.

Für kompliziertere Umwandlungen stehen Ihnen eine Reihe von Bibliotheksfunktionen zur Verfügung. Welche Bibliotheksfunktionen es in ESQL/C für die einzelnen Datentypen gibt, steht unter der Beschreibung zum Datentyp. Die Funktionen sind im Kapitel Bibliotheksfunktionen beschrieben. Ihnen stehen natürlich auch die Funktionen der C-Bibliothek dafür zur Verfügung. Durch explizite Datentyp-Umwandlungen haben Sie eine bessere Kontrolle über Ihr Programm und die möglichen Ergebnisse.

2.3.1 CHARACTER- Zeichenketten fester Länge

Der Datentyp CHARACTER ist im zentralen SQL-Handbuch [2], Kapitel 3 beschrieben.

Der korrespondierende Datentyp in der Sprache C ist der Datentyp char. Die Hostvariable muß 1 Byte größer sein als die zugehörige CHAR-Spalte der Datenbank, um das Nullbyte der CHAR-Spalte mitaufnehmen zu können.

Verwenden Sie bei der Vereinbarung der CHAR-Hostvariablen einen Zeiger, müssen Sie dafür sorgen, daß ausreichend Speicherplatz für die Variable zur Verfügung steht.

Der ESQL/C-Precompiler akzeptiert außer CHAR noch die zwei vordefinierten Datentypen STRING und FIXCHAR für Hostvariable.

- Vereinbaren Sie die Hostvariable mit dem Datentyp CHAR, wird der Speicherplatz ggfs. mit Leerzeichen aufgefüllt und mit einem Nullbyte abgeschlossen.
- Vereinbaren Sie die Hostvariable mit dem Datentyp STRING, wird nicht mit Leerzeichen aufgefüllt und sofort nach dem Ende der Zeichenkette mit einem Nullbyte abgeschlossen.
- Der Datentyp FIXCHAR entspricht dem Datentyp CHAR bis auf den Unterschied, daß kein abschließendes Nullbyte eingefügt wird. Zur Unterscheidung verwenden wir den Begriff Zeichenfolge für Variablen vom Datentyp FIXCHAR. Eine Hostvariable, die Werte einer Spalte aufnehmen soll, die mit dem SQL-Datentyp CHAR(n) vereinbart wurde, können Sie als FIXCHAR vereinbaren, ohne ein Zeichen bei der Übernahme aus der Datenbank zu verlieren.

  Vorsicht
  Da eine Variable vom Typ FIXCHAR kein abschließendes Nullbyte besitzt, kann es passieren, daß Sie beim Übergeben einer solchen Hostvariable an eine Funktion wie z.B. printf mehr lesen, als Sie wollten. Dies geschieht, wenn die auf die entsprechende FIXCHAR-Variable folgenden Speicherplätze auch von einer Variablen vom Typ CHAR belegt sind. Ohne entsprechend begrenzte Formatangabe wird bis zur abschließenden Nullbyte gelesen.

Beispiel

```
$char hostvar[n+1];        /* Da in C Zeichenvektoren mit einem Nullbyte
                           abgeschlossen werden,sollte die Host-Variable,
                           die CHAR(n) aufnehmen soll, mit einer Größe
                           von n + 1 vereinbart werden. Es kann sonst
                           zum Verlust des letzten Zeichens aus dem
                           Datenbankfeld kommen.                    */

$char *hostvar;           /* Wobei hostvar auf einen freien Speicherbereich
                          der Länge n+1 zeigen muß                  */

$string hostvar[n+1];

$fixchar hostvar[n];
```

2.3.2 VARCHAR - Zeichenketten variabler Länge

Der Datentyp VARCHAR ist im zentralen SQL-Handbuch [2], Kapitel 3 beschrieben.

Die Hostvariable vom Datentyp VARCHAR können Sie nach einer der folgenden Regeln vereinbaren:

```
$varchar hostvar[n];
```

```
$string hostvar[n];
```

hostvar
ist der Name der Hostvariablen.

n
gibt die maximale Länge der Variablen an. Hier müssen Sie die maximale Länge der zugehörigen Spalte + 1 für das Nullbyte angeben.

Die beiden Vereinbarungen unterscheiden sich in der Behandlung der abschließenden Leerzeichen. Vereinbaren Sie die Variable mit VARCHAR, bleiben abschließende Leerzeichen bei der Wertübergabe erhalten. Vereinbaren Sie die Variable mit STRING, werden abschließende Leerzeichen abgeschnitten.

Bei der Wertübergabe von einer VARCHAR-Datenbankspalte an eine Hostvariable vom Datentyp CHAR oder FIXCHAR oder von einer Hostvariable beliebigen Typs an eine Datenbankspalte vom Datentyp VARCHAR bleiben alle benutzereingefügten Leerzeichen erhalten.

Für den Datentyp VARCHAR sind vier Makros in der Include-Datei varchar.h vordefiniert. Sie dienen dazu, Längeninformationen aus syscolumns zu erhalten:

VCLENGTH(size) bestimmt, wie groß die Längenangabe in der Vereinbarung der Hostvariable vom Datentyp VARCHAR sein muß.

```
#define VCLENGTH(len) (VCMAX(len) + 1)
```

VCMIN(size) bestimmt die minimale Länge einer Hostvariablen vom Datentyp VARCHAR aus dem entsprechenden Wert in der Systemtabelle syscolumns.

```
#define VCMIN(size)     (((size) >> 8) & 0x00ff)
```

VCMAX(size) bestimmt die maximale Länge einer Hostvariablen vom Datentyp VARCHAR aus dem entsprechenden Wert in der Systemtabelle syscolumns.

```
#define VCMAX(size)     ((size)     & 0x00ff)
```

VCSIZ(max,min) verschlüsselt die maximale und minimale Länge.

```
#define VCSIZ(max,min) ((((min) << 8) &
                        0x00ff) + ((max) &
                        0x00ff))
```

Der Inhalt der Include-Datei varchar.h ist im Anhang aufgelistet.

2.3.3 DECIMAL, NUMERIC - Festpunkt- bzw. Gleitpunktzahlen

Der SQL-Datentyp DECIMAL ist im zentralen SQL-Handbuch [2], Kapitel 3 beschrieben.

Für Hostvariablen, die Spalten vom Typ DECIMAL entsprechen sollen, stellt ESQL/C eine spezielle Struktur zur Verfügung: struct decimal.

Die Struktur ist in der Include-Datei decimal.h definiert. (vgl. Kapitel3 und Anhang)

In decimal.h ist die Struktur folgendermaßen definiert:

```
struct decimal
{
short dec_exp;
short dec_pos;
short dec_ndgts;
char  dec_dgts[DECSIZE];
};

typedef struct decimal dec_t;
```

dec_exp
ist der Exponent der normalisierten DECIMAL-Zahl. Die Basis ist 100. Dies bedeutet, der größtmögliche Wert ist 99.

dec_pos
ist das Vorzeichen der DECIMAL-Zahl. Für Zahlen mit positiven Vorzeichen (auch für 0) ist dec_pos 1, für Zahlen mit negativen Vorzeichen ist dec_pos 0.

dec_ndgts
ist die Anzahl der signifikanten Ziffern der DECIMAL-Zahl. Dies ist die Anzahl der signifikanten Ziffern in dec_dgts. Basis ist 100.

dec_dgts
ist ein Zeichenvektor, der die signifikanten Ziffern des normalisierten DECIMAL-Zahl enthält. dec_dgts[0] darf nie 0 werden. Jedes Zeichen im Vektor stellt eine 1 Byte große Binärzahl zur Basis 100 dar. In der Komponente dec_ndgts ist die Anzahl der signifikanten Ziffern in dec_dgts gespeichert.

ESQL/C stellt Ihnen Datentyp-Konvertierungsfunktionen von DECIMAL nach char, int, long, double und umgekehrt zur Verfügung.

Außerdem sind Funktionen definiert, die die 4 Grundrechenarten auf dem Typ DECIMAL realisieren. Sie finden die Funktionen im Kapitel Bibliotheksfunktionen.

Übersicht über die Bibliotheksfunktionen zum Datentyp DECIMAL:

deccvasc()	CHAR-Wert in DECIMAL umwandeln
dectoasc()	DECIMAL-Wert in CHAR umwandeln
deccvint()	int-Wert in DECIMAL umwandeln
dectoint()	DECIMAL-Wert in int umwandeln
deccvlong()	long-Wert in DECIMAL umwandeln
dectolong()	DECIMAL-Wert in long umwandeln
deccvdbl()	double-Wert in DECIMAL umwandeln
dectodbl()	DECIMAL-Wert in double umwandeln
decadd()	2 DECIMAL-Werte addieren
decsub()	DECIMAL-Wert von DECIMAL-Wert subtrahieren
decmul()	DECIMAL-Werte multiplizieren
deccmp()	2 DECIMAL-Werte vergleichen
dececvt()	DECIMAL-Wert in Zeichenkette umwandeln
decfcvt()	DECIMAL-Wert in Zeichenkette umwandeln

2.3.4 DATE

Der SQL-Datentyp DATE ist im zentralen SQL-Handbuch [2], Kapitel 3 beschrieben.

Hostvariable zum Datentyp DATE können Sie folgendermaßen vereinbaren:

```
$char[] hostvar;
oder
$long hostvar;
```

Da ein Datum intern als long-Wert dargestellt wird, ist es notwendig, daß lesbare Darstellungen von Kalenderdaten (als Zeichenketten) folgendermaßen umgewandelt werden:

- in das interne Format, wenn die Hostvariable als Konstante in einer WHERE-Klausel oder als Input-Wert bei einer INSERT- oder UPDATE-Anweisung benutzt wird.
- oder aus dem internen Format, wenn die Hostvariable bei einer SELECT-Anweisung DATE-Werte aus der Datenbank zugewiesen bekommt.

In INFORMIX werden Umwandlungen automatisch vorgenommen, wenn die Werte der Hostvariablen gewisse Bedingungen erfüllen:

interne Darstellung ——> $char var[];	Der Vektor ist groß genug, die in der Umgebungsvariable DBDATE vereinbarte Datums-darstellung aufzunehmen.
interne Darstellung <—— $char var[];	Die Darstellung des Datums entspricht der in der Umgebungsvariable DBDATE vereinbarten Datumsdarstellung

Zusätzlich stellt Ihnen INFORMIX dazu Konvertierungsfunktionen in beide Richtungen zur Verfügung, die eine vielseitigere Darstellung von Daten in Hostvariablen vom Typ CHAR [] ermöglichen.

Übersicht über die Bibliotheksfunktionen zum Datentyp DATE:

rdatestr	Konvertiert das interne Format in eine Zeichenkette
rdayofweek	Liefert von einem internen Format den Wochentag
rdefmtdate	Konvertiert Zeichenketten in internes Format
rfmtdate	Konvertiert internes Format in eine Zeichenkette
rjulmdy	Liefert Tag, Monat und Jahr vom internen Format
rleapyear	Erkennt Schaltjahre
rmdyjul	Liefert Tag, Monat und Jahr im internen Format
rstrdate	Konvertiert eine Zeichenkette in internes Format
rtoday	Wandelt Systemdatum in internes Format um

2.3.5 DATETIME - Zeitpunkte

Der SQL-Datentyp DATETIME ist im zentralen SQL-Handbuch [2], Kapitel 3 beschrieben.

Für die Hostvariable zum Datentyp DATETIME gibt es die vordefinierte Struktur dtime_t. Die Struktur steht in der Include-Datei datetime.h. Diese Datei müssen Sie einbinden, wenn Sie Hostvariablen vom Datentyp DATETIME vereinbaren wollen (siehe Kapitel 3 und Anhang).

Sie vereinbaren eine Hostvariable vom Datentyp DATETIME folgendermaßen:

```
$datetime [komponente to komponente] hostname [,...];
```

komponente
ist die Datumskomponente, wie sie für DATETIME-Werte vorgesehen ist.

Die dtime_t-Struktur sieht folgendermaßen aus:

```
typedef struct dtime {
                    short dt_qual;
                    dec_t dt_dec;
            } dtime_t;
```

Die Variable dt_qual repräsentiert die Datumskomponenten. Der Speicherbedarf des DATETIME-Wertes steht in dt_dec. Die für die dec_t-Struktur erforderliche Include-Datei decimal.h wird automatisch mitgebunden, wenn Sie sie nicht schon angegeben haben.

Der DATETIME-Wert wird dargestellt als DECIMAL-Zahl ohne Dezimalstellen; der Wert entspricht der Anzahl an Zeichen, die die Datumskomponenten festlegen. Ist z.B. eine Tabellenspalte definiert als DATETIME YEAR TO DAY, so bedeutet dies 4 Zeichen für YEAR, 2 Zeichen für MONTH und 2 Zeichen für DAY, zusammen also 8 Zeichen. Sie wird also gespeichert als DECIMAL(8,0).

Übertragen von DATETIME-Datenbankwerten in DATETIME-Hostvariablen

Bevor Sie aus einer Datenbankspalte vom Datentyp DATETIME einen Wert in eine DATETIME-Hostvariable bringen, müssen Sie den Makro TU_DTENCODE mit dem der Variablen entsprechenden Bereich aufrufen. Dieser Makro weist der Variable dt_qual in der dtime_t-Struktur einen gültigen Wert zu. Paßt der Datenbankwert nicht zu der Hostvariable, wird er mit der Funktion dtextend() an die Hostvariable angepaßt.

Hat die Variable dt_qual keinen gültigen Wert, werden der Datenbankwert und die Datumskomponenten aus der Datenbankspalte übernommen. Die Variable dt_qual wird auf 0 gesetzt, das bedeutet: der Datenbankwert wurde nicht angepaßt.

Sie können DATETIME-Werte aus der Datenbank auch Hostvariablen vom Datentyp CHAR, STRING oder FIXCHAR zuweisen. Es findet automatisch eine Konvertierung statt. Ist die Hostvariable zu klein, wird der Wert abgeschnitten und sqlca.sqlwarn.sqlwarn1 wird auf W gesetzt. Falls Sie eine Indikatorvariable an die Hostvariable gebunden haben, enthält sie die eigentliche Länge des DATETIME-Wertes. DATETIME-Werte können nicht automatisch in numerische Hostvariablen übertragen werden.

Schreiben von DATETIME-Hostvariablen in eine DATETIME-Datenbankspalte

Die Hostvariable vom Datentyp DATETIME muß einen gültigen Wert in der dt_qual-Variable haben, bevor Sie den Hostvariablen-Wert in die Datenbank schreiben können.

Die Datumskomponenten der Hostvariable können sich von denen der Datenbankspalte unterscheiden. Der Wert der Hostvariable wird der Datenbankspalte angepaßt.

Ist die Variable dt_qual nicht initialisiert, erhalten Sie einen negativen Wert in der sqlca.sqlcode und UPDATE bzw. INSERT wird nicht ausgeführt.

Ist die Hostvariable vom Datentyp CHAR, STRING oder FIXCHAR und Sie wollen einen Hostvariablen-Wert in eine Datenbankspalte vom Datentyp DATETIME schreiben, so versucht ESQL/C eine automatische Konvertierung. Ist die Konvertierung nicht sinnvoll, erhält die sqlca.sqlcode einen negativen Wert und UPDATE bzw. INSERT wird nicht ausgeführt.

Konvertierung zwischen DATETIME und DATE

Es gibt keine Bibliotheksfunktionen für die Konvertierung von DATETIME nach DATE und umgekehrt. Sie können eine Konvertierung mit folgenden Funktionen erreichen.

Konvertierung von DATETIME in DATE:

1. Mit der Funktion dtextend() erreichen Sie, daß die Hostvariable die Datumskomponenten YEAR TO DAY hat.
2. Mit der Funktion dttoasc() erhalten Sie eine Zeichenkette der Form yyyy-mm-dd.
3. Mit der Funktion rdefmtdate() mit dem Argument yyyy-mm-dd konvertieren Sie die Zeichenkette in einen DATE-Wert.

Konvertierung von DATE in DATETIME:

1. Vereinbaren Sie eine DATETIME-Hostvariable mit den Komponenten YEAR TO DAY. Dabei müssen Sie mit dem Makro TU_DTENCODE(TU_YEAR,TU_DAY) die Variable dt_qual initialisieren.
2. Mit der Funktion rfmtdate() mit dem Argument yyyy-mm-dd konvertieren Sie die DATE-Variable in eine Zeichenkette.
3. Mit der Funktion dtcvasc() konvertieren Sie die Zeichenkette in die durch den Makro vorbereitete DATETIME-Variable.
4. Falls notwendig, benutzen Sie die Funktion dtextend(), um die Datumskomponenten anzupassen.

Übersicht über die Bibliotheksfunktionen zum Datentyp DATETIME

dttoasc	konverteirt eine DATETIME-Variable in eine Zeichenkette
dtcvasc	konverteirt eine Zeichenkette in eine DATETIME-Variable
dtcurrent	gibt die aktuelle Zeit an
dtextend	ändert Datumskomponenten

Vor jedem Funktionsaufruf muß die dt_qual-Variable aus der dtime_t-Struktur mit dem Makro TU_DTENCODE(f,t) vorbereitet werden. Dabei entsprechen f und t der Anfangs- und Endkomponente der DATETIME-Hostvariable. Andernfalls ist keine Ausgabe der Variablen möglich. Der Makro ist in der Include-Datei datetime.h definiert.

2.3.6 INTERVAL - Zeitspannen

Der SQL-Datentyp INTERVAL ist im zentralen SQL-Handbuch [2], Kapitel 3 beschrieben.

Für die Hostvariable zum Datentyp INTERVAL gibt es die vordefinierte Struktur intrvl_t. Die Struktur steht in der Include-Datei datetime.h. Diese Datei müssen Sie einbinden, wenn Sie Hostvariablen vom Datentyp INTERVAL vereinbaren wollen (siehe Kapitel 3 und Anhang).

Sie vereinbaren eine Hostvariable vom Datentyp INTERVAL folgendermaßen:

```
$interval [komponente to komponente] hostname [,...];
```

komponente
ist die Datumskomponente, wie sie für INTERVAL-Werte vorgesehen ist.

Die intrvl_t-Struktur sieht folgendermaßen aus:

```
typedef struct intrvl {
                    short in_qual;
                    dec_t in_dec;
              } intrvl_t;
```

Die Variable in_qual repräsentiert die Datumskomponenten. Der Speicherbedarf des INTERVAL-Wertes steht in in_dec. Die für die dec_t-Struktur erforderliche Include-Datei decimal.h wird automatisch mitgebunden, wenn Sie sie nicht schon angegeben haben.

Der INTERVAL-Wert wird dargestellt als DECIMAL-Zahl ohne Dezimalstellen; der Wert entspricht der Anzahl an Zeichen, die die Datumskomponenten festlegen. Ist z.B. eine Tabellenspalte definiert als INTERVAL YEAR TO DAY, so bedeutet dies 4 Zeichen für YEAR, 2 Zeichen für MONTH und 2 Zeichen für DAY, zusammen also 8 Zeichen. Sie wird also gespeichert als DECIMAL(8,0).

Übertragen von INTERVAL-Datenbankwerten in INTERVAL-Hostvariablen

Bevor Sie aus einer Datenbankspalte vom Datentyp INTERVAL einen Wert in eine INTERVAL-Hostvariable bringen, müssen Sie den Makro TU_DTENCODE mit dem der Variablen entsprechenden Bereich aufrufen. Dieser Makro weist der Variable in_qual in der intrvl_t-Struktur einen gültigen Wert zu. ESQL/C überprüft dann, ob in_qual zu den Datumskomponenten der Datenbankspalte paßt.

Hat die Variable in_qual keinen gültigen Wert, werden der Datenbankwert und die Datumskomponenten aus der Datenbankspalte übernommen. Die Variable in_qual wird initialisiert.

Sie können INTERVAL-Werte aus der Datenbank auch Hostvariablen vom Datentyp CHAR, STRING oder FIXCHAR zuweisen. Es findet automatisch eine Konvertierung statt. Ist die Hostvariable zu klein, wird der Wert abgeschnitten und sqlca.sqlwarn.sqlwarn1 wird auf W gesetzt. Falls Sie eine Indikatorvariable an die Hostvariable gebunden haben, enthält sie die eigentliche Länge des INTERVAL-Wertes. INTERVAL-Werte können nicht automatisch in numerische Hostvariablen übertragen werden.

Schreiben von INTERVAL-Hostvariablen in eine INTERVAL-Datenbankspalte

Die Hostvariable vom Datentyp INTERVAL muß einen gültigen Wert in der in_qual-Variable haben, bevor Sie den Hostvariablen-Wert in die Datenbank schreiben können.

Die Datumskomponenten der Hostvariable müssen vergleichbar zu denen der Datenbankspalte sein.

Ist die Variable in_qual nicht initialisiert, erhalten Sie einen negativen Wert in der sqlca.sqlcode und UPDATE bzw. INSERT wird nicht ausgeführt.

Ist die Hostvariable vom Datentyp CHAR, STRING oder FIXCHAR und Sie wollen einen Hostvariablen-Wert in eine Datenbankspalte vom Datentyp INTERVAL schreiben, so versucht ESQL/C eine automatische Konvertierung. Ist die Konvertierung nicht sinnvoll, erhält die sqlca.sqlcode einen negativen Wert und der UPDATE bzw. INSERT wird nicht ausgeführt.

Übersicht über die Bibliotheksfunktionen zum Datentyp INTERVAL

intoasc	konvertiert eine INTERVAL-Variable in eine Zeichenkette
incvasc	konvertiert eine Zeichenkette in eine INTERVAL-Variable

2.3.7 BYTE und TEXT - Blobdaten

Die Datentypen BYTE und TEXT sind im zentralen SQL-Handbuch [2], Kapitel 3 beschrieben. Diese Datentypen stehen Ihnen nur mit dem Backend ONLINE zur Verfügung.

In ESQL/C müssen Sie Locator-Variablen vereinbaren, um Blobdaten aus der Datenbank holen oder in die Datenbank schreiben zu können. Eine Locator-Variable ist eine Struktur, die beschreibt woher das Blobdatum für einen INSERT oder UPDATE kommt, oder beschreibt wohin das Blobdatum geschrieben werden soll, wenn Sie es aus der Datenbank lesen. Die Locator-Variable enthält das Blobdatum nicht selbst; sie enthält nur Informationen über die Quelle oder das Ziel des Wertes.

Die Beschreibung der Locator-Struktur folgt. Zum Verständnis der Kommentare müssen folgende Begriffe bekannt sein:

USER zeigt an, daß die zugehörige Angabe von Ihnen anzugeben ist. Die Ausgabe wird dann vom System ausgewertet.

SYSTEM zeigt an , daß das System die Angabe macht und Sie diese Angabe auswerten können.

INTERNAL zeigt an, daß der Bereich ein Arbeitsbereich für ESQL/C ist.

Teile der Locator-Struktur sind als union realisiert. Welche Variante benutzt wird, hängt davon ab, ob das Blobdatum in einer SINIX-Datei oder im Speicher steht. In der Variable loc_loctype geben Sie an, wo das Blobdatum steht.

Die Locator-Struktur:

```
typedef struct
  {
  short loc_loctype;          /* USER: type of locator - see below    */
  union                       /* variant on 'loc'                     */
      {
      struct                  /* case LOCMEMORY                       */
          {
          long  lc_bufsize;   /* USER: buffer size */
          char *lc_buffer;    /* USER: memory buffer to use           */
          char *lc_currdata_p;/* INTERNAL: current memory buffer      */
          int   lc_mflags;    /* INTERNAL: memory flags (see below)   */
          } lc_mem;

      struct                  /* cases LOCFNAME & LOCFILE             */
          {
          char *lc_fname;     /* USER: file name                      */
          int   lc_mode;      /* USER: perm. bits used if creating    */
          int   lc_fd;        /* USER: os file descriptior            */
          long  lc_position;  /* INTERNAL: seek position              */
          } lc_file;
      } lc_union;

  long  loc_indicator;        /* USER SYSTEM: indicator               */
  long  loc_type;             /* SYSTEM: type of blob                 */
  long  loc_size;             /* USER SYSTEM: num bytes in blob or -1 */
  int   loc_status;           /* SYSTEM: status return of locator ops */
  char *loc_user_env;         /* USER: for the user's PRIVATE use     */
  long  loc_xfercount;        /* INTERNAL/SYSTEM: Transfer count      */

  int (*loc_open)();          /* USER: open function                  */
  int (*loc_close)();         /* USER: close function                 */
  int (*loc_read)();          /* USER: read function                  */
  int (*loc_write)();         /* USER: write function                 */

  int   loc_oflags;           /* USER/INTERNAL: see flag definitions below */
  } loc_t;

#define loc_fname       lc_union.lc_file.lc_fname
#define loc_fd          lc_union.lc_file.lc_fd
#define loc_position    lc_union.lc_file.lc_position
#define loc_bufsize     lc_union.lc_mem.lc_bufsize
#define loc_buffer      lc_union.lc_mem.lc_buffer
#define loc_currdata_p  lc_union.lc_mem.lc_currdata_p
#define loc_mflags      lc_union.lc_mem.lc_mflags

 /* Enumeration literals for loc_loctype */

#define LOCMEMORY       1               /* memory storage */
#define LOCFNAME        2               /* File storage with file name */
#define LOCFILE         3               /* File storage with fd */
#define LOCUSER         4               /* User define functions */

/* passed to loc_open and stored in loc_oflags */
#define LOC_RONLY       0x1             /* read only */
#define LOC_WONLY       0x2             /* write only */
/* LOC_APPEND can be set when the locator is created
 * if the file is to be appended to instead of created
 */
#define LOC_APPEND      0x4             /* write with append */
#define LOC_TEMPFILE    0x8             /* 4GL tempfile blob */

/* passed to loc_open and stored in loc_mflags */
#define LOC_ALLOC       0x1             /* free and alloc memory */

#endif  /* LOCATOR_INCL */
```

Im folgenden sind die vom Benutzer zu versorgenden Variablen bzw. die Variablen der Struktur erläutert, die vom System gesetzt werden und die der Benutzer abfragen kann.

loc_indicator

USER das Blobdatum soll mit dem NULL-Wert in die Datenbank geschrieben werden.

SYSTEM es wird eine -1 gesetzt, wenn das Blobdatum beim Holen aus der Datenbank den NULL-Wert hat.

loc_status

SYSTEM ist 0, wenn eine Operation erfolgreich war; hat einen negativen Wert, wenn ein Fehler auftritt.

loc_type

SYSTEM gibt an, um welches Blobdatum es sich handelt. Die Zahl entspricht dem Datentyp TEXT oder BYTE, wie er in der Include-Datei sqltypes.h angegeben ist.

loc_size

USER gibt den maximalen Speicherbedarf an Bytes an, wenn Sie das Blobdatum lesen oder einfügen wollen. Sie können -1 angeben, dann wird das Blobdatum bis zum Dateiende (EOF) gelesen bzw. eingefügt.

SYSTEM gibt an, wieviel Bytes das ausgewählte Blobdatum hat.

loc_user_env und
loc_union

USER sind Zeiger auf Speicherbereiche, die das Benutzerprogramm für eigenen Bedarf benutzen kann. Das System prüft diese Bereiche nicht.

loc_loctype

USER der Variable können Sie einen der folgenden Werte zuweisen: LOCMEMORY LOCFILE LOCFNAME LOCUSER Die Bedeutung der Werte ist im folgenden beschrieben, ebenso wie die Bedeutung der davon abhängigen Variablen.

Das Blobdatum im Hauptspeicher

Setzen Sie die Variable loc_loctype auf LOCMEMORY, wird das Blobdatum im Hauptspeicher gespeichert.
Setzen Sie loc_bufsize auf -1, so besorgt das Backend mit Hilfe von malloc() den Speicherplatz und setzt dann die Variablen loc_buffer und loc_bufsize.

Sie können selbst Speicherplatz für das Blobdatum besorgen, z.B. mit der C-Funktion malloc(). Das Ergebnis dieser Funktion weisen Sie der Variablen loc_buffer zu. loc_buffer gibt die Adresse des Speicherbereichs an. Dieser Speicherbereich ist ein Puffer; er muß nicht das ganze Blobdatum auf einmal enthalten. Die Größe des Speicherbereichs geben Sie in der Variable loc_bufsize an. Diese Angabe entspricht der Speicheranforderung.

Formulieren Sie aufeinanderfolgende FETCH-Anweisungen und die Menge der Daten übersteigt die Kapazität des Puffers, so wird der Puffer freigegeben und der erforderliche Speicherbedarf zur Verfügung gestellt. In Ihrem Programm müssen Sie berücksichtigen, daß sich auf diese Weise die Adresse des Blobdatums ändern kann.

Paßt der Blobwert nicht in den Speicherbereich, wird bei der FETCH-Anweisung loc_status ein negativer Wert zugewiesen und die aktuelle Länge des Blobdatums in der Variable loc_indicator abgelegt. Ist loc_bufsize kleiner als loc_size, erhalten Sie eine Fehlermeldung, wenn Sie ein Blobdatum in der Datenbank speichern wollen.

Das Blobdatum in einer geöffneten Datei

Soll das Blobdatum in eine geöffneten Datei gespeichert werden, oder aus einer geöffneten Datei gelesen werden, müssen Sie folgende Angaben machen:

Sie öffenen z.B. mit der CES-Funktion fopen() die Datei. Der Variablen loc_loctype weisen Sie den Wert LOCFILE zu. Die Dateikennzahl (file descriptor) weisen Sie der Variablen loc_fd zu. Das Backend liest oder schreibt die Blobwerte von dieser bzw. in diese Position.
Der Variablen loc_oflags müssen Sie einen der folgenden Werte zuweisen: LOC_RONLY LOC_WONLY LOC_APPEND gemäß der Zugriffsart, die Sie für die geöffnete Datei angegeben haben.

Der Wert von loc_size entscheidet, wieviele Bytes übertragen werden. Ist loc_size -1, liest das Backend bis zum Ende der Datei.

Holen Sie einen Blobwert aus der Datenbank in die geöffnete Datei, schreibt das Backend den ganzen Wert in die Datei und meldet in loc_size, wieviele Bytes übertragen wurden.

Beim Wechsel von Schreiben auf Lesen oder umgekehrt, müssen Sie mit der entsprechenden CES-Funktion die Datei mit der entsprechenden Zugriffsart erneut eröffen.

Das Blobdatum in einer nicht geöffneten Datei

Soll das Blobdatum in eine nicht geöffneten Datei gespeichert werden, oder aus dieser gelesen werden, müssen Sie folgende Angaben machen:

Sie weisen der Variablen loc_loctype den Wert LOCFNAME zu.
Der Variablen loc_fname weisen Sie den Pfadnamen der Datei zu.
Der Variablen loc_oflags weisen Sie einen der folgenden Werte zu:
LOC_RONLY
LOC_WONLY
LOC_APPEND
gemäß der Zugriffsart, die Sie für die Datei vorgesehen haben. Das Backend öffnet die Datei und setzt die Variable loc_fd. Der weitere Vorgang entspricht dem bei der geöffneten Datei.

Das Blobdatum soll von Benutzerprogrammen gespeichert werden

Setzen Sie loc_loctype auf LOCUSER, wird das Blobdatum von C-Funktionen verwaltet, die Sie selbst schreiben. Die Adressen der Funktionen werden gespeichert in loc_open, loc_close, loc_read und loc_write. Das Backend ruft diese Funktionen auf, um das Blobdatum zu verwalten.

Jede dieser Funktionen erhält die Adresse von der loc_t-Struktur als ersten oder einzigen Parameter. Die Variable loc_user_env ist von diesen Funktionen benutzbar; z.B. kann loc_user_env die Adresse des gemeinsamen Arbeitsbereichs enthalten. Außerdem sind alle Variabeln der Unterstruktur loc_union und die Variable loc_xfercount von diesen Funktionen benutzbar.

Benutzerprogrammierte Open-Funktion:

Folgende Komponenten muß die Open-Funktion enthalten.
Die Funktion hat zwei Parameter; der erste Parameter erhält die Adresse der loc_t-Struktur, der zweite Parameter erhält entweder den Wert LOC_RONLY, wenn das Blobdatum in die Datenbank geschrieben werden soll, oder LOC_WONLY, wenn das Blobdatum aus der Datenbank geholt werden soll. Der Returnwert der Funktion ist 0, wenn sie erfolgreich ist und -1, im Fehlerfall.

```
openblob(adloc, oflags)
        loc_t *adloc;
        int    oflags;
{
        adloc -> loc_status = 0;
        adloc -> loc_xfercount = 0L;
        if ( 0 == (oflags & adloc -> loc_oflags))
                  return(-1);
        if (oflags & LOC_RONLY)
           /*** prepare for store to db ***/
        else
           /*** prepare for fetch to program ***/
        return(0);
}
```

Benutzerprogrammierte Close-Funktion:

Wenn eine Datenübertragung abgeschlossen ist, wird die von Ihnen geschriebene Close-Funktion aufgerufen. Folgende Elemente muß die Funktion enthalten:

```
closeblob(adloc)
        loc_t *adloc;
{
        adloc -> loc_status = 0;
        if (adloc -> loc_oflags $ LOC_WONLY) /* if fetching */
        {
            adloc -> loc_indicator =0;  /* clear indicator */
            adloc -> loc_size = adloc -> loc_xfercount;
        }
        return(0);
}
```

Benutzerprogrammierte Read-Funktion:

Um ein Blobdatum in die Datenbank zu schreiben, ruft der Datenbankserver die von Ihnen geschriebene Read-Funktion auf. Dabei wird das Blobdatum in Teilen gelesen, bis es vollständig gelesen ist. In der Variable loc_size steht die Größe des Blobs. Haben Sie loc_size auf -1 gesetzt, wird der Blob bis zum EOF gelesen.

Die Funktion gibt als Rückgabewert die Anzahl an Bytes an, die übertragen wurden. Ist diese Zahl nicht gleich der Anzahl der geforderten Bytes, nimmt das Backend EOF an.

```
readblob(adloc, bufp, ntoread)
        loc_t *adloc;
        char  *budp;
        int   ntoread;
{
        int ntoxfer;
        ntoxfer = ntoread;
        if (adloc -> loc_size != -1)
             ntoxfer = min(ntoread,
                    adloc -> loc_size - adloc -> loc_xfercount);

         /*** transfer "ntoread" bytes to *bufp ***/

        adloc -> loc_xfercoutn += ntoxfer;
        return(ntoxfer);
}
```

Benutzerprogrammierte Output-Funktion:

Um ein Blobdatum aus der Datenbank zu holen, ruft der Datenbankserver die von Ihnen geschriebene Output-Funktion auf. Ist der Returnwert ≠ 0, ist ein Fehler aufgetreten.

```
writeblob(adloc, bufp, ntowrite)
          loc_t  *adloc;
          char   *bufp;
          int    ntowrite;
{
          /*** transfer ntowrite bytes from *bufp ***/
          adloc -> loc_xfercount += ntowrite;
          return(0);
}
```

Beispiel 1

In Beispiel 1 ist die Variable loc_loctype auf LOCMEMORY gesetzt.

```
$char name[20];
$loc_t photo;
$char *buff;

photo.loc_loctype = LOCMEMORY;
photo.loc_buffer = buff;
photo.loc_bufsize = -1;

$SELECT name, photo into $name, $photo
          FROM kunde WHERE kdnr =1234;

$INSERT INTO kunde(name, photo) VALUES($name, $photo);
```

Beispiel 2

In Beispiel 2 ist die Variable loc_loctype auf LOCFNAME gesetzt.

```
$loc_t photo;
$char dateiname[50];
$char name[20];

strcpy(dateiname, "/usr/gast/datei_photo");

photo.loc_loctype = LOCFNAME;
photo.loc_locsize = -1;
photo.loc_locbufsize = -1;
photo.loc_fname = dateiname;
photo.loc_oflags = LOC_WONLY;

$SELECT name, photo INTO $name, $photo
          FROM kunde WHERE kdnr = 1234;

strcpy(name, "Meyer");
$INSERT INTO kunde (name, photo)
               VALUES ($name, $photo);
```

2.4 NULL-Wert und Indikatorvariablen

Die Darstellung eines NULL-Wertes hängt vom Datentyp und von der Maschine ab. Oft entspricht die Darstellung nicht dem zulässigen Wert eines C-Datentyps. Sie sollten daher keine arithmetischen oder anderen Operationen auf einer Hostvariablen ausführen, die einen NULL-Wert annehmen kann.

Es gibt zwei Möglichkeiten festzustellen, ob einer Hostvariablen ein NULL-Wert aus der Datenbank zugewiesen wurde:

- Sie verwenden die Bibliotheksfunktion risnull().
- Sie binden eine Indikatorvariable an die Hostvariable und testen diese nach der Zuweisung des Wertes aus der Datenbank. Ist die Indikatorvariable negativ, wurde der Hostvariablen ein NULL-Wert zugewiesen.

Programme ohne Indikatorvariablen können beim Übersetzen überprüft werden, ob NULL-Werte an Hostvariable übergeben wurden (-icheck, siehe Kapitel 3). Mit den obengenannten Sprachmitteln ist aber eine Kontrolle im Programm möglich.

Es gibt zwei Möglichkeiten, einer Hostvariable einen NULL-Wert zuzuweisen, z. B. für eine INSERT- oder UPDATE-Anweisung:

- Sie verwenden die Bibliotheksfunktion rsetnull ().
- Sie binden eine Indikatorvariable an die Hostvariable. Die Indikatorvariable muß einen negativen Wert besitzen. Dies ist aber nur bei den Anweisungen INSERT und UPDATE möglich.

  Wenn Sie mit einer Hostvariablen einen NULL-Wert in der WHERE-Klausel einer SELECT-Anweisung darstellen wollen, dürfen Sie dabei keine negative Indikatorvariable an die Hostvariable binden. Verwenden Sie in diesem Fall entweder die Funktion rsetnull() oder den Ausdruck IS NULL.

Sie finden die beiden Bibliotheksfunktionen rsetnull() und risnull() im Kapitel 4 Bibliotheksfunktionen. Der Gebrauch von Indikatorvariablen ist im folgenden beschrieben.

2.4.1 Indikatorvariablen

Indikatorvariablen sind C-Variablen, die es Ihnen ermöglichen:

- zu prüfen, ob einer Hostvariablen ein NULL-Wert zugewiesen wurde.
- zu prüfen, ob bei der Zuweisung eines Wertes aus der Datenbank an eine Hostvariable typbedingte Fehler gemacht wurden.
- einen NULL-Wert darzustellen, wenn ein Wert mittels einer Hostvariablen in die Datenbank eingefügt bzw. in der Datenbank verändert werden soll.

Vereinbarung von Indikatorvariablen und Bindung an eine Hostvariable

Die Indikatorvariable ist eine C-Variable. Sie kann jeden Datentyp außer DATETIME und INTERVAL haben. Die Vereinbarung entspricht der Vereinbarung von Hostvariablen. Üblicherweise ist die Indikatorvariable ganzzahlig.

```
 $datentyp ind_var;

oder

EXEC SQL BEGIN DECLARE SECTION
 datentyp ind_var;
.
.
.
EXEC SQL END DECLARE SECTION
```

Die zweite Möglichkeit entspricht dem ANSI-Standard.

Sie binden eine Indikatorvariable an eine bestimmte Hostvariable innerhalb einer SQL-Anweisung. Dazu fügen Sie den Namen der Indikatorvariablen direkt an den der Hostvariablen an, nur getrennt durch das :-Zeichen bzw. durch das $-Zeichen oder getrennt durch das Schlüsselwort INDICATOR. Die Verwendung des Schlüsselworts entspricht dem ANSI-Standard.

In dynamisch formulierten SQL-Anweisungen binden Sie die Indikatorvariable folgendermaßen an die Hostvariable: Sie weisen der Komponente sqlind bzw. sqlidata der einer Hostvariablen entsprechenden sqlvar_struct-Struktur die Adresse der Indikatorvariable zu. (siehe Dynamisch formulierte Anweisungen, 2.5)

Beispiel

```
$char    name[16];
$char    comp[20];
$short   nameind;
$short   compind;
   .
   .
   .
$database versand;
$select nachname, firma
            into $name:nameind,   $comp INDICATOR compind
            from kunde
            where kunden_nr = 105;
```

Die Spalte nachname in der Tabelle kunde ist als Zeichenkette mit einer Länge von 15 Zeichen definiert. Dies hat folgende Auswirkung:

- der Komponente sqlca.sqlwarn1 in der sqlca-Struktur wird der Wert W zugewiesen
- nameind wird die tatsächliche Länge des Inhalts der Spalte nachname derTabelle kunde zugewiesen.
- name enthält nur die ersten 15 Zeichen der Spalte nachname (Die Zeichenkette name muß mit '\0' abgeschlossen werden).

Ist der Inhalt von nachname kürzer als 15 Zeichen, so werden nur die abschließenden Leerzeichen abgeschnitten.

Indikatorvariable zur Erkennung von NULL-Werten

Sie können Zuweisungen eines Spalteninhalts an eine Hostvariable überwachen,wenn Sie eine Indikatorvariable vereinbaren und an die Hostvariable binden.

Wird einer Hostvariablen ein NULL-Wert zugewiesen, so erhält die Indikatorvariable einen negativen Wert. Die Hostvariable ist jetzt mit einem undefinierten Wert belegt.

Indikatorvariable zur Erkennung von Informationsverlust

Ist die alphanumerische Hostvariable für den beim SELECT ausgewählten Wert zu klein, so wird der Wert gekürzt.

ESQL/C weist der an die Hostvariable gebundenen Indikatorvariablen die gesamte Länge des Spalteninhalts zu; diese kann kleiner als die definierte Spaltenlänge sein.

Eine Kürzung wird in der sqlca-Struktur mit einer Warnung gekennzeichnet (siehe im Abschnitt 2.7 Erfolgskontrolle).

Ist das gelieferte Ergebnis weder gekürzt worden, noch ein NULL-Wert, so erhält die Indikatorvariable den Wert 0.

Statt das Schlüsselwort NULL in einer INSERT- bzw. UPDATE-Anweisung zu verwenden, können Sie auch eine Hostvariable mit zugehöriger negativer Indikatorvariable benutzen.

2.5 Dynamisch formulierte SQL-Anweisungen

In ESQL/C gibt es die Möglichkeit, SQL-Anweisungen erst zur Laufzeit zu formulieren. Wir nennen diese Anweisungen dynamisch formulierte SQL-Anweisungen. Sie werden z.B. dazu gebraucht, interaktive Programme zu schreiben, bei denen der Benutzer erst zur Ablaufzeit eine SELECT-Anweisung am Bildschirm eingibt. Ein anderes Beispiel ist ein Programm, das mit einer oder mehreren Datenbanken arbeitet, deren Strukturen sich ändern können.

Die Problematik ist:

- Sie wissen bei der Programmerstellung nicht, welche Anweisung bei der Programmausführung eingegeben wird.
- Sie wissen bei dynamisch formulierten SELECT-Anweisungen im Quellprogramm noch nicht, wieviele Elemente die Spaltenauswahl hat und auf welche Spalteninhalte sich diese Elemente beziehen.

  Zusätzlich können auch die Anzahl und die Datentypen der Parameter in der WHERE-Klausel dem Programm unbekannt sein.
- Sie kennen bei dynamisch formulierten Nicht-SELECT-Anweisungen weder die Anzahl noch den Typ von optionalen Parametern der Anweisung.

Die dynamisch formulierten Anweisungen müssen im Quellprogramm vorbereitet und verwaltet werden. Dafür gibt es folgenden SQL-Anweisungen:

PREPARE
: übernimmt eine Zeichenkette, interpretiert sie als SQL-Anweisung und weist sie einem Anweisungbezeichner zu.

  Die übergebene Anweisung wird vorübersetzt. Die auf PREPARE folgenden Anweisungen zur dynamischen Verwaltung erhalten ihren Bezug zu der SQL-Anweisung durch den Anweisungbezeichner.

DESCRIBE
: entscheidet, ob die mit PREPARE vorübersetze Anweisung eine SELECT-Anweisung ohne INTO TEMP ist und verschafft sich im positiven Fall Information über die für einen Satz benötigte Speichergröße.

Ist die Anweisung eine INSERT-Anweisung mit in der VALUES-Klausel durch Fragezeichen gekennzeichneten Parametern, so liefert DESCRIBE für jede durch ein Fragezeichen identifizierte Spalte eine Beschreibung dieser Spalte.

Andernfalls liefert DESCRIBE nur die Art der Anweisungen als Ergebnis.

EXECUTE
führt dazu, daß die mit PREPARE an den Anweisungbezeichner übergebene Anweisung ausgeführt wird. Sie dürfen die EXECUTE-Anweisung nicht in Verbindung mit einer mittels PREPARE vorbereiteten SELECT-Anweisung ohne INTO TEMP verwenden . Für diesen Fall müssen Sie einen Satzzeiger vereinbaren (siehe DECLARE).

DECLARE
vereinbart einen Satzzeiger für eine durch PREPARE übergebene SELECT-Anweisung ohne INTO TEMP.

OPEN
bringt den mit DECLARE vereinbarten Satzzeiger zur Ausführung.

FREE
gibt Betriebsmittel des Backens frei, die für dynamisch formulierte Anweisungen oder für Satzzeiger belegt wurden.

EXECUTE IMMEDIATE
faßt die Funktionen von PREPARE, EXECUTE und FREE in einer Anweisung zusammen. Sie bereitet eine dynamisch formulierte Anweisung vor, führt sie aus und gibt die Betriebsmittel wieder frei.

Hinweis

Es ist nicht erlaubt, Anweisungen zur Satzzeigerverwaltung und Anweisungen zur Behandlung von dynamisch formulierten Anweisungen erst zur Laufzeit zu formulieren.

2.5.1 Die sqlda-Struktur

Die sqlda-Struktur ist die zentrale Datenstruktur für dynamisch formulierte Anweisungen. Sie soll mehreren Zwecken dienen.

Die dynamisch formulierte Anweisung muß hinsichtlich ihrer Art analysiert werden. Dafür steht die Anweisung DESCRIBE zur Verfügung. Diese Anweisung liefert Information darüber, ob eine mit einem Anweisungsbezeichner (der vorher mit PREPARE für die Anweisung vereinbart wurde) übergebene Anweisung eine SELECT- oder Nicht-SELECT-Anweisung ist.

- Ist die Anweisung keine SELECT-Anweisung, so wird eine genaue Beschreibung der Art der Anweisung in sqlca.sqlcode zurückgegeben.

  Besitzt die mit DESCRIBE beschriebene Anweisung Parameter, so läßt sich eine von Ihnen bereitgestellte sqlda-Struktur, die auf einen von Ihnen bereitgestellten Vektor von sqlvar_struct-Strukturen zeigt, bei richtiger Belegung ihrer Komponenten später bei der Ausführung der Anweisung durch EXECUTE genau wie eine Liste von Hostvariablen benutzen (siehe Abschnitt 2.5.3, Nicht-SELECT-Anweisungen mit Parametern).

- Ist die Anweisung eine INSERT-Anweisung, so liefert DESCRIBE für jeden in der VALUES-Klausel mit einem Fragezeichen gekennzeichneten Parameter eine Beschreibung der entsprechenden Spalte. Die Beschreibung einer jeden solchen Spalte befindet sich in den sqlvar_struct-Strukturen, auf die die sqlvar-Komponente der sqlda-Struktur zeigt.

- Bei dynamisch formulierten SELECT-Anweisungen ist die Anzahl der Elemente der Spaltenauswahl und deren Datentypen dem Programm selbst nicht bekannt.
  DESCRIBE liefert in diesem Fall mehr Information als bei Nicht-SELECT-Anweisungen. DESCRIBE erkennt die Anzahl der Elemente der Spaltenauswahl, die entsprechenden Datentypen, ihre Größe und ihre Namen und trägt diese Informationen in die von DESCRIBE bereitgestellte sqlda-Struktur ein (siehe auch Abschnitt 2.5.2, SELECT-Anweisungen ohne Parameter).

DESCRIBE benötigt auf jeden Fall einen vorher vereinbarten Zeiger auf eine sqlda-Struktur als Parameter. Durch DESCRIBE zeigt die Komponente sqlda.sqlvar bei einer SELECT-Anweisung ohne INTO TEMP auf das erste Element eines Vektors von sqlvar_struct-Strukturen (d.h. entsprechender Speicherplatz wird reserviert).

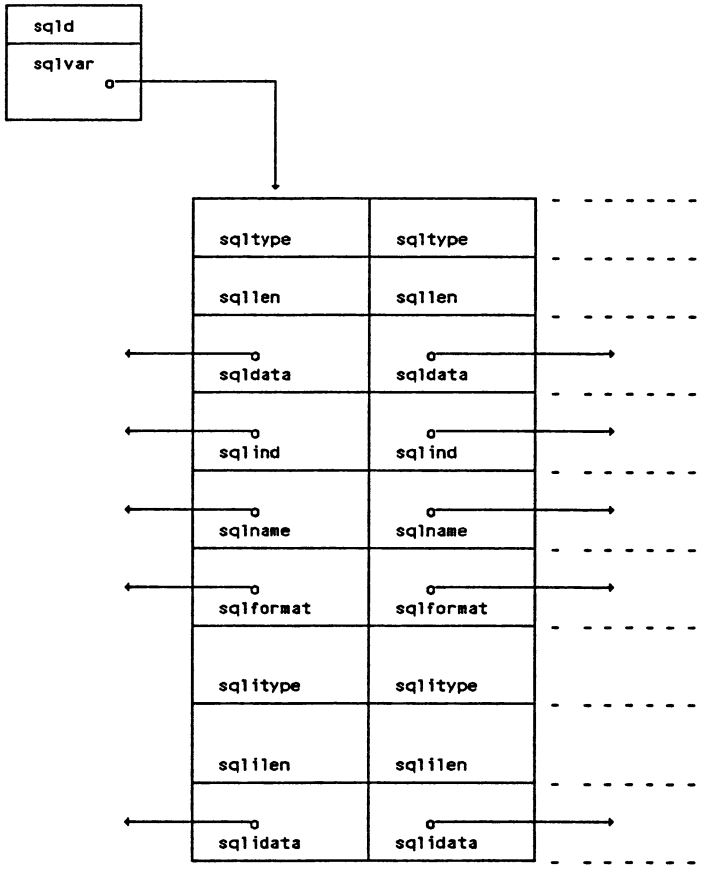
Hinweis

Die Anweisung DESCRIBE liefert keine Informationen über die mit einem Fragezeichen markierten Parameter einer dynamisch formulierten Anweisung. Dies bedeutet, daß man bei dynamisch formulierten Anweisungen mit Parametern schon zur Übersetzungszeit deren Anzahl, Reihenfolge und Typ kennen sollte.

Andernfalls wäre es notwendig, die Anweisung auf die Anzahl der Fragezeichen und der zugehörigen Spalten mit entsprechenden Typinformationen hin zu analysieren. Die sqlda-Struktur ist wie folgt definiert:

```
struct sqlvar_struct {
   short       sqltype;
   short       sqllen;
   char        *sqldata;
   short       *sqlind;
   char        *sqlname;
   char        *sqlformat;
   short       sqlitype;
   short       sqlilen;
   char        *sqlidata;
        };

struct         sqlda {
   short       sqld;
   struct sqlvar_struct        *sqlvar;
   };
```

Die Struktur läßt sich zum besseren Verständnis auch grafisch darstellen.



sqltype

Wird eine SELECT-Anweisung ohne INTO TEMP analysiert, so wird hier der SQL-Datentyp des entsprechenden Elements der Spaltenauswahl eingetragen. Die hier eingetragenen Werte entsprechen dann den Konstanten SQLCHAR - SQLINTERVAL, die in der Datei sqltypes.h definiert sind. Sie erhalten also hier Information über den SQL-Typ der entsprechenden Spalte der Ergebnistabelle.

Wird eine INSERT-Anweisung analysiert, so steht hier der SQL-Datentyp der Spalte der VALUES-Klausel, für die ein Fragezeichen angegeben wurde. Die eingetragenen Werte entsprechen den Konstanten SQLCHAR - SQLINTERVAL, die in der Datei sqltypes.h definiert sind.

Soll die sqlda-Struktur Werte aus der Datenbank zugewiesen bekommen oder anstatt einer Liste von Hostvariablen benutzt werden, um einer dynamisch formulierten Anweisung Parameterwerte zu übergeben, so müssen Sie in sqltype die Konstanten aus sqltypes.h benützen, die mit C beginnen, entsprechend dem C-Datentyp. SQL muß wissen, wie der Speicherplatz, auf die die Komponente sqldata (siehe unten) zeigt, interpretiert werden soll.

sqllen
ist eine ganze Zahl, die die Länge in Bytes eines alphanumerischen Datums angibt oder den Datumskomponentenbezeichner eines DATETIME- oder INTERVAL-Wertes angibt.
Dies gilt sowohl für die Übernahme von Werten aus der Datenbank als auch für die Ersetzung von Fragezeichen durch die sqlda-Struktur.

sqldata
ist der Zeiger auf die Daten. Wollen Sie mit einer sqlda-Struktur für eine SELECT-Anweisung Werte aus der Ergebnistabelle übernehmen, müssen Sie hier ausreichend Speicherplatz reservieren, um den entsprechenden Spaltenwert des aktuellen Satzes der Ergebnistabelle aufnehmen zu können. Verwenden Sie die sqlda-Struktur, um einer Anweisung Parameter zu übergeben, weisen Sie hier die entsprechenden Werte zu.

sqlind
ist ein Zeiger auf eine Indikatorvariable vom Datentyp short int. Der Zeiger wird bei der Anweisung DESCRIBE (für eine SELECT-Anweisung ohne INTO TEMP) zuerst auf den Nullzeiger (NIL) gesetzt.
sqlind wird als Indikatorvariable für die sqlvar_struct-Stuktur verwendet.
Schaffen Sie sich selbst eine sqlda-Struktur (z.B. mit malloc()) und wollen keine Indikatorvariable benutzen, ist es notwendig, daß Sie sqlind den Nullzeiger ((char *) NULL) zuweisen.

Verwenden Sie eine sqlda-Struktur, um Fragezeichen in einer Anweisung zu ersetzen, so können Sie hier einen Speicherbereich mit negativen Inhalt zuweisen, um einen NULL-Wert darzustellen.

Dies ist aber nur bei den Anweisungen UPDATE und INSERT möglich, um Werte in die Datenbank einzufügen oder Werte in der Datenbank zu verändern.

sqlname
zeigt auf einen Zeichenvektor, der den übergebenen Spaltennamen oder die übergebene Spaltenüberschrift enthält (bei SELECT-Anweisungen ohne INTO TEMP).

Bei INSERT-Anweisungen, die in der VALUES-Klausel Fragezeichen besitzen, wird hier der Spaltenname der entsprechenden Spalte geschrieben.

sqlformat
ist für zukünftige Anwendungen reserviert.

sqlitype
ist eine Zahl vom Datentyp int, die den Datentyp der Indikatorvariablen angibt, entsprechend der Zuordnung in sqltypes.h.

sqlilen
gibt die Länge der Indikatorvariable in Bytes an.

sqlidata
ist der Zeiger auf die Indikatorvariable.

sqld
ist eine short int-Zahl, die für eine SELECT-Anweisung die Anzahl der mit jedem Satz gelieferten Werte repräsentiert.

Wird eine INSERT-Anweisung analysiert, so steht hier die Anzahl der Fragezeichen in der VALUES-Klausel.

Wird die sqlda-Struktur zur Übergabe von Parametern benutzt, muß hier die exakte Anzahl der zu übergebenden Parameter eingetragen werden. Diese entspricht genau der Anzahl der mit Fragezeichen (?) gekennzeichneten Parameter der entsprechenden SQL-Anweisung.

sqlvar
zeigt auf eine sqlvar_struct-Struktur

Die sqlda-Struktur zeigt nur dann auf einen Vektor von sqlvar_struct-Strukturen, und diese werden nur dann in ihren Komponenten gefüllt, wenn eine SELECT-Anweisung ohne INTO TEMP oder eine INSERT-Anweisung mit Fragezeichen in der VALUES-Klausel durch die DESCRIBE-Anweisung beschrieben wird.

2.5.2 SELECT-Anweisungen ohne INTO TEMP

SELECT-Anweisungen ohne INTO TEMP ohne Parameter

Bei SELECT-Anweisungen werden die als Ergebnis gelieferten Werte der Abfrage Hostvariablen zugewiesen, die in einer INTO-Klausel aufgelistet werden.

Wenn Sie aber nun eine SELECT-Anweisung erst zur Laufzeit formulieren, nachdem Sie Ihr Programm bereits übersetzt haben, können Sie keine INTO-Klausel verwenden, weil die Hostvariablen nicht direkt zugänglich sind. Sie müssen in diesem Fall eine sqlda-Struktur verwenden.

Mit Hilfe dieser Struktur können Sie sich informieren:

- wieviele Werte pro Satz der Ergebnistabelle erwartet werden,
- welchen SQL-Datentypen diese Spalten entsprechen,
- um welche Spaltennamen es sich dabei handelt und
- ob der Inhalt einer Spalte einem NULL-Wert entspricht.

Mit der sqlda-Struktur können Sie außerdem für die übergebenen Werte entsprechend Speicherplatz reservieren und dafür sorgen, daß die übergebenen Werte dort auch entsprechend richtig abgelegt werden.

Dynamisch formulierte SELECT-Anweisungen realisieren Sie folgendermaßen:

1. Zuerst vereinbaren Sie einen Zeiger auf eine sqlda-Struktur (wir nennen ihn z.B. anw_beschr).

```
sqlda *anw_beschr;
```

   Der Zeiger auf die sqlda-Struktur wird nicht als Hostvariable vereinbart, also ohne vorangestelltes $-Zeichen. Auch beim Einbau in die entsprechende Anweisung DESCRIBE wird kein $-Zeichen vorangestellt.

2. Übersetzen Sie die dynamisch formulierte Anweisung mit PREPARE vor und weisen Sie ihr dadurch einen Anweisungbezeichner zu (wir nennen ihn z.B. abfrage). Die SELECT-Anweisung ist dabei in der Hostvariablen anweisung gespeichert.

```
$prepare abfrage from $anweisung;
```

3. Mit der Anweisung DESCRIBE analysieren Sie die dynamisch formulierte Anweisung. Als Ergebnis wird dann die sqlda-Struktur anw_beschr entsprechend belegt.

```
$describe abfrage into anw_beschr;
```

Diese Anweisung führt dazu, daß anw_beschr.sqlvar auf eine Reihe von sqlvar_struct-Strukturen zeigt, deren Komponenten teilweise belegt sind. sqld enthält die Anzahl der belegten sqlvar_struct-Strukturen.

Jede sqlvar_struct-Struktur, jeweils für ein Element der Spaltenauswahl zuständig, wird durch die DESCRIBE-Anweisung in den Komponenten sqltype, sqllen (für alphanumerische Daten oder für Datumskomponenten der DATETIME oder INTERVAL-Daten) und sqlname besetzt:

sqlind wird mit dem Nullzeiger ((char *) NULL) belegt.

In sqltype wird der SQL-Datentyp des gefundenen Elements der Spaltenauswahl beschrieben. Um diese Konstanten in Ihrem Programm verwenden zu können,müssen Sie mit $include sqltypes.h die Datei, in der die Konstanten definiert sind, in Ihren Quellprogramm einfügen.

sqldata erhält den entsprechenden Spaltennamen.

4. Belegen Sie nun die sqlda-Struktur:

Besorgen Sie sich Speicherplatz für die erwarteten Werte und weisen Sie dessen Anfangsadresse in jeder sqlvar_struct-Struktur der Komponente zu, die auf den Wert zeigt, den Sie für jeden Satz der Ergebnistabelle erhalten.

Diese Komponente ist der Zeiger auf den erhaltenen Wert: sqldata. Hier wird bei der späteren Ausführung einer FETCH-Anweisung der aus dem aktuellen Satz der Ergebnistabelle erhaltene Wert geschrieben. Dies setzt aber voraus, daß Sie dafür genügend Speicherplatz zur Verfügung gestellt haben.

Sie müssen also z.B.die Funktion malloc() verwenden und die richtigen Grenzen zuweisen. Diese hängen vom Datentyp und bei alphanumerischen Variablen von der Länge der Variablen ab.

Wollen Sie für jeden gelieferten Wert eine Indikator-Variable vereinbaren, so müssen Sie sqlind auf einen Speicherbereich zeigen lassen, der eine short int-Zahl aufnehmen kann.

In die Komponente sqltype müssen Sie jetzt den C-Typ, codiert nach den in der Include-Datei sqltypes.h definierten Konstanten, eintragen, für den Sie Speicherplatz reserviert haben.

sqllen muß, wenn die entsprechende Spalte vom Datentyp CHAR bzw. VCHAR ist, die Länge des Zeichenvektors zugewiesen werden.

5. Vereinbaren Sie nun einen Satzzeiger für den Anweisungsbezeichner (wir nennen den Satzzeiger z.B. zeiger). Für dynamisch formulierte SELECT-Anweisungen muß in jedem Fall ein Satzzeiger vereinbart werden. Dies ist auch notwendig, wenn die SELECT-Anweisung nur einen Satz als Ergebnis liefert.

   Sie können entweder

   — einen einfachen Satzzeiger vereinbaren, mit dem sukzessive vom ersten bis zum letzten Satz der Ergebnistabelle zugegriffen werden kann

   — oder einen Scroll-Satzzeiger vereinbaren, der es erlaubt, direkt auf jeden Satz der Ergebnistabelle zuzugreifen.

   Näheres über die beiden Satzzeiger-Typen für SELECT-Anweisungen finden Sie im zentralen SQL-Handbuch [2], Kapitel 2.

   In unserem Beispiel wird ein einfacher Satzzeiger vereinbart.

```
$declare zeiger  cursor  for abfrage;
```

6. Der Satzzeiger muß nun geöffnet und die SELECT-Anweisung dadurch zur Ausführung gebracht werden. Dies geschieht mit der Anweisung OPEN.

```
$open zeiger;
```

   Der Satzzeiger zeigt jetzt vor den ersten Satz der Ergebnistabelle.

Der Satzzeiger muß nun auf den ersten Satz der Ergebnistabelle positioniert werden. Dies geschieht mit der Anweisung FETCH.

```
$fetch zeiger using descriptor anw_beschr;
```

Der Satzzeiger zeigt jetzt auf den ersten Satz der Ergebnistabelle. Falls diese nicht existiert, erhält sqlca.sqlcode den Wert SQLNOTFOUND zugewiesen.

Bei erfolgreicher Ausführung von FETCH wird in die Speicherbereiche, auf die die verschiedenen sqldata-Zeiger der sqlvar_struct-Strukturen zeigen, die Inhalte der Spalten des aktuellen Satzes geschrieben. Diese können dann entsprechend weiter bearbeitet werden.
Der Zugriff erfolgt dabei über die sqldata-Zeiger der sqlvar_struct-Strukturen.

8. Falls erforderlich, können Sie die FETCH-Anweisung wiederholen.

Alle dynamisch formulierten SELECT-Anweisungen müssen einen für ihren Anweisungbezeichner vereinbarten Satzzeiger besitzen.

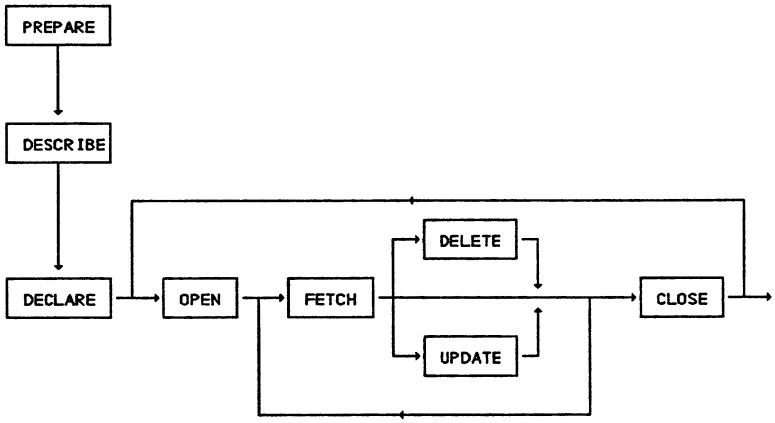
In dynamisch formulierten SELECT-Anweisungen ist es nicht erlaubt, Hostvariable explizit in der Anweisung aufzuführen. Parameter können nur durch Fragezeichen gekennzeichnet werden. Sie finden die Erklärung dazu im Abschnitt: SELECT-Anweisungen ohne INTO TEMP mit Parametern. In diesem Abschnitt sind die einzig möglichen Variablen einer SELECT-Anweisung die Speicherbereiche, die die Spalteninhalte je eines Satzes der Ergebnistabelle aufnehmen sollen. Die sqlda-Struktur dazu wird in der USING DESCRIPTOR-Klausel der FETCH-Anweisung als Parameter übergeben und steht so stellvertretend für eine Liste von Hostvariablen (mit optionalen Indikator-Variablen), deren Anzahl der Anzahl der Spalten in der Spaltenauswahl der SELECT-Anweisung entspricht.

Übersicht

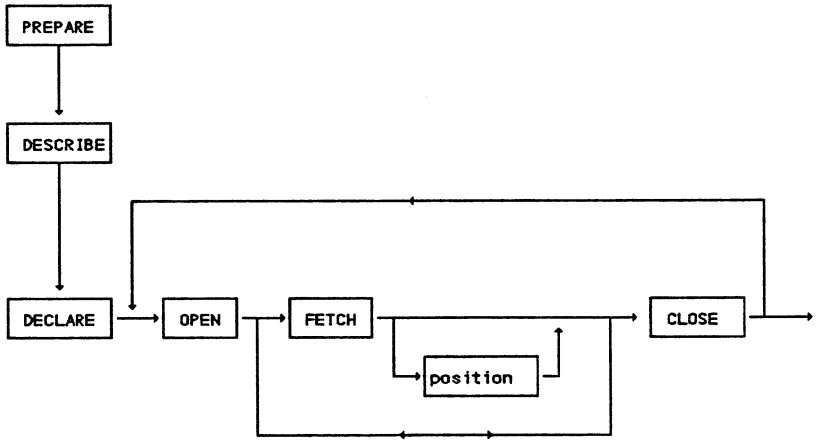
Die Reihenfolge, um dynamisch formulierte SELECT-Anweisungen zur Ausführung zu bringen, ersehen Sie aus folgenden Diagrammen für einfache und Scroll-Satzzeiger.

Der einfache Satzzeiger

Um mit einem einfachen Satzzeiger den aktuellen Satz zu verändern, ist es notwendig, die Klausel FOR UPDATE in die SELECT-Anweisung anzugeben und diese Zeichenkette der Anweisung PREPARE zu übergeben.



Der Scroll-Satzzeiger



Wobei position entweder NEXT, PREVIOUS, PRIOR, FIRST, LAST, CURRENT, RELATIVE n oder ABSOLUTE n sein kann.

Beispiel

Um diesen Vorgang besser verstehen zu können, sehen Sie sich bitte folgendes Programm an. Zur Übersichtlichkeit wurde die Fehlerbehandlung weggelassen. Wir gehen davon aus, daß zur Laufzeit eine SELECT-Anweisung eingegeben und in der Zeichenkette anweisung abgespeichert wird. Die Ergebnistabelle soll dann auf der Standardausgabe erscheinen. Nehmen wir dafür folgende Anweisung:

```
$select auftrags_nr, auftragsdatum from auftrag ;
```

Das Programm verarbeitet jede SELECT-Anweisung ohne INTO TEMP.

Zuerst werden die benötigten Include-Dateien eingefügt und entsprechende Variablen vereinbart:

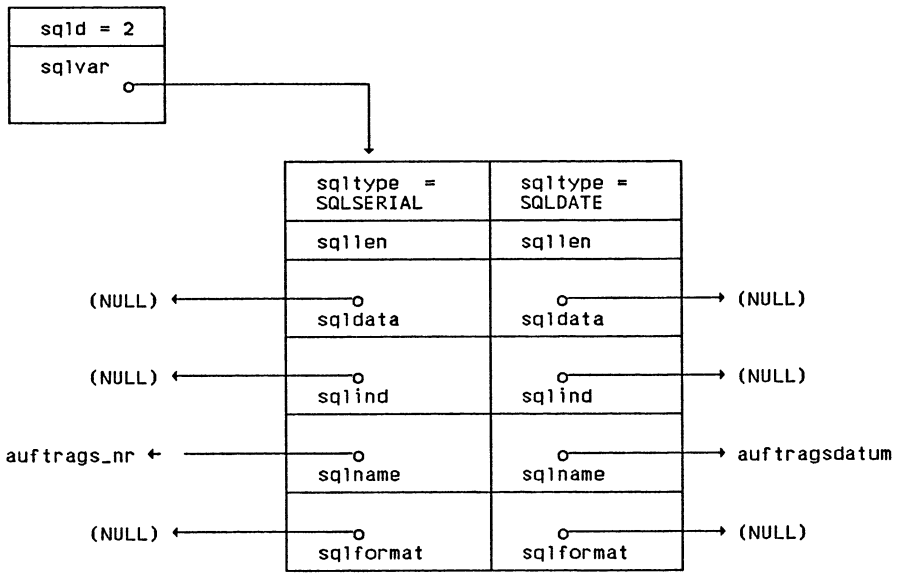
```
$include sqlca;
$include sqlda;
#include <stdio.h>
#include <decimal.h>

main()
{
        struct sqlda      *anw_beschr;  /* Zeiger auf sqlda-Struktur */
        int buffersize;                 /* benötigter Speicherplatz */
        int i;                          /* Laufvariable             */
        struct sqlvar_struct *col;  /* Hilfszeiger              */
        char *malloc( );                  /* Funkt. zur Speicherreserv.*/
        char datumsstring[17];          /* nimmt lesbares Datum auf  */
        $    char    anweisung[128]; /* Variable für Anweisung   */
```

versand ist die aktuelle Datenbank.

Der Hostvariablen anweisung wird nun die dynamisch formulierte Anweisung zugewiesen (z.B. mit einer Funktion wie getline()). Es folgen die Anweisungen PREPARE und DESCRIBE, um die SELECT-Anweisung vorzubereiten und zu analysieren.

```
$    database versand;        /* Datenbank wird ausgewählt*/

printf("BITTE SELECT-ANWEISUNG eingeben\n");
getline(anweisung,128);

$    prepare  anw_bez from $anweisung;
$    describe anw_bez into  anw_beschr;
```

Die sqlda-Struktur ist nach der Anweisung DESCRIBE folgendermaßen belegt:



Nun folgt der wichtigste Teil des Programms. Sie müssen für die Werte, die die SELECT-Anweisung liefern wird, genügend Speicherplatz reservieren und die entsprechenden Komponenten der sqlvar-Srukturen auf diese Speicherplätze zeigen lassen.

In diesem Beispiel werden keine Indikatorvariablen verwendet.

```
for (col = anw_beschr->sqlvar, i = 0; i < anw_beschr->sqld; col++, i++)
{
            /* Ausgabe des Feldnamens     */
            printf("%10s\t",col->sqlname);

            switch(col->sqltype)
          {
            case SQLSERIAL:
                 col->sqltype = CLONGTYPE;
                 buffersize = rtypmsize(CLONGTYPE);
                 col->sqldata = malloc(buffersize);
                 break;
            case SQLINT:
                 col->sqltype = CLONGTYPE;
                 buffersize = rtypmsize(CLONGTYPE);
                 col->sqldata = malloc(buffersize);
                 break;
            case SQLSMINT:
                 col->sqltype = CSHORTTYPE;
                 buffersize = rtypmsize(CSHORTTYPE,0);
                 col->sqldata = malloc(buffersize);
                 break;
            case SQLSMFLOAT:
                 col->sqltype = CFLOATTYPE;
                 buffersize = rtypmsize(CFLOATTYPE,0);
                 col->sqldata = malloc(buffersize);
                 break;
            case SQLFLOAT:
                 col->sqltype = CDOUBLETYPE;
                 buffersize = rtypmsize(CDOUBLETYPE,0);
                 col->sqldata = malloc(buffersize);
                 break;
            case SQLMONEY:
            case SQLDECIMAL:
                 col->sqltype = CDECIMALTYPE;
                 buffersize = rtypmsize(CDECIMALTYPE,0);
                 col->sqldata = malloc(buffersize);
                 break;
            case SQLDATE:
                 col->sqltype = CDATETYPE;
                 buffersize = rtypmsize(CDATETYPE,0);
                 col->sqldata = malloc(buffersize);
                 break;
            case SQLCHAR:
                 col->sqltype = CCHARTYPE;
                 buffersize = col->sqllen + 1;
                 col->sqllen += 1;
                 col->sqldata = malloc(buffersize);
                 break;
            default:
                 printf("DATENTYP BEI DESCRIBE NICHT BEKANNT\n");
                 printf("%d\n",col->sqltype);
                 exit(1);
          }
    }
            printf("\n");
```

Nun wird ein Satzzeiger für die Anweisung vereinbart. Als nächstes wird der Satzzeiger geöffnet und die Anweisung dadurch zur Ausführung gebracht.

```
$declare zeiger cursor for anw_bez;
$open zeiger;
```

Der Satzzeiger ist jetzt vor den ersten Satz der Ergebnistabelle positioniert. Solange sqlca.sqlcode nicht den Wert SQLNOTFOUND zugewiesen bekommt, wird der Satzzeiger auf den nächsten Satz positioniert und die für jeden Satz übergebenen Werte auf der Standardausgabe gezeigt. Geeignete cast-Operationen wandeln den Zeiger sqldata in den entsprechenden Typ-Zeiger um, damit die übergebenen Werte richtig interpretiert werden.

```
    $   fetch zeiger using descriptor anw_beschr;

    while (sqlca.sqlcode != SQLNOTFOUND && sqlca.sqlcode >= 0 )
    {
            for (col=anw_beschr->sqlvar, i =  anw_beschr->sqld;i > 0;
                 col++, i--)
             {
            /* Jeder Wert wird entsprechend an der Standardausgabe
                            ausgegeben      */
            switch (col->sqltype)
                    {

            case CLONGTYPE:
            case CINTTYPE:
                    printf("%10d",*(long *)(col->sqldata));
                    printf("\t");
                    break;
            case CSHORTTYPE:
                    printf("%10d",*(int *)(col->sqldata));
                    printf("\t");
                    break;
            case CFLOATTYPE:
                    printf("%10.2f",*(float *)(col->sqldata));
                    printf("\t");
                    break;
            case CDOUBLETYPE:
                    printf("%10.2f",*(double *)(col->sqldata));
                    printf("\t");
                    break;
            case CDECIMALTYPE:
                    dectodbl((dec_t *)col->sqldata,&lang);
                    printf("%10.2f",lang);
                    printf("\t");
                    break;

            case CDATETYPE:
                    rdatestr(*(long *)(col->sqldata), datumsstring);
                    printf("%10s",datumsstring);
                    printf("\t");
                    break;
            case CCHARTYPE:
                    printf("%10s",col->sqldata);
                    printf("\t");
                    break;
            default:
                    printf("DATENTYP BEI FETCH NICHT BEKANNT\n");
                    exit(1);
                    }       /* Ende case    */
             }       /* Ende for     */
            printf("\n");
            $    fetch zeiger using descriptor anw_beschr;
    }       /* Ende while   */
    if ( sqlca.sqlcode != SQLNOTFOUND )
            fprintf(stderr,"FEHLER,sqlcode = %d\n", sqlca.sqlcode);
    printf(" Kein Satz mehr in der Ergebnistabelle\n");
} /* Ende MAIN */
```

Zusammenfassung

Es liegt in Ihrer Verantwortung, für jeden durch die Abfrage gelieferten Spalteninhalt in Ihrem Programm ausreichend Speicherplatz zur Verfügung zu stellen.

Sie müssen für jeden als Ergebnis gelieferten Wert eines Satzes der Ergebnistabelle den sqldata-Zeiger des entsprechenden sqlvar_struct-Elements auf den Speicherbereich zeigen lassen, in den der Wert geschrieben werden soll.

In unserem Beispiel wird durch die DESCRIBE-Anweisung Speicherplatz für die notwendige Anzahl von sqlvar_struct-Strukturen bereitgestellt. sqltype und sqlname werden Werte zugewiesen und die Zeiger sqldata und sqlind automatisch auf den Nullzeiger gesetzt.

Es liegt nun an Ihnen, ob Sie sqldata und sqlind auf einen geeigneten Speicherplatz zeigen lassen.

Verwenden Sie keine DESCRIBE-Anweisung, um die sqlvar_struct-Strukturen zu schaffen, so dürfen Sie nicht vergessen, die Zeiger auf die Indikator Variablen mit (char *)0 zu initialisieren, falls Sie diese nicht benötigen. Andernfalls könnten andere Speicherbereiche überschrieben werden.

SELECT-Anweisungen ohne INTO TEMP mit Parametern

SELECT-Anweisungen mit Parametern sind Anweisungen, bei denen Eingabeparameter erst zur Laufzeit formuliert werden. Da mit der DESCRIBE-Anweisung nur die Liste der Spaltennamen oder Ausdrücke der Spaltenauswahl der SELECT-Anweisung geprüft wird, erhält man damit keine Information über die Parameter der WHERE-Klausel.

Sie müssen die Anzahl und den Datentyp der Parameter der SELECT-Anweisungen kennen. Wenn Sie einen allgemein verwendbaren, interaktiven Interpretierer schreiben, haben Sie gewöhnlich diese Informationen.

Haben Sie diese Information nicht, müssen Sie Ihr Programm so formulieren, daß die Anzahl der Fragezeichen (?) und die entsprechenden zugehörigen Datentypen festgelegt sind.

Kennen Sie die Anzahl und den Datentyp der Parameter, so stehen Ihnen zwei Möglichkeiten offen: Entweder vereinbaren Sie eine entsprechende (gleiche) Anzahl von Hostvariablen oder Sie schaffen Speicherplatz für eine sqlda-Struktur, die Sie dazu benützen, Daten an die Anweisung zu übergeben.

Der Gebrauch von Hostvariablen entspricht bis auf einige Änderungen den Regeln für Hostvariablen, wie sie im vorherigen Abschnitt beschrieben sind.

Sie kennzeichnen in der SELECT-Anweisung die Hostvariablen mit Fragezeichen. In der OPEN-Anweisung ersetzen Sie die Fragezeichen durch aktuelle Werte, indem Sie eine USING-Klausel angeben. An das Schlüsselwort USING schließt sich eine durch Kommata getrennte Liste von Hostvariablen an, entsprechend der Anzahl an Fragezeichen. Die Reihenfolge der Fragezeichen in der SELECT-Anweisung entspricht der Reihenfolge der Hostvariablen.

```
$ open zeiger using hostvar1,hostvar2,...;
```

Eine erneute OPEN-Anweisung mit der Hostvariablen-Liste führt zu einer neuen Ergebnistabelle, die von der aktuellen Belegung der Hostvariablen und vom Inhalt der Datenbank abhängt.

Der Gebrauch der sqlda-Struktur

Sie vereinbaren im Vereinbarungsblock ihrer C-Funktion eine sqlda-Struktur und die dazugehörigen sqlvar_struct-Strukturen oder erzeugen diese dynamisch mit einer Funktion wie malloc().

Im Beispiel ersetzen 3 sqlvar_struct-Strukturen 3 Fragezeichen. Der Speicherplatz wird bei der Vereinbarung zur Verfügung gestellt:

```
struct sqlda *meinzeiger; /* Der Anweisung OPEN muß ein Zeiger auf eine
                             sqlda-Struktur als Argument übergeben
                             werden */
struct sqlda meinsqlda;
struct sqlvar_struct meinvektor[3];
```

meinzeiger muß auf die sqlda-Struktur meinsqlda zeigen, und der Zeiger sqlvar der sqlda-Struktur muß nun auf den Vektor zeigen:

```
meinzeiger = &meinsqlda;
sqlda.sqlvar = meinvektor;
```

Die Komponenten sind folgendermaßen zu belegen:

meinsqlda
: ist eine C-Variable vom Typ einer sqlda-Struktur. Die sqlvar-Komponente dieser Struktur muß dabei auf einen Vektor von sqlvar_struct-Strukturen zeigen. Wollen Sie die Fragezeichen in der Anweisung nun durch entsprechende Werte ersetzen, müssen Sie jeweils eine sqlvar_struct-Struktur mit Information über den Parameter, den Sie besetzen wollen, füllen.

  Zuerst weisen Sie der sqld-Komponente der sqlda-Struktur die Anzahl der sqlvar_struct-Strukturen zu, die Sie belegen werden und die der Anzahl der Fragezeichenn in Ihrer Anweisung entsprechen müssen.

Entsprechend dem ersten Fragezeichen in Ihrer mit PREPARE vorbereiteten Anweisung weisen Sie nun den entsprechenden Komponenten der ersten sqlvar-Struktur Werte zu:

sqltype wird ein Wert zugewiesen,der den C-Typ des zu übergebenden Wertes kodiert. Die entsprechenden Kodierungen finden Sie in der Datei sqltypes.h.

sqllen erhalt, wenn die Variable ein Zeichenvektor ist, die Länge des Zeichenvektors.

sqldata muß die Adresse des Speicherbereichs enthalten, in dem der zu verwendende Wert steht.

Sie starten die vorhergegangene Abfrage mit folgender Anweisung

```
$ open zeiger using descriptor meinzeiger;
```

Eine neue OPEN-Anweisung führt zu einer neuen Ergebnistabelle, die von der aktuellen Belegung der sqlda-Struktur und des Inhalts der Datenbank abhängt.

2.5.3 Nicht-SELECT-Anweisungen und SELECT-Anweisungen mit INTO TEMP

Dynamisch formulierte Nicht-SELECT-Anweisungen bereiten Sie mit PREPARE vor und führen Sie mit EXECUTE aus.

Eine zusätzliche DESCRIBE-Anweisung informiert Sie folgendermaßen über die mit PREPARE vorbereitete Anweisung:

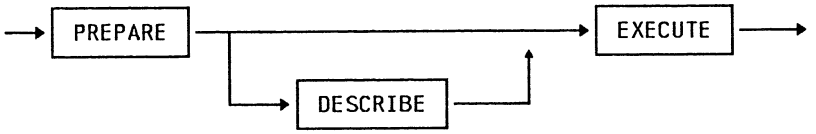
- Ist die Anweisung eine Nicht-SELECT-Anweisung bzw. SELECT-Anweisung mit INTO TEMP?
- Hat die Anweisung besondere Merkmale?

Diese Informationen liefert die DESCRIBE-Anweisung in der sqlca.sqlcode codiert nach den in der Datei sqlstype.h definierten Konstanten (siehe Abschnitt 2.7 Erfolgskontrolle und sqlca-Struktur).

Dabei ist es zusätzlich möglich, Parameter in der auszuführenden Anweisung mit Fragezeichen zu kennzeichnen und mit der Anweisung EXECUTE die entsprechenden aktuellen Werte zu übergeben (siehe nächster Abschnitt 2.5.3).

Wenn Sie für dynamisch eingegebene INSERT-Anweisung einen Satzzeiger vereinbaren möchten, um die Sätze blockweise in die Datenbank einzufügen, so erfordert dies eine spezielle Behandlung. Satzzeiger für INSERT-Anweisungen sind nur dann sinnvoll, wenn die VALUES-Klausel durch Fragegzeichen gekennzeichnete Parameter enthält. Die einzufügenden Sätze werden sonst nicht zwischengespeichert.

Sehen Sie dazu das Beispiel im Abschnitt: Nicht-SELECT-Anweisungen mit Parametern.



Beispiel für Nicht-SELECT-Anweisungen ohne Parameter

```
#include <stdio.h>
$include sqlca;

main()
{

      $char  puffer[264];
      for(;;)
      {
      printf("Bitte geben Sie Ihr Non-Select-Statement ohne Parameter ein:\n");

      /*      Lesen der Anweisungen in $puffer                */

      getline(puffer,264);

      /*   Bei Eingabe von end wird das Programm abgebrochen  */

      if (strcmp(puffer,"end") == 0)
                  exit(0);
      /*   Ein Bezeichner wird für die Anweisung vereinbart   */
      /*   und die Anweisung wird vorübersetzt                */

      $ prepare anweisung from $puffer;
      if (sqlca.sqlcode != 0)
      {
              if (sqlca.sqlcode == -201)
              {
                      printf("EIN SYNTAXFEHLER WURDE ERKANNT\n");
                      continue;
              }

              printf("FEHLERCODE BEI PREPARE:%d\n",sqlca.sqlcode);
                      exit(1);
      }
      /*  Falls kein Fehler auftritt,wird die Anweisung ausgeführt */

      $ execute anweisung;
      if (sqlca.sqlcode != 0)
      {
              printf("FEHLERCODE BEI EXECUTE:%d\n",sqlca.sqlcode);
              exit(1);
      }
      printf("Die Anweisung wurde erfolgreich abgeschlossen");
      } /*    Ende FOR(;;)    */
}     /*      Ende MAIN( )    */
```

Hier wird zuerst eine Hostvariable vereinbart, die später die Anweisung aufnehmen soll. Die Anweisung wird von der Standardeingabe eingelesen und mit der Anweisung PREPARE einem Anweisungsbezeichner übergeben. Die eingegebene Anweisung wird vorübersetzt.

Durch Prüfen von sqlca.sqlcode wird bei reinen Syntaxfehlern in der Anweisung die Anweisung ignoriert und eine neue Anweisung eingelesen. Die Eingabe von end führt zum Abbruch des Programms. Bei Fehlern endet das Programm mit der Ausgabe des Fehlercodes.

Sonst wird die Anweisung durch EXECUTE zur Ausführung gebracht.Bei erfolgreicher Ausführung wird die erfolgreiche Ausführung bestätigt und um Eingabe der nächsten Anweisung gebeten.

Nicht-SELECT-Anweisungen mit Parametern

Dynamisch formulierte Anweisungen, die zusätzlich mit Fragezeichen (?) gekennzeichnete Parameter enthalten, müssen Sie mit zusätzlichem Aufwand behandeln.

Mit der EXECUTE-Anweisung übergeben Sie die aktuellen Werte, die die Fragezeichen in der ursprünglich übergebenen Anweisung ersetzen sollen.

Sie haben dabei zwei Möglichkeiten: entweder Sie verwenden eine Liste von Hostvariablen (wobei die Anzahl der Hostvariablen genau der Anzahl der Fragezeichen entsprechen muß), oder Sie verwenden eine sqlda-Struktur.

Die DESCRIBE-Anweisung liefert nur dann Informationen über die Datentypen bzw. Namen der Spalten der mit einem Fragezeichen gekennzeichneten Parameter, wenn die Anweisung eine INSERT-Anweisung ist:

DESCRIBE stellt dazu eine sqlda- Struktur zur Verfügung, deren sqlvar-Komponente auf einen Vektor von sqlvar_struct-Strukturen zeigt. Jede sqlvar_struct-Struktur trägt dann Information über das entsprechende Element der VALUES-Klausel der INSERT-Anweisung, für die Sie ein Fragezeichen angeben. Der übergebene sqlda-Zeiger zeigt nach Ausführung von DESCRIBE auf eine sqlda-Struktur, deren sqlvar-Komponente bei der Beschreibung einer INSERT-Anweisung auf eine Folge von sqlvar_struct-Strukturen zeigt: Diese Anzahl ist in der Komponente sqlda.sqld beschrieben. Es sind genau soviele sqlvar-Strukturen mit Information belegt, wie Elemente in der VALUES-Klausel der Anweisung mit einem Fragezeichen gekennzeichnet sind. In der Komponente sqltype einer sqlvar_struct-Struktur steht nun der SQL-Datentyp des entsprechenden Elements der VALUES-Klausel verschlüsselt nach den in der Datei sqltypes.h definierten Konstanten. Ist der Datentyp alphanumerisch, so wird in der Komponente sqllen die Länge der Spalte angegeben, wie sie bei CREATE TABLE vereinbart wurde. Bei den Datentypen DATETIME und INTERVAL finden Sie in sqllen die durch die Datumskomponenten festgeschriebene Anzahl an Bytes. Beim Datentyp VCHAR gibt sqllen die maximale Länge der Spalte an.

Die Komponente sqlname jeder sqlvar_struct-Struktur zeigt auf den Spaltennamen des zugehörigen Elements der VALUES-Klausel, das mit einem Fragezeichen gekennzeichnet ist.

Die Komponenten sqldata und sqlind sind mit dem Nullzeiger initialisiert.

Für alle Anweisungen außer SELECT ohne INTO TEMP und INSERT gilt:

Die sqlda-Struktur ist nicht belegt, wird aber zur Verfügung gestellt (Speicherplatz wird bereitgestellt).

Es werden keine sqlva_struct-Strukturen zur Verfügung gestellt. Die Komponente sqlvar der sqlda-Struktur ist daher undefiniert.

Einzige Ausnahme ist der Fall, daß Sie für eine dynamisch eingegebene INSERT-Anweisung einen Satzzeiger vereinbaren möchten, um die Sätze blockweise in die Datenbank einzufügen. Sie können die Anweisung dann nicht mit EXECUTE ausführen lassen.

Ist die Anzahl und die Art der Parameter zur Übersetzungszeit bekannt, so kann eine mit dem Bezeichner anw_bez von PREPARE vorbereitete Anweisung durch die Anweisung

```
$ execute anw_bez using $hvar1, $hvar2, $hvar3;
```

gestartet werden, wenn die vorübersetzte Anweisung 3 Fragezeichen enthält.

Der Gebrauch der sqlda-Struktur

Ist die Anzahl der Parameter bis zur Laufzeit nicht bekannt und werden die Parameterdaten in die sqlda-Struktur übergeben, auf die in_vals zeigt, so wird die dynamisch formulierte Anweisung mit folgender Anweisung zum Laufen gebracht:

```
$ execute stateid using descriptor in_vals;
```

Sie müssen dabei Ihre Anweisung auf mögliche Parameter selbst analysieren (Ausnahme ist die VALUES-Klausel der INSERT-Anweisungen). Dies ist nicht unbedingt einfach und verlangt entsprechenden Aufwand, wie z.B. die Untersuchung der Anweisung auf entsprechende Fragezeichen hin. Dabei sind Ihnen aber auch die entsprechenden notwendigen Parametertypen noch nicht bekannt.

Im folgenden Beispiel wird zur Vereinfachung angenommen, daß Sie die Parameter der auszuführenden Anweisung schon kennen und nur die sqlda-Struktur vereinbaren und die entsprechenden Werte eintragen müssen.

```
#include <stdio.h>
$include sqlca;
$include sqlda;
$include sqltypes;

main( )
{

      struct sqlvar_struct *col;
      struct sqlda *beschreibung; /* Ein Zeiger auf eine sqlda-Struktur wird
                                     vereinbart */
      struct sqlda meinsqlda; /*eine sqlda-Struktur wird vereinbart */

      struct sqlvar_struct meinsqlvar; /* Eine sqlvar_struct zur Ersetzung
                                     eines Fragezeichens wird vereinbart */
      long variable;

      $database versand;

      $prepare bezeichner from "insert into auftrag (kunden_nr) values (?)";
      /* Ein unvollständiger Satz soll eingefügt werden     */

      beschreibung = &meinsqlda;
      meinsqlda.sqlvar = &meinsqlvar;

      /* Einer Variablen wird ein Wert zugewiesen, der das Fragezeichen
         in der Anweisung ersetzen soll       */

      variable = 999999;

      /* Die sqlda-Struktur wird richtig belegt                */

      beschreibung->sqld = 1;         /* Ein Fragezeichen wird ersetzt  */
      col = beschreibung->sqlvar;
      col->sqltype = CLONGTYPE;

      col->sqldata = &variable ;

      /* Die Anweisung soll ausgeführt werden. Die Argumente werden
             aus der sqlda-Struktur übernommen                */
      $execute bezeichner using beschreibung;
}
```

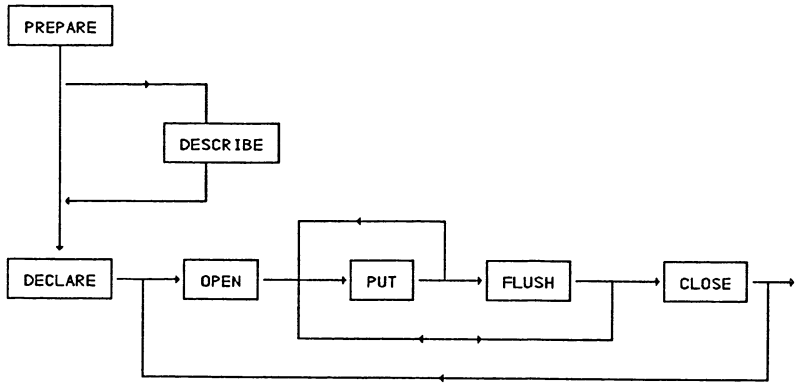
Einfügen mit Satzzeiger

Wenn Sie mit einer dynamisch formulierten INSERT-Anweisung die einzufügenden Sätze blockweise in die Datenbank eintragen wollen, müssen Sie einen Satzzeiger vereinbaren. Da die einzufügenden Sätze aber nur zwischengespeichert werden, wenn die INSERT-Anweisung mindestens ein Fragezeichen in der VALUES-Klausel enthält, müssen Sie zusätzlich diese Parameter bei Ausführung ersetzen.

Dies führt zu folgender Vorgehensweise:

- Zuerst wird die INSERT-Anweisung mit PREPARE vorbereitet. Die vorbereitete Anweisung erhält dabei einen Anweisungsbezeichner zugewiesen.
- Ein Satzzeiger wird für die vorbereitete Anweisung mit DECLARE vereinbart.
- Der Satzzeiger wird mit OPEN geöffnet.
- Mit PUT werden die Sätze in den Puffer eingetragen. PUT erhält die aktuellen Werte durch eine Liste von Hostvariablen oder durch eine sqlda-Struktur.
- Ein anschließendes FLUSH leert den Puffer. Die Sätze werden als Block in die Datenbank eingefügt.

Übersicht



Im folgenden Beispiel werden die aktuellen Werte durch eine Liste von Hostvariablen übergeben.

```
#include <stdio.h>
$include sqlca.h;

main()
{

      $char fname[4] ;
      $char lname[6] ;
      int i;
      sprintf(fname , "OHO");
      sprintf(lname , "OHOLD");

      $database versand;
      $prepare bezeichner from "insert into
               hersteller values (?,?)";

      $declare zeiger cursor for bezeichner;
      $open zeiger;
      $put zeiger from $fname,$lname;
      sprintf(fname , "AHA");
      sprintf(lname , "AHALD");
      $put zeiger from $fname,$lname;
      $flush zeiger;
      $close zeiger;
}
```

2.6 Satzzeiger

Satzzeiger, wie sie vereinbart, verwaltet und geschlossen werden, ist im SQL-Handbuch, Kapitel 2 umfassend beschrieben. Im folgenden finden Sie kurze Hinweise zur Gültigkeit von Satzzeigern in Programmen, sowie zur Verwendung von Hostvariablen im Zusammenhang mit Satzzeigern. Einige ESQL/C-Beispiele sollen die Zusammenhänge noch einmal verdeutlichen.

Gültigkeitsbereich von Satzzeigern

Satzzeiger sind nur in der Datei ansprechbar, in der sie vereinbart werden. Alle Anweisungen zur Satzzeigerverwaltung müssen Sie in der Datei geben, in der Sie den Satzzeiger für eine Anweisung vereinbaren. Sie können in keiner Quellprogramm-Datei auf einen Satzzeiger zugreifen, den Sie in einer anderen Datei vereinbart haben.

Der ESQL/C-Präprozessor akzeptiert Anweisungen zur Satzzeigermanipulation (OPEN, FETCH, CLOSE) nur, wenn Sie physisch nach der Vereinbarung des Satzzeigers (DECLARE) in Ihrer Datei stehen.

Ein Satzzeiger ist innerhalb einer Datei ab dem Punkt der Vereinbarung in verschiedenen C-Funktionen wie eine globale Variable ansprechbar.

2.6.1 Satzzeiger und Hostvariable

Wenn Sie Hostvariable als Konstanten in der SELECT-Anweisung benutzen wollen, so gibt es zwei Möglichkeiten, die Hostvariablen einzubauen.

- Sie führen die Hostvariablen direkt in der SELECT-Anweisung auf. Dann dürfen Sie beim Öffnen des Satzzeigers keine USING-Klausel formulieren. Die aktuellen Werte der Hostvariablen werden benutzt. Dies ist für dynamisch formulierte SELECT-Anweisungen grundsätzlich nicht zulässig.
- Sie kennzeichnen in der SELECT-Anweisung die zu belegenden Konstanten mit Fragezeichen. Dies bedeutet, daß Sie die Anweisung zuerst mit PREPARE vorübersetzen und die Fragezeichen bei der OPEN-Anweisung mit aktuellen Werten ersetzen werden. (siehe dazu 2.5 Dynamisch formulierte SQL-Anweisungen)

Hostvariablen zur Übernahme von Spalteninhalten des aktuellen Satzes

Wollen Sie mit Hostvariablen die Spalteninhalte des aktuellen Satzes der Ergebnistabelle übernehmen, stehen Ihnen zwei Möglichkeiten zur Auswahl. Entweder

- Sie führen die Hostvariablen direkt in der SELECT-Anweisung in einer INTO-Klausel auf oder
- Sie führen die Hostvariablen in einer INTO-Klausel der FETCH-Anweisung auf.

  Für dynamisch formulierte SELECT-Anweisungen ist bei der Anweisung FETCH die Übernahme der Werte eines Satzes mittels einer sqlda-Struktur vorgesehen. Lesen Sie dazu bitte den Abschnitt 2.5

Dabei gilt für die oben genannten Fälle: die Anzahl der Hostvariablen muß der Anzahl der Elemente der Spaltenauswahl entsprechen.

2.6.2 INSERT-Satzzeiger und Hostvariable

Die INSERT-Anweisung sollte zumindest eine Variable in der VALUES-Klausel besitzen. Ist dies nicht der Fall, so werden die einzufügenden Sätze nicht gepuffert.

Da es sich immer um die gleichen Sätze handelt, merkt sich INFORMIX-ESQL/C nur die Anzahl der gegebenen PUT-Anweisungen. Nur eine anschließende FLUSH- oder CLOSE-Anweisung führt dazu, daß entsprechend viele gleiche Sätze in die Datenbank eingefügt werden.

Wird der Satzzeiger nicht mit CLOSE geschlossen bzw. der Puffer mit FLUSH geleert, gehen die einzufügenden, identischen Sätze verloren.

Um innerhalb von INSERT-Anweisungen Konstanten durch Hostvariablen darzustellen, gibt es zwei Möglichkeiten:

— In Ihrer INSERT-Anweisung führen Sie die Hostvariablen in der VALUES-Klausel explizit auf.

  Den Satzzeiger vereinbaren Sie dann durch

```
$ declare zeiger cursor for insert-anweisung;
```

  Sie weisen nun in Ihrem Programm diesen Hostvariablen Werte zu. Ohne eine USING oder FROM-Klausel geben Sie nur die Anweisung

```
$ put zeiger;
```

  und der einzufügende Satz wird in den Puffer geschrieben. Die Werte entsprechen dabei den aktuellen Werten der in der VALUES-Klausel aufgeführten Hostvariablen.

— Sie ersetzen in der VALUES-Klausel Ihrer INSERT-Anweisung alle Werte, die mit Hostvariablen übergeben werden sollen, durch Fragezeichen (?).

  Sie müssen in diesem Fall die Anweisung mit PREPARE vorübersetzen und vereinbaren damit auch einen Anweisungsbezeichner für die INSERT-Anweisung. Diesen Namen übergeben Sie der Anweisung DECLARE, um einen Satzzeiger für die INSERT-Anweisung zu vereinbaren.

Sie weisen Ihren Hostvariablen die gewünschten Werte zu. Bei einer folgenden PUT-Anweisungen müssen Sie die Fragezeichen durch die aktuellen Werte der Hostvariablen ersetzen. Dies geschieht durch die FROM- bzw. USING DESCRIPTOR-Klausel der PUT-Anweisung:

Verwenden Sie die FROM-Klausel (die einfachere Form), so führen Sie nach dem Schlüsselwort FROM eine durch Kommata getrennte Liste von Hostvariablen auf. Die Anzahl der Hostvariablen muß der Anzahl den Fragezeichen in der VALUES-Klausel der INSERT-Anweisung entsprechen.

```
$ put zeiger from $hostvar1,$hostvar2,.. ;
```

Verwenden Sie die USING-DESCRIPTOR-Klausel, so müssen Sie die Werte mittels einer sqlda-Struktur übergeben (siehe 2.5).

Beispiel 1

```
#include <stdio.h>
$include sqlca;

main()
{

      $       char var[12];
      $       int  nummer;
      $       int  anz;

      $database    versand;
      $declare zeiger cursor for
              select posten_nr into $nummer
              from posten
              where menge > $anz;

      for(;;)
      {
      printf("Bitte geben Sie das Minimum für die Menge ein:\n");
      getline(var, 12);

      anz = atoi(var);
      if ( anz == 9999) /* Abbruchkriterium */
              break;
      $open zeiger ;
      for (;;)
      {
              $fetch zeiger;
              if (sqlca.sqlcode == SQLNOTFOUND)
                      break;
              if (sqlca.sqlcode != 0L)
              {
              printf("Fehlernr.:  %d beim Positionieren des Satzzeigers\n",
                                                      sqlca.sqlcode);
                      exit(1);
              }

              printf("Bestellmenge der Postennummer %d groesser als %d\n",
                                      nummer, anz);
      }       /* ENDE FOR(;;) NR.2    */
      $close zeiger;
      }       /* ENDE FOR(;;) NR.1    */

} /* ENDE MAIN()  */
```

In diesem Beispiel werden alle Postennummern ausgegeben, deren Bestellmenge eine gewisse, interaktiv eingegebene Menge überschreitet. In einer for(;;)-Schleife wird das nächste Mengenkriterium abgefragt. Gibt man als Kriterium die Zahl 9999 ein, kommt das Programm zum Ende.

Beispiel 2

Dieses Beispiel entspricht in seiner Wirkung vollkommen Beispiel 1. Beispiel 2 zeigt nur, daß ein Satzzeiger innerhalb der Datei global nach seiner Vereinbarung mit DECLARE ansprechbar ist.

```
#include <stdio.h>
$include sqlca;

$     char var[12];
$     int  nummer;
$     int  anz;

main()
{

      $database       versand;
      $declare zeiger cursor for
              select posten_nr into $nummer
              from posten
              where menge > $anz;

      weiter();
} /* Ende MAIN() */

weiter()
{
      for(;;)
      {
      printf("Bitte geben Sie das Minimum für die Menge ein:\n");
      getline(var, 12);

      anz = atoi(var);
      if ( anz == 9999) /* Abbruchkriterium */
              break;
      $open zeiger ;
      for (;;)
      {
              $fetch zeiger;
              if (sqlca.sqlcode == SQLNOTFOUND)
                      break;
              if (sqlca.sqlcode != 0L)
              {
              printf("Fehlernr.:  %d beim Positionieren des Satzzeiger\n",
                                              sqlca.sqlcode);
                      exit(1);
              }
              printf("Bestellmenge der Postennummer %d groesser als %d\n",
                                      nummer, anz);
      }       /* ENDE FOR(;;) NR.2    */
      $close zeiger;
      }       /* ENDE FOR(;;) NR.1    */
} /* Ende weiter() */
```

Beispiel INSERT-Anweisungen

Das folgende Beispiel zeigt die Verwendung eines Satzzeigers für eine INSERT-Anweisung. Die Wirkung entspricht der sukzessiven Ausführung von INSERT-Anweisungen, aber mit der zusätzlichen Zusicherung, daß die Sätze erst zwischengespeichert werden.

```
#include <stdio.h>
$include sqlca;

main( )
{
                .
                .
      $ code  char[4];
      $ name  char[16];

      $ database versand;
      $ declare zeiger cursor for insert into hersteller
                                        values ( $code,$name);

      for(;;)
      {

      printf("Bitte Herstellercode eingeben\n");
      getline(code, 3);

      /* Bei Eingabe von "end" Leeren des Puffers und Programmende    */

      if(strcmp(code,"end") == 0)
      {
              $ flush zeiger;
              $ close zeiger;
              return;
      }

      printf("Bitte Herstellernamen eingeben\n");
      getline(name, 15);

      $ put zeiger ;
      } /* Ende FOR   */

      $ flush zeiger;
      $ close zeiger;
              .
              .
} /* Ende MAIN( ) */
```

2.7 Erfolgskontrolle

Eine ordnungsgemäße Datenbankverwaltung verlangt, daß alle logisch zusammenhängenden Anweisungssequenzen, die die Datenbank verändern, auch erfolgreich beendet werden. Dies ist z.B. bei allen Buchungen der doppelten Buchführung notwendig, um nicht die verschiedenen Bilanzen zu verfälschen. Es kann sonst zu Inkonsistenzen kommen, die eine Arbeit mit der Datenbank unmöglich machen, weil darin enthaltene Daten falsch oder unvollständig sein können.

Ebenso wird es Ihnen nicht gelingen, auf eine Tabelle Ihrer Datenbank mittels einer SELECT-Anweisung zuzugreifen, wenn die vorhergehende Anweisung, die Ihre Datenbank zur aktuellen Datenbank macht, nicht vorschriftsmäßig ausgeführt wurde.

Sie sollten also jede in Ihr C-Programm eingebettete SQL-Anweisung auf eine erfolgreiche Ausführung hin überprüfen.

Dies wird in ESQL/C durch eine spezielle Struktur ermöglicht, die nach jeder Ausführung einer SQL-Anweisung Information darüber enthält, ob die entsprechende Anweisung erfolgreich ausgeführt wurde. Diese Struktur ist als sqlca-Struktur (SQL-communication area) in der Include-Datei sqlca.h vereinbart.

Die SQL-Anweisung WHENEVER wertet die sqlca-Strukur aus und ermöglicht eine komfortable Fehlerbehandlung im Programm.

2.7.1 sqlca-Struktur

Nach jeder SQL-Anweisung (außer DECLARE) wird in ESQL/C ein Ergebniscode in eine sqlca-Struktur geschrieben. Diese Struktur ist in der Include-Datei sqlca.h wie folgt definiert:

```
struct sqlca_s {
   long       sqlcode;
   char       sqlerrm[72];
   char       sqlerrp[8];
   long       sqlerrd[6];
   struct sqlcaw_s{
      char       sqlwarn0;
      char       sqlwarn1;
      char       sqlwarn2;
      char       sqlwarn3;
      char       sqlwarn4;
      char       sqlwarn5;
      char       sqlwarn6;
      char       sqlwarn7;
               } sqlwarn;
      } sqlca;
```

sqlcode
zeigt das Ergebnis einer ausgeführten ESQL/C-Anweisung an.

Das Ergebnis ist bei einer erfolgreichen Ausführung 0 bis auf zwei Ausnahmen; diese Ausnahmen sind die Anweisungen DECLARE und DESCRIBE.

DECLARE liefert keinen Ergebniscode in sqlcode.

DESCRIBE liefert in sqlcode

- eine 0, wenn eine SELECT-Anweisung ohne INTO TEMP erkannt wurde oder
- einen Wert größer 0, der die Art der analysierten Anweisung beschreibt.

  Dabei entsprechen die Werte den in der Include-Datei sqlstype.h definierten Konstanten.

  Lesen Sie dazu bitte auch die Beschreibung der Erfolgskontrolle bei den entsprechenden Beschreibungen der Anweisungen im SQL-Handbuch.

Bei einer mißlungenen Ausführung wird sqlcode negativ. Der negative Wert verschlüsselt eine entsprechende Fehlermeldung. Lesen Sie dazu das Manual Fehlermeldungen für INFORMIX-Produkte [7], das die entsprechenden Fehlercodes mit ihrer Bedeutung enthält.

Einige dieser Fehlermeldungen verlangen eine zusätzliche Fehlerbeschreibung. Diese bezieht sich auf C-ISAM-Fehler. Die entsprechende Komponente der sqlca-Struktur, die diesen Fehlercode zugewiesen bekommt, ist sqlca.sqlerrd[1]. Auch zu diesen Fehlercodes findet sich eine ausführliche Beschreibung im Handbuch Fehlermeldungen für INFORMIX-Produkte [7].

sqlca.sqlcode wird der Wert SQLNOTFOUND (= 100) zugewiesen, wenn mit einer FETCH-Anweisung der Satzzeiger hinter den letzten Satz der Ergebnistabelle bzw. auf einen Satz in der Ergebnistabelle positioniert wird, der in der Ergebnistabelle nicht existiert (z.B. den 2.ten Satz).

sqlerrm
Zur Zeit nicht in Gebrauch

sqlerrp
Zur Zeit nicht in Gebrauch

sqlerrd
Ein Feld von 6 long integer Zahlen

sqlerrd[0]	zur Zeit nicht in Gebrauch
sqlerrd[1]	zurückgegebener SERIAL-Wert oder Fehlercode von 'C-ISAM'
sqlerrd[2]	Anzahl der verarbeiteten Sätze; der Wert ist abfragbar, wenn die SQL-Anweisung durchgeführt ist. Bei der Verwendung von Satzzeigern ist nach dem 1. FETCH die Anzahl der verarbeiteten Sätze bekannt.
sqlerrd[3]	zur Zeit nicht in Gebrauch
sqlerrd[4]	Fehlerposition in der SQL-Anweisung
sqlerrd[5]	ROWID des letzten Satzes nach dem INSERT.

sqlwarn
Dies ist eine aus 8 Zeichen zusammengesetzte Struktur, deren Komponenten jeweils verschiedenste Bedingungen für Warnungen anzeigen (im Gegensatz zu Fehlern), die der Ausführung einer SQL-Anweisung folgen. Sind die Zeichen Leerzeichen, so wurden keine entsprechenden Probleme erkannt.

sqlwarn0
wird auf W gesetzt, wenn ein oder mehrere der anderen Warnzeichen auf W gesetzt wurden. Ist sqlwarn0 gleich dem Leerzeichen, so müssen die restlichen Warnzeichen nicht mehr überprüft werden.

sqlwarn1
wird auf W gesetzt, wenn ein oder mehrere Daten abgeschnitten werden mußten, um in eine alphanumerische Hostvariable zu passen. Um zu erfahren, welche Daten gekürzt wurden, müssen Sie die entsprechenden Indikatorvariablen prüfen. Dabei wird die definierte Länge der Spalte und die Länge der Hostvariablen verglichen. Es wird also auch eine Warnung signalisiert, wenn der Spalteninhalt kürzer als die Hostvariable ist.

sqlwarn1 wird nach einer DATABASE-Anweisung auf W gesetzt, wenn die Datenbank mit Transaktionssicherung arbeitet.

sqlwarn2
wird auf W gesetzt, wenn bei der Ausführung einer Mengenfunktion (COUNT, SUM, MAX, AVG, MIN) ein NULL-Wert entdeckt wurde.

sqlwarn2 wird nach einer DATABASE-Anweisung auf W gesetzt, wenn eine MODE ANSI-Datenbank mit Transaktionssicherung gewählt wurde.

sqlwarn3
wird auf W gesetzt, wenn die Anzahl der in der Spaltenauswahl einer SELECT-Anweisung genannten Spalten nicht der Anzahl der in der INTO-Klausel genannten Hostvariablen entspricht.

sqlwarn3 wird nach einer DATABASE-Anweisung auf W gesetzt, wenn eine INFORMIX-ONLINE-Datenbank gewählt wurde.

sqlwarn4
: wird bei einer DESCRIBE-Anweisung auf W gesetzt, wenn eine UPDATE- oder DELETE-Anweisung ohne WHERE-Klausel analysiert werden soll. Ohne WHERE-Klausel arbeitet UPDATE oder DELETE auf der ganzen Tabelle. Durch Abfrage dieser Variablen können Sie ungewollte globale Änderungen in Ihrer Tabelle vermeiden.

  sqlwarn4 wird auf W gesetzt, wenn eine FLOAT nach DECIMAL-Konversion durchgeführt wurde.

sqlwarn5
: wird auf W gesetzt, wenn Ihr Programm eine INFORMIX-Erweiterung zum ANSI-Standard benutzt und die Umgebungsvariable DBANSIWARN gesetzt ist.

sqlwarn6
: Zur Zeit nicht in Gebrauch

sqlwarn7
: Zur Zeit nicht in Gebrauch

2.7.2 WHENEVER-Anweisung

Die SQL-Anweisung WHENEVER hat die Funktion einer Fehlerbehandlungsroutine. Nach jeder SQL-Anweisung können Sie mit dieser Anweisung auf einen eventuellen Fehler abfragen und ggfs. zu einem gewünschten Programmteil verzweigen.

Eine WHENEVER-Anweisung zu einer Fehlersituation gilt für alle folgenden SQL-Anweisungen, bis Sie eine neue WHENEVER-Anweisung schreiben. Sie können mehrere WHENEVER-Anweisungen angeben, wenn diese sich auf unterschiedliche Fehlersituationen beziehen.

```
$WHENEVER fehler verzweigung;
```

fehler
ist die Fehlersituation, in der verzweigt werden soll; für fehler können Sie eins der folgenden Schlüsselwörter angeben:

[SQL]ERROR es soll verzweigt werden, wenn ein Fehler bei der vorherigen SQL-Anweisung aufgetreten ist.

NOT FOUND es soll verzweigt werden, wenn der Satzzeiger hinter den letzten Satz der Ergebnistabelle zeigt oder bei einem SELECT keine Sätze gefungen wurden.

SQLWARNING es soll verzweigt werden, wenn bei der Durchführung der SQL-Anweisung eine Warnung aufgetreten ist. Haben Sie die Umgebungsvariable DBANSIWARN gesetzt oder übersetzen Sie mit die Option -ansi Ihr Programm, erzeugt diese Angabe eine Warnung.

verzweigung
gibt an wo das Programm fortgesetzt werden soll; für verzweigung können Sie eine der folgenden Angaben machen:

GO TO marke oder
GOTO marke marke ist die Anweisungsmarke, zu der verzweigt wird, wenn die angegebene Fehlersituation aufgetreten ist. Wenn Sie dem ANSI-Standard genügen wollen, müssen Sie vor marke das :-Zeichen schreiben. Benutzen Sie diese Angabe, muß jede folgende Funktion vor ihrem Namen eine Marke haben.

CONTINUE zeigt an, daß das Programm nicht auf den Fehler reagieren soll. Diese Angabe können Sie verwenden, um eine vorher gesetzte Angabe außer Kraft zu setzen. Im Zusammenhang mit der Angabe SQLERROR ist CONTINUE die Voreinstellung; ESQL/C testet dann nicht auf Fehler.

Beispiele zur Erfolgskontrolle und Fehlerbehandlung finden Sie im Anhang (siehe A.1).

3 Übersetzen von ESQL/C-Programmen

Ausgangspunkt für die Übersetzung ist Ihr ESQL/C-Programm. Es enthält:

- Präprozessor-Anweisungen
- C-Quellprogrammzeilen
- integrierte SQL-Anweisungen
- Host- und Indikatorvariablen

Dieses Programm kann nicht dem C-Compiler cc übergeben werden, denn es enthält Anweisungen und Kennzeichnungen, die in der Programmiersprache C nicht erlaubt sind.

Das ESQL/C-Quellprogramm muß zuerst vom ESQL/C-Präprozessor in C-Programmcode umgewandelt werden.

Das ESQL/C-Quellprogramm muß in einer Datei stehen, deren Pfadname mit dem Suffix .ec endet.

In Abschnitt 3.1 sind die Präprozessor-Anweisungen beschrieben.

Im Abschnitt 3.2 finden Sie die Beschreibung der in ESQL/C vordefinierten Include-Dateien.

Im Abschnitt 3.3 steht das Systemkommando esql, mit dem Sie den ESQL/C-Präprozessor starten.

Abschnitt 3.4 erklärt, wie man in ESQL/C modulare Programm aufbauen kann.

3.1 Die ESQL/C Präprozessor-Anweisungen

Die Präprozessor-Anweisungen dienen dazu, SQL-Quellprogramm-Text in ein ESQL/C-Programm einzufügen und bedingtes Umformen der SQL-Anweisungen durch den ESQL/C-Präprozessor zu ermöglichen. Die Präprozessor-Anweisungen entsprechen den Präprozessor-Anweisungen für den C-Compiler; sie wirken aber nur während des ESQL/C-Präprozessor-Laufs.

Das Programm kann auch C-Präprozessor-Anweisungen enthalten, die dann vom C-Compiler bearbeitet werden.

Die ESQL/C-Präprozessor-Anweisungen können überall im ESQL/C-Programm stehen, nicht nur am Anfang des Programms.

Es gibt folgende Präprozessor-Anweisungen:

$include
$define
$undef
$ifdef
$ifndef
$else
$endif

Eine Entsprechung zur #if-Anweisung gibt es beim ESQL/C-Präprozessor nicht.

$include-Anweisung

Die $include-Anweisung bewirkt, daß anstelle der $include-Anweisung im Programm der Quellprogramm-Text aus der in der Anweisung angegebenen Datei eingefügt wird.

```
$include pfadname;
  oder
$include "pfadname";
  oder
EXEC SQL include pfadname;
```

Benutzen Sie die erste Form der include-Anweisung, sucht der Präprozessor nach der angegebenen Datei in folgender Reihenfolge:

1) im aktuellen Dateiverzeichnis

2) im Dateiverzeichnis $INFORMIXDIR/incl, wobei $INFORMIXDIR für den Inhalt der Umgebungsvariable INFORMIXDIR steht

3) im Dateiverzeichnis /usr/include

Benutzen Sie die zweite Form der include-Anweisung, sucht der Präprozessor unter diesem Pfadnamen absolut.

Das dritte Format entspricht ANSI Standard. Der Präprozessor sucht in der gleichen Reihenfolge, wie beim ersten Format.

Sie können $include-Anweisungen schachteln, und zwar bis zu achtmal.

$define-Anweisung

Die $define-Anweisung weist einem Namen einen integer-Wert zu. Sie unterstützt nicht die Definition von Zeichenkonstanten oder parametrisierten Makros, wie es die #define-Anweisung in C tut.

```
$define NAME [int_wert];
oder
EXEC SQL define NAME [int_wert];
```

int_wert
: ist ein INTEGER-Wert. Dieser Wert kann in SQL-Anweisungen für symbolische Konstanten vergleichbar der #define-Anweisung in C benutzt werden. Geben Sie int_wert nicht an, so ist Name definiert. Damit kann NAME in den Anweisungen $ifdef, $ifndef und $else abgefragt werden.

$undef

Die $undef-Anweisung löscht die Definition aus der entsprechenden $define-Anweisung.

```
$undef NAME;
oder
EXEC SQL undef NAME;
```

$ifdef

Die $ifdef-Anweisung überprüft, ob NAME mit der $define-Anweisung definiert ist. Ist dies der Fall, werden alle bis zur nächsten $endif-Anweisung folgenden Anweisungen vom ESQL/C-Präprozessor bearbeitet. Mit dieser Anweisung können Sie bedingtes Umformen des Quellprogramms durch den ESQL/C-Präprozessor veranlassen.

```
$ifdef NAME;
oder
EXEC SQL ifdef NAME;
```

$ifndef

Die $ifndef-Anweisung überprüft, ob NAME mit der $define-Anweisung definiert ist. Ist dies nicht der Fall, werden alle bis zur nächsten $endif-Anweisung folgenden Anweisungen vom ESQL/C-Präprozessor bearbeitet. Mit dieser Anweisung können Sie bedingtes Umformen des Quellprogramms durch den ESQL/C-Präprozessor veranlassen.

```
$ifndef NAME;
oder
EXEC SQL ifndef NAME;
```

$else

Die $else-Anweisung verzweigt zu einer Anweisungsfolge alternativ zu einer $ifdef- oder $ifndef-Anweisung, die dann vom ESQL/C-Präprozessor bearbeitet werden soll.

```
$else;
oder
EXEC SQL else;
```

$endif

Die $endif-Anweisung schließt eine $ifdef- bzw. $ifndef-Bedingung ab.

```
$endif;
oder
EXEC SQL endif;
```

3.2 Vordefinierte Include-Dateien

In INFORMIX-ESQL/C werden Ihnen Dateien zur Verfügung gestellt, die Definitionen von speziell für ESQL/C notwendigen Datenstrukturen und wichtigen Konstanten enthalten.

Diese Dateien müssen Sie, falls Sie darin definierte Typen oder Konstanten verwenden wollen, gemäß einer speziellen Syntax am Anfang Ihrer ESQL/C-Programme einbinden. Der ESQL/C-Präprozessor kopiert dann an der Stelle, an der Sie die Anweisung angeben, den Inhalt der entsprechenden Datei ein.

In Ihren ESQL/C-Programmen müssen Sie diese Dateien nur dann einbinden, wenn Sie Bezug auf Datentypen oder Konstanten nehmen, die in diesen definiert sind. Die enthaltenen Konstanten- und Strukturdefinitionen sind dem Programm dann bekannt.

Die Namen der zur Verfügung gestellten Dateien enden alle mit dem Suffix .h und werden im folgenden vorgestellt:

sqlca.h
: enthält die Vereinbarung der Struktur, in der Fehlermeldungen gespeichert werden (siehe Fehlerbehandlung und sqlca-Struktur). Sie sollten immer sqlca.h in Ihr ESQL/C Programm miteinbeziehen, um die erfolgreiche Ausführung Ihrer SQL-Anweisungen kontrollieren zu können.

sqlda.h
: enthält die Vereinbarung der Struktur, die der Behandlung von zur Laufzeit formulierten SQL-Anweisungen dient (siehe Kapitel 2.5, dynamische SQL-Anweisungen).

sqlstype.h
: enthält INTEGER-Konstantendefinitionen für SQL-Anweisungen (siehe Kapitel 2.5, dynamische SQL-Anweisungen).

sqltypes.h
: enthält Konstantendefinitionen für C und SQL-Datentypen

decimal.h
: enthält die Strukturdefinition der dec_t-Struktur.

datetime.h
: enthält die Strukturdefinition der dtime_t und intrvl_t-Strukturen und die für diese Datentypen notwendigen Makros.

ctools.h
: enthält Konstantendefinitionen für den ACE- und PERFORM-Anschluß.

varchar.h
: enthält Makros für die Benutzung des Datentyps VARCHAR.

locator.h
: enthält die Strukturdefinition für die Locator-Variable, die Sie vereinbaren müssen, um die Datentypen BYTE und TEXT verwenden zu können.

Sie können die Dateien mit jeweils einer der beiden folgenden Anweisungen in Ihr Programm einbeziehen:

```
$include  sqlca;                EXEC SQL include  sqlca;
$include  sqlda;                EXEC SQL include  sqlda;
$include  sqlstype;             EXEC SQL include  sqlstype;
$include  sqltypes;             EXEC SQL include  sqltypes;
$include  decimal.h;            EXEC SQL include  decimal.h;
$include  datetime.h;           EXEC SQL include  datetime.h;
$include  ctools.h;             EXEC SQL include  ctools.h;
$include  varchar.h;            EXEC SQL include  varchar.h;
$include  locator.h;            EXEC SQL include  locator.h;
```

Diese Dateien sind vollständig im Anhang aufgelistet.

Der Präprozessor sucht diese Dateien im Dateiverzeichnis /usr/include. Sind sie dort nicht installiert, gibt der Precompiler eine Meldung aus, daß er die Dateien nicht finden kann.

3.3 Übersetzungskommando esql

Mit dem Kommando esql auf Systemebene starten Sie den ESQL/C-Präprossorlauf. Der ESQL/C-Präprozessor bearbeitet die ESQL/C-Präprozessor-Anweisungen und wandelt entsprechend den ESQL/C-Präprozessor-Anweisungen die SQL-Anweisungen in C-Quellprogrammcode um. Das Ergebnis ist eine Datei mit dem Suffix .c, die Sie mit dem C-Compiler zusammen mit weiteren .c bzw. .o Objektdateien übersetzen können. So entsteht eine .o Objektdatei, die an den Linker ld weitergereicht werden kann. Dieser bindet die angegebenen Objektdateien, soweit keine Fehler vorliegen, zu einem ausführbaren Programm. Bei diesem Aufruf müssen Sie zusätzlich die ESQL/C-Bibliothek durch die Angabe -lsql an den Binder weiterreichen. Diese Bibliothek muß im Kommando vor allen anderen Bibliotheken genannt werden.

Alle diese einzelen Schritte können Sie einzeln durchführen oder mit dem esql-Kommando erledigen.

Vor dem Aufruf beachten

Vor dem Aufruf des esql-Kommandos, muß das ESQL/C-Quellprogramm in einer Datei stehen, deren Suffix .ec ist.

```
esql[␣-e][␣-esqlcargs][␣-otherargs ...][␣-o␣outfile]
    [␣-ansi]␣esqlfile.ec[␣othersrc.c ...]
      [␣otherobj.o ...][␣-lyourlib.a ...][␣-lsyslib ...]
```

-e
dieser Schalter bewirkt, daß nur der ESQL/C-Precompiler aktiviert wird. Das Quellprogramm wird nicht vom C-Compiler übersetzt und nicht gebunden.

-esqlcargs
steht für folgende Schalterargumente für den Präprozessor ESQL/C:

esqlc [␣-g|G][␣-nln][␣-V][␣-ansi][␣-EDname[=val]]
[␣-icheck][␣-EUname]

-g
durchnummerieren jeder Zeile (wird vom Debugger benutzt)

-G
keine Zeilennummerierung (wird vom Debugger benutzt;entspricht -nln)

-nln
keine Zeilennummerierung (s. -G)

-V
druckt Präprozessor-Versionsinformation

-ansi
überprüft die SQL-Anweisungen auf die Einhaltung der ANSI-Standard-Regeln.

-EDname[=val]
definiert einen Benutzernamen für den Präprozessor. Mit dem Zusatz =val können Sie dem Namen einen Wert zuweisen, z. B. -EDMACNAME=62

-icheck
überprüft, ob ein NULL-Wert an eine Hostvariable übergeben wird, die keine Indikator-Variable hat und erzeugt eine Fehlermeldung, falls dies der Fall ist.

-EUname
undefiniert

-otherargs
hier können Sie andere Schalter angeben, die an den C-Compiler cc bzw. an den Linking Editor ld weitergereicht werden.

-o outfile
outfile wird, falls Sie nicht den Schalter -e angegeben haben, der Name Ihrer ausführbaren Datei. Wenn Sie -o outfile nicht angeben, so steht Ihr ausführbares Programm in der Datei a.out.

-ansi
überprüft die SQL-Anweisungen auf die Einhaltung der ANSI-Standard-Regeln.

esqlfile.ec
ist Ihr ESQL/C-Programm .

othersrc.c
sind andere C-Quellprogramme, die an den C-Compiler übergeben werden und später an Ihr Programm gebunden werden.

otherobj.o
sind andere Objektcode Dateien, die an Ihr Programm gebunden werden.

-lyourlib.a
sind von Ihnen benötigte Bibliotheken, die zu Ihrem Programm gebunden werden.

-lsyslib
sind Systembibliotheken, die zu Ihrem Programm gebunden werden.

Hinweis

- wird das Kommando zu lang für eine Zeile, so schreiben Sie am Zeilenende einfach weiter, ohne ↵ zu drücken.
- eine Zeile fortsetzen können Sie auch, indem Sie am Ende der Zeile einen Gegenschrägstrich (\) schreiben.

Zur Überprüfung der ANSI-Standard-Kompatibilität:

Sie haben zwei Möglichkeiten ein Programm zu überprüfen ,ob es kompatibel zum ANSI-Standard ist:

- Sie setzen die Umgebungsvariable DBANSIWARN (siehe Kapitel Umgebungsvariablen). Danach wird das Programm, immer wenn es übersetzt wird oder abläuft, automatisch auf ANSI-Kompatibilität überprüft.
- Sie benutzen den Schalter -ansi. Dann wird das Programm zur Übersetzungszeit auf ANSI-Kompatibilität überprüft. Zur Laufzeit findet keine Überprüfung statt.

In beiden Fällen gibt esql im Abweichungsfall während der Übersetzung Warnungen auf dem Bildschirm aus. Ist DBANSIWARN gesetzt, wird im Abweichungsfall während des Ablaufs in der sqlca-Struktur sqlwarn 5 auf W gesetzt. Das übersetzte Benutzerprogramm kann Laufszeitüberprüfungen auf INFORMIX-Erweiterungen zur ANSI SQL Syntax machen, indem es die splca-Struktur benutzt. Insbesondere kann folgendes überprüft werden:

- werden Transaktionen durchgeführt?
- arbeitet die Datenbank als MODE ANSI?
- wird mit einer SE-Datenbank oder einer ONLINE-Datenbank gearbeitet?
- wird irgendeine Erweiterung zur ANSI SQL-Syntax benutzt?

3.4 Modularisierbarkeit

In der Programmiersprache C ist es möglich, in verschiedenen Objektdateien mit den gleichen Variablen zu arbeiten. Die Hostvariablen werden dazu in einer Datei global vereinbart und in einer anderen Datei, in der man auf sie zugreifen will, als extern deklariert.

Besonderheiten sind bei folgenden ESQL/C-Sprachelementen zu beachten:

- Satzzeiger sind nur in der Datei ansprechbar, in der sie vereinbart werden. Alle Anweisungen zur Satzzeigerverwaltung müssen Sie in der Datei angeben, in der Sie den Satzzeiger für eine Anweisung vereinbaren. Sie können in keiner anderen Datei auf einen Satzzeiger zugreifen, der in einer anderen Datei vereinbart wurde.

  Der ESQL/C-Präprozessor akzeptiert Anweisungen zur Satzzeigermanipulation (z.B. OPEN, FETCH, CLOSE) nur, wenn Sie physisch nach der Vereinbarung des Satzzeigers (DECLARE) in Ihrer Datei stehen.

  Ein Satzzeiger ist innerhalb einer Datei ab dem Punkt der Vereinbarung in verschiedenen C-Funktionen wie eine globale Variable ansprechbar.

- Für eine mit PREPARE vorübersetzte Anweisung wird ein Anweisungsbezeichner vereinbart, mit dem die vorübersetzte Anweisung ansprechbar ist. Dieser Anweisungsbezeichner ist nur in der Datei bekannt, in der er mit PREPARE vereinbart wird.

  Ist die Anweisung eine SELECT-Anweisung, so müssen Sie im allgemeinen einen Satzzeiger vereinbaren, um die Anweisung auszuführen. Bei der Vereinbarung dieses Satzzeigers wird der zuvor mit PREPARE vereinbarte Anweisungsbezeichner als Argument übergeben, um die SELECT-Anweisung zu identifizieren.

  Dies bedeutet, daß in diesem Fall alle Satzzeiger-Anweisungen und die PREPARE-Anweisung in einer Datei stehen müssen.

  Ist die Anweisung eine Nicht-SELECT-Anweisung bzw. SELECT-Anweisung mit INTO TEMP, so wird die Anweisung mit EXECUTE ausgeführt. PREPARE- und EXECUTE-Anweisung müssen in einer Datei stehen, falls sie sich auf den gleichen Anweisungsbezeichner beziehen.

Da Satzzeiger und Anweisungsbezeichner nur in der Datei bekannt sind, in der sie vereinbart werden, ist es möglich, daß Sie in verschiedenen Dateien die gleichen Namen für diese Elemente vergeben.

Dabei wird in jeder Datei immer nur das dort vereinbarte Element angesprochen.

4 Die Bibliotheksfunktionen

Dieses Kapitel beschreibt die Bibliotheksfunktionen in alphabetischer Reihenfolge. Die Funktionen befinden sich in der Bibliothek libsql.a. Diese wird beim Aufruf des Kommandos esql automatisch an Ihr C-Programm gebunden. Benutzen Sie den Aufruf esql nicht, müssen Sie diese Bibliothek explizit beim Aufrufen des C-Compilers in Ihr C-Programm einbinden.

Das Kapitel ist folgendermaßen gegliedert:

In 4.1 finden Sie die Beschreibung der Metasyntax der Funktionsbeschreibung und allgemeine Vorbemerkungen.

In 4.2 stehen die Bibliotheksfunktionen in alphabetischer Reihenfolge; eine Übersicht der Funktionen nach Themen geordnet finden Sie zu Beginn des Abschnitts 4.2.

4.1 Metasyntax der Funktionsbeschreibung

Die Funktionsbeschreibungen in diesem Kapitel folgen alle demselben Prinzip, das im folgenden erläutert wird.

Die Beschreibung einer Funktion ist in folgende Informationsabschnitte aufgeteilt:

- Funktionsname und deutsche Kurzbeschreibung
- Allgemeine Funktionsbeschreibung
- Definition der Funktion
- Parameterbeschreibung
- Returnwert
- Hinweise
- Beispiel

Die letzten drei der o.g. Abschnitte können fehlen, wenn sie für die Funktion ohne Bedeutung sind.

Allgemeine Funktionsbeschreibung

In diesem Überblick finden Sie die allgemeine Beschreibung der Arbeitsweise der Funktion.

Definition der Funktion

Die Definition beschreibt den Funktionskopf:

- Datentyp und Name der Funktion
- Liste der formalen Parameter
- Deklaration der Parameter

Parameterbeschreibung

Im Anschluß an die Funktionsdefinition finden Sie die ausführliche Beschreibung der Parameter: Bedeutung, mögliche Werte, davon abhängige Wirkungsweise usw.
Wir unterscheiden zwischen Eingabe- und Ergebnisparametern. Eingabeparameter sind in der Beschreibung durch → gekennzeichnet. Ergebnisparameter sind in der Beschreibung durch einen ← gekennzeichnet. Im Unterschied zu Eingabeparametern wird bei Ergebnisparametern der Inhalt der beim Aufruf übergebenen Variablen durch die Funktion verändert. Man spricht hier auch von impliziten Funktionsergebnissen. Ist ein Parameter als Ergebnisparameter gekennzeichnet, müssen Sie vor dem Aufruf den für das Ergebnis benötigten Speicherplatz zur Verfügung stellen, z.B. indem Sie eine passende Variable vereinbaren. Beim Aufruf übergeben Sie einen Zeiger auf den bereitgestellten Bereich.

Returnwert

Hier sind die möglichen Funktionsergebnisse aufgelistet, d.h. die dem Funktionstyp entsprechenden Returnwerte.

Hinweise

In diesem Abschnitt finden Sie folgende Informationen:

- mögliche Fehlerquellen
- Programmier- und Anwendungstips
- Zusammenhang mit anderen Funktionen

Beispiel

Kurzes Beispiel, das die Anwendung der beschriebenen Funktion illustriert. Ergebnisse und Ausgaben der Funktionen sind durch eine gestrichelte Linie vom Programmbeispiel getrennt.

4.1.1 Erläuterungen zu den Formatierfunktionen

Die Formatierfunktionen rfmtdec, rfmtdouble und rfmtlong formatieren numerische Werte nach angegebenen Formatierregeln. Die Formatierregeln ergeben sich aus dem Formatier-String, den Sie als Parameter der Funktion übergeben. Der Formatier-String ist eine Zeichenkette, die sich aus den Zeichen * & # < , . — + ( ) $ zusammensetzt. Die Bedeutung der Zeichen ist im folgenden erläutert.

Die Zeichen — + ( ) $ stehen der Zahl voran. Geben Sie das Zeichen mehrfach an, erscheint das Zeichen in der Ausgabe nur einmal. Es wandert möglichst um soviele Positionen nach rechts, wie es im Formatier-String angegeben ist.

* Stern ersetzt Leerzeichen.

& ersetzt Leerzeichen durch 0

verändert keine Leerzeichen-Position. Dieses Zeichen können Sie benutzen, um die maximale Breite der Ausgabe zu beschreiben.

< bewirkt, daß die Zahl linksbündig ausgerichtet wird.

, wird abhängig von der Umgebungsvariable DBMONEY als Komma (amerikanisch) oder als Punkt (deutsch) zur Einteilung der Zahl in Tausenderblocks in die Ausgabe geschrieben.

. ist abhangig von der Umgebungsvariable DBMONEY der Dezimalpunkt (amerikanisch) oder das Dezimalkomma (deutsch). Der Punkt darf nur einmal im Formatier-String stehen.

— ist das Minuszeichen für eine negative Zahl.

\+ ist das Pluszeichen vor einer Zahl ≥ 0.

(Die Klammer kann benutzt werden für negative Zahlen an Stelle des Minuszeichens.

) schließt die negative Zahl ab.

$ Abhängig von der Umgebungsvariable DBMONEY wird ein $-Zeichen oder DM vor die Zahl geschrieben.

Die Beispiele zu diesen Formatierregeln stehen bei den Funktionen.

Begriffserklärungen

In diesem Abschnitt finden Sie Erklärungen zu Begriffen, die häufig in den Funktionsbeschreibungen vorkommen und dort als bekannt vorausgesetzt werden.

Zeichenfolge
Zeichenfolge bedeutet eine Hostvariable vom Datentyp FIXCHAR. Den Datentyp FIXCHAR gibt es nur in ESQL/C. FIXCHAR entspricht dem Datentyp CHAR bis auf den Unterschied, daß kein abschließendes Nullbyte eingefügt ist.

Beachten Sie, daß die meisten C-Funktionen, die Zeichenketten verarbeiten, wie z.B. printf, erwarten, daß die übergebene Zeichenkette mit einem Nullbyte abgeschlossen ist.

Eine Hostvariable, die Werte einer Spalte aufnehmen soll, die mit dem SQL-Datentyp CHAR(n) vereinbart wurde, können Sie als FIXCHAR vereinbaren, ohne ein Zeichen bei der Übernahme aus der Datenbank zu verlieren.

Zeichenkette
Zeichenkette bedeutet eine mit dem Nullbyte (\0) abgeschlossene C-Zeichenkette. Eine Zeichenketten-Konstante ist eine in Anführungsstrichen eingeschlossene Folge von beliebigen Zeichen, z.B. "ABC123+=*". Eine Zeichenkette-Variable ist ein Zeiger auf einen Vektor von Zeichen (char *zg), dessen letztes Byte das Nullbyte ist.

Benutzung der Bibliotheksfunktionen

Die Bibliotheksfunktionen können wie selbstgeschriebene Funktionen benutzt, d.h. im Programm aufgerufen und mit Argumenten versorgt werden.

Es gibt einige Voraussetzungen, die im Programm für die Benutzung der Bibliotheksfunktionen geschaffen werden müssen:

- Deklaration einer Funktion

  Alle Funktionen, die einen Returnwert ungleich INTEGER zurückliefern, müssen vor dem Aufruf deklariert werden. Dies kann erfolgen durch Einfügen der entsprechenden Include-Datei, aber auch durch explizite Deklaration.

- Einfügen einer Include-Datei durch die $include-Anweisung

  Das Einfügen einer Include-Datei ist dann nötig, wenn die Funktion dort definierte Konstanten, Datentypen usw. benutzt werden. Bei der Beschreibung einer Funktion erhalten Sie den Hinweis, ob und welche Include-Datei Sie benötigen. Die Beschreibung der vordefinierten Include-Dateien finden Sie im Kapitel 3,Übersetzen und Binden.

4.2 Übersicht über die Bibliotheksfunktionen

Funktionen zum Datentyp FIXCHAR

bycmpr	vergleicht zwei Zeichenfolgen
bycopy	kopiert eine Zeichenfolge
byfill	füllt einen Bereich mit einem Zeichen auf
byleng	zählt die Zeichen einer Zeichenfolge
ldchar	kopiert eine Zeichenfolge in eine Zeichenkette

Funktionen zum SQL-Datentyp DECIMAL

decadd	addiert zwei Zahlen vom Datentyp DECIMAL
deccmp	vergleicht zwei Zahlen vom Datentyp DECIMAL
deccopy	kopiert eine Zahl vom Datentyp DECIMAL
deccvasc	konvertiert eine Zeichenkette vom Datentyp CHAR in eine Variable vom Datentyp DECIMAL
deccvdbl	konvertiert eine Zahl vom C-Datentyp double in den Datentyp DECIMAL
deccvint	konvertiert eine Zahl vom C-Datentyp int in den Datentyp DECIMAL
declong	konvertiert eine Zahl vom C-Datentyp long in den Datentyp DECIMAL
decdiv	dividiert zwei Zahlen vom Datentyp DECIMAL
dececvt	konvertiert eine Zahl vom Datentyp DECIMAL in eine Zeichenkette
decfcvt	konvertiert eine Zahl vom Datentyp DECIMAL in eine Zeichenkette
decmul	multipliziert zwei Zahlen vom Datentyp DECIMAL
decround	rundet eine Zahl vom Datentyp DECIMAL
decsub	subtrahiert zwei Zahlen vom Datentyp DECIMAL
dectoasc	konvertiert eine Zahl vom Datentyp DECIMAL in eine Zeichenkette
dectodbl	konvertiert eine Zahl vom Datentyp DECIMAL in den C-Datentyp double
dectoint	konvertiert eine Zahl vom Datentyp DECIMAL in den C-Datentyp int
dectolong	konvertiert eine Zahl vom Datentyp DECIMAL in den C-Datentyp long
dectrunc	schneidet eine Zahl vom Datentyp DECIMAl ab

Funktionen zum SQL-Datentyp DATETIME und INTERVAL

dtcurrent	gibt die aktuelle Zeit ein
dtcvasc	konvertiert eine Variable vom C-Datentyp CHAR in eine Variable vom Datentyp DATETIME
dtextend	kopiert Datumskomponenten
dttoasc	konvertiert die Datumskomponenten einer Variablen vom Datentyp DATETIME in eine Zeichenkette
incvasc	konvertiert eine Variable vom C-Datentyp CHAR in eine Variable vom Datentyp INTERVAL
intoasc	konvertiert die Datumskomponenten einer Variablen vom Datentyp INTERVAL in eine Zeichenkette

Funktionen zum SQL-Datentyp DATE

rdatestr	konvertiert das interne Format eines Datums in eine Zeichenkette
rdayofweek	gibt den Wochentag aus
rdefmtdate	bringt ein Datum ins interne Format
rfmtdate	stellt ein Datum als Zeichenkette dar
rjulmdy	zerlegt ein Datum in Tages-, Monats- und Jahreszahl
rleapyear	findet Schaltjahre heraus
rmdyjul	erzeugt ein internes Datumformat
rstrdate	bringt ein Datum ins interne Format
rtoday	liefert das Datum aus der Systemuhr im internen Format

Numerische Formatier-Funktionen

rfmtdec	konvertiert ein dec_t internes Format in eine formatierte Zeichenkette
rfmtdouble	konvertiert ein double internes Format in eine formatierte Zeichenkette
rfmtlong	konvertiert ein long internes Format in eine formatierte Zeichenkette

Funktionen zum NULL-Wert

risnull	prüft, ob eine C-Variable einen NULL-Wert repräsentiert
rsetnull	weist einer C-Variablen einen NULL-Wert zu

Funktionen zur Konversion von Zeichenketten in numerische C-Datentypen

rstod	konvertiert eine Zeichenkette in den C-Datentyp double
rstoi	konvertiert eine Zeichenkette in den C-Datentyp int
rstol	konvertiert eine Zeichenkette in den C-Datentyp long

Funktionen Ausrichtung und Speicherbedarf der Datentypen

rtypalign	richtet Zeiger auf Typgrenze aus
rtypmsize	gibt den Speicherplatzbedarf eines Datentyps an
rtypname	liefert den Namen eines SQL-Datentyps
rtypwidth	gibt den Platzbedarf nach Konvertierung eines Datentyps an

Funktionen für allgemeine Aufgaben

rgetmsg	konvertiert Fehlermeldungsnummern in Fehlermeldungstext
rupshift	konvertiert alle Buchstaben in Großbuchstaben
rdownshift	konvertiert alle Buchstaben in Kleinbuchstaben

Funktionen zum Beenden und Aufrufen eines Datenbank-Prozesses

sqlbreak	unterbricht einen Datenbankprozeß
sqlexit	beendet einen Datenbankprozeß
sqlstart	startet einen Datenbankprozeß

Funktionen für Zeichenketten

stcat	verkettet zwei Zeichenketten
stchar	kopiert eine Zeichenkette
stcmpr	vergleicht zwei Zeichenketten
stcopy	kopiert eine Zeichenkette
stleng	zählt die Anzahl an Bytes in einer Zeichenkette

4.3 Die Bibliotheksfunktionen in alphabetischer Reihenfolge

bycmpr Zwei Zeichenfolgen vergleichen

bycmpr vergleicht zwei Zeichenfolgen auf einer vorgegebenen Länge Zeichen für Zeichen bis der erste Unterschied feststellt ist. Das Ergebnis ist ein Wert vom Datentyp INTEGER. Das Vorzeichen zeigt die alphabetische Reihenfolge der Zeichenfolgen an.

```
int bycmpr(byte1, byte2, length)
   char *byte1;
   char *byte2;
   int length;
```

-→ byte1
ist der Zeiger auf die Startposition in der ersten Zeichenfolge.

-→ byte2
ist der Zeiger auf die Startposition in der zweiten Zeichenfolge.

-→ length
gibt die Länge an, auf der verglichen werden soll.

Returnwert

Das Ergebnis ergibt sich aus der Subtraktion des ASCII-Codes von byte1 und byte2.

=0 wenn die beiden Zeichenfolgen identisch sind.
<0 wenn die Zeichenfolge byte1 kleiner ist als die Zeichenfolge byte2.
>0 wenn die Zeichenfolge byte1 größer ist als die Zeichenfolge byte2.

Beispiel

```
main()

{
$fixchar *str1;
char *str2;
int lge;

printf("=0  a gleich b      <0  a kleiner b      >0  a groesser b\n\n");

str1 = "beispiel";
str2 = "beistand";
lge = 3;
printf("Vergleich von %s und %s bis zum %d. Zeichen :\n", str1, str2, lge);
printf("Ergebnis : %d\n\n", bycmpr(str1, str2, 3));

lge = 5;
printf("Vergleich von %s und %s bis zum %d. Zeichen :\n", str1, str2, lge);
printf("Ergebnis : %d\n\n", bycmpr(str1, str2, lge));

str1 = "informix";
printf("Vergleich von %s und %s bis zum 6. Zeichen :\n", str1, str2);
printf("Ergebnis : %d\n\n", bycmpr(str1, str2, 6));
}
```

```
=0  a gleich b      <0  a kleiner b      >0  a groesser b

Vergleich von beispiel und beistand bis zum 3. Zeichen :
Ergebnis : 0

Vergleich von beispiel und beistand bis zum 5. Zeichen :
Ergebnis : -1

Vergleich von informix und beistand bis zum 6. Zeichen :
Ergebnis : 1
```

bycopy Zeichenfolge kopieren

bycopy kopiert eine vorgegebene Anzahl von Zeichen an eine andere Stelle.

```
int bycopy(from, to, length)
  char *from;
  char *to;
  int length;
```

→ from
ist ein Zeiger auf die erste Position der zu kopierenden Zeichenfolge.

→ to
ist ein Zeiger auf die Position, an die kopiert werden soll. Achten Sie darauf, daß der Bereich mindestens die Länge length hat.

→ length
ist die Anzahl von Zeichen, die zu kopieren sind.

Beispiel

```
main()

{
$fixchar *str1;
$fixchar *str2;
char *str2_begin;

str1 = "spiel";
printf("1. Zeichenfolge vor Kopieren : %s\n", str1);
str2 = "bei.....programm";
printf("2. Zeichenfolge vor Kopieren : %s\n\n", str2);

printf("1. Zeichenfolge nach Kopieren : %s\n", str1);
printf("2. Zeichenfolge nach Kopieren : %s\n", bycopy(str1, str2 + 3, 5));
}
```

```
1. Zeichenfolge vor Kopieren : spiel
2. Zeichenfolge vor Kopieren : bei.....programm

1. Zeichenfolge nach Kopieren : spiel
2. Zeichenfolge nach Kopieren : beispielprogramm
```

byfill Bereich mit einem Zeichen auffüllen

byfill füllt einen durch einen Zeiger bezeichneten Speicherbereich in der vorgegebenen Länge mit einem Zeichen auf.

```
int byfill(to, length, ch)
  char *to;
  int length;
  char ch;
```

→ to
ist der Zeiger auf den Bereich, in den das Zeichen geschrieben wird. Der Bereich muß mindestens die Länge length haben.

→ length
gibt an, wie oft das Zeichen geschrieben werden soll.

→ ch
ist das Füllzeichen

Beispiel

```
main()

{
$fixchar *str;

str = "wieder_holen";
printf("Zeichenfolge vor Auffuellen : %s\n", str);

printf("Zeichenfolge nach Auffuellen : %s\n", byfill(str + 7,5,'w'));
}

------------------------------------------------------------------------

Zeichenfolge vor Auffuellen : wieder_holen
Zeichenfolge nach Auffuellen : wieder_wwwww
```

byleng Zeichen einer Zeichenfolge zählen

byleng gibt die Anzahl von Zeichen einer Zeichenfolge bis zur angegebenen Länge aus. Abschließende Leerzeichen werden nicht mitgezählt.

```
int byleng(from, count)
  char *from;
  int count;
```

→ from
ist der Zeiger auf eine Zeichenfolge.

→ count
gibt an, bis zum wievielten Byte der Zeichenfolge gezählt werden soll. Für count dürfen Sie keinen Wert angeben, der größer als die reale Länge der Zeichenfolge ist, da die Funktion sonst ein unsinniges Ergebnis liefert.

Returnwert

Der Returnwert der Funktion gibt die Länge der Zeichenfolge an.

Beispiel

```
main()

{
$fixchar *str;
int lge;

str = "    Zeichenfolge fester Laenge    ";
printf("Zeichenfolge : '%s'\n", str);

lge = 3;
printf("Laenge der Zeichenfolge : %d\n", byleng(str, lge));

lge = 23;
printf("Laenge der Zeichenfolge : %d\n", byleng(str, lge));

printf("Laenge der Zeichenfolge : %d\n", byleng(str, 33));
}
```

```
Zeichenfolge : '    Zeichenfolge fester Laenge   '
Laenge der Zeichenfolge : 0
Laenge der Zeichenfolge : 23
Laenge der Zeichenfolge : 30
```

decadd Zwei Zahlen vom Datentyp DECIMAL addieren

decadd addiert zwei Eingabewerte vom Datentyp DECIMAL und gibt das Ergebnis auf den Ergbnisparameter erg aus.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden.

```
#include <decimal.h>

int decadd(n1, n2, erg)          /* erg = n1 + n2 */
     dec_t *n1;
     dec_t *n2;
     dec_t *erg;
```

→ n1, n2
sind Zeiger auf die Eingabewerte vom Datentyp DECIMAL.

← erg
ist der Zeiger auf die dec_t-Struktur, die das Rechenergebnis aufnimmt. erg darf auch mit n1 oder n2 übereinstimmen.

Returnwert

0	Die Funktion war erfolgreich
-1200	Das Ergebnis ist zu groß für den Typ decimal (Überlauf)
-1201	Das Ergebnis ist zu klein für den Typ decimal (Unterlauf)

Beispiel

```
#include <decimal.h>

main()

{
dec_t dezimal_zahl_1;
dec_t dezimal_zahl_2;
dec_t dezimal_zahl_3;
double double_zahl;
int fehler;

printf("1. Summand : 44.321\n");
deccvasc("44.321", 80, &dezimal_zahl_1);

printf("2. Summand : 25\n");
deccvlong(25, &dezimal_zahl_2);

fehler = decadd(&dezimal_zahl_1, &dezimal_zahl_2, &dezimal_zahl_3);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n", fehler);
   exit(1);
   }

dectodbl(&dezimal_zahl_3, &double_zahl);
printf("Summe      : %f\n", double_zahl);
}
```

```
1. Summand : 44.321
2. Summand : 25
Summe      : 69.321000
```

deccmp Zwei Zahlen vom Datentyp DECIMAL vergleichen

deccmp vergleicht zwei Zahlen vom Datentyp DECIMAL.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden.

```
#include <decimal.h>

int deccmp(n1, n2)
     dec_t *n1;
     dec_t *n2;
```

→ n1, n2
sind Zeiger auf die zu vergleichenden dec_t-Strukturen.

Returnwert

0	die beiden Zahlen sind gleich
-1	die erste Zahl ist kleiner
+1	die erste Zahl ist größer
DECUNKNOWN	einer der Werte ist NULL

Beispiel

```
#include <decimal.h>

main()

{
dec_t dezimal_zahl_1, dezimal_zahl_2;
int ergebnis;

printf("1. Zahl vom Typ string : 44.0\n");
deccvasc("44.0", 80, &dezimal_zahl_1);

printf("2. Zahl vom Typ long   : 25\n\n");
deccvlong(25, &dezimal_zahl_2);

ergebnis = deccmp(&dezimal_zahl_1, &dezimal_zahl_2);
if (ergebnis == 0)
   printf("Die beiden Zahlen sind gleich\n\n");
else if (ergebnis == -1)
   printf("Die 1. Zahl ist kleiner\n\n");
else if (ergebnis == 1)
   printf("Die 1. Zahl ist groesser\n\n");
```

```
else
   printf("Einer der Werte ist NULL\n\n");

printf("1. Zahl vom Typ double : 25.0\n");
deccvdbl(25.0, &dezimal_zahl_1);

printf("2. Zahl vom Typ long   : 25\n\n");
deccvlong(25, &dezimal_zahl_2);

ergebnis = deccmp(&dezimal_zahl_1, &dezimal_zahl_2);
if (ergebnis == 0)
   printf("Die beiden Zahlen sind gleich\n\n");
else if (ergebnis == -1)
   printf("Die 1. Zahl ist kleiner\n\n");
else if (ergebnis == 1)
   printf("Die 1. Zahl ist groesser\n\n");
else
   printf("Einer der Werte ist NULL\n\n");
}
```

```
1. Zahl vom Typ string : 44.0
2. Zahl vom Typ long   : 25

Die 1. Zahl ist groesser

1. Zahl vom Typ double : 25.0
2. Zahl vom Typ long   : 25

Die beiden Zahlen sind gleich
```

deccopy Zahl vom Datentyp DECIMAL kopieren

deccopy kopiert den Inhalt einer dec_t-Struktur in eine andere dec_t-Struktur.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden.

```
#include <decimal.h>

int deccopy(n1, n2)
     dec_t *n1;
     dec_t *n2;
```

→ n1
ist der Zeiger auf die zu kopierende dec_t-Struktur.

← n2
ist der Zeiger auf die aufnehmende dec_t-Struktur.

Beispiel

```
#include <decimal.h>

main()

{
dec_t von, nach;
int i;

printf("Zahl vom Typ long              : -123456\n");
deccvlong(-123456L, &von);

printf("Entsprechende decimal-Struktur :\n");
printf("Exponent                       : %hd\n", von.dec_exp);
printf("Vorzeichen                     : %hd\n", von.dec_pos);
printf("# signifikante Ziffern         : %hd\n", von.dec_ndgts);
printf("signifikante Ziffern (dezimal) : ");
i = 0;
while (von.dec_dgts[i] != '\0')
   {
   printf("%d", von.dec_dgts[i]);
   ++i;
   }
printf("\n\n");

deccopy(&von, &nach);

printf("Kopierte decimal-Struktur :\n");
printf("Exponent                       : %hd\n", von.dec_exp);
printf("Vorzeichen                     : %hd\n", von.dec_pos);
```

```
printf("# signifikante Ziffern         : %hd\n", von.dec_ndgts);
printf("signifikante Ziffern (dezimal) : ");
i = 0;
while (von.dec_dgts[i] != '\0')
   {
   printf("%d", von.dec_dgts[i]);
   ++i;
   }
printf("\n\n");
}
```

```
Zahl vom Typ long              : -123456
Entsprechende decimal-Struktur :
Exponent                       : 3
Vorzeichen                     : 0
# signifikante Ziffern        : 3
signifikante Ziffern (dezimal) : 123456

Kopierte decimal-Struktur :
Exponent                       : 3
Vorzeichen                     : 0
# signifikante Ziffern        : 3
signifikante Ziffern (dezimal) : 123456
```

deccvasc CHAR in DECIMAL umwandeln

deccvasc wandelt einen abdruckbaren Wert vom Datentyp CHAR in eine Zahl vom Datentyp DECIMAL um. Führende Leerzeichen werden unterdrückt. Der CHAR-Wert kann ein Vorzeichen, einen Dezimalpunkt und Zahlen rechts davon haben. Auch ein Exponent (e oder E) kann enthalten sein. Der Exponent kann ein Vorzeichen besitzen.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden.

```
#include <decimal.h>

int deccvasc(cp, laenge, np)
    char *cp;
    int laenge;
    dec_t *np;
```

→ cp
ist der Zeiger auf die Zeichenkette, die umgewandelt werden soll.

→ laenge
gibt die Länge der Zeichenkette an (inklusive führender Leerzeichen). Geben Sie hier eine kürzere Länge an, als die Zeichenkette lang ist, wird nur die Anzahl laenge an Zeichen übertragen, wobei führende Leerzeichen mitzählen, obwohl sie in der Zahl unterdrückt werden.

← np
Zeiger auf die dec_t-Struktur, in die das Ergebnis der Umwandlung geschrieben werden soll.

Returnwert

0	Die Funktion war erfolgreich
-1200	Die Zahl ist zu groß für den Typ DECIMAL
-1201	Die Zahl ist zu klein für den Typ DECIMAL
-1213	Die Zeichenkette enthält nicht-numerische Zeichen
-1216	Exponent falsch

Beispiel

```
#include <decimal.h>

main()

{
dec_t dezimal_zahl;
int fehler;
int i;

printf("Zahl vom Typ char : '12345.67'\n");

fehler = deccvasc("12345.67", 8, &dezimal_zahl);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }

printf("Zahl vom Typ decimal :\n");
printf("Exponent                          : %hd\n", dezimal_zahl.dec_exp);
printf("Vorzeichen                        : %hd\n", dezimal_zahl.dec_pos);
printf("# signifikante Ziffern            : %hd\n", dezimal_zahl.dec_ndgts);
printf("signifikante Ziffern (dezimal) : ");
i = 0;
while (dezimal_zahl.dec_dgts[i] != '\0')
   {
   printf("%d", dezimal_zahl.dec_dgts[i]);
   ++i;
   }
printf("\n");
}
```

```
Zahl vom Typ char : '12345.67'
Zahl vom Typ decimal :
Exponent                          : 3
Vorzeichen                        : 1
# signifikante Ziffern            : 4
signifikante Ziffern (dezimal) : 1234567
```

deccvdbl double in DECIMAL umwandeln

deccvdbl konvertiert eine Zahl vom C-Datentyp double in den Datentyp DECIMAL.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden.

```
#include <decimal.h>

int deccvdbl(dbl, np)
     double dbl;
     dec_t *np;
```

→ dbl
ist die zu konvertierende Zahl vom Datentyp double.

← np
ist der Zeiger auf die dec_t-Struktur, in die das Ergebnis geschrieben werden soll.

Beispiel

```
#include <decimal.h>

main()

{
dec_t dezimal_zahl;
int i;

printf("Zahl vom Typ double : 3.14159\n");

deccvdbl(3.14159,  &dezimal_zahl);

printf("Zahl vom Typ decimal :\n");
printf("Exponent                        : %hd\n", dezimal_zahl.dec_exp);
printf("Vorzeichen                      : %hd\n", dezimal_zahl.dec_pos);
printf("# signifikante Ziffern          : %hd\n", dezimal_zahl.dec_ndgts);
printf("signifikante Ziffern (dezimal) : ");
i = 0;
while (dezimal_zahl.dec_dgts[i] != '\0')
   {
   printf("%d", dezimal_zahl.dec_dgts[i]);
   ++i;
   }
printf("\n");
}
```

```
Zahl vom Typ double : 3.14159
Zahl vom Typ decimal :
Exponent                        : 1
Vorzeichen                      : 1
# signifikante Ziffern          : 4
signifikante Ziffern (dezimal)  : 3141590
```

deccvint int in DECIMAL umwandeln

deccvint konvertiert eine Zahl vom C-Datentyp int in den Datentyp DECIMAL.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden.

```
#include <decimal.h>

int deccvint(integer, np)
     int integer;
     dec_t *np;
```

→ integer
ist die umzuwandelnde Zahl vom C-Datentyp int.

← np
ist der Zeiger auf die dec_t-Struktur, in die das Ergebnis geschrieben werden soll.

Beispiel

```
#include <decimal.h>

main()

{
dec_t dezimal_zahl;
int i;

printf("Zahl vom Typ int : 999\n");

deccvint(999, &dezimal_zahl);

printf("Zahl vom Typ decimal :\n");
printf("Exponent                        : %hd\n", dezimal_zahl.dec_exp);
printf("Vorzeichen                      : %hd\n", dezimal_zahl.dec_pos);
printf("# signifikante Ziffern          : %hd\n", dezimal_zahl.dec_ndgts);
printf("signifikante Ziffern (dezimal) : ");
i = 0;
while (dezimal_zahl.dec_dgts[i] != '\0')
   {
   printf("%d", dezimal_zahl.dec_dgts[i]);
   ++i;
   }
printf("\n");
}
```

```
Zahl vom Typ int : 999
Zahl vom Typ decimal :
Exponent                          : 2
Vorzeichen                        : 1
# signifikante Ziffern            : 2
signifikante Ziffern (dezimal)    : 999
```

deccvlong long in DECIMAL umwandeln

deccvlong konvertiert eine Zahl vom C-Datentyp long in eine Zahl in den Datentyp DECIMAL.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden.

```
#include <decimal.h>

int deccvlong(lang, np)
     long lang;
     dec_t *np;
```

→ lang
ist die zu konvertierende Zahlvom C-Datentyp long.

← np
ist der Zeiger auf die dec_t-Struktur, in die das Ergebnis geschrieben werden soll.

Beispiel

```
#include <decimal.h>

main()

{
dec_t dezimal_zahl;
int i;

printf("Zahl vom Typ long : 123456\n");

deccvlong(123456L, &dezimal_zahl);

printf("Zahl vom Typ decimal :\n");
printf("Exponent                        : %hd\n", dezimal_zahl.dec_exp);
printf("Vorzeichen                      : %hd\n", dezimal_zahl.dec_pos);
printf("# signifikante Ziffern          : %hd\n", dezimal_zahl.dec_ndgts);
printf("signifikante Ziffern (dezimal) : ");
i = 0;
while (dezimal_zahl.dec_dgts[i] != '\0')
   {
   printf("%d", dezimal_zahl.dec_dgts[i]);
   ++i;
   }
printf("\n");
}
```

```
Zahl vom Typ long : 123456
Zahl vom Typ decimal :
Exponent                          : 3
Vorzeichen                        : 1
# signifikante Ziffern            : 3
signifikante Ziffern (dezimal)    : 123456
```

decdiv Zwei Zahlen vom Datentyp DECIMAL dividieren

decdiv dividiert n1 durch n2 und gibt das Ergebnis nach erg aus.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden.

```
#include <decimal.h>

int decdiv(n1, n2, erg)          /* erg = n1 / n2 */
     dec_t *n1;
     dec_t *n2;
     dec_t *erg;
```

→ n1, n2
sind Zeiger auf die Eingabewerte vom Datentyp DECIMAL.

← erg
ist der Zeiger auf die dec_t-Struktur, die das Rechenergebnis aufnimmt. erg darf auch mit n1 oder n2 übereinstimmen.

Returnwert

0	Die Funktion war erfolgreich
-1200	Das Ergebnis ist zu groß für den Typ DECIMAL (Überlauf)
-1201	Das Ergebnis ist zu klein für den Typ DECIMAL (Unterlauf)
-1202	n2 hat den Wert 0.

Beispiel

```
#include <decimal.h>

main()

{
dec_t dezimal_zahl_1;
dec_t dezimal_zahl_2;
dec_t dezimal_zahl_3;
double double_zahl;
int fehler;

printf("Divident : 44.321\n");
deccvasc("44.321", 80, &dezimal_zahl_1);

printf("Divisor  : 25\n");
deccvlong(25, &dezimal_zahl_2);
```

```
fehler = decdiv(&dezimal_zahl_1, &dezimal_zahl_2, &dezimal_zahl_3);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n", fehler);
   exit(1);
   }

dectodbl(&dezimal_zahl_3, &double_zahl);
printf("Quotient : %f\n", double_zahl);
}
```

```
Divident : 44.321
Divisor  : 25
Quotient : 1.772840
```

dececvt DECIMAL in Zeichenkette umwandeln

dececvt wandelt eine Zahl vom Datentyp DECIMAL in eine mit dem Nullbyte abgeschlossene Zeichenkette um und liefert einen Zeiger auf den Anfang der Zeichenkette.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden.

```
#include <decimal.h>

char *dececvt(np, nstellen, dezimal_pkt, vorzeichen)
     dec_t *np;
     int nstellen;
     int *dezimal_pkt;
     int *vorzeichen;
```

→ np
ist der Zeiger auf die Zahl vom Datentyp DECIMAL, die umgewandelt werden soll.

→ nstellen
bestimmt die Anzahl der auszugebenen Zeichen der gesamten Zeichenkette. Das letzte Zeichen wird gerundet.

← dezimal_pkt
ist der Zeiger auf eine Zahl vom Datentyp int, die die Stellung des Dezimalpunktes bezogen auf den Anfang der Zeichenkette angibt. Ein negativer Wert drückt aus, wie weit ein gedachter Dezimalpunkt links vor dem gelieferten Wert steht.

← vorzeichen
ist der Zeiger auf das Vorzeichen.
vorzeichen ist 0, wenn die umgewandelte Zahl größer oder gleich 0 ist.

Returnwert

Die Ergebnis-Zeichenkette ist der Returnwert.

Hinweis

Bei der Umwandlung wird die niedrigwertigste Stelle gerundet.

Beispiel

```
#include <decimal.h>

char *dececvt();
char *decfcvt();

main()

{
dec_t dezimal_zahl;
int nstellen;
int dez_pkt;
int vorzeichen;

printf("-123.45, Zahl vom Typ double wird in decimal-Struktur");
printf(" umgewandelt\n\n");
deccvdbl(-123.45, &dezimal_zahl);

nstellen = 4;
printf("Zeichenkette mit Zeichenanzahl 4 : ");
printf("%s\n", dececvt(&dezimal_zahl, nstellen, &dez_pkt, &vorzeichen));
printf("Dezimalpunkt : %d\n", dez_pkt);
printf("Vorzeichen   : %d\n\n", vorzeichen);

printf("zum Vergleich\n");
printf("Zeichenkette mit Zeichenanzahl 4 rechts vom Dezimalpunkt : ");
printf("%s\n", decfcvt(&dezimal_zahl, 4, &dez_pkt, &vorzeichen));
printf("Dezimalpunkt : %d\n", dez_pkt);
printf("Vorzeichen   : %d\n\n", vorzeichen);
}
```

```
-123.45, Zahl vom Typ double wird in decimal-Struktur umgewandelt

Zeichenkette mit Zeichenanzahl 4 : 1235
Dezimalpunkt : 3
Vorzeichen   : 1

zum Vergleich
Zeichenkette mit Zeichenanzahl 4 rechts vom Dezimalpunkt : 1234500
Dezimalpunkt : 3
Vorzeichen   : 1
```

decfcvt DECIMAL in Zeichenkette umwandeln

decfcvt wandelt eine Zahl vom Datentyp DECIMAL in eine mit dem Nullbyte abgeschlossene Zeichenkette um und liefert einen Zeiger auf den Anfang der Zeichenkette.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden mit der Anweisung:

```
#include <decimal.h>

char *decfcvt(np, nstellen, dezimal_pkt, vorzeichen)
     dec_t *np;
     int nstellen;
     int *dezimal_pkt;
     int *vorzeichen;
```

Die Funktionen dececvt und decfcvt sind bis auf die Bedeutung des Parameters nstellen identisch.

→ np
ist der Zeiger auf die dec_t-Struktur, die umgewandelt werden soll.

→ nstellen
bestimmt die Anzahl der auszugebenen Zeichen: Bei decfcvt bestimmt nstellen die Zeichenanzahl rechts vom Dezimalpunkt. Das letzte Zeichen wird gerundet.

← dezimal_pkt
zeigt auf eine Zahl vom Datentyp int, die die Stellung des Dezimalpunktes bezogen auf den Anfang der Zeichenkette angibt. Ein negativer Wert drückt aus, wie weit ein gedachter Dezimalpunkt links vor dem gelieferten Wert steht.

← vorzeichen
ist ein Zeiger auf das Vorzeichen.
vorzeichen ist 0, wenn die umgewandelte Zahl größer oder gleich 0 ist.

Returnwert

Die Ergenis-Zeichenkette ist der Returnwert.

Hinweis

Bei der Umwandlung wird die niedrigwertigste Stelle gerundet.

Beispiel

```
#include <decimal.h>

char *dececvt();
char *decfcvt();

main()

{
dec_t dezimal_zahl;
int nstellen;
int dez_pkt;
int vorzeichen;

printf("0.0666, Zahl vom Typ double wird in decimal-Struktur");
printf(" umgewandelt\n\n");
deccvdbl(0.0666, &dezimal_zahl);

printf("Zeichenkette mit Zeichenanzahl 4 rechts vom Dezimalpunkt : ");
printf("%s\n", decfcvt(&dezimal_zahl, 4, &dez_pkt, &vorzeichen));
printf("Dezimalpunkt : %d\n", dez_pkt);
printf("Vorzeichen   : %d\n\n", vorzeichen);

printf("zum Vergleich\n");
nstellen = 4;
printf("Zeichenkette mit Zeichenanzahl 4 : ");
printf("%s\n", dececvt(&dezimal_zahl, nstellen, &dez_pkt, &vorzeichen));
printf("Dezimalpunkt : %d\n", dez_pkt);
printf("Vorzeichen   : %d\n\n", vorzeichen);
}
```

```
0.0666, Zahl vom Typ double wird in decimal-Struktur umgewandelt

Zeichenkette mit Zeichenanzahl 4 rechts vom Dezimalpunkt : 666
Dezimalpunkt : -1
Vorzeichen   : 0

zum Vergleich
Zeichenkette mit Zeichenanzahl 4 : 6660
Dezimalpunkt : -1
Vorzeichen   : 0
```

decmul Zwei Zahlen multiplizieren

decmul multipliziert zwei Zahlen vom Datentyp DECIMAL und gibt das Ergebnis nach erg aus.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden.

```
#include <decimal.h>

int decmul(n1, n2, erg)          /* erg = n1 * n2 */
     dec_t *n1;
     dec_t *n2;
     dec_t *erg;
```

→ n1, n2
sind Zeiger auf die Eingabewerte vom Typ DECIMAL.

← erg
ist der Zeiger auf die dec_t-Struktur, die das Rechenergebnis aufnimmt. erg darf auch mit n1 oder n2 übereinstimmen.

Returnwert

0	Die Funktion war erfolgreich
-1200	Das Ergebnis ist zu groß für den Typ decimal (Überlauf)
-1201	Das Ergebnis ist zu klein für den Typ decimal (Unterlauf)

Beispiel

```
#include <decimal.h>

main()

{
dec_t dezimal_zahl_1;
dec_t dezimal_zahl_2;
dec_t dezimal_zahl_3;
double double_zahl;
int fehler;

printf("1. Faktor : 44.321\n");
deccvasc("44.321", 80, &dezimal_zahl_1);

printf("2. Faktor : 25\n");
deccvlong(25, &dezimal_zahl_2);

fehler = decmul(&dezimal_zahl_1, &dezimal_zahl_2, &dezimal_zahl_3);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n", fehler);
   exit(1);
   }

dectodbl(&dezimal_zahl_3, &double_zahl);
printf("Produkt    : %f\n", double_zahl);
}
```

```
------------------------------------------------------------------------

1. Faktor : 44.321
2. Faktor : 25
Produkt   : 1108.025000
```

decround Zahl vom Datentyp DECIMAL runden

decround rundet eine Zahl vom Datentyp DECIMAL auf die angegebenen Stellen hinter dem Komma. Der Rundungsfaktor ist $5 \times 10^{-s-1}$. Das Runden wird erreicht, indem zu einer positiven Zahl der Rundungsfaktor addiert wird; bei einer negativen Zahl wird er subtrahiert. Dann wird auf s Zeichen abgeschnitten.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden.

```
#include <decimal.h>

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

↔ d
ist der Zeiger auf die dec_t-Struktur, deren Wert gerundet werden soll.

→ s
ist die Anzahl an Stellen hinter dem Komma, auf die gerundet werden soll.

Beispiel

```
#include <decimal.h>

main()

{
dec_t dezimal_zahl;
int s;
double double_zahl;

printf("Wert          s             Wert gerundet\n");
printf("-------------------------------------------\n");

s = 0;
deccvdbl(1.4, &dezimal_zahl);
decround(&dezimal_zahl, s);
dectodbl(&dezimal_zahl, &double_zahl);
printf("1.4           %d             %f\n", s, double_zahl);

deccvdbl(-1.4, &dezimal_zahl);
decround(&dezimal_zahl, s);
```

```
dectodbl(&dezimal_zahl, &double_zahl);
printf("-1.4         %d            %f\n", s, double_zahl);

s = 2;
deccvdbl(1.684, &dezimal_zahl);
decround(&dezimal_zahl, s);
dectodbl(&dezimal_zahl, &double_zahl);
printf("1.684        %d            %f\n", s, double_zahl);

deccvdbl(1.685, &dezimal_zahl);
decround(&dezimal_zahl, s);
dectodbl(&dezimal_zahl, &double_zahl);
printf("1.685        %d            %f\n", s, double_zahl);

decround(&dezimal_zahl, 1);
dectodbl(&dezimal_zahl, &double_zahl);
printf("1.685        1             %f\n", double_zahl);

decround(&dezimal_zahl, 0);
dectodbl(&dezimal_zahl, &double_zahl);
printf("1.685        0             %f\n", double_zahl);
}
```

```
------------------------------------------------------------------------
```

Wert	s	Wert gerundet
1.4	0	1.000000
-1.4	0	-1.000000
1.684	2	1.680000
1.685	2	1.690000
1.685	1	1.700000
1.685	0	2.000000

decsub Zwei Zahlen vom Datentyp DECIMAL subtrahieren

decsub subtrahiert zwei Zahlen vom Datentyp DECIMAL voneinander und gibt das Ergebnis nach erg aus.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden.

```
#include <decimal.h>

int decsub(n1, n2, erg)          /* erg = n1 - n2 */
      dec_t *n1;
      dec_t *n2;
      dec_t *erg;
```

-→ n1, n2
sind Zeiger auf die Eingabewerte vom Datentyp DECIMAL.

←- erg
ist der Zeiger auf die dec_t-Struktur, die das Rechenergebnis aufnimmt. erg darf auch mit n1 oder n2 übereinstimmen.

Returnwert

0	Die Funktion war erfolgreich
-1200	Das Ergebnis ist zu groß für den Typ DECIMAL
-1201	Das Ergebnis ist zu klein für den Typ DECIMAL

Beispiel

```
#include <decimal.h>

main()

{
dec_t dezimal_zahl_1;
dec_t dezimal_zahl_2;
dec_t dezimal_zahl_3;
double double_zahl;
int fehler;

printf("Minuend    : 44.321\n");
deccvasc("44.321", 80, &dezimal_zahl_1);

printf("Subtrahent : 25\n");
deccvlong(25, &dezimal_zahl_2);
```

```
fehler = decsub(&dezimal_zahl_1, &dezimal_zahl_2, &dezimal_zahl_3);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n", fehler);
   exit(1);
   }

dectodbl(&dezimal_zahl_3, &double_zahl);
printf("Differenz  : %f\n", double_zahl);
}

------------------------------------------------------------------------

Minuend    : 44.321
Subtrahent : 25
Differenz  : 19.321000
```

dectoasc DECIMAL in CHAR umwandeln

dectoasc wandelt den Inhalt einer dec_t-Struktur in eine abdruckbare Zeichenkette um. Die Zeichenkette wird ggf. nach rechts mit Leerzeichen aufgefüllt.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden.

```
#include <decimal.h>

int dectoasc(np, cp, laenge, rechts)
      dec_t *np;
      char *cp;
      int laenge;
      int rechts;
```

→ np
ist der Zeiger auf die dec_t-Struktur, deren Inhalt in eine Zeichenkette umgewandelt werden soll.

← cp
ist die Anfangsadresse der Zeichenkette, die die umgewandelte Zahl aufnehmen soll.

→ laenge
gibt die signifikante Länge der Zeichenkette an.

→ rechts
gibt die Anzahl der Kommastellen an.
Wenn rechts den Wert -1 hat, dann bestimmt die dec_t-Struktur die Anzahl.

Returnwert

0 Funktion war erfolgreich
-1 Funktion war nicht erfolgreich; dies ist z.B. der Fall, wenn die Zahl nicht in die Zeichenkette paßt.

Beispiel

```
#include <decimal.h>

main()

{
char str[80];
dec_t dezimal_zahl;
int fehler;
int i;

printf("Zahl vom Typ char : 12345.67\n\n");

deccvasc("12345.67", 8, &dezimal_zahl);

printf("Zahl vom Typ decimal :\n");
printf("Exponent                          : %hd\n", dezimal_zahl.dec_exp);
printf("Vorzeichen                        : %hd\n", dezimal_zahl.dec_pos);
printf("# signifikante Ziffern            : %hd\n", dezimal_zahl.dec_ndgts);
printf("signifikante Ziffern (dezimal) : ");
i = 0;
while (dezimal_zahl.dec_dgts[i] != '\0')
   {
   printf("%d", dezimal_zahl.dec_dgts[i]);
   ++i;
   }
printf("\n\n");

fehler = dectoasc(&dezimal_zahl, str, 80, -1);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }

printf("Zahl vom Typ char war : %s\n", str);
}
```

```
------------------------------------------------------------------------

Zahl vom Typ char : 12345.67

Zahl vom Typ decimal :
Exponent                          : 3
Vorzeichen                        : 1
# signifikante Ziffern            : 4
signifikante Ziffern (dezimal) : 1234567

Zahl vom Typ char war : 12345.67
```

dectodbl DECIMAL in double umwandeln

dectodbl konvertiert eine Zahl vom Datentyp DECIMAL in den C-Datentyp double.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden.

```
#include <decimal.h>

int dectodbl(np, dbl_zeiger)
      dec_t  *np;
      double *dbl_zeiger;
```

→ np
ist der Zeiger auf die dec_t-Struktur, deren Wert in eine Zahl vom Datentyp double umgewandelt wird.

← dbl_zeiger
ist der Zeiger auf die Variable vom Datentyp double, die das Ergebnis der Umwandlung aufnehmen soll.

Beispiel

```
#include <decimal.h>

main()

{
dec_t dezimal_zahl;
double double_zahl;
int i;

printf("Zahl vom Typ double : 3.14159\n\n");

deccvdbl(3.14159,  &dezimal_zahl);

printf("Zahl vom Typ decimal :\n");
printf("Exponent                        : %hd\n", dezimal_zahl.dec_exp);
printf("Vorzeichen                      : %hd\n", dezimal_zahl.dec_pos);
printf("# signifikante Ziffern          : %hd\n", dezimal_zahl.dec_ndgts);
printf("signifikante Ziffern (dezimal) : ");
i = 0;
while (dezimal_zahl.dec_dgts[i] != '\0')
   {
   printf("%d", dezimal_zahl.dec_dgts[i]);
   ++i;
   }
printf("\n\n");
```

```
dectodbl(&dezimal_zahl, &double_zahl);

printf("Zahl vom Typ double war : %f\n", double_zahl);
}
```

```
Zahl vom Typ double : 3.14159

Zahl vom Typ decimal :
Exponent                        : 1
Vorzeichen                      : 1
# signifikante Ziffern          : 4
signifikante Ziffern (dezimal) : 3141590

Zahl vom Typ double war : 3.141590
```

dectoint DECIMAL in int umwandeln

dectoint konvertiert eine Zahl vom Datentyp DECIMAL in den C-Datentyp int.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden.

```
#include <decimal.h>

int dectoint(np, ip)
      dec_t *np;
      int *ip;
```

→ np
ist der Zeiger auf die dec_t-Struktur, deren Inhalt in eine Zahl vom C-Datentyp int umgewandelt wird.

← ip
ist der Zeiger auf die Variable vom C-Datentyp int, die das Ergebnis aufnehmen soll.

Returnwert

0	Funktion war erfolgreich.
-1200	Die DECIMAL-Zahl ist größer als 32767.

Beispiel

```
#include <decimal.h>

main()

{
dec_t dezimal_zahl;
int int_zahl;
int fehler;
int i;

printf("Zahl vom Typ int : 999\n\n");

deccvint(999, &dezimal_zahl);

printf("Zahl vom Typ decimal :\n");
printf("Exponent                    : %hd\n", dezimal_zahl.dec_exp);
printf("Vorzeichen                  : %hd\n", dezimal_zahl.dec_pos);
```

```
printf("# signifikante Ziffern         : %hd\n", dezimal_zahl.dec_ndgts);
printf("signifikante Ziffern (dezimal) : ");
i = 0;
while (dezimal_zahl.dec_dgts[i] != '\0')
   {
   printf("%d", dezimal_zahl.dec_dgts[i]);
   ++i;
   }
printf("\n\n");

fehler = dectoint(&dezimal_zahl, &int_zahl);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }

printf("Zahl vom Typ int war : %d\n", int_zahl);
}
```

```
Zahl vom Typ int : 999

Zahl vom Typ decimal :
Exponent                       : 2
Vorzeichen                     : 1
# signifikante Ziffern         : 2
signifikante Ziffern (dezimal) : 999

Zahl vom Typ int war : 999
```

dectolong DECIMAL in long umwandeln

dectolong konvertiert eine Zahl vom Datentyp DECIMAL in den C-Datentyp long.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden.

```
#include <decimal.h>

int dectolong(np, lang_zeiger)
     dec_t *np;
     long *lang_zeiger;
```

→ np
ist der Zeiger auf die dec_t-Struktur, deren Inhalt in eine Zahl vom C-Datentyp long umgewandelt wird.

← lang_zeiger
ist der Zeiger auf eine Variable vom C-Datentyp long, die das Ergebnis der Umwandlung aufnehmen soll.

Returnwert

0	Die Funktion war erfolgreich
-1200	Die DECIMAL-Zahl ist größer als 2 147 483 647

Beispiel

```
#include <decimal.h>

main()

{
dec_t dezimal_zahl;
long long_zahl;
int fehler;
int i;

printf("Zahl vom Typ long : 123456\n\n");

deccvlong(123456L, &dezimal_zahl);

printf("Zahl vom Typ decimal :\n");
printf("Exponent                    : %hd\n", dezimal_zahl.dec_exp);
printf("Vorzeichen                  : %hd\n", dezimal_zahl.dec_pos);
```

```
printf("# signifikante Ziffern          : %hd\n", dezimal_zahl.dec_ndgts);
printf("signifikante Ziffern (dezimal) : ");
i = 0;
while (dezimal_zahl.dec_dgts[i] != '\0')
   {
   printf("%d", dezimal_zahl.dec_dgts[i]);
   ++i;
   }
printf("\n\n");

fehler = dectolong(&dezimal_zahl, &long_zahl);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }

printf("Zahl vom Typ long war : %ld\n", long_zahl);
}
```

```
Zahl vom Typ long : 123456

Zahl vom Typ decimal :
Exponent                       : 3
Vorzeichen                     : 1
# signifikante Ziffern         : 3
signifikante Ziffern (dezimal) : 123456

Zahl vom Typ long war : 123456
```

dectrunc DECIMAL-Zahl abschneiden

dectrunc schneidet eine Zahl vom Datentyp DECIMAL auf die vorgegebene Anzahl an Stellen hinter dem Komma ab, nachdem die Zahl gerundet wurde.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dec_t-Strukturen die Include-Datei decimal.h in Ihr Programm einbinden.

```
#include <decimal.h>

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

↔ d
ist der Zeiger auf eine dec_t Struktur, die die DECIMAL-Zahl darstellt, deren Stellen abgeschnitten werden sollen.

→ s
bezeichnet die Anzahl an Stellen hinter dem Komma, auf die die Zahl abgeschnitten werden soll.

Beispiel

```
#include <decimal.h>

main()

{
dec_t dezimal_zahl;
int s;
double dbl_round, dbl_trunc;

printf("ungerundet   s        gerundet     abgeschnitten\n");
printf("-------------------------------------------------------\n");

s = 0;
deccvdbl(1.4, &dezimal_zahl);
decround(&dezimal_zahl, s);
dectodbl(&dezimal_zahl, &dbl_round);
dectrunc(&dezimal_zahl, s);
dectodbl(&dezimal_zahl, &dbl_trunc);
printf("1.4          %d        %8.3f %17.3f\n", s, dbl_round, dbl_trunc);

deccvdbl(-1.4, &dezimal_zahl);
decround(&dezimal_zahl, s);
dectodbl(&dezimal_zahl, &dbl_round);
dectrunc(&dezimal_zahl, s);
```

```
dectodbl(&dezimal_zahl, &dbl_trunc);
printf("-1.4         %d        %8.3f %17.3f\n", s, dbl_round, dbl_trunc);

s = 2;
deccvdbl(1.684, &dezimal_zahl);
decround(&dezimal_zahl, s);
dectodbl(&dezimal_zahl, &dbl_round);
dectrunc(&dezimal_zahl, s);
dectodbl(&dezimal_zahl, &dbl_trunc);
printf("1.684        %d        %8.3f %17.3f\n", s, dbl_round, dbl_trunc);

deccvdbl(1.685, &dezimal_zahl);
decround(&dezimal_zahl, s);
dectodbl(&dezimal_zahl, &dbl_round);
dectrunc(&dezimal_zahl, s);
dectodbl(&dezimal_zahl, &dbl_trunc);
printf("1.685        %d        %8.3f %17.3f\n", s, dbl_round, dbl_trunc);

decround(&dezimal_zahl, 1);
dectodbl(&dezimal_zahl, &dbl_round);
dectrunc(&dezimal_zahl, 1);
dectodbl(&dezimal_zahl, &dbl_trunc);
printf("1.685        1        %8.3f %17.3f\n", dbl_round, dbl_trunc);

decround(&dezimal_zahl, 0);
dectodbl(&dezimal_zahl, &dbl_round);
dectrunc(&dezimal_zahl, 0);
dectodbl(&dezimal_zahl, &dbl_trunc);
printf("1.685        0        %8.3f %17.3f\n", dbl_round, dbl_trunc);
}
```

---

ungerundet	s	gerundet	abgeschnitten
1.4	0	1.000	1.000
-1.4	0	-1.000	-1.000
1.684	2	1.680	1.680
1.685	2	1.690	1.690
1.685	1	1.700	1.700
1.685	0	2.000	2.000

dtcurrent Aktuelle Zeitangabe eingeben

dtcurrent weist die aktuelle Zeit und/oder das aktuelle Datum einer Variablen vom Datentyp DATETIME zu.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dtime_t-Strukturen die Include-Datei datetime.h in Ihr Programm einbinden.

Außerdem müssen Sie die Datumskomponenten initialisieren, indem Sie den Makro TU_DTENCODE aufrufen.

```
#include <datetime.h>

int dtcurrent(d)
    dtime_t *d;
```

← d
ist der Zeiger auf eine dtime_t-Struktur, für die Sie den Speicherplatz zur Verfügung stellen müssen.

Beispiel

```
#include <datetime.h>

main()

{
$datetime year to day zeit;
char str[26];

zeit.dt_qual = TU_DTENCODE(TU_YEAR, TU_DAY);
dtcurrent(&zeit);

zeit.dt_qual = TU_DTENCODE(TU_YEAR, TU_DAY);
dttoasc(&zeit, str);
printf("Datum : %s\n", str);
}
```

```
Datum : 1990-03-27
```

dtcvasc CHAR in DATETIME konvertieren

dtcvasc konvertiert eine Variable vom C-Datentyp CHAR in eine Variable vom Datentyp DATETIME.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dtime_t-Strukturen die Include-Datei datetime.h in Ihr Programm einbinden.

Außerdem müssen Sie den Macro TU_DTENCODE aus dieser Include-Datei mit den gewünschten Parametern aufrufen, um damit die Datums-Komponenten zu initialisieren.

```
#include <datetime.h>

int dtcvasc(str,d)
     char     *str;
     dtime_t *d;
```

→ str
ist der Zeiger auf eine Zeichenkette. Die Zeichenkette kann führende und angehängte Leerzeichen enthalten. Vom ersten signifikanten Zeichen angefangen, darf sie jedoch nur noch Zeichen und Begrenzer enthalten, die den Datumskomponenten entsprechen. Wenn ein Jahr mit ein oder zwei Zeichen angegeben ist, wird 1900 addiert.

← d
ist der Zeiger auf eine dtime_t Hostvariable, für die Sie den Speicherplatz zur Verfügung stellen müssen.

Returnwert

Ist die Zeichenkette übertragbar, so wird der Wert der Variablen zugewiesen und die Funktion hat den Returnwert 0. Andernfalls wird die Variable nicht geändert und die Funktion hat einen der folgenden negativen Werte als Returnwert:

- 1260 eine Konversion zwischen den angegebenen Datentypen ist nicht möglich.
- 1261 es sind zuviele Zeichen im ersten Feld von DATETIME oder INTERVAL.

- 1262 ein nicht-numerisches Zeichen in DATETIME oder INTERVAL.
- 1263 der Wert einer Datumskomponente ist außerhalb des Wertebereichs.
- 1264 es bleiben Zeichen am Ende der DATETIME-Variable übrig.
- 1265 Überlauf bei einer DATETIME oder INTERVAL-Operation
- 1266 INTERVAL oder DATETIME ist unverträglich mit dieser Operation
- 1267 das Ergebnis einer DATETIME-Berechnung ist außerhalb des Wertebereichs
- 1268 ungültige Datumskomponente.

Beispiel

```
#include <datetime.h>

main()

{
$datetime year to second zeit;
char *str_ein;
char str_aus[26];
int fehler;

str_ein = "  90-03-28 14:00:05    ";

zeit.dt_qual = TU_DTENCODE(TU_YEAR, TU_SECOND);
fehler = dtcvasc(str_ein, &zeit);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }

zeit.dt_qual = TU_DTENCODE(TU_YEAR, TU_SECOND);
dttoasc(&zeit, str_aus);
printf("Datum und Uhrzeit : %s\n\n", str_aus);

zeit.dt_qual = TU_DTENCODE(TU_YEAR, TU_DAY);
fehler = dtcvasc("90-03-28 16:00:00", &zeit);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }

zeit.dt_qual = TU_DTENCODE(TU_YEAR, TU_DAY);
dttoasc(&zeit, str_aus);
printf("Datum und Uhrzeit : %s\n\n", str_aus);
}
```

```
Datum und Uhrzeit : 1990-03-28 14:00:05

Fehlernummer : -1264
```

dtextend Datumskomponenten kopieren

dtextend kopiert Datumskomponenten auf die Datumskomponenten einer anderen Variablen vom Datentyp DATETIME. Die Datumskomponenten der Zielvariablen bestimmen den Kopiervorgang.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dtime_t-Strukturen die Include-Datei datetime.h in Ihr Programm einbinden.

Außerdem müssen Sie den Macro TU_DTENCODE aus dieser Include-Datei mit den gewünschten Parametern aufrufen, um damit die Datums-Komponenten zu initialisieren.

```
#include <datetime.h>

int dtextend(id,od)
     dtime_t *id, *od;
```

→ id
ist der Zeiger auf die Variable, die kopiert werden soll. Komponenten, die in id nicht vorkommen, werden nicht beachtet.

← od
ist der Zeiger auf die Variable, in die kopiert werden soll. Komponenten in od, die in id nicht vorkommen, werden wie folgt aufgefüllt: Komponenten links von der höchstwertigen Komponente von id werden mit der aktuellen Zeit und dem aktuellen Datum aufgefüllt. Komponenten rechts von der niedrigstwertigen Komponente von id werden mit Nullen aufgefüllt.

Returnwert

-1268 ungültige Datumskomponente

Beispiel

```
#include <datetime.h>

main()

{
$datetime month to day von;
$datetime year to day nach;
char *str_ein;
char str_aus[26];
int fehler;

str_ein = "03-29";
printf("zu kopierendes Datum : %s\n", str_ein);

von.dt_qual = TU_DTENCODE(TU_MONTH, TU_DAY);
dtcvasc(str_ein, &von);

/* hier normalerweise Fehlerroutine */

nach.dt_qual = TU_DTENCODE(TU_YEAR, TU_DAY);
fehler = dtextend(&von, &nach);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }

nach.dt_qual = TU_DTENCODE(TU_YEAR, TU_DAY);
dttoasc(&nach, str_aus);
printf("kopiertes Datum : %s\n\n", str_aus);

printf("zu kopierendes Datum : 88-11-11\n");
von.dt_qual = TU_DTENCODE(TU_YEAR, TU_DAY);
dtcvasc("88-11-11", &von);

/* hier normalerweise Fehlerroutine */

nach.dt_qual = TU_DTENCODE(TU_YEAR, TU_DAY);
fehler = dtextend(&von, &nach);

/* hier normalerweise Fehlerroutine */

nach.dt_qual = TU_DTENCODE(TU_YEAR, TU_DAY);
dttoasc(&nach, str_aus);
printf("kopiertes Datum : %s\n\n", str_aus);
}
```

```
zu kopierendes Datum : 03-29
kopiertes Datum : 1990-03-29

zu kopierendes Datum : 88-11-11
kopiertes Datum : 1988-11-11
```

dttoasc DATETIME in eine Zeichenkette umwandeln

dttoasc konvertiert die Datumskomponenten einer Variable vom Datentyp DATETIME in eine Zeichenkette. Dabei werden die Begrenzer mitausgegeben.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der dtime_t-Strukturen die Include-Datei datetime.h in Ihr Programm einbinden.

Außerdem müssen Sie den Macro TU_DTENCODE aus dieser Include-Datei mit den gewünschten Parametern aufrufen.

```
#include <datetime.h>

int dttoasc(d,str)
      dtime_t *d;
      char *str;
```

→ d
ist der Zeiger auf eine dtime_t-Struktur, für die Sie vor dem Aufruf Speicherplatz zur Verfügung stellen müssen. Zeigt der übergebene Zeiger nicht auf eine belegte Struktur, gibt die Funktion einen undefinierten Wert von maximal 26 Bytes in str.

← str
ist der Zeiger auf eine Zeichenkette. Der Speicherbedarf für die Zeichenkette berechnet sich wie folgt:
1 Byte für jeden Begrenzer 4 Zeichen für year 2 Zeichen für month, day, hour, minute und second für fraction soviele Zeichen, wie für die Genauigkeit angegeben.

Die maximale Länge für str ergibt sich durch eine DATETIME-Variable, die qualifiziert ist mit year to fraction(5). Sie enthält 19 Zeichen, 6 Begrenzer und das abschließende Nullbyte, als Summe also 26 Bytes.

Beispiel

Siehe Beispiele zu den Bibliotheksfunktionen "dtcurrent", "dtcvasc" und "dtextend".

incvasc Zeichenkette in INTERVAL umwandeln

incvasc konvertiert eine Zeichenkette in eine Variable vom Datentyp INTERVAL.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der intrvl_t-Strukturen die Include-Datei datetime.h in Ihr Programm einbinden.
Außerdem müssen Sie den Macro TU_IENCODE aus dieser Include-Datei mit den gewünschten Parametern aufrufen.

```
#include <datetime.h>

int incvasc(str,i)
     char *str;
     intrvl_t *i;
```

→ str
ist der Zeiger auf die Zeichenkette. str kann führende oder angehängte Leerzeichen haben. Die Funktion ignoriert die Leerzeichen; sie berücksichtigt nur Zeichen und Begrenzer, die für die Datumskomponenten geeignet sind.

← i
ist der Zeiger auf eine intrvl_t-Strukur, für die Sie vor dem Aufruf Speicherplatz zur Verfügung stellen müssen.

Returnwert

Ist die Funktion erfolgreich, so ist der Returnwert 0. Im Fehlerfall wird i nicht verändert und folgender Fehlercode ausgegeben:

- 1260 eine Konvertierung zwischen den angegebenen Datentypen ist nicht möglich.
- 1261 es sind zuviele Zeichen im ersten Feld von DATETIME oder INTERVAL.
- 1262 ein nicht-numerisches Zeichen in DATETIME oder INTERVAL.
- 1263 der Wert einer Datumskomponente ist außerhalb des Wertebereichs.
- 1264 es bleiben Zeichen am Ende der DATETIME-Variable übrig
- 1265 Überlauf bei einer DATETIME oder INTERVAL-Operation
- 1266 INTERVAL oder DATETIME ist unverträglich mit dieser Operation

— 1267 das Ergebnis einer DATETIME-Berechnung ist außerhalb des Wertebereichs
— 1268 ungültige Datumskomponente.

Beispiel

```
#include <datetime.h>

main()

{
$interval hour to second spanne;
char *str_ein;
char str_aus[26];
int fehler;

str_ein = "15:00:05";
printf("eingegebene Zeitspanne     : %s\n", str_ein);

spanne.in_qual = TU_IENCODE(2, TU_HOUR, TU_SECOND);
fehler = incvasc(str_ein, &spanne);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }

spanne.in_qual = TU_IENCODE(2, TU_HOUR, TU_SECOND);
intoasc(&spanne, str_aus);
printf("eingegebene Zeitspanne war :%s\n", str_aus);
printf("(Genauigkeit fuer das erste Feld : 2)\n\n");

spanne.in_qual = TU_IENCODE(1, TU_HOUR, TU_SECOND);
intoasc(&spanne, str_aus);
printf("eingegebene Zeitspanne war : %s\n", str_aus);
printf("(Genauigkeit fuer das erste Feld : 1)\n\n\n");

printf("eingegebene Zeitspanne     : 11:55:00\n");

spanne.in_qual = TU_IENCODE(1, TU_HOUR, TU_SECOND);
fehler = incvasc("11:55:00", &spanne);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }

spanne.in_qual = TU_DTENCODE(2, TU_DAY, TU_SECOND);
intoasc(&spanne, str_aus);
printf("eingegebene Zeitspanne war : %s\n\n", str_aus);
}
```

```
eingegebene Zeitspanne     : 15:00:05
eingegebene Zeitspanne war : 15:00:05
(Genauigkeit fuer das erste Feld : 2)

eingegebene Zeitspanne war :  5:00:05
(Genauigkeit fuer das erste Feld : 1)


eingegebene Zeitspanne     : 11:55:00
Fehlernummer : -1261
```

intoasc INTERVAL in eine Zeichenkette umwandeln

intoasc konvertiert die Datumskomponenten einer Variablen vom Datentyp INTERVAL in eine Zeichenkette.

Vor Aufruf der Funktion

Bevor Sie diese Funktion aufrufen, müssen Sie zur Definition und zur Versorgung der intrvl_t-Strukturen die Include-Datei datetime.h in Ihr Programm einbinden.
Außerdem müssen Sie den Macro TU_IENCODE aus dieser Include-Datei mit den gewünschten Parametern aufrufen.

```
#include <datetime.h>

int intoasc(i,str)
      intrvl_t *i;
      char     *str;
```

→ i
ist der Zeiger auf eine intrvl_t-Struktur, für die Sie vor dem Aufruf Speicherplatz zur Verfügung stellen müssen. Die Zeichen der Datumskomponenten und die Begrenzer dazwischen werden konvertiert. Zeigt der übergebene Zeiger nicht auf eine belegte intrvl_t-Struktur speichert die Funktion einen undefinierten Wert von maximal 21 Bytes in str.

← str
ist der Zeiger auf eine Zeichenkette. Der Speicherbedarf für die Zeichenkette berechnet sich wie folgt:

1 Byte für jeden Begrenzer 4 Zeichen für year 2 Zeichen für month, day, hour, minute und second für fraction soviele Zeichen, wie für die Genauigkeit angegeben.

Die maximale Länge für str ergibt sich durch eine INTERVAL-Variable, die qualifiziert ist mit year to fraction(5). Sie enthält 19 Zeichen, 6 Begrenzer und das abschließende Nullbyte, als Summe also 26 Bytes.

Beispiel

Siehe Beispiel zu der Bibliotheksfunktion incvasc.

ldchar Zeichenfolge kopieren

ldchar kopiert eine Zeichenfolge fester Länge in eine mit Nullbyte abgeschlossene Zeichenkette. Dabei werden ggf. abschliessende Leerzeichen nicht kopiert.

```
int ldchar(from, count, to)
      char *from;
      char *to;
      int count;
```

→ from
ist der Zeiger auf die zu kopierende Zeichenfolge.

→ count
ist die Länge der zu kopierenden Zeichenfolge.

← to
ist der Zeiger auf eine Ziel-Zeichenkette.

Hinweis

Die Zeiger from und to können auf dieselbe Speicheradresse zeigen. Sie müssen darauf achten, daß für die Zeichenkette genügend Speicherplatz zur Verfügung steht, nämlich count Zeichen + Nullbyte.

Beispiel

```
main()

{
$fixchar *str1;
char str2[80];
int lge;
int i;

str1 = "Zeichenfolge fester Laenge   ";
lge = 26;
printf("Urspruengliche Zeichenfolge : %s\n", str1);

ldchar(str1, lge, &str2);
printf("Kopierte Zeichenkette       : ");
i = 0;
while (str2[i] != '\0')
   printf("%c", str2[i++]);
printf("\n");

ldchar(str1, 12, &str2);
printf("Kopierte Zeichenkette       : ");
i = 0;
while (str2[i] != '\0')
```

```
   printf("%c", str2[i++]);

printf("\n");
}
```

```
------------------------------------------------------------------------

Urspruengliche Zeichenfolge : Zeichenfolge fester Laenge
Kopierte Zeichenkette       : Zeichenfolge fester Laenge
Kopierte Zeichenkette       : Zeichenfolge
```

rdatestr Datum als Zeichenkette darstellen

rdatestr konvertiert ein Datum in eine Zeichenkette. Die interne numerische Darstellung wird in das Format umgewandelt, das Sie in der Umgebungsvariablen DBDATE festlegen. Lesen Sie dazu Kapitel 5 Umgebungsvariable. Der voreingestellte Wert ist mm/tt/jjjj.

```
int rdatestr(jdate,str)
        long jdate;
        char *str;
```

→ jdate
ist das interne Format einer Variablen vom Datentyp DATE.

← str
ist der Zeiger auf eine Zeichenkette, die mindestens 12 Zeichen aufnehmen kann. Hier steht nach Ausführung der Funktion das lesbare Datum im Format, das Sie in der Umgebungsvariablen DBDATE festgelegt haben. Der voreingestellte Wert ist mm/tt/jjjj.

Beispiel

```
main()

{
long datum;
char str[12];

printf("Interne Datumsdarstellung : 32944\n");

rdatestr(32944L, str);

printf("Datum : %s\n", str);
}
```

```
Interne Datumsdarstellung : 32944
Datum : 13.03.1990
```

rdayofweek Wochentag ausgeben

rdayofweek errechnet den Wochentag eines Datums, das in internem Format gegeben ist.

```
int rdayofweek(jdate)
        long jdate;
```

→ jdate
ist das interne Format einer Variablen vom Datentyp DATE.

Returnwert

0 für Sonntag
1 für Montag
2 für Dienstag
3 für Mittwoch
4 für Donnerstag
5 für Freitag
6 für Samstag

Beispiel

```
main()

{
long datum;
int wochentag;

printf("Interne Datumsdarstellung : 32944\n");

wochentag = rdayofweek(32944);

switch (wochentag)
   {
   case 0:
      printf("Sonntag\n");
      break;
   case 1:
      printf("Montag\n");
      break;
   case 2:
      printf("Dienstag\n");
      break;
   case 3:
      printf("Mittwoch\n");
      break;
   case 4:
      printf("Donnerstag\n");
      break;
   case 5:
      printf("Freitag\n");
      break;
   default:
```

```
        printf("Samstag\n");
        break;
    }
}
```

```
Interne Datumsdarstellung : 32944
Dienstag
```

rdefmdate Datum ins interne Format bringen

rdefmtdate konvertiert ein in einer Zeichenkette gespeichertes Datum in das interne Format. Das Format des Eingabedatums können Sie selbst bestimmen.

```
int rdefmtdate(jdate,fmtstring,input)
       long *jdate;
       char *fmtstring;
       char *input;
```

← jdate
ist das interne Format einer Variablen vom Datentyp DATE.

→ fmtstring
ist der Zeiger auf eine Zeichenkette, die ein Datumformat enthält. fmtstring besteht aus einer Kombination der folgenden Angaben:

fmtstring enthält	Datum enthält
dd	zweistellige Tageszahl
ddd	dreibuchstabiges Tageskürzel
mm	zweistellige Monatszahl
mmm	dreibuchstabiges Monatskürzel
yy	zweistellige Jahreszahl
yyyy	vierstellige Jahreszahl
ascii-Zeichen	dasselbe ascii-Zeichen

→ input
ist der Zeiger auf eine Zeichenkette, die das Datum enthält. Das Datum muß zum angegebenen Format passen, d.h. die Reihenfolge von Tag, Monat und Jahr muß mit dem Format übereinstimmen.

Returnwert

0	Funktion war erfolgreich
-1204	ungültige Jahresangabe
-1205	ungültige Monatsangabe
-1206	ungültige Tagesangabe
-1209	Wenn das Datum keine Trennzeichen zwischen der Tages-, Monats- und Jahresangabe hat, dann muß das Datum genau 6 oder 8 Zeichen lang sein.
-1212	Das Format muß eine Tages-, Monats- und Jahresangabe enthalten.

Hinweis

- Diese Funktion benötigt die INFORMIX-Meldungstext-Dateien.
- Die Ausführung dieser Funktion ist daher wesentlich aufwendiger als z.B. der Gebrauch der Funktion rstrdate. Verwenden Sie daher rdefmtdate nur, wenn es unbedingt nötig ist.

Lesen Sie dazu im Kapitel 5, Umgebungsvariable bei den Umgebungsvariablen INFORMIXDIR und DBLANG nach.

Beispiel

```
main()

{
long datum;
char *format;
char *str;
int fehler;

str = "90/03/13";
printf("Eingabe : %s\n", str);

format = "yy.mm.dd";
fehler = rdefmtdate(&datum, format, str);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n", fehler);
   exit(1);
   }

printf("Interne Datumsdarstellung : %d\n\n", datum);

printf("Eingabe : 031390\n");

rdefmtdate(&datum, "mmm. dd. yyyy", "031390");

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n", fehler);
   exit(1);
   }

printf("Interne Datumsdarstellung : %d\n\n", datum);
}
```

```
Eingabe : 90/03/13
Interne Datumsdarstellung : 32944

Eingabe : 031390
Interne Datumsdarstellung : 32944
```

rdownshift Großbuchstaben in Kleinbuchstaben umwandeln

rdownshift wandelt in einer mit Nullbyte abgeschlossenen Zeichenkette alle Buchstaben in Kleinbuchstaben um.

```
int rdownshift(s)
     char *s;
```

↔ s
ist ein Zeiger auf eine mit Nullbyte abgeschlossene Zeichenkette, in der alle Buchstaben in Kleinbuchstaben umgewandelt werden.

Beispiel

```
main()

{
char *str;

str = "GrOSS unD KLeIn wiRd nUN klEiN GESCHrieBeN";
printf("Zeichenfolge vor Umwandlung :\n");
printf("%s\n\n", str);

rdownshift(str);

printf("Zeichenfolge nach Umwandlung :\n");
printf("%s\n\n", str);
}
```

```
Zeichenfolge vor Umwandlung :
GrOSS unD KLeIn wiRd nUN klEiN GESCHrieBeN

Zeichenfolge nach Umwandlung :
gross und klein wird nun klein geschrieben
```

rfmtdate Datum als Zeichenkette darstellen

rfmtdate konvertiert das interne Format eines Datums in eine Zeichenkette. Das Format der Zeichenkette können Sie selbst bestimmen.

```
int rfmtdate(jdate,fmtstring,erg)
      long jdate;
      char *fmtstring;
      char *erg;
```

→ jdate
ist das interne Format einer Variablen vom Datentyp DATE.

→ fmtstring
ist der Zeiger auf eine Zeichenkette, die ein Datumformat enthält.
fmtstring besteht aus einer Kombination der folgenden Angaben:

fmtstring enthält	Datum enthält
dd	zweistellige Tageszahl
ddd	dreibuchstabiges Tageskürzel
mm	zweistellige Monatszahl
mmm	dreibuchstabiges Monatskürzel
yy	zweistellige Jahreszahl
yyyy	vierstellige Jahreszahl
ascii-Zeichen	dasselbe ascii-Zeichen

← erg
ist der Zeiger auf eine Zeichenkette, in die das konvertierte Datum geschrieben werden soll.

Returnwert

0 Funktion war erfolgreich
-1210 Das Datum konnte nicht in das Format Monat/Tag/Jahr konvertiert werden
-1211 Arbeitsspeicher-Ueberlauf

Hinweis

— Diese Funktion benötigt die INFORMIX-Meldungstext-Dateien.

— Die Ausführung dieser Funktion ist daher wesentlich aufwendiger als z.B. der Gebrauch der Funktion rdatestr. Verwenden Sie daher rfmtdate nur, wenn es unbedingt nötig ist.

  Lesen Sie dazu im Kapitel 5 Umgebungsvariable bei den Umgebungsvariablen INFORMIXDIR und DBLANG nach.

Beispiel

```
main()

{
long datum;
char *str;
char ergebnis[40];
int fehler;

datum = 32944;
printf("Internes Format : %d\n", datum);

str = "mmddyy";
fehler = rfmtdate(datum, str, ergebnis);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }

printf("Zeichenkette    : %s\n", ergebnis);

/*
 * Auf Fehlerbehandlungsroutinen nach Aufruf von rfmtdate wird aus
 * Platzgruenden verzichtet
 */

str = "yymmdd";
rfmtdate(datum, str, ergebnis);
printf("Zeichenkette    : %s\n", ergebnis);

rfmtdate(datum, "yy mm dd", ergebnis);
printf("Zeichenkette    : %s\n", ergebnis);

rfmtdate(datum, "mmm. dd yyyy", ergebnis);
printf("Zeichenkette    : %s\n", ergebnis);

rfmtdate(datum, "yy mm dd", ergebnis);
printf("Zeichenkette    : %s\n", ergebnis);

rfmtdate(datum, "(ddd) mmm dd yyyy", ergebnis);
printf("Zeichenkette    : %s\n", ergebnis);
}
```

```
------------------------------------------------------------------------

Internes Format : 32944
Zeichenkette    : 031390
Zeichenkette    : 900313
Zeichenkette    : 90 03 13
Zeichenkette    : Mar. 13 1990
Zeichenkette    : 90 03 13
Zeichenkette    : (Tue) Mar 13 1990
```

rfmtdec DECIMAL nach CHAR umwandeln

rfmtdec konvertiert ein dec_t internes Format in eine formatierte Zeichenfolge gemäß den angegebenen Formatierregeln. Die Bedeutung der Formatier-Zeichen ist im Abschnitt 4.1.1 erklärt.

```
int rfmtdec(dec, format, outbuf)
     dec_t *dec;
     char  *format;
     char  *outbuf;
```

→ dec
ist der Zeiger auf die dec_t-Struktur, die formatiert werden soll.

→ format
ist der Zeiger auf die Formatier-Zeichenfolge.

← outbuf
ist der Zeiger auf die Zeichenkette, in die das Ergebnis geschrieben werden soll.

Returnwert

0	Die Konvertierung war erfolgreich
-1211	Speicherüberlauf
-1217	Die Formatier-Zeichenfolge ist zu lang.

Beispiel

```
#include <decimal.h>

main()

{
double double_zahl_alt;
double double_zahl_neu;
double double_zahl;
dec_t dezimal_zahl;
char *str;
char ausgabe[13];
int fehler;

printf("    Format            Wert       formatierter Wert\n");
printf("-------------------------------------------------\n");

str = "-$$$,$$$.&&";
double_zahl_alt = -12345.67;
deccvdbl(double_zahl_alt, &dezimal_zahl);

fehler = rfmtdec(&dezimal_zahl, str, ausgabe);

if (fehler != 0)
```

```
    {
    printf("Fehlernummer : %d\n", fehler);
    exit(1);
    }

dectodbl(&dezimal_zahl, &double_zahl_neu);
printf("%12s     %8.2f        %13s\n", str, double_zahl_neu, ausgabe);

/*
 * Auf Fehlerabfrage nach Aufruf von rfmtdec wird aus Platzgruenden
 * verzichtet
 */

deccvdbl(-1234.56, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "␣$$$,$$$.&&", ausgabe);
printf(" ␣$$$,$$$.&&      -1234.56        %13s\n", ausgabe);
deccvdbl(-123.45, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "␣$$$,$$$.&&", ausgabe);
printf(" ␣$$$,$$$.&&       -123.45        %13s\n", ausgabe);
deccvdbl(-12345.67, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "␣␣$$,$$$.&&", ausgabe);
printf(" ␣␣$$,$$$.&&     -12345.67        %13s\n", ausgabe);
deccvdbl(-1234.56, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "␣␣$$,$$$.&&", ausgabe);
printf(" ␣␣$$,$$$.&&      -1234.56        %13s\n", ausgabe);
deccvdbl(-123.45, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "␣␣$$,$$$.&&", ausgabe);
printf(" ␣␣$$,$$$.&&       -123.45        %13s\n", ausgabe);
deccvdbl(-12.34, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "␣␣$$,$$$.&&", ausgabe);
printf(" ␣␣$$,$$$.&&        -12.34        %13s\n", ausgabe);
deccvdbl(-1.23, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "␣␣$$,$$$.&&", ausgabe);
printf(" ␣␣$$,$$$.&&         -1.23        %13s\n\n", ausgabe);

deccvdbl(-12345.67, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "␣␣␣␣,␣␣$.&&", ausgabe);
printf(" ␣␣␣␣,␣␣$.&&     -12345.67        %13s\n", ausgabe);
deccvdbl(-1234.56, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "␣␣␣␣,␣␣$.&&", ausgabe);
printf(" ␣␣␣␣,␣␣$.&&      -1234.56        %13s\n", ausgabe);
deccvdbl(-123.45, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "␣␣␣␣,␣␣$.&&", ausgabe);
printf(" ␣␣␣␣,␣␣$.&&       -123.45        %13s\n", ausgabe);
deccvdbl(-12.34, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "␣␣␣␣,␣␣$.&&", ausgabe);
printf(" ␣␣␣␣,␣␣$.&&        -12.34        %13s\n", ausgabe);
deccvdbl(-1.23, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "␣␣␣␣,␣␣$.&&", ausgabe);
printf(" ␣␣␣␣,␣␣$.&&         -1.23        %13s\n", ausgabe);
deccvdbl(-.12, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "␣␣␣␣,␣␣$.&&", ausgabe);
printf(" ␣␣␣␣,␣␣$.&&          -.12        %13s\n\n", ausgabe);

deccvdbl(12345.67, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "$***,***.&&", ausgabe);
printf(" $***,***.&&      12345.67        %13s\n", ausgabe);
deccvdbl(1234.56, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "$***,***.&&", ausgabe);
printf(" $***,***.&&       1234.56        %13s\n", ausgabe);
deccvdbl(123.45, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "$***,***.&&", ausgabe);
printf(" $***,***.&&        123.45        %13s\n", ausgabe);
deccvdbl(12.34, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "$***,***.&&", ausgabe);
printf(" $***,***.&&         12.34        %13s\n", ausgabe);
deccvdbl(1.23, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "$***,***.&&", ausgabe);
printf(" $***,***.&&          1.23        %13s\n", ausgabe);
deccvdbl(.12, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "$***,***.&&", ausgabe);
printf(" $***,***.&&           .12        %13s\n\n", ausgabe);

deccvdbl(-12345.67, &dezimal_zahl);
```

```
rfmtdec(&dezimal_zahl, "($$$,$$$.&&)", ausgabe);
printf("($$$,$$$.&&)     -12345.67        %13s\n", ausgabe);
deccvdbl(-1234.56, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "($$$,$$$.&&)", ausgabe);
printf("($$$,$$$.&&)      -1234.56        %13s\n", ausgabe);
deccvdbl(-123.45, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "($$$,$$$.&&)", ausgabe);
printf("($$$,$$$.&&)       -123.45        %13s\n", ausgabe);
deccvdbl(-12345.67, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "(($$,$$$.&&)", ausgabe);
printf("(($$,$$$.&&)     -12345.67        %13s\n", ausgabe);
deccvdbl(-1234.56, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "(($$,$$$.&&)", ausgabe);
printf("(($$,$$$.&&)      -1234.56        %13s\n", ausgabe);
deccvdbl(-123.45, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "(($$,$$$.&&)", ausgabe);
printf("(($$,$$$.&&)       -123.45        %13s\n", ausgabe);
deccvdbl(-12.34, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "(($$,$$$.&&)", ausgabe);
printf("(($$,$$$.&&)        -12.34        %13s\n", ausgabe);
deccvdbl(-1.23, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "(($$,$$$.&&)", ausgabe);
printf("(($$,$$$.&&)         -1.23        %13s\n\n", ausgabe);

deccvdbl(-12345.67, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "((((,(($.&&)", ausgabe);
printf("((((,(($.&&)     -12345.67        %13s\n", ausgabe);
deccvdbl(-1234.56, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "((((,(($.&&)", ausgabe);
printf("((((,(($.&&)      -1234.56        %13s\n", ausgabe);
deccvdbl(-123.45, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "((((,(($.&&)", ausgabe);
printf("((((,(($.&&)       -123.45        %13s\n", ausgabe);
deccvdbl(-12.34, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "((((,(($.&&)", ausgabe);
printf("((((,(($.&&)        -12.34        %13s\n", ausgabe);
deccvdbl(-1.23, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "((((,(($.&&)", ausgabe);
printf("((((,(($.&&)         -1.23        %13s\n", ausgabe);
deccvdbl(-.12, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "((((,(($.&&)", ausgabe);
printf("((((,(($.&&)          -.12        %13s\n\n", ausgabe);

deccvdbl(12345.0, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "<<<,<<<", ausgabe);
printf("     <<<,<<<        12345        %13s\n", ausgabe);
deccvdbl(1234.0, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "<<<,<<<", ausgabe);
printf("     <<<,<<<         1234        %13s\n", ausgabe);
deccvdbl(123.0, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "<<<,<<<", ausgabe);
printf("     <<<,<<<          123        %13s\n", ausgabe);
deccvdbl(12.0, &dezimal_zahl);
rfmtdec(&dezimal_zahl, "<<<,<<<", ausgabe);
printf("     <<<,<<<           12        %13s\n\n", ausgabe);
}
```

Format	Wert	formatierter Wert
␣$$$,$$$.&&	-12345.67	␣$12,345.67
␣$$$,$$$.&&	-1234.56	- $1,234.56
␣$$$,$$$.&&	-123.45	- $123.45
␣␣$$,$$$.&&	-12345.67	␣$12,345.67
␣␣$$,$$$.&&	-1234.56	␣$1,234.56
␣␣$$,$$$.&&	-123.45	- $123.45
␣␣$$,$$$.&&	-12.34	- $12.34
␣␣$$,$$$.&&	-1.23	- $1.23
␣␣␣␣,␣␣$.&&	-12345.67	␣$12,345.67
␣␣␣␣,␣␣$.&&	-1234.56	␣$1,234.56
␣␣␣␣,␣␣$.&&	-123.45	␣$123.45
␣␣␣␣,␣␣$.&&	-12.34	␣$12.34

```
␣␣␣␣,␣␣$.&&          -1.23              ␣$1.23
␣␣␣␣,␣␣$.&&           -.12               ␣$.12

$***,***.&&       12345.67          $*12,345.67
$***,***.&&        1234.56          $**1,234.56
$***,***.&&         123.45          $****123.45
$***,***.&&          12.34          $*****12.34
$***,***.&&           1.23          $******1.23
$***,***.&&            .12          $*******.12

($$$,$$$.&&)      -12345.67         ($12,345.67)
($$$,$$$.&&)       -1234.56         ( $1,234.56)
($$$,$$$.&&)        -123.45         (   $123.45)
(($$,$$$.&&)      -12345.67         ($12,345.67)
(($$,$$$.&&)       -1234.56          ($1,234.56)
(($$,$$$.&&)        -123.45          (  $123.45)
(($$,$$$.&&)         -12.34          (   $12.34)
(($$,$$$.&&)          -1.23          (    $1.23)

((((,(($.&&)      -12345.67         ($12,345.67)
((((,(($.&&)       -1234.56          ($1,234.56)
((((,(($.&&)        -123.45            ($123.45)
((((,(($.&&)         -12.34             ($12.34)
((((,(($.&&)          -1.23              ($1.23)
((((,(($.&&)           -.12               ($.12)

     <<<,<<<          12345             12,345
     <<<,<<<           1234              1,234
     <<<,<<<            123                123
     <<<,<<<             12                 12
```

rfmtdouble double nach CHAR umwandeln

rfmtdouble konvertiert ein double-internes Format in eine formatierte Zeichenfolge gemäß den angegebenen Formatierregeln. Die Bedeutung der Formatier-Zeichen ist im Abschnitt 4.1.1 erklärt.

```
int rfmtdouble(dvalue, format, outbuf)
     double dvalue;
     char  *format;
     char  *outbuf;
```

→ dvalue
ist die Zahl vom Datentyp double, die formatiert werden soll.

→ format
ist der Zeiger auf die Formatier-Zeichenfolge.

← outbuf
ist der Zeiger auf die Zeichenkette, in die die Ausgabe geschrieben werden soll.

Returnwert

0	Die Konvertierung war erfolgreich
-1211	Speicherüberlauf
-1217	Die Formatier-Zeichenfolge ist zu lang.

Beispiel

```
main()

{
double double_zahl;
char *str;
char ausgabe[10];
int fehler;

printf("    Format          Wert        formatierter Wert\n");
printf("-------------------------------------------------\n");

str = "##,###.##";
double_zahl = 12345.67;

fehler = rfmtdouble(double_zahl, str, ausgabe);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n", fehler);
   exit(1);
   }

printf("%10s     %8.2f           %10s\n", str, double_zahl, ausgabe);
```

```
/*
 * Auf Fehlerabfrage nach Aufruf von rfmtdouble wird aus Platzgruenden
 * verzichtet
 */

rfmtdouble(1234.56, "##,###.##", ausgabe);
printf(" ##,###.##      1234.56          %10s\n", ausgabe);
rfmtdouble(123.45, "##,###.##", ausgabe);
printf(" ##,###.##       123.56          %10s\n", ausgabe);
rfmtdouble(12.34, "##,###.##", ausgabe);
printf(" ##,###.##        12.34          %10s\n", ausgabe);
rfmtdouble(1.23, "##,###.##", ausgabe);
printf(" ##,###.##         1.23          %10s\n", ausgabe);
rfmtdouble(0.12, "##,###.##", ausgabe);
printf(" ##,###.##         0.12          %10s\n", ausgabe);
rfmtdouble(0.01, "##,###.##", ausgabe);
printf(" ##,###.##         0.01          %10s\n", ausgabe);
rfmtdouble(-0.01, "##,###.##", ausgabe);
printf(" ##,###.##        -0.01          %10s\n", ausgabe);
rfmtdouble(-1.0, "##,###.##", ausgabe);
printf(" ##,###.##        -1             %10s\n\n", ausgabe);

rfmtdouble(12345.67, "&&,&&&.&&", ausgabe);
printf(" &&,&&&.&&     12345.67          %10s\n", ausgabe);
rfmtdouble(1234.56, "&&,&&&.&&", ausgabe);
printf(" &&,&&&.&&      1234.56          %10s\n", ausgabe);
rfmtdouble(123.45, "&&,&&&.&&", ausgabe);
printf(" &&,&&&.&&       123.45          %10s\n", ausgabe);
rfmtdouble(0.01, "&&,&&&.&&", ausgabe);
printf(" &&,&&&.&&         0.01          %10s\n\n", ausgabe);

rfmtdouble(12345.67, "$$,$$$.$$", ausgabe);
printf(" $$,$$$.$$     12345.67          %10s (Ueberlauf)\n", ausgabe);
rfmtdouble(1234.56, "$$,$$$.$$", ausgabe);
printf(" $$,$$$.$$      1234.56          %10s\n", ausgabe);
rfmtdouble(0.0, "$$,$$$.##", ausgabe);
printf(" $$,$$$.##         0.00          %10s\n", ausgabe);
rfmtdouble(1234.00, "$$,$$$.##", ausgabe);
printf(" $$,$$$.##      1234.00          %10s\n", ausgabe);
rfmtdouble(0.0, "$$,$$$.&&", ausgabe);
printf(" $$,$$$.&&         0.00          %10s\n", ausgabe);
rfmtdouble(1234.00, "$$,$$$.&&", ausgabe);
printf(" $$,$$$.&&      1234.00          %10s\n\n", ausgabe);

rfmtdouble(-12345.67, "␣##,###.##", ausgabe);
printf("␣##,###.##    -12345.67          %10s\n", ausgabe);
rfmtdouble(-123.45, "␣##,###.##", ausgabe);
printf("␣##,###.##       -123.45          %10s\n", ausgabe);
rfmtdouble(-12.34, "␣##,###.##", ausgabe);
printf("␣##,###.##        -12.34          %10s\n", ausgabe);
rfmtdouble(-12.34, "␣␣#,###.##", ausgabe);
printf("␣␣#,###.##        -12.34          %10s\n", ausgabe);
rfmtdouble(-12.34, "␣␣-,###.##", ausgabe);
printf("␣␣-,###.##        -12.34          %10s\n", ausgabe);
rfmtdouble(-12.34, "␣␣-,␣##.##", ausgabe);
printf("␣␣-,␣##.##        -12.34          %10s\n", ausgabe);
rfmtdouble(-1.00, "␣␣-,␣␣#.##", ausgabe);
printf("␣␣-,␣␣#.##         -1.00          %10s\n\n", ausgabe);

rfmtdouble(12345.67, "␣##,###.##", ausgabe);
printf("␣##,###.##     12345.67          %10s\n", ausgabe);
rfmtdouble(123.45, "␣##,###.##", ausgabe);
printf("␣##,###.##       123.45          %10s\n", ausgabe);
rfmtdouble(12.34, "␣##,###.##", ausgabe);
printf("␣##,###.##        12.34          %10s\n", ausgabe);
rfmtdouble(12.34, "␣␣#,###.##", ausgabe);
printf("␣␣#,###.##        12.34          %10s\n", ausgabe);
rfmtdouble(12.34, "␣␣-,###.##", ausgabe);
printf("␣␣-,###.##        12.34          %10s\n", ausgabe);
rfmtdouble(12.34, "␣␣-,␣##.##", ausgabe);
printf("␣␣-,␣##.##        12.34          %10s\n", ausgabe);
rfmtdouble(1.00, "␣␣-,␣␣#.##", ausgabe);
printf("␣␣-,␣␣#.##         1.00          %10s\n", ausgabe);
rfmtdouble(-.01, "␣␣-,␣␣-.␣␣", ausgabe);
```

```
printf("␣␣-,␣␣-.␣␣          -.01          %10s\n", ausgabe);
rfmtdouble(-.01, "␣␣-,␣␣-.&&", ausgabe);
printf("␣␣-,␣␣-.&&          -.01          %10s\n\n", ausgabe);
}
```

Format	Wert	formatierter Wert
##,###.##	12345.67	12,345.67
##,###.##	1234.56	1,234.56
##,###.##	123.56	123.45
##,###.##	12.34	12.34
##,###.##	1.23	1.23
##,###.##	0.12	.12
##,###.##	0.01	.01
##,###.##	-0.01	.01
##,###.##	-1	1.00
&&,&&&.&&	12345.67	12,345.67
&&,&&&.&&	1234.56	01,234.56
&&,&&&.&&	123.45	000123.45
&&,&&&.&&	0.01	000000.01
$$,$$$.$$	12345.67	********* (Ueberlauf)
$$,$$$.$$	1234.56	$1,234.56
$$,$$$.##	0.00	$.00
$$,$$$.##	1234.00	$1,234.00
$$,$$$.&&	0.00	$.00
$$,$$$.&&	1234.00	$1,234.00
␣##,###.##	-12345.67	-12,345.67
␣##,###.##	-123.45	- 123.45
␣##,###.##	-12.34	- 12.34
␣␣#,###.##	-12.34	- 12.34
␣␣-,###.##	-12.34	- 12.34
␣␣-,␣##.##	-12.34	-12.34
␣␣-,␣␣#.##	-1.00	-1.00
␣##,###.##	12345.67	12,345.67
␣##,###.##	123.45	123.45
␣##,###.##	12.34	12.34
␣␣#,###.##	12.34	12.34
␣␣-,###.##	12.34	12.34
␣␣-,␣##.##	12.34	12.34
␣␣-,␣␣#.##	1.00	1.00
␣␣-,␣␣-.␣␣	-.01	-.01
␣␣-,␣␣-.&&	-.01	-.01

rfmtlong long nach CHAR umwandeln

rfmtlong konvertiert ein long-internes Format in eine formatierte Zeichenfolge gemäß den angegebenen Formatierregeln. Die Bedeutung der Formatier-Zeichen ist im Abschnitt 4.1.1 erklärt.

```
int rfmtlong(lvalue, format, outbuf)
     long  lvalue;
     char  *format;
     char  *outbuf;
```

→ lvalue
ist die Zahl vom Datentyp long, die formatiert werden soll.

→ format
ist der Zeiger auf die Formatier-Zeichenfolge.

← outbuf
ist der Zeiger auf die Zeichenkette, in die die Ausgabe geschrieben werden soll.

Returnwert

0	Die Konvertierung war erfolgreich
-1211	Speicherüberlauf
-1217	Die Formatier-Zeichenfolge ist zu lang.

Beispiel

```
main()

{
long long_zahl;
char *str;
char ausgabe[6];
int fehler;

printf(" Format      Wert        formatierter Wert\n");
printf("-------------------------------------------\n");

str = "#####";
long_zahl = 0;

fehler = rfmtlong(long_zahl, str, ausgabe);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n", fehler);
   exit(1);
   }

printf("%6s     %5d          %6s\n", str, long_zahl, ausgabe);
```

```
/*
 * Auf Fehlerabfrage nach Aufruf von rfmtlong wird aus Platzgruenden
 * verzichtet
 */

rfmtlong(0, "&&&&&", ausgabe);
printf(" &&&&&          0          %6s\n", ausgabe);
rfmtlong(0, "$$$$$", ausgabe);
printf(" $$$$$          0          %6s\n", ausgabe);
rfmtlong(0, "*****", ausgabe);
printf(" *****          0          %6s\n", ausgabe);
rfmtlong(0, "<<<<<", ausgabe);
printf(" <<<<<          0          %6s\n\n", ausgabe);

rfmtlong(12345, "##,###", ausgabe);
printf("##,###      12345          %6s\n", ausgabe);
rfmtlong(1234, "##,###", ausgabe);
printf("##,###       1234          %6s\n", ausgabe);
rfmtlong(123, "##,###", ausgabe);
printf("##,###        123          %6s\n", ausgabe);
rfmtlong(12, "##,###", ausgabe);
printf("##,###         12          %6s\n", ausgabe);
rfmtlong(1, "##,###", ausgabe);
printf("##,###          1          %6s\n", ausgabe);
rfmtlong(-1, "##,###", ausgabe);
printf("##,###         -1          %6s\n", ausgabe);
rfmtlong(0, "##,###", ausgabe);
printf("##,###          0          %6s\n\n", ausgabe);

rfmtlong(12345, "&&,&&&", ausgabe);
printf("&&,&&&      12345          %6s\n", ausgabe);
rfmtlong(1234, "&&,&&&", ausgabe);
printf("&&,&&&       1234          %6s\n", ausgabe);
rfmtlong(123, "&&,&&&", ausgabe);
printf("&&,&&&        123          %6s\n", ausgabe);
rfmtlong(12, "&&,&&&", ausgabe);
printf("&&,&&&         12          %6s\n", ausgabe);
rfmtlong(1, "&&,&&&", ausgabe);
printf("&&,&&&          1          %6s\n", ausgabe);
rfmtlong(-1, "&&,&&&", ausgabe);
printf("&&,&&&         -1          %6s\n", ausgabe);
rfmtlong(0, "&&,&&&", ausgabe);
printf("&&,&&&          0          %6s\n\n", ausgabe);

rfmtlong(12345, "$$,$$$", ausgabe);
printf("$$,$$$      12345          %6s (Ueberlauf)\n", ausgabe);
rfmtlong(1234, "$$,$$$", ausgabe);
printf("$$,$$$       1234          %6s\n", ausgabe);
rfmtlong(123, "$$,$$$", ausgabe);
printf("$$,$$$        123          %6s\n", ausgabe);
rfmtlong(12, "$$,$$$", ausgabe);
printf("$$,$$$         12          %6s\n", ausgabe);
rfmtlong(1, "$$,$$$", ausgabe);
printf("$$,$$$          1          %6s\n", ausgabe);
rfmtlong(-1, "$$,$$$", ausgabe);
printf("$$,$$$         -1          %6s\n", ausgabe);
rfmtlong(0, "$$,$$$", ausgabe);
printf("$$,$$$          0          %6s\n\n", ausgabe);

rfmtlong(12345, "**,***", ausgabe);
printf("**,***      12345          %6s\n", ausgabe);
rfmtlong(1234, "**,***", ausgabe);
printf("**,***       1234          %6s\n", ausgabe);
rfmtlong(123, "**,***", ausgabe);
printf("**,***        123          %6s\n", ausgabe);
rfmtlong(12, "**,***", ausgabe);
printf("**,***         12          %6s\n", ausgabe);
rfmtlong(1, "**,***", ausgabe);
printf("**,***          1          %6s\n", ausgabe);
rfmtlong(0, "**,***", ausgabe);
printf("**,***          0          %6s\n", ausgabe);
}
```

Format	Wert	formatierter Wert
#####	0	
&&&&&	0	00000
$$$$$	0	$
*****	0	*****
<<<<<	0	
##,###	12345	12,345
##,###	1234	1,234
##,###	123	123
##,###	12	12
##,###	1	1
##,###	-1	1
##,###	0	
&&,&&&	12345	12,345
&&,&&&	1234	01,234
&&,&&&	123	000123
&&,&&&	12	000012
&&,&&&	1	000001
&&,&&&	-1	000001
&&,&&&	0	000000
$$,$$$	12345	****** (Ueberlauf)
$$,$$$	1234	$1,234
$$,$$$	123	$123
$$,$$$	12	$12
$$,$$$	1	$1
$$,$$$	-1	$1
$$,$$$	0	$
,*	12345	12,345
,*	1234	*1,234
,*	123	***123
,*	12	****12
,*	1	*****1
,*	0	******

rgetmsg Fehlermeldungsnummern umsetzen

rgetmsg setzt eine INFORMIX-Fehlermeldungsnummer in den entsprechenden Fehlermeldungstext um und speichert ihn in einer Zeichenkette ab.

```
int rgetmsg(msgnum,msgstr,lenmsgstr)
     short msgnum;
     char *msgstr;
     short lenmsgstr;
```

→ msgnum
ist die Fehlermeldungsnummer.

← msgstr
ist der Zeiger auf die Zeichenkette.

→ lenmsgstr
ist die Länge des Meldungstextes.

Returnwert

0	Die Umsetzung war erfolgreich.
-1232	Unbekannte Fehlermeldungsnummer.

Hinweis

Die Fehlermeldungsnummer ist unter sqlca.sqlcode abgelegt. rgetmsg benutzt die Systemdatei usr/informix/msg für den Meldungstext.

Beispiel

```
 $include sqlca;

main()

{
char str[80];
short lge;
int fehler;

lge = 80;

$database unbekannt;

if (sqlca.sqlcode != 0L)
   {

   fehler = rgetmsg(sqlca.sqlcode, str, lge);
   if (fehler != 0L)
      {
      printf("Fehlermeldungsnummer : %d\n", fehler);
```

```
      exit(1);
      }

   printf("Fehlermeldung : %s\n", str);
   exit(1);
   }
}
```

```
Fehlermeldung : Datenbank nicht gefunden oder keine Zugriffserlaubnis.
```

risnull C-Variable auf NULL-Wert prüfen

risnull prüft, ob eine Hostvariable einen NULL-Wert enthält.

```
int risnull(typ,ptrvar)
      int typ;
      char *ptrvar;
```

-→ typ
symbolische Konstante, die dem Datentyp der zu prüfenden Hostvariable zugeordnet ist. Die symbolische Konstante ist in der Datei sqltypes.h definiert.

-→ zeiger;
ist der Zeiger auf die zu prüfende Hostvariable

Returnwert

1 die Variable ist mit einem NULL-Wert belegt
0 die Variable ist nicht mit einem NULL-Wert belegt

Beispiel

```
$include sqltypes;

main()

{
$long variable;

long *zeiger;
int typ;

zeiger = &variable;
typ = CLONGTYPE;

if (risnull(typ, zeiger))
   printf("C-Variable mit NULL-Wert belegt\n");
else
   printf("C-Variable nicht mit NULL-Wert belegt\n");

printf("C-Variable wird NULL-Wert zugewiesen\n");
rsetnull(typ, zeiger);

if (risnull(typ, zeiger))
   printf("C-Variable mit NULL-Wert belegt\n");
else
   printf("C-Variable nicht mit NULL-Wert belegt\n");
}
```

```
C-Variable nicht mit NULL-Wert belegt
C-Variable wird NULL-Wert zugewiesen
C-Variable mit NULL-Wert belegt
```

rjulmdy Datum zerlegen

rjulmdy zerlegt ein Datum in die Tages-, Monats- und Jahreszahl. Das Datum muß im internen Format vorliegen.

```
int rjulmdy(jdate,mdy)
      long jdate;
      short mdy[3];
```

→ jdate
ist das interne Format einer Variablen vom Datentyp DATE.

← mdy[3]

mdy[0]
wird der Monat aus jdate zugewiesen.

mdy[1]
wird der Tag aus jdate zugewiesen.

mdy[2]
wird das Jahr aus jdate zugewiesen.

Returnwert

0 Funktion war erfolgreich

Beispiel

```
main()

{
short datum[3];

printf("Internes Format : 32944\n");

rjulmdy(32944L, datum);

printf("Tag   : %d\n", datum[1]);
printf("Monat : %d\n", datum[0]);
printf("Jahr  : %d\n", datum[2]);
}
```

```
------------------------------------------------------------------------

Internes Format : 32944
Tag   : 13
Monat : 3
Jahr  : 1990
```

rleapyear Schaltjahre herausfinden

rleapyear stellt fest, ob ein Jahr ein Schaltjahr ist.

```
int rleapyear(year)
      int year;
```

→ year
muß eine Jahreszahl sein, kein vollständiges Datum.

Returnwert

1 wenn das Jahr ein Schaltjahr ist.
0 wenn das Jahr kein Schaltjahr ist.

Beispiel

```
main()

{
long datum;

datum = 1990;

while (!rleapyear(datum))
   printf("%d ist kein Schaltjahr\n", datum++);
printf("%d ist Schaltjahr\n", datum);
}
```

```
1990 ist kein Schaltjahr
1991 ist kein Schaltjahr
1992 ist Schaltjahr
```

rmdyjul Tages-, Monats- und Jahreszahl in ein internes Format bringen

rmdyjul setzt die einzelnen Komponenten eines Datums zusammen und konvertiert das Datum in das interne Format.

```
int rmdyjul(mdy,jdate)
     short mdy[3];
     long *jdate;
```

→ mdy[3]

mdy[0]
muß eine Zahl zwischen 1 und 12 für den Monat enthalten.

mdy[1]
muß eine Zahl zwischen 1 und 31 für den Tag enthalten.

mdy[2]
muß eine Zahl zwischen 1 und 9999 für das Jahr enthalten.

← jdate
ist der Zeiger auf den Speicherbereich, in den das konvertierte Datum geschrieben werden soll.

Returnwert

0	Funktion war erfolgreich
-1204	ungültige Jahreszahl in mdy[2]
-1205	ungültige Monatszahl in mdy[0]
-1206	ungültige Tageszahl in mdy[1]

Beispiel

```
main()

{
short datum[3];
long ergebnis;
int fehler;

datum[0] = 6;
datum[1] = 10;
datum[2] = 1963;
datum[3] = '\0';
printf("Eingabedatum : 06.10.1963\n");

fehler = rmdyjul(datum, &ergebnis);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }

printf("Internes Format : %d\n", ergebnis);
}
```

```
Eingabedatum : 06.10.1963
Internes Format : 23171
```

rsetnull Hostvariablen einen NULL-Wert zuweisen

rsetnull weist einer Hostvariablen einen NULL-Wert zu.

```
int rsetnull(typ,ptrvar)
      int typ;
      char *ptrvar;
```

-→ typ
ist die symbolische Konstante, die dem Datentyp der Hostvariablen zugeordnet ist. Die symbolische Konstante ist in sqltypes.h definiert.

←- ptrvar;
ist der Zeiger auf die zu belegende C-Variable.

Beispiel

Siehe Beispiel zur Funktion risnull

rstod Zeichenkette in double umwandeln

rstod konvertiert eine mit Nullbyte abgeschlossene Zeichenkette in den Datentyp double.

```
int rstod(string, double_val)
     char *string;
     double *double_val;
```

→ string
ist der Zeiger auf die Zeichenkette.

← double_val
ist der Zeiger auf die Variable vom Datentyp double, die das Ergebnis aufnehmen soll.

Returnwert

0 Die Umwandlung war erfolgreich
≠ keine Konvertierung möglich.

Beispiel

```
main()

{
char *str;
double double_zahl;
short fehler;

str = "3.14159";
printf("Zeichenkette : 3.14159\n");

fehler = rstod(str, &double_zahl);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }

printf("Zahl vom Typ double : %f\n\n", double_zahl);

str = "a 3.14159";
printf("Zeichenkette : a 3.14159\n");

fehler = rstod(str, double_zahl);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }

printf("Zahl vom Typ double : %f\n\n", double_zahl);
```

```
}
```

```
Zeichenkette : 3.14159
Zahl vom Typ double : 3.141590

Zeichenkette : a 3.14159
Fehlernummer : -1213
```

rstoi Zeichenkette in int umwandeln

rstoi konvertiert eine mit Nullbyte abgeschlossene Zeichenkette in den Datentyp int.

```
int rstoi(string, int_val)
     char *string;
     int *int_val;
```

→ string
ist der Zeiger auf die Zeichenkette. Der zulässige Wertebereich liegt zwischen -32767 und 32767.

← int_val
ist der Zeiger auf die Variable vom Datentyp int, die das Ergebnis aufnehmen soll.

Returnwert

0	Konvertierung war erfolgreich
≠	Konvertierung war nicht erfolgreich

Beispiel

```
main()

{
char *str;
short integer_zahl;
short fehler;

str = "14159";
printf("Zeichenkette : 14159\n");

fehler = rstoi(str, &integer_zahl);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }

printf("Zahl vom Typ integer : %d\n\n", integer_zahl);

str = "999999";
printf("Zeichenkette : 999999\n");

fehler = rstoi(str, &integer_zahl);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }
```

```
printf("Zahl vom Typ integer : %d\n\n", integer_zahl);
}
```

```
Zeichenkette : 14159
Zahl vom Typ integer : 14159

Zeichenkette : 999999
Fehlernummer : -1214
```

rstol Zeichenkette in long umwandeln

rstol konvertiert eine mit dem Nullbyte abgeschlossene Zeichenkette in den Datentyp long.

```
int rstol(string, long_val)
     char *string;
     long *long_val;
```

→ string
ist der Zeiger auf die Zeichenkette. Der zugelassene Wertebereich liegt zwischen -2147483647 und 2147483647.

← long_val
ist der Zeiger auf die Variable vom Datentyp long, die das Ergebnis aufnehmen soll.

Returnwert

0	Konvertierung war erfolgreich
≠	Konvertierung war nicht erfolgreich

Beispiel

```
main()

{
char *str;
long long_zahl;
short fehler;

str = "-654321";
printf("Zeichenkette : -654321\n");

fehler = rstol(str, &long_zahl);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }

printf("Zahl vom Typ long : %d\n\n", long_zahl);

str = "1a -654321";
printf("Zeichenkette : 1a -654321\n");

fehler = rstol("1a -654321", &long_zahl);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }
```

```
printf("Zahl vom Typ long : %d\n\n", long_zahl);
}
```

```
Zeichenkette : -654321
Zahl vom Typ long : -654321

Zeichenkette : 1a -654321
Fehlernummer : -1213
```

rstrdate Datum ins interne Format bringen

rdefmtdate konvertiert ein Datum in das interne Format. Das Eingabedatum muß ein fest vorgegebenes Format haben. Dieses Format legen Sie durch die Umgebungsvariable DBDATE fest. Lesen Sie dazu den entsprechenden Unterpunkt im Kapitel 5, Umgebungsvariable.

```
int rstrdate(str,jdate)
     char *str;
     long *jdate;
```

→ str
ist der Zeiger auf die Zeichenfolge, die das Datum darstellt. Die Zeichenfolge muß das Format besitzen, das Sie in der Umgebungsvariable DBDATE festgelegt haben. Voreinstellung ist das Format mm/tt/jjjj.

mm
zweistellige Monatsangabe

tt
zweistellige Tagesangabe

jjjj
vierstellige Jahresangabe

Als Trennzeichen ist auch jedes andere Zeichen erlaubt, ausgenommen Zahlen.

← jdate
ist der Zeiger auf den Speicherbereich, in den das konvertierte Datum geschrieben werden soll.

Returnwert

0	Die Funktion war erfolgreich
< 0	Fehler

Beispiel

```
main()

{
long datum;
char *str;
int fehler;

str = "04/05/1990";
printf("Zeichenkette : '%s'\n", str);

rstrdate(str, &datum);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }

printf("Internes Format : %d\n\n", datum);

printf("Zeichenkette : 'sinnlos'\n");

rstrdate("sinnlos", &datum);

if (fehler != 0)
   {
   printf("Fehlernummer : %d\n\n", fehler);
   exit(1);
   }

printf("Internes Format : %d\n\n", datum);
}
```

```
Zeichenkette : '04/05/1990'
Internes Format : 32996

Zeichenkette : 'sinnlos'
Internes Format : -2147483648
```

rtoday Datum aus der Systemuhr holen

rtoday liefert das im Rechner gespeicherte Datum im internen Format.

```
int rtoday(today)
    long *today;
```

← today
ist der Zeiger auf den Speicherbereich, in den das Datum geschrieben werden soll.

Beispiel

```
main()

{
long datum;

rtoday(&datum);

printf("Datum aus Systemuhr im internen Format : %d\n", datum);
}
```

```
Datum aus Systemuhr im internen Format : 32944
```

rtypalign Zeiger auf Typgrenze ausrichten

rtypalign richtet einen Zeiger auf den angegebenen Datentyp aus. rtypalign und rtypmsize sind nützlich beim Ausfüllen einer sqlda-Struktur, die von FETCH ausgewertet wird.

```
char *rtypalign(ptr, type)
        char *ptr;
        int type;
```

→ ptr
ist der Zeiger. Er zeigt auf den auszurichtenden Bereich.

→ type
ist die symbolische Konstante, die dem SQL- oder C-Datentyp in der Datei sqltypes.h zugeordnet ist. Diese Konstante wird z.B. von DESCRIBE nach sqlda.sqlvar->sqltype geschrieben.

Returnwert

Ein Zeiger auf die für den Datentyp ausgerichtete Adresse.

Beispiel

```
$include sqltypes;

char *rtypalign();
char *malloc();

main()

{
char *zeiger;
int konstante;

zeiger = malloc(100);                    /* Speicherplatz anfordern */
printf("Zeiger              : %d\n", zeiger);

konstante = SQLFLOAT;
zeiger = rtypalign(zeiger, konstante);
printf("Zeiger nach SQLFLOAT : %d\n\n", zeiger);

zeiger += rtypmsize(konstante, 0);       /* Siehe rtypmsize */

printf("Zeiger                    : %d\n", zeiger);
zeiger = rtypalign(zeiger, CDECIMALTYPE);
printf("Zeiger nach CDECIMALTYPE : %d\n\n", zeiger);

zeiger += rtypmsize(zeiger,CDECIMALTYPE);    /* s. o. */

printf("Zeiger : %d\n", zeiger);

free(zeiger);                            /* Speicher freigeben */
}
```

```
Zeiger               : 53804
Zeiger nach SQLFLOAT : 53804

Zeiger                    : 53812
Zeiger nach CDECIMALTYPE : 53812

Zeiger : 53834
```

rtypmsize Speicherplatzbedarf eines Datentyps ermitteln

rtypmsize gibt den Speicherplatzbedarf von SQL- oder C-Datentypen an. rtypmsize dient dem Zweck, eine von DESCRIBE gelieferte sqlda-Struktur auszufüllen.

```
rtypmsize(type, len)
      int type;
      int len;
```

→ type
ist die symbolische Konstante, die dem SQL- oder C-Datentyp in der Datei sqltypes.h (siehe Anhang) zugeordnet ist.

→ len
wird nur bei Datentypen ausgewertet, die Zeichenfolgen enthalten, und muß die Länge der Zeichenfolge angeben. Bei CCHARTYPE und CSTRINGTYPE erhöht rtypmsize die Länge um 1 für das abschließende Nullbyte. Bei FIXCHAR gibt es kein Nullbyte.

Returnwert

type	len	Returnwert
SQLCHAR	n	n + 1
SQLSMINT	-	sizeof(short)
SQLINT	-	sizeof(long)
SQLFLOAT	-	sizeof(double
SQLSMFLOAT	-	sizeof(float)
SQLDECIMAL	-	sizeof(struct decimal)
SQLSERIAL	-	sizeof(long)
SQLDATE	-	sizeof(long)
SQLMONEY	-	sizeof(struct decimal)
CCHARTYPE	n	n + 1
CSHORTTYPE	-	sizeof(short)
CINTTYPE	-	sizeof(int)
CLONGTYPE	-	sizeof(long)
CFLOATTYPE	-	sizeof(float)
CDOUBLETYPE	-	sizeof(double
CDECIMALTYPE	-	sizeof(struct decimal)
CFIXCHARTYPE	n	n
CSTRINGTYP	n	n + 1
CDATETYPE	-	sizeof(long)
CMONEYTYPE	-	sizeof(struct decimal)
CVCHARTYPE		sizeof(struct var)
kein gültiger SQL-Typ	-	0

Beispiel

```
$include sqltypes;

main()

{
int konstante;
int lge;

konstante = SQLCHAR;
lge = 5;
printf("SQL-Datentyp : SQLCHAR\n");
printf("Rueckkehrwert : %d\n\n", rtypmsize(konstante, lge));

printf("SQL-Datentyp : SQLSERIAL\n");
printf("Rueckkehrwert : %d\n\n", rtypmsize(SQLSERIAL, 999));

printf("SQL-Datentyp : CDOUBLETYPE\n");
printf("Rueckkehrwert : %d\n\n", rtypmsize(105, lge));

printf("SQL-Datentyp : CFIXCHARTYPE\n");
printf("Rueckkehrwert : %d\n\n", rtypmsize(CFIXCHARTYPE, 40));
}
```

```
------------------------------------------------------------------------

SQL-Datentyp : SQLCHAR
Rueckkehrwert : 6

SQL-Datentyp : SQLSERIAL
Rueckkehrwert : 4

SQL-Datentyp : CDOUBLETYPE
Rueckkehrwert : 8

SQL-Datentyp : CFIXCHARTYPE
Rueckkehrwert : 40
```

rtypname Bezeichnungen der SQL-Datentypen ausgeben

rtypname gibt eine mit Nullbyte abgeschlossene Zeichenkette aus, die den Namen des zugeordneten SQL-Datentyps enthält.

```
char *rtypname(sqltype)
          int sqltype;
```

→ sqltype
ist die symbolische Konstante, die dem SQL-Datentyp in der Datei sqltypes.h (sieheAnhang) zugeordnet ist.

Returnwert

sqltype	Zeichenfolge
SQLCHAR	"char"
SQLSMINT	"smallint"
SQLINT	"integer"
SQLFLOAT	"float"
SQLSMFLOAT	"smallfloat"
SQLDECIMAL	"decimal"
SQLSERIAL	"serial"
SQLDATE	"date"
SQLMONEY	"money"
SQLDTIME	"datetime"
SQLINTERVAL	"interval"
SQLBYTES	"byte"
SQLTEXT	"text"
SQLVCHAR	"vchar"
ungültiger Typ	""

Beispiel

```
$include sqltypes;

main()

{
int konstante;

konstante = SQLINT;
printf("SQL-Datentyp : SQLINT\n");
printf("Zeichenfolge : %s\n\n", rtypname(konstante));

printf("SQL-Datentyp : SQLDATE\n");
printf("Zeichenfolge : %s\n\n", rtypname(SQLDATE));

printf("SQL-Datentyp : SQLDECIMAL\n");
printf("Zeichenfolge : %s\n\n", rtypname(5));
}
```

```
------------------------------------------------------------------------

SQL-Datentyp : SQLINT
Zeichenfolge : integer

SQL-Datentyp : SQLDATE
Zeichenfolge : date

SQL-Datentyp : SQLDECIMAL
Zeichenfolge : decimal
```

rtypwidth Platzbedarf des konvertierten Datentyps ermitteln

rtypwidth ermittelt die kleinste Länge, die eine Zeichenfolge haben muß, um einen konvertierten SQL-Datentyp aufnehmen zu können.

```
rtypwidth(sqltype, sqllen)
      int sqltype;
      int sqllen;
```

→ sqltype
ist die symbolische Konstante, die dem SQL-Datentyp in der Datei sqltypes.h zugeordnet ist.

→ sqllen
ist die Anzahl der Bytes in einer Datei für den entsprechenden SQL-Datentyp.

Returnwert

0 der angegebene Datentyp ist kein gültiger SQL-Datentyp.
n > 0 ist die Anzahl an Bytes, die für den Datentyp benötigt wird.

Beispiel

```
$include sqltypes;

#include <decimal.h>

main()

{
int konstante;
int lge;

konstante = SQLCHAR;
lge = 1;
printf("SQL-Datentyp : SQLCHAR\n");
printf("Rueckkehrwert : %d\n\n", rtypwidth(konstante, lge));

printf("SQL-Datentyp : SQLSMFLOAT\n");
printf("Rueckkehrwert : %d\n\n", rtypwidth(SQLSMFLOAT, 0));

printf("SQL-Datentyp : SQLDECIMAL\n");
printf("Rueckkehrwert : %d\n\n", rtypwidth(SQLDECIMAL, PRECTOT(12)));
}
```

```
SQL-Datentyp : SQLCHAR
Rueckkehrwert : 1

SQL-Datentyp : SQLSMFLOAT
Rueckkehrwert : 14

SQL-Datentyp : SQLDECIMAL
Rueckkehrwert : 2
```

rupshift Klein- in Großbuchstaben umwandeln

rupshift wandelt in einer mit Nullbyte abgeschlossenen Zeichenkette alle Buchstaben in Großbuchstaben um.

```
rupshift(s)
   char *s;
```

↔ s
ist ein Zeiger auf eine mit Nullbyte abgschlossene Zeichenkette, in der alle Buchstaben in Großbuchstaben umgewandelt werden.

Beispiel

```
main()

{
char *str;

str = "GrOSS unD KLeIn wiRd nUN groSs GESCHrieBeN";
printf("Zeichenfolge vor Umwandlung :\n");
printf("%s\n\n", str);

rupshift(str);

printf("Zeichenfolge nach Umwandlung :\n");
printf("%s\n\n", str);
}
```

```
Zeichenfolge vor Umwandlung :
GrOSS unD KLeIn wiRd nUN groSs GESCHrieBeN

Zeichenfolge nach Umwandlung :
GROSS UND KLEIN WIRD NUN GROSS GESCHRIEBEN
```

sqlbreak Datenbankprozeß unterbrechen

sqlbreak schickt an das INFORMIX-Backend die Anforderung, den Datenbank-Prozeß zu unterbrechen.

```
sqlbreak()
```

Erhält das INFORMIX-Backend das Unterbrechungssignal, reagiert es wie bei einer fehlerhaften SQL-Anweisung: Er gibt Status und Kontrolle an den Anwendungsprozeß ab.

sqlexit Datenbank-Prozeß beenden

sqlexit beendet einen Datenbank-Prozeß und gibt den belegten Speicherplatz frei.
Die Funktion kann dazu verwendet werden, den Speicherplatzbedarf einer Datenbankanwendung zu reduzieren.

```
sqlexit()
```

Die Funktion ist erforderlich, wenn von einem Anwendungsprogramm, das bereits SQL-Anweisungen verarbeitet hat, ein Sohnprozeß erzeugt werden soll. Da der Sohnprozeß die Verbindungen zu dem SQLEXEC miterbt, wird ggfs. zuviel Speicherplatz belegt. Mit sqlexit können die Verbindungen des Sohnprozesses zum SQLEXEC abgebrochen werden. Die Kommunikation des Vaterprozesses zum SQLEXEC bleibt dabei unberührt.

Hinweis

Sie sollten die Funktion sqlexit nur aufrufen, wenn keine Datenbank eröffnet ist. Es ist empfehlenswert, ggf. eine CLOSE DATABASE-Anweisung vor dem Aufruf zu benutzen.
Rufen Sie sqlexit auf, solange noch eine Datenbank geöffnet ist, werden alle laufenden Transaktionen zurückgesetzt und die Datenbank dann geschlossen.

sqlstart Datenbank-Prozeß starten

sqlstart startet einen Datenbank-Prozeß. Sie sollten sqlstart nur aufrufen, wenn keine Datenbank eröffnet ist, sonst bewirkt sqlstart nichts.

```
sqlstart()
```

Hinweis

sqlstart hat die gleiche Wirkung wie die Anweisung $database; allerdings öffnet sqlstart gleichzeitig die Datenbank.

Returnwert

0 Datenbank-Prozeß ist gestartet.
≠ Funktion konnte nicht ordnungsgemäß ausgeführt werden.

stcat Zwei Zeichenketten verketten

stcat verkettet zwei mit Nullbyte abgeschlossene Zeichenketten.

```
stcat(s,dest)
     char *s, *dest;
```

→ s
ist der Zeiger auf den Anfang der Zeichenkette, die an die Zeichenkette, auf die dest zeigt, angehängt werden soll.

←→ dest
ist der Zeiger auf den Anfang der Ziel-Zeichenkette. stcat hängt s an das Ende der Zeichenkette an. Der Bereich, auf den dest zeigt, muß groß genug sein.

Beispiel

```
main()

{
char *str1;
char *str2;

str1 = "konkatenieren ";
printf("1. Zeichenkette : '%s'\n", str1);
str2 = "von Zeichenketten";
printf("2. Zeichenkette : '%s'\n", str2);

stcat(str2, str1);

printf("1. Zeichenkette : '%s'\n", str1);
}
```

```
1. Zeichenkette : 'konkatenieren '
2. Zeichenkette : 'von Zeichenketten'
1. Zeichenkette : 'konkatenieren von Zeichenketten'
```

stchar Zeichenkette kopieren

stchar kopiert eine mit dem Nullbyte abgeschlossene Zeichenkette in eine Zeichenkette der angegebenen Länge und füllt ggf. am Ende mit Leerzeichen auf.

```
stchar(from, to, count)
  char *from;
  char *to;
  int count;
```

→ from
ist ein Zeiger auf eine mit dem Nullbyte abgeschlossene Zeichenkette, die kopiert werden soll.

← to
ist der Zeiger auf die aufnehmende Zeichenkette, die mindestens die Länge count haben muß.

→ count
ist die Anzahl von Zeichen in der aufnehmenden Zeichenkette.

Hinweis

— ist die Anzahl count kleiner als die Länge der Zeichenkette from, so wird gekürzt.

— Die Zeiger from und to können auf dieselbe Speicheradresse zeigen.

Beispiel

```
main()

{
char *str1;
$fixchar str2[50];
int lge;
int i;

str1 = "Zeichenfolge fester Laenge   ";
printf("Urspruengliche Zeichenkette : %s\n", str1);

lge = 12;
stchar(str1, &str2, lge);
printf("Kopierte Zeichenfolge       :\n");
i = 0;
while ((i <= 50) && (str2[i] != '\0'))
   printf("%c", str2[i++]);
printf("!!!!\n");
```

```
stchar(str1, &str2, 45);
printf("Kopierte Zeichenfolge       :\n");
i = 0;
while ((i <= 50) && (str2[i] != '\0'))
   printf("%c", str2[i++]);
printf("!!!!\n");
}
```

```
Urspruengliche Zeichenkette : Zeichenfolge fester Laenge
Kopierte Zeichenfolge       :
Zeichenfolge!!!!
Kopierte Zeichenfolge       :
Zeichenfolge fester Laenge                    !!!!
```

stcmpr Zeichenketten vergleichen

stcmpr vergleicht zwei mit Nullbyte abgeschlossene Zeichenketten. s1 ist größer als s2, wenn s1 in der ASCII-Sortierfolge nach s2 kommt.

```
stcmpr(s1,s2)
  char *s1, *s2;
```

-→ s1
ist der Zeiger auf die erste Zeichenkette.

-→ s2
ist der Zeiger auf die zweite Zeichenkette.

Returnwert

Das Ergebnis des Vergleichs ergibt sich aus der Subtraktion des ASCII-Codes von s1 und s2.

= 0 die beiden Zeichenketten sind gleich;
< 0 s1 ist kleiner s2
> 0 s1 ist größer s2

Beispiel

```
main()

{
char *str1;
char *str2;
int lge;

printf("=0  a gleich b      <0  a kleiner b      >0  a groesser b\n\n");

str1 = "beispiel";
str2 = "beistand";
printf("Vergleich von '%s' und '%s' :\n", str1, str2);
printf("Ergebnis : %d\n\n", stcmpr(str1, str2));

str2 = "beispiel";
printf("Vergleich von '%s' und '%s' :\n", str1, str2);
printf("Ergebnis : %d\n\n", stcmpr(str1, str2));

str1 = "informix";
printf("Vergleich von '%s' und '%s' :\n", str1, str2);
printf("Ergebnis : %d\n\n", stcmpr(str1, str2));
}
```

```
-------------------------------------------------------------------------

=0  a gleich b      <0  a kleiner b      >0  a groesser b

Vergleich von 'beispiel' und 'beistand' :
Ergebnis : -1
```

```
Vergleich von 'beispiel' und 'beispiel' :
Ergebnis : 0

Vergleich von 'informix' und 'beispiel' :
Ergebnis : 1
```

stcopy Zeichenkette kopieren

stcopy kopiert eine mit Nullbyte abgeschlossene Zeichenkette.

```
stcopy(from,to)
   char *from, *to;
```

from
: ist der Zeiger auf die Zeichenkette, die kopiert werden soll.

to
: ist der Zeiger auf den Bereich im Speicher, in den kopiert werden soll. Der Bereich muß ausreichend groß sein.

Beispiel

```
main()

{
char *str1;
char str2[80];

str1 = "zu kopierender String";
printf("zu kopierende Zeichenkette : '%s'\n", str1);

stcopy(str1, str2);

printf("kopierte Zeichenkette      : '%s'\n", str2);
}
```

```
--------------------------------------------------------------------------

zu kopierende Zeichenkette : 'zu kopierender String'
kopierte Zeichenkette      : 'zu kopierender String'
```

stleng Zeichen einer Zeichenkette zählen

stleng gibt die Länge einer mit Nullbyte abgeschlossenen Zeichenkette in Byte an. Dabei zählt das Nullbyte nicht mit.

```
stleng(string)
  char *string;
```

→ string
ist der Zeiger auf die Zeichenkette.

Returnwert

Die Länge der Zeichenkette ist der Returnwert.

Beispiel

```
main()

{
char *str;

str = "Laenge der Zeichenfolge ohne abschliessene Null ";
printf("Zeichenfolge : '%s'\n", str);

printf("Laenge der Zeichenfolge : %d\n", stleng(str));
}
```

```
---------------------------------------------------------------------

Zeichenfolge : 'Laenge der Zeichenfolge ohne abschliessene Null '
Laenge der Zeichenfolge : 48
```

5 Umgebungsvariablen

Umgebungsvariablen

Sie können die "Umgebung" beeinflussen, in der INFORMIX ESQL/C arbeitet. Dies geschieht über Variablen, deren Inhalt Sie individuell verändern können und die INFORMIX ESQL/C anschließend entsprechend auswertet. Die folgende Tabelle faßt alle von INFORMIX ESQL/C verwendeten Umgebungsvariablen kurz zusammen:

Umgebungsvariable	Bedeutung
DBANSIWARN	ANSI-Standards überwachen
DBDATE	Ein- und Ausgabeformat für das Datum
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
SQLEXEC	INFORMIX-Backend bestimmen
TBCONFIG	Datei für das INFORMIX-ONLINE-System

Ist eine Umgebungsvariable nicht gesetzt, so benutzt INFORMIX einen Standardwert.

Setzen von Umgebungsvariablen

Umgebungsvariablen setzen Sie in der Shell. Beachten Sie, daß Sie Umgebungsvariablen in Großbuchstaben angeben müssen.

Beispiel (Bourne-Shell): `DBTEMP=/abc`
Beispiel (C-Shell) : `setenv DBTEMP /abc`

Damit setzen Sie die Umgebungsvariable DBTEMP auf den Wert /abc. Diese Wertzuweisung gilt jedoch nur für die login-Shell. Soll der Wert auch in einer Sub-Shell gelten, so müssen Sie die Variable exportieren.

Beispiel: `export DBTEMP`

Wenn Sie diese Eingaben nicht nach jedem Anmelden neu vornehmen wollen, dann müssen Sie sie in der Datei

.profile (Bourne-Shell),
.login (C-Shell)

speichern.

Mit unset (bzw. unsetenv) können Sie die Variablen zurücksetzen.

Abfragen von Umgebungsvariablen

`echo_$variable`
gibt den Wert aus, auf den die angegebene Variable variable gesetzt ist.

`set`
gibt alle Variablen aus, die für die aktuelle Shell gesetzt sind.

`export`
gibt die Namen aller exportierten Variablen aus.

Umgebungsvariable DBANSIWARN

Mit DBANSIWARN überprüfen Sie die Erweiterungen von INFORMIX ESQL/C gegenüber dem ANSI-Standard. DBANSIWARN wird kein Wert zugewiesen. Die Funktion entspricht der des Schalters -ansi.

Zur Laufzeit des Progamms wird die Variable sqlwarn5 aus der sqlca-Struktur auf W gesetzt.

Beim Übersetzen des Quellprogramms werden die Warnungen auf dem Bildschirm ausgegeben.

Umgebungsvariable DBDATE

definiert das Ein- und Ausgabeformat eines Datums.

Erlaubte Werte:

Der Wert von DBDATE besteht aus den Buchstaben D, M und Y in beliebiger Reihenfolge. Hinter dem Y muß eine 2 oder 4 stehen. Der Wert muß mit Punkt "." oder Schrägstrich "/" oder Bindestrich "-" abgeschlossen sein.

- Die Reihenfolge von D, M und Y bestimmt die Reihenfolge der Tages-, Monats- und Jahreszahl im Datum. Dabei bedeutet:

  D eine zweistellige Tageszahl
  M eine zweistellige Monatszahl
  Y2 eine zweistellige Jahreszahl
  Y4 eine vierstellige Jahreszahl; (zweistelligen Angaben werden automatisch in vierstellige umgewandelt: 19xx).

- Das letzte Zeichen des Wertes ist das Trennzeichen, das die Tages-, Monats- und Jahreszahlen im Datum trennt.

Standardwert: `DBDATE=DMY4.`

Das bedeutet: Als Reihenfolge gilt Tag, Monat, Jahr; die Jahreszahl kann 4-stellig eingegeben werden; die Angaben Tag, Monat und Jahr sind durch das Zeichen "." (Punkt) zu trennen.

Beispiel

- DBDATE=DMY4. (Standardwert)

  Ausgabe: 12.05.1987

- DBDATE=MDY2-

  Ausgabe: 05-12-87

Umgebungsvariable DBLANG

bestimmt das Dateiverzeichnis für fremdsprachige Meldungstexte.

Das angegebene Dateiverzeichnis muß in dem Dateiverzeichnis enthalten sein, das die Umgebungsvariable INFORMIXDIR bezeichnet. Dort ist standardmäßig /usr/lib/informix definiert. Als Wert darf nur der Name des Unterdateiverzeichnisses angegeben werden.

Standardwert: `DBLANG=msg`

Das Standard-Dateiverzeichnis für Meldungstexte heißt msg. Wenn Sie ein anderes Meldungsdateiverzeichnis verwenden wollen, müssen Sie dieses Dateiverzeichnis unter $INFORMIXDIR einrichten. Die Zugriffsrechte sind folgendermaßen zu setzen:

USER und GROUP: informix Lese-, Schreib-, Ausführberechtigung: 755

Die zum Meldungsdateiverzeichnis gehörenden .iem-Dateien müssen Sie unter das Dateiverzeichnis $INFORMIXDIR/$DBLANG kopieren. Diese Dateien benötigen folgende Zugriffsrechte:

USER und GROUP: informix Lese-, Schreib-, Ausführberechtigung: 644

Umgebungsvariable DBMONEY

legt für die Ausgabe von Werten

- in DECIMAL- und MONEY-Spalten fest, welches Zeichen zwischen Vor- und Nachkommastellen gesetzt wird: Komma oder Punkt. Dies gilt unabhängig davon, ob die Eingabe mit Komma oder Punkt erfolgt.
- in MONEY-Spalten ein Präfix und/oder ein Suffix fest, das zusammen mit dem Spaltenwert ausgegeben wird.

Syntax:

DBMONEY=[präfix]{,|.}[suffix]

Standardwert: DBMONEY=,

präfix
Zeichenfolge, die am Anfang des Wertes ausgegeben wird.
Max. Länge: 7 Zeichen
Verbotene Zeichen: Zahlen, Komma, Punkt.

suffix
Zeichenfolge, die am Ende des Wertes ausgegeben wird. Länge und Zeichenvorrat wie bei präfix (siehe dort).

,
Zahlen mit Nachkommastellen erhalten (unabhängig von der Eingabe) bei der Ausgabe ein Komma als Trennzeichen.

.
Zahlen mit Nachkommastellen erhalten (unabhängig von der Eingabe) bei der Ausgabe einen Punkt als Trennzeichen.

Beispiel

— DBMONEY=, (Standardwert)

Ausgabe: 4768,36

— DBMONEY=.Dollar

Ausgabe: 4768.36 Dollar

Umgebungsvariable DBPATH

legt Pfade zu Dateiverzeichnissen fest, die INFORMIX zusätzlich zum aktuellen Dateiverzeichnis durchsuchen soll. DBPATH wird nur ausgewertet, wenn INFORMIX im aktuellen Dateiverzeichnis nicht fündig wird und zwar

— beim Suchen der Datenbank,
— beim Suchen eines Formats.

Für INFORMIX-STAR gibt die Umgebungsvariable DBPATH an, in welchem Informixsystem die Datenbank gesucht wird.

Die einzelnen Pfade sind mit Doppelpunkt ":" zu trennen.

Standardmäßig sucht INFORMIX nur im aktuellen Dateiverzeichnis.

Beispiel

```
DBPATH=/usr/a:/usr/b:/usr/c
```

Wird die Datenbank im aktuellen Dateiverzeichnis nicht gefunden, durchsucht INFORMIX die Dateiverzeichnisse /usr/a, /usr/b und /usr/c.

Umgebungsvariable DBPRINT

bestimmt das Ausgabeprogramm für die Listen des Programms.

Standardwert: `DBPRINT=lpr`

Listen werden mit dem Kommando lpr ausgedruckt.

Umgebungsvariable DBTEMP

bestimmt das Dateiverzeichnis, das die temporären Dateien aufnimmt.

Standardwert: `DBTEMP=/tmp`

Temporäre Dateien werden im Dateiverzeichnis /tmp abgelegt.

Umgebungsvariable INFORMIXDIR

bestimmt interne Dateiverzeichnisse von INFORMIX.

Standardwert: `INFORMIXDIR=/usr/lib/informix`

Die INFORMIX-Dateien sind unter /usr/lib/informix abgelegt.

Umgebungsvariable SQLEXEC

Die Umgebungsvariable SQLEXEC muß gesetzt sein, wenn INFORMIX-SE und INFORMIX-ONLINE auf dem gleichen Rechner installiert sind und Sie Zugriff auf INFORMIX-SE haben wollen.

Beispiel: `SQLEXEC=$INFORMIXDIR/lib/sqlexec`

Ist SQLEXEC nicht gesetzt, ist INFORMIX-ONLINE der Standard.

Standardwert: `SQLEXEC=$INFORMIXDIR/lib/sqlturbo`

Umgebungsvariable TBCONFIG

Die Umgabungsvariable TBCONFIG gibt das INFORMIX-ONLINE-System an, wenn mehrere Systeme auf einem Rechner installiert sind.

Standardwert: `$DINFORMIXDIR/etc/tbconfig`

6 ACE und PERFORM Anschluß

Kapitelübersicht

Dieses Kapitel behandelt die Schnittstelle zwischen Funktionen der Programmiersprache C und den INFORMIX-SQL Komponenten PERFORM und ACE. Steht Ihnen nur INFORMIX-ESQL/C zur Verfügung, ist dieses Kapitel für Sie irrelevant und kann übersprungen werden.

Die Produktkomponenten ACE und PERFORM sind mächtige Listen- und Formatgeneratoren des relationalen Datenbanksystems INFORMIX-SQL von Informix. Zur Übersetzung eines vom Benutzer erstellten Programms, das eine bestimmte Liste oder ein Bildschirmformat beschreibt, verwendet ACE das Programm ACEPREP und PERFORM das Programm FORMBUILD.

Dieses Kapitel ergänzt die Informationen, die im Handbuch INFORMIX-SQL Nachschlagen enthalten sind, und beschreibt, wie Sie C-Funktionen sowohl für ACE als auch für PERFORM vom Quellprogramm aus aufrufen können. In diesem Kapitel wird vorausgesetzt, daß Sie als Anwender von PERFORM und ACE mit dem Stoff der entsprechenden Kapiteln des Handbuches INFORMIX-SQL Nachschlagen vertraut sind.

Normalerweise können sowohl ACE als auch PERFORM alle Ihre Datenbanklisten und Bildschirmdialoge ohne zusätzliche Änderung bearbeiten, es könnte jedoch einmal vorkommen, daß Sie eine von ACE bzw. PERFORM nicht bereitgestellte Funktion hinzufügen müssen. 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. PERFORM könnte C-Funktionen zur Prüfung der Gültigkeit von Daten, zur Aufzeichnung des Datums, der Zeit und des Namens der Person, die die Sätze aktualisiert, sowie zur Überarbeitung und Aktualisierung der Datenbank aufrufen.

C-Funktionen können sich der in diesem Handbuch in Kapitel 4 beschriebenen Bibliotheksfunktionen bedienen. Was PERFORM betrifft, so können C-Funktionen auch die am Ende dieses Kapitels erläuterten Sonderfunktionen verwenden. C-Funktionen können INFORMIX-ESQL/C-Anweisungen enthalten und Mathematikfunktionen oder andere C-Funktionen aufrufen.

Da für Sie der Aufruf benutzerdefinierter C-Funktionen möglich ist, erhöht sich natürlich die Leistungsfähigkeit und Flexibilität von ACE und PERFORM.

Dieses Kapitel beschäftigt sich mit dem Aufruf von C-Funktionen innerhalb der ACEPREP- und FORMBUILD-Programme, mit dem Aufbau der C-Funktionen und der Generierung eigener Versionen von ACE und PERFORM, die Ihre C-Funktion(en) enthalten.

ACE-Listenprogramme

Das allgemeine Format eines ACE-Listenprogramms zeichnet sich durch sieben Abschnitte aus:

DATABASE-Abschnitt (obligatorisch)
DEFINE-Abschnitt
INPUT-Abschnitt
OUTPUT-Abschnitt
SELECT-Abschnitt (SELECT oder READ obligatorisch)
READ-Abschnitt (SELECT oder READ obligatorisch)
FORMAT-Abschnitt (obligatorisch)

Der Aufruf einer C-Funktion innerhalb eines Listenprogramms erfolgt durch Vereinbarung des Funktionsnamens im DEFINE-Abschnitt und durch Verwendung der Funktion 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
ist ein notwendiges Schlüsselwort.

FUNCTION
ist ein notwendiges Schlüsselwort.

benfunkt
ist der Name, mit dem die C-Funktion im Programm angesprochen wird. benfunkt muß den Syntaxregeln eines ACE-Bezeichners entsprechen.

END
ist ein notwendiges Schlüsselwort.

Hinweise

1. Sie können mehrere Funktionen auf einmal vereinbaren, indem Sie das Schlüsselwort FUNCTION, jeweils gefolgt vom nächsten Funktionsnamen, wiederholen.
2. Schreiben Sie keine Klammern nach dem Funktionsnamen.
3. Neben der FUNCTION-Anweisung können Sie auch PARAM-, VARIABLE- und ASCII-Anweisungen innerhalb des DEFINE-Abschnitts verwenden.

Aufruf von C-Funktionen

Im FORMAT-Abschnitt des Listenprogramms sind ein oder mehrere Kontrollblöcke enthalten, die festlegen, wann ACE eine bestimmte Aktion durchführen wird.

```
FORMAT
        PAGE HEADER-Kontrollblock
        PAGE TRAILER-Kontrollblock
        FIRST PAGE HEADER-Kontrollblock
        ON EVERY ROW-Kontrollblock
        ON LAST ROW-Kontrollblock
        BEFORE GROUP OF-Kontrollblock
        AFTER GROUP OF-Kontrollblock
END
```

Jeder Kontrollblock enthält eine oder mehrere Anweisungen, die die durchzuführende Aktion für ACE definieren. In Kapitel 5 des Handbuches INFORMIX-SQL Nachschlagen werden die von ACE erlaubten Anweisungen beschrieben. Neben den dort definierten Anweisungen können Sie sich auch eines C-Funktionsaufrufs bedienen.

[CALL] benfunkt([ausdr,...])

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

ausdr,...
sind 1 bis 10 Ausdrücke, die durch Kommata getrennt sind.

Hinweise

1. Ein Ausdruck kann von einer einfachen numerischen oder alphabetischen Konstante bis zu einer komplexeren Reihe von Spaltennamen, ACE-Variablen, ACE-Parametern, ACE-Funktionen (wie etwa Mengen- und Datumsfunktionen), unter Hochkommata gestellten Zeichenketten, arithmetischen und logischen Operatoren und Schlüsselwörtern alles sein.
2. ACE-Anweisungen werden aus Schlüsselwörtern und Ausdrücken gebildet. Eine C-Funktion kann in einem Ausdruck überall dort vorkommen, wo eine Konstante verwendet werden kann. In diesem Fall darf das Schlüsselwort CALL nicht eingefügt werden und die C-Funktion muß einen Wert als Ergebnis liefern.

Beispiele

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

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

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

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

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

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.

Übersetzen des Listenprogramms

Sie können ACEPREP zum Übersetzen des Listenprogramms verwenden, egal ob es C-Funktionsaufrufe enthält oder nicht. Geben Sie dem Listenprogramm einen Dateinamen mit dem Suffix .ace, wie beispielsweise progdat.ace. Um ACEPREP für diese Datei aufzurufen, geben Sie folgenden Befehl ein:

```
saceprep progdat
```

Lesen Sie dazu bitte mehr in Kapitel 5 des Handbuches INFORMIX-SQL Nachschlagen.

PERFORM-Formatprogramme

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

DATABASE-Abschnitt (obligatorisch)
SCREEN-Abschnitt (obligatorisch)
TABLES-Abschnitt (obligatorisch)
ATTRIBUTES-Abschnitt (obligatorisch)
INSTRUCTIONS-Abschnitt

Sie können C-Funktionen in den Kontrollblöcken des INSTRUCTIONS-Abschnitts eines Formatprogramms verwenden.

Aufruf der C-Funktion

In Kontrollblöcken können überall dort, wo Sie einen Ausdruck verwenden können, auch C-Funktionen stehen. Eine C-Funktion kann aber auch als Einzelaktion allein stehen. Die Aufrufsyntax für eine benutzerdefinierte Funktion lautet wie folgt:

[CALL] benfunkt([ausdr,...])

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.

ausdr,...
sind 0 bis 10 Ausdrücke. Ein Ausdruck ist definiert als:

- ein Feldbezeichner
- eine Konstante
- ein Mengenfunktion
- eine C-Funktion
- das Schlüsselwort TODAY

- das Schlüsselwort CURRENT
- jede beliebige Kombination der vorhergehenden Ausdrücke, die durch die Verwendung der arithmetischen Operatoren +,-,* und / gebildet wird.

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()
```

Wenn Sie mehr über die LET- und IF-THEN-ELSE-Aktionen und Ausdrücke wissen möchten, verweisen wir auf das Handbuch INFORMIX-SQL Nachschlagen.

ON BEGINNING und ON ENDING

ON BEGINNING und ON ENDING sind zwei Kontrollblöcke, die nur zusammen mit Aufrufen von C-Funktionen verwendet werden können. Diese Kontrollblöcke sind im INSTRUCTIONS-Abschnitt des Formatprogramms angegeben. ON BEGINNING startet unmittelbar nach dem Aufruf von PERFORM und ON ENDING unmittelbar nach dem Befehl EXIT.

- Mit ON BEGINNING können Sie Befehle eingeben, ein spezielles Paßwort anfordern oder eine temporäre Arbeitsdatei zur Speicherung eines Stapels von Transaktionssätzen anlegen.
- Mit ON ENDING können Sie Berechnungen durchführen, um die während der gerade zu Ende gegangenen PERFORM-Sitzung in der Datenbank getätigten Änderungen zusammenzufassen, Zusammenstellungen der hinzugefügten Sätze zu drucken oder Arbeitsdateien zu löschen.

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

```
ON BEGINNING
        CALL benfunkt(ausdr,...)

ON ENDING
        CALL benfunkt(ausdr,...)
```

Die Syntaxerklärung entspricht jener im vorhergehenden Abschnitt.

Übersetzen des Formatprogramms

Sie können FORMBUILD zur Übersetzung des Formatprogramms verwenden, egal ob Ihr Formatprogramm C-Funktionsaufrufe enthält oder nicht. Geben Sie dem Formatprogramm einen Dateinamen mit dem Suffix .per, wie beispielsweise progdat.per. Um FORMBUILD mit dieser Datei als Argument aufzurufen, geben Sie folgenden Befehl ein:

sformbld progdat

Weitere Informationen stehen im Handbuch INFORMIX-SQL Nachschlagen.

Schreiben des C-Programms

Sprechen Sie innerhalb eines ACE-Listenprogramms oder eines PERFORM-Formatprogramms C-Funktionen an, so ist eine eigene Version von ACE oder PERFORM zu erzeugen, der diese Funktionen bekannt sind. Sie müssen 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. Dieser Abschnitt zeigt Ihnen, wie Sie Ihr C-Programm aufzubauen, Ihre Funktionen zu vereinbaren haben und wie Werte an Ihre Funktionen übergeben und von diesen zurückgeliefert werden.

C-Programmstruktur

Zur Erzeugung einer Version von ACE oder PERFORM, die 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.

Nachstehende Programmliste zeigt die allgemeine Struktur eines solchen C-Programms, das zwei benutzerdefinierte Funktionen beinhaltet:

```
#include "ctools.h"
/* Weitere includes moeglich wie benoetigt */

valueptr funct1();
valueptr funct2();

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

/* Weitere globale Vereinbarungen hinzufuegen */

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

valueptr funct2(arg1, arg2)
valueptr arg1, arg2;
{
      .
      .
      .
/* funct2 erhaelt zwei Argumente
   und liefert kein Ergebnis      */
      .
      .
      .
}
```

Der Aufbau der Struktur ist im folgenden erläutert.

1. Am Beginn Ihres C-Programms muß folgende Zeile stehen:

```
#include "ctools.h"
```

   Die Include-Datei ctools.h ist im Anhang aufgelistet.

   Sie können natürlich - je nach Anwendung - andere Dateien, wie etwa math.h oder stdio.h, einfügen. Wenn Sie INFORMIX-ESQL/C verwenden, können Sie sqlca.h und andere Include-Dateien einfügen.

2. Bevor Sie das erforderliche Struktur-Array **ufunc** initialisieren, müssen Sie Ihre Funktionen vereinbaren. ctools.h beinhaltet die Definition der Struktur **value** sowie der 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();
```

   Wenn sie die Initialisierung des Arrays **userfuncs** nach der Definition Ihrer Funktionen plazieren, können Sie diesen Schritt überspringen.

3. 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.

4. Den letzten Teil des C-Programms bildet der Code Ihrer Funktionen. 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 in einem späteren Abschnitt beschrieben.

Eingabeparameter

Die Include-Datei ctools.h soll die Schnittstelle zwischen C-Funktionen und ACE bzw. PERFORM vereinfachen. Wenn beispielsweise der an die C-Funktion übergebene Parameter arg ist, können mit INFORMIX-SE bzw. INFORMIX ONLINE die folgenden Definitionen zur Feststellung des Datentyps von arg und zur Extraktion des Wertes von arg verwendet werden, gleichgültig um welchen Datentyp es sich dabei handelt.

```
arg->v_charp     Zeiger auf string
arg->v_len       Länge von string
arg->v_int       INTEGER-Wert
arg->v_long      LONG-Wert
arg->v_float     FLOAT-Wert
arg->v_double    DOUBLE-Wert
arg->v_decimal   DECIMAL-, MONEY-, DATETIME-, oder
                 oder INTERVAL-Wert
arg->v_type      Datentyp
arg->v_ind       Indikatorvariabel auf NULL
arg->v_prec      DATETIME bzw. INTERVAL-Datumskomponente
```

nur unter ONLINE

```
arg->v_charp     Zeiger auf VARCHAR
arg->v_len       Länge von VARCHAR
arg->v_blocator  Zeiger auf eine Locator-Struktur für
                 BYTE oder TEXT
```

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
		nur unter ONLINE
SQLVCHAR	VARCHAR	char
SQLTEXT	TEXT	loc_t
SQLBYTES	BYTE	loc_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.

Datentyp-Konvertierung

Die Include-Datei ctools.h bietet eine Alternative zur Bestimmung des von ACE oder PERFORM übernommenen Parametertyps. Die unten angeführten Funktionen führen die Konvertierung eines Parameters, der als Zeiger auf eine Struktur vom Typ **value** übergeben wurde, in einen C-Datentyp Ihrer Wahl durch.

Funktion	**Ergebnis**
toint	int
tolong	long
tofloat	double
todouble	double
todate	long
todecimal	dec_t
todatetime	dtime_t
tointerval	intrvl_t

Diese Funktionen übernehmen einen Zeiger auf eine Struktur vom Typ **value** und liefern als Ergebnis einen Wert des angegebenen Typs. **todecimal** hat ein zweites Argument, das ein Zeiger auf eine Struktur vom Typ **dec_t** ist. **todatetime** hat ein zweites Argument, das ein Zeiger auf eine Struktur vom Typ **dtime_t** ist. **tointerval** hat ein zweites Argument, das ein Zeiger auf eine Struktur vom Typ **intrvl_t** ist. 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.

Ergebnisse

Wenn Ihre Funktion einen Wert an ACE oder PERFORM liefern 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, sind in ctools.h folgende Makros enthalten:

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

Verwenden Sie das entsprechende Makro selbst 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 folgenden Makro aus der Include-Datei ctools.h verwenden, um VARCHAR-Werte zurückzugeben:

```
vcharreturn(s,c)
  char *s;
  int   c;
```

Sie können keine TEXT und BYTE zurückgeben.

Spezielle Bibliotheksfunktionen von PERFORM

Fünf C-Funktionen sollen 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.

Hinweis

Das Argument von **pf_getval** und **pf_putval**, das sich auf den gelieferten oder ausgegebenen Wert bezieht, muß ein Zeiger auf die den Wert enthaltende Variable sein. Ein oft begangener Programmierfehler besteht darin, die Variable selbst zu verwenden. Dies führt zu einem Laufzeit-Systemfehler, den der Compiler nicht erkennt.

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 Bildschirmfel des tagname beschreibt.

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

Hinweis

Hier eine Aufstellung der möglichen Werte von type. Sie sind in ctools.h definiert:

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

Returnwert

0	Die Funktion war erfolgreich; das Bildschirmfeld wurde 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, INTERVAL oder VARCHAR, die **pf_getval** liefern kann. Bei TEXT und BYTE zeigt *retvalue* auf eine loc_t-Struktur. Perform kopiert seine interne Locator-Struktur auf Ihre Struktur.

valtype
ist eine Zahl von Typ short integer. Sie gibt den Typ des Wertes an, auf den retvalue zeigen sollte. Die Angabe für valtype für VARCHAR und Blob ist folgende: valtype: CVCHARTYPE SQLTyp: SQLVCHAR
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 muß vallen die Anzahl an Bytes enthalten, die die Puffer haben kann.

Hinweise

1. Für valtype gibt es folgende Auswahlmöglichkeiten:

valtype	**SQL-Typ**
CCHARTYPE CFIXCHARTYPE CSTRINGTYPE	CHAR
CINTTYPE	INTEGER
CSHORTTYPE	SMALLINT
CLONGTYPE	INTEGER, DATE, SERIAL
CFLOATTYPE	SMALLFLOAT
CDOUBLETYPE	FLOAT
CDECIMALTYPE	DECIMAL, MONEY
CDATETIME	DATETIME
CINTERVAL	INTERVAL
	nur unter ONLINE
CVCHARTYPE	VARCHAR
CLOCATORTYPE	SQLTEXT bzw. SQLBYTES

2. 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.

3. 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, ist die Zahl, auf die retvalue zeigt, 0. Ist valtype vom Typ CHAR und das Bildschirmfeld ein numerisches Feld, erfolgt eine Konvertierung in eine Zeichenkette. Entspricht die Zeichenkette nicht der von vallen spezifizierten Länge, enthält retvalue die gekürzte Zeichenkette mit einem Nullbyte am Ende.

Returnwert

0	Die Funktion war erfolgreich; das Bildschirmfeld wurde 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, INTERVAL oder VARCHAR. Der Wert wird in das durch tagname bestimmte Bildschirmfeld eingetragen. Bei TEXT und BYTE zeigt pvalue auf eine loc_t-Struktur. 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 Blobwert 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.

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

Hinweise

1. Für valtype gibt es folgende Auswahlmöglichkeiten:

valtype	**SQL-Typ**
CCHARTYPE, CFIXCHARTYPE, CSTRINGTYPE	CHAR
CINTTYPE	INTEGER
CSHORTTYPE	SMALLINT
CLONGTYPE	INTEGER, DATE
CFLOATTYPE	SMALLFLOAT
CDOUBLETYPE	FLOAT
CDECIMALTYPE	DECIMAL, MONEY
CDATETIME	DATETIME
CINTERVAL	INTERVAL
	nur unter ONLINE
CVCHARTYPE	VARCHAR
CLOCATORTYPE	SQLTEXT bzw. SQLBYTES

2. Der Datentyp von pvalue wird durch den Wert des Parameters valtype bestimmt.
3. 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.
4. 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.
5. Paßt ein Zahlenwert nicht in ein numerisches Bildschirmfeld, wird das Feld mit Sternchen gefüllt.

Returnwert

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.

Hinweise

1. Wird die Funktion **pf_nxfield** während eines BEFORE EDITADD- oder BEFORE EDITUPDATE-Vorgangs einer Tabelle aufgerufen, legt sie das erste zu bearbeitende Bildschirmfeld fest.
2. Wird **pf_nxfield** während eines AFTER EDITADD- oder AFTER EDITUPDATE-Vorganges einer Tabelle aufgerufen, so wird der Cursor auf das angegebene Bildschirmfeld tagname zur weiteren Bearbeitung gesetzt und der Satz nicht eingetragen.
3. Wird die Funktion **pf_nxfield** entweder vor einem BEFORE EDITADD/EDITUPDATE-Vorgang oder nach einem AFTER EDITADD/EDITUPDATE-Vorgang eines Feldes aufgerufen, legt sie das nächste zu bearbeitende Feld fest.

4. Wird die Funktion **pf_nxfield** entweder nach einem AFTER ADD- oder AFTER UPDATE-Vorgang aufgerufen, ist sie wirkungslos, da der Satz bereits eingetragen worden ist.
5. 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.

Returnwert

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.

reverseflag
: ist eine Zahl von Typ short integer, die angibt, ob die Meldung in Inversdarstellung ausgegeben wird (0 = normale Anzeige, 1 = invers).

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

Hinweise

1. 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.
2. 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.

Der Übersetzungs-, Binde- und Ablaufprozeß

Nach Erstellung der Datei, die Ihre C-Funktionen enthält, müssen Sie die C-Funktionen übersetzen und die notwendigen Bibliotheksfunktionen hinzufügen, 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

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

Shell-Skripts werden zur Vereinfachung des Übersetzungs- und Bindeprozesses bereitgestellt. Sie können diese Shell-Skripts genauso verwenden wie das standardmäßige C-Übersetzungs- und Bindeprogramm **cc**. Die Kenntnis der Namen spezieller ACE-, PERFORM- oder INFORMIX-ESQL/C-Bibliotheken oder Präprozessoren sowie der Verzeichnisse der Include-Dateien dieser Programme ist für Sie nicht notwendig.

```
[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.

-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.

Hinweis

Sie können mehrere C-Programme gleichzeitig übersetzen.

Beispiele

Dieser Abschnitt behandelt 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.

ACE

Beipiel 1

Der folgende Abschnitt zeigt ein Quellprogramm, das eine Benutzerfunktion zur Ausführung eines Systemkommandos aufruft. Der Name des Programms in der Beispieldatenbank lautet a_ex1.ace. Abschnitt 2 des Programms zeigt die Benutzerfunktion **to_unix.c**.

```
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
```

Abschnitt 2:

```
#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);
}
```

Beipiel 2

Der nächste Abschnitt zeigt ein ACE-Programm, das eine C-Funktion aufruft, die die Quadratwurzel einer Zahl vom Typ DECIMAL berechnet. Die C-Funktion verwendet einige der in Kapitel 4 beschriebenen Dezimalfunktionen.

Das Programm a_ex2.ace der Beispieldatenbank berechnet den Mittelwert und die Standardabweichung der Gesamtkosten aller Aufträge der Datenbank versand. Abschnitt 2 zeigt die Benutzerfunktion **decsqrt.c**.

```
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
```

Abschnitt 2:

```
#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);
}
```

PERFORM

Beispiel 1

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.

Abschnitt 1 zeigt ein Formatprogramm zur Aufnahme neuer Kunden in die Datenbank versand. 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 START, wird die C-Funktion **stamptime** aufgerufen.

Abschnitt 1:

```
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
```

Abschnitt 2 zeigt die Funktion **stamptime**. Diese wird vom Formatprogramm aufgerufen, wenn der Erfasser zur Beendigung der Transaktion START drückt. Zusätzlich zu der in diesem Kapitel bereits definierten Sonderfunktion **pf_putval** verwendet **stamptime** die Systemfunktionen **time**, **localtime** und **getlogin**.

Die Benutzerkennung 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 vom Typ string das Bildschirmfeld Zeitpunkt der Erfassung geschrieben. Danach nimmt PERFORM den Satz unter Verwendung der Daten des Bildschirmes in die Tabelle kunde auf.

Abschnitt 2:

```
#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

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 von START holt Ihr C-Programm die Daten vom Bildschirm und schreibt die Sätze in die Tabellen.

Abschnitt 1 zeigt ein Auftragsformular. Es können darin 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-Felder 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 eintippen.

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 solange ein Artikel nach dem anderen bearbeitet, bis Sie START 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.

Abschnitt 1:

```
database versand

screen
{
                              AUFTRAGSFORMULAR

   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 START die Funktion **add_order**. In Abschnitt 2 werden diese Funktionen dargestellt. Neben den in diesem Kapitel definierten C-Funktionen verwendet das Programm einige Funktionen aus dem Kapitel Bibliotheksfunktionen sowie INFORMIX-ESQL/C.

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 in mult_item.ec enthalten.

Abschnitt 2:

```
#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);
}
```

A Anhang

A.1 C-Beispiele

Einführung in die C-Beispiele

Das Beispeilprogramm unload bildet die INFORMIX-Anweisung UNLOAD nach. Die zu entladenden Daten werden auf stdout ausgegeben. Das Format der Ausgabe-Daten ist mit denen der UNLOAD-Anweisung identisch. Das Feldtrennzeichen ist | (senkrechter Strich). Es kann über die Umgebungsvariable DBLIMITER verändert werden. Da die UNLOAD-Anweisung nur interaktiv und bei INFORMIX-4GL verwendet werden kann, ist es sinnvoll das Beispielprogramm einzugeben.

Aufruf des Programmes : unload dbname select

dbname : Name der Datenbank, aus der die Daten entladen werden sollen.

select : Beliebige SELECT-Anweisung (jedoch ohne die Klausel INTO TEMP ...). Die SELECT-Anweisung ist nur ein Parameter, sie muß daher in ' oder " eingeschlossen werden.

Beispiel:

Der gesamte Inhalt der Tabelle kunde wird in die Datei kunde.unl entladen.

```
unload firma 'select * from kunde' >kunde.unl
```

Das Beispeilprogramm load bildet die INFORMIX-Anweisung LOAD nach. Die zu ladenden Daten werden von stdin gelesen. Das Format der Eingabe-Daten ist mit denen der LOAD-Anweisung identisch. Das Feldtrennzeichen ist | (senkrechter Strich). Es kann über die Umgebungsvariable DBLIMITER verändert werden.

Da die UNLOAD-Anweisung nur interaktiv und bei INFORMIX-4GL verwendet werden kann, ist es sinnvoll das Beispielprogramm einzugeben.

Aufruf des Programmes: load dbname insert

dbname : Name der Datenbank, in die die Daten geladen werden sollen.

insert : INSERT-Anweisung in folgendem Format:

INSERT INTO tabelle [(feld,...)] [VALUES (wert,...)]

Ist die Angabe (feld,...) vorhanden, so werden durch das Programm lediglich die genannten Felder angesprochen. Alle übrigen Felder der Tabelle werden mit NULL-Value gefüllt. Sie dürfen daher nicht mit NOT NULL definiert worden sein. Fehlt die Angabe (feld,...) werden alle Felder der Tabelle angesprochen.

Ist die VALUES-Klausel angegeben, so mueßen für alle angesprochenen (s.o.) Felder entweder Werte (Konstanten) oder ? als Platzhalter angegeben werden. Die einzufügenden Werte werden in diesem Falle 1:1 den Fragezeichen zugeordnet (d.h.: Die Eingabe-Datei muß genau so viele Werte je Satz enthalten, wie Fragezeichen angegeben wurden). Fehlt die Klausel VALUES, so werden die Werte der Eingabedatei je Satz 1:1 den angesprochenen (s.o.) Feldern der Tabelle zugeordnet.

Die INSERT-Anweisung ist nur ein Parameter, sie muss daher in ' oder " eingeschlossen werden.

Beispiel 1:

Die Datei kunde.unl soll in die Tabelle kunde geladen werden. Die Datei kunde.unl enthält für alle Felder der Tabelle einen durch das Zeichen | abgeschlossenen Wert.

load firma 'insert into kunde' <kunde.unl

Beispiel 2:

Die Tabelle kunde hat folgenden Aufbau:

```
nr      serial,
kname   char(30),
plz     integer,
ort     char(30)
```

Die Datei name.unl enthält Namen von Kunde, die alle im selben Ort wohnen. Diese Namen sollen nun in die Tabelle kunde geladen werden.

Inhalt der Datei name.unl:

```
Schmidt Hans| Maier Peter| Mueller Klaus|
```

```
load firma 'insert into kunde
            values (0, ?, 8000, "Muenchen")' <name.unl
```

Beispiel1 unload

```
/*
 * unload:
 * Beispeilprogramm das die INFORMIX-Anweisung UNLOAD nachbildet.
 * Die zu entladenden Daten werden auf stdout ausgegeben. Das
 * Format der Ausgabe-Daten ist auch mit denen der UNLOAD-Anweisung
 * identisch.
 *
 * Aufruf des Programmes:   unload dbname select
 *
 * Das Argument select ist eine SQL SELECT-Anweisung (jedoch ohne
 * die Klausel INTO TEMP ...). Die SELECT-Anweisung ist nur ein
 * Parameter, sie muss daher in ' oder " eingeschlossen werden.
 *
 */

$include sqlca;          /* fuer Struktur zur Fehlerbehandlung      */
$include sqlda;          /* fuer dynamisch formulierte SQL-Anweisungen */
$include sqltypes;       /* fuer SQL- und C-Datentypen              */

#include <stdio.h>
#include <decimal.h>

typedef struct sqlvar_struct var_t;

/*
 * Funktionen , die nicht ( int ) zurueckliefern
 */

char *rtypalign();
char *malloc();
char *getenv();

#define TRUE  1
#define FALSE 0
#define FUNCOK  0
#define FUNCERR 1

#define NSTRSIZE 40              /* Max. Laenge num. Zeichen */

#define DELIMITER '|'            /* DEFAULT-Trennzeichen */

/*
 * Globale Definitionen
 */

FILE *outfile = stdout;          /* Ausgabe-Datei */

char delimiter = DELIMITER;      /* Feldtrennzeichen */

long rowcnt = 0;                 /* Anzahl ausgegebener Saetze */

/*
 * Zeiger auf (anzufordernden) Speicher fuer Satzpuffer
 */

char *buffer;


main(argc, argv)
int argc;
char *argv[];

{
register char *delim;
$char *dbname;

$whenever error continue;    /* Kein Programmabbruch durch SQL-Fehler */

if (argc <= 2)               /* Zuwenig Argumente angegeben ? */
    {
    fprintf(stderr, "Gebrauch: %s dbname select\n", argv[0]);
```

```
    exit(1);
    }

/* Initialisierung */

dbname = argv[1];           /* Parameter 1 ist der Name der DB */

$database $dbname;          /* Datenbank eroeffnen */

if (sqlca.sqlcode != 0L)    /* DATABASE-Anweisung erfolgreich
                               ausgefuehrt ? */
    {
    fprintf(stderr,
        "Datenbank '%s' kann nicht eroeffnet werden.\n", dbname);
    prsqlerror((char *) 0);     /* SQL-Fehlermeldung ausgeben */
    exit(1);
    }

delim = getenv("DBDELIMITER");  /* Environment Variable DBDELIMITER */
if (delim != (char *) 0)
    delimiter = *delim;         /* 1. Zeichen ersetzt DEFAULT-Wert */

/* Verarbeitung */

if (unload(argv[2]) != FUNCOK)  /* Ausfuehren UNLOAD */
    exit(1);

/* Abschluss */

fprintf(stderr, "Saetze ausgegeben: %ld\n", rowcnt);

$close database;                /* Datenbank schliessen */

if (sqlca.sqlcode != 0L)        /* Datenbank geschlossen ? */
    {
    fprintf(stderr,
        "Datenbank '%s' kann nicht geschlossen werden\n", dbname);
    prsqlerror((char *) 0);
    exit(1);
    }

exit(0);
}


/*
 * unload()
 * Diese Funktion "prepariert" die als Argument uebergebene
 * SELECT-Anweisung und baut mit DESCRIBE eine sqlda-Struktur
 * auf, die mit der Funktion fillsqlda() vervollstaendigt wird.
 * Der Lese-CURSOR wird deklariert und die Funktion runselect()
 * aufgerufen.
 */

unload(selectstmt)
$char *selectstmt;                  /* SELECT-Anweisung */

{
register int retc;
struct sqlda *pdescr;               /* Zeiger auf sqlda-Struktur */

$prepare selectid from $selectstmt; /* SELECT-Anweisung voruebersetzen */

if (sqlca.sqlcode != 0L)            /* Fehler bei PREPARE ? */
    {
    prsqlerror("Fehler beim Vorbereiten der SELECT-Anweisung");
    return FUNCERR;
    }

$describe selectid into pdescr;     /* SELECT-Anweisung analysieren */

if (sqlca.sqlcode != 0L)            /* SELCET-Anweisung angegeben ? */
    {
    prerror("Keine SELECT-Anweisung angegeben");
```

```
    return FUNCERR;
    }

if (fillsqlda(pdescr) != FUNCOK)      /* sqlda-Struktur fuellen;
                                       * Fehler bei fillsqlda ?
                                       */
    {
    prerror("Fehler bei der Speicheranforderung");
    return FUNCERR;
    }

/*
 * Satzzeiger fuer SELECT-Anweisung ohne INTO TEMP vereinbaren
 */

$declare selectcurs cursor for selectid;

retc = runselect(pdescr);             /* Ablauf SELECT */

free(buffer);                         /* Speicher freigeben */

return retc;
}


/*
 * runselect()
 * Eroeffnen des CURSOR's zum Lesen aus der DB.
 * Mit fetch werden dann Datensaetze gelesen und
 * mit der Funktion output() ausgegeben.
 */

runselect(pdescr)
struct sqlda *pdescr;              /* Zeiger auf die sqlda-Struktur */

{
register int retc;

$open selectcurs;                  /* Satzzeiger oeffnen */

if (sqlca.sqlcode != 0L)           /* Fehler bei OPEN ? */
    {
    prsqlerror("Fehler beim Eroeffnen des Cursors");
    return FUNCERR;
    }

retc = FUNCOK;

while (TRUE)
    {

/*
 * Mittels Zeiger auf sqlda-Struktur Satzzeiger positionieren
 */

    $fetch selectcurs using descriptor pdescr;

    if (sqlca.sqlcode == SQLNOTFOUND)   /* Ende erreicht ? */
        break;

/*
 * FETCH-Anweisung erfolgreich ausgefuehrt ?
 */

    if (sqlca.sqlcode != 0L)
        {
        prsqlerror("Fehler beim Lesen aus der DB");
        retc = FUNCERR;
        break;
        }

    if (output(pdescr) != FUNCOK)       /* Satz ausgeben;
                                         * Fehler bei output() ?
                                         */
```

```
        {
        retc = FUNCERR;
        break;
        }

    rowcnt++;                           /* Ausgegebene Saetze zaehlen */
    }

$close selectcurs;                      /* Satzzeiger schliessen */

if (sqlca.sqlcode != 0L)                /* Fehler bei CLOSE ? */
    {
    prsqlerror("Fehler beim Schliessen des Cursers");
    return FUNCERR;
    }

return retc;
}


/*
 * fillsqlda()
 * Diese Funktion traegt in die mit DESCRIBE aufgebauten sqlda-Struktur
 * die Typen der C-Variablen ein in denen die einzelnen
 * Datenfelder bereitgestellt werden sollen. Im selben Durchlauf
 * wird Groesse des anzufordernden Speichers errechnet.
 * Dann wird der errechnete Speicherbedarf angefordert.
 * In einem zweiten Durchlauf werden die sqldata-Zeiger versorgt.
 */

fillsqlda(psqlda)
struct sqlda *psqlda;           /* Zeiger auf sqlda-Struktur */

{
register var_t *pvar;
register char *workp;
register int len;
register int i;

workp = (char *) 0;

pvar = psqlda->sqlvar;          /* Initialisierung mit Startadresse */

/*
 *  Eintragen des Typs der C-Variable mittels setctype();
 *  Berechnen des benoetigten Speicherplatzes
 */

for (i = psqlda->sqld; i > 0; i--)
    {
    setctype(pvar);
    len    = rtypmsize(pvar->sqltype, pvar->sqllen);
    workp = rtypalign(workp, pvar->sqltype);

    pvar->sqldata = workp;
    pvar->sqlind  = (short *) 0;

    workp += len;
    pvar++;
    }


/*
 * Anfordern des errechneten Speicherbedarfs
 */

buffer = malloc((unsigned int) (workp - (char *) 0));
if (buffer == (char *) 0)       /* kein Speicherbedarf ? */
    return FUNCERR;

workp = buffer;

pvar = psqlda->sqlvar;          /* Initialisierung mit Startadresse */
```

```
/*
 * 'sqldata'-Zeiger versorgen
 */

for (i = psqlda->sqld; i > 0; i--)
    {
    pvar->sqldata = workp  + (pvar->sqldata - (char *) 0);
    pvar++;
    }

return FUNCOK;
}


/*
 * setctype()
 * In die Variable sqltype wird der gewuenschte
 * C-Datentyp eingetragen.
 */

setctype(pvar)
register var_t *pvar;

{

/*
 * SQL-Datentyp in entsprechenden C-Datentyp umwandeln
 */

switch (pvar->sqltype & SQLTYPE)
    {
    case SQLSMINT:
        pvar->sqltype = CSHORTTYPE;
        break;

    case SQLSERIAL:
    case SQLINT:
        pvar->sqltype = CLONGTYPE;
        break;

    case SQLSMFLOAT:
    case SQLFLOAT:
        pvar->sqltype = CDECIMALTYPE;
        pvar->sqllen  = PRECMAKE(16,255);
        break;

    case SQLMONEY:
    case SQLDECIMAL:
        pvar->sqltype = CDECIMALTYPE;
        break;

    case SQLDATE:
        pvar->sqltype = CDATETYPE;
        break;

    default:
        pvar->sqltype = CFIXCHARTYPE;
        break;
    }
}


/*
 * output()
 * Mit dieser Funktion wird ein ueber die sqlda-Struktur
 * adressierter Datensatz ausgegeben. Das Format der Ausgabe
 * entspricht dabei dem der INFORMIX-Anweisung UNLOAD.
 */

output(psqlda)
struct sqlda *psqlda;  /* Zeiger auf die sqlda-Struktur */

{
register var_t *pvar;
```

```
register int i;
char numstr[NSTRSIZE + 1];

pvar = psqlda->sqlvar;
for (i = psqlda->sqld; i > 0; i--)

/* C-Variable ungleich Null-Wert ? */

    {
    if (!risnull(pvar->sqltype, pvar->sqldata))
        {
        switch (pvar->sqltype)
            {
            case CSHORTTYPE:
                sprintf(numstr, "%d", *((short *) pvar->sqldata));
                fputs(numstr, outfile);
                break;

            case CLONGTYPE:
                sprintf(numstr, "%ld", *((long *) pvar->sqldata));
                fputs(numstr, outfile);
                break;

            case CDECIMALTYPE:
                dectoasc((dec_t *) pvar->sqldata, numstr, NSTRSIZE, -1);
                numstr[byleng(numstr, NSTRSIZE)] = '\0';
                dpointcomma(numstr);
                fputs(numstr, outfile);
                break;

            case CDATETYPE:
                rdatestr(*((long *) pvar->sqldata), numstr);
                fputs(numstr, outfile);
                break;

            default:
                outstr(pvar->sqldata, pvar->sqllen);
                break;
            }
        }

    fputc(delimiter, outfile);
    pvar++;
    }

fputc('\n', outfile);

return FUNCOK;
}


/*
 * outstr()
 * Ausgeben Zeichenreihe
 */

outstr(str, len)
char *str;
int len;

{
register char *cp;
register int count;

cp    = str;
count = byleng(cp, len);

/*
 * Mindestens 1 Zeichen muss ausgegeben werden.
 */

do
    {
    if (((*cp == delimiter) || (*cp == '\\')) && (delimiter != '\\'))
```

```
        fputc('\\', outfile);
    fputc(*(cp++), outfile);
    }
while (--count > 0);

}


/*
 * prsqlerror()
 * Ausgeben SQL-Fehlermeldung
 */

prsqlerror(errmsg)
char *errmsg;

{
if (errmsg != (char *) 0)
    prerror(errmsg);

fprintf(stderr, "SQLCODE: %ld\n", sqlca.sqlcode);
}

/*
 * prerror()
 * Ausgeben Fehlermeldung.
 */

prerror(errmsg)
char *errmsg;

{
fprintf(stderr, "%s.\n", errmsg);
 }
```

Beispiel2 load

```
/*
 * load:
 * Beispeilprogramm das die INFORMIX-Anweisung LOAD nachbildet.
 * Die zu ladenden Daten werden von stdin gelesen. Das Format
 * der Eingabe-Daten ist auch mit denen der LOAD-Anweisung
 * identisch.
 *
 * Aufruf des Programmes:   load dbname insert
 *
 * Das Argument insert ist eine INSERT-Anweisung (jedoch ohne die
 * Klausel SELECT ...). Die Klausel VALUES kann wahlweise ange-
 * geben werden. Die INSERT-Anweisung ist nur ein Parameter, sie
 * muss daher in ' oder " eingeschlossen werden.
 *
 */

$include sqlca;      /* fuer Struktur zur Fehlerbehandlung             */
$include sqlda;      /* fuer dynamisch formulierte SQL-Anweisugen */
$include sqltypes;   /* fuer SQL- und C-Datentypen                   */
$include sqlstype;   /* fuer SQL-Anweisungstypen                     */

#include <stdio.h>
#include <ctype.h>

typedef struct sqlvar_struct var_t;

#define TRUE     1
#define FALSE    0

#define FUNCOK   0
#define FUNCERR  1
#define EOFINP  -1

#define MAXBUF (16*1024) /* Maximale Laenge des Eingabesatzes */

#define DELIMITER '|'    /* DEFAULT-Trennzeichen */

#define NAMELEN 18       /* Max. Laenge eines Tabellen-/Feldnamens */

#define LOCKMODE "share" /* Sperrmodus fuer die Tabelle */

/*
 * Art der INSERT-Anweisung
 */

#define INSERTLD   1    /* ohne VALUES oder SELECT                */
#define INSERTVAL  2    /* mit  VALUES; INSERT INTO ... VALUES ... */
#define INSERTSEL  3    /* mit  SELECT; INSERT INTO ... SELECT ... */
#define NOINSERT  -1    /* Kein INSERT */

/*
 * Funktionen, die nicht (int) zurueckliefern
 */

char *malloc();
char *getenv();
var_t *allocvart();
char *chkword();
char *skipspace();
char *skipword();
char *cpystr();

/*
 * Gloable Definitionen
 */

FILE *inpfile = stdin;          /* Eingabedatei */

char delimiter = DELIMITER;     /* Feldtrennzeichen */

$char *insertstmt;              /* Zeiger auf INSERT-Anweisung */
```

```
int inserttype;                 /* Art der INSERT-Anweisung */

char tabname[NAMELEN + 1];      /* Name der Tabelle */

int numcols = 0;                /* Anz. Felder, in die eingefuegt wird */

long rowcnt = 0;                /* Anzahl eingefuegte Saetze */

char buffer[MAXBUF];            /* Speicher fuer Eingabedaten */

main(argc, argv)
int argc;
char *argv[];

{
register char *cp;
$char *dbname;

$whenever error continue; /* Kein Programmabbruch durch SQL-Fehler */

if (argc != 3)            /* Falsche Anzahl Argumente angegeben ? */
    {
    fprintf(stderr, "Gebrauch: %s dbname insert\n", argv[0]);
    exit(1);
    }

/* Initialisierung */

dbname = argv[1];

$database $dbname;        /* Datenbank eroeffnen */

if (sqlca.sqlcode != 0L)  /* DATABASE-Anweisung erfolgreich ausgefuehrt ? */
    {
    fprintf(stderr,
        "Datenbank '%s' kann nicht eroeffnet werden.\n", dbname);
    prsqlerror((char *) 0);
    exit(1);
    }

if (startup(argv[2]) != FUNCOK)
    exit(1);

/* Verarbeitung */

if (load() != FUNCOK)     /* Ausfuehren LOAD */
    exit(1);

/* Abschluss */

fprintf(stderr, "Saetze eingefuegt: %ld\n", rowcnt);

$close database;          /* Datenbank schliessen */

if (sqlca.sqlcode != 0L)  /* Datenbank geschlossen ? */
    {
    fprintf(stderr,
        "Datenbank '%s' kann nicht geschlossen werden.\n", dbname);
    prsqlerror((char *) 0);
    exit(1);
    }
exit(0);
}

/*
 * startup()
 */

startup(inspar)
char *inspar;

{
register char *delim;
```

```
delim = getenv("DBDELIMITER");      /* Environment Variable DBDELIMITER */
if (delim != (char *) 0)
    delimiter = *delim;             /* 1. Zeichen ersetzt DEFAULT-Wert  */

inserttype = parseinsert(inspar);  /* INSERT-Parameter untersuchen */

if (inserttype == NOINSERT)         /* Kein INSERT */
    {
    prerror("Keine oder fehlerhafte INSERT-Anweisung");
    return FUNCERR;
    }

if (inserttype == INSERTSEL)        /* INSERT mit SELECT */
    {
    prerror("INSERT mit SELECT ist hier nicht erlaubt");
    return FUNCERR;
    }

if (numcols == 0)                   /* Anzahl Felder bekannt ? */
    {
    if (getnumcols() != FUNCOK)     /* Aus Systemtabelle ermitteln;
                                     * Fehler bei getnumcols() ?
                                     */
        return FUNCERR;
    }

if (inserttype == INSERTLD)         /* INSERT ohne VALUES */
    {
    bldnewinsert(inspar, buffer);  /* Ergaenze um 'VALUES (?,...)' */
    insertstmt = buffer;
    }
else
    insertstmt = inspar;            /* INSERT mit VALUES */

#ifdef DEBUG
fprintf(stderr, "%s\n", insertstmt);
#endif /* DEBUG */

return FUNCOK;
}

/*
 * load()
 * Steuerung.
 */

load()

{
register struct sqlda *psqlda;
struct sqlda ldsqlda;                   /* Eingabe sqlda-Struktur */

psqlda = &ldsqlda;                      /* Zeiger auf sqlda-Struktur */

psqlda->sqlvar = allocvart(numcols);  /* Variablentabelle aufbauen */

if (psqlda->sqlvar == (var_t *) 0)
    {
    prerror("Fehler bei Anforderung von Arbeitsspeicher");
    return FUNCERR;
    }

$prepare insertid from $insertstmt;   /* INSERT-Anweisung voruebersetzen */

if (sqlca.sqlcode != 0L)              /* Fehler bei PREPARE ? */
    {
    prsqlerror("INSERT-Anweisung fehlerhaft");
    return FUNCERR;
    }

$declare inscur cursor for insertid;  /* Satzzeiger vereinbaren */

/*
 * ESQL/C-Precompiler erkennt Fehler
```

```
 */
$begin work;                            /* Transaktion starten */

if (sqlca.sqlcode != 0L)                /* Transaktion gestartet ? */
    {
    prsqlerror("BEGIN WORK-Anweisung fehlerhaft");
    return FUNCERR;
    }

if (locktable() != FUNCOK)              /* Tabelle sperren;
                                         * Fehler in locktable() ?
                                         */
    {
    prsqlerror("Tabelle kann nicht gesperrt werden");
    $rollback work;                     /* Transaktion zuruecksetzen */

    if (sqlca.sqlcode != 0L)            /* Fehler bei ROLLBACK WORK ? */
        prsqlerror("Fehler beim Zuruecksetzen der Transaktion");

    return FUNCERR;
    }

if (runinsert(psqlda) != FUNCOK)        /* Insert ausfuehren;
                                         * Fehler in runinsert() ?
                                         */
    {
    $rollback work;                     /* s. o. */

    if (sqlca.sqlcode != 0L)
        prsqlerror("Fehler beim Zuruecksetzen der Transaktion");

    return FUNCERR;
    }

$commit work;                           /* Transaktion beenden */

if (sqlca.sqlcode != 0L)                /* Transaktion beendet ? */
    {
    prsqlerror("Fehler beim Beenden der Transaktion");
    return FUNCERR;
    }

free(psqlda->sqlvar);                   /* Speicher wieder freigeben */

return FUNCOK;
}

/*
 * runinsert()
 * Ausfuehren INSERT-Anweisung.
 * Lesen Eingabedaten und schreiben in DB-Tabelle.
 */

runinsert(psqlda)
struct sqlda *psqlda;

{
register int retc;

$open inscur;                           /* Satzzeiger eroeffnen */

if (sqlca.sqlcode != 0L)                /* Fehler ? */
    {
    prsqlerror("CURSOR konnte nicht geoeffnet werden");

    $rollback work;                     /* s. o. */

    if (sqlca.sqlcode != 0L)
        prsqlerror("Fehler beim Beenden der Transaktion");

    return FUNCERR;
    }
```

```
while ((retc = input(psqlda)) == FUNCOK)  /* bis EOF || Fehler in input() */
    {
    $put inscur using descriptor psqlda;  /* Satz schreiben */

    if (sqlca.sqlcode != 0L)              /* Fehler ? */
        {
        prsqlerror("Fehler beim Schreiben in die Tabelle");
        retc = FUNCERR;
        break;
        }

    rowcnt++;                             /* Wieder einer geschafft ! */
    }

if (retc == FUNCERR)                      /* Ist ein Fehler aufgetreten ? */
    return FUNCERR;

$close inscur;                            /* Satzzeiger schliessen */

if (sqlca.sqlcode != 0L)                  /* Fehler ? */
    {
    prsqlerror("CURSOR konnte nicht geschlossen werden");
    return FUNCERR;
    }

return FUNCOK;
}

/*
 * allocvart()
 * Speicher fuer die Variablen-Tabelle (Struktur var_t)
 * anfordern und konstante Werte eintragen.
 */

var_t *
allocvart(ncols)
int ncols;

{
register int i;
register var_t *pvar;
register var_t *vartab;

/*
 * Speicherplatz anfordern
 */

vartab = (var_t *) malloc((unsigned int) (sizeof(var_t) * ncols));

if (vartab != (var_t *) 0)              /* Speicheranforderung o.k. ? */
    {
    pvar = vartab;
    for (i = ncols; i > 0; i--)
        {
        pvar->sqltype = CFIXCHARTYPE;   /*
                                         * Konstanten in Variablentabelle
                                         * eintragen
                                         */
        pvar->sqlind  = (short *) 0;
        pvar++;
        }
    }

return vartab;
}

/*
 * locktable()
 * Tabelle sperren.
 */

locktable()

{
```

```
$char lockstmt[80];                 /* Host-Variable */

sprintf(lockstmt, "lock table %s in %s mode", tabname, LOCKMODE);

$prepare lockid from $lockstmt;     /* LOCK-Anweisung voruebersetzen */

if (sqlca.sqlcode != 0L)            /* Fehler ? */
    return FUNCERR;

$execute lockid;                    /* LOCK-Anweisung ausfuehren  */

if (sqlca.sqlcode != 0L)            /* Fehler ? */
    return FUNCERR;

return FUNCOK;
}


/*
 * input():
 * Diese Funktion liest eine Satz (abgeschlossen durch \n) von
 * inpfile in den Arbeitsspeicher (buffer) ein. Sie traegt die
 * Anfangsadressen und die Laengen, der durch delimiter abge-
 * schlossenen Felder in die sqldata-Pointer bzw. sqllen-Variablen
 * der Variablen-Tabelle ein.
 */

static char nullchar = '\0';    /* Fuer NULL-Wert */

input(psqlda)
struct sqlda *psqlda;

{
register int ch;                /* Eingelesenes Zeichen */
register char *pbuf;            /* Zeiger in Satzpuffer */
register int nbytes;            /* Freie Zeichen in Satzpuffer */
register int varsize;           /* Anzahl Zeichen aktuelles Feld */
register var_t *pcols;          /* Aktuelles Feld in Variablentabelle */
register int cntvars;           /* Anzahl eingelesener Felder */
register int maxvars;           /* Maximale Anzahl Felder */
register char *vardata;         /* Beginn aktuelles Feld in Satzpuffer */
register int escape;            /* Schalter fuer escape-Zeichen */
int retc;

/*
 * Initialisierung der Variablen fuer Einlesen
 */

pcols   = psqlda->sqlvar;
maxvars = numcols;
pbuf    = buffer;
nbytes  = MAXBUF;
cntvars = 0;
varsize = 0;

retc = FUNCOK;

while ((ch = fgetc(inpfile)) != EOF)      /* Lesen Zeichen bis EOF */
    {
    if (ch == '\n')                       /* Satzende erreicht ? */
        {
        if (varsize > 0)                  /* Nicht nach DELIMITER ? */
            {
            prerror("Letztes Feld nicht mit DELIMITER abgeschlossen");
            retc = FUNCERR;
            }

        break;                            /* Satzende */
        }

    if (varsize == 0)                     /* Beginn eines Feldes */
        {
        if (!(cntvars < maxvars))         /* Zuviele Eingabe-Werte ? */
            {
```

```
            prerror("Zuviele Werte im Eingabesatz");
            retc = FUNCERR;
            break;
            }

        escape = FALSE;                     /* escape-Schalter aus */

        vardata = pbuf;                     /* Adresse Beginn Feld */
        }

    if ((ch == delimiter) && (!escape))     /* Feldende erreicht ? */
        {
        if (varsize == 0)                   /* Kein Zeichen gelesen ? */
            {
            pcols->sqldata = &nullchar;     /* NULL-Wert */
            pcols->sqllen  = 1;
            }
        else
            {
            pcols->sqldata = vardata;       /* Daten-Adresse */
            pcols->sqllen  = varsize;       /* Daten-Laenge */
            }

        cntvars++;              /* Anzahl Eingabefelder inkrementieren */

        pcols++;                /* Naechstes Feld in Variablentabelle */
        varsize = 0;            /* Kennzeichen Feldbeginn */
        continue;
        }

    if (((ch == '\\') && (!escape)) && (delimiter != '\\'))
        {
        escape = TRUE;
        continue;               /* backslash ueberspringen */
        }

    escape = FALSE;

    if (!(nbytes-- > 0))        /* Kein Platz mehr ? */
        {
        prerror("Eingabesatz groesser als Satzpuffer");
        retc = FUNCERR;
        break;
        }

    *pbuf++ = ch;               /* Abspeichern des eingelesenen Zeichens */
    varsize++;
    }

if (retc != FUNCOK)             /* Ist ein Fehler aufgetreten ? */
    return retc;

if (ch == EOF)                  /* Dateiende erreicht ? */
    {
    if (nbytes != MAXBUF)       /* Dateiende mitten im Satz ? */
        {
        prerror("Unvorhergesehenes Ende der Eingabe");
        return FUNCERR;
        }

    return EOFINP;              /* EOF waehrend Eingabe */
    }

if (inserttype == INSERTLD)     /* INSERT ohne VALUES ? */
    {
    if (cntvars < maxvars)      /* Zuwenig Eingabe-Werte ? */
        {
        prerror("Zuwenig Werte im Eingabesatz");
        return FUNCERR;
        }
    }

psqlda->sqld = cntvars;         /* Anzahl Felder */
```

```
return FUNCOK;
}


/*
 * bldnewinsert()
 * INSERT-Anweisung ohne VALUES-Angabe um VALUES (?,...) ergaenzen.
 */

bldnewinsert(oldins, newins)
char *oldins;
char *newins;

{
register int i;
register char *cp;

cp = cpystr(oldins, newins);     /* Adresse abschliessendes '\0' */

/*
 * Ergaenzen um VALUES-Angabe
 */

cp = cpystr(" values(", cp);

for (i = numcols - 1; i > 0; i--)
    cp = cpystr("?,", cp);

cp = cpystr("?)", cp);
}


/*
 * getnumcols()
 * Die Anzahl der Felder der Tabelle aus dem Feld ncols
 * in der Systemtabelle systables ermitteln.
 * Da es auch erlaubt sein sollte an Stelle eines Tabellen-
 * namens einen Synonymnamen dafuer anzugeben, ergibt sich
 * eine etwas kompliziertere SELECT-Anweisung.
 */

getnumcols()

{
$char *tbname;
$int cols;

tbname = tabname;

/*
 * Systemtabellen abfragen ...
 */

$select ncols into $cols from systables
    where tabname = $tbname                          /* ... nach Tabellenname */
       or tabid = (select tabid from syssynonyms /* ... nach Synonymname */
                        where synname = $tbname);

if (sqlca.sqlcode == SQLNOTFOUND)                    /* Nanu ?!? */
    {
    prerror("Die Tabelle ist in der Datenbank nicht vorhanden");
    return FUNCERR;
    }

if (sqlca.sqlcode != 0L)                             /* Fehler ? */
    {
    prsqlerror("Fehler bei Zugriff auf Systemtabellen");
    return FUNCERR;
    }

numcols = cols;

return FUNCOK;
}
```

```
/*
 * parseinsert():
 * Untersucht eine Anweisung:
 * INSERT INTO tab [ ( feld, ... ) ] [ VALUES ...  | SELECT ... ]
 * Dabei wird versucht den Namen der Tabelle tab und, wenn ange-
 * geben, die Anzahl der Felder zu ermitteln.
 */

parseinsert(inspar)
char *inspar;

{
register char *cp;
register int cnt;
register char *ptabbeg;
register char *ptabend;

cp = inspar;                          /* Initialisierung */

if (!(cp = chkword(cp, "insert")))    /* Kein INSERT ? */
    return NOINSERT;
cp = skipword(cp);                    /* Ueberspringe INSERT  */
if (!(cp = chkword(cp, "into")))      /* Kein INTO ? */
    return NOINSERT;
cp = skipword(cp);                    /* Ueberspringe INTO    */

cp = skipspace(cp);                   /* Suche Tabellenname   */

ptabbeg = cp;                         /* Markiere Beginn Tabellenname */

cp = skipword(cp);                    /* Ueberspringe Tabellenname */

ptabend = cp;                         /* Markiere Ende Tabellenname */

cnt = (int) (ptabend - ptabbeg);      /* Laenge Tabellenname */
if (cnt > NAMELEN)                    /* Zulange ? */
    return NOINSERT;

bycopy(ptabbeg, tabname, cnt);        /* Abspeichern Tabellenname */
tabname[cnt] = '\0';

cp = skipspace(cp);                  /* Suche '(' | VALUES/SELECT | '\0' */

cnt = 0;                             /* Zaehler fuer Feldnamen */

if (*cp == '(')                      /* Wenn '(', dann Feldnamen vorhanden */
    {
    cp++;
    while (*cp && (*cp != ')'))      /* Wenn ')', dann Ende der Feldnamen */
        {
        cp = skipspace(cp);          /* Suche Feldnamen */
        cp = skipword(cp);           /* Ueberspringe Feldname */
        cnt++;
        cp = skipspace(cp);          /* Suche ',' oder ')' */
        if (*cp == ',')              /* Wenn ',', dann ueberspringen */
            cp++;
        }

    if (*cp)
        cp++;                        /* Ueberspringe ')' */
    }

numcols = cnt;                       /* Anzahl Felder abspeichern */

cp = skipspace(cp);                  /* Suche VALUES | SELECT | '\0' */

if (*cp == '\0')                     /* '\0' */
    return INSERTLD;

if (chkword(cp, "values"))           /* VALUES */
    return INSERTVAL;

if (chkword(cp, "insert"))           /* INSERT */
    return INSERTSEL;
```

```
return NOINSERT;                    /* Fehler */
}


/*
 * Pruefen ob die Zeichenfolge str mit dem Wort word beginnt.
 * Leerzeichen oder Kommentare am Beginn werden uebersprungen.
 * Das Wort in str wird in Kleinbuchstaben umgewandelt.
 * Rueckkehrwert ist die Position von word innerhalb str
 * oder NULL.
 */

char *
chkword(str, word)
register char *str;
register char *word;

{
register char *begin;

str = skipspace(str);          /* Ueberspringe Leerzeichen */

begin = str;                   /* Markiere Beginn */

while (*word && *str)          /* Zeichenfolge und Wort ungleich '\0' */
    {
                                        /* Umwandlung */
    if ((*(word++)) != (isupper(*str) ? tolower(*str) : *str))
        break;
    str++;
    }

return (*word) ? (char *) 0 : begin;       /* Position von Wort */
}


/*
 * Ueberspringe ein SQL-Schluesselwort oder einen
 * SQL-Bezeichner.
 * Es wird mindestens ein Zeichen uebersprungen.
 */

char *
skipword(s)
char *s;

{
register char *cp;

cp = s;
while (*cp)
    {
    cp++;
    if ((!isalnum(*cp)) && (*cp != '_'))   /* SQL-Sprachelement */
        break;
    }
return cp;
}


/*
 * Ueberspringe Leerzeichen (auch Tab's) und Kommentare
 * Rueckkehrwert ist die Adresse des ersten Nicht-Leer-
 * zeichens oder des Endes der Zeichenkette s.
 */

char *
skipspace(s)
char *s;

{
register char *cp;

cp = s;                                          /* Initialisierung */
```

```
while ((*cp) && ((isspace(*cp)) || (*cp == '{')))  /* ' ' | Kommentar ? */
    {
    if (*cp == '{')             /* Ueberspringe Kommentar */
        {
        do
            cp++;
        while (*cp && (*cp != '}'));

        if (!(*cp))             /* Nicht geschlossener Kommentar */
            break;
        }

    cp++;
    }

return cp;
}


/*
 * Kopieren Zeichenfolge from nach to.
 * Rueckkehrwert ist die Adresse des abschliessenden
 * '\0' der Zeichenfolge to.
 */

char *
cpystr(from, to)
register char *from;
register char *to;

{
while (*from)
    *to++ = *from++;   /* Zeichenweises kopieren */

*to = '\0';

return to;
}


/*
 * prsqlerror()
 * Ausgeben SQL-Fehlermeldung
 */

prsqlerror(errmsg)
char *errmsg;

{
if (errmsg != (char *) 0)
    prerror(errmsg);

fprintf(stderr, "SQLCODE: %ld\n", sqlca.sqlcode);
}


/*
 * prerror()
 * Ausgeben Fehlermeldung mit Nummer des Satzes bei dem
 * der Fehler aufgetreten ist.
 * Satznummer plus 1, da in runinsert() der Satzzaehler
 * erst nach korrektem Einfuegen erhoeht wird.
 */

prerror(errmsg)
char *errmsg;

{
fprintf(stderr, "Satz %ld: %s.\n", rowcnt + 1, errmsg);
 }
```

Beispiel3 dpoint.c

```
/*
 * Umwandeln Dezimalpunkt in Dezimalkomma bzw.
 * umgekehrt.
 */


/*
 * Funktion, die nicht ( int ) zurueckliefert
 */

char *getenv();

/*
 * Globale Definitionen
 */

static short initflag = 0;      /* Initialisierungs-Schalter */
static char  decpoint = ',';    /* DEFAULT-Dezimalpunkt/-komma */


/*
 * dcommapoint()
 * Dezimalkomma in Dezimalpunkt vertauschen.
 */

dcommapoint(str)
register char *str;

{
if (!initflag)
    dpinit();

if (decpoint == ',')
    {
    while (*str)
        {
        if (*str == ',')
            {
            *str = '.';
            break;
            }
        str++;
        }
    }
}

/*
 * dpointcomma()
 * Dezimalpunkt in Dezimalkomma vertauschen.
 */

dpointcomma(str)
register char *str;

{
if (!initflag)
    dpinit();

if (decpoint == ',')
    {
    while (*str)
        {
        if (*str == '.')
            {
            *str = ',';
            break;
            }
        str++;
        }
    }
}
```

```
/*
 * dpinit()
 * Initialisieren decimal-point
 */

static
dpinit()

{
register char *cp;
register short i;

cp = getenv("DBMONEY");

if (cp != (char *) 0)
    {
    while (*cp)
        {
        if (*cp == '.' || *cp == ',')
            {
            decpoint = *cp;
            break;
            }
        cp++;
        }
    }

initflag = 1;
 }
```

A.2 Include-Dateien

Die sqlca-Struktur

```
struct sqlca_s
   {
   long sqlcode;
   char sqlerrm[72]; /* error message parameters */
   char sqlerrp[8];
   long sqlerrd[6];
                /* 0 - reserved        */
                /* 1 - serial value after insert or  ISAM error code */
                /* 2 - number of rows processed */
                /* 3 - not used at present */
                /* 4 - offset into of an error */
                /* 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;
  };

extern struct sqlca_s sqlca;

#define SQLNOTFOUND 100
```

sqlda.h-Struktur

```
 #ifndef _SQLDA
#define _SQLDA

struct sqlvar_struct
    {
    short sqltype;              /* variable type                */
    short sqllen;               /* length in bytes              */
    char *sqldata;              /* pointer to data              */
    short *sqlind;              /* pointer to indicator         */
    char  *sqlname;             /* variable name                */
    char  *sqlformat;           /* reserved for future use      */
    short sqlitype;             /* ind variable type            */
    short sqlilen;              /* ind length in bytes          */
    char *sqlidata;             /* ind data pointer             */
    };

struct sqlda
    {
    short sqld;
    struct sqlvar_struct *sqlvar;
    };

 #endif /* _SQLDA */
```

sqlstype.h

```
#define SQ_DATABASE     1
#define SQ_SELECT       2
#define SQ_SELINTO      3
#define SQ_UPDATE       4
#define SQ_DELETE       5
#define SQ_INSERT       6
#define SQ_UPDCURR      7
#define SQ_DELCURR      8
#define SQ_LDINSERT     9
#define SQ_LOCK         10
#define SQ_UNLOCK       11
#define SQ_CREADB       12
#define SQ_DROPDB       13
#define SQ_CRETAB       14
#define SQ_DRPTAB       15
#define SQ_CREIDX       16
#define SQ_DRPIDX       17
#define SQ_GRANT        18
#define SQ_REVOKE       19
#define SQ_RENTAB       20
#define SQ_RENCOL       21
#define SQ_CREAUD       22
#define SQ_STRAUD       23
#define SQ_STPAUD       24
#define SQ_DRPAUD       25
#define SQ_RECTAB       26
#define SQ_CHKTAB       27
#define SQ_REPTAB       28
#define SQ_ALTER        29
#define SQ_STATS        30
#define SQ_CLSDB        31
#define SQ_DELALL       32
#define SQ_UPDALL       33
#define SQ_BEGWORK      34
#define SQ_COMMIT       35
#define SQ_ROLLBACK     36
#define SQ_SAVEPOINT    37
#define SQ_STARTDB      38
#define SQ_RFORWARD     39
#define SQ_CREVIEW      40
#define SQ_DROPVIEW     41
#define SQ_DEBUG        42
#define SQ_CREASYN      43
#define SQ_DROPSYN      44
#define SQ_CTEMP        45
#define SQ_WAITFOR      46
#define SQ_ALTIDX       47
#define SQ_ISOLATE      48
#define SQ_SETLOG       49
#define SQ_EXPLAIN      50
#define SQ_SCHEMA       51
```

sqltypes.h

```
#ifndef CCHARTYPE
 /* C language types */

#define CCHARTYPE        100
#define CSHORTTYPE       101
#define CINTTYPE         102
#define CLONGTYPE        103
#define CFLOATTYPE       104
#define CDOUBLETYPE      105
#define CDECIMALTYPE     107
#define CFIXCHARTYPE     108
#define CSTRINGTYPE      109
#define CDATETYPE        110
#define CMONEYTYPE       111
#define CDTIMETYPE       112
#define CLOCATORTYPE     113
#define CVCHARTYPE       114
#define CINVTYPE         115
#define CFILETYPE        116

#define USERCOLL(x)      ((x))

/*
 * Define all possible database types
 *   include C-ISAM types here as well as in isam.h
 */

#define SQLCHAR          0
#define SQLSMINT         1
#define SQLINT           2
#define SQLFLOAT         3
#define SQLSMFLOAT       4
#define SQLDECIMAL       5
#define SQLSERIAL        6
#define SQLDATE          7
#define SQLMONEY         8
#define SQLNULL          9
#define SQLDTIME         10
#define SQLBYTES         11
#define SQLTEXT          12
#define SQLVCHAR         13
#define SQLINTERVAL      14
#define SQLTYPE          0xF     /* type mask              */
#define SQLNONULL        0x100   /* disallow nulls         */
#define SQLMAXTYPES      15

/* this is not a real type but a flag to show that the
 * value is from a host variable
 */
#define SQLHOST          01000
#define SQLNETFLT        02000

#define SIZCHAR          1
#define SIZSMINT         2
#define SIZINT           4
#define SIZFLOAT         (sizeof(double))
#define SIZSMFLOAT       (sizeof(float))
#define SIZDECIMAL       17      /* decimal(32) */
#define SIZSERIAL        4
#define SIZDATE          4
#define SIZMONEY         17      /* decimal(32) */
#define SIZDTIME         7       /* decimal(12,0) */
#define SIZVCHAR         1

#define ISDECTYPE(t) (ISSQLTYPE(t) && (((t)&SQLTYPE)==SQLDECIMAL ||
         ((t)&SQLTYPE)==SQLMONEY ||
         ((t)&SQLTYPE) == SQLDTIME || ((t)&SQLTYPE) == SQLINTERVAL))
#define ISBLOBTYPE(type)   (ISBYTESTYPE (type) || ISTEXTTYPE(type))
#define ISBYTESTYPE(type)  (ISSQLTYPE(type) && ((type) & SQLTYPE) ==
          SQLBYTES)
#define ISTEXTTYPE(type)  (ISSQLTYPE(type) && ((type) & SQLTYPE) ==
```

```
        SQLTEXT)
#define ISBLOBCTYPE(type) (ISLOCTYPE(type) || ISFILETYPE(type))
#define ISLOCTYPE(type) ((type) == CLOCATORTYPE)
#define ISFILETYPE(type) ((type) == CFILETYPE)
#define ISVCTYPE(t)     (ISSQLTYPE(t) && ((t) & SQLTYPE) == SQLVCHAR)

#define ISSQLTYPE(t)    ((t) >= SQLCHAR && ((t)&SQLTYPE) < SQLMAXTYPES)

#define DEFDECIMAL      9       /* default decimal(16) size */
#define DEFMONEY        9       /* default decimal(16) size */
#define DATENULL        0x80000000L
#define DATEDOOM        0x80000001L

#define SYSPUBLIC       "public"

#endif /* CCHARTYPE */
```

decimal.h

```
/*
 * Unpacked Format (format for program usage)
 *
 *    Signed exponent "dec_exp" ranging from  -64 to +63
 *    Separate sign of mantissa "dec_pos"
 *    Base 100 digits (range 0 - 99) with decimal point
 *      immediately to the left of first digit.
 */

#ifndef DECSIZE
#define DECSIZE 16
#define DECUNKNOWN -2

struct decimal
    {
    short dec_exp;              /* exponent base 100           */
    short dec_pos;              /* sign: 1=pos, 0=neg, -1=null */
    short dec_ndgts;            /* number of significant digits */
    char  dec_dgts[DECSIZE];    /* actual digits base 100      */
    };
typedef struct decimal dec_t;

/*
 *  A decimal null will be represented internally by setting dec_pos
 *  equal to DECPOSNULL
 */

#define DECPOSNULL      (-1)

/*
 * DECLEN calculates minumum number of bytes
 * necessary to hold a decimal(m,n)
 * where m = total # significant digits and
 *       n = significant digits to right of decimal
 */

#define DECLEN(m,n)     (((m)+((n)&1)+3)/2)
#define DECLENGTH(len)  DECLEN(PRECTOT(len),PRECDEC(len))

/*
 * DECPREC calculates a default precision given
 * number of bytes used to store number
 */

#define DECPREC(size)   (((size-1)<<9)+2)

/* macros to look at and make encoded decimal precision
 *
 *  PRECTOT(x)          return total precision (digits total)
 *  PRECDEC(x)          return decimal precision (digits to right)
 *  PRECMAKE(x,y)       make precision from total and decimal
 */

#define PRECTOT(x)      (((x)>>8) & 0xff)
#define PRECDEC(x)      ((x) & 0xff)
#define PRECMAKE(x,y)   (((x)<<8) + (y))
```

```
 /*
 * Packed Format  (format in records in files)
 *
 *    First byte =
 *        top 1 bit = sign 0=neg, 1=pos
 *        low 7 bits = Exponent in excess 64 format
 *    Rest of bytes = base 100 digits in 100 complement format
 *    Notes --  This format sorts numerically with just a
 *              simple byte by byte unsigned comparison.
 *              Zero is represented as 80,00,00,... (hex).
 *              Negative numbers have the exponent complemented
 *              and the base 100 digits in 100's complement
 */

#endif /* DECSIZE */
```

datetime.h

```
#ifndef DECSIZE
#include "../incl/decimal.h"
#endif

typedef struct dtime
        {
        short dt_qual;
        dec_t dt_dec;
        }       dtime_t;

typedef struct intrvl
        {
        short in_qual;
        dec_t in_dec;
        }       intrvl_t;

/* time units for datetime qualifier */

#define TU_YEAR 0
#define TU_MONTH 2
#define TU_DAY 4
#define TU_HOUR 6
#define TU_MINUTE 8
#define TU_SECOND 10
#define TU_FRAC 12
#define TU_F1 11
#define TU_F2 12
#define TU_F3 13
#define TU_F4 14
#define TU_F5 15

/* TU_END - end time unit in datetime qualifier */
/* TU_START - start time unit in datetime qualifier */
/* TU_LEN - length in  digits of datetime qualifier */

#define TU_END(qual) (qual & 0xf)
#define TU_START(qual) ((qual>>4) & 0xf)
#define TU_LEN(qual) ((qual>>8) & 0xff)

/* TU_ENCODE - encode length, start and end time unit to form qualifier */
/* TU_DTENCODE - encode datetime qual */
/* TU_IENCODE - encode interval qual */

#define TU_ENCODE(len,s,e) (((len)<<8) | ((s)<<4) | (e))
#define TU_DTENCODE(s,e) TU_ENCODE(((e)-(s)+((s)==TU_YEAR?4:2)), s, e)
#define TU_IENCODE(len,s,e) TU_ENCODE(((e)-(s)+(len)),s,e)
#define TU_FLEN(len) (TU_LEN(len)-(TU_END(len)-TU_START(len)))

/* TU_CURRQUAL - default qualifier used by current */

#define TU_CURRQUAL TU_ENCODE(17,TU_YEAR,TU_F3)
```

varchar.h

```
/*
 * VARCHAR macros
 */

#define MAXVCLEN                (255)
#define VCLENGTH(len)           (VCMAX(len)+1)
#define VCMIN(size)             (((size) >> 8) & 0x00ff)
#define VCMAX(size)             ((size) & 0x00ff)
#define VCSIZ(max, min)         ((((min) << 8) & 0xff00) + ((max) & 0x00ff))
```

locator.h

```
#ifndef LOCATOR_INCL                /* avoid multiple includes */
#define LOCATOR_INCL

/*
```

Locators are used to store TEXT or BYTE fields (blobs) in ESQL programs. The "loc_t" structure is described below. Fields denoted USER should be set by the user program and will be examined by the DBMS system. Those denoted SYSTEM are set by the system and may be examined by the user program. Those denoted INTERNAL contain data only the system manupilates and examines.

If "loc_loctype" is set to LOCMEMORY, then the blob is stored in primary memory. The memory buffer is pointed to by the variant "loc_buffer". The field "loc_bufsize" gives the size of "loc_buffer". If the "loc_bufsize" is set to "-1" and the locator is used for a fetch, memory is obtained using "malloc" and "loc_buffer" and "loc_bufsize" are set.

If "loc_loctype" is set to LOCFILE, then the blob is stored in a file. The file descriptor of an open operating system file is specified in "loc_fd".

If "loc_loctype" is set to LOCFNAME, the the blob is stored in a file and the name of the file is given. The DBMS will open or created the file at the correct time and in the correct mode.

If the "loc_loctype" is set to LOCUSER, "loc_(open/close/read/write)" are called. If the blob is an input to a SQL statement, "loc_open" is called with the parameter "LOC_RONLY". If the blob is an output target for an SQL statement, "loc_open" is called with the parameter "LOC_WONLY".

"loc_size" specifies the maximum number of bytes to use when the locator is an input target to an SQL statement. It specifies the number of bytes returned if the locator is an output target. If "loc_loctype" is LOCFILE or LOCUSER, it can be set to -1 to indicate transfer until end-of-file.

"loc_indicator" is set by the user to -1 to indicate a NULL blob. It will be set to -1 if a NULL blob is retrieved. If the blob to be retrieved will not fit in the space provided, the indicator contains the size of the blob.

"loc_status" is the status return of locator operations.

"loc_type" is the "blob" type (SQLTEXT, SQLBYTES, ...).

"loc_user_env" is a pointer for the user's private use. It is neither set nor examined by the system. "loc_user_env" as well as the "loc_union" fieds may be used by user supplied routines to store and communicate information. */

```
typedef struct
    {
    short loc_loctype;          /* USER: type of locator - see below    */
    union                       /* variant on 'loc'                     */
        {
        struct                  /* case LOCMEMORY                       */
            {
            long  lc_bufsize;   /* USER: buffer size */
            char *lc_buffer;    /* USER: memory buffer to use           */
            char *lc_currdata_p;/* INTERNAL: current memory buffer      */
            int   lc_mflags;    /* INTERNAL: memory flags (see below)   */
            } lc_mem;

        struct                  /* cases LOCFNAME & LOCFILE             */
            {
            char *lc_fname;     /* USER: file name                      */
            int   lc_mode;      /* USER: perm. bits used if creating    */
            int   lc_fd;        /* USER: os file descriptior            */
            long  lc_position;  /* INTERNAL: seek position              */
            } lc_file;
        } lc_union;

    long  loc_indicator;        /* USER SYSTEM: indicator               */
    long  loc_type;             /* SYSTEM: type of blob                 */
    long  loc_size;             /* USER SYSTEM: num bytes in blob or -1 */
    int   loc_status;           /* SYSTEM: status return of locator ops */
    char *loc_user_env;         /* USER: for the user's PRIVATE use     */
    long  loc_xfercount;        /* INTERNAL/SYSTEM: Transfer count      */

    int (*loc_open)();          /* USER: open function                  */
    int (*loc_close)();         /* USER: close function                 */
    int (*loc_read)();          /* USER: read function                  */
    int (*loc_write)();         /* USER: write function                 */

    int   loc_oflags;           /* USER/INTERNAL: see flag definitions below */
    } loc_t;

#define loc_fname       lc_union.lc_file.lc_fname
#define loc_fd          lc_union.lc_file.lc_fd
#define loc_position    lc_union.lc_file.lc_position
#define loc_bufsize     lc_union.lc_mem.lc_bufsize
#define loc_buffer      lc_union.lc_mem.lc_buffer
#define loc_currdata_p  lc_union.lc_mem.lc_currdata_p
#define loc_mflags      lc_union.lc_mem.lc_mflags

/* Enumeration literals for loc_loctype */

#define LOCMEMORY       1               /* memory storage */
#define LOCFNAME        2               /* File storage with file name */
#define LOCFILE         3               /* File storage with fd */
#define LOCUSER         4               /* User define functions */

/* passed to loc_open and stored in loc_oflags */
#define LOC_RONLY       0x1             /* read only */
#define LOC_WONLY       0x2             /* write only */
/* LOC_APPEND can be set when the locator is created
 * if the file is to be appended to instead of created
 */
#define LOC_APPEND      0x4             /* write with append */
#define LOC_TEMPFILE    0x8             /* 4GL tempfile blob */

/* passed to loc_open and stored in loc_mflags */
#define LOC_ALLOC       0x1             /* free and alloc memory */

#endif  /* LOCATOR_INCL */
```

blob.h

```
#ifndef BLOB_DOT_H      /* To handle multiple includes */
#define BLOB_DOT_H

/*
 * Structure of the BlobLocation
 */

/*
 * blob definitions
 * tblob_t is the data that is stored in the tuple - it describes the blob
 * NOTE: tb_fd is expected to be the first member and TB_FLAGS gives offset
 *       to the tb_flags member.
 */

typedef struct tblob
    {
    short       tb_fd;          /* blob file descriptor (must be first) */
    short       tb_coloff;      /* Blob column offset in row            */
    long        tb_tblspace;    /* blob table space                     */
    long        tb_start;       /* starting byte                        */
    long        tb_end;         /* ending byte-0 for end of blob        */
    long        tb_size;        /* Size of blob                         */
    long        tb_addr;        /* Starting Sector or BlobPage          */
    long        tb_family;      /* Family ID                            */
    long        tb_volume;      /* Family Volume                        */
    short       tb_medium;      /* Medium - odd if removable            */
    short       tb_bstamp;      /* first BlobPage Blob stamp            */
    short       tb_sockid;      /* socket id of remote blob             */
    short       tb_flags;       /* flags - see below                    */
    long        tb_reserved1;   /* reserved for the future              */
    long        tb_reserved2;   /* reserved for the future              */
    long        tb_reserved3;   /* reserved for the future              */
    long        tb_reserved4;   /* reserved for the future              */
    } tblob_t;

                                /* offest to tb_flags in tblob_t */
#define TB_FLAGS                (2*INTSIZE+7*LONGSIZE+3*INTSIZE)
#define SIZTBLOB                (11*LONGSIZE + 6*INTSIZE)

/* 'flags' definitions */
#define BLOBISNULL      (0x0001)        /* BLOB is NULL */
#define BLOBALIEN       (0x0002)        /* BLOB is ALIEN */
#define BL_BSBLOB       (0x0004)        /* blob is stored in blobspace */
#define BL_PNBLOB       (0x0008)        /* store in tablespace */

/*
 * this struture is used to pass "useful" information back to
 * the user.
 */

typedef struct blobinfo
    {
    long        bi_size;                /* Size of blob                 */
    long        bi_addr;                /* Starting Sector or BlobPage  */
    long        bi_family;              /* Family ID                    */
    long        bi_volume;              /* Family Volume                */
    short       bi_flags;               /* flags                        */
    short       bi_medium;              /* Medium - odd if removable    */
    } blobinfo_t;

#endif  /* BLOB_DOT_H : To handle multiple includes */
```

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;
typedef struct value * perfvalue;

extern struct value retstack;

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

#define lngreturn(i)     {retstack.v_type=SQLINT;\
                          retstack.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;\
                          retstack.v_double=(d);\
                          return(&retstack);}
#endif /* NOFLOAT */

#define strreturn(s,c)   {retstack.v_type=SQLCHAR;\
                          retstack.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;\
                          deccopy(&d, &retstack.v_decimal);\
                          return(&retstack);}

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

#define invreturn(inv)   {retstack.v_type=SQLINTERVAL;\
                          retstack.v_prec=(inv).in_qual;\
                          deccopy(&(inv).in_dec, &retstack.v_decimal);\
                          return(&retstack);}

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

extern long tolong();    /* tolong() takes a pointer to value structure
                          *          as an argument.
                          *          Returns the value (converted to long)
                          *          of the value structure (v_long).
                          */
#ifndef 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 structure
                        *          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, dtime, dtqual)
                        *      valueptr val;
                        *      dtime_t *dtime;
                        *      int dtqual;
                        *
                        * input args:  val, dtqual
                        * output args: dtime
                        */

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

#ifndef 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 statement of ACE.
 *        0,0 <-------------------Note that this array must be terminated
 *                                      by two zeros.
 *        };
 *
 * These structures are required so that ACE can call the user subroutines
 * at run time.
 */
```

A.3 Dienstprogramme

Folgende Dienstprogramme, die auf Betriebssystemebene aufzurufen sind, stehen dem INFORMIX-Anwender bzw. INFORMIX-Datenbankverwalter zur Verfügung:

Dienstprogramm	Bedeutung	gültig für Backend SE=INFORMIX-SE ON=INFORMIX-ONLINE
bcheck	Indexprüfprogramm	SE
dbexport	Datenbank exportieren	SE, ON
dbimport	Datenbank importieren	SE, ON
dbload	Daten aus Datei in Datenbank einlesen	SE,ON
dblog	Transaktionsprotokoll-Dateien ausgeben	SE
dbschema	Anweisungen für Datenbankaufbau erzeugen	SE, ON
tbcheck	Indexprüfprogramm	ON 1)
tbload	Datenbank oder Tabelle binär laden	ON 1)
tblog	logische Protokolle ausgeben	ON 1)
tbunload	Datenbank oder Tabelle binär entladen	ON 1)

1) Die Beschreibung dieses Dienstprogramms finden Sie im ONLINE-Handbuch [6].

A.3.1 bcheck - Indexprüfprogramm

bcheck ist ein Dienstprogramm, das Indexdateien prüft oder auch repariert. Es kann nur für INFORMIX-SE-Datenbanken verwendet werden. INFORMIX-ONLINE-Anwender benutzen anstelle von *bcheck* das Dienstprogramm *tbcheck* (siehe INFORMIX-ONLINE-Handbuch [6]). *bcheck* vergleicht die Indexdatei mit der Datendatei. Geprüft wird die Übereinstimmung von Spalten, die als Index vereinbart wurden, mit den in der Indexdatei geschriebenen Schlüsselwerten. Wenn es keine Übereinstimmung gibt, fragt Sie das Prüfprogramm, ob Sie den defekten Index löschen und neu aufbauen wollen. Wenn Sie erneuern lassen wollen, liest *bcheck* die ganze Datendatei sequentiell durch und schreibt die gefundenen Indexwerte wieder in die Indexdatei. Voraussetzung ist natürlich, daß die Indexbeschreibung in der Indexdatei noch in Ordnung ist.

```
bcheck[␣-ilnjqs]␣datei[...]
```

-i Nur die Indexdatei wird geprüft.
-l Alle Einträge in den b-trees werden aufgelistet.
-n Alle Fragen, die gestellt werden, werden mit nein beantwortet.
-j Alle Fragen, die gestellt werden, werden mit ja beantwortet.
-q Ohne Meldungen.
-s Verändert die Knotengröße in der Indexdatei.

datei Name der zu prüfenden C-ISAM-Datei.

Wenn Sie keinen Parameter angeben, wartet *bcheck* auf eine Antwort von Ihnen, wenn ein Fehler gefunden wurde. Verwenden Sie den Parameter **-j** immer mit Vorsicht und erst ab der 2. Überprüfung der Datei.

Beispiel für eine normal verlaufende bcheck-Sitzung

```
$ bcheck kunde__100

BCHECK  C-ISAM B-tree Pruefprogramm   Version 4.00.U
Copyright (C) 1981-1989 Informix Software, Inc.


C-ISAM Datei: kunde__100

Pruefen der Dateibeschreibung und Datei-Groessen.
Aktuelle Knotengroesse der C_ISAM Index-Datei = 1024
Pruefen der Datendatei Saetze
Pruefen der Indizes- und Schluesselbeschreibungen.
Index 1 = Schluessel Eindeutig:
    0 benutzte Index-Knoten — 1 benutzte Index b-tree Ebenen
Index 2 = Schluessel Eindeutig:  (0,4,2)
    1 benutzte Index-Knoten — 1 benutzte Index b-tree Ebenen
Index 3 = Schluessel Duplikate:  (111,5,0)
    1 benutzte Index-Knoten — 1 benutzte Index b-tree Ebenen
Pruefen der Freiplatzlisten fuer Datensaetze und Index-Knoten
4 benutzte Index-Knoten, 0 frei — 18 benutzte Datensaetze, 0 frei
```

bcheck fand keine Fehler. Die Zahlen, die für jeden Index ausgegeben wurden, bedeuten die Stellung des Schlüsselwertes im Datensatz. Es können bis zu 8 Zahlengruppen für jeden Index angegeben werden.

Beispiele für fehlerhafte Indexdateien

Im folgenden Fall soll *bcheck* keine Änderungen vornehmen. Beantworten Sie alle Fragen mit "nein", indem Sie das Programm mit -n aufrufen:

```
$ bcheck -n kunde__100

BCHECK  C-ISAM B-tree Pruefprogramm   Version 4.00.U
Copyright (C) 1981-1989 Informix Software, Inc.


C-ISAM Datei: kunde__100

Pruefen der Dateibeschreibung 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 benutzte Index-Knoten — 1 benutzte Index b-tree Ebenen
Index 2 = Schluessel Eindeutig:  (0,4,2)
    1 benutzte Index-Knoten — 1 benutzte Index b-tree Ebenen

FEHLER: 1 fehlende Zeiger auf Datensaetze
Index entfernen ? nein

Index 3 = Schluessel Duplikate:  (111,5,0)
    1 benutzte Index-Knoten — 1 benutzte Index b-tree Ebenen

FEHLER: 1 fehlende Zeiger auf Datensaetze
Index entfernen ? nein

Pruefen der Freiplatzlisten fuer Datensaetze und Index-Knoten

FEHLER: 1 mehrfache Zeiger auf Datensaetze
Freiplatzliste fuer Datensaetze wiederherstellen ?nein

4 benutzte Index-Knoten, 0 frei — 19 benutzte Datensaetze, 0 frei
```

Im folgenden Fall wollen Sie die Indizes löschen und neu erstellen. Beantworten Sie diesmal alle Fragen mit "ja", indem Sie das Programm mit -j aufrufen:

```
$ bcheck -j kunde__100

BCHECK  C-ISAM B-tree Pruefprogramm   Version 4.00.U
Copyright (C) 1981-1989 Informix Software, Inc.
```

```
C-ISAM Datei: kunde__100

Pruefen der Dateibeschreibung 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 benutzte Index-Knoten — 1 benutzte Index b-tree Ebenen
Index 2 = Schluessel Eindeutig:  (0,4,2)
    1 benutzte Index-Knoten — 1 benutzte Index b-tree Ebenen

FEHLER: 3 fehlende Zeiger auf Datensaetze
Index entfernen ? ja

Index wiederherstellen ? ja

Index 3 = Schluessel Duplikate:  (111,5,0)
    1 benutzte Index-Knoten — 1 benutzte Index b-tree Ebenen

FEHLER: 3 fehlende Zeiger auf Datensaetze
Index entfernen ? ja

Index wiederherstellen ? ja

Pruefen der Freiplatzlisten fuer Datensaetze und Index-Knoten

FEHLER: 3 mehrfache Zeiger auf Datensaetze
Freiplatzliste fuer Datensaetze wiederherstellen ?ja

Freiplatzliste fuer Datensaetze wird wiederhergestellt.
Index 3 wird wiederhergestellt.
Index 2 wird wiederhergestellt.

FEHLER: Mehrfach vorhandener Schluesselwert, Datensatz 2

FEHLER: Mehrfach vorhandener Schluesselwert, Datensatz 12

FEHLER: Mehrfach vorhandener Schluesselwert, Datensatz 13
4 benutzte Index-Knoten, 0 frei — 21 benutzte Datensaetze, 0 frei
```

A.3.2 dbexport - Datenbank exportieren

Mit diesem Dienstprogramm erzeugen Sie aus einer Datenbank ASCII-Dateien, die Sie mit dem Dienstprogramm *dbimport* in eine andere Datenbankumgebung importieren können.

dbexport kann nur vom Datenbankverwalter (Zugriffsrecht DBA) oder dem Benutzer *informix* aufgerufen werden.

Ausgabe auf Platte: dbexport erzeugt im angegebenen Dateiverzeichnis (Schalter **-o**) das Dateiverzeichnis *datenbank.exp*, das folgende Dateien enthält:

tabelle1.*unl* ... ... tabellen.*unl*	Je Tabelle eine Datei, die die Daten enthält.
datenbank.*sql*	Datei, die die SQL-Anweisungen zum Datenbankaufbau enthält.
dbexport.out	Datei, die das Ablaufprotokoll enthält.

datenbank.exp wird der Gruppe *informix* zugeordnet.

Ausgabe auf Band: dbexport erzeugt kein Dateiverzeichnis. Die Dateien werden direkt auf Band ausgegeben.

Stellen Sie vor dem *dbexport*-Aufruf sicher, daß die Umgebungsvariable SQLEXEC auf die gewünschte Umgebung (ONLINE oder SE gesetzt ist).

```
dbexport[␣-c][␣-q][␣{-o␣dateiverzeichnis                                    }]␣datenbank
                    {-t␣gerät␣-b␣blockgr␣-s␣bandgr[␣-f␣datei]}
```

-c
gibt an, daß das Programm fortgesetzt werden soll, wenn Fehler auftauchen, ausgenommen folgende schwerwiegende Fehler:

- Das angegebene Bandgerät kann nicht eröffnet werden.
- Fehlerhafte Schreibzugriffe auf Band oder Platte.
- Ungültige Parameter im Kommando.
- Datenbank kann nicht eröffnet werden oder keine Zugriffsberechtigung.

-q
unterdrückt evtl. vorhandene Meldungen von SQL-Anweisungen am Bildschirm.

-o dateiverzeichnis
Exportiert die Datenbank auf Platte.
Geben Sie den absoluten oder relativen Pfadnamen eines Dateiverzeichnisses an, in dem die von *dbexport* erzeugten Dateien abgelegt werden sollen. Fehlt die Angabe, so gilt das aktuelle Dateiverzeichnis.

-t gerät
Exportiert die Datenbank auf Band.
Geben Sie die Gerätedatei des Bandgerätes an.
Wenn die Datenbank auf ein Band exportiert wird, dürfen Sie *dbexport* nicht als Hintergrundprozeß aufrufen.

-b blockgr
Blockgröße des Bandes in Kbyte.

Für INFORMIX-ONLINE gilt: Die Blockgröße muß ein Vielfaches der Page-Größe sein.

-s bandgr
Kapazität des Bandes in Kbyte. Die Größe, die Sie hier angeben, sollte mindestens 10% unter der tatsächlichen Größe des Bandes liegen. Sie könnten sonst beim Sichern Schwierigkeiten bekommen.

-f datei
schreibt die von *dbexport* erzeugten SQL-Anweisungen nicht auf Band, sondern in die angegebene Datei auf Platte. Fehlt die Angabe, so werden die SQL-Anweisungen in die Datei datenbank**.sql** auf Band ausgegeben.

datenbank
Name der Datenbank, die exportiert werden soll.

Hinweis

— Während des Programmablaufs ist die Datenbank exklusiv gesperrt. Falls dies nicht möglich ist, erscheint eine Fehlermeldung.

— Mit der Taste DEL können Sie den Programmablauf abbrechen. Dabei werden Sie aufgefordert, dies nochmals zu bestätigen.

— Die Datenbank wird als "Datenbank ohne Transaktionssicherung" exportiert. Soll die Datenbank beim Import (*dbimport*) wieder mit Transaktionssicherung arbeiten, so müssen Sie dies bei *dbimport* enstprechend angeben (Schalter **-l**).

Beispiel

Das folgende Kommando exportiert die Datenbank *versand* auf Band. Die Blockgröße des Bandes beträgt 16 Kbytes; jedes Band soll eine Datenmenge von 40000 Kbytes aufnehmen. Bei Fehlern soll das Programm fortfahren.

```
dbexport -c -t /dev/rts -b 16 -s 40000 versand
```

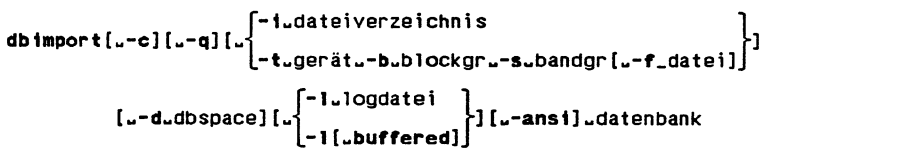
Das folgende Kommando exportiert die Datenbank *buecher* in das Dateiverzeichnis */usr/lomata/port*:

```
dbexport -c -o /usr/lomata/port buecher
```

A.3.3 dbimport - Datenbank importieren

Mit diesem Dienstprogramm erzeugen Sie eine Datenbank aus den Dateien, die Sie zuvor mit *dbexport* erzeugt haben. Die Kennung, unter der *dbimport* abläuft, erhält das DBA-Zugriffsrecht auf die Datenbank.

Stellen Sie vor dem *dbimport*-Aufruf sicher, daß die Umgebungsvariable SQLEXEC auf die gewünschte Umgebung (ONLINE oder SE gesetzt ist).



-c
gibt an, daß das Programm fortgesetzt werden soll, wenn Fehler auftauchen, ausgenommen folgende schwerwiegende Fehler:

- Das angegebene Bandgerät kann nicht eröffnet werden.
- Fehlerhafte Lesezugriffe auf Band oder Platte.
- Ungültige Parameter im Kommando.
- Datenbank kann nicht eröffnet werden oder keine Zugriffsberechtigung.

-q
unterdrückt Meldungen von SQL-Anweisungen am Bildschirm.

-i dateiverzeichnis
Importiert die Datenbank von Platte.
Geben Sie den absoluten oder relativen Pfadnamen eines Dateiverzeichnisses an, in dem die von *dbexport* erzeugten Dateien enthalten sind. Fehlt die Angabe, so gilt das aktuelle Dateiverzeichnis, d. h. *dbimport* sucht das Dateiverzeichnis datenbank.**exp** im aktuellen Dateiverzeichnis.

-t gerät
Importiert die Datenbank vom Band.
Geben Sie die Gerätedatei des Bandgerätes an.
Wenn die Datenbank vom Band importiert wird, dürfen Sie *dbimport* nicht als Hintergrundprozeß aufrufen.

-b blockgr
Blockgröße des Bandes in Kbyte. Die Angabe muß mit der Blockgröße übereinstimmen, die Sie bei *dbexport* angegeben haben.

-s bandgr
Kapazität des Bandes in Kbyte. Die Angabe muß mit der Bandgröße übereinstimmen, die Sie bei *dbexport* angegeben haben.

-f datei
Name der Datei, die auch bei *dbexport* (Schalter **-f**) angegeben ist.

-d dbspace
Nur für INFORMIX-ONLINE:
Name des Dbspaces, in dem die Datenbank eingerichtet werden soll. Fehlt die Angabe, so wird die Datenbank im Root-Dbspace eingerichtet.

-l
schaltet für die importierte Datenbank die Transaktionssicherung ein. Fehlt die Angabe, so wird keine Transaktionssicherung durchgeführt. Der Schalter **-l** muß angegeben werden, wenn die neue Datenbank als ANSI-Datenbank erzeugt wird (Angabe **-ansi**).

logdatei
Nur für INFORMIX-SE:
Absoluter Pfadname der Datei, in die das Transaktionsprotokoll geschrieben werden soll.

buffered
Nur für INFORMIX-ONLINE:
wählt gepufferte Protokollierung. Diese Angabe ist bei ANSII-Datenbanken verboten. Fehlt die Angabe, so gilt als Standardwert: ungepufferte Protokollierung.

-ansi
erzeugt die neue Datenbank als ANSI-Datenbank (MODE ANSI). Wenn Sie diese Angabe verwenden, so müssen Sie auch den Schalter **-l** angeben.

datenbank
Bei Import von Platte gilt: Name der Datenbank, wie er bei *dbexport* angegeben war. Die importierte Datenbank erhält denselben Namen.

Bei Import von Band gilt: Name, den die importierte Datenbank erhalten soll. Er muß nicht mit dem bei *dbexport* angegebenen Datenbanknamen übereinstimmen.

Hinweis

- Während des Programmablaufs ist die Datenbank exklusiv gesperrt.
- Beim Laden der Datenbank führt *dbimport* einen impliziten LOCK TABLE und UNLOCK TABLE durch.
- INFORMIX-SE: Die Datenbankdateien der importierten Datenbank werden im aktuellen Dateiverzeichnis angelegt.
- Ist der Schalter -l nicht angegeben, d. h. die Transaktionssicherung nicht eingeschaltet, so können Sie dies nachträglich durchführen:
  INFORMIX-ONLINE: über den DB-Monitor (siehe ONLINE-Handbuch [6])
  INFORMIX-SE: über die SQL-Anweisung START DATABASE (siehe SQL-Handbuch [2])
- Mit der Taste DEL können Sie den Programmablauf abbrechen. Dabei werden Sie aufgefordert, dies nochmals zu bestätigen.
- *dbimport* erzeugt die Meldungsdatei *dbimport.out*. Diese enthält alle Fehlermeldungen und Warnungen, die während des Programmablaufes entstehen.
- INFORMIX-ONLINE: Extent-Größe festlegen

  Die Extent-Größe für Tabellen stellt einen wichtigen Faktor für die Performance dar. Der Standardwert für die Größe des Initial-Extent von importierten Tabellen ist die Größe der importierten Tabelle. Der Standardwert für die nachfolgenden Extents beträgt 10 % der Größe des Initial-Extent. Falls zu erwarten ist, daß die importierte Tabelle wesentlich größer werden wird, können Sie diese Standardwerte durch Angabe eigener Größen außer Kraft setzen (siehe Beispiel im ONLINE-Handbuch [6]).
  Dazu editieren Sie die Anweisungsdatei datenbank.**sql** im **.exp**-Dateiverzeichnis, die *dbexport* beim Exportieren erzeugt hat oder - falls Schalter -f verwendet wurde - die dort angegebene Datei.
  Die Größen für die Initial-Extents und die nachfolgenden Extents können Sie in der CREATE TABLE-Anweisung hinzufügen (Beschreibung siehe SQL-Handbuch [2]).
  Danach kann *dbimport* mit der geänderten Anweisungsdatei datenbank.**sql** gestartet werden.

Beispiel

Eine Datenbank auf Band (Gerät /dev/rts soll als INFORMIX-ONLINE-Datenbank in den Dbspace *dbspace2* importiert werden. Die Datenbank soll den Namen *versand* erhalten. Bildschirmmeldungen durch SQL-Anweisungen sollen unterdrückt werden. Bei Fehlern soll das Programm fortgesetzt werden:

```
dbimport -c -t /dev/rts -b 16 -s 40000 -q -d dbspace2 versand
```

Die Datenbank *buecher* befindet sich auf Platte und soll als INFORMIX-ONLINE-Datenbank in den Root-Dbspace importiert werden. Das Dateiverzeichnis *versand.exp* befindet sich im Dateiverzeichnis */usr/lomata/port*:

```
dbimport -i /usr/lomata/port buecher
```

Beachten Sie, daß für den Import in die INFORMIX-ONLINE-Umgebung die Umgebungsvariable SQLEXEC entsprechend gesetzt sein muß.

A.3.4 dbload - Daten aus Datei in Datenbank einlesen

dbload ist ein Dienstprogramm, mit dem Sie Daten im ASCII-Format aus einer Betriebssystemdatei in eine Datenbank übertragen können. Es bietet Ihnen einen einfachen und leistungsfähigen Transfer von Daten aus einer anderen INFORMIX-Datenbank oder einem völlig anderem Datenbanksystem in Ihre Datenbank.

Die wichtigsten Funktionen von *dbload* sind:

- Daten aus bestimmten Feldern in einer oder mehreren Dateien können in ausgewählte Spalten einer oder mehrerer Datenbank-Tabellen geladen werden.
- Das Laden kann ab einer beliebig definierbaren Zeile der Eingabedatei beginnen.
- Beim Ladevorgang wird jeweils eine bestimmte von Ihnen definierbare Anzahl an Sätzen gemeinsam bearbeitet.
- Die Dateien können Sätze mit fester oder variabler Länge enthalten.
- Für jedes Feld kann ein bestimmter Wert als NULL-Wert definiert werden. Enthält ein Satz in diesem Feld den definierten Wert, so wird anstelle dieses Wertes ein NULL-Wert eingetragen.
- In der Anweisungsdatei können Sie auch direkt Werte angeben.
- Sätze, die nicht geladen werden können, werden zusammen mit einer Diagnose-Information in einer Fehlerdatei protokolliert.
- Sie können ein Fehlerlimit definieren, d. h. angeben, wieviel fehlerhafte Sätze während des Ladevorgangs maximal auftreten dürfen, ohne daß der Ladevorgang abgebrochen wird.
- Für Datenbanken mit Transaktionen können Sie festlegen, ob bei Abbruch des Ladevorgangs wegen Überschreitung des Fehlerlimits alle seit letztem COMMIT erfolgreich eingelesenen Sätze zurückgesetzt werden sollen.

Vorgehensweise

Für den *dbload*-Aufruf benötigen Sie die *Eingabedateien*, die die einzulesenden Daten enthalten, sowie eine *Anweisungsdatei*, die das Format der Eingabesätze beschreibt und entsprechend den Tabellenspalten zuordnet.

Die folgenden Abschnitte beschreiben den Aufbau von Eingabe- und Anweisungsdatei. Im Anschluß daran finden Sie die Syntax des *dbload*-Aufrufs.

Aufbau der Eingabedatei

Analog zur Aufteilung von Tabellensätzen in einzelne Spalten müssen die Sätze der Eingabedatei in einzelne Felder unterteilbar sein. Ein Feld des Eingabesatzes läßt sich somit genau einer Tabellenspalte zuordnen. Welche Bereiche des Eingabesatzes jeweils zu einem Feld zusammengefaßt werden, definieren Sie in der Anweisungsdatei.

Weitere Merkmale der Eingabedatei:

- Sie darf nur abdruckbare Zeichen enthalten.
- Die einzelnen Sätze müssen mit dem Zeilenende-Zeichen abgeschlossen sein. Soll das Zeilenendezeichen als Feldinhalt verwendet werden, so muß es mit Gegenschrägstrich \ entwertet werden.
- Die Satzlänge kann entweder fest oder variabel sein:
  - *Feste Satzlänge* bedeutet, daß alle Sätze gleich lang sind und daß die Feldpositionen der sich entsprechenden Inhalte in jedem Satz übereinstimmen müssen. Enthält z. B. ein Eingabesatz ab Spalte 9 als Feldinhalt einen Nachnamen, so müssen sinngemäß in allen Eingabesätzen die Nachnamen ab Spalte 9 enthalten sein und auch dieselbe Länge haben (ggf. mit Leerzeichen auffüllen).
    In der Anweisungsdatei können Sie für Sätze fester Länge einen bestimmten Feldinhalt als NULL-Wert definieren. Beim Ladevorgang wird dann anstelle des dieses Feldinhaltes ein NULL-Wert übertragen.
  - *Variable Satzlänge* bedeutet, daß die sich entsprechenden Felder jedes Satzes (und damit jeder Satz) unterschiedliche Länge haben. Die Feldgrenzen müssen daher durch ein bestimmtes Zeichen markiert sein. Welches Zeichen als Feldtrennzeichen interpretiert werden soll, legen Sie in der Anweisungsdatei fest. Soll das Feldtrennzeichen als Feldinhalt interpretiert werden, so muß es mit Gegenschrägstrich \ entwertet werden.
    Zwei aufeinanderfolgeden Feldtrennzeichen bezeichnen ein leeres Feld. Beim Ladevorgang wird dafür ein NULL-Wert übertragen.

- Für alphanumerische Feldinhalte gilt: Stimmt die Länge nicht überein mit der Länge der Tabellenspalte, so wird beim Ladevorgang enrsprechend gekürzt oder mit Leerzeichen aufgefüllt.
- Die Feldinhalte müssen den Datentypen der Tabellenspalten entsprechen, denen Sie später zugeordnet werden:
  - Enthält ein Feldinhalt führende Leerzeichen, so darf er nur Tabellenspalten zugeordnet werden, für die der Datentyp DATE, MONEY oder ein numerischer Datentyp definiert ist. Felder, die Tabellenspalten vom Datentyp MONEY zugeordnet werden, dürfen zusätzlich das Währungssymbol enthalten. Es muß an erster Stelle stehen.
  - Felder, die Tabellenspalten vom Datentyp DATE zugeordnet werden, müssen das Datum in dem Format enthalten, das in der Umgebungsvariablen DBDATE definiert ist.
  - Felder, die Tabellenspalten vom Datentyp DATETIME oder INTERVAL zugeordnet werden, müssen die einzelnen Zeitangaben in der Form yyyy-mm-dd-hh:mm:ss enthalten (siehe Beschreibung dieser Datentypen im SQL-Handbuch [2].
  - INFORMIX-ONLINE: Felder, die Tabellenspalten vom Datentyp BYTE zugeordnet sind, müssen Daten im ASCII-Format enthalten. Führende Leerzeichen sind nicht erlaubt.

Ein einfaches Beispiel für eine zulässige Eingabedatei stellt die von der SQL-Anweisung UNLOAD erzeugte Datei dar. Diese enthält Sätze variabler Länge. Die einzelnen Felder durch ein bei UNLOAD definiertes Feldtrennzeichen gekennzeichnet (UNLOAD-Beschreibung siehe SQL-Handbuch [2]).

Aufbau der Anweisungsdatei

Die Anweisungsdatei ordnet die Daten der Eingabedateien den Tabellen der Datenbank zu. Sie enthält zweierlei Arten von Anweisungen:

- FILE-Anweisung: Dort geben Sie den Namen einer Eingabedatei an und definieren die Felder der Eingabesätze.
- INSERT-Anweisung: Dort ordnen Sie die in der FILE-Anweisung definierten Felder den Tabellenspalten zu.

Die Anweisungsdatei enthält immer eine Folge von FILE- und INSERT-Anweisungen, wobei für die Anordnung gilt: Auf eine FILE-Anweisung folgen immer eine oder mehrere INSERT-Anweisungen, die sich auf diese FILE-Anweisung beziehen. In einer Anweisungsdatei lassen sich mehrere solcher FILE-INSERT-Folgen definieren.

Syntax der Anweisungen

Kommentare innerhalb der Anweisungsdatei sind in geschweifte Klammern { } zu setzen. Mit Ausnahme der letzten Anweisung müssen sämtliche Anweisungen mit Semikolon ; abgeschlossen werden.

Die erste FILE-Anweisung zeigt die Syntax für Sätze variabler Länge, die zweite die für Sätze fester Länge.

```
FILE␣"datei"␣DELIMITER␣"c"␣feldanzahl;     {für variable Satzlänge}
FILE␣"datei"                                  {für feste Satzlänge}
     (feld1␣start[-end][:␣...] [␣NULL = "str1"],
      feld2␣start[-end][:␣...] [␣NULL = "str2"],
      ...
      ...
      feldn␣start[-end][:␣...] [␣NULL = "strn"]);
INSERT␣INTO␣tabelle [(spaltenname,...)] [VALUES␣(wert,...)]; [...;]
```

FILE
: definiert die Eingabedatei und die Eingabefelder. Je Eingabedatei muß 1 FILE-Anweisung definiert werden.

"datei"
: Name der Eingabedatei, eingeschlossen in Anführungszeichen "". Die Benutzerkennung, unter der *dbload* abläuft, muß Leseberechtigung auf die Datei haben.

DELIMITER
: bezeichnet das Feldtrennzeichen für Sätze variabler Länge. Das Feldtrennzeichen muß innerhalb der gesamten Eingabedatei einheitlich sein.
dbload ordnet automatisch jedem Feld des Eingabesatzes die Feldnamen **f01, f02, f03, ...** usw. zu. Diese Feldnamen müssen Sie in der INSERT-Anweisung angeben, wenn Sie auf die Felder Bezug nehmen.

"c"
Feldtrennzeichen, eingeschlossen in Anführungszeichen "". Das Feldtrennzeichen muß auch am Ende des letzten Feldes (vor dem Zeilenendezeichen) gesetzt werden. Wenn Sie das Feldtrennzeichen am Ende weglassen, läuft *dbload* auf Fehler, falls das letzte Feld leer ist, weil dann die Feldanzahl nicht mehr stimmt.

feldanzahl
Anzahl Felder im Eingabesatz; die Anzahl muß in allen Sätzen übereinstimmen.

feld1 ... feldn
hier definieren Sie Feldnamen für Eingabesätze fester Länge. Die nachfolgenden INSERT-Anweisungen, die sich auf diese FILE-Anweisung beziehen, verwenden diese Feldnamen.

start-end
bezeichnet den Wertebereich im Eingabesatz, der unter dem definierten Feldnamen angesprochen werden soll. *start* bezeichnet die Anfangsspalte, *end* die Endspalte des Wertes im Eingabesatz. Sie können hier für den angegebenen Feldnamen ein Feld aus mehreren Wertebereichen zusammensetzen; Sie müssen dann die einzelnen Wertebereiche *start - end* durch den Doppelpunkt trennen (z. B. 1-10 : 15-18 usw.). Dabei dürfen sich die angegebenen Wertebereiche auch überlappen, z. B. 1-10 : 5-22.
Fehlt die Angabe von *end*, so gilt als Standardwert: 1 Zeichen ab *start*.

NULL="str1"..."strn"
definiert den Feldinhalt, der als NULL-Wert interpretiert werden soll, d. h. anstelle dieses Feldinhalts wird beim Laden der NULL-Wert übertragen. Für jedes Feld läßt sich ein eigene Zeichenfolge definieren, die als NULL-Wert zu interpretieren ist.

INSERT INTO
hier definieren Sie eine INSERT-Anweisung. Die Angabe des Operanden *select-anweisung* ist nicht erlaubt. Ansonsten entspricht die Syntax der von INSERT (siehe SQL-Handbuch [2]).

tabelle
bezeichnet die Tabelle, in die die Daten geladen werden sollen. Die Benutzerkennung, unter der *dbload* läuft, muß INSERT-Zugriffsrecht für die angegebene Tabelle haben.
Der Tabellenname darf auch in der Form *eigentümer.tabelle* angegeben werden.

(spaltenname,...)
bezeichnet die Spalten der Tabelle, in die Werte einzutragen sind. Ausführliche Beschreibung siehe INSERT-Anweisung im SQL-Handbuch [2]. Für Tabellenspalten, die hier nicht angegeben sind, versucht INFORMIX NULL-Werte zu übertragen. Falls dies aufgrund der Tabellendefinition nicht möglich ist, erscheint eine Fehlermeldung bei Ablauf des Programms.
Fehlt die Angabe, so gelten alle Spalten der Tabelle in der dort definierten Reihenfolge.

VALUES
bezeichnet die Liste der Werte, die den angegebenen Spalten zuzuordnen sind.

(wert,...)
Mit *wert* bezeichnen Sie entweder direkt eine Konstante, die in die Spalte übertragen werden soll oder Sie geben den Feldnamen an, wie er in der vorangehenden FILE-Anweisung definiert wurde.
Für die Zuordnung der Werte gilt: die Werte der Werteliste werden in der definierten Reihenfolge den Spalten der angegebenen Spaltenliste zugeordnet (1. Wert der Wertelist wird 1. Spalte der Spaltenliste
zugeordnet usw.).
Die Anzahl der Werte in der Werteliste muß übereinstimmen mit der Anzahl der Spalten in der Spaltenliste.

Syntax des dbload-Aufrufs

Vor dem Aufruf beachten

Fehlt eine der Angaben *-d datenbank, -c anweisungsdatei* oder *-l fehlerprotokoll*, so fragt *dbload* unabhängig vom Standardwert sämtliche im Aufruf fehlenden Operanden im Dialog ab. Soll für einen Operanden der Standardwert angenommen werden, so müssen Sie bei Abfrage dieses Operanden eine "leere" Eingabe machen, d. h. die Taste ↵ drücken.

Enthält der *dbload*-Aufruf alle drei oben genannten Operanden, so setzt *dbload* für alle anderen nicht angegebenen Operanden automatisch den definierten Standardwert ein.

```
dbload[␣-d[␣datenbank][␣-c[␣anweisungsdatei][␣-l[␣fehlerprotokoll]]
      [␣-e[␣anz1][␣-n[␣anz2][␣-i[␣anz3][␣-p][␣-r][␣-s [> ausgabedatei]]
```

-d datenbank
Name der Datenbank, in die die Daten übertragen werden sollen. Fehlt hier der Datenbankname, so wird er von *dbload* automatisch angefordert.

-c anweisungsdatei
Name der Anweisungsdatei, die die erforderlichen Anweisungen enthält. Fehlt hier der Name der Anweisungsdatei, so wird er von *dbload* automatisch angefordert.

-l fehlerprotokoll
Name für das Fehlerprotokoll. Fehlt hier der Name des Fehlerprotokolls, so wird er von *dbload* automatisch angefordert.

-e anz1
bezeichnet die Anzahl der fehlerhaften Sätze, die bearbeitet werden, ehe *dbload* abbricht. Standardwert: 10 Das heißt: *dbload* bricht beim elften fehlerhaften Satz ab.

-n anz2
bezeichnet die Anzahl der Sätze, die von *dbload* erfolgreich gelesen werden müssen, ehe sie in die Datenbank übertragen werden. Bevor diese Anzahl nicht erreicht ist, werden keine Sätze in die Datenbank übertragen. Standardwert: 100 Das heißt: Jeweils Einheiten zu 100 Sätzen werden übertragen. *dbload* gibt nach jeder übertragen Einheit eine Meldung aus.

-i anz3
bezeichnet die Anzahl der Sätze in der Datei, die beim Einlesen übersprungen werden sollen. Das heißt: Der erste eingelesene Satz ist der Satz, der auf die angegebene Anzahl Sätze folgt. Diese Angabe ist vor allem nützlich,

- wenn Sie einen zuvor abgebrochenen Ladevorgang an der unterbrochenen Stelle fortsetzen wollen.
- wenn Sie allgemeine Kommentare am Anfang der Eingabedatei überspringen wollen.

-p
Nur für Datenbanken mit Transaktionssicherung:
Bewirkt, daß bei Abbruch von *dbload* eine Abfrage erscheint, ob die bisher eingelesenen (aber nicht übertragenen) Sätze übertragen (≙ COMMIT) oder zurückgesetzt (≙ ROLLBACK) werden sollen. Standardwert: Die Sätze werden übertragen.

-r
weist *dbload* an, keine Tabellen zu sperren.

-s
prüft nur die Syntax der Anweisungen in der Anweisungsdatei. Es werden keine Daten in die Datenbank übertragen. Diese Angabe verwenden Sie vor dem eigentlichen Laden der Daten. Die Anweisungsdatei wird zeilenweise auf dem Bildschirm ausgegeben. Dabei werden Syntaxfehler gekennzeichnet.

> ausgabedatei
schreibt das Ergebnis der Syntaxprüfung in die angegebene Datei.

Hinweis

— Sämtliche Dateinamen können auch mit dem Pfadnamen angegeben werden.

— Für Datenbanken mit Transaktionssicherung gilt: Wenn Sie *dbload* selbst abbrechen (Taste DEL), dann werden alle seit dem letzten COMMIT eingelesenen Sätze wieder zurückgesetzt.

— Die Geschwindigkeit, in der die Daten übertragen werden, kann beeinträchtigt werden, wenn die Empfänger-Tabelle Indizes hat. Daher empfiehlt es sich, die Indizes vor dem *dbload*-Aufruf zu löschen und danach neu zu erzeugen.

Beispiel für den dbload-Aufruf

dbload-Aufruf, der mit Ausnahme von **-s** sämtliche Operanden zeigt:

```
dbload -d db -c adat -l fdat -e 5 -n 75 -i 20 -p -r
```

Die Angaben bedeuten:

-d db	Die Sätze sollen in die Datenbank *db* übertragen werden.
-c adat	Die Anweisungsdatei heißt *adat*.
-l fdat	Die Datei für das Fehlerprotokoll heißt *fdat*.
-e 5	*dbload* bricht beim sechsten fehlehaften Satz ab.

-e 75 *dbload* überträgt die Sätze jeweils in Einheiten zu 75 fehlerfreien Sätzen.

-i 20 *dbload* ignoriert die ersten 20 Zeilen der Eingabedatei.

-p bei Abbruch von *dbload* erscheint eine Abfrage, was mit bereits eingelesenen, aber noch nicht übertragenen Sätzen geschehen soll.

-r die Tabellen, in die Daten übertragen werden, sollen während des Ladevorgangs nicht gesperrt werden.

Beispiel für den Aufbau einer Anweisungsdatei

```
FILE datei1 (feld1  1 - 10 : 13      :  5 - 22  NULL = "string1",
             feld2 10 - 21 : 28 - 32,
             feld3  8 - 10 : 33 - 50 : 29 - 33  NULL = "string2",
             ...
             ...
             feldn  9      : 16 - 19            NULL = "stringn");

INSERT INTO tab1 (col1, col2, col9, ... , coln) ;

INSERT INTO tab2 VALUES (feld1, feld3, "abc", ... , feldn) ;

INSERT INTO tab3 ;     {ohne Angabe von Feldern und Werten }

FILE "datei.2" DELIMITER "|" num       {variable Feldlänge}

INSERT INTO tab1 VALUES (f01, f02, "abc", "234", ... , f0n) ;

INSERT INTO tab4 ;
```

Bedeutung der einzelnen Anweisungen:

```
-  FILE datei1 (feld1  1 - 10 : 13      :  5 - 22  NULL = "string1",
                feld2 10 - 21 : 28 - 32,
                feld3  8 - 10 : 33 - 50 : 29 - 33  NULL = "string2",
                ...
                ...
                feldn  9      : 16 - 19            NULL = "stringn");
```

dateil ist der Name der Eingabedatei.

feld1 ... feldn Feldnamen, die Datenfelder fester Länge identifizieren.

In dem gezeigten Beispiel setzt sich *feld1* zusammen aus den Zeichen 1 bis 10, Zeichen 13 und Zeichen 5 bis 22 eines jeden Satzes der Datei.

Zeichenpositionen dürfen beliebig wiederholt werden. Im Beispiel erscheinen die Zeichenpositionen 5 bis 10 und 13 zweimal in *feld1*; die Zeichenpositionen 10, 13 und 21 erscheinen sowohl in *feld1* als auch in *feld2*.

In *feld1* ist der NULL-Wert als "string1" definiert. *dbload* setzt in das Feld immer dann einen NULL-Wert, wenn es auf "string1" trifft.

feld2 besteht aus den Zeichen in Position 10 bis 21 und 28 bis 32.

feld3 besteht aus den Zeichen in Position 8 bis 10, 33 bis 50 und 29 bis 33. Der NULL-Wert in diesem Feld ist als "string2" definiert.

Die Felddefinition wird fortgesetzt, bis das letzte Feld erreicht ist. Hier enthält *feldn* die Zeichen in Postionen 9 und 16 bis 19. Der NULL-Wert in diesem Feld ist als "stringn" definiert.

Die Felddefinitionen sind in runde Klammern () einzuschließen. Zeichen und Zeichenfolgen sind durch einen Doppelpunkt : zu trennen.

- `INSERT INTO tab1 (col1, col2, col9, ... , coln) ;`

  Es folgt eine INSERT-Anweisung. Da es keine Werteliste gibt, nimmt *dbload* die Werteliste der vorhergehenden FILE-Anweisung. *dbload* überträgt die Daten von *feld1* nach *col1*, die von *feld2* nach *col2* und die von *feld3* nach *col9* der Tabelle *tab1*. (In diesem Fall werden die Tabellenspalten 4 bis 8 übergangen). Dieser Vorgang wird fortgesetzt, bis die Daten von *feldn* in das Feld *coln* übertragen sind.

- `INSERT INTO tab2 VALUES (feld1, feld3, "abc", ... , feldn) ;`

  Da es keine Spaltenliste gibt, gelten aus *tab2* alle Spalten, und zwar in der Reihenfolge, in der sie in den Systemtabellen abgspeichert sind. Die Werte, die in jedes Feld geladen werden sollen, werden eigens angegeben. *dbload* überträgt die Daten von *feld1* in die 1. Spalte der Tabelle, die von *feld3* in die 2. Spalte und die Konstante "abc" in die 3. Spalte der Tabelle *tab2*. Dieser Vorgang wird fortgesetzt, bis der Wert von *feldn* in das letzte Feld übertragen ist.

- `INSERT INTO tab3 ;  {ohne Angabe von Feldern und Werten }`

  Da weder Feld- noch Werteliste angegeben ist, gelten alle Felder von *tab3* in der abgespeicherten Reihenfolge und die Werteliste aus der vorhergehenden FILE-Anweisung.

  Hier wird der Wert von *feld1* in die 1. Spalte übertragen, der von *feld2* in die 2. Spalte usw.

  Anmerkung: Diese Anweisung erfordert es, daß die Feldliste in der FILE-Anweisung eine genaue (1:1) Entsprechung zu den Feldern der Tabelle enthält. Ist diese Entsprechung nicht gegeben, lädt *dbload* die Sätze nicht. Sie erhalten stattdessen für jeden Satz eine Fehlermeldung. *dbload* beendet den Ladevorgang, wenn die Gesamtzahl der fehlerhaften Sätze die vom Benutzer definierte Grenze (falls angegeben) oder den Standardwert von 10 fehlerhaften Sätzen überschreitet.

- `FILE "datei.2" DELIMITER "|" num  {variable Feldlänge}`

  Die Felder in *datei.2* haben variable Länge. Mit der DELIMITER-Klausel wird als Feldtrennzeichen ein senkrechter Strich "|" definiert. Dieses Feldtrennzeichen muß in der gesamten Datei verwendet werden und ist in Anführungszeichen einzuschließen. 'num' bezeichnet die Anzahl der Felder, die einen Satz in der Datendatei bilden. Den Feldern werden automatisch die Namen *f01, f02,f03* usw. zugewiesen. Jedem Feld, das kein Zeichen enthält, wird ein NULL-Wert zugewiesen.

  Der Kommentar "{variable Feldlänge}" ist in geschweiften Klammern eingeschlossen.

- `INSERT INTO tab1 VALUES (f01, f02, "abc", "234", ... , f0n) ;`

  Da es keine Spaltenliste gibt, gelten alle Tabellenspalten von *tab1* in der abgespeicherten Reihenfolge. Werte, die in jede Spalte von *tab1* übertragen werden sollen, werden eigens angegeben. Hier wird der Wert von *f01* in die 1. Spalte übertragen, der von *f02* in die 2. Spalte, die Konstante "abc" in die 3. Spalte, der Wert "234" in die 4. Spalte usw.

- `INSERT INTO tab4 ;`

  Da weder Spalten- noch Werteliste angegeben ist, gelten alle Spalten der Tabelle *tab4* in der abgespeicherten Reihenfolge. Die Werteliste wird aus der vorhergehenden FILE-Anweisung genommen. Die Feldliste in der FILE-Anweisung muß eine genaue (1:1) Entsprechung zu den Spalten der Tabelle *tab4* enthalten. Hier wird der Wert von *f01* in die 1. Spalte übertragen, der von *f02* in die 2. Spalte usw.

A.3.5 dblog - Transaktionsprotokoll-Dateien ausgeben

Das Dienstprogramm *dblog* gibt den Inhalt von Transaktionsprotokoll-Dateien aus, die mit INFORMIX-SE oder C-ISAM erstellt wurden.

Dieses Dienstprogramm steht nur dem Anwender von INFORMIX-SE zur Verfügung, INFORMIX-ONLINE-Anwender benutzen anstelle von *dblog* das Dienstprogramm *tblog* (siehe Handbuch INFORMIX-ONLINE [6]).

```
dblog[␣-v][␣-l][␣-h␣anzahl]␣protokolldatei
      [␣-d␣anfangsdatum␣endedatum]
      [␣-t␣anfangszeit␣endezeit]
      [␣-u␣benutzer]
      [␣-f␣dateiname]
      [␣-r␣anfangsposition␣endeposition]
```

-v
zusätzliche Information für spezielle Typen von Protokollsätzen ausgeben.

-l
Information über Postition der Protokollsätze in der Protokolldatei ausgeben.

-h anzahl
gibt an, nach wieviel Zeilen die Überschrift bei der Ausgabe wiederholt werden soll.
Wenn Sie 0 angeben, wird die Überschrift nur einmal am Anfang ausgegeben.
Standard ist 20.

protokolldatei
Name der Transaktionsprotokoll-Datei.

-d anfangsdatum endedatum
Es werden nur die Aktivitäten von Anfangsdatum bis Endedatum ausgegeben. Das Format ist mm/tt; mm = Monat, tt = Tag

-t anfangszeit endezeit
Es werden nur die Aktivitäten von Anfangszeit bis Endezeit ausgegeben.

-u benutzer
Es werden nur die Aktivitäten der angegebenen Benutzerkennung ausgegeben.

-f dateiname
Es werden nur die Aktivitäten ausgegeben, die sich auf die Datenbank-Tabelle beziehen, die in *dateiname* gespeichert ist.

-r anfangsposition endeposition
Es werden nur die Aktivitäten von Anfangs- bis Endeposition der Protokollsätze ausgegeben. Die Position eines Protokollsatzes wird in Anzahl Bytes vom Anfang der Protokolldatei angegeben.

Ausgabe der Protokollsätze

Für jeden Protokollsatz werden standardmäßig folgende Felder ausgegeben:

TY	Typ des Protokollsatzes (siehe Ausgabe bei -v)
TrxID	Transaktionsnummer
User	SINIX-Benutzernummer, die zum Protokollsatz gehört.
User Name	Benutzerkennung, die zur Benutzernummer gehört.
Date/Time	Datum und Zeit, zu der der Protokollsatz geschrieben wurde.
Lngth	Länge des Protokollsatzes.
FD	Dateizeiger (File-Deskripotor) der betroffenen C-ISAM-Datei.
Recno	Rowid des Datensatzes, den der Protokollsatz betrifft.
Filename	Dateiname der vom Protokollsatz betroffenen Datei; für Satztypen open, close, build, erase und rename.

Ausgabe bei -l

Wenn Sie mit dem Schalter -l die Ausgabe der Position der Protokollsätze anfordern, werden zusätzlich folgende drei Felder ausgegeben:

Location	Position des Protokollsatzes; die Position wird in Anzahl Bytes vom Dateianfang der Protokolldatei ausgegeben.
Prev Loc	Position des vorhergehenden Protokollsatzes in derselben Transaktion wie der aktuelle Protokollsatz (nicht bei allen Protokollsatz-Typen verwendet)
Prev Len	Länge des vorhergehenden Protokollsatzes in derselben Transaktion wie der aktuelle Protokollsatz (nicht bei allen Protokollsatz-Typen verwendet)

Ausgabe bei -v

Wenn Sie mit dem Schalter -v Zusatzausgaben anfordern, werden für einige Protokollsatz-Typen zusätzliche Felder ausgegeben. Die folgende Übersicht zeigt die Protokollsatz-Typen und die zusätzlich ausgegebenen Felder.

Protokollsatz	Code	zusätzlich ausgegebene Felder
ISAM-Datei aufbauen	BU	Row Length (ISAM-Satzlänge) Build Mode (C-ISAM-Modus der ISAM-Datei)
ISAM-Datei schließen	FC	keine zusätzlichen Felder
ISAM-Datei öffenen	FO	keine zusätzlichen Felder
Datensatz aus ISAM-Datei löschen	DE	Image (Hexadezimal und ASCII-Ausgabe des Protokollsatzes)
Datensatz in ISAM-Datei einfügen	IN	Image (Hexadezimal- und ASCII-Ausgabe des Protokollsatzes)
ISAM-Datei umbenennen	RE	Neuer Dateiname, der im Protokollsatz enthalten ist
Datensatz in ISAM-Datei ändern	UP	Pre-Image (Hexadezimal- und ASCII-Ausgabe des Before Image des Datensatzes im Protokollsatz) Post-Image (Hexadezimal- und ASCII-Ausgabe des Aftere Image des Datensatzes im Protokollsatz)

Unique Id für ISAM-Datei anfordern	UN	keine zusätzlichen Felder
Unique Id für ISAM-Datei setzen	SU	keine zusätzlichen Felder
ISAM-Datei löschen	ER	keine zusätzlichen Felder
BEGIN WORK	BW	keine zusätzlichen Felder
COMMIT WORK	CW	keine zusätzlichen Felder
ROLLBACK WORK	RW	keine zusätzlichen Felder
Index für ISAM-Datei erzeugen	CI	Beschreibung der Index-Spalten (Key)
Index einer ISAM-Datei löschen	DI	Beschreibung der Index-Spalten (Key)
Datensätze physisch nach Schlüsselwerten sortieren (Cluster-Index)	CL	Beschreibung der Index-Spalten (Key)

Die drei Protokollsatz-Typen, die Indizes betreffen (CI, DI und CL) enthalten genaue Information über die Index-Spalten (Keys). Format und Inhalt der Index-Spalten werden in folgenden Feldern ausgegeben:

```
key.k_nparts = n   key.k_flags = n  key.k_len = n
key.k_part:   kp_start    kp_leng   kp_type
```

key.k_nparts — Anzahl der Spalten des Index, wie sie in der Index-Beschreibung des Protokollsatzes enthalten sind.

key.k_flags — Anzahl der Index-Flags, die zur Index-Beschreibung des Protokollsatzes gehören.

key.k_len — Länge der Index-Beschreibung im Protokollsatz

key.k_part — beschreibt für jede Index-Spalte:

- kp_start — Anfangsbyte der Index-Spalte im gesamten (evtl.zusammengesetzten) Index
- kp_leng — Länge der Index-Spalte
- kp_type — Sortierreihenfolge der Index-Spalte

Hinweise

- Sie können die Schalter -v, -l, und -h zusammen angeben. Die Angabe der Position des Protokollsatzes (Schalter -l) wird vor allem während des Rücksetzens einer Transaktion mit ROLLBACK benutzt.
- Wenn Sie mit dem Schalter -t eine Anfangs- und Endezeit auswählen, ohne gleichzeitig mit dem Schalter -d ein Anfangs- und Endedatum auszuwählen, wird der aktuelle Tag verwendet.
- Mit dem Schalter -f geben Sie die gewünschte ISAM-Datei an, in der die Datenbank-Tabelle gespeichert ist. Den Namen der Datei können Sie mit folgender SQL-Anweisung abfragen:

```
SELECT dirpath
  FROM systables
  WHERE tabname = tabellenname
```

Beispiele

Die folgenden Beispiele benützen die Protokolldatei *allcall.log*.

1. Alle Protokollsätze im Standardformat ausgeben:
   dblog allcall.log
2. Alle Protokollsätze mit Position und zusätzlicher Information ausgeben:
   dblog -l -v allcall.log
3. Alle Protokollsätze ausgeben, ohne zusätzliche Überschriften:
   dblog -h 0 allcall.log
4. Alle Protokollsätze ausgeben, mit einer Überschrift für jeden Protokollsatz:
   dblog -h 1 allcall.log
5. Alle Protokollsätze zwischen 9/11/89 und 9/20/89 ausgeben; Sätze vom Anfangs- und Endedatum sind in der Ausgabe enthalten.
   dblog -d 9/11/89 9/20/89 allcall.log
6. Alle Protokollsätze des heutigen Tages zwischen 10:01:01 und 15:59:59, einschließlich:
   dblog -t 10:01:01 15:59:59 allcall.log
7. Alle Protokollsätze ausgeben, die der Benutzer *informix* geschrieben hat:
   dblog -u informix allcall.log
8. Alle Protokollsätze ausgeben, die die C-ISAM-Datei *isfile* betreffen:
   dblog -f isfile allcall.log

9. Alle Protokollsätze ausgeben, der Position von 100 Bytes bis 550 Bytes in der Protokolldatei reicht.
   dblog -r 100 550 allcall.log

Beispielausgaben

Standardausgabe:

```
DBLOG: Transaction Log File Display C-ISAM Version 4.00.U
Copyright (C) 1981-1989 Informix Software, Inc.
Software Serial Number RDS#N000000

TY Trx ID User User Name    Date/Time     Lngth FD Recno Filename
-- ------ ---- ------------ ------------- ----- -- ----- --------
BU    688  200 informix    9/ 8 17:16:55    37  0        isfile
BW    688  200 informix    9/ 8 17:16:55    20
FO    688  200 informix    9/ 8 17:16:55    29  0        isfile
IN    688  200 informix    9/ 8 17:16:55    36  0     1  isfile
FC    688  200 informix    9/ 8 17:16:55    29  0        isfile
CW    688  200 informix    9/ 8 17:16:55    20
BW    688  200 informix    9/ 8 17:16:56    20
FO    688  200 informix    9/ 8 17:16:56    29  0        isfile
IN    688  200 informix    9/ 8 17:16:56    36  0     2  isfile
FC    688  200 informix    9/ 8 17:16:56    29  0        isfile
RW    688  200 informix    9/ 8 17:16:56    20
BW    688  200 informix    9/ 8 17:16:56    20
FO    688  200 informix    9/ 8 17:16:56    29  0        isfile
CL    688  200 informix    9/ 8 17:16:56    28  0
FC    688  200 informix    9/ 8 17:16:57    33  0        issaa00688
CW    688  200 informix    9/ 8 17:16:57    20
BW    688  200 informix    9/ 8 17:16:58    20
FO    688  200 informix    9/ 8 17:16:58    29  0        isfile
IN    688  200 informix    9/ 8 17:16:58    36  0     2  isfile
IN    688  200 informix    9/ 8 17:16:58    36  0     3  isfile

TY Trx ID User User Name    Date/Time     Lngth FD Recno Filename
-- ------ ---- ------------ ------------- ----- -- ----- --------
UP    688  200 informix    9/ 8 17:16:58    48  0     1  isfile
DE    688  200 informix    9/ 8 17:16:58    36  0     2  isfile
FC    688  200 informix    9/ 8 17:16:58    29  0        isfile
CW    688  200 informix    9/ 8 17:16:58    20
BW    688  200 informix    9/ 8 17:16:58    20
FO    688  200 informix    9/ 8 17:16:58    29  0        isfile
CI    688  200 informix    9/ 8 17:16:58    34  0
FC    688  200 informix    9/ 8 17:16:58    29  0        isfile
CW    688  200 informix    9/ 8 17:16:58    20
BW    688  200 informix    9/ 8 17:16:58    20
FO    688  200 informix    9/ 8 17:16:58    29  0        isfile
DI    688  200 informix    9/ 8 17:16:58    34  0
FC    688  200 informix    9/ 8 17:16:58    29  0        isfile
CW    688  200 informix    9/ 8 17:16:58    20
BW    688  200 informix    9/ 8 17:16:59    20
FO    688  200 informix    9/ 8 17:16:59    29  0        isfile
UN    688  200 informix    9/ 8 17:16:59    26  0     1
SU    688  200 informix    9/ 8 17:16:59    26  0   100
UN    688  200 informix    9/ 8 17:16:59    26  0   100
FC    688  200 informix    9/ 8 17:16:59    29  0        isfile

TY Trx ID User User Name    Date/Time     Lngth FD Recno Filename
-- ------ ---- ------------ ------------- ----- -- ----- --------
CW    688  200 informix    9/ 8 17:16:59    20
RE    688  200 informix    9/ 8 17:16:59    39           isfile
ER    688  200 informix    9/ 8 17:17:00    28           isfile2

Program over.
```

Ausgabe mit dblog -l

```
DBLOG: Transaction Log File Display C-ISAM Version 4.00.U
Copyright (C) 1981-1989 Informix Software, Inc.
Software Serial Number RDS#N000000

TY Trx ID User User Name    Date/Time     Lngth FD Recno Filename      Location Prev Loc Pr Ln
-- ------ ---- ------------ ------------- ----- -- ----- ------------ -------- -------- -- --
BU    688  200 informix     9/ 8 17:16:55    37  0       isfile              0        0  0
BW    688  200 informix     9/ 8 17:16:55    20                          37        0  0
FO    688  200 informix     9/ 8 17:16:55    29  0       isfile              57        0  0
IN    688  200 informix     9/ 8 17:16:55    36  0     1 isfile              86       37 20
FC    688  200 informix     9/ 8 17:16:55    29  0       isfile             122        0  0
CW    688  200 informix     9/ 8 17:16:55    20                         151       86 36
BW    688  200 informix     9/ 8 17:16:56    20                         171      151 20
FO    688  200 informix     9/ 8 17:16:56    29  0       isfile             191        0  0
IN    688  200 informix     9/ 8 17:16:56    36  0     2 isfile             220      171 20
FC    688  200 informix     9/ 8 17:16:56    29  0       isfile             256        0  0
RW    688  200 informix     9/ 8 17:16:56    20                         285      220 36
BW    688  200 informix     9/ 8 17:16:56    20                         305      285 20
FO    688  200 informix     9/ 8 17:16:56    29  0       isfile             325        0  0
CL    688  200 informix     9/ 8 17:16:56    28  0                          354        0  0
FC    688  200 informix     9/ 8 17:16:57    33  0       issaa00688         382        0  0
CW    688  200 informix     9/ 8 17:16:57    20                         415      305 20
BW    688  200 informix     9/ 8 17:16:58    20                         435      415 20
FO    688  200 informix     9/ 8 17:16:58    29  0       isfile             455        0  0
IN    688  200 informix     9/ 8 17:16:58    36  0     2 isfile             484      435 20
IN    688  200 informix     9/ 8 17:16:58    36  0     3 isfile             520      484 36

TY Trx ID User User Name    Date/Time     Lngth FD Recno Filename      Location Prev Loc Pr Ln
-- ------ ---- ------------ ------------- ----- -- ----- ------------ -------- -------- -- --
UP    688  200 informix     9/ 8 17:16:58    48  0     1 isfile             556      520 36
DE    688  200 informix     9/ 8 17:16:58    36  0     2 isfile             604      556 48
FC    688  200 informix     9/ 8 17:16:58    29  0       isfile             640        0  0
CW    688  200 informix     9/ 8 17:16:58    20                         669      604 36
BW    688  200 informix     9/ 8 17:16:58    20                         689      669 20
FO    688  200 informix     9/ 8 17:16:58    29  0       isfile             709        0  0
CI    688  200 informix     9/ 8 17:16:58    34  0                          738        0  0
FC    688  200 informix     9/ 8 17:16:58    29  0       isfile             772        0  0
CW    688  200 informix     9/ 8 17:16:58    20                         801      689 20
BW    688  200 informix     9/ 8 17:16:58    20                         821      801 20
FO    688  200 informix     9/ 8 17:16:58    29  0       isfile             841        0  0
DI    688  200 informix     9/ 8 17:16:58    34  0                          870        0  0
FC    688  200 informix     9/ 8 17:16:58    29  0       isfile             904        0  0
CW    688  200 informix     9/ 8 17:16:58    20                         933      821 20
BW    688  200 informix     9/ 8 17:16:59    20                         953      933 20
FO    688  200 informix     9/ 8 17:16:59    29  0       isfile             973        0  0
UN    688  200 informix     9/ 8 17:16:59    26  0     1                   1002        0  0
SU    688  200 informix     9/ 8 17:16:59    26  0   100                   1028        0  0
UN    688  200 informix     9/ 8 17:16:59    26  0   100                   1054        0  0
FC    688  200 informix     9/ 8 17:16:59    29  0       isfile             1080        0  0

TY Trx ID User User Name    Date/Time     Lngth FD Recno Filename      Location Prev Loc Pr Ln
-- ------ ---- ------------ ------------- ----- -- ----- ------------ -------- -------- -- --
CW    688  200 informix     9/ 8 17:16:59    20                        1109      953 20
RE    688  200 informix     9/ 8 17:16:59    39       isfile           1129        0  0
ER    688  200 informix     9/ 8 17:17:00    28       isfile2          1168        0  0

Program over.
```

Ausgabe bei dblog -v

```
DBLOG: Transaction Log File Display C-ISAM Version 4.00.U
Copyright (C) 1981-1989 Informix Software, Inc.
Software Serial Number RDS#N000000

TY Trx ID User User Name    Date/Time     Lngth FD Recno Filename
-- ------ ---- ------------ ------------- ----- -- ----- ------------
BU    688  200 informix     9/ 8 17:16:55    37  0       isfile
    Row Length = 10    Build Mode = 0x806
BW    688  200 informix     9/ 8 17:16:55    20
FO    688  200 informix     9/ 8 17:16:55    29  0       isfile
IN    688  200 informix     9/ 8 17:16:55    36  0     1 isfile
```

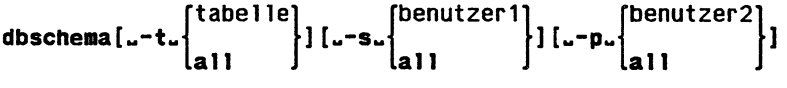
```
    Image:
             0001 0000 0000 0000 0000                          .....
FC     688  200 informix    9/ 8 17:16:55    29  0          isfile
CW     688  200 informix    9/ 8 17:16:55    20
BW     688  200 informix    9/ 8 17:16:56    20
FO     688  200 informix    9/ 8 17:16:56    29  0          isfile
IN     688  200 informix    9/ 8 17:16:56    36  0       2 isfile
    Image:
             0001 0000 0000 0000 0000                          .....
FC     688  200 informix    9/ 8 17:16:56    29  0          isfile
RW     688  200 informix    9/ 8 17:16:56    20
BW     688  200 informix    9/ 8 17:16:56    20
FO     688  200 informix    9/ 8 17:16:56    29  0          isfile
CL     688  200 informix    9/ 8 17:16:56    28  0
FC     688  200 informix    9/ 8 17:16:57    33  0          issaa00688
CW     688  200 informix    9/ 8 17:16:57    20
BW     688  200 informix    9/ 8 17:16:58    20
FO     688  200 informix    9/ 8 17:16:58    29  0          isfile
IN     688  200 informix    9/ 8 17:16:58    36  0       2 isfile
    Image:
             0001 0000 0000 0000 0000                          .....
IN     688  200 informix    9/ 8 17:16:58    36  0       3 isfile
    Image:
             0001 0000 0000 0000 0000                          .....

TY Trx ID User User Name     Date/Time     Lngth FD Recno Filename

UP     688  200 informix    9/ 8 17:16:58    48  0       1 isfile
    Pre-Image:
             0001 0000 0000 0000 0000                          .....
    Post-Image:
             0002 0000 0000 0000 0000                          .....
DE     688  200 informix    9/ 8 17:16:58    36  0       2 isfile
    Image:
             0001 0000 0000 0000 0000                          .....
FC     688  200 informix    9/ 8 17:16:58    29  0          isfile
CW     688  200 informix    9/ 8 17:16:58    20
BW     688  200 informix    9/ 8 17:16:58    20
FO     688  200 informix    9/ 8 17:16:58    29  0          isfile
CI     688  200 informix    9/ 8 17:16:58    34  0
    key.k_nparts = 1    key.k_flags = 1    key.k_len = 2
    key.k_part:  kp_start  kp_leng  kp_type

                        0         2         1
FC     688  200 informix    9/ 8 17:16:58    29  0          isfile
CW     688  200 informix    9/ 8 17:16:58    20
BW     688  200 informix    9/ 8 17:16:58    20
FO     688  200 informix    9/ 8 17:16:58    29  0          isfile
DI     688  200 informix    9/ 8 17:16:58    34  0
    key.k_nparts = 1    key.k_flags = 1    key.k_len = 2
    key.k_part:  kp_start  kp_leng  kp_type

                        0         2         1
FC     688  200 informix    9/ 8 17:16:58    29  0          isfile
CW     688  200 informix    9/ 8 17:16:58    20
BW     688  200 informix    9/ 8 17:16:59    20
FO     688  200 informix    9/ 8 17:16:59    29  0          isfile
UN     688  200 informix    9/ 8 17:16:59    26  0       1
SU     688  200 informix    9/ 8 17:16:59    26  0     100
UN     688  200 informix    9/ 8 17:16:59    26  0     100
FC     688  200 informix    9/ 8 17:16:59    29  0          isfile

TY Trx ID User User Name     Date/Time     Lngth FD Recno Filename

CW     688  200 informix    9/ 8 17:16:59    20
RE     688  200 informix    9/ 8 17:16:59    39          isfile
    New Filename = isfile2
ER     688  200 informix    9/ 8 17:17:00    28          isfile2
BU     699 4116 jac         9/ 8 17:18:27    37  0       isfile
      Row Length = 10    Build Mode = 0x806

Program over.
```

A.3.6 dbschema - Anweisungen für Datenbank-Aufbau erzeugen

dbschema erzeugt eine Anweisungsdatei, die CREATE TABLE-, CREATE INDEXund CREATE VIEW-Anweisungen enthält, die für den Aufbau einer Datenbank oder einer bestimmten Tabelle nötig sind. Außerdem kann *dbschema* alle CREATE SYNONYM- und GRANT-Anweisungen erzeugen, die für eine Datenbank, eine bestimmte Tabelle oder einen View definiert sind.

Standardmäßig erzeugt *dbschema* alle CREATE TABLE-, CREATE VIEW-, CREATE INDEX-, CREATE SYNONYM- und GRANT-Anweisungen für die gesamte Datenbank. Durch Auswahl der entsprechenden Operanden können Sie die Ausgabe beschränken
— auf eine bestimmte Tabelle oder View bzw.
— auf Synonyme und Zugriffsrechte für einen bestimmten Benutzer.

dbschema setzt in die erzeugten Anweisungen jeweils den aktuellen Benutzernamen des Eigentümers ein. Wenn Sie für den erneuten Datenbank-Aufbau andere Eigentümer einsetzen wollen, so müssen Sie die Anweisungsdatei vor Ablauf entsprechend ändern.



-t tabelle
Name der Tabelle oder des Views, für die CREATE TABLE (bzw. CREATE VIEW) und CREATE INDEX-Anweisungen erzeugt werden sollen.

all
bezeichnet alle Tabellen und Views in der Datenbank.

-s benutzer1
Name des Benutzers, für den die CREATE SYNONYM-Anweisungen erzeugt werden sollen. Ist der Schalter **-t** angegeben, so werden CREATE SYNONYM-Anweisungen nur für die dort angegebene Tabelle bzw. View erzeugt. Ohne Schalter **-t** werden alle CREATE SYNONYM-Anweisungen erzeugt, die der angegebene Benutzer in der Datenbank gegeben hat.

all
bezeichnet alle Benutzer.

-p benutzer2
Name des Benutzer, für den die GRANT-Anweisungen erzeugt werden sollen. Ist der Schalter **-t** angegeben, so werden GRANT-Anweisungen nur für die dort angegebene Tabelle bzw. View erzeugt. Ohne Schalter **-t** werden alle GRANT-Anweisungen erzeugt, die der angegebene Benutzer in der Datenbank gegeben hat. Die GRANT-Anweisung enthält auch den Namen des Benutzers, der die GRANT-Anweisung gegeben hat (GRANT ... AS benutzer).
Für den späteren Aufruf der Anweisungsdatei müssen folgende Voraussetzungen erfüllt sein, damit die enthaltenen GRANT-Anweisungen ablaufen können:
Der Benutzer (AS benutzer), anstelle dessen das Zugriffsrecht vergeben wird, muß das Datenbankzugriffsrecht CONNECT haben und, falls er nicht Eigentümer der Tabelle ist, das entsprechende Tabellenzugriffsrecht inklusive der Berechtigung WITH GRANT OPTION.

all
bezeichnet alle Benutzer.

-d datenbank
Name der Datenbank. Für INFORMIX-SE gilt: Ist die Datenbank nicht im aktuellen Dateiverzeichnis vorhanden, durchsucht *dbschema* die Dateiverzeichnisse, die in der Umgebungsvariablen DBPATH angegeben sind. *dbschema* wird abgewiesen, wenn die Datenbank nicht gefunden wird.

datei
Name der Datei, in die *dbschema* die erzeugten Anweisungen schreiben soll. Fehlt die Angabe, so werden die Anweisungen auf dem Bildschirm ausgegeben.

Hinweis

— Spalten vom Datentyp SERIAL erhalten in der CREATE TABLE-Anweisung den Anfangswert 1, unabhängig von ihrem ursprünglichen Anfangswert.

Beispiel

Für die Datenbank *versand* werden die Anweisungen erzeugt, die zum Aufbau der Tabelle *auftrag* und deren Indizes nötig sind:

```
dbschema -t auftrag -d versand
```

```
DBSCHEMA Schema Utility       INFORMIX-SQL Version 4.00.U
Copyright (C) Informix Software, Inc., 1984-1989
Software Serial Number RDS#N000000
{ TABLE "lomata".auftrag row size = 80 number of columns = 10
  index size = 24 }
create table "lomata".auftrag
  (
    auftrags_nr serial not null,
    auftragsdatum date,
    kunden_nr integer,
    lieferhinweis char(40),
    offen char(1),
    fremd_nr char(10),
    lieferdatum date,
    liefergewicht decimal(8,2),
    zustellgebuehr money(6,2),
    zahldatum date
  );
revoke all on "lomata".auftrag from "public";

create unique index "lomata".a_nr_ix on "lomata".auftrag (auftrags_nr);
create index "lomata".a_knr_ix on "lomata".auftrag (kunden_nr);
```

Literatur

Die mit * gekennzeichneten Titel sind nicht von Siemens herausgegeben.

[1] Programmieren in C
Kernighan/Ritchie
2. Ausgabe ANSI C mit C-Lösungsbuch

[2] Betriebssystem SINIX
INFORMIX
SQL
Sprachbeschreibung

[3] Betriebssystem SINIX
INFORMIX-SQL
Nachschlagen

[4] Betriebssystem SINIX
INFORMIX-SQL
Datenbanksystem
Kennenlernen

[5] Betriebssystem SINIX
C-ISAM
Indexsequentielle Zugriffsmethode

[6] Betriebssystem SINIX
INFORMIX-ONLINE
Datenbank-Server
Administrator-Handbuch

[7] Betriebssystem SINIX
Fehlermeldungen für
INFORMIX-Produkte

[8] Betriebssystem SINIX
INFORMIX-NET
Netzkomponente für INFORMIX-SE

[9] Betriebssystem SINIX
INFORMIX-STAR
Netzkomponente für INFORMIX-ONLINE

[10] Betriebssystem SINIX
INFORMIX-ESQL/COBOL
COBOL-Schnittstelle für das
Datenbanksystem INFORMIX

[11] Betriebssystem SINIX
INFORMIX-4GL
Nachschlagen

[12] Betriebssystem SINIX
INFORMIX-4GL
Kennenlernen

[13] Betriebssystem SINIX
Systemverwalter-Handbuch

[14] Betriebssystem SINIX
Kommandos

* C.J. Date
An Introduction to Database Systems
Addison-Wesley
1986

* C.J. Date
A Guide to The SQL Standard
Addison-Wesley
1990

Bestellen von Handbüchern

Die aufgeführten Handbücher finden Sie mit ihren Bestellnummern im *Druckschriftenverzeichnis Datentechnik*. Dort ist auch der Bestellvorgang erklärt. Neu erschienene Titel finden Sie in den *Druckschriften-Neuerscheinungen Datentechnik*.

Beide Veröffentlichungen erhalten Sie regelmäßig, wenn Sie in den entsprechenden Verteiler aufgenommen sind. Wenden Sie sich bitte hierfür an Ihre zuständige Siemens-Zweigniederlassung; außerhalb der Bundesrepublik Deutschland hilft Ihnen die zuständige Siemens-Vertretung.

Stichwörter

S

Herausgegeben von
Siemens Nixdorf Informationssysteme AG
Postfach 2160, W-4790 Paderborn
Postfach 830951, W-8000 München 83

Bestell-Nr. U3518-J-Z95-3
Printed in the Federal Republic of Germany
6160 AG 10901. (7700)

SIEMENS
NIXDORF

SINIX

INFORMIX-ESQL/C V4.0

C-Schnittstelle für das Datenbanksystem
INFORMIX

Beschreibung

Herausgegeben von
Siemens Nixdorf Informationssysteme AG
Postfach 2160, W-4790 Paderborn
Postfach 830951, W-8000 München 83

Bestell-Nr. U3518-J-Z95-3
Printed in the Federal Republic of Germany
6160 AG 10901. (7700)