SIEMENS
NIXDORF

SINIX

INFORMIX V4.0
SQL

Sprachbeschreibung

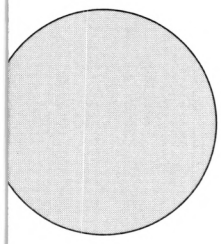
Sie haben
uns zu diesem Handbuch
etwas mitzuteilen?
Schicken Sie uns bitte
Ihre Anregungen
unter Angabe der Bestellnummer
dieses Handbuches.

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

Fax:
(089) 636-40443

email im EUnet:
man@sieqm2.uucp

DOPPEL



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

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

Fax:
(089) 636-40443

email im EUnet:
man@sieqm2.uucp

Einführung

Konzepte

INFORMIX (SINIX) SQL

Lexikalische Elemente und Namen

Sprachbeschreibung

Datentypen und Werte

Zusammengesetzte Sprachelemente

SQL-Anweisungen

Anhang

Ausgabe Oktober 1990 (INFORMIX V 4.0)

02377

...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 2/91 636-48999

Siemens Nixdorf Training Center
Postfach 830951, W-8000 München 83

Herausgegeben von
Siemens Nixdorf Informationssysteme AG

Vorwort

Dieses Handbuch ist eine vollständige Beschreibung der Datenbanksprache INFORMIX-SQL für alle INFORMIX-Produkte, die eine SQL-Benutzerschnittstelle zur Verfügung stellen.

Die Unterschiede zwischen den einzelnen Produkten, soweit sie SQL betreffen, sind herausgestellt. Außerdem sind die Abweichungen und Erweiterungen gegenüber dem ANSI-Standard beschrieben.

An wen richtet sich das Handbuch?

Das Handbuch richtet sich an alle Benutzer von INFORMIX-Datenbanksystemen, die mit SQL arbeiten.

Welche Vorkenntnisse benötigen Sie?

Für SQL benötigen Sie grundlegende Kenntnisse über relationale Datenbanken. Außerdem müssen Sie mit der Bedienung der INFORMIX-Produkte vertraut sein.

Wie ist das Benutzerhandbuch aufgebaut?

Dieses Handbuch beschreibt SQL im Stil einer Programmiersprachenbeschreibung. Es besteht aus 6 Kapiteln und 5 Anhängen mit folgendem Inhalt:

Kapitel 1	gibt einen Überblick über INFORMIX im Hinblick auf SQL. Es erklärt, was SQL ist und bei welchen Produkten SQL eingesetzt wird. Außerdem sind die Darstellungsmittel des Handbuchs zusammengestellt.
Kapitel 2	ist eine Zusammenstellung der Datenbank-Konzepte, auf die sich die SQL-Anweisungen beziehen. Die Datenbankobjekte sind erklärt und es wird beschrieben, welche Operationen jeweils mit SQL-Anweisungen ausgeführt werden können.
Kapitel 3	beschreibt die lexikalischen Einheiten der Sprache SQL sowie Syntax und Verwendung von Namen in SQL-Anweisungen.
Kapitel 4	ist ein Nachschlageteil über alle INFORMIX-Datentypen. Sie erfahren die Einzelheiten für die Datentypdefinition und für die Verwendung der Werte eines Datentyps.

Kapitel 5	beschreibt in einheitlichem Stil die zusammengesetzten Sprachelemente von SQL: Funktionen, Unterabfragen. Ausdrücke, Prädikate und Bedingungen. Die Sprachelemente sind in der Reihenfolge beschrieben, in der sie aufeinander aufbauen.
Kapitel 6	ist ein alphabetischer Nachschlageteil über die INFORMIX-SQL-Anweisungen. Alle Anweisungen sind in einem einheitlichen Format ausführlich beschrieben. Außerdem finden Sie dort Tabellen, in denen die Anweisungen nach unterschiedlichen Gesichtspunkten zusammengestellt sind.
Anhang 1	beschreibt die Beispieldatenbank, auf die sich die Beispiele in diesem Handbuch beziehen.
Anhang 2	ist eine Zusammenstellung der Systemtabellen. Aufbau und Inhalt der Tabellen sind erklärt.
Anhang 3	beschreibt die Umgebungsvariablen, die für INFORMIX von Bedeutung sind.
Anhang 4	beschreibt ein Verfahren zum Abschätzen der Tabellengrößen bei INFORMIX-ONLINE.
Anhang 5	enthält die Liste aller resevierten Wörter.

Eine Bitte an Sie

Schreiben Sie uns, wie Sie mit dem Handbuch zurechtkommen. Wenn Sie zufrieden sind, dann 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

Inhalt

	Vorwort	
1	**Einführung**	**1-1**
1.1	INFORMIX im Überblick	1-2
1.1.1	Frontend	1-3
1.1.2	Backend	1-4
	INFORMIX-SE	1-4
	INFORMIX-ONLINE	1-4
	INFORMIX-Netzprodukte	1-6
1.1.3	Datenbank	1-6
1.2	Was ist INFORMIX-SQL?	1-7
1.2.1	Kompatibilität	1-8
1.3	INFORMIX-Komponenten und SQL	1-9
1.4	Darstellungsmittel	1-10
2	**Konzepte**	**2-1**
2.1	Datenbank	2-2
2.1.1	Datenbankobjekte	2-2
2.1.2	Interne Organisation	2-2
2.1.3	Datenbank definieren	2-6
2.1.4	Datenbank löschen	2-6
2.1.5	Datenbank öffnen	2-6
2.1.6	Aktuelle Datenbank bearbeiten	2-6
2.1.7	Datenbank wechseln	2-7
2.1.8	Datenbank schließen	2-7
2.1.9	ANSI-Datenbank	2-7
	ANSI-Datenbank erstellen	2-7
	Von Nicht-ANSI auf ANSI wechseln	2-8
2.1.10	Entfernte Datenbank	2-8
2.2	Tabelle	2-9
2.2.1	Basistabelle	2-9
	Basistabelle definieren	2-10
	Basistabelle löschen	2-10
	Basistabelle bearbeiten	2-10
	Systemtabellen	2-11
2.2.2	View	2-12
	View definieren	2-12
	View löschen	2-13
	View bearbeiten	2-13
	Vorteile von Views	2-15
2.2.3	Temporäre Tabelle	2-16
	Temporäre Tabelle definieren	2-16
	Temporäre Tabelle löschen	2-16

	Temporäre Tabelle bearbeiten	2-16
2.3	Synonym	2-18
2.3.1	Synonym definieren	2-18
2.3.2	Synonym löschen	2-19
2.3.3	Synonym verwenden	2-19
2.4	Spalten	2-20
2.4.1	Spalte definieren	2-20
2.5	Werte und Datentypen	2-21
2.6	Index	2-23
2.6.1	Index definieren	2-24
2.6.2	Index löschen	2-24
2.6.3	Index ändern	2-24
2.7	Constraint	2-25
2.7.1	Constraint definieren	2-25
2.7.2	Constraint löschen	2-25
2.8	Benutzer	2-26
2.8.1	Wem werden Datenbankzugriffsrechte zugeteilt?	2-27
2.8.2	ANSI-Datenbank	2-27
2.9	Zugriffsrechte	2-28
2.9.1	Welche Zugriffsrechte gibt es?	2-28
2.9.2	Voreingestellte Zugriffsrechte	2-28
2.9.3	Eigentümer	2-29
2.9.4	Zugriffsrechte vergeben und entziehen	2-29
2.9.5	Überprüfung der Zugriffsrechte	2-30
2.9.6	Zusammenwirken von Datenbank- und Tabellenzugriffsrechten	2-30
2.10	Transaktionssicherung	2-33
2.10.1	Was ist eine Transaktion?	2-33
2.10.2	Transaktionsprotokoll	2-35
2.10.3	Ausführung von Transaktionen	2-36
2.10.4	Möglichkeiten der Transaktionssicherung	2-37
2.10.5	Transaktionssicherung einstellen	2-39
2.11	Sperren	2-40
2.11.1	Sperrgranulat	2-40
2.11.2	Einzelsperren	2-41
2.11.3	Wie werden Sperren gesetzt?	2-44
2.11.4	Anzahl der Sperren	2-45
2.11.5	Wie lange werden Sperren gehalten?	2-46
2.12	Datenschutz	2-47
2.12.1	Datensicherheit	2-47
2.12.2	Datenintegrität	2-47
2.12.3	Datensicherung	2-48
	INFORMIX-SE-Sicherungsverfahren	2-48
	INFORMIX-ONLINE-Sicherungsverfahren	2-49
2.13	Programmeinbettung	2-50

2.13.1 Hostvariable 2-50
2.13.2 Indikatorvariable 2-52
2.13.3 Satzzeiger 2-53
2.13.4 Dynamisch formulierte Anweisung 2-58
2.13.5 Erfolgskontrolle 2-60

3 Lexikalische Elemente und Namen 3-1
3.1 Lexikalische Einheiten 3-2
3.2 Namen 3-4
3.2.1 SQL-Syntax für einfache Namen 3-5
3.2.2 Name definieren 3-6
3.2.3 Name ändern 3-7
3.2.4 Eindeutigkeit von Namen 3-8
3.2.5 Qualifizierung von Namen 3-9

4 Datentypen und Werte 4-1
4.1 Überblick 4-3
4.1.1 Einteilung der Datentypen 4-3
4.1.2 Datentyp und Wertebereich 4-4
4.2 Datentyp definieren 4-4
4.2.1 Übersicht über SQL-Syntax für Datentypen 4-5
4.2.2 Alphanumerische Datentypen 4-6
CHARACTER - Zeichenketten fester Länge 4-6
VARCHAR - Zeichenketten variabler Länge 4-7
4.2.3 Numerische Datentypen 4-10
SMALLINT - Kleine Ganzzahlen 4-10
INTEGER - Ganzzahlen 4-11
SERIAL - Nummern 4-12
DECIMAL, NUMERIC - Festpunktzahlen 4-14
MONEY - Festpunktzahlen als Geldbeträge 4-16
DECIMAL, NUMERIC - Gleitpunktzahlen 4-17
SMALLFLOAT, REAL- Gleitpunktzahlen mit einfacher Genauigkeit 4-18
FLOAT, DOUBLE PRECISION - Doppelte Genauigkeit . . . 4-19
4.2.4 Zeit-Datentypen 4-20
DATE - Datum 4-20
DATETIME - Zeitpunkte 4-21
INTERVAL - Zeitspannen 4-24
4.2.5 BLOB-Datentypen 4-28
TEXT - Texte 4-28
BYTE - Binäre BLOB-Daten 4-32
4.3 Werte verwenden 4-36
4.3.1 NULL-Wert 4-37
4.3.2 Alphanumerische Werte 4-39
4.3.3 Numerische Werte 4-42
4.3.4 Zeitwerte 4-44

	Datum	4-45
	Zeitpunkt	4-49
	Zeitspanne	4-53
5	**Zusammengesetzte Sprachelemente**	**5-1**
5.1	Funktionen	5-3
	Inhaltliche Zusammenstellung der SQL-Funktionen	5-3
	AVG() - Arithmetisches Mittel	5-5
	COUNT(*) - Elemente zählen	5-7
	COUNT() - Verschiedene Elemente zählen	5-8
	CURRENT - Aktuelles Datum	5-10
	DATE() - Datum erzeugen	5-12
	DAY() - Tag bestimmen	5-14
	EXTEND() - Zeitpunkt anpassen	5-16
	LENGTH() - Länge einer Zeichenkette bestimmen	5-18
	MAX() - Maximum bestimmen	5-19
	MDY() - Datum erzeugen	5-21
	MIN() - Minimum bestimmen	5-22
	MONTH() - Monat bestimmen	5-24
	SITENAME- Aktuelles Informixsystem	5-26
	SUM() - Summe berechnen	5-27
	TODAY - Aktuelles Datum	5-29
	USER - Aktueller Benutzer	5-30
	WEEKDAY() - Wochentag bestimmen	5-31
	YEAR() - Jahr bestimmen	5-33
5.2	Unterabfragen	5-35
	Korrelierte Unterabfragen	5-36
5.3	Ausdrücke	5-38
5.4	Prädikate	5-43
	Vergleich von zwei Werten	5-45
	Vergleichbare Werte	5-46
	Bereichsabfrage	5-50
	Elementabfrage	5-53
	Mustervergleich	5-55
	Platzhalter	5-55
	Entwertungszeichen	5-55
	Vergleich auf NULL	5-59
	Existenzabfrage	5-60
5.5	Bedingung	5-61
	Prioritäten	5-63
5.6	Join	5-65
	Einfacher Join	5-65
	Zusammengesetzter Join	5-65
	Join-Typen	5-65
	Normaler Join	5-66
	Zusammengesetzter normaler Join	5-67

	Outer-Join	5-69
	Zusammengesetzter Outer-Join	5-70
6	**SQL-Anweisungen**	**6-1**
6.1	Inhaltliche Zusammenstellung	6-2
6.2	Zusammenstellung nach Verwendungsmöglichkeit	6-5
6.3	Zusammenstellung nach Backend	6-7
6.4	Erweiterungen des ANSI-Standards	6-9
6.5	Beschreibungsformat und Parameter	6-12
6.5.1	Beschreibungsformat	6-12
6.5.2	Parameter	6-16
6.6	Alphabetischer Nachschlageteil	6-17
	ALTER INDEX - Index ändern	6-18
	ALTER TABLE - Tabellenstruktur ändern	6-21
	BEGIN WORK - Transaktion starten	6-31
	CLOSE - Satzzeiger schließen	6-33
	CLOSE DATABASE - Datenbank schließen	6-36
	COMMIT WORK - Transaktion beenden	6-38
	CREATE AUDIT - Audit-Protokoll erzeugen	6-40
	CREATE DATABASE - INFORMIX-SE-Datenbank erzeugen	6-44
	CREATE DATABASE - INFORMIX-ONLINE-Datenbank erzeugen	6-48
	CREATE INDEX - Index erzeugen	6-51
	CREATE SCHEMA - Datenbankschema erzeugen	6-56
	CREATE SYNONYM - Synonym definieren	6-59
	CREATE TABLE - Tabelle erzeugen	6-63
	CREATE VIEW - View erzeugen	6-72
	DATABASE - Datenbank eröffnen	6-77
	DECLARE - Satzzeiger vereinbaren	6-80
	DELETE - Sätze löschen	6-88
	DESCRIBE - Dynamische Anweisung analysieren	6-93
	DROP AUDIT - Audit-Protokoll löschen	6-96
	DROP DATABASE - Datenbank löschen	6-97
	DROP INDEX - Index löschen	6-99
	DROP SYNONYM - Synonym löschen	6-101
	DROP TABLE - Tabelle löschen	6-103
	DROP VIEW - View löschen	6-105
	EXECUTE - Dynamische Anweisung ausführen	6-107
	EXECUTE IMMEDIATE - Dynamische Anweisung sofort ausführen	6-109
	FETCH - Satzzeiger positionieren	6-111
	FLUSH - Pufferinhalt in Datenbank einfügen	6-116
	FREE - Betriebsmittel freigeben	6-119
	GRANT - Zugriffsrechte vergeben	6-123
	INFO - Tabellen-Informationen ausgeben	6-131
	INSERT- Sätze in Tabelle einfügen	6-135

LOAD - Sätze aus Datei in Tabelle einfügen 6-140
LOCK TABLE - Tabelle sperren 6-146
OPEN - Satzzeiger öffnen 6-149
OUTPUT - Sätze aus Tabelle ausgeben 6-154
PREPARE - Dynamische Anweisungen vorbereiten 6-156
PUT - Satz in Einfügepuffer schreiben 6-161
RECOVER TABLE - Tabelle aktualisieren 6-165
RENAME COLUMN - Spaltennamen ändern 6-167
RENAME TABLE - Tabellennamen ändern 6-169
REPAIR TABLE - Index für Tabelle wiederherstellen . . . 6-171
REVOKE - Zugriffsrechte entziehen 6-172
ROLLBACK WORK - Transaktion zurücksetzen 6-176
ROLLFORWARD DATABASE - Datenbanksicherung aktualisieren 6-179
SELECT - Daten abfragen 6-181
SELECT/Spaltenauswahl - Ergebnisspalten auswählen . . . 6-185
SELECT/FROM - Tabellen angeben 6-188
SELECT/WHERE - Ergebnissätze auswählen 6-193
SELECT/GROUP BY - Ergebnissätze gruppieren 6-197
SELECT/HAVING - Gruppen auswählen 6-199
SELECT/ORDER BY - Ergebnistabelle sortieren 6-201
SELECT/INTO TEMP - Ergebnistabelle benennen 6-204
SELECT/UNION - SELECT-Anweisungen verbinden . . . 6-206
SELECT (eingebettet) 6-207
SELECT/INTO - Satz in Variable ausgeben 6-208
SET EXPLAIN - Suchstrategie ausgeben 6-210
SET ISOLATION - Isolationsstufen wählen 6-213
SET LOCK MODE - Warte-Modus definieren 6-214
SET LOG - Protokollierungsmodus ändern 6-217
START DATABASE - Transaktionssicherung einschalten . . 6-219
UNLOAD - Sätze aus Tabelle in Ausgabedatei schreiben . . 6-222
UNLOCK TABLE - Tabellensperre aufheben 6-227
UPDATE - Spaltenwerte ändern 6-228
UPDATE STATISTICS - Satzanzahl aktualisieren 6-234

A Anhang A-1
A.1 Beispieldatenbank versand A-1
Struktur der Tabellen A-1
Tabelle kunde A-1
Tabelle auftrag A-2
Tabelle posten A-2
Tabelle artikel A-3
Tabelle hersteller A-4
Tabelle bundesland A-4
Verbindung der Tabellen durch Join-Spalten A-5
Join-Spalten in den Tabellen kunde und auftrag A-6

	Join-Spalten in den Tabellen auftrag und posten	A-7
	Join-Spalten in den Tabellen posten und artikel	A-8
	Join-Spalten in den Tabellen artikel und hersteller	A-9
	Join-Spalten in den Tabellen kunde und bundesland	A-10
	Daten in der Datenbank versand	A-11
	Tabelle kunde	A-11
	Tabelle auftrag	A-12
	Tabelle posten	A-13
	Tabelle artikel	A-14
	Tabelle hersteller	A-14
	Tabelle bundesland	A-14
A.2	Systemtabellen	A-15
	Systemtabelle systables	A-16
	Systemtabelle syscolumns	A-17
	Systemtabelle sysindexes	A-18
	Systemtabelle systabauth	A-19
	Systemtabelle syscolauth	A-19
	Systemtabelle sysdepend	A-20
	Systemtabelle syssynonyms	A-20
	Systemtabelle sysusers	A-20
	Systemtabelle sysviews	A-21
	Systemtabelle sysconstraints	A-21
	Systemtabelle syssyntable	A-21
A.3	Umgebungsvariable	A-23
	Umgebungsvariable DBANSIWARN	A-24
	Umgebungsvariable DBDATE	A-25
	Umgebungsvariable DBDELIMITER	A-26
	Umgebungsvariable DBMONEY	A-26
	Umgebungsvariable DBPATH	A-27
	Umgebungsvariable DBTEMP	A-27
	Umgebungsvariable INFORMIXDIR	A-28
	Umgebungsvariable SQLEXEC	A-28
	Umgebungsvariable TBCONIFG	A-28
A.4	Tabellengröße schätzen	A-29
A.5	Reservierte Wörter	A-35
	Literatur	A-41

Stichwörter

1 Einführung

1.1 INFORMIX im Überblick
1.2 Was ist INFORMIX-SQL?
1.3 Wo wird SQL verwendet?
1.4 Darstellungsmittel

Dieses Kapitel beschreibt, welche Komponenten INFORMIX bereitstellt, wo SQL benutzt wird und welche Darstellungsmittel in diesem Handbuch verwendet werden.

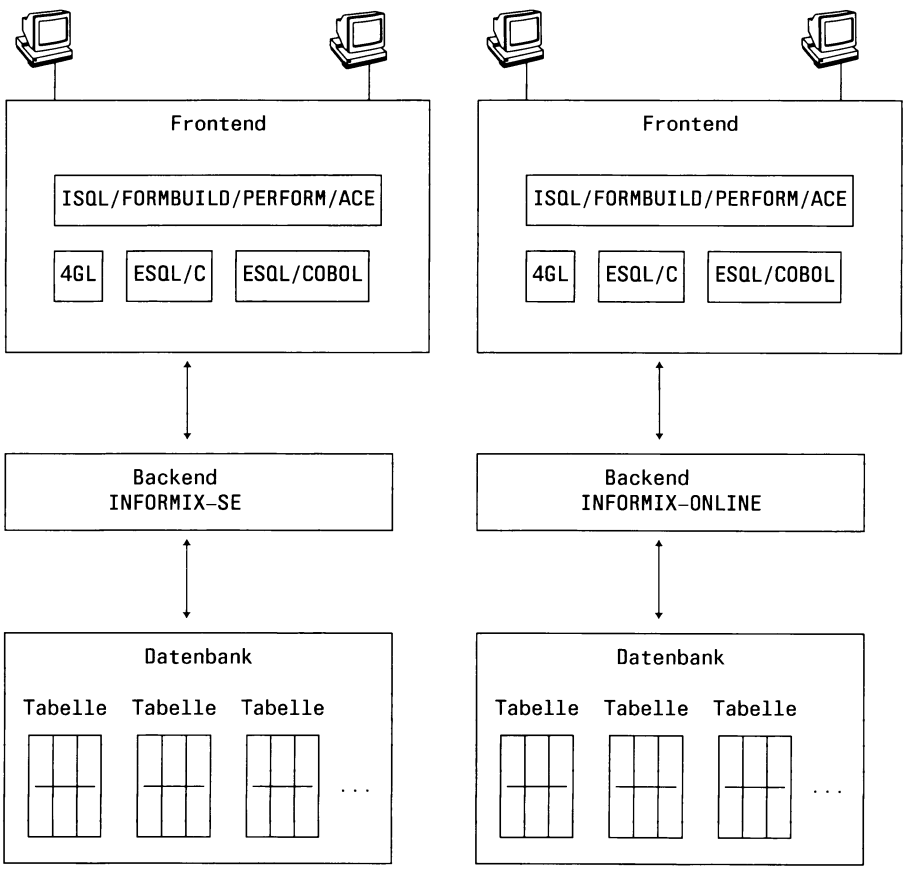
1.1 INFORMIX im Überblick

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.

In diesem Handbuch sind die Frontends berücksichtigt, die eine SQL-Schnittstelle bieten. In Abschnitt 1.3 sind diese Komponenten zusammengestellt.

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 Speicherplatzes, 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. Einzelheiten hierzu finden Sie in Kapitel 2, Abschnitt 2.1 *Datenbank*.

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. Einzelheiten hierzu finden Sie in Kapitel 2, Abschnitt 2.11.2 *Einzelsperren*.

- Bessere Möglichkeiten für die Datensicherung:
  Einen Überblick über die einzelnen Möglichkeiten finden Sie in Kapitel 2, Abschnitt 2.12 *Datenschutz*. Die ausführliche Beschreibung der Datensicherungsmethoden ist im ONLINE-Handbuch [10] beschrieben.
- Datentyp VARCHAR:
  Er ermöglicht es, Spalten zu definieren, die Zeichenketten unterschiedlicher Länge enthalten können. Die ausführliche Beschreibung finden Sie in Kapitel 4, Abschnitt 4.2.2 *Alphanumerische Datentypen*.
- 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 in Kapitel 4, Abschnitt 4.2.5 *BLOB-Datentypen*.
- Zugriff auf externe Tabellen:
  Sie haben die Möglichkeit, auf Tabellen einer Datenbank zuzugreifen, die nicht die aktuelle Datenbank ist. Einzelheiten hierzu finden Sie in Kapitel 2, Abschnitt 2.1 *Datenbank* und Abschnitt 2.2 *Tabelle*.
- 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.

Kennzeichnung der Unterschiede

In diesem Handbuch sind an allen Stellen die Unterschiede zwischen den beiden Backends gekennzeichnet und, falls erforderlich, in einem gesonderten Abschnitt beschrieben.

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 in Kapitel 2 beschrieben.

1.2 Was ist INFORMIX-SQL?

INFORMIX-SQL ist die Datenbanksprache von INFORMIX, in der die Anweisungen zum Bearbeiten der Datenbank formuliert werden.

Gemäß ihrer Funktion sind die SQL-Anweisungen in folgende Gruppen unterteilt:

- Anweisungen für die Datendefinition
- Anweisungen für die Datenmanipulation
  - Datenabfrage
  - Datenänderung
- Anweisungen für Vergabe und Entzug von Zugriffsrechten
- Anweisungen zur Gewährleistung der Datenintegrität
- Anweisungen für die Datensicherung
- Anweisungen für die Programmeinbettung
  - Satzzeiger
  - Dynamische Bearbeitung von Anweisungen

Die Anweisungen sind vollständig und systematisch im Nachschlageteil in Kapitel 6 beschrieben. Die SQL-Sprachelemente, aus denen die Anweisungen bestehen, sind in den Kapitel 3, 4 und 5 beschrieben und zwar in der Reihenfolge, in der sie aufeinander aufbauen.

1.2.1 Kompatibilität

SQL ist eine genormte Sprache. Das Normungsgremium ANSI (American National Standard Institute) hat Standards für SQL (Structured Query Language) definiert.

INFORMIX-SQL Version 4.0 erfüllt den Standard ANSI LEVEL I (X3.135-1986). Außerdem wird der ANSI LEVEL II Standard erfüllt, bis auf folgende Ausnahmen:

- INFORMIX-SE
  - serialisierbare Transaktionen
  - Modul-Language
- INFORMIX-ONLINE

  Modul-Language

Darüberhinaus bietet INFORMIX eine Anzahl von Erweiterungen zum ANSI-Standard. Diese sind in Kapitel 6, Abschnitt 6.4 *Erweiterungen des ANSI-Standards* tabellarisch zusammengestellt. Im Nachschlageteil in Kapitel 6 finden Sie bei jeder Anweisung, die eine Erweiterung ist, oder bei der es Abweichungen gegenüber dem Standard gibt, den Abschnitt mit der Überschrift *ANSI-Standard*. Dort sind diese Unterschiede beschrieben.

1.3 INFORMIX-Komponenten und SQL

In diesem Abschnitt sind die INFORMIX-Komponenten zusammengestellt, die eine SQL-Schnittstelle zur Verfügung Stellen.

Für den interaktiven Betrieb gibt es die Komponenten:

- ISQL (Dialog-Abfragesprache)
- ACE (Listengenerator)

Für die Einbettung der SQL-Anweisungen in ein Programm gibt es folgende Komponenten:

- INFORMIX-4GL (**4**th **G**eneration **L**angunge)
- INFORMIX-ESQL/C (**e**mbedded SQL für die Programmiersprache C)
- INFORMIX-ESQL/COBOL (**e**mbedded SQL für die Programmiersprache COBOL)

Bezüglich SQL gibt es einige Unterschiede zwischen dem interaktiven und dem eingetteten Gebrauch. An einigen Stellen gibt es außerdem noch Unterschiede zwischen den einzelnen Programmiersprachen.

Die Unterschiede sind in diesem Handbuch beschrieben. Insbesondere finden Sie im Kapitel 2, Abschnitt 2.13 *Programmeinbettung* die speziellen Konzepte für die Programmeinbettung, soweit sie SQL betreffen. Im Nachschlageteil in Kapitel 6 sind für jede SQL-Anweisung, bei der es Unterschiede gibt, die Unterschiede in der Syntax gekennzeichnet und in einem gesonderten Abschnitt beschrieben.

1.4 Darstellungsmittel

In diesem Handbuch, insbesondere in den Kapiteln 3, 4, 5 und 6 verwenden wir folgende Darstellungsmittel:

`Raster`	Syntaxdefinition
GROSS	SQL-Schlüsselwörter in Syntaxdefinitionen und im Fließtext
fett	In Syntaxdefinitionen konstante Teile, die keine SQL-Schlüsselwörter sind. Hervorhebungen im Fließtext.
kursiv	Parameter in Syntaxdefinitionen und im Fließtext. Außerdem konstante Teile im Fließtext, die keine SQL-Schlüsselwörter sind.
`Schreibma-schinenschrift`	Programmtext in Beispielen
[ ]	Optionale Angaben Die eckigen Klammern werden nicht angegeben.
	Alternative Angaben in Syntaxdefinitionen Die geschweiften Klammern werden nicht angegeben.
{ }	Zusammenfassung von Klauseln in Syntaxdefinitionen, die gemeinsam wiederholt werden können. Die geschweiften Klammern werden nicht angegeben.

...	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.
DISTINCT UNIQUE	Bei INFORMIX-SQL ist das Schlüsselwort UNIQUE ein Synonym für DISTINCT. In diesem Handbuch wird immer DISTINCT angegeben. Sie können an allen Stellen DISTINCT durch UNIQUE ersetzen.

Verweise

Verweise werden wie folgt angegeben.

- Verweis auf einen Abschnitt in demselben Kapitel:
  siehe Abschnitt *Abschnitts-Titel*
- Verweis auf einen Abschnitt in einem anderen Kapitel:
  siehe Kapitel Kapitelnummer, Abschnitt *Abschnitts-Titel*
- Verweis auf ein anderes Handbuch:
  siehe Handbuchtitel [Literaturlistennummer]

2 Konzepte

2.1 Datenbank
2.2 Tabelle
2.3 Synonym
2.4 Spalte
2.5 Werte und Datentypen
2.6 Index
2.7 Constraint
2.8 Benutzer
2.9 Zugriffsrechte
2.10 Transaktionssicherung
2.11 Sperren
2.12 Datenschutz
2.13 Programmeinbettung

Dieses Kapitel ist eine Zusammenstellung der Konzepte, auf die sich die SQL-Anweisungen beziehen.

Die Datenbankobjekte sind erklärt und es wird beschrieben, welche Operationen jeweils mit SQL-Anweisungen ausgeführt werden können. Außerdem wird auf die Unterschiede zwischen den Backends (INFORMIX-SE, INFORMIX-ONLINE), zwischen interaktivem und eingebettetem Gebrauch sowie auf Besonderheiten bei ANSI-Datenbanken hingewiesen.

Die Einzelheiten erfahren Sie in Kapitel 6 bei der ausführlichen Beschreibung der jeweiligen SQL-Anweisungen. Die Syntax und Beschreibung der SQL-Sprachelemente finden Sie in den Kapiteln 3, 4 und 5.

Die Beschreibung der INFORMIX-ONLINE-spezifischen Teile für den INFORMIX-ONLINE-Verwalter finden Sie im ONLINE-Handbuch [10]. Die INFORMIX-Netzprodukte sind in den INFORMIX-Netzhandbüchern [7, 8] beschrieben. Näheres über die Konzepte von relationalen Datenbanksystemen finden Sie in der Standardliteratur (siehe z.B. [13, 14]).

2.1 Datenbank

Eine Datenbank bildet eine Verwaltungseinheit eines Datenbanksystems. In einem relationalen Datenbanksystem besteht sie aus einer Reihe von benutzerdefinierten Tabellen und Systemtabellen. In den Systemtabellen werden alle Definitionen eingetragen, die zur Verwaltung der Datenbank erforderlich sind.

Jede Tabelle gehört zu genau einer Datenbank. Die organisatorische Zusammengehörigkeit der Tabellen einer Datenbank regelt das Datenbankverwaltungssystem. Der Benutzer ist für den Aufbau und den Inhalt der benutzerdefinierten Tabellen verantwortlich.

2.1.1 Datenbankobjekte

Datenbankobjekt ist der Oberbegriff für alle Elemente der Datenbank, die in der Datenbank erzeugt und benannt werden können. Zu den Datenbankobjekten gehören:

- Tabellen
  - Basistabellen
  - Views
  - Synonyme
  - temporäre Tabellen
- Spalten
- Indizes
- Constraints

Die einzelnen Datenbankobjekte sind in den folgenden Abschitten beschrieben.

2.1.2 Interne Organisation

Die interne Abspeicherung einer Datenbank ist bei INFORMIX-SE und INFORMIX-ONLINE unterschiedlich.

INFORMIX-SE

Die Datenbank wird als Dateiverzeichnis im SINIX-Dateisystem abgelegt und mit den Mitteln der SINIX-Dateiverwaltung verwaltet:

- Der Dateiname des Datenbankverzeichnisses hat das Suffix *.dbs*.
- Für jede Tabelle werden zwei Dateien angelegt, die das Suffix *.dat* für Daten bzw. *.idx* für Index erhalten.
- Benutzer, die auf die Tabelle zugreifen wollen, müssen Ausführungsberechtigung (x-Bit) für alle Dateiverzeichnisse im Pfad und Leseberechtigung (r-Bit) für die Tabellen- und Index-Dateien besitzen.
- Benutzer, die die Tabelle verändern wollen, müssen Schreibberechtigung (w-Bit) für die Tabellen- und Index-Dateien besitzen.

Die Organisation über das Dateisystem hat den Vorteil, daß der Benutzer sich nicht um den Speicherplatzbedarf und die Organisation der Daten seiner Datenbank kümmern muß. Das regelt die SINIX-Dateiverwaltung automatisch, solange die Plattenkapazität ausreicht.

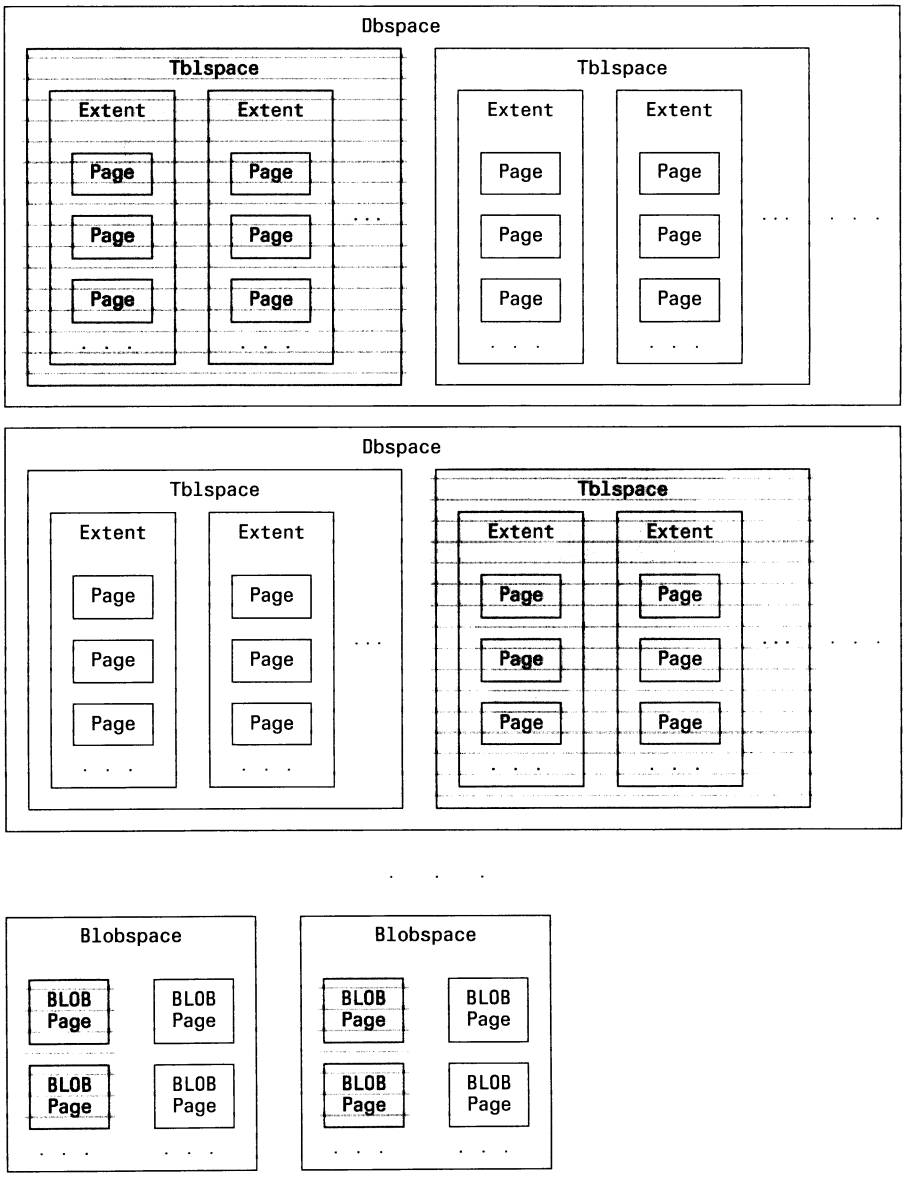
Nachteile der Organisation über das Dateisystem sind:

- Der Benutzer muß die Vorschriften der SINIX-Dateiverwaltung berücksichtigen. Dazu gehören:
  - Auswirkungen beim Löschen von Dateien und eines Dateiverzeichnisses
  - Auswirkungen der Dateizugriffsrechte
  - systemspezifische Grenzwerte, zum Beispiel für offene Dateien
  - Sicherungsmethoden.
- Die Sicherheit des Datenbanksystems ist eingeschränkt.
  - Benutzer können ohne Kontrolle des Datenbankverwaltungssystems die Dateien eines Datenbankverzeichnisses manipulieren.
  - Datenbank, Transaktions- und Audit-Protokolle werden nur über den SINIX-Ein/Ausgabe-Mechanismus abgespeichert.
  - Die Sicherung der Datenbank, Transaktions- und Audit-Protokolle liegt in voller Verantwortung des Benutzers.
- Die Performance ist schlechter, da die SINIX-Dateiverwaltung nicht auf die besonderen Anforderungen eines Datenbankverwaltungssystems zugeschnitten ist.

INFORMIX-ONLINE

Das INFORMIX-ONLINE-Datenbankverwaltungssystem hat eine eigene Plattenverwaltung für seine Datenbanken, unabhängig vom SINIX-Dateisystem.

Dazu wird aus Sicht der INFORMIX-ONLINE-Datenbankverwaltung der zur Verfügung stehende Platz in verschiedene Einheiten aufgeteilt:



Die genaue Beschreibung dieser Darstellung finden Sie im ONLINE-Handbuch [10]. Im folgenden sind nur die wichtigsten Begriffe zusammengestellt.

Eine Tabelle wird in einem **Tblspace** abgespeichert. Ein Tblspace setzt sich aus mehreren Extents zusammen.

Ein **Extent** besteht aus einer Anzahl von **Pages**. Die Extent-Größe wird bei der Erstellung der Tabelle festgelegt. Die Page-Größe ist im System fest vorgegeben.

Mehrere Tblspaces zusammen bilden einen **Dbspace**. Ein Dbspace besteht aus einem oder mehreren physischen **Chunks**, wobei ein Chunk ein Raw Device, Teil eines Raw Device oder eine Datei sein kann.

INFORMIX-ONLINE unterstützt BLOB-Daten (**B**inary **L**arge **O**bjects, siehe Abschnitt 2.5 und Kapitel 4). Für BLOB-Daten kann zusätzlich ein **Blobspace** angelegt werden, der aus mehreren **BLOB-Pages** besteht. Im Unterschied zu den Pages für Dbspaces legt der INFORMIX-ONLINE-Verwalter die Größe der BLOB-Pages fest.

Für eine Datenbank gibt es keine eigene logische Einheit. Die Tabellen und BLOB-Daten einer Datenbank können in verschiedenen Dbspaces und Blobspaces organisiert sein. Die im Bild gerasterten Teile können zum Beispiel zu einer Datenbank gehören.

Die Zuweisung der Dbspaces und Blobspaces übernimmt der Informix-ONLINE-Verwalter (siehe ONLINE-Handbuch [10]):

- Er muß die Größe der Datenbanken und ihrer Tabellen abschätzen, um genügend Speicherplatz bereitzustellen.
- Er muß die Größe der BLOB-Pages sinnvoll einstellen und Auswirkungen der Einteilung berücksichtigen (z.B. Anzahl der Sperren, Abspeicherung von variabel langen Daten).

Bei INFORMIX-ONLINE muß der Benutzer bei der Erstellung seiner Tabellen die Extent-Größe angeben. Verwendet er BLOB-Daten, muß er zusätzlich angeben, wo die BLOB-Daten abgelegt werden sollen.

Ansonsten muß er sich nicht um die interne Organisation der Datenbank kümmern.

2.1.3 Datenbank definieren

Gibt es die Datenbank noch nicht, müssen Sie die Datenbank mit CREATE DATABASE definieren. Bei der Definition wird die Datenbank geöffnet und zur aktuellen Datenbank gemacht.

INFORMIX-Netzprodukte:
Setzen Sie INFORMIX-NET oder INFORMIX-STAR ein, können Sie eine Datenbank in einem anderen Informixsystem erstellen.

2.1.4 Datenbank löschen

Wird eine Datenbank nicht mehr benötigt, kann sie mit DROP DATABASE gelöscht werden. Alle zugehörigen Tabellen werden automatisch mitgelöscht.

2.1.5 Datenbank öffnen

Für die Bearbeitung muß die Datenbank geöffnet sein. Eine Datenbank, die neu erstellt wird, wird automatisch bei der Definition mit CREATE DATABASE geöffnet.
Eine Datenbank, die bereits vorhanden ist, wird mit der Anweisung DATABASE geöffnet.

Bei INFORMIX kann immer nur eine Datenbank geöffnet sein. Sie heißt **aktuelle** Datenbank.

2.1.6 Aktuelle Datenbank bearbeiten

In der aktuellen Datenbank können Sie alle SQL-Anweisungen ausführen, für die Sie Berechtigung haben.

INFORMIX-ONLINE:
Sie können Daten aus Tabellen einer fremden Datenbank abfragen, die nicht die aktuelle Datenbank ist. Zur Unterscheidung von den Tabellen der aktuellen Datenbank werden diese Tabellen **externe** Tabellen genannt.

Die fremde Datenbank kann sein:

- Eine Datenbank in demselben ONLINE-System
- Nur mit INFORMIX-STAR
  - Eine Datenbank in einem anderen ONLINE-System auf demselben Rechner

- Eine Datenbank in einem ONLINE-System auf einem anderen Rechner

2.1.7 Datenbank wechseln

Mit DATABASE können Sie die Datenbank wechseln. Die aktuelle Datenbank wird geschlossen und die angegebene Datenbank wird geöffnet und zur aktuellen Datenbank gemacht.

2.1.8 Datenbank schließen

Nach der Bearbeitung müssen Sie die Datenbank schließen. Dazu gibt es die Anweisung CLOSE DATABASE.

Die aktuelle Datenbank wird automatisch geschlossen, wenn Sie mit DATABASE die Datenbank wechseln.

2.1.9 ANSI-Datenbank

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.

ANSI-Datenbank erstellen

Sie können eine ANSI-Datenbank erstellen, indem Sie bei CREATE DATABASE die Klausel MODE ANSI angeben.

Von Nicht-ANSI auf ANSI wechseln

Bei INFORMIX-SE können Sie eine Nicht-ANSI-Datenbank mit START DATABASE in eine ANSI-Datenbank umwandeln.

Bei INFORMIX-ONLINE kann nur der INFORMIX-ONLINE-Verwalter eine Nicht-ANSI-Datenbank in eine ANSI-Datenbank umwandeln und zwar mit dem DB-Monitor im Menü LOGICAL LOGS mit der Menüfunktion *Databases*.

[!] Eine ANSI-Datenbank kann nicht mehr in eine Nicht-ANSI-Datenbank umgewandelt werden.

2.1.10 Entfernte Datenbank

Wenn Sie die INFORMIX-Netzprodukte einsetzen, können Sie mit entfernten Datenbanken arbeiten.

INFORMIX-SE plus **INFORMIX-NET**

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

Eine entfernte Datenbank ist eine Datenbank auf einem anderen Rechner.

INFORMIX-ONLINE plus **INFORMIX-STAR**

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

Zusätzlich haben Sie die Möglichkeit, Daten aus externen Tabellen abzufragen (siehe Abschnitt 2.1.6 *Aktuelle Datenbank bearbeiten*).

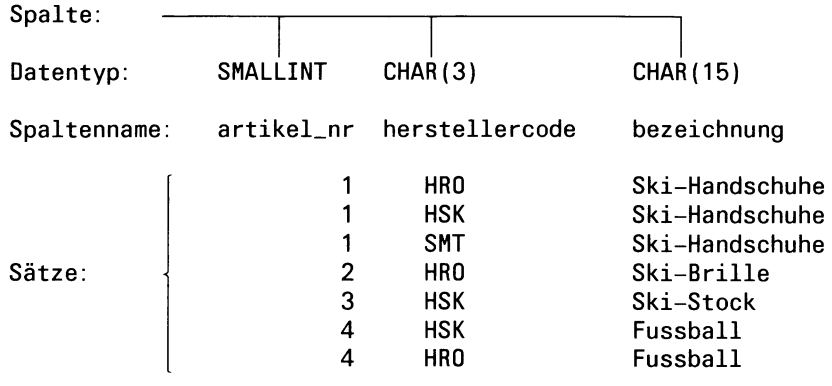
Eine entfernte Datenbank kann sein:

- eine Datenbank auf einem anderen Rechner
- eine Datenbank aus einem anderen INFORMIX-ONLINE-System auf demselben Rechner.

2.2 Tabelle

Die Daten einer Datenbank sind in Tabellen organisiert. Eine Tabelle besteht aus Sätzen und Spalten. Sie hat einen Eigentümer und ein festes Format, genannt **Tabellenschema**.

Das Tabellenschema legt Anzahl und Reihenfolge der Spalten fest, sowie die Art einer Spalte, ihren Namen und Datentyp:



Die Sätze einer Tabelle sind nicht geordnet. Dagegen ist ein Satz selbst eine geordnete Folge von Werten. Die Ordnung ist durch die Reihenfolge der Spalten definiert. Jeder Satz einer Tabelle hat dieselbe Anzahl von Spalten und enthält für jede Spalte einen Wert. Ein Satz ist die kleinste Einheit von Daten, die in eine Tabelle eingefügt oder aus einer Tabelle gelöscht werden kann.

Es gibt drei Arten von Tabellen:

- Basistabellen
- Views
- temporäre Tabellen.

2.2.1 Basistabelle

Die Basistabellen sind die Tabellen, in denen die Daten der Datenbank abgespeichert und gehalten werden. Eine Basistabelle wird nach ihrer Definition permanent gespeichert, bis sie gelöscht wird.

Eigentümer der Basistabelle ist der Benutzer, der die Tabelle entweder selbst erzeugt hat oder von einem DBA (**Daten**bank**a**dministrator) als Eigentümer eingesetzt wurde.

Basistabelle definieren

Eine Basistabelle wird mit CREATE TABLE in der aktuellen Datenbank erstellt. Bei der Definition geben Sie das Tabellenschema an (Tabellennamen, Spaltendefinitionen, usw.).

Basistabelle löschen

Wird eine Basistabelle nicht mehr benötigt, kann sie mit DROP TABLE gelöscht werden. Dabei werden alle Daten, die diese Tabelle betreffen, automatisch gelöscht. Die Auswirkungen sind im einzelnen in Kapitel 6 bei DROP TABLE beschrieben.

Wird die Datenbank gelöscht, werden die zugehörigen Tabellen automatisch mitgelöscht.

Basistabelle bearbeiten

Sie können auf jede existierende Basistabelle der aktuellen Datenbank zugreifen, vorausgesetzt Sie haben die notwendigen Zugriffsrechte (siehe Abschnitt 2.9 *Zugriffsrechte*).

Sie können Daten abfragen und Sätze eintragen, ändern und löschen. Dazu stehen Ihnen alle SQL-Anweisungen zur Datendefinition und -manipulation zur Verfügung.

Sie können das Tabellenschema einer Basistabelle nach der Definition mit ALTER TABLE verändern, zum Beispiel Spalten hinzufügen oder löschen und mit RENAME TABLE den Tabellennamen ändern.

Zusätzlich können Sie mit CREATE SYNONYM weitere Namen für eine Basistabelle definieren (siehe Kapitel 3, Abschnitt 3.2 *Namen*).

Sie können mit CREATE INDEX Indizes auf Basistabellen definieren, um Eindeutigkeit der Spaltenwerte zu erzwingen oder Ihre Abfragen zu optimieren (siehe Abschnitt 2.6, *Index*).

Mit GRANT können Sie Zugriffsrechte vergeben, um anderen Benutzern Zugriff auf Ihre Daten zu erlauben oder zu verbieten (siehe Abschnitt 2.9 *Zugriffsrechte*).

INFORMIX-ONLINE:
Sie können zusätzlich Daten aus einer externen Basistabelle abfragen. Setzen Sie zusätzlich INFORMIX-STAR ein, kann die Datenbank, zu der die externe Tabelle gehört, auch in einem anderen Informixsystem liegen.

Die Datenabfrage aus einer externen Tabelle ist nur möglich, wenn gilt:

- Die Datenbanken sind entweder beide ANSI-Datenbanken oder beide Nicht-ANSI-Datenbanken.
- Die Datenbanken sind entweder beide mit Transaktionssicherung definiert oder beide ohne.

Die Daten der externen Tabelle werden mit derselben Isolationsstufe gelesen, die für die aktuelle Datenbank eingestellt ist (siehe Abschnitt 2.11 *Sperren*).

Systemtabellen

Zu den Basistabellen gehören auch die von INFORMIX vordefinierten Systemtabellen, in denen die Datenbankdefinitionen abgespeichert werden.

Im Anhang A.2 finden Sie eine Liste aller Systemtabellen mit Erklärung der Inhalte.

Eigentümer der Systemtabellen ist die Kennung *informix*.

Die Eintragungen in den Systemtabellen werden von INFORMIX vorgenommen.

INFORMIX läßt zwar zu, daß ein DBA Sätze in Systemtabellen einträgt, ändert und löscht. Doch davon sollte aber nur in Notfällen Gebrauch gemacht werden!

2.2.2 View

Ein View ist keine echte Tabelle, sondern eine Sicht auf eine oder mehrere Tabellen, die durch eine SELECT-Abfrage definiert wird.

Beispiel:

Tabelle konto:

konto_nr	kundenname	ort	betrag
0011	Julius	Frankfurt	-1233.56
0023	Anton	Muenchen	120000.00
0058	Becker	Muenchen	23000.56
0089	Franz	Frankfurt	5.00
0077	Krug	Muenchen	-670.00

Die gerasterten Teile zeigen eine Sicht auf die Tabelle *konto*, bei der nur die Kontonummer, der Kundenname und der Ort derjenigen Kunden zu sehen sind, die aus München kommen.

Die Tabellen, auf die sich ein View bezieht, sind entweder Basistabellen oder zuvor definierte Views. Auf diese Weise kann der Benutzer eine Hierarchie von Views auf eine oder mehrere Tabellen aufbauen.

Die Besonderheit eines View besteht darin, daß lediglich die SELECT-Abfrage abgespeichert wird, die den View definiert. Die Daten sind nur in den zugrundeliegenden Basistabellen vorhanden.

Eigentümer des View ist der Benutzer, der den View entweder selbst erzeugt hat oder für den ein DBA den View erzeugt hat.

View definieren

Ein View wird mit der Anweisung CREATE VIEW durch eine SELECT-Abfrage definiert, in der die zugrundeliegenden Tabellen angegeben sind und die Bedingung, mit der aus diesen Tabellen der gewünschte Ausschnitt ausgewählt wird. Den View-Namen geben Sie bei der Definition an. Das restliche Tabellenschema ist das Tabellenschema der Ergebnistabelle der SELECT-Anweisung.

Beispiel:

View aus obigem Beispiel definieren:

```
CREATE VIEW muc_konto
AS SELECT konto_nr, kundenname, ort
FROM konto
WHERE ort="Muenchen"
```

Die SELECT-Abfrage wird bei der Definition noch nicht ausgewertet.

Die View-Definition wird in der Systemtabelle *sysviews* abgespeichert, der View-Name in der Systemtabelle *systables*.

Sie können einen View mit der WITH CHECK OPTION-Klausel definieren, damit in die Tabelle nur Sätze eingefügt werden können, die die SELECT-Abfrage erfüllen (siehe unten, *Änderbare Views*).

Spätere Umbenennungen der zugrundeliegenden Tabellen oder Spalten werden nicht nachgezogen.

View löschen

Ein View wird automatisch gelöscht, wenn die zugrundeliegenden Tabellen gelöscht werden.

Wenn Sie nur eine Spalte mit ALTER TABLE löschen, die an einem View beteiligt ist, können Sie nicht mehr mit diesem View arbeiten.

Außerdem können Sie einen View mit DROP VIEW löschen.

View bearbeiten

Sie können einen View zur Datenabfrage und unter Einschränkungen (siehe unten) auch zur Datenmanipulation verwenden. Dabei bedeutet Bearbeiten eines View, die zugrundeliegenden Basistabellen bearbeiten.

Wenn Sie in einer SQL-Anweisung den View angeben, wird die in der View-Definition angegebene SELECT-Abfrage ausgeführt. Auf diese Weise werden die Sätze und Spalten der zugrundeliegenden Basistabellen bestimmt, die die Anweisung eigentlich betrifft.

Bei der Datenabfrage stellt sich ein View dem Benutzer wie eine echte Tabelle dar.

Setzen Sie INFORMIX-ONLINE ein, gilt das auch für externe Views. Für Views eines anderen Informixsystems ist zusätzlich INFORMIX-STAR erforderlich.

Beispiel:

Kontonummer und Name aller Kunden abfragen:

```
SELECT *
 FROM muc_konto

konto_nr  kundenname      ort
------------------------------
    0023  Anton       Muenchen
    0058  Becker      Muenchen
    0077  Krug        Muenchen
```

Änderbare Views

Einfüge- und Änderungsanweisungen sind bei einem View nur erlaubt, wenn aus der View-Definition eindeutig hervorgeht, welche Sätze einer Basistabelle betroffen sind und zusätzlich für die betroffenen Sätze die jeweilige Operation erlaubt ist. Bei CREATE VIEW sind die Bedingungen für änderbare Views im einzelnen zusammengestellt.

Wurde ein änderbarer View mit der WITH CHECK OPTION-Klausel definiert, können Sie nur Sätze über den View eingeben oder ändern, die die View-Bedingung erfüllen. Sätze, die die Bedingung nicht erfüllen, werden abgewiesen.

Ohne diese Klausel können Sätze über den View eingefügt oder geändert werden, die die Bedingung der SELECT-Abfrage nicht erfüllen. Diese Sätze sind anschließend nicht mehr über den View zugreifbar.

Nicht erlaubte Anweisungen

In folgenden Anweisungen ist die Angabe eines Views verboten:

Anweisung	Bedeutung
ALTER TABLE	Tabellenstruktur ändern
CHECK TABLE	Tabelle prüfen
CREATE AUDIT	AUDIT-Protokoll erzeugen
CREATE INDEX	Index erzeugen
DROP AUDIT	AUDIT-Protokoll löschen
DROP TABLE	Tabelle löschen
LOCK TABLE	Tabelle sperren
RECOVER TABLE	Tabelle aktualisieren
RENAME COLUMN	Spaltennamen ändern
RENAME TABLE	Tabellennamen ändern
REPAIR TABLE	Index für Tabelle wiederherstellen
UNLOCK TABLE	Tabellensperre aufheben

Außerdem können Sie für einen View wie für eine Basistabelle weitere Namen definieren und Zugriffsrechte vergeben. In Kapitel 6 sind bei CREATE VIEW alle erlaubten Anweisungen explizit aufgeführt.

Vorteile von Views

Die Vorteile von Views sind Speicherplatzersparnis und Datenschutz. Daten, die in verschiedenen Kombinationen benötigt werden, müssen nur einmal in ihrer Gesamtheit tatsächlich abgespeichert werden. Durch geeignet definierte Views erreicht man, daß je nach Bedarf nur die benötigten Ausschnitte herausgesucht werden. So können zum Beispiel aus Sicherheitsgründen kritische Daten (wie im Beispiel der Kontostand) vor nichtprivilegierten Benutzern verborgen werden, während privilegierte Benutzer auf alle Daten zugreifen dürfen.

2.2.3 Temporäre Tabelle

Eine temporäre Tabelle ist eine Tabelle, die am Ende einer Datenbanksitzung automatisch gelöscht wird. Solange die Tabelle existiert, sind ihre Daten gespeichert.

Eigentümer der temporären Tabelle ist der **Prozeß**, der die Tabelle erzeugt.

Temporäre Tabelle definieren

Eine temporäre Tabelle wird nicht der aktuellen Datenbank zugeordnet. Es gibt zwei Möglichkeiten, temporäre Tabellen zu erstellen.

Mit der Anweisung CREATE TEMP TABLE erzeugen Sie eine benannte temporäre Tabelle (siehe CREATE TABLE), wobei Sie wie bei der Definition einer Basistabelle die Tabellenstruktur angeben (Tabellennamen, Spaltendefinitionen, usw.).

Mit SELECT erzeugen Sie eine temporäre Ergebnistabelle. Mit der INTO TEMP-Klausel können Sie die Ergebnistabelle benennen, ansonsten erhält sie keinen Namen. Eine Ergebnistabelle entsteht aus einer oder mehreren Tabellen bei der Ausführung einer SELECT-Abfrage. Die Tabellenstruktur wird aus den zugrundeliegenden Tabellen abgeleitet (siehe Kapitel 6, SELECT/Spaltenauswahl).

Temporäre Tabelle löschen

Die Lebensdauer einer temporären Tabelle hängt davon ab, ob sie benannt ist oder nicht.

Eine benannte temporäre Tabelle kann mit DROP TABLE innerhalb der Sitzung gelöscht werden. Ansonsten wird sie automatisch am Ende der Datenbanksitzung gelöscht.

Eine Ergebnistabelle ohne Namen wird im interaktiven Betrieb nach der Ausgabe gelöscht. Im eingebetteten Betrieb können Sie solange auf die Ergebnistabelle zugreifen, wie der zugehörige Satzzeiger geöffnet ist.

Temporäre Tabelle bearbeiten

Nur der Prozeß, der die temporäre Tabelle erstellt, kann die Tabelle bearbeiten.

Im Unterschied zu einer Basistabelle werden beim Wechsel der Datenbank innerhalb desselben Informixsystems die temporären Tabellen mitgenommen. Daher kann der Prozeß, der die temporäre Tabelle erstellt hat, beim Wechsel der Datenbank weiterhin auf die Tabelle, solange sie existiert, mit demselben Namen zugreifen.

Für die Bearbeitung temporärer Tabellen gelten einige Einschränkungen gegenüber Basistabellen. Folgende Anweisungen dürfen Sie nicht mit temporären Tabellen ausführen:

Anweisung	Bedeutung
ALTER INDEX	Index ändern
ALTER TABLE	Tabellenstruktur ändern
CHECK TABLE	Tabelle prüfen
CREATE AUDIT	AUDIT-Protokoll erzeugen
CREATE SYNONYM	Synonym definieren
CREATE VIEW	View erzeugen
DROP AUDIT	AUDIT-Protokoll löschen
DROP SYNONYM	Synonym löschen
GRANT	Zugriffsrechte vergeben
INFO	Tabellen-Informationen ausgeben
RECOVER TABLE	Tabelle aktualisieren
RENAME COLUMN	Spaltennamen ändern
RENAME TABLE	Tabellennamen ändern
REPAIR TABLE	Index für Tabelle wiederherstellen
REVOKE	Zugriffsrechte entziehen

Ansonsten können Sie wie bei Basistabellen Daten abfragen und Sätze eintragen, ändern und löschen.

2.3 Synonym

Für Basistabellen und Views können Sie zusätzlich zu dem in der Definition festgelegten Namen weitere Namen vergeben. Die zusätzlichen Namen heißen Synonyme.

Mit Synonymen können Sie kürzere oder aussagekräftige Namen für die Tabellen vergeben, die Sie in Ihren Anweisungen verwenden.

Dies ist insbesondere interessant, wenn Sie auf externe Tabellen einer entfernten Datenbank zugreifen.

2.3.1 Synonym definieren

Sie definieren ein Synonym mit CREATE SYNONYM. Sie können ein Synonym für eine Tabelle der aktuellen Datenbank definieren.

Ein Synonym für eine Tabelle in derselben Datenbank wird automatisch nachgezogen, wenn die Tabelle später umbenannt wird.

[!] Das gilt nicht für Synonyme von externen Tabellen (siehe nächsten Abschnitt).

INFORMIX-ONLINE

Sie können auch für externe Tabellen ein Synonym definieren. Setzen Sie zusätzlich INFORMIX-STAR ein, kann die Datenbank, zu der die externe Tabelle gehört, in einem anderen Informixsystem liegen.

Dabei gilt:

- Bei der Definition dürfen Sie die externe Tabelle auch mit einem Synonym ansprechen, das für diese Tabelle in der fremden Datenbank definiert wurde.

  Beispiel: In der fremden Datenbank *sachbuecher* des Informixsystems *alpha* ist für die Tabelle *autoren* das Synonym *aut* definiert. Sie dürfen dann in der aktuellen Datenbank ein Synonym für diese Tabelle wie folgt definieren (siehe Abschnitt 3.2.5 *Qualifizierung von Namen*) :

  ```
  CREATE SYNONYM sach_aut FOR //alpha/sachbuecher:aut
  ```

- Ein Synonym für eine externe Tabelle wird nicht gelöscht, wenn die Tabelle gelöscht wird und nicht nachgezogen, wenn die Tabelle umbenannt wird. Das hat folgende Auswirkungen:

- Wird in der fremden Datenbank eine neue Tabelle mit dem Namen erzeugt, für den das Synonym erzeugt wurde, kann diese neue Tabelle jetzt mit dem Synonym angesprochen werden.
- Bei einer Umbenennung kann die ursprüngliche Tabelle, für die das Synonym definiert wurde, nicht mehr mit dem Synonym angesprochen werden.

2.3.2 Synonym löschen

Ein Synonym für eine interne Tabellen (in derselben Datenbank), wird automatisch gelöscht, wenn die Tabelle gelöscht wird.

[!] Das gilt nicht für Synonyme von externen Tabellen (siehe Abschnitt 2.3.1 *Synonym definieren*, Unterabschnitt *INFORMIX-ONLINE*).

Außerdem können Sie ein Synonym mit der Anweisung DROP SYNONYM löschen.

2.3.3 Synonym verwenden

Ein Synonym kann wie der Haupttabellenname dazu verwendet werden, die Tabelle in SQL-Anweisungen anzusprechen. Allerdings ist ein Synonym nicht ganz gleichwertig mit dem Hauptnamen der Tabelle.
In folgenden Anweisungen ist die Verwendung eines Synonyms verboten:

Anweisung	Bedeutung
CHECK TABLE	Tabelle prüfen
CREATE SYNONYM	Synonym definieren
REPAIR TABLE	Index für Tabelle wiederherstellen

Ansonsten können Sie das Synonym immer statt des Tabellennamens in einer SQL-Anweisung angeben. In Kapitel 6 sind bei CREATE SYNONYM alle erlaubten Anweisungen explizit aufgeführt.

2.4 Spalten

Die Sätze einer Tabelle sind in Spalten aufgeteilt. Eine Spalte besitzt einen Namen und einen Datentyp.

2.4.1 Spalte definieren

Die Spalten einer Tabelle werden bei der Tabellendefinition festgelegt. Diese definiert:

- Anzahl und Reihenfolge der Spalten
- Spaltennamen für jede Spalte
- Datentyp für jede Spalte (siehe Abschnitt 2.5, *Werte und Datentypen*) Alle Werte bzw. BLOB-Daten in einer Spalte haben denselben Datentyp.

Bei Basistabellen und mit CREATE TABLE erzeugten temporären Tabellen können zusätzlich:

- für jede Spalte NULL-Werte erlaubt oder verboten werden.
- Eindeutigkeitsbedingungen für eine oder mehrere Spalten mit einem Constraint festgelegt werden (siehe Abschnitt 2.7 *Constraint*).

Bei Basistabellen und mit CREATE TABLE erzeugten temporären Tabellen erfolgt die Spaltendefinition explizit durch Angaben in der CREATE TABLE-Anweisung.

Bei Views und Ergebnistabellen werden die Spaltendefinitionen von den Spalten der zugrundeliegenden Basistabellen abgeleitet.

2.4.2 Spalten bearbeiten

Sie können Werte in Spalten eintragen (INSERT), ändern (UPDATE) und aus Spalten abfragen (SELECT).

Dabei unterscheidet INFORMIX wie jedes relationale Datenbanksystem NULL-Werte und Nicht-NULL-Werte (siehe Abschnitt 2.5).

Ein einzelner Spaltenwert ist die kleinste Einheit von Daten, die in eine Tabelle eingetragen oder aus einer Tabelle ausgewählt oder geändert werden kann.

2.5 Werte und Datentypen

Die Inhalte der Tabellenspalten werden bei INFORMIX unterteilt in:

- Werte
  Sie werden zum Eintragen (INSERT) und Ändern (UPDATE) von Spaltenwerten verwendet und um Berechnungen und Vergleiche durchzuführen (SELECT, DELETE, UPDATE). Wie in jedem relationalen Datenbanksystem gibt es bei INFORMIX NULL-Werte und Nicht-NULL-Werte:

  – NULL-Werte
    NULL-Werte sind eine Besonderheit von relationalen Datenbanken. Enthält eine Spalte einen NULL-Wert bedeutet das, daß der Inhalt der Spalte undefiniert oder unbekannt ist.

    Die Konstante für NULL-Werte heißt NULL.

    Die INFORMIX-spezifische Behandlung von NULL-Werten ist in Kapitel 4, Abschnitt 4.3 *Werte verwenden* ausführlich beschrieben.

  – Nicht-NULL-Werte
    Nicht-NULL-Werte sind entweder numerische Werte, alphanumerische Werte oder Zeitwerte (bei INFORMIX: Datum, Zeitpunkt oder Zeitspanne).

    Die Konstanten für Nicht-NULL-Werte müssen in der von INFORMIX festgelegten Syntax angegeben werden. In Kapitel 4 ist die INFORMIX-Syntax für alle Konstanten beschrieben.

- BLOB-Daten
  BLOB-Daten sind Byte-Folgen bis zu einer theoretischen Länge von 2^{31}. Es gibt BLOB-Daten vom Typ TEXT und vom Typ BYTE.
  TEXT-Daten sind Folgen aus ASCII-Zeichen und den Formatsteuerzeichen für Tabulator, neue Zeile und neue Seite. BYTE-Daten sind beliebige Byte-Folgen, die aus INFORMIX-Sicht nicht weiter strukturiert sind.

  BLOB-Daten können nur in Spalten eingetragen und wieder ausgewählt werden. Sie können nicht wie Werte in Berechnungen und Vergleichen verwendet werden. Daher werden sie zur Unterscheidung **BLOB-Daten** genannt.

  Wie Sie mit BLOB-Spalten und -Daten arbeiten, ist ausführlich in Kapitel 4, Abschnitt 4.2 *Datentypen definieren* beschrieben.

Für jede Spalte einer Tabelle muß ein Datentyp angegeben werden. Dieser Datentyp legt fest, welche Werte bzw. Daten in die Spalte eingetragen werden können. INFORMIX unterstützt folgende Datentypen:

- Alphanumerische Datentypen für alphanumerische Werte
  - CHARACTER
  - VARCHAR (nur INFORMIX-ONLINE)
- Numerische Datentypen für numerische Werte
  - Ganzzahlige Datentypen:
    INTEGER, SMALLINT, SERIAL
  - Realzahl-Datentypen:
    DECIMAL bzw. NUMERIC, MONEY, SMALLFLOAT bzw. REAL, FLOAT bzw. DOUBLE PRECISION
- Zeit-Datentypen für Zeitwerte:
  DATE, DATETIME, INTERVAL
- BLOB-Datentypen für BLOB-Daten (nur INFORMIX-ONLINE):
  TEXT, BYTE.

Jeder Datentyp außer SERIAL beinhaltet zusätzlich den NULL-Wert. Soll für eine Spalte der NULL-Wert ausgeschlossen werden, muß dies bei der Definition mit CREATE TABLE durch die Klausel NOT NULL angegeben werden (siehe Abschnitt 2.4 *Spalten*).

In Kapitel 4 sind alle INFORMIX-Datentypen mit ihren zugehörigen Werten bzw. Daten ausführlich beschrieben.

2.6 Index

Unter Verwendung eines Index kann der Zugriff auf eine Tabelle beschleunigt werden.

Ein Index wird für eine oder mehrere Spalten einer Tabelle definiert. Er legt fest, in welcher Reihenfolge bei einer Abfrage die Sätze aus der Tabelle gesucht werden. Die Werte der Indexspalten legen das Sortierkriterium fest und der Index gibt zusätzlich an, ob auf- oder absteigend sortiert wird.

Die zusätzliche Verwaltung des Index erhöht den Aufwand bei Einfüge- und Änderungsanweisungen. Daher sollten Sie folgende Richtlinien berücksichtigen:

- Erzeugen Sie keine Indizes für Tabellen mit weniger als 200 Sätzen. Die Geschwindigkeit, die Sie mit einem Index erzielen, ist nicht größer als die Zeit, die nötig ist, um die Indexdatei kleiner Tabellen zu öffnen und zu durchsuchen.
- Erzeugen Sie keinen Index für eine Spalte, die wenig unterschiedliche Werte enthält.

INFORMIX unterstützt folgende Index-Arten:

- Normalen Index
- Eindeutigen Index
  Die Spaltenwerte der Indexspalten müssen alle verschieden sein.
- Cluster-Index
  Die Sätze der Tabelle werden in der Reihenfolge des Index sortiert und physisch auf der Platte in dieser Reihenfolge abgespeichert.

Eigentümer des Index ist der Benutzer, der den Index entweder selbst erzeugt hat oder für den ein DBA den Index erzeugt hat.

2.6.1 Index definieren

Ein Index wird mit CREATE INDEX definiert, wobei Sie mit der Klausel ASC bzw. DESC festlegen, ob ein aufsteigender oder absteigender Index erzeugt wird.

Mit der Klausel UNIQUE können Sie einen eindeutigen Index erzeugen, mit der Klausel CLUSTER einen Cluster-Index.

Sie müssen darauf achten, daß der Index tatsächlich zu einer Optimierung der Abfragen führt.

Außerdem müssen Sie bereits bestehende Indizes und Eindeutigkeitsbedingungen berücksichtigen, insbesondere, wenn Sie einen eindeutigen oder Cluster-Index definieren. Die Einzelheiten, auch die möglichen Konfliktfälle, sind in Kapitel 6 bei CREATE INDEX ausführlich beschrieben.

Ein Index wird automatisch nachgezogen, wenn die Indexspalten später umbenannt werden.

2.6.2 Index löschen

Ein Index wird automatisch gelöscht, wenn die zugehörigen Indexspalten bzw. die ganze Tabelle gelöscht wird.

Außerdem können Sie einen Index mit DROP INDEX löschen.

2.6.3 Index ändern

Mit ALTER INDEX können Sie einen Index in einen Cluster-Index umwandeln bzw. einen Cluster-Index in einen normalen Index.

2.7 Constraint

Ein Constraint ist eine Eindeutigkeitsbedingung, die für Tabellenspalten festgelegt werden kann.

Sie können einen einfachen Constraint auf eine Spalte definieren oder einen zusammengesetzten Constraint auf mehrere Spalten.

Die Werte in den Constraint-Spalten müssen unterschiedlich sein.

Bei INFORMIX wird der Constraint durch einen eindeutigen Index realisiert, der aufsteigend sortiert ist.

Eigentümer des Constraint ist der Benutzer, der den Constraint entweder selbst erzeugt hat oder für den ein DBA den Constraint erzeugt hat.

2.7.1 Constraint definieren

Ein Constraint wird durch Angabe der Constraint-Klausel mit der CREATE TABLE- oder ALTER TABLE-Anweisung definiert.

Sie müssen bereits bestehende Indizes und Eindeutigkeitsbedingungen berücksichtigen. Die Einzelheiten, auch die möglichen Konfliktfälle, sind in Kapitel 6 bei CREATE INDEX, CREATE TABLE und ALTER TABLE ausführlich beschrieben.

2.7.2 Constraint löschen

Ein Constraint wird automatisch gelöscht, wenn die zugehörigen Constraint-Spalten bzw. die ganze Tabelle gelöscht wird.

Außerdem können Sie einen Constraint mit ALTER TABLE löschen, indem Sie die Klausel DROP CONSTRAINT angeben.

2.8 Benutzer

INFORMIX hat keine eigene Benutzerverwaltung. Zur Identifizierung eines Benutzers wird die SINIX-Benutzerkennung verwendet.

Der Datenbankzugriff und die Festlegung, ob und welche Datenbankobjekte ein Benutzer definieren kann, wird über die Datenbankzugriffsrechte geregelt.

Es gibt drei Datenbankzugriffsrechte, die eine Hierarchie von Benutzerklassen definieren:

- Connect
  Connect ist das eingeschränkteste Datenbankzugriffsrecht. Benutzer mit Connect-Zugriffsrecht dürfen lediglich:
  - auf bestehende Tabellen der Datenbank zugreifen (SELECT, INSERT, UPDATE und DELETE), vorausgesetzt sie haben das erforderliche Tabellenzugriffsrecht.
  - Views, temporäre Tabellen und Indizes auf temporären Tabellen erstellen, ändern und löschen.
- Resource
  Benutzer mit Resource-Zugriffsrecht besitzen automatisch das Connect-Zugriffsrecht. Zusätzlich dürfen sie Basistabellen erstellen, die sie ändern und löschen können. Außerdem dürfen sie für diese Basistabellen Indizes erzeugen, ändern und löschen sowie Zugriffsrechte vergeben und entziehen.
- DBA (**D**aten**bank**administrator)
  DBA ist das umfassendste Datenbankzugriffsrecht. Benutzer mit DBA-Zugriffsrecht besitzen automatisch das Resource-Zugriffsrecht. Zusätzlich dürfen sie:
  - Alle Tabellenzugriffsrechte für alle Tabellen der Datenbank vergeben und entziehen (GRANT...AS *benutzer*)
  - Tabellen, Views, Synonyme, Indizes und Constraints für andere Benutzer erzeugen (CREATE ...) und löschen (DROP ...)
  - Die Datenbank löschen (DROP DATABASE)
  - Transaktionssicherung einschalten bei einer SE-Datenbank (START DATABASE)

- Bei einer SE-Datenbank die Datenbanksicherung aktualisieren (ROLLFORWARD DATABASE)
- Datenbankzugriffsrechte Connect, Resource und DBA vergeben und entziehen (GRANT, REVOKE).

Es kann mehrere Benutzer mit Zugriffsrecht DBA geben.

Wenn Sie eine Datenbank mit CREATE DATABASE erzeugen, besitzen Sie automatisch das DBA-Zugriffsrecht für diese Datenbank.

2.8.1 Wem werden Datenbankzugriffsrechte zugeteilt?

Datenbankzugriffsrechte darf nur ein DBA vergeben und entziehen. Dazu verwendet er die GRANT-Anweisung. Er kann Datenbankzugriffsrechte zuteilen an:

- Einzelne Benutzer
  Ein Benutzer wird über seinen SINIX-Benutzernamen angesprochen.
- Allgemeinheit
  Die Allgemeinheit wird mit PUBLIC bezeichnet. Zugriffsrechte, die an PUBLIC vergeben wurden, gelten als Standardzugriffsrechte für alle Benutzer der Datenbank (siehe Abschnitt 2.9 *Zugriffsrechte*).

2.8.2 ANSI-Datenbank

Bei einer ANSI-Datenbank sind die Datenbankobjekte den einzelnen Benutzern zugeordnet. Das hat folgende Auswirkungen:

- Die Namen der Objekte müssen nur pro Datenbankbenutzer eindeutig sein.
- Bei Zugriff auf ein Objekt eines anderen Benutzers muß der Objektname mit dem Benutzernamen qualifiziert werden.

Nähere Beschreibung siehe Kapitel 3, Abschnitt 3.2, *Namen*.

2.9 Zugriffsrechte

Die Zugriffsrechte regeln den Zugriff auf die Datenbank und ihre Objekte. Bevor eine SQL-Anweisung ausgeführt wird, überprüft INFORMIX, ob der Benutzer das erforderliche Zugriffsrecht besitzt. Besteht das Zugriffsrecht, wird die Anweisung ausgeführt, ansonsten wird sie mit einer Fehlermeldung abgewiesen.

2.9.1 Welche Zugriffsrechte gibt es?

Bei INFORMIX gibt es folgende Arten von Zugriffsrechten:

- Datenbankzugriffsrechte (siehe Abschnitt 2.8, *Benutzer*)
- Tabellenzugriffsrechte
  - Select-Zugriffsrecht:
    erlaubt das Auswählen von Sätzen der Tabelle (SELECT).
  - Insert-Zugriffsrecht:
    erlaubt das Einfügen von Sätzen in die Tabelle (INSERT).
  - Delete-Zugriffsrecht:
    erlaubt das Löschen von Sätzen der Tabelle (DELETE).
  - Update-Zugriffsrecht:
    erlaubt das Ändern von Sätzen der Tabelle (UPDATE).
  - Index-Zugriffsrecht:
    erlaubt das Erzeugen eines Index für die Tabelle.
  - Alter-Zugriffsrecht:
    erlaubt das Ändern des Tabellennamens (RENAME TABLE) und des Tabellenschemas (ALTER TABLE).

2.9.2 Voreingestellte Zugriffsrechte

INFORMIX setzt standardmäßig folgende Zugriffsrechte:

- Datenbankzugriffsrechte

  Der Benutzer, der die Datenbank erstellt, erhält das DBA-Zugriffsrecht für die Datenbank. Sonst gibt es keine standardmäßig gesetzten Datenbankzugriffsrechte.

- Tabellenzugriffsrechte

  In einer Nicht-ANSI-Datenbank werden automatisch bei jeder neu erzeugten Tabelle folgende Tabellenzugriffsrechte gesetzt:

  ```
  GRANT DELETE,INSERT,SELECT,UPDATE TO PUBLIC
  ```

  ANSI-Datenbank:
  Bei einer ANSI-Datenbank hat kein anderer Benutzer als der Eigentümer standardmäßig Zugriffsrechte auf die Tabelle. Alle Zugriffsrechte müssen Sie explizit mit GRANT vergeben.

2.9.3 Eigentümer

Der Eigentümer eines Datenbankobjekts ist der Benutzer, der das Objekt entweder selbst erzeugt hat, oder für den ein DBA das Objekt erzeugt hat.

Der Eigentümer einer Tabelle hat automatisch alle Tabellenzugriffsrechte. Diese können ihm nicht entzogen werden. Sie werden auch bei INFO nicht angezeigt.

2.9.4 Zugriffsrechte vergeben und entziehen

Zugriffsrechte werden mit GRANT vergeben und mit REVOKE entzogen.

Wer darf Zugriffsrechte vergeben und entziehen?

Der Eigentümer eines Datenbankobjekts und ein DBA dürfen anderen Benutzern Zugriffsrechte für dieses Objekt zuteilen.

Ein Zugriffsrecht darf nur derjenige entziehen, der das Zugriffsrecht vergeben hat oder anstelle dessen ein DBA das Zugriffsrecht vergeben hat.

Außerdem kann ein Benutzer die Tabellenzugriffsrechte weitergeben, die ihm mit der Klausel WITH GRANT OPTION zugeteilt wurden.

Wem werden Zugriffsrechte zugeteilt?

Zugriffsrechte werden zugeteilt an:

- Einzelne Benutzer
  Ein Benutzer wird über seinen SINIX-Benutzernamen angesprochen.
- Allgemeinheit
  Die Allgemeinheit wird mit PUBLIC bezeichnet. Zugriffsrechte, die an PUBLIC vergeben wurden, gelten als minimale Zugriffsrechte für alle Benutzer der Datenbank. Sie können nur PUBLIC, aber nicht einzelnen Benutzern entzogen werden.

2.9.5 Überprüfung der Zugriffsrechte

INFORMIX überprüft die Zugriffsrechte an Hand der Systemtabellen, in denen die vergebenen Rechte eingetragen sind.

Bei der Überprüfung wird wie folgt vorgegangen:

- Zuerst wird überprüft, ob dem Benutzer das Zugriffsrecht explizit zugeteilt wurde.
- Sind keine Einträge für den speziellen Benutzer vorhanden, wird überprüft, ob das Zugriffsrecht PUBLIC zugeteilt wurde.

2.9.6 Zusammenwirken von Datenbank- und Tabellenzugriffsrechten

Die Datenbankzugriffsrechte legen fest, welche Rechte ein Benutzer hat, wenn ihm keine anderen Rechte zugeteilt wurden.

Tabellenzugriffsrechte wirken nur, wenn die erforderlichen Datenbankzugriffsrechte vorliegen.

Die folgende Tabelle beschreibt die Zusammenhänge zwischen Datenbank- und Tabellenzugriffsrechten. Sie zeigt, welche Zugriffsrechte **mindestens** für die Ausführung einer SQL-Anweisung nötig sind.

Es müssen jeweils die angegebenen Tabellen- **und** Datenbankzugriffsrechte erfüllt sein.

[!] Da das DBA-Zugriffsrecht alle Tabellenrechte für alle Tabellen der Datenbank umfaßt, kann ein Benutzer mit DBA-Zugriffsrecht alle folgenden Anweisungen ausführen.

Anweisung	Mindest-Zugriffsrechte	
	Tabellenebene	Datenbankebene
ALTER INDEX	Tabelleneigentümer oder Index	Connect
ALTER TABLE	Tabelleneigentümer oder Alter	Connect
CHECK TABLE	Tabelleneigentümer	Connect
CREATE AUDIT	Tabelleneigentümer	Connect
CREATE INDEX	Tabelleneigentümer oder Index	Resource
CREATE SYNONYM	-	Connect
CREATE TABLE	-	Resource
CREATE TEMP TABLE	-	Connect
CREATE VIEW	Tabelleneigentümer oder Select	Connect
DATABASE	-	Connect
DELETE	Tabelleneigentümer oder Delete	Connect
DROP AUDIT	Tabelleneigentümer	Connect
DROP DATABASE	-	DBA
DROP INDEX	Indexeigentümer	Connect
DROP SYNONYM	Synonymeigentümer	Connect
DROP TABLE	Tabelleneigentümer	CONNECT
DROP VIEW	Vieweigentümer	Connect
GRANT	Tabelleneigentümer Grant-Berechtigung -	Connect Connect DBA
INFO	-	Connect
INSERT	Tabelleneigentümer oder Insert	Connect
LOAD	Tabelleneigentümer oder Insert	Connect
LOCK TABLE	Tabelleneigentümer	Connect
OUTPUT	Tabelleneigentümer oder Select	Connect

Anweisung	Mindest-Zugriffsrechte	
	Tabellenebene	Datenbankebene
RECOVER TABLE	Tabelleneigentümer	Connect
RENAME COLUMN	Tabelleneigentümer oder Alter	Connect
RENAME TABLE	Tabelleneigentümer oder Alter	Resource
REPAIR TABLE	Tabelleneigentümer	Connect
REVOKE Tabellen-zugriffsrecht	Benutzer, der das Tabellen-zugriffsrecht vergeben hat	Connect
REVOKE Datenbank-zugriffsrecht	-	DBA
ROLLFORWARD DATABASE	-	DBA
SELECT	Tabelleneigentümer oder Select	Connect
START DATABASE	-	DBA
UNLOAD	Tabelleneigentümer oder Select	Connect
UPDATE	Tabelleneigentümer oder Update	Connect
UPDATE STATISTICS	-	Connect

2.10 Transaktionssicherung

Das Konzept der Transaktionssicherung ermöglicht die Einhaltung und Wiederherstellung von konsistenten Zuständen in der Datenbank. Im allgemeinen erfordert der Übergang von einem konsistenten Zustand in den nächsten konsistenten Zustand eine Folge von Datenbankoperationen.

Beispiel:

> In einer Datenbank für den Flugverkehr gibt es mehrere Tabellen, in denen Flugzeiten abgespeichert sind. Wird jetzt bei einem Flug die Abflugzeit geändert, muß in allen Tabellen, die diesen Flug enthalten, die Abflugzeit entsprechend geändert werden. Das heißt, es müssen eventuell mehrere UPDATE-Anweisungen ausgeführt werden, damit die Daten inhaltlich korrekt bleiben. Während der Ausführung der einzelnen UPDATE-Anweisungen entstehen inkonsistente Zustände, da ein Teil der Tabellen noch die alte Abflugzeit enthält, während einige bereits geändert sind. Wird jetzt die Ausführung aus irgendeinem Grund unterbrochen, bleibt die Datenbank in einem inkonsistenten Zustand, da nur ein Teil der UPDATE-Anweisungen ausgeführt werden konnte.

Das Konzept der Transaktionssicherung hilft dem Benutzer, solche inhaltlichen Inkonsistenzen zu vermeiden.

Was ist Transaktionssicherung?

Transaktionssicherung bedeutet, daß

- Anweisungsfolgen als Transaktion ausgeführt werden können
- Über die Ausführung von Transaktionen ein Protokoll geführt wird.

2.10.1 Was ist eine Transaktion?

Eine Transaktion ist eine Folge von SQL-Anweisungen, die eine logische Einheit bilden. Bei der Ausführung einer Transaktion sorgt INFORMIX dafür, daß die zu der Transaktion gehörenden Anweisungen entweder vollständig ausgeführt oder vollständig zurückgesetzt werden.

Anweisungsfolge als Transaktion kennzeichen

Soll eine Anweisungsfolge als Transaktion gelten, wird sie in eine **Transaktionsklammer** gesetzt.

Eine Transaktionsklammer besteht aus der Anweisung BEGIN WORK zum Starten einer Transaktion und den Anweisungen COMMIT WORK bzw. ROLLBACK WORK zum Beenden der Transaktion:

```
BEGIN WORK;

  Anweisung;
  Anweisung;
...
COMMIT WORK bzw. ROLLBACK WORK;
```

Alle zwischen BEGIN WORK und COMMIT WORK bzw. ROLLBACK WORK angegebenen Anweisungen werden zu einer Transaktion zusammengefaßt.

COMMIT WORK wird angegeben, wenn nach Ausführung der Transaktion alle innerhalb der Transaktion ausgeführten Änderungen in der Datenbank **festgeschrieben** werden sollen. Festschreiben bedeutet, die Änderungen werden auf die Platte geschrieben und sind dann in der Datenbank vorhanden.

ROLLBACK WORK wird angegeben, wenn alle innerhalb der Transaktion ausgeführten Änderungen **zurückgesetzt** werden sollen. Zurücksetzen bedeutet, die Änderungen werden nicht auf die Platte geschrieben und sind nicht in der Datenbank vorhanden.

ANSI-Datenbank

Bei einer ANSI-Datenbank ist jede Anweisung automatisch innerhalb einer Transaktionsklammer, da BEGIN WORK implizit bei Beginn der Datenbanksitzung und nach jedem COMMIT WORK bzw. ROLLBACK WORK gesetzt wird.

Die Anweisungen BEGIN WORK, COMMIT WORK und ROLLBACK WORK sind ausführlich in Kapitel 6 beschrieben.

2.10.2 Transaktionsprotokoll

Bei Transaktionssicherung führt INFORMIX ein Protokoll über die Änderungen, die während einer Transaktion ausgeführt werden. Dieses Protokoll heißt Transaktionsprotokoll. Es wird für die gesamte Datenbank geführt.

An Hand des Transaktionsprotokolls kann INFORMIX nach einem Systemabsturz oder Plattenfehler die gesicherte Datenbank wieder so herstellen, daß gilt:

- Alle vor dem Systemabsturz festgeschriebenen Transaktionen bleiben erhalten.
- Die zum Fehlerzeitpunkt offenen Transaktionen sind zurückgesetzt, hinterlassen also in der Datenbank keinerlei Effekte.

(siehe auch Abschnitt 2.12 *Datenschutz*).

INFORMIX-ONLINE

Sie können wählen zwischen ungepufferter und gepufferter Transaktionsprotokollierung:

- Bei ungepufferter Transaktionsprotokollierung wird das Protokoll sofort auf Platte geschrieben.
- Bei gepufferter Transaktionprotokollierung wird das Protokoll zuerst in einen Puffer und später der Puffer auf die Platte geschrieben. Es ist daher möglich, daß geänderte Sätze nach abgeschlossenen Transaktionen nur im Puffer gespeichert sind und der Puffer noch nicht auf Platte geschrieben ist. Dadurch können bei der Restaurierung nach einem Systemfehler abgeschlossene Transaktionen fehlen. Die Performance wird durch die gepufferte Protokollierung gesteigert.

Sie können den Protokollierungsmodus bei der Erstellung der Datenbank angeben. Außerdem kann jeder Benutzer den Protokollierungsmodus mit der SET LOG-Anweisung nach seinen Wünschen einstellen. Die Anweisung SET LOG ist ausführlich in Kapitel 6 beschrieben.

2.10.3 Ausführung von Transaktionen

Wird eine als Transaktion gekennzeichnete Anweisungsfolge ausgeführt, geht INFORMIX wie folgt vor:

- Die Anweisungen werden nacheinander ausgeführt.
- Änderungen werden im Transaktionsprotokoll festgehalten.
- Ist zum Abschluß COMMIT WORK angegeben, wird die Transaktion beendet und die in der Transaktion durchgeführten Änderungen werden auf der Datenbank festgeschrieben.
- Ist zum Abschluß ROLLBACK WORK angegeben, wird die Transaktion auch beendet, aber die in der Transaktion durchgeführten Änderungen werden zurückgesetzt.

  Beachten Sie, daß nicht alle Anweisungen zurückgesetzt werden können. Bei welchen Anweisungen dies möglich ist, ist auch abhängig vom Backend. In Kapitel 6 ist bei jeder Anweisung für jedes Backend angegeben, ob sie zurückgesetzt werden kann.

Wird die Ausführung einer Transaktion unterbrochen, müssen Sie angeben, ob die Transaktion zurückgesetzt oder festgeschrieben werden soll.

Auswirkungen von Transaktionen auf Sperren

Transaktionen haben Auswirkungen auf:

- Tabellensperren
- Schreibsperren (INFORMIX-SE)
- Exclusive-Einzelsperren (INFORMIX-ONLINE)
- Share-Einzelsperren bei Isolationsstufe Repeatable Read (INFORMIX-ONLINE).

Diese Sperren werden immer bis zum Transaktionsende gehalten.

Die Sperren sind ausführlich im Abschnitt 2.11 beschrieben.

2.10.4 Möglichkeiten der Transaktionssicherung

Abhängig davon, ob Sie mit einer NICHT-ANSI-Datenbank oder einer ANSI-Datenbank arbeiten und welches Backend (INFORMIX-SE, INFORMIX-ONLINE) Sie benutzen, gibt es verschiedene Möglichkeiten der Transaktionssicherung:

- Datenbank ohne Transaktionssicherung
- Datenbank mit Transaktionssicherung

  INFORMIX-ONLINE:
  - Mit ungepufferter Transaktionsprotokollierung
  - Mit gepufferter Transaktionsprotokollierung

Die einzelnen Möglichkeiten sind im folgenden beschrieben.

Datenbank ohne Transaktionssicherung

Dieser Modus bedeutet keine Sicherheit.
Es gibt keine Transaktionen. Alle SQL-Anweisungen werden ungesichert ausgeführt und können nicht zurückgesetzt werden.

Bei einer Datenbank ohne Transaktionssicherung können bereits bei Ausführung einer einzelnen Änderungsanweisung inkonsistente Zustände in der Datenbank entstehen.

Beispiel:

> Wird eine UPDATE-Anweisung, mit der 100 Sätze geändert werden sollen, nach Änderung des 50. Satzes abgebrochen, bleibt die Datenbank in einem inkonsistenten Zustand. 50 Sätze enthalten bereits die geänderten Werte, die restlichen 50 sind noch auf dem alten Stand.

Datenbank mit Transaktionssicherung

Sie arbeiten mit Transaktionen.

Nicht-ANSI-Datenbank

Sie können wählen, ob Sie Anweisungen außerhalb einer Transaktionsklammer oder innerhalb einer Transaktionsklammer angeben.

- Ohne Transaktionsklammer

  Dieser Modus bedeutet Sicherheit für eine einzelne Anweisung.
  Wenn Sie keine Transaktionsklammer angeben, wird jede einzelne Anweisung implizit als Transaktion betrachtet. Das hat folgende Auswirkungen:

  – Eine Anweisung wird implizit entweder vollständig ausgeführt oder zurückgesetzt. Im Unterschied zur Datenbank ohne Transaktionssicherung können daher bei Ausführung einer einzelnen Änderungsanweisung keine inkonsistenten Zustände in der Datenbank entstehen.

  – Eine einzelne Anweisung kann nicht mit ROLLBACK WORK zurückgesetzt werden.

  – Mehrere Anweisungen können nicht zu einer Transaktion zusammengefaßt werden.

- Mit Transaktionsklammer

  Dieser Modus bedeutet Sicherheit für eine Anweisungsfolge.
  Wenn Sie eine Transaktionsklammer angeben, werden alle in der Klammer angegebenen Anweisungen zu einer Transaktion zusammengefaßt. Das hat folgende Auswirkungen:

  – Alle in der Klammer angegebenen Anweisungen werden entweder vollständig ausgeführt oder zurückgesetzt. Das heißt, es kann jetzt auch bei Ausführung mehrerer Anweisungen sichergestellt werden, daß keine inkonsistenten Zustände entstehen.

  – Die Anweisungen können mit ROLLBACK WORK zurückgesetzt werden.

ANSI-Datenbank

Bei einer ANSI-Datenbank ist immer Transaktionssicherung eingestellt und BEGIN WORK wird implizit gesetzt. Das heißt bei Start der Datenbanksitzung wird mit BEGIN WORK die erste Transaktion gestartet und nach jeder beendeten Transaktion automatisch mit BEGIN WORK die nächste Transaktion eröffnet, sodaß jede Anweisung innerhalb einer Transaktionsklammer BEGIN WORK und COMMIT WORK bzw. ROLLBACK WORK steht. Das hat folgende Auswirkungen:

- BEGIN WORK sollte nicht mehr explizit angegeben werden. Die Einzelheiten sind bei BEGIN WORK im Kapitel 6 beschrieben.
- Eine Transaktion wird solange nicht beendet, bis Sie explizit COMMIT WORK bzw. ROLLBACK WORK angeben. Darauf sollten Sie achten, da die von Ihnen gehaltenen Sperren vor Transaktionsende nicht freigegeben werden. Das gilt auch für Share-Sperren, die bei Isolationsstufe Repeatable Read gesetzt werden.

2.10.5 Transaktionssicherung einstellen

Nicht-ANSI-Datenbank

Bei einer Nicht-ANSI-Datenbank kann die Transaktionssicherung wie folgt eingestellt werden:

- bei der Definition der Datenbank mit CREATE DATABASE
- während der Datenbanksitzung:

  INFORMIX-SE:
  Mit START DATABASE kann der DBA eine Datenbank ohne Transaktionssicherung auf eine Datenbank mit Transaktionssicherung umschalten.

  INFORMIX-ONLINE:
  Nur der INFORMIX-ONLINE-Verwalter kann nach einer Archivierung die Transaktionssicherung ein- bzw. ausschalten (siehe ONLINE-Handbuch [10]).

ANSI-Datenbank

Bei einer ANSI-Datenbank ist automatisch Transaktionssicherung eingeschaltet. Sie kann weder ein- noch ausgeschaltet werden.

2.11 Sperren

Sperren werden benötigt, um bei paralleler Arbeit mehrerer Benutzer auf derselben Datenbank die Konsistenz der Daten gewährleisten zu können.

Eine Sperre wird bei Ausführung einer SQL-Anweisung gesetzt. Der kleinste sperrbare Bereich ist ein Satz.

Die Sperre wird dem **Prozeß** zugeordnet, der die SQL-Anweisung ausführen will. Er ist der Sperrenhalter.

Welche Operationen für andere Benutzer noch erlaubt sind, ist abhängig von der Art der Sperre.

Es gibt Unterschiede hinsichtlich:

- des Sperrgranulats:
  es gibt den Bereich an, der gesperrt wird.
- des Sperren-Typs:
  er legt die Wirkung fest, die eine Sperre hat.

2.11.1 Sperrgranulat

Das Sperrgranulat richtet sich nach der Größe des gesperrten Bereichs. Es gibt folgende Bereiche, die gesperrt werden können:

- Datenbank
  Die Datenbank steht exklusiv einem Benutzer zur Verfügung. Andere Benutzer dürfen nicht mit der Datenbank arbeiten, solange sie gesperrt ist.
- Tabelle
  Die ganze Tabelle wird gesperrt.
  Es gibt zwei Sperren-Typen für Tabellen:
  Sie können eine Tabelle im Share-Modus oder im Exclusive-Modus sperren. Bei einer Tabellensperre im Share-Modus dürfen andere Benutzer die Tabelle noch lesen, aber nicht mehr verändern. Bei einer Tabellensperre im Exclusive-Modus dürfen andere Benutzer die Tabelle weder lesen noch verändern.

  Auf eine Tabelle darf zu einem Zeitpunkt immer nur eine Tabellensperre gesetzt werden.

- Satz bzw. Page
  Ein Satz bzw. eine Page (nur INFORMIX-ONLINE) wird gesperrt. Satz- und Pagesperren werden in diesem Handbuch **Einzelsperren** genannt. Die Sperren-Typen für Einzelsperren sind im nächsten Abschnitt erklärt.

  INFORMIX-ONLINE:
  Bei einer Satzsperre wird nur der jeweilige Satz gesperrt. Wenn der Satz über mehrere Pages geht, werden diese Pages gesperrt, aber nur soweit, wie sie von dem zu sperrenden Satz belegt sind.
  Bei einer Pagesperre wird die ganze Page gesperrt.

Das gröbste Sperrgranulat ergibt sich beim Sperren der gesamten Datenbank, das feinste beim Sperren eines Satzes.

2.11.2 Einzelsperren

Welche Einzelsperren es gibt, ist abhängig vom INFORMIX-Backend.

Einzelsperren bei INFORMIX-SE

Es gibt nur einen Sperren-Typ, die Schreibsperre für Änderungsanweisungen. Bei allen Anweisungen, die einen Satz einfügen, verändern oder löschen (INSERT, LOAD, DELETE, UPDATE) wird eine Schreibsperre gesetzt. Diese Sperre hat folgende Wirkung:

Andere Benutzer dürfen die gesperrten Sätze nicht ändern aber noch lesen. Dies entspricht bei INFORMIX-ONLINE der Exclusive-Sperre mit Isolationsstufe Dirty Read.

Auf einen gesperrten Satz kann keine weitere Sperre gesetzt werden.

Einzelsperren bei INFORMIX-ONLINE

INFORMIX-ONLINE unterstützt verschiedene Typen von Einzelsperren. Zusätzlich gibt es eine Isolationsstufe, die festlegt, ob beim Lesen fremde Sperren berücksichtigt und eigene gesetzt werden.

Sperren-Typen

- Share-Sperre
  Andere Benutzer dürfen die gesperrten Sätze nicht mehr ändern aber noch lesen.

  Auf einen mit einer Share-Sperre gesperrten Bereich können noch weitere Share-Sperren und noch eine Update-Sperre gesetzt werden, aber keine Exclusive-Sperre.

- Exclusive-Sperre
  Andere Benutzer dürfen die gesperrten Sätze nicht ändern und nur noch abhängig von der Isolationsstufe lesen.

  Auf einen mit einer Exclusive-Sperre gesperrten Bereich können keine anderen Sperren gesetzt werden.

- Update-Sperre
  Die Update-Sperre hat dieselben Eigenschaften wie eine Share-Sperre, außer daß auf einen Bereich, auf dem bereits eine Update-Sperre ist, keine weitere Update-Sperre gesetzt werden kann. Die Update-Sperre wird bei Programmeinbettung bei einem Select-Satzzeiger verwendet, der mit FOR UPDATE definiert wurde. Bei Ausführung der SELECT-Anweisung wird auf den gelesenen Satz eine Update-Sperre gesetzt.

  Wird der gelesene Satz anschließend mit UPDATE...WHERE CURRENT OF geändert bzw. mit DELETE...WHERE CURRENT OF gelöscht, wird die Update-Sperre in eine Exclusive-Sperre umgewandelt. Daher heißt die Update-Sperre im Englischen auch **promotable lock**.

Isolationsstufe

Die Isolationsstufe legt fest, ob beim Lesen (SELECT) fremde Exclusive-Sperren berücksichtigt und eigene Share-Sperren gesetzt werden.
Sie gibt an, inwieweit die gelesenen Sätze vor Änderungen fremder Prozesse geschützt sind.

Die Isolationsstufe gilt auch für Tabellensperren.

Es gibt vier Isolationsstufen mit unterschiedlichem Sicherheitsgrad. Diese Isolationsstufen vom niedrigsten bis zum höchsten Sicherheitsgrad sind:

- Dirty Read
  Keine Sicherheit.

  Es werden keine fremden Sperren berücksichtigt und keine eigenen Share-Sperren gesetzt.
  Das gilt auch für fremde Exclusive-Sperren. Ein Prozeß mit Isolationsstufe Dirty Read kann also Sätze lesen, auf die ein anderer Prozeß eine Exclusive-Sperre hält. Er liest somit auf eigene Verantwortung. Er kann zum Beispiel Sätze lesen, die noch nicht in der Datenbank festgeschrieben sind, weil sie ein anderer Prozeß innerhalb einer Transaktion eingefügt hat, die später zurückgesetzt wird.

- Committed Read
  Sicherheit, daß beim Lesen der Satz noch vorhanden ist.

  Wirkt nur bei Datenbank mit Transaktionssicherung.

  Fremde Exclusive-Sperren werden berücksichtigt, indem versucht wird, eine eigene Share-Sperre zu setzen. Ist dies möglich, bedeutet es, daß keine Exclusive-Sperre besteht und die Daten werden gelesen ohne eine eigene Share-Sperre zu setzen.
  Ein Prozeß mit Isolationsstufe Committed Read kann also keine Sätze lesen, die es nicht in der Datenbank gibt. Allerding kann, während er die Sätze liest, ein anderer Prozeß die Sätze ändern oder löschen.

- Cursor Stability
  Sicherheit, daß der Satz während des Lesens nicht geändert wird.

  Wirkt nur bei Datenbank mit Transaktionssicherung.

  Fremde Exclusive-Sperren werden berücksichtigt. Zusätzlich wird eine eigene Share-Sperre auf den Satz gesetzt, der gelesen werden soll. Die Sperre wird freigegeben, sobald der nächste Satz gelesen wird.
  Während ein Prozeß mit Isolationsstufe Cursor Stability einen Satz liest, kann also kein anderer Prozeß diesen Satz ändern oder löschen.

- Repeatable Read
  Sicherheit, daß innerhalb der Transaktion alle gelesenen Sätze nicht geändert werden.

  Wirkt nur bei Datenbank mit Transaktionssicherung.

  Fremde Exclusive-Sperren werden berücksichtigt. Zusätzlich wird eine eigene Share-Sperre auf alle Sätze gesetzt, die mit der SELECT-Anweisung innerhalb einer Transaktion gelesen werden. Die Sperre wird bis zum Transaktionsende gehalten. Ein Prozeß mit Isolationsstufe Repeatable Read kann daher innerhalb einer Transaktion dieselben Sätze mehrfach lesen und sicher sein, daß kein anderer Prozeß diese Sätze geändert oder gelöscht hat.

Isolationsstufe einstellen:

Von INFORMIX-ONLINE ist die Isolationsstufe wie folgt voreingestellt:

- Nicht-ANSI-Datenbank
  - Ohne Transaktionssicherung: Dirty Read
  - Mit Transaktionssicherung: Committed Read
- ANSI-Datenbank: Repeatable Read

Die Isolationsstufe kann bei einer Datenbank mit Transaktionssicherung mit der Anweisung SET ISOLATION (siehe Kapitel 6) eingestellt werden.

2.11.3 Wie werden Sperren gesetzt?

Datenbanksperre

Eine Datenbank wird mit DATABASE EXCLUSIVE gesperrt.

Tabellensperre

Eine Tabelle kann explizit mit LOCK TABLE gesperrt werden. Dabei können Sie angeben, ob eine Share- oder Exclusive-Tabellensperre gesetzt werden soll.

Implizit setzt INFORMIX Exclusive-Tabellensperren bei Anweisungen, die die Tabellenstruktur betreffen, wie zum Beispiel ALTER TABLE.

Einzelsperre

Einzelsperren werden nur implizit vergeben und zwar:

INFORMIX-SE:
Schreibsperren bei Änderungsanweisungen (INSERT, LOAD, UPDATE, DELETE).

INFORMIX-ONLINE:
- Exclusive-Sperren bei Änderungsanweisungen (INSERT, LOAD, UPDATE, DELETE).
- Share-Sperren bei SELECT-Anweisungen abhängig von der Isolationsstufe.

In Kapitel 6 ist für jede Anweisung beschrieben, ob und welche Sperren gesetzt werden.

2.11.4 Anzahl der Sperren

Die Anzahl der Sperren, die insgesamt gehalten werden können, ist abhängig davon, welches Backend (INFORMIX-SE, INFORMIX-ONLINE) Sie benutzen.

- INFORMIX-SE:
  Die Anzahl ist betriebssystemabhängig und pro Rechner beschränkt.
- INFORMIX-ONLINE:
  Die Anzahl ist durch Shared Memory Parameter festgelegt und pro Informixsystem beschränkt.
  Verwenden Sie BLOB-Spalten, müssen Sie berücksichtigen, daß für BLOB-Daten mehr Sperren benötigt werden und zwar:
  - bei INSERT, LOAD und DELETE werden pro BLOB-Page 2 Sperren benötigt.
  - bei UPDATE werden pro BLOB-Page 4 Sperren benötigt.
  - bei SELECT werden pro BLOB-Page 2 Sperren benötigt.

2.11.5 Wie lange werden Sperren gehalten?

Wie lange Sperren gehalten werden, ist bei Tabellensperren, SE-Schreibsperren und ONLINE-Einzelsperren abhängig von Transaktionen:

- Tabellensperre
  - Datenbank ohne Transaktionssicherung: bis die Anweisung UNLOCK gegeben wird
  - Datenbank mit Transaktionssicherung: bis Transaktionsende
- Exclusive-Satzsperre
  - Datenbank ohne Transaktionssicherung: bis der nächste Satz geändert wird
  - Datenbank mit Transaktionssicherung: bis Transaktionsende

Bei ONLINE-Share-Einzelsperren ist es abhängig von der Isolationsstufe, wie lange die Sperren gehalten werden (siehe oben).

Verhalten bei gesperrten Sätzen

Will ein Prozeß Sätze bearbeiten, die gesperrt sind, wartet er standardmäßig nicht auf die Freigabe, sondern kehrt sofort mit einer Fehlermeldung zurück.

Dieses Verhalten können Sie mit der Anweisung SET LOCK MODE (siehe Kapitel 6) ändern.

Mit der Klausel WAIT legen Sie fest, daß der Prozeß nicht zurückkehrt, sondern auf die Freigabe wartet. Dabei müssen Sie darauf achten, daß es nicht zu Verklemmungen kommt, wenn zwei Prozesse gegenseitig auf die Freigabe von Sperren warten.

Mit der Klausel NO WAIT können Sie wieder das Standardverhalten einstellen.

INFORMIX-ONLINE:
Sie können zusätzlich eine maximale Wartezeit festlegen.

2.12 Datenschutz

Der Datenschutz ist ein sehr wesentlicher Bereich eines Datenbanksystems, insbesondere bei großen Datenbanken, auf denen mehrere Benutzer gleichzeitig arbeiten.

Datenschutz umfaßt Datensicherheit, Datenintegrität und Datensicherung.

2.12.1 Datensicherheit

Unter Datensicherheit versteht man den Schutz der Daten eines Benutzers vor fremden Zugriffen. SQL-Konzepte, die zur Datensicherheit beitragen, sind:

- Views (siehe Abschnitt 2.2 *Tabellen*)
- Zugriffsrechte (siehe Abschnitt 2.9 *Zugriffsrechte*)
- Transaktionssicherung (siehe Abschnitt 2.10 *Transaktionssicherung*)
- Sperren (siehe Abschnitt 2.11 *Sperren*).

2.12.2 Datenintegrität

Datenintegrität bedeutet, daß die Daten in der Datenbank inhaltlich korrekt sind.

Beispiel:

> Bei einem Datum darf die Monatsangabe nur eine Zahl von 1 bis 12 sein.

SQL-Konzepte, die den Benutzer bei der Einhaltung der Datenintegrität unterstützen, sind:

- Datentyp-Festlegungen (siehe Kapitel 4, Abschnitt 4.2 *Datentyp definiern*)
- Indizes und Constraints (siehe Abschnitte 2.6 *Index* und 2.7 *Constraint*)
- Transaktionssicherung (siehe Abschnitt 2.10 *Transaktionssicherung*)
- Sperren (siehe Abschnitt 2.11 *Sperren*).

2.12.3 Datensicherung

Zur Datensicherung gehören Verfahren, die es ermöglichen, eine Datenbank oder einzelne Tabellen auf eine andere Platte oder auf Band zu sichern, um sie nach Systemfehlern, insbesondere Plattenfehlern, zu restaurieren. Die Möglichkeiten, die INFORMIX bereitstellt, sind abhängig davon, welches Backend Sie verwenden und davon, ob Sie eine Datenbank mit oder ohne Transaktionssicherung einsetzen. Sie sind in diesem Abschnitt zusammengestellt.

INFORMIX-SE-Sicherungsverfahren

Sie können nur mit Betriebssystemmitteln die Datenbank sichern und nach Fehlern die Sicherungskopie wiedereinspielen (siehe Handbuch SINIX-Systemverwaltung [11]).

Mögliche Datensicherungsverfahren im Zusammenhang mit den von INFORMIX bereitgestellten Protokollierungsmöglichkeiten sind:

- Ohne Transaktionssicherung

  Sie können für einzelne Tabellen ein Audit-Protokoll erstellen (siehe Kapitel 6, CREATE AUDIT) und sichern. Nach einem Fehler muß dann nur die Tabelle und, falls diese nicht mehr vorhanden ist, das zugehörige Audit-Protokoll eingespielt werden, um die Tabelle mit RECOVER TABLE wiederherzustellen.

- Mit Transaktionssicherung

  Sie führen eine Datenbanksicherung **unmittelbar** nach der Anweisung CREATE DATABASE bzw. START DATABASE durch, in der Sie angegeben haben, in welche Datei das Transaktionsprotokoll geschrieben wird. Dieser Dateiname wird dann mit auf Band gesichert.

  Das Transaktionsprotokoll können Sie unabhängig von der Datenbank sichern.

  Zum Restaurieren spielen Sie die gesicherte Datenbank und das zugehörige Transaktionsprotokoll ein, falls es nicht mehr auf der Platte vorhanden ist. Anschließend können Sie mit ROLLFORWARD DATABASE und dem Transaktionsprotokoll die Datenbank wiederherstellen, sodaß alle seit der Sicherung festgeschriebenen Transaktionen wieder nachgezogen sind.

INFORMIX-ONLINE-Sicherungsverfahren

Die Datensicherung übernimmt der INFORMIX-ONLINE-Verwalter.

Er sichert die Datenbank über die Archive-Funktion des DB-Monitors mit der Menüfunktion Backup auf Band. Dazu spielt er die Sicherungskopie über die Archive-Funktion des DB-Monitors mit der Menüfunktion Restore wieder ein (siehe INFORMIX-ONLINE-Handbuch[10]).

INFORMIX-ONLINE bietet die Möglichkeit, eine 3-stufige, inkrementelle Sicherung durchzuführen.

Mögliche Datensicherungsverfahren im Zusammenhang mit den von INFORMIX bereitgestellten Protokollierungsmöglichkeiten sind:

- Ohne Transaktionssicherung

  Es gibt keine weiteren Möglichkeiten.

- Mit Transaktionssicherung

  Bei INFORMIX-ONLINE findet automatisch beim nächsten Start nach einem Systemabsturz ein schneller Wiederanlauf statt, bei dem die zum Absturzzeitpunkt offenen Transaktionen automatisch zurückgesetzt werden.

  Für Absicherung bei Plattenfehlern sichern Sie das Transaktionsprotokoll unabhängig von der Datenbank über das Menü LOGICAL LOGS des DB-Monitors mit einer Backup-Menüfunktion auf Band (siehe ONLINE-Handbuch [10]).

  Nach einem Fehler spielen Sie die Datenbank wieder ein. Das Transaktionsprotokoll wird automatisch nachgefordert. Mit dem Transaktionsprotokoll kann die Datenbank wiederhergestellt werden, sodaß die seit der Sicherung festgeschriebenen Transaktionen wieder nachgezogen sind. Hatten Sie gepufferte Transaktionsprotokollierung eingestellt, kann es sein, daß die letzten Transaktionen verloren gegangen sind (siehe Abschnitt 2.10 *Transaktionssicherung*).

Datenbank-Spiegelung

INFORMIX-ONLINE bietet die Möglichkeit der Datenbank-Spiegelung. Dabei werden die Chunks eines Dbspace oder Blobspace doppelt geführt. Werden Chunks beschädigt, greift INFORMIX automatisch auf die Spiegel-Chunks zu.

2.13 Programmeinbettung

Die SQL-Anweisungen können in eine Programmiersprache eingebettet werden. Dadurch besteht die Möglichkeit, aus Programmen heraus auf die Datenbank zuzugreifen.

INFORMIX bietet Schnittstellen für die Programmiersprachen:

- INFORMIX-4GL
- C
- COBOL

Für die Programmeinbettung gibt es bei SQL zusätzlich folgende Konzepte:

- Hostvariable
- Indikatorvariable
- Satzzeiger
- Dynamisch formulierte Anweisung
- Erfolgskontrolle über die *sqlca*-Struktur

Im folgenden sind diese Konzepte allgemein erklärt. Die Einzelheiten sind sprachspezifisch und daher in den INFORMIX-Handbüchern für die jeweilige Sprache beschrieben.

2.13.1 Hostvariable

Eine Hostvariable ist eine Variable, die dazu dient, in einer SQL-Anweisung Werte aus der Datenbank aufzunehmen oder in die Datenbank zu speichern und Werte bereitzustellen, die in Berechnungen benötigt werden.

Die Hostvariable muß im Programm entsprechend den Konventionen der Programmiersprache deklariert werden.

Der Datentyp der Hostvariablen richtet sich nach dem Datentyp der INFORMIX-Werte, für die diese Hostvariable verwendet werden soll. Für Datentypen, die in der Programmiersprache nicht vorhanden sind, stellt die jeweilige INFORMIX-Sprachschnittstelle vordefinierte Datentypen bereit, die für die Hostvariablen verwendet werden müssen. In den INFORMIX-Handbüchern für die einzelnen Programmiersprachen sind Datentypzuordnungen und vordefinierte INFORMIX-Datentypen jeweils speziell für die Programmiersprache angegeben.

INFORMIX-ONLINE

Sie können auch Hostvariablen für BLOB-Daten verwenden, die aus der Datenbank abgefragt oder in die Datenbank gespeichert werden.

Die Hostvariablen für BLOB-Daten können nicht in Berechnungen verwendet werden.

Bei INFORMIX-4GL unter INFORMIX-ONLINE können Sie mit FREE den Speicherplatz freigeben, den Hostvariablen für BLOB-Daten belegen.

2.13.2 Indikatorvariable

Eine Hostvariable kann mit einer Indikatorvariablen kombiniert sein. Sie dient dazu, den in Programmiersprachen nicht vorhandenen NULL-Wert auszudrücken und die Übertragung der Werte von der und in die Datenbank zu kontrollieren.

Die Indikatorvariable ist eine Variable. Sie muß wie die Hostvariable im Programm entsprechend den Konventionen der Programmiersprache deklariert werden.

Sie kann in SELECT-, FETCH-, INSERT-, UPDATE-Anweisungen verwendet werden und wird dann hinter der zugehörigen Hostvariablen angegeben. Die genaue Syntax ist sprachspezifisch und im INFORMIX-Handbuch für die jeweilige Programmiersprache beschrieben.

Die Indikatorvariable hat folgende Wirkung:

Bei der Abfrage von Werten oder BLOB-Daten aus der Datenbank mit Zuweisung an die Hostvariable (SELECT, FETCH) wird die Indikatorvariable automatisch von INFORMIX wie folgt belegt:

0	Die Hostvariable enthält einen definierten Wert. Die Zuweisung war fehlerfrei.
< 0	Der Wert, der zugewiesen werden sollte, ist der NULL-Wert.
> 0	Bei alphanumerischen Werten: Es wurde eine Zeichenkette zugewiesen, die verkürzt wurde. Der Wert gibt die Originallänge an.

Wenn Sie Sätze in die Datenbank einfügen (INSERT) oder ändern (UPDATE) und Spaltenwerte über Hostvariablen angeben, können Sie die Indikatorvariable verwenden, um einen NULL-Wert einzutragen. Dazu müssen Sie der Indikatorvariablen vor Aufruf der Anweisung einen Wert zuweisen, der kleiner als 0 ist.

Die Indikatorvariable darf nur bei der Übertragung von Werten aus der oder in die Datenbank verwendet werden.

Sie darf nicht in Berechnungen verwendet werden, das heißt in Funktionen, Ausdrücken, Prädikaten und Bedingungen, wenn ein Wert über eine Hostvariable angegeben wird.

2.13.3 Satzzeiger

Da es in einigen Programmiersprachen für den Typ Tabelle keine Entsprechung gibt, wird für die Programmeinbettung das Konzept des Satzzeigers verwendet.

Ein Satzzeiger ist einer Tabelle zugeordnet und ermöglicht es, die Sätze einer Tabelle einzeln anzusprechen. Er gibt immer eine aktuelle Position in der Tabelle an. Wird ein Satzzeiger geöffnet, steht er vor dem ersten Satz der Tabelle. Während der Bearbeitung ist die aktuelle Position entweder ein Satz oder zwischen zwei Sätzen, wenn ein Satz gelöscht wurde.

INFORMIX sorgt dafür, daß bei Ausführung einer SQL-Anweisung, in der der Satzzeiger verwendet wird, der Satzzeiger entsprechend der Anweisung verschoben wird.

INFORMIX unterstützt:

- Select-Satzzeiger für SELECT-Anweisungen
  Mit dem Select-Satzzeiger können Sie auf die einzelnen Sätze einer Ergebnistabelle zugreifen.
- Insert-Satzzeiger für INSERT-Anweisungen
  Der Insert-Satzzeiger wird verwendet, um Sätze beim Einfügen in die Datenbank zu puffern.

Ein Satzzeiger muß definiert, vor Gebrauch geöffnet und nach Gebrauch geschlossen werden.

Satzzeiger definieren

Sie definieren einen Satzzeiger mit DECLARE.

Dabei gibt es folgende Besonderheiten:

- Sie können einen Satzzeiger mit WITH HOLD definieren, um den Satzzeiger über das Transaktionsende hinaus offen zu halten. Die Auswirkungen insbesondere bezüglich Sperren sind ausführlich in Kapitel 6 bei DECLARE beschrieben.
- Sie können einen Select-Satzzeiger mit SCROLL definieren, um in beliebiger Reihenfolge auf die Sätze der Ergebnistabelle zugreifen zu können (siehe Abschnitt *Select-Satzzeiger verwenden*).
- Für Basistabellen und änderbare Views können Sie einen Select-Satzzeiger mit FOR UPDATE definieren, um Sätze in der zugrundeliegenden Basistabelle, zu ändern oder zu löschen (siehe Abschnitt *Select-Satzzeiger verwenden*). Die SELECT-Abfrage ist in diesem Fall sehr eingeschränkt. Sie darf sich insbesondere nur auf eine Basistabelle beziehen. Die Einschränkungen sind ausführlich in Kapitel 6 bei DECLARE beschrieben.

Sie **müssen** einen Select-Satzzeiger in folgenden Fällen definieren:

- Die erzeugte Ergebnistabelle hat mehr als einen Satz.
- Für eine SELECT-Anweisung ohne INTO TEMP-Klausel, die dynamisch formuliert wurde (siehe unten, Abschnitt *Dynamisch formulierte Anweisung*). In diesem Fall müssen Sie einen Satzzeiger definieren, auch wenn die Ergebnistabelle nur einen Satz enthält.

Sie können einen Select-Satzzeiger mit FOR UPDATE definieren, um Änderungen in einer Basistabelle schneller durchzuführen.

Sie können einen Insert-Satzzeiger definieren, um Sätze schneller in eine Tabelle einzutragen.

Satzzeiger öffnen

Satzzeiger sind nur in der Quellprogrammdatei ansprechbar, in der sie mit DECLARE vereinbart werden. Sie öffnen einen Satzzeiger mit OPEN. Bei den beiden Arten von Satzzeigern hat OPEN folgende Wirkung:

- Bei einem Select-Satzzeiger bestimmt OPEN die aktuellen Werte der Variablen, die in der SELECT-Anweisung vorkommen. Diese Werte gelten dann bei der Ausführung der SELECT-Anweisung. Die SELECT-Anweisung wird erst beim ersten FETCH ausgeführt. Das heißt, die Ergebnistabelle wird erst dann bestimmt.
- Bei einem Insert-Satzzeiger stellt OPEN einen Puffer für die einzufügenden Sätze bereit.

Haben Sie den Satzzeiger für eine dynamisch formulierte Anweisung mit Platzhaltern vereinbart, geben Sie bei OPEN die aktuellen Werte an.

Die Betriebsmittel, die der Satzzeiger belegt, können mit der FREE-Anweisung wieder freigegeben werden.

Satzzeiger schließen

Bei Verwendung von Transaktionen wird ein Satzzeiger automatisch geschlossen, wenn die Transaktion beendet wird und der Satzzeiger kein Satzzeiger ist, der mit WITH HOLD definiert wurde.

Außerdem können Sie einen Satzzeiger mit CLOSE schließen.

War der Satzzeiger für eine INSERT-Anweisung vereinbart, wird der Insert-Puffer als Block in die Datenbank eingefügt. Nach CLOSE existiert der Puffer nicht mehr. Ein leerer Puffer für den Satzzeiger kann erneut mit OPEN bereitgestellt werden.

Select-Satzzeiger verwenden

Ein geöffneter Select-Satzzeiger wird dazu verwendet, die Sätze einer Ergebnistabelle zu lesen. Ein Select-Satzzeiger, der mit FOR UPDATE definiert wurde, kann zusätzlich zum Ändern der Sätze verwendet werden.

Select-Satzzeiger zum Lesen

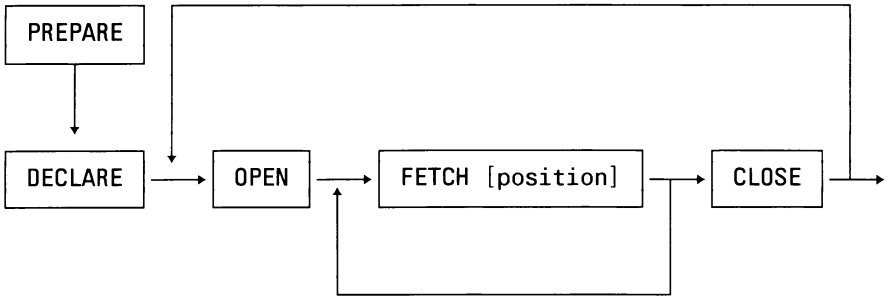
Zum Lesen verwenden Sie die Anweisung FETCH. Die erste FETCH-Anweisung nach dem OPEN führt die SELECT-Anweisung aus. Dabei werden die bei OPEN bestimmten Werte der Variablen verwendet. Der Satzzeiger steht anschließend auf dem ersten Satz der Ergebnistabelle.

Alle weiteren FETCH-Anweisungen positionieren den Satzzeiger auf einen Satz in der Ergebnistabelle und machen diesen zum aktuellen Satz. Die Werte des aktuellen Satzes können Hostvariablen oder einer *sqlda*-Struktur (ESQL/C) zugewiesen werden.

Bei einem Select-Satzzeiger ohne SCROLL können Sie nur auf den jeweils nächsten Satz positionieren.

Bei einem Scroll-Satzzeiger können Sie den Satzzeiger beliebig in der Ergebnistabelle positionieren, indem Sie bei FETCH zusätzlich eine Position angeben. Das ist möglich, da INFORMIX bei einem Scroll-Satzzeiger eine temporäre Tabelle für die Ergebnistabelle aufbaut. Bei einer FETCH-Anweisung erweitert INFORMIX die temporäre Tabelle bis zum aktuellen Satz, wenn dieser noch nicht in der temporären Tabelle enthalten ist. Die Arbeitsweise von Scroll-Satzzeigern ist ausführlich in Kapitel 6 bei DECLARE CURSOR beschrieben.

Die möglichen Anweisungen bei einem Select-Satzzeiger zum Lesen und ihre Reihenfolge zeigt folgende Zusammenfassung:

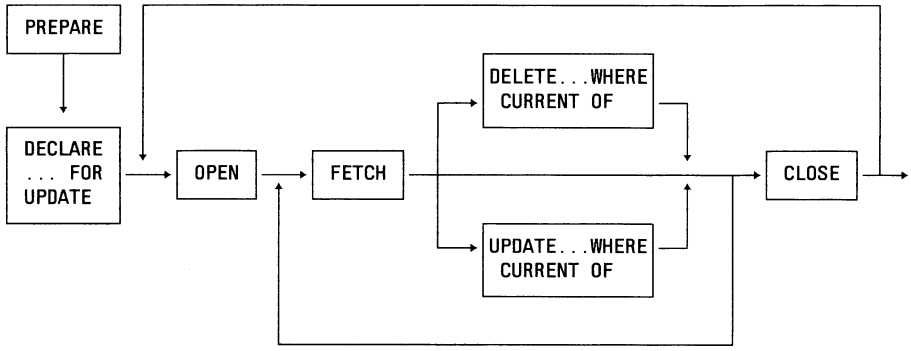


Einzelheiten sind in Kapitel 6 bei den jeweiligen Anweisungen beschrieben.

Select-Satzzeiger zum Lesen und Ändern:

Bei einem Select-Satzzeiger, der mit FOR UPDATE definiert wurde, können Sie nach dem Lesen mit einer Anweisung UPDATE...WHERE CURRENT OF bzw. DELETE...WHERE CURRENT OF einen Satz in der zugrundeliegenden Basistabelle ändern bzw. löschen. INFORMIX bestimmt den zu ändernden Satz in der Basistabelle aus der aktuellen Position des Satzzeigers in der Ergebnistabelle. Die Position des Satzzeigers wird durch die Änderungsoperation nicht verändert. Diese Art eine Basistabelle zu ändern ist effizienter als die Änderung ohne Satzzeiger.

Die möglichen Anweisungen bei einem Select-Satzzeiger, der mit FOR UPDATE definiert wurde, und ihre Reihenfolge zeigt folgende Zusammenfassung:



Einzelheiten sind in Kapitel 6 bei den jeweiligen Anweisungen beschrieben.

Insert-Satzzeiger verwenden

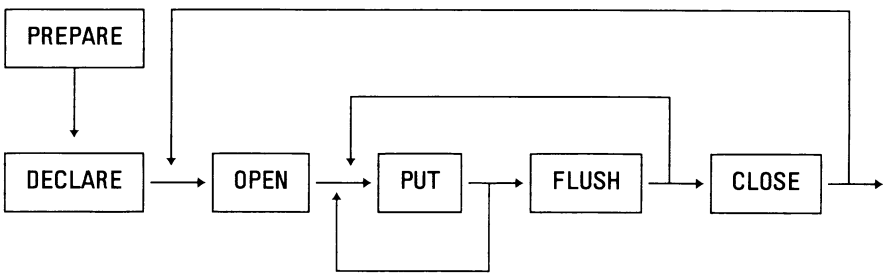
Ein geöffneter Insert-Satzzeiger wird dazu verwendet, Sätze effizienter in eine Tabelle einzutragen.

Die einzufügenden Sätze werden zuerst in einen Puffer geschrieben, der dann als Ganzes in die Tabelle eingetragen wird.

Die gepufferten Sätze werden automatisch in die Datenbank geschrieben und der Puffer geleert, wenn er voll ist oder der Satzzeiger geschlossen wird.

Außerdem können Sie auch explizit mit FLUSH die gepufferten Sätze in die Datenbank schreiben und den Puffer leeren.

Die möglichen Anweisungen bei einem Insert-Satzzeiger und ihre Reihenfolge zeigt folgende Zusammenfassung:



Einzelheiten sind in Kapitel 6 bei den jeweiligen Anweisungen beschrieben.

Satzzeiger und Transaktionen

Bei der Verwendung von Satzzeigern müssen Sie die Zusammenhänge mit Transaktionen (siehe Abschnitt 2.10, *Transaktionssicherung*) berücksichtigen. Die Art des Satzzeigers hat Auswirkungen darauf, wann Sie einen Satzzeiger öffnen können und wann er wieder automatisch geschlossen wird. In Kapitel 6 sind bei jeder SQL-Anweisung, die Satzzeiger verwendet, die Möglichkeiten im einzelnen angegeben.

2.13.4 Dynamisch formulierte Anweisung

Eine dynamisch formulierte Anweisung besteht aus einer Anweisung oder mehreren Anweisungen, die zur Übersetzungszeit noch nicht bekannt sein müssen. Eine dynamisch formulierte Anweisung muß erst zur Laufzeit vorhanden sein.

Sie können auch einen Select-Satzzeiger für eine dynamische SELECT-Anweisung zuweisen.

Eine dynamisch formulierte Anweisung muß vorbereitet und getrennt davon ausgeführt werden.

Dynamisch formulierte Anweisung vorbereiten

Mit PREPARE bereiten Sie eine dynamisch formulierte Anweisung vor. Sie definieren einen Namen für die dynamisch formulierte Anweisung, genannt Anweisungsbezeichner, unter dem sie bei der späteren Verwendung (DECLARE, EXECUTE) angesprochen werden kann.

Für die noch nicht bekannten SQL-Anweisungen, für die der Anweisungsbezeichner steht, geben Sie eine alphanumerische Hostvariable an. Dieser Hostvariable weisen Sie dann im Programm die gewünschten SQL-Anweisungen als Zeichenkette zu. Sie können zum Beispiel die SQL-Anweisungen über ein Dialogprogramm einlesen und daraus die Zeichenkette zusammenbauen, die mit der Hostvariablen übergeben wird.

Sie können die SQL-Anweisungen bereits vollständig als konstante Zeichenkette angeben. Dieser Fall ist nicht so interessant, da die Anweisungen nicht dynamisch zusammengestellt werden.

Wird PREPARE ausgeführt, muß die dynamisch formulierte Anweisung bekannt sein. Ist sie nicht korrekt, wird mit Fehler abgebrochen.

Dynamisch formulierte Anweisung ausführen

Eine dynamisch formulierte Anweisung für eine SELECT-Anweisung, wird bei der ersten FETCH-Anweisung ausgeführt, nachdem der zugehörige Satzzeiger mit OPEN geöffnet wurde.

Jede andere dynamisch formulierte Anweisung muß mit EXECUTE ausgeführt werden. Bei ESQL/C gibt es auch die Möglichkeit, eine dynamische Anweisung direkt mit EXECUTE IMMEDIATE vorzubereiten und auszuführen.

Zum Ausführungszeitpunkt müssen alle benötigten Werte bekannt sein. Bei einer dynamischen Anweisung mit Satzzeiger können Werte für Platzhalter bei OPEN angegeben werden.

Bei einer mit EXECUTE ausgeführten Anweisung können Werte in der USING-Klausel der EXECUTE-Anweisung angegeben werden.

Speicherplatz freigeben

Die Betriebsmittel, die die vorbereitete Anweisung belegt, können mit der FREE-Anweisung wieder freigegeben werden.

2.13.5 Erfolgskontrolle

Um zu überprüfen, ob eine SQL-Anweisung erfolgreich war, wird von den Sprachschnittstellen die *sqlca*-Struktur bereitgestellt.

Sie wird von INFORMIX automatisch bei der Ausführung einer SQL-Anweisung belegt. Sie kann nach jeder SQL-Anweisung im Programm abgefragt werden.

Die *sqlca*-Struktur besteht aus mehreren Komponenten. Name und Anordung der Komponenten ist für jede Sprache unterschiedlich. Sie ist daher in den INFORMIX-Handbüchern für die Programmeinbettung für jede Programmiersprache beschrieben.

Die *sqlcode*-Komponente (ESQL/C) gibt Auskunft darüber, ob eine SQL-Anweisung erfolgreich ausgeführt wurde oder nicht:

```
sqlcode = 0    die Anweisung wurde erfolgreich ausgeführt
sqlcode < 0    die Anweisung wurde nicht erfolgreich ausgeführt
```

Im Erfolgsfall kann die *sqlca*-Struktur abhängig von der Anweisung zusätzliche Informationen enthalten (zum Beispiel die Anzahl der tatsächlich eingefügten Sätze).

In Kapitel 6 sind bei den entsprechenden Anweisungen im Abschnitt *Erfolgskontrolle bei Programmeinbettung* die relevanten Komponenten, ihre Bedeutung und mögliche Belegung angegeben. Die *sqlcode*-Komponente (ESQL/C) bzw. STATUS (4GL) wird bei jeder Anweisung gesetzt und daher nicht gesondert für jede Anweisung beschrieben.

3 Lexikalische Elemente und Namen

3.1 Lexikalische Einheiten
3.2 Namen

Dieses Kapitel beschreibt zuerst, aus welchen lexikalischen Einheiten eine SQL-Anweisung zusammengesetzt ist. Anschließend sind die Namen beschrieben. In den nachfolgenden Kapiteln werden diese Definitionen als bekannt vorausgesetzt.

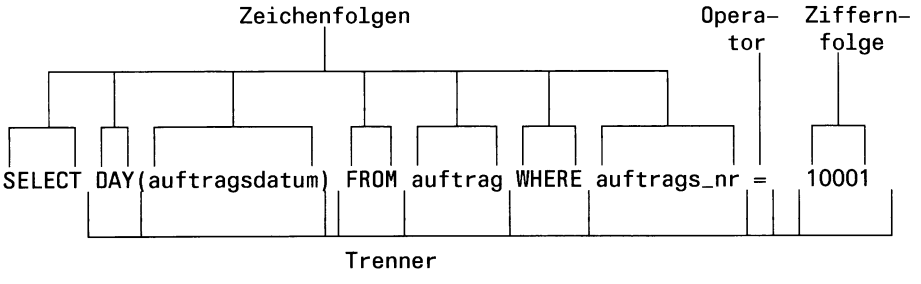
In den Syntaxdefinitionen werden für dieselben Parametertypen dieselben Bezeichnungen verwendet. Folgende Übersicht zeigt, welche Bezeichnungen in diesem Kapitel vorkommen, und wo Sie nähere Informationen zu dem jeweiligen Parametertyp finden.

Parameterbezeichnung	Bedeutung	Nähere Information
buchstabe	Buchstabe A–Z oder a–z	Keine
datenbank	Name einer Datenbank, eventuell qualifiziert	3.2.1, 3.2.5 und 2.1
datenbankname	Einfacher Datenbankname	3.2.1
eigner	Name eines Eigentümers	SINIX-Handbuch [12]
objname	Einfacher Name eines Datenbankobjekts	3.2.1 und 2.1
tabelle	Name einer Tabelle (Basistabelle, View, temporäre Tabelle) eventuell qualifiziert	3.2.1, 3.2.5 und 2.2
spaltenname	Einfacher Spaltenname	3.2.1 und 2.4
system	Name des Informixsystems	3.2.5 und ONLINE-Handbuch [10]
ziffer	Ziffer 0–9	Keine

Gibt es für einen Parameter Einschränkungen, ist dies bei der Beschreibung des Parameters angegeben; zum Beispiel, wenn für *tabelle* nur eine Basistabelle angegeben werden darf.

3.1 Lexikalische Einheiten

Eine SQL-Anweisung besteht aus den lexikalischen Einheiten Zeichenfolge, Ziffernfolge, Spezialzeichen und Trenner.



Zeichenfolgen

Es gibt Zeichenfolgen, die nicht in Anführungszeichen gesetzt werden, und Zeichenfolgen, die in Anführungszeichen gesetzt werden müssen.

Schlüsselwörter und Namen

Zeichenfolge, die nicht in Anführungszeichen eingeschlossen sind, bilden die SQL-Schlüsselwörter und die Namen.

Ein SQL-Schlüsselwort ist eine Folge aus Buchstaben, die groß oder klein geschrieben werden können. In diesem Handbuch sind alle SQL-Schlüsselwörter zur Unterscheidung groß geschrieben. *Beispiel*: SELECT

Eine Auflistung aller Schlüsselwörter finden Sie im Anhang A.5.

Die Namen und ihre Syntax sind im Abschnitt 3.2 erklärt.

Zeichenfolgen in Anführungszeichen

Zeichenfolgen in Anführungszeichen können sein:

- Namen, bei denen die Großschreibung beibehalten werden soll (siehe Abschnitt 3.2 *Namen*)
- Alphanumerische Konstanten (siehe Kapitel 4, Abschnitt 4.3.2 *Alphanumerische Werte*) *Beispiel*: "Maier"
- Zeitkonstanten (siehe Kapitel 4, Abschnitt 4.3.4 *Zeitwerte*)

  Beispiel: '10-4 4'

Bei Konstanten können zur Begrenzung statt Anführungszeichen " Hochkommas ' verwendet werden.

Zeichenketten

Als Oberbegriff für die Zeichenfolgen, die alphanumerische Werte oder Zeitwerte darstellen, wird in diesem Handbuch der Begriff **Zeichenkette** verwendet.

Konstante Zeichenketten werden in Anführungszeichen bzw. Hochkommas eingeschlossen (siehe oben).

Ziffernfolgen

Die Ziffernfolgen bilden die numerischen Konstanten.
Beispiel: 3.14

Die numerischen Konstanten sind mit ihrer Syntax in Kapitel 4, Abschnitt 4.3.3 *Numerische Werte* beschrieben.

Spezialzeichen

Zu den SQL-Spezialzeichen gehören Anführungszeichen " und Hochkomma ' zur Begrenzung von Zeichenfolgen sowie die Operatorzeichen:

( ) = + * / < >= > >= <> != ? % _ [] ^

Die Operatorzeichen werden in Ausdrücken und Prädikaten verwendet.
Beispiel: (3 + anzahl) * 12

Zum Teil haben sie verschiedene Bedeutungen, abhängig davon, in welchem Zusammenhang sie verwendet werden.

Sie sind in Kapitel 5 bei den Ausdrücken und Prädikaten erklärt.

Trenner

Trenner sind Leerzeichen, Tabulator- und Neue-Zeile-Zeichen. Die Operatorzeichen dienen auch als Trenner.

3.2 Namen

Namen sind Zeichenfolgen, die verwendet werden, um Objekte zu identifizieren. INFORMIX unterscheidet einfache Namen und qualifizierte Namen für die Identifikation gleichnamiger Objekte.

In SQL gibt es Namen für die Datenbank und folgende Datenbankobjekte:

- Datenbanken
- Tabellen
  - Basistabellen
  - Views
  - Synonyme
  - temporäre Tabellen
- Spalten
- Indizes
- Constraints

Bei Programmeinbettung gibt es zusätzlich Namen für:

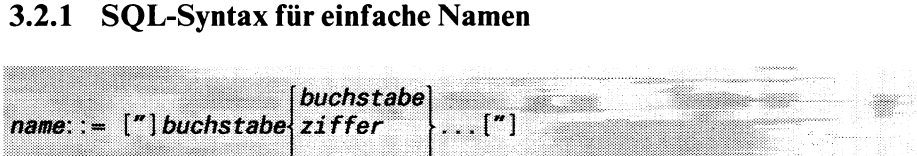
- Variablen:
  Die Namen für Variablen müssen den Konventionen der jeweiligen Programmiersprache entsprechen. Sie sind hier nicht weiter erklärt.
- Satzzeiger
- Dynamisch formulierte Anweisungen:
  Die Namen für dynamische Anweisungen werden in diesem Handbuch **Anweisungsbezeichner** genannt, um sie von den eigentlichen Namen der Anweisung, wie z.B. SELECT, zu unterscheiden.

Außerdem können in SQL-Anweisungen folgende Namen vorkommen:

- Dateinamen:
  Dateinamen werden nach den SINIX-Konventionen gebildet und in Anführungszeichen " eingeschlossen. Sie sind hier nicht weiter erklärt.
- Namen von Benutzern:
  Benutzernamen werden nach den SINIX-Konventionen gebildet. Sie sind hier nicht weiter erklärt.

– Namen für INFORMIX-ONLINE-Speicherbereiche (Dbspace, Blobspace):
Diese Namen werden vom INFORMIX-ONLINE-Verwalter, vergeben (siehe ONLINE-Handbuch [10]). Sie sind hier nicht weiter erklärt.

3.2.1 SQL-Syntax für einfache Namen



buchstabe
Kleinbuchstabe zwischen a und z oder Großbuchstabe zwischen A und Z.

Wenn Sie den Namen nicht in Anführungszeichen setzen, werden Großbuchstaben automatisch in Kleinbuchstaben umgewandelt.
Wenn Sie den Namen in Anführungszeichen einschließen, wird die angegebene Schreibweise beibehalten.

Einschränkung:
Schlüsselwörter sind reserviert. Sie sind nicht als Namen zugelassen (siehe Anhang A.5).

ziffer
Ziffer zwischen 0 und 9.

_ Unterstrich

Länge von Namen

Ein einfacher Namen darf maximal 18 Zeichen lang sein. Bei Namen in Anführungszeichen werden die Anführungszeichen nicht mitgezählt.

Bei Programmeinbettung kann es abhängig von der Programmiersprache noch zusätzlich Einschränkungen geben, zum Beispiel, daß nur die ersten 8 Zeichen signifikant sind.

3.2.2 Name definieren

Der Name eines Objekts wird bei der Definition des Objekts in der entsprechenden SQL-Anweisung festgelegt. Damit ist der Name eingeführt und das Objekt kann in allen weiteren Anweisungen mit diesem Namen angesprochen werden. In der folgenden Tabelle sind diese Anweisungen für die Datenbankobjekte zusammengestellt:

Datenbankobjekt	SQL-Anweisung
Datenbank	CREATE DATABASE

Tabelle
  Basistabelle
  View
  Synonym
  temporäre Tabelle
	
	CREATE TABLE
	CREATE VIEW
	CREATE SYNONYM
	CREATE TEMP TABLE
	SELECT INTO TEMP

Spalte	CREATE TABLE, ALTER TABLE CREATE VIEW, SELECT
Index	CREATE INDEX
Constraint	CREATE TABLE...UNIQUE CONSTRAINT, ALTER TABLE...ADD CONSTRAINT UNIQUE

Folgende Tabelle zeigt die Anweisungen für die Definition der Namen, die bei Programmeinbettung zusätzlich verwendet werden können:

Objekt	SQL-Anweisung
Dynamisch formulierte Anweisung	PREPARE
Satzzeiger	DECLARE

Zusätzliche Namen für Tabellen

Für Tabellen können zusätzlich zu dem Namen, der in der CREATE TABLE-Anweisung definiert wird, weitere Namen festgelegt werden, und zwar:

- Synonyme, die mit CREATE SYNONYM definiert werden.
  Synonyme können insbesondere dazu verwendet werden, kürzere oder sprechendere Namen für eine Tabelle zu definieren.
- Referenzen, die in der SELECT-Anweisung definiert werden.
  Eine Referenz ist eine Umbennung der Tabelle, die nur für die Dauer der SELECT-Anweisung gilt. Sie wird benötigt, um Spalten eindeutig angeben zu können (nähere Beschreibung siehe Kapitel 6, SELECT/ FROM).

3.2.3 Name ändern

Die Namen von Basistabellen und Spalten können mit der RENAME-Anweisung geändert werden. Der alte Name wird dadurch automatisch gelöscht.

Bei einer Umbenennung muß an allen Stellen, an denen der alte Name verwendet wurde, der neue Name eingesetzt werden.

Bei einer Tabelle wird intern nicht der Name sondern eine Tabellennummer zur Identifizierung verwendet. Daher müssen Sie nur die Stellen ändern, in denen tatsächlich der Name steht.
Folgende Übersicht zeigt, an welchen Stellen Sie die Umbenennung einer Tabelle nachziehen müssen.

Verwendung des Tabellennamens	Umbennung nachziehen
Synonym	interne Tabelle: nein externe Tabelle: ja
View	ja
Index	nein
Audit-Protokoll	nein
Programme	ja

Folgende Übersicht zeigt, an welchen Stellen Sie die Umbenennung einer Spalte nachziehen müssen.

Verwendung des Spaltennamens	Umbenennung nachziehen
View	ja
Index	nein
Programme	ja

3.2.4 Eindeutigkeit von Namen

Namen müssen in festgelegten Bereichen eindeutig sein.

Die folgende Tabelle zeigt, in welchem Bereich die Namen der verschiedenen Datenbankobjekte eindeutig sein müssen:

Datenbankobjekt	Bereich
Datenbank	SE: Datenbankdateiverzeichnis ONLINE: INFORMIX-ONLINE-System
Tabelle Basistabelle View Synonym temporäre Tabelle	Nicht-ANSI: Unter allen Tabellennamen der Datenbank ANSI: Unter allen Tabellennamen eines Benutzers
Referenz	Anweisung (SELECT)
Spalte	Tabelle
Index	Datenbank
Constraint	Nicht-ANSI: Datenbank ANSI: Datenbankbenutzer

Für die Programmeinbettung gilt zusätzlich:

Die Anweisungsbezeichner und Satzzeigernamen müssen pro Quellprogrammdatei eindeutig sein.

3.2.5 Qualifizierung von Namen

Um in einer SQL-Anweisung gleichnamige Objekte eindeutig identifizieren zu können, gibt es die Möglichkeit, den Namen zu qualifizieren.

Folgende Qualifikationen sind möglich:

- Qualifikation mit dem Namen des Eigentümers
  für Basistabelle, Synonyme, View, Indizes und Constraints
- Qualifikation mit dem Namen der Tabelle für Spalten.
- Bei INFORMIX-ONLINE gibt es zusätzlich:
  Qualifikation mit dem Namen der Datenbank für externe Basistabellen, Views und Synonyme.

Bei Einsatz der INFORMIX-Netzprodukte (INFORMIX-NET, INFORMIX-STAR) gibt es zusätzlich:

- Qualifikation mit dem Namen des Informixsystem
  für Datenbanken eines anderen Informixsystems

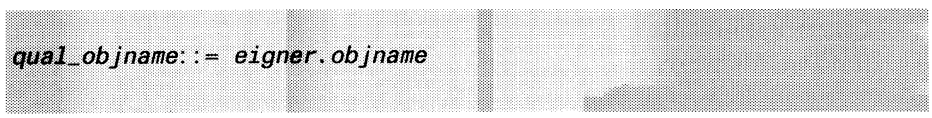
Qualifikation mit dem Eigentümernamen

Die Qualifikation mit dem Namen des Eigentümers ermöglicht es, Tabellen, Synonyme und Indizes eines anderen Benutzers eindeutig anzusprechen.

Bei einer ANSI-Datenbank ist die Qualifikation mit dem Eigentümernamen Pflicht, wenn Objekte eines anderen Benutzers angesprochen werden. Wird keine Qualifikation angegeben, erfolgt eine Fehlermeldung.

Bei einer Nicht-ANSI-Datenbank ist die Qualifikation optional, da die Objekte innerhalb der gesamten Datenbank eindeutig benannt werden müssen. Wenn eine Qualifikation angegeben ist, wird sie überprüft.

Um Ihre eigenen Objekte anzusprechen, müssen Sie keine Qualifikation angeben. Die Qualifikation mit dem eigenen Namen ist sowohl bei einer ANSI- als auch bei einer Nicht-ANSI-Datenbank optional.



eigner
Name des Eigentümers.
Der Eigentümer ist der Benutzer, der das Objekt entweder selbst erzeugt hat oder für den ein DBA das Objekt erzeugt hat.

Bei Systemtabellen heißt der Eigentümer *informix*.

objname
Einfacher Name des Datenbankobjekts.
Das Objekt kann eine Basistabelle, ein View, ein Synonym, ein Index oder ein Constraint sein.

Beispiel:

Die Benutzerin *sis* erzeugt den Index *ind1* für die Tabelle *auftrag* des Benutzers *lomata*.

```
CREATE INDEX ind1 ON lomata.auftrag (auftrags_nr)
```

Beachten Sie, daß der Index *sis* gehört, und folglich die qualifizierte Form *sis.ind1* heißt.

Qualifikation mit dem Tabellennamen

Spaltennamen müssen nur innerhalb einer Tabelle eindeutig sein. Die Qualifikation mit dem Namen der Tabelle ermöglicht es, Spalten eindeutig anzusprechen.

Der qualifizierte Spaltenname ist in der RENAME COLUMN-Anweisung erforderlich und in der SELECT-Anweisung erlaubt.
In den restlichen Anweisungen, in denen Spalten angegeben werden, wie zum Beispiel CREATE TABLE, ist nur der einfache Spaltenname erlaubt.

tabelle
Name der Tabelle, zu der die Spalte gehört.
Die Tabelle kann eine Basistabelle, ein View oder eine temporäre Tabelle sein. Sie können den einfachen Namen angeben und bei Basistabellen und Views auch ein Synonym oder eine qualifizierte Form (Eigentümer, Datenbank, Informixsystem).

spaltenname
Einfacher Name der Spalte.

Beispiel:

Artikelnummer aus den Tabellen *posten* und *artikel* der Beispieldatenbank *versand* abfragen:

```
SELECT posten.artikel_nr, artikel.artikel_nr
   FROM posten,artikel
```

Qualifikation mit dem Datenbanknamen

Nur INFORMIX-ONLINE

INFORMIX-ONLINE bietet die Möglichkeit, eine Tabelle aus einer anderen Datenbank anzusprechen. Dazu müssen Sie den Tabellennamen mit dem Namen der Datenbank qualifizieren.

```
qual_tabellenname: := datenbank: tabelle
```

datenbank
Name der Datenbank, zu der die Tabelle gehört.
Wenn Sie fremde Datenbanken in einem anderen Informixsystem verwenden, können Sie auch einen qualifizierten Datenbanknamen angeben.

tabelle
Name einer Basistabelle oder eines View.
Sie können den einfachen Namen, ein Synonym bzw. die mit dem Eigentümer qualifizierte Form angeben.

Bei einem Synonym ist die Datenbankqualifikation nicht möglich, wenn das Synonym bereits für einen Namen mit Datenbankqualifikation definiert wurde.

Beispiel:

Es sollen alle Sätze aus der Tabelle *auftrag* des Eigentümers *lomata* ausgewählt werden.
Die Tabelle steht in der Datenbank *versand*. Für die Tabelle *auftrag* hat *lomata* das Synonym *auf* definiert:

```
SELECT * FROM versand:lomata.auf
```

Sie können auch zuerst ein Synonym für den qualifizierten Namen definieren und anschließend über das Synonym auf die Tabelle zugreifen:

```
CREATE SYNONYM vla FOR versand:lomata.auf
SELECT * FROM vla
```

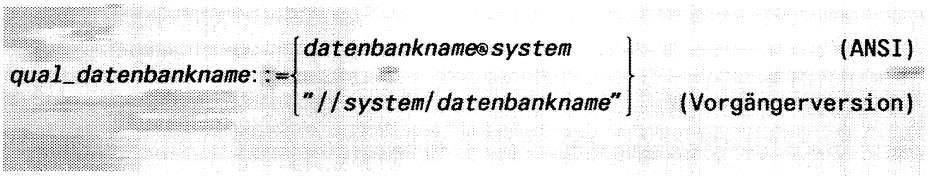
Qualifikation mit dem Informixsystemnamen

Nur INFORMIX-ONLINE plus INFORMIX-STAR oder
INFORMIX-SE plus INFORMIX-NET

Wenn Sie Datenbanken eines anderen Informixsystems verwenden, müssen Sie den Datenbanknamen mit dem Namen des Informixsystems qualifizieren.

Es gibt zwei Schreibweisen:

- ANSI-Schreibweise
- Schreibweise, die mit der Vorgängerversion kompatibel ist.



datenbankname
Einfacher Name der Datenbank.

system
Name des Informixsystems, zu dem die Datenbank gehört. Bei INFORMIX-SE ist dieser Name der Rechnername.
Bei INFORMIX-ONLINE wird dieser Name als Servername vom INFORMIX-ONLINE-Verwalter bei der Initialisierung vergeben.

Beispiel:

Datenbank *versand* im Informixsystem *muc* definieren:

```
CREATE DATABASE versand@muc

oder

CREATE DATABASE "//muc/versand/"
```

4 Datentypen und Werte

4.1 Überblick
4.2 Datentyp definieren
4.3 Werte verwenden

Dieses Kapitel hat zwei Hauptteile. Nach einem Überblick über die INFORMIX-Datentypen und die zugehörigen Wertebereiche werden im ersten Teil alle Einzelheiten erklärt, die Sie bezüglich Datentypen bei der Definition der Tabellenspalten wissen müssen:

- Syntax
- benötigter Speicherplatz
- Wertebereich, den der Datentyp festlegt.

Im zweiten Teil werden dann alle Einzelheiten erklärt, die Sie bei der Verwendung der Werte eines Datentyps kennen müssen:

- Syntax der Konstanten
- Regeln für das Eintragen der Werte in Tabellenspalten
- Regeln für die Verwendung der Werte in Berechnungen.

In den Syntaxdefinitionen werden für dieselben Parametertypen dieselben Bezeichnungen verwendet. Folgende Übersicht zeigt die wichtigen Bezeichnungen aus diesem Kapitel in alphabetischer Reihenfolge, und wo Sie nähere Informationen zu dem jeweiligen Parametertyp finden.

Parameterbezeichnung	Bedeutung	Nähere Information
ausdruck	Ausdruck	5.3
blobspace	Name eines Blobspace	ONLINE-Handbuch [10]
ganzzahl	Ganzzahl	4.3.3
komponente	Komponente eines Zeitwerts	4.2.4, 4.3.4
zeichen	Abdruckbares Einzelzeichen	Keine
zeichenkette	Folge aus einem oder mehreren Zeichen. (Alphanumerischer Wert, Zeitwert) Konstante Zeichenketten werden in Anführungs-zeichen " oder Hochkommas ' gesetzt.	3.1 und 4.3.2 und 4.3.4
ziffer	Ziffer 0-9	Keine

Gibt es für einen Parameter Einschränkungen, ist dies bei der Beschreibung des Parameters angegeben; zum Beispiel, wenn für *ganzzahl* nur eine Zahl von 1 bis 32 angegeben werden darf.

4.1 Überblick

Wie in Kapitel 2, Abschnitt 2.5 bereits beschrieben, sind die Werte bzw. Daten, die als Inhalte der Tabellen vorkommen können, in verschiedene Wertebereiche unterteilt.
Die Wertebereiche werden durch Datentypen festgelegt.

4.1.1 Einteilung der Datentypen

Die Datentypen, die INFORMIX zur Verfügung stellt, sind in folgende Gruppen eingeteilt:

- Alphanumerische Datentypen:
  - CHARACTER
  - VARCHAR (nur INFORMIX-ONLINE)
- Numerische Datentypen
  - Ganzzahlige Datentypen:
    - INTEGER
    - SMALLINT
    - SERIAL
  - Realzahl-Datentypen:
    - DECIMAL bzw. NUMERIC
    - MONEY
    - SMALLFLOAT bzw. REAL
    - FLOAT bzw. DOUBLE PRECISION
- Zeit-Datentypen:
  - DATE
  - DATETIME
  - INTERVAL
- BLOB-Datentypen (nur INFORMIX-ONLINE):
  - TEXT
  - BYTE.

4.1.2 Datentyp und Wertebereich

Jeder Datentyp definiert einen zugehörigen Wertebereich. Entsprechend den Haupt-Datentypgruppen gibt es alphanumerische Werte, numerische Werte, Zeitwerte und BLOB-Daten. Zusätzlich gibt es NULL-Werte. Jeder Datentyp außer SERIAL beinhaltet auch den NULL-Wert.

Die BLOB-Daten zählen nicht zu den Werten, da auf SQL-Basis keine Berechnungen mit oder Bearbeitung von BLOB-Daten möglich ist. Wie Sie BLOB-Spalten verwenden können, ist in Abschnitt 4.2 für jeden BLOB-Typ angegeben.

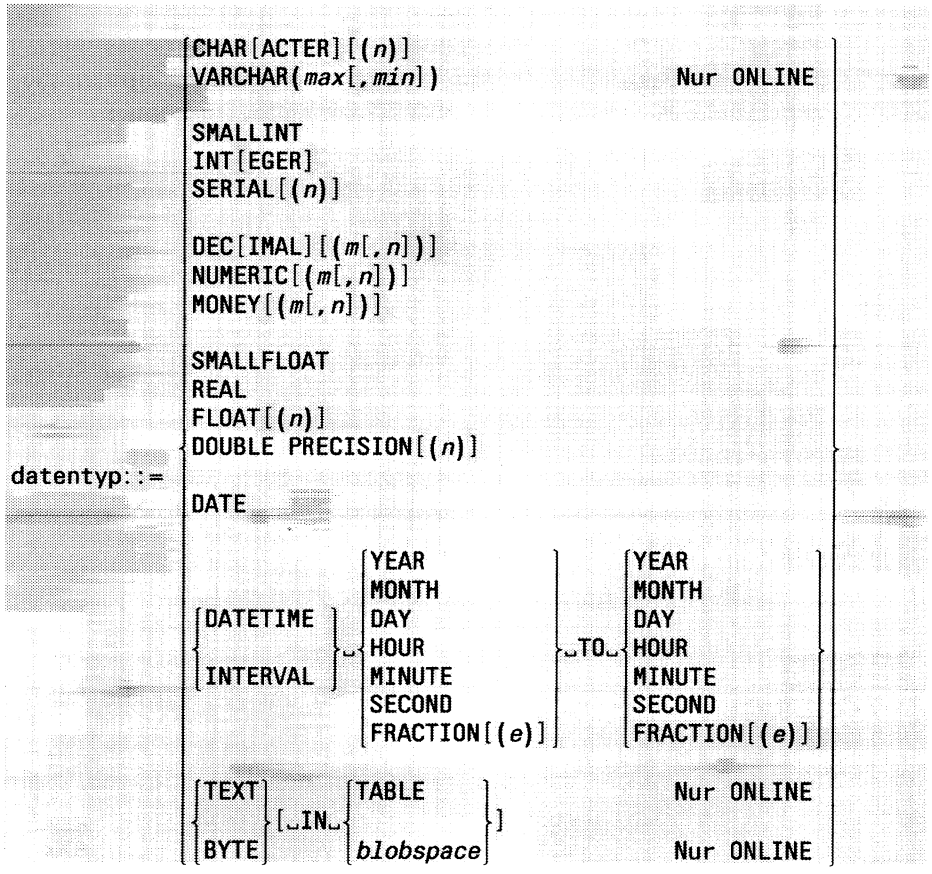
Für die eigentlichen Werte gibt es entsprechende Konstanten und Regeln, wie die Werte verwendet werden dürfen. Diese sind in Abschnitt 4.3 *Werte verwenden* beschrieben.

4.2 Datentyp definieren

Die Angabe des Datentyps ist bei der Definition einer Spalte erforderlich. Damit wird festgelegt, welche Werte die Spalte aufnehmen kann. Soll für eine Spalte der NULL-Wert ausgeschlossen werden, muß dies bei der Definition mit CREATE TABLE durch die Klausel NOT NULL angegeben werden (siehe Kapitel 6, CREATE TABLE).

In der folgenden Übersicht ist die Syntax für alle Datentypen zusammengestellt:

4.2.1 Übersicht über SQL-Syntax für Datentypen



Alles, was Sie bezüglich Datentypen bei der Spaltendefinition wissen müssen, ist in diesem Abschnitt 4.2 beschrieben. Die Datentypen sind in der Reihenfolge beschrieben, in der sie in der Übersicht aufgelistet sind.

4.2.2 Alphanumerische Datentypen

CHARACTER - Zeichenketten fester Länge

Der Datentyp CHARACTER wird für Spalten verwendet, die alphanumerische Werte (siehe Abschnitt 4.3.2) fester Länge enthalten können.

```
CHAR[ACTER][(n)]
```

n

Ganzzahl zwischen zwischen 1 und 32511, die die Länge der Spalte angibt.

n nicht angegeben:
Es gilt $n = 1$.

Speicherplatz für CHARACTER-Spalten

Für eine CHARACTER-Spalte werden *n* Bytes reserviert.

Wertbereich für CHARACTER-Spalten

Eine CHARACTER-Spalte kann Zeichenketten mit der für die Spalte festgelegten Länge enthalten.

Wenn Sie in eine CHARACTER-Spalte eine Zeichenkette eintragen, kann die Länge von der Spaltenlänge abweichen:

- Tragen Sie eine kürzere Zeichenkette ein, wird diese automatisch am Ende mit Leerzeichen aufgefüllt.
- Versuchen Sie eine längere Zeichenkette einzutragen, wird diese automatisch auf die Länge gekürzt.

Beispiel

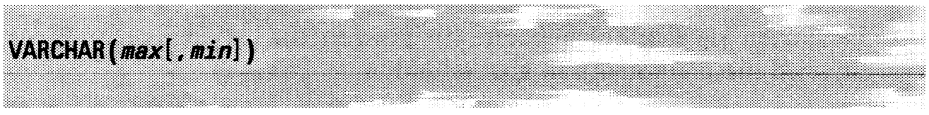
Die Tabelle *kunde* der Beispieldatenbank *versand* enthält 9 CHARACTER-Spalten unterschiedlicher Länge. Die Werte, die die Spalten enthalten können, sind Zeichenketten der Länge 2, 5 15, 18 bzw. 20. Beispiele dieser CHARACTER-Spalten sind:

```
vorname         CHAR(15)
nachname        CHAR(15)
plz             CHAR(5)
telefon         CHAR(18)
```

VARCHAR - Zeichenketten variabler Länge

Nur INFORMIX-ONLINE

Der Datentyp VARCHAR wird für Spalten verwendet, die alphanumerische Werte (siehe Abschnitt 4.3.2) variabler Länge enthalten können.



max
Ganzzahl zwischen zwischen 1 und 255, die die maximale Länge der VARCHAR-Spalte festlegt.

min
Ganzzahl zwischen zwischen 0 und 255, die die minimale Länge der VARCHAR-Spalte festlegt.

min nicht angegeben:
Es gilt $min = 0$.

Länge abfragen

Die Spalte *collength* der Systemtabelle *syscolumns* enthält für eine VARCHAR-Spalte die minimale und maximale Länge in folgender codierten Form:

```
(min * 256) + max
```

Speicherplatz für VARCHAR-Spalten

Für eine VARCHAR-Spalte werden nur $min + 1$ Bytes reserviert.

Wertebereich für VARCHAR-Spalten

Eine VARCHAR-Spalte kann Zeichenketten unterschiedlicher Länge enthalten. Die minimale Länge ist durch *min* festgelegt. Zeichenketten, die kürzer als *min* sind, werden wie bei CHARACTER automatisch am Ende mit Leerzeichen aufgefüllt.

Im Unterschied zu CHARACTER-Spalten gilt:

- Wird eine Zeichenkette in eine VARCHAR-Spalte eingetragen, die länger als *min* Bytes ist, wird erst beim Eintragen der erforderlich Platz besorgt.
- Zeichenketten, die länger als *min* sind, belegen immer nur so viele Bytes, wie sie tatsächlich benötigen.
- Versuchen Sie eine Zeichenkette einzutragen, die länger als *max* Bytes ist, wird diese automatisch abgeschnitten.

VARCHAR-Spalten ändern

Da VARCHAR-Spalten unterschiedlich lang werden können, kann sich nach einer UPDATE-Anweisung die neue Satzlänge von der vorherigen Satzlänge unterscheiden:

- Ist der geänderte Satz kürzer, wird der nicht mehr benötigte Speicherplatz freigegeben.
  Findet die Änderung in einer Transaktion statt, ist der freizugebende Speicherplatz noch gesperrt, bis die Transaktion beendet ist.
- Ist der geänderte Satz länger, besorgt INFORMIX-ONLINE automatisch den Platz, um den Satz abzuspeichern. Dabei ist sichergestellt, daß die Satzadresse, die Sie mit SELECT ROWID abfragen können, unverändert bleibt.

Minimale Länge festlegen

Die Bereitstellung des Platzes für die einzutragenden Zeichenketten kostet Zeit, da Sätze verschoben werden müssen, wenn der anfangs bereitgestellte Platz nicht ausreicht.

Daher ist es sinnvoll, eine minimale Länge festzulegen, wenn vorauszusehen ist, daß die anfangs eingetragenen Spaltenwerte später bis zu dieser Länge wachsen. In diesem Fall belegt INFORMIX bereits beim ersten Eintragen des Satzes in die Tabelle den erforderlichen Platz. Allerdings wird dann wie bei CHARACTER-Spalten bei kürzeren Zeichenketten Platz unnötigt belegt.

Beispiel

Tabelle *buecher* mit der VARCHAR-Spalte *titel* definieren:

```
CREATE TABLE buecher
(
   bestell_nr      INT,
   titel           VARCHAR(50,20)
)
```

Für die Spalte *titel* werden 20 Bytes reserviert. Jede Zeichenkette in *titel* hat somit mindestens die Länge 20. Wenn Sie einen kürzeren Titel eintragen, wird dieser mit Leerzeichen aufgefüllt. Wenn Sie einen längeren Titel eintragen, wird der benötigte Platz automatisch bereitgestellt:

```
INSERT INTO buecher VALUES (3456, "Morgen geht die Sonne auf")
INSERT INTO buecher VALUES (5777, "Kinderlieder")
INSERT INTO buecher
   VALUES (7888,
     "Das ist ein ueberlanger Titel mit mehr als fuenfzig Zeichen")
```

4.2.3 Numerische Datentypen

SMALLINT - Kleine Ganzzahlen

Der Datentyp SMALLINT wird für Spalten verwendet, die Ganzzahlen (siehe Abschnitt 4.3.3) aus dem Bereich von -32767 bis +32767 aufnehmen können.

```
SMALLINT
```

Speicherplatz für SMALLINT-Spalten

Für eine SMALLINT-Spalte werden 2 Bytes reserviert.

Wertebereich für SMALLINT-Spalten

Eine SMALLINT-Spalte kann Ganzzahlen aus dem Bereich von -32767 bis +32767 enthalten.

Beispiel

Die Tabelle *posten* der Beispieldatenbank *versand* hat drei SMALLINT-Spalten:

```
posten_nr       SMALLINT
artikel_nr      SMALLINT
menge           SMALLINT
```

INTEGER - Ganzzahlen

Der Datentyp INTEGER wird für Spalten verwendet, die Ganzzahlen (siehe Abschnitt 4.3.3) aus dem Bereich von -2147483647 bis +2147483647 aufnehmen können.

Speicherplatz für INTEGER-Spalten

Für eine INTEGER-Spalte werden 4 Bytes reserviert.

Wertebereich für INTEGER-Spalten

Eine INTEGER-Spalte kann Ganzzahlen aus dem Bereich von -2147483647 bis +2147483647 enthalten.

Beispiel

Die Tabelle *posten* der Beispieldatenbank *versand* hat eine INTEGER-Spalte:

```
auftrags_nr     INTEGER
```

SERIAL - Nummern

Der Datentyp SERIAL ist für eine ganzzahlige Spalte gedacht, in der die Tabellensätze fortlaufend numeriert werden.

Die Besonderheit von SERIAL besteht darin, daß INFORMIX automatisch einen Wert in eine SERIAL-Spalte einträgt, wenn kein Wert angegeben wird.

Wenn INFORMIX die Werte einträgt, werden die Sätze in der Reihenfolge, in der sie eingefügt werden, durchnumeriert.

Pro Tabelle ist nur eine SERIAL-Spalte erlaubt.

```
SERIAL[(n)]
```

n

Ganzzahl, die den Anfangswert für die automatische Numerierung angibt.
Die höchste Satznummer ist 2147483647.

n nicht angegeben:
INFORMIX startet mit dem Anfangswert 1.

Speicherplatz für SERIAL-Spalten

Für eine SERIAL-Spalte werden 4 Bytes reserviert.

Wertebereich für SERIAL-Spalten

Der Datentyp SERIAL enthält nicht den NULL-Wert. Eine SERIAL-Spalte kann daher keinen NULL-Wert enthalten. Ansonsten können die Werte in SERIAL-Spalten Ganzzahlen wie bei einer INTEGER-Spalte sein.

Wenn Sie beim Einfügen eines Satzes für eine SERIAL-Spalte keinen Wert oder 0 angeben, trägt INFORMIX automatisch einen Wert ein und zwar den bisher höchsten Wert plus 1.

Sie können die automatische Numerierung umgehen, indem Sie selbst einen Wert ungleich 0 eintragen. Sie können jede Ganzzahl wie bei INTEGER eintragen.
Sollen die Werte eindeutig sein, müssen Sie für die Eindeutigkeit sorgen, zum Beispiel, indem Sie einen Constraint oder eindeutigen Index definieren.

[!] Beachten Sie, daß Sie bei einem INSERT ohne Spaltenangabe für die SERIAL-Spalte einen Wert explizit angeben müssen. Wünschen Sie automatische Numerierung, geben Sie 0 an, ansonsten einen anderen Wert.

Besonderheiten von SERIAL-Spalten

Für SERIAL-Spalten gelten folgende Besonderheiten:

- Sie können keinen NULL-Wert enthalten.
- Sie können nicht geändert werden (UPDATE).

Beispiel

Die SERIAL-Spalte *kunden_nr* der Tabelle *kunde* der Beispieldatenbank *versand* ist wie folgt definiert:

```
CREATE TABLE  kunde
(
   kunden_nr        SERIAL(101)
   ...
)
```

Wenn Sie keinen Wert angeben, trägt INFORMIX einen Wert ein. Die Numerierung beginnt bei 101. Wenn Sie einen Wert angeben, wird dieser Wert eingetragen. Ist er höher als die bisherigen Werte, setzt INFORMIX die automatische Numerierung mit diesem Wert fort:

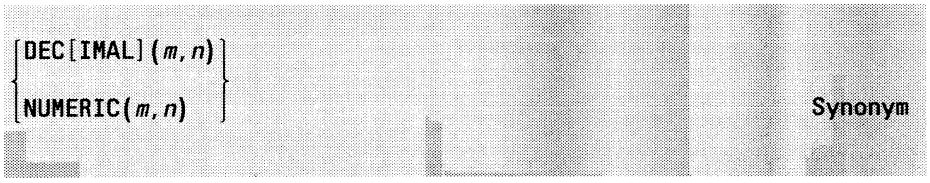
```
INSERT INTO kunde
  VALUES (1001, "Max","Rebel","Comet",NULL,NULL,"Muenchen",NULL,8000,"1234567");

INSERT INTO kunde
  VALUES (0,"Leo,"Huber","Comet",NULL,NULL,"Muenchen",NULL,8000,"2345678");
```

DECIMAL, NUMERIC - Festpunktzahlen

Für den Datentyp DECIMAL gibt es zwei Formate, ein Format für Festpunkt- und ein Format für Gleitpunktzahlen.

NUMERIC ist ein Synonym für DECIMAL.

Das folgende Format ist für Festpunktzahlen (siehe Abschnitt 4.3.3):



m

Ganzzahl zwischen 1 und 32, die die Gesamtanzahl der Dezimalstellen angibt.

n

Ganzzahl zwischen 0 und *m*, die die Anzahl der Nachkommastellen angibt.
Wenn Sie mehr als *n* Nachkommastellen angeben, rundet INFORMIX automatisch auf *n* Nachkommastellen.

Speicherplatz für DECIMAL-Festpunkt-Spalten

Die Anzahl der Bytes, die für eine DECIMAL-Spalte reserviert werden, ist abhängig davon, ob *m* und *n* gerade oder ungerade sind:

m	*n*	Spaltenlänge
gerade	gerade	1+*m*/2
gerade	ungerade	1+(*m*+2)/2
ungerade	gerade oder ungerade	1+(*m*+1)/2

Wertebereich für DECIMAL-Festpunkt-Spalten

Eine DECIMAL-Festpunkt-Spalte kann Festpunktzahlen enthalten, deren Betrag im Bereich von $0.5 * 10^{-m}$ bis $10^{n-m}-10^{-n}$ liegt.

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

Beispiel

Die Tabelle *auftrag* der Beispieldatenbank *versand* hat eine Festpunkt-Spalte:

```
liefergewicht  DECIMAL(8,2)
```

Sie können Festpunktzahlen angeben, die insgesamt 8 Dezimalstellen haben, wobei 2 Nachkommastellen vorhanden sein können.

```
INSERT INTO auftrag (liefergewicht) VALUES (345678,45)
```

MONEY - Festpunktzahlen als Geldbeträge

Der Datentyp MONEY ist für Spalten gedacht, in denen Geldbeträge stehen.
Intern entspricht MONEY dem Festpunkt-Datentyp DECIMAL.

```
MONEY[(m[,n])]
```

m

Ganzzahl zwischen 1 und 32, die die Gesamtanzahl der Dezimalstellen angibt.

n nicht angegeben:
Es gilt $n=2$.

m und *n* nicht angegeben:
Es gilt $m=16$, $n=2$.

Wertebereich für MONEY-Spalten

Eine MONEY-Spalte kann Festpunktzahlen aus demselben Bereich enthalten wie eine DECIMAL-Festpunkt-Spalte.

Besonderheit von MONEY-Spalten

Die Besonderheit von MONEY besteht darin, daß ein Ausgabeformat für die Werte von MONEY-Spalten festgelegt werden kann, sodaß diese Werte als Geldbeträge zu identifizieren sind (siehe Abschnitt 4.3.3).

Beispiel

Die Tabelle *auftrag* der Beispieldatenbank *versand* hat eine MONEY-Spalte:

```
zustellgebuehr   MONEY(6,2)
```

Sie können Festpunktzahlen angeben, die insgesamt 6 Dezimalstellen haben, wobei 2 Nachkommastellen vorhanden sein können.

```
INSERT INTO auftrag (zustellgebuehr) VALUES (345.67)
```

DECIMAL, NUMERIC - Gleitpunktzahlen

Für den Datentyp DECIMAL gibt es zwei Formate, ein Format für Festpunkt- und ein Format für Gleitpunktzahlen.

NUMERIC ist ein Synonym für DECIMAL.

Das folgende Format ist für Gleitpunktzahlen (siehe Abschnitt 4.3.3):

m

Ganzzahl zwischen 1 und 32, die die Anzahl der Dezimalstellen in der Mantisse angibt.

m nicht angegeben:
Es gilt $m = 16$.

Speicherplatz für DECIMAL-Gleitpunkt-Spalten

Für eine DECIMAL-Gleitpunkt-Spalte werden $1 + m/2$ Bytes reserviert.

Wertebereich für DECIMAL-Gleitpunkt-Spalten

Eine DECIMAL-Gleitpunkt-Spalte kann Gleitpunktzahlen enthalten, deren Betrag im Bereich von 10^{-128} bis 10^{126} liegt.

Beispiel

Tabelle *versuch1* mit einer Gleitpunktspalte definieren:

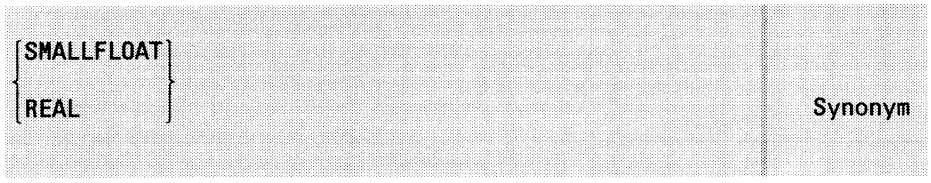
```
CREATE TABLE versuch1(gleitp DECIMAL(8));

INSERT INTO versuch1 VALUES (34E12);
INSERT INTO versuch1 VALUES (0.5E-32)
```

SMALLFLOAT, REAL - Gleitpunktzahlen mit einfacher Genauigkeit

Der Datentyp SMALLFLOAT wird für Spalten verwendet, die Gleitpunktzahlen (siehe Abschnitt 4.3.3) mit einfacher Genauigkeit aufnehmen können. Die Genauigkeit beträgt 8 Dezimalstellen.

SMALLFLOAT entspricht dem Datentyp *float* in C.

REAL ist ein Synonym für SMALLFLOAT.



Speicherplatz für SMALLFLOAT-Spalten

Für eine SMALLFLOAT-Spalte werden 4 Bytes reserviert.

Wertebereich für SMALLFLOAT-Spalten

Eine SMALLFLOAT-Spalte kann Gleitpunktzahlen enthalten, deren Betrag im Bereich von $8{,}43 * 10^{-37}$ bis $3{,}37 * 10^{+38}$ liegt.

Beispiel

Tabelle *versuch2* mit einer SMALLFLOAT-Spalte definieren:

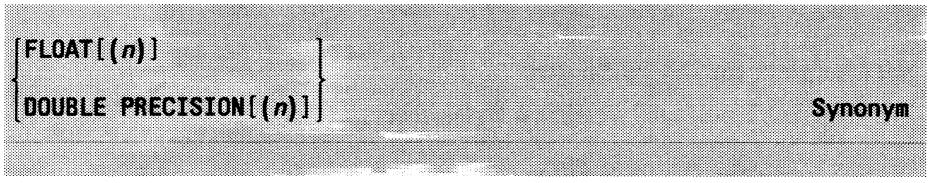
```
CREATE TABLE versuch2(sm SMALLFLOAT);

INSERT INTO versuch2 VALUES (34E4);
INSERT INTO versuch2 VALUES (0.5E-32)
```

FLOAT, DOUBLE PRECISION - Doppelte Genauigkeit

Der Datentyp FLOAT wird für Spalten verwendet, die Gleitpunktzahlen (siehe Abschnitt 4.3.3) mit doppelter Genauigkeit aufnehmen können. Die Genauigkeit beträgt 16 Dezimalstellen.

FLOAT entspricht dem Datentyp *double* in C.

DOUBLE PRECISION ist ein Synonym für FLOAT.



n

Ganzzahl zwischen 1 und 14, die die Genauigkeit angibt. Derzeit ohne Bedeutung.

Speicherplatz für FLOAT-Spalten

Für eine FLOAT-Spalte werden 8 Bytes reserviert.

Wertebereich für FLOAT-Spalten

Eine FLOAT-Spalte kann Gleitpunktzahlen enthalten, deren Betrag im Bereich von $4{,}19 * 10^{-307}$ bis $1{,}67 * 10^{+308}$ liegt.

Beispiel

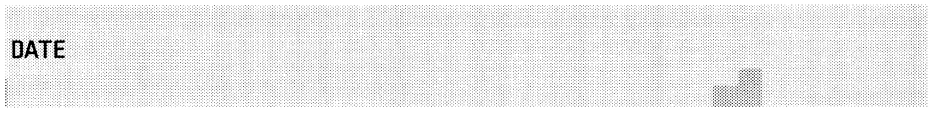
Tabelle *versuch3* mit einer FLOAT-Spalte definieren:

```
CREATE TABLE versuch3(fl FLOAT);

INSERT INTO versuch3 VALUES (34E206);
INSERT INTO versuch3 VALUES (0.5E-301)
```

4.2.4 Zeit-Datentypen

DATE - Datum

Der Datentyp DATE wird für Spalten verwendet, die ein Datum aufnehmen (siehe Abschnitt 4.3.4) können.



Speicherplatz für DATE-Spalten

Für eine DATE-Spalte werden 4 Bytes reserviert.

Wertebereich für DATE-Spalten

Ein Datum wird intern umgerechnet in die Anzahl Tage seit 31.12.1899. Der 1.1.1900 gilt als 1.

Eine DATE-Spalte kann daher Ganzzahlen enthalten wie eine INTEGER-Spalte.

Eine Datumskonstante, die Sie in eine DATE-Spalte eintragen, können Sie entweder als Ganzzahl angeben, die die Anzahl Tage seit 31.12.1899 angibt, oder als Zeichenkette im festgelegten Datumsformat. Zur Angabe des aktuellen Datums können Sie die Funktion TODAY verwenden. Die genaue Beschreibung finden Sie im Abschnitt 4.3, *Werte verwenden*.

Beispiel

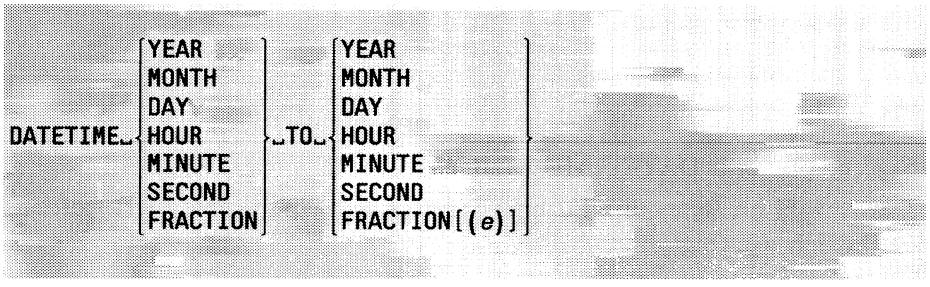
Die Tabelle *auftrag* der Beispieldatenbank *versand* enthält drei DATE-Spalten:

```
auftragsdatum   DATE
lieferdatum     DATE
zahldatum       DATE
```

DATETIME - Zeitpunkte

Der Datentyp DATETIME wird für Spalten verwendet, die Zeitpunkte (siehe Abschnitt 4.3.4) aufnehmen können.

Sie legen fest, welche Komponenten die Zeitpunkte enthalten. Sie geben die Anfangskomponente und die Endkomponente an. Die Anfangskomponente muß größer oder gleich der Endkomponente sein.



YEAR
: Die Jahrkomponente ist enthalten.
Sie ist eine vierstellige Ganzzahl, die Werte von 1 bis 9999 aufnehmen kann.

MONTH
: Die Monatkomponente ist enthalten.
Sie ist eine zweistellige Ganzzahl, die Werte von 1 bis 12 aufnehmen kann.

DAY
: Die Tagkomponente ist enthalten.
Sie ist eine zweistellige Ganzzahl, die Werte von 1 bis 31 aufnehmen kann.

HOUR
: Die Stundenkomponente ist enthalten.
Sie ist eine zweistellige Ganzzahl, die Werte von 0 bis 23 aufnehmen kann.

MINUTE
: Die Minutenkomponente ist enthalten.
Sie ist eine zweistellige Ganzzahl, die Werte von 0 bis 59 aufnehmen kann.

SECOND
Die Sekundenkomponente ist enthalten.
Sie ist eine zweistellige Ganzzahl, die Werte von 0 bis 59 aufnehmen kann.

FRACTION[(*e*)]
Die Sekundenbruchteilkomponente ist enthalten.

e
Ganzzahl zwischen 1 und 5, die die Anzahl der Stellen für die Komponente angibt.

e nicht angegeben:
Es gilt $e=3$.

Speicherplatz für DATETIME-Spalten

Die Anzahl an Bytes, die für eine DATETIME-Spalte reserviert werden, ergibt sich wie folgt:

Summe der Stellen aller vorhandenen Komponenten (eventuell aufgerundet auf die nächste gerade Zahl) geteilt durch 2 plus 1.

Wertebereich für DATETIME-Spalten

Die Werte in einer DATETIME-Spalte sind Zeitpunkte, die die Komponenten enthalten, die für die Spalte festgelegt wurden.

Eine Zeitpunkt-Konstante, die Sie in eine DATETIME-Spalte eintragen, können Sie entweder in der DATETIME-Schreibweise oder als Zeichenkette angeben. Bei der DATETIME-Schreibweise müssen Sie nicht alle Komponentenwerte angeben, allerdings immer eine zusammenhängende Folge von Komponenten. Zur Angabe des aktuellen Zeitpunkts können Sie die Funktion CURRENT verwenden. Die genaue Beschreibung finden Sie in Abschnitt 4.3, *Werte verwenden*.

Beispiel 1

Spalte vom Typ DATETIME definieren:

```
CREATE TABLE zeiten (zeitpunkt DATETIME MONTH TO HOUR)
```

Die Spalte *zeitpunkt* kann Zeitpunkte aufnehmen, die den Monat, den Tag und die Stunde angeben.

Beispiel 2

Das ist ein Fehler:
Die Anfangskomponente ist kleiner als die Endkomponente:

```
CREATE TABLE versuch1 (zeit DATETIME HOUR TO YEAR)
```

Richtig heißt es:

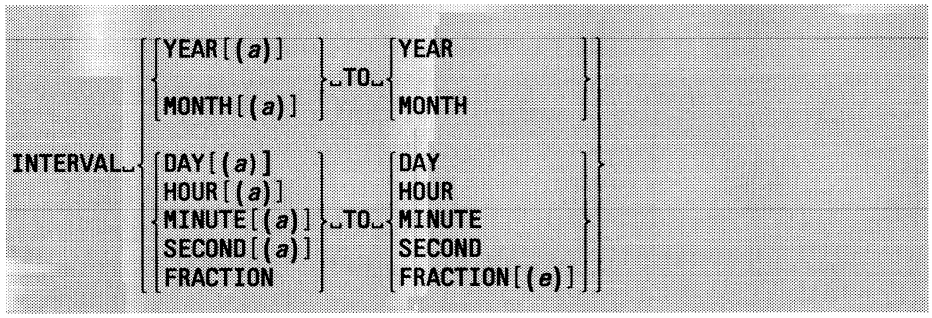
```
CREATE TABLE versuch1 (zeit DATETIME YEAR TO HOUR)
```

INTERVAL - Zeitspannen

Der Datentyp INTERVAL wird für Spalten verwendet, die Zeitspannen (siehe Abschnitt 4.3.4) aufnehmen können.

Sie legen fest, welche Komponenten die Zeitspannen enthalten. Sie geben die Anfangskomponente und die Endkomponente an. Die Anfangskomponente muß größer oder gleich der Endkomponente sein und es sind nur folgende Kombinationen erlaubt:

- Anfangs- und Endkomponente aus YEAR und MONTH oder
- Anfangs- und Endkomponente aus DAY, HOUR, MINUTE, SECOND und FRACTION.



YEAR[(*a*)]
Die Jahrkomponente ist enthalten.
Sie gibt eine Anzahl von Jahren an.

a
Nur erlaubt für Anfangskomponente!
Ganzzahl zwischen 1 und 9, die die Anzahl der Stellen für die Komponente angibt.

a nicht angegeben:
Es gilt $a = 4$.

MONTH[(*a*)]
Die Monatkomponente ist enthalten.
Sie gibt eine Anzahl von Monaten an.

a
Nur erlaubt für Anfangskomponente!
Ganzzahl zwischen 1 und 9, die die Anzahl der Stellen für die Komponente angibt.

a nicht angegeben:
Es gilt $a=2$.

DAY[(*a*)]
Die Tagkomponente ist enthalten.
Sie gibt eine Anzahl von Tagen an.

a
Nur erlaubt für Anfangskomponente!
Ganzzahl zwischen 1 und 9, die die Anzahl der Stellen für die Komponente angibt.

a nicht angegeben:
Es gilt $a=2$.

HOUR[(*a*)]
Die Stundenkomponente ist enthalten.
Sie gibt eine Anzahl von Stunden an.

a
Nur erlaubt für Anfangskomponente!
Ganzzahl zwischen 1 und 9, die die Anzahl der Stellen für die Komponente angibt.

a nicht angegeben:
Es gilt $a=2$.

MINUTE
Die Minutenkomponente ist enthalten.
Sie gibt eine Anzahl von Minuten an.

a
Nur erlaubt für Anfangskomponente!
Ganzzahl zwischen 1 und 9, die die Anzahl der Stellen für die Komponente angibt.

a nicht angegeben:
Es gilt $a=2$.

SECOND
Die Sekundenkomponente ist enthalten.
Sie gibt eine Anzahl von Sekunden an.

a
Nur erlaubt für Anfangskomponente!
Ganzzahl zwischen 1 und 9, die die Anzahl der Stellen für die Komponente angibt.

a nicht angegeben:
Es gilt $a=2$.

FRACTION[(*e*)]
Die Sekundenbruchteilkomponente ist enthalten.
Sie gibt eine Anzahl von Sekundenbruchteilen an.

e
Nur erlaubt für Endkomponente!
Ganzzahl zwischen 1 und 5, die die Anzahl der Stellen für die Komponente angibt.

e nicht angegeben:
Es gilt $e=3$.

Speicherplatz für INTERVAL-Spalten

Die Anzahl an Bytes, die für eine INTERVAL-Spalte reserviert werden, ergibt sich wie folgt:

Summe der Stellen aller vorhandenen Komponenten (eventuell aufgerundet auf die nächste gerade Zahl) geteilt durch 2 plus 1.

Wertebereich für INTERVAL-Spalten

Die Werte in einer INTERVAL-Spalte sind Zeitspannen, die zum Zeitspannentyp passen, der für die die Spalte festgelegt wurde (entweder Jahr/Monat-Typ oder Tag/Stunde/Minute/Sekunde/Bruchteile-Typ).

Eine Zeitspannen-Konstante, die Sie in eine INTERVAL-Spalte eintragen, können Sie entweder in der INTERVAL-Schreibweise oder mit UNITS oder als Zeichenkette angeben. Bei der INTERVAL-Schreibweise müssen Sie nicht alle Komponentenwerte angeben, allerdings immer eine zusammenhängende Folge von Komponenten. Die genaue Beschreibung finden Sie in Abschnitt 4.3, *Werte verwenden*.

Beispiel 1

Spalte vom Typ INTERVAL definieren:

```
CREATE TABLE wettlauf (teilnehmer CHAR(20), dauer INTERVAL HOUR TO FRACTION)
```

Die Spalte *dauer* kann Zeitspannen aufnehmen, die eine Anzahl von Stunden, Minuten, Sekunden und Sekundenbruchteilen angeben.

Beispiel 2

Das ist ein Fehler:
Die Endkomponente HOUR darf keine Stellenangabe enthalten.

```
CREATE TABLE versuch2 (stunden INTERVAL HOUR(5) TO HOUR(3))
```

Richtig heißt es:

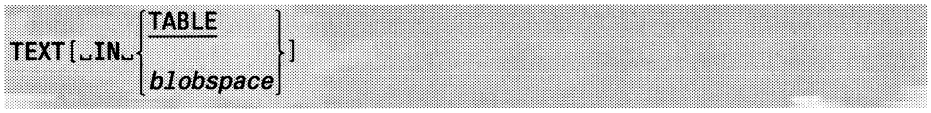
```
CREATE TABLE versuch2 (stunden INTERVAL HOUR(5) TO HOUR)
```

4.2.5 BLOB-Datentypen

TEXT - Texte

Nur INFORMIX-ONLINE

Der Datentyp TEXT wird für Spalten verwendet, die Texte beliebiger Länge aufnehmen können.



IN TABLE
Die BLOB-Daten der TEXT-Spalte werden im Tblspace abgelegt.
Wenn die Texte klein sind und ihre Größe nicht stark variiert, können Sie die Spalte im Tblspace abspeichern.

IN *blobspace*
Die BLOB-Daten der TEXT-Spalte werden im Blobspace *blobspace* abgelegt.
Der Blobspace ist ein vom Tblspace getrennter Bereich.

Sie sollten einen Blobspace angeben, dessen BLOB-Page-Größe für die zu erwartenden BLOB-Daten geeignet ist (siehe ONLINE-Handbuch [10]).

Wenn die Texte umfangreich werden können, sollten Sie die Spalte in einem Blobspace abspeichern.

IN-Klausel nicht angegeben:
Es gilt IN TABLE.

Speicherplatz für TEXT-Spalten

Tblspace und Blobspace werden vom INFORMIX-ONLINE-Verwalter definiert. Ein Blobspace besteht aus BLOB-Pages. Die Daten einer BLOB-Spalte im Blobspace werden auf BLOB-Pages aufgeteilt, die als Liste verkettet sind. Der INFORMIX-ONLINE-Verwalter legt die Größe der BLOB-Pages fest. Die BLOB-Page-Größe muß sinnvoll auf die möglich Größe der BLOB-Spalten abgestimmt sein, da sich die Anzahl der belegten BLOB-Pages auf die Anzahl der Sperren auswirkt. Wird ein Satz mit BLOB-Spalten gesperrt, werden für jede belegte BLOB-Page bei UPDATE 4 und ansonsten 2 Sperren beansprucht.
Nähere Beschreibung siehe ONLINE-Handbuch [10].

Wertebereich für TEXT-Spalten

Eine TEXT-Spalte kann Texte bis zu einer theoretischen Länge von 2^{31} enthalten. Ein Text ist eine Folge von abdruckbaren ASCII-Zeichen, in der folgende Steuerzeichen zur Formatierung vorkommen können:

- Tabulator-Zeichen
- Neue-Zeile-Zeichen
- Neue-Seite-Zeichen.

Beispiele sind Programm-Quelltexte und Dokumentationen.

TEXT-Spalten verwenden

Texte sind BLOB-Daten. Sie zählen nicht zu den SQL-Werten, da auf SQL-Basis keine Berechnungen mit oder Bearbeitung von Texten möglich sind.

Sie können BLOB-Daten in TEXT-Spalten eintragen, abfragen, ändern und löschen sowie eine TEXT-Spalte mit IS [NOT] NULL auf den NULL-Wert abfragen.

Eintragen

Die BLOB-Daten für eine TEXT-Spalte können Sie auf eine der folgenden Arten eintragen:

- Mit der LOAD-Anweisung.
  Dabei müssen Sie für jede Spalte (auch die letzte) ein Trennzeichen angeben, auch wenn für die Spalte kein Wert eingetragen wird. Der einzutragenden Text muß im ASCII-Code sein.
- Mit der INSERT-Anweisung, bei der die einzufügenden Daten als Ergebnistabelle einer SELECT-Anweisung geliefert werden.

Die Anweisungen sind ausführlich in Kapitel 6 beschrieben.

Pro BLOB-Page werden bei INSERT 2 Sperren benötigt.

Abfragen

Sie können die BLOB-Daten aus TEXT-Spalten mit SELECT abfragen. Dabei ist es möglich, durch Angabe einer Anfangs- und Endposition nur einen Ausschnitt der TEXT-Spalte auszuwählen:

```
SELECT text_spalte[a[,e]]      . . .
```

Beachten Sie, daß die äußeren eckigen Klammern keine Metasymbole sind, sondern angegeben werden müssen. Die inneren eckigen Klammern sind dagegen die üblichen Metasymbole für optionale Angabe.

a

Ganzzahl, die die Anfangsposition angibt, ab der ein Ausschnitt ausgewählt werden soll. Die Positionen in der Spalte sind beginnend mit 1 durchnumeriert.

e

Ganzzahl, die die Endposition angibt, bis zu der ein Ausschnitt ausgewählt werden soll.

e nicht angegeben:
Es wird nur das Zeichen an der Position *a* ausgewählt.

Ist die Isolationsstufe so eingestellt, daß beim Lesen Sperren gesetzt werden, werden pro BLOB-Page 2 Sperren benötigt.

Ändern

Sie können die BLOB-Daten in TEXT-Spalten mit einer UPDATE-Anweisung ändern, bei der die die einzutragenden BLOB-Daten als Ergebnistabelle einer SELECT-Anweisung geliefert werden.

Pro BLOB-Page werden 4 Sperren benötigt.

Löschen

Die BLOB-Daten einer TEXT-Spalte werden gelöscht, wenn der entsprechende Satz mit DELETE gelöscht wird.
Pro BLOB-Page werden 2 Sperren benötigt.

Programmeinbettung

Bei Programmeinbettung haben Sie zusätzlich die Möglichkeit, spezielle Hostvariablen für BLOB-Daten zu verwenden.
Über diese Hostvariablen können Sie BLOB-Daten bei INSERT und UPDATE bereitstellen und bei Abfragen übernehmen.

Die Realisierung von BLOB-Variablen ist bei den einzelnen Programmiersprachen unterschiedlich und in dem jeweiligen INFORMIX-Handbuch für diese Programmiersprache beschrieben.

Beispiel

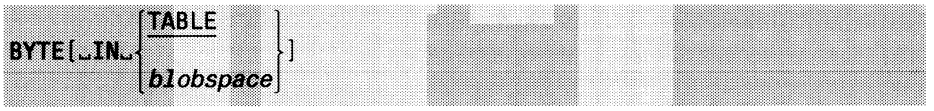
Spalte *doku* vom Typ TEXT definieren, deren Werte im Tblspace abgelegt werden sollen:

```
CREATE TABLE programme
(
   pname  CHAR(20),
   modul  BYTE IN blob1,
   doku   TEXT
)
```

BYTE - Binäre BLOB-Daten

Nur INFORMIX-ONLINE

Der Datentyp BYTE wird für Spalten verwendet, die binäre BLOB-Daten beliebiger Länge aufnehmen können.



IN TABLE
Die BLOB-Daten der BYTE-Spalte werden im Tblspace abgelegt.
Wenn die Byte-Folgen für die Spalte klein sind und ihre Größe nicht stark variiert, können Sie die Spalte im Tblspace abspeichern.

IN *blobspace*
Die BLOB-Daten der BYTE-Spalte werden im Blobspace *blobspace* abgelegt.
Der Blobspace ist ein vom Tblspace getrennter Bereich.

Sie sollten einen Blobspace angeben, dessen BLOB-Page-Größe für die zu erwartenden BLOB-Daten geeignet ist (siehe ONLINE-Handbuch [10]).

Wenn die Byte-Folgen für die Spalte groß werden können, sollten Sie die Spalte in einem Blobspace abspeichern.

IN-Klausel nicht angegeben:
Es gilt IN TABLE.

Speicherplatz für BYTE-Spalten

Tblspace und Blobspace werden vom INFORMIX-ONLINE-Verwalter definiert. Ein Blobspace besteht aus BLOB-Pages. Die Daten einer BLOB-Spalte im Blobspace werden auf BLOB-Pages aufgeteilt, die als Liste verkettet sind. Der INFORMIX-ONLINE-Verwalter legt die Größe der BLOB-Pages fest. Die BLOB-Page-Größe muß sinnvoll auf die möglich Größe der BLOB-Spalten abgestimmt sein, da sich die Anzahl der belegten BLOB-Pages auf die Anzahl der Sperren auswirkt. Wird ein Satz mit BLOB-Spalten gesperrt, werden für jede belegte BLOB-Page bei UPDATE 4 und ansonsten 2 Sperren beansprucht.

Nähere Beschreibung siehe ONLINE-Handbuch [10].

Wertebereich für BYTE-Spalten

Eine BLOB-Spalte kann Byte-Folgen bis zu einer theoretischen Länge von 2^{31} enthalten. Aus Sicht von INFORMIX haben sie keine weitere Struktur.

Beispiele sind Objektmodule von Programmen und digitalisierte Bilder.

BYTE-Spalten verwenden

BYTE-Daten sind BLOB-Daten. Sie zählen nicht zu den SQL-Werten, da auf SQL-Basis keine Berechnungen mit oder Bearbeitung von Byte-Folgen möglich ist.

Sie können BYTE-Daten in BYTE-Spalten eintragen, abfragen, ändern und löschen sowie eine BYTE-Spalte mit IS [NOT] NULL auf den NULL-Wert abfragen.

Eintragen

Die BLOB-Daten für eine BYTE-Spalte können Sie auf eine der folgenden Arten eintragen:

- Mit der LOAD-Anweisung.
  Dabei müssen Sie für jede Spalte (auch die letzte) ein Trennzeichen angeben, auch wenn für die Spalte kein Wert eingetragen wird.

  BLOB-Daten werden im Hexadezimal-Format angegeben.
- Mit der INSERT-Anweisung, bei der die einzufügenden Daten als Ergebnistabelle einer SELECT-Anweisung geliefert werden.

Die Anweisungen sind ausführlich in Kapitel 6 beschrieben.

Abfragen

Wenn Sie im interaktiven Betrieb BLOB-Daten aus BYTE-Spalten mit SELECT abfragen, wird lediglich eine Meldung ausgegeben, daß es sich um eine BYTE-Spalte handelt.

Ist die Isolationsstufe so eingestellt, daß beim Lesen Sperren gesetzt werden, werden pro BLOB-Page 2 Sperren benötigt.

Ändern

Sie können die BLOB-Daten in BYTE-Spalten mit einer UPDATE-Anweisung ändern, bei der die einzutragenden BLOB-Daten als Ergebnistabelle einer SELECT-Anweisung geliefert werden.

Löschen

Die BLOB-Daten einer BYTE-Spalte werden gelöscht, wenn der entsprechende Satz mit DELETE gelöscht wird.

Pro BLOB-Page werden zwei Sperren benötigt.

Programmeinbettung

Bei Programmeinbettung haben Sie zusätzlich die Möglichkeit, spezielle Hostvariablen für BLOB-Daten zu verwenden. Über diese Hostvariablen können Sie wie üblich BLOB-Daten bei INSERT und UPDATE bereitstellen und bei SELECT übernehmen.

Die Realisierung von BLOB-Variablen ist bei den einzelnen Programmiersprachen unterschiedlich und in dem jeweiligen INFORMIX-Handbuch für diese Programmiersprache beschrieben.

Beispiel

Spalte *modul* vom Typ BYTE definieren, deren Werte in dem Blobspace *blob1* abgelegt werden sollen:

```
CREATE TABLE programme
(
   pname  CHAR(20),
   modul  BYTE IN blob1,
   doku   TEXT
)
```

Datentyp-Konvertierung

Die Konvertierung eines Wertes von einem Ausgangsdatentyp in einen Zieldatentyp ist in folgenden Fällen erforderlich:

- In eine Spalte soll ein Wert eingetragen werden, der einen anderen Datentyp hat als die Spalte.
- Werte unterschiedlichen Datentyps sollen verglichen werden.
- Ein Ausdruck mit Werten unterschiedlichen Datentyps soll berechnet werden.

INFORMIX versucht soweit wie möglich die Werte zu konvertieren. Ist keine Konvertierung möglich, erfolgt eine Fehlermeldung.

Bei Programmeinbettung gibt es zusätzliche Regeln für die Konvertierung von INFORMIX-Datentypen und Datentypen der Hostvariablen. Außerdem haben Sie die Möglichkeit, mit Bibliotheksfunktionen explizite Typkonvertierungen vorzunehmen. Die Konvertierung ist sprachspezifisch und in dem jeweiligen Handbuch beschrieben.

4.3 Werte verwenden

Werte werden in SQL-Anweisungen angegeben, um:

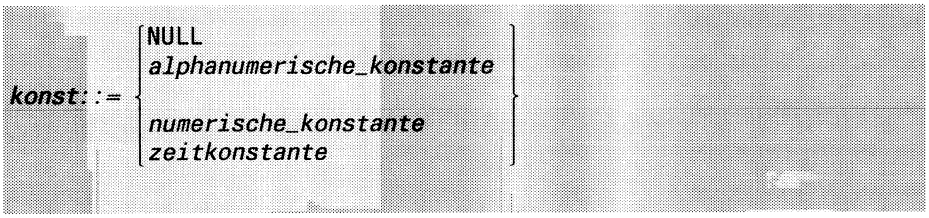
- Spaltenwerte einzutragen und zu ändern (INSERT, UPDATE)
- Berechnungen und Vergleiche durchzuführen (z.B. SELECT-Spaltenauswahl, WHERE- und HAVING-Bedingungen)

INFORMIX unterscheidet zwischen NULL-Werten und Nicht-NULL-Werten (siehe Kapitel 2, Abschnitt 2.5). Die Nicht-NULL-Werte sind entsprechend den Datentypen unterteilt. Die BLOB-Daten für BLOB-Spalten zählen nicht zu den Werten, da auf Basis von SQL keine Berechnungen mit BLOB-Daten durchgeführt werden können.

Es gibt daher folgende Gruppen von Werten:

- NULL-Werte
- Alphanumerische Werte
- Numerische Werte
- Zeitwerte.

Für jede dieser Gruppen gibt es entsprechende Konstanten, die in folgender Übersicht zusmmengestellt sind:



In diesem Abschnitt werden für jede der genannten Gruppen von Werten die Syntax für die zugehörigen Konstanten beschrieben und die Regeln erklärt, die bei der Verwendung der Werte berücksichtigt werden müssen. Die Gruppen sind nachfolgend in der oben angegebenen Reihenfolge beschrieben.

4.3.1 NULL-Wert

NULL-Werte sind eine Besonderheit von relationalen Datenbanken. Ein Null-Wert bedeutet undefiniert oder unbekannt.

Er ist von allen anderen Werten verschieden, insbesondere darf er nicht mit dem Leerzeichen oder der numerischen 0 verwechselt werden.

Konstante für NULL-Wert

Die Konstante für den NULL-Wert ist das Wort NULL. Die Konstante NULL darf nur beim Einfügen (INSERT) und Ändern (UPDATE) angeben werden, um einen Spaltenwert explizit auf den NULL-Wert zu setzen.

Beispiel:

In die Tabelle *artikel* einen Artikel eintragen, dessen Herstellercode unbekannt ist:

```
INSERT INTO artikel (5, NULL, "Pralinen",150.00,"Box","6/Box")
```

NULL-Wert in Tabellenspalten

Sie können den NULL-Wert für eine Spalte einer Basistabelle oder einer mit CREATE TEMP TABLE definierten Tabelle verbieten, indem Sie bei der Spaltendefinition NOT NULL angeben.

Wird der NULL-Wert bei der Definition nicht ausgeschlossen, kann die Spalte den NULL-Wert enthalten, sofern sie keine SERIAL-Spalte ist.

Ist für die Spalte eine Eindeutigkeitsbedingung mit einem eindeutigen Index (siehe Kapitel 2, Abschnitt 2.6 *Index*) oder einem Constraint (siehe Abschnitt 2.7, *Constraint*) definiert, darf der NULL-Wert höchstens einmal vorkommen.

Geben Sie beim Einfügen eines Satzes (INSERT) für eine Spalte, die NULL-Werte enthalten darf, keinen Wert an, trägt INFORMIX automatisch den NULL-Wert ein.

NULL-Wert in Berechnungen

Die Konstante NULL darf nicht in Berechnungen (Funktionen, Ausdrücke, Prädikate, Bedingungen) angegeben werden. Allerdings können Sie in Berechnungen Ausdrücke angeben (zum Beispiel einen Spaltennamen), die den NULL-Wert ergeben.

Kommt in einer Berechnung ein NULL-Wert vor, ist das Ergebnis in den meisten Fällen auch der NULL-Wert. Allerdings gibt es Ausnahmen wie zum Beispiel das Prädikat IS [NOT] NULL. In Kapitel 5 ist für jede Funktion, jeden Operator, jedes Prädikat und jede Bedingung das Ergebnis angegeben, wenn ein Operand der NULL-Wert ist.

Eine Bedingung, die den NULL-Wert ergibt, gilt als falsch.

NULL-Wert in GROUP BY

Bei der Gruppenbildung mit der GROUP BY-Klausel in einer SELECT-Anweisung werden alle Sätze, die in den Gruppierungsspalten den NULL-Wert enthalten, zu einer Gruppe zusammengefaßt.

NULL-Wert bei ORDER BY

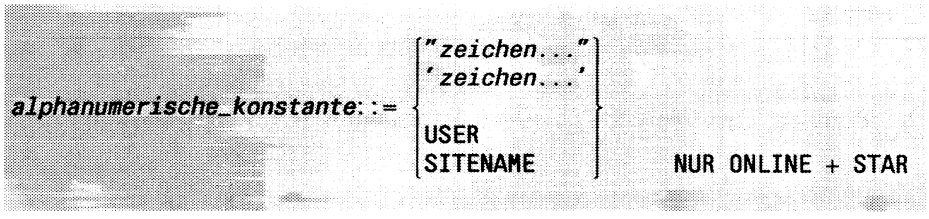
Beim Sortieren mit der ORDER BY-Klausel in einer SELECT-Anweisung sind NULL-Werte kleiner als andere Werte.

4.3.2 Alphanumerische Werte

Alphanumerische Werte sind Zeichenketten.

Alphanumerische Konstante

Die Syntax für eine alphanumerische Konstante ist wie folgt definiert:



zeichen
Abdruckbares Zeichen.

USER
Funktion, die den aktuellen Benutzernamen liefert. Sie können USER überall angeben, wo ein alphanumerischer Wert verlangt wird.

SITENAME
Nur INFORMIX-ONLINE plus INFORMIX-STAR
Funktion, die den Namen des aktuellen Informixsystems liefert.
Sie können SITENAME überall angeben, wo ein alphanumerischer Wert verlangt wird.

Alphanumerische Werte in Tabellenspalten

Sie können einen alphanumerischen Wert in eine CHARACTER- oder VARCHAR-Spalte (nur INFORMIX-ONLINE) eintragen.

Wenn Sie einen alphanumerischen Wert in eine CHARACTER-Spalte eintragen und die Länge von der Spaltenlänge abweicht, gilt:

- Ist die Zeichenkette kürzer, wird sie am Ende mit Leerzeichen aufgefüllt.
- Ist die Zeichenkette länger, wird sie am Ende auf die Spaltenlänge gekürzt.

Wenn Sie eine Zeichenkette in eine VARCHAR-Spalte eintragen, die länger als die minimale Spaltenlänge ist, wird der benötigte Platz besorgt. Allerdings werden höchstens so viele Bytes besorgt, wie maximal für die Spalte definiert wurde.

Die folgenden Beispiele beziehen sich auf die Tabelle *kunde* der Beispieldatenbank *versand*.

Beispiel 1

Vor- und Nachname einer Kundin eintragen:

```
INSERT INTO kunde (vorname, nachname)
  VALUES ("Hanna", "Gold")
```

Beispiel 2

Kundenname auf den aktuellen Benutzernamen ändern:

```
UPDATE kunde SET nachname=USER WHERE kunden_nr=101
```

Beispiel 3

Das ist ein Fehler:
Zeichenketten müssen in Hochkomma oder Anführungszeichen eingeschlossen werden.

```
INSERT INTO kunde (vorname, nachname)
  VALUES (Alfred, Schmiergel)
```

Richtig heißt es:

```
INSERT INTO kunde (vorname, nachname)
  VALUES ('Alfred', 'Schmiergel')
```

Alphanumerische Werte in Berechnungen

Alphanumerische Werte können verwendet werden in:

- Prädikaten
- Funktion LENGTH, die die Länge einer Zeichenkette bestimmt
- Funktion DATE, die eine Zeichenkette mit Datumsangaben in ein Datum umwandelt.

Zwei alphanumerische Werte werden von links nach rechts Zeichen für Zeichen verglichen.

Sind die beiden Werte unterschiedlich lang, wird die kürzere Zeichenkette rechts mit Leerzeichen aufgefüllt, so daß beide gleich lang sind.

Zwei Zeichenketten sind gleich, wenn sie an jeder Position das gleiche Zeichen haben.

Wenn zwei Zeichenketten nicht gleich sind, bestimmen die ersten beiden unterschiedlichen Zeichen, welche Zeichenkette größer bzw. kleiner ist. Ein Zeichen ist größer als ein anderes, wenn es einen höheren ASCII-Code hat. Beachten Sie, daß beim Abspeichern Ihrer Daten in der Datenbank die Codierung verwendet wird, die für Ihre Datensichtstation eingestellt ist.

Das kann sich insbesondere bei Umlauten auswirken, da diese bei verschiedenen Codes unterschiedlich codiert werden.

Beispiel

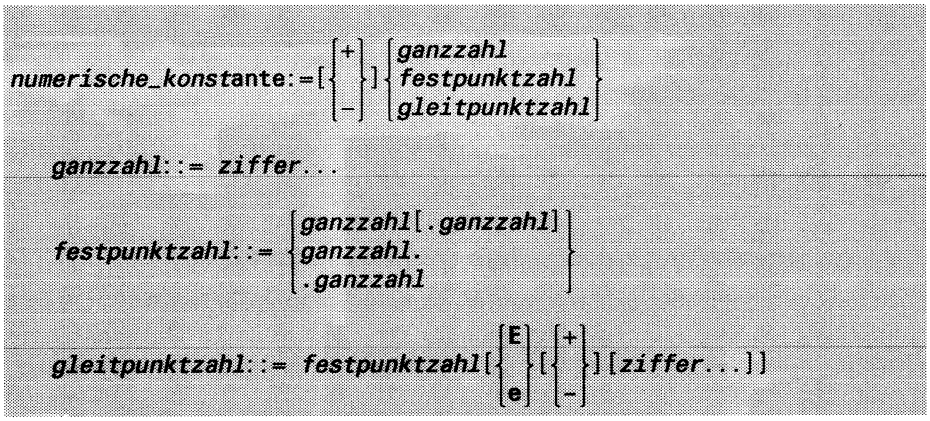
```
"Mai" < "Maier"       ist wahr
'Majer' < 'Maier'     ist falsch
```

4.3.3 Numerische Werte

Numerische Werte sind Ganzzahlen oder Realzahlen. Bei den Realzahlen unterscheidet man zwischen Festpunkt- und Gleitpunktzahlen.

Numerische Konstanten

Die Syntax für numerische Konstanten ist wie folgt definiert:



Numerische Werte in Tabellenspalten

Sie können einen numerischen Werte in eine Spalte mit numerischem Datentyp eintragen. Stimmen die numerischen Datentypen nicht überein, wird der Wert automatisch in den Datentyp der Spalte konvertiert.

Die folgenden Beispiele beziehen sich auf die Tabelle *posten* der Beispieldatenbank *versand*.

Beispiele:

Auftragsnummer eintragen:

```
INSERT INTO posten (auftrags_nr) VALUES (5000)
```

Auftragsmenge ändern.
Der angegebene Wert wird automatisch in eine Ganzzahl konvertiert:

```
UPDATE posten SET menge=34.75 WHERE auftrags_nr=1005
```

Das ist ein Fehler:
Der angegebene Wert ist nicht numerisch:

```
UPDATE posten SET menge="viel" WHERE auftrags_nr=1005
```

Numerische Werte in Berechnungen

Numerische Werte können in allen SQL-Prädikaten, -Ausdrücken und -Funktionen verwendet werden (siehe Kapitel 5).

Konvertierung

Wenn verschiedene numerische Datentypen in einer Berechnung vorkommen, konvertiert INFORMIX die Werte in den umfassendsten dieser Datentypen.

Wenn Sie statt einem numerischen Wert einen alphanumerischen Wert angeben, konvertiert INFORMIX diesen Wert. Stellt der alphanumerische Wert eine Zahl dar, ist das Ergebnis diese Zahl, ansonsten 0.

Geldbeträge

Ein Spezialfall von Festpunktzahlen sind Festpunktzahlen in MONEY-Spalten. Für diese kann mit der Umgebungsvariablen DBMONEY ein Ausgabeformat festgelegt werden, sodaß sie bei der Ausgabe als Geldbeträge zu erkennen sind.

Beispiel:

```
DBMONEY=DM.
export DBMONEY
```

Wenn Sie diese Anweisungen auf Shell-Ebene ausgeführt haben, werden Geldbeträge in INFORMIX mit DM und Punkt als Dezimalpunkt ausgegeben, zum Beispiel:

```
SELECT (gesamtpreis) FROM posten
```

```
gesamtpreis
  DM250.00
  DM960.00
  DM240.00
  ...
```

4.3.4 Zeitwerte

SQL unterscheidet folgende Arten von Zeitwerten:

- Datum
  Ein Datum enthält wie üblich die Angaben Tag, Monat und Jahr.
- Zeitpunkt
  Ein Zeitpunkt kann Datum und Uhrzeit enthalten.
- Zeitspanne
  Eine Zeitspanne gibt eine Anzahl von Jahren und Monaten bzw. Tagen, Stunden, Minuten und Sekunden an.

Die Syntax der zugehörigen Zeitkonstanten ist unterschiedlich und umfangreich. Außerdem werden die verschiedenen Zeitwerte unterschiedlich verwendet. Sie sind daher im folgenden einzeln erklärt.

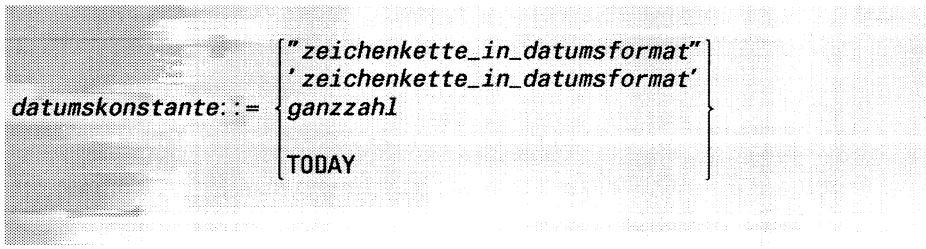
Datum

Ein Datum besteht aus den Angaben Tag, Monat und Jahr. Ein Datum wird intern umgerechnet in die Anzahl Tage seit 31.12.1899.

Für die Ein-/Ausgabe gibt es ein Datumsformat, sodaß ein Datum als Zeichenkette eingegeben und ausgegeben werden kann. Das Datumsformat kann mit der Umgebungsvariablen DBDATE eingestellt werden (siehe Anhang A.3).

Datumskonstante

Die Syntax für eine Datumskonstante ist wie folgt definiert:



zeichenkette_in_datumsformat
Zeichenkette, die folgende Angaben enthält:
tt, *mm* und [*jj*]*jj*.
Die Reihenfolge sowie das Trennzeichen zwischen den Angaben sind durch das Datumsformat festgelegt.
INFORMIX prüft bei dieser Darstellung, ob die Zeichenkette ein gültiges Datum ist.

tt
Ganzzahl zwischen 1 und 31 (passend zum Monat), die den Tag angibt.

mm
Ganzzahl zwischen 1 und 12, die den Monat angibt.

[*jj*]*jj*
Ganzzahl, die das Jahr angibt.
Wenn Sie nur eine zweistellige Zahl angeben, wird diese automatisch als Jahreszahl interpretiert und das Jahrhundert 19*jj* ergänzt.

ganzzahl
Ganzzahl, die die Anzahl der Tage seit dem 31.12.1899 angibt.
Der 1.1.1900 gilt als 1.
Ein Datum vor dem 31.12.1899 ist eine negative Ganzzahl. Ein Datum nach dem 31.12.1899 ist eine positive Ganzzahl.

TODAY
Funktion, die das aktuelle Datum liefert. Sie können TODAY überall angeben, wo als Wert ein Datum verlangt wird.

Datumsformat festlegen

Das Datumsformat für die Ein- und Ausgabe können Sie festlegen, indem Sie die Umgebungsvariable DBDATE setzen (siehe Anhang A.3).

Beispiel:

```
DBDATE=DMJ4-
export DBDATE
```

Wenn Sie diese Anweisungen auf Shell-Ebene ausführen, können Sie ein Datum wie folgt eingeben:

```
INSERT INTO auftrag (auftragsdatum) VALUES ("30-3-90")
```

Datum in Tabellenspalten

Ein Datum kann in eine Tabellenspalte vom Type DATE oder DATETIME eingetragen werden.

Die folgenden Beispiele beziehen sich auf die Tabelle *auftrag* der Beispieldatenbank *versand*.

Beispiele:

Aktuelles Datum in die Spalte *auftragsdatum* eintragen:

```
INSERT INTO auftrag (auftragsdatum) VALUES(TODAY)
```

Lieferdatum für Auftrag 1005 ändern:

```
UPDATE auftrag SET lieferdatum="1.10.90" WHERE auftrags_nr=1005
```

Das ist ein Fehler:
Der Wert 33 für einen Tag ist nicht erlaubt.

```
INSERT INTO auftrag (auftragsdatum) VALUES ("33.7.90")
```

Datum in Berechnungen

Ein Datum kann verwendet werden in:

- Vergleichen
  Ein Datum kann mit einem anderen Datum und mit einem Zeitpunkt verglichen werden. Für den Vergleich gilt:
  Ein Datum ist größer als ein anderes, wenn es jünger ist.
- Ausdrücken
  - Sie können zu einem Datum eine Ganzzahl oder eine Zeitspanne addieren oder subtrahieren. Das Ergebnis ist das entsprechend spätere bzw. frühere Datum.
  - Sie können von einem Datum ein anderes Datum subtrahieren. Das Ergebnis ist eine Ganzzahl, die die Differenz zwischen den Daten in Tagen angibt.
  - Sie können von einem Datum einen Zeitpunkt subtrahieren. Das Ergebnis ist die Zeitspanne zwischen den beiden Zeitpunkten.
- Datumsfunktionen
  INFORMIX stellt eine Reihe von Funktionen zur Verfügung, um ein Datum zu erstellen oder Teile eines Datums abzufragen.

Funktionen, Ausdrücke und Prädikate sind ausführlich in Kapitel 5 beschrieben.

Beispiele:

"12.10.90" < "1.1.92"	ergibt wahr
"24.4.90" + 10	ergibt "4.5.90"
"1.3.90" - "20.2.90"	ergibt 8
WEEKDAY("17.5.90")	liefert 4 für Donnerstag

Zeitpunkt

Mit einem Zeitpunkt kann ein Datum inklusive Uhrzeit angegeben werden. Er besteht aus einer Folge von Komponenten für die Angaben: Jahr, Monat, Tag, Stunde, Minute, Sekunde, Sekundenbruchteile.

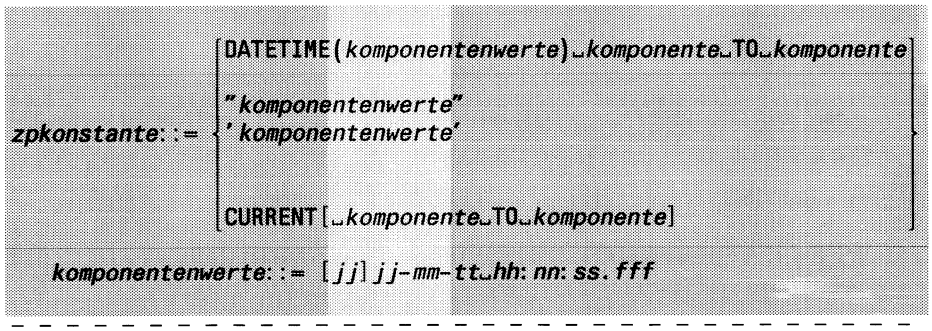
Ein Zeitpunkt kann alle oder nur Teile der Komponenten enthalten; es muß aber immer eine zusammenhängende Folge von Komponenten sein. Enthält ein Zeitpunkt alle Komponenten, gibt er ein vollständiges Datum mit Uhrzeit inklusive Sekundenbruchteilen an.
Enthält er nur einen Teil der Komponenten, gibt er nur die Werte für diese Komponenten an.

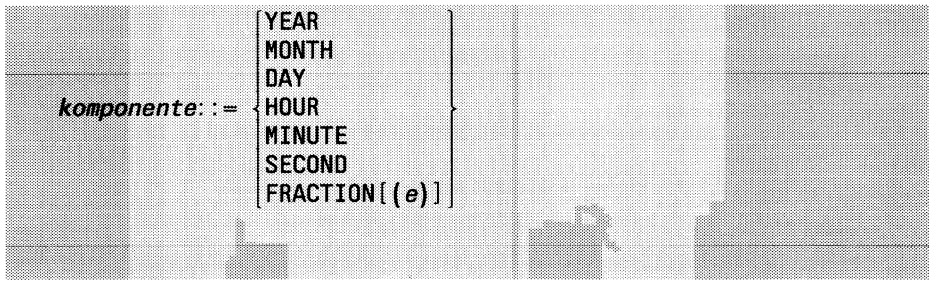
Beispiel: Enthält er nur die Tagkomponente, gibt er lediglich einen Tag an.

Zeitpunkt-Konstanten

Eine Zeitpunkt-Konstante wird angegeben, indem für ihre Komponenten die konkreten Werte aufgelistet werden. Es gibt zwei Schreibweisen für Zeitpunkt-Konstanten:

- Mit DATETIME sowie Anfangs- und Endkomponente
  Die Syntax für diese Schreibweise entspricht der Definition einer DATETIME-Spalte.
- Als Zeichenkette.





[*jj*]*jj*
: Jahr
Wenn Sie nur eine zweistellige Zahl angeben, wird diese als Jahreszahl interpretiert und das Jahrhundert 19*jj* automatisch ergänzt.
Wenn Sie eine zweistellige Jahreszahl angeben möchten, müssen Sie diese mit mindestens einer führenden Null angeben, zum Beispiel 089 für die Jahreszahl 89.

mm
: Monat.
Ganzzahl zwischen 1 und 12.

tt
: Tag
Ganzzahl zwischen 1 und 31 passend zum Monat.

hh
: Stunde
Ganzzahl zwischen 0 und 23.

nn
: Minute
Ganzzahl zwischen 0 und 59.

ss
: Sekunde
Ganzzahl zwischen 0 und 59.

fff
: Sekundenbruchteile als Ganzzahl
Die Anzahl der Stellen ist abhängig von der Angabe bei FRACTION.

CURRENT [*komponente* TO *komponente*]
: Funktion, die einen Zeitpunkt liefert, der das aktuelle Datum und die aktuelle Uhrzeit angibt. CURRENT hat die Komponenten YEAR TO FRACTION.
Sie können CURRENT überall angeben, wo als Wert ein Zeitpunkt verlangt wird.

[!] Die Trennzeichen zwischen den Komponentenwerten müssen genau eingehalten werden:

- Bindestrich - zwischen Jahr, Monat und Tag
- Leerzeichen ␣ zwischen Tag und Stunde
- Doppelpunkt : zwischen Stunde, Minute und Sekunde
- Punkt . zwischen Sekunde und Sekundenbruchteilen.

Zeitpunkte in Tabellenspalten

Zeitpunkte können in eine DATETIME-Spalte eingetragen werden.

Bei der Schreibweise als Zeichenkette müssen die Komponentenwerte für alle Komponenten angegeben werden, die für die Spalte festgelegt wurden.

Bei der Schreibweise mit DATETIME können Komponentenwerte am Anfang oder Ende fehlen. Es werden dann automatisch Standardwerte wie folgt eingesetzt:

- Fehlen Werte am Anfang, werden die entsprechenden Werte des aktuellen Datums bzw. der aktuellen Uhrzeit eingesetzt.

  Beispiel: Bei DATETIME (12 30) YEAR TO DAY ist das Jahr nicht angegeben. Es wird automatisch das aktuelle Jahr eingetragen.

- Fehlen Werte am Ende, werden Konstanten eingetragen, und zwar 1 bei Monat und Tag, 0 bei allen anderen Komponenten.

  Beispiel: Bei DATETIME (22:45) HOUR TO SECOND ist die Sekundenkomponente nicht angegeben. Sie wird automatisch auf 0 gesetzt.

Die folgenden Beispiele beziehen sich auf die Spalte *zeitpunkt* der Tabelle *zeiten*:

```
CREATE TABLE zeiten (zeitpunkt  DATETIME MONTH TO HOUR)
```

Beispiele:

In die Spalte *zeitpunkt* wird der Zeitpunkt 23 Uhr am 23.11. eingetragen:

```
INSERT INTO zeiten (zeitpunkt)
   VALUES (DATETIME (11-23 23) MONTH TO HOUR
```

In die Spalte *zeitpunkt* wird der Zeitpunkt 0 Uhr am 24. des aktuellen Monats eingetragen:

```
INSERT INTO zeiten (zeitpunkt) VALUES (DATETIME (24) DAY TO DAY)
```

In die Spalte *zeitpunkt* wird der Zeitpunkt 4 Uhr am 2.10. eingetragen:

```
INSERT INTO zeiten (zeitpunkt) VALUES("10-2 4")
```

Das ist ein Fehler:
Es fehlt der Monat, da bei einem Zeitpunkt, der als Zeichenkette angegeben wird, die Komponentenwerte vollständig vorhanden sein müssen.

```
INSERT INTO zeiten (zeitpunkt) VALUES("2 4")
```

Zeitpunkte in Berechnungen

Ein Zeitpunkt kann verwendet werden in:

- Vergleichen
  Ein Zeitpunkt kann mit einem anderen Zeitpunkt oder mit einem Datum verglichen werden, wobei gilt:
  Ein Zeitpunkt ist **größer** als ein anderer Zeitpunkt bzw. ein anderes Datum, wenn er **jünger** ist.
- Ausdrücken
  - Sie können von einem Zeitpunkt einen anderen Zeitpunkt oder ein Datum subtrahieren. Das Ergebnis ist eine Zeitspanne. Die Komponenten der Zeitspanne sind abhängig vom Ausgangstyp, angepaßt an die erlaubten Kombinationen für Zeitspannen.

    Beispiel: CURRENT - DATETIME(10-1) MONTH TO DAY ergibt z.B.: INTERVAL(-61 00:00:00.000) DAY TO FRACTION
  - Sie können zu einem Zeitpunkt eine Zeitspanne addieren oder von ihm subtrahieren.
    Sie müssen darauf achten, daß der Zeitpunkt zu dem Typ der Zeitspanne paßt, d.h. Komponenten enthält, die in der Zeitspanne vorkommen. Das Ergebnis ist der entsprechend spätere bzw. frühere Zeitpunkt. Er hat dieselben Komponenten wie der Ausgangszeitpunkt.
- Funktion EXTEND
  Sie dient dazu, Komponenten von Zeitpunkten zu ergänzen bzw. zu streichen.

Funktionen, Ausdrücke und Prädikate sind ausführlich in Kapitel 5 beschrieben.

Komponenten anpassen

Für alle Operationen, in denen Zeitpunkte erlaubt sind, gilt:

Enthalten Zeitpunkte (bzw. Daten) unterschiedliche Komponenten, werden sie implizit mit der Funktion EXTEND angeglichen, sodaß beide dieselben Komponenten haben:

- Für Komponentenwerte, die am Anfang ergänzt werden, wird der entsprechende Wert des aktuellen Datums bzw. der aktuellen Uhrzeit eingetragen.
- Für Komponentenwerte, die am Ende ergänzt werden, wird eine Konstante eingetragen, und zwar 1 bei Monat und Tag und 0 sonst.

Beispiele:

```
DATETIME(1990-5-17) YEAR TO DAY
< DATETIME(1990-5-30 20) YEAR TO HOUR
```

ergibt wahr.

```
DATETIME(1990-5) YEAR TO MONTH
- "1.6.1995"
```

ergibt:

```
INTERVAL(5-1) YEAR TO MONTH
```

```
EXTEND (DATETIME(1990-8-1) YEAR TO DAY, YEAR TO MINUTE)
- INTERVAL (720) MINUTE (3) TO MINUTE
```

ergibt:

```
DATETIME(1990-07-31 12:00) YEAR TO MINUTE
```

```
DAYTIME (1989-9-30 12:30) YEAR TO MINUTE
- CURRENT
```

ergibt (abhängig von aktuellem Datum und aktueller Uhrzeit):

```
INTERVAL (60 12:30) DAY TO MINUTE
```

```
CURRENT
- DAYTIME (10-1) MONTH TO DAY
```

ergibt (abhängig von aktuellem Datum und aktueller Uhrzeit):

```
INTERVAL (-60 00:00:00.000) DAY TO FRACTION
```

Zeitspanne

Eine Zeitspanne besteht wie ein Zeitpunkt aus einer Anzahl von Komponenten. Sie gibt im Unterschied zum Zeitpunkt eine Zeitdauer an.

Zeitspannentypen

Es gibt zwei Zeitspannentypen:

- Typ 1:
  Zeitspannen, die nur Komponenten aus Jahr und Monat enthalten.
- Typ 2:
  Zeitspannen, die nur Komponenten aus Tag, Stunden, Minuten, Sekunden und Sekundenbruchteilen enthalten.

Bei beiden Typen kann eine Zeitspanne alle oder nur Teile der erlaubten Komponenten enthalten; es muß aber immer eine zusammenhängende Folge von Komponenten sein.
Enthält sie nur einen Teil der Komponenten, gibt sie auch nur die Anzahl für diese Komponenten an.

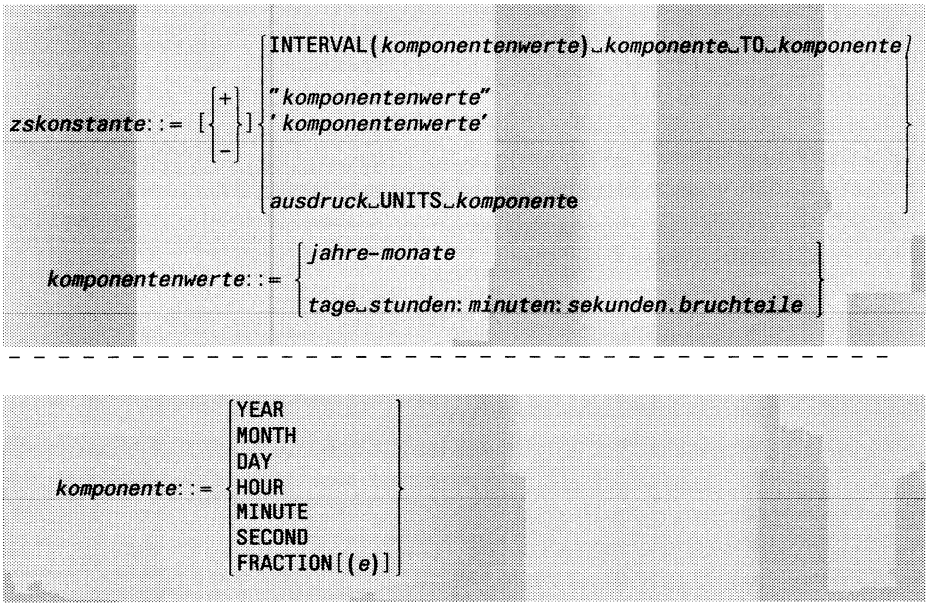
Beispiel:

Enthält sie nur die Tagkomponente, gibt sie eine Anzahl Tage an.

Zeitspannen-Konstante

Es gibt drei Möglichkeiten, Zeitspannen-Konstanten anzugeben:

- Mit INTERVAL und Anfangs-/Endkomponenten:
  Die Syntax für diese Schreibweise entspricht der Definition einer INTERVAL-Spalte.
- Als Zeichenkette
- Mit UNITS (Zeitspannen mit einer Komponente).



Bei der Schreibweise mit INTERVAL gilt wie beim Datentyp INTERVAL:

- Anfangs- und Endkomponente müssen entweder aus YEAR und MONTH oder aus DAY, HOUR, MINUTE, SECOND und FRACTION sein.
- Außer bei FRACTION kann für die Anfangskomponente eine Stellenzahl festgelegt werden.
- Bei FRACTION kann eine Stellenzahl festgelegt werden, wenn FRACTION Endkomponente ist.

jahre

Ganzzahl, die die Anzahl der Jahre angibt.
Die festgelegte Stellenzahl bestimmt, welche Werte Sie eintragen dürfen.
Standardmäßig sind 4 Stellen festgelegt und Sie können eine Zahl zwischen 0 und 9999 angeben.

monate

Ganzzahl, die die Anzahl der Monate angibt.
Wenn die Monatkomponente die Anfangskomponente ist, bestimmt die festgelegte Stellenzahl, welche Werte Sie eintragen dürfen.
Standardmäßig sind 2 Stellen festgelegt und Sie können eine Zahl zwischen 0 und 99 angeben.

Wenn die Monatkomponente nicht die Anfangskomponente ist, können Sie eine Zahl zwischen 1 und 12 angeben.

tage

Ganzzahl, die die Anzahl der Tage angibt.
Die festgelegte Stellenzahl bestimmt, welche Werte Sie eintragen dürfen.
Standardmäßig sind 2 Stellen festgelegt und Sie können eine Zahl zwischen 0 und 99 angeben.

stunden

Ganzzahl, die die Anzahl der Stunden angibt.
Wenn die Stundenkomponente die Anfangskomponente ist, bestimmt die festgelegte Stellenzahl, welche Werte Sie eintragen dürfen.
Standardmäßig sind 2 Stellen festgelegt und Sie können eine Zahl zwischen 0 und 99 angeben.

Wenn die Stundenkomponente nicht die Anfangskomponente ist, können Sie eine Zahl zwischen 0 und 23 angeben.

minuten

Ganzzahl, die die Anzahl der Minuten angibt.
Wenn die Minutenkomponente die Anfangskomponente ist, bestimmt die festgelegte Stellenzahl, welche Werte Sie eintragen dürfen.
Standardmäßig sind 2 Stellen festgelegt und Sie können eine Zahl zwischen 0 und 99 angeben.

Wenn die Minutenkomponente nicht die Anfangskomponente ist, können Sie eine Zahl zwischen 0 und 59 angeben.

sekunden
Ganzzahl, die die Anzahl der Sekunden angibt.
Wenn die Sekundenkomponente die Anfangskomponente ist, bestimmt die festgelegte Stellenzahl, welche Werte Sie eintragen dürfen.
Standardmäßig sind 2 Stellen festgelegt und Sie können eine Zahl zwischen 0 und 99 angeben.

Wenn die Sekundenkomponente nicht die Anfangskomponente ist, können Sie eine Zahl zwischen 0 und 59 angeben.

bruchteile
Ganzzahl, die die Sekundenbruchteile angibt.
Die möglichen Werte sind abhängig von der festgelegten Stellenanzahl.

ausdruck UNITS *komponente*
Zeitspanne mit nur einer Komponente *komponente*.
ausdruck ist ein Ausdruck, aus dem der Wert für die Komponente *komponente* berechnet wird. Der Ausdruck wird ausgewertet. Das Ergebnis wird, wenn nötig, in eine Ganzzahl konvertiert und ergibt den Komponentenwert. Die Stellenanzahl wird automatisch dem berechneten Wert angepaßt. Sie wird nicht angegeben.

Beispiel: Zeitspanne mit der Komponente Monat, die als Wert 15 Monate hat: 15 UNITS MONTH oder (3*5) UNITS MONTH.

Die Trennzeichen zwischen den Komponentenwerten müssen genau eingehalten werden:

- Bindestrich - zwischen Jahr, Monat und Tag
- Leerzeichen ␣ zwischen Tag und Stunde
- Doppelpunkt : zwischen Stunde, Minute und Sekunde
- Punkt . zwischen Sekunde und Sekundenbruchteilen

Zeitspanne in Tabellenspalten

Eine Zeitspanne kann in INTERVAL-Spalten desselben Typs eingetragen werden.

Bei der Schreibweise als Zeichenkette müssen die Komponentenwerte für alle Komponenten angegeben werden, die für die Spalte festgelegt wurden.

Bei der Schreibweise mit INTERVAL können Komponentenwerte am Anfang oder Ende fehlen. Für einen fehlenden Komponentenwert wird dann automatisch 0 eingesetzt.

Die folgenden Beispiele beziehen sich auf die Spalte *dauer* der Tabelle *wettlauf*:

```
dauer  INTERVAL HOUR(3) TO FRACTION
```

Beispiele:

In die Spalte *dauer* die Zeitspanne 27 Stunden eintragen:

```
INSERT INTO wettlauf (dauer) VALUES (INTERVAL (27) HOUR TO HOUR)
```

In die Spalte *dauer* die Zeitspanne 27 Stunden, 30 Minuten, 20 Sekunden und 12 Bruchteile eintragen:

```
INSERT INTO wettlauf (dauer)
  VALUES (INTERVAL(27:30:20.12) HOUR TO FRACTION)
```

In die Spalte *dauer* dieselbe Zeitspanne als Zeichenkette eintragen:

```
INSERT INTO wettlauf (dauer) VALUES ("27:30:20.12")
```

Zeitspanne in Berechnungen

Bei allen für Zeitspannen definierten Operationen müssen die Zeitspannentypen der Operanden zusammenpassen. Ansonsten ist die Operation nicht erlaubt.

Eine Zeitspanne kann verwendet werden in:

- Vergleichen
  Eine Zeitspanne kann mit einer anderen Zeitspanne vom gleichen Typ verglichen werden.
- Ausdrücken
  - Sie können zu einer Zeitspanne einen Zeitpunkt bzw. ein Datum addieren. Das Ergebnis ist der entsprechend spätere Zeitpunkt bzw. das spätere Datum.
  - Sie können von einem Zeitpunkt bzw. Datum eine Zeitspanne subtrahieren. Das Ergebnis ist der entsprechend frühere Zeitpunkt bzw. das frühere Datum.

- Sie können zu einer Zeitspanne eine andere Zeitspanne addieren oder von ihr subtrahieren. Das Ergebnis ist die entsprechend größere bzw. kleinere Zeitspanne. Das Ergebnis enthält alle Komponenten, die in den Operanden vorkommen.
- Sie können eine Zeitspanne mit einem numerischen Wert multiplizieren oder dividieren.
  Jeder Komponentenwert wird mit dem numerischen Wert multipliziert bzw. dividiert und der resultierende Wert wird ganzzahlig abgerundet. Das Ergebnis ist die Zeitspanne mit den neuen Komponentenwerten. Sie enthält alle Komponenten der Ausgangszeitspanne.

Bei allen Operationen werden die Komponentenwerte automatisch so berechnet, daß sie den oben angegebenen Bedingungen für Datums- und Uhrzeitangaben genügen.

Funktionen, Ausdrücke und Prädikate sind ausführlich in Kapitel 5 beschrieben.

Beispiele:

```
INTERVAL (5-3) YEAR TO MONTH
+ 15 UNITS MONTH
```

ergibt:

```
INTERVAL (6-06) YEAR TO MONTH
```

```
("2.5.1989" - "6.4.1954") UNITS DAY
```

ergibt:

```
INTERVAL (12810) DAY(5) TO DAY
```

```
EXTEND("2.5.1989", YEAR TO MONTH)
- "6.4.1954"
```

ergibt:

```
INTERVAL (35-01) YEAR TO MONTH
```

```
CURRENT
+ INTERVAL (3-5) YEAR TO MONTH
```

ergibt (abhängig von aktuellem Datum und der aktuellen Uhrzeit):

```
DATETIME(1993-01-01 00:00:00.000) YEAR TO FRACTION
```

5 Zusammengesetzte Sprachelemente

5.1 Funktionen
5.2 Unterabfragen
5.3 Ausdrücke
5.4 Prädikate
5.5 Bedingungen
5.6 Join

Dieses Kapitel beschreibt die zusammengesetzten Sprachelemente, die in SQL-Anweisungen vorkommen können.

Diese Sprachelemente setzen sich aus Grundelementen wie Namen und Konstanten und anderen Sprachelementen zusammen. Sie sind in der Reihenfolge beschrieben, in der sie aufeinander aufbauen.

In der Beschreibung der SQL-Anweisungen in Kapitel 6 werden die Sprachelemente als bekannt vorausgesetzt.

In den Syntaxdefinitionen werden für dieselben Parametertypen dieselben Bezeichnungen verwendet. Folgende Übersicht zeigt die wichtigen Bezeichnungen aus diesem Kapitel in alphabetischer Reihenfolge, und wo Sie nähere Informationen zu dem jeweiligen Parametertyp finden.

Parameterbezeichnung	Bedeutung	Nähere Information
ausdruck	Ausdruck	5.3
d	Datum	4.2.4, 4.3.4
funktionsaufruf	Auruf einer Funktion	5.1
ganzzahl	Ganzzahl	4.3.3
komponente	Komponente eines Zeitwerts	4.2.4, 4.3.4
konst	Konstante	4.3
muster	Suchmuster für Mustervergleich	5.4
praedikat	Prädikat	5.4
spalte	Name einer Spalte eventuell qualifiziert	3.2.1, 3.2.5 und 2.4
select-anweisung	SELECT-Anweisung	6, SELECT
unterabfrage	Unterabfrage	5.2
vergleichs_op	Vergleichsoperator	5.4
zeichen	Abdruckbares Einzelzeichen	Keine
zeichenkette	Folge aus einem oder mehreren Zeichen. (Alphanumerischer Wert, Zeitwert) Konstante Zeichenketten werden in Anführungs-zeichen " oder Hochkommas ' gesetzt.	3.1 und 4.3.2 und 4.3.4
ziffer	Ziffer 0-9	Keine
zeitpunkt	Zeitpunkt	4.2.4, 4.3.4

Gibt es für einen Parameter Einschränkungen, ist dies bei der Beschreibung des Parameters angegeben; zum Beispiel, wenn für *ganzzahl* nur eine Zahl von 1 bis 12 angegeben werden darf.

5.1 Funktionen

Funktionen berechnen einen Wert. Funktionsaufrufe können innerhalb von Ausdrücken vorkommen. Bei Auswertung eines Ausdrucks mit Funktionsaufrufen werden zuerst die Funktionen ausgeführt und dann die Funktionsaufrufe durch die berechneten Werte ersetzt.

Inhaltliche Zusammenstellung der SQL-Funktionen

Die SQL-Funktionen sind in folgende Gruppen eingeteilt:

- Zeichenkettenfunktionen
- Zeitfunktionen
- Mengenfunktionen.
  Mengenfunktion dürfen nur in der Spaltenauswahl und der HAVING-Bedingung einer SELECT-Anweisung vorkommen.

Zeichenkettenfunktionen

Funktion	Bedeutung
LENGTH()	Länge einer Zeichenkette abfragen
SITENAME	Aktuelles Informixsystem (nur ONLINE plus STAR)
USER	Aktueller Benutzername

Zeitfunktionen

Funktion	Bedeutung
CURRENT	Aktueller Zeitpunkt
DATE()	Datum erstellen
DAY()	Tag im Monat bestimmen
EXTEND()	Zeitpunkt anpassen
MDY()	Datum erstellen
MONTH()	Monat im Jahr bestimmen
TODAY	Aktuelles Datum
WEEKDAY()	Wochentag bestimmen
YEAR()	Jahreszahl bestimmen

Mengenfunktionen

Funktion	Bedeutung
AVG()	Durchschnittswert berechnen
COUNT(*)	Zeilen einer Ergebnistabelle zählen
COUNT()	Elemente einer Spalte zählen
MAX()	Maximum einer Spalte bestimmen
MIN()	Minimum einer Spalte bestimmen
SUM()	Summe einer Spalte berechnen

Funktionsargumente

Für jede Funktion ist festgelegt, von welchem Datentyp die Argumente sein müssen. In der Funktionsbeschreibung in diesem Kapitel sind für jede Funktion die erlaubten Argumenttypen angegeben. Im Funktionsaufruf dürfen Sie nur Argumente der jeweils festgelegten Datentypen angeben.

Für ein Argument können Sie angeben:

- Eine Nicht-NULL-Konstante vom passenden Datentyp
- Einen Ausdruck (siehe 5.3), der einen Wert vom passendem Argumenttyp liefert (z.B. einen Spaltennamen). Der Ausdruck wird zuerst ausgewertet und dann die Funktion auf den berechneten Wert angewendet. Bei Programmeinbettung kann der Wert auch über eine Hostvariable angegeben werden.

  Einschränkung:
  Mengenfunktionen dürfen nicht geschachtelt werden. Das heißt, bei einer Mengenfunktion dürfen Sie als Argument keinen Ausdruck angeben, der selbst wieder eine Mengenfunktion enthält.
- Die Konstante NULL als Funktionsargument ist nicht erlaubt. Allerdings kann ein Ausdruck, der als Argument angegeben wird, bei der Auswertung den NULL-Wert ergeben. In den Funktionsbeschreibungen ist jeweils angegeben, welchen Wert die Funktion in diesem Fall liefert.

AVG() - Arithmetisches Mittel

Funktionsgruppe: Mengenfunktion

AVG() berechnet das arithmetische Mittel aus einer Menge von Werten.

ALL
: Alle Werte werden berücksichtigt, auch solche die doppelt vorkommen.

DISTINCT
: Nur verschiedene Werte werden berücksichtigt. Duplikate werden ignoriert.

spalte
: Spalte.
Die Spalte muß numerisch oder vom Datentyp INTERVAL sein.

Addition und Division von Zeitspannen sind im Kapitel 4 Abschnitt 4.3.4 *Zeitwerte* beschrieben.

wert
: Numerischer Wert oder Zeitspanne.

Ergebnis

Datentyp: numerisch

- Bei Argument *spalte*

  Enthält die Spalte nur NULL-Werte, ist das Ergebnis der NULL-Wert. Ansonsten werden NULL-Werte ignoriert und das Ergebnis ist wie folgt:

  Ohne GROUP-BY-Klausel der SELECT-Anweisung:
  : Arithmetisches Mittel der Werte aus *spalte*.

  Mit GROUP-BY-Klausel der SELECT-Anweisung:
  : Pro Gruppe in der Ergebnistabelle, das arithmetische Mittel der Werte von *spalte* für diese Gruppe (siehe Beispiel).

- Bei Argument *wert*

  AVG() liefert *wert* zurück.

Beispiel

SELECT ohne GROUP BY:

Durchschnittspreis der Posten in der Tabelle *posten* der Beispieldatenbank *versand* berechnen:

```
SELECT AVG(gesamtpreis) FROM posten
```

```
   (avg)
DM306.10
```

Wenn Sie in die Tabelle einen Satz eintragen, dessen Spalte *gesamtpreis* der NULL-Wert ist, ändert sich das Ergebnis nicht.

SELECT mit GROUP BY:

Für jede Auftragsnummer wird der Durchschnittsgesamtpreis berechnet:

```
SELECT auftrags_nr, AVG(gesamtpreis)
FROM posten
GROUP BY (auftrags_nr)
```

```
auftrags_nr       (avg)
       1001    DM250.00
       1002    DM600.00
       1003    DM316.67
       1004    DM531.50
       . . .
```

COUNT(*) - Elemente zählen

Funktionsgruppe: Mengenfunktion

COUNT(*) zählt die Elemente einer Menge.

Ergebnis

Datentyp: Numerisch

Ohne GROUP-BY-Klausel der SELECT-Anweisung:
Anzahl der Sätze der Ergebnistabelle. Doppelte Sätze und Sätze, die nur NULL-Werte enthalten, werden mitgezählt.

Mit GROUP-BY-Klausel der SELECT-Anweisung:
Pro Gruppe in der Ergebnistabelle, die Anzahl der Elemente in dieser Gruppe (siehe Beispiel).

Beispiel

SELECT ohne GROUP BY:

Aus der Tabelle *kunde* der Beispieldatenbank *versand* abfragen, wieviele Kunden in München wohnen:

```
SELECT COUNT(*) FROM kunde WHERE ort="Muenchen"
```

```
(count(*))
         5
```

SELECT mit GROUP BY:

Für jeden Ort zählen, wieviele Kunden aus diesem Ort vorhanden sind:

```
SELECT ort, COUNT(*)
FROM kunde
GROUP BY (ort)
```

```
ort             (count(*))
Augsburg                 3
Wasserburg               1
Rosenheim                2
Muenchen                 5
        . . .
```

COUNT() - Verschiedene Elemente zählen

Funktionsgruppe: Mengenfunktion

COUNT(DISTINCT) zählt die verschiedenen Werte einer Menge. Duplikate und NULL-Werte werden nicht mitgezählt.

```
COUNT(DISTINCT␣spalte)
```

spalte
Spalte mit numerischem, alphanumerischem Datentyp oder Zeitdatentyp.

Einschränkung:
Sie dürfen keine Spalte vom Datentyp TEXT oder BYTE angeben.

Ergebnis

Datentyp: Numerisch

Ohne GROUP-BY-Klausel der SELECT-Anweisung:
Anzahl der verschiedenen Werte von *spalte*.

Mit GROUP-BY-Klausel der SELECT-Anweisung:
Pro Gruppe in der Ergebnistabelle, die Anzahl verschiedener Elemente in dieser Gruppe.

Enthält die Menge nur NULL-Werte, ist das Ergebnis 0.
Enthält die Menge Nicht-NULL-Werte, werden NULL-Werte nicht mitgezählt.

Beispiel

SELECT ohne GROUP BY:

Aus der Tabelle *artikel* der Beispieldatenbank *versand* die Anzahl verschiedener Artikelbezeichnungen bestimmen:

```
SELECT COUNT(DISTINCT bezeichnung) FROM artikel
```

```
count()
      9
```

Wenn Sie in die Tabelle einen Artikel eintragen, dessen Bezeichnung unbekannt ist (NULL-Wert), ändert sich das Ergebnis nicht.

SELECT mit GROUP BY:

Für jede Auftragsnummer die Anzahl verschiedener Artikel zählen:

```
SELECT auftrags_nr, COUNT(DISTINCT artikel_nr)
FROM posten
GROUP BY (auftrags_nr)
```

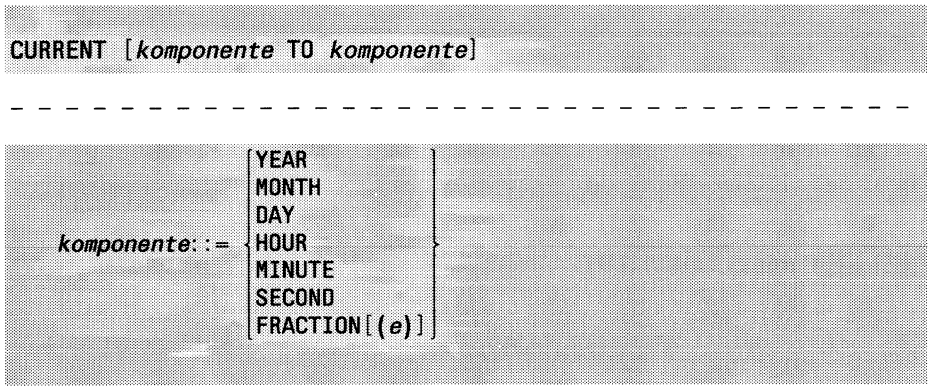
```
auftrags_nr     (count)
       1001           1
       1002           2
       1003           3
       1004           3
       . . .
```

Wenn Sie in die Tabelle einen Satz eintragen, dessen Spalte *gesamtpreis* der NULL-Wert ist, ändert sich das Ergebnis nicht.

CURRENT - Aktuelles Datum

Funktionsgruppe: Zeitfunktion

CURRENT liefert aktuelles Datum und aktuelle Uhrzeit im Datentyp DATETIME.



komponente TO *komponente*
Angabe der Komponenten, die der zurückgelieferte Zeitpunkt haben soll.
Die möglichen Kombinationen sind ausführlich in Kapitel 4, Abschnitt 4.2.4 *Zeit-Datentypen* und 4.3.4 *Zeitwerte* beschrieben.

komponente TO *komponente* nicht angegeben:
Es gilt YEAR TO FRACTION, das heißt der zurückgelieferte Zeitpunkt enthält alle Komponenten.

Ergebnis

Datentyp: DATETIME

Aktuelles Datum und aktuelle Uhrzeit.

- Rufen Sie CURRENT innerhalb einer Anweisung mehrmals auf, ist nicht sichergestellt, daß jedesmal ein anderer Wert zurückgeliefert wird. Es kann mehrmals derselbe Wert vorkommen.
- INFORMIX führt die CURRENT-Aufrufe nicht unbedingt in der Reihenfolge aus, in der sie in einer Anweisung angegeben sind. Daher können Sie CURRENT nicht dazu verwenden, Ausführungszeitpunkte innerhalb der Anweisung zu markieren.

Beispiel

Aktuellen Zeitpunkt in die Tabelle *zeiten* eintragen:

```
CREATE TABLE zeiten (zeitpunkt  DATETIME MONTH TO HOUR);
INSERT INTO zeiten (zeitpunkt) VALUES (CURRENT);
SELECT * FROM zeiten
```

```
zeitpunkt
1990.12.5 22:45:30.111
```

DATE() - Datum erzeugen

Funktionsgruppe: Zeitfunktion

DATE() erzeugt ein Datum des Datentyps DATE.

```
DATE(d)
```

d

Wert, der in ein Datum umgewandelt werden soll.
Sie können angeben:
ganzzahl, *zeichenkette*, *zeitpunkt*.

ganzzahl

Ganzzahl, die die Anzahl Tage seit dem 31.12.1899 angibt.

zeichenkette

Zeichenkette, die ein gültiges Datum angibt. Die Komponenten Tag, Monat und Jahr müssen in der Reihenfolge angegeben werden, die im aktuellen Datumsformat festgelegt ist.
Als Trennzeichen dürfen Sie jedes gültige Trennzeichen für ein Datum verwenden (., -, und /).

Beispiel: Ist `DBDATE=DMY4.`, dann dürfen Sie angeben:
`DATE("12/4/90")`, aber nicht `DATE("90/31/3")`.

zeitpunkt

Zeitpunkt.
Es gelten die üblichen Regeln für die Anpassung der Komponenten (siehe Kapitel 4, Abschnitt 4.3.4 *Zeitwerte*):

Enthält *zeitpunkt* Komponenten, die nicht zu einem Datum gehören, werden diese Komponenten abgeschnitten.

Beispiel: Liefert `CURRENT` den Zeitpunkt:
`1990-05-24 10:58:19.000`, dann liefert `DATE(CURRENT)` das Datum:
`24.5.1990`.

Fehlen in *zeitpunkt* Datumskomponenten, werden sie automatisch ergänzt und mit Standardwerten wie folgt belegt:

- Für Komponentenwerte, die am Anfang ergänzt werden, wird der entsprechende Wert des aktuellen Datums eingesetzt.

  Beispiel: `DATE(DATETIME(12-5) MONTH TO DAY)` liefert: 12.5.1990.

- Werden Monat oder Tag am Ende ergänzt, wird 1 eingesetzt.

  Beispiel: `DATE(DATETIME(92-12)YEAR TO MONTH)` liefert: 1.12.1992.

Ergebnis

Datentyp: DATE

NULL-Wert, wenn *d* den NULL-Wert ergibt.

Der Wert von *d* als Datum, sonst.
Bei der Ausgabe wird *d* im aktuellen Datumsformat ausgegeben.

Beispiel

In der Tabelle *auftrag* der Beispieldatenbank *versand* das Auftragsdatum des Auftrags 1001 ändern:

```
UPDATE auftrag SET auftragsdatum=DATE(33600) WHERE auftrags_nr=1001;
SELECT auftragsdatum FROM auftrag WHERE auftrags_nr=1001
```

```
auftragsdatum
26.12.1991
```

DAY() - Tag bestimmen

Funktionsgruppe: Zeitfunktion

DAY() bestimmt aus einem Datum die Tageszahl.

```
DAY(d)
```

d

Datum, aus dem der Tag bestimmt werden soll.
Sie können angeben:
ganzzahl, *zeichenkette*, *zeitpunkt*.

ganzzahl

Ganzzahl, die die Anzahl Tage seit dem 31.12.1899 angibt.

zeichenkette

Zeichenkette, die ein gültiges Datum angibt. Die Komponenten Tag, Monat und Jahr müssen in der Reihenfolge angegeben werden, die im aktuellen Datumsformat festgelegt ist.
Als Trennzeichen dürfen Sie jedes gültige Trennzeichen für ein Datum verwenden (., -, und /).

Beispiel: Ist `DBDATE=DMY4.`, dann dürfen Sie angeben:
`DAY("12/4/90")`, aber nicht `DAY("90/31/3")`.

zeitpunkt

Zeitpunkt.
Es gelten die üblichen Regeln für die Anpassung der Komponenten (siehe Kapitel 4, Abschnitt 4.3.4 *Zeitwerte*):

Fehlt in *zeitpunkt* die Tagkomponente, wird sie automatisch ergänzt und mit Standardwerten wie folgt belegt:

- Wird sie am Anfang ergänzt, wird der aktuelle Tag eingesetzt.

  Beispiel: `DAY(DATETIME(23:45) HOUR TO MINUTE)` liefert:
  24, wenn der aktuelle Tag der 24. ist.

- Wird sie am Ende ergänzt, wird 1 eingesetzt.
  Beispiel: `DAY(DATETIME(92-12)YEAR TO MONTH)` liefert: 1.

Ergebnis

Datentyp: Numerisch

NULL-Wert, wenn *d* den NULL-Wert ergibt.

Ganzzahl zwischen 1 und 31, die den Tag aus *d* angibt, sonst.

Beispiel

Vom Auftragsdatum des Auftrags 1001 in der Tabelle *auftrag* der Beispieldatenbank *versand* den Tag ausgeben:

```
SELECT DAY(auftragsdatum) FROM auftrag WHERE auftrags_nr=1001
```

```
auftragsdatum
           20
```

EXTEND() - Zeitpunkt anpassen

Funktionsgruppe: Zeitfunktion

EXTEND() dient dazu, Komponenten von Zeitpunkten zu ergänzen bzw. zu streichen. Solche Anpassungen sind in Berechnungen notwendig, wenn Zeitpunkte mehr oder weniger Komponenten haben, als benötigt werden.

```
EXTEND(zeitpunkt[,komponente␣TO␣komponente])
```

```
                    ┌YEAR     ┐
                    │MONTH    │
                    │DAY      │
komponente::=       {HOUR     }
                    │MINUTE   │
                    │SECOND   │
                    └FRACTION[(e)]┘
```

zeitpunkt
Zeitpunkt, der angepaßt werden soll.

komponente TO *komponente*
Angabe der Komponenten, die der Zeitpunkt haben soll.
Für die Komponentenangabe gelten dieselben Bedingungen wie beim Datentyp DATETIME (siehe Kapitel 4, Abschnitt 4.2.4 *Zeit-Datentypen*).

Komponenten von *zeitpunkt*, die nicht in den angegebenen Komponenten enthalten sind, werden entfernt.

Angegebene Komponenten, die nicht in *zeitpunkt* enthalten sind, werden ergänzt und mit folgenden Werten belegt:

- für Komponenten, die am Anfang ergänzt werden, wird der entsprechende Wert des aktuellen Datums bzw. der aktuellen Uhrzeit eingetragen.
- für Komponenten, die am Ende ergänzt werden, wird eine Konstante eingetragen, und zwar 1 bei Monat und Tag und 0 sonst.

komponente TO *komponente* nicht angegeben:
Der Zeitpunkt enthält alle Komponenten YEAR TO FRACTION.

Ergebnis

Datentyp: DATETIME

NULL-Wert, wenn *zeitpunkt* den NULL-Wert ergibt.

Angepaßter Zeitpunkt, sonst.

Beispiel

Das Beispiel bezieht sich auf die Spalte *zeitpunkt* der Tabelle *zeiten*:

```
CREATE TABLE zeiten (zeitpunkt  DATETIME MONTH TO HOUR)
```

Passenden Zeitpunkt eintragen:

```
INSERT INTO zeiten (zeitpunkt)
   VALUES (CURRENT);
```

Komponente HOUR nicht ausgeben:

```
SELECT EXTEND(zeitpunkt, MONTH TO DAY) FROM zeiten
```

```
(expression)
 05-24
```

LENGTH() - Länge einer Zeichenkette bestimmen

Funktionsgruppe: Zeichenkettenfunktion

LENGTH() bestimmt die Länge einer Zeichenkette.

```
LENGTH(zeichenkette)
```

zeichenkette
Zeichenkette, deren Länge bestimmt werden soll.

Sie dürfen auch eine Spalte vom Datentyp TEXT oder BYTE angeben. LENGTH liefert die aktuelle Länge der Daten in Bytes.

Ergebnis

Datentyp: Numerisch

NULL-Wert, wenn *zeichenkette* den NULL-Wert ergibt.

0, wenn *zeichenkette* nur aus Leerzeichen besteht.

Ganzzahl, die die Länge von *zeichenkette* angibt, sonst.
Abschließende Leerzeichen werden nicht mitgezählt.

Beispiel

Für den Auftrag 1001 in der Tabelle *auftrag* der Beispieldatenbank *versand* die Länge des Lieferhinweises abfragen:

```
SELECT LENGTH(lieferhinweis) FROM auftrag WHERE auftrags_nr=1001
```

```
(expression)
           9
```

MAX() - Maximum bestimmen

Funktionsgruppe: Mengenfunktion

MAX() bestimmt den größten Wert einer Menge.

ALL
DISTINCT
Die Angabe ALL oder DISTINCT ist syntaktisch erlaubt, hat aber keine Bedeutung.

spalte
Spalte mit numerischem, alphanumerischem Datentyp oder Zeitdatentyp.

Der Vergleich von alphanumerischen Werten und Zeitwerten ist in Kapitel 4 bei diesen Werten jeweils beschrieben.

Einschränkung:
Sie dürfen keine Spalte vom Datentyp TEXT oder BYTE angeben.

wert
Numerischer Wert, alphanumerischer Wert oder Zeitwert.

Ergebnis

Datentyp: Elementtyp

- Bei Argument *spalte*

  Enthält die Spalte nur NULL-Werte, ist das Ergebnis der NULL-Wert. Ansonsten werden NULL-Werte ignoriert und das Ergebnis ist wie folgt:

  Ohne GROUP-BY-Klausel der SELECT-Anweisung:
  Größter Wert aus *spalte*.

  Mit GROUP-BY-Klausel der SELECT-Anweisung:
  Pro Gruppe in der Ergebnistabelle, der größte Wert von *spalte* für diese Gruppe (siehe Beispiel).

- Bei Argument *wert*

  MAX() liefert *wert* zurück.

Beispiel

SELECT ohne GROUP BY:

Aus der Tabelle *posten* der Beispieldatenbank *versand* den höchsten Gesamtpreis für Auftrag 1006 abfragen:

```
SELECT MAX(gesamtpreis) FROM posten WHERE auftrags_nr=1006
```

```
(max)
 190.00
```

Wenn Sie für den Auftrag 1006 einen Posten eintragen, dessen Gesamtpreis der NULL-Wert ist, ändert sich das Ergebnis nicht.

SELECT mit GROUP BY:

Für jede Auftragsnummer den höchsten Gesamtpreis berechnen:

```
SELECT auftrags_nr, MAX(gesamtpreis)
FROM posten
GROUP BY (auftrags_nr)
```

```
auftrags_nr     (max)
       1001     DM250.00
       1002     DM960.00
       1003     DM840.00
       1004     DM960.00
       . . .
```

MDY() - Datum erzeugen

Funktionsgruppe: Zeitfunktion

MDY() erstellt aus drei Zahlen ein Datum des Datentyps DATE.

```
MDY(mm, tt, jjjj)
```

mm
Ganzzahl zwischen 1 und 12, die den Monat angibt.

tt
Ganzzahl zwischen 1 und 31, die den Tag angibt. Der Tag muß zu dem Monat *m* passen.

jjjj
Ganzzahl, die das Jahr angibt. Sie müssen die Jahreszahl vollständig angeben.

Beispiel: 1990

Ergebnis

Datentyp: DATE

NULL-Wert, wenn *mm*, oder *tt* oder *jjjj* den NULL-Wert ergibt.

Datum aus den angegebenen Werten *tt*, *mm*, *jjjj*, sonst.
Bei der Ausgabe wird der erzeugte Wert im aktuell eingestellten Datumsformat ausgegeben.

Beispiel

In der Tabelle *auftrag* der Beispieldatenbank *versand* das Auftragsdatum des Auftrags 1001 auf den 24.5.1990 setzen:

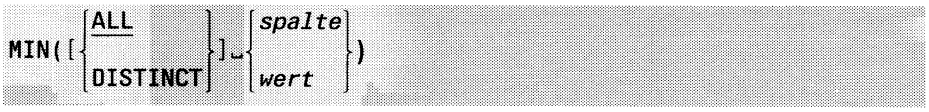
```
UPDATE auftrag SET auftragsdatum=MDY(5,24,1990) WHERE auftrags_nr=1001;
SELECT auftragsdatum FROM auftrag WHERE auftrags_nr=1001
```

```
auftragsdatum
    24.5.1990
```

MIN() - Minimum bestimmen

Funktionsgruppe: Mengenfunktion

MIN() bestimmt das kleinste Element einer Menge.



ALL
DISTINCT
Die Angabe ALL oder DISTINCT ist syntaktisch erlaubt, hat aber keine Bedeutung.

spalte
Spalte mit numerischem, alphanumerischem Datentyp oder Zeitdatentyp.

Der Vergleich von alphanumerischen Werten und Zeitwerten ist im Kapitel 4 bei diesen Werten jeweils beschrieben.

Einschränkung:
Sie dürfen keine Spalte vom Datentyp TEXT oder BYTE angeben.

wert
Numerischer Wert, alphanumerischer Wert oder Zeitwert.

Ergebnis

Datentyp: Elementtyp

- Bei Argument *spalte*

  Enthält die Spalte nur NULL-Werte, ist das Ergebnis der NULL-Wert. Ansonsten werden NULL-Werte ignoriert und das Ergebnis ist wie folgt:

  Ohne GROUP-BY-Klausel der SELECT-Anweisung:
  Kleinstes Element von *spalte*.

  Mit GROUP-BY-Klausel der SELECT-Anweisung:
  Pro Gruppe in der Ergebnistabelle, das kleinste Element von *spalte* für diese Gruppe (siehe Beispiel).

- Bei Argument *wert*

  MIN() liefert *wert* zurück.

Beispiel

SELECT ohne GROUP BY:

Aus der Tabelle *posten* der Beispieldatenbank *versand* den niedrigsten Gesamtpreis für Auftrag 1006 abfragen:

```
SELECT MIN(gesamtpreis) FROM posten WHERE auftrags_nr=1006
```

```
(min)
  36.00
```

Wenn Sie für Auftrag 1006 einen Posten eintragen, dessen Spalte *gesamtpreis* der NULL-Wert ist, ändert sich das Ergebnis nicht.

SELECT mit GROUP BY:

Für jede Auftragsnummer den niedrigsten Gesamtpreis berechnen:

```
SELECT auftrags_nr, MIN(gesamtpreis)
FROM posten
GROUP BY (auftrags_nr)
```

```
auftrags_nr     (min)
       1001     DM250.00
       1002     DM240.00
       1003     DM 20.00
       1004     DM126.00
       . . .
```

MONTH() - Monat bestimmen

Funktionsgruppe: Zeitfunktion

MONTH() bestimmt aus einem Datum die Nummer des Monats.

```
MONTH(d)
```

d

Datum, aus dem der Monat bestimmt werden soll.
Sie können angeben:
ganzzahl, *zeichenkette*, *zeitpunkt*.

ganzzahl

Ganzzahl, die die Anzahl Tage seit dem 31.12.1899 angibt.

zeichenkette

Zeichenkette, die ein gültiges Datum angibt. Die Komponenten Tag, Monat und Jahr müssen in der Reihenfolge angegeben werden, die im aktuellen Datumsformat festgelegt ist.
Als Trennzeichen dürfen Sie jedes gültige Trennzeichen für ein Datum verwenden (., -, und /).

Beispiel: Ist `DBDATE=DMY4.`, dann dürfen Sie angeben:
`MONTH("12/4/90")`, aber nicht `MONTH("90/31/3")`.

zeitpunkt

Zeitpunkt.
Es gelten die üblichen Regeln für die Anpassung der Komponenten (siehe Kapitel 4, Abschnitt 4.3.4 *Zeitwerte*):

Fehlt in *zeitpunkt* die Monatkomponente, wird sie automatisch ergänzt und mit Standardwerten wie folgt belegt:

- Wird sie am Anfang ergänzt, wird der aktuelle Monat eingesetzt.

  Beispiel: `DAY(DATETIME(23:45) HOUR TO MINUTE)` liefert:
  5, wenn der aktuelle Monat Mai ist.

- Wird sie am Ende ergänzt, wird 1 eingesetzt.

  Beispiel: `DAY(DATETIME(92)YEAR TO YEAR )` liefert: 1.

Ergebnis

Datentyp: Numerisch

NULL-Wert, wenn *d* den NULL-Wert ergibt.

Ganzzahl zwischen 1 und 12, die den Monat aus *d* angibt, sonst.

Beispiel

Von dem Auftragsdatum für Auftrags 1001 in der Tabelle *auftrag* der Beispieldatenbank *versand* den Monat ausgeben:

```
SELECT MONTH(auftragsdatum) FROM auftrag WHERE auftrags_nr=1001
```

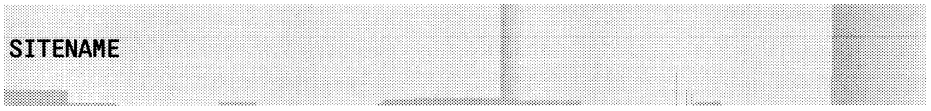
```
auftragsdatum
            1
```

SITENAME - Aktuelles Informixsystem

Nur INFORMIX-ONLINE plus INFORMIX-STAR

Funktionsgruppe: Zeichenkettenfunktion

SITENAME liefert den Namen des aktuellen Informixsystems.



Ergebnis

Datentyp: alphanumerisch

Name des aktuellen Informixsystems als Zeichenkette. Dieser Name wird als Servername vom INFORMIX-ONLINE-Verwalter bei der Initialisierung vergeben (siehe INFORMIX-ONLINE-Handbuch [10]).

SUM() - Summe berechnen

Funktionsgruppe: Mengenfunktion

SUM() berechnet die Summe aller Werte einer Menge.

ALL
: Alle Werte werden berücksichtigt, auch solche die doppelt vorkommen.

DISTINCT
: Nur verschiedene Werte werden berücksichtigt. Duplikate werden ignoriert.

spalte
: Spalte.
Die Spalte muß numerisch sein oder vom Datentyp INTERVAL.

Addition von Zeitspannen sind im Kapitel 4, Abschnitt 4.3.4 *Zeitwerte* beschrieben.

wert
: Numerischer Wert oder Zeitspanne.

Ergebnis

Datentyp: Numerisch

- Bei Argument *spalte*

  Enthält die Spalte nur NULL-Werte, ist das Ergebnis der NULL-Wert. Ansonsten werden NULL-Werte ignoriert und das Ergebnis ist wie folgt:

  Ohne GROUP-BY-Klausel der SELECT-Anweisung:
  : Summe der Werte aus *spalte*.

  Mit GROUP-BY-Klausel der SELECT-Anweisung:
  : Pro Gruppe in der Ergebnistabelle, die Summe von *spalte* für diese Gruppe (siehe Beispiel).

- Bei Argument *wert*

  SUM() liefert: *wert* * Anzahl der Ergebnissätze.

Beispiel

SELECT ohne GROUP BY:

Aus der Tabelle *posten* der Beispieldatenbank *versand* die Summe aller Gesamtpreise abfragen:

```
SELECT SUM(gesamtpreis) FROM posten
```

```
(expression)
  DM11937.80
```

Wenn Sie in die Tabelle einen Satz eintragen, dessen Gesamtpreis unbekannt ist (NULL-Wert), ändert sich das Ergebnis nicht.

SELECT mit GROUP BY:

Für jede Auftragsnummer die Summe seiner Gesamtpreise berechnen:

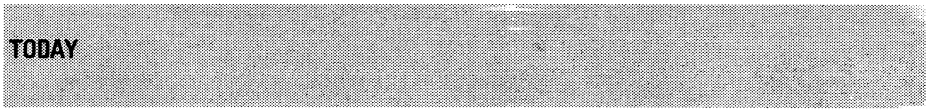
```
SELECT auftrags_nr, SUM(gesamtpreis)
FROM posten
GROUP BY (auftrags_nr)
```

```
auftrags_nr     (sum)
       1001     DM250.00
       1002     DM1200.00
       1003     DM959.00
       1004     DM2126.00
       . . .
```

TODAY - Aktuelles Datum

Funktionsgruppe: Zeitfunktion

TODAY liefert das aktuelle Datum im Datentyp DATE.



Ergebnis

Datentyp: DATE

Aktuelles Datum als Ganzzahl.

Beispiel

In der Tabelle *auftrag* der Beispieldatenbank *versand* das Auftragsdatum des Auftrags 1001 auf das aktuelle Datum setzen:

```
UPDATE auftrag SET auftragsdatum=TODAY WHERE auftrags_nr=1001;
SELECT auftragsdatum FROM auftrag WHERE auftrags_nr=1001
```

```
auftragsdatum
    24.5.1990
```

USER - Aktueller Benutzer

Funktionsgruppe: Zeichenkettenfunktion

USER liefert den Namen des aktuellen Benutzers.

```
USER
```

Ergebnis

Datentyp: alphanumerisch

Name des aktuellen Benutzers als Zeichenkette.

Beispiel

In der Tabelle *kunde* der Beispieldatenbank *versand* den Nachnamen des Kunden 101 auf den aktuellen Benutzernamen setzen:

```
UPDATE kunde SET nachname=USER WHERE kunden_nr=101;
SELECT nachname FROM kunde WHERE kunden_nr=101
```

```
nachname
    jager
```

WEEKDAY() - Wochentag bestimmen

Funktionsgruppe: Zeitfunktion

WEEKDAY() bestimmt aus einem Datum den Wochentag.

d

Datum, aus dem der Wochentag bestimmt werden soll.
Sie können angeben:
ganzzahl, *zeichenkette*, *zeitpunkt*.

ganzzahl

Ganzzahl, die die Anzahl Tage seit dem 31.12.1899 angibt.

zeichenkette

Zeichenkette, die ein gültiges Datum angibt. Die Komponenten Tag, Monat und Jahr müssen in der Reihenfolge angegeben werden, die im aktuellen Datumsformat festgelegt ist.
Als Trennzeichen dürfen Sie jedes gültige Trennzeichen für ein Datum verwenden (., -, und /).

Beispiel: Ist `DBDATE=DMY4.`, dann dürfen Sie angeben:
`WEEKDAY("12/4/90")`, aber nicht `WEEKDAY("90/31/3")`.

zeitpunkt

Zeitpunkt.
Es gelten die üblichen Regeln für die Anpassung der Komponenten (siehe Kapitel 4, Abschnitt 4.3.3 *zeitwerte*):

Fehlt in *zeitpunkt* die Tagkomponente, wird sie automatisch ergänzt und mit Standardwerten wie folgt belegt:

- Wird sie am Anfang ergänzt, wird der aktuelle Tag eingesetzt.

  Beispiel: `WEEKDAY(DATETIME(23:45) HOUR TO MINUTE)` liefert: 4, wenn der aktuelle Tag Donnerstag ist.

- Wird sie am Ende ergänzt, wird 1 eingesetzt.

  Beispiel: `WEEKDAY(DATETIME(5)MONTH TO MONTH )` liefert: 2, wenn der 1.5 ein Dienstag ist.

Ergebnis

Datentyp: Numerisch

NULL-Wert, wenn *d* den NULL-Wert ergibt.

Ganzzahl zwischen 0 und 6, die den Wochentag angibt, sonst.
0 steht für Sonntag, 6 für Samstag.

Beispiel

Vom Auftragsdatum des Auftrags 1001 in der Tabelle *auftrag* der Beispieldatenbank *versand* den Wochentag ausgeben:

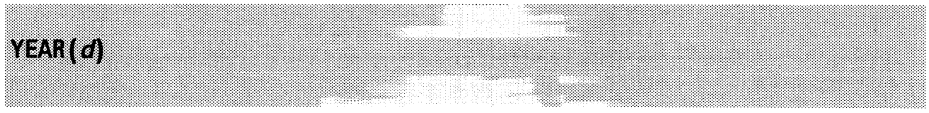
```
SELECT WEEKDAY(auftragsdatum) FROM auftrag
```

```
(expression)
          3
```

YEAR() - Jahr bestimmen

Funktionsgruppe: Zeitfunktion

YEAR() bestimmt aus einem Datum das Jahr.



d

Datum, aus dem das Jahr bestimmt werden soll.
Sie können angeben:
ganzzahl, *zeichenkette*, *zeitpunkt*.

ganzzahl

Ganzzahl, die die Anzahl Tage seit dem 31.12.1899 angibt.

zeichenkette

Zeichenkette, die ein gültiges Datum angibt. Die Komponenten Tag, Monat und Jahr müssen in der Reihenfolge angegeben werden, die im aktuellen Datumsformat festgelegt ist.
Als Trennzeichen dürfen Sie jedes gültige Trennzeichen für ein Datum verwenden (., -, und /).

Beispiel: Ist `DBDATE=DMY4.`, dann dürfen Sie angeben:
`YEAR("12/4/90")`, aber nicht `YEAR("90/31/3")`.

zeitpunkt

Zeitpunkt.
Es gelten die üblichen Regeln für die Anpassung der Komponenten (siehe Kapitel 4, Abschnitt 4.3.4 *Zeitwerte*):

Fehlt in *zeitpunkt* die Jahrkomponente, wird sie automatisch ergänzt und mit dem aktuellen Jahr belegt.

Ergebnis

Datentyp: Numerisch

NULL-Wert, wenn *d* den NULL-Wert ergibt.

Ganzzahl, die das Jahr aus *d* angibt, sonst.

Beispiel

Vom Auftragsdatum des Auftrags 1001 in der Tabelle *auftrag* der Beispieldatenbank *versand* das Jahr ausgeben:

```
SELECT YEAR(auftragsdatum) FROM auftrag WHERE auftrags_nr=1001
```

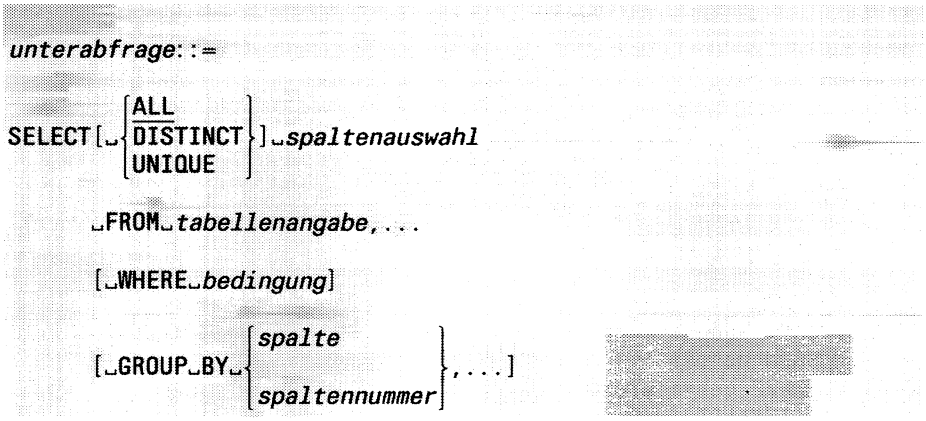
```
auftragsdatum
         1990
```

5.2 Unterabfragen

Eine Unterabfrage ist eine eingeschränkte SELECT-Anweisung, die eine **einspaltige** Ergebnistabelle liefert. Unterabfragen können in folgenden Fällen verwendet werden:

- In den Prädikaten, die einen Vergleich mit einer Ergebnisspalte durchführen (ANY-, SOME-, ALL- und IN).
- In Ausdrücken.
  In diesem Fall muß die Unterabfrage genau einen Wert liefert. Sie kann dann wie ein konstanter Wert in dem Ausdruck verwendet werden (siehe Abschnitt 5.3 *Ausdrücke*).

Eine Unterabfrage wird immer in runde Klammern eingeschlossen.



Gegenüber einer SELECT-Anweisung gibt es folgende Einschränkungen:

- Es darf nur eine Spaltenauswahl geben, da die Ergebnistabelle einspaltig sein muß.
- Die Klauseln ORDER BY, INTO, INTO TEMP und UNION sind nicht erlaubt.
- Die Spaltenauswahl darf keine Spalte vom Datentyp TEXT oder BYTE enthalten.

Ansonsten sind die Klauseln bei der SELECT-Anweisung in Kapitel 6 beschrieben.

Korrelierte Unterabfragen

Bei einer geschachtelten SELECT-Abfrage heißt eine innere Unterabfrage **korrelierte Unterabfrage**, wenn sie sich auf Spalten einer äußeren Tabelle bezieht, das heißt einer Tabelle, die in einer der äußeren SELECT-Abfragen verwendet wird.

Mit Hilfe von korrelierten Unterabfragen können Sie Beziehungen zwischen den Spaltenwerten einer Spalte bestimmen.

Beispiel:

Bei einer Personentabelle mit einer Spalte, die für jede Person das Alter enthält, können Sie feststellen, welche Personen genau das Durchschnittsalter haben (siehe auch Beispiel unten).

Nicht korrelierte Unterabfragen werden nur einmal ausgewertet. Korrelierte Unterabfragen werden für **jeden** Satz der äußeren Tabelle ausgewertet. Ist die Unterabfrage geschachtelt, erfolgt die Auswertung von innen nach außen.

Beispiel

Für jeden Auftrag aus der Tabelle *posten* der Beispieldatenbank *versand* die Posten heraussuchen, deren Gesamtpreis mindestens das Doppelte des billigsten Posten dieses Auftrags beträgt:

```
SELECT auftrags_nr, artikel_nr, herstellercode, gesamtpreis
       FROM posten x
       WHERE gesamtpreis
       > (SELECT 2 * MIN (gesamtpreis)
                FROM posten
                WHERE auftrags_nr = x.auftrags_nr)
```

```
auftrags_nr artikel_nr herstellercode gesamtpreis
       1002          4 HSK                960,00
       1003          8 ANZ                840,00
       1003          5 ANZ                 99,00
       1004          1 HRO                960,00
       1004          1 HSK                800,00
       1005          5 NRG                280,00
       1005          5 ANZ                198,00
       1006          5 SMT                125,00
       1006          5 NRG                190,00
       1006          5 ANZ                 99,00
       1007          4 HRO                480,00
       1007          7 HRO                600,00
       1008          8 ANZ                840,00
       1012          8 ANZ                840,00
       1013          6 ANZ                 48,00
       1013          9 ANZ                 40,00
```

Die Unterabfrage bezieht sich auf die Spalte *auftrags_nr* der äußeren Tabelle *posten*.

Die Auswertung ist wie folgt:

- Für jede Auftragsnummer in der Tabelle *posten* wird die Unterabfrage ausgewertet.
- Sie liefert als Ergebnis zu einem Auftrag den doppelten minimalen Gesamtpreis seiner Posten.
- Dieser Wert wird als Vergleichswert in die äußere Abfrage eingesetzt und dann werden die Posten herausgesucht, deren Gesamtpreis über diesem Wert liegt.

Beachten Sie, daß für die äußere Tabelle eine Referenz angegeben werden muß, um die Spalten eindeutig ansprechen zu können.

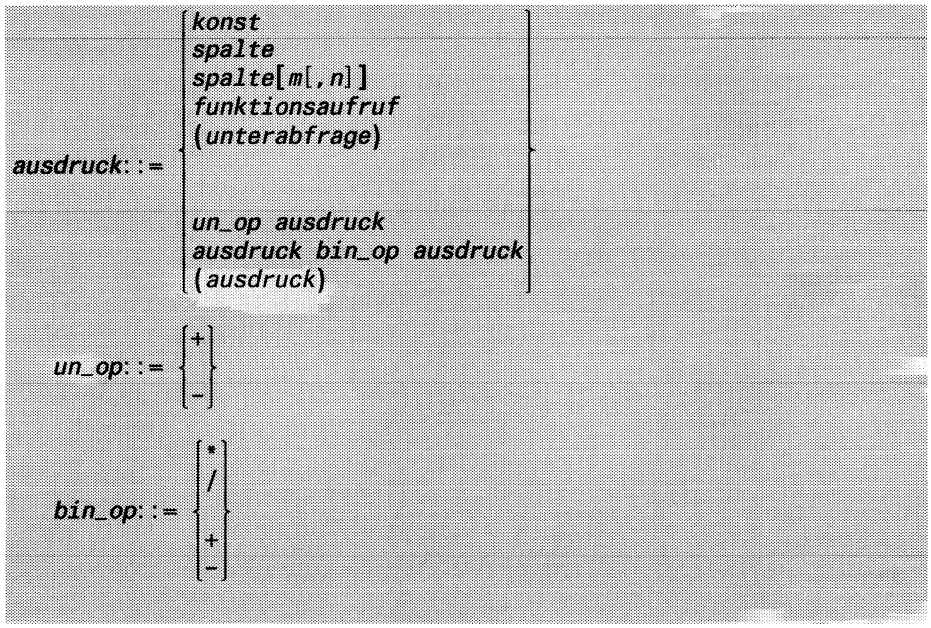
5.3 Ausdrücke

Ein Ausdruck ergibt einen Wert. Ausdrücke können vorkommen in:

- Spaltenauswahl (SELECT-Anweisung)
- Prädikaten von Bedingungen (WHERE- und HAVING-Klausel)
- Zuweisungen (UPDATE-Anweisung).

Ein Ausdruck besteht aus Operanden und eventuell Operatoren. Ein Ausdruck wird wie folgt ausgewertet:

- Zuerst werden die Operanden berechnet.
- Sind Operatoren vorhanden, werden diese anschließend auf die berechneten Werte angewendet.
- Ist irgendein Operand der NULL-Wert, ist das Ergebnis der NULL-Wert. Ansonsten ist das Ergebnis einer Auswertung ein alphanumerischer, numerischer Wert oder ein Zeitwert.



konst

Alphanumerische Konstante, numerische Konstante oder Zeitkonstante (siehe Kapitel 4).

Bei Programmeinbettung können Sie auch eine Hostvariable angeben, die die Konstante enthält.

spalte
Name einer Tabellenspalte, aus der die Werte genommen werden.

spalte[*m*[,*n*]]
Ausschnitt der Tabellenspalte *spalte* auswählen.
Die Spalte muß vom Datentyp CHARACTER, VARCHAR oder TEXT sein. Sie geben den Ausschnitt der Spalte durch Anfangs- und Endposition an.

Die äußeren eckigen Klammern müssen angegeben werden. Die innere eckige Klammer ist das übliche Metasymbol für optionale Angabe.

m
Anfangsposition. Sie muß kleiner oder gleich *n* sein. Die Positionen in einer Spalte sind beginnend mit 1 durchnumeriert.

n
Endposition. Sie muß größer oder gleich *m* sein.

n nicht angegeben:
Es wird das Zeichen an der Position *m* ausgewählt.

Einschränkungen:
- Sie dürfen in einem Ausdruck nicht gleichzeitig eine Mengenfunktion und eine weitere Spalte angeben.
- In den meisten Fällen, in denen Ausdrücke verwendet werden, dürfen Sie keine Spalte vom Typ TEXT oder BYTE angeben. Eine BLOB-Spalte ist nur erlaubt bei dem Prädikat IS [NOT] NULL und der Funktion LENGTH sowie in der Spaltenauswahl einer SELECT-Anweisung.

funktionsaufruf
Aufruf einer SQL-Funktion.

unterabfrage
Unterabfrage, die genau einen Wert liefert.

un_op
Einstelliger Operator, der das Vorzeichen angibt. Das Ergebnis von *ausdruck* muß numerisch oder vom Datentyp INTERVAL sein.

\+ Positiv

\- Negativ

bin_op
Zweistelliger Operator.

Einschränkung:
Sie dürfen keine Spalte vom Datentyp TEXT oder BYTE angeben.

a * *b*

Multiplikation von *a* mit *b*.
Die Ausdrücke *a* und *b* müssen entweder beide numerisch sein oder ein Operand muß numerisch und der andere vom Datentyp INTERVAL sein:

Datentyp von a	Datentyp von b	Ergebnistyp
numerisch	numerisch	numerisch
numerisch	INTERVAL	INTERVAL
INTERVAL	numerisch	INTERVAL

Sind beide Operanden numerisch, ist * die arithmetische Multiplikation. Das Ergebnis wird in den größeren der beiden Datentypen umgewandelt.

Ist ein Operand eine Zeitspanne, wird jeder Komponentenwert mit dem numerischen Wert multipliziert und der resultierende Wert wird, wenn nötig, ganzzahlig abgerundet. Das Ergebnis ist die Zeitspanne mit den neuen Komponentenwerten. Sie enthält alle Komponenten der Ausgangszeitspanne.

a / *b*

Division von *a* durch *b*
Die Operanden *a* und *b* müssen entweder beide numerisch sein oder ein Operand muß numerisch und der andere vom Typ INTERVAL sein:

Datentyp von a	Datentyp von b	Ergebnistyp
numerisch	numerisch	numerisch
numerisch	INTERVAL	INTERVAL
INTERVAL	numerisch	INTERVAL

Sind beide Operanden numerisch, ist / die arithmetische Division. Das Ergebnis wird in den größeren der beiden Datentypen umgewandelt.

Ist ein Operand eine Zeitspanne, wird jeder Komponentenwert durch den numerischen Wert dividiert und der resultierende Wert wird, wenn nötig, ganzzahlig abgerundet. Das Ergebnis ist die Zeitspanne mit den neuen Komponentenwerten. Sie enthält alle Komponenten der Ausgangszeitspanne.

a + *b*

Addition von *a* und *b*.
Die Operanden *a* und *b* können numerische Werte oder Zeitwerte sein. Das Ergebnis ist abhängig vom Datentyp der Operanden *a* und *b*. Folgende Kombinationen sind erlaubt:

Datentyp von a	Datentyp von b	Ergebnistyp	Ergebnis
numerisch	numerisch	numerisch	num. Wert a+b
numerisch	DATE	DATE	Datum a+b
DATE	numerisch	DATE	Datum a+b
DATE	INTERVAL	DATETIME	Zeitpunkt a+b
DATETIME	INTERVAL	DATETIME	Zeitpunkt a+b
INTERVAL	DATE	DATETIME	Zeitpunkt a+b
INTERVAL	DATETIME	DATETIME	Zeitpunkt a+b
INTERVAL	INTERVAL	INTERVAL	Zeitspanne a+b

Die Addition von Zeitwerten ist bei den Zeitwerten in Kapitel 4, Abschnitt 4.3.4 *Zeitwerte* erklärt.

a - *b*

Subtraktion b von a.
Die Operanden *a* und *b* können numerische Werte oder Zeitwerte sein. Das Ergebnis ist abhängig vom Datentyp der Operanden *a* und *b*. Folgende Kombinationen sind erlaubt:

Datentyp von a	Datentyp von b	Ergebnistyp	Ergebnis
numerisch	numerisch	numerisch	num. Wert a-b
DATE	numerisch	DATE	Datum a-b
DATE	INTERVAL	DATETIME	Zeitpunkt a-b
DATETIME	INTERVAL	DATETIME	Zeitpunkt a-b
INTERVAL	INTERVAL	INTERVAL	Zeitspanne a-b

Die Subtraktion von Zeitwerten ist bei den Zeitwerten in Kapitel 4, Abschnitt 4.3.4 *Zeitwerte* erklärt.

Prioritäten

- Klammerausdrücke haben die höchste Priorität.
- Die einstelligen Operatoren haben Vorrang vor den zweistelligen.
- Die multiplikativen Operatoren * und / haben höhere Priorität als die additiven + und -.
- Die multiplikativen Operatoren haben gleiche Priorität.
- Die additiven zweistelligen Operatoren haben gleiche Priorität.
- Bei mehreren Operatoren gleicher Priorität wird von links nach rechts ausgewertet.

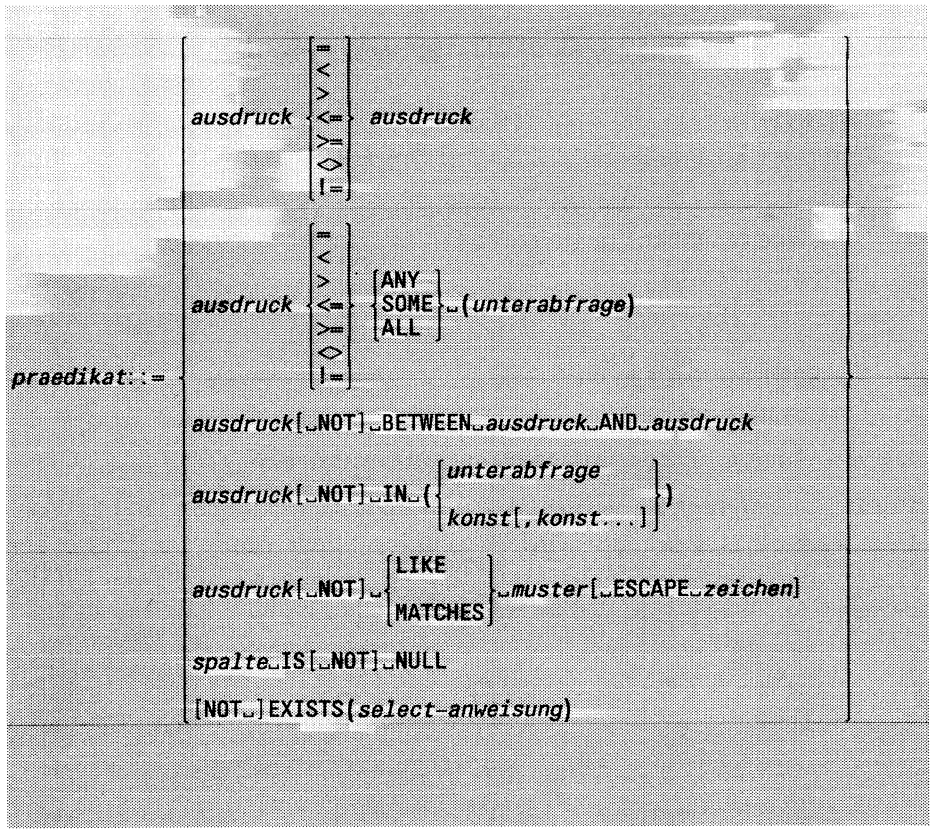
5.4 Prädikate

Prädikate sind die Bestandteile von Bedingungen (siehe Abschnitt 5.5 *Bedingung*).
Ein Prädikat besteht aus Operanden und Operatoren. Entsprechend den Operatoren sind Prädikate in folgende Gruppen unterteilt:

- Vergleich von zwei Werten
- Vergleich mit einer Ergebnisspalte
- Bereichsabfrage
- Elementabfrage
- Mustervergleich
- Vergleich auf NULL-Wert
- Existenzabfrage.

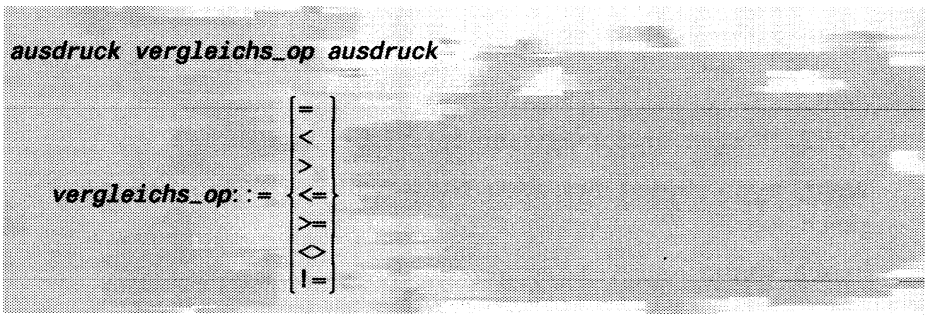
Die einzelnen Gruppen sind nachfolgend beschrieben.

Ein Prädikat liefert den Wahrheitswert wahr, falsch oder den NULL-Wert. Der Wert eines Prädikats wird berechnet, indem zuerst die Werte der Operanden berechnet werden und dann die jeweiligen Operatoren auf die berechneten Werte angewendet werden.


praedikat::=
ausdruck { = | < | > | <= | >= | <> | != } ausdruck
ausdruck { = | < | > | <= | >= | <> | != } { ANY | SOME | ALL } (unterabfrage)
ausdruck[NOT] BETWEEN ausdruck AND ausdruck
ausdruck[NOT] IN ({ unterabfrage | konst[,konst...] })
ausdruck[NOT] { LIKE | MATCHES } muster[ESCAPE zeichen]
spalte IS[NOT] NULL
[NOT]EXISTS(select-anweisung)

Vergleich von zwei Werten

Zwei Operanden mit vergleichbarem Datentyp werden gemäß dem angegebenen Vergleichsoperator verglichen.



ausdruck

Ausdruck.

Die beiden Ausdrücke müssen vergleichbare Werte ergeben (siehe unten, Abschnitt *Vergleichbare Werte*).

Eine für relationale Datenbanken charakteristische Verwendung des Vergleichs von Werten ist der Join. Bei einem Join wird als Join-Bedingung ein Vergleich angegeben, bei dem beide Operanden Spalten sind, deren Werte miteinander verglichen werden. Der Join ist ausführlich im Abschnitt 5.6 *Join* beschrieben.

Einschränkung:

- Sie dürfen nicht die Konstante NULL angeben.
- Sie dürfen keine Spalte vom Datentyp TEXT oder BYTE angeben.

vergleichs_op

=

Vergleich auf Gleichheit

<

Vergleich auf kleiner

>

Vergleich auf größer

< =
Vergleich auf kleiner oder gleich

> =
Vergleich auf größer oder gleich

**< >, != **
Vergleich auf Ungleichheit

Ergebnis

NULL-Wert, wenn ein Operand der NULL-Wert ist.

Wahr, wenn beide Operanden Nicht-Null-Werte sind und der Vergleich zutrifft.

Falsch, sonst.

Vergleichbare Werte

Wie eine Vergleichsoperation ausgeführt wird, ist abhängig vom Datentyp der Operanden. Sie ist ausführlich in Kapitel 4 für jeden Wert beschrieben. Im folgenden sind die wichtigsten Punkte nochmals zusammengestellt.

NULL-Werte

Ist ein Operand der NULL-Wert, liefern alle Vergleiche als Ergebnis den NULL-Wert (siehe auch Kapitel 4, Abschnitt 4.3.1 *NULL-Wert*).

Alphanumerische Werte

Zwei Zeichenketten sind gleich, wenn sie an jeder Position das gleiche Zeichen haben. Sind sie ungleich, bestimmt der ASCII-Code der ersten beiden unterschiedlichen Zeichen, welche Zeichenkette größer bzw. kleiner ist. Bei unterschiedlich langen Zeichenketten wird die kürzere mit Leerzeichen aufgefüllt (siehe auch Kapitel 4, Abschnitt 4.3.2 *Alphanumerische Werte*).

Numerische Werte

Zwei numerische Werte sind gleich, wenn sie dasselbe Vorzeichen und denselben Betrag haben (siehe auch Kapitel 4, Abschnitt 4.3.3 *Numerische Werte*).

Zeitwerte

Datum und Zeitpunkt können verglichen werden. Bei unterschiedlichen Komponenten werden die für den Vergleich notwendigen Anpassungen automatisch durchgeführt. Ein Datum oder Zeitpunkt ist größer als ein anderes Datum oder ein anderer Zeitpunkt, wenn es jünger ist.

Zeitspannen können nur mit Zeitspannen verglichen werden (siehe auch Kapitel 4, Abschnitt 4.3.4 *Zeitwerte*).

Beispiel 1

Alphanumerische Werte vergleichen:
Aus der Tabelle *kunde* der Beispieldatenbank *versand* die Kunden mit Firma heraussuchen, die aus einem Ort kommen, dessen Name mit *Aug* beginnt:

```
SELECT vorname, nachname, firma, ort
       FROM kunde
       WHERE ort[1,3] ="Aug"
```

```
vorname         nachname        firma               ort
Ludwig          Pauli           Pauli Sport         Augsburg
Janette         Millet          Sportausstattung    Augsburg
Frank           Keyser          Sport Keyser        Augsburg
```

Beispiel 2

Vergleich mit Unterabfrage, die genau einen Wert liefert:
Aus der Tabelle *posten* der Beispieldatenbank *versand* den Auftrag heraussuchen, bei dem die größte Menge des Artikels Nummer 9 bestellt wurde:

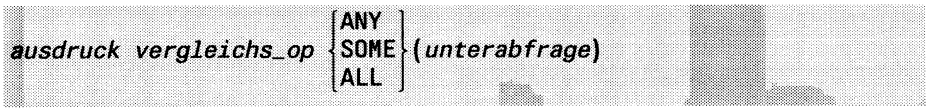
```
SELECT auftrags_nr
       FROM posten
       WHERE artikel_nr = 9
       AND
       menge = (SELECT MAX(menge)
                FROM posten
                WHERE artikel_nr = 9)
```

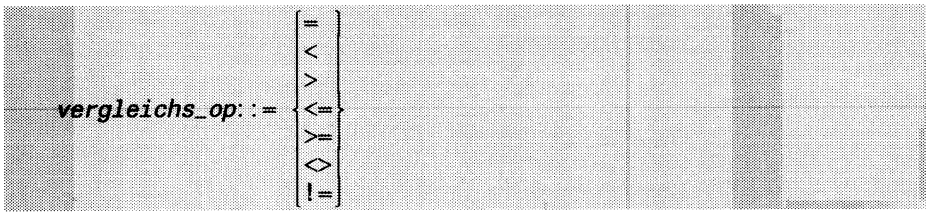
Da das Maximum immer ein eindeutiger Wert ist, liefert die Unterabfrage genau einen Wert und darf im Vergleich als Operand angegeben werden.

```
auftrags_nr
       1012
```

Vergleich mit Ergebnisspalte

Der Wert eines Operanden wird mit den Werten einer Ergebnisspalte verglichen, die eine Unterabfrage liefert.





ausdruck
Ausdruck.
Der Ausdruck muß einen Wert ergeben, der mit den Elementen der Ergebnisspalte verglichen werden kann (siehe *Vergleich von zwei Werten*).

Einschränkung:
- Sie dürfen nicht die Konstante NULL angeben.
- Sie dürfen keine Spalte vom Datentyp TEXT oder BYTE angeben.

unterabfrage
Unterabfrage, die die Ergebnisspalte liefert, mit deren Werten *ausdruck* verglichen wird.

Ergebnis

ANY
NULL-Wert, wenn ein Operand der NULL-Wert ist.

Wahr, wenn der Vergleich mit mindestens einem Wert der Ergebnisspalte wahr ergibt.

Falsch, wenn die Ergebnisspalte leer ist oder der Vergleich mit allen Werten der Ergebnisspalte falsch ergibt.

SOME
Gleichbedeutend mit ANY.

ALL
NULL-Wert, wenn ein Operand der NULL-Wert ist.

Wahr, wenn die Ergebnisspalte leer ist oder der Vergleich mit allen Werten der Ergebnisspalte wahr ergibt.

Falsch, wenn der Vergleich mit mindestens einem Wert falsch ergibt.

Beispiel

Aus der Tabelle *posten* der Beispieldatenbank *versand* die Aufträge heraussuchen, die einen Posten enthalten, dessen Gesamtpreis höher ist als der Gesamtpreis aller Posten des Auftrags Nummer 1011.

```
SELECT UNIQUE auftrags_nr
       FROM posten
       WHERE gesamtpreis
       > ALL (SELECT gesamtpreis
              FROM posten
              WHERE auftrags_nr = 1011)
```

```
auftrags_nr
       1001
       1002
       1003
       1004
       1005
       1006
       1007
       1008
       1009
       1012
       1014
       1015
```

Bereichsabfrage

Es wird geprüft, ob der Wert des ersten Ausdrucks in dem angegebenen Wertebereich liegt.

```
ausdruck1 [ NOT] BETWEEN ausdruck2 AND ausdruck3
```

ausdruck1, *ausdruck2*, *ausdruck3*
Ausdrücke
Die Ausdrücke müssen vergleichbare Werte ergeben (siehe *Vergleich von zwei Werten*).

Einschränkung:
- Sie dürfen nicht die Konstante NULL angeben.
- Sie dürfen keine Spalte vom Datentyp TEXT oder BYTE angeben.

Ergebnis

Ohne NOT:
NULL-Wert, wenn ein Operand den NULL-Wert ergibt.

Wahr, wenn *ausdruck2* $\leq$ *ausdruck1* $\leq$ *ausdruck3*.

Falsch, sonst.

Mit NOT:
NULL-Wert, wenn ein Operand den NULL-Wert ergibt.

Wahr, wenn *ausdruck1* $<$ *ausdruck2* oder *ausdruck1* $>$ *ausdruck3*

Falsch, sonst.

Beispiel 1

Numerischen Bereich abprüfen:
Aus der Tabelle *artikel* der Beispieldatenbank *versand* alle Artikel mit Herstellercode und Preis heraussuchen, deren Preis zwischen 125.- und 200.- DM liegt:

```
SELECT artikel_nr, herstellercode, preis
       FROM artikel
       WHERE preis between 125.00 and 200.00
```

```
artikel_nr herstellercode    preis
         2 HRO              126,00
```

Beispiel 2

Datumsbereich abprüfen:
Aus den Tabellen *auftrag* und *posten* der Beispieldatenbank *versand* Kunde, Artikel, Herstellercode und Auftragsdatum der Aufträge heraussuchen, die in der Zeit zwischen 6.1.90 und 6.7.90 gestellt wurden:

```
SELECT UNIQUE kunden_nr, artikel_nr, herstellercode, auftragsdatum
       FROM auftrag, posten
       WHERE auftragsdatum
            BETWEEN "6/1/90" AND "6/7/90"
            AND auftrag.auftrags_nr = posten.auftrags_nr
```

```
kunden_nr artikel_nr herstellercode auftragsdatum
      104          1 HRO            20.01.1990
      101          4 HSK            01.06.1990
      101          3 HSK            01.06.1990
      106          1 HRO            12.04.1990
      106          2 HRO            12.04.1990
      106          3 HSK            12.04.1990
      106          1 HSK            12.04.1990
      117          1 HRO            25.03.1990
      117          2 HRO            25.03.1990
      117          3 HSK            25.03.1990
      117          4 HRO            25.03.1990
      117          7 HRO            25.03.1990
      111          1 SMT            14.02.1990
      115          6 SMT            29.05.1990
      115          6 ANZ            29.05.1990
      104          5 ANZ            23.03.1990
      117          8 ANZ            05.06.1990
      117          9 ANZ            05.06.1990
      106          4 HSK            01.05.1990
      106          4 HRO            01.05.1990
```

Beispiel 3

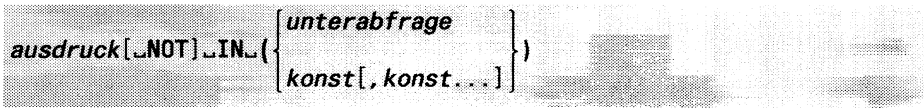
Alphanumerischen Bereich abprüfen:
Aus der Tabelle *kunde* der Beispieldatenbank *versand* die Kunden mit Postleitzahl heraussuchen, die in einem Ort sind, dessen Postleitzahl nicht zwischen 8070 und 8200 liegt:

```
SELECT vorname, nachname, plz
       FROM kunde
       WHERE plz
       NOT BETWEEN "8070" and "8200"
```

```
vorname         nachname        plz
Liane           Bergen          8300
Alfred          Grantella       8300
Dieter          Bachmann        8390
Ludwig          Pauli           8900
Janette         Millet          8900
Frank           Keyser          8900
Georg           Watt            7900
Johannes        Partellman      7900
Anton           Hochfeld        8000
Martin          Korting         8000
Roland          Jaeger          8000
Frank           Albert          8000
Arnold          Sipell          8000
```

Elementabfrage

Es wird geprüft, ob ein Wert unter den Elementen einer Menge vorkommt.



ausdruck
Ausdruck
Der Ausdruck muß einen Wert ergeben, der mit den Elementen der Menge vergleichbar ist (siehe *Vergleich von zwei Werten*).

unterabfrage
Unterabfrage.
Die Unterabfrage liefert die Menge der Elemente in Form einer Ergebnisspalte.

(*konst* [,*konst*...])
Menge von Konstanten, auf die sich die Elementabfrage bezieht.

Bei Programmeinbettung können Sie auch Hostvariablen angeben, die die Konstanten enthalten.

Einschränkung:
Sie dürfen nicht die Konstante NULL angeben.

Ergebnis

Ohne NOT:
NULL-Wert, wenn ein Operand den NULL-Wert ergibt.

Wahr, wenn der Wert des Ausdrucks in der angegebenen Menge von Werten vorkommt.

Diese Form entspricht dem Prädikat:
ausdruck = ANY(*unterabfrage*)
(siehe *Vergleich mit Ergebnisspalte*)

Falsch, sonst.

Mit NOT:
NULL-Wert, wenn ein Operand den NULL-Wert ergibt.

Wahr, wenn der Wert des Ausdrucks nicht in in der angegebenen Menge von Werten vorkommt.

Diese Form entspricht dem Prädikat:
ausdruck != ALL(*unterabfrage*)
(siehe *Vergleich mit Ergebnisspalte*)

Falsch, sonst.

Beispiel 1

Elementabfrage mit alphanumerischen Werten:
Aus der Tabelle *kunde* der Beispieldatenbank *versand* alle Kunden mit Firma und Ort heraussuchen, die aus Ulm oder Augsburg sind:

```
SELECT nachname, vorname, firma, ort
       FROM kunde
       WHERE ort IN ("Ulm","Augsburg")
```

```
nachname        vorname         firma               ort
Pauli           Ludwig          Pauli Sport         Augsburg
Watt            Georg           Sport Watt          Ulm
Millet          Janette         Sportausstattung    Augsburg
Keyser          Frank           Sport Keyser        Augsburg
Partellman      Johannes        Olympia Sport       Ulm
```

Beispiel 2

Elementabfrage mit Ergebnisspalte.
Aus den Tabellen *auftrag* und *posten* der Beispieldatenbank *versand* die Kunden heraussuchen, die keinen Artikel Nummer 1 (Ski-Handschuhe) bestellt haben:

```
SELECT UNIQUE kunden_nr
       FROM auftrag
       WHERE auftrags_nr
       NOT IN (SELECT auftrags_nr
               FROM posten
               WHERE artikel_nr = 1)
```

```
kunden_nr
      101
      104
      116
      112
      110
      115
      117
      106
```

Mustervergleich

Es wird geprüft, ob ein alphanumerischer Wert zu einem angegebenen Muster paßt.

Ein Muster ist eine Zeichenkette, die folgende Zeichen enthalten kann:

- Platzhalter
- Entwertungszeichen
- normale Zeichen.

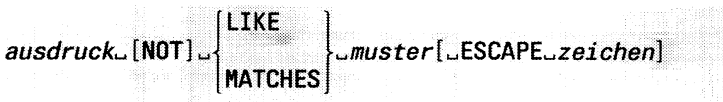
Platzhalter

Ein Platzhalter steht für ein oder mehrere andere Zeichen. Platzhalter können auch als normales Zeichen im Muster erscheinen, wenn sie mit dem Entwertungszeichen entwertet werden.
Die Platzhalter sind bei LIKE und MATCHES unterschiedlich.

Entwertungszeichen

Ein Entwertungszeichen hebt die Sonderbedeutung von Platzhaltern auf. Das vordefinierte Entwertungszeichen ist der Gegenschrägstrich \ (ASCII-Code 92). Beachten Sie, daß das Zeichen Ö im deutschen ISO-7-Bit-Code dem Gegenschrägstrich entspricht und deshalb entwertet werden muß.

Sie können mit der ESCAPE-Klausel ein eigenes Entwertungszeichen definieren.



ausdruck
Ausdruck
Der Ausdruck muß einen alphanumerischen Wert ergeben.

LIKE
Das Muster kann folgende Platzhalter enthalten:

Platzhalter	Bedeutung
_ (Unterstrich)	beliebiges Zeichen
%	beliebige (auch leere) Folge von Zeichen

MATCHES

Das Muster kann folgende Platzhalter enthalten:

Platzhalter	Bedeutung
?	ein beliebiges Zeichen
*	eine beliebige (auch leere) Folge von Zeichen
[{ zeichen | zeichen-zeichen } ...]	ein Zeichen, das zu den angegebenen Zeichen oder Zeichenbereichen gehört
[{ ^zeichen | ^zeichen-zeichen } ...]	ein Zeichen, das nicht zu den angegebenen Zeichen oder Zeichenbereichen gehört Die eckigen Klammern müssen Sie eingeben.

muster

Muster, zu dem der Wert von *ausdruck* passen soll.

ESCAPE-Klausel

Mit der ESCAPE-Klausel können Sie ein Entwertungszeichen definieren. Entwertungszeichen vor Platzhaltern bewirken, daß die Platzhalter ihre Funktion als Platzhalter verlieren und statt dessen als normale Zeichen interpretiert werden.

zeichen

Zeichen, das in dem Vergleich als Entwertungszeichen gelten soll. *zeichen* kann jedes abdruckbare Zeichen sein. Sie müssen das Zeichen in Anführungszeichen " oder Hochkomma ' setzen.

Bei Programmeinbettung können Sie *zeichen* auch über eine Hostvariable angeben.

ESCAPE-Klausel nicht angegeben:
Der Gegenschrägstrich \ ist das Entwertungszeichen.

Ergebnis

Ohne NOT:

NULL-Wert, wenn *ausdruck* den NULL-Wert ergibt.

Wahr, wenn der Wert des Ausdrucks zu dem Muster paßt.

Falsch, sonst.

Mit NOT:

NULL-Wert, wenn *ausdruck* den NULL-Wert ergibt.

Wahr, wenn der Wert des Ausdrucks nicht zu dem Muster paßt.

Falsch, sonst.

Beispiel 1

Aus der Tabelle *kunde* der Beispieldatenbank *versand* alle Kunden heraussuchen, deren Nachname mit *Kor* beginnt:

```
SELECT vorname, nachname
       FROM kunde
       WHERE nachname MATCHES "Kor*"
```

```
vorname         nachname
Philipp         Korres
Martin          Korting
```

Beispiel 2

Aus der Tabelle *kunde* der Beispieldatenbank *versand* alle Kunden mit Firma und Ort heraussuchen, deren Ortsanschrift mit einem Buchstaben zwischen A und L beginnt:

```
SELECT kunden_nr, firma, ort
       FROM kunde
       WHERE ort MATCHES "[A-L]*"
```

```
kunden_nr firma               ort
      101 Pauli Sport         Augsburg
      105 Der Laden           Ingolstadt
      109 Sportausstattung    Augsburg
      111 Sport Keyser        Augsburg
      112 Larsinger & Partner Ingolstadt
      113 Sporthaus           Landshut
      115 Sportladen          Landshut
```

Beispiel 3

Aus der Tabelle *auftrag* der Beispieldatenbank *versand* alle Aufträge heraussuchen, deren Lieferhinweis das Prozentzeichen % enthält:

```
INSERT INTO auftrag (auftrags_nr, lieferhinweis)
  VALUES(1111, "Lieferung nur 50%");

SELECT auftrags_nr, lieferhinweis
       FROM auftrag
       WHERE lieferhinweis LIKE "%!%%" ESCAPE "!"
```

```
auftrags_nr   lieferhinweis
       1111   Lieferung nur 50%
```

Vergleich auf NULL

Es wird geprüft, ob eine Spalte den NULL-Wert enthält.

spalte
Spalte, die auf NULL-Werte überprüft werden soll. Die Spalte kann jeden SQL-Datentyp haben, BLOB-Datentypen sind auch erlaubt.

Ergebnis

Ohne NOT:
Wahr, wenn der Wert der Spalte der NULL-Wert ist.

Falsch, sonst.

Mit NOT:
Wahr, wenn der Wert der Spalte nicht der NULL-Wert ist.

Falsch, sonst.

Beispiel

Aus der Tabelle *auftrag* der Beispieldatenbank *versand* Kunde und Auftragsdatum zu Aufträgen heraussuchen, deren Zahlungsdatum noch unbekannt, d.h der NULL-Wert ist:

```
SELECT kunden_nr, auftragsdatum
       FROM auftrag
       WHERE zahldatum IS NULL
```

```
kunden_nr auftragsdatum
      106 12.04.1990
      112 19.09.1990
      117 25.03.1990
      117 05.06.1990
```

Existenzabfrage

Es wird geprüft, ob eine Ergebnistabelle leer ist.

```
[NOT␣]EXISTS(select-anweisung)
```

select-anweisung
SELECT-Anweisung.

Einschränkung:
Die SELECT-Anweisung darf keine ORDER BY-, INTO-, INTO TEMP- und UNION-Klausel enthalten.

Ergebnis

Ohne NOT:
Wahr, wenn die Ergebnistabelle nicht leer ist.

Falsch, wenn die Ergebnistabelle leer ist.

Mit NOT:
Wahr, wenn die Ergebnistabelle leer ist.

Falsch, wenn die Ergebnistabelle nicht leer ist.

Beispiel

Aus den Tabellen *kunde* und *auftrag* der Beispieldatenbank *versand* alle Kunden heraussuchen, die keinen Auftrag vergeben haben:

```
SELECT nachname
    FROM kunde
    WHERE NOT EXISTS
    (SELECT auftrags_nr
            FROM auftrag, kunde
            WHERE auftrag.kunden_nr = kunde.kunden_nr)
```

Es ist kein Kunde ohne Auftrag vorhanden:

```
nachname
```

5.5 Bedingung

Bedingungen dienen dazu, die Menge der Sätze einzuschränken, die von einer Tabellenoperation betroffen sind. Es werden nur die Sätze berücksichtigt, die die angegebene Bedingung erfüllen. Sie können Bedingungen angeben beim Löschen (DELETE), Ändern (UPDATE) und Auswählen von Sätzen (SELECT).

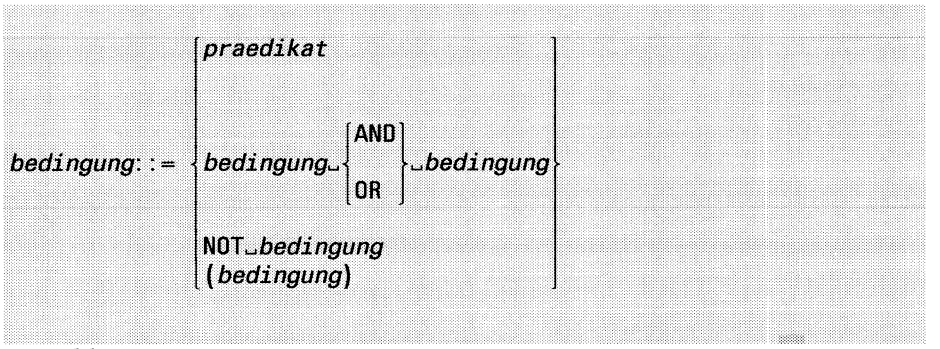
Eine für relationale Datenbanken charakteristische Bedingung beim Auswählen von Sätzen ist die Join-Bedingung. Die Join-Bedingung ist ein Vergleich von Spaltenwerten und dient dazu, aus dem Kartesischen Produkt mehrerer Tabellen nur die Sätze auszuwählen, deren Spalten die Join-Bedingung erfüllen. Der Join ist gesondert im Abschnitt 5.6 *Join* beschrieben.

Sie formulieren die Bedingung in einer WHERE- oder HAVING-Klausel, die im einzelnen in folgenden Anweisungen vorkommen können:

- WHERE-Klausel
  - DELETE-Anweisung
  - UPDATE-Anweisung
  - SELECT-Anweisung und in SELECT-Abfragen bei CREATE VIEW, INSERT, OUTPUT, UNLOAD
- HAVING-Kausel
  SELECT-Anweisung und in SELECT-Abfragen bei CREATE VIEW, INSERT, OUTPUT, UNLOAD

Eine Bedingung besteht aus Prädikaten und eventuell logischen Operatoren. Die Prädikate sind die Operanden der logischen Operatoren. Eine Bedingung wird wie folgt ausgewertet:

- Zuerst werden die Operanden berechnet.
- Sind Operatoren vorhanden, werden sie anschließend auf die berechneten Werte angewendet.
- Das Ergebnis ist ein Wahrheitswert wahr, falsch oder der NULLWert.
  In der Operatorbeschreibung ist jeweils angegeben, welches Ergebnis die Auswertung für NULL-Werte liefert.
- Eine Bedingung, die als Ergebnis den NULL-Wert liefert, zählt als falsch.



praedikat
Prädikat.

AND
Logisches UND.

Ergebnis:
NULL-Wert, wenn ein Operand wahr und ein Operand der NULL-Wert ist.

Wahr, wenn beide Operanden wahr sind.

Falsch, wenn ein Operand falsch ist.

OR
Logisches ODER.

Ergebnis:
NULL-Wert, wenn ein Operand wahr und ein Operand der NULL-Wert ist.

Wahr, wenn ein Operand wahr ist.

Falsch, wenn beide Operanden falsch sind.

NOT
Negation.

Ergebnis:
NULL-Wert, wenn der Operand der NULL-Wert ist.

Wahr, wenn der Operand falsch ist.

Falsch, wenn der Operand wahr ist.

Prioritäten

- Klammerausdrücke haben die höchste Priorität.
- NOT hat Vorrang vor AND und OR.
- AND hat Vorrang vor OR.
- Bei mehreren Operatoren gleicher Priorität wird von links nach rechts ausgewertet.

Beispiel 1

Aus den Tabellen *auftrag* und *kunde* der Beispieldatenbank *versand* alle Aufträge mit der zugehörigen Firma heraussuchen, die nach dem 6.4.90 gestellt wurden.

```
SELECT auftrags_nr, firma, auftragsdatum
       FROM auftrag a, kunde k
       WHERE a.auftragsdatum > "6/04/90"
       AND a.kunden_nr = k.kunden_nr
```

```
auftrags_nr firma                 auftragsdatum
       1002 Pauli Sport           01.06.1990
       1003 Spielball             12.10.1990
       1004 Sport Watt            12.04.1990
       1005 Olympia Sport         04.12.1990
       1006 Larsinger & Partner   19.09.1990
       1008 Sportwaren            17.11.1990
       1010 Sportladen            29.05.1990
       1012 Sportecke             05.06.1990
       1013 Spielball             01.09.1990
       1014 Sport Watt            01.05.1990
       1015 Sportwaren            10.07.1990
```

Beispiel 2

Aus der Tabelle *posten* der Beispieldatenbank *versand* alle Posten und zugehörige Aufträge heraussuchen, deren Gesamtpreis größer als 1000.00 ist und deren Herstellercode aus dem Buchstaben H und einer beliebigen Zeichenfolge besteht.

```
SELECT posten_nr, auftrags_nr, gesamtpreis
       FROM posten
       WHERE gesamtpreis > 1000.00
       AND herstellercode LIKE "H%"
```

Es gibt keinen Satz, der beide Bedingungen erfüllt:

```
posten_nr auftrags_nr gesamtpreis
```

Beispiel 3

Aus der Tabelle *kunde* der Beispieldatenbank *versand* alle Kunden mit Anschrift heraussuchen, die aus einem Ort mit einer Postleitzahl zwischen 8070 und 8300 sind oder nicht aus Muenchen, Ulm und Passau:

```
SELECT nachname, kunden_nr, plz, ort
       FROM kunde
       WHERE plz BETWEEN "8070" AND "8300"
       OR ort NOT IN("Muenchen","Ulm","Passau")
```

```
nachname        kunden_nr plz   ort
Pauli                 101 8900  Augsburg
Sadler                102 8090  Wasserburg
Korres                103 8200  Rosenheim
Viktor                105 8070  Ingolstadt
Remark                107 8200  Rosenheim
Millet                109 8900  Augsburg
Keyser                111 8900  Augsburg
Larsinger             112 8070  Ingolstadt
Bergen                113 8300  Landshut
Grantella             115 8300  Landshut
```

5.6 Join

Ein Join besteht aus einer Join-Bedingung. Sie wird in der WHERE-Klausel einer SELECT-Abfrage über mehrere Tabellen angegeben und dient dazu, aus dem Kartesischen Produkt von mehreren Tabellen ausgewählte Sätze herauszusuchen.

Eine Join-Bedingung vergleicht korrespondierende Spalten aus den verknüpften Tabellen. Die Spalten heißen **Join-Spalten**. Bei einem Join werden alle Sätze ausgewählt, deren Join-Spalten den Vergleich erfüllen.

Dieselbe Tabelle kann mehrmals vorkommen. Ein Join, der eine Tabelle mit sich selbst verknüpft, heißt **Self-Join**.

Die Join-Bedingung kann einfach oder zusammengesetzt sein, man spricht dann von einfachen bzw. zusammengesetzten Joins.

Einfacher Join

Ein einfacher Join wählt Sätze aus dem Kartesischen Produkt von zwei Tabellen aus.

Die dazugehörige einfache Join-Bedingung, ist ein Vergleich von zwei Spalten aus zwei Tabellen.

Wie bei jedem Vergleich, müssen die Werte der Join-Spalten vergleichbar sein.
Darüberhinaus sollten die Spalten inhaltlich zusammenpassen, sodaß zwischen den abgefragten Daten eine sinnvolle Beziehung besteht.
Für die Tabellen der Beispieldatenbank sind die sinnvollen Join-Spalten im Anhang A.1 angezeigt.

Zusammengesetzter Join

Ein zusammengesetzter Join wählt Sätze aus dem Kartesischen Produkt von mehreren Tabellen aus.

Die dazugehörige Join-Bedingung ist eine mit AND oder OR zusammengesetzte Bedingung aus einfachen Join-Bedingungen.

Die Reihenfolge, in der einfache Join-Bedingungen angegeben werden, ist ohne Bedeutung (siehe unten, Beispiele).

Join-Typen

INFORMIX unterstützt den normalen Join und den Outer-Join. Für beide Typen und ihre Kombination sind im folgenden einfache und zusammengesetzte Joins erklärt und am Beispiel verdeutlicht.

Normaler Join

Bei einem normalen Join enthält die Ergebnistabelle nur die Sätze, deren Join-Spalten die Join-Bedingung erfüllen.

Beispiel:

Kundenname und zugehörige Auftragsnummer aus den Tabellen *kunde* und *auftrag* der Beispieldatenbank *versand* heraussuchen:

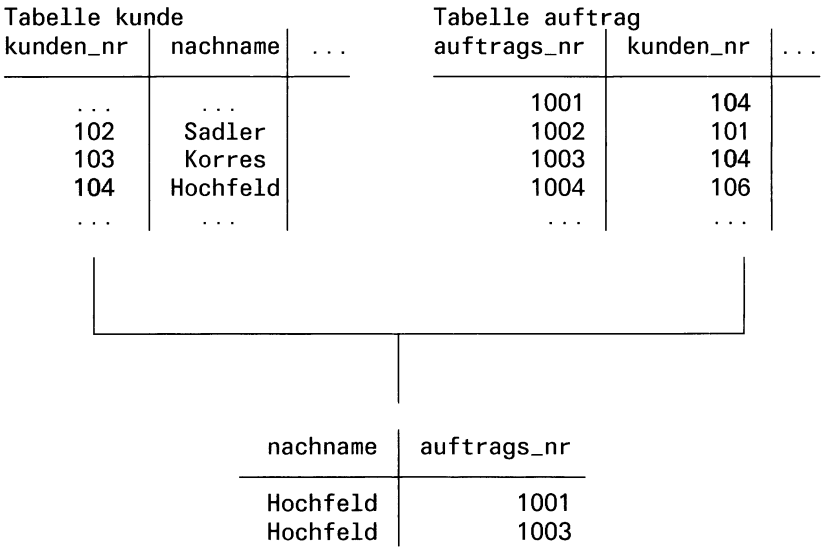


Tabelle kunde

kunden_nr	nachname	...
...	...	
102	Sadler	
103	Korres	
104	Hochfeld	
...	...	

Tabelle auftrag

auftrags_nr	kunden_nr	...
1001	104	
1002	101	
1003	104	
1004	106	
...	...	

nachname	auftrags_nr
Hochfeld	1001
Hochfeld	1003

```
SELECT nachname, auftrags_nr
  FROM kunde, auftrag
  WHERE kunde.kunden_nr=auftrag.kunden_nr
```

Kunden, die keinen Auftrag erteilt haben, zum Beispiel Kunde Korres mit Nummer 103, sind in der Ergebnistabelle nicht enthalten.

```
nachname     auftrags_nr
Hochfeld           1001
Pauli              1002
Hochfeld           1003
Watt               1004
. . .
```

Zusammengesetzter normaler Join

Bei einem zusammengesetzten Join können mehrere Spalten in Bezug zu derselben Spalte verglichen werden. Diese Join-Spalte wird dann in jeder Teil-Join-Bedingung verwendet.

Beispiel:

Aus den Tabellen *hersteller*, *posten* und *auftrag* der Beispieldatenbank *versand* Herstellercode, Postennummer und Artikelbezeichnung für Herstellercodes heraussuchen, die in allen drei Tabellen vorkommen:

```
SELECT h.herstellercode, p.posten_nr, a.bezeichnung
  FROM hersteller h, artikel a, posten p
  WHERE h.herstellercode=p.herstellercode
  AND h.herstellercode=a.herstellercode
  AND h.herstellercode LIKE "N%"
```

```
herstellercode   posten_nr      bezeichnung
NRG                     1      Tennisschlaeger
NRG                     2      Tennisschlaeger
```

Außerdem können bei einem zusammengesetzten Join in jeder Teil-Join-Bedingung unterschiedliche Join-Spalten verwendet werden.

Beispiel:

Folgendes Beispiel sucht aus den Tabellen *kunde*, *auftrag* und *posten* der Beispieldatenbank *versand* für jeden Kunden, der einen Auftrag gestellt hat, die zugehörigen Posten heraus.

```
SELECT k.nachname, a.auftrags_nr, p.posten_nr
   FROM kunde k, auftrag a, posten p
   WHERE k.kunden_nr=a.kunden_nr
   AND a.auftrags_nr=p.auftrags_nr
```

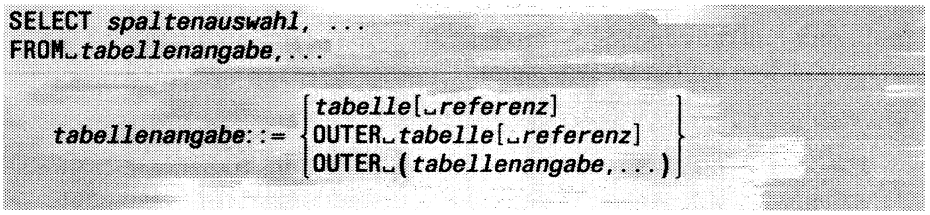
```
nachname     auftrags_nr      posten_nr
Hochfeld            1001              1
Pauli               1002              1
Pauli               1002              2
Hochfeld            1003              1
Hochfeld            1003              2
Hochfeld            1003              3
Watt                1004              1
. . .
```

Obiges Beispiel liefert dasselbe Ergebnis, wenn Sie es wie folgt formulieren:

```
SELECT k.nachname, a.auftrags_nr, p.posten_nr
   FROM kunde k, auftrag a, posten p
   WHERE a.auftrags_nr=p.auftrags_nr
   AND k.kunden_nr=a.kunden_nr
```

Outer-Join

Eine besondere Form des Join ist der Outer-Join. Er wird gebildet, indem Sie in der Tabellenangabe der SELECT-Abfrage das Schlüsselwort OUTER verwenden (siehe Kapitel 6, SELECT-Anweisung).



Im Gegensatz zum normalen Join gilt bei einem Outer-Join:

Es gibt eine **dominierende** Tabelle und eine **abhängige** Tabelle. Gibt es für Werte in der Join-Spalte der dominierenden Tabelle keinen übereinstimmenden Wert in der abhängigen Tabelle, bleibt der Satz trotzdem erhalten. Für die nicht vorhandenen Werte in der abhängigen Tabelle wird der NULL-Wert eingesetzt.

Beispiel:

Wie im ersten Join-Beispiel möchten Sie aus den Tabellen *kunde* und *auftrag* der Beispieldatenbank *versand* Kundenname und zugehörige Auftragsnummer heraussuchen. Es sollen aber alle Kunden aufgelistet werden, auch solche, die noch keinen Auftrag gestellt haben. Dazu formulieren Sie folgenden Outer-Join:

```
SELECT nachname, auftrags_nr
  FROM kunde, OUTER auftrag
  WHERE kunde.kunden_nr=auftrag.kunden_nr
```

Kunden, die keinen Auftrag erteilt haben, zum Beispiel Kunde Korres mit Nummer 103, sind jetzt in der Ergebnistabelle enthalten. Für die fehlende Auftragsnummer ist der NULL-Wert eingetragen.

```
nachname  auftrags_nr
Pauli           1002
Sadler
Korres
Hochfeld        1001
Hochfeld        1003
Hochfeld        1011
. . . .
```

Zusammengesetzter Outer-Join

Wenn Sie mehr als zwei Tabellen über Joins verbinden, können Sie mehrmals OUTER verwenden. Sie können lineare oder geschachtelte Outer-Joins bilden.

Bei einem linearen Outer-Join gibt es mehrere abhängige Tabellen zu einer dominanten Tabelle.
In diesem Fall muß in jedem Join eine Join-Spalte der dominanten Tabelle verwendet werden (siehe Beispiele).

Beispiel:

Um den Effekt des Outer-Join zu verdeutlichen, tragen Sie folgende Sätze in die Tabellen *hersteller* und *artikel* der Beispieldatenbank *versand* ein:

```
INSERT INTO hersteller VALUES("NPP", "Nupp");
INSERT INTO posten  (posten_nr, herstellercode)
  VALUES(99,"NPP")
```

Jetzt enthält die Hersteller-Tabelle und die Posten-Tabelle den Herstellercode *NPP*, der nicht in der Artikel-Tabelle vorkommt.

Die Herstellertabelle ist die dominante Tabelle, die Posten- und Artikel-Tabelle sind abhängige Tabellen.

Beachten Sie, daß Sie in jeder Join-Bedingung die Join-Spalte *h.herstellercode* der dominanten Tabelle angeben müssen.

```
SELECT h.herstellercode, p.posten_nr, a.bezeichnung
  FROM hersteller h, OUTER artikel a, OUTER posten p
  WHERE h.herstellercode=p.herstellercode
  AND h.herstellercode=a.herstellercode
  AND h.herstellercode LIKE "N%"
```

```
herstellercode     posten_nr      bezeichnung
NPP                        99
NRG                         1     Tennisschlaeger
NRG                         2     Tennisschlaeger
```

Bei einem geschachtelten Outer-Join können Sie durch Klammerung der Tabellen, die Sie bei OUTER angeben, eine Hierarchie von dominierenden und abhängigen Tabellen festlegen.

Geben Sie geklammerte Tabellen hinter OUTER an, werden zuerst die Joins oder Outer-Joins der Tabellen in der Klammer berechnet und dann

die erzeugte Ergebnistabelle als abhängige Tabelle eingesetzt. Sie können bei den inneren Tabellen wieder OUTER, auch geschachtelt, angeben, sodaß eine Hierarchie von Outer-Joins ensteht (siehe Kapitel 6, SELECT). Wenn Sie kein OUTER angeben, wird ein normaler Join berechnet.

Die folgenden Beispiele suchen aus den drei Tabellen Herstellercode, Postennummer und zugehörige Artikelbezeichnung heraus. Sie sehen die unterschiedlichen Ergebnisse abhängig davon, welche Joins verwendet werden.
Um die Unterschiede deutlicher herauszustellen, beschränken wir die Abfragen auf die Herstellercodes, die mit dem Buchstaben N beginnen.

Beispiele:

1. Es werden nur Herstellercodes berücksichtigt, die in allen drei Tabellen vorkommen:

```
SELECT h.herstellercode, p.posten_nr, a.bezeichnung
  FROM hersteller h, artikel a, posten p
  WHERE h.herstellercode=p.herstellercode
  AND p.herstellercode=a.herstellercode
  AND h.herstellercode LIKE "N%"
```

```
herstellercode   posten_nr      bezeichnung
NRG                     1       Tennisschlaeger
NRG                     2       Tennisschlaeger
```

2. Die Posten-Tabelle ist die dominante Tabelle. Das heißt, es werden alle Herstellercodes berücksichtigt, die in der Posten-Tabelle vorkommen, auch wenn sie nicht in der Artikel-Tabelle sind, wobei für die fehlende Artikelbezeichnung der Null-Wert eingetragen wird:

```
SELECT p.herstellercode, p.posten, a.bezeichnung
  FROM posten p, OUTER artikel a
  WHERE p.herstellercode=a.herstellercode
  AND p.herstellercode LIKE "N%"
```

```
herstellercode   posten_nr      bezeichnung
NRG                     1       Tennisschlaeger
NRG                     2       Tennisschlaeger
NPP                    99
```

3. Die Hersteller-Tabelle ist die dominante Tabelle. Das heißt, es werden alle Herstellercodes berücksichtigt, die in der Hersteller-Tabelle vorkommen. Bei Postennummer und Artikelbezeichnung werden allerdings nur die Herstellercodes berücksichtigt, die sowohl in der Posten- als auch in der Artikel-Tabelle enthalten sind:

```
SELECT h.herstellercode, p.posten_nr, a.bezeichnung
  FROM hersteller h, OUTER (artikel a, posten p)
  WHERE h.herstellercode=p.herstellercode
  AND p.herstellercode=a.herstellercode
  AND h.herstellercode LIKE "N%"
```

NRG ist der einzige Herstellercode, der sowohl in der Posten- als auch in der Artikel-Tabelle vorkommt. Zusätzlich enthält die Ausgabe den Herstellercode *NPP* aus der Hersteller-Tabelle. Für die fehlenden Postennummer und Bezeichnung sind NULL-Werte eingesetzt.

```
herstellercode     posten_nr      bezeichnung
NPP
NRG                        1      Tennisschlaeger
NRG                        2      Tennisschlaeger
```

4. Die Hersteller-Tabelle ist die oberste dominante Tabelle. Das heißt, es werden alle Herstellercodes berücksichtigt, die in der HerstellerTabelle vorkommen. Zusätzlich ist die Posten-Tabelle innere dominante Tabelle. Das heißt, es werden alle Posten berücksichtigt, auch, wenn ihr Herstellercode nicht in der Artikel-Tabelle vorkommt.

```
SELECT h.herstellercode, p.posten_nr, a.bezeichnung
  FROM hersteller h, OUTER (artikel a, OUTER posten p)
  WHERE h.herstellercode=p.herstellercode
  AND p.herstellercode=a.herstellercode
  AND h.herstellercode LIKE "N%"
```

Den Unterschied zu Beispiel 3 erkennen Sie am Ergebnis, das den Posten 99 enthält:

```
herstellercode     posten_nr      bezeichnung
NPP                       99
NRG                        1      Tennisschlaeger
NRG                        2      Tennisschlaeger
```

6 SQL-Anweisungen

6.1 Inhaltliche Zusammenstellung
6.2 Zusammenstellung nach Verwendungsmöglichkeit
6.3 Zusammenstellung nach Backend
6.4 Erweiterung des ANSI-Standards
6.5 Beschreibungsformat und Parameterbezeichnungen
6.6 Alphabetischer Nachschlageteil

Kapitel 6 beschreibt die INFORMIX-SQL-Anweisungen.

6.1 Inhaltliche Zusammenstellung

Datendefinition

SQL-Anweisung	Bedeutung
CREATE DATABASE	Datenbank erzeugen
DATABASE	Datenbank eröffnen
CLOSE DATABASE	Datenbank schließen
CREATE TABLE	Tabelle erzeugen
CREATE INDEX	Index erzeugen
CREATE SCHEMA	Datenbankschema erzeugen
CREATE SYNONYM	Synonym definieren
CREATE VIEW	View erzeugen
ALTER INDEX	Indexdefinition ändern
ALTER TABLE	Tabellendefinition ändern
RENAME COLUMN	Spaltennamen ändern
RENAME TABLE	Tabellennamen ändern
DROP TABLE	Tabelle löschen
DROP INDEX	Index löschen
DROP SYNONYM	Synonym löschen
DROP VIEW	View löschen
DROP DATABASE	Datenbank löschen
UPDATE STATISTICS	Satzanzahl aktualisieren

Datenabfrage

SQL-Anweisung	Bedeutung
SELECT	Daten abfragen
SET EXPLAIN	Suchstrategie ausgeben
FETCH	Satzzeiger positionieren
OUTPUT	Sätze in Datei ausgeben
UNLOAD	Sätze in Datei ausgeben
INFO	Tabellen-Information ausgeben

Datenänderung

SQL-Anweisung	Bedeutung
DELETE	Sätze löschen
INSERT	Sätze einfügen
UPDATE	Spaltenwerte ändern
LOAD	Sätze aus Datei einfügen

Zugriffsrechte

SQL-Anweisung	Bedeutung
GRANT	Zugriffsrechte vergeben
REVOKE	Zugriffsrechte entziehen
INFO	Tabellen-Information ausgeben

Datenintegrität

SQL-Anweisung	Bedeutung
START DATABASE	Transaktionssicherung einschalten
BEGIN WORK	Transaktion starten
COMMIT WORK	Transaktion beenden
ROLLBACK WORK	Transaktion rücksetzen
SET ISOLATION	Isolationsstufe wählen
SET LOCK MODE	Warte-Modus definieren
SET LOG	Protokollierungsmodus ändern
LOCK TABLE	Tabelle sperren
UNLOCK TABLE	Tabellensperre aufheben
CHECK TABLE	Tabelle prüfen
REPAIR TABLE	Index wiederherstellen

Datensicherung

SQL-Anweisung	Bedeutung
START DATABASE	Transaktionssicherung einschalten
ROLLFORWARD DATABASE	Datenbanksicherung aktualisieren
CREATE AUDIT	Audit-Protokoll erzeugen
RECOVER TABLE	Tabelle aktualisieren
DROP AUDIT	Audit-Protokoll löschen

Satzzeiger

SQL-Anweisung	Bedeutung
DECLARE	Satzzeiger vereinbaren
OPEN	Satzzeiger öffnen
CLOSE	Satzzeiger schließen
FETCH	Satzzeiger positionieren
PUT	Satz in Einfügepuffer schreiben
FLUSH	Einfügepuffer leeren
FREE	Betriebsmittel freigeben

Dynamische Bearbeitung von Anweisungen

SQL-Anweisung	Bedeutung
DESCRIBE	Anweisung analysieren
PREPARE	dynamische Anweisung vorbereiten
EXECUTE	dynamische Anweisung ausführen
EXECUTE IMMEDIATE	dynamische Anweisung sofort ausführen
FREE	Betriebsmittel freigeben

6.2 Zusammenstellung nach Verwendungsmöglichkeit

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

6.3 Zusammenstellung nach Backend

INFORMIX-SE und INFORMIX-ONLINE

SQL-Anweisung	Bedeutung
ALTER INDEX	Indexdefinition ändern
ALTER TABLE	Tabellendefinition ändern
BEGIN WORK	Transaktion starten
CLOSE	Satzzeiger schließen
CLOSE DATABASE	Datenbank schließen
COMMIT WORK	Transaktion beenden
CREATE DATABASE	Datenbank erzeugen
CREATE INDEX	Index erzeugen
CREATE SCHEMA	Datenbankschema erzeugen
CREATE SYNONYM	Synonym definieren
CREATE TABLE	Tabelle erzeugen
CREATE VIEW	View erzeugen
DATABASE	Datenbank eröffnen
DECLARE	Satzzeiger vereinbaren
DELETE	Sätze löschen
DESCRIBE	Anweisung analysieren
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
EXECUTE	dynamische Anweisung ausführen
EXECUTE IMMEDIATE	dynamische Anweisung sofort ausführen
FETCH	Satzzeiger positionieren
FLUSH	Einfügepuffer leeren
FREE	Betriebsmittel freigeben
GRANT	Zugriffsrechte vergeben
INFO	Tabellen-Information ausgeben
INSERT	Sätze einfügen
LOAD	Sätze aus Datei einfügen
LOCK TABLE	Tabelle sperren
OPEN	Satzzeiger öffnen
OUTPUT	Sätze in Datei ausgeben
PREPARE	dynamische Anweisung vorbereiten
PUT	Satz in Einfügepuffer schreiben
RENAME COLUMN	Spaltennamen ändern
RENAME TABLE	Tabellennamen ändern
REVOKE	Zugriffsrechte entziehen
ROLLBACK WORK	Transaktion rücksetzen

SQL-Anweisung	Bedeutung
SELECT	Daten abfragen
SET EXPLAIN	Suchstrategie ausgeben
SET LOCK MODE	Warte-Modus definieren
UNLOAD	Sätze in Datei ausgeben
UNLOCK TABLE	Tabellensperre aufheben
UPDATE	Spaltenwerte ändern
UPDATE STATISTICS	Satzanzahl aktualisieren

INFORMIX-SE

SQL-Anweisung	Bedeutung
CHECK TABLE	Tabelle prüfen
CREATE AUDIT	Audit-Protokoll erzeugen
RECOVER TABLE	Tabelle aktualisieren
REPAIR TABLE	Index wiederherstellen
ROLLFORWARD DATABASE	Datenbanksicherung aktualisieren
START DATABASE	Transaktionssicherung einstellen

INFORMIX-ONLINE

SQL-Anweisung	Bedeutung
SET ISOLATION	Isolationsstufe wählen
SET LOG	Protokollierungsmodus ändern

6.4 Erweiterungen des ANSI-Standards

SQL-Anweisung	Bedeutung
ALTER INDEX	Indexdefinition ändern
ALTER TABLE	Tabellendefinition ändern
BEGIN WORK	Transaktion starten
CHECK TABLE	Tabelle prüfen
CLOSE DATABASE	Datenbank schließen
CREATE AUDIT	Audit-Protokoll erzeugen
CREATE DATABASE	Datenbank erzeugen
CREATE INDEX	Index erzeugen
CREATE SYNONYM	Synonym definieren
DATABASE	Datenbank eröffnen
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
EXECUTE IMMEDIATE	dynamische Anweisung sofort ausführen
FLUSH	Einfügepuffer leeren
FREE	Betriebsmittel freigeben
INFO	Tabellen-Information ausgeben
LOAD	Sätze aus Datei einfügen
LOCK TABLE	Tabelle sperren
OUTPUT	Sätze in Datei ausgeben
PUT	Satz in Einfügepuffer schreiben
RECOVER TABLE	Tabelle aktualisieren
RENAME COLUMN	Spaltennamen ändern
RENAME TABLE	Tabellennamen ändern
REPAIR TABLE	Index wiederherstellen
REVOKE	Zugriffsrechte entziehen
ROLLFORWARD DATABASE	Datenbanksicherung aktualisieren
SET EXPLAIN	Suchstrategie ausgeben
SET ISOLATION	Isolationsstufe wählen
SET LOCK MODE	Warte-Modus definieren
SET LOG	Protokollierungsmodus ändern
START DATABASE	Transaktionssicherung einstellen
UNLOAD	Sätze in Datei ausgeben
UNLOCK TABLE	Tabellensperre aufheben
UPDATE STATISTICS	Satzanzahl aktualisieren

Die Anweisungen CREATE TABLE, CREATE VIEW und GRANT sind im ANSI-Standard enthalten, dort aber im Gegensatz zu INFORMIX nur innerhalb der Anweisung CREATE SCHEMA erlaubt.

Die folgenden Anweisungen sind im ANSI-Standard enthalten, einige ihrer Schlüsselwörter aber nicht.

SQL-Anweisung	Schlüsselwort nicht im ANSI-Standard
CREATE TABLE	TEMP WITH NO LOG UNIQUE CONSTRAINT DISTINCT IN *"datei"* IN *dbspace* EXTENT SIZE NEXT SIZE LOCK MODE Datentypen: DATE MONEY SMALLFLOAT DATETIME INTERVAL VARCHAR BYTE TEXT
DECLARE	SCROLL WITH HOLD FOR UPDATE INSERT
GRANT	AS *benutzer* CONNECT RESOURCE DBA

SQL-Anweisung	Schlüsselwort nicht im ANSI-Standard
SELECT	UNIQUE OUTER INTO TEMP UNITS MATCHES Verwendung von Spaltennummern Ausschnitte aus Spaltenwerten Funktionen: DAY() MONTH() YEAR() WEEKDAY() DATE() MDY() TODAY LENGTH() CURRENT EXTEND() SITENAME
UPDATE	Verwendung von Spaltenlisten

6.5 Beschreibungsformat und Parameter

In diesem Abschnitt sind die Konventionen für die Anweisungsbeschreibungen im Nachschlageteil erklärt.

6.5.1 Beschreibungsformat

Die SQL-Anweisungen sind in einem einheitlichen Stil beschrieben. Sie sind alphabetisch angeordnet und pro Anweisung gibt es in den meisten Fällen einen Eintrag. Ausnahmen sind lediglich die Anweisungen CREATE DATABASE und SELECT, die mehrere Einträge benötigen.

Unterschiede der Produkte

Jeder Eintrag hat eine Kopfzeile, die den Namen der Anweisung enthält.

Da bei der Beschreibung alle INFORMIX-Komponenten berücksichtigt werden, die eine SQL-Schnittstelle anbieten, gibt es folgende Konventionen, um die Unterschiede kenntlich zu machen:

- Zusatz in der Kopfzeile
- Zusatz in der Syntax

Zusatz in der Kopfzeile:

- Existiert die Anweisung in allen Produkten, hat der Name der Anweisung in der Kopfzeile keinen Zusatz
- Existiert die Anweisung nur bei INFORMIX-SE, hat der Name der Anweisung in der Kopfzeile den Zusatz **(SE)**
- Existiert die Anweisung nur bei INFORMIX-ONLINE, hat der Name der Anweisung in der Kopfzeile den Zusatz **(ONLINE)**
- Existiert die Anweisung nur im interaktiven Betrieb, hat der Name der Anweisung in der Kopfzeile den Zusatz **(interaktiv)**
- Existiert die Anweisung nur bei INFORMIX-SE im interaktiven Betrieb, hat der Name der Anweisung in der Kopfzeile den Zusatz **(SE, interaktiv)**
- Existiert die Anweisung nur bei Programmeinbettung, hat der Name der Anweisung in der Kopfzeile den Zusatz **(eingebettet)**

Für die Ausnahmen CREATE DATABASE und SELECT gilt:

- Für CREATE DATABASE gibt es je einen Eintrag CREATE DATABASE (SE) und CREATE DATABASE (ONLINE), da die Anweisung für die beiden Backends sehr verschieden ist.
- Für SELECT gibt es einen Eintrag pro Klausel (SELECT/Spaltenauswahl, SELECT/FROM, usw.), da die Syntax so umfangreich ist, daß ein Eintrag zu unübersichtlich würde.

Zusatz in der Syntax:

Gibt es die Anweisung bei allen INFORMIX-Komponenten, allerdings mit Unterschieden in der Syntax, sind die unterschiedlichen Klauseln in der Syntax durch einen Zusatz gekennzeichnet. Es gibt folgende Zusätze:

- nur ONLINE
  Die Klausel gibt es für die Anweisung nur bei INFORMIX-ONLINE.
- nur SE
  Die Klausel gibt es für die Anweisung nur bei INFORMIX-SE.
- nur eingebettet
  Die Klausel gibt es für die Anweisung nur bei Programmeinbettung.

Aufbau eines Eintrags

Jeder Eintrag besteht aus mehreren Abschnitten.

Bei einem Eintrag können Abschnitte fehlen, wenn Sie für die entsprechende Anweisung keine Bedeutung haben.

Außerdem können nach der Erklärung der Syntax weitere Abschnitte vorhanden sein, die Besonderheiten der jeweiligen Anweisung beschreiben.

Auf den folgenden Seiten sind die wichtigsten Abschnitte zusammengestellt.

Anweisungsname - Kurzbeschreibung

Nach der Überschrift wird die Wirkungsweise der Anweisung beschrieben.

Vor dem Aufruf beachten

Dieser Abschnitt enthält wichtige Voraussetzungen, die für die erfolgreiche Ausführung der Anweisung erfüllt sein müssen. Insbesondere sind die notwendigen Zugriffsrechte zusammengestellt.

```
ANWEISUNGSNAME␣KLAUSEL␣parameter . . .
```

```
Syntaxdefinition für Teile der Anweisung, die in diesem Eintrag
nur aufgelistet, aber nicht beschrieben werden. Die Beschreibung
ist an einer anderen Stelle im Handbuch zu finden.
```

parameter
Erklärungen, zu dem Parameter.

Die Klauseln und Parameter sind in der Reihenfolge beschrieben, in der sie in der Syntaxdefinition vorkommen.

Anweisung **in Transaktionen**

In diesem Abschnitt wird beschrieben, wie die Anweisung in einer Transaktion verwendet werden kann und welche Besonderheiten berücksichtigt werden müssen, zum Beispiel, ob die Anweisung zurückgesetzt werden kann oder nicht.

Sperren beim ...

In diesem Abschnitt wird beschrieben, welche Sperren bei Ausführung der Anweisung gesetzt werden.

Erfolgskontrolle

Dieser Abschnitt ist für Programmeinbettung. Er gibt an, aus welchen Komponenten der *sqlca*-Struktur nach Ausführung anweisungsspezifische Informationen über die Ausführung abgefragt werden können. Die Komponente *sqlcode*, die Auskunft gibt, ob die Ausführung erfolgreich bzw. nicht erfolgreich war, ist hier nicht aufgeführt, da sie für alle Anweisungen gleich ist.

ANSI-Standard

Dieser Abschnitt beschreibt Abweichungen vom ANSI-Standard.

Beispiele

Dieser Abschnitt zeigt ein oder mehrere Beispiele für die Verwendung der Anweisung. Die Beispiele beziehen sich auf die im Anhang A.1 angegebene Beispieldatenbank.

> > > > Verwandte Anweisungen

6.5.2 Parameter

In den Syntaxdefinitionen werden für dieselben Parametertypen dieselben Bezeichnungen verwendet. Folgende Übersicht zeigt die wichtigen Bezeichnungen aus diesem Kapitel in alphabetischer Reihenfolge, und wo Sie nähere Informationen zu dem jeweiligen Parametertyp finden.

Parameterbezeichnung	Bedeutung	Nähere Information
anweisungsbezeichner	Name einer dynamisch formulierten Anweisung	3.2.1 und 2.13.4
ausdruck	Ausdruck	5.3
bedingung	Bedingung	5.5 und 5.6
benutzer	SINIX Name eines Benutzers	SINIX-Handbuch [11]
constrname	Name eines Constraint eventuell qualifiziert	3.2.1, 3.2.5 und 2.7
datei	Name einer SINIX-Datei	SINIX-Handbuch [11]
datenbank	Name einer Datenbank eventuell qualifiziert	3.2.1, 3.2.5 und 2.1
datentyp	INFORMIX-Datentyp	4.2
dbspace	Name eines Dbspace	ONLINE-Handbuch [10]
index	Name eines Index eventuell qualifiziert	3.2.1, 3.2.5 und 2.6
konst	Konstante	4.3
muster	Suchmuster für Mustervergleich	5.4
praedikat	Prädikat	5.4
referenz	Umbenennung einer Tabelle für SELECT	3.2.1 und SELECT
spalte	Name einer Spalte eventuell qualifiziert	3.2.1, 3.2.5 und 2.4
spaltenname	Einfacher Spaltenname	3.2.1
select-anweisung	SELECT-Anweisung	6.6 SELECT

Parameterbezeichnung	Bedeutung	Nähere Information
synonym	Synonym eventuell qualifiziert	3.2.1, 3.2.5 und 2.3
tabelle	Name einer Tabelle (Basistabelle, View, temporäre Tabelle) eventuell qualifiziert	3.2.1, 3.2.5 und 2.2
unterabfrage	Unterabfrage	5.2
variable	Hostvariable eventuell kombiniert mit Indikatorvariable	2.13.1 und 2.13.2
view	Name eines View eventuell qualifiziert	3.2.1 und 3.2.5 und 2.2
zeichen	Abdruckbares Einzelzeichen	Keine

Gibt es für einen Parameter Einschränkungen, ist dies bei der Beschreibung des Parameters angegeben; zum Beispiel, wenn für *tabelle* nur eine Basistabelle angegeben werden darf.

6.6 Alphabetischer Nachschlageteil

In diesem Abschnitt sind die SQL-Anweisungen alphabetisch beschrieben.

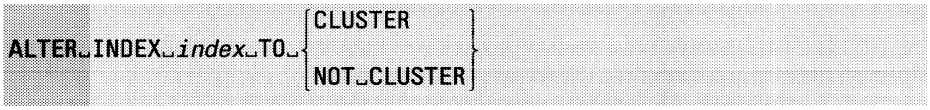
ALTER INDEX - Index ändern

ALTER INDEX ändert die Zuordnung des Cluster-Attributs zu einem Index. Wird das Cluster-Attribut einem Index zugeordnet, so wird die Basistabelle physisch nach der Reihenfolge des Index sortiert.

Vor dem Aufruf beachten

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich entweder das Index-Zugriffsrecht haben oder aber Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

ALTER INDEX TO CLUSTER sperrt die Tabelle exklusiv gegen jeglichen Zugriff anderer Prozesse. ALTER INDEX TO CLUSTER wird abgewiesen, wenn ein anderer Prozeß die Tabelle bearbeitet. Dies gilt auch dann, wenn nur lesend zugegriffen wird.



index
Name des Index, dessen Cluster-Attribut geändert werden soll.

CLUSTER
Die Sätze der Basistabelle werden nach dem angegebenen Index sortiert. Dem Index wird das Cluster-Attribut zugeordnet. Ist das Cluster-Attribut einem anderen Index derselben Tabelle zugeordnet, so müssen Sie erst diese Zuordnung aufheben (ALTER INDEX TO NOT CLUSTER).

ALTER INDEX TO CLUSTER ist in erster Linie sinnvoll, wenn Sie eine Tabelle regelmäßig in Reihenfolge eines Index verarbeiten. Aufgrund der physischen Anordnung der Sätze in der passenden Reihenfolge ergeben sich optimale Verhältnisse bezüglich der Plattenzugriffe und damit eine entsprechend verbesserte Performance.

NOT CLUSTER
Die Zuordnung des Cluster-Attributs zum angegebenen Index wird aufgehoben. Die zugehörige Tabelle wird weder verändert noch gesperrt. Es erfolgt lediglich eine Änderung der Systemtabellen, so daß dem Index das Cluster-Attribut nicht mehr zugeordnet ist.

Arbeitsweise von ALTER INDEX

ALTER INDEX TO CLUSTER sortiert die Sätze der Tabelle, indem vorübergehend eine zweite Tabelle aufgebaut wird und die Sätze aus der Original-Tabelle sortiert in die zweite Tabelle geschrieben werden. Dies kann einige Zeit dauern und erfordert vorübergehend Plattenspeicherplatz für die Kopie der Tabelle. Nach dem Sortieren wird die Original-Tabelle automatisch gelöscht und der Speicherplatz freigegeben.

Durch die Anweisung ALTER INDEX TO CLUSTER wird die Tabelle einmalig in der Reihenfolge des Index sortiert. Wenn Sie danach die Tabelle mit INSERT-, LOAD-, UPDATE- oder DELETE-Anweisungen bearbeiten, so geht dadurch im Laufe der Zeit die Sortierung verloren. Diese können Sie dann durch eine weitere Anweisung ALTER INDEX TO CLUSTER wiederherstellen. Es ist nicht nötig ALTER INDEX TO NOT CLUSTER anzugeben, wenn danach ALTER INDEX TO CLUSTER auf den gleichen Index angegeben wird.

ALTER INDEX in Transaktionen

Die Anweisung ALTER INDEX kann bei einer INFORMIX-SE-Datenbank nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

INFORMIX-ONLINE kann die Anweisung ALTER INDEX zurücksetzen.

ANSI-Standard

Die Anweisung ALTER INDEX ist nicht im ANSI-Standard enthalten.

Beispiel

Im folgenden Beispiel werden für die Tabelle *auftrag* zwei Indizes erzeugt, wobei die Sätze in der Tabelle aufsteigend nach Kundennummer sortiert werden (CREATE CLUSTER INDEX ix_kun), wie die Ausgabe zeigt. Anschließend sollen die Sätze mit ALTER INDEX nach Auftragsnummer sortiert werden. Zuvor muß jedoch die Zuordnung des Cluster-Attributs zum Index *ix_kun* aufgehoben werden. Die SELECT-Anweisung zeigt, daß die Sätze jetzt nach Auftragsnummer sortiert abgespeichert sind.

```
CREATE UNIQUE INDEX ix_auf
       ON auftrag (auftrags_nr);
CREATE CLUSTER INDEX ix_kun
       ON auftrag (kunden_nr);

SELECT auftrags_nr,kunden_nr
          FROM auftrag
```

auftrags_nr	kunden_nr
1002	101
1001	104
1003	104
1011	104
1013	104
1004	106
1014	106
1008	110
1015	110
1009	111
1006	112
1010	115
1005	116
1007	117
1012	117

```
ALTER INDEX ix_kun TO NOT CLUSTER;
ALTER INDEX ix_auf TO CLUSTER;

SELECT auftrags_nr, kunden_nr
          FROM auftrag
```

auftrags_nr	kunden_nr
1001	104
1002	101
1003	104
1004	106
1005	116
1006	112
1007	117
1008	110
1009	111
1010	115
1011	104
1012	117
1013	104
1014	106
1015	110

> > > > CREATE INDEX, DROP INDEX

ALTER TABLE - Tabellenstruktur ändern

ALTER TABLE ändert die Struktur einer Basistabelle. Folgende Änderungen sind möglich:

- Spalten einfügen (ADD-Klausel)
- Spalten löschen (DROP-Klausel)
- Datentyp von Spalten ändern (MODIFY-Klausel)
- NULL-Werte für Spalten erlauben oder verbieten (MODIFY-Klausel, ADD-Klausel)
- UNIQUE CONSTRAINT hinzufügen (ADD CONSTRAINT-Klausel, ADD-Klausel)
- UNIQUE CONSTRAINT löschen (DROP CONSTRAINT-Klausel)

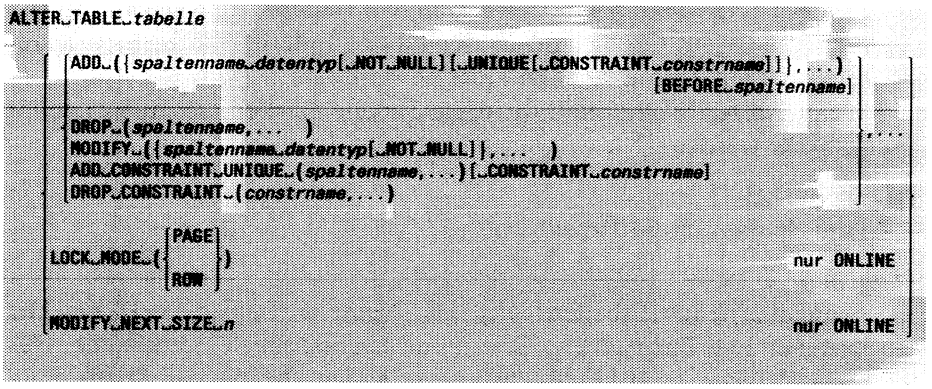
Bei INFORMIX-ONLINE sind zusätzlich folgende Änderungen möglich:

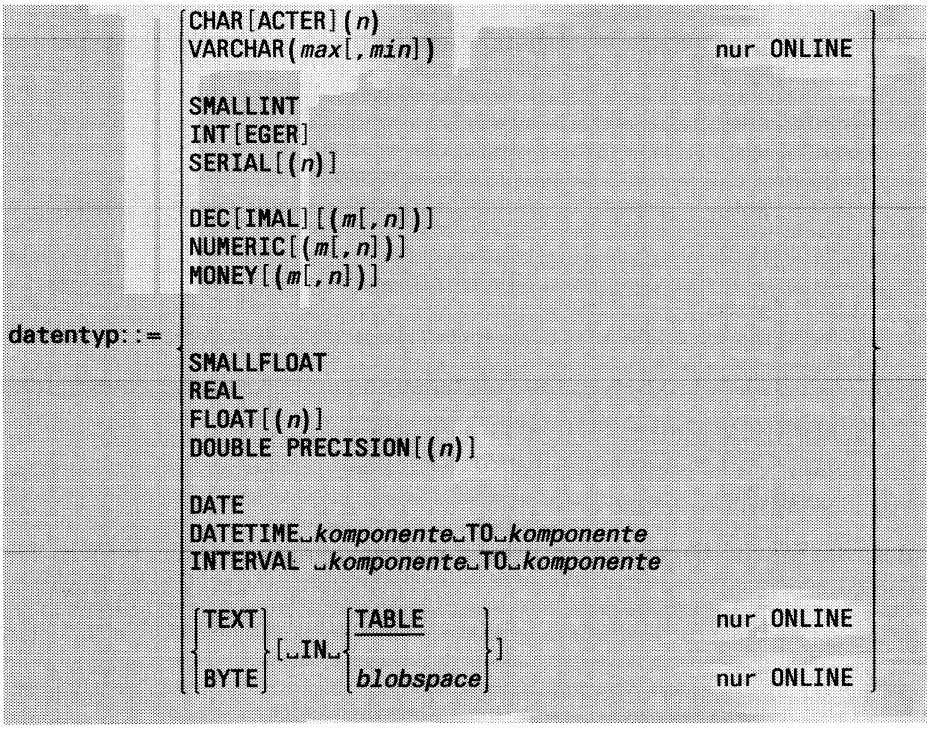
- Sperrbereich ändern (LOCK MODE-Klausel)
- Extent-Größe ändern (MODIFY NEXT SIZE-Klausel)

Vor dem Aufruf beachten

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich entweder das Alter-Zugriffsrecht haben oder aber Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

ALTER TABLE sperrt die Tabelle exklusiv gegen jeglichen Zugriff anderer Prozesse. ALTER TABLE wird abgewiesen, wenn ein anderer Prozeß die Tabelle bearbeitet. Dies gilt auch dann, wenn nur lesend zugegriffen wird.





tabelle

Name oder Synonym der Basistabelle, deren Struktur geändert werden soll. Bei INFORMIX-ONLINE kann ein Benutzer mit DBA-Zugriffsrecht die Extent-Größe einer Systemtabelle ändern. Alle anderen Klauseln sind auf Systemtabellen nicht anwendbar.
Wenn ein Audit-Protokoll für diese Tabelle existiert, so muß es zuvor gelöscht werden.

ADD-Klausel

Fügt Spalten in die Tabelle ein.

spaltenname

Name der neuen Tabellenspalte. Der Name darf aus max. 18 Zeichen bestehen und Buchstaben, Ziffern und Unterstriche (_) enthalten. Das erste Zeichen muß ein Buchstabe sein. Der Name der Spalte muß innerhalb der Tabelle eindeutig sein.

Bereits existierende Sätze der Tabelle erhalten in dieser Spalte den NULL-Wert.

datentyp
Datentyp für die neue Spalte. Der Datentyp SERIAL darf nur angegeben werden, wenn die Tabelle leer ist (siehe auch *SERIAL-Spalten nachträglich einfügen*). Eine Beschreibung der aufgeführten Datentypen finden Sie in Kapitel 4. Dort ist auch beschrieben, welche Werte in Spalten welchen Datentyps eingegeben werden können.

NOT NULL
Die Spalte läßt keine NULL-Werte als Eingabe zu. NOT NULL kann für eine neue Spalte nur angegeben werden, wenn die Tabelle leer ist.
NOT NULL nicht angegeben:
NULL-Werte sind als Eingabe für die Spalte zugelassen.

UNIQUE-Klausel
Erzwingt die Eindeutigkeit der Werte in der direkt zuvor genannten Spalte. Die Spalte darf nicht vom Datentyp TEXT oder BYTE sein. Zur Realisierung des Constraint wird ein eindeutiger Index über die Spalte erzeugt.

Achtung:
Ein Constraint ist für neue Spalten nur möglich, wenn die Tabelle max. 1 Satz enthält, da neue Spalten in bereits existierenden Sätzen der Tabelle den NULL-Wert erhalten. Dadurch ist die Eindeutigkeit der Spaltenwerte nicht mehr vorhanden.

UNIQUE-Klausel nicht angegeben:
Es können doppelte Werte in der Spalte vorkommen.

CONSTRAINT *constrname*
Name für den Constraint. Der Name darf aus max. 18 Zeichen bestehen und Buchstaben, Ziffern und Unterstriche (_) enthalten. Das erste Zeichen muß ein Buchstabe sein. Der Name darf bei einer Nicht-ANSI-Datenbank nicht mit einem bereits existierenden Constraint-Namen der aktuellen Datenbank übereinstimmen. Bei ANSI-Datenbanken muß der Constraint-Name pro Datenbankbenutzer eindeutig sein.

Ein Benutzer mit DBA-Zugriffsrecht kann einen Constraint für einen anderen Benutzer erzeugen, indem er *constrname* mit dem Namen des Eigentümers qualifiziert. Ohne Qualifikation wird die zur Ablaufzeit aktuelle Benutzerkennung Eigentümer des Constraint.

CONSTRAINT *constrname* nicht angegeben:
Wird kein Constraint-Name angegeben, so vergibt INFORMIX einen Namen (siehe unten).

BEFORE *spaltenname*
Name einer bereits existierenden Spalte, vor die die neue Spalte eingefügt wird.

BEFORE *spaltenname* nicht angegeben:
Die neue Spalte wird am Ende der Tabelle angefügt.

DROP-Klausel
Löscht Spalten aus der Tabelle.

spaltenname
Name der Spalte, die gelöscht werden soll. Es werden automatisch auch alle Indizes und UNIQUE CONSTRAINTs gelöscht, die die angegebene Spalte enthalten.

Wenn Sie eine Spalte löschen, die an einem View (siehe CREATE VIEW) beteiligt ist, können Sie nicht mehr mit diesem View arbeiten.

MODIFY-Klausel
Ändert den Datentyp einer Spalte. Für eine CHAR-Spalte kann die Länge geändert werden. NULL-Werte können für Spalten nachträglich erlaubt oder verboten werden.

spaltenname
Name der Spalte, die geändert werden soll. Es werden automatisch auch alle Indizes angepaßt, die die angegebene Spalte enthalten.

datentyp
Neuer Datentyp für die Spalte. Eine Beschreibung der aufgeführten Datentypen finden Sie in Kapitel 4. Soweit möglich konvertiert INFORMIX die Werte der Tabelle in den neuen Datentyp.

Soll nur die Zulässigkeit von NULL-Werten verändert werden, so ist für *datentyp* der bisherige Datentyp anzugeben.

Beim Datentyp SERIAL ist die Angabe eines Startwertes wirkungslos (siehe auch *SERIAL-Spalten nachträglich einfügen*).

NOT NULL
Die Spalte läßt keine NULL-Werte als Eingabe zu. Sie dürfen NOT NULL nur angeben, wenn die Spalte noch keine NULL-Werte enthält.

NOT NULL nicht angegeben:
NULL-Werte sind als Eingabe für die Spalte zugelassen.

ADD CONSTRAINT-Klausel

Ein Constraint wird zur Tabelle hinzugefügt. Ein Constraint erzwingt die Eindeutigkeit der Werte in den angegebenen Spalten.

UNIQUE (*spaltenname,...*)

Erzwingt die Eindeutigkeit der Werte in den angegebenen Spalten. Die Spalte darf nicht vom Datentyp TEXT oder BYTE sein.
Sie können bei einer INFORMIX-SE-Datenbank bis zu 8 Spalten angeben, deren Gesamtlänge 120 Byte nicht überschreiten darf.
Bei einer INFORMIX-ONLINE-Datenbank können Sie bis zu 16 Spalten angeben, deren Gesamtlänge 255 Byte nicht überschreiten darf.

Zur Realisierung des Constraint wird ein zusammengesetzter eindeutiger Index über die Spalten erzeugt.

CONSTRAINT *constrname*

siehe oben

CONSTRAINT *constrname* nicht angegeben:
Wird kein Constraint-Name angegeben, so vergibt INFORMIX einen Namen (siehe unten).

DROP CONSTRAINT-Klausel

Ein Constraint wird gelöscht. Der zugrundeliegende Index wird ebenfalls gelöscht.

constrname

Name des Constraint, der gelöscht werden soll. Wenn Sie bei der Definition des Constraint keinen Namen angegeben haben, müssen Sie hier den von INFORMIX erzeugten Namen angeben (siehe unten).

LOCK MODE-Klausel

Die LOCK MODE-Klausel legt fest, ob Sperren auf den Bereich einer Page oder eines Satzes gesetzt werden.

Einschränkung:
Die LOCK MODE-Klausel kann nur bei einer INFORMIX-ONLINE-Datenbank angegeben werden.

PAGE

Sperrbereich ist Page.

ROW

Sperrbereich ist Satz.

MODIFY NEXT SIZE *n*

Die Größe der Extents, die beim Wachsen der Tabelle hinzugefügt werden, wird verändert. *n* gibt die neue Größe in Kbyte an, wobei *n* eine Zahl zwischen 8 und 32767 ist. Die Größe bereits bestehender Extents kann nicht verändert werden. Hinweise zur Einschätzung der Größe einer Tabelle finden Sie im Anhang A.4 beschrieben.

Einschränkung:
Die Klausel MODIFY NEXT SIZE kann nur bei einer INFORMIX-ONLINE-Datenbank angegeben werden.

Arbeitsweise von ALTER TABLE

Enthält die ALTER TABLE-Anweisung die Klauseln ADD, DROP oder MODIFY, so baut INFORMIX eine neue Tabelle auf und kopiert die Daten aus der alten in die neue Tabelle. Dies kann einige Zeit dauern und erfordert vorübergehend den Plattenspeicherplatz für die Kopie der Tabelle. Die alte Tabelle wird danach automatisch gelöscht und der Speicherplatz freigegeben.

Constraint-Name

Wenn Sie dem Constraint keinen Namen geben, so erzeugt INFORMIX einen Namen nach dem Schema:
u*tabid*_*indexnummer*.

Der zugehörende Index hat den Namen:
␣*tabid*_*indexnummer*.

u

von INFORMIX vergebenes Präfix (für UNIQUE).

tabid

von INFORMIX vergebene Nummer der Tabelle.

indexnummer

von INFORMIX vergebene Nummer des Index.

Zur Realisierung des Constraint wird ein eindeutiger Index über die Spalten erzeugt. Der Index ist aufsteigend sortiert. Sie dürfen deshalb keinen Index erzeugen, der genau diesem, durch Constraint erzeugten Index in allen Merkmalen gleicht (siehe auch CREATE INDEX).

Der von INFORMIX erzeugte Name des Constraint kann von der Systemtabelle *sysconstraints* abgefragt werden.

```
SELECT constrname FROM sysconstraints
       WHERE tabid = (SELECT tabid FROM systables
                             WHERE tabname = "tabelle")
```

SERIAL-Spalten nachträglich einfügen

Eine Spalte mit dem Datentyp SERIAL kann mit ALTER TABLE nicht nachträglich eingefügt werden, wenn die Tabelle bereits Sätze enthält. Diese Restriktion kann folgendermaßen umgangen werden:

1. Fügen Sie eine Spalte vom Datentyp INTEGER ein (ALTER TABLE-Anweisung, ADD-Klausel).
2. Belegen Sie diese Spalte mit den gewünschten Werten (UPDATE-Anweisung).
3. Wandeln Sie den Datentyp der Spalte in SERIAL um (ALTER TABLE-Anweisung, MODIFY-Klausel).

Reihenfolge der Klauseln

Innerhalb einer ALTER TABLE-Anweisung können jeweils mehrere Klauseln in beliebiger Reihenfolge angegeben werden. Es ist lediglich zu beachten, daß die einzelnen Klauseln genau in der angegebenen Reihenfolge abgearbeitet werden. Wenn eine der Klauseln auf einen Fehler läuft, wird die ganze Anweisung ALTER TABLE abgewiesen.

Beispiel:

```
richtig:  ALTER TABLE tab001 ADD (spalte02 DATE BEFORE spalte01),
                             DROP (spalte01 )

falsch:   ALTER TABLE tab001 DROP (spalte01 ),
                             ADD (spalte02 DATE BEFORE spalte01)
```

Das zweite Beispiel führt zu einem Fehler, da zu dem Zeitpunkt, in dem ADD ausgeführt werden soll, die Spalte *spalte01* bereits gelöscht ist (DROP).

```
falsch:   ALTER TABLE tab001 ADD (spalte03 FLOAT BEFORE spalte02 ,
                                  spalte02 DATE  BEFORE spalte01)
```

Das dritte Beispiel ist ebenfalls fehlerhaft, da zu dem Zeitpunkt, in dem die Spalte *spalte03* eingefügt werden soll, die Spalte *spalte02* noch nicht existiert.

Datenkonvertierung bei eindeutigem Index bzw. Constraint

Wird der Datentyp einer Spalte mit der MODIFY-Klausel verändert, so konvertiert INFORMIX soweit möglich die bestehenden Werte dieser Spalte in den neuen Datentyp.

Wird z.B. ein numerischer Datentyp mit Nachkommastellen in einen numerischen Datentyp ohne Nachkommastellen umgewandelt, so werden bei der Konvertierung nur die Vorkommastellen übernommen. Die Nachkommastellen der Spaltenwerte werden abgeschnitten.
Das kann zu Problemen führen, wenn für diese Spalte ein Constraint oder ein eindeutiger Index definiert ist.

Beispiel:

Eine Spalte *zahl* mit dem Datentyp MONEY enthält die Werte 33.11, 33.77 und 44.47.

```
ALTER TABLE ... MODIFY (zahl INTEGER)
```

Durch die Konvertierung von MONEY nach INTEGER ergeben sich die Werte 33, 33 und 44. Der Wert 33 tritt zweimal auf. INFORMIX ist daher nicht in der Lage, über der INTEGER-Spalte einen eindeutigen Index aufzubauen und bricht ALTER TABLE mit einer Fehlermeldung ab.

ALTER TABLE in Transaktionen

Die Anweisung ALTER TABLE kann bei einer INFORMIX-SE-Datenbank nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

INFORMIX-ONLINE kann die Anweisung ALTER TABLE zurücksetzen.

ANSI-Standard

Die Anweisung ALTER TABLE ist nicht im ANSI-Standard enthalten.

Beispiel 1

Das erste Beispiel verdeutlicht die Syntax der ALTER TABLE-Anweisung.

```
ALTER TABLE tab1 ADD (datum DATE);

ALTER TABLE tab1 ADD (nummer INTEGER);

ALTER TABLE tab1 ADD CONSTRAINT UNIQUE (datum, nummer)
                 CONSTRAINT datnum;

ALTER TABLE tab1 MODIFY (datum DATE NOT NULL);

ALTER TABLE tab1 ADD (autonummer CHAR(10) UNIQUE);
```

Beispiel 2

Das folgende Beispiel zeigt zwei verschiedene ALTER TABLE-Anweisungen, die dasselbe bewirken: zwei neue Spalten werden eingefügt.

```
ALTER TABLE tab001 ADD (spalte02 DATE  BEFORE spalte01 ),
                   ADD (spalte03 FLOAT BEFORE spalte02 );

ALTER TABLE tab001 ADD (spalte02 DATE  BEFORE spalte01 ,
                        spalte03 FLOAT BEFORE spalte02 )
```

Beispiel 3

Im folgenden Beispiel werden für die Tabelle *posten* folgende Änderungen vorgenommen:

- vor die Spalte *gesamtpreis* wird eine neue Spalte *posten_gewicht* eingefügt
- die Spalte *gesamtpreis* wird gelöscht
- die Länge der Spalte *herstellercode* wird geändert in CHAR(4).

```
INFO COLUMNS FOR posten
```

```
Spaltenname          Typ                                  NULL's
posten_nr            smallint                             yes
auftrags_nr          integer                              yes
artikel_nr           smallint                             yes
herstellercode       char(3)                              yes
menge                smallint                             yes
gesamtpreis          money(8,2)                           yes
```

```
ALTER TABLE posten
      ADD (posten_gewicht DECIMAL(6,2) BEFORE gesamtpreis),
      DROP (gesamtpreis),
      MODIFY (herstellercode CHAR(4));

INFO COLUMNS FOR posten
```

```
Spaltenname          Typ                                  NULL's
posten_nr            smallint                             yes
auftrags_nr          integer                              yes
artikel_nr           smallint                             yes
herstellercode       char(4)                              yes
menge                smallint                             yes
posten_gewicht       decimal(6,2)                         yes
```

Beispiel 4

Im folgenden Beispiel für INFORMIX-ONLINE wird der Sperrbereich der Tabelle geändert und die Größe des NEXT EXTENT SIZE auf 30 Kbyte festgesetzt. Die neue Größe des NEXT EXTENT SIZE gilt für alle folgenden Vergrößerungen der Tabelle. Zur Kontrolle werden die entsprechenden Einträge in der Systemtabelle *systables* ausgegeben.

```
ALTER TABLE posten LOCK MODE (ROW), MODIFY NEXT SIZE 30;

SELECT locklevel, nextsize FROM systables WHERE tabname = "posten"
```

```
locklevel    nextsize

R                  30
```

>>>> CREATE INDEX, CREATE TABLE, RENAME COLUMN, RENAME TABLE

BEGIN WORK - Transaktion starten

BEGIN WORK startet eine Transaktion und kennzeichnet ihren Anfang im Transaktionsprotokoll. BEGIN WORK wird ignoriert, wenn bereits eine Transaktion eröffnet ist.

Vor dem Aufruf beachten

BEGIN WORK ist nur erlaubt, wenn die Transaktionssicherung für die Datenbank eingeschaltet ist. Die Transaktionssicherung wird entweder mit CREATE DATABASE beim Erzeugen der Datenbank bzw. danach mit START DATABASE oder vom INFORMIX-ONLINE-Verwalter eingeschaltet.

BEGIN WORK und ANSI-Datenbanken

BEGIN WORK sollten Sie nicht bei einer ANSI-Datenbank verwenden, da dort Transaktionen implizit nach COMMIT WORK bzw. ROLLBACK WORK gestartet werden.
BEGIN WORK führt bei einer ANSI-Datenbank zu einem Laufzeit-Fehler, außer wenn BEGIN WORK unmittelbar nach folgenden Anweisungen verwendet wird:

- COMMIT WORK
- CREATE DATABASE
- DATABASE
- START DATABASE
- ROLLBACK WORK

In diesem Fall tritt kein Fehler, sondern nur eine Warnung auf.

Auswirkung von BEGIN WORK auf Sperren

Sätze, die Sie in einer Transaktion mit DELETE, INSERT, LOAD oder UPDATE bearbeiten, erhalten eine Sperre. Die Sperre wird erst am Ende der Transaktion freigegeben.

Die Anzahl der möglichen Sperren ist bei INFORMIX-SE durch das Betriebssystem bzw. bei INFORMIX-ONLINE durch Shared Memory Parameter beschränkt. Wenn eine Transaktion zu viele Sperren verursachen würde, empfiehlt es sich, unmittelbar nach Beginn der Transaktion die ganze betroffene Tabelle mit LOCK TABLE zu sperren. Nach LOCK TABLE werden keine Einzelsperren mehr auf die gesperrte Tabelle gesetzt (siehe LOCK TABLE).

Weitere Informationen über Transaktionen und Sperren finden Sie in Kapitel 2.

ANSI-Standard

Die Anweisung BEGIN WORK ist nicht im ANSI-Standard enthalten.

Beispiel

siehe COMMIT WORK und ROLLBACK WORK.

>>>> COMMIT WORK, LOCK TABLE, ROLLBACK WORK, START DATABASE

CHECK TABLE - Tabelle prüfen

CHECK TABLE stellt Widersprüche zwischen den Datensätzen einer Tabelle und den zugehörigen Indizes fest. Ein fehlerhafter Index kann dann mit REPAIR TABLE richtiggestellt werden.

Vor dem Aufruf beachten

CHECK TABLE kann nur für eine INFORMIX-SE-Datenbank verwendet werden.

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

CHECK TABLE sperrt die Tabelle exklusiv gegen jeglichen Zugriff anderer Prozesse. CHECK TABLE wird abgewiesen, wenn ein anderer Prozeß die Tabelle bearbeitet. Dies gilt auch dann, wenn nur lesend zugegriffen wird.

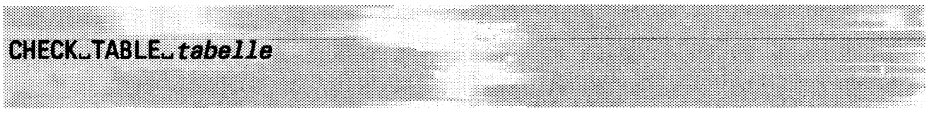


tabelle
Name der Basistabelle, die geprüft werden soll. Die Angabe eines Synonyms und der Systemtabelle *systables* ist hier nicht erlaubt.

Arbeitsweise von CHECK TABLE

CHECK TABLE ruft intern das Dienstprogramm BCHECK auf (siehe Handbuch für das jeweilige Frontend-Produkt [1, 3, 4, 6]). CHECK TABLE *tabelle* ist gleichbedeutend mit folgendem Aufruf auf Betriebssystemebene:

```
bcheck -n tabnnn.
```

ANSI-Standard

Die Anweisung CHECK TABLE ist nicht im ANSI-Standard enthalten.

>>>> REPAIR TABLE

CLOSE - Satzzeiger schließen

CLOSE schließt einen Satzzeiger, den Sie mit der Anweisung DECLARE für eine INSERT- oder SELECT-Anweisung vereinbart haben.

Haben Sie den Satzzeiger für eine SELECT-Anweisung vereinbart, ist er nach CLOSE nicht länger mit der Ergebnistabelle verbunden. Die Anweisung FETCH können Sie erst wieder verwenden, wenn der Satzzeiger mit OPEN neu geöffnet wird.

War der Satzzeiger für eine INSERT-Anweisung vereinbart, wird der Insert-Puffer als Block in die Datenbank eingefügt. Nach CLOSE existiert der Puffer nicht mehr. Ein leerer Puffer für den Satzzeiger kann erneut mit OPEN bereitgestellt werden.

Vor dem Aufruf beachten

Satzzeiger sind nur in der Quellprogrammdatei ansprechbar, in der sie mit DECLARE vereinbart werden. Der mit CLOSE angesprochene Satzzeiger muß zuvor mit DECLARE für eine SELECT- oder INSERT-Anweisung vereinbart und mit OPEN geöffnet werden.

```
CLOSE satzzeiger
```

satzzeiger
Name des Satzzeigers, den Sie schließen wollen.

Satzzeiger und Transaktionen

Die Anweisungen COMMIT WORK und ROLLBACK WORK schließen alle offenen Satzzeiger, außer solchen, die mit WITH HOLD definiert wurden. Sie sollten aber einen Insert-Satzzeiger immer mit der Anweisung CLOSE schließen, um überprüfen zu können, ob die Sätze aus dem Puffer erfolgreich in die Datenbank geschrieben wurden (siehe *Erfolgskontrolle*).

Schließen eines Insert-Satzzeigers

Werden einem Insert-Satzzeiger mit PUT nur konstante Sätze übergeben, so zählt INFORMIX nur die Anzahl der einzufügenden "gleichen" Sätze und fügt diese erst bei einer folgenden FLUSH- bzw. CLOSE-Anweisung in die Tabelle ein.

Wird in diesem Fall das Programm beendet, ohne daß zuvor der Insert-Satzzeiger mit CLOSE geschlossen oder der Puffer mit FLUSH in die Datenbank geschrieben wurde, geht der Inhalt des Puffers verloren.
Es gibt keine Variable, in der INFORMIX alle aus dem Insert-Puffer eingefügten Sätze zählt. Daher sollten Sie, wenn Sie Information über die Anzahl der tatsächlich eingefügten Sätze haben wollen, selbst eine entsprechende Zählvariable vereinbaren und für jede PUT-Anweisung erhöhen.

Erfolgskontrolle bei Programmeinbettung

Wird mit CLOSE ein Insert-Satzzeiger geschlossen, so geben die folgenden Variablen Auskunft über die Anzahl der mit der CLOSE-Anweisung erfolgreich eingefügten Sätze. Im Fehlerfall bleibt der Satzzeiger geöffnet.

	Positivfall	Fehlerfall
INFORMIX-4GL	status = 0 SQLCA.SQLERRD[3] ist die Anzahl der mit CLOSE eingefügten Sätze.	status < 0 SQLCA.SQLERRD[3] ist die Anzahl der mit CLOSE erfolgreich eingefügten Sätze. Der Rest der Sätze ist verloren.
INFORMIX-ESQL/C	sqlca.sqlcode = 0 sqlca.sqlerrd[2] ist die Anzahl der mit CLOSE eingefügten Sätze.	sqlca.sqlcode < 0 sqlca.sqlerrd[2] ist die Anzahl der mit CLOSE erfolgreich eingefügten Sätze. Der Rest der Sätze ist verloren.
INFORMIX-ESQL/COBOL	SQLCODE OF SQLCA = 0 SQLERRD[3] ist die Anzahl der mit CLOSE eingefügten Sätze.	SQLCODE OF SQLCA < 0 SQLERRD[3] ist die Anzahl der mit CLOSE erfolgreich eingefügten Sätze. Der Rest der Sätze ist verloren.

ANSI-Standard

Die Anweisung CLOSE ist im ANSI-Standard enthalten.

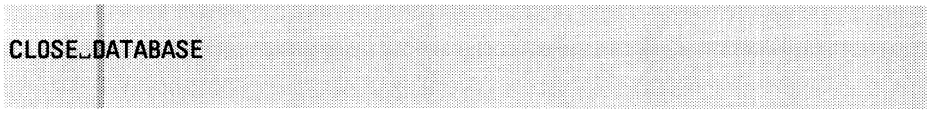
>>>> DECLARE, FETCH, FLUSH, OPEN, PUT

CLOSE DATABASE - Datenbank schließen

CLOSE DATABASE schließt die aktuelle Datenbank. Dabei werden auch alle offenen Satzzeiger geschlossen. Ist keine Datenbank eröffnet, so wird CLOSE DATABASE ignoriert.

Vor dem Aufruf beachten

CLOSE DATABASE wird abgewiesen, wenn noch eine Transaktion offen ist. Beenden Sie zuerst die Transaktion.



Verwendung von CLOSE DATABASE

CLOSE DATABASE müssen Sie vor folgenden Anweisungen eingeben, wenn diese die aktuelle Datenbank betreffen:

DROP DATABASE	Datenbank löschen
ROLLFORWARD DATABASE	Datenbanksicherung aktualisieren
START DATABASE	Transaktionssicherung einstellen

Nach CLOSE DATABASE können Sie nur folgende Anweisungen eingeben:

CREATE DATABASE	Datenbank erzeugen
DATABASE	Datenbank eröffnen
DROP DATABASE	Datenbank löschen
ROLLFORWARD DATABASE	Datenbanksicherung aktualisieren
START DATABASE	Transaktionssicherung einstellen

CLOSE DATABASE als dynamische Anweisung

Wird CLOSE DATABASE mit PREPARE für die dynamische Ausführung mit EXECUTE vorbereitet, so muß CLOSE DATABASE die einzige Anweisung in dieser PREPARE-Anweisung sein.

ANSI-Standard

Die Anweisung CLOSE DATABASE ist nicht im ANSI-Standard enthalten.

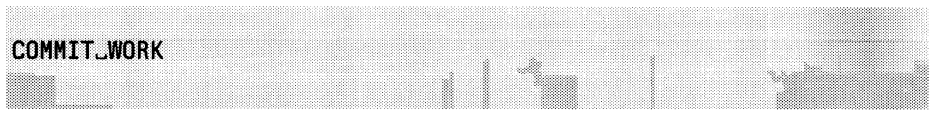
>>>> CREATE DATABASE, DATABASE, DROP DATABASE, ROLLFORWARD DATABASE, START DATABASE

COMMIT WORK - Transaktion beenden

COMMIT WORK beendet eine Transaktion und schreibt die während der Transaktion durchgeführten Änderungen auf der Datenbank fest. Die Transaktion wird im Transaktionsprotokoll als abgeschlossen gekennzeichnet. COMMIT WORK wird ignoriert, wenn keine Transaktion eröffnet ist.
Bei einer ANSI-Datenbank schließt COMMIT WORK eine Transaktion ab und eröffnet implizit die nächste Transaktion.

Vor dem Aufruf beachten

COMMIT WORK ist nur erlaubt, wenn die Transaktionssicherung für die Datenbank eingeschaltet ist. Die Transaktionssicherung wird entweder mit CREATE DATABASE beim Erzeugen der Datenbank bzw. danach mit START DATABASE oder vom INFORMIX-ONLINE-Verwalter eingeschaltet.



Auswirkung von COMMIT WORK auf Satzzeiger

COMMIT WORK schließt alle offenen Satzzeiger, außer diejenigen, die mit WITH HOLD definiert wurden.

Ein Satzzeiger ohne WITH HOLD, der für eine SELECT-Anweisung vereinbart wurde, ist nicht mehr mit der Ergebnistabelle verbunden. Nach COMMIT WORK ist nur eine folgende OPEN-Anweisung für diesen Satzzeiger sinnvoll und möglich.

Wurde ein Satzzeiger für eine INSERT-Anweisung vereinbart, so wird bei COMMIT WORK der Inhalt des Puffers in die Datenbank geschrieben. Sie haben dabei keine Kontrolle darüber, ob wirklich alle Sätze erfolgreich in die Datenbank eingefügt wurden. Leeren Sie daher den Puffer explizit mit FLUSH oder mit CLOSE.

Wird COMMIT WORK in einem 4GL-Programm innerhalb einer FOREACH-Schleife verwendet, so muß der Satzzeiger mit WITH HOLD definiert sein. Ansonsten wird der Satzzeiger geschlossen und beim nächsten Schleifendurchlauf tritt ein Fehler auf.

Weitere Informationen über Transaktionen und Sperren finden Sie in Kapitel 2.

Auswirkung von COMMIT WORK auf Sperren

COMMIT WORK hebt alle Einzelsperren auf, die während der Transaktion gesetzt wurden. Ebenfalls werden Tabellensperren aufgehoben, die Sie mit LOCK TABLE gesetzt haben.

ANSI-Standard

COMMIT WORK ist im ANSI-Standard enthalten.

Beispiel

Im folgenden Beispiel wird für die Datenbank *versand* Transaktionssicherung eingestellt und ein Transaktionsprotokolldatei definiert. Anschließend wird eine Transaktion gestartet, in der mit INSERT ein Satz in die Tabelle *hersteller* aufgenommen wird. Da die Transaktion mit COMMIT WORK beendet wird, bleibt der eingefügte Satz in der Tabelle erhalten. Wird anstelle des COMMIT WORK ein ROLLBACK WORK gegeben, so wird der Satz nicht in die Tabelle eingefügt.

```
BEGIN WORK;
INSERT INTO hersteller (herstellercode,herstellername)
       VALUES ("NEU","Neusport");

COMMIT WORK;
SELECT * FROM hersteller;
```

```
herstellercode herstellername

SMT            Schmitt
ANZ            Anzabel
NRG            Norgesta
HSK            Hasken
HRO            Herold
NEU            Neusport
```

>>>> BEGIN WORK, LOCK TABLE, ROLLBACK WORK, START DATABASE

CREATE AUDIT - Audit-Protokoll erzeugen

CREATE AUDIT erzeugt eine Audit-Protokolldatei für eine Tabelle und startet die Protokollierung aller Änderungen, die Sie mit INSERT, DELETE, LOAD und UPDATE am Inhalt der Tabelle vornehmen.

Existiert zur angegebenen Tabelle bereits ein Audit-Protokoll mit anderem Namen, dann wird CREATE AUDIT abgewiesen. Bei gleichem Namen bleibt die Anweisung wirkungslos.

Vor dem Aufruf beachten

CREATE AUDIT darf nur für eine INFORMIX-SE-Datenbank verwendet werden.

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

CREATE AUDIT sperrt die Tabelle exklusiv gegen jeglichen Zugriff anderer Prozesse. CREATE AUDIT wird abgewiesen, wenn ein anderer Prozeß die Tabelle bearbeitet. Dies gilt auch dann, wenn nur lesend zugegriffen wird.

```
CREATE␣AUDIT␣FOR␣tabelle␣IN␣"datei"
```

tabelle
Name oder Synonym der Basistabelle, deren Änderungen Sie protokollieren wollen. *tabelle* darf keine Systemtabelle sein.

Achtung:
Für diese Tabelle kann danach kein Cluster-Index mehr definiert werden (siehe ALTER INDEX bzw. CREATE INDEX, CLUSTER-Klausel).

datei
Name der Datei, in der die Tabellenänderungen protokolliert werden sollen. *datei* muß mit dem absoluten Pfadnamen angegeben werden. Der Dateiname darf maximal 64 Zeichen lang sein. Alle Dateiverzeichnisse im Pfad müssen bereits existieren.

Zugriffsrechte zur Audit-Datei auf Betriebssystemebene

Benutzer, die auf INFORMIX-Ebene berechtigt sind, die Tabelle zu ändern, können dennoch die Tabelle nicht ändern, wenn sie auf Betriebssystemebene keine Zugriffserlaubnis auf die Audit-Datei haben.
Alle Benutzer, die auf die Tabelle zugreifen wollen, müssen Ausführungsberechtigung (x-Bit) für alle Dateiverzeichnisse im Pfad besitzen. Alle Benutzer, die die Tabelle verändern wollen, müssen Schreibberechtigung (w-Bit) für die Audit-Datei besitzen.

Audit-Protokollierung und Datensicherung

Für eine INFORMIX-SE-Datenbank ohne Transaktionssicherung bieten Audit-Protokolle die einzige Möglichkeit, die Sicherungskopie einer Tabelle zu aktualisieren.

Die Tabelle muß nach CREATE AUDIT und vor der ersten Änderung der Tabelle mit Betriebssystemmitteln gesichert werden. Andernfalls kann RECOVER TABLE die Sicherungskopie nicht verarbeiten oder erzeugt möglicherweise eine logisch inkonsistente Sicherungstabelle. Es liegt in der vollen Verantwortung des Benutzers, dafür zu sorgen, daß Audit-Protokoll und Sicherungskopie zusammenpassen!

Bei einer Datenbank mit Transaktionssicherung werden zum Aktualisieren einer Sicherungskopie der Datenbank die Transaktionsprotokolle verwendet.
Sie können zusätzlich Audit-Protokollierung verwenden, um die Änderungen in der Tabelle zu protokollieren. Zur Kontrolle der Änderungen können Sie dann das Audit-Protokoll mit einem selbst erstellten Programm auswerten (siehe *Aufbau der Audit-Datei*).

Verwenden Sie entweder Audit- oder Transaktionsprotokolle zum Aktualisieren von Sicherungskopien. Eine Mischung dieser beiden Sicherungsverfahren kann zur Inkonsistenz der Daten führen.

INFORMIX-ONLINE verwendet keine Audit-Protokollierung.

CREATE AUDIT in Transaktionen

Die Anweisung CREATE AUDIT kann nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

Aufbau der Audit-Datei

Das Satzformat des Audit-Protokolls gliedert sich logisch in drei Teile:

1. Vorspann von 14 Byte

   Aufbau des Vorspanns:

Byte	entspricht INFORMIX-Datentyp	Inhalt
0	CHAR(1)	Folgende Werte sind möglich: a bedeutet: eingefügter Satz d bedeutet: gelöschter Satz r bedeutet: geänderter Satz vor Änderung w bedeutet: geänderter Satz nach Änderung
1	CHAR(1)	Identisch zu Byte 0
2 - 5	INTEGER	Binäre Ganzzahl; enthält die Differenz in Sekunden zwischen dem Zeitpunkt der Änderung und einem Zeitpunkt, der vom Betriebssystem abhängig ist; in der Regel 1.1.1970 00:00:00.
6 - 7	SMALLINT	Binäre Ganzzahl; enthält die Prozeßnummer (PID) des INFORMIX-Backends (sqlexec), das die Änderung durchgeführt hat.
8 - 9	SMALLINT	Binäre Ganzzahl; enthält die Benutzerkennung (UID) des Benutzers, der die Änderung durchgeführt hat.
10 - 13	INTEGER	Binäre Ganzzahl; enthält die Satzadresse des betroffenen Satzes (ROWID). Die ROWID ist innerhalb der Tabelle eindeutig.

   Sie können die Audit-Datei per Programm auswerten. Verwenden Sie dabei für die Werte in der Audit-Datei Variablen, die den angegebenen INFORMIX-Datentypen entsprechen, wie z.B *long int* für INTEGER in der Sprache C.

2. Der betroffene Satz der Tabelle, in Format und Länge identisch mit dem entsprechenden Satz in der Datenbank.

3. Ein Nachspann von 1 Byte. Dieses Byte enthält das Steuerzeichen LINEFEED (Zeilenvorschub, hexadezimal X'0A').

Beispiel für den Aufbau einer Audit-Datei

Im folgenden Beispiel werden in die Tabelle *hersteller* zuerst drei Sätze eingefügt, dann ein Satz geändert und ein weiterer gelöscht.

```
INSERT INTO hersteller VALUES("NEU","Neusport GmbH");
INSERT INTO hersteller VALUES("STA","Star Sport");
INSERT INTO hersteller VALUES ("MAX","Maxis Sportkist");
UPDATE hersteller SET herstellercode="STS" WHERE herstellercode="STA";
DELETE FROM hersteller WHERE herstellercode="NEU";
```

Danach enthält die Audit-Datei folgende Daten. Die Ausgabe in diesem Beispiel wurde mit dem SINIX-Kommando xd aufbereitet:

```
  0  616124d9 b853017d  00090000 00014e45  aa$  S }      NE
 10  554e6575 73706f72  7420476d 62482020  UNeusport GmbH
 20  0a616124 d9b85501  7d000900 00000253   aa$  U }      S
 30  54415374 61722053  706f7274 20202020  TAStar Sport
 40  200a6161 24d9b855  017d0009 00000003    aa$  U }
 50  4d41584d 61786973  2053706f 72746b69  MAXMaxis Sportki
 60  73740a72 7224d9b8  56017d00 09000000  st rr$  V }
 70  02535441 53746172  2053706f 72742020   STAStar Sport
 80  2020200a 777724d9  b856017d 00090000     ww$  V }
 90  00025354 53537461  72205370 6f727420   STSStar Sport
 a0  20202020 0a646424  d9b87701 7d000900      dd$  w }
 b0  0000014e 45554e65  7573706f 72742047    NEUNeusport G
 c0  6d624820 200a                          mbH
```

ANSI-Standard

Die Anweisung CREATE AUDIT ist nicht im ANSI-Standard enthalten.

Beispiel

Das folgende Beispiel erzeigt eine Audit-Datei für die Tabelle *hersteller* der Datenbank *versand*.

```
CREATE AUDIT FOR hersteller IN "/usr1/save/herst.aud"
```

>>>> DROP AUDIT, RECOVER TABLE

CREATE DATABASE - INFORMIX-SE-Datenbank erzeugen

CREATE DATABASE schließt die aktuelle Datenbank und erzeugt die neue Datenbank. Dabei werden die Systemtabellen für das Data Dictionary erzeugt, das die Struktur der Datenbank beschreibt. Anschließend wird die neue Datenbank eröffnet.

Die Anweisung CREATE DATABASE hat zwei Formate: eines für INFORMIX-SE, das andere für INFORMIX-ONLINE. Beide Formate der Anweisung CREATE DATABASE sind getrennt beschrieben, weil sie sich grundlegend unterscheiden. Im folgenden wird die Anweisung CREATE DATABASE für INFORMIX-SE beschrieben.

Vor dem Aufruf beachten

CREATE DATABASE wird abgewiesen, wenn noch eine Transaktion offen ist. Beenden Sie zuerst die Transaktion.

```
CREATE␣DATABASE␣datenbank[␣WITH␣LOG␣IN␣"datei"[␣MODE␣ANSI]]
```

datenbank
: Name der neuen Datenbank. Der Name darf aus max. 10 Zeichen bestehen und Buchstaben, Ziffern und Unterstriche (_) enthalten. Das erste Zeichen muß ein Buchstabe sein. Der Name einer INFORMIX-SE-Datenbank muß im aktuellen Dateiverzeichnis eindeutig sein. Eigentümer der Datenbank wird die zur Ablaufzeit aktuelle Benutzerkennung.

  Wird die Anweisung CREATE DATABASE in eine Programmiersprache eingebettet, so kann für *datenbank* auch eine alphanumerische Hostvariable benutzt werden.

WITH LOG IN
: Die Transaktionssicherung wird eingeschaltet.

  WITH LOG IN "*datei*" nicht angegeben:
  Die Datenbank wird ohne Transaktionssicherung erzeugt. Ohne Transaktionssicherung können Sie keine Transaktionen und keine Restaurierung der Datenbank bei einem Systemfehler durchführen.

datei

Name der Transaktionsprotokoll-Datei. *datei* muß mit dem absoluten Pfadnamen angegeben werden. Der Dateiname darf maximal 64 Zeichen lang sein. Alle Dateiverzeichnisse im Pfad müssen bereits existieren. Existiert die Transaktionsprotokoll-Datei bereits, so wird CREATE DATABASE abgewiesen.

MODE ANSI

Die Datenbank wird als ANSI-Datenbank erzeugt. Transaktionen sind implizit (siehe auch Kapitel 2). Eine ANSI-Datenbank kann nicht mehr in eine Nicht-ANSI-Datebank umgewandelt werden.

MODE ANSI nicht angegeben:
Die Datenbank wird als Nicht-ANSI-Datenbank erzeugt. Transaktionen müssen explizit mit BEGIN WORK begonnen werden.

CREATE DATABASE in Transaktionen

CREATE DATABASE kann nicht mit ROLLBACK WORK zurückgesetzt werden.

INFORMIX-SE-Datenbank auf Betriebssystemebene

INFORMIX erzeugt im aktuellen Dateiverzeichnis ein Unterverzeichnis mit dem Namen *datenbank.dbs*. Beim Erzeugen der Datenbank mit CREATE DATABASE werden in diesem Dateiverzeichnis die Daten und Indexinformationen der Systemtabellen gespeichert. Der Aufbau der Systemtabellen ist in Anhang A.2 beschrieben.

Im Dateiverzeichnis der Datenbank dürfen Sie keine eigenen Dateien anlegen. Die Dateien im Dateiverzeichnis der Datenbank *datenbank.dbs* dürfen Sie nicht auf Betriebssystemebene verändern oder löschen.

Alle Benutzer, die auf die Datenbank zugreifen wollen, müssen Ausführungsberechtigung (x-Bit) für alle Dateiverzeichnisse im Pfad und Leseberechtigung (r-Bit) für die Datenbank-Dateien besitzen.
Alle Benutzer, die die Datenbank verändern wollen, müssen Schreibberechtigung (w-Bit) für die Datenbank-Dateien besitzen.

Transaktionssicherung bei INFORMIX-SE

Wenn Sie eine Datenbank mit Transaktionssicherung erzeugt haben, können Sie den Dateinamen des Transaktionsprotokolls über die Systemtabelle *systables* erfragen.

```
SELECT dirpath FROM systables WHERE tabname="syslog"
```

Die Transaktionssicherung für eine Nicht-ANSI-Datenbank kann nachträglich mit folgender Anweisung ausgeschaltet werden:

```
DELETE FROM systables WHERE tabname = "syslog"
```

Damit ist die Transaktionssicherung ausgeschaltet. Mit START DATABASE kann sie wieder eingeschaltet werden.

Benutzer, die auf INFORMIX-Ebene berechtigt sind, eine Tabelle zu ändern, können dennoch die Tabelle nicht ändern, wenn sie auf Betriebssystemebene keine Zugriffserlaubnis auf die Transaktionsprotokoll-Datei haben.
Alle Benutzer, die auf eine Tabelle zugreifen wollen, müssen Ausführungsberechtigung (x-Bit) für alle Dateiverzeichnisse im Pfad besitzen. Alle Benutzer, die eine Tabelle verändern wollen, müssen Schreibberechtigung (w-Bit) für die Transaktionsprotokoll-Datei besitzen.

Mit der Menüfunktion *Datenbank* im INFORMIX-Hauptmenü können Sie ebenfalls eine Datenbank erzeugen. Diese Datenbank besitzt jedoch keine Transaktionssicherung.

CREATE DATABASE als dynamische Anweisung

Wird CREATE DATABASE mit PREPARE für die dynamische Ausführung mit EXECUTE vorbereitet, so muß CREATE DATABASE die einzige Anweisung in dieser PREPARE-Anweisung sein.

Eigenschaften von ANSI-Datenbanken

Für ANSI-Datenbanken gilt im Unterschied zu Nicht-ANSI-Datenbanken:

- Die Transaktionssicherung ist immer eingeschaltet. ANSI-Datenbanken arbeiten immer mit Transaktionen.
- Fremde Datenbankobjekte müssen mit dem Namen des Eigentümers qualifiziert werden.
- Bei Anweisungen, die den ANSI-Standard verletzen, werden Warnungen ausgegeben.

Eine ANSI-Datenbank kann nicht mehr in eine Nicht-ANSI-Datenbank umgewandelt werden.

ANSI-Standard

Die Anweisung CREATE DATABASE ist nicht im ANSI-Standard enthalten.

Beispiel

Dieses Beispiel erzeugt die Datenbank *neudb* als Datenbank mit Transaktionssicherung. Der Name des Transaktionsprotokolls kann über die Systemtabelle *systables* abgefragt werden.

```
CREATE DATABASE neudb WITH LOG IN "/usr1/lomata/neudb.log1";

SELECT dirpath FROM systables WHERE tabname="syslog"

dirpath

/usr1/lomata/neudb.log1
```

> > > > DROP DATABASE, GRANT, START DATABASE

CREATE DATABASE - INFORMIX-ONLINE-Datenbank erzeugen

CREATE DATABASE schließt die aktuelle Datenbank und erzeugt die neue Datenbank. Dabei werden die Systemtabellen für das Data Dictionary erzeugt, das die Struktur der Datenbank beschreibt. Anschließend wird die neue Datenbank eröffnet.

Die Anweisung CREATE DATABASE hat zwei Formate: eines für INFORMIX-SE-Datenbanken, das andere für INFORMIX-ONLINE-Datenbanken. Beide Formate der Anweisung CREATE DATABASE sind getrennt beschrieben, weil sie sich grundlegend unterscheiden. Im folgenden wird die Anweisung CREATE DATABASE für INFORMIX-ONLINE beschrieben.

Vor dem Aufruf beachten

CREATE DATABASE wird abgewiesen, wenn noch eine Transaktion offen ist. Beenden Sie zuerst die Transaktion.

datenbank
Name der neuen Datenbank. Der Name darf aus max. 18 Zeichen bestehen und Buchstaben, Ziffern und Unterstriche (_) enthalten. Das erste Zeichen muß ein Buchstabe sein. Der Name der Datenbank muß im INFORMIX-ONLINE-System eindeutig sein. Eigentümer der Datenbank wird die zur Ablaufzeit aktuelle Benutzerkennung.

Wird die Anweisung CREATE DATABASE in eine Programmiersprache eingebettet, so kann für *datenbank* auch eine alphanumerische Hostvariable benutzt werden.

IN *dbspace*
Name des Dbspace, in dem INFORMIX-ONLINE die Systemtabellen der Datenbank und standardmäßig auch die Tabellen speichert.

IN *dbspace* nicht angegeben:
Die Datenbank wird im Root-Dbspace gespeichert.

WITH-Klausel
Gibt die Art der Transaktionssicherung an.

WITH-Klausel nicht angegeben:
Die Datenbank wird ohne Transaktionssicherung erzeugt. Ohne Transaktionssicherung können Sie keine Transaktionen und keine Restaurierung der Datenbank bei einem Systemfehler durchführen.

LOG
Schaltet die Transaktionssicherung für die Datenbank ein. Die Transaktionsprotokollierung erfolgt ungepuffert.

BUFFERED LOG
Schaltet die Transaktionssicherung für die Datenbank ein. Die Transaktionsprotokollierung erfolgt gepuffert. Bei gepufferter Transaktionprotokollierung ist die Transaktionssicherheit geringer als bei ungepufferter Protokollierung. Dadurch können bei einer Restaurierung nach einem Systemfehler die letzten Transaktionen fehlen, die Performance wird jedoch durch gepufferte Protokollierung verbessert.

LOG MODE ANSI
Die Datenbank wird als ANSI-Datenbank erzeugt. Transaktionen sind implizit (siehe auch Kapitel 2). Die Transaktionsprotokollierung erfolgt ungepuffert. Eine ANSI-Datenbank kann nicht mehr in eine Nicht-ANSI-Datenbank umgewandelt werden.

Transaktionssicherung bei INFORMIX-ONLINE

Ist die Transaktionssicherung für eine Datenbank eingeschaltet, kann ein Benutzer zwischen gepufferter und ungepufferter Protokollierung wechseln (siehe Anweisung SET LOG).

Wird die Transaktionssicherung nicht beim Erzeugen der Datenbank mit CREATE DATABASE festgelegt, kann sie nachträglich nur der INFORMIX-ONLINE-Verwalter verändern.

Mit der Menüfunktion *Datenbank* im INFORMIX-Hauptmenü können Sie ebenfalls eine Datenbank erzeugen. Diese Datenbank besitzt jedoch keine Transaktionssicherung.

CREATE DATABASE in Transaktionen

CREATE DATABASE kann nicht mit ROLLBACK WORK zurückgesetzt werden.

CREATE DATABASE als dynamische Anweisung

Wird CREATE DATABASE mit PREPARE für eine dynamische Ausführung mit EXECUTE vorbereitet, so muß CREATE DATABASE die einzige Anweisung in dieser PREPARE-Anweisung sein.

Eigenschaften von ANSI-Datenbanken

Für ANSI-Datenbanken gilt im Unterschied zu Nicht-ANSI-Datenbanken:

- Die Transaktionssicherung ist immer eingeschaltet. ANSI-Datenbanken arbeiten immer mit Transaktionen.
- Die voreingestellte Isolationsstufe ist Repeatable Read.
  Die Isolationsstufe Repeatable Read gibt die Sicherheit, daß gelesene Sätze innerhalb der Transaktion von anderen Prozessen nicht verändert werden können. Zu diesem Zweck werden die bei Leseoperationen gesetzten Sperren bis zum Ende der Transaktion gehalten. Dadurch werden möglicherweise gleichzeitig sehr viele Sperren gehalten und die maximal zulässige Anzahl von Sperren kann überschritten werden.
- Fremde Datenbankobjekte müssen mit dem Namen des Eigentümers qualifiziert werden.
- Bei Anweisungen, die den ANSI-Standard verletzen, werden Warnungen ausgegeben.

Eine ANSI-Datenbank kann nicht mehr in eine Nicht-ANSI-Datenbank umgewandelt werden.

ANSI-Standard

Die Anweisung CREATE DATABASE ist nicht im ANSI-Standard enthalten.

Beispiel

Dieses Beispiel erzeugt die Datenbank *neudb* als Datenbank mit Transaktionssicherung. Die Transaktionsprotokollierung erfolgt ungepuffert. Die Datenbank wird im Dbspace *werbung* gespeichert.

```
CREATE DATABASE neudb IN werbung WITH LOG
```

>>>> DROP DATABASE, GRANT, START DATABASE

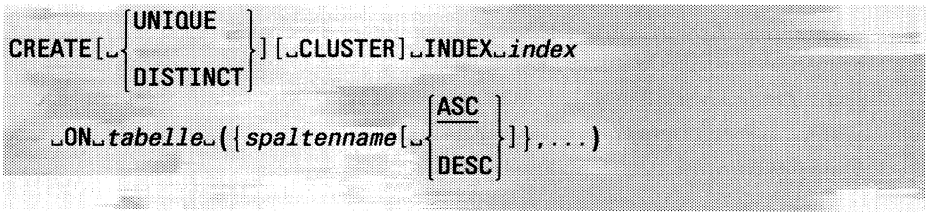
CREATE INDEX - Index erzeugen

CREATE INDEX erzeugt einen Index für eine oder mehrere Spalten einer Tabelle. Damit kann der Zugriff auf die Tabelle beschleunigt werden. Zusätzlich können Sie die Sätze der Tabelle in Reihenfolge des Index sortieren lassen (Cluster-Attribut).

Vor dem Aufruf beachten

Besitzen Sie das Resource-Zugriffsrecht, so müssen Sie zusätzlich entweder das INDEX-Zugriffsrecht haben oder aber Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

CREATE INDEX sperrt die Tabelle exklusiv gegen jeglichen Zugriff anderer Prozesse. CREATE INDEX wird abgewiesen, wenn ein anderer Prozeß die Tabelle bearbeitet. Dies gilt auch dann, wenn nur lesend zugegriffen wird.



UNIQUE
: Der Index muß eindeutige Werte enthalten. Ein Satz läßt sich nur dann in die Tabelle einfügen, wenn der Spaltenwert in der Tabelle noch nicht vorkommt. Auch ein NULL-Wert darf nur einmal vorkommen.

  Bei einem zusammengesetzten Index bezieht sich die Eindeutigkeit auf die Kombination der Spalten, die zum Index gehören. Ein Satz läßt sich nur dann in die Tabelle einspeichern, wenn die Werte-Kombination noch nicht vorkommt.

  UNIQUE bzw. DISTINCT nicht angegeben:
  Der Index kann doppelte Werte enthalten.

DISTINCT
: Diese Angabe ist gleichbedeutend mit UNIQUE.

CLUSTER

Die Sätze der Tabelle werden in der Reihenfolge des Index sortiert. Das ist in erster Linie sinnvoll, wenn Sie die Tabelle regelmäßig in Reihenfolge des Index verarbeiten. Aufgrund der physischen Anordnung der Sätze in der passenden Reihenfolge ergeben sich optimale Verhältnisse bezüglich der Plattenzugriffe und damit eine entsprechend verbesserte Performance.

Einschränkung:
Das Cluster-Attribut kann immer nur einem Index der Tabelle zugeordnet sein. Ist das Cluster-Attribut bereits einem anderen Index zugeordnet, so müssen Sie erst diese Zuordnung aufheben (siehe ALTER INDEX).
Sie können für eine Tabelle mit Audit-Protokollierung kein Cluster-Attribut festlegen.

CLUSTER nicht angegeben:
Die Sätze der Tabelle werden nicht sortiert gespeichert.

index

Name des neuen Index. Der Indexname darf aus max. 18 Zeichen bestehen und Buchstaben, Ziffern und Unterstriche (_) enthalten. Das erste Zeichen muß ein Buchstabe sein. Der Indexname darf nicht mit einem bereits existierenden Indexnamen der aktuellen Datenbank übereinstimmen.

Den Indexnamen benötigen Sie zur Identifizierung des Index, beispielsweise beim Löschen mit DROP INDEX.

Ein Benutzer mit DBA-Zugriffsrecht kann einen Index für einen anderen Benutzer erzeugen, indem er *index* mit dem Namen des Eigentümers qualifiziert. Ohne Qualifikation wird die zur Ablaufzeit aktuelle Benutzerkennung Eigentümer des Index.

tabelle

Name oder Synonym der Tabelle, für die der Index definiert wird. Es kann auch der Name einer temporären Tabelle angegeben werden. Dagegen ist der Name oder das Synonym eines View nicht erlaubt.

spaltenname

Spalte der oben genannten Tabelle, die den Index erhalten soll. Die Spalte darf nicht vom Datentyp TEXT oder BYTE sein. Wenn Sie mehrere Spalten angeben, entsteht ein zusammengesetzter Index.
Sie können bei einer INFORMIX-SE-Datenbank bis zu 8 Spalten angeben, deren Gesamtlänge 120 Byte nicht überschreiten darf.
Bei einer INFORMIX-ONLINE-Datenbank können Sie bis zu 16 Spalten angeben, deren Gesamtlänge 255 Byte nicht überschreiten darf.

Einschränkung:
Sie dürfen keinen Index erzeugen, der mit einem bereits vorhandenen Index vollständig übereinstimmt. Zwei Indizes dürfen nicht in allen folgenden Merkmalen übereinstimmen: Spalten, Sortierreihenfolge je Spalte oder Kombination der Spalten bei einem zusammengesetzten Index.

Beachten Sie, daß durch die UNIQUE CONSTRAINT-Klausel bei den Anweisungen CREATE TABLE bzw. ALTER TABLE ebenfalls ein Index erzeugt wird. Sie dürfen keinen Index erzeugen, der mit diesem durch UNIQUE CONSTRAINT erzeugten Index in allen Merkmalen übereinstimmt.

ASC

Der Index wird für die angegebene Spalte aufsteigend sortiert. Ein aufsteigend sortierter Index wird für die Optimierung von Suchfragen und die Ausgabe der Sätze in aufsteigender Reihenfolge benötigt (siehe SELECT, ORDER BY-Klausel). ASC ist Standard.

DESC

Der Index wird für die angegebene Spalte absteigend sortiert. Ein absteigend sortierter Index wird für die Ausgabe der Sätze in absteigender Reihenfolge benötigt (siehe SELECT, ORDER BY-Klausel).

Weder ASC noch DESC angegeben:
Es wird die Standard-Sortierreihenfolge ASC (aufsteigend) verwendet.

Arbeitsweise von CREATE CLUSTER INDEX

CREATE CLUSTER INDEX sortiert die Sätze der Tabelle, indem vorübergehend eine zweite Tabelle aufgebaut wird und die Sätze aus der Original-Tabelle sortiert in die zweite Tabelle geschrieben werden. Dies kann einige Zeit dauern und erfordert vorübergehend den Plattenspeicherplatz für die Kopie der Tabelle. Nach dem Sortieren wird die Original-Tabelle automatisch gelöscht und der Speicherplatz freigegeben.

Durch die Anweisung CREATE CLUSTER INDEX wird die Tabelle einmalig in der Reihenfolge des Index sortiert. Wenn Sie danach die Tabelle mit INSERT-, LOAD-, UPDATE- oder DELETE-Anweisungen bearbeiten, so geht dadurch im Laufe der Zeit die Sortierung verloren. Diese können Sie dann durch die Anweisung ALTER INDEX TO CLUSTER wiederherstellen.

CREATE INDEX in Transaktionen

Die Anweisung CREATE INDEX kann bei einer INFORMIX-SE-Datenbank nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

INFORMIX-ONLINE kann die Anweisung CREATE INDEX zurücksetzen.

ANSI-Standard

Die Anweisung CREATE INDEX ist nicht im ANSI-Standard enthalten.

Beispiel 1

Im ersten Beispiel wird für die Tabelle *auftrag* ein Index über die Spalte *kunden_nr* definiert. Der Index wird standardmäßig aufsteigend sortiert.

```
CREATE INDEX i1_kun ON auftrag (kunden_nr);

INFO INDEXES FOR auftrag
```

```
Indexname           Eigner    Typ         Cluster  Spalten

i1_kun              lomata    Duplikate   Nein     kunden_nr
```

Beispiel 2

Im zweiten Beispiel wird für die Tabelle *auftrag* ein Cluster-Index über die Spalte *auftrags_nr* definiert, d.h. die Sätze werden physisch nach dem Index sortiert. Die Sortierreihenfolge ist hier absteigend (DESC).

```
CREATE CLUSTER INDEX i2_auf ON auftrag (auftrags_nr DESC);

INFO INDEXES FOR auftrag
```

```
Indexname           Eigner    Typ        Cluster  Spalten
i1_kun              lomata    Duplikate  Nein     kunden_nr
i2_auf              lomata    Duplikate  Ja       auftrags_nr  Absteigend
```

```
SELECT auftrags_nr FROM auftrag
```

```
auftrags_nr
       1015
       1014
       1013
       1012
       1011
       1010
       1009
       1008
       1007
       1006
       1005
       1004
       1003
       1002
       1001
```

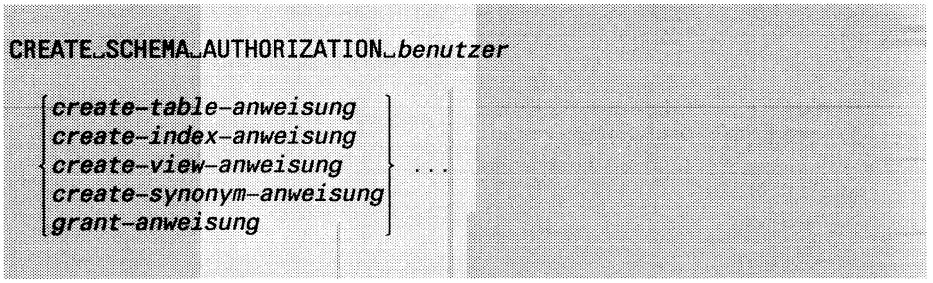
>>>> ALTER INDEX, ALTER TABLE, CREATE TABLE, DROP INDEX

CREATE SCHEMA - Datenbankschema erzeugen

CREATE SCHEMA erzeugt die Definition von Tabellen, Indizes, Views, Synonymen und Zugriffsrechten für die aktuelle Datenbank.

Vor dem Aufruf beachten

Die Datenbank, für die das Datenbankschema erzeugt wird, muß bereits mit CREATE DATABASE erzeugt sein. Sie müssen mindestens Resource-Zugriffsrecht auf diese Datenbank besitzen.



benutzer
Benutzerkennung, die Eigentümer der Datenbank und der Tabellen, Indizes, Views, Synonyme und Zugriffsrechte wird. Ein Benutzer mit Resource-Zugriffsrecht kann nur seine eigene Benutzerkennung angeben. Nur ein Benutzer mit DBA-Zugriffsrecht kann eine fremde Benutzerkennung angeben.

Achtung:
In den einzelnen CREATE-Anweisungen muß der Name des Benutzers nicht als Eigentümer angegeben werden. Werden die Namen von Tabellen, Indizes, Views und Synonymen mit einem Benutzernamen qualifiziert, so muß der bei CREATE SCHEMA angegebene Benutzer auch dort angegeben werden.

CREATE/GRANT-Klausel
Die einzelnen Anweisungen werden durch ein Leerzeichen getrennt oder sie beginnen in einer neuen Zeile. Ein Strichpunkt (;) oder das Datei-Ende schließt die Anweisung CREATE SCHEMA ab.

create-table-anweisung
CREATE TABLE-Anweisung, die eine Basistabelle für die Datenbank erzeugt.

create-index-anweisung
CREATE INDEX-Anweisung, die einen Index für eine Tabelle

erzeugt.

create-view-anweisung
CREATE VIEW-Anweisung, die einen View erzeugt.

create-synonym-anweisung
CREATE SYNONYM-Anweisung, die ein Synonym für eine Tabelle oder einen View definiert.

grant-anweisung
GRANT-Anweisung, die Zugriffsrechte für eine Tabelle oder einen View vergibt. Die GRANT-Anweisung darf nur Tabellenzugriffsrechte vergeben. Die Klausel AS *benutzer* darf nur die oben angegebene Benutzerkennung enthalten.

Arbeitsweise von CREATE SCHEMA

Die Anweisungen CREATE TABLE, CREATE INDEX, CREATE VIEW, CREATE SYNONYM und GRANT, die innerhalb der CREATE SCHEMA-Anweisung angegeben werden, werden genau in der angegebenen Reihenfolge ausgeführt. Anweisungen, die sich auf bereits existierende Tabellen oder Views beziehen, müssen deshalb nach der Anweisung stehen, die diese Tabellen bzw. Views erzeugt.

CREATE SCHEMA in Transaktionen

Die Anweisung CREATE SCHEMA kann bei einer INFORMIX-SE-Datenbank nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

INFORMIX-ONLINE kann die Anweisung CREATE SCHEMA zurücksetzen.

ANSI-Standard

Die Anweisung CREATE SCHEMA ist im ANSI-Standard enthalten.

Beispiel

Im folgenden Beispiel wird zuerst die Datenbank *natur* erzeugt. Die aktuelle Benutzerkennung besitzt dadurch DBA-Zugriffsrecht auf die Datenbank und kann für fremde Benutzer das Datenbankschema erzeugen.

```
CREATE DATABASE natur;

CREATE SCHEMA AUTHORIZATION karin
CREATE TABLE tiere (tname CHAR(20),
                    tgruppe CHAR(25),
                    rote_liste CHAR(1)
                   )
CREATE TABLE pflanzen (pname CHAR(20),
                       pgruppe CHAR(25),
                       rote_liste CHAR(1)
                      )
CREATE UNIQUE INDEX tnam ON tiere (tname)
CREATE UNIQUE INDEX pnam ON pflanzen (pname)
CREATE VIEW tiere_rot
            AS SELECT * FROM tiere WHERE rote_liste='j'
CREATE VIEW pflanzen_rot
            AS SELECT * FROM pflanzen WHERE rote_liste='j'
CREATE SYNONYM tr FOR tiere_rot
CREATE SYNONYM pr FOR pflanzen_rot
GRANT SELECT ON tiere TO PUBLIC
GRANT SELECT ON pflanzen TO PUBLIC
GRANT ALL ON tiere TO rudi
GRANT ALL ON pflanzen TO eckart
```

CREATE SYNONYM - Synonym definieren

CREATE SYNONYM definiert einen zusätzlichen Namen für eine Tabelle oder einen View.

Vor dem Aufruf beachten

Sie müssen das Connect-Zugriffsrecht besitzen.

```
CREATE␣SYNONYM␣synonym␣FOR␣tabelle
```

synonym

Synonym für die Tabelle oder den View. Das Synonym darf aus max. 18 Zeichen bestehen und Buchstaben, Ziffern und Unterstriche (_) enthalten. Das erste Zeichen muß ein Buchstabe sein. Bei einer Nicht-ANSI-Datenbank darf das Synonym nicht mit einem bereits existierenden Tabellen-, View- oder Synonymnamen der aktuellen Datenbank übereinstimmen. Für ANSI-Datenbanken muß der Name unter den bereits existierenden Tabellen-, View- oder Synonymnamen desselben Datenbankbenutzers eindeutig sein.

Ein Benutzer mit DBA-Zugriffsrecht kann ein Synonym für einen anderen Benutzer erzeugen, indem er *synonym* mit dem Namen des Eigentümers qualifiziert. Ohne Qualifikation wird die zur Ablaufzeit aktuelle Benutzerkennung Eigentümer des Synonyms.

tabelle

Name einer Basistabelle oder eines Views. Für eine temporäre Tabelle kann kein Synonym definiert werden.

Verwendung von Synonymen

In folgenden Anweisungen ist die Verwendung eines Synonyms erlaubt:

ALTER TABLE
CREATE AUDIT
CREATE INDEX
DELETE
DROP AUDIT
DROP SYNONYM
DROP VIEW
DROP TABLE
GRANT
INFO
INSERT
LOAD
LOCK TABLE
OUTPUT
RECOVER TABLE
RENAME COLUMN
RENAME TABLE
REVOKE
SELECT
UNLOAD
UNLOCK TABLE
UPDATE
UPDATE STATISTICS

Die bestehenden Zugriffsrechte für eine Tabelle gelten auch dann, wenn Sie unter dem Synonym auf die Tabelle zugreifen.

In folgenden Anweisungen ist die Verwendung eines Synonyms verboten:

CHECK TABLE
CREATE SYNONYM
REPAIR TABLE

CREATE SYNONYM in Transaktionen

Die Anweisung CREATE SYNONYM kann bei einer INFORMIX-SE-Datenbank nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

INFORMIX-ONLINE kann die Anweisung CREATE SYNONYM zurücksetzen.

ANSI-Standard

Die Anweisung CREATE SYNONYM ist nicht im ANSI-Standard enthalten.

Beispiel 1

Im ersten Beispiel wird für die Tabelle *hersteller* das Synonym *h* erzeugt. Der Synonymname wird anstelle des Tabellennamens bei der Abfrage verwendet.

```
CREATE SYNONYM h FOR hersteller;

SELECT * FROM h
```

```
herstellercode herstellername
SMT            Schmitt
ANZ            Anzabel
NRG            Norgesta
HSK            Hasken
HRO            Herold
```

Beispiel 2

Im folgenden Beispiel wird das Synonym *sk* für die Tabelle *kunde* des Benutzers *karin* definiert. Synonyme eignen sich besonders als Abkürzung für lange Tabellennamen, z.B. wenn bei einer ANSI-Datenbank die Tabelle eines anderen Benutzers verwendet wird.

```
CREATE SYNONYM sk FOR karin.neukund
```

Beispiel 3

Bei INFORMIX-ONLINE und INFORMIX-STAR können Sie Synonyme für externe Tabellen definieren (siehe Kapitel 3). Im folgenden Beispiel wird das Synonym *flager* für die Tabelle *lager* des Benutzers *mike* in der Datenbank *fremddb* definiert. Danach wird der Eintrag in der Systemtabelle *syssyntable* ausgegeben.

```
CREATE SYNONYM flager FOR fremddb:mike.lager

SELECT * FROM syssyntable WHERE
         tabid = (SELECT tabid FROM systables
                         WHERE tabname = "flager")
```

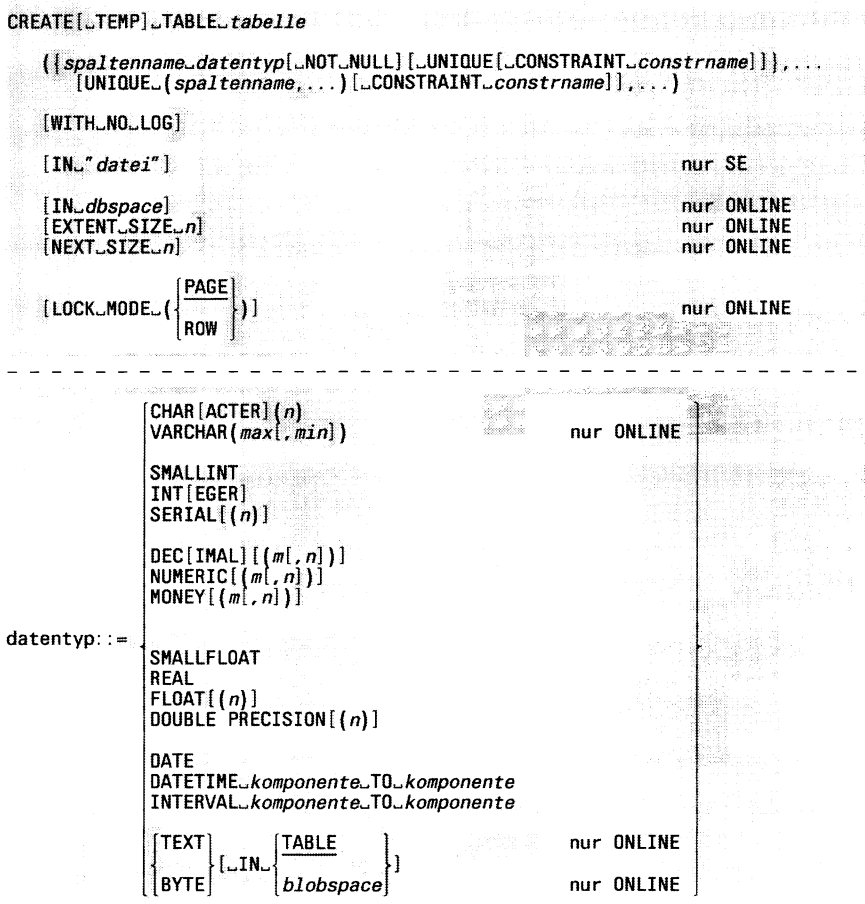
```
tabid       115
servername
dbname      fremddb
owner       mike
tabname     lager
btabid
```

CREATE TABLE - Tabelle erzeugen

CREATE TABLE erzeugt eine neue Tabelle in der aktuellen Datenbank. Für die Tabelle definieren Sie die Spalten, den Datentyp der Spalten und wahlweise Eindeutigkeitsbedingungen für Spaltenwerte.

Vor dem Aufruf beachten

Für das Erzeugen einer Basistabelle müssen Sie mindestens das Resource-Zugriffsrecht besitzen. Für eine temporäre Tabelle genügt das Connect-Zugriffsrecht.



TEMP

Die erzeugte Tabelle bleibt nur temporär bestehen, sie wird automatisch am Programmende gelöscht. Eine temporäre Tabelle kann nur der Prozeß bearbeiten, der sie erzeugt hat.

Bei einer INFORMIX-SE-Datenbank werden temporäre Tabellen standardmäßig in dem Dateiverzeichnis gespeichert, das durch die Umgebungsvariable DBTEMP festgelegt ist (standardmäßig /tmp).
Bei einer INFORMIX-ONLINE-Datenbank werden temporäre Tabellen im Root-Dbspace gespeichert.

TEMP nicht angegeben:
Es wird eine permanent gespeicherte Basistabelle erzeugt.

tabelle

Name der neuen Tabelle. Der Name darf aus max. 18 Zeichen bestehen und Buchstaben, Ziffern und Unterstriche (_) enthalten. Das erste Zeichen muß ein Buchstabe sein. Bei einer Nicht-ANSI-Datenbank darf der Name der Tabelle nicht mit einem bereits existierenden Tabellen-, View- oder Synonymnamen der aktuellen Datenbank übereinstimmen.

Für ANSI-Datenbanken muß der Name unter den bereits existierenden Tabellen-, View- oder Synonymnamen desselben Datenbankbenutzers eindeutig sein.

Ein Benutzer mit DBA-Zugriffsrecht kann eine Basistabelle für einen anderen Benutzer erzeugen, indem er *tabelle* mit dem Namen des Eigentümers qualifiziert. Ohne Qualifikation wird die zur Ablaufzeit aktuelle Benutzerkennung Eigentümer der Tabelle.

spaltenname
Name der Tabellenspalte. Der Name darf aus max. 18 Zeichen bestehen und Buchstaben, Ziffern und Unterstriche (_) enthalten. Das erste Zeichen muß ein Buchstabe sein. Der Name der Spalte muß innerhalb der Tabelle eindeutig sein.

datentyp
Datentyp für die Spalte. Eine Beschreibung der aufgeführten Datentypen finden Sie in Kapitel 4. Dort ist auch beschrieben, welche Werte in Spalten welchen Datentyps eingegeben werden können.

NOT NULL
Die Spalte läßt keine NULL-Werte als Eingabe zu.

NOT NULL nicht angegeben:
NULL-Werte sind als Eingabe für die Spalte zugelassen.

UNIQUE-Klausel
Erzwingt die Eindeutigkeit der Werte in der direkt zuvor genannten Spalte. Die Spalte darf nicht vom Datentyp TEXT oder BYTE sein. Zur Realisierung des Constraint wird ein eindeutiger Index über die Spalte erzeugt.

UNIQUE-Klausel nicht angegeben:
Es können doppelte Werte in der Spalte vorkommen.

CONSTRAINT *constrname*
Name für den Constraint. Der Name darf aus max. 18 Zeichen bestehen und Buchstaben, Ziffern und Unterstriche (_) enthalten. Das erste Zeichen muß ein Buchstabe sein. Der Name darf bei einer Nicht-ANSI-Datenbank nicht mit einem bereits existierenden Constraint-Namen der aktuellen Datenbank übereinstimmen. Bei ANSI-Datenbanken muß der Constraint-Name pro Datenbankbenutzer eindeutig sein.

Ein Benutzer mit DBA-Zugriffsrecht kann einen Constraint für einen anderen Benutzer erzeugen, indem er *constrname* mit dem Namen des Eigentümers qualifiziert. Ohne Qualifikation wird die zur Ablaufzeit aktuelle Benutzerkennung Eigentümer des Constraint.

CONSTRAINT *constrname* nicht angegeben:
Wird kein Constraint-Name angegeben, so vergibt INFORMIX einen Namen (siehe unten).

UNIQUE (*spaltenname,...*)
Im Unterschied zur obigen UNIQUE-Klausel kann hier die Eindeutigkeit mehrerer Spalten erzwungen werden. Die Spalte darf nicht vom Datentyp TEXT oder BYTE sein. Sie können bei einer INFORMIX-SE-Datenbank bis zu 8 Spalten angeben, deren Gesamtlänge 120 Byte nicht überschreiten darf.
Bei einer INFORMIX-ONLINE-Datenbank können Sie bis zu 16 Spalten angeben, deren Gesamtlänge 255 Byte nicht überschreiten darf.

Zur Realisierung des Constraint wird ein zusammengesetzter eindeutiger Index über die Spalten erzeugt.

CONSTRAINT *constrname*
siehe oben

CONSTRAINT *constrname* nicht angegeben:
Wird kein Constraint-Name angegeben, so vergibt INFORMIX einen Namen (siehe unten).

WITH NO LOG
Schaltet die Transaktionsprotokollierung für die temporäre Tabelle aus. Wurde die Transaktionsprotokollierung ausgeschaltet, so kann sie nicht mehr eingeschaltet werden. Die Klausel WITH NO LOG kann nur für eine temporäre Tabelle angegeben werden.
WITH NO LOG kann auch für eine temporäre Tabelle in einer ANSI-Datenbank verwendet werden.

WITH NO LOG nicht angegeben:
Auch temporäre Tabellen unterliegen der Transaktionsprotokollierung, wenn die Datenbank mit Transaktionssicherung definiert wurde.

IN "*datei*"

Name der Datei, in der die Tabelle einer INFORMIX-SE-Datenbank gespeichert wird. *datei* muß mit dem absoluten Pfadnamen angegeben werden. Alle Dateiverzeichnisse im Pfad müssen bereits existieren. Der gesamte Pfadname darf maximal 64 Zeichen lang sein. Davon dürfen maximal 10 Zeichen auf den Basisnamen der Datei selbst entfallen.

Der Basisname der Datei bestimmt den Namen, den die Tabellen-Dateien erhalten (siehe *Tabellen einer INFORMIX-SE-Datenbank auf Betriebssystemebene*).

Bei einer temporären Tabelle wird die Klausel IN "*datei*" ignoriert.

Einschränkung:
Die Klausel IN "*datei*" kann nur bei einer INFORMIX-SE-Datenbank angegeben werden.

IN "*datei*" nicht angegeben:
Die Tabelle wird bei einer INFORMIX-SE-Datenbank im Dateiverzeichnis der Datenbank gespeichert. Die Dateinamen leiten sich aus dem Tabellennamen ab (siehe *Tabellen einer INFORMIX-SE-Datenbank auf Betriebssystemebene*).

IN *dbspace*

Name des Dbspace, in dem INFORMIX-ONLINE die Tabelle speichern soll.

Einschränkung:
Die Klausel IN *dbspace* kann nur bei einer INFORMIX-ONLINE-Datenbank angegeben werden.

IN *dbspace* nicht angegeben:
Die Tabelle wird im Dbspace der Datenbank gespeichert.

EXTENT SIZE *n*

Größe des Initial-Extent für die Tabelle und deren Indizes. *n* gibt die Größe in Kbyte an, wobei *n* eine Zahl zwischen 8 und 32767 ist. Hinweise zur Einschätzung der Größe einer Tabelle finden Sie im Anhang A.4 beschrieben.

Einschränkung: Die Klausel EXTENT SIZE kann nur bei einer INFORMIX-ONLINE-Datenbank angegeben werden.

EXTENT SIZE *n* nicht angegeben:
INFORMIX-ONLINE verwendet den Standardwert (16 Kbyte = 8 Pages).

NEXT SIZE *n*

Größe der weiteren Extents, die bei Bedarf hinzugefügt werden. *n* gibt die Größe in Kbyte an, wobei *n* eine Zahl zwischen 8 und 32767 ist. Hinweise zur Einschätzung der Größe einer Tabelle finden Sie im Anhang A.4 beschrieben.

Einschränkung:
Die Klausel NEXT SIZE kann nur bei einer INFORMIX-ONLINE-Datenbank angegeben werden.

NEXT SIZE *n* nicht angegeben:
INFORMIX-ONLINE verwendet den Standardwert (16 Kbyte = 8 Pages).

LOCK MODE-Klausel
Die LOCK MODE-Klausel legt fest, ob Sperren auf den Bereich einer Page oder eines Satzes gesetzt werden.

Einschränkung:
Die Klausel LOCK MODE kann nur bei einer INFORMIX-ONLINE-Datenbank angegeben werden.

LOCK MODE-Klausel nicht angegeben:
Sperrbereich ist Page.

PAGE
Sperrbereich ist Page. PAGE ist Voreinstellung.

ROW
Sperrbereich ist Satz.

Constraint-Name

Wenn Sie dem Constraint keinen Namen geben, so erzeugt INFORMIX einen Namen nach dem Schema:
u*tabid*_*indexnummer*.

Der zugehörende Index hat den Namen:
␣*tabid*_*indexnummer*.

u
von INFORMIX vergebenes Präfix (für UNIQUE).

tabid
von INFORMIX vergebene Nummer der Tabelle.

indexnummer
von INFORMIX vergebene Nummer des Index.

Zur Realisierung des Constraint wird ein eindeutiger Index über die Spalten erzeugt. Der Index ist aufsteigend sortiert. Sie dürfen deshalb keinen Index erzeugen, der genau diesem, durch den Constraint erzeugten Index in allen Merkmalen gleicht (siehe auch CREATE INDEX).

Der von INFORMIX erzeugte Name des Constraint kann von der Systemtabelle *sysconstraints* abgefragt werden.

```
SELECT constrname FROM sysconstraints
       WHERE tabid = (SELECT tabid FROM systables
                              WHERE tabname = "tabelle")
```

Tabellen einer INFORMIX-SE-Datenbank auf Betriebssystemebene

INFORMIX-SE legt für jede Tabelle zwei Dateien an. Die Dateien erhalten das Suffix *.dat* für Daten und *.idx* für Index.

Wird mit der Klausel IN "*datei*" ein Dateiname für die Tabelle angegeben, so erhalten die Tabellen-Dateien folgende Namen: *basisname.dat* und *basisname.idx*.

Beispiel:

CREATE TABLE adresse ..... IN "/usr1/karin/db/adr"

Die Tabellen-Dateien heißen in diesem Fall *adr.dat* und *adr.idx* und sind im Dateiverzeichnis */usr1/karin/db* gespeichert.

Wird die Klausel IN "*datei*" nicht angegeben, so erzeugt INFORMIX-SE die Dateinamen nach dem Muster *tabnnn.dat* bzw. *tabnnn.idx*. Die Dateien werden im Dateiverzeichnis der Datenbank *datenbank.dbs* abgespeichert.

tab
: Ergibt sich aus den ersten 7 Stellen des Tabellennamens. Ist der Name kürzer, so werden Unterstriche (_) ergänzt.

nnn
: Von INFORMIX vergebene Nummer.

.dat
: Von INFORMIX vergebenes Suffix.

.idx
: Von INFORMIX vergebenes Suffix.

Alle Benutzer, die auf die Tabelle zugreifen wollen, müssen Ausführungsberechtigung (x-Bit) für alle Dateiverzeichnisse im Pfad und Leseberechtigung (r-Bit) für die Tabellen- und Index-Dateien besitzen.
Alle Benutzer, die die Tabelle verändern wollen, müssen Schreibberechtigung (w-Bit) für die Tabellen- und Index-Dateien besitzen.

Standard-Zugriffsrechte

In einer Nicht-ANSI-Datenbank werden automatisch bei jeder neu erzeugten Tabelle folgende Tabellenzugriffsrechte gesetzt:

```
GRANT DELETE,INSERT,SELECT,UPDATE TO PUBLIC
```

Nur der Eigentümer der Tabelle oder ein Benutzer mit DBA-Zugriffsrecht kann standardmäßig die Tabellenstruktur ändern oder die Tabelle löschen. Die Zugriffsrechte können mit GRANT und REVOKE verändert werden.

Bei einer ANSI-Datenbank hat kein anderer Benutzer als der Eigentümer standardmäßig Zugriffsrechte auf die Tabelle. Alle Zugriffsrechte müssen Sie explizit mit GRANT vergeben.

CREATE TABLE in Transaktionen

Die Anweisung CREATE TABLE kann bei einer INFORMIX-SE-Datenbank nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

INFORMIX-ONLINE kann die Anweisung CREATE TABLE zurücksetzen.

ANSI-Standard

Die Anweisung CREATE TABLE ist im ANSI-Standard enthalten, dort aber im Gegensatz zu INFORMIX nur innerhalb der Anweisung CREATE SCHEMA erlaubt. Die Klauseln TEMP, DISTINCT, UNIQUE CONSTRAINT, WITH NO LOG und IN sind nicht im ANSI-Standard enthalten. Ebenfalls nicht im ANSI-Standard enthalten sind die Datentypen SERIAL, DATE, MONEY, SMALLFLOAT, DATETIME und INTERVAL sowie alle Klauseln und Datentypen, die speziell für INFORMIX-ONLINE gelten.

Beispiel 1

Das folgende Beispiel erstellt die Tabelle *kunde* der Beispieldatenbank *versand*.

```
CREATE TABLE kunde
(
   kunden_nr        SERIAL(101),
   vorname          CHAR(15),
   nachname         CHAR(15),
   firma            CHAR(20),
   adresse1         CHAR(20),
   adresse2         CHAR(20),
   ort              CHAR(15),
   bundesland       CHAR(2),
   plz              CHAR(5),
   telefon          CHAR(18)
)
```

Beispiel 2

Das folgende Beispiel erstellt die Tabelle *bilder*. Die Datentypen VARCHAR, TEXT und BYTE können nur bei einer INFORMIX-ONLINE-Datenbank verwendet werden.

```
CREATE TABLE bilder
(             bildnum INTEGER,
              ueberschrift VARCHAR(100,20),
              beschreibung TEXT IN TABLE,
              bild BYTE
)
```

>>>> ALTER TABLE, DROP TABLE, GRANT, RENAME COLUMN, RENAME TABLE, REVOKE

CREATE VIEW - View erzeugen

CREATE VIEW erzeugt einen View aus bereits bestehenden Tabellen und Views. Auf einer temporären Tabelle kann kein View erzeugt werden.

Vor dem Aufruf beachten

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich entweder das Select-Zugriffsrecht haben oder aber Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

```
CREATE␣VIEW␣view[␣(spaltenname,...)]␣AS␣select-anweisung
          [␣WITH␣CHECK␣OPTION]
```

view
Name des neuen View. Der Name darf aus max. 18 Zeichen bestehen und Buchstaben, Ziffern und Unterstriche (_) enthalten. Das erste Zeichen muß ein Buchstabe sein. Bei einer Nicht-ANSI-Datenbank darf der Name des Views nicht mit einem bereits existierenden Tabellen-, View- oder Synonymnamen der aktuellen Datenbank übereinstimmen. Für ANSI-Datenbanken muß der Name unter den bereits existierenden Tabellen-, View- oder Synonymnamen desselben Datenbankbenutzers eindeutig sein.

Ein Benutzer mit DBA-Zugriffsrecht kann einen View für einen anderen Benutzer erzeugen, indem er *view* mit dem Namen des Eigentümers qualifiziert. Ohne Qualifikation wird die zur Ablaufzeit aktuelle Benutzerkennung Eigentümer des View.

spaltenname
Name der Spalte des Views. Der Name darf aus max. 18 Zeichen bestehen und Buchstaben, Ziffern und Unterstriche (_) enthalten. Das erste Zeichen muß ein Buchstabe sein. Der Name der Spalte muß innerhalb des Views eindeutig sein.

Die Viewspalten müssen nur dann benannt werden, wenn die Spaltennamen der zugrundeliegenden Tabellen nicht eindeutig sind oder wenn virtuelle Spalten (bei Berechnungen) vorkommen. Wenn die Spalten des View benannt werden, dann müssen alle Spalten einen Namen erhalten.

Beispiel:

```
CREATE VIEW sicht1
       (artikelnum,hstcode1,bezeichnung,hstcode2,hname)
       AS SELECT artikel_nr,artikel.herstellercode,
                 bezeichnung, hersteller.herstellercode,
                 herstellername
          FROM artikel,hersteller
          WHERE artikel.herstellercode=hersteller.herstellercode
```

Im obigen Beispiel tritt die Spalte *herstellercode* zweimal auf. Deshalb müssen alle Spalten im View einen Namen erhalten.

spaltenname,... nicht angegeben:
Es gelten die Spaltennamen aus der SELECT-Anweisung.

select-anweisung

SELECT-Anweisung, die aus bereits bestehenden Tabellen und Views die Spalten und Sätze auswählt, die den neuen View bilden sollen. Wenn Sie in der SELECT-Anweisung Synonyme verwenden, können Sie mit dem View nicht mehr arbeiten, wenn diese Synonyme gelöscht werden.

Die Spalten des Views besitzen denselben Datentyp wie die zugrundeliegende Spalte aus der SELECT-Anweisung. Spalten, die als Ergebnis von Berechnungen entstehen, haben den Datentyp, der sich aus der Berechnung ergibt.

Werden im View Spalten benannt, so muß die Anzahl der Spalten in der Ergebnistabelle des SELECT mit der Anzahl der im View genannten Spalten übereinstimmen.

Wenn Sie mit SELECT * alle Spalten einer Tabelle für den View auswählen, dann gelten für den View alle Spalten der Tabelle, die zum Zeitpunkt des CREATE VIEW vorhanden sind.

Einschränkung:
Die SELECT-Anweisung darf keine ORDER BY-, keine INTO-, keine INTO TEMP- und keine UNION-Klausel enthalten. Die SELECT-Anweisung darf keine Hostvariablen enthalten.

WITH CHECK OPTION

Sätze, die Sie über den View eingeben oder ändern, werden auf die Einhaltung der in der SELECT-Anweisung definierten Bedingung (WHERE-Klausel) überprüft. Sätze, die die Bedingung nicht erfüllen, werden abgewiesen.

WITH CHECK OPTION nicht angegeben:
Es können Sätze in den View eingefügt oder geändert werden, die die Bedingung der SELECT-Anweisung nicht erfüllen. Solche Sätze sind anschließend nicht mehr über den View zugreifbar.

Verwendung von Views

In folgenden Anweisungen ist die Verwendung eines Views erlaubt:

CREATE SYNONYM	LOAD
CREATE VIEW	OUTPUT
DELETE	REVOKE
DROP VIEW	SELECT
GRANT	UNLOAD
INFO	UPDATE
INSERT	UPDATE STATISTICS

Die bestehenden Zugriffsrechte auf eine Tabelle gelten auch dann, wenn Sie durch einen View auf die Tabelle zugreifen.

In folgenden Anweisungen ist die Angabe eines Views verboten:

ALTER TABLE	LOCK TABLE
CHECK TABLE	RECOVER TABLE
CREATE AUDIT	RENAME COLUMN
CREATE INDEX	RENAME TABLE
DROP AUDIT	REPAIR TABLE
DROP TABLE	UNLOCK TABLE

Änderbare Views

Einfüge- und Änderungsanweisungen sind bei einem View nur erlaubt, wenn aus der View-Definition eindeutig hervorgeht, welche Sätze einer Basistabelle betroffen sind.
Änderbare Views dürfen deshalb in der SELECT-Anweisung die Klauseln ORDER BY, GROUP BY, DISTINCT bzw. UNIQUE und HAVING nicht enthalten; in der Spaltenauswahl dürfen keine Mengenfunktionen auftreten. Die SELECT-Anweisung darf in der FROM-Klausel nur eine Tabelle enthalten. Die Verwendung von anderen Tabellen in Unterabfragen ist möglich, wenn die Unterabfrage nicht korreliert ist.

DELETE-Anweisungen sind auf alle änderbaren Views möglich.

INSERT- und LOAD-Anweisungen sind möglich für Views, die zusätzlich zu den oben genannten Einschränkungen alle mit NOT NULL definierten Spalten der Basistabelle enthalten und die keine berechneten Spalten besitzen.

UPDATE-Anweisungen können nur die Spalten ändern, die nicht durch Berechnungen entstehen.

Die Zugriffsrechte auf einen View können Sie mit der INFO-Anweisung abfragen. Es wird ausgegeben, ob das Delete-, Update- oder Insert-Zugriffsrecht auf den View besteht und welche Spalten mit UPDATE geändert werden können.

CREATE VIEW in Transaktionen

Die Anweisung CREATE VIEW kann bei einer INFORMIX-SE-Datenbank nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

INFORMIX-ONLINE kann die Anweisung CREATE VIEW zurücksetzen.

ANSI-Standard

Die Anweisung CREATE VIEW ist im ANSI-Standard enthalten, dort aber im Gegensatz zu INFORMIX nur innerhalb der Anweisung CREATE SCHEMA erlaubt.

Beispiel

Im folgenden Beispiel wird ein View aus der Tabelle *kunde* erzeugt. Der View enthält alle Spalten der Tabelle *kunde*, aber nur die Sätze, die in der Spalte *ort* den Wert *Muenchen* haben.

```
CREATE VIEW muenchen AS SELECT * FROM kunde WHERE ort="Muenchen";

SELECT vorname,nachname,ort FROM muenchen
```

```
vorname         nachname        ort
Anton           Hochfeld        Muenchen
Martin          Korting         Muenchen
Roland          Jaeger          Muenchen
Frank           Albert          Muenchen
Arnold          Sipell          Muenchen
```

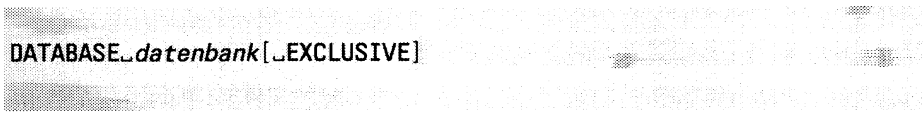
> > > > CREATE TABLE, DROP VIEW

DATABASE - Datenbank eröffnen

DATABASE eröffnet eine Datenbank. Wenn bereits eine Datenbank eröffnet ist, wird diese zuerst geschlossen.

Vor dem Aufruf beachten

Sie müssen das Connect-Zugriffsrecht für die Datenbank besitzen.



datenbank
Name der Datenbank, die eröffnet werden soll.
INFORMIX-SE sucht die Datenbank, d.h. das Dateiverzeichnis *datenbank.dbs* im aktuellen Dateiverzeichnis. Ist die Datenbank dort nicht vorhanden, sucht INFORMIX-SE im Dateiverzeichnis, das durch die Umgebungsvariable DBPATH angegeben wird.
INFORMIX-ONLINE verwaltet eine Datenbank unabhängig vom Dateisystem. Die Umgebungsvariable DBPATH gibt bei INFORMIX-STAR an, in welchem Informixsystem die Datenbank gesucht wird (siehe Handbuch für INFORMIX-STAR [8]).

Bei Programmeinbettung können Sie *datenbank* über eine alphanumerische Variable angeben. In diesem Fall können Sie bei INFORMIX-SE auch den absoluten Pfadnamen angeben, allerdings ohne das Suffix *.dbs*.

EXCLUSIVE
Datenbank exklusiv für den aktuellen Benutzer öffnen, der diese Anweisung ausführt. Die Datenbank bleibt solange für alle anderen Benutzer gesperrt, bis sie mit CLOSE DATABASE geschlossen wird.

EXCLUSIVE nicht angegeben: Auch andere Benutzer können auf die Datenbank zugreifen.

Datenbank-Anweisungen ohne eröffnete Datenbank

Wenn keine Datenbank eröffnet ist, können Sie nur die folgenden Anweisungen verwenden:

CREATE DATABASE
DROP DATABASE
DATABASE
ROLLFORWARD DATABASE
START DATABASE

DATABASE bei INFORMIX-4GL

Bei INFORMIX-4GL erfüllt die Anweisung DATABASE zusätzlich die Aufgabe, dem 4GL-Precompiler die Datenbank bekanntzugeben, die Informationen zu allen mit LIKE definierten Variablen enthält (siehe INFORMIX-4GL-Handbuch [4], Anweisung DEFINE).
In diesem Fall muß die DATABASE-Anweisung die erste Anweisung eines Moduls sein, darf die Klausel EXCLUSIVE nicht enthalten und der Name der Datenbank darf nicht mit einer Variablen angegeben werden.

Mit LIKE definierte Variablen können sich in einem Modul nur auf eine Datenbank beziehen. Wenn in einem 4GL-Modul eine Datenbank geschlossen und eine zweite Datenbank eröffnet wird, kann daher für die zweite Datenbank keine Variable mit LIKE definiert werden.

Abfragen des Datenbanktyps

Nach Ausführung der DATABASE-Anweisung können Sie über die Variable SQLCA den Typ der Datenbank feststellen.

Abfrage bei INFORMIX-4GL:
SQLCA.SQLAWARN[2] = 'W' bei Datenbank mit Transaktionssicherung
SQLCA.SQLAWARN[3] = 'W' bei ANSI-Datenbank
SQLCA.SQLAWARN[4] = 'W' bei INFORMIX-ONLINE-Datenbank

Abfrage bei INFORMIX-ESQL/C:
sqlca.sqlwarn1 = 'W' bei Datenbank mit Transaktionssicherung
sqlca.sqlwarn2 = 'W' bei ANSI-Datenbank
sqlca.sqlwarn3 = 'W' bei INFORMIX-ONLINE-Datenbank

Abfrage bei INFORMIX-ESQL/COBOL:
SQLAWARN[2] = 'W' bei Datenbank mit Transaktionssicherung
SQLAWARN[3] = 'W' bei ANSI-Datenbank
SQLAWARN[4] = 'W' bei INFORMIX-ONLINE-Datenbank

DATABASE als dynamische Anweisung

Wird DATABASE mit PREPARE für eine dynamische Ausführung mit EXECUTE vorbereitet, so muß DATABASE die einzige Anweisung in dieser PREPARE-Anweisung sein.

DATABASE in Transaktionen

Die Anweisung DATABASE kann nicht mit ROLLBACK WORK zurückgesetzt werden.

ANSI-Standard

Die Anweisung DATABASE ist nicht im ANSI-Standard enthalten.

Beispiel

Die Datenbank *versand* wird eröffnet. Andere Benutzer können ebenfalls auf die Datenbank zugreifen.

```
DATABASE versand
```

>>>> CLOSE DATABASE, CREATE DATABASE, DROP DATABASE

DECLARE - Satzzeiger vereinbaren

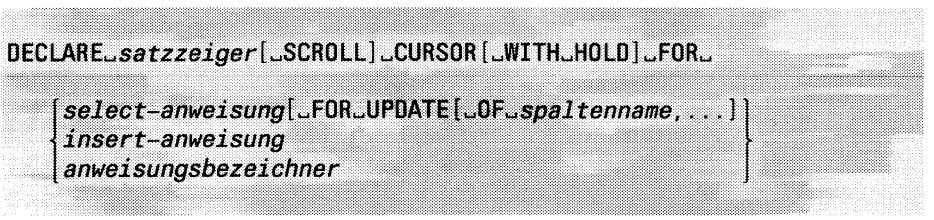
DECLARE vereinbart einen Satzzeiger für eine SELECT- oder eine INSERT-Anweisung.

Mit dem Satzzeiger für SELECT-Anweisungen können Sie auf die einzelnen Sätze der Ergebnistabelle zugreifen. Sie können die Sätze ausgeben, löschen, verändern oder die Werte der Spalten auf andere Art in Ihrem Programm verarbeiten. Ein mit DECLARE vereinbarter Select-Satzzeiger wird für nachfolgende OPEN-, FETCH-, FOREACH-, DELETE-, UPDATE- oder CLOSE-Anweisungen benötigt.

Satzzeiger für INSERT-Anweisungen vereinbaren Sie, um Sätze beim Einfügen in die Datenbank zu puffern. Ein mit DECLARE vereinbarter Insert-Satzzeiger wird für nachfolgende OPEN-, PUT-, FLUSH- oder CLOSE-Anweisungen benötigt.

Vor dem Aufruf beachten

Die DECLARE-Anweisung muß in der Quellprogrammdatei statisch vor der ersten Anweisung stehen, die den vereinbarten Satzzeiger benutzt. Alle Anweisungen, die diesen Satzzeiger verwenden, müssen in derselben Datei stehen.



satzzeiger
: Name für den Satzzeiger. Der Name darf aus max. 18 Zeichen bestehen und Buchstaben, Ziffern und Unterstriche (_) enthalten. Das erste Zeichen muß ein Buchstabe sein. Der Name des Satzzeigers muß in der Quellprogrammdatei eindeutig sein. Nur in dieser Datei ist er global gültig.

SCROLL
: Der Satzzeiger kann in beliebiger Reihenfolge auf die Sätze der Ergebnistabelle zugreifen (siehe FETCH).

  SCROLL dürfen Sie nur angeben, wenn der Satzzeiger für eine SELECT-Anweisung ohne FOR UPDATE vereinbart wird.

SCROLL nicht angegeben:
Wenn Sie SCROLL nicht angeben, dann können Sie mit dem Satzzeiger nur einen Satz nach dem anderen ansprechen. Sie können nicht zurückpositionieren und auch keine Sätze überspringen.

WITH HOLD
Der Satzzeiger bleibt über das Ende einer Transaktion hinaus geöffnet. Der Satzzeiger muß mit CLOSE explizit geschlossen werden.

Achtung:
Da am Ende einer Transaktion alle Einzelsperren freigegeben werden, kann der Fall auftreten, daß andere Prozesse Sätze ändern oder löschen, auf die der Satzzeiger mit WITH HOLD zeigt. Dieser Fall kann auch durch die Isolationsstufen bei INFORMIX-ONLINE nicht beeinflußt werden.

WITH HOLD nicht angegeben:
Der Satzzeiger wird am Ende einer Transaktion geschlossen.

select-anweisung
SELECT-Anweisung, auf deren Ergebnistabelle mit dem Satzzeiger zugegriffen wird. Die SELECT-Anweisung darf keine INTO TEMP-Klausel enthalten. Die SELECT-Anweisung kann Hostvariablen in der WHERE-Klausel und in der Klausel INTO *variable,...* enthalten. Die Variablen nehmen dann die Spaltenwerte des jeweils aktuellen Satzes auf. Zum Zeitpunkt der Auswertung von Variablen siehe unten.

FOR UPDATE
Der aktuelle Satz der zugrundeliegenden Basistabelle soll mit den Anweisungen UPDATE oder DELETE mit der Klausel WHERE CURRENT OF geändert oder gelöscht werden.

Wenn Sie einen Satzzeiger mit FOR UPDATE vereinbaren, dann setzt INFORMIX eine Sperre auf den jeweils aktuellen Satz (siehe *Sperren zum Ändern*).

Einschränkung:
Der Satzzeiger darf nicht mit SCROLL vereinbart sein. Die SELECT-Anweisung darf nur auf eine einzelne Tabelle zugreifen (kein Join) und nicht die Klauseln GROUP BY, HAVING, ORDER BY und DISTINCT bzw. UNIQUE enthalten. Es dürfen keine Mengenfunktionen im SELECT verwendet werden.

FOR UPDATE nicht angegeben:
Der aktuelle Satz der zugrundeliegenden Basistabelle kann nicht mit einem nachfolgenden UPDATE WHERE CURRENT OF oder DELETE WHERE CURRENT OF bearbeitet werden.

OF *spaltenname,...*
Das Ändern des aktuellen Satzes mit UPDATE WHERE CURRENT OF wird auf die angegebenen Spalten eingeschränkt.
Geben Sie für *spaltenname* eine Spalte der Tabelle an, die im SELECT bei FROM genannt ist. Die Spalte selbst muß nicht in der SELECT-Anweisung vorkommen. Die Angabe der Spalten beschleunigt die UPDATE-Anweisung, die sich auf diese Spalten bezieht.

Für eine DELETE-Anweisung bleibt diese Angabe ohne Wirkung, da sie den ganzen Satz löscht, auf den der Satzzeiger positioniert ist.

OF *spalte,...* nicht angegeben:
Ein UPDATE WHERE CURRENT OF kann jede Spalte ändern, für die die entsprechenden Zugriffsrechte bestehen.

insert-anweisung
INSERT-Anweisung, die Hostvariable, aber keine untergeordnete SELECT-Anweisung enthalten darf.

anweisungsbezeichner
Bezeichner einer dynamischen SELECT- oder INSERT-Anweisung, die mit PREPARE vorbereitet wurde.
Die dynamische SELECT-Anweisung kann eine FOR UPDATE-Klausel enthalten (siehe PREPARE).

Auswertung von Variablen

Der Zeitpunkt, zu dem der Wert von Hostvariablen ausgewertet wird, unterscheidet sich je nach dem, ob einfache Variablen oder Arrays verwendet werden.

- Einfache Variablen werden beim Öffnen des Satzzeigers mit OPEN bzw. FOREACH ausgewertet.
- Bei Array-Variablen wird die Indexvariable bei der Definition des Satzzeigers mit DECLARE ausgewertet, der Wert des Array-Elements selbst erst beim Öffnen des Satzzeigers mit OPEN bzw. FOREACH.

Beispiel für INFORMIX/ESQL/C

```
$int zahl[5] ;
int i;

zahl[1] = 10;
zahl[2] = 20;
zahl[3] = 30;
zahl[4] = 40;
zahl[5] = 50;
i=3;

$ DATABASE versand
$ DECLARE selzeig CURSOR FOR
     SELECT * FROM kunde WHERE kunden_nr > $zahl[i];
zahl[3] = 100;
i=1;
$ OPEN selzeig;
```

i wird zum Zeitpunkt des DECLARE ausgewertet, also mit dem Wert 3. Beim OPEN wird dann der zu diesem Zeitpunkt aktuelle Wert von *zahl[3]* verwendet, also 100.

Sperren zum Ändern

Wenn Sie einen Satzzeiger mit FOR UPDATE vereinbaren, dann setzt INFORMIX eine Sperre auf den jeweils aktuellen Satz, wenn der Satz mit FETCH gelesen wird. Welche Sperre gesetzt wird und wie lange sie gehalten wird, hängt davon ab, ob es sich um eine INFORMIX-SE- oder eine INFORMIX-ONLINE-Datenbank handelt.

INFORMIX-SE-Datenbank:
Auf den aktuellen Satz wird eine Exclusive-Sperre gesetzt. Gelingt dies nicht, weil bereits ein anderer Benutzer eine Sperre auf den Satz hält, wird der FETCH abgewiesen. Wird der aktuelle Satz nicht verändert, so bleibt die Exklusiv-Sperre erhalten, bis Sie den Satzzeiger mit FETCH auf den nächsten Satz positionieren. Wird der aktuelle Satz mit UPDATE oder DELETE verändert, so bleibt die Exclusive-Sperre bei einer Datenbank mit Transaktionssicherung bis zum Ende der Transaktion erhalten.

INFORMIX-ONLINE-Datenbank:
Auf den aktuellen Satz wird eine Update-Sperre gesetzt. Wird der aktuelle Satz nicht verändert, so wird die Update-Sperre freigegeben, wenn Sie den Satzzeiger mit FETCH auf den nächsten Satz positionieren. Wird der aktuelle Satz mit UPDATE oder DELETE WHERE CURRENT OF verändert, so versucht INFORMIX-ONLINE die Update-Sperre in eine Exclusive-Sperre umzuwandeln. Gelingt dies nicht, weil eine anderer Benutzer eine Share-Sperre auf den Satz hält, so wird der UPDATE bzw. DELETE abgewiesen. Die gesetzte Exclusive-Sperre wird bei einer Datenbank mit Transaktionssicherung bis zum Ende der Transaktion gehalten.

Verwendung von Satzzeigern in Transaktionen

Für eine Datenbank mit Transaktionssicherung gelten folgende Regeln zur Verwendung von Satzzeigern:

- Select-Satzzeiger ohne FOR UPDATE
  Ein Select-Satzzeiger ohne FOR UPDATE kann beliebig innerhalb und außerhalb einer Transaktion geöffnet, geschlossen und mit FETCH gelesen werden.
- Select-Satzzeiger mit FOR UPDATE
  - ohne WITH HOLD
    Ein Select-Satzzeiger mit FOR UPDATE und ohne WITH HOLD kann nur innerhalb einer Transaktion mit OPEN geöffnet und mit CLOSE geschlossen werden. Jede FETCH-Anweisung auf diesen Satzzeiger muß ebenfalls innerhalb der gleichen Transaktion stattfinden, da der Satzzeiger ohne WITH HOLD am Ende der Transaktion automatisch geschlossen wird. UPDATE- und DELETE-Anweisungen für diesen Satzzeiger müssen innerhalb der gleichen Transaktion stattfinden.
  - mit WITH HOLD
    Ein Select-Satzzeiger mit FOR UPDATE und WITH HOLD kann außerhalb einer Transaktion mit OPEN geöffnet und mit CLOSE geschlossen werden. Eine FETCH-Anweisung auf diesen Satzzeiger kann außerhalb einer Transaktion stattfinden. Soll ein Satz aber mit DELETE/UPDATE WHERE CURRENT OF bearbeitet werden, so muß die FETCH-Anweisung, die den Satz liest und die DELETE- bzw. UPDATE-Anweisung innerhalb einer Transaktion stattfinden.

- Insert-Satzzeiger ohne WITH HOLD
  OPEN-, PUT-, FLUSH und CLOSE-Anweisungen für den Insert-Satzzeiger ohne WITH HOLD können nur innerhalb einer Transaktion stattfinden.
- Insert-Satzzeiger mit WITH HOLD
  OPEN-, PUT-, FLUSH und CLOSE-Anweisungen für den Insert-Satzzeiger mit WITH HOLD können auch außerhalb einer Transaktion stattfinden.

Für eine ANSI-Datenbank sind diese Regeln automatisch erfüllt, da dort implizite Transaktionen verwendet werden.

Arbeitsweise des Scroll-Satzzeigers

Wenn Sie den Scroll-Satzzeiger mit OPEN öffnen, wird eine temporäre Tabelle für die Ergebnistabelle aufgebaut. Bei jeder FETCH-Anweisung erweitert INFORMIX die temporäre Tabelle bis zum aktuellen Satz, wenn dieser noch nicht in der temporären Tabelle enthalten ist.

Das Aufbauen der temporären Tabelle kann Auswirkungen auf die Ablaufgeschwindigkeit eines Programms haben. Wenn Sie beispielsweise den Satzzeiger auf den letzten Satz der Ergebnistabelle positionieren, dann muß INFORMIX zuvor alle anderen Sätze in die temporäre Tabelle aufnehmen.

Wird ein Scroll-Satzzeiger in der Ergebnistabelle zurückpositioniert (z.B. mit FETCH PREVIOUS), so greift INFORMIX auf den bereits gelesenen Satz in der temporären Tabelle zu und nicht erneut auf die Tabelle in der Datenbank. Wurden inzwischen Änderungen an der Datenbank-Tabelle vorgenommen, so entsprechen sich temporäre Tabelle und Datenbank-Tabelle nicht mehr.
Dieser Fall kann nur ausgeschlossen werden, wenn der Scroll-Satzzeiger ohne WITH HOLD definiert wurde und entweder die Tabelle mit LOCK TABLE exklusiv gesperrt wird oder bei INFORMIX-ONLINE mit der Isolationsstufe Repeatable Read gearbeitet wird (siehe SET ISOLATION).

Ist der Scroll-Satzzeiger mit WITH HOLD definiert, so können nach Ende der Transaktion und Freigabe aller Sperren Inkonsistenzen zwischen temporärer Tabelle und Datenbank-Tabelle auftreten, die auch durch die Isolationsstufe Repeatable Read nicht verhindert werden können.

Erfolgskontrolle bei Programmeinbettung

DECLARE kennt keine Fehlerkontrolle zur Laufzeit. Wird der Satzzeiger falsch vereinbart, erkennt dies schon der Precompiler zum Übersetzungszeitpunkt.

ANSI-Standard

Die Anweisung DECLARE ist im ANSI-Standard enthalten. Die Vereinbarung von Insert-Satzzeigern sowie die Klauseln SCROLL, WITH HOLD und FOR UPDATE sind nicht im ANSI-Standard enthalten.

Beispiel für INFORMIX-4GL

Das erste Beispiel verdeutlicht die Syntax der DECLARE-Anweisung.

```
DEFINE artikelnr SMALLINT
DEFINE hcode CHAR(3)
DEFINE bezeich CHAR(15)
DEFINE preis MONEY(6,2)
DEFINE leinh CHAR(4)
DEFINE stl CHAR(15)

DECLARE szeiger CURSOR FOR
        SELECT * FROM kunde

DECLARE uzeiger CURSOR FOR
        SELECT * FROM kunde WHERE kunden_nr > 100
        FOR UPDATE OF nachname, vorname

DECLARE izeiger CURSOR FOR
        INSERT INTO artikel
        VALUES (artikelnr, hcode, bezeich, preis, leinh, stl)

DECLARE scrollzeiger SCROLL CURSOR FOR
        SELECT * from auftrag
        WHERE kunden_nr < 104
```

Beispiel für INFORMIX-ESQL/C

Das folgende Beispiel zeigt die Verwendung eines Satzzeigers mit WITH HOLD in Kombination mit einem einfachen Satzzeiger. Der Satzzeiger mit WITH HOLD wird dazu benutzt, die Tabelle *auftrag* nach offenen Aufträgen zu durchsuchen. Wird ein offener Auftrag gefunden, so werden die dazugehörigen Posten aus der Tabelle *posten* gesucht, ausgegeben und auf Abfrage der Gesamtpreis um 2,5 % erhöht.

```
$ int auftrnum, pnum, mn, gespreis ;
int aendern;

$ DECLARE hauptzeiger CURSOR WITH HOLD FOR
          SELECT auftrags_nr FROM auftrag WHERE offen = 'j';

$ DECLARE unterzeiger CURSOR FOR
          SELECT posten_nr, menge, gesamtpreis
          FROM posten WHERE auftrags_nr = $auftrnum
          FOR UPDATE;

$ OPEN hauptzeiger;
for (;;)      /* solange es offene Auftraege gibt */
    {
    $ FETCH hauptzeiger INTO $auftrnum;
    $ BEGIN WORK;
    $ OPEN unterzeiger; /* mit aktuellem Wert von $auftrnum */

    for (;;)   /* solange es Posten zu dem offenen Auftrag gibt */
        {
        $ FETCH unterzeiger INTO $pnum, $mn, $gespreis;
        /* Ausgeben auf dem Bildschirm und Abfrage, */
        /* ob geaendert werden soll                 */
        if (aendern)
           $ UPDATE posten SET menge = menge + 2
                    WHERE CURRENT OF unterzeiger;
        }
    $ CLOSE unterzeiger;
    $ COMMIT WORK;
    }  /* Ende Hauptschleife */
$ CLOSE hauptzeiger;
```

>>>> CLOSE, DELETE, FETCH, FLUSH, INSERT, OPEN, PREPARE, PUT, SELECT, UPDATE

DELETE - Sätze löschen

DELETE löscht Sätze aus einer Tabelle.

Vor dem Aufruf beachten

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich entweder das Delete-Zugriffsrecht haben oder aber Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

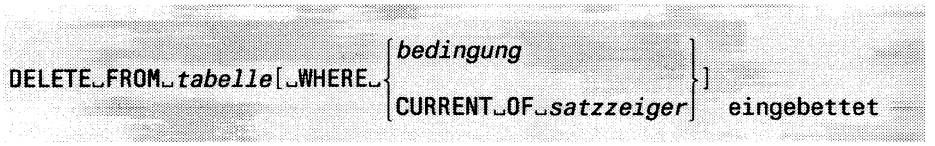


tabelle
Name oder Synonym der Tabelle, aus der Sätze gelöscht werden sollen. Die Tabelle kann eine Basistabelle, eine temporäre Tabelle oder ein änderbarer View (siehe CREATE VIEW) sein.

WHERE-Klausel
Die WHERE-Klausel gibt an, welche Sätze gelöscht werden.

WHERE-Klausel nicht angegeben:
Alle Sätze der Tabelle werden gelöscht.

bedingung
Bedingung, die die zu löschenden Sätze erfüllen müssen. Ein Satz wird nur gelöscht, wenn er die angegebene Bedingung erfüllt. Eine nähere Beschreibung von *bedingung* finden Sie in Kapitel 5.

Einschränkung:
Spaltenangaben in *bedingung* außerhalb von Unterabfragen dürfen sich nur auf die angegebene Tabelle beziehen. Bei Unterabfragen in *bedingung* dürfen Sie in der FROM-Klausel nicht die Tabelle *tabelle* angeben.

CURRENT OF *satzzeiger*

Bei Programmeinbettung kann der zu löschende Satz über den Satzzeiger angegeben werden.
DELETE löscht den aktuellen Satz, auf den der Satzzeiger zeigt. Nach DELETE zeigt der Satzzeiger zwischen den vorhergehenden und den nachfolgenden Satz der Ergebnistabelle. Für eine weitere DELETE-Anweisung müssen Sie den Satzzeiger zuerst wieder mit FETCH auf einen Satz der Ergebnistabelle positionieren.

Voraussetzung:
Der Satzzeiger muß zuvor in derselben Quellprogrammdatei mit DECLARE für eine SELECT-Anweisung mit der Klausel FOR UPDATE und ohne die SCROLL-Klausel vereinbart worden sein. Der Satzzeiger muß mit OPEN geöffnet und mit FETCH auf einen Satz der Ergebnistabelle positioniert worden sein.

Bei einer Datenbank mit Transaktionssicherung müssen Sie DELETE mit WHERE CURRENT OF innerhalb einer Transaktion verwenden. Die FETCH-Anweisung, die den Satz bereitgestellt hat, muß in der gleichen Transaktion durchgeführt worden sein.

Sperren beim Löschen

Bei einer Datenbank ohne Transaktionssicherung wird jeweils nur der eine Satz exklusiv gesperrt, der momentan gelöscht wird.

Wenn Sie auf einer Datenbank mit Transaktionssicherung arbeiten und die DELETE-Anweisung außerhalb einer Transaktionsklammer ausführen, dann werden alle betroffenen Sätze exklusiv gesperrt, bis die DELETE-Anweisung komplett abgearbeitet ist.

Wird die DELETE-Anweisung innerhalb einer Transaktionsklammer ausgeführt, dann werden alle betroffenen Sätze bis zum Ende der Transaktion exklusiv gesperrt.

INFORMIX-ONLINE sperrt beim Löschen eines Satzes zusätzlich den benachbarten Wert im eindeutigen Index. Dies verhindert, daß mit INSERT ein Satz mit dem gleichen Wert in den eindeutigen Index eingefügt wird, bevor die Sperre freigegeben wird.

Ist die Zahl der betroffenen Sätze sehr groß, kann die maximal zulässige Anzahl an Sperren überschritten werden. Die Anzahl der möglichen Sperren ist bei INFORMIX-SE durch das Betriebssystem bzw. bei INFORMIX-ONLINE durch Shared Memory Parameter beschränkt.

Sie sollten in diesem Fall die Anzahl der zu löschenden Sätze reduzieren oder die ganze Tabelle vor der Ausführung der DELETE-Anweisung sperren (siehe LOCK TABLE).

Datenintegrität beim Löschen

Datenbank ohne Transaktionssicherung

Wenn eine DELETE-Anweisung abgebrochen wird, dann ist sie teilweise ausgeführt worden. Die bis zum Abbruch der Anweisung gelöschten Sätze wurden aus der Tabelle entfernt, die restlichen Sätze sind nach wie vor vorhanden.

Beispiel:

Es sollen 100 Sätze mit DELETE gelöscht werden. Beim 57. Satz liegt eine Satzsperre vor, weil dieser Satz gerade von einem anderen Benutzer mit UPDATE bearbeitet wird.
DELETE wird abgebrochen. Vom 1. bis zum 56. Satz sind die zu löschenden Sätze aus der Tabelle entfernt, vom 57. bis zum 100. Satz sind sie noch in der Tabelle vorhanden.

Datenbank mit Transaktionssicherung

Bei einer Nicht-ANSI-Datenbank wird jede DELETE-Anweisung außerhalb einer Transaktion automatisch als einzelne Transaktion behandelt. Eine unvollständig ausgeführte DELETE-Anweisung wird automatisch zurückgesetzt. Unvollständige DELETE-Anweisungen, wie in obigem Beispiel, können also nicht vorkommen.

Bei einer Nicht-ANSI-Datenbank innerhalb einer Transaktion und bei einer ANSI-Datenbank müssen Sie die Transaktion, in der die DELETE-Anweisung ausgeführt wird, explizit mit ROLLBACK WORK bzw. COMMIT WORK beenden.

Erfolgskontrolle bei Programmeinbettung

	Positivfall	Fehlerfall
INFORMIX-4GL	status = 0 SQLCA.SQLERRD[3] ist die Anzahl der mit DELETE gelöschten Sätze.	status < 0 SQLCA.SQLERRD[3] ist die Anzahl der mit DELETE erfolgreich gelöschten Sätze. Der Rest der Sätze ist nicht gelöscht.
INFORMIX-ESQL/C	sqlca.sqlcode = 0 sqlca.sqlerrd[2] ist die Anzahl der mit DELETE gelöschten Sätze.	sqlca.sqlcode < 0 sqlca.sqlerrd[2] ist die Anzahl der mit DELETE erfolgreich gelöschten Sätze. Der Rest der Sätze ist nicht gelöscht.
INFORMIX-ESQL/COBOL	SQLCODE OF SQLCA = 0 SQLERRD[3] ist die Anzahl der mit DELETE gelöschten Sätze.	SQLCODE OF SQLCA < 0 SQLERRD[3] ist die Anzahl der mit DELETE erfolgreich gelöschten Sätze. Der Rest der Sätze ist nicht gelöscht.

ANSI-Standard

Die Anweisung DELETE ist im ANSI-Standard enthalten.

Beispiel 1

Im folgenden Beispiel werden aus der Tabelle *posten* alle Posten mit der Auftragsnummer 1013 gelöscht.

```
SELECT * FROM posten WHERE auftrags_nr = 1013;
```

```
posten_nr auftrags_nr artikel_nr herstellercode  menge gesamtpreis

        1       1013          5 ANZ                 1       19.80
        2       1013          6 SMT                 1       36.00
        3       1013          6 ANZ                 1       48.00
        4       1013          9 ANZ                 2       40.00
```

```
DELETE FROM posten WHERE auftrags_nr = 1013;

SELECT * FROM posten WHERE auftrags_nr = 1013
```

```
posten_nr auftrags_nr artikel_nr herstellercode  menge gesamtpreis
```

Beispiel 2 für INFORMIX-4GL

Im folgenden Beispiel werden alle Posten gelöscht, die zum Auftrag gehören, dessen Auftragsnummer in der Variablen *anum* gespeichert ist.

```
DEFINE anum INTEGER
LET anum= 1013

DELETE FROM posten WHERE auftrags_nr = anum
```

Beispiel 3 für INFORMIX-4GL

Im folgenden Beispiel wird aus der Tabelle *auftrag* der Satz gelöscht, auf den der Satzzeiger *zn* zeigt.

```
DEFINE kunde LIKE auftrag.kunden_nr

DECLARE zn CURSOR FOR
          SELECT kunden_nr FROM auftrag WHERE auftrags_nr > 1010
          FOR UPDATE
OPEN zn
BEGIN WORK
FETCH zn INTO kunde
DELETE FROM auftrag WHERE CURRENT OF zn
COMMIT WORK

CLOSE zn
```

>>>> DECLARE, INSERT, UPDATE

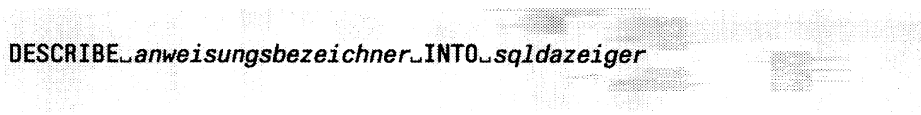
DESCRIBE - Dynamische Anweisung analysieren

Mit der Anweisung DESCRIBE wird eine mit PREPARE vorbereitete dynamische Anweisung analysiert. DESCRIBE erkennt welche SQL-Anweisung mit PREPARE vorbereitet wurde. Bei einer SELECT-Anweisung wird zusätzlich die Struktur der Sätze der Ergebnistabelle analysiert.

Dadurch ist es möglich, dynamische Anweisungen zu bearbeiten, auch wenn zum Übersetzungszeitpunkt weder die Art der Anweisung noch die Anzahl oder der Datentyp der mit SELECT ausgewählten Spalten bekannt ist.

Vor dem Aufruf beachten

Die DESCRIBE-Anweisung können Sie nur bei der Einbettung in C-Programme verwenden.



anweisungsbezeichner
Bezeichner für die dynamischen Anweisung. *anweisungsbezeichner* wurde mit PREPARE vereinbart.

sqldazeiger
Zeiger auf eine *sqlda*-Struktur. Der Zeiger muß als Hostvariable vereinbart sein. Die *sqlda*-Struktur selbst wird von INFORMIX automatisch erzeugt. Genauere Informationen zur *sqlda*-Struktur finden Sie im Handbuch für die C-Einbettung [3].

Erfolgskontrolle bei Programmeinbettung

Positivfall	Fehlerfall
sqlca.sqlcode = 0 Es wurde eine SELECT-Anweisung (ohne INTO TEMP) analysiert. sqlca.sqlcode > 0 kodiert die Art der Anweisung nach den Konstanten, die in der Header-Datei **sqlstype.h** definiert sind. Folgende Konstanten stehen für die einzelnen Anweisungen: Konstante — SQL-Anweisung SQ_DATABASE — DATABASE SQ_SELECT — SELECT (nicht in Gebrauch) SQ_SELINTO — SELECT INTO TEMP SQ_UPDATE — UPDATE SQ_DELETE — DELETE SQ_INSERT — INSERT SQ_UPDCURR — UPDATE (mit Satzzeiger) SQ_DELCURR — DELETE (mit Satzzeiger) SQ_LDINSERT — FLUSH (nicht in Gebrauch) SQ_LOCK — LOCK TABLE SQ_UNLOCK — UNLOCK TABLE SQ_CREADB — CREATE DATABASE SQ_DROPDB — DROP DATABASE SQ_CRETAB — CREATE TABLE SQ_DRPTAB — DROP TABLE SQ_CREIDX — CREATE INDEX SQ_DRPIDX — DROP INDEX SQ_GRANT — GRANT SQ_REVOKE — REVOKE SQ_RENTAB — RENAME TABLE SQ_RENCOL — RENAME COLUMN SQ_CREAUD — CREATE AUDIT SQ_STRAUD — START AUDIT (nicht in Gebr.)	sqlca.sqlcode < 0

Positivfall		Fehlerfall
SQ_STPAUD	STOP AUDIT (nicht in Gebr.)	
SQ_DRPAUD	DROP AUDIT	
SQ_RECTAB	RECOVER TABLE	
SQ_CHKTAB	CHECK TABLE (nicht in Gebr.)	
SQ_REPTAB	REPAIR TABLE (nicht in Gebr.)	
SQ_ALTER	ALTER TABLE	
SQ_STATS	UPDATE STATISTICS	
SQ_CLSDB	CLOSE DATABASE	
SQ_DELALL	DELETE (ohne WHERE-Klausel)	
SQ_UPDALL	UPDATE (ohne WHERE-Klausel)	
SQ_BEGWORK	BEGIN WORK	
SQ_COMMIT	COMMIT WORK	
SQ_ROLLBACK	ROLLBACK WORK	
SQ_SAVEPOINT	nicht in Gebrauch	
SQ_STARTDB	START DATABASE	
SQ_RFORWARD	ROLLFORWARD DATABASE	
SQ_CREVIEW	CREATE VIEW	
SQ_DROPVIEW	DROP VIEW	
SQ_DEBUG	nicht in Gebrauch	
SQ_CREASYN	CREATE SYNONYM	
SQ_DROPSYN	DROP SYNONYM	
SQ_CTEMP	CREATE TEMP TABLE	
SQ_WAITFOR	SET LOCK MODE	
SQ_ALTIDX	ALTER INDEX	
SQ_ISOLATE	SET ISOLATION	
SQ_SETLOG	SET LOG	
SQ_EXPLAIN	SET EXPLAIN	
SQ_SCHEMA	CREATE SCHEMA	

ANSI-Standard

Die Anweisung DESCRIBE ist im ANSI-Standard enthalten.

Beispiel

Beispiele für die Anwendung von DESCRIBE finden Sie im Handbuch für die C-Einbettung [3].

> > > > EXECUTE, OPEN, PREPARE

DROP AUDIT - Audit-Protokoll löschen

DROP AUDIT beendet die Protokollierung der Tabellenänderungen, die Sie zuvor mit CREATE AUDIT begonnen haben und löscht die zugehörige Audit-Datei.

Vor dem Aufruf beachten

DROP AUDIT darf nur für eine INFORMIX-SE-Datenbank verwendet werden.

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

DROP AUDIT sperrt die Tabelle exklusiv gegen jeglichen Zugriff anderer Prozesse. DROP AUDIT wird abgewiesen, wenn ein anderer Prozeß die Tabelle bearbeitet. Dies gilt auch dann, wenn nur lesend zugegriffen wird.

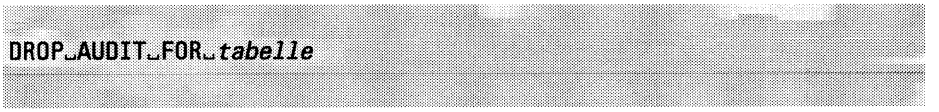


tabelle
Name oder Synonym der Basistabelle, deren Audit-Protokoll gelöscht werden soll.

DROP AUDIT in Transaktionen

Die Anweisung DROP AUDIT kann nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

ANSI-Standard

Die Anweisung DROP AUDIT ist nicht im ANSI-Standard enthalten.

> > > > CREATE AUDIT, RECOVER TABLE

DROP DATABASE - Datenbank löschen

DROP DATABASE löscht eine Datenbank. Alle Tabellen, Sätze und Indizes werden gelöscht.

Vor dem Aufruf beachten

Sie müssen das DBA-Zugriffsrecht für die Datenbank besitzen.

Die Datenbank muß geschlossen sein. Wenn sie noch geöffnet ist, dann müssen Sie diese zuerst mit CLOSE DATABASE schließen.

DROP DATABASE sperrt die Datenbank exklusiv gegen jeglichen Zugriff anderer Prozesse. DROP DATABASE wird abgewiesen, wenn ein anderer Prozeß die Datenbank bearbeitet. Dies gilt auch dann, wenn nur lesend zugegriffen wird.

```
DROP␣DATABASE␣datenbank
```

datenbank
Name der Datenbank, die gelöscht werden soll.

Wird die Anweisung DROP DATABASE in eine Programmiersprache eingebettet, so kann für *datenbank* auch eine alphanumerische Hostvariable benutzt werden.

Auswirkung von DROP DATABASE bei INFORMIX-SE

DROP DATABASE löscht die angegebene Datenbank und alle zu ihr gehörenden Dateien. Die Dateien stehen in dem Dateiverzeichnis *datenbank.dbs* oder in dem Dateiverzeichnis, das beim Erstellen einer Tabelle dieser Datenbank bei CREATE TABLE angegeben wurde. Soweit vorhanden, löscht DROP DATABASE auch Transaktions- und Audit-Protokolle. Beachten Sie, daß Dateien, deren Name nicht auf *.dat* oder *.idx* endet, nicht gelöscht werden. Wenn das Datenbank-Dateiverzeichnis *datenbank.dbs* solche Dateien enthält, dann führt DROP DATABASE zu einer Fehlermeldung, weil DROP DATABASE zum Löschen das SINIX-Kommando *rmdir* benutzt. *rmdir* ist nur erfolgreich, wenn das zu löschende Dateiverzeichnis leer ist.

DROP DATABASE in Transaktionen

DROP DATABASE kann nicht mit ROLLBACK WORK zurückgesetzt werden.

ANSI-Standard

Die Anweisung DROP DATABASE ist nicht im ANSI-Standard enthalten.

Beispiel

Die folgende Anweisung DROP DATABASE löscht die Datenbank *neudb*. Alle Daten und Indizes werden gelöscht.

```
DROP DATABASE neudb
```

>>>> CLOSE DATABASE, CREATE DATABASE

DROP INDEX - Index löschen

DROP INDEX löscht einen Index.

Vor dem Aufruf beachten

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich Indexeigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

DROP INDEX sperrt die Tabelle exklusiv gegen jeglichen Zugriff anderer Prozesse. DROP INDEX wird abgewiesen, wenn ein anderer Prozeß die Tabelle bearbeitet. Dies gilt auch dann, wenn nur lesend zugegriffen wird.

index
Name des Index, der gelöscht werden soll. Der Index einer Systemtabelle oder ein Index, der in einem Bildschirmformat für einen Join benötigt wird, darf nicht gelöscht werden.

Achtung:
Mit DROP INDEX können Sie keinen Constraint löschen. Dies ist nur mit der DROP CONSTRAINT-Klausel bei der Anweisung ALTER TABLE möglich.

DROP INDEX in Transaktionen

Die Anweisung DROP INDEX kann bei einer INFORMIX-SE-Datenbank nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

INFORMIX-ONLINE kann die Anweisung DROP INDEX zurücksetzen.

ANSI-Standard

Die Anweisung DROP INDEX ist nicht im ANSI-Standard enthalten.

Beispiel

Im folgenden Beispiel werden zuerst mit der INFO-Anweisung die definierten Indizes ausgegeben, dann wird mit DROP INDEX ein Index gelöscht.

```
INFO INDEXES FOR auftrag
```

```
Indexname           Eigner      Typ        Cluster  Spalten

a_nr_ix             lomata      Eindeutig  Nein     auftrags_nr

a_knr_ix            lomata      Duplikate  Nein     kunden_nr
```

```
DROP INDEX a_nr_ix;

INFO INDEXES FOR auftrag
```

```
Indexname           Eigner      Typ        Cluster  Spalten

a_knr_ix            lomata      Duplikate  Nein     kunden_nr
```

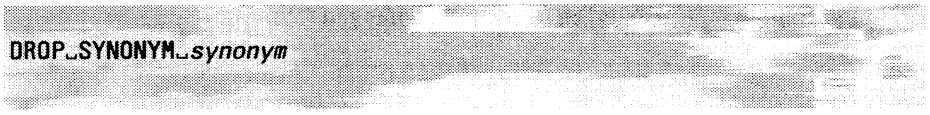
> > > > ALTER INDEX, CREATE INDEX

DROP SYNONYM - Synonym löschen

DROP SYNONYM löscht ein Synonym einer Basistabelle oder eines Views.

Vor dem Aufruf beachten

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich Synonymeigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.



synonym
Name des Synonyms, das gelöscht werden soll.

Auswirkung von DROP SYNONYM

Wurde das Synonym in bereits compilierten Programmen verwendet, so hat DROP SYNONYM keine Auswirkung darauf, da bei der Compilierung der Synonymname durch den Tabellennamen ersetzt wurde.
Wird das Synonym dagegeben in dynamisch formulierten Anweisungen oder in Shell-Prozeduren verwendet oder soll das Quellprogramm neu übersetzt werden, so müssen diese zuerst angepaßt werden.

Wurde das Synonym in der Definition eines Views verwendet, müssen Sie den View neu definieren.

DROP SYNONYM in Transaktionen

Die Anweisung DROP SYNONYM kann bei einer INFORMIX-SE-Datenbank nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

INFORMIX-ONLINE kann die Anweisung DROP SYNONYM zurücksetzen.

ANSI-Standard

Die Anweisung DROP SYNONYM ist nicht im ANSI-Standard enthalten.

Beispiel

Im folgenden Beispiel werden zuerst die Namen der Synonyme aus der Systemtabelle *systables* abgefragt, anschließend wird ein Synonym gelöscht.

```
SELECT tabname FROM systables WHERE tabtype="S"
```

```
tabname

h
sykun
```

```
DROP SYNONYM h
```

> > > > CREATE SYNONYM

DROP TABLE - Tabelle löschen

DROP TABLE löscht eine Basistabelle. Damit werden automatisch auch gelöscht:

- Alle in der Tabelle enthaltenen Sätze
- Synonyme für die Tabelle
- Indizes, die auf der Tabelle definiert wurden
- Zugriffsrechte, die für die Tabelle vergeben wurden
- Views, die auf der Tabelle definiert wurden
- Audit-Protokolle zu der Tabelle
- Bei INFORMIX-SE die zugehörigen von INFORMIX erzeugten Dateien und Transaktionsprotokolle

Vor dem Aufruf beachten

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

DROP TABLE sperrt die Tabelle exklusiv gegen jeglichen Zugriff anderer Prozesse. DROP TABLE wird abgewiesen, wenn ein anderer Prozeß die Tabelle bearbeitet. Dies gilt auch dann, wenn nur lesend zugegriffen wird.

```
DROP␣TABLE␣tabelle
```

tabelle
Name oder Synonym der Tabelle, die gelöscht werden soll. Die Tabelle kann eine Basistabelle oder eine temporäre Tabelle sein. Sie dürfen keine Systemtabelle angeben.

DROP TABLE in Transaktionen

Die Anweisung DROP TABLE kann bei einer INFORMIX-SE-Datenbank nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

INFORMIX-ONLINE kann die Anweisung DROP TABLE zurücksetzen.

ANSI-Standard

Die Anweisung DROP TABLE ist nicht im ANSI-Standard enthalten.

Beispiel

Die folgende DROP TABLE-Anweisung löscht die Tabelle *auftrag* der Beispieldatenbank *versand*.

```
DROP TABLE auftrag
```

> > > > ALTER TABLE, CREATE TABLE

DROP VIEW - View löschen

DROP VIEW löscht einen View. Damit werden automatisch auch gelöscht:

- Synonyme für den View
- Zugriffsrechte, die für den View vergeben wurden
- Views, die auf diesem View definiert sind.

Vor dem Aufruf beachten

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich Vieweigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

view
Name oder Synonym des View, der gelöscht werden soll.

DROP VIEW in Transaktionen

Die Anweisung DROP VIEW kann bei einer INFORMIX-SE-Datenbank nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

INFORMIX-ONLINE kann die Anweisung DROP VIEW zurücksetzen.

ANSI-Standard

Die Anweisung DROP VIEW ist nicht im ANSI-Standard enthalten.

Beispiel

Im folgenden Beispiel werden zuerst die Namen der Views aus der Systemtabelle *systables* abgefragt, anschließend wird ein View gelöscht.

```
SELECT tabname FROM systables WHERE tabtype="V"
```

```
tabname

muenchen
vtest
```

```
DROP VIEW muenchen
```

> > > > CREATE VIEW, DROP TABLE

EXECUTE - Dynamische Anweisung ausführen

Mit der Anweisung EXECUTE wird eine mit PREPARE vorbereitete Anweisung ausgeführt. Vorbereitete SELECT-Anweisungen können nur dann mit EXECUTE ausgeführt werden, wenn sie die Klausel INTO TEMP enthalten, sonst werden dazu die Anweisungen DECLARE, OPEN, FETCH und CLOSE bzw. die 4GL-Anweisung FOREACH verwendet.

Eine EXECUTE-Anweisung kann beliebig oft für eine mit PREPARE vorbereitete Anweisung zur Ausführung kommen. Die Betriebsmittel, die die vorbereitete Anweisung belegt, können mit der FREE-Anweisung wieder freigegeben werden.

Vor dem Aufruf beachten

Anweisungsbezeichner sind nur in der Datei ansprechbar, in der sie mit PREPARE vereinbart werden.

anweisungsbezeichner
Bezeichner für die dynamische Anweisung. *anweisungsbezeichner* wurde mit PREPARE vereinbart.

USING
Die USING-Klausel dürfen Sie nur angeben, wenn die dynamische Anweisung Fragezeichen als Platzhalter für Werte enthält.

USING nicht angegeben:
Die dynamische Anweisung enhält keine Fragezeichen als Platzhalter für Werte.

variable
Hostvariable, deren Wert einem Fragezeichen der dynamischen Anweisung zugewiesen wird. Der Datentyp der Hostvariablen muß zum entsprechenden Wert passen, für den das Fragezeichen als Platzhalter in der Anweisung steht. Die Werte der Hostvariablen werden in der aufgeführten Reihenfolge den Fragezeichen der dynamischen Anweisung zugewiesen. Die Anzahl der angegebenen Hostvariablen muß dabei der Anzahl der Fragezeichen entsprechen. Die Anzahl und der Datentyp der Platzhalter in der dynamischen Anweisung muß bereits zum Übersetzungszeitpunkt bekannt sein.

Wenn es die vorbereitete Anweisung erlaubt, können den Hostvariablen Indikatorvariablen zugeordnet werden. Bindet man an eine Hostvariable eine negative Indikatorvariable, so stellt die Hostvariable einen NULL-Wert dar. Dies ist z.B sinnvoll beim Einfügen von Sätzen mit INSERT oder beim Ändern mit UPDATE.

DESCRIPTOR *sqldazeiger*
Die Werte für die Fragezeichen der dynamischen Anweisung werden durch eine *sqlda*-Struktur angegeben.

Achtung:
Die USING DESCRIPTOR-Klausel können Sie nur bei der Einbettung in C-Programme verwenden.

sqldazeiger
Zeiger auf eine *sqlda*-Struktur, die auf die Werte zeigt, die den Fragezeichen der dynamischen Anweisung zugewiesen werden. Genauere Informationen zur *sqlda*-Struktur finden Sie im Handbuch für die C-Einbettung [3].

ANSI-Standard

Die Anweisung EXECUTE ist im ANSI-Standard enthalten.

Beispiel für INFORMIX-ESQL/C

Im folgenden Beispiel wird die dynamische UPDATE-Anweisung in der Hostvariablen *aend* übergeben und mit PREPARE vorbereitet. Die aktuellen Werte für die UPDATE-Anweisung werden mit EXECUTE ... USING übergeben.

```
sprintf(aend, "%s",
       "UPDATE artikel SET preis = ? WHERE bezeichnung = ?");

$ PREPARE aend_bez FROM $aend;
$ EXECUTE aend_bez USING $neupreis, $bez;
```

>>>> DECLARE, EXECUTE IMMEDIATE, FREE, OPEN, PREPARE

EXECUTE IMMEDIATE - Dynamische Anweisung sofort ausführen

Mit der Anweisung EXECUTE IMMEDIATE wird eine dynamische Anweisung in einem Schritt vorbereitet, ausgeführt und ihre Betriebsmittel wieder freigegeben.
EXECUTE IMMEDIATE kann also PREPARE, EXECUTE und FREE ersetzen.

Vor dem Aufruf beachten

Die Anweisung EXECUTE IMMEDIATE gibt es nur bei INFORMIX-ESQL/C.

anweisung
Genau eine SQL-Anweisung, die dynamisch ausgeführt wird. *anweisung* kann eine in " oder ' eingeschlosse alphanumerische Konstante sein oder eine Hostvariable für alphanumerische Werte.

Innerhalb von *anweisung* dürfen keine Hostvariablen und keine Fragezeichen als Platzhalter verwendet werden.

SQL-Anweisungen für EXECUTE IMMEDIATE

Folgende Anweisungen können bei EXECUTE IMMEDIATE verwendet werden:

ALTER INDEX
ALTER TABLE
BEGIN WORK
CLOSE DATABASE
COMMIT WORK
CREATE AUDIT
CREATE DATABASE
CREATE INDEX
CREATE SYNONYM
CREATE TABLE
CREATE VIEW
DATABASE
DELETE
DROP VIEW
GRANT
INSERT
LOCK TABLE
RECOVER TABLE
RENAME COLUMN
RENAME TABLE
REVOKE
ROLLBACK WORK
ROLLFORWARD DATABASE
SELECT INTO TEMP
SET ISOLATION
SET LOCK MODE

DROP AUDIT	SET LOG
DROP DATABASE	START DATABASE
DROP INDEX	UNLOCK TABLE
DROP SYNONYM	UPDATE
DROP TABLE	UPDATE STATISTICS

DELETE, INSERT und UPDATE dürfen eine SELECT-Anweisung als Unterabfrage enthalten.

Folgende Anweisungen können nicht bei EXECUTE IMMEDIATE verwendet werden:

CLOSE	FREE
DECLARE	OPEN
DESCRIBE	PREPARE
EXECUTE	PUT
EXECUTE IMMEDIATE	SELECT ohne INTO TEMP
FETCH	WHENEVER
FLUSH	

ANSI-Standard

Die Anweisung EXECUTE IMMEDIATE ist nicht im ANSI-Standard enthalten.

Beispiel für INFORMIX-ESQL/C

Im folgenden Beispiel wird eine dynamische Anweisung als Eingabe abgefragt und mit der Hostvariablen *anw1* an die Anweisung EXECUTE IMMEDIATE übergeben und ausgeführt.

```
char anw1[50];

printf("Anweisung eingeben");
scanf("%s",anw1);
$ EXECUTE IMMEDIATE anw1;
```

> > > > EXECUTE, FREE, PREPARE

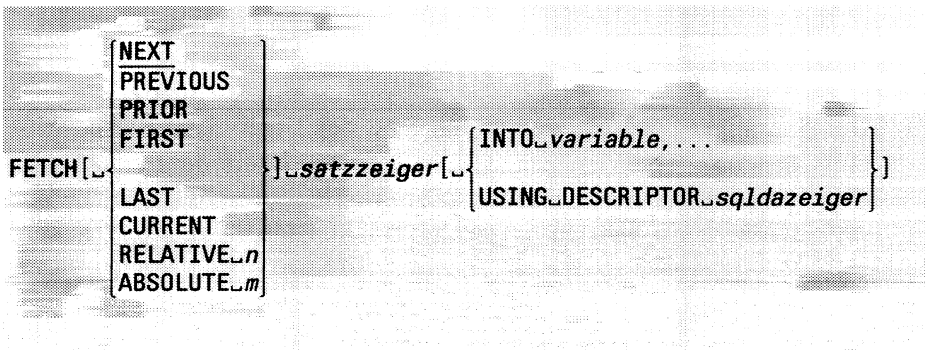
FETCH - Satzzeiger positionieren

FETCH positioniert einen Satzzeiger auf einen Satz in einer Ergebnistabelle und macht diesen zum aktuellen Satz. Die Werte des aktuellen Satzes können Hostvariablen oder einer *sqlda*-Struktur zugewiesen werden. Der aktuelle Satz kann von einer nachfolgenden UPDATE- oder DELETE-Anweisung geändert oder gelöscht werden.
Die 1. FETCH-Anweisung nach dem OPEN führt zuerst die SELECT-Anweisung aus, bestimmt die Ergebnistabelle und positioniert den Satzzeiger auf den ersten Satz.

Vor dem Aufruf beachten

Satzzeiger sind nur in der Quellprogrammdatei ansprechbar, in der sie mit DECLARE vereinbart werden. Der mit FETCH angesprochene Satzzeiger muß zuvor mit DECLARE für eine SELECT-Anweisung vereinbart und mit OPEN geöffnet werden.

Man kann einen Satzzeiger nur frei auf die Sätze der Ergebnistabelle positionieren, wenn er mit SCROLL vereinbart ist. Andernfalls erhält man mit jedem FETCH nur den nächsten Satz (siehe NEXT-Klausel).



NEXT
: Der Satzzeiger wird auf den nächsten Satz der Ergebnistabelle positioniert. Wenn Sie einen Satzzeiger ohne SCROLL vereinbart haben, dann können Sie nur NEXT verwenden.
NEXT ist Standard.

PREVIOUS
: Der Satzzeiger wird auf den vorhergehenden Satz der Ergebnistabelle positioniert. Sie dürfen PREVIOUS nur verwenden, wenn Sie den Satzzeiger mit SCROLL vereinbart haben.

PRIOR
ist gleichbedeutend mit PREVIOUS.

FIRST
Der Satzzeiger wird auf den ersten Satz der Ergebnistabelle positioniert. Sie dürfen FIRST nur verwenden, wenn Sie den Satzzeiger mit SCROLL vereinbart haben.

LAST
Der Satzzeiger wird auf den letzten Satz der Ergebnistabelle positioniert. Sie dürfen LAST nur verwenden, wenn Sie den Satzzeiger mit SCROLL vereinbart haben.

CURRENT
Der Satzzeiger wird auf den aktuellen Satz der Ergebnistabelle positioniert. Sie dürfen CURRENT nur verwenden, wenn Sie den Satzzeiger mit SCROLL vereinbart haben.

RELATIVE *n*
Der Satzzeiger wird um *n* Sätze von seiner aktuellen Position verschoben. Dabei bewirkt ein negatives *n* eine Bewegung vor den aktuellen Satz, ein positives *n* hinter diesen. *n* kann dabei eine ganzzahlige Konstante sein oder eine Variable, die eine ganzzahlige Konstante liefert. Sie dürfen RELATIVE nur verwenden, wenn Sie den Satzzeiger mit SCROLL vereinbart haben.

ABSOLUTE *m*
Der Satzzeiger wird auf den *m*-ten Satz der Ergebnistabelle positioniert. *m* kann dabei eine ganzzahlige Konstante sein oder eine Variable, die eine ganzzahlige Konstante liefert. Sie dürfen ABSOLUTE nur verwenden, wenn Sie den Satzzeiger mit SCROLL vereinbart haben.

satzzeiger
Name des Select-Satzzeigers.

INTO- und USING-Klausel:
Sie dürfen die INTO- und USING-Klausel nicht verwenden, wenn Sie bereits bei der DECLARE-Anweisung SELECT INTO *variable,...* angegeben haben.
Sie müssen entweder die INTO- oder die USING-Klausel verwenden, wenn Sie in der SELECT-Anweisung keine Variablen mit INTO angegeben haben, um die Spaltenwerte aufzunehmen.

INTO *variable,...*

Hostvariable, in die die Spaltenwerte des aktuellen Satzes geschrieben werden. Die Anzahl und der Datentyp der Hostvariablen muß zur Spaltenauswahl in der SELECT-Anweisung passen.

Wenn eine der Hostvariablen ein Array-Element ist, dann darf INTO *variable,..* nicht in der SELECT-Anweisung stehen. Sie müssen dann INTO *variable,...* hier bei der FETCH-Anweisung angeben.

USING DESCRIPTOR *sqldazeiger*

Der Speicherplatz für den aktuellen Satz wird durch eine *sqlda*-Struktur angegeben.

Achtung:
Die USING DESCRIPTOR-Klausel können Sie nur bei der Einbettung in C-Programme verwenden.

sqldazeiger

Zeiger auf eine *sqlda*-Struktur, die auf den Speicherplatz zeigt, der die Spaltenwerte des aktuellen Satzes aufnimmt. Genauere Informationen zur *sqlda*-Struktur finden Sie im Handbuch für die C-Einbettung [3].

Indikatorvariable und NULL-Werte

Um festzustellen, ob einer Hostvariablen NULL-Werte zugewiesen wurden bzw. ob eine alphanumerische Hostvariable zu klein war, um einen Spaltenwert aus der Datenbank zu übernehmen, können Sie Indikatorvariable an die entsprechenden Hostvariable binden. Genauere Informationen zu Indikatorvariablen und NULL-Werten finden Sie in Kapitel 2 und 4 und im Handbuch für die jeweilige Programmeinbettung [3, 4, 6].

Von FETCH gesetzte Sperren

Bei einer INFORMIX-SE-Datenbank setzt FETCH nur dann eine Sperre auf den aktuellen Satz, wenn die zuvor mit DECLARE dem Satzzeiger zugewiesene SELECT-Anweisung eine FOR UPDATE-Klausel enthält (siehe DECLARE). Die gesetzte Sperre ist eine Exclusive-Sperre.

Bei einer INFORMIX-ONLINE-Datenbank hängt das Setzen einer Share-Sperre vom der eingestellten Isolationsstufe ab (siehe Anweisung SET ISOLATION).
Wurde der Satzzeiger mit FOR UPDATE definiert, so setzt FETCH eine Update-Sperre auf den aktuellen Satz. Wird der aktuelle Satz mit DELETE/UPDATE WHERE CURRENT OF geändert, so wird die Update-Sperre in eine Exclusive-Sperre umgewandelt (siehe auch Kapitel 2, Abschnitt 2.11 *Sperren*).

Erfolgskontrolle bei Programmeinbettung

	Positivfall		Fehlerfall
INFORMIX-4GL	status = 0	status = SQLNOTFOUND	status < 0
INFORMIX-ESQL/C	sqlca.sqlcode = 0	sqlca.sqlcode =SQLNOTFOUND	sqlca.sqlcode < 0
INFORMIX-ESQL/COBOL	SQLCODE OF SQLCA = 0	SQLCODE OF SQLCA = SQLNOTFOUND	SQLCODE OF SQLCA < 0
	Der Satzzeiger wurde positioniert und der Satz gelesen.	Der Satzzeiger wurde nicht positioniert, weil es entweder keinen Satz gibt oder weil der Satzzeiger über die Ergebnistabelle hinaus bewegt werden sollte.	

ANSI-Standard

Die Anweisung FETCH ist im ANSI-Standard enthalten.

Beispiel 1 für INFORMIX-4GL

Beim ersten Beispiel werden die Hostvariablen, die die Spaltenwerte aufnehmen, direkt in der SELECT-Anweisung angegeben. Jede FETCH-Anweisung verwendet diese Hostvariablen.

```
DEFINE hcode_var CHAR(3)
DEFINE hname_var CHAR(15)

DECLARE selzeiger SCROLL CURSOR FOR
        SELECT herstellercode, herstellername
               INTO hcode_var, hname_var
               FROM hersteller

OPEN selzeiger

FETCH selzeiger

FETCH LAST selzeiger

CLOSE selzeiger
```

Beispiel 2 für INFORMIX-4GL

Beim folgenden Beispiel werden die Hostvariablen, die die Spaltenwerte aufnehmen, beim FETCH angegeben. Dadurch können bei jedem FETCH andere Hostvariablen angegeben werden.

```
DEFINE hcode_var1, hcode_var2 CHAR(3)
DEFINE hname_var1, hname_var2 CHAR(15)

DECLARE selzeiger SCROLL CURSOR FOR
        SELECT herstellercode, herstellername
               FROM hersteller

OPEN selzeiger

FETCH selzeiger INTO hcode_var1, hname_var1

FETCH LAST selzeiger INTO hcode_var2, hname_var2

CLOSE selzeiger
```

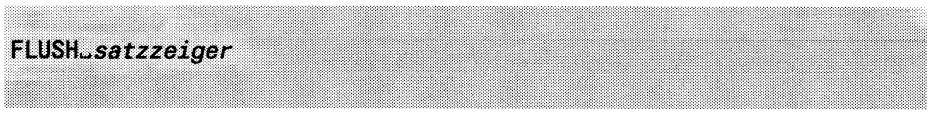
> > > > CLOSE, DECLARE, DELETE, OPEN, UPDATE

FLUSH - Pufferinhalt in Datenbank einfügen

FLUSH leert den Puffer, der zu einem Insert-Satzzeiger gehört. Im Gegensatz zur CLOSE-Anweisung, die ebenfalls den Puffer in die Datenbank schreibt, bleibt bei FLUSH der Insert-Satzzeiger geöffnet.

Vor dem Aufruf beachten

Satzzeiger sind nur in der Quellprogrammdatei ansprechbar, in der sie mit DECLARE vereinbart werden. Der mit FLUSH angesprochene Satzzeiger muß zuvor mit DECLARE für eine INSERT-Anweisung vereinbart und mit OPEN geöffnet werden.



satzzeiger
Name des Satzzeigers, dessen Puffer Sie leeren wollen.

Leeren des Puffers

Der Puffer wird nicht nur mit der FLUSH-Anweisung geleert. Trifft eine PUT-Anweisung auf einen vollen Puffer, so wird der Pufferinhalt in die Datenbank eingefügt (implizites FLUSH). Im Puffer steht dann der mit PUT zugewiesene Satz.

Die CLOSE-Anweisung leert ebenfalls den Puffer, wobei der Insert-Satzzeiger aber dann geschlossen ist und vor einem erneuten Öffnen des Satzzeigers nicht mehr verwendet werden kann.

Werden mit der PUT-Anweisung nur konstante Sätze in den Puffer geschrieben, so zählt INFORMIX nur die Zahl der "gleichen" Sätze und fügt sie nicht in den Puffer ein. Ohne ein explizites FLUSH oder CLOSE werden die Sätze nie in die Datenbank eingefügt.
Wird das Programm beendet, ohne daß vorher der Satzzeiger mit CLOSE geschlossen oder der Puffer mit FLUSH in die Datenbank geschrieben wurde, geht der Inhalt des Puffers verloren.

Es gibt keine Variable, in der INFORMIX alle aus dem Insert-Puffer eingefügten Sätze zählt. Daher sollten Sie, wenn Sie Information über die Anzahl der tatsächlich eingefügten Sätze haben wollen, selbst eine entsprechende Zählvariable vereinbaren und für jede PUT-Anweisung erhöhen.

Erfolgskontrolle bei Programmeinbettung

	Positivfall	Fehlerfall
INFORMIX-4GL	status = 0 SQLCA.SQLERRD[3] ist die Anzahl der mit FLUSH eingefügten Sätze.	status < 0 SQLCA.SQLERRD[3] ist die Anzahl der mit FLUSH erfolgreich eingefügten Sätze. Der Rest der Sätze ist verloren.
INFORMIX-ESQL/C	sqlca.sqlcode = 0 sqlca.sqlerrd[2] ist die Anzahl der mit FLUSH eingefügten Sätze.	sqlca.sqlcode < 0 sqlca.sqlerrd[2] ist die Anzahl der mit FLUSH erfolgreich eingefügten Sätze. Der Rest der Sätze ist verloren.
INFORMIX-ESQL/COBOL	SQLCODE OF SQLCA = 0 SQLERRD[3] ist die Anzahl der mit FLUSH eingefügten Sätze.	SQLCODE OF SQLCA < 0 SQLERRD[3] ist die Anzahl der mit FLUSH erfolgreich eingefügten Sätze. Der Rest der Sätze ist verloren.

ANSI-Standard

Die Anweisung FLUSH ist nicht im ANSI-Standard enthalten.

Beispiel für INFORMIX-4GL

Im folgenden Beispiel wird eine INSERT-Anweisung mit PREPARE vorbereitet, mit DECLARE ein Satzzeiger dafür definiert und mit OPEN geöffnet. Nach einigen PUT-Anweisungen wird der Satzzeiger mit FLUSH in die Datenbank geschrieben. Danach können weitere PUT-Anweisungen für den gleichen Satzzeiger erfolgen.

```
DEFINE her_code1, her_code2, ... CHAR(3)
DEFINE her_name1, her_name2, ... CHAR(15)

PREPARE ins_anw FROM
        "INSERT INTO hersteller VALUES (?,?)"

DECLARE inszeiger CURSOR FOR ins_anw

OPEN inszeiger

PUT inszeiger FROM her_code1, her_name1
PUT inszeiger FROM her_code2, her_name2
  .
  .
FLUSH inszeiger

PUT inszeiger FROM her_code5, her_name5
PUT inszeiger FROM her_code6, her_name6
  .
  .
CLOSE inszeiger
```

>>>> CLOSE, DECLARE, OPEN, PUT

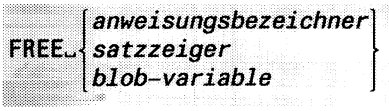
FREE - Betriebsmittel freigeben

FREE gibt Betriebsmittel des Datenbankservers frei, die für dynamische Anweisungen (mit PREPARE) oder für Satzzeiger (mit OPEN) belegt wurden.

Bei INFORMIX-4GL unter INFORMIX-ONLINE können Sie mit FREE zusätzlich den Speicherplatz freigeben, den Variablen vom Datentyp TEXT oder BYTE belegen.

Vor dem Aufruf beachten

Anweisungsbezeichner und Satzzeiger sind nur in der Quellprogrammdatei ansprechbar, in der sie mit PREPARE bzw. DECLARE vereinbart wurden. Die FREE-Anweisung muß deshalb in derselben Datei stehen.



anweisungsbezeichner
Name der dynamischen Anweisung, wie er bei der PREPARE-Anweisung festgelegt wurde.

satzzeiger
Name des Satzzeigers, wie er bei der DECLARE-Anweisung für eine SELECT- oder INSERT-Anweisung festgelegt wurde. Der Satzzeiger muß zuvor mit CLOSE geschlossen werden.

Achtung:
Wurde der Satzzeiger für einen durch PREPARE festgelegten Anweisungsbezeichner vereinbart, so müssen Sie die FREE-Anweisung mit diesem Anweisungsbezeichner benutzen und nicht mit dem Satzzeiger.

blob-variable
Name der Variable, deren Speicherplatz freigegeben wird. Die Variable muß vom Datentyp TEXT oder BYTE sein. Wurde die Variable mit LOCATE IN MEMORY definiert, so wird der zugehörige Speicherplatz im Hauptspeicher freigegeben.
Wurde die Variable mit LOCATE IN FILE definiert, so wird die zugehörige Datei gelöscht.

Auswirkung von FREE

Ein Satzzeiger, der mit FREE freigegeben wurde, kann nicht mehr verwendet werden, bis er erneut mit OPEN geöffnet wurde.

Ein Anweisungsbezeichner, der mit FREE freigegeben wurde, kann nicht mehr verwendet werden.

Eine BLOB-Variable, die mit LOCATE IN MEMORY definiert wurde, kann erst wieder verwendet werden, wenn sie erneut mit LOCATE IN MEMORY initialisiert wird.

Eine BLOB-Variable, die mit LOCATE IN FILE *datei* definiert wurde, kann nach einem FREE nicht wieder verwendet werden.

Wurde *datei* nicht angegeben (die BLOB-Variable wird in einer temporären Datei gespeichert), so kann die BLOB-Variable nach einem erneuten LOCATE IN FILE wieder verwendet werden.

ANSI-Standard

Die Anweisung FREE ist nicht im ANSI-Standard enthalten.

Beispiel 1 für INFORMIX-4GL

Im folgenden Beispiel gibt die FREE-Anweisung eine Satzzeiger frei.

```
DEFINE knum INTEGER

LET knum = 101
DECLARE sel_zeiger CURSOR FOR SELECT * FROM auftrag
                   WHERE kunden_nr = knum

OPEN sel_zeiger
  .
  .
CLOSE sel_zeiger
FREE sel_zeiger
```

Beispiel 2 für INFORMIX-4GL

Im folgenden Beispiel gibt die FREE-Anweisung den Anweisungsbezeichner einer dynamischen Anweisung frei.

```
DEFINE h_neu, h_alt CHAR(3)

LET h_neu = "NEU"
LET h_alt = "ALT"

PREPARE aend_bez FROM
    "UPDATE posten SET herstellercode = ? WHERE herstellercode = ?"

EXECUTE aend_bez USING h_neu, h_alt
FREE aend_bez
```

Beispiel 3 für INFORMIX-4GL

Im folgenden Beispiel gibt die FREE-Anweisung den Anweisungsbezeichner der dynamischen SELECT-Anweisung frei. Der Satzzeiger darf nicht freigegeben werden.

```
DEFINE knum INTEGER

LET knum = 101

PREPARE suche_1 FROM
        "SELECT * FROM auftrag WHERE kunden_nr = ?"
DECLARE sel_zeiger CURSOR FOR suche_1

OPEN sel_zeiger USING knum
   .
   .
CLOSE sel_zeiger
FREE suche_1
```

Beispiel 4 für INFORMIX-4GL

Im folgenden Beispiel gibt die FREE-Anweisung den Speicherplatz für eine TEXT-Variable frei.

```
DEFINE ber1 TEXT
LOCATE ber1 in MEMORY

SELECT bericht INTO ber1 FROM ablage WHERE nummer=345
   .
   .
FREE ber1
```

> > > > CLOSE, OPEN, PREPARE

GRANT - Zugriffsrechte vergeben

GRANT vergibt Zugriffsrechte für Basistabellen und Views oder die gesamte Datenbank. Sie können mit GRANT festlegen, welcher Benutzer welche Zugriffsrechte besitzt. Zugriffsrechte aus früheren GRANT-Anweisungen bleiben weiterhin wirksam, d.h. eine GRANT-Anweisung hebt nicht die vorhergehende auf. Nur mit der Anweisung REVOKE kann ein Zugriffsrecht wieder entzogen werden.

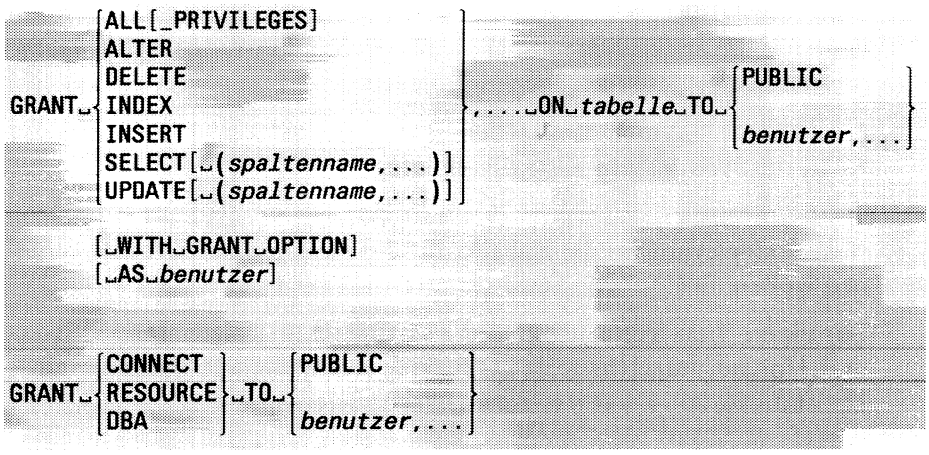
Die GRANT-Anweisung hat zwei Formate: das erste für die Vergabe von Zugriffsrechten auf Tabellen, das zweite für die Vergabe von Zugriffsrechten auf die gesamte Datenbank.

Vor dem Aufruf beachten

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie für die Vergabe von Tabellenzugriffsrechten entweder die Grant-Berechtigung für die entsprechenden Tabellenzugriffsrechte haben (siehe Klausel WITH GRANT OPTION) oder Tabelleneigentümer sein.

Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt. Sie müssen aber die Klausel AS *benutzer* verwenden.

Für die Vergabe von Datenbankzugriffsrechten müssen Sie DBA-Zugriffsrecht besitzen.



ALL[PRIVILEGES]

Alle Tabellenzugriffsrechte werden vergeben. ALL PRIVILEGES umfaßt die Tabellenzugriffsrechte Alter, Delete, Index, Insert, Select und Update.

ALTER
Tabellenzugriffsrecht, das das Ändern der Tabellenstruktur erlaubt. Dazu gehört das Einfügen und Löschen von Spalten, das Ändern des Namens und des Datentyps einer Spalte sowie das Ändern des Tabellennamens.

DELETE
Tabellenzugriffsrecht, das das Löschen von Sätzen der Tabelle erlaubt.

INDEX
Tabellenzugriffsrecht, das das Erzeugen eines Index für die Tabelle erlaubt.

INSERT
Tabellenzugriffsrecht, das das Neuaufnehmen von Sätzen in die Tabelle erlaubt.

SELECT[(*spaltenname*,...)]
Tabellenzugriffsrecht, das das Lesen von Sätzen erlaubt. Das Lesen kann auf die angegebenen Spalten eingeschränkt werden. *spaltenname* ist eine Spalte der angegebenen Tabelle.

(*spaltenname*,...) nicht angegeben:
Es ist das Lesen von allen Spalten der Tabelle erlaubt.

UPDATE[(*spaltenname*,...)]
Tabellenzugriffsrecht, das das Ändern von Sätzen erlaubt. Das Ändern kann auf die angegebenen Spalten eingeschränkt werden. *spaltenname* ist eine Spalte der angegebenen Tabelle.

(*spaltenname*,...) nicht angegeben:
Es ist das Ändern von allen Spalten der Tabelle erlaubt.

tabelle
Name oder Synonym der Tabelle, für die Sie die Zugriffsrechte vergeben wollen. Die Tabelle kann eine Basistabelle oder ein View sein.

Für einen View können nur die Zugriffsrechte Delete, Insert, Select und Update vergeben werden. Besteht der View aus mehr als einer Basistabelle, so ist nur das Zugriffsrecht Select möglich.

PUBLIC
Die angegebenen Tabellenzugriffsrechte gelten für die Allgemeinheit. Jeder Benutzer besitzt zusätzlich zu seinen eigenen die Zugriffsrechte der Allgemeinheit.

benutzer

Benutzerkennung, für die die angegebenen Tabellenzugriffsrechte gelten sollen. Für eine Benutzerkennung, die gleichzeitig Tabelleneigentümer ist, hat die Anweisung keine Wirkung.
GRANT speichert nur den Namen, nicht die Nummer der Benutzerkennung.

WITH GRANT OPTION

Die angegebenen Benutzer erhalten zusätzlich zu den Zugriffsrechten die Grant-Berechtigung, d.h. die Benutzer sind berechtigt, die erhaltenen Tabellenzugriffsrechte an andere Benutzer weiterzugeben.

WITH GRANT OPTION nicht angegeben:
Der Benutzer kann die verliehenen Zugriffsrechte nicht weitergeben.

AS *benutzer*

Die Zugriffsrechte werden anstelle eines anderen Benutzers vergeben. Es wird geprüft, ob dieser Benutzer die angegebenen Zugriffsrechte vergeben darf. Nur der in der Klausel AS *benutzer* genannte Benutzer kann die vergebenen Zugriffsrechte wieder entziehen.

Einschränkung:
Nur ein Benutzer mit dem DBA-Zugriffsrecht kann Zugriffsrechte anstelle eines anderen Benutzers vergeben.

AS *benutzer* nicht angegeben:
Der zur Ablaufzeit aktuelle Benutzer muß die angegebenen Zugriffsrechte vergeben dürfen. Nur dieser Benutzer kann die vergebenen Zugriffsrechte wieder entziehen.

CONNECT

Datenbankzugriffsrecht, das den Zugriff auf bestehende Tabellen erlaubt, wenn die entsprechenden Tabellenzugriffsrechte bestehen. Außerdem dürfen Views, temporäre Tabellen und Indizes auf temporären Tabellen erstellt und gelöscht werden.

RESOURCE

Umfaßt das Connect-Zugriffsrecht. Zusätzlich kann der Benutzer Basistabellen erstellen, die er ändern und löschen kann. Zusätzlich darf er für diese Basistabellen Indizes erzeugen, ändern und löschen sowie Zugriffsrechte vergeben und entziehen.

Bogen 12

DBA

Umfaßt das Resource-Zugriffsrecht. Zusätzlich darf der Benutzer:

- Alle Tabellenzugriffsrechte für alle Tabellen der Datenbank vergeben (GRANT ... AS *benutzer*)
- Tabellen, Views, Synonyme, Indizes und Constraints für andere Benutzer erzeugen und löschen (CREATE ..., DROP ...)
- Die Datenbank löschen (DROP DATABASE)
- Bei einer SE-Datenbank die Transaktionssicherung einschalten (START DATABASE)
- Die Datenbanksicherung aktualisieren (ROLLFORWARD DATABASE)
- Datenbankzugriffsrechte Connect, Resource und DBA vergeben und entziehen (GRANT, REVOKE).

Wenn Sie eine Datenbank mit CREATE DATABASE erzeugen, werden Sie automatisch Datenbankadministrator dieser Datenbank und besitzen DBA-Zugriffsrecht.

PUBLIC

Die angegebenen Datenbankzugriffsrechte gelten für die Allgemeinheit. Jeder Benutzer besitzt zusätzlich zu seinen eigenen die Zugriffsrechte der Allgemeinheit.

benutzer

Benutzerkennung, für die die angegebenen Datenbankzugriffsrechte gelten sollen. GRANT speichert nur den Namen, nicht die Nummer der Benutzerkennung.

Standard-Tabellenzugriffsrechte

In einer Nicht-ANSI-Datenbank werden automatisch bei jeder neu erzeugten Tabelle folgende Tabellenzugriffsrechte gesetzt:

```
GRANT DELETE,INSERT,SELECT,UPDATE TO PUBLIC
```

Um Tabellenzugriffsrechte einzuschränken, empfiehlt es sich, bei einer Nicht-ANSI-Datenbank zuerst alle Zugriffsrechte aufzuheben und anschließend die gewünschten Rechte zu erteilen.

Bei einer ANSI-Datenbank hat kein anderer Benutzer als der Eigentümer standardmäßig Zugriffsrechte auf die Tabelle. Alle Zugriffsrechte müssen Sie explizit mit GRANT vergeben.

GRANT in Transaktionen

Die Anweisung GRANT kann bei einer INFORMIX-SE-Datenbank nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

INFORMIX-ONLINE kann die Anweisung GRANT zurücksetzen.

Zusammenwirken von Datenbank- und Tabellenzugriffsrechten

Die folgende Tabelle beschreibt die Zusammenhänge zwischen Datenbank- und Tabellenzugriffsrechten. Es sind die Zugriffsrechte angegeben, die **mindestens** für die Ausführung der jeweiligen SQL-Anweisung nötig sind. Es müssen jeweils die angegebenen Tabellen- **und** Datenbankzugriffsrechte erfüllt sein.

Da das DBA-Zugriffsrecht alle Tabellenrechte für alle Tabellen der Datenbank umfaßt, kann ein Benutzer mit DBA-Zugriffsrecht alle folgenden Anweisungen ausführen.

Anweisung	Mindest-Zugriffsrechte	
	Tabellenebene	Datenbankebene
ALTER INDEX	Tabelleneigentümer oder Index	Connect
ALTER TABLE	Tabelleneigentümer oder Alter	Connect
CHECK TABLE	Tabelleneigentümer	Connect
CREATE AUDIT	Tabelleneigentümer	Connect
CREATE INDEX	Tabelleneigentümer oder Index	Resource
CREATE SYNONYM	-	Connect
CREATE TABLE	-	Resource
CREATE TEMP TABLE	-	Connect
CREATE VIEW	Tabelleneigentümer oder Select	Connect
DATABASE	-	Connect
DELETE	Tabelleneigentümer oder Delete	Connect
DROP AUDIT	Tabelleneigentümer	Connect
DROP DATABASE	-	DBA

Anweisung	Mindest-Zugriffsrechte	
	Tabellenebene	Datenbankebene
DROP INDEX	Indexeigentümer	Connect
DROP SYNONYM	Synonymeigentümer	Connect
DROP TABLE	Tabelleneigentümer	CONNECT
DROP VIEW	Vieweigentümer	Connect
GRANT	Tabelleneigentümer Grant-Berechtigung -	Connect Connect DBA
INFO	-	Connect
INSERT	Tabelleneigentümer oder Insert	Connect
LOAD	Tabelleneigentümer oder Insert	Connect
LOCK TABLE	Tabelleneigentümer	Connect
OUTPUT	Tabelleneigentümer oder Select	Connect
RECOVER TABLE	Tabelleneigentümer	Connect
RENAME COLUMN	Tabelleneigentümer oder Alter	Connect
RENAME TABLE	Tabelleneigentümer oder Alter	Resource
REPAIR TABLE	Tabelleneigentümer	Connect
REVOKE Tabellen-zugriffsrecht REVOKE Datenbank-zugriffsrecht	Benutzer, der das Tabellen-zugriffsrecht vergeben hat -	Connect DBA
ROLLFORWARD DATABASE	-	DBA
SELECT	Tabelleneigentümer oder Select	Connect
START DATABASE	-	DBA
UNLOAD	Tabelleneigentümer oder Select	Connect
UPDATE	Tabelleneigentümer oder Update	Connect
UPDATE STATISTICS	-	Connect

ANSI-Standard

Die Anweisung GRANT ist im ANSI-Standard enthalten, dort aber im Gegensatz zu INFORMIX nur innerhalb der Anweisung CREATE SCHEMA erlaubt. Die Datenbankzugriffsrechte CONNECT, RESOURCE und DBA und die Klausel AS *benutzer* sind nicht im ANSI-Standard enthalten.

Beispiel 1

Im folgenden Beispiel gibt die INFO-Anweisung zuerst die aktuellen Tabellen-Zugriffsrechte für die Tabelle *kunde* aus: Alle Benutzer haben mit Ausnahme des Alter-Zugriffsrechts sämtliche Tabellenzugriffsrechte (Standard bei einer Nicht-ANSI-Datenbank). Die anschließenden Anweisungen bewirken folgende Änderungen:

- für PUBLIC werden alle Rechte entzogen
- die Benutzer *usera* und *userb* erhalten sämtliche Tabellenzugriffsrechte außer das Alter-Zugriffsrecht
- der Benutzer *boss* erhält alle Tabellenzugriffsrechte
- der Benutzer *wichtl* darf nur lesen

```
INFO PRIVILEGES FOR kunde
```

Benutzer	Select	Update	Insert	Delete	Index	Alter
public	Alle	Alle	Ja	Ja	Ja	Nein

```
REVOKE ALL ON kunde FROM public;
GRANT ALL ON kunde TO usera,userb;
REVOKE ALTER ON kunde FROM usera,userb;
GRANT ALL ON kunde TO boss;
GRANT SELECT ON kunde TO wichtl;
INFO PRIVILEGES FOR kunde
```

Benutzer	Select	Update	Insert	Delete	Index	Alter
boss	Alle	Alle	Ja	Ja	Ja	Ja
usera	Alle	Alle	Ja	Ja	Ja	Nein
userb	Alle	Alle	Ja	Ja	Ja	Nein
wichtl	Alle	Keine	Nein	Nein	Nein	Nein

Beispiel 2

Im folgenden Beispiel wird das Select-Zugriffsrecht auf einzelne Spalten eingeschränkt.

```
REVOKE SELECT ON kunde FROM wichtl;
GRANT SELECT (vorname,nachname) ON kunde TO wichtl;
INFO PRIVILEGES FOR kunde
```

```
Benutzer     Select          Update          Insert  Delete  Index  Alter
boss          Alle            Alle            Ja      Ja      Ja     Ja
usera         Alle            Alle            Ja      Ja      Ja     Nein
userb         Alle            Alle            Ja      Ja      Ja     Nein
wichtl                                        Nein    Nein    Nein   Nein
              vorname
              nachname
```

Beispiel 3

Im nächsten Beispiel werden folgende Datenbankzugriffsrechte erteilt:

- Der Benutzer *boss* erhält das Datenbankzugriffsrecht DBA, d.h. er hat dieselben Rechte wie der ursprüngliche Eigentümer.
- Die Benutzer *usera* und *userb* erhalten Connect-Zugriffsrecht.

Durch Abfragen der Systemtabelle *sysusers* erhält man alle Benutzer, die Datenbankzugriffsrechte haben. Die Spalte *usertype* zeigt, welcher Art dieses Zugriffsrecht ist:
D = DBA, R = Resource, C = Connect.

```
GRANT DBA TO boss;
GRANT CONNECT TO usera,userb;
SELECT username, usertype FROM sysusers
```

```
username usertype

lomata   D
public   C
karin    C
sissi    C
boss     D
usera    C
userb    C
```

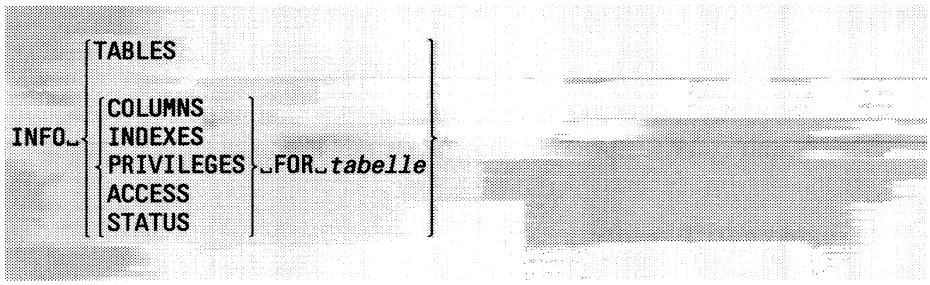
> > > > REVOKE

INFO - Tabellen-Informationen ausgeben

INFO informiert über die Tabellen der aktuellen Datenbank. Die Namen der Tabellen und Spalten, Datentypen der Spalten, Indizes, Zugriffsrechte und Statusinformationen werden ausgegeben.

Vor dem Aufruf beachten

Sie müssen das Connect-Zugriffsrecht für die Datenbank besitzen.



TABLES
: Die Namen und Synonyme von Basistabellen und Views der aktuellen Datenbank werden ausgegeben. Wenn Sie nicht der Eigentümer der Tabelle sind, wird die entsprechende Benutzerkennung dem Tabellennamen vorangestellt. Die Namen von Systemtabellen werden nicht ausgegeben.

COLUMNS
: Für die angegebene Tabelle wird der Name und der Datentyp der Spalten ausgegeben. Zusätzlich wird ausgegeben, ob die Spalte NULL-Werte enthalten darf.

INDEXES
: Für die angegebene Tabelle werden der Name, der Eigentümer und die Spalten der Indizes ausgegeben. Zusätzlich wird ausgeben, ob der Index eindeutig sein muß oder Duplikate zuläßt und ob dem Index das Cluster-Attribut zugeordnet ist.
Die Namen von Indizes, die für Constraints erzeugt werden, werden ebenfalls ausgegeben.

PRIVILEGES
: Für die angegebene Tabelle werden die Tabellenzugriffsrechte der Benutzer ausgegeben.
Ein Benutzer kann die angezeigten Tabellenzugriffsrechte nur anwenden, wenn er auch das dazu nötige Datenbankzugriffsrecht besitzt

(siehe GRANT-Anweisung).

ACCESS
Diese Angabe ist gleichbedeutend mit PRIVILEGES.

STATUS
Für die angegebene Tabelle werden folgende Informationen ausgegeben: Tabellenname, Eigentümer, Satzlänge, Anzahl Sätze, Anzahl Spalten, Erstellungsdatum, Name des Audit-Protokolls (falls vorhanden). Die Anzahl der Sätze ist in der Regel nur dann richtig, falls unmittelbar vor INFO STATUS ein UPDATE STATISTICS durchgeführt worden ist.

FOR *tabelle*
Name oder Synonym der Tabelle, über die Sie sich informieren wollen. Der Name einer temporären Tabelle ist nicht erlaubt.

Datenbankzugriffsrechte

Datenbankzugriffsrechte werden durch INFO PRIVILEGES nicht angezeigt. Diese lassen sich über die Systemtabelle *sysusers* erfragen:

```
SELECT * FROM sysusers
```

Zugriffsrechte auf Betriebssystemebene werden mit der INFO-Anweisung nicht ausgegeben.

ANSI-Standard

Die Anweisung INFO ist nicht im ANSI-Standard enthalten.

Beispiel 1

Die folgende INFO-Anweisung gibt die Namen aller Tabellen der aktuellen Datenbank *versand* aus. Die Namen von Synonymen und Views werden ebenfalls ausgegeben.

```
INFO TABLES
```

```
Tabellenname

artikel          auftrag          h                hersteller
kunde            muenchen         posten
```

Beispiel 2

Die folgende INFO-Anweisung gibt die Spalten der Tabelle *kunde* aus.

```
INFO COLUMNS FOR kunde
```

```
Spaltenname          Typ                                  NULL's

kunden_nr            serial                               no
vorname              char(15)                             yes
nachname             char(15)                             yes
firma                char(20)                             yes
adresse1             char(20)                             yes
adresse2             char(20)                             yes
ort                  char(15)                             yes
bundesland           char(2)                              yes
plz                  char(5)                              yes
telefon              char(18)                             yes
```

Beispiel 3

Die folgende INFO-Anweisung gibt die Namen der Indizes aus, die für die Tabelle *kunde* definiert sind.

```
INFO INDEXES FOR kunde
```

```
Indexname            Eigner    Typ        Cluster  Spalten

k_nr_ix              lomata    Eindeutig  Nein     kunden_nr

k_plz_ix             lomata    Duplikate  Nein     plz
```

Beispiel 4

Die folgende INFO-Anweisung gibt die Tabellenzugriffsrechte für die Tabelle *kunde* aus.

```
INFO PRIVILEGES FOR kunde
```

```
Benutzer     Select          Update          Insert  Delete  Index  Alter

public       Alle             Alle            Ja      Ja      Ja     Nein
```

Beispiel 5

Die folgende INFO-Anweisung gibt den Status der Tabelle *kunde* aus.

```
INFO STATUS FOR kunde
```

```
Tabellen-Name       kunde
Eigentuemer         lomata
Satzlaenge          134
Anzahl Saetze       18
Anzahl Spalten      10
Erstellungsdatum    07.04.1990
```

INSERT- Sätze in Tabelle einfügen

INSERT fügt in eine bestehende Tabelle einen oder mehrere Sätze ein.

Vor dem Aufruf beachten

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich entweder das Insert-Zugriffsrecht haben oder aber Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

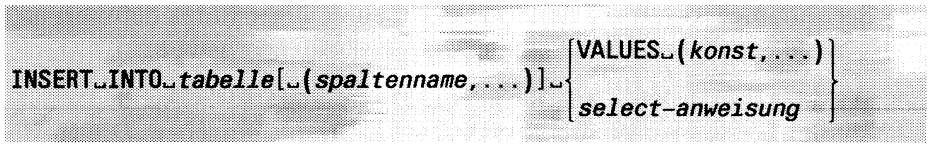


tabelle
Name oder Synonym der Tabelle, in die Sätze eingefügt werden sollen. Die Tabelle kann eine Basistabelle, eine temporäre Tabelle oder ein änderbarer View (siehe CREATE VIEW) sein.

(*spaltenname*,...**)**
Die einzufügenden Sätze enthalten nur Werte für die angegebenen Spalten. Geben Sie für *spaltenname* eine Spalte der angegebenen Tabelle ein. Die Reihenfolge der Spalten muß nicht mit der Tabelle übereinstimmen.
Sie müssen mindestens alle Spalten angeben, die mit NOT NULL definiert sind. Wenn Sie eine Spalte nicht angeben, wird sie automatisch mit dem NULL-Wert besetzt.

(*spaltenname*,...**)** nicht angegeben:
Die folgenden Wert-Angaben beziehen sich auf alle Spalten der Tabelle *tabelle* in der Reihenfolge, wie sie bei CREATE/ALTER TABLE bzw. CREATE VIEW angegeben wurden.
Mit INFO COLUMNS FOR *tabelle* können Sie die Reihenfolge der Spalten der Tabelle feststellen.

VALUES (*konst*,...)

Konstante, die den Wert für eine Spalte angibt.
Der *i*-te Wert wird der *i*-ten Spalte in der Spaltenliste zugewiesen, falls eine Spaltenliste angegeben wurde, ansonsten der *i*-ten Spalte der Tabelle.
Die Werte müssen mit dem Datentyp der Spalte verträglich sein (siehe Kapitel 4).

Wenn Sie einer Spalte mit Datentyp SERIAL den Wert 0 zuweisen, dann weist INFORMIX automatisch eine Satznummer als Spaltenwert zu. Wenn Sie einen Wert größer 0 zuweisen, so gilt dieser Wert als Satznummer. Wenn auf der Spalte kein eindeutiger Index definiert ist, können Sie dieselbe Satznummer auch mehrfach vergeben.

Wird die Anweisung INSERT in eine Programmiersprache eingebettet, können Sie die Konstante über eine Hostvariable eingeben. Die Hostvariablen können auch vom Datentyp TEXT oder BYTE sein. Bei Programmeinbettung kann statt einer Liste von Werten auch eine Record-Variable oder eine Struktur angeben werden (siehe Handbuch für die entsprechende Programmeinbettung [3, 4, 6]).

select-anweisung

SELECT-Anweisung, deren Ergebnistabelle die einzufügenden Sätze liefert. Die Werte der Ergebnistabelle müssen mit den Datentypen der entsprechenden Spalten verträglich sein (siehe Kapitel 4).

Einschränkung:
Die SELECT-Anweisung darf die Klauseln INTO, INTO TEMP, UNION und ORDER BY nicht enthalten. In der FROM-Klausel dürfen Sie nicht die Tabelle angeben, in die die neuen Sätze eingetragen werden.

Sperren beim Einfügen

Bei einer Datenbank ohne Transaktionssicherung wird jeweils nur der eine Satz exklusiv gesperrt, der momentan eingefügt wird.

Wenn Sie auf einer Datenbank mit Transaktionssicherung arbeiten und die INSERT-Anweisung außerhalb einer Transaktionsklammer ausführen, dann werden alle betroffenen Sätze exklusiv gesperrt, bis die INSERT-Anweisung komplett abgearbeitet ist.

Wird die INSERT-Anweisung innerhalb einer Transaktionsklammer ausgeführt, dann werden alle betroffenen Sätze bis zum Ende der Transaktion exklusiv gesperrt.

Ist die Zahl der betroffenen Sätze sehr groß, kann die maximal zulässige Anzahl an Sperren überschritten werden. Die Anzahl der möglichen Sperren ist bei INFORMIX-SE durch das Betriebssystem bzw. bei INFORMIX-ONLINE durch Shared Memory Parameter beschränkt.

Sie sollten in diesem Fall die Anzahl der zu einzufügenden Sätze reduzieren oder die ganze Tabelle vor der Ausführung der INSERT-Anweisung sperren (siehe LOCK TABLE).

Datenintegrität beim Einfügen

Datenbank ohne Transaktionssicherung

Wenn eine INSERT-Anweisung abgebrochen wird, dann ist sie teilweise ausgeführt worden. Die bis zum Abbruch der Anweisung eingefügten Sätze sind in der Tabelle vorhanden

Beispiel:

Sie möchten mit INSERT ... SELECT ... 100 Sätze einfügen. Beim 57. Satz tritt ein Konvertierungsfehler auf und INSERT wird abgebrochen. Die ersten 56 Sätze, die bereits eingetragen wurden, sind in der Tabelle vorhanden.

Datenbank mit Transaktionssicherung

Bei einer Nicht-ANSI-Datenbank wird jede INSERT-Anweisung außerhalb einer Transaktion automatisch als einzelne Transaktion behandelt. Eine unvollständig ausgeführte INSERT-Anweisung wird automatisch zurückgesetzt. Unvollständige INSERT-Anweisungen wie in obigem Beispiel können also nicht vorkommen.

Bei einer Nicht-ANSI-Datenbank innerhalb einer Transaktion und bei einer ANSI-Datenbank müssen Sie die Transaktion, in der die INSERT-Anweisung ausgeführt wird, explizit mit ROLLBACK WORK bzw. COMMIT WORK beenden.

Erfolgskontrolle bei Programmeinbettung

	Positivfall	Fehlerfall
INFORMIX-4GL	status = 0 SQLCA.SQLERRD[3] ist die Anzahl der mit INSERT eingefügten Sätze.	status < 0 SQLCA.SQLERRD[3] ist die Anzahl der mit INSERT erfolgreich eingefügten Sätze. Der Rest der Sätze wurde nicht eingefügt.
INFORMIX-ESQL/C	sqlca.sqlcode = 0 sqlca.sqlerrd[2] ist die Anzahl der mit INSERT eingefügten Sätze.	sqlca.sqlcode < 0 sqlca.sqlerrd[2] ist die Anzahl der mit INSERT erfolgreich eingefügten Sätze. Der Rest der Sätze wurde nicht eingefügt.
INFORMIX-ESQL/COBOL	SQLCODE OF SQLCA = 0 SQLERRD[3] ist die Anzahl der mit INSERT eingefügten Sätze.	SQLCODE OF SQLCA < 0 SQLERRD[3] ist die Anzahl der mit INSERT erfolgreich eingefügten Sätze. Der Rest der Sätze wurde nicht eingefügt.

ANSI-Standard

Die Anweisung INSERT ist im ANSI-Standard enthalten.

Beispiel 1

Die folgende INSERT-Anweisung fügt einen Satz in die Tabelle *artikel* ein. Die Spaltenwerte, die nicht angegeben sind, werden auf NULL gesetzt.

```
INSERT INTO artikel (artikel_nr, herstellercode, bezeichnung)
            VALUES (7,"ANZ","Softball")
```

Beispiel 2

Die folgende INSERT-Anweisung fügt mehrere Sätze in die Tabelle *artikel* ein. Die Sätze werden durch eine SELECT-Anweisung bereitgestellt.

```
INSERT INTO artikel
           SELECT * FROM art2 WHERE herstellercode = "NEU"
```

Beispiel 3 für INFORMIX-4GL

Die folgende INSERT-Anweisung fügt einen Satz in die Tabelle *artikel* ein. Die Spaltenwerte werden durch die Record-Variable *art* angegeben.

```
DEFINE art RECORD LIKE artikel.*
    .
    .
    .
INSERT INTO artikel VALUES (art.*)
```

>>>> DELETE, INFO, SELECT, UPDATE

LOAD - Sätze aus Datei in Tabelle einfügen

LOAD fügt Sätze aus einer Eingabedatei in eine Tabelle ein.

Vor dem Aufruf beachten

Die LOAD-Anweisung kann nur interaktiv und bei INFORMIX-4GL verwendet werden.

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich entweder das Insert-Zugriffsrecht haben oder aber Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

Bei einer Datenbank mit Transaktionssicherung muß die Anweisung LOAD immer innerhalb einer Transaktion ablaufen.

"*datei*"
Name der Eingabedatei, die die einzufügenden Sätze enthält.
Wird die Anweisung LOAD in INFORMIX-4GL verwendet, können Sie auch eine alphanumerische Variable statt "*datei*" angeben.

DELIMITER "*zeichen*"
Einzelnes Zeichen, das als Trennzeichen zwischen den Datenfeldern in der Eingabedatei verwendet wird.
Wird die Anweisung LOAD in INFORMIX-4GL verwendet, können Sie auch eine alphanumerische Variable statt "*zeichen*" angeben.

Sie können auch Buchstaben, Ziffern, Komma (,) oder Punkt (.) als Trennzeichen definieren. Sie müssen aber sicherstellen, daß das Trennzeichen nicht in den einzufügenden Werten vorkommt bzw. entwertet wird.
Wird das Trennzeichen oder der Gegenschrägstrich innerhalb der Sätze der Eingabedatei verwendet, so müssen sie durch einen vorangestellten Gegenschrägstrich (\ = ASCII 92) entwertet werden (UNLOAD macht das automatisch so). Beachten Sie, daß das Zeichen Ö im deutschen ISO-7-Bit-Code dem Gegenschrägstrich entspricht und deshalb

entwertet werden muß.

DELIMITER "*zeichen*" nicht angegeben:
Das mit der Umgebungsvariablen DBDELIMITER definierte Zeichen wird als Trennzeichen benutzt. Ist DBDELIMITER nicht definiert, so gilt der senkrechte Strich (| = ASCII 124) als Standardtrennzeichen.

Achtung:
Wenn Sie Sätze aus einer Tabelle mit UNLOAD entladen und sie anschließend mit LOAD in eine Tabelle einfügen wollen, müssen Sie bei UNLOAD und LOAD dasselbe Trennzeichen verwenden.

INSERT INTO *tabelle*
Name oder Synonym der Tabelle, in die die Sätze eingefügt werden sollen. Für *tabelle* dürfen Sie eine Basistabelle, eine temporäre Tabelle oder einen änderbaren View (siehe CREATE VIEW) angeben.

(*spaltenname,...***)**
Namen der Spalten, in die die Werte aus der Eingabedatei eingetragen werden. Sie müssen mindestens die Spalten angeben, die mit NOT NULL definiert sind, ansonsten wird LOAD abgewiesen. Die übrigen Spalten können fehlen. Sie werden dann mit NULL-Werten besetzt.

(*spaltenname,...***)** nicht angegeben:
Es gelten alle Spalten der Tabelle, und zwar in der Reihenfolge, in der sie für die Tabelle definiert sind.

variable
Wird die Anweisung LOAD in INFORMIX-4GL verwendet, können Sie auch eine Variable statt INSERT... angeben. Die Variable muß eine INSERT-Anweisung ohne VALUES- und SELECT-Klausel enthalten.

Datentyp und Format der Werte in der Eingabedatei

Die Anzahl der eingelesenen Werte pro Datensatz in der Eingabedatei muß der Anzahl der (angegebenen) Spalten entsprechen.

Soll eine Spalte mit NULL besetzt werden, wird in der Eingabedatei für diese Spalte zwischen den entsprechenden beiden Trennzeichen kein Wert angegeben.

Numerische Werte dürfen den Wertebereich der Spalte nicht überschreiten (siehe Kapitel 4).

Alphanumerische Werte werden abgeschnitten, wenn sie länger sind als die Spalte. Eine alphanumerischer Wert aus Leerzeichen wird durch ein Leerzeichen erzeugt.

Enthalten alphanumerische Werte das Trennzeichen, den Gegenschrägstrich oder "neue Zeile", so müssen diese mit dem Gegenschrägstrich entwertet werden.

Werte für Spalten der Datentypen DATE und MONEY dürfen führende Leerzeichen haben.

Werte des Datentyps DATE müssen im aktuellen DATE-Format angegeben sein (durch die Umgebungsvariable DBDATE bestimmt).

Ein Wert vom Datentyp MONEY kann ein führendes Währungssymbol enthalten.

Bei Werten vom Datentyp DATETIME und INTERVAL müssen die einzelnen Komponenten als Zeichen in der Form *yyyy-mm-dd hh:mm:ss* angegeben werden (siehe auch Kapitel 4).

Werte vom Datentyp BYTE sind im Hexadezimal-Format anzugegeben. Führende Leerzeichen sind nicht erlaubt.

Weicht ein Wert vom Datentyp ab, der für die Spalte definiert ist, so versucht INFORMIX, die Werte in das passende Datenformat zu konvertieren.

Mit UNLOAD entladene Sätze besitzen genau das Format, das die LOAD-Anweisung als Eingabe benötigt. Beachten Sie daher auch die Hinweise zu UNLOAD.

Sperren beim Laden

Bei einer Datenbank ohne Transaktionssicherung wird jeweils nur der eine Satz exklusiv gesperrt, der momentan eingefügt wird.

Wenn Sie auf einer Datenbank mit Transaktionssicherung arbeiten, muß die LOAD-Anweisung innerhalb einer Transaktionsklammer ausgeführt werden. Alle eingefügten Sätze werden bis zum Ende der Transaktion exklusiv gesperrt.

Ist die Zahl der eingefügten Sätze sehr groß, kann die maximal zulässige Anzahl an Sperren überschritten werden. Die Anzahl der möglichen Sperren ist bei INFORMIX-SE durch das Betriebssystem bzw. bei INFORMIX-ONLINE durch Shared Memory Parameter beschränkt.

Sie sollten in diesem Fall die Anzahl der Sätze reduzieren oder die ganze Tabelle vor der Ausführung der LOAD-Anweisung sperren (siehe LOCK TABLE).

Datenintegrität beim Einfügen von Sätzen aus einer Datei

Ohne Transaktionssicherung

Ein abgebrochener LOAD wird auch tatsächlich nur teilweise ausgeführt.

Beispiel:

Eine LOAD-Anweisung, die eigentlich 100 Sätze laden würde, läuft beim 57. Satz auf einen Konvertierungsfehler. Die LOAD-Anweisung wird jetzt zwar abgebrochen, die bereits eingespeicherten 56 Sätze verbleiben jedoch in der Tabelle.

Mit Transaktionssicherung

LOAD kann nur innerhalb einer Transaktion ablaufen, außerhalb einer Transaktion wird LOAD abgewiesen. Alle Sätze, die über LOAD eingefügt werden, werden bis zum Ende der Transaktion gesperrt.

Das obige Beispiel einer abgebrochenen LOAD-Anweisung kommt bei einer Datenbank mit Transaktionen nicht zum Tragen. Falls eine LOAD-Anweisung nicht korrekt beendet wird, können Sie die Transaktion explizit mit ROLLBACK WORK oder COMMIT WORK beenden.

Erfolgskontrolle bei Programmeinbettung

	Positivfall	Fehlerfall
INFORMIX-4GL	status = 0 SQLCA.SQLERRD[3] ist die Anzahl der mit LOAD eingefügten Sätze.	status < 0 SQLCA.SQLERRD[3] ist die Anzahl der mit LOAD erfolgreich eingefügten Sätze. Der Rest der Sätze wurde nicht eingefügt.

ANSI-Standard

Die Anweisung LOAD ist nicht im ANSI-Standard enthalten.

Beispiel 1

In die Tabelle *hersteller* sollen Sätze aus der Datei *l.herst* eingefügt werden.

Inhalt der Datei *l.herst*:

```
NEU|neusport
OPA|opafit
HIT|juniorhit
```

```
LOAD FROM l.herst INSERT INTO hersteller;
SELECT * FROM hersteller
```

```
herstellercode herstellername
SMT            Schmitt
ANZ            Anzabel
NRG            Norgesta
HSK            Hasken
HRO            Herold
NEU            neusport
OPA            opafit
HIT            juniorhit
```

Beispiel 2

In die Tabelle *kunde* sollen Sätze aus der Datei *l.herst* eingefügt werden. Jeder Satz in der Ladedatei enthält nur Werte für die Spalten *kunden_nr*, *firma* und *ort*. Die übrigen Spalten werden beim Laden automatisch mit NULL-Werten versorgt. Voraussetzung dafür ist natürlich, daß diese Spalten NULL-Werte zulassen.

Inhalt der Datei */usr/lomata/b/l.herst*:

```
119|Otto trimm|Muenchen
120|Opa fit|Muenchen
121|Junior Sport|Muenchen
```

```
LOAD FROM "/usr/lomata/b/l.herst"
       INSERT INTO kunde (kunden_nr,firma,ort);
SELECT kunden_nr,firma,ort,telefon
       FROM kunde
       WHERE telefon IS NULL
```

```
kunden_nr firma                ort             telefon
      121 Junior Sport         Muenchen
      120 Opa fit              Muenchen
      119 Otto trimm           Muenchen
```

LOCK TABLE - Tabelle sperren

LOCK TABLE sperrt eine Tabelle gegen andere Prozesse. LOCK TABLE wird abgewiesen, wenn die Tabelle oder mindestens ein Satz der Tabelle bereits durch einen anderen Prozeß gesperrt ist.

Vor dem Aufruf beachten

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

Bei einer Nicht-ANSI-Datenbank mit Transaktionssicherung muß LOCK TABLE innerhalb einer Transaktionsklammer ausgeführt werden.

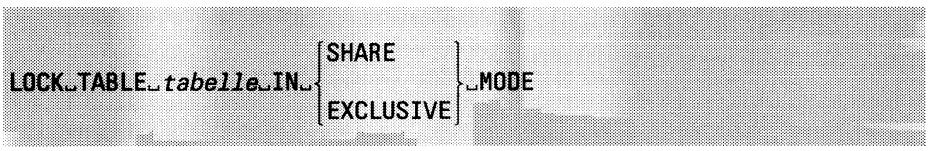


tabelle
Name oder Synonym der Basistabelle, die gesperrt werden soll. Es kann keine Systemtabelle angegeben werden. Auf die Sätze der gesperrten Tabelle werden nach einem LOCK TABLE keine Einzelsperren mehr gesetzt.

SHARE
Die Tabelle kann von anderen Prozessen noch gelesen, aber nicht mehr verändert werden.

EXCLUSIVE
Auf die Tabelle wird eine exklusive Sperre gesetzt. Andere Prozesse können die Tabelle weder ändern noch lesen.

Bei INFORMIX-ONLINE können Prozesse mit der Isolationsstufe Dirty Read die Tabelle weiterhin lesen, da Dirty Read Sperren anderer Prozesse ignoriert.

LOCK TABLE bei Datenbanken mit Transaktionssicherung

Arbeitet eine Datenbank mit Transaktionssicherung, so ist LOCK TABLE nur innerhalb einer Transaktion erlaubt. Außerhalb einer Transaktion wird LOCK TABLE abgewiesen.

Bei einer Datenbank mit Transaktionssicherung setzen die Anweisungen DELETE, INSERT, LOAD und UPDATE Exklusiv-Sperren auf die geänderten Sätze, die bis zum Ende der Transaktion gehalten werden.
Die Anzahl der möglichen Sperren ist bei INFORMIX-SE durch das Betriebssystem bzw. bei INFORMIX-ONLINE durch Shared Memory Parameter beschränkt.
Wenn eine Transaktion zu viele Sperren verursachen würde, empfiehlt es sich, unmittelbar nach Beginn der Transaktion die ganze betroffene Tabelle mit LOCK TABLE zu sperren. Nach LOCK TABLE werden keine Einzelsperren mehr auf die gesperrte Tabelle gesetzt.

Bei Transaktionsende (COMMIT WORK oder ROLLBACK WORK) werden automatisch alle durch LOCK TABLE erzeugten Sperren zurückgesetzt.

Dauer der Tabellensperre

Durch LOCK TABLE erzeugte Sperren werden spätestens mit der Beendigung von INFORMIX zurückgesetzt.

Bei einer Datenbank ohne Transaktionssicherung kann die Tabellensperre mit der Anweisung UNLOCK TABLE freigegeben werden, bei einer Datenbank mit Transaktionen hingegen nicht. Das Beenden einer Transaktion mit den Anweisungen COMMIT WORK oder ROLLBACK WORK gibt dort die Tabellensperre frei.

Sie können beliebig viele LOCK-Anweisungen auf die gleiche Tabelle nacheinander absetzen, wenn Sie immer denselben Sperrmodus (SHARE bzw. EXCLUSIVE) angeben. Die erste LOCK-Anweisung wird ausgeführt, die nachfolgenden werden ignoriert.

Wollen Sie die Tabellensperre ändern (von SHARE auf EXCLUSIVE und umgekehrt), so müssen Sie erst mit UNLOCK TABLE die bestehende Sperre aufheben. Andernfalls wird LOCK TABLE abgewiesen.

ANSI-Standard

Die Anweisung LOCK TABLE ist nicht im ANSI-Standard enthalten.

Beispiel

Mit folgender Anweisung reservieren Sie sich die Tabelle *auftrag* exklusiv, d.h. kein anderer Prozeß kann auf die Tabelle zugreifen.

```
LOCK TABLE auftrag IN EXCLUSIVE MODE
```

>>>> BEGIN WORK, COMMIT WORK, ROLLBACK WORK, UNLOCK TABLE

OPEN - Satzzeiger öffnen

OPEN öffnet einen Satzzeiger, den Sie mit der Anweisung DECLARE vereinbart haben.

Haben Sie den Satzzeiger für eine SELECT-Anweisung vereinbart, dann wertet OPEN die SELECT-Anweisung aus. Dabei verwendet OPEN die aktuellen Werte der Variablen, die in der SELECT-Anweisung vorkommen und überprüft die Zugriffsrechte. Die Ergebnistabelle der SELECT-Anweisung wird erst durch den 1. FETCH auf den Satzzeiger bestimmt (siehe FETCH-Anweisung).

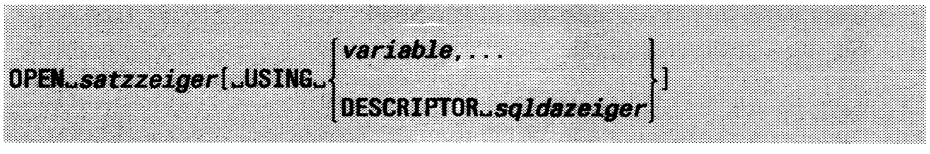
Haben Sie den Satzzeiger für eine INSERT-Anweisung vereinbart, dann stellt OPEN einen Puffer für die einzufügenden Sätze bereit.

Wenn Sie in der OPEN-Anweisung einen bereits geöffneten Satzzeiger angeben, so wird dieser zuerst geschlossen und dann mit den aktuellen Werten der Hostvariablen neu geöffnet.

Die Betriebsmittel, die der Satzzeiger belegt, können mit der FREE-Anweisung wieder freigegeben werden.

Vor dem Aufruf beachten

Satzzeiger sind nur in der Quellprogrammdatei ansprechbar, in der sie mit DECLARE vereinbart werden. Der mit OPEN angesprochene Satzzeiger muß zuvor mit DECLARE für eine SELECT- oder INSERT-Anweisung vereinbart werden.



satzzeiger
Name des Satzzeigers, den Sie öffnen wollen.

USING
Die USING-Klausel dürfen Sie nur für eine dynamische SELECT-Anweisung angeben, die mit PREPARE vorbereitet wurde. Die SELECT-Anweisung muß Fragezeichen als Platzhalter für Werte enthalten.

USING nicht angegeben:
Die mit dem Satzzeiger verbundene Anweisung ist eine INSERT-Anweisung oder eine SELECT-Anweisung, die keine Fragezeichen als Platzhalter enthält.

variable
Hostvariable, deren Wert einem Fragezeichen der dynamischen SELECT-Anweisung zugewiesen wird. Der Datentyp der Hostvariablen muß zum entsprechenden Wert passen, für den das Fragezeichen als Platzhalter in der SELECT-Anweisung steht. Die Werte der Hostvariablen werden in der aufgeführten Reihenfolge den Fragezeichen der dynamischen SELECT-Anweisung zugewiesen. Die Anzahl der angegebenen Hostvariablen muß dabei der Anzahl der Fragezeichen entsprechen.

Achtung:
Sie dürfen keine Indikatorvariablen an die Hostvariablen binden.

DESCRIPTOR *sqldazeiger*
Die Werte für die Fragezeichen der dynamischen SELECT-Anweisung werden durch eine *sqlda*-Struktur angegeben.

Achtung:
Die USING DESCRIPTOR-Klausel können Sie nur bei der Einbettung in C-Programme verwenden.

sqldazeiger
Zeiger auf eine *sqlda*-Struktur, die auf die Werte zeigt, die den Fragezeichen der dynamischen SELECT-Anweisung zugewiesen werden. Genauere Informationen zur *sqlda*-Struktur finden Sie im Handbuch für die C-Einbettung [3].

Auswertung von Variablen

Der Zeitpunkt, zu dem der Wert von Hostvariablen ausgewertet wird, unterscheidet sich je nach dem, ob einfache Variablen oder Arrays verwendet werden.

- Einfach Variablen werden beim Öffnen des Satzzeigers mit OPEN bzw. FOREACH ausgewertet.
- Bei Array-Variablen wird die Indexvariable bei der Definition des Satzzeigers mit DECLARE ausgewertet, der Wert des Array-Elements selbst erst beim Öffnen des Satzzeigers mit OPEN bzw. FOREACH.

Beispiel für INFORMIX-ESQL/C

```
$int zahl[5] ;
int i;

zahl[1] = 10;
zahl[2] = 20;
zahl[3] = 30;
zahl[4] = 40;
zahl[5] = 50;
i=3;

$ DATABASE versand
$ DECLARE selzeig CURSOR FOR
     SELECT * FROM kunde WHERE kunden_nr > $zahl[i];
zahl[3] = 100;
i=1;
$OPEN selzeig;
```

i wird zum Zeitpunkt des DECLARE ausgewertet, also mit dem Wert 3. Beim OPEN wird dann der zu diesem Zeitpunkt aktuelle Wert von *zahl[3]* verwendet, also 100.

Erfolgskontrolle bei Programmeinbettung

Wird mit OPEN ein Insert-Satzzeiger eröffnet, der noch offen war, so werden zuerst die gepufferten Sätze in die Datenbank eingefügt und dann der Satzzeiger neu eröffnet.
Die folgenden Variablen geben Auskunft über die Anzahl der erfolgreich eingefügten Sätze.

	Positivfall	Fehlerfall
INFORMIX-4GL	status = 0 SQLCA.SQLERRD[3] ist die Anzahl der mit OPEN eingefügten Sätze.	status < 0 SQLCA.SQLERRD[3] ist die Anzahl der mit OPEN erfolgreich eingefügten Sätze. Der Rest der Sätze ist verloren.
INFORMIX-ESQL/C	sqlca.sqlcode = 0 sqlca.sqlerrd[2] ist die Anzahl der mit OPEN eingefügten Sätze.	sqlca.sqlcode < 0 sqlca.sqlerrd[2] ist die Anzahl der mit OPEN erfolgreich eingefügten Sätze. Der Rest der Sätze ist verloren.

	Positivfall	Fehlerfall
INFORMIX-ESQL/COBOL	SQLCODE OF SQLCA = 0 SQLERRD[3] ist die Anzahl der mit OPEN eingefügten Sätze.	SQLCODE OF SQLCA < 0 SQLERRD[3] ist die Anzahl der mit OPEN erfolgreich eingefügten Sätze. Der Rest der Sätze ist verloren.

ANSI-Standard

Die Anweisung OPEN ist im ANSI-Standard enthalten.

Beispiel 1 für INFORMIX-4GL

Beim ersten Beispiel werden die Hostvariablen direkt in der SELECT-Anweisung angegeben.

```
DEFINE hcode CHAR(3)

DECLARE selzeiger CURSOR FOR
        SELECT * FROM hersteller WHERE herstellercode = hcode

OPEN selzeiger
```

Beispiel 2 für INFORMIX-4GL

Beim folgenden Beispiel wird eine dynamische SELECT-Anweisung mit PREPARE vorbereitet und die Hostvariablen werden bei der OPEN-Anweisung übergeben.

```
DEFINE hcode CHAR(3)

PREPARE anwbez
        FROM "SELECT * FROM hersteller WHERE herstellercode = ?"

DECLARE selzeiger CURSOR FOR anwbez

OPEN selzeiger USING hcode
```

> > > > CLOSE, DECLARE, FETCH, FLUSH, PREPARE, PUT

OUTPUT - Sätze aus Tabelle ausgeben

OUTPUT speichert Sätze, die Sie über die SELECT-Anweisung auswählen, in einer Datei oder übergibt sie an ein Programm als Standardeingabe. Das Ausgabeformat entspricht dem Format der jeweiligen Bildschirmausgabe.

Vor dem Aufruf beachten

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich entweder das SELECT-Zugriffsrecht für die entsprechenden Spalten haben oder aber Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

TO "*datei*"
Name der Datei, die die Sätze aufnehmen soll. Existiert die angegebene Datei bereits, so wird sie überschrieben.

TO PIPE "*programm*"
Geben Sie das Programm an, das die Sätze weiterverarbeiten soll (als Standardeingabe).

WITHOUT HEADINGS
Die Spaltenwerte werden ohne den Spaltennamen ausgegeben.

WITHOUT HEADINGS nicht angegeben:
Die Spaltenwerte werden mit dem Spaltennamen ausgegeben.

select-anweisung
SELECT-Anweisung, deren Ergebnistabelle ausgegeben wird. Die SELECT-Anweisung darf nicht die Klauseln INTO TEMP und INTO enthalten.

Zusammenhang von OUTPUT und UNLOAD

Dieselbe Funktion wie OUTPUT bietet auch die Anweisung UNLOAD. Ein wesentlicher Unterschied zeigt sich lediglich in der Ausgabe: UNLOAD gibt genau in dem Format aus, das die LOAD-Anweisung als Eingabe benötigt.

ANSI-Standard

Die Anweisung OUTPUT ist nicht im ANSI-Standard enthalten.

Beispiel

Alle Sätze der Tabelle *hersteller* werden in die Datei */usr/lomata/b/out.herst* ausgegeben.

```
OUTPUT TO "/usr/lomata/b/out.herst"
       SELECT * FROM hersteller
```

Inhalt der Datei */usr/lomata/b/out.herst*:

```
herstellercode herstellername
SMT            Schmitt
ANZ            Anzabel
NRG            Norgesta
HSK            Hasken
HRO            Herold
```

> > > > LOAD, SELECT, UNLOAD

PREPARE - Dynamische Anweisungen vorbereiten

PREPARE bereitet eine oder mehrere dynamische SQL-Anweisungen auf die spätere Ausführung vor. Eine SELECT-Anweisung ohne INTO TEMP wird mit DECLARE und OPEN ausgeführt, alle anderen Anweisungen mit EXECUTE.

Die Betriebsmittel, die die vorbereitete Anweisung belegt, können mit der FREE-Anweisung wieder freigegeben werden.

Vor dem Aufruf beachten

Die PREPARE-Anweisung muß in der Quellprogrammdatei statisch vor der ersten Anweisung stehen, die den vereinbarten Anweisungsbezeichner verwendet. Alle Anweisungen, die diesen mit PREPARE vereinbarten Anweisungsbezeichner verwenden, müssen in derselben Datei stehen.

anweisungsbezeichner
Name für die dynamische Anweisung. Mit diesem Namen ist die vorbereitete Anweisung in der Quellprogrammdatei global ansprechbar. Der Name muß innerhalb der Datei eindeutig sein. Der Name muß mit einem Buchstaben beginnen und darf aus Buchstaben, Ziffern oder Unterstrichen (_) bestehen. Die ersten acht Zeichen des Namens sind signifikant.

anweisung
Eine oder mehrere SQL-Anweisungen, die für die dynamische Ausführung vorbereitet werden. *anweisung* kann eine in " oder ' eingeschlossene alphanumerische Konstante sein oder eine Hostvariable für alphanumerische Werte der jeweiligen Programmiersprache (siehe Handbuch für die Einbettung [3, 4, 6]).

Innerhalb von *anweisung* dürfen keine Hostvariablen verwendet werden. Statt dessen schreiben Sie jeweils als Platzhalter für einen noch unbekannten Eingabewert ein Fragezeichen (?). Den Fragezeichen weisen Sie später Werte mit einer EXECUTE-, OPEN- oder PUT-Anweisung zu. Fragezeichen können nicht als Platzhalter für die Namen von Datenbanken, Tabellen, Spalten und Benutzern verwendet werden.

Werden mehrere Anweisungen gleichzeitig mit PREPARE vorbereitet, müssen die einzelnen Anweisungen mit Strichpunkt (;) voneinander getrennt werden (siehe unten).

Wenn Sie eine SELECT-Anweisung vorbereiten, dürfen Sie der SELECT-Anweisung FOR UPDATE anhängen (die FOR UPDATE-Klausel finden Sie bei der DECLARE-Anweisung beschrieben).

SQL-Anweisungen für PREPARE

Folgende Anweisungen können einzeln oder zu mehreren mit PREPARE vorbereitet werden:

ALTER INDEX
ALTER TABLE
BEGIN WORK
COMMIT WORK
CREATE AUDIT
CREATE INDEX
CREATE SYNONYM
CREATE TABLE
CREATE VIEW
DELETE
DROP AUDIT
DROP INDEX
DROP SYNONYM
DROP TABLE
DROP VIEW
GRANT
INSERT
LOCK TABLE
RECOVER TABLE
RENAME COLUMN
RENAME TABLE
REVOKE
ROLLBACK WORK
ROLLFORWARD DATABASE
SET ISOLATION
SET LOCK MODE
SET LOG
START DATABASE
UNLOCK TABLE
UPDATE
UPDATE STATISTICS

DELETE, INSERT und UPDATE dürfen eine SELECT-Anweisung als Unterabfrage enthalten.

Die folgenden SQL-Anweisungen können nur einzeln mit PREPARE vorbereitet werden:

CLOSE DATABASE
CREATE DATABASE
DATABASE
DROP DATABASE
SELECT (darf die Klausel INTO *variable*,... nicht enthalten)

Folgende Anweisungen können nicht mit PREPARE vorbereitet werden:

CHECK TABLE
CLOSE
DECLARE
DESCRIBE
EXECUTE
EXECUTE IMMEDIATE
FETCH
FLUSH
FREE
und alle 4GL-Anweisungen
LOAD
OPEN
PREPARE
PUT
REPAIR TABLE
SELECT mit INTO *variable,...*
UNLOAD
WHENEVER

PREPARE und Satzzeiger

Eine mit PREPARE vorbereitete SELECT-Anweisung ohne INTO TEMP dürfen Sie nicht mit der Anweisung EXECUTE ausführen, sondern müssen mit DECLARE einen Satzzeiger dafür vereinbaren.

Da DECLARE für mit PREPARE vorbereitete SELECT-Anweisungen keine FOR UPDATE-Klausel zuläßt, müssen Sie die FOR UPDATE-Klausel bereits bei PREPARE angeben. Dies ist aber nur dann nötig, wenn Sie den aktuellen Satz verändern oder löschen wollen.

Mehrere SQL-Anweisungen in einem PREPARE

Ein PREPARE, der mehrere SQL-Anweisungen gleichzeitig auf die dynamische Ausführung vorbereitet, bewirkt nichts anderes, als mehrere PREPARE-Anweisungen, die die dynamischen SQL-Anweisungen einzeln vorbereiten.
Die Performance erhöht sich jedoch, wenn die Anzahl an PREPARE- und EXECUTE-Anweisungen reduziert wird, bei denen Anwendungsprogramm und Backend Informationen austauschen müssen.

Werden mehrere SQL-Anweisungen gleichzeitig mit PREPARE vorbereitet, so werden sie mit EXECUTE als Einheit ausgeführt und nicht sequentiell abgearbeitet. Es dürfen deshalb in einem PREPARE keine Anweisungen auftreten, die vom Ergebnis einer Anweisung des gleichen PREPARE abhängen.

Tritt bei einem PREPARE mit mehreren Anweisungen ein Fehler auf, so wird dies wie üblich in der Variablen SQLCA angezeigt. Es wird aber nur der erste aufgetretene Fehler gemeldet. Sie sind selbst dafür verantwortlich die fehlerhafte Anweisung zu identifizieren.
Es ist deshalb empfehlenswert, zuerst die dynamischen SQL-Anweisungen einzeln mit PREPARE vorzubereiten und erst dann mehrere SQL-Anweisungen zu einem PREPARE zusammenzufassen, wenn sie ausgetestet sind.

ANSI-Standard

Die Anweisung PREPARE ist im ANSI-Standard enthalten.

Beispiel 1 für INFORMIX-4GL

Im ersten 4GL-Beispiel wird die SQL-Anweisung direkt in PREPARE angegeben. Zwei Fragezeichen fungieren als Platzhalter für Werte, die zum Zeitpunkt des PREPARE nicht bekannt sind. Die Fragezeichen müssen später bei der OPEN-Anweisung mit Werten versorgt werden.

```
DEFINE knum INTEGER
DEFINE datum DATE

PREPARE suche_1 FROM "SELECT * FROM auftrag",
        "WHERE kunden_nr = ? AND auftragsdatum > ?"

DECLARE sel_zeiger CURSOR FOR suche_1

OPEN sel_zeiger USING knum, datum
```

Beispiel 2 für INFORMIX-ESQL/C

Das folgende ESQL/C-Beispiel bereitet mehrere SQL-Anweisungen gleichzeitig mit PREPARE vor. Die einzelnen Anweisungen sind durch Strichpunkt (;) getrennt.

```
$ char aenderung[300];
$ char h_neu[3], h_alt[3];

sprintf(h_neu, "%s","NEU");
sprintf(h_alt, "%s","ALT");
sprintf(aenderung, "%s, %s, %s, %s, %s, %s, %s, %s",
       "BEGIN WORK;",
       "UPDATE posten SET herstellercode = ? ",
       "WHERE herstellercode = ?;",
       "UPDATE artikel SET herstellercode = ? ",
       "WHERE herstellercode = ?;",
       "UPDATE hersteller SET herstellercode = ? ",
       "WHERE herstellercode = ?;",
       "COMMIT WORK;");
$ PREPARE aend_bez FROM $aenderung;
$ EXECUTE aend_bez USING $h_neu, $h_alt, $h_neu, $h_alt, $h_neu, $h_alt;
```

Beispiel 3 für INFORMIX-4GL

Das folgende Beispiel zeigt die gleichen Anweisungen wie bei Beispiel 2 als 4GL-Programmteil.

```
DEFINE h_neu, h_alt CHAR(3)
DEFINE aendern CHAR(300)

LET aendern = "BEGIN WORK;",
                "UPDATE posten SET herstellercode = ? ",
              "WHERE herstellercode = ?;",
              "UPDATE artikel SET herstellercode = ? ",
              "WHERE herstellercode = ?;",
              "UPDATE hersteller SET herstellercode = ? ",
              "WHERE herstellercode = ?;",
              "COMMIT WORK;"
PREPARE aend_bez FROM aendern
EXECUTE aend_bez USING h_neu, h_alt, h_neu, h_alt,
                       h_neu, h_alt
```

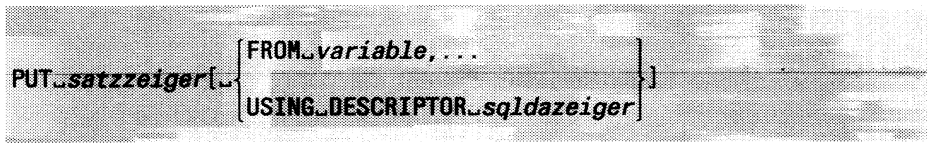
>>>> DECLARE, EXECUTE, FETCH, FOREACH (4GL), FREE, OPEN, PUT

PUT - Satz in Einfügepuffer schreiben

PUT schreibt einen Satz in den Einfügepuffer, der zum Insert-Satzzeiger gehört.

Vor dem Aufruf beachten

Satzzeiger sind nur in der Quellprogrammdatei ansprechbar, in der sie mit DECLARE vereinbart werden. Der mit PUT angesprochene Satzzeiger muß zuvor mit DECLARE für eine INSERT-Anweisung vereinbart und mit OPEN geöffnet werden.



satzzeiger
Name des Insert-Satzzeigers, in dessen Puffer die Sätze geschrieben werden.

FROM- und USING-Klausel:
Wenn die INSERT-Anweisung keine dynamische Anweisung ist, dürfen Sie die FROM- und USING-Klausel nicht angeben. Enthält die INSERT-Anweisung Hostvariablen, so werden die aktuellen Werte zum Zeitpunkt der PUT-Anweisung verwendet.
Sie müssen entweder die FROM- oder die USING-Klausel angeben, wenn die INSERT-Anweisung eine dynamische Anweisung mit Fragezeichen als Platzhalter für Werte ist.

FROM *variable,...*
Die Werte, die den Fragezeichen der dynamischen INSERT-Anweisung entsprechen, werden mit Hostvariablen angegeben. Die INSERT-Anweisung wurde mit PREPARE vorbereitet. Die Werte der Hostvariablen werden in der aufgeführten Reihenfolge den Fragezeichen der dynamischen INSERT-Anweisung zugewiesen. Die Anzahl der angegebenen Hostvariablen muß dabei der Anzahl der Fragezeichen entsprechen.

variable
Hostvariable, deren Wert einem Fragezeichen der dynamischen INSERT-Anweisung zugewiesen wird. Der Datentyp der Hostvariablen muß zum entsprechenden Wert passen, für den das Fragezeichen als Platzhalter in der INSERT-Anweisung steht.

USING DESCRIPTOR *sqldazeiger*
Die Werte für die Fragezeichen der dynamischen INSERT-Anweisung werden durch eine *sqlda*-Struktur angegeben.

Achtung:
Die USING DESCRIPTOR-Klausel können Sie nur bei der Einbettung in C-Programme verwenden.

sqldazeiger
Zeiger auf eine *sqlda*-Struktur, die auf die Werte zeigt, die den Fragezeichen der dynamischen INSERT-Anweisung zugewiesen werden. Genauere Informationen zur *sqlda*-Struktur finden Sie im Handbuch für die C-Einbettung [3].

Leeren des Puffers

Trifft eine PUT-Anweisung auf einen vollen Puffer, so wird der Pufferinhalt in die Datenbank eingefügt (implizites FLUSH). Im Puffer steht dann der mit PUT zugewiesene Satz.

Es gibt keine Variable, in der INFORMIX alle durch den Insert-Puffer eingefügten Sätze zählt. Daher sollten Sie, wenn Sie Informationen über die Anzahl der tatsächlich eingefügten Sätze haben wollen, selbst eine entsprechende Zählvariable vereinbaren und für jede PUT-Anweisung erhöhen.

Enthält der mit PUT übergebene Satz nur Konstanten, so wird er nicht gepuffert. INFORMIX zählt die Anzahl der einzufügenden "gleichen" Sätze und fügt diese nur bei einer folgenden FLUSH- bzw. CLOSE-Anweisung in die Tabelle ein.
Wird das Programm beendet, ohne daß vorher der Satzzeiger mit CLOSE geschlossen oder der Puffer mit FLUSH in die Datenbank geschrieben wurde, geht der Inhalt des Puffers verloren.

Erfolgskontrolle bei Programmeinbettung

	Positivfall	Fehlerfall
INFORMIX-4GL	status = 0 SQLCA.SQLERRD[3] = 0 der Satz wurde in den Puffer eingefügt. SQLCA.SQLERRD[3] > 0 ist die Anzahl der mit PUT in die Datenbank eingefügten Sätze.	status < 0 SQLCA.SQLERRD[3] ist die Anzahl der mit PUT erfolgreich eingefügten Sätze. Der Rest der Sätze ist verloren.
INFORMIX-ESQL/C	sqlca.sqlcode = 0 sqlca.sqlerrd[2] = 0 der Satz wurde in den Puffer eingefügt. sqlca.sqlerrd[2] > 0 ist die Anzahl der mit PUT in die Datenbank eingefügten Sätze.	sqlca.sqlcode < 0 sqlca.sqlerrd[2] ist die Anzahl der mit PUT erfolgreich eingefügten Sätze. Der Rest der Sätze ist verloren.
INFORMIX-ESQL/COBOL	SQLCODE OF SQLCA = 0 SQLERRD[3] = 0 der Satz wurde in den Puffer eingefügt. SQLERRD[3] > 0 ist die Anzahl der mit PUT in die Datenbank eingefügten Sätze.	SQLCODE OF SQLCA < 0 SQLERRD[3] ist die Anzahl der mit PUT erfolgreich eingefügten Sätze. Der Rest der Sätze ist verloren.

ANSI-Standard

Die Anweisung PUT ist nicht im ANSI-Standard enthalten.

Beispiel 1 für INFORMIX-4GL

Beim ersten Beispiel werden die Hostvariablen direkt in der INSERT-Anweisung angegeben.

```
DEFINE hcode CHAR(3)
DEFINE hname CHAR(15)

DECLARE inszeiger CURSOR FOR
        INSERT INTO hersteller VALUES (hcode, hname)

OPEN inszeiger

PUT inszeiger
     .
     .
     .
CLOSE inszeiger
```

Beispiel 2 für INFORMIX-4GL

Beim folgenden Beispiel wird eine dynamische INSERT-Anweisung zuerst mit PREPARE vorbereitet. Die Werte werden bei der PUT-Anweisung mit Hostvariablen übergeben.

```
DEFINE hcode CHAR(3)
DEFINE hname CHAR(15)

PREPARE ins_anw FROM
        "INSERT INTO hersteller VALUES (?,?)"

DECLARE inszeiger CURSOR FOR ins_anw

OPEN inszeiger

PUT inszeiger FROM hcode, hname
       .
       .
       .
CLOSE inszeiger
```

> > > > CLOSE, DECLARE, FLUSH, OPEN

RECOVER TABLE - Tabelle aktualisieren

Mit RECOVER TABLE können Sie die Sicherungskopie einer Tabelle mit dem Audit-Protokoll auf den aktuellen Stand bringen. Sie benötigen dazu

- eine Sicherungskopie der Tabelle, die Sie mit Betriebssystemmitteln erstellt haben und
- das Audit-Protokoll, das Sie unmittelbar vor Erstellen der Sicherungskopie erzeugt haben (siehe CREATE AUDIT).

Es wird empfohlen RECOVER TABLE nur bei Datenbanken ohne Transaktionssicherung zu verwenden. Bei Datenbanken mit Transaktionssicherung steht mit ROLLFORWARD DATABASE eine komfortablere Möglichkeit zur Verfügung.
Verwenden Sie entweder Audit- oder Transaktionsprotokolle zum Aktualisieren von Sicherungskopien. Eine Mischung dieser beiden Sicherungsverfahren kann zur Inkonsistenz der Daten führen.

Vor dem Aufruf beachten

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

RECOVER TABLE sperrt die Tabelle exklusiv gegen jeglichen Zugriff anderer Prozesse. RECOVER TABLE wird abgewiesen, wenn ein anderer Prozeß die Tabelle bearbeitet. Dies gilt auch dann, wenn nur lesend zugegriffen wird.

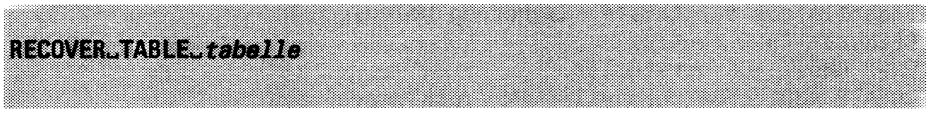


tabelle
Name oder Synonym der Basistabelle, die Sie aktualisieren wollen.

Arbeitsweise von RECOVER TABLE

RECOVER TABLE bricht ab, wenn das Audit-Protokoll nicht zur Sicherungskopie paßt. Das ist z.B. der Fall, wenn ein zu löschender oder zu ändernder Satz nicht mehr existiert oder ein einzulesender Satz in der Tabelle schon vorhanden ist.

Vorgehensweise

Nach dem Einspielen des Audit-Protokolls in die Tabelle sollten Sie folgendermaßen vorgehen:

1. Löschen Sie das eingespielte Audit-Protokoll (DROP AUDIT).
2. Erzeugen Sie ein neues Audit-Protokoll (CREATE AUDIT).
3. Sichern Sie die Tabelle erneut mit Betriebssystemmitteln auf Magnetband.

RECOVER TABLE und Transaktionen

Die Anweisung RECOVER TABLE kann nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

ANSI-Standard

Die Anweisung RECOVER TABLE ist nicht im ANSI-Standard enthalten.

Beispiel

Die folgenden Anweisungen zeigen den Verlauf einer Tabellenaktualisierung. Dabei wird vorausgesetzt, daß das Audit-Protokoll unmittelbar vor der letzten Sicherung erzeugt wurde.

```
{Tabelle aus der letzten Sicherung holen}
RECOVER TABLE hersteller;
DROP AUDIT FOR hersteller;
CREATE AUDIT FOR hersteller IN "/usr1/save/herst.aud"
{Aktualisierte Tabelle wieder sichern}
```

>>>> CREATE AUDIT, DROP AUDIT

RENAME COLUMN - Spaltennamen ändern

RENAME COLUMN ändert den Namen einer Tabellenspalte.

Vor dem Aufruf beachten

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich entweder das Alter-Zugriffsrecht haben oder aber Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

```
RENAME␣COLUMN␣tabelle.spaltenname␣TO␣spaltenname
```

tabelle
Name oder Synonym der Basistabelle, deren Spalte umbenannt werden soll. Sie dürfen keine Systemtabelle angeben.

spaltenname
Name der Spalte, die umbenannt werden soll.

TO *spaltenname*
Neuer Name der Spalte. Der Name darf aus max. 18 Zeichen bestehen und Buchstaben, Ziffern und Unterstriche (_) enthalten. Das erste Zeichen muß ein Buchstabe sein. Der Name der Spalte muß innerhalb der Tabelle eindeutig sein.

Auswirkung von RENAME COLUMN

RENAME COLUMN wirkt nicht auf Views, die auf der Tabelle definiert sind. Die Spaltennamen dieser Views bleiben unverändert.

Nach Ausführung von RENAME COLUMN müssen Sie alle INFORMIX-Programme ändern, die die umbenannte Spalte verwenden.

RENAME COLUMN in Transaktionen

Die Anweisung RENAME COLUMN kann bei einer INFORMIX-SE-Datenbank nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

INFORMIX-ONLINE kann die Anweisung RENAME COLUMN zurücksetzen.

ANSI-Standard

Die Anweisung RENAME COLUMN ist nicht im ANSI-Standard enthalten.

Beispiel

Im folgenden Beispiel werden zuerst mit der INFO-Anweisung die Spaltennamen der Tabelle *hersteller* ausgegeben. Dann wird der Name der Spalte *herstellercode* geändert und die geänderten Spaltennamen erneut mit der INFO-Anweisung ausgegeben.

```
INFO COLUMNS FOR hersteller
```

```
Spaltenname          Typ                                    NULL's

herstellercode       char(3)                                yes
herstellername       char(15)                               yes
```

```
RENAME COLUMN hersteller.herstellercode TO hcode;
INFO COLUMNS FOR hersteller
```

```
Spaltenname          Typ                                    NULL's

hcode                char(3)                                yes
herstellername       char(15)                               yes
```

>>>> ALTER TABLE, CREATE TABLE, DROP TABLE, INSERT, RENAME TABLE

RENAME TABLE - Tabellennamen ändern

RENAME TABLE ändert den Namen einer Basistabelle.

Vor dem Aufruf beachten

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich entweder das Alter-Zugriffsrecht haben oder aber Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

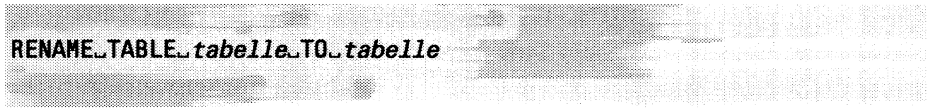


tabelle
Name der Basistabelle, die umbenannt werden soll. Sie dürfen kein Synonym und keine Systemtabelle angeben.

TO *tabelle*
Neuer Name der Tabelle. Der Name darf aus max. 18 Zeichen bestehen und Buchstaben, Ziffern und Unterstriche (_) enthalten. Das erste Zeichen muß ein Buchstabe sein. Bei einer Nicht-ANSI-Datenbank darf der Name der Tabelle nicht mit einem bereits existierenden Tabellen-, View- oder Synonymnamen der aktuellen Datenbank übereinstimmen. Bei ANSI-Datenbanken muß der Name unter allen bereits existierenden Tabellen-, View- oder Synonymnamen desselben Datenbankbenutzers eindeutig sein.

Der Eigentümer einer Tabelle kann mit RENAME TABLE nicht geändert werden, der neue Tabellenname darf deshalb nicht mit dem Eigentümer qualifiziert werden.

Auswirkungen von RENAME TABLE

Bei Synonymen erfolgt die Anpassung an den neuen Tabellennamen automatisch.

Dagegen müssen Sie bei allen Views, die auf der Tabelle definiert sind, den Tabellennamen selbst ändern. Sie müssen zuerst mit DROP VIEW die alte View-Definition löschen und anschließend mit CREATE VIEW den View neu definieren, wobei Sie den neuen Tabellennamen angeben.

Außerdem müssen Sie nach Ausführung der Anweisung RENAME TABLE alle INFORMIX-Programme ändern, die den alten Tabellennamen verwenden.

Bei INFORMIX-SE werden die Namen der Dateien, die zu der umbenannten Tabelle gehören, automatisch an den neuen Tabellennamen angepaßt, unabhängig davon, ob sie im Dateiverzeichnis *datenbank.dbs* der Datenbank oder in einem anderen Dateiverzeichnis liegen.

RENAME TABLE in Transaktionen

Die Anweisung RENAME TABLE kann bei einer INFORMIX-SE-Datenbank nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

INFORMIX-ONLINE kann die Anweisung RENAME TABLE zurücksetzen.

ANSI-Standard

Die Anweisung RENAME TABLE ist nicht im ANSI-Standard enthalten.

> > > > ALTER TABLE, CREATE TABLE, DROP TABLE, INSERT, RENAME COLUMN

REPAIR TABLE - Index für Tabelle wiederherstellen

REPAIR TABLE behebt Widersprüche zwischen den Sätzen einerTabelle und den zugehörigen Indexinformationen. Die Indexinformationen werden dabei an die Tabellensätze angepaßt. Prüfen Sie zuerst mit CHECK TABLE, ob eine Wiederherstellung der Indexinformationen benötigt wird.

Vor dem Aufruf beachten

REPAIR TABLE kann nur für eine INFORMIX-SE-Datenbank verwendet werden.

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.

REPAIR TABLE sperrt die Tabelle exklusiv gegen jeglichen Zugriff anderer Prozesse. REPAIR TABLE wird abgewiesen, wenn ein anderer Prozeß die Tabelle bearbeitet. Dies gilt auch dann, wenn nur lesend zugegriffen wird.

tabelle
Name der Basistabelle, die Sie wiederherstellen wollen. Die Angabe eines Synonyms oder der Systemtabelle *systables* ist hier nicht erlaubt.

Arbeitsweise von REPAIR TABLE

REPAIR TABLE ruft intern das Dienstprogramm BCHECK auf (siehe Handbuch für das jeweilige Frontend-Produkt [1, 3, 4, 6]). REPAIR TABLE *tabelle* ist gleichbedeutend mit folgendem Aufruf auf Betriebssystemebene:

```
bcheck -j tabnnn.
```

ANSI-Standard

Die Anweisung REPAIR TABLE ist nicht im ANSI-Standard enthalten.

>>>> CHECK TABLE

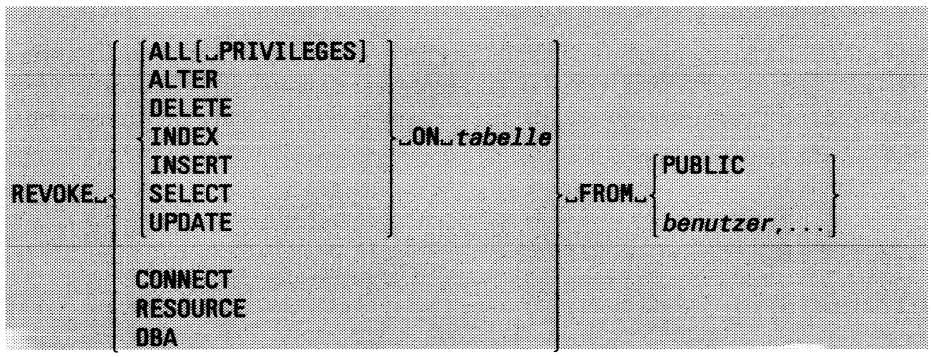
REVOKE - Zugriffsrechte entziehen

REVOKE entzieht Benutzern Datenbank- oder Tabellenzugriffsrechte.

Vor dem Aufruf beachten

Tabellenzugriffsrechte können einem Benutzer nur von dem Benutzer entzogen werden, der das Tabellenzugriffsrecht vergeben hat (siehe auch GRANT-Anweisung).

Datenbankzugriffsrechte können einem Benutzer nur durch einen Benutzer mit DBA-Zugriffsrecht entzogen werden.



ALL[PRIVILEGES]
: Alle Tabellenzugriffsrechte werden entzogen. ALL PRIVILEGES umfaßt die Tabellenzugriffsrechte Alter, Delete, Index, Insert, Select und Update.

ALTER
: Das Tabellenzugriffsrecht zum Ändern der Tabellenstruktur wird entzogen.

DELETE
: Das Tabellenzugriffsrecht zum Löschen von Sätzen der Tabelle wird entzogen.

INDEX
: Das Tabellenzugriffsrecht zum Erzeugen eines Index für die Tabelle wird entzogen.

INSERT
: Das Tabellenzugriffsrecht zum Neuaufnehmen von Sätzen in die Tabelle wird entzogen.

SELECT
Das Tabellenzugriffsrecht zum Lesen von Sätzen aus der Tabelle wird entzogen. Das Select-Zugriffsrecht kann nur insgesamt und nicht einzelnen Spalten entzogen werden.

UPDATE
Das Tabellenzugriffsrecht zum Ändern von Sätzen in der Tabelle wird entzogen. Das Update-Zugriffsrecht kann nur insgesamt und nicht einzelnen Spalten entzogen werden.

tabelle
Name oder Synonym der Tabelle, für die Sie die Zugriffsrechte entziehen wollen. Die Tabelle kann eine Basistabelle oder ein View sein.

CONNECT
Der Zugriff zur Datenbank wird entzogen.

RESOURCE
Das Recht Tabellen und Indizes zu erzeugen wird entzogen. Der Benutzer erhält stattdessen das Connect-Zugriffsrecht.

DBA
Das DBA-Zugriffsrecht wird entzogen. Der Benutzer erhält stattdessen das Connect-Zugriffsrecht.

PUBLIC
Alle angegebenen Zugriffsrechte werden der Allgemeinheit entzogen. Wurden Zugriffsrechte mit GRANT an einzelne Benutzerkennungen vergeben, so bleiben diese Rechte bestehen.

benutzer
Benutzerkennung, der die angegebenen Zugriffsrechte entzogen werden sollen. Ein an PUBLIC vergebenen Zugriffsrecht kann nicht einer einzelnen Benutzerkennung entzogen werden. Wenn Sie Ihre eigene Benutzerkennung angeben, wird REVOKE ignoriert. Ein Benutzer kann sich selbst keine Zugriffsrechte entziehen.

Arbeitsweise von REVOKE

Sie können das Select- oder Update-Zugriffsrecht nicht für einzelne Spalten entziehen. Sie müssen in diesem Fall zuerst das Select- bzw. Update-Zugriffsrecht entziehen und dann mit GRANT die gewünschten Zugriffsrechte auf die einzelnen Spalten neu vergeben.

Wollen Sie mit REVOKE einem Benutzer nur die Grant-Berechtigung nicht aber die Tabellenzugriffsrechte entziehen, so müssen Sie zuerst die entsprechenden Tabellenzugriffsrechte entziehen und dann mit der Anweisung GRANT genau dieselben Tabellenzugriffsrechte wieder vergeben, aber diesmal ohne die Klausel WITH GRANT OPTION.

REVOKE in Transaktionen

Die Anweisung REVOKE kann bei einer INFORMIX-SE-Datenbank nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

INFORMIX-ONLINE kann die Anweisung REVOKE zurücksetzen.

ANSI-Standard

Die Anweisung REVOKE ist nicht im ANSI-Standard enthalten.

Beispiel 1

Tabellenzugriffsrechte können Sie im Dialog mit INFO abfragen. Im folgenden Beispiel gibt die INFO-Anweisung zuerst die aktuellen Tabellen-Zugriffsrechte für die Tabelle *kunde* aus. Die anschließenden REVOKE-Anweisungen haben folgende Wirkung:

- dem Benutzer *usera* wird das Update-Zugriffsrecht entzogen
- dem Benutzer *userb* wird das Delete-Zugriffsrecht entzogen
- dem Benutzer *boss* werden alle Tabellenzugriffsrechte entzogen

```
INFO PRIVILEGES FOR kunde
```

Benutzer	Select	Update	Insert	Delete	Index	Alter
boss	Alle	Alle	Ja	Ja	Ja	Ja
usera	Alle	Alle	Ja	Ja	Ja	Nein
userb	Alle	Alle	Ja	Ja	Ja	Nein

```
REVOKE UPDATE ON kunde FROM usera;
REVOKE DELETE ON kunde FROM userb;
REVOKE ALL ON kunde FROM boss;
INFO PRIVILEGES FOR kunde
```

```
Benutzer     Select          Update          Insert  Delete  Index  Alter
usera         Alle            Keine           Ja      Ja      Ja     Nein
userb         Alle            Alle            Ja      Nein    Ja     Nein
```

Beispiel 2

Datenbankzugriffsrechte erfragen Sie über die Systemtabelle *sysusers*. Im folgenden Beispiel werden dann folgende Änderungen der Datenbankzugriffsrechte vorgenommen:

- Der Benutzer *usera* darf nicht mehr auf die aktuelle Datenbank zugreifen.
- Benutzer *userb* wird das Resource-Zugriffsrecht entzogen. Er hat danach automatisch Connect-Zugriffsrecht, darf also noch auf die Datenbank zugreifen. Um diesem Benutzer alle Rechte zu entziehen, müßte man anschließend noch das Connect-Zugriffsrecht aufheben.
- Benutzer *boss* wird das DBA-Zugriffsrecht entzogen. Er hat danach automatisch Connect-Zugriffsrecht.

```
SELECT username, usertype FROM sysusers
```

```
username usertype
lomata   D
boss     D
usera    C
userb    R
```

```
REVOKE CONNECT FROM usera;
REVOKE RESOURCE FROM userb;
REVOKE DBA FROM boss;
SELECT username, usertype FROM sysusers
```

```
username usertype
lomata   D
boss     C
userb    C
```

>>>> CREATE TABLE, GRANT

ROLLBACK WORK - Transaktion zurücksetzen

ROLLBACK WORK beendet eine Transaktion und setzt die Änderungen zurück, die seit Beginn der Transaktion durchgeführt wurden. ROLLBACK WORK wird abgewiesen, wenn keine Transaktion offen ist.
Bei einer ANSI-Datenbank setzt ROLLBACK WORK eine Transaktion zurück und eröffnet implizit die nächste Transaktion.

Vor dem Aufruf beachten

ROLLBACK WORK ist nur erlaubt, wenn die Transaktionssicherung für die Datenbank eingeschaltet ist. Die Transaktionssicherung wird entweder mit CREATE DATABASE beim Erzeugen der Datenbank oder danach mit START DATABASE oder vom INFORMIX-ONLINE-Verwalter eingeschaltet.

```
ROLLBACK WORK
```

Welche Anweisungen können mit ROLLBACK WORK zurückgesetzt werden?

Bei einer INFORMIX-SE-Datenbank können folgende Datenänderungsanweisungen zurückgesetzt werden:

INSERT
UPDATE
DELETE
LOAD

Bei einer INFORMIX-ONLINE-Datenbank können alle Anweisungen außer CREATE DATABASE und DROP DATABASE zurückgesetzt werden.

Auswirkung von ROLLBACK WORK auf Satzzeiger

ROLLBACK WORK schließt alle offenen Satzzeiger, außer solchen, die mit WITH HOLD definiert wurden.

Ein Satzzeiger, der für eine SELECT-Anweisung vereinbart wurde, ist nicht mehr mit der Ergebnistabelle verbunden. Nach ROLLBACK WORK ist nur eine folgende OPEN-Anweisung für diesen Satzzeiger sinnvoll und möglich.

Wird ROLLBACK WORK in einem 4GL-Programm innerhalb einer FOREACH-Schleife verwendet, so muß der Satzzeiger mit WITH HOLD definiert sein. Sonst wird der Satzzeiger geschlossen und beim nächsten Schleifendurchlauf tritt ein Fehler auf.

Auswirkung von ROLLBACK WORK auf Sperren

ROLLBACK WORK hebt alle Einzelsperren auf, die während der Transaktion gesetzt wurden. Tabellensperren, die Sie mit LOCK TABLE gesetzt haben, werden ebenfalls aufgehoben.

Weitere Informationen über Transaktionen und Sperren finden Sie in Kapitel 2.

ROLLBACK WORK in Fehlerbehandlungsroutine

Wenn Sie ROLLBACK WORK innerhalb einer Fehlerbehandlungsroutine verwenden, die mit WHENEVER aufgerufen wird, dann sollten Sie in dieser Routine vor der Anweisung ROLLBACK WORK die folgenden beiden Anweisungen stehen haben:

```
WHENEVER ERROR CONTINUE
WHENEVER WARNING CONTINUE
```

Sie vermeiden dadurch eine Endlosschleife, falls ROLLBACK WORK selbst auf einen Fehler oder eine Warnung läuft. Es wird empfohlen, die Fehlerroutine ans Dateiende zu setzen.

ROLLBACK WORK im Audit-Protokoll

Wird zusätzlich zur Transaktionsprotokollierung noch die Audit-Protokollierung benutzt, wird das Zurücksetzen der Transaktion auch im Audit-Protokoll gespeichert.

ANSI-Standard

ROLLBACK WORK ist im ANSI-Standard enthalten. Bei einer ANSI-Datenbank setzt ROLLBACK WORK eine Transaktion zurück und eröffnet implizit die nächste Transaktion.

Beispiel

Im folgenden Beispiel wird eine Transaktion gestartet, in der mit INSERT ein Satz in die Tabelle *hersteller* aufgenommen wird. Da die Transaktion mit ROLLBACK WORK beendet wird, wird der Satz nicht in die Tabelle eingefügt.
Wird anstelle des ROLLBACK WORK ein COMMIT WORK gegeben, so bleibt der INSERT erhalten.

```
BEGIN WORK;
INSERT INTO hersteller (herstellercode,herstellername)
       VALUES ("NEU","neusport");

SELECT * FROM hersteller;
```

```
herstellercode herstellername
SMT            Schmitt
ANZ            Anzabel
NRG            Norgesta
HSK            Hasken
HRO            Herold
NEU            neusport
```

```
ROLLBACK WORK;

SELECT * FROM hersteller;
```

```
herstellercode herstellername
SMT            Schmitt
ANZ            Anzabel
NRG            Norgesta
HSK            Hasken
HRO            Herold
```

> > > > BEGIN WORK, COMMIT WORK, LOCK TABLE

ROLLFORWARD DATABASE - Datenbanksicherung aktualisieren

Mit ROLLFORWARD DATABASE können Sie die Sicherungskopie einer Datenbank mit einer Transaktionsprotokoll-Datei auf den aktuellen Stand bringen. Sie benötigen dazu

- eine Sicherungskopie der Datenbank und
- das Transaktionsprotokoll, das Sie unmittelbar vor dem Erstellen der Sicherungskopie der Datenbank erzeugt haben.

Vor dem Aufruf beachten

ROLLFORWARD DATABASE wird für die aktuelle Datenbank abgewiesen. Sie müssen die aktuelle Datenbank zuerst mit CLOSE DATABASE schließen.

Sie müssen DBA-Zugriffsrecht für die Datenbank haben.

ROLLFORWARD DATABASE sperrt die Tabelle exklusiv gegen jeglichen Zugriff anderer Prozesse. ROLLFORWARD DATABASE wird abgewiesen, wenn ein anderer Prozeß die Tabelle bearbeitet. Dies gilt auch dann, wenn nur lesend zugegriffen wird.

datenbank
Name der Datenbank, die aktualisiert werden soll. Diese Datenbank ist nach ROLLFORWARD DATABASE die aktuelle Datenbank, bleibt aber für andere Prozesse gesperrt, bis Sie CLOSE DATABASE ausführen.

Vorgehensweise

Nach dem Einspielen des Transaktionsprotokolls in die Sicherungskopie der Datenbank sollten prinzipiell folgende Schritte durchgeführt werden:

1. Erzeugen Sie ein neues Transaktionsprotokoll (START DATABASE). Der Dateiname des neuen Transaktionsprotokolls darf nicht identisch sein mit dem Dateinamen des alten Transaktionsprotokolls.
2. Schließen Sie die Datenbank mit CLOSE DATABASE.
3. Sperren Sie die Datenbank exklusiv mit DATABASE EXCLUSIV.

4. Sichern Sie die Datenbank mit Betriebssystemmitteln.
5. Schließen Sie die Datenbank mit CLOSE DATABASE.

Danach können Sie die alte Transaktionsprotokoll-Datei mit einem Betriebssystemkommando löschen.
Wenn Sie den Dateinamen des Transaktionsprotokolls nicht ändern wollen, müssen Sie anstatt Schritt 1 lediglich folgendes Betriebssystemkommando eingeben:

```
cat /dev/null > alter-dateiname
```

Dieses Kommando richtet eine neue, leere Transaktionsprotokoll-Datei ein.

ANSI-Standard

Die Anweisung ROLLFORWARD DATABASE ist nicht im ANSI-Standard enthalten.

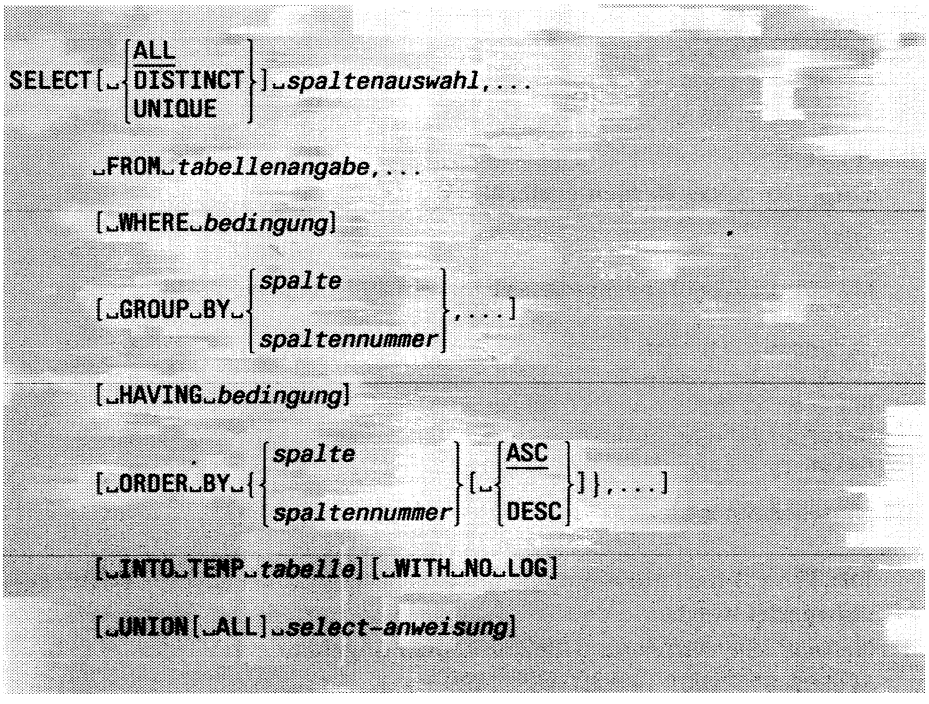
>>>> BEGIN WORK, COMMIT WORK, ROLLBACK WORK, START DATABASE

SELECT - Daten abfragen

SELECT wählt Daten aus Tabellen aus und liefert bei Erfolg eine Ergebnistabelle zurück.

Vor dem Aufruf beachten

Besitzen Sie das CONNECT-Zugriffsrecht, so müssen Sie zusätzlich entweder Eigentümer der Tabelle sein oder das SELECT-Zugriffsrecht für jede verwendete Spalte besitzen.
Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.



Für alle Klauseln gilt:

- Die angegebene Reihenfolge der Klauseln muß eingehalten werden.
- Spaltennamen müssen eindeutig sein. Kommt ein Spaltenname in mehreren Tabellen vor, so müssen Sie ihn mit dem Tabellennamen qualifizieren. Wenn Sie eine Tabelle mit einer *referenz* für die Dauer der SELECT-Anweisung umbenennen (siehe FROM), dürfen Sie nur noch die Umbenennung verwenden.

Beispiel:

```
SELECT K.vorname, auftrags_nr
FROM kunde K, auftrag
ORDER BY K.vorname
```

ALL

Doppelte Sätze in der Ergebnistabelle bleiben erhalten. ALL ist Voreinstellung.

Beispiel:

```
SELECT nachname, vorname FROM person
```

nachname	vorname
Maier	Hima
Maier	Hugo
Maier	Hugo
Maier	Karl

DISTINCT

Doppelte Sätze werden entfernt.

Beispiel:

```
SELECT DISTINCT nachname, vorname FROM person
```

nachname	vorname
Maier	Hima
Maier	Hugo
Maier	Karl

Einschränkung:

DISTINCT darf nur einmal in **derselben** Ebene einer SELECT-Abfrage verwendet werden.
Beispiel: Sie dürfen zum Beispiel nicht angeben:

```
SELECT DISTINCT  COUNT(DISTINCT ...) ...
```

UNIQUE

Gleichbedeutend mit DISTINCT; doppelte Sätze werden entfernt.

Sperren beim Lesen

INFORMIX-SE

Es werden keine Sperren beim Lesen gesetzt.

INFORMIX-ONLINE

Beim Lesen wird abhängig von der Isolationsstufe eine Lesesperre gesetzt. Es gibt vier Isolationsstufen mit unterschiedlichem Sicherheitsgrad. Die ausführliche Beschreibung finden Sie in Kapitel 2, Abschnitt 2.11 *Sperren*. Diese Isolationsstufen vom niedrigsten bis zum höchsten Sicherheitsgrad sind:

- Dirty Read:
  Es werden keine fremden Sperren berücksichtigt. Das gilt auch für fremde Exclusive-Sperren. Es werden keine eigenen Share-Sperren gesetzt.
- Committed Read:
  Für Datenbank mit Transaktionssicherung
  Fremde Exclusive-Sperren werden berücksichtigt, indem versucht wird, eine eigene Share-Sperre zu setzen. Ist dies möglich, bedeutet das, daß keine Exclusive-Sperre besteht und die Daten werden gelesen ohne eine eigene Share-Sperre zu setzen.
- Cursor Stability:
  Für Datenbank mit Transaktionssicherung
  Fremde Exclusive-Sperren werden berücksichtigt. Zusätzlich wird eine eigene Share-Sperre auf den Satz gesetzt, der gelesen werden soll.
- Repeatable Read:
  Für Datenbank mit Transaktionssicherung
  Fremde Exclusive-Sperren werden berücksichtigt. Zusätzlich wird eine eigene Share-Sperre auf alle Sätze gesetzt, die mit der SELECT-Anweisung innerhalb einer Transaktion gelesen werden.

Welche Isolationsstufe voreingestellt ist, ist abhängig davon, ob es eine NICHT-ANSI-Datenbank oder ANSI-Datenbank ist und ob für die Datenbank Transaktionssicherung eingestellt ist. Es gibt folgende Möglichkeiten:

- Nicht-ANSI-Datenbank
  - Ohne Transaktionssicherung: immer Dirty Read
  - Mit Transaktionssicherung: voreingestellt Committed Read
- ANSI-Datenbank: voreingestellt Repeatable Read

Die Isolationsstufe kann mit der Anweisung SET ISOLATION eingestellt werden.

Wenn Sie mit einer Isolationsstufe arbeiten, die Sperren beim Lesen setzt (insbesondere Repeatable Read), müssen Sie berücksichtigen, daß die maximal zulässige Anzahl an Sperren überschritten werden kann. Die Anzahl der möglichen Sperren ist bei bei INFORMIX-ONLINE durch Shared Memory Parameter beschränkt.

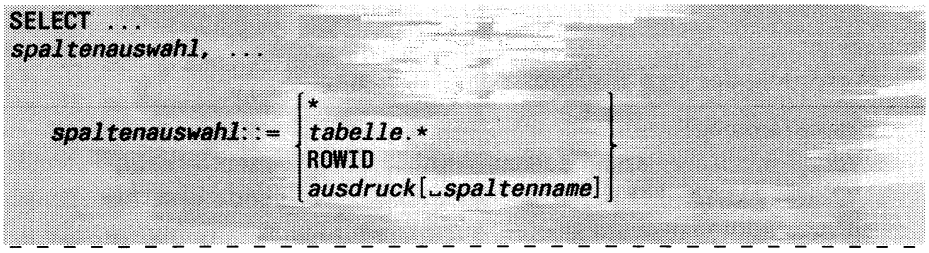
Sie sollten in diesem Fall die Anzahl der auszuwählenden Sätze reduzieren oder die ganze Tabelle vor der Ausführung der SELECT-Anweisung sperren (siehe LOCK TABLE).

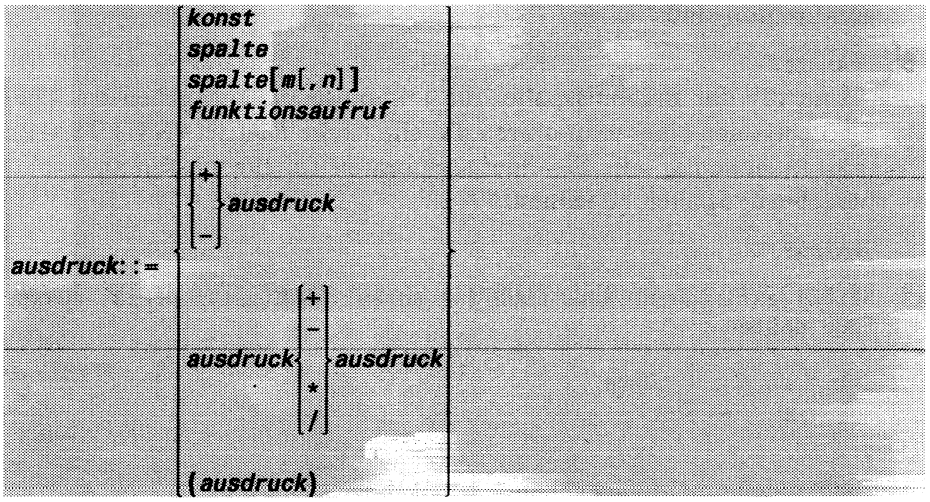
ANSI-Standard

Die Anweisung SELECT ist im ANSI-Standard enthalten. Die Klausel INTO TEMP und die Angabe von UNIQUE, OUTER und GROUP BY mit *spaltennummer* ist nicht im ANSI-Standard enthalten.

SELECT/Spaltenauswahl - Ergebnisspalten auswählen

Mit der Spaltenauswahl legen Sie die Ergebnisspalten fest.





Beachten Sie, daß ein Ausdruck in der Spaltenauswahl keine Unterabfrage sein darf.

* Alle Spalten auswählen.
Die Reihenfolge und die Namen der Spalten der in der FROM-Klausel angegebenen Tabellen werden übernommen. Bei mehreren Tabellen gilt die Reihenfolge der Tabellen in der FROM-Klausel (siehe SELECT/FROM: Ergebnis bei mehreren Tabellen).

tabelle.*
Alle Spalten der Tabelle *tabelle* auswählen. Die Reihenfolge und die Namen der Spalten von *tabelle* werden übernommen.

ROWID

Ergebnisspalte für Satzadresse hinzufügen. Jedem Satz in einer Tabelle ist eine feste Satzadresse zugeordnet, die den Satz eindeutig identifiziert. Wenn Sie ROWID angeben, erhält die Ergebnistabelle eine Spalte, in der die Satzadresse des ausgewählten Satzes eingetragen ist.

ausdruck

Ausdruck, der eine Ergebnisspalte bezeichnet.

Einschränkung:
Wenn in einer Spaltenausauswahl eine Mengenfunktion (AVG, COUNT, MAX, MIN, SUM) vorkommt, gilt folgende Einschränkung: In der Spaltenauswahl dürfen nur Spaltennamen vorkommen, die in der GROUP BY-Klausel aufgeführt sind oder Argument einer Mengenfunktion sind.

spaltenname

Name für die Ergebnisspalte, die mit *ausdruck* angegeben ist. Die Spaltennamen einer temporären Tabelle (siehe Klausel INTO TEMP) müssen eindeutig sein. Bei einer temporären Tabelle müssen Sie daher berechnete Spalten benennen.

Beispiel:

```
SELECT auftrags_nr anr, COUNT(*) anzahl, SUM(gesamtpreis) summe
FROM posten
```

anr	anzahl	summe

. . .

spaltenname nicht angegeben:
Wenn *ausdruck* ein Spaltenname ist, erhält die Ergebnisspalte denselben Namen. Ansonsten vergibt INFORMIX im interaktiven Betrieb einen Namen.

Beispiel:

```
SELECT auftrags_nr, COUNT(*)
FROM posten
```

auftrags_nr	(count(*))

. . .

Spalten der Ergebnistabelle

Die Reihenfolge der Spalten in der Ergebnistabelle entspricht der Reihenfolge der Spalten in der Spaltenauswahl.

Die Attribute einer Ergebnisspalte (Datentyp, Länge, Genauigkeit, Nachkommastellen) werden entweder von der zugrundeliegenden Spalte übernommen oder ergeben sich aus dem angegebenen Ausdruck.

SELECT/FROM - Tabellen angeben

In der FROM-Klausel geben Sie die Tabellen an, aus denen Daten ausgewählt werden sollen.

INFORMIX-ONLINE

Sie können Daten aus externen Tabellen abfragen.

Die Datenbank, zu der die externe Tabelle gehört, ist eine andere Datenbank als die aktuelle Datenbank. Die andere Datenbank kann sein:

- Eine Datenbank in demselben ONLINE-System
- Nur mit INFORMIX-STAR
  - Eine Datenbank in einem anderen ONLINE-System auf demselben Rechner
  - Eine Datenbank in einem ONLINE-System auf einem anderen Rechner

```
SELECT ...

FROM tabellenangabe,...

                          ⎡tabelle[ referenz]                ⎤
   tabellenangabe::= {    ⎢OUTER tabelle[ referenz]           ⎥ }
                          ⎣OUTER (tabellenangabe,...)        ⎦
```

tabelle

Name oder Synonym einer Basistabelle oder eines View oder Name einer temporären Tabelle.

Dieselbe Tabelle kann mehrmals in der FROM-Klausel vorkommen. In diesem Fall müssen Sie Referenzen verwenden.

INFORMIX-ONLINE:
Für eine externe Tabelle geben Sie den qualifizierten Namen an (siehe Kapitel 3, Abschnitt 3.2.5 *Qualifizierte Namen*). Liegt die Datenbank mit der externen Tabelle in demselben Informixsystem, qualifizieren Sie den Tabellennamen mit dem einfachen Namen der Datenbank. Liegt die Datenbank mit der externen Tabelle in einem anderen Informixsystem, qualifizieren Sie den Tabellennamen mit dem qualifizierten Namen der Datenbank.

referenz

Tabellenname, der für die Dauer der SELECT-Anweisung eine Umbenennung für die Tabelle *tabelle* ist.
Bei jeder Spaltenangabe, die sich auf diese Angabe von *tabelle* bezieht, müssen Sie den Spaltennamen mit dem neuen Namen *referenz* qualifizieren.

Der neue Name muß eindeutig sein, d.h. *referenz* darf nur einmal in einer FROM-Klausel dieser SELECT-Anweisung vorkommen. Sie **können** eine Tabelle umbenennen, um z.B. wie bei Synonymen durch sprechende Namen Ihre SELECT-Anweisung verständlicher zu formulieren oder um lange Namen abzukürzen.
Sie **müssen** eine Tabelle umbenennen, wenn innerhalb einer SELECT-Anweisung dieselbe Tabelle mehrfach verwendet wird, d.h. *tabelle* mehrmals in FROM-Klauseln vorkommt.

Beispiel:

Tabelle mit sich selbst verknüpfen

```
SELECT A.nachname B.nachname          /* Kunden abfragen, die in */
FROM kunde A, kunde B
WHERE A.ort = B.ort                   /* demselben Ort wohnen */
AND A.knr < B.knr                     /* und gleiche Paare vermeiden */
```

(siehe auch SELECT/WHERE: Unterabfragen)

OUTER

OUTER-Operator zum Bilden eines Outer-Join.
Bei einem Outer-Join gibt es eine dominante Tabelle und abhängige Tabellen. Die hinter OUTER genannte Tabelle ist eine abhängige Tabelle.

tabelle1, OUTER *tabelle2*

tabelle1 ist die dominante Tabelle, *tabelle2* ist die abhängige Tabelle.

Für die Tabellen eines Outer-Join muß eine Join-Bedingung angegeben werden.

Gibt es für Werte in der Join-Spalte der dominante Tabelle keinen übereinstimmenden Wert in der abhängigen Tabelle, bleibt der Satz trotzdem erhalten. Für die nicht vorhandenen Werte der abhängigen Tabelle wird der NULL-Wert eingesetzt.

Beispiel:

Aus den Tabellen *kunde* und *auftrag* der Beispieldatenbank *versand* sollen Kundenname und zugehörige Auftragsnummer herausgesucht werden und zwar für alle Kunden, auch solche, die noch keinen Auftrag gestellt haben:

```
SELECT nachname, auftrags_nr
  FROM kunde, OUTER auftrag
  WHERE kunde.kunden_nr=auftrag.kunden_nr
```

Kunden, die keinen Auftrag erteilt haben, zum Beispiel Kunde *Korres* mit Nummer 103, sind in der Ergebnistabelle enthalten. Für die fehlende Auftragsnummer ist der NULL-Wert eingetragen.

```
nachname  | auftrgs_nr
----------+-----------
Pauli            1002
Sadler
Korres
. . .
```

Sie können OUTER mehrmals in *tabellenangabe* angeben und zusammengesetzte (Outer-)Joins bilden.
Beispiele für mögliche Kombinationen sind:

tabelle1, OUTER *tabelle2*, OUTER *tabelle3*
Linearer Outer-Join.
tabelle1 ist die dominante Tabelle für *tabelle2* und *tabelle3*.

tabelle1, OUTER (*tabelle2*, *tabelle3*)
Geschachtelter Outer-Join mit innerem normalen Join.
tabelle1 ist die dominante Tabelle.
Die Ergebnistabelle des Join von *tabelle2* und *tabelle3* ist die abhängige Tabelle.

tabelle1, OUTER (*tabelle2*, OUTER *tabelle3*)
Hierarchischer Outer-Join.
tabelle1 ist die dominante Tabelle für die Ergebnistabelle des Outer-Join von *tabelle2* und *tabelle3*, bei dem *tabelle2* die dominante Tabelle für *tabelle3* ist.

In Kapitel 5, Abschnitt 5.6 *Join* sind Outer-Joins ausführlich beschrieben und für die aufgeführten Kombinationen Beispiele angegeben.

Einschränkung:
Wird OUTER mehrmals auf der **derselben** Klammerebene einer SELECT-Abfrage verwendet, handelt es sich um einen linearen Outer-Join. In diesem Fall muß in jeder Teil-Join-Bedingung immer dieselbe Join-Spalte der dominanten Tabelle verwendet werden.

Ergebnis bei mehreren Tabellen

Wenn Sie mehrere Tabellen angeben und keine einschränkenden Auswahl-Bedingungen, dann ist die Ergebnistabelle das Kartesische Produkt aus den angegebenen Tabellen:
Jeder Satz einer Tabelle wird mit jedem Satz der anderen Tabellen verknüpft.

Beispiel:

Jedem Auto aus der Tabelle *luxusauto* alle Teile aus der Tabelle *ausstattung* zuordnen:

```
Tabelle: luxusauto
```

anr	atyp
01	saab1
02	saab2

```
Tabelle: ausstattung
```

nr	name
10	radio
20	telefon
30	leder

```
SELECT *
FROM luxusauto, ausstattung
```

anr	atyp	nr	name
01	saab1	10	radio
01	saab1	20	telefon
01	saab1	30	leder
02	saab2	10	radio
02	saab2	20	telefon
02	saab3	30	leder

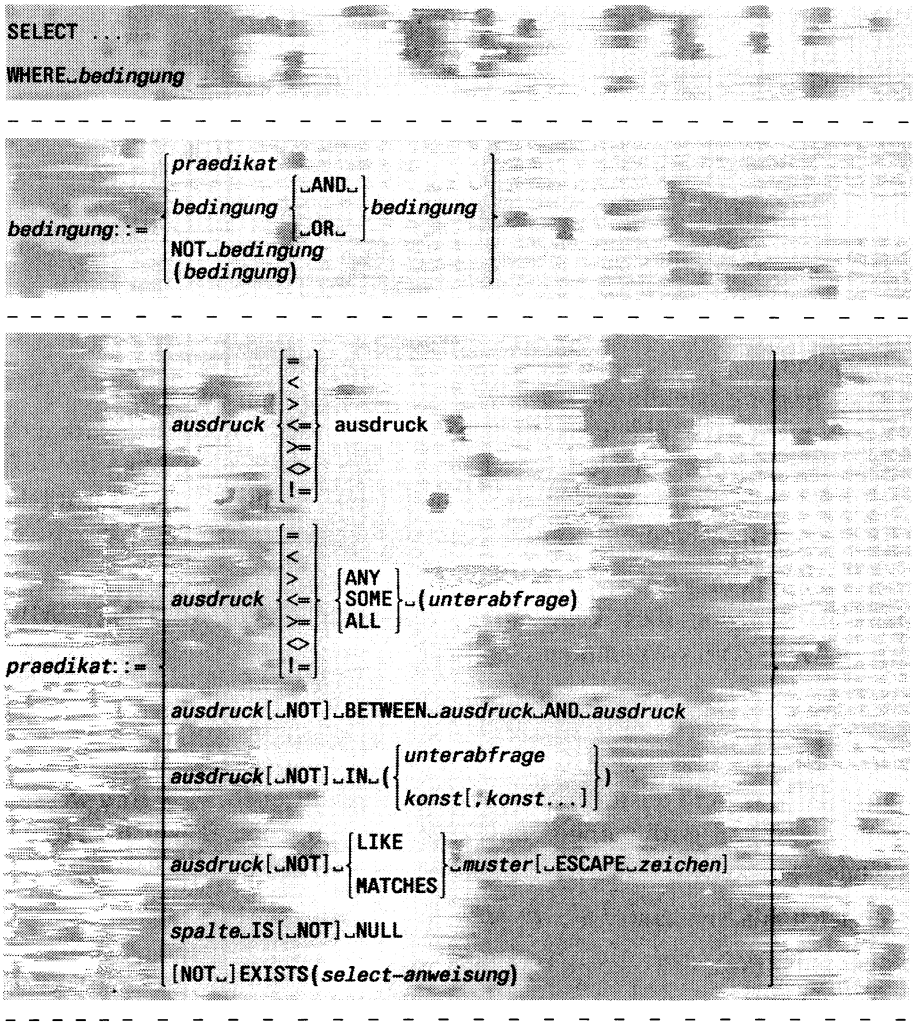
Im allgemeinen ist das vollständige Kartesische Produkt uninteressant, da viele inhaltlich aussagelose Kombinationen entstehen.

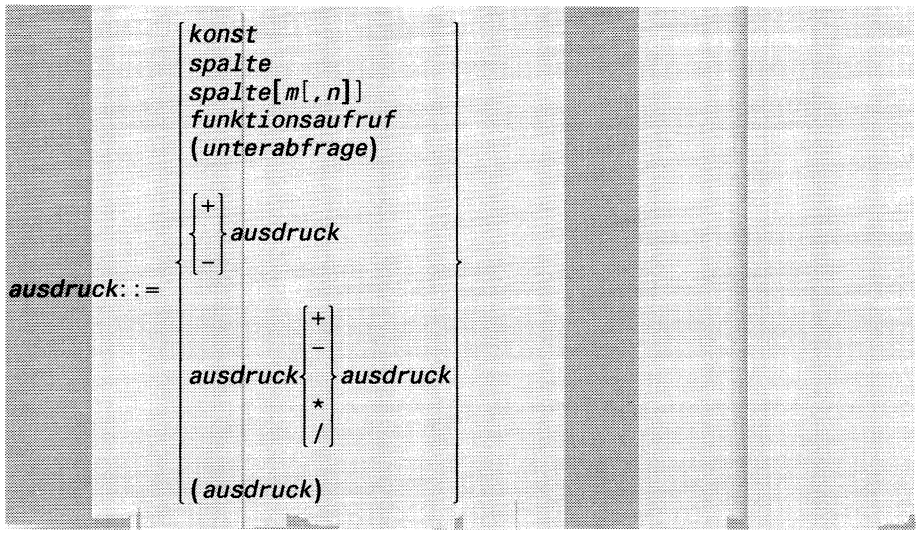
In den meisten Fällen wird daher die Ergebnistabelle eingeschränkt, wobei sehr häufig Joins verwendet werden. Durch Festlegung geeigneter Join-Spalten und Angabe der Join-Bedingung in der WHERE-Klausel kann eine eingeschränkte Ergebnistabelle erzeugt werden, deren Sätze sinnvolle Kombinationen ergeben.

Joins sind ausführlich in Kapitel 5, Abschnitt 5.6 *Join* beschrieben.

SELECT/WHERE - Ergebnissätze auswählen

In der WHERE-Klausel geben Sie Bedingungen an, um Sätze für die Ergebnistabelle auszuwählen. Die Ergebnistabelle enthält nur die Sätze, die die angegebenen Bedingungen erfüllen.





bedingung

Bedingung, die die auszuwählenden Sätze erfüllen müssen.
Bei mehreren Tabellen können Sie insbesondere eine Join-Bedingung angeben, um sinnvolle Kombinationen aus dem Kartesischen Produkt dieser Tabellen auszuwählen. Bedingungen, Prädikate und Ausdrücke sind ausführlich in in Kapitel 5 beschrieben.

Einschränkung:
In *bedingung* darf keine Mengenfunktion (AVG, COUNT, MIN, MAX, SUM) vorkommen.

Beispiel

Die Prädikate sind ausführlich in Kapitel 5 beschrieben. Hier sind wesentliche Arten von Bedingungen an Hand von einfachen Beispielen zusammengestellt.

Vergleich mit Konstante: =, <, <=, >, >=, <>, !=

```
SELECT artikel.nr
FROM artikel
WHERE preis > 3000
```

Vergleich mit Zeichenketten-Muster: [NOT] LIKE, [NOT] MATCHES

```
SELECT *
FROM kunde
WHERE nachname LIKE "H%"
```

Bereichsabfrage: BETWEEN, NOT BETWEEN

```
SELECT nachname
FROM kunde
WHERE nachname BETWEEN "BA" AND "Bz"
```

Vergleich auf NULL-Wert: IS NULL, IS NOT NULL

```
SELECT posten_nr
FROM posten
WHERE herstellercode IS NULL
```

Vergleich auf mehrere Werte: IN, NOT IN

```
SELECT artikel_nr
FROM artikel
WHERE preis IN (145,152,257)
```

Innere SELECT-Anweisung: [NOT] EXISTS

```
SELECT kunde.nachname
FROM kunde
WHERE EXISTS
     ( SELECT *
       FROM auftrag
       WHERE auftrag.kunden_nr= kunde.kunden_nr)
```

Unterabfrage:

Unterabfrage, die eine Ergebnisspalte liefert: ALL, ANY, SOME, [NOT] IN

```
SELECT nachname
FROM kunde
WHERE kunde.kunden_nr =  SOME
      (SELECT kunden_nr
       FROM auftrag
       WHERE lieferdatum = "10.2.90")
```

Korrelierte Unterabfrage:

Für jeden Auftrag die Posten heraussuchen, deren Gesamtpreis mindestens das Doppelte beträgt, wie der billigste Posten dieses Auftrags:

```
SELECT auftrags_nr, artikel_nr, herstellercode, gesamtpreis
       FROM posten x
       WHERE gesamtpreis
       > (SELECT 2 * MIN (gesamtpreis)
              FROM posten
              WHERE auftrags_nr = x.auftrags_nr)
```

Bedingung: AND, OR, NOT

```
SELECT posten_nr
FROM posten
WHERE herstellercode NOT LIKE "A*" AND gesamtpreis < 10000
```

Join:

Join über drei Tabellen:
Welcher Kunde hat welche Posten bestellt?

```
SELECT nachname, posten_nr
   FROM kunde, auftrag, posten
   WHERE kunde.kunden_nr=auftrag.kunden_nr
   AND auftrag.auftrags_nr=posten.auftrags_nr
```

Outer-Join:

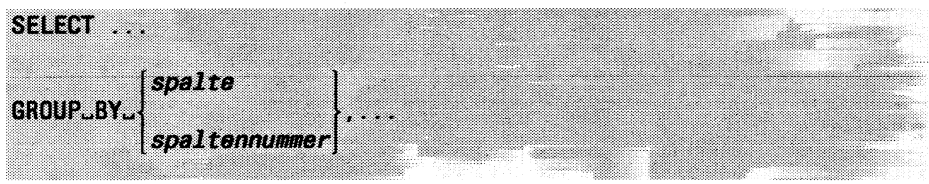
Hersteller mit zugehöriger Artikelbezeichnung heraussuchen, auch für Hersteller, die noch keinen Artikel in der Artikeltabelle haben:

```
SELECT h.herstellercode, a.bezeichnung
  FROM hersteller h, outer artikel a
  WHERE h.herstellercode=a.herstellercode
```

SELECT/GROUP BY - Ergebnissätze gruppieren

Mit Hilfe der GROUP BY-Klausel werden Tabellensätze in Gruppen zusammengefaßt. Die Ergebnistabelle enthält dann für jede Gruppe einen Satz. Alle NULL-Werte in einer Spalte werden als gleich betrachtet und bilden daher eine Gruppe.

Wenn Sie die GROUP BY-Klausel angeben, wirken Mengenfunktionen gruppenweise (siehe Beispiel unten und Kapitel 5, Abschnitt 5.1 *Funktionen*).



spalte
Spalte, die ein Gruppierungsmerkmal angibt. Denken Sie daran, daß Sie Spaltennamen gegebenenfalls qualifizieren müssen, wobei Sie den neuen Tabellennamen verwenden, wenn Sie die Tabelle in der FROM-Klausel umbenannt haben. Die Reihenfolge der Spaltenangaben hat keine Bedeutung.

spaltennummer
Positionsnummer einer Ergebnisspalte.
$1 \leq$ *spaltennummer* $\leq$ Anzahl der Ergebnisspalten.
Die Ergebnisspalten sind entsprechend den Angaben in der Spaltenauswahl von links nach rechts, beginnend mit 1, durchnumeriert. Auf diese Weise können Sie auch Spalten als Gruppierungsmerkmal aufnehmen, die keinen Spaltennamen haben.

Einschränkung

Wenn Sie die GROUP BY-Klausel angeben, dürfen in der Spaltenauswahl nur noch die Spaltennamen vorkommen, die bei GROUP BY aufgeführt oder Argument einer Mengenfunktion sind.

Die Spalten dürfen nicht vom Datentyp TEXT und BYTE sein.

Wie werden die Gruppen gebildet?

Die Sätze, die in allen angegebenen Spalten den gleichen Wert haben, bilden eine Gruppe. Sätze, die in diesen Spalten den NULL-Wert enthalten, bilden eine eigene Gruppe.
Wenn Sie Ergebnissätze in Gruppen zusammenfassen, wirken Mengenfunktionen gruppenweise.

Beispiel:

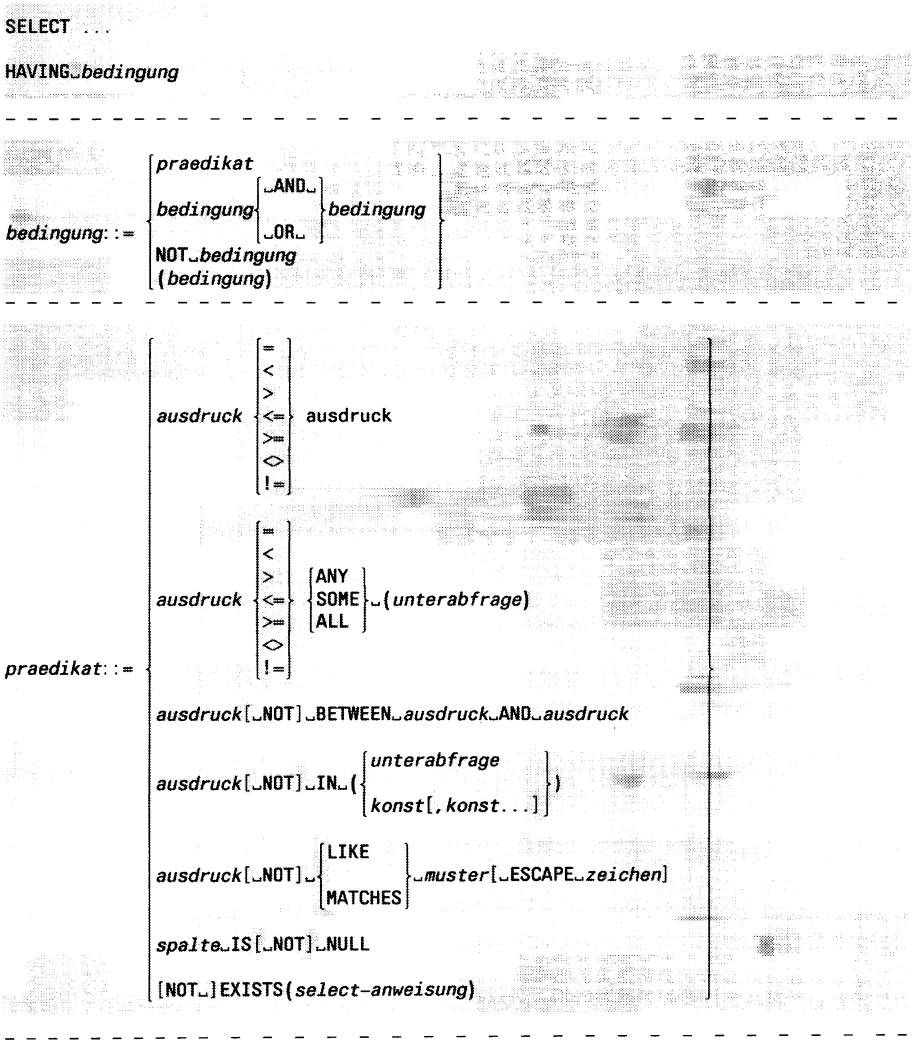
Für jede Auftragsnummer die Summe der Gesamtpreise bilden:

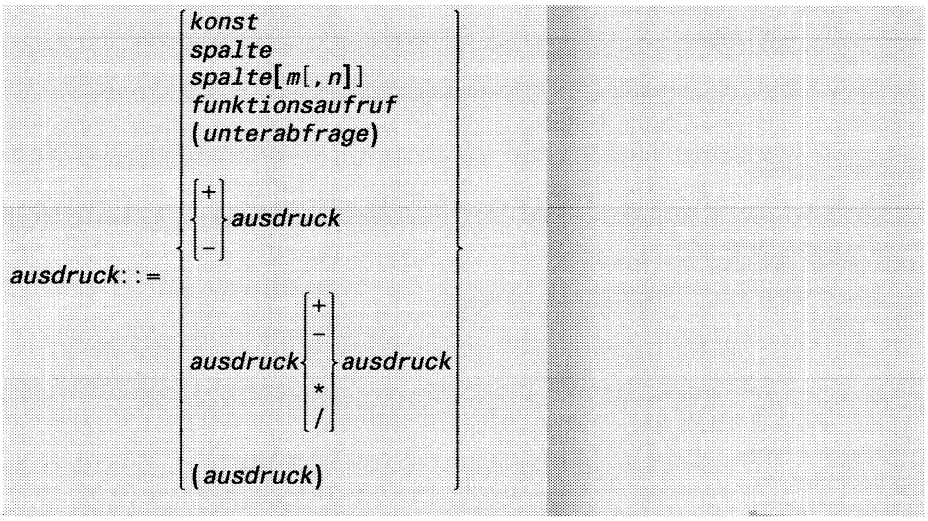
```
SELECT auftrags_nr, SUM(gesamtpreis)
FROM posten
GROUP BY (auftrags_nr)
```

```
auftrags_nr     (sum)
       1001     DM250.00
       1002     DM1200.00
       1003     DM959.00
       1004     DM2126.00
       . . .
```

SELECT/HAVING - Gruppen auswählen

In der HAVING-Klausel geben Sie Bedingungen an, um Gruppen auszuwählen. Die Ergebnistabelle enthält den Satz für eine Gruppe, wenn die Gruppe die angegebene Bedingung erfüllt.





bedingung
Bedingung, die eine Gruppe erfüllen muß. Ist keine GROUP BY-Klausel angegeben, gelten alle Sätze als eine Gruppe.
Im Unterschied zur WHERE-Bedingung dürfen Sie in der HAVING-Bedingung Mengenfunktionen angeben.

Einschränkung

Ein Spaltenname darf in der HAVING-Klausel nur vorkommen, wenn gilt:

- Der Spaltenname ist in der GROUP BY-Klausel aufgeführt.
- Der Spaltenname ist Argument einer Mengenfunktion (AVG(), SUM(), ...). Wenn der Spaltenname auch in der Spaltenauswahl erscheint, darf er dort auch nur als Argument einer Mengenfunktion vorkommen.

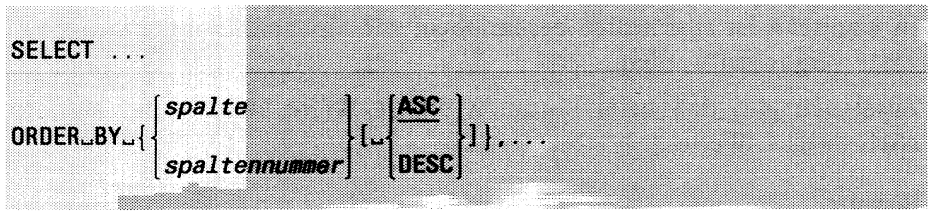
Beispiel:

Für jede Artikelnummer, die größer als 100 ist, die Summe der Gesamtpreise bilden:

```
SELECT SUM(gesamtpreis)
FROM posten
GROUP BY artikel_nr
HAVING artikel_nr > 100
```

SELECT/ORDER BY - Ergebnistabelle sortieren

In der ORDER BY-Klausel geben Sie Spalten an, nach denen die Ergebnistabelle sortiert werden soll.



spalte
Spaltenname der Spalte, nach der sortiert werden soll.
Die Spalte muß in der Spaltenauswahl vorkommen. Denken Sie daran, daß Sie Spaltennamen gegebenenfalls qualifizieren müssen, wobei Sie den neuen Namen verwenden, wenn Sie die Tabelle in der FROM-Klausel umbenannt haben.

spaltennummer
Spaltennummer der Spalte, nach der sortiert werden soll:
$1 \leq$ *spaltennummer* $\leq$ Anzahl der Ergebnisspalten.
Die Ergebnisspalten sind entsprechend den Angaben in der Spaltenauswahl von links nach rechts beginnend mit 1 durchnumeriert. Auf diese Weise können Sie auch Spalten als Sortierbegriff verwenden, die keinen Spaltennamen haben.

ASC
Werte aufsteigend sortieren. ASC ist Voreinstellung.

DESC
Werte absteigend sortieren.

Einschränkung

Die Spalten dürfen nicht vom Datentyp TEXT und BYTE sein.

Wie wird sortiert?

Es wird zuerst nach den Werten der ersten angegebenen Spalte sortiert. Wenn Sätze mit gleichen Werten in der Spalte vorkommen, werden diese gemäß der zweiten Sortierspalte sortiert usw. bis die Sätze eindeutig sind oder keine Sortierspalten mehr vorliegen.

In Kapitel 4 ist ausführlich beschrieben, wie alphanumerische und Zeitwerte verglichen werden.

Zusätzlich gilt, daß beim Sortieren NULL-Werte kleiner sind als alle Nicht-NULL-Werte.

Beispiel

Aus der Tabelle *artikel* der Beispieldatenbank *versand* die Artikelbezeichnungen mit Herstellercode heraussuchen; die Ausgabe soll nach den Artikelbezeichnungen sortiert sein:

```
SELECT bezeichnung, herstellercode
FROM artikel
ORDER BY bezeichnung
```

```
bezeichnung         herstellercode

Basketball                     HRO
Fussball                       HRO
Fussball                       HSK
Ski-Brille                     HRO
Ski-Handschuhe                 SMT
Ski-Handschuhe                 HSK
Ski-Handschuhe                 HRO
Ski-Stock                      HSK
. . .
```

Um die Sortierreihenfolge bei NULL-Werten zu sehen, tragen Sie in die Tabelle folgenden Satz mit unbekanntem Herstellercode ein:

```
INSERT INTO artikel (bezeichnung, preis)
VALUES ("Ski-Handschuhe", 450.00)
```

Da Bezeichnungen wie zum Beispiel *Ski-Handschuhe* mehrfach vorkommen, soll die Ausgabe auch noch nach dem Herstellercode sortiert werden:

```
SELECT bezeichnung, herstellercode
FROM artikel
ORDER BY bezeichnung, herstellercode
```

Die Herstellercodes sind jetzt auch sortiert. Unterschiede zum Ergebnis oben sind gerastert. Der NULL-Wert bei den Herstellercodes von *Ski-Handschuhe* wird als niedrigster Wert einsortiert.

```
bezeichnung          herstellercode
Basketball                      HRO
Fussball                        HRO
Fussball                        HSK
Ski-Brille                      HRO
Ski-Handschuhe
Ski-Handschuhe                  HRO
Ski-Handschuhe                  HSK
Ski-Handschuhe                  SMT
Ski-Stock                       HSK
. . .
```

SELECT/INTO TEMP - Ergebnistabelle benennen

Mit der INTO TEMP-Klausel können Sie eine benannte temporäre Ergebnistabelle erzeugen. Die temporäre Tabelle existiert, bis Sie sie mit der Anweisung DROP TABLE löschen oder die INFORMIX-Sitzung beenden. Auf eine temporäre Tabelle können Sie auch zugreifen, wenn Sie die Datenbank wechseln. Die Datenbank muß auf demselben Rechner sein.

Eine temporäre Tabelle kann nicht qualifiziert werden und Sie dürfen kein Synonym definieren. Die Spalten einer temporären Tabelle dürfen immer NULL-Werte enthalten.

```
SELECT ...
INTO␣TEMP␣tabelle[␣WITH␣NO␣LOG]
```

tabelle

Name der temporären Tabelle. Der Name darf aus max. 18 Zeichen bestehen und Buchstaben, Ziffern und Unterstriche (_) enthalten. Das erste Zeichen muß ein Buchstabe sein. Bei einer Nicht-ANSI-Datenbank darf der Name der Tabelle nicht mit einem bereits existierenden Tabellen-, View- oder Synonymnamen der aktuellen Datenbank übereinstimmen. Für ANSI-Datenbanken muß der Name unter allen bereits existierenden Tabellen-, View- oder Synonymnamen desselben Benutzers eindeutig sein.

Eigentümer der Tabelle wird der zur Ablaufzeit aktuelle Prozeß.

Einschränkung:

- Jede Ergebnisspalte muß einen Namen haben. Wenn Sie Ausdrücke als Ergebnisspalten angegeben haben, müssen Sie diese Ausdrücke mit einem Spaltennamen benennen (siehe SELECT/ Spaltenauswahl).
- INTO TEMP dürfen Sie nicht zusammen mit ORDER BY angeben.

WITH NO LOG

Schaltet die Transaktionsprotokollierung für die temporäre Tabelle aus. Wurde die Transaktionsprotokollierung ausgeschaltet, so kann sie nicht mehr eingeschaltet werden.

WITH NO LOG kann auch für eine ANSI-Datenbank verwendet werden.

WITH NO LOG nicht angegeben:
Auch temporäre Tabellen unterliegen der Transaktionsprotokollierung, wenn die Datenbank mit Transaktionssicherung definiert wurde.

SELECT/UNION - SELECT-Anweisungen verbinden

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

```
SELECT ...
UNION [ ALL ] select-anweisung
```

ALL
Doppelte Sätze in der Ergebnistabelle bleiben erhalten.

ALL nicht angegeben:
Doppelte Sätze werden entfernt.

select-anweisung
SELECT-Anweisung.
Die Spaltennamen, die sich aus der Spaltenauswahl der ersten SELECT-Anweisung ergeben, bilden die Spaltennamen der gesamten Ergebnistabelle.

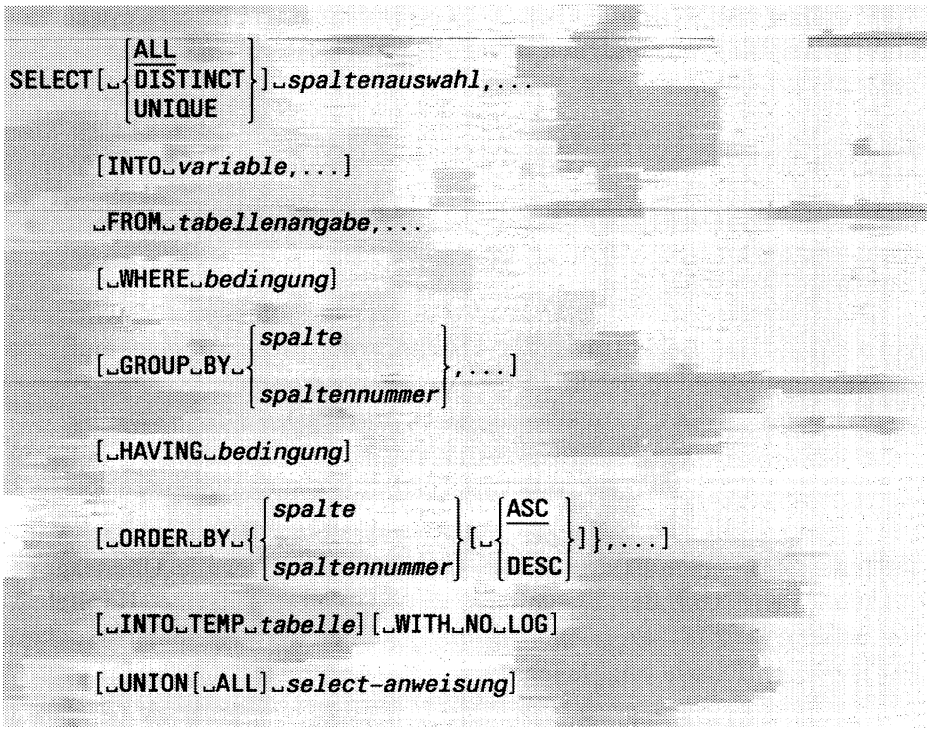
Einschränkung:

- Die Spaltenauswahl muß in allen mit UNION verbundenen SELECT-Anweisungen zusammenpassen, d.h. Anzahl und Datentyp der ausgewählten Spalten müssen in allen SELECT-Anweisungen übereinstimmen.
- Nur in der letzten SELECT-Anweisung darf die ORDER BY-Klausel verwendet werden. Dabei darf der Sortierbegriff nur in Form von Spaltennummern angegeben sein. Diese Nummern sind die Positionsangaben der Spalten der Ergebnistabelle, die als Ergebnis aller verknüpften Abfragen entsteht.
- Nur in der letzten SELECT-Anweisung darf die INTO TEMP-Klausel verwendet werden.

>>>> DECLARE, INSERT, DELETE, UPDATE

SELECT (eingebettet)

Bei Programmeinbettung sind alle Klauseln der SELECT-Anweisung erlaubt, die im interaktiven Betrieb definiert sind. Zusätzlich gibt es die INTO-Klausel, die im folgenden beschrieben wird.

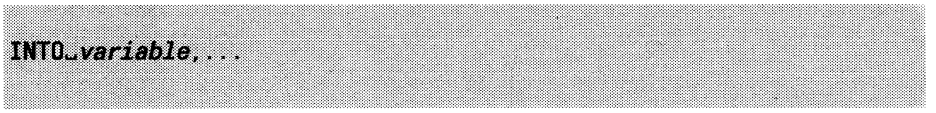


Die Klauseln INTO *variable*,... und INTO TEMP dürfen nicht gleichzeitig verwendet werden.

SELECT/INTO - Satz in Variable ausgeben

Die INTO-Klausel dient dazu, eine Ergebnistabelle in Variablen auszugeben. Kann die Ergebnistabelle mehrere Sätze enthalten, müssen Sie mit DECLARE einen Satzzeiger definieren, wobei Sie die SELECT-Anweisung angeben.

Die Klausel INTO *variable* darf auch in der ersten SELECT-Anweisung von mit UNION verbundenen SELECT-Anweisungen vorkommen.



variable
Hostvariable, in die die Spaltenwerte des Ergebnissatzes geschrieben werden. Die Anzahl und der Datentyp der Hostvariablen muß zur Spaltenauswahl in der SELECT-Anweisung passen.
Wird die SELECT-Anweisung in einer DECLARE-Anweisung verwendet, darf *variable* kein Array-Element sein. Sie müssen in diesem Fall das Array-Element in der FETCH-Anweisung angeben.

variable kann eine mit einer Indikatorvariable kombinierte Hostvariable sein. Die Indikatorvariable gibt Auskunft über die Übertragung der Werte von der Datenbank in die Variable (siehe Kapitel 2, Abschnitt 2.13 *Programmeinbettung*). Sie wird automatisch von INFORMIX wie folgt belegt:

0	Die Hostvariable enthält einen definierten Wert. Die Zuweisung war fehlerfrei.
< 0	Der Wert, der zugewiesen werden sollte, ist der NULL-Wert.
> 0	Bei alphanumerischen Werten: Es wurde eine Zeichenkette zugewiesen, die verkürzt wurde. Der Wert gibt die Originallänge an.

SELECT als dynamische Anweisung

SELECT ... INTO *variable* kann nicht dynamisch ausgeführt werden. Alle anderen SELECT-Anweisungen können einzeln mit PREPARE für eine dynamische Ausführung vorbereitet werden.

Eine mit PREPARE vorbereitete SELECT-Anweisung mit INTO TEMP wird mit EXECUTE ausgeführt.

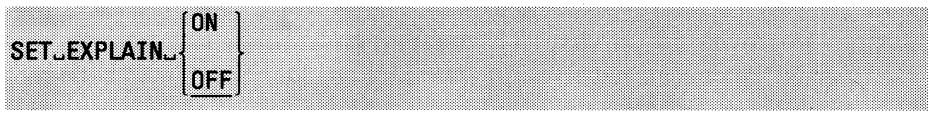
Eine mit PREPARE vorbereitete SELECT-Anweisung ohne INTO TEMP dürfen Sie nicht mit EXECUTE ausführen, sondern Sie müssen mit DECLARE einen Satzzeiger dafür vereinbaren. Der Satzzeiger wird mit OPEN geöffnet und die Sätze der Ergebnistabelle mit FETCH gelesen.

Nähere Beschreibung von dynamischen Anweisungen siehe Kapitel 2, Abschnitt 2.13, *Programmeinbettung*.

> > > > DECLARE, EXECUTE, FETCH, OPEN, PREPARE

SET EXPLAIN - Suchstrategie ausgeben

SET EXPLAIN ON schaltet die Ausgabe der Suchstrategie in eine Datei ein. Die Suchstrategie aller nachfolgenden SELECT-Anweisungen wird in eine Datei ausgegeben, bis die Ausgabe mit SET EXPLAIN OFF wieder ausgeschaltet wird.



ON
Die Ausgabe der Suchstrategie in die Datei *sqexplain.out* wird eingeschaltet. Die Datei wird im aktuellen Dateiverzeichnis angelegt. Existiert die Datei bereits im aktuellen Dateiverzeichnis, wird die Ausgabe angehängt.

Achtung:
Befindet sich Ihre aktuelle Datenbank auf einem anderem Rechner, wird die Datei *sqexplain.out* im HOME-Dateiverzeichnis des Benutzers auf dem anderen Rechner angelegt.

OFF
Die Ausgabe der Suchstrategie wird ausgeschaltet. OFF ist Standard.

Ausgabe der Suchstrategie

Folgende Daten werden ausgegeben:

- geschätzte Kosten der SELECT-Anweisung aus der Anzahl der Plattenzugriffe und der Anzahl der verarbeiteten Sätze
- geschätzte Anzahl Sätze
- Reihenfolge der Bearbeitung der Tabellen
- für jede Tabelle wird ausgegeben:
  - die Zugriffsart:

    SEQUENTIAL SCAN
    Sequentielle Suche in der Tabelle.

    INDEX PATH
    Suche über permanenten Index.

AUTOINDEX SCAN
Suche über temporär erzeugten Index.

– die Spalten, die für die Auswahl benutzt werden

ANSI-Standard

Die Anweisung SET EXPLAIN ist nicht im ANSI-Standard enthalten.

Beispiel

Im folgenden Beispiel wird die Suchstrategie von drei SELECT-Anweisungen ausgegeben.

Inhalt der Datei *sqexplain.out*:

```
QUERY:
------
SELECT vorname, nachname, firma FROM kunde;

Estimated Cost: 2
Estimated # of Rows Returned: 10

1) lomata.kunde: SEQUENTIAL SCAN


QUERY:
------
SELECT vorname, nachname, firma FROM kunde
   WHERE firma MATCHES "Sport*" AND
   kunden_nr BETWEEN 110 AND 115
   ORDER BY nachname;

Estimated Cost: 8
Estimated # of Rows Returned: 1
Temporary Files Required For: Order By

1) lomata.kunde: INDEX PATH

    Filters: lomata.kunde.firma MATCHES 'Sport*'

    (1) Index Keys: kunden_nr
        Lower Index Filter: lomata.kunde.kunden_nr >= 110
        Upper Index Filter: lomata.kunde.kunden_nr <= 115
```

```
QUERY:
------
SELECT * FROM kunde, auftrag, posten
   WHERE kunde.kunden_nr = auftrag.kunden_nr
   AND auftrag.auftrags_nr= posten.auftrags_nr;

Estimated Cost: 68
Estimated # of Rows Returned: 12

1) lomata.posten: SEQUENTIAL SCAN

2) lomata.auftrag: INDEX PATH

    (1) Index Keys: auftrags_nr
        Lower Index Filter: lomata.auftrag.auftrags_nr = lomata.posten.auftrags_nr

3) lomata.kunde: INDEX PATH

    (1) Index Keys: kunden_nr
        Lower Index Filter: lomata.kunde.kunden_nr = lomata.auftrag.kunden_nr
```

> > > > SELECT

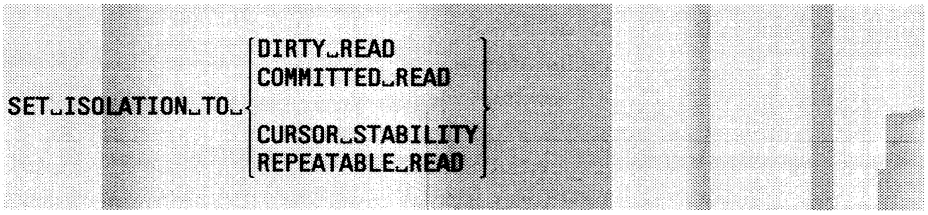
SET ISOLATION - Isolationsstufen wählen

SET ISOLATION bestimmt die Stufe der Isolation zwischen Prozessen, die gleichzeitig auf bestimmte Sätze lesend zugreifen wollen. Mit SET ISOLATION können Sie nicht das Setzen von Exclusive-Sperren zum Schreiben beeinflussen.

Die Isolationsstufe gilt bis zur nächsten Anweisung SET ISOLATION bzw. bis zum Programmende.

Vor dem Aufruf beachten

Diese Anweisung können Sie nur verwenden, wenn Ihre Datenbank unter INFORMIX-ONLINE arbeitet und die Transaktionssicherung eingeschaltet ist.



DIRTY READ

Der Prozeß erhält lesenden Zugriff auf einen Satz auch dann, wenn dieser durch einen anderen Prozeß exklusiv gesperrt ist. Es ist daher möglich, daß sogenannte "Phantomsätze" gelesen werden können. Phantomsätze treten z.B. auf, wenn ein Benutzer Sätze innerhalb einer Transaktion einfügt und die Transaktion am Ende zurücksetzt. Hat ein anderer Benutzer die eingefügten Sätze inzwischen gelesen, arbeitet er auf "Phantomdaten". Bei der Isolationsstufe Dirty Read setzt der Prozeß selbst keine Sperre.

Dirty Read ist die Isolationsstufe für Datenbanken ohne Transaktionssicherung.

COMMITTED READ

Der Prozeß erhält lesenden Zugriff auf einen Satz, sofern dieser durch keinen anderen Prozeß exklusiv gesperrt ist. Bei der Isolationsstufe Committed Read prüft der Prozeß, ob er Share-Sperre auf den Satz setzen könnte. Dadurch können keine "Phantomsätze" gelesen werden, der Satz kann aber, nachdem er gelesen wurde, von einem anderen Prozeß verändert werden.

Committed Read ist die voreingestellte Isolationsstufe für Nicht-

ANSI-Datenbanken mit Transaktionssicherung.

CURSOR STABILITY

Der Prozeß erhält lesenden Zugriff auf einen Satz, sofern dieser durch keinen anderen Prozeß exklusiv gesperrt ist. Bei der Isolationsstufe Cursor Stability setzt der Prozeß selbst eine Share-Sperre auf den Satz, auf den der Satzzeiger zeigt. Der Satz bleibt gesperrt, bis der Prozeß, der die Sperre herbeigeführt hat, auf den nächsten Satz (mit FETCH) zugreift. Dadurch ist der Satz bei einem nachfolgenden UPDATE WHERE CURRENT OF bzw. DELETE WHERE CURRENT OF unverändert vorhanden.

Ausnahme:
Wird mit einem SELECT ohne Satzzeiger auf einen Einzelsatz zugegriffen, so wird die Share-Sperre sofort nach dem Lesen des Satzes wieder freigegeben.

REPEATABLE READ

Der Prozeß erhält lesenden Zugriff auf einen Satz, sofern dieser durch keinen anderen Prozeß exklusiv gesperrt ist. Bei der Isolationsstufe Repeatable Read setzt der Prozeß selbst eine Share-Sperre auf jeden gelesenen Satz. Alle gesperrten Sätze bleiben bis zum Ende der Transaktion gesperrt. Innerhalb der Transaktion kann deshalb eine Suche mit dem gleichen Ergebnis wiederholt werden.
Repeatable Read ist die voreingestellte Isolationsstufe für ANSI-Datenbanken.

Interaktiver Betrieb

Im interaktiven Betrieb können Sie keine Satzzeiger verwenden. Der Unterschied zwischen den Isolationsstufen Committed Read und Cursor Stability tritt deshalb im interaktiven Betrieb nicht in Erscheinung. Cursor Stability kann aber den Zugriff auf die Sätze verlangsamen.

ANSI-Standard

Die Anweisung SET ISOLATION ist nicht im ANSI-Standard enthalten. ANSI-Datenbanken arbeiten standardmäßig mit der Isolationsstufe Repeatable Read. Die Stufe der Isolation kann auch für ANSI-Datenbanken mit SET ISOLATION geändert werden.

>>>> COMMIT WORK, OPEN, ROLLBACK WORK

SET LOCK MODE - Warte-Modus definieren

SET LOCK MODE legt fest, ob nachfolgende SQL-Anweisungen auf einen gesperrten Satz warten. SET LOCK MODE wirkt nur auf Einzelsperren und nicht auf Tabellensperren. Bei INFORMIX-ONLINE kann die Anweisung SET LOCK MODE zusätzlich die maximale Wartezeit auf einen gesperrten Satz festlegen.

NOT WAIT
: Zugriffsversuche auf gesperrte Sätze werden mit Fehler abgewiesen. Dies entspricht dem Standardfall, wenn noch kein SET LOCK MODE gegeben wurde.

WAIT
: Bei einem Zugriffsversuch auf gesperrte Sätze wird gewartet, bis die Sperren vom anderen Prozeß freigegeben werden.

Achtung:
Bei
INFORMIX-SE-Datenbanken gibt es kein Limit für die Wartezeit auf die Freigabe der Sperre. Wenn der Prozeß, der die Sperre verursacht hat, den Satz nicht mehr freigibt (z.B. wegen einer Endlos-Schleife), dann wird der Prozeß, der auf die Freigabe des gesperrten Satzes wartet, in einen unbegrenzten Wartezustand versetzt.
Bei INFORMIX-ONLINE-Datenbanken kann der INFORMIX-ONLINE-Verwalter eine maximale Wartezeit festlegen, bis zu der maximal auf die Sperrfreigabe gewartet wird. Existiert keine maximale Wartezeit, wird unbegrenzt auf Freigabe der Sperren gewartet.

WAIT *n*
: Bei INFORMIX-ONLINE-Datenbanken kann eine Wartezeit von *n* Sekunden auf einen gesperrten Satz vereinbart werden. Wird die Wartezeit überschritten, so wird der Zugriffsversuch auf den gesperrten Satz abgewiesen.

Zusammenhang mit INFORMIX-STAR

Wenn auf Ihrem Rechner INFORMIX-STAR installiert ist, kann der INFORMIX-ONLINE-Verwalter global eine maximale Wartezeit auf Sperren festlegen. Die Wartezeit auf Sperren, die Sie mit SET LOCK MODE festlegen, ist nur wirksam, wenn sie kleiner ist als die vom INFORMIX-ONLINE-Verwalter festgelegte.

Reichweite von SET LOCK MODE

SET LOCK MODE TO WAIT wirkt nicht bei exklusiven Tabellen- oder Datenbank-Sperren. Das heißt, daß in den nachfolgenden drei Fällen trotz LOCK MODE WAIT jeglicher Zugriffsversuch abgewiesen wird.

- Die Datenbank ist explizit durch DATABASE... EXCLUSIVE gesperrt.
- Eine Tabelle ist mit LOCK TABLE... IN EXCLUSIVE MODE explizit gesperrt.
- Eine Tabelle oder auch die ganze Datenbank ist implizit durch eine Anweisung gesperrt, z.B. ALTER TABLE, ROLLFORWARD DATABASE, usw.

Im Verlauf einer INFORMIX-Sitzung kann der LOCK MODE beliebig oft gewechselt werden.

ANSI-Standard

Die Anweisung SET LOCK MODE ist nicht im ANSI-Standard enthalten.

> > > > DATABASE, LOCK TABLE

SET LOG - Protokollierungsmodus ändern

SET LOG schaltet die Transaktionsprotokollierung für die aktuelle Anwendung zwischen gepuffert und ungepuffert um.
Sie können jedoch damit weder die Protokollierung ein- oder ausschalten, noch den Standard-Protokolliermodus der Datenbank ändern. Das kann nur der INFORMIX-ONLINE-Verwalter.

Vor dem Aufruf beachten

SET LOG ist nur erlaubt, wenn die Transaktionssicherung für die Datenbank eingeschaltet ist. Die Transaktionssicherung wird entweder mit CREATE DATABASE beim Erzeugen der Datenbank oder danach vom INFORMIX-ONLINE-Verwalter eingeschaltet.

BUFFERED

Wählt gepufferte Protokollierung. Bei gepufferter Transaktionsprotokollierung ist es möglich, daß geänderte Sätze nach der Anweisung COMMIT WORK nicht in den Protokollen auf der Platte gespeichert sind. Dadurch können bei der Restaurierung nach einem Systemfehler Transaktionen fehlen. Die Performance wird durch die gepufferte Protokollierung jedoch gesteigert.

Achtung:
Da die Pufferung der Transaktionsprotokollierung eines INFORMIX-ONLINE-Systems gemeinsam im Shared Memory stattfindet, ist nur dann die gepufferte Transaktionsprotokollierung wirksam, wenn alle Prozesse mit gepufferter Protokollierung arbeiten. Wünscht nur ein Prozeß ungepufferte Protokollierung, so arbeitet das gesamte INFORMIX-ONLINE-System mit ungepufferter Protokollierung.

BUFFERED nicht angegeben:
Wählt ungepufferte Protokollierung. Nach einer mit der Anweisung COMMIT WORK abgeschlossenen Transaktion sind die geänderten Sätze auf der Platte gespeichert. Bei einem Systemfehler können alle beendeten Transaktionen restauriert werden.

ANSI-Standard

Die Anweisung SET LOG ist nicht im ANSI-Standard enthalten und kann nicht auf eine ANSI-Datenbank angewendet werden. ANSI-Datenbanken besitzen immer ungepufferte Protokollierung.

> > > > COMMIT WORK

START DATABASE - Transaktionssicherung einschalten

START DATABASE schaltet die Transaktionssicherung für eine INFORMIX-SE-Datenbank ein oder wechselt den Namen der Transaktionsprotokoll-Datei. Dabei kann festgelegt werden, daß die Datenbank eine ANSI-Datenbank mit impliziten Transaktionen wird.

Vor dem Aufruf beachten

START DATABASE wird für die aktuelle Datenbank abgewiesen. Sie müssen die aktuelle Datenbank zuerst mit CLOSE DATABASE schließen.

Sie müssen DBA-Zugriffsrecht für die Datenbank haben.

START DATABASE sperrt die Tabelle exklusiv gegen jeglichen Zugriff anderer Prozesse. START DATABASE wird abgewiesen, wenn ein anderer Prozeß die Tabelle bearbeitet. Dies gilt auch dann, wenn nur lesend zugegriffen wird.

datenbank
: Name der Datenbank, für die die Transaktionssicherung eingeschaltet wird. Diese Datenbank ist nach START DATABASE die aktuelle Datenbank, bleibt aber für andere Prozesse gesperrt, bis Sie CLOSE DATABASE ausführen.

datei
: Name der Transaktionsprotokoll-Datei. *datei* muß mit dem absoluten Pfadnamen angegeben werden. Der Dateiname darf maximal 64 Zeichen lang sein. Alle Dateiverzeichnisse im Pfad müssen bereits existieren. Existiert die Transaktionsprotokoll-Datei bereits, so wird START DATABASE abgewiesen.

MODE ANSI
: Die Datenbank wird eine ANSI-Datenbank mit impliziten Transaktionen (siehe Kapitel 2). Eine ANSI-Datenbank kann nicht mehr in eine Nicht-ANSI-Datenbank umgewandelt werden.

  MODE ANSI nicht angegeben:
  Transaktionen müssen explizit mit BEGIN WORK begonnen werden.

Transaktionsprotokoll-Datei auf Betriebssystemebene

Benutzer, die auf INFORMIX-Ebene berechtigt sind, eine Tabelle zu ändern, können die Tabelle dennoch nicht ändern, wenn sie auf Betriebssystemebene keine Zugriffserlaubnis auf die Transaktionsprotokoll-Datei haben.
Alle Benutzer, die auf eine Tabelle zugreifen wollen, müssen Ausführungsberechtigung (x-Bit) für alle Dateiverzeichnisse im Pfad besitzen.
Alle Benutzer, die eine Tabelle verändern wollen, müssen Schreibberechtigung (w-Bit) für die Transaktionsprotokoll-Datei besitzen.

Transaktionssicherung bei INFORMIX-SE

Den Namen der Transaktionsprotokoll-Datei für die aktuelle Datenbank können Sie über die Systemtabelle *systables* erfragen:

```
SELECT dirpath FROM systables WHERE tabname="syslog"
```

Die Transaktionssicherung für eine Nicht-ANSI-Datenbank kann nachträglich mit folgender Anweisung ausgeschaltet werden:

```
DELETE FROM systables WHERE tabname = "syslog"
```

Damit ist die Transaktionssicherung ausgeschaltet. Mit START DATABASE kann sie wieder eingeschaltet werden.

Eigenschaften von ANSI-Datenbanken

Für ANSI-Datenbanken gilt im Unterschied zu Nicht-ANSI-Datenbanken:

- Es ist automatisch Transaktionssicherung eingeschaltet. ANSI-Datenbanken arbeiten immer mit Transaktionen.
- Fremde Datenbankobjekte müssen mit dem Namen des Eigentümers qualifiziert werden.
- Bei Anweisungen, die den ANSI-Standard verletzen, werden Warnungen ausgegeben.

Eine ANSI-Datenbank kann nicht mehr in eine Nicht-ANSI-Datenbank umgewandelt werden.

ANSI-Standard

Die Anweisung START DATABASE ist nicht im ANSI-Standard enthalten.

Beispiel

Für die aktuelle Datenbank *versand* soll die Transaktionssicherung eingeschaltet und ein Transaktionsprotokoll eingerichtet werden. Die Datenbank muß zuvor mit CLOSE DATABASE geschlossen werden. Mit der nachfolgenden SELECT-Anweisung wird der Name des aktuell zugewiesenen Transaktionsprotokolls erfragt.

```
CLOSE DATABASE;
START DATABASE versand WITH LOG IN "/usr1/lomata/versand.log1";
SELECT dirpath FROM systables WHERE tabname="syslog"

dirpath

/usr1/lomata/versand.log1
```

>>>> BEGIN WORK, COMMIT WORK, CREATE DATABASE, ROLLBACK WORK, ROLLFORWARD DATABASE

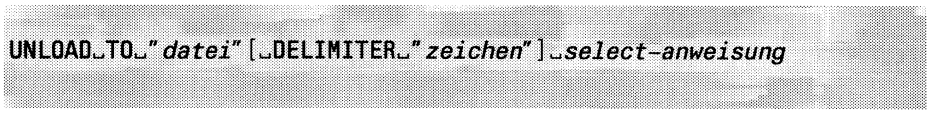
UNLOAD - Sätze aus Tabelle in Ausgabedatei schreiben

UNLOAD gibt die Sätze, die Sie über eine SELECT-Anweisung auswählen, in eine Datei aus. Das Ausgabeformat der Sätze entspricht genau dem Format, das die LOAD-Anweisung als Eingabe benötigt. Die Sätze werden nicht aus der Tabelle gelöscht.

Vor dem Aufruf beachten

Die UNLOAD-Anweisung kann nur interaktiv und bei INFORMIX-4GL verwendet werden.

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich entweder das SELECT-Zugriffsrecht für die entsprechenden Spalten haben oder Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.



"*datei*"
Name der Datei, in die die Sätze ausgegeben werden. Existiert die angegebene Datei bereits, so wird sie überschrieben.
Wird die Anweisung UNLOAD in INFORMIX-4GL verwendet, können Sie auch eine alphanumerische Variable statt "*datei*" angeben.

DELIMITER "*zeichen*"
Einzelnes Zeichen, das als Trennzeichen zwischen den Datenfeldern in der Ausgabedatei verwendet wird.
Wird die Anweisung UNLOAD in INFORMIX-4GL verwendet, können Sie auch eine alphanumerische Variable statt "*zeichen*" angeben.

Wird das Trennzeichen oder der Gegenschrägstrich innerhalb der auszugebenden Sätze verwendet, so werden sie von INFORMIX automatisch durch einen vorangestellten Gegenschrägstrich (\ = ASCII 92) entwertet. Beim Einlesen der Sätze mit LOAD werden die Gegenschrägstriche wieder entfernt. Beachten Sie, daß das Zeichen Ö im deutschen ISO-7-Bit-Code dem Gegenschrägstrich entspricht und deshalb entwertet wird.

Sie können auch Buchstaben, Ziffern, Komma (,) oder Punkt (.) als Trennzeichen definieren. Sie müssen aber sicherstellen, daß diese Zeichen nicht in den auszugebenden Werten vorkommen, da diese Zeichen bei der Ausgabe nicht entwertet werden.

DELIMITER "*zeichen*" nicht angegeben:
Das mit der Umgebungsvariablen DBDELIMITER definierte Zeichen wird als Trennzeichen benutzt. Ist DBDELIMITER nicht definiert, so gilt der senkrechte Strich (| = ASCII 124) als Standardtrennzeichen.

select-anweisung
SELECT-Anweisung, deren Ergebnistabelle ausgegeben wird. Die SELECT-Anweisung darf nicht die Klauseln INTO TEMP und INTO enthalten. Wird die Anweisung UNLOAD in INFORMIX-4GL verwendet, können Sie auch eine alphanumerische Variable statt *select-anweisung* angeben.

Datentyp und Format der ausgegebenen Spaltenwerte

Mit UNLOAD ausgegebene Sätze besitzen genau das Format, das die LOAD-Anweisung als Eingabe benötigt.

Enthält die Spalte den Wert NULL, so wird zwischen zwei Trennzeichen nichts ausgegeben.

Numerische Werte werden ohne führende Leerzeichen ausgegeben. Bei den Datentypen INTEGER und SMALLINT wird der Wert 0 als 0 ausgegeben. Bei den Datentypen FLOAT, SMALLFLOAT, DECIMAL und MONEY wird der Wert 0 als 0.0 bzw. 0,0 ausgegeben, je nach Wert der Umgebungsvariablen DBMONEY.

Alphanumerische Werte werden ohne Leerzeichen am Ende ausgegeben. Sind das Trennzeichen, der Gegenschrägstrich oder "neue Zeile" enthalten, so werden sie mit dem Gegenschrägstrich entwertet.

Werte des Datentyps DATE werden im aktuellen DATE-Format ausgegeben (durch die Umgebungsvariable DBDATE bestimmt).

Werte vom Datentyp MONEY werden ohne ein Währungssymbol ausgegeben.

Bei Werten vom Datentyp DATETIME und INTERVAL werden die einzelnen Komponenten als Zeichen in der Form *yyyy-mm-dd hh:mm:ss* ausgegeben.

Werte vom Datentyp BYTE werden im Hexadezimal-Format in die Datei ausgegeben.

Erfolgskontrolle bei Programmeinbettung

	Positivfall	Fehlerfall
INFORMIX-4GL	status = 0 SQLCA.SQLERRD[3] ist die Anzahl der mit UNLOAD ausgegebenen Sätze.	status < 0 SQLCA.SQLERRD[3] ist die Anzahl der mit UNLOAD erfolgreich ausgegebenen Sätze. Der Rest der Sätze wurde nicht ausgegeben.

ANSI-Standard

Die Anweisung UNLOAD ist nicht im ANSI-Standard enthalten.

Beispiel 1

Aus der Tabelle *kunde* werden alle Sätze in die Datei *kunde.unl* ausgegeben, die in der Spalte *ort* den Wert *Muenchen* haben. Es werden nur ausgewählte Spalten der gefundenen Sätze ausgegeben.
Das Ausgabeformat der Daten entspricht genau dem Format, das die LOAD-Anweisung als Eingabe benötigt.

```
UNLOAD TO "unl.kunde"
       SELECT kunden_nr, nachname, firma, ort, telefon
              FROM kunde WHERE ort = "Muenchen"
```

Inhalt der Datei *unl.kunde*:

```
104|Hochfeld|Spielball|Muenchen|089/25430|
108|Korting|Martin's Shop|Muenchen|089/33730|
110|Jaeger|Sportwaren|Muenchen|089/5032|
114|Albert|Der Sport-Albert|Muenchen|089/6367|
117|Sipell|Sportecke|Muenchen|089/6321|
```

Beispiel 2

Die nachfolgenden Anweisungen erzeugen eine Tabelle *tab1* und speichern dann 7 Sätze in die Tabelle. Anschließend wird die Tabelle in die Datei */usr/lomata/test/unl1* entladen. Die Umgebungsvariable DBDELIMITER ist auf **?**, DBDATE auf **Y4MD/** und DBMONEY auf **,** gesetzt.

```
CREATE TABLE tab1 ( f1 CHAR (9), f2 SMALLINT,
                    f3 INTEGER, f4 DECIMAL (6),
                    f5 DECIMAL (6,4), f6 SMALLFLOAT,
                    f7 FLOAT, f8 MONEY (4),
                    f9 SERIAL (4711), f10 DATE
                  );
INSERT INTO tab1 VALUES
       ("␣␣␣␣␣␣␣␣␣",0,0,0,0,0,0,0,0,"1899.12.31");
INSERT INTO tab1 VALUES
       ("AAA␣␣␣␣␣␣",1.1,1.1,1.1,1.1,1.1,1.1,1.1,1.1,0,today);
INSERT INTO tab1 VALUES
       ("␣␣␣BBB␣␣␣",1.9,1.9,1.9,1.9,1.9,1.9,1.9,0,null);
INSERT INTO tab1 VALUES
       ("␣␣␣␣␣␣CCC",-1.1,-1.1,-1.1,-1.1,-1.1,-1.1,-1.1,0,"87-1-1");
INSERT INTO tab1 VALUES
       ("AAABBBCCC+++",-1.9,-1.9,-1.9,-1.9,-1.9,-1.9,-1.9,0,"19871231");
INSERT INTO tab1
       (f1,f4,f5,f6,f7,f8,f9)
       values (user,1.234567e+2,12.34567,1234567890.123e-11,
               1234567890.123e-11,0.005,7777);
INSERT INTO tab1 (f9)
       VALUES  0);

UNLOAD TO "/usr/lomata/test/unl1" SELECT * FROM tab1
```

Inhalt der Datei */usr/lomata/test/unl1*:

```
␣?0?0?0,0?0,0?0,0?0,0?0,0?4711?1899/12/31?
AAA?1?1?1,1?1,1?1,1?1,1?1,1?4712?1987/09/10?
␣␣␣BBB?1?1?1,9?1,9?1,9?1,9?1,9?4713??
␣␣␣␣␣␣CCC?-1?-1?-1,1?-1,1?-1,1?-1,1?-1,1?4714?1987/01/01?
AAABBBCCC?-1?-1?-1,9?-1,9?-1,9?-1,9?-1,9?4715?1987/12/31?
lomata???123,457?12,3457?0,01234567?0,0123456789012?0,01?7777??
??????????7778??
```

Beispiel 3 für INFORMIX-4GL

Im folgenden Beispiel werden die SELECT-Anweisung und das Trennzeichen durch 4GL-Variable angegeben.

```
DEFINE selvar CHAR (100)
DEFINE del CHAR(1)

LET selvar = "SELECT * FROM posten"
LET del="%"

UNLOAD TO "xxx.unl" DELIMITER del selvar
```

> > > > LOAD, OUTPUT, SELECT

UNLOCK TABLE - Tabellensperre aufheben

UNLOCK TABLE hebt eine Tabellensperre auf, die Sie zuvor mit LOCK TABLE gesetzt haben. UNLOCK TABLE wird abgewiesen, falls nicht vorher vom gleichen Prozeß ein LOCK TABLE auf die gleiche Tabelle abgesetzt worden ist. Es ist natürlich nicht möglich, mit UNLOCK TABLE eine Tabellensperre aufzuheben, die ein anderer Prozeß gesetzt hat.

Vor dem Aufruf beachten

UNLOCK TABLE ist nur bei Datenbanken ohne Transaktionen erlaubt.

tabelle
Name oder Synonym der Basistabelle, die Sie freigeben wollen.

UNLOCK TABLE und Transaktionen

Bei Datenbanken mit Transaktionen wird UNLOCK TABLE abgewiesen. Dort wird automatisch bei COMMIT WORK bzw. ROLLBACK WORK die gesperrte Tabelle freigegeben.

ANSI-Standard

Die Anweisung UNLOCK TABLE ist nicht im ANSI-Standard enthalten.

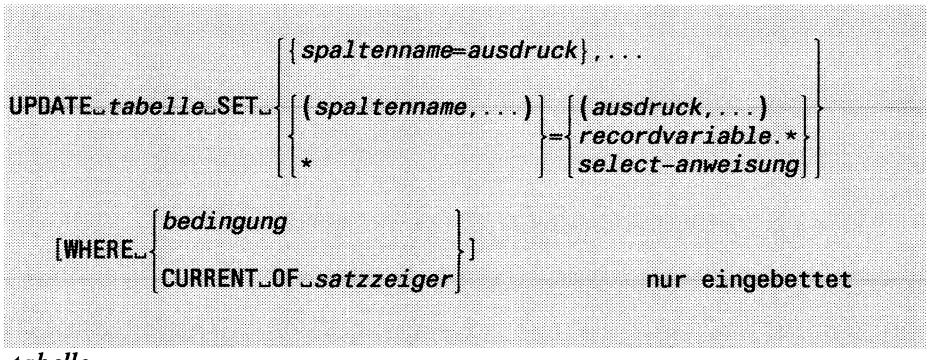
> > > > COMMIT WORK, LOCK TABLE, ROLLBACK WORK

UPDATE - Spaltenwerte ändern

UPDATE ändert Spaltenwerte von Sätzen in einer Tabelle.

Vor dem Aufruf beachten

Besitzen Sie das Connect-Zugriffsrecht, so müssen Sie zusätzlich entweder das UPDATE-Zugriffsrecht für die entsprechenden Spalten haben oder aber Tabelleneigentümer sein. Besitzen Sie das DBA-Zugriffsrecht, so werden keine weiteren Zugriffsrechte benötigt.



tabelle

Name oder Synonym der Tabelle, in der Sie Spaltenwerte ändern wollen. Die Tabelle kann eine Basistabelle, eine temporäre Tabelle oder ein änderbarer View sein.

spaltenname = ausdruck

Jeder Spalte wird einzeln ein Wert zugewiesen.

spaltenname

Spalte, deren Inhalt geändert werden soll. Eine Spalte vom Datentyp SERIAL darf nicht angegeben werden.

ausdruck

Ausdruck, dessen Wert der entsprechenden Spalte zugeordnet wird. *ausdruck* muß einen Wert liefern, der mit dem Datentyp der Spalte verträglich ist. Eine nähere Beschreibung von *ausdruck* finden Sie in Kapitel 5.

(*spaltenname,...***)**

Liste von Spalten, deren Inhalt geändert werden soll. Die Spalten müssen in der Tabelle enthalten sein und dürfen nicht vom Datentyp SERIAL sein.

*

Die Werte werden allen Spalten der Tabelle in der definierten Reihenfolge zugewiesen. Enthält die Tabelle eine SERIAL-Spalte, so ist nur die Zuweisung * = *recordvariable.** erlaubt, da in diesem Fall bei der Zuweisung der Werte SERIAL-Spalten automatisch übersprungen werden.
Sie können bei INFORMIX-4GL statt * auch *tabelle.** angeben.

(*ausdruck*,...)

Ausdruck, dessen Wert der entsprechenden Spalte zugeordnet wird. Der *i*-te Wert wird der *i*-ten Spalte in der Spaltenliste zugewiesen, falls eine solche angegeben wurde, ansonsten der *i*-ten Spalte der Tabelle. *ausdruck* muß einen Wert liefern, der mit dem Datentyp der Spalte verträglich ist. Ist *ausdruck* eine SELECT-Anweisung, so darf sie sich nicht auf die zu ändernde Tabelle beziehen. Eine nähere Beschreibung von *ausdruck* finden Sie in Kapitel 5.

Wird die Anweisung UPDATE in eine Programmiersprache eingebettet, können Sie einen Wert auch über eine Hostvariable eingeben. Die Hostvariablen können auch vom Datentyp TEXT oder BYTE sein. Bei Programmeinbettung kann statt einer Liste von Werten auch eine Record-Variable oder eine Struktur angeben werden (siehe Handbuch für die entsprechende Programmeinbettung [3, 4, 6]).

*recordvariable.**

Bei INFORMIX-4GL können Sie statt einer Liste von Ausdrücken eine Record-Variable angeben.

select-anweisung

SELECT-Anweisung, die einen Satz mit der passenden Spaltenanzahl als Ergebnis liefert. Die SELECT-Anweisung darf sich nicht auf die zu ändernde Tabelle beziehen und nicht die Klausel INTO und INTO TEMP enthalten.

WHERE-Klausel

Die WHERE-Klausel gibt an, welche Sätze geändert werden.

WHERE-Klausel nicht angegeben:
Alle Sätze der Tabelle werden geändert.

bedingung

Bedingung, die die zu ändernden Sätze erfüllen müssen. Ein Satz wird nur geändert, wenn er die angegebene Bedingung erfüllt. Eine nähere Beschreibung von *bedingung* finden Sie in Kapitel 5.

Einschränkung:
Enthält die Bedingung eine Unterabfrage, so darf diese nicht die Klausel UNIQUE bzw. DISTINCT enthalten.

CURRENT OF *satzzeiger*
Bei Programmeinbettung kann der zu ändernde Satz über den Satzzeiger angegeben werden.
UPDATE ändert den aktuellen Satz, auf den der Satzzeiger zeigt. Danach zeigt der Satzzeiger auf den geänderten Satz.

Wenn Sie bei DECLARE ... FOR UPDATE OF Spalten angegeben haben, so können Sie mit UPDATE nur diese Spalten ändern.

Voraussetzung:
Der Satzzeiger muß zuvor in derselben Quellprogrammdatei mit DECLARE für eine SELECT-Anweisung mit der Klausel FOR UPDATE und ohne die SCROLL-Klausel vereinbart worden sein. Der Satzzeiger muß mit OPEN geöffnet und mit FETCH auf einen Satz der Ergebnistabelle positioniert worden sein.

Bei einer Datenbank mit Transaktionssicherung müssen Sie UPDATE mit WHERE CURRENT OF innerhalb einer Transaktion verwenden. Die FETCH-Anweisung, die den Satz bereitgestellt hat, muß in der gleichen Transaktion durchgeführt worden sein.

Sperren beim Ändern

Wenn Sie auf einer Datenbank mit Transaktionssicherung arbeiten und die UPDATE-Anweisung außerhalb einer Transaktionsklammer ausführen, dann werden alle betroffenen Sätze gesperrt, bis die UPDATE-Anweisung komplett abgearbeitet ist.

Wird die UPDATE-Anweisung innerhalb einer Transaktionsklammer ausgeführt, dann werden alle betroffenen Sätze bis zum Ende der Transaktion gesperrt.

Ist die Zahl der betroffenen Sätze sehr groß, kann die maximal zulässige Anzahl an Sperren überschritten werden. Die Anzahl der möglichen Sperren ist bei INFORMIX-SE durch das Betriebssystem bzw. bei INFORMIX-ONLINE durch Shared Memory Parameter beschränkt.

Sie sollten in diesem Fall die Anzahl der zu ändernden Sätze reduzieren oder die ganze Tabelle vor der Ausführung der UPDATE-Anweisung sperren (siehe LOCK TABLE).

Datenintegrität beim Ändern

Datenbank ohne Transaktionssicherung

Wenn eine UPDATE-Anweisung abgebrochen wird, dann ist sie teilweise ausgeführt worden. Die bis zum Abbruch der Anweisung geänderten Sätze sind in der Tabelle verändert, die restlichen Sätze sind unverändert vorhanden.

Beispiel:

Eine UPDATE-Anweisung, die eigentlich 100 Sätze verändern würde, läuft beim 57. Satz auf einen Konvertierungsfehler. Die UPDATE-Anweisung wird zwar abgebrochen, die Sätze 1 bis 56 verbleiben aber im veränderten Zustand in der Tabelle.

Datenbank mit Transaktionssicherung

Bei einer Nicht-ANSI-Datenbank wird jede UPDATE-Anweisung außerhalb einer Transaktion automatisch als einzelne Transaktion behandelt. Eine unvollständig ausgeführte UPDATE-Anweisung wird automatisch zurückgesetzt. Unvollständige UPDATE-Anweisungen wie in obigem Beispiel können also nicht vorkommen.

Bei einer Nicht-ANSI-Datenbank innerhalb einer Transaktion und bei einer ANSI-Datenbank müssen Sie die Transaktion, in der die UPDATE-Anweisung ausgeführt wird, explizit mit COMMIT WORK bzw. ROLLBACK WORK beenden.

Erfolgskontrolle bei Programmeinbettung

	Positivfall	Fehlerfall
INFORMIX-4GL	status = 0 SQLCA.SQLERRD[3] ist die Anzahl der mit UPDATE geänderten Sätze.	status < 0 SQLCA.SQLERRD[3] ist die Anzahl der mit UPDATE erfolgreich geänderten Sätze. Der Rest der Sätze ist nicht geändert.
INFORMIX-ESQL/C	sqlca.sqlcode = 0 sqlca.sqlerrd[2] ist die Anzahl der mit UPDATE geänderten Sätze.	sqlca.sqlcode < 0 sqlca.sqlerrd[2] ist die Anzahl der mit UPDATE erfolgreich geänderten Sätze. Der Rest der Sätze ist nicht geändert.
INFORMIX-ESQL/COBOL	SQLCODE OF SQLCA = 0 SQLERRD[3] ist die Anzahl der mit UPDATE geänderten Sätze.	SQLCODE OF SQLCA < 0 SQLERRD[3] ist die Anzahl der mit UPDATE erfolgreich geänderten Sätze. Der Rest der Sätze ist nicht geändert.

ANSI-Standard

Die Anweisung UPDATE ist im ANSI-Standard enthalten, aber jeder Spalte kann nur einzeln ein Wert zugewiesen werden (*spaltenname = ausdruck*).

Beispiel 1

Die folgende UPDATE-Anweisung ändert einen ganzen Satz der Tabelle *artikel*.

```
UPDATE artikel SET * = (6,"SMT","Tennisschlaeger",12,"Box","1/BOX")
               WHERE artikel_nr=6 AND herstellercode="SMT"
```

Beispiel 2 für INFORMIX-4GL

Die folgende UPDATE-Anweisung ändert einen ganzen Satz der Tabelle *artikel* durch die Zuweisung der Record-Variablen *art*.

```
DEFINE art RECORD LIKE artikel.*

LET art.artikel_nr     = 6
LET art.herstellercode = "SMT"
LET art.bezeichnung    = "Tennisschlaeger"
LET art.preis          = 12
LET art.liefereinheit  = "Box"
LET art.stueckliste    = "1/Box"

UPDATE artikel SET * = art.*
               WHERE artikel_nr=6 AND herstellercode="SMT"
```

Beispiel 3 für INFORMIX-4GL

Im folgenden Beispiel wird der Satz geändert, auf den der Satzzeiger *zn* aktuell zeigt.

```
DEFINE kundnr LIKE kunde.kunden_nr

DECLARE zn CURSOR FOR
        SELECT kunden_nr FROM auftrag WHERE auftrags_nr > 1010
        FOR UPDATE

OPEN zn

FETCH zn INTO kundnr

UPDATE auftrag SET kunden_nr=0 WHERE CURRENT OF zn
```

> > > > DECLARE, DELETE, FETCH, INSERT

UPDATE STATISTICS - Satzanzahl aktualisieren

UPDATE STATISTICS aktualisiert den Eintrag über die Anzahl der Sätze der Basistabellen und Views.

INFORMIX-ONLINE aktualisiert zusätzlich Indexinformationen in der Systemtabelle *sysindexes*.

Vor dem Aufruf beachten

Sie müssen das Connect-Zugriffsrecht besitzen.

UPDATE STATISTICS wird abgewiesen, wenn ein anderer Prozeß die Tabelle exklusiv gesperrt hat (LOCK TABLE EXCLUSIVE).

```
UPDATE␣STATISTICS[␣FOR␣TABLE␣tabelle]
```

FOR TABLE *tabelle*
Name oder Synonym der Tabelle, deren Eintrag Sie aktualisieren wollen. Sie können eine Basistabelle oder einen View angeben.

FOR TABLE *tabelle* nicht angegeben:
Die Einträge aller Tabellen der aktuellen Datenbank werden aktualisiert.

Auswirkung der Anweisung UPDATE STATISTICS

INFORMIX verwaltet für jede Basistabelle und jeden View einen Eintrag über die Anzahl der Tabellensätze in der Systemtabelle *systables*. Dieser Eintrag wird zum Optimieren von Suchanweisungen verwendet. Da der Eintrag nicht automatisch aktualisiert wird, wenn sich die Anzahl der Tabellensätze ändert, sollte er nach gravierenden Änderungen mit der Anweisung UPDATE STATISTICS auf den aktuellen Stand gebracht werden.

UPDATE STATISTICS in Transaktionen

Die Anweisung UPDATE STATISTICS kann bei einer INFORMIX-SE-Datenbank nicht mit ROLLBACK WORK zurückgesetzt werden. ROLLBACK WORK wird zwar in diesem Fall nicht abgewiesen, führt aber zu einem nicht definierten Ergebnis, z.B. wird die Transaktion nur teilweise zurückgesetzt.

INFORMIX-ONLINE kann die Anweisung UPDATE STATISTICS zurücksetzen.

ANSI-Standard

Die Anweisung UPDATE STATISTICS ist nicht im ANSI-Standard enthalten.

Beispiel

Das folgende Beispiel zeigt die Status-Ausgabe der INFO-Anweisung vor einer Aktualisierung der Satzanzahl (*Anzahl Saetze* = 0). Durch die Anweisung UPDATE STATISTICS wird die Satzanzahl auf den aktuellen Wert gesetzt. Die zweite INFO-Anweisung gibt die aktualisierte Satzanzahl aus (*Anzahl Saetze* = 14).

```
INFO STATUS FOR auftrag
```

```
Tabellen-Name       auftrag
Eigentuemer         lomata
Satzlaenge          80
Anzahl Saetze       0
Anzahl Spalten      10
Erstellungsdatum    07.04.1990
```

```
UPDATE STATISTICS FOR TABLE auftrag

INFO STATUS FOR auftrag
```

```
Tabellen-Name       auftrag
Eigentuemer         lomata
Satzlaenge          80
Anzahl Saetze       14
Anzahl Spalten      10
Erstellungsdatum    07.04.1990
```

A Anhang

A.1 Beispieldatenbank versand

In diesem Abschnitt wird die Beispieldatenbank *versand* beschrieben, die für die Beispiele in diesem Handbuch verwendet wird. Die Struktur der Datenbank-Tabellen werden beschrieben, für welche Spalten ein Index definiert ist und über welche Spalten eine Verbindung der Tabellen (Join) möglich ist.

Für jede Tabelle werden die in ihr enthaltenen Daten aufgelistet.

Struktur der Tabellen

Die Datenbank *versand* enthält Informationen über einen fiktiven Sportartikelgroßhändler, der Läden in Süddeutschland beliefert. Die Datenbank besteht aus sechs Tabellen:

- kunde
- auftrag
- posten
- artikel
- hersteller
- bundesland

Tabelle kunde

Die Tabelle *kunde* enthält Informationen über 18 Läden, die Sportartikel vom Großhändler beziehen. Für jeden Laden wird der Name des Ladeninhabers und der Name und die Adresse des Ladens gespeichert. Die Tabelle *kunde* hat folgende Spalten:

```
Spaltenname          Datentyp

kunden_nr            SERIAL
vorname              CHAR(15)
nachname             CHAR(15)
firma                CHAR(20)
adresse1             CHAR(20)
adresse2             CHAR(20)
ort                  CHAR(15)
```

```
bundesland          CHAR(2)
plz                 CHAR(5)
telefon             CHAR(18)
```

Für die Spalte *kunden_nr* ist ein eindeutiger Index definiert. Für die Spalte *plz* ist ebenfalls ein Index definiert; bei diesem sind doppelt vorkommende Werte erlaubt.

Tabelle auftrag

Die Tabelle *auftrag* enthält die Aufträge, die die Kunden dem Großhändler erteilt haben. Für jeden Auftrag ist die Nummer des Auftrags gespeichert; zusätzlich das Auftragsdatum, die Kundennummer, Versandanweisungen, evtl. Rückstände, die Nummer des Kundenbestellscheins, Versandkosten und das Datum, an dem der Kunde für den Auftrag bezahlt hat. Die Spalten in der Tabelle *auftrag* sind:

```
Spaltenname         Datentyp

auftrags_nr         SERIAL
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
```

Für die Spalte *auftrags_nr* ist ein eindeutiger Index definiert; die Spalte *kunden_nr* ist ebenfalls indiziert, läßt aber doppelt vorkommende Werte zu.

Tabelle posten

Die Tabelle *posten* enthält die einzelnen Positionen eines Auftrags. Beispiel: Einige Aufträge enthalten nur einen Posten, während andere Aufträge fünf Posten haben. Die Tabelle *posten* enthält die Postennummer, Auftragsnummer, Artikelnummer, Herstellercode, Menge und den Gesamtpreis für jeden Artikel, der bestellt wurde. Die Spalten in der Tabelle *posten* sind:

```
Spaltenname         Datentyp

posten_nr           SMALLINT
auftrags_nr         INTEGER
artikel_nr          SMALLINT
herstellercode      CHAR(3)
menge               SMALLINT
gesamtpreis         MONEY(8,2)
```

Die Spalte *auftrags_nr* ist indiziert und läßt doppelt vorkommende Werte zu. Die Spalten *artikel_nr* und *herstellercode* bilden einen zusammengesetzen Index, doppelt vorkommende Werte sind erlaubt.

Tabelle artikel

Der Großhändler bietet seinen Kunden fünfzehn verschiedene Arten von Sportartikeln an. Beispiel: Der Großhändler bietet Ski- Handschuhe von drei verschiedenen Herstellern und Basketbälle von einem Hersteller an.

Die Tabelle *artikel* ist ein Katalog der Artikel, die vom Großhändler verkauft werden. Für jeden Artikel im Lager hat die Tabelle *artikel* eine Nummer, die den Artikel kennzeichnet, einen Code, der den Hersteller bezeichnet, eine Beschreibung des Artikels, den Preis der Packung, die Mindestmenge, die bestellt werden muß und eine Beschreibung der Pakkung (z.B.: Eine Kiste enthält zehn Fußbälle).

Die Tabelle *artikel* enthält folgende Spalten:

```
Spaltenname         Datentyp

artikel_nr          SMALLINT
herstellercode      CHAR(3)
bezeichnung         CHAR(15)
preis               MONEY(6,2)
liefereinheit       CHAR(4)
stueckliste         CHAR(15)
```

Die Spalten *artikel_nr* und *herstellercode* bilden einen zusammengesetzten Index, der eindeutige Werte enthalten muß.

Tabelle hersteller

Der Großhändler hat Sportartikel von fünf Herstellern. Informationen über diese Hersteller sind in der Tabelle *hersteller* gespeichert. Diese Informationen bestehen aus einem Herstellercode und dem Namen des Herstellers. Die Spalten der Tabelle *hersteller* sind:

```
Spaltenname          Datentyp

herstellercode       CHAR(3)
herstellername       CHAR(15)
```

Für die Spalte *herstellercode* ist ein eindeutiger Index definiert.

Tabelle bundesland

Die Tabelle *bundesland* enthält die Namen und Abkürzungen der 11 Bundesländer der Bundesrepublik Deutschland. Die Spalten der Tabelle *bundesland* sind:

```
Spaltenname          Datentyp

bcode                CHAR(2)
laendername          CHAR(20)
```

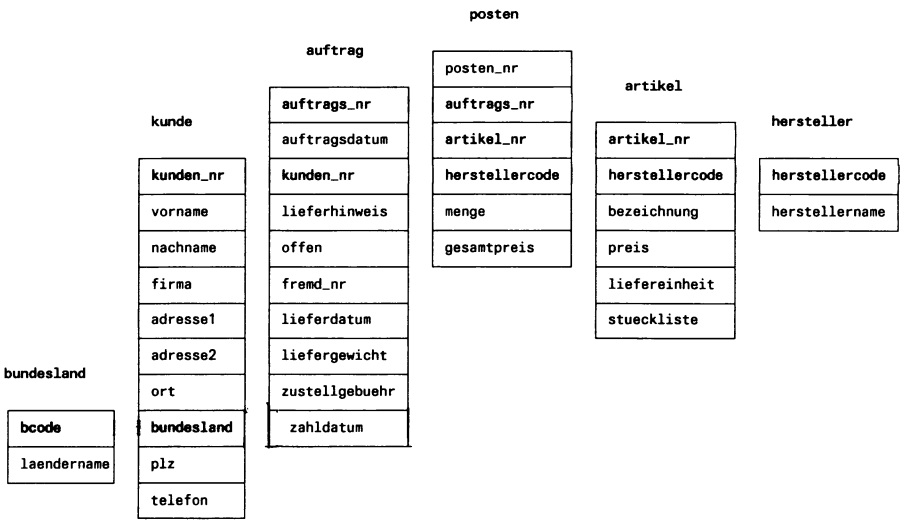
Für die Spalte *bcode* ist ein eindeutiger Index definiert.

Verbindung der Tabellen durch Join-Spalten

Die sechs Tabellen der Datenbank *versand* lassen sich durch Spalten verbinden, die als Join-Spalten geeignet sind. Das ermöglicht eine Datenbankabfrage über mehrere Tabellen gleichzeitig.

Übersicht

Die gerasterten Spalten zeigen die sinnvollen Join-Verbindungen zwischen den Tabellen der Datenbank.



Im folgenden wird jeweils die Verbindung von zwei Tabellen betrachtet.

Join-Spalten in den Tabellen kunde und auftrag

Die Tabellen *kunde* und *auftrag* lassen sich durch die Spalte *kunden_nr* in der Tabelle *kunde* und *kunden_nr* in der Tabelle *auftrag* verbinden.

Ausschnitt aus der Tabelle *kunde*:

kunden_nr	vorname	nachname
101	Ludwig	Pauli
102	Karola	Sadler
103	Philipp	Korres
104	Anton	Hochfeld

Ausschnitt aus der Tabelle *auftrag*:

auftrags_nr	auftragsdatum	kunden_nr
1001	20.01.1990	104
1002	01.06.1990	101
1003	12.10.1990	104
1004	12.04.1990	106

Die Tabelle *kunde* enthält in der Spalte *kunden_nr* für jeden Kunden eine eindeutige Nummer und zusätzlich Spalten für Namen, Firma, Adresse, Telefonnummer etc. Die Tabelle *auftrag* enthält ebenfalls eine Spalte *kunden_nr*, die die Nummer des Kunden enthält, der den Auftrag gab und weitere Spalten, die den Auftrag betreffen.

Will man die Aufträge von Anton Hochfeld abfragen, so verbindet man die Tabellen *kunde* und *auftrag* über die Spalte *kunden_nr*. Anton Hochfeld hat die Kundennummer 104 (aus Tabelle *kunde* gelesen). In der Tabelle *auftrag* gibt es zwei Aufträge mit der Kundennummer 104, Anton Hochfeld hat also zur Zeit zwei Aufträge gegeben.

Durch die Verbindung der Tabellen *kunde* und *auftrag* über die Join-Spalte *kunden_nr* kann der Name und die Adresse eines Kunden gleichzeitig mit den zugehörigen Aufträgen abgefragt werden.

Join-Spalten in den Tabellen auftrag und posten

Die Tabellen *auftrag* und *posten* lassen sich durch die Spalte *auftrags_nr* in der Tabelle *auftrag* und *auftrags_nr* in der Tabelle *posten* verbinden. In der Tabelle *auftrag* wird jeder Auftrag eines Kunden durch eine eindeutige Auftragsnummer identifiziert. Für jeden Posten dieses Auftrags wird ein Satz mit der zugehörigen Auftragsnummer und Informationen über den bestellten Artikel in die Tabelle *posten* geschrieben.

Durch die Join-Spalte *auftrags_nr* wird die Zuordnung zwischen Auftrag und zugehörigen Posten hergestellt.

Ausschnitt aus der Tabelle *auftrag*:

auftrags_nr	auftragsdatum	kunden_nr
1001	20.01.1990	104
1002	01.06.1990	101
1003	12.10.1990	104

Ausschnitt aus der Tabelle *posten*:

posten_nr	auftrags_nr	artikel_nr	herstellercode
1	1001	1	HRO
1	1002	4	HSK
2	1002	3	HSK
1	1003	9	ANZ
2	1003	8	ANZ
3	1003	5	ANZ

Join-Spalten in den Tabellen posten und artikel

Die Tabellen *posten* und *artikel* lassen sich durch zwei Spalten verbinden: *artikel_nr* und *herstellercode* in der Tabelle *posten* und *artikel_nr* und *herstellercode* in der Tabelle *artikel*.

Beide Spalten werden zur eindeutigen Identifizierung eines Artikels benötigt; z.B. ist der Artikel mit der Artikelnummer 1 und dem Herstellercode HRO ein Herold Ski-Handschuh, während der Artikel mit der Artikelnummer 1 und dem Herstellercode HSK ein Hasken Ski-Handschuh ist.

Die gleichen Artikelnummern und Herstellercodes können in mehreren Sätzen der Tabelle *posten* auftreten, wenn dieser Artikel in verschiedenen Aufträgen bestellt wurde.

Durch diese beiden Spalten *artikel_nr* und *herstellercode* wird der Zusammenhang zwischen Posten eines Auftrags (= bestellter Artikel) in der Tabelle *posten* und dem Artikel in der Tabelle *artikel* hergestellt.

Ausschnitt aus der Tabelle *posten*:

posten_nr	auftrags_nr	artikel_nr	herstellercode
1	1001	1	HRO
1	1002	4	HSK
2	1002	3	HSK
1	1003	9	ANZ
2	1003	8	ANZ
3	1003	5	ANZ
1	1004	1	HRO

Ausschnitt aus der Tabelle *artikel*:

artikel_nr	herstellercode	bezeichnung
1	HRO	Ski-Handschuhe
1	HSK	Ski-Handschuhe
1	SMT	Ski-Handschuhe

Join-Spalten in den Tabellen artikel und hersteller

Die Tabellen *artikel* und *hersteller* lassen sich durch die Spalte *herstellercode* in der Tabelle *artikel* und *herstellercode* in der Tabelle *hersteller* verbinden. Dieser Herstellercode erscheint in mehreren Sätzen der Tabelle *artikel*, in der Tabelle *hersteller* ist er eindeutig.

Durch die Verbindung der Tabellen *artikel* und *hersteller* über die Spalte *herstellercode* kann zu einem bestimmten Artikel der volle Name des Herstellers gefunden werden.

Ausschnitt aus der Tabelle *artikel*:

artikel_nr	herstellercode	bezeichnung
1	HRO	Ski-Handschuhe
1	HSK	Ski-Handschuhe
1	SMT	Ski-Handschuhe
2	HRO	Ski-Brille

Ausschnitt aus der Tabelle *hersteller*:

herstellercode	herstellername
NRG	Norgesta
HSK	Hasken
HRO	Herold

Join-Spalten in den Tabellen kunde und bundesland

Die Tabellen *kunde* und *bundesland* lassen sich durch die Spalte *bundesland* in der Tabelle *kunde* und *bcode* in der Tabelle *bundesland* verbinden. Wenn mehrere Kunden im selben Bundesland wohnen, tritt die Abkürzung für das Bundesland in mehreren Sätzen der Tabelle *kunde* auf.

Ausschnitt aus der Tabelle *kunde*:

kunden_nr	vorname	nachname	...	bundesland
101	Ludwig	Pauli		BY
102	Karola	Sadler		BY
103	Philipp	Korres		BY

Ausschnitt aus der Tabelle *bundesland*:

bcode	laendername
BW	Baden-Wuerttemberg
BY	Bayern
BE	Berlin (West)
BR	Bremen

Daten in der Datenbank versand

Die Daten, die in der Datenbank *versand* gespeichert sind, finden Sie in den folgenden Tabellen.

Tabelle kunde

kunden_nr	vorname	nachname	firma	adresse1	adresse2	ort	bundesland	plz	telefon
101	Ludwig	Pauli	Pauli Sport	Forstweg 47		Augsburg	BY	8900	0821/8075
102	Karola	Sadler	Sport-Aktiv	Blumenstr. 12		Wasserburg	BY	8090	08071/121289
103	Philipp	Korres	Philipp's Sportwaren	Sandgasse 2	Postfach 3498	Rosenheim	BY	8200	08031/4543
104	Anton	Hochfeld	Spielball	Goethestr. 34	Lilienstr.42	Muenchen	BY	8000	089/25430
105	Raimund	Viktor	Der Laden	Hofstr. 34		Ingolstadt	BY	8070	0841/2321
106	Georg	Watt	Sport Watt	Am Bergsteig 88		Ulm	BY	7900	0731/7334421
107	Karl	Remark	- Sportwaren -	Allee 13		Rosenheim	BY	8200	08031/560324
108	Martin	Korting	Martin's Shop	Buchenstr. 51		Muenchen	BY	8000	089/33730
109	Janette	Millet	Sportausstattung	Munsterstr. 1	Wofgangstr 66d	Augsburg	BY	8900	0821/358789
110	Roland	Jaeger	Sportwaren	Herbststr. 77		Muenchen	BY	8000	089/5032
111	Frank	Keyser	Sport Keyser	Im Tal 2		Augsburg	BY	8900	0821/7845
112	Margarete	Larsinger	Larsinger & Partner	Wiesenweg 11		Ingolstadt	BY	8070	0841/797235
113	Liane	Bergen	Sporthaus	Hochstr. 59		Landshut	BY	8300	0871/6699182
114	Frank	Albert	Der Sport-Albert	Torstr. 33		Muenchen	BY	8000	089/6367
115	Alfred	Grantella	Sportladen	Forchenstr. 99		Landshut	BY	8300	0871/498822
116	Johannes	Partellman	Olympia Sport	Spinosastr. 10		Ulm	BY	7900	0731/601123
117	Arnold	Sipell	Sportecke	Marktplatz 4		Muenchen	BY	8000	089/6321
118	Dieter	Bachmann	Sportausstatter	Lindenweg 4	y	Passau	BY	8390	0851/560011

Tabelle auftrag

auftrags_nr	auftragsdatum	kunden_nr	lieferhinweis	offen	fremd_nr	lieferdatum	liefergewicht	zustellgebuehr	zahldatum
1001	20.01.1990	104	Spedition	n	B77836	01.02.1990	20,40	10,00	22.03.1990
1002	01.06.1990	101	Hintertuere klingeln	n	9270	06.06.1990	50,60	15,30	03.07.1990
1003	12.10.1990	104	Spedition	n	B77890	13.10.1990	35,60	10,80	04.11.1990
1004	12.04.1990	106	zweimal klingeln	j	8006	30.04.1990	95,80	19,20	
1005	04.12.1990	116	vorher anrufen	n	2865	19.12.1990	80,80	16,20	30.12.1990
1006	19.09.1990	112	nach 10.00 Uhr	j	013557		70,80	14,20	
1007	25.03.1990	117		n	278693	23.04.1990	125,90	25,20	
1008	17.11.1990	110	Samstags geschlossen	j	LZ230	06.12.1990	45,60	13,80	21.12.1990
1009	14.02.1990	111	zweite Tuere rechts	n	4745	04.03.1990	20,40	10,00	21.04.1990
1010	29.05.1990	115	wenn geschlossen, nebenan abliefern	n	4290	08.06.1990	40,60	12,30	22.07.1990
1011	23.03.1990	104	Spedition	n	B77897	13.04.1990	10,40	5,00	01.06.1990
1012	05.06.1990	117		n	278701	09.06.1990	70,80	14,20	
1013	01.09.1990	104	Spedition	n	B77930	18.09.1990	60,80	12,20	10.10.1990
1014	01.05.1990	106	mehrfach klingeln	n	8052	10.05.1990	40,60	12,30	18.07.1990
1015	10.07.1990	110	Samstags geschlossen	n y	MA003	01.08.1990	20,60	6,30	31.08.1990

Tabelle posten

posten_nr	auftrags_nr	artikel_nr	herstellercode	menge	gesamtpreis
1	1001	1	HRO	1	250,00
1	1002	4	HSK	1	960,00
2	1002	3	HSK	1	240,00
1	1003	9	ANZ	1	20,00
2	1003	8	ANZ	1	840,00
3	1003	5	ANZ	5	99,00
1	1004	1	HRO	1	960,00
2	1004	2	HRO	1	126,00
3	1004	3	HSK	1	240,00
4	1004	1	HSK	1	800,00
1	1005	5	NRG	10	280,00
2	1005	5	ANZ	10	198,00
3	1005	6	SMT	1	36,00
4	1005	6	ANZ	1	48,00
1	1006	5	SMT	5	125,00
2	1006	5	NRG	5	190,00
3	1006	5	ANZ	5	99,00
4	1006	6	SMT	1	36,00
5	1006	6	ANZ	1	48,00
1	1007	1	HRO	1	250,00
2	1007	2	HRO	1	126,00
3	1007	3	HSK	1	240,00
4	1007	4	HRO	1	480,00
5	1007	7	HRO	1	600,00
1	1008	8	ANZ	1	840,00
2	1008	9	ANZ	5	100,00
1	1009	1	SMT	1	450,00
1	1010	6	SMT	1	36,00
2	1010	6	ANZ	1	48,00
1	1011	5	ANZ	5	99,00
1	1012	8	ANZ	1	840,00
2	1012	9	ANZ	10	200,00
1	1013	5	ANZ	1	19,80
2	1013	6	SMT	1	36,00
3	1013	6	ANZ	1	48,00
4	1013	9	ANZ	2	40,00
1	1014	4	HSK	1	960,00
2	1014	4	HRO	1	480,00
1	1015	1	SMT	1	450,00

Tabelle artikel

artikel_nr	herstellercode	bezeichnung	preis	liefereinheit	stueckliste
1	HRO	Ski-Handschuhe	250,00	Box	10/Box
1	HSK	Ski-Handschuhe	800,00	Box	10/Box
1	SMT	Ski-Handschuhe	450,00	Box	10/Box
2	HRO	Ski-Brille	126,00	Box	24/Box
3	HSK	Ski-Stock	240,00	Col.	12/Colli
4	HSK	Fussball	960,00	Col.	24/Colli
4	HRO	Fussball	480,00	Col.	24/Colli
5	NRG	Tennisschlaeger	28,00	solo	solo
5	SMT	Tennisschlaeger	25,00	solo	solo
5	ANZ	Tennisschlaeger	19,80	solo	solo
6	SMT	Tennisball	36,00	Box	24 Dosen/Box
6	ANZ	Tennisball	48,00	Box	24 Dosen/Box
7	HRO	Basketball	600,00	Box	24/Box
8	ANZ	Volleyball	840,00	Box	24/Box
9	ANZ	Volleyb. profi	20,00	solo	solo

Tabelle hersteller

herstellercode	herstellername
SMT	Schmitt
ANZ	Anzabel
NRG	Norgesta
HSK	Hasken
HRO	Herold

Tabelle bundesland

bcode	laendername
BW	Baden-Wuerttemberg
BY	Bayern
BE	Berlin (West)
BR	Bremen
HA	Hamburg
HE	Hessen
NI	Niedersachsen
NW	Nordrhein-Westfalen
RP	Rheinland-Pfalz
SA	Saarland
SH	Schleswig-Holstein

A.2 Systemtabellen

INFORMIX legt für jede Datenbank automatisch Tabellen an, die Informationen über die gesamte Datenbank enthalten. Diese Systemtabellen können von allen Benutzern gelesen werden.
Folgende Systemtabellen werden angelegt:

systables	informiert über Tabellen
syscolumns	informiert über Spalten
sysindexes	informiert über Indizes
systabauth	informiert über Tabellenzugriffsrechte
syscolauth	informiert über Spaltenzugriffsrechte
sysdepend	informiert darüber, wie Views von Tabellen abhängen
syssynonyms	wird nicht mehr verwendet; bleibt aus Kompatibilitätsgründen erhalten
sysusers	informiert über Datenbankzugriffsrechte
sysviews	informiert über die Definition von Views
sysconstraints	informiert über Constraints
syssyntable	informiert über definierte Synonyme

Die Struktur und Bedeutung der Systemtabellen wird im folgenden beschrieben.

Systemtabelle systables

Spaltenname	Datentyp	Bedeutung
tabname	CHAR(18)	Tabellenname
owner	CHAR(8)	Benutzerkennung des Tabelleneigentümers
dirpath	CHAR(64)	nur INFORMIX-SE: Pfad für die Tabellendateien
partnum	INTEGER	nur INFORMIX-ONLINE: Nummer des Tblspace, in dem die Tabelle gespeichert ist
tabid	SERIAL	fortlaufende Tabellennummer Die INFORMIX-eigenen Tabellen haben die Nummern 1 - 99, Benutzertabellen beginnen mit Nr. 100.
rowsize	SMALLINT	Satzlänge der Tabelle
ncols	SMALLINT	Anzahl der Spalten
nindexes	SMALLINT	Anzahl der Indizes
nrows	INTEGER	Anzahl der Sätze Dieser Wert wird nicht ständig aktualisiert. Erst UPDATE STATISTICS trägt den neuesten Stand ein
created	DATE	Erstellungsdatum
version	INTEGER	Versionsnummer Alle Anweisungen, die das Schema der Tabelle verändern, erhöhen die Nummer um 1.
tabtype	CHAR(1)	Tabellentyp T = Basistabelle S = Synonym V = View
audpath	CHAR(64)	nur INFORMIX-SE: Pfad für AUDIT-Datei, wenn AUDIT-Protokollierung eingeschaltet ist

Die folgenden Spalten treten nur bei INFORMIX-ONLINE auf:

locklevel	CHAR(1)	Sperrbereich für Einzelsperren P = Sperrbereich ist Page R = Sperrbereich ist Satz
npused	SMALLINT	Anzahl belegte Pages
fextsize	INTEGER	Größe des Initial-Extent
nextsize	INTEGER	Größe der weiteren Extents
flags	SMALLINT	z. Zt. nicht belegt
site	CHAR(18)	z. Zt. nicht belegt
dbname	CHAR(18)	z. Zt. nicht belegt

Systemtabelle syscolumns

Spaltenname	Datentyp	Bedeutung
colname	CHAR(18)	Spaltenname
tabid	INTEGER	Nummer der zugehörigen Tabelle
colno	SMALLINT	fortlaufende Nummer der Spalte in der Tabelle
coltype	SMALLINT	Datentyp der Spalte Die möglichen Inhalte sind in der in der Include-Datei sqltypes.h definiert.
collength	SMALLINT	Länge der Spalte Bei einer VARCHAR-Spalte sind die minimale und maximale Länge folgendermaßen codiert: (*min* * 256) + *max*

Die folgenden Spalten treten nur bei INFORMIX-ONLINE auf:

colmin	INTEGER	zweitkleinster Wert der Spalte
colmax	INTEGER	zweitgrößter Wert der Spalte

Die Spalten colmin, colmax werden nur belegt, wenn ein Index auf der Spalte definiert ist. UPDATE STATISTICS aktualisiert die Werte, die INFORMIX-ONLINE zur Optimierung von SELECT-Anweisungen benutzt. Ist der Datentyp der Spalte nicht INTEGER, so werden die Werte in INTEGER-Werte umgewandelt.

Systemtabelle sysindexes

Spaltenname	Datentyp	Bedeutung
idxname	CHAR(18)	Indexname
owner	CHAR(8)	Benutzerkennung des Indexeigentümers
tabid	INTEGER	Nummer der zugehörigen Tabelle
idxtype	CHAR(1)	Indextyp U = Eindeutiger Index D = Duplikate zugelassen
clustered	CHAR(1)	CLUSTER-Attribut – kein CLUSTER-Index C = CLUSTER-Index
part1	SMALLINT	Nummer der Spalte, für die der Index aufgebaut ist. Erstreckt sich ein Index über mehrere Spalten, so sind in part1 bis part8 die einzelnen Spaltennummern enthalten. Für absteigende Indizes hat die Spaltennummer ein negatives Vorzeichen
part2	SMALLINT	
part3	SMALLINT	
part4	SMALLINT	
part5	SMALLINT	
part6	SMALLINT	
part7	SMALLINT	
part8	SMALLINT	

Die folgenden Spalten treten nur bei INFORMIX-ONLINE auf:

part9	SMALLINT	zusätzliche Spaltennummern für zusammengesetzte Indizes von INFORMIX-ONLINE
:		
part16	SMALLINT	
levels	SMALLINT	Anzahl der B-Tree Ebenen
leaves	INTEGER	Anzahl der Blätter des B-Tree
nunique	INTEGER	Anzahl der eindeutigen Indexwerte
clust	INTEGER	z.Z. nicht belegt

Die Indexinformationen werden bei UPDATE STATISTICS aktualisiert.

Systemtabelle systabauth

Spaltenname	Datentyp	Bedeutung
grantor	CHAR(8)	Eigentümer des Zugriffsrechts
grantee	CHAR(8)	Benutzerkennung, für die das Zugriffsrecht gilt
tabid	INTEGER	Nummer der zugehörigen Tabelle
tabauth	CHAR(7)	Zugriffsrechte, die von *grantor* an *grantee* vergeben wurden. Jede Stelle in dieser Spalte entspricht einem bestimmten Zugriffsrecht. Ein Bindestrich bedeutet, daß die Zugriffsart nicht erlaubt ist. Ein Stern * an der 3. Stelle bedeutet, daß Zugriffsrechte spaltenweise vergeben sind (siehe Tabelle syscolauth).

Stelle	Zugriffsrechte	Wert
1	SELECT-Zugriffsrecht auf alle Spalten	s
2	UPDATE-Zugriffsrecht auf alle Spalten	u
3	SELECT- oder UPDATE-Zugriffsrecht auf Spalten eingeschränkt	*
4	INSERT-Zugriffsrecht	i
5	DELETE-Zugriffsrecht	d
6	INDEX-Zugriffsrecht	x
7	ALTER-Zugriffsrecht	a

Systemtabelle syscolauth

Spaltenname	Datentyp	Bedeutung
grantor	CHAR(8)	Eigentümer des Zugriffsrechts
grantee	CHAR(8)	Benutzerkennung, für die das Zugriffsrecht gilt
tabid	INTEGER	Nummer der zugehörigen Tabelle
colno	SMALLINT	zugehörige Spaltennummer
colauth	CHAR(2)	Zugriffsrechte

Stelle	Zugriffsrechte	Wert
1	SELECT-Zugriffsrecht wird auf die Spalte eingeschränkt	s
2	UPDATE-Zugriffsrecht wird auf die Spalte eingeschränkt	u

Systemtabelle sysdepend

Spaltenname	Datentyp	Bedeutung
btabid	INTEGER	Nummer der Basistabelle oder des Views
btype	CHAR(1)	Tabellentyp T = Basistabelle V = View
dtabid	INTEGER	Nummer der abhängigen Tabelle
dtype	CHAR(1)	Tabellentyp der abhängigen Tabelle V = View

Systemtabelle syssynonyms

Die Systemtabelle *syssynonyms* wird nicht mehr verwendet. Die Synonyme werden jetzt in der Tabelle *syssyntable* gespeichert. Die Tabelle *syssynonyms* ist aus Kompatibilitätsgründen weiterhin definiert.

Systemtabelle sysusers

Spaltenname	Datentyp	Bedeutung
username	CHAR(8)	Benutzerkennung, für die Zugriffsrecht gilt
usertype	CHAR(1)	Zugriffsrecht auf Datenbank-Ebene D = DBA R = RESOURCE C = CONNECT
priority	SMALLINT	z. Zt. nicht belegt
password	CHAR(8)	z. Zt. nicht belegt

Systemtabelle sysviews

Spaltenname	Datentyp	Bedeutung
tabid	INTEGER	Nummer der zugehörigen Tabelle
seqno	SMALLINT	Folgenummer
viewtext	CHAR(64)	Teil aus der SELECT-Anweisung

Systemtabelle sysconstraints

Spaltenname	Datentyp	Bedeutung
constrid	SERIAL	z. Zt. nicht belegt
constrname	CHAR(18)	Name des Constraint
owner	CHAR(8)	Eigentümer des Constraint
tabid	INTEGER	Nummer der zugehörigen Tabelle
constrtype	CHAR(1)	z. Zt. nicht belegt
idxname	CHAR(18)	Name des zugehörigen Index

Systemtabelle syssyntable

Spaltenname	Datentyp	Bedeutung
tabid	INTEGER	Nummer des zugehörigen Synonyms
servername	CHAR(18)	Name des INFORMIX-ONLINE-Systems bei externen Tabellen
dbname	CHAR(18)	Datenbankname bei externen Tabellen (nur INFORMIX-ONLINE)
owner	CHAR(8)	Eigentümer der externen Tabelle (nur INFORMIX-ONLINE)
tabname	CHAR(18)	Name der externen Tabelle (nur INFORMIX-ONLINE)
btabid	INTEGER	Nummer der zum Synonym gehörigen internen Tabelle

A.3 Umgebungsvariable

Sie können die "Umgebung" beeinflussen, in der INFORMIX arbeitet. Dies geschieht über Variablen, deren Inhalt Sie individuell verändern können und die INFORMIX anschließend entsprechend auswertet. Die folgende Tabelle faßt alle von INFORMIX verwendeten Umgebungsvariablen kurz zusammen:

Umgebungsvariable	Bedeutung
DBANSIWARN	Überwachung des ANSI-Standards
DBDATE	Ein- und Ausgabeformat für das Datum
DBDELIMITER	Trennzeichen definieren
DBEDIT	Texteditor bestimmen
DBMONEY	Ausgabeformat für Geldbeträge
DBPATH	Pfade für Datenbanken und Dateien
DBPRINT	Ausgabeprogramm definieren
DBTEMP	Dateiverzeichnis für temporäre Dateien
DBLANG	Dateiverzeichnis für Meldungsdateien
INFORMIXDIR	Dateiverzeichnis für INFORMIX-Dateien
INFORMIXTERM	Bildschirmsteuerung termcap/terminfo
SQLEXEC	INFORMIX-Backend bestimmen
TERMCAP	Termcap-Datei bestimmen
TBCONFIG	*tbconfig*-Datei für das INFORMIX-ONLINE-System

Im folgenden werden nur die Umgebungsvariablen beschrieben, die Auswirkungen auf SQL-Anweisungen haben. Im Handbuch für das jeweilige Frontend-Produkt [1, 3, 4, 6] finden Sie die Beschreibung der übrigen Umgebungsvariablen.

Setzen von Umgebungsvariablen

Umgebungsvariablen setzen Sie in der Shell. Beachten Sie, daß Sie Umgebungsvariablen in Großbuchstaben angeben müssen.

Beispiel (Bourne-Shell):

```
DBTEMP=/abc
export DBTEMP
```

Beispiel (C-Shell):

```
setenv DBTEMP /abc
```

Wenn Sie diese Eingaben nicht nach jedem LOGIN neu vornehmen wollen, dann müssen Sie sie in die Datei *.profile* bzw. *.login* eintragen.

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 gegenüber dem ANSI-Standard. DBANSIWARN wird kein Wert zugewiesen.

Umgebungsvariable DBDATE

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 Zeichen, 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: 12-05-87

Umgebungsvariable DBDELIMITER

DBDELIMITER vereinbart das Zeichen, das die Anweisungen LOAD und UNLOAD als Trennzeichen zwischen den Feldern der Ein- bzw. Ausgabedatei benutzen.

Standardwert: `DBDELIMITER ist |`

| = senkrechter Strich (ASCII 124)

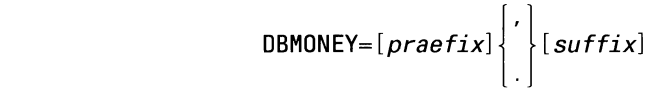
Beachten Sie, daß das Zeichen ö im deutschen ISO-7-Bit-Code dem senkrechten Strich entspricht.

Umgebungsvariable DBMONEY

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:



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*.

,
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

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 einer INFORMIX-SE-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 DBTEMP

DBTEMP bestimmt das Dateiverzeichnis, das die temporären Dateien aufnimmt.

Standardwert: `DBTEMP=/tmp`

Temporäre Dateien werden im Dateiverzeichnis /tmp abgelegt.

Umgebungsvariable INFORMIXDIR

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 TBCONIFG

Die Umgebungsvariable TBCONFIG gibt das INFORMIX-ONLINE-System an, wenn mehrere Systeme auf einem Rechner initialisiert sind.

Standardwert: `$INFORMIXDIR/etc/tbconfig`

A.4 Tabellengröße schätzen

Bei INFORMIX-ONLINE können Sie die Speicherplatzausnutzung für eine Tabelle optimieren, wenn Sie die Extent-Größen vernünftig angeben. Beim Erzeugen der Tabelle mit CREATE TABLE können Sie den Initial-Extent und die Größe für die weiteren Extents angeben. Die Größe für die weiteren Extents kann mit ALTER TABLE verändert werden.

Der Standardwert für beide Extent-Größen ist 8 Pages (16 KByte). Verwenden Sie den Standardwert für große Tabellen, so wird möglicherweise die maximale Extent-Anzahl überschritten.

Der Speicherplatz für den Initial-Extent einer Tabelle wird beim Erzeugen der Tabelle sofort im Dbspace reserviert. Sie sollten diesen Speicherplatz voll ausnutzen, da er exklusiv für die Tabelle reserviert ist.
Der Speicherplatz für weitere Extents wird im Dbspace dynamisch reserviert. Der Speicherplatz wird erst angefordert, wenn die bisher angelegten Extents gefüllt sind. Die automatische Reservierung von Extents ist durch die maximale Extent-Anzahl begrenzt.
Der Wert für die Größen der Extents sollte weder zu groß noch zu klein sein: zu große Extents, die nicht ganz gefüllt werden können, vergeuden Speicherplatz; zu kleine Extents können den Zugriffs-Overhead vergrößern.

INFORMIX-ONLINE kann Tabellen mit bis zu acht Extents ohne zusätzlichen Aufwand verwalten. Sie sollten deswegen dafür sorgen, daß keine Tabelle mehr als acht Extents benötigt.

Maximale Extent-Anzahl

Die maximale Extent-Anzahl wird durch die Spalten und Indizes der Tabelle beeinflußt. Die maximale Extent-Anzahl kann wie folgt berechnet werden:

```
1964 (Page-Größe minus interne Daten)

minus Anzahl Spalten * 4
minus Anzahl Indizes * 12
minus Anzahl Indexspalten * 4
-----------------------------------------------
= freier Platz für Extent-Verwaltung

maximale Extent-Anzahl = (freier Platz für Extent-Verwaltung) / 8
```

Wird die maximale Extent-Anzahl überschritten, so muß die Tabelle neu erzeugt werden. Dies kann sehr zeitaufwendig sein, da alle Sätze der Tabelle entladen und neu geladen werden müssen. Folgende Schritte sind in diesem Fall durchzuführen:

- Tabelle entladen (UNLOAD-Anweisung)
- Tabelle löschen (DROP TABLE-Anweisung)
- Tabelle mit neuem Wert für Initial-Extent und Next-Extent neu erzeugen (CREATE TABLE-Anweisung)
- Tabelle neu laden (LOAD-Anweisung)

Um das Erreichen der maximalen Extent-Anzahl zu vermeiden, verdoppelt INFORMIX-ONLINE automatisch die Größe der weiteren Extents, wenn die Nummer des Extent ein Vielfaches von 64 ist.

Extent-Größe schätzen

Um die erforderliche Größe des Initial-Extent und der weiteren Extents schätzen zu können, müssen Sie die Länge der Sätze in der Tabelle und die Anzahl der Sätze bestimmen. Aus diesen Zahlen kann dann die erforderliche Größe der Extents berechnet werden.

Speicherbedarf für einen Satz berechnen

Bevor Sie die Größe des Initial-Extent und der weitereren Extents berechnen, müssen Sie zunächst den Speicherbedarf für einen Satz der Tabelle berechnen.
Ein Satz benötigt Speicherplatz für Spalten und für Indexeinträge. Die Länge eines Satzes berechnet sich aus der Summe der einzelnen Spaltenlängen.

Die Länge der Spalten hängt vom definierten Datentyp ab und ist in Kapitel 3 beim jeweiligen Datentyp beschrieben. Die Länge einer Spalte der Tabelle *tabelle* kann aus der Systemtabelle *syscolumns* abgefragt werden.

```
SELECT collength FROM syscolumns WHERE colname="spaltenname"
       AND tabid= (SELECT tabid FROM systables
                           WHERE tabname="tabelle")
```

Die Länge eines Indexeintrags ergibt sich aus der Länge der indizierten Spalten plus acht Byte für die Adressierung des dazugehörigen Satzes.

Satzlänge bei VARCHAR-Spalten

Enthält die Tabelle eine VARCHAR-Spalte, so kann keine exakte Satzlänge und Länge eines Indexeintrags bestimmt werden. Sie müssen die mittlere Länge der VARCHAR-Spalte schätzen. Wenn eine minimale Länge für die VARCHAR-Spalte definiert wurde, müssen Sie bei der Schätzung der mittleren Spaltenlänge beachten, daß mindestens die definierte Minimallänge belegt wird. Für jede VARCHAR-Spalte benötigt INFORMIX-ONLINE zusätzlich ein Byte Speicherplatz.

Satzlänge bei TEXT- und BYTE-Spalten

Die Länge einer TEXT- oder BYTE-Spalte ist innerhalb des Satzes immer 28 Byte. Wenn die TEXT- oder BYTE-Spalte getrennt von der Tabelle in einem Blobspace gespeichert wird, wird die Größe der gespeicherten TEXT- oder BYTE-Daten nicht zur Extent-Berechnung herangezogen.
Werden die TEXT- oder BYTE-Daten im Tblspace gespeichert (IN TABLE), so müssen diese Spalten bei der Berechnung der Satzlänge berücksichtigt werden. Für jede TEXT- oder BYTE-Spalte werden 28 Byte zur Satzlänge addiert. Die durchschnittliche Belegung der Spalten muß geschätzt werden. Berücksichtigen Sie dabei, daß die Daten in TEXT- oder BYTE-Spalten in ganzen Page-Einheiten gespeichert werden.

Speicherplatz für Verwaltungsinformationen

Zusätzlich zum Platzbedarf, der sich aus der Speicherung der Daten ergibt, müssen Sie noch Speicherplatz für Verwaltungsinformationen berücksichtigen:

- Verwaltung der Pages
- Anpassung der Datensätze an die Page-Größe
- B-Tree für Indizes

Dieser zusätzliche Platzbedarf für die Verwaltungsinformationen ist in der Berechnung der Extentgröße bereits enthalten.

Berechnung der Extent-Größe

Schritt		
1	Länge jedes Index bestimmen	8 Byte zu jeder Indexlänge addieren
2	Gesamtlänge der Indizes bestimmen	alle Indexeinträge aus Schritt 1 addieren
3	Verwaltungsinformation für die Indizes berücksichtigen	25 % zur Summe aus Schritt 2 addieren
4	Initial-Extent schätzen	anfängliche Anzahl der Tabellensätze schätzen
5	Platzbedarf für Indizes bestimmen	Ergebnis von Schritt 3 mit der Satzanzahl aus Schritt 4 multiplizieren
6	Platzbedarf in KByte umrechnen	Ergebnis aus Schritt 5 durch 1024 teilen
7	verfügbare Page-Größe berechnen	2048 minus 32
8	Satzlänge bestimmen	Spaltenlängen addieren
9	Zahl der Sätze bestimmen, die in eine Page passen	verfügbare Page-Größe aus Schritt 7 durch Satzlänge aus Schritt 8 dividieren
10	Page-Anzahl für Daten im Initial-Extent bestimmen	Satzanzahl aus Schritt 4 durch Satzanzahl pro Page aus Schritt 9 dividieren; aufrunden
11	Speicherplatz für Daten im Initial-Extent bestimmen	Page-Anzahl aus Schritt 10 mit der Größe einer vollen Page multiplizieren
12	in KByte umrechnen	Ergebnis aus Schritt 11 durch 1024 teilen

Schritt		
13	Größe des Initial-Extent in KByte bestimmen	Ergebnis aus Schritt 12 zum Ergebnis aus Schritt 6 addieren
14	Tabellenwachstum schätzen	Anzahl der Sätze schätzen, die zur Tabelle hinzugefügt werden
15	Größe der weiteren Extents bestimmen	die Schritte 5 bis 6 für den Platzbedarf von Indizes und die Schritte 10 bis 13 für den Platzbedarf von Daten wiederholen. Die Schätzung aus Schritt 14 wird als Satzanzahl in den Schritten 5 und 10 verwendet. Das Endergebnis durch 7 dividieren.

Beispiel

Im folgenden Beispiel wird die Extent-Größe für die Tabelle *posten* der Beispieldatenbank *versand* berechnet.

Tabelle *posten*:

Spalte	Datentyp	Länge in Byte
posten_nr	SMALLINT	2
auftrags_nr	INTEGER	4
artikel_nr	SMALLINT	2
herstellercode	CHAR(3)	3
menge	SMALLINT	2
gesamtpreis	MONEY(8)	5

Für die Tabelle *posten* ist ein einfacher Index für *auftrags_nr* und ein zusammengesetzter Index für *artikel_nr* und *herstellercode* definiert.

Im folgenden Beispiel wird angenommen, daß die Tabelle anfangs 20000 Sätze enthalten soll und bis auf 55000 Sätze wachsen wird.

```
1. Länge jedes Index bestimmen
   Index für auftrags_num: 4 + 8 = 12
   Index für artikel=nr und herstellercode: 2 + 3 + 8 = 13

2. Gesamtlänge der Indizes bestimmen
   Gesamtlänge: 12 + 13 = 25

3. Verwaltungsinformation für die Indizes berücksichtigen
   bereinigte Indexlänge: 25 * 1,25 = 31,3

4. anfängliche Anzahl der Tabellensätze schätzen
   geschätzte Satzzahl: 20000

5. Platz für Indizes bestimmen
   Satzanzahl * bereinigte Indexlänge: 20000 * 31,3 = 626000 Byte

6. Platzbedarf in KByte umrechnen: 626000 / 1024 = 612 KByte

7. verfügbare Page-Größe berechnen: 2048 - 32 = 2016

8. Satzlänge bestimmen
   Satzlänge: 2 + 4 + 2 + 3 + 2 + 5 = 18 Byte

9. Zahl der Sätze bestimmen, die in eine Page passen
   Satzanzahl pro Page: 2016 / 18 = 112

10. Page-Anzahl für Daten im Initial-Extent bestimmen
    20000 / 112 = 178,6 —> 179

11. Speicherplatz für Daten im Initial-Extent bestimmen
    Speicherplatz für Daten-Pages: 179 * 2048 = 366592 Byte

12. in KByte umrechnen: 366592 / 1024 = 358 KByte

13. Größe des Initial-Extent in KByte bestimmen
    Größe des Initial- Extent: 358 + 612 = 970 KByte

14. Tabellenwachstum schätzen
    nachträglich hinzugefügte Sätze: 35000

15. Größe der weiteren Extents bestimmen

       Satzanzahl * bereinigte Indexlänge: 35000 * 31,3 = 1095500
       Byte Platzbedarf in KByte umrechnen: 1095500 / 1024 = 1070 KByte
       Page-Anzahl für Daten: 35000 / 112 = 313
       Speicherplatz für Daten-Pages: 313 * 2048 = 641024 Byte
       Platzbedarf in KByte umrechnen: 641024 / 1024 = 626 KByte
       Platzbedarf für Daten und Index: 626 + 1070 KByte =1696 KByte

       Größe eines weiteren Extent: 1696 / 7 = 243 KByte
```

A.5 Reservierte Wörter

Die folgende Liste enthält Wörter, die als Schlüsselwörter für die INFORMIX-Produkte reserviert sind. Werden die SQL-Anweisungen in eine Programmiersprache eingebettet, sind zusätzlich die reservierten Wörter der jeweiligen Programmiersprache zu beachten.

Nicht jedes der folgenden Wörter ist in allen INFORMIX-Produkten reserviert. Sie sollten aber aus Gründen der Portabilität z.B. 4GL-Schlüsselwörter nicht in ESQL/C-Programmen verwenden, obwohl diese im ESQL/C-Programm nicht zu einem Fehler führen.

abort
absolute
accept
access
add
after
all
allowing
alter
and
ansi
any
array
as
asc
ascending
ascii
at
attribute
attributes
audit
auto
autonext
average
avg
background
before
begin
beginning
bell
between
black
blanks
blink
blue
bold
border
bottom
buffered
by
byte
call
case
char
character
check
clear
clipped
close
cluster
col
color
colors
column
columns
command
comment
comments
commit

committed
composites
compress
connect
constant
constraint
construct
continue
convert
count
create
current
cursor
cyan
database
date
datetime
date_type
day
dba
debug
dec
dec_t
decimal
decimal_type
declare
default
defaults
defer
define
delete
delimiter
delimiters
desc
descending
describe
descriptor
dim
dirty
display
displayonly
distinct
do
dominant
double
down
downshift
drop
dtime
dtime_t
eco-*
editadd
editupdate
else
end
end-exec
endif
ending
error
escape
every
exclusive
exec
execute
exists
exit
exitnow
exits
explain
extend
extent
extern
external
false
fetch
field
file
finish
first
fixchar
float
flush

for
foreach
form
format
formonly
found
fraction
free
from
function
globals
go
go to
goto
grant
green
group
having
header
headings
help
hold
hour
identified
if
ifdef
ifndef
immediate
in
include
index
indicator
infield
info
initialize
input
insert
instructions
int
integer
interrupt
intersect
interval
into
intrvl_t
inverse
invisible
is
isam
isolation
join
joining
key
label
last
left
len
length
let
like
line
lineno
lines
load
locate
locator
lock
loc_t
log
long
long_float
long_integer
lookup
loop
magenta
main
margin
master
matches
max
mdy
memory

menu
message
min
minus
minute
mod
mode
modify
module
money
month
name
natural
need
new
next
nextfield
no
nocr
noentry
normal
not
not found
notfound
noupdate
now
null
numeric
of
off
on
open
option
options
or
order
otherwise
out
outer
output
package
page
pageno
param
pause
percent
perform
picture
pipe
positive
precision
prepare
previous
print
printer
prior
privilege
privileges
program
prompt
public
put
query
queryclear
quit
raise
range
read
readonly
real
record
recover
red
register
relative
remove
rename
repair
repeatable
report
required
resource

return
returning
reverse
revoke
right
rollback
rollforward
row
rowid
rows
run
savepoint
screen
scroll
second
section
select
serial
serial_type
set
share
shift
short
short_float
short_integer
sitename
size
skip
sleep
smallfloat
smallint
some
space
spaces
sql
sql*
sqlca
sqlchar_type
sqlda
sqldecimal_type
sqlerr
sqlerror
sqlfloat_type
sqlint_type
sqlmoney_type
sqlsmfloat_type
sqlsmint_type
sqlwarning
stability
start
startlog
static
statistics
status
stdv
step
stop
string
struct
subtract
subtype
sum
synonym
systab|es
table
tables
temp
text
then
through
thru
time
tiny_integer
to
today
top
total
trailer
trailing
true
type
typedef

undef
underline
union
unique
units
unload
unlock
up
update
upshift
user
using
validate
values
varchar
variable
vc_t
verify
view
wait
waiting
warning
weekday
when
whenever
where
while
white
window
with
without
wordwrap
work
wrap
year
yellow
yes
zerofill

Literatur

Die mit * gekennzeichneten Titel sind nicht von Siemens herausgegeben.

[1] Betriebssystem SINIX
INFORMIX-SQL
Datenbanksystem
Nachschlagen

[2] Betriebssystem SINIX
INFORMIX-SQL
Datenbanksystem
Kennenlernen

[3] Betriebssystem SINIX
INFORMIX-ESQL/C
C-Schnittstelle für das
Datenbanksystem INFORMIX

[4] Betriebssystem SINIX
INFORMIX-4GL
Nachschlagen

[5] Betriebssystem SINIX
INFORMIX-4GL
Kennenlernen

[6] Betriebssystem SINIX
INFORMIX-ESQL/COBOL
COBOL-Schnittstelle für das
Datenbanksystem INFORMIX

[7] Betriebssystem SINIX
INFORMIX-NET
Netzkomponente für INFORMIX-SE

[8] Betriebssystem SINIX
INFORMIX-STAR
Netzkomponente für INFORMIX-ONLINE

[9] Betriebssystem SINIX
Fehlermeldungen für
INFORMIX-Produkte

[10] Betriebssystem SINIX
INFORMIX-ONLINE
Datenbank-Server
Administrator-Handbuch

[11] Betriebssystem SINIX
Systemverwalter-Handbuch

[12] 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

abfragen
- BYTE-Spalte 4-33
- Daten 6-179
- Daten, Sperre 6-181
- TEXT-Spalte 4-29

abhängige Tabelle 5-69, 6-187
abschließen, Transaktion 2-34
ABSOLUTE 6-110
Addition, Ausdruck 5-41
änderbarer View 2-14
ändern
- BYTE-Spalte 4-33
- Datentyp 6-24
- Extent-Größe 6-26
- Index 2-24
- Index (ALTER INDEX) 6-16
- Name 3-7
- Spalte 6-24
- Spaltenname (RENAME COLUMN) 6-167
- Spaltenwert (UPDATE) 6-228
- Tabellenstruktur (ALTER TABLE) 6-21
- TEXT-Spalte 4-30
- Transaktionsprotokollierung 6-49, 6-217
- Transaktionssicherung 6-49
- VARCHAR-Spalte 4-8

aktualisieren
- Datenbanksicherung 6-179
- Tabelle 6-165

aktuelle Datenbank 2-6
aktueller
- Benutzer, USER 5-30
- Satz (FETCH) 6-111

aktuelles
- Datum, CURRENT 5-10
- Datum, TODAY 5-29
- Informixsystem, SITENAME 5-26

ALL, Operator 5-49
ALL-Klausel
- AVG() 5-5
- MAX() 5-19
- MIN() 5-22
- SUM() 5-27

ALL-Zugriffsrecht 6-123, 6-172
alphanumerische
- Konstante 4-39
- Werte, Vergleich 5-46

alphanumerischer
- Datentyp 4-6, 4-7
- Wert 4-39
- Wert, Berechnung 4-41
- Wert, eintragen 4-39
- Wert, Funktion 4-41
- Wert, Prädikat 4-41
- Wert, vergleichen 4-41

ALTER INDEX 6-18
ALTER TABLE 6-21
Alter, Tabellenzugriffsrecht 2-28, 6-124, 6-172
AND, Operator 5-62
angeben
- Funktionsargument 5-4
- Tabelle 6-188

anpassen, Komponente 4-52
ANSI, voreingestellte Tabellenzugriffsrechte 2-29
ANSI-Datenbank 2-7, 6-49, 6-219
- Datenbankzugriffsrecht 2-27
- erstellen 2-7
- erzeugen 6-45
- Transaktionssicherung 2-39

ANSI-Standard überprüfen A-24
Anweisung
- analysieren (DESCRIBE) 6-93
- ausführen (EXECUTE) 6-107
- ausführen (EXECUTE IMMEDIATE) 6-109
- dynamisch 2-58, 6-209
- dynamisch (DESCRIBE) 6-93
- dynamisch (EXECUTE

IMMEDIATE) 6-109
– dynamisch (EXECUTE) 6-107
– dynamisch (PREPARE) 6-156
– dynamisch formuliert 2-58
– vorbereiten 2-58
– vorbereiten (PREPARE) 6-156
Anweisungsbezeichner freigeben (FREE) 6-119
ANY, Operator 5-48
Argument, Funktion 5-4
arithmetisches Mittel, AVG() 5-5
Audit-Datei
– Aufbau 6-42
– Zugriffsrechte 6-41
Audit-Protokoll 6-42
– erzeugen (CREATE AUDIT) 6-40
– löschen (DROP AUDIT) 6-96
Aufbau, Audit-Datei 6-42
aufheben, Tabellensperre 6-227
Ausdruck 5-38
– Addition 5-41
– berechnen 5-38
– Bereichsabfrage 5-50
– Datum 4-47
– Division 5-40
– Elementabfrage 5-53
– Funktionsargument 5-4
– Funktionsaufruf 5-39
– Hostvariable 5-38
– Konstante 5-38
– Multiplikation 5-40
– Mustervergleich 5-55
– NULL 4-38
– NULL-Wert 5-38
– numerischer Wert 4-43
– Operand 5-38
– Operator 5-38
– Prädikat 5-38
– Priorität 5-42
– Spalte 5-39
– Spaltenauswahl 5-38
– Subtraktion 5-41
– Unterabfrage 5-39, 5-48
– Vergleichsprädikat 5-45, 5-48
– Zeitpunkt 4-51
– Zeitspanne 4-57
– Zuweisung 5-38
ausführen
– Anweisung 6-107, 6-109
– dynamisch formulierte Anweisung 2-59
– Funktionsaufruf 5-3
– Transaktion 2-36
Ausgabeformat, MONEY 4-16
ausschließen, NULL-Wert 2-22
auswählen
– Ergebnissätze 6-193
– Ergebnisspalten 6-185
– Gruppen 6-199
auswerten
– Ausdruck 5-38
– Bedingung 5-61
– Prädikat 5-43
Auswertung, Unterabfrage 5-37
AVG()
– ALL-Klausel 5-5
– arithmetisches Mittel 5-5
– DISTINCT-Klausel 5-5
– GROUP BY-Klausel 5-5
– Mengenfunktion 5-5

Basistabelle 2-9
– bearbeiten 2-10
– definieren 2-10
– externe 2-11
– löschen 2-10
BCHECK, Dienstprogramm 6-33, 6-171
bearbeiten
– Basistabelle 2-10
– Datenbank 2-6
– Spalte 2-20
– temporäre Tabelle 2-16
– View 2-13
Bedingung 5-61
– auswerten 5-61
– DELETE 5-61
– HAVING-Klausel 5-61
– Join 5-65
– NULL 4-38
– NULL-Wert 5-61

– Operator 5-61
– Prädikat 5-43, 5-61
– Priorität 5-63
– SELECT 5-61
– UPDATE 5-61
– Wahrheitswert 5-61
– WHERE (SELECT) 6-194
– WHERE-Klausel 5-61
beenden, Transaktion 6-38
BEGIN WORK 6-31
– Transaktionsklammer 2-34
beginnen, Transaktion 6-31
Beispieldatenbank A-1
benannte temporäre Tabelle 6-204
benennen, Ergebnistabelle 2-16, 6-204
Benutzer 2-26
– aktuell 5-30
– Datenbankzugriffsrecht 2-26
Benutzerklasse 2-26
Benutzername 3-4
berechnen
– Ausdruck 5-38
– Prädikat 5-43
– Satz-Größe A-30
Berechnung
– alphanumerischer Wert 4-41
– Datum 4-47
– NULL-Wert 4-38
– numerischer Wert 4-43
– Zeitpunkt 4-51
– Zeitspanne 4-57
Bereichsabfrage, Prädikat 5-50
Beschreibungsformat, Anweisungen 6-12
Betriebsmittel freigeben (FREE) 6-119
BLOB-Daten 2-21, 4-4
BLOB-Datentyp 4-28, 4-32
BLOB-Page 2-5
Blobspace 2-5
BYTE-Datentyp 4-32
BYTE-Spalte 4-32
– abfragen 4-33
– ändern 4-33
– eintragen 4-33
– löschen 4-34
– verwenden 4-33
CHARACTER-Datentyp 4-6
CHARACTER-Spalte 4-6
CHECK OPTION 6-73
CHECK TABLE 6-33
Chunk 2-5
CLOSE 6-33
CLOSE DATABASE 6-36
Cluster-Attribut 6-18, 6-51
Cluster-Index 2-23
CLUSTER-Klausel, Index 2-24
COMMIT WORK 6-38
– Transaktionsklammer 2-34
Committed Read 6-213
– Isolationsstufe 2-43, 6-183
Connect-Datenbankzugriffsrecht 2-26, 6-125, 6-173
Constraint
– Datenintegrität 2-47
– definieren 2-25
– Definition 2-25
– Eindeutigkeitsbedingung 2-25
– einfach 2-25
– erzeugen 6-65
– löschen 2-25, 6-25
– Name 6-26, 6-68
– Index 2-25
– zusammengesetzt 2-25
COUNT()
– Duplikat 5-8
– Elemente zählen 5-8
– GROUP BY-Klausel 5-8
– Mengenfunktion 5-8
– NULL-Wert 5-8
COUNT(*)
– Elemente zählen 5-7
– GROUP BY-Klausel 5-7
– Mengenfunktion 5-7
CREATE AUDIT 6-40
CREATE DATABASE 6-44, 6-48
CREATE INDEX 6-51
CREATE SCHEMA 6-56
CREATE SYNONYM 6-59
CREATE TABLE 6-63
CREATE VIEW 6-72

CURRENT 6-112
– aktuelles Datum 5-10
– DATETIME 5-10
– Funktion 4-51
– Zeitfunktion 5-10
Cursor Stability 6-214
– Isolationsstufe 2-43, 6-183

DATABASE 6-77
DATE()
– Datum erzeugen 5-12
– Funktion 4-41
– Zeitfunktion 5-12
DATE-Datentyp 4-20
DATE-Spalte 4-20
Datei
– Index 2-3
– Tabelle 2-3
Dateiname 3-4
Dateiverzeichnis
– für Datenbanken A-27
– für INFORMIX-Dateien A-28
– für temporäre Datei A-27
– Datenbank 2-3
Daten abfragen (SELECT) 6-181
Datenbank
– aktuelle 2-6
– ANSI 2-7
– auf ANSI, wechseln 2-8
– bearbeiten 2-6
– definieren 2-6
– Definition 2-2
– entfernte 2-8
– eröffnen (DATABASE) 6-77
– erzeugen (CREATE DATABASE) 6-44, 6-48
– fremde 2-6
– interne Organisation 2-2
– löschen 2-6
– löschen (DROP DATABASE) 6-97
– mit Transaktionssicherung 2-38
– öffnen 2-6
– ohne Transaktionssicherung 2-37
– Qualifikation 3-12
– schließen 2-7
– schließen (CLOSE DATABASE) 6-36
– Sperre 2-40
– wechseln 2-7
– und Tabellenzugriffsrecht 2-30
Datenbankobjekt 2-2
– Name 3-4
Datenbankschema erzeugen (CREATE SCHEMA) 6-56
Datenbanksicherung aktualisieren (ROLLFORWARD DATABASE) 6-179
Datenbanksperre, setzen 2-44
Datenbankverzeichnis 2-3
Datenbankzugriffsrecht 2-26, 2-28
– ANSI-Datenbank 2-27
– Benutzer 2-26
– Connect 2-26
– DBA 2-26
– PUBLIC 2-27
– voreingestellt 2-28
– zuteilen 2-27
Datenbankzustand, konsistent 2-33
Datenintegrität 2-47
– Constraint 2-47
– Datentyp 2-47
– Index 2-47
– Sperren 2-47
– Transaktionssicherung 2-47
Datenkonvertierung, eindeutiger Index 6-28
Datenschutz 2-47
Datensicherheit 2-47
– Sperren 2-47
– Transaktionssicherung 2-47
– View 2-47
– Zugriffsrecht 2-47
Datensicherung 2-48
Datentyp 2-21
– ändern 6-24
– alphanumerisch 4-6, 4-7
– BLOB 4-28, 4-32
– BYTE 4-32
– CHARACTER 4-6
– DATE 4-20
– Datenintegrität 2-47

- DATETIME 4-21
- DECIMAL 4-14, 4-17
- definieren 4-4
- DOUBLE PRECISION 4-19
- Einteilung 4-3
- Festpunkt 4-14
- FLOAT 4-19
- Gleitpunkt 4-17, 4-19
- INTEGER 4-11
- INTERVAL 4-24
- Konvertierung 4-35
- MONEY 4-16
- NUMERIC 4-14, 4-17
- numerisch 4-10, 4-11, 4-12, 4-14, 4-16, 4-17, 4-18, 4-19
- REAL 4-18
- Realzahl 4-18
- SERIAL 4-12
- SMALLFLOAT 4-18
- SMALLINT 4-10
- TEXT 4-28
- Übersicht 4-5
- VARCHAR 4-7
- Wertebereich 4-4
- Zeit 4-20, 4-21, 4-24

DATETIME
- CURRENT 5-10
- Datentyp 4-21
- Schreibweise, Zeitspanne 4-48
- Spalte 4-22

Datum 4-45
- erzeugen ,DATE() 5-12
- erzeugen, MDY() 5-21
- aktuell 5-29
- Ausdruck 4-47
- Berechnung 4-47
- Ein/Ausgabeformat definieren A-25
- eintragen 4-46
- Ganzzahl 5-12, 5-14, 5-24, 5-31, 5-33
- Prädikat 4-47
- vergleichen 4-47
- Zeichenkette 5-12, 5-14, 5-24, 5-31, 5-33
- Zeitpunkt 5-12, 5-14, 5-24, 5-31, 5-33
- Zeitwert 4-44

Datumsformat 4-46
Datumsfunktion 4-47
Datumskonstante 4-45
DAY()
- Tag bestimmen 5-14
- Zeitfunktion 5-14

DBA, Datenbankzugriffsrecht 2-26, 6-126, 6-173
DBANSIWARN (Umgebungsvariable) A-24
DBDATE, (Umgebungsvariable) 4-46, A-25
DBDELIMITER (Umgebungsvariable) A-26
DBMONEY (Umgebungsvariable) 4-43, A-26
DBPATH (Umgebungsvariable) A-27
Dbspace 2-5, 6-48, 6-67, A-29
DBTEMP (Umgebungsvariable) A-27
DECIMAL-Datentyp 4-14, 4-17
DECIMAL-Spalte 4-14
DECLARE 6-80
definieren
- ANSI-Datenbank 2-7
- Basistabelle 2-10
- Constraint 2-25
- Datenbank 2-6
- Datentyp 4-4
- Index 2-24
- Name 3-6
- Satzzeiger 2-53, 6-80
- Spalte 2-20
- Synonym 2-18
- Synonym (CREATE SYNONYM) 6-59
- temporäre Tabelle 2-16
- View 2-12
- Warte-Modus 6-215

DELETE 6-88
Delete, Tabellenzugriffsrecht 2-28, 6-124, 6-172
DESCRIBE 6-93

Dienstprogramm BCHECK 6-33, 6-171
Dirty Read 6-213
– Isolationsstufe 2-43, 6-183
DISTINCT-Klausel
– AVG() 5-5
– MAX() 5-19
– MIN() 5-22
– SUM() 5-27
Division, Ausdruck 5-40
dominante Tabelle 6-189
dominierende Tabelle 5-69
DOUBLE PRECISION
– Datentyp 4-19
– Spalte 4-19
DROP AUDIT 6-96
DROP DATABASE 6-97
DROP INDEX 6-99
DROP SYNONYM 6-101
DROP TABLE 6-103
DROP VIEW 6-105
Duplikat, Count() 5-8
dynamisch formulierte Anweisung 2-58
– ausführen 2-59
– vorbereiten 2-58
dynamische Anweisung 2-58, 6-93, 6-107, 6-109, 6-156
– (SELECT) 6-209

Eigentümer
– Definition 2-29
– Index 2-23
– Qualifikation 3-10
– Tabellenzugriffsrechte 2-29
eindeutiger Index 2-23
– Datenkonvertierung 6-28
Eindeutigkeit, Name 3-8
Eindeutigkeitsbedingung
– Constraint 2-25
– Index 2-24
– Spalte 2-20
einfacher
– Constraint 2-25
– Join 5-65
– Name 3-5
einfügen
– Satz (INSERT) 6-135
– SERIAL-Spalte 6-27
– Spalte 6-22
eingebettet, SELECT 6-207
einschalten, Transaktionssicherung 6-49, 6-219
einspaltige Ergebnistabelle 5-35
einstellen
– Isolationsstufe 2-44
– Transaktionssicherung 2-39
einstelliger Operator 5-39
Einteilung
– Datentyp 4-3
– Werte 4-36
eintragen
– alphanumerischer Wert 4-39
– BYTE-Spalte 4-33
– Datum 4-46
– numerischer Wert 4-42
– TEXT-Spalte 4-29
– Zeitpunkt 4-50
– Zeitspanne 4-56
Einzelsperre 2-41
– Exclusive 2-42
– setzen 2-45
– Share 2-42
– Update 2-42
Elementabfrage, Prädikat 5-53
Elemente zählen
– COUNT() 5-8
– COUNT(*) 5-7
entfernte Datenbank 2-8
Entwertungszeichen 5-55
entziehen, Zugriffsrechte 2-29, 6-172
Erfolgskontrolle 2-60
Ergebnissätze
– gruppieren, (SELECT/GROUP BY) 6-197
– auswählen (SELECT/Spaltenauswahl) 6-185
– auswählen (SELECT/WHERE) 6-193
Ergebnistabelle 2-16, 6-181
– benennen, (SELECT/INTO TEMP) 6-204
– sortieren, (SELECT/ORDER BY)

6-201
– benannt 2-16
– einspaltig 5-35
erstellen, ANSI-Datenbank 2-7
erzeugen
– ANSI-Datenbank 6-45
– Audit-Protokoll (CREATE AUDIT) 6-40
– Constraint 6-65
– Datenbank (CREATE DATABASE) 6-44, 6-48
– Index (CREATE INDEX) 6-51
– Synonym (CREATE SYNONYM) 6-59
– Tabelle (CREATE TABLE) 6-63
– temporäre Tabelle 6-64
– View (CREATE VIEW) 6-72
ESCAPE-Klausel, Mustervergleich 5-56
Exclusive-Einzelsperre 2-42
– und Transaktion 2-36
EXCLUSIVE-Sperre 6-146
Exclusive-Tabellensperre 2-40
EXECUTE 6-107
EXECUTE IMMEDIATE 6-109
Existenzabfrage , Prädikat 5-60
EXTEND(), Zeitfunktion 5-16
– anpassen 5-16
EXTEND, Funktion 4-51
Extent 2-5
EXTENT SIZE-Klausel 6-67
Extent-Größe 6-67
– ändern 6-26
– schätzen A-29
externe
– Basistabelle 2-11
– Tabelle 2-6, 2-11, 6-188
externer, View 2-13

festlegen
– Sperrbereich 6-25, 6-68
– Warte-Modus 6-215
Festpunkt-Datentyp 4-14
Festpunkt-Spalte 4-14
Festpunktzahl 4-42
festschreiben, Transaktion 2-34
FETCH 6-111
FIRST 6-112
FLOAT-Datentyp 4-19
FLOAT-Spalte 4-19
FLUSH 6-116
FOR UPDATE
– Satzzeiger (DECLARE) 6-81
– -Klausel, Satzzeiger 2-53
FREE 6-119
freigeben
– Anweisungsbezeichner 6-119
– Betriebsmittel 6-119
– Satzzeiger 6-119
– Speicherplatz 6-119
fremde Datenbank 2-6
FROM-Klausel (SELECT) 6-188
Funktion 5-3
– alphanumerischer Wert 4-41
– AVG() 5-5
– COUNT() 5-8
– COUNT(*) 5-7
– CURRENT 4-51, 5-10
– DATE 4-41
– DATE() 5-12
– Datum 4-47
– DAY() 5-14
– EXTEND 4-51
– EXTEND() 5-16
– LENGTH 4-41
– LENGTH() 5-18
– MAX() 5-19
– MDY() 5-21
– Menge 5-4, 5-5, 5-7, 5-8, 5-19, 5-22, 5-27
– MIN() 5-22
– MONTH() 5-24
– NULL 4-38
– numerischer Wert 4-43
– SITENAME 4-39, 5-26
– SUM() 5-27
– TODAY 4-46, 5-29
– USER 4-39, 5-30
– WEEKDAY() 5-31
– YEAR() 5-33
– Zeichenkette 5-18, 5-26, 5-30

– Zeichenketten 5-3
– Zeit 5-3, 5-10, 5-12, 5-14, 5-16, 5-21, 5-24, 5-29, 5-31, 5-33
– Zeitpunkt 4-51
– Zeitspanne 4-57
Funktionsargument 5-4
– angeben 5-4
– Ausdruck 5-4
– Konstante 5-4
– NULL 5-4
Funktionsaufruf 5-3
– Ausdruck 5-39
– ausführen 5-3

Ganzzahl 4-42
– Datum 5-12, 5-14, 5-24, 5-31, 5-33
Geldbetrag 4-43
– Ausgabeformat definieren A-26
gepufferte Transaktions-protokollierung 2-35, 6-217
geschachtelter Outer-Join 5-70
Gleitpunkt-Datentyp 4-17, 4-19
Gleitpunkt-Spalte 4-19
Gleitpunktzahl 4-42
GRANT 6-123
Grant-Berechtigung 6-125
GROUP BY-Klausel
– AVG() 5-5
– COUNT() 5-8
– COUNT(*) 5-7
– MAX() 5-19
– MIN() 5-22
– NULL-Wert 4-38
– (SELECT) 6-197
– SUM() 5-27
Gruppe 6-198
Gruppen, auswählen (SELECT/HAVING) 6-199
gruppieren, Ergebnissätze 6-197

halten Sperre 2-46
HAVING-Bedingung 6-200
HAVING-Klausel (SELECT) 6-199
hinzufügen, CONSTRAINT 6-25
Hostvariable 2-50
– Ausdruck 5-38

Index
– ändern 2-24
– ändern (ALTER INDEX) 6-18
– Cluster 2-23
– CLUSTER-Klausel 2-24
– Datenintegrität 2-47
– definieren 2-24
– Definition 2-23
– Eigentümer 2-23
– eindeutig 2-23
– Eindeutigkeitsbedingung 2-24
– erzeugen (CREATE INDEX) 6-51
– löschen 2-24
– löschen (DROP INDEX) 6-99
– normal 2-23
– sortieren (CREATE INDEX) 6-52
– Tabelle sortieren (ALTER INDEX) 6-18
– Tabellenzugriffsrecht 2-28
– und Constraint 2-25
– UNIQUE-Klausel 2-24
Index-Datei 2-3
Index-Zugriffsrecht 6-124, 6-172
Indikatorvariable 2-51
INFO 6-131
INFORMIX-ONLINE-Speicherbereich, Name 3-5
INFORMIX-STAR 6-216
INFORMIXDIR (Umgebungsvariable) A-28
Informixsystem, Qualifikation 3-13
Initial-Extent A-29
INSERT 6-135
Insert, Tabellenzugriffsrecht 2-28, 6-124, 6-172
Insert-Satzzeiger 2-52
– (CLOSE) 6-33
Insert-Satzzeiger, verwenden 2-57
INTEGER-Datentyp 4-11
INTEGER-Spalte 4-11
INTERVAL-Datentyp 4-24
INTERVAL-Schreibweise, Zeitspanne 4-54
INTERVAL-Spalte 4-26
INTO TEMP-Klausel (SELECT) 6-204

INTO-Klausel (SELECT) 6-208
Isolationsstufe 2-43, 6-183, 6-211
- Committed Read 2-43, 6-183
- Cursor Stability 2-43, 6-183
- Dirty Read 2-43, 6-183
- einstellen 2-44
- Repeatable Read 2-44, 6-183

Jahr
- bestimmen, YEAR() 5-33
- Komponente 4-48, 4-53, 5-10, 5-16

Join 5-65
- einfach 5-65
- normal 5-65, 5-66
- OUTER 5-65, 5-69
- self 5-65
- Typ 5-65
- Vergleich 5-65
- zusammengesetzt 5-65

Join-Bedingung 5-65
- einfach 5-65

Join-Spalte 5-65

Kartesisches Produkt 6-191
kennzeichnen, Transaktion 2-34
Komponente
- anpassen, Zeitpunkt 4-52
- Jahr 4-48, 4-53, 5-10, 5-16
- Minute 4-48, 4-53, 5-10, 5-16
- Monat 4-48, 4-53, 5-10, 5-16
- Sekunde 4-48, 4-53, 5-10, 5-16
- Sekundenbruchteil 4-48, 4-53, 5-10, 5-16
- Stunde 4-48, 4-53, 5-10, 5-16
- Tag 4-48, 4-53, 5-10, 5-16
- Zeitpunkt 4-48, 5-10, 5-16
- Zeitspanne 4-53

konsistenter
- Datenbankzustand 2-33
- Zustand 2-33

Konstante
- Menge, Elementabfrage 5-53
- alphanumerisch 4-39
- Ausdruck 5-38
- Datum 4-45
- Funktionsargument 5-4
- NULL 4-37
- NULL-Wert 4-37
- numerisch 4-42
- Zeitpunkt 4-48
- Zeitspanne 4-54

korrelierte Unterabfrage 5-36

Länge
- einer Zeichenkette, LENGTH() 5-18
- Name 3-5

LAST 6-112
leeren, Puffer 6-116, 6-162
LENGTH()
- Funktion 4-41
- Länge einer Zeichenkette 5-18
- Zeichenkettenfunktion 5-18

lesen, Sperre 6-182
Lesesperre 6-213
lexikalische Einheiten 3-2
LIKE,Operator 5-55
linearer Outer-Join 5-70
LOAD 6-141
LOCK MODE 6-25, 6-68
LOCK TABLE 6-146
löschen
- Datenbank 2-6
- Audit-Protokoll (DROP AUDIT) 6-96
- Basistabelle 2-10
- BYTE-Spalte 4-34
- Constraint 2-25
- CONSTRAINT 6-25
- Datenbank (DROP DATABASE) 6-97
- Index 2-24
- Index (DROP INDEX) 6-99
- Satz (DELETE) 6-88
- Spalte 6-24
- Synonym 2-19
- Synonym (DROP SYNONYM) 6-101
- Tabelle (DROP TABLE) 6-103
- temporäre Tabelle 2-16
- TEXT-Spalte 4-30
- View 2-13

– View (DROP VIEW) 6-105
logischer Operator 5-61
– AND 5-62
– NOT 5-62
– OR 5-62
MATCHES ,Operator 5-56
MAX()
– ALL-Klausel 5-19
– DISTINCT-Klausel 5-19
– GROUP BY-Klausel 5-19
– Maximum bestimmen 5-19
– Mengenfunktion 5-19
Maximum bestimmen, MAX() 5-19
MDY()
– Datum erzeugen 5-21
– Zeitfunktion 5-21
Menge, Elementabfrage 5-53
Mengenfunktion 5-5, 5-7, 5-8, 5-19, 5-22, 5-27, 6-197
– AVG() 5-5
– COUNT() 5-8
– COUNT(*) 5-7
– MAX() 5-19
– MIN() 5-22
– SUM() 5-27
MIN()
– ALL-Klausel 5-22
– DISTINCT-Klausel 5-22
– GROUP BY-Klausel 5-22
– Mengenfunktion 5-22
– Minimum bestimmen 5-22
– NULL-Wert 5-22
Minimum bestimmen, MIN() 5-22
Minute, Komponente 4-48, 4-53, 5-10, 5-16
MODE ANSI 6-45, 6-49, 6-219
Möglichkeiten, Transaktionssicherung 2-37
Monat
– bestimmen, MONTH() 5-24
– Komponente 4-48, 4-53, 5-10, 5-16
MONEY
– Ausgabeformat 4-16, 4-43
– Datentyp 4-16
– Spalte 4-16
MONTH()
– Monat bestimmen 5-24
– Zeitfunktion 5-24
Multiplikation, Ausdruck 5-40
Muster 5-55
Mustervergleich, Prädikat 5-55
Name 3-4
– ändern 3-7
– Benutzer 3-4
– Constraint 6-26, 6-68
– Datei 3-4
– Datenbankobjekt 3-4
– definieren 3-6
– Eindeutigkeit 3-8
– einfach 3-5
– INFORMIX-ONLINE-Speicherbereich 3-5
– Länge 3-5
– qualifiziert 3-9
– Tabelle 3-7
NEXT 6-111
NEXT SIZE-Klausel 6-68
Nicht-ANSI, voreingestellte Tabellenzugriffsrechte 2-29
Nicht-NULL-Wert 2-21
normaler
– Index 2-23
– Join 5-65, 5-66
NOT, Operator 5-62
NULL
– Ausdruck 4-38
– Bedingung 4-38
– Funktion 4-38
– Funktionsargument 5-4
– Konstante 4-37
– Prädikat 4-38
NULL-Wert 2-21, 4-37
– Ausdruck 5-38
– ausschließen 2-22
– Bedingung 5-61
– Berechnung 4-38
– COUNT() 5-8
– GROUP BY-Klausel 4-38
– MIN() 5-22
– ORDER BY-Klausel 4-38

– Spalte 2-20, 4-37
– SUM() 5-27
– Vergleich 5-46
NULL-Werte verbieten 6-65
NUMERIC-Datentyp 4-14, 4-17
NUMERIC-Spalte 4-14
numerische
– Konstante 4-42
– Werte, Vergleich 5-46
numerischer Datentyp 4-10, 4-11, 4-12, 4-14, 4-16, 4-17, 4-18, 4-19
numerischer Wert 4-42
– Ausdruck 4-43
– Berechnung 4-43
– eintragen 4-42
– Funktion 4-43
– Konvertierung 4-43
– Prädikat 4-43

öffnen
– Datenbank 2-6
– Satzzeiger 2-54, 6-149
OPEN 6-149
Operand, Ausdruck 5-38
Operator
– ALL 5-49
– AND 5-62
– ANY 5-48
– Ausdruck 5-38
– Bedingung 5-61
– einstellig 5-39
– LIKE 5-55
– logisch 5-61
– MATCHES 5-56
– NOT 5-62
– OR 5-62
– Prädikat 5-43
– SOME 5-49
– Vergleich 5-45
– zweistellig 5-39
Operatorzeichen 3-3
OR, Operator 5-62
ORDER BY-Klausel
– (SELECT) 6-201
– NULL-Wert 4-38
OUTER 6-189
Outer-Join 5-65, 5-69
– geschachtelt 5-70
– linear 5-70
– zusammengesetzt 5-70

Page 2-5
Page-Sperre 2-41
Platzhalter 5-55
positionieren, Satzzeiger 6-111
Prädikat 5-43
– alphanumerischer Wert 4-41
– auswerten 5-43
– Bedingung 5-43, 5-61
– berechnen 5-43
– Bereichsabfrage 5-50
– Datum 4-47
– Elementabfrage 5-53
– Existenzabfrage 5-60
– Mustervergleich 5-55
– NULL 4-38
– numerischer Wert 4-43
– Operator 5-43
– Übersicht 5-43
– Vergleich auf NULL 5-59
– Vergleich mit Ergebnisspalte 5-48
– Vergleich von zwei Werten 5-45
– Wahrheitswert 5-43
– Zeitpunkt 4-51
– Zeitspanne 4-57
PREPARE 6-156
PREVIOUS 6-111
PRIOR 6-112
Priorität
– Ausdruck 5-42
– Bedingung 5-63
Programmeinbettung 2-50
– Hostvariable 2-50
– Indikatorvariable 2-51
– Insert-Satzzeiger 2-52
– Satzzeiger 2-52
– Select-Satzzeiger 2-52
– Sprachschnittstellen 2-50
Protokollierung
– gepufferte 6-49
– ungepufferte 6-49
Protokollierungsart siehe Protokollie-

rungsmodus 6-49
Protokollierungsmodus 6-49, 6-217
– Transaktion 2-35
prüfen, Tabelle 6-33
PUBLIC 2-27
Puffer
– leeren 6-162
– leeren (FLUSH) 6-116
PUT 6-161

Qualifikation
– Datenbank 3-12
– Eigentümer 3-10
– Informixsystem 3-13
– Tabelle 3-11
qualifizierter Name 3-9

REAL-Datentyp 4-18
REAL-Spalte 4-18
Realzahl 4-42
– Datentyp 4-18
– Spalte 4-18
RECOVER TABLE 6-165
Referenz, Tabelle 6-181, 6-189
RELATIVE 6-112
RENAME COLUMN 6-167
RENAME TABLE 6-169
REPAIR TABLE 6-171
Repeatable Read
– Isolationsstufe 2-44, 6-183, 6-214
Reservierte Wörter A-35
Resource-Zugriffsrecht 6-125, 6-173
REVOKE 6-172
ROLLBACK WORK 6-176
– Transaktionsklammer 2-34
ROLLFORWARD DATABASE 6-179
ROWID, Satzadresse 6-186

Satz 2-9
– aktueller (FETCH) 6-111
– aus Datei einlesen (LOAD) 6-140
– einfügen (INSERT) 6-135
– in Datei ausgeben (OUTPUT) 6-154
– in Datei ausgeben (UNLOAD) 6-222
– in Tabelle einfügen (LOAD) 6-140
– in Variable, (SELECT (eingebettet)/INTO) 6-208
– löschen (DELETE) 6-88
Satzadresse 6-186
Satzanzahl aktualisieren (UPDATE STATISTICS) 6-234
Satzgröße A-30
Satzsperre 2-41
Satzzeiger 2-52
– definieren 6-80
– freigeben (FREE) 6-119
– öffnen (OPEN) 6-149
– positionieren (FETCH) 6-111
– schließen (CLOSE) 6-33
– schließen (COMMIT WORK) 6-38
– schließen (ROLLBACK WORK) 6-176
– und Transaktion 2-58
– vereinbaren (DECLARE) 6-80
– definieren 2-53
– FOR UPDATE (DECLARE) 6-81
– FOR UPDATE-Klausel 2-53
– INSERT-Anweisung (DECLARE) 6-82
– öffnen 2-54
– Satz in Puffer schreiben (PUT) 6-161
– schließen 2-54
– SCROLL 6-80
– SCROLL-Klausel 2-53
– SELECT-Anweisung (DECLARE) 6-81
– verwenden 2-57
– WITH HOLD 6-81
– WITH HOLD-Klausel 2-53
schätzen, Extent-Größe A-29
schließen
– Datenbank 2-7, 6-36
– Insert-Satzzeiger 6-33
schließen Satzzeiger 2-54, 6-33
– (COMMIT WORK) 6-38
– (ROLLBACK WORK) 6-176
Schreibsperre, SE 2-41
SCROLL-Klausel, Satzzeiger 2-53
Scroll-Satzzeiger 2-53, 6-80

– Arbeitsweise 6-85
SE-Schreibsperre 2-41
– und Transaktion 2-36
Sekunde, Komponente 4-48, 4-53, 5-10, 5-16
Sekundenbruchteil, Komponente 4-48, 4-53, 5-10, 5-16
SELECT 6-181
– eingebettet 6-207
SELECT-Anweisung
– verbinden, (SELECT/UNION) 6-206
– Existenzabfrage 5-60
Select-Satzzeiger 2-52
– zum Ändern 2-56
– zum Lesen 2-55
– verwenden 2-54
Select-Tabellenzugriffsrecht 2-28, 6-124, 6-173
SELECT/FROM 6-188
SELECT/GROUP BY 6-197
SELECT/HAVING 6-199
SELECT/INTO 6-208
SELECT/INTO TEMP 6-204
SELECT/ORDER BY 6-201
SELECT/Spaltenauswahl 6-185
SELECT/UNION 6-206
SELECT/WHERE 6-193
SERIAL-Datentyp 4-12
SERIAL-Spalte 4-12
– einfügen 6-27
SET ISOLATION 6-213
SET LOCK MODE 6-215
SET LOG 6-217
setzen
– Datenbanksperre 2-44
– Einzelsperre 2-45
– Sperre 2-44
– Tabellensperre 2-44
Share-Einzelsperren und Transaktion (INFORMIX-ONLINE) 2-36
SHARE-Sperre 2-42, 6-146, 6-213
Share-Tabellensperre 2-40
SITENAME
– aktuelles Informixsystem 5-26
– Funktion 4-39
– Zeichenkettenfunktion 5-26
SMALLFLOAT
– Datentyp 4-18
– Spalte 4-18
SMALLINT
– Datentyp 4-10
– Spalte 4-10
SOME, Operator 5-49
sortieren
– Ergebnistabelle 6-201
– Index 6-52
– nach Index (ALTER INDEX) 6-18
Sortierreihenfolge 6-53
Spalte 2-9, 2-20
– ändern 6-24
– Ausdruck 5-39
– bearbeiten 2-20
– BYTE 4-32
– CHARACTER 4-6
– DATE 4-20
– DATETIME 4-22
– DECIMAL 4-14
– definieren 2-20
– DOUBLE PRECISION 4-19
– Eindeutigkeitsbedingung 2-20
– einfügen 6-22
– FLOAT 4-19
– Gleitpunkt 4-19
– Inhalt ändern (UPDATE) 6-228
– INTEGER 4-11
– INTERVAL 4-26
– Join 5-65
– Länge ändern 6-24
– löschen 6-24
– MONEY 4-16
– Name ändern (RENAME COLUMN) 6-167
– NULL-Wert 2-20, 4-37
– NUMERIC 4-14
– REAL 4-18
– Realzahl 4-18
– SERIAL 4-12
– SMALLFLOAT 4-18
– SMALLINT 4-10

– TEXT 4-28
– VARCHAR 4-7
– Vergleich auf NULL 5-59
Spaltennummer 6-197, 6-199
Speicherbedarf (Satz) A-30
Speicherplatz freigeben (FREE) 6-119
Sperrbereich 6-25, 6-68
– festlegen 6-25, 6-68
Sperre 6-25, 6-68
– Anzahl 2-45
– aufheben, Tabelle 6-227
– Datenbank 2-40
– Definition 2-40
– Exclusive 2-42
– (FETCH) 6-113
– für Tabelle (LOCK TABLE) 6-146
– halten 2-46
– Page 2-41
– Satz 2-41
– setzen 2-44
– Share 2-42
– und Prozeß 2-46
– Update 2-42
– Tabelle 2-40
– Warte-Modus definieren 6-215
– zurücksetzen 6-147
Sperrebene siehe Sperrbereich 6-25, 6-68
Sperren
– beim Lesen 6-183
– und Transaktion 2-36
– Datenintegrität 2-47
– Datensicherheit 2-47
– Tabelle 6-146
Sperrenhalter 2-40
Sperrgranulat 2-40
– festlegen 6-25, 6-68
Sperrmodus (LOCK TABLE) 6-146
Spezialzeichen 3-3
sqlca-Struktur 2-60
Standard-Isolationsstufe 6-213, 6-214
Standard-Zugriffsrechte 6-71, 6-126
START DATABASE 6-219
starten, Transaktion 6-31
Struktur, *sqlca* 2-60
Stunde, Komponente 4-48, 4-53, 5-10, 5-16
Subtraktion, Ausdruck 5-41
SUM()
– ALL-Klausel 5-27
– DISTINCT-Klausel 5-27
– GROUP BY-Klausel 5-27
– Mengenfunktion 5-27
– NULL-Wert 5-27
– Summe berechnen 5-27
Summe berechnen, SUM() 5-27
Synonym
– definieren 2-18
– definieren (CREATE SYNONYM) 6-59
– Definition 2-18
– erzeugen (CREATE SYNONYM) 6-59
– löschen 2-19
– löschen (DROP SYNONYM) 6-101
– verwenden 2-19
syscolauth (Systemtabelle) A-20
syscolumns (Systemtabelle) A-17
sysconstraints (Systemtabelle) A-21
sysdepend (Systemtabelle) A-20
sysindexes (Systemtabelle) A-18
syssynonyms (Systemtabelle) A-20
syssyntable (Systemtabelle) A-22
systabauth (Systemtabelle) A-19
systables (Systemtabelle) 2-13, A-16
Systemabsturz, Transaktionsprotokoll 2-35
Systemtabelle 2-2, 2-11
– *systables* 2-13
– *sysviews* 2-13
Systemtabellen A-15ff
sysusers (Systemtabelle) A-21
sysviews (Systemtabelle) 2-13, A-21

Tabelle
– aktualisieren (RECOVER TABLE) 6-165
– erzeugen (CREATE TABLE) 6-63
– löschen (DROP TABLE) 6-103
– prüfen (CHECK TABLE) 6-33

– sperren (LOCK TABLE) 6-146
– wiederherstellen (REPAIR TABLE) 6-171
– abhängig 5-69, 6-189
– Arten 2-9
– Datei 2-3
– Definition 2-9
– dominant 5-69, 6-189
– externe 2-6, 6-188
– Qualifikation 3-11
– Referenz 6-181, 6-189
– temporär 2-16, 6-181, 6-203
– Umbenennung 6-181
Tabellen angeben (SELECT/FROM) 6-188
Tabellen- und Datenbankzugriffsrecht 2-30
Tabellen-Dateien 6-69
Tabellen-Information (INFO) 6-131
Tabellenname 3-7
– ändern (RENAME TABLE) 6-169
Tabellenschema 2-9
Tabellensperre 2-40, 6-146
– aufheben (UNLOCK TABLE) 6-227
– Exclusive 2-40
– setzen 2-44
– Share 2-40
– und Transaktion 2-36
Tabellenstruktur ändern (ALTER TABLE) 6-21
Tabellenzugriffsrecht 2-28
– Alter 2-28
– Delete 2-28
– Index 2-28
– Insert 2-28
– Select 2-28
– Update 2-28
– voreingestellt 2-29
Tabellenzugriffsrechte, Eigentümer 2-29
Tag
– bestimmen,DAY() 5-14
– Komponente 4-48, 4-53, 5-10, 5-16
Tblspace 2-5
temporäre Tabelle 2-16, 6-181, 6-202
– erzeugen 6-64
– bearbeiten 2-16
– benannt 6-204
– definieren 2-16
– löschen 2-16
TEXT-Datentyp 4-28
TEXT-Spalte 4-28
– abfragen 4-29
– ändern 4-30
– eintragen 4-29
– löschen 4-30
– verwenden 4-29
TODAY
– aktuelles Datum 5-29
– Funktion 4-46
– Zeitfunktion 5-29
Transaktion 2-33
– beenden (COMMIT WORK) 6-39
– beginnen (BEGIN WORK) 6-31
– starten (BEGIN WORK) 6-31
– und Satzzeiger 2-58
– und Sperren 2-36
– zurücksetzen (ROLLBACK WORK) 6-176
– abschließen 2-34
– ausführen 2-36
– festschreiben 2-34
– kennzeichnen 2-34
– Protokollierungsmodus 2-35
– zurücksetzen 2-34
Transaktionsklammer 2-34
– BEGIN WORK 2-34
– COMMIT WORK 2-34
– ROLLBACK WORK 2-34
Transaktionsprotokoll 2-35
– Systemabsturz 2-35
Transaktionsprotokollierung 6-49
– ändern 6-49, 6-217
– gepuffert 2-35
– gepufferte 6-49, 6-217
– ungepuffert 2-35
– ungepufferte 6-49, 6-217
Transaktionssicherung 2-33, 6-49
– ändern 6-49

– einschalten 6-44, 6-49, 6-219
– ANSI-Datenbank 2-39
– Datenintegrität 2-47
– Datensicherheit 2-47
– einstellen 2-39
– mit 2-38
– Möglichkeiten 2-37
– ohne 2-37
Trenner 3-3
Trennzeichen
– definieren A-26
– Zeitpunkt 4-50
– Zeitspanne 4-56
Typ, Join 5-65

überprüfen
– ANSI-Standard A-24
– Zugriffsrechte 2-30
Umbenennung, Tabelle 6-181
Umgebungsvariable
– DBANSIWARN A-24
– DBDATE 4-46, A-25
– DBDELIMITER A-26
– DBMONEY 4-43, A-26
– DBPATH A-27
– DBTEMP A-27
– INFORMIXDIR A-28
ungepufferte Transaktionsprotokollierung 2-35, 6-217
UNION-Klausel (SELECT) 6-206
UNIQUE-Klausel, Index 2-24
UNITS, Zeitspanne 4-54
UNLOAD 6-222
UNLOCK TABLE 6-227
Unterabfrage 5-35
– Ausdruck 5-39
– Auswertung 5-37
– Elementabfrage 5-53
– korreliert 5-36
UPDATE 6-228
UPDATE STATISTICS 6-234
Update, Tabellenzugriffsrecht 2-28, 6-124, 6-173
Update-Einzelsperre 2-42
USER
– aktueller Benutzer 5-30
– Funktion 4-39
– Zeichenkettenfunktion 5-30
USING DESCRIPTOR
– (EXECUTE) 6-108
– (FETCH) 6-113
– (OPEN) 6-150
– (PUT) 6-162

VARCHAR-Datentyp 4-7
VARCHAR-Spalte 4-7
– ändern 4-8
verbieten, NULL-Werte 6-65
verbinden, SELECT-Anweisung 6-206
vereinbaren, Satzzeiger 6-80
vergeben, Zugriffsrechte 2-29, 6-123
Vergleich
– alphanumerische Werte 5-46
– auf NULL, Prädikat 5-59
– Join 5-65
– mit Ergebnisspalte, Prädikat 5-48
– NULL-Wert 5-46
– numerische Werte 5-46
– von zwei Werten, Prädikat 5-45
– Zeitwerte 5-47
vergleichbare Werte 5-46
vergleichen
– alphanumerischer Wert 4-41
– Datum 4-47
– Zeitpunkt 4-51
– Zeitspanne 4-57
Vergleichsoperator 5-45
versand, Beispieldatenbank A-1
verwenden
– BYTE-Spalte 4-33
– Insert-Satzzeiger 2-57
– Satzzeiger 2-54, 2-57
– Select-Satzzeiger 2-54
– Synonym 2-19
– TEXT-Spalte 4-29
View
– änderbar 2-14
– bearbeiten 2-13
– Datensicherheit 2-47
– definieren 2-12
– Definition 2-12
– erzeugen (CREATE VIEW) 6-72

– Komponenete anpassen 4-52
– Komponente 4-48, 5-10, 5-16
– Konstante 4-48
– Prädikat 4-51
– Trennzeichen 4-50
– vergleichen 4-51
– Zeichenkette 4-48
– Zeitwert 4-44
Zeitspanne 4-53
– Ausdruck 4-57
– Berechnung 4-57
– DATETIME-Schreibweise 4-48
– eintragen 4-56
– Funktion 4-57
– INTERVAL-Schreibweise 4-54
– Komponente 4-53
– Konstante 4-54
– Prädikat 4-57
– Trennzeichen 4-56
– Typ 4-53
– UNITS 4-54
– vergleichen 4-57
– Zeitwert 4-44
Zeitwert 4-44
– Datum 4-44
– Zeitpunkt 4-44
– Zeitspanne 4-44
– Vergleich 5-47
Ziffernfolge, 3-3
Zugriffsrecht
– Alter 2-28
– Arten 2-28
– Audit-Datei 6-41
– Connect 2-26
– Datenbank 2-28
– Datensicherheit 2-47
– DBA 2-26
– Definition 2-28
– Delete 2-28
– entziehen 2-29
– entziehen (REVOKE) 6-173
– Index 2-28
– Insert 2-28
– Select 2-28
– Tabelle 2-28
– überprüfen 2-30
– Update 2-28
– vergeben 2-29
– vergeben (GRANT) 6-123
– voreingestellt 2-28
zurücksetzen
– Sperre 6-147
– Transaktion 2-34, 6-176
zusammengesetzter
– Constraint 2-25
– Join 5-65
– Outer-Join 5-70
Zustand, konsistent 2-33
zuteilen, Datenbankzugriffsrecht 2-27
zweistelliger Operator 5-39

– externer 2-13
– löschen 2-13
– löschen (DROP VIEW) 6-105
– Vorteile 2-15
– WITH CHECK OPTION-Klausel 2-13
vorbereiten
– Anweisung 2-58
– dynamisch formulierte Anweisung 2-58
voreingestellte
– Datenbankzugriffsrechte 2-28
– Tabellenzugriffsrechte 2-29
– Tabellenzugriffsrechte, ANSI 2-29
– Tabellenzugriffsrechte, Nicht-ANSI 2-29
– Zugriffsrechte 2-28

Wahrheitswert, Bedingung 5-61
Wahrheitswert, Prädikat 5-43
Warte-Modus
– definieren (SET LOCK MODE) 6-215
– festlegen 6-215
wechseln
– auf ANSI-Datenbank 2-8
– Datenbank 2-7
WEEKDAY()
– Wochentag bestimmen 5-31
– Zeitfunktion 5-31
Wert 2-21
– alphanumerisch 4-39
– Datum 4-44
– numerisch 4-42
– Zeit 4-44
– Zeitpunkt 4-44
– Zeitspanne 4-44
Werte 4-36
– Einteilung 4-36
– NULL-Wert 4-37
– vergleichbar 5-46
Wertebereich 4-4
WHERE-Bedingung 6-194
WHERE-Klausel (SELECT) 6-193
wiederherstellen, Tabelle 6-171
WITH CHECK OPTION 6-73
– Klausel, View 2-13
WITH GRANT OPTION 6-125
WITH HOLD, Satzzeiger 6-81
– Klausel, Satzzeiger 2-53
WITH NO LOG-Klausel (SELECT) 6-205
Wochentag bestimmen, WEEKDAY() 5-31

YEAR()
– Jahr bestimmen 5-33
– Zeitfunktion 5-33

Zeichenfolge 3-2
– Name 3-2
– Schlüsselwort 3-2
Zeichenkette 4-39
– Datum 5-12, 5-14, 5-24, 5-31, 5-33
– LENGTH() 5-18
– Zeitpunkt 4-48
Zeichenkettenfunktion 5-3, 5-18, 5-26, 5-30
– LENGTH() 5-18
– SITENAME 5-26
– USER 5-30
Zeitdatentyp 4-20, 4-21, 4-24
Zeitdauer 4-53
Zeitfunktion 5-10, 5-12, 5-14, 5-16, 5-21, 5-24, 5-29, 5-31, 5-33
– CURRENT 5-10
– DATE() 5-12
– DAY() 5-14
– EXTEND() 5-16
– MDY() 5-21
– MONTH() 5-24
– TODAY 5-29
– WEEKDAY() 5-31
– YEAR() 5-33
Zeitfunktionen 5-3
Zeitpunkt 4-48
– anpassen,EXTEND() 5-16
– Ausdruck 4-51
– Berechnung 4-51
– Datum 5-12, 5-14, 5-24, 5-31, 5-33
– eintragen 4-50
– Funktion 4-51

Herausgegeben von
Siemens Nixdorf Informationssysteme AG
Postfach 2160, W-4790 Paderborn
Postfach 830951, W-8000 München 83

Bestell-Nr. U 6492-J-Z95-1
Printed in the Federal Republic of Germany
9000 AG 10903. (11250)