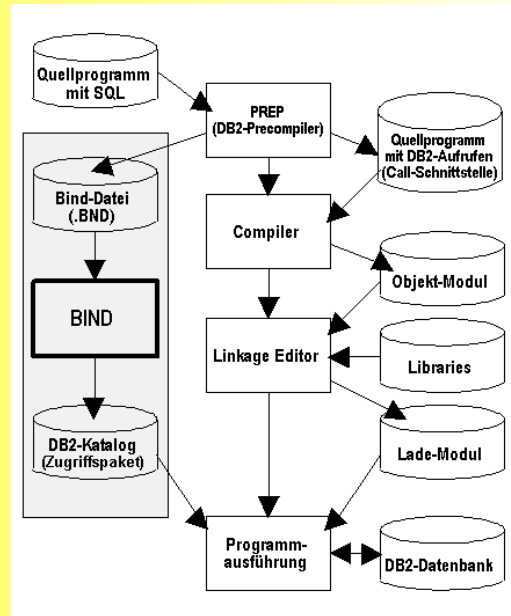




Heinz Axel Pürner

ESQL für Newbies



Eine Einführung in die Anwendungsentwicklung

für DB2 UDB mit Embedded SQL

Leitfaden zur Anwendungsentwicklung

Pürner Publikationen Dortmund

Heinz Axel Pürner

ESQL für Newbies

**Eine Einführung in die Anwendungsentwicklung
für DB2 UDB mit Embedded SQL**

Leitfaden zur Anwendungsentwicklung

Mit 4 Abbildungen

Pürner Publikationen, Dortmund

Pürner, Heinz Axel

ESQL für Newbies

Eine Einführung in die Anwendungsentwicklung für DB2 UDB mit Embedded SQL

Leitfaden zur Anwendungsentwicklung

Pürner Publikationen, Dortmund, 2003

Autoren und Verlag übernehmen für die Fehlerfreiheit der Programme keine Gewährleistung oder Haftung.

Der Verlag übernimmt keine Gewähr dafür, daß die beschriebenen Verfahren, Programme usw. frei von Schutzrechten Dritter sind.

Das Werk einschließlich aller seiner Teile ist urheberrechtlich geschützt. Jede Verwertung außerhalb der engen Grenzen des Urheberrechtsgesetzes ist ohne Zustimmung des Verlages unzulässig und strafbar. Das gilt insbesondere für Vervielfältigungen, Übersetzungen, Mikroverfilmungen und die Speicherung und Verarbeitung in elektronischen Systemen.

© Pürner Publikationen, Von-der-Tann-Str. 14, 44143 Dortmund, 2003

Layout und Herstellung: Beate Pürner, Pürner Technische Dokumentation, Dortmund

Inhalt

1. Überblick Anwendungsprogrammierung	1-1
2. Grundsätzliches zur Anwendungsprogrammierung	2-1
3. Anwendungsprogrammierung in C.....	3-1
4. Anwendungsprogrammierung in COBOL	4-1
5. ESQL-Befehle	5-1
5.1 BEGIN DECLARE SECTION.....	5-2
5.2 END DECLARE SECTION	5-2
5.3 INCLUDE	5-3
5.4 DECLARE CURSOR	5-4
5.5 OPEN.....	5-7
5.6 FETCH.....	5-9
5.7 CLOSE.....	5-10
5.8 DELETE.....	5-10
5.9 INSERT.....	5-11
5.10 UPDATE	5-13
5.11 SELECT INTO	5-15
5.12 CALL.....	5-17
5.13 FREE LOCATOR.....	5-19
5.14 VALUES INTO.....	5-19
5.15 Compound SQL.....	5-20
5.16 WHENEVER	5-22

6. Dynamic SQL-Befehle	6-1
6.1 DESCRIBE	6-1
6.2 PREPARE.....	6-3
6.3 EXECUTE.....	6-5
6.4 EXECUTE IMMEDIATE.....	6-6
7. Kommandos zum Übersetzen und Binden.....	7-1
7.1 PREP (PRECOMPILE PROGRAM)	7-1
7.2 BIND	7-11
8. Anhang.....	8-1
8.1 SQLDA als COBOL-COPY	8-1
8.2 SQLDA für C.....	8-3
9. Stichwortverzeichnis.....	9-1

Vorwort

In diesem Leitfaden wende ich mich an Anfänger, die in einer klassischen Programmiersprache wie COBOL oder C Anwendungsprogramme für DB2-Datenbanken schreiben wollen.

- Ich gebe Ihnen zunächst in Kapitel 1, *Überblick Anwendungsprogrammierung*, einen kurzen Überblick, in welcher vielfältiger Form DB2 die Anwendungsprogrammierung unterstützt.

In den folgenden Kapiteln lege ich den Schwerpunkt auf die Anwendungsentwicklung mit Programmiersprachen der 3. Generation.

- In Kapitel 2, *Grundsätzliches zur Anwendungsprogrammierung*, erläutere ich einige grundsätzliche Aspekte zur Anwendungsentwicklung mit ESQL.
- In Kapitel 3, *Anwendungsprogrammierung in C*, zeige ich Ihnen zwei Beispiele in der Programmiersprache C.
- Kapitel 4, *Anwendungsprogrammierung in COBOL*, enthält ein Beispiel zu Embedded SQL (ESQL) in COBOL, weil ich Ihnen zeigen will, daß eine Migration bestehender DB2-Anwendungen vom Mainframe in eine Client-Server-Umgebung ohne weitgehende Neuprogrammierung nicht so schwierig ist. Auf dem Mainframe ist COBOL immer noch die meist benutzte Programmiersprache.

In den drei folgenden Kapiteln biete ich Ihnen eine Zusammenstellung der ESQL- und DSQL¹-Befehle, der Dienstprogramme und Kommandos, die die Anwendungsprogrammierung unterstützen. Dabei verzichte ich bewußt auf die das vollständigen Aufzählen

¹ DSQL = Dynamisches SQL

aller Befehle und aller Parameter zu den erläuterten Befehlen, weil ich dem Anfänger für den Einstieg in die Anwendungsentwicklung nur die wichtigsten erläutern will, statt ihn mit der Fülle der Möglichkeiten zu verwirren.

- Kapitel 5, *ESQL-Befehle*, enthält die wichtigsten Befehle zu Embedded SQL.
- Kapitel 6, *Dynamic SQL-Befehle*, erläutert die speziellen Befehle für dynamisches ESQL.
- In Kapitel 7, *Kommandos zum Übersetzen und Binden*, finden Sie die Kommandos für den Precompiler und den BIND.
- Im Kapitel 8, *Anhang* finden Sie die SQLDA als Copystrecken für COBOL und für C.

JAVA behandle ich in diesem Leitfaden nicht, weil die JAVA-Datenbankschnittstellen kein Embedded SQL sind: JDBC zählt zu den CALL-Schnittstellen wie auch ODBC und SQLJ ist eine sehr spezielle Variante von ESQL, die ich hier nicht behandeln möchte. JAVA-Anwender finden Beispiele ebenso wie COBOL- oder C-Anwender im Lieferumfang von DB2 UDB im Unterverzeichnis SAMPLES.

1. Überblick Anwendungsprogrammierung

DB2 unterstützt den Zugang zu seinen Datenbanken aus Programmen heraus in vielfältiger Weise. Für die zu kompilierenden Programmiersprachen C, COBOL und FORTRAN stellt IBM SQL-Schnittstellen und einen Precompiler (PREP – Precompile Program) mit der DB2-Software zur Verfügung. Für PL/I ist der Precompiler in dem PL/I-Entwicklungssystem enthalten, für JAVA kann das Dienstprogramm SQLJ benutzt werden. Da die SQL-Befehle quasi in die Befehle der Programmiersprache eingebettet werden, spricht man von ESQL (embedded SQL). Im Normalfall sind die ESQL-Befehle statisch, das heißt sie sind fest codiert und können vor ihrer Ausführung genauso übersetzt werden wie die Befehle der Wirtssprache. Alternativ können Sie die SQL-Befehle aber auch erst während der Ausführung Ihres Programms zusammensetzen und an DB2 übergeben. Dann werden diese Befehle erst während der Programmausführung übersetzt. Man spricht in diesem Fall von dynamischem SQL.

Statisches und
dynamisches
SQL

Der Vorteil der statischen ESQL-Befehle liegt darin, daß die Umsetzung der Befehle in eine ausführbare Form vor der Programmausführung liegt und somit die Ausführungszeit nicht belastet. Außerdem ist die Programmierung wohl etwas einfacher als mit dynamischem SQL. Dynamisches SQL hat dagegen den Vorteil, daß der Optimizer immer den aktuell besten Zugriffspfad sucht, während die einmal vor längerer Zeit übersetzten statischen ESQL-Befehle nach vielleicht schon überholten Entscheidungen des Optimizers ausgeführt werden.

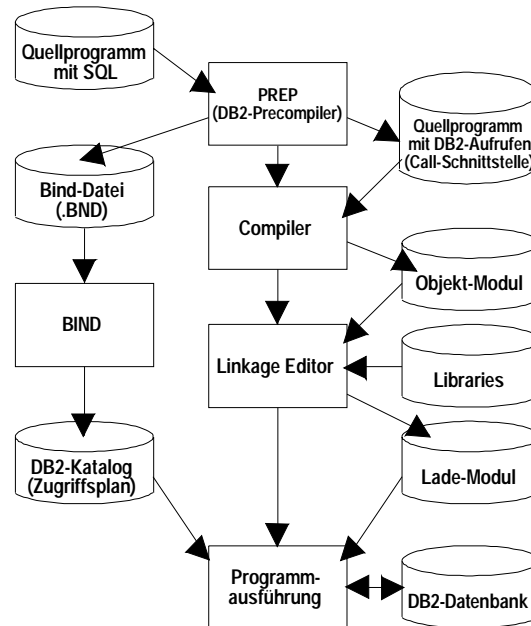
Ein weiterer Vorteil von Programmen mit statischem ESQL liegt in der Sicherheit. Der Benutzer erhält nur die Berechtigung, das Zugriffspaket (package) ausführen zu dürfen, aber keine Berechtigungen für den Zugriff auf die darin angesprochenen Tabellen. Mit DSQL muß der Benutzer auch die Berechtigung zur entsprechenden Manipulation der Tabellen besitzen.

API Für die interpretierte Sprache REXX stellt IBM eine Call-Schnittstelle oder API (application programming interface) zur Verfügung. Damit wird nur dynamisches SQL unterstützt.

ODBC-Unterstützung Zusätzlich zur SQL-Schnittstelle verfügt DB2 auch über ein Call-Level-Interface (CLI) zur Unterstützung von Microsofts ODBC und X/Open CLI. Über CLI können alle SQL-Befehle ausgeführt werden, die unter Dynamic SQL erlaubt sind, sowie Compound SQL. CLI-Programme kennen keine Vorübersetzung (PREP) und keinen Bind. Da sie also auch unabhängig von einem Precompiler sind, können sie ohne Kenntnis des späteren DBMS-Produktes entwickelt werden, wenn jenes über ein kompatibles ODBC- oder X/Open-Laufzeitsystem verfügt. Sie erreichen somit ein sehr hohes Maß an Portabilität. Sie besitzen aber auch die Nachteile von DSQL-Programmen.

Gleiches gilt für JAVA-Programme, die mit der JDBC-Schnittstelle arbeiten.

Bild 1-1:
Quellprogramm
bearbeiten



Für die wichtigsten Programmiersprachen besitzt DB2 einen Precompiler PREP (Pre-compile Program), der die SQL-Befehle im Quellprogramm in DB2-Aufrufe umsetzt und zugleich in eine Binde-Datei (.BND)¹ übernimmt.

PREP erzeugt als Ausgabedateien:

- das vorübersetzte Quellprogramm mit DB2-Aufrufen und
- die Binde-Datei.

Das vorübersetzte Quellprogramm wird anschließend wie gewohnt mit dem Compiler und dem Linker (linkage editor) in ein ausführbares Lade-Modul umgewandelt.

Die Binde-Datei wird standardmäßig bereits vom Precompiler mit BIND an die Datenbank gebunden, das heißt der DB2-Optimizer ermittelt zu den SQL-Befehlen die optimalen Zugriffspfade und speichert diese als Zugriffspaket (package) im Datenbank-Katalog ab (Katalog-Tabellen SYSIBM.SYSPLAN und SYSIBM.SYSSECTION). Sie können PREP durch Aufruf-Parameter das Binden untersagen und das Zugriffspaket mit Hilfe von BIND getrennt erstellen.

Precompiler und BIND werden über DB2-Kommandos aufgerufen.

Zusätzlich zu der SQL-Unterstützung bietet Ihnen DB2 die Möglichkeit, DB2-Kommandos über Call-Schnittstellen abzusetzen. Für C, COBOL und FORTRAN müssen Sie dazu jeweils dedizierte Schnittstellen-Programme aufrufen. In REXX übergeben Sie das Kommando als Zeichenkette über eine einheitliche Schnittstelle.

¹ Unter DB2 für OS/390 wird diese Datei DBRM genannt.

2. Grundsätzliches zur Anwendungsprogrammierung

Programmiersprachen der sogenannten 3. Generation sind in der Dateiverarbeitung grundsätzlich einzelsatzorientiert. Das bedeutet, sie können jeden Satz einer Datei nur einzeln lesen, schreiben, verändern oder löschen. SQL dagegen ist mengenorientiert. Die Selektion, Änderung oder Löschung *einer* Zeile ist als Sonderfall einer Operation mit einer Menge anzusehen, die aus genau *einem* Tupel besteht, und nur über die WHERE-Bedingung zu erzwingen.

Wegen der Beschränkungen der klassischen Programmiersprachen können in Programmen mit Embedded SQL nur einzelne Zeilen bearbeitet werden¹. So beschränkt sich ein SELECT-Befehl, den Sie direkt in einem Programm codieren, auf das Lesen genau einer Zeile. Ist die Ergebnis-Tabelle größer, erhalten Sie einen Fehlercode.

Cursor

Für die satzweise Bearbeitung von relationalen Mengen in einem Programm wurde das Konstrukt des Cursor eingeführt. Mit seiner Hilfe können Sie die Ergebnis-Tabelle eines SELECT zeilenweise lesen, die gelesene Zeile ändern oder löschen. Dabei wird der SELECT in die Vereinbarung *DECLARE cr-name CURSOR FOR* übernommen und mit dem Befehl *OPEN cr-name* ausgeführt. Mit FETCH werden die Zeilen der Ergebnis-Tabelle gelesen, mit *CLOSE cr-name* wird die Ergebnis-Tabelle freigegeben. Die SQL-Befehle UPDATE und DELETE nehmen mit *WHERE CURRENT OF cr-name* Bezug auf die aktuelle Zeile. Weitere Erläuterungen zu den Befehlen finden Sie in Kapitel 5, *ESQL-Befehle*.

¹ Sie können natürlich aus Programmen heraus mengenorientierte SQL-Befehle absetzen, wenn die Zeilen der angesprochenen Tabellen dabei nicht im Programm bearbeitet werden.

NULL-Wert

Ein Problem im Zusammenspiel von relationalen Datenbanken und klassischen Programmiersprachen ist die Behandlung des Wertes NULL: Die Programmiersprachen besitzen keine Möglichkeiten, einen unbestimmten Wert zu bearbeiten. Daher gibt es für die Übergabe von NULL zwischen Datenbank und Programm NULL-Indikatoren. Wird eine solche Indikator-Variable von der Datenbank-Schnittstelle auf -1 gesetzt, so enthält die zugehörige Programm-Variable *keinen* sinnvollen Wert, da der entsprechende Wert in der Tabelle NULL ist. Sonst nimmt die Indikator-Variable den Wert 0 an. Setzen Sie im Programm bei einem Befehl eine Indikator-Variable auf -1, wird in der Datenbank mit NULL statt mit dem Wert der zugehörigen Programm-Variablen gearbeitet.



IBM nutzt die Indikator-Variable nicht nur für den NULL-Wert, sondern auch für andere Angaben, die über den SQL-Standard hinaus gehen! In DB2 enthält sie auch die ursprüngliche Länge abgeschnittener Zeichenketten oder die Sekunden von Zeitangaben, die bei der Variablenzuweisung abgeschnitten wurden. Außerdem zeigt sie Fehler bei der Datenkonvertierung an.

Fehler-
behandlung

Fehler und Warnungen werden in zwei Variablen vom DBMS zurückgegeben, in SQLCODE und SQLSTATE. SQLCODE ist die ältere und geläufigere Variable. Fehler werden hierin als negative Zahlen, Warnungen als positive Zahlen angezeigt. 0 bedeutet, daß der Befehl erfolgreich war. Nach dem SQL-Standard von 1992 ist die Variable SQLSTATE als Status-Variable der Variablen SQLCODE in Zukunft vorzuziehen. Für SQLSTATE gibt es genormte Status-Werte und individuelle Bereiche für den Benutzer - im Gegensatz zu SQLCODE. Letztere wird wohl nur aus Gründen der Kompatibilität die nächsten Jahre überleben.

Üblicherweise wird im DB2-Umfeld ein SQL-Kommunikationsbereich (SQLCA) benutzt. Dieser Bereich enthält beide Variablen und noch weitere Felder, ist aber nicht im SQL-Standard definiert. Wer auf Portabilität achten muß, sollte die SQLCA nicht nutzen, sondern nur die Variablen SQLCODE und/oder SQLSTATE. Wer seine Programme nur im Gültigkeitsbereich von IBMs SAA einsetzt, sollte dagegen die SQLCA für eine ausführliche Fehlerbehandlung nutzen. Auch die Hersteller anderer relationaler Systeme, z.B. ORACLE, unterstützen einen solchen Bereich mit sehr ähnlichem Aufbau

Berechtigungen Zur Ausführung der Programme mit statischen ESQL-Befehlen benötigen die Anwender nur eine EXECUTE-Berechtigung, die für das Zugriffspaket (package) vergeben wird. Damit erhalten die Benutzer des Programms keinerlei Rechte an den DB2-Objekten (Tabellen, Indizes etc.), auf die im Programm zugegriffen werden. D.h. die Benutzer können ohne das Programm nicht auf die Datenbank zugreifen, auch nicht über eine interaktive Schnittstelle wie die Befehlszentrale.

Die Programmierer dagegen benötigen die direkten Berechtigungen für die benutzten DB2-Objekte, um den BIND explizit oder als Teil der Vorübersetzung (PREP) ausführen zu können.

Bei der Nutzung von dynamischem SQL müssen zunächst auch die Anwender des Programms die direkten Berechtigungen an den benutzten DB2-Objekten besitzen. Allerdings läßt sich diese Regel durch einen BIND-Parameter (DYNAMICRULES) ändern. Weitere Informationen zu DYNAMICRULES findet der fortgeschrittene Entwickler im DB2 Command Reference Handbuch.

3. Anwendungsprogrammierung in C

Zwei Beispiel-Programme sollen den Umgang mit Embedded SQL allgemein und speziell in C verdeutlichen. Das erste Beispiel lehnt sich an die Beispiel-Tabellen der Datenbank SAMPLE von IBM an und liest Zeilen mit Hilfe eines Cursor und ändert diese. Das zweite Beispiel knüpft an unsere alte Fallstudie „Yachthafen“¹ an und arbeitet mit einem Cursor, einem SELECT INTO und einem INSERT.

Beispiel-
Programm
Manager-Boni

In diesem Programm werden alle Manager der Tabelle EMPLOYEE gelesen, die Anzahl der ihnen unterstellten Mitarbeiter ermittelt und entsprechend der Anzahl einen bescheidenen Bonus festsetzen. Als Grund-Bonus sind 500 fest vorgegeben.

Es werden mit Hilfe des Cursor C1 alle Manager mit Personalnummer (EMPNO) und Nachnamen (LASTNAME) gelesen. In einem SELECT-Befehl, der in dieser Form nur jeweils eine Zeile als Ergebnis liefern darf, wird die Anzahl der direkt unterstellten Mitarbeiter gelesen. In Abhängigkeit dieser Anzahl (NUM_EMPLOYEES) wird der zusätzliche Bonus bemessen. Mit einem UPDATE-Befehl mit WHERE CURRENT OF wird der neue Bonus in der Tabellenzeile geändert.

Beachten Sie bitte, daß die Spalte LASTNAME als VARCHAR(15) definiert ist. Sie besitzt also ein Längengebiet von 2 Bytes, das auch an das Programm übertragen wird. Daher ist im Programm eine Struktur zu definieren, die aus dem Längengebiet und der eigentlichen Zeichenkette besteht. So ist es auch in meinem Beispiel-Programm gemacht worden. Allerdings kann in C auch ohne Struktur nur mit einer Zeichenkette

¹ Siehe unser Buch DB2 Common Server, Vieweg Verlag Wiesbaden 1997

gearbeitet werden, weil dann die binäre Null (\0) als Terminator das Ende markiert. Hier nun der Quellcode als Eingabe für den DB2-Precompiler:

```

Quellcode für      /*
Beispiel-          +-----+
Programm          | Name       : CursorUpd.SQC
Manager-Boni     | Purpose    : Demo mit Cursor lesen und ändern
                 | Platform   : DB2 UDB
                 | Author     : H.A. Pürner
                 |              Pürner Unternehmensberatung, Dortmund
                 | Disclaimer : This "sample" code is for demonstrations only, no
                 |              warrenties are made or implied as to correct
                 |              function. You should carefully test this code in
                 |              your own environment before using it.
                 +-----+
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sqlenv.h>
#include <sqlca.h>
#include "util.h"

#define CHECKERR(CE_STR)  if (check_error (CE_STR, &sqlca) != 0) return 1;

int main(int argc, char *argv[]) {

    EXEC SQL BEGIN DECLARE SECTION;
        char  MANAGER_ID[7];
        struct {
            short  nl;
            char   la_na[18];
        } lname;
        double base_bonus, BONUSPRM;
        long NUM_EMPLOYEES;
    EXEC SQL END DECLARE SECTION;

    double tot_bonus = 0;
    char LName[24];
    int END_TABLE = 0;

```

```

/* Declare Variables for CALL USING */
struct sqlca    sqlca;

base_bonus = 500.0;
tot_bonus  = 0;

/* Datenbank-Connect */
printf("Connect to Database SAMPLE.\n");
EXEC SQL CONNECT TO SAMPLE;
CHECKERR ("CONNECT TO Sample");

/* Declare Cursor */
EXEC SQL DECLARE C1 CURSOR FOR
        SELECT EMPNO, LASTNAME
        FROM EMPLOYEE
        WHERE JOB = 'MANAGER'
        FOR UPDATE OF BONUS;

EXEC SQL OPEN C1;
CHECKERR ("OPEN C1");

while( END_TABLE == 0) {
EXEC SQL FETCH C1
        INTO :MANAGER_ID, :lname;

if (sqlca.sqlcode == 100)
    { END_TABLE = 7; break; }
CHECKERR ("FETCH C1");

/* Single Select */
EXEC SQL SELECT COUNT(DISTINCT EMP.EMPNO)
        INTO :NUM_EMPLOYEES
        FROM DEPARTMENT DEPT,
        EMPLOYEE EMP
        WHERE EMP.WORKDEPT = DEPT.DEPTNO
        AND DEPT.MGRNO = :MANAGER_ID;
CHECKERR ("SELECT INTO");

```

```

if( NUM_EMPLOYEES > 10 ) BONUSPRM = 1000.00;
else {
    if( NUM_EMPLOYEES > 5 ) BONUSPRM = 500.00;
    else {
        if( NUM_EMPLOYEES > 0 ) BONUSPRM = 100.00;
        else BONUSPRM = 0.00;
    }
}

/* UPDATE mit WHERE CURRENT OF.. */
EXEC SQL UPDATE EMPLOYEE
        SET BONUS = :base_bonus + :BONUSPRM
        WHERE CURRENT OF C1;
CHECKERR ("UPDATE C1");

lname.la_na[lname.nl] = '\0';
printf("Manager %s %15s Bonus: %9.2f \n", MANAGER_ID, lname.la_na,
base_bonus+BONUSPRM);

tot_bonus = tot_bonus + base_bonus + BONUSPRM;
}

EXEC SQL CLOSE C1;
CHECKERR ("CLOSE C1");

printf(" Grund-Bonus %9.2f      Summe Boni: %9.2f \n", base_bonus, tot_bonus);
printf("\n");

/* Disconnect from Database */
EXEC SQL CONNECT RESET;
CHECKERR ("CONNECT RESET");
return 0;
}
/* end of program */

```

Da die Zeilen des Cursor C1 geändert werden, habe ich den Cursor mit der Klausel FOR UPDATE OF deklariert. Diese Klausel ist optional, wenn Sie nicht entsprechend den SAA-Definitionen von IBM (SAA - *systems application architecture*) entwickeln wollen.

Zur Fehlerbehandlung nach SQL-Befehlen benutze ich ein Unterprogramm *check_error()*, das aus den älteren DB2-Beispielprogrammen stammt und die Informationen des SQL-Kommunikationsbereichs SQLCA aufbereitet und ausgibt. Die SQLCA hat folgende Struktur:

SQL-Kommuni-
kationsbereich
SQLCA

```

*****
*
* Source File Name = SQLCA.H
*
* (C) COPYRIGHT International Business Machines Corp. 1987, 1997
* All Rights Reserved
* Licensed Materials - Property of IBM
*
* US Government Users Restricted Rights - Use, duplication or
* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
*
* Function = Include File defining:
*           SQL Communications Area
*
* Operating System = Common C Include File
*
*****/

#if !(defined(SQLCODE) || defined(SQLCADE)) /* Permit Duplicate Includes */

#include "sqlsystem.h" /* System dependent defines */

#if (defined(DB2OS2) || defined(DB2NT))
#pragma pack(4)
#elif (defined(DB2WIN))
#pragma pack(2)
#elif (defined(DB2MAC))
#if (defined(DB2PPC))
#pragma align 4
#elif (defined(DB268K))
#pragma align 2
#endif
#elif (defined(DB2HP) || defined(DB2SNI))

#elif (defined(DB2SUN) && defined(__xlc__))
#pragma options align=natural
#elif (defined(DB2AIX) && defined(__64BIT__))
#pragma options align=natural
#elif (defined(DB2AIX))
#pragma options align=power
#endif

```

```

/* SQL Communication Area - SQLCA */
/* _SQLLOLDCHAR defaults to 'char'. See sqlsystem.h for details. */

SQL_STRUCTURE sqlca
{
    _SQLLOLDCHAR    sqlcaid[8];           /* Eyecatcher = 'SQLCA  ' */
    sqlint32        sqlcabc;             /* SQLCA size in bytes = 136 */
#ifdef DB2_SQL92E
    sqlint32        sqlcade;           /* SQL return code */
#else
    sqlint32        sqlcode;           /* SQL return code */
#endif
#ifdef
    short          sqlerrml;           /* Length for SQLERRMC */
    _SQLLOLDCHAR    sqlerrmc[70];     /* Error message tokens */

    _SQLLOLDCHAR    sqlerrp[8];       /* Diagnostic information */

    sqlint32        sqlerrd[6];       /* Diagnostic information */
    _SQLLOLDCHAR    sqlwarn[11];     /* Warning flags */

#ifdef DB2_SQL92E
    _SQLLOLDCHAR    sqlstat[5];       /* State corresponding to SQLCODE */
#else
    _SQLLOLDCHAR    sqlstate[5];     /* State corresponding to SQLCODE */
#endif
#endif
};

#ifdef DB2_SQL92E
#define SQLCADE    sqlca.sqlcade
#else
#define SQLCODE    sqlca.sqlcode
#endif
#define SQLWARN0    sqlca.sqlwarn[0]
#define SQLWARN1    sqlca.sqlwarn[1]
#define SQLWARN2    sqlca.sqlwarn[2]
#define SQLWARN3    sqlca.sqlwarn[3]
#define SQLWARN4    sqlca.sqlwarn[4]
#define SQLWARN5    sqlca.sqlwarn[5]
#define SQLWARN6    sqlca.sqlwarn[6]
#define SQLWARN7    sqlca.sqlwarn[7]
#define SQLWARN8    sqlca.sqlwarn[8]
#define SQLWARN9    sqlca.sqlwarn[9]

```

```

#define    SQLWARNA            sqlca.sqlwarn[10]

/* sqlerrd tokens updated when compound SQL statements processed */

#define    SQL_CMP_NA_ERRORS        1
#define    SQL_CMP_ROWS_AFFECTED    2
#define    SQL_CMP_STMTS_COMPLETED  3
#define    SQL_CMP_REF_INT_ROWS     4

/* sqlerrd tokens updated when CONNECT statements processed */

#define    SQL_CONNECT_DB_APP2DB_CONVFACTOR    0
#define    SQL_CONNECT_DB_DB2APP_CONVFACTOR    1
#define    SQL_CONNECT_DB_UPDATEABILITY_IN_UOW  2
#define    SQL_CONNECT_DB_COMMIT_TYPE          3

/* Values returned for sqlerrd[SQL_CONNECT_DB_UPDATEABILITY_IN_UOW] */

#define    SQL_DB_UPDATEABLE                1
#define    SQL_DB_READ_ONLY                 2

/* Values returned for sqlerrd[SQL_CONNECT_DB_COMMIT_TYPE] */

#define    SQL_DB_ONE_PHASE_COMMIT          1
#define    SQL_DB_ONE_PHASE_READ_ONLY      2
#define    SQL_DB_TWO_PHASE_COMMIT          3

#if (defined(DB2OS2) || defined(DB2NT) || defined(DB2WIN))
#pragma pack()
#elif (defined(DB2MAC))
#pragma align
#elif (defined(DB2HP) || defined(DB2SNI))

#elif (defined(DB2AIX) || (defined(DB2SUN) && defined(__xlc__)))
#pragma options align=reset
#endif

#endif /* SQLCODE */

```

Precompiler
PREP



Der Precompiler PREP erwartet, daß die angesprochenen Datenbank-Objekte zur Übersetzungszeit angelegt sind und prüft Programm-Angaben gegen den Katalog. So entdeckt er Fehler, die sonst erst bei den Testläufen offenkundig würden. Denken Sie daran: je früher Fehler entdeckt werden, desto kostengünstiger ist ihre Beseitigung.

Im Gegensatz zu DB2 für OS/390, dessen Precompiler üblicherweise nicht auf den System-Katalog zugreift, kennt DB2 UDB keinen DECLARE TABLE-Befehl, aus dem die Tabellenstruktur ersichtlich ist. Dies erhöht die Datenunabhängigkeit der Programme gegenüber DB2 unter OS/390.

PREP erzeugt neben einigen Meldungen eine Datei mit den vorübersetzten SQL-Befehlen für den C- oder C++-Compiler. Es werden die üblichen C- und C++-Compiler der jeweiligen Betriebssystem-Umgebung unterstützt. Mit dem Parameter TARGET können Sie den gewünschten Compiler explizit vorgeben.

PREP erzeugt standardmäßig in einem integrierten BIND-Aufruf ein Zugriffspaket (package) in der Datenbank. Mit dem Parameter BINDFILE erhalten Sie bindefähiges Modul (progame.BND), mit SYNTAX können Sie eine reine Syntax-Prüfung durchführen.

Zugriffspaket
(package)

Ein Zugriffspaket (package) enthält die Zugriffe, die der Optimizer aufgrund der physischen Gegebenheiten in der Datenbank für die SQL-Befehle eines Programms ausgewählt hat. Ändern sich die physischen Gegebenheiten der Datenbank, zum Beispiel durch einen neuen Index oder durch andere Statistikwerte im DB2-Katalog, kann ein erneutes Binden der Zugriffspakete notwendig oder sinnvoll sein.

Ein Zugriffspaket unterteilt sich in Abschnitte (section), die die Zugriffe für einen SQL-Befehl aufnehmen. Je Zugriffspaket wird eine Zeile mit allgemeinen Informationen in der Katalog-Tabelle `SYSIBM.SYSPLAN` und je Abschnitt mindestens eine Zeile mit den verschlüsselten Zugriffsdaten in der Katalog-Tabelle `SYSIBM.SYSSECTION` gespeichert. Die Abhängigkeiten des Zugriffspakets von Datenbank-Objekten wie Tabellen oder Indizes werden in `SYSIBM.SYSPLANDEP` festgehalten, Berechtigungen in `SYSIBM.SYPLANAUTH`. Die zugehörigen SQL-Befehle werden in der Tabelle `SYSIBM.SYSSTMT` gespeichert.

Die Aufrufe von C- oder C++-Compiler und Linker (linkage editor) sind abhängig von den Gegebenheiten Ihrer Installation.

Das fertige Programm CursorUpd ist, wie Sie sicher schon erkannt haben, kein Programm mit Schnittstellen zu einer grafischen Benutzerschnittstelle. Es kann im Befehlszeilen-Umfeld aufgerufen werden.

Beispiel-
Programm
*Liegeplatz-
Buchung*

In diesem Programm werden für einen vorgegebenen Zeitraum alle freien Liegeplätze mit Hilfe des Cursor C0 gelesen und angezeigt. Es wird vorausgesetzt, daß die Yacht, für die reserviert werden soll, bereits in der Tabelle YACHT registriert ist. Daher reichen zur Buchung des Liegeplatzes nur die Eingabe der Yacht und des gewünschten Liegeplatzes aus. Es wird mit Einzel-SELECT-Befehlen überprüft, daß die Yacht registriert ist und daß der gewählte Liegeplatz tatsächlich zu den freien gehört. Die Buchung wird durch einen INSERT in die Tabelle BELEGUNG durchgeführt. Dabei prüft im Hintergrund ein Trigger, daß die Yacht mit Größe auch in den Liegeplatz hineinpaßt.

Hier nun der Quellcode als Eingabe für den DB2-Precompiler:

Quellcode für
Beispiel-
Programm
*Liegeplatz-
Buchung*

```

/*
+-----+
| Name      : BUCHMARI.SQC
| Purpose   : Buchung freier Liegeplatz
| Platform  : DB2 UDB
| Author    : H. A. Pürner
|            Pürner Unternehmensberatung, Dortmund
| Disclaimer: This "sample" code is for demonstrations only, no
|            warranties are made or implied as to correct
|            function. You should carefully test this code in
|            your own environment before using it.
|
+-----+
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sqlenv.h>
#include "util.h"

EXEC SQL INCLUDE SQLCA;

#define CHECKERR(CE_STR)  if (check_error (CE_STR, &sqlca) != 0) return 1;

int main(int argc, char *argv[]) {

    EXEC SQL BEGIN DECLARE SECTION;

```

```

char   von_datum [11];
char   bis_datum [11];
char   nix[2];
short  LPNR;
double LAENGE;
double BREITE;
double WTIEFE;
short  WTind = 0;
char   STROM[5];
char   WAN[5];
short  YNR;
char   yname[25];
char   reg_ort[25];
EXEC SQL END DECLARE SECTION;

/* aufruf-prüfung */
if (argc != 3) {
    printf ("\nAufruf: buchmari <von-datum> <bis-datum> \n\n");
    return 1;
}
printf( "\nBeispiel-Programm \n Buchung Liegeplatz für Yacht\n");
strcpy (von_datum, argv[1]);
strcpy (bis_datum, argv[2]);

EXEC SQL CONNECT TO marina;
CHECKERR ("CONNECT TO MARINA");

/* Anzeige der freien Liegeplätze in einer Schleife */
printf("Freie Liegeplätze von %s bis %s \n", von_datum, bis_datum);

EXEC SQL DECLARE c0 CURSOR FOR
    SELECT LPNR, LAENGE, BREITE, WASSERTIEFE,
           STROM, WASSERANSCHLUSS
    FROM LIEGEPLATZ L
    WHERE NOT EXISTS (
        SELECT * FROM BELEGUNG
        WHERE LPNR = L.LPNR AND
              (VON BETWEEN :von_datum AND :bis_datum OR
               BIS BETWEEN :von_datum AND :bis_datum OR
               :von_datum BETWEEN VON AND BIS OR
               :bis_datum BETWEEN VON AND BIS )
    )
    ORDER BY L.LPNR ASC;

```

```

EXEC SQL OPEN c0;
CHECKERR ("OPEN CURSOR 0");
printf(" Platz-Nr Länge Breite Wassertiefe Strom Wasser \n");

do {
    EXEC SQL FETCH c0 INTO :LPNR, :LAENGE, :BREITE, :WTIEFE, :STROM, :WAN;
    if (SQLCODE != 0) break;

    printf( "   %3i   %5.2f %5.2f   %5.2f   %s   %s\n",
           LPNR, LAENGE, BREITE, WTIEFE, STROM, WAN );

} while ( 1 );

EXEC SQL CLOSE c0;
CHECKERR ("CLOSE CURSOR 0");

/* Buchungszweig */

Ein_YNR: /* Einlesen Yacht Nr */

    printf("Bitte Yacht-Nr. für Buchung eingeben: \n");
    scanf("%d", &YNR);

/* Yacht auch bekannt ? */

EXEC SQL
    SELECT name, laenge, breite, tiefgang, reg_ort
    INTO :yname, :LAENGE, :BREITE, :WTIEFE:WTind, :reg_ort
    FROM YACHT
    WHERE YNR = :YNR;

if(SQLCODE == 100) {
    printf(" Yacht %4i nicht bekannt \n", YNR);
    goto Ein_YNR;
}
CHECKERR ("SELECT Yacht");
if(WTind == -1) WTIEFE = 0;

printf(" Yacht %s aus %s \n Länge %5.2f Breite %5.2f Tiefgang %5.2f \n",
       yname, reg_ort, LAENGE, BREITE, WTIEFE);

Ein_LPNR: /* Einlesen Liegeplatz Nr */

```

```

printf("Bitte Liegeplatz-Nr. für Buchung eingeben:\n");
scanf("%d", &LPNR);

/* Liegeplatz wirklich frei? */

EXEC SQL
  SELECT LPNR INTO :LPNR
  FROM LIEGEPLATZ L
  WHERE NOT EXISTS (
    SELECT * FROM BELEGUNG
    WHERE LPNR = L.LPNR AND
      (VON BETWEEN :von_datum AND :bis_datum OR
      BIS BETWEEN :von_datum AND :bis_datum OR
      :von_datum BETWEEN VON AND BIS OR
      :bis_datum BETWEEN VON AND BIS )
    ) AND L.LPNR = :LPNR;

if(SQLCODE == 100) {
  printf(" Liegplatz %i nicht frei \n", LPNR);
  goto Ein_LPNR;
}
CHECKERR ("SELECT Liegeplatz");

/* alles klar, also reservieren */

EXEC SQL
  INSERT INTO BELEGUNG (LPNR, YNR, VON, BIS)
  VALUES (:LPNR, :YNR, :von_datum, :bis_datum);
CHECKERR ("INSERT Belegung");

printf("Liegeplatz %i für Yacht %i vom %s bis %s reserviert \n",
  LPNR, YNR, von_datum, bis_datum);

EXEC SQL CONNECT RESET;
CHECKERR ("DISCONNECT");
return 0;
}
/* end of program */

```

Da die Spalte TIEFGANG in der Tabelle YACHT NULL-Werte enthalten darf, muß für sie im Programm eine Indikatorvariable (WTind) definiert und nach jedem FETCH abgefragt werden, ob durch den Wert -1 ein NULL-Wert angezeigt wird. In einem solchen Fall wäre der Inhalt der zugehörigen Variable WTIEFE nicht verwertbar. Der Einfachheit halber setze ich diese dann auf 0.

Zur Fehlerbehandlung nach SQL-Befehlen benutze ich auch in diesem Beispiel das Unterprogramm *check_error()*, das aus den älteren DB2-Beispielprogrammen stammt und die Informationen des SQL-Kommunikationsbereichs SQLCA aufbereitet und ausgibt.

Auch das Programm BUCHMARI hat keine Schnittstellen zu einer grafischen Benutzerschnittstelle. Es wird im Befehlszeilen-Umfeld mit Angabe des Zeitraums (buchmari <von-datum> <bis-datum>) aufgerufen.

Der Trigger, der überprüft, daß die Yacht mit ihren Abmessungen auch in den ausgewählten Liegeplatz paßt, ist SQL-Code, der in der Datenbank verankert ist. Das Programm hat keinen Einfluss auf seine Funktionsweise. Lediglich im Falle einer ausführlichen Fehlerbehandlung könnte das Programm auf den speziellen Abbruch des INSERT-Befehls durch den Trigger abfragen.

Der Trigger wurde folgendermaßen definiert:

```
-- connect to marina;
drop trigger passend;
CREATE TRIGGER PASSEND
  NO CASCADE BEFORE INSERT ON BELEGUNG
  REFERENCING NEW AS N
  FOR EACH ROW MODE DB2SQL
  WHEN (NOT EXISTS (
    SELECT * FROM YACHT Y, LIEGEPLATZ L
    WHERE YNR = N.YNR AND LPNR = N.LPNR
    AND Y.LAENGE <= L.LAENGE
    AND Y.BREITE < L.BREITE
    AND Y.TIEFGANG < L.WASSERTIEFE
  )
  )
  SIGNAL SQLSTATE '90002' ('Yacht größer Liegeplatz')
;
```

4. Anwendungsprogrammierung in COBOL

Beispiel-
Programm
*Liegeplatz-
Gebühren
kassieren*

Für das Beispiel-Programm wähle ich eine Aufgabenstellung aus unserer Yachthafen-Fallstudie: Der Hafenmeister unserer Marina verwaltet nicht nur die Belegung der Liegeplätze, er muß auch die Liegeplatz-Gebühren kassieren. Dazu erstellt ihm ein COBOL-Programm Rechnungen für alle Yachten, die ab dem ausgewählten Tag einen Liegeplatz belegen, das heißt das VON-Datum der Belegung muß gleich dem vorgegebenen Datum sein. Die Rechnung wird für die gebuchte Liegedauer und auf den registrierten Eigner ausgestellt. Für die Rechnungsanschrift nehme ich vereinfachend die erste Adresse des Eigners, das heißt diejenige seiner Adressen, die die niedrigste Adreßnummer besitzt (solche Vereinfachungen sind zwar im Sinne einer korrekten Datenmodellierung nicht vernünftig, aber leider durchaus praxisnah). Eindeutige Rechnungsnummer, Kostensätze und andere Basisdaten erhalte ich aus einer jahresbezogenen Tabelle für Geschäftsdaten.

Hier nun der Quellcode als Eingabe für den DB2-Precompiler:

Quellcode für
Beispiel-
Programm
*Liegeplatz-
Gebühren
kassieren*

```

IDENTIFICATION DIVISION.
*****
*           MARINA - DEMO-ANWENDUNG
* RECHNUNGSSCHREIBUNG FÜR ALLE YACHTEN, DIE AB HEUTE EINEN LIEGE-
* PLATZ BENUTZEN.
* ALLE RECHTE BEI UNTERNEHMENSBERATUNG PUERNER, DORTMUND
*****

PROGRAM-ID.    MARIRECH.
AUTHOR.        PUERNER.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
```

```
SOURCE-COMPUTER.  OS2-PC.
OBJECT-COMPUTER.  OS2-PC.
SPECIAL-NAMES.  DECIMAL-POINT IS COMMA.

INPUT-OUTPUT SECTION.

FILE-CONTROL.
    SELECT AUSGABE ASSIGN TO "RECHNUNG.LST".

DATA DIVISION.
FILE SECTION.
FD  AUSGABE LABEL RECORDS OMITTED.
01  ZEILE          PIC X(82).
WORKING-STORAGE SECTION

*****
*  SQL - DEKLARATIONEN
*****

    EXEC SQL
        INCLUDE SQLCA
    END-EXEC.

    EXEC SQL
        BEGIN DECLARE SECTION
    END-EXEC

01  NAME1          PIC X(24).
01  NAME2          PIC X(24).
01  ANSCHRIFT1     PIC X(24).
01  ANSCHRIFT2     PIC X(24).
01  TELEFON       PIC X(15).
01  TELEFAX       PIC X(15).
01  KOSTENSATZ    PIC S99V99  COMP-3.
01  ZUSCHLAG      PIC S99V99  COMP-3.
01  ZIND          PIC S9(4)   COMP-5.
01  MWST          PIC S99V9   COMP-3.
01  RECHNR        PIC S9(9)   COMP-5.
01  YNAME         PIC X(24).
01  LAENGE        PIC S9999V99 COMP-3.
01  LPNR          PIC S9(4)   COMP-5.
01  ZEITRAUM      PIC S9(4)   COMP-5.
```

```

01 STROM                PIC X(4).
01 WASSER               PIC X(4).
01 PNR                 PIC S9(4)   COMP-5.
01 PNAME               PIC X(24).
01 VORNAME             PIC X(24).
01 NKZ                 PIC X(3).
01 PLZ                 PIC X(6).
01 ORT                 PIC X(24).
01 STRASSE             PIC X(32).
01 STRIND              PIC S9(4)   COMP-5.
01 WJAHR               PIC X(4).
01 DATUMV              PIC X(10).

EXEC SQL
  END DECLARE SECTION
END-EXEC.
*****
* SQL - DEKLARATIONEN
*****

EXEC SQL
  DECLARE CR001 CURSOR FOR
  SELECT NAME,
  Y.LAENGE,
  B.LPNR,
  DAYS(BIS)-DAYS(VON),
  STROM,
  WASSERANSCHLUSS,
  PNR
  FROM BELEGUNG B, YACHT Y, LIEGEPLATZ L
  WHERE B.LPNR = L.LPNR
        AND B.YNR = Y.YNR
        AND VON = :DATUMV
  ORDER BY B.LPNR
END-EXEC.

*****
*   L O K A L E   V A R I A B L E
*****

01 PROGNAME            PIC X(8)      VALUE "MARIN01".
01 EOF                 PIC S9(4) COMP-5 VALUE +100.
01 FEHLER              PIC --999.

```

```

01  DAT.
    05  TT          PIC X(2).
    05  FILLER      PIC X.
    05  MM          PIC X(2).
    05  FILLER      PIC X.
    05  JJ          PIC X(4).
01  LGEBUEHR       PIC 9(6)V99 COMP-3.
01  MWSTB          PIC 9(6)V99 COMP-3.
01  ZUSCHLAG1     PIC 9(6)V99 COMP-3.
01  ZUSCHLAG2     PIC 9(6)V99 COMP-3.
01  SUMME1        PIC 9(6)V99 COMP-3.
01  SUMME2        PIC 9(6)V99 COMP-3.
*****
*   DRUCKAUSGABE
*****

01  KOPF1.
    02  FILLER      PIC X(54) VALUE SPACE.
    02  NAME1K     PIC X(24).
01  KOPF2.
    02  FILLER      PIC X(54) VALUE SPACE.
    02  NAME2K     PIC X(24).
01  KOPF3.
    02  FILLER      PIC X(54) VALUE SPACE.
    02  ANSCHRIFT1K PIC X(24).
01  KOPF4.
    02  FILLER      PIC X(54) VALUE SPACE.
    02  ANSCHRIFT2K PIC X(24).
01  KOPF5.
    02  FILLER      PIC X(54) VALUE SPACE.
    02  FILLER      PIC X(09) VALUE "Telefon ".
    02  TELEFONK   PIC X(15).
01  KOPF6.
    02  FILLER      PIC X(54) VALUE SPACE.
    02  FILLER      PIC X(09) VALUE "Telefax ".
    02  TELEFAXK   PIC X(15).
01  DATZEILE.
    02  FILLER      PIC X(68) VALUE SPACE.
    02  DATUM.
        05  TT          PIC X(2).
        05  FILLER      PIC X   VALUE ". ".
        05  MM          PIC X(2).
        05  FILLER      PIC X   VALUE ". ".

```

```

    05  JJ                PIC X(4).
01  BETREFF1.
    02  FILLER           PIC X(8) VALUE SPACE.
    02  FILLER           PIC X(14)
                           VALUE "Rechnung Nr.: ".
    02  RECH-NR         PIC ZZZ99.
01  BETREFF2.
    02  FILLER           PIC X(8) VALUE SPACE.
    02  FILLER           PIC X(16)
                           VALUE "Liegeplatz Nr.: ".
    02  LIEG-NR        PIC ZZ9.
    02  FILLER           PIC X(17)
                           VALUE " - Schiffslänge: ".
    02  LAENGEA         PIC Z9,9.
    02  FILLER           PIC X(6) VALUE " Meter".
01  AZEILE1.
    02  FILLER           PIC X(4) VALUE SPACE.
    02  NAME-VORN       PIC X(50).
01  AZEILE2.
    02  FILLER           PIC X(4) VALUE SPACE.
    02  STRASSEA        PIC X(24).
01  AZEILE3.
    02  FILLER           PIC X(4) VALUE SPACE.
    02  NKZA            PIC X(3).
    02  FILLER           PIC XX VALUE "- ".
    02  PLZA            PIC X(6).
    02  FILLER           PIC X VALUE SPACE.
    02  ORTA            PIC X(24).
01  RZEILE1.
    02  FILLER           PIC X(8) VALUE SPACE.
    02  TAGE            PIC ZZZ9.
    02  FILLER           PIC X(13) VALUE " Tage zu €/m ".
    02  KOSTENSATZA     PIC 99,99.
    02  FILLER           PIC X(25) VALUE SPACE.
    02  LIEGEKOSTEN     PIC ZZZZ99,99.
01  RZEILE2.
    02  FILLER           PIC X(35)
                           VALUE "
                           Zuschlag für Wasseranschluß".
    02  FILLER           PIC X(21) VALUE SPACE.
    02  WZUSCHLAG       PIC ZZZZ99,99.
01  RZEILE3.
    02  FILLER           PIC X(34)
                           VALUE "
                           Zuschlag für Stromanschluß".

```

```
02 FILLER PIC X(22) VALUE SPACE.
02 SZUSCHLAG PIC ZZZZ99,99.
01 RZEILE4.
02 FILLER PIC X(22)
   VALUE " Summe Gebühren".
02 FILLER PIC X(34) VALUE SPACE.
02 SUMME-1 PIC ZZZZ99,99.
01 RZEILE5.
02 FILLER PIC X(23)
   VALUE " Mehrwertsteuer ".
02 MWST-SATZ PIC 99.
02 FILLER PIC X VALUE "%".
02 FILLER PIC X(30) VALUE SPACE.
02 MWST-BETRAG PIC ZZZZ99,99.
01 RZEILE6.
02 FILLER PIC X(19)
   VALUE " Gesamtsumme".
02 FILLER PIC X(37) VALUE SPACE.
02 SUMME-2 PIC ZZZZ99,99.
01 STZEILE.
02 FILLER PIC X(8) VALUE SPACE.
02 STRICHE PIC X(58) VALUE ALL "-".
* -----
```

PROCEDURE DIVISION.

STEUERUNG SECTION.

```
*****
* STANDARD STEUERUNG
*****
```

```
PERFORM VORLAUF
PERFORM RECHNUNG
PERFORM NACHLAUF
.
Z. STOP RUN.
```

* -----

VORLAUF SECTION.

```

*****
*   PRUEFEN UEBERGEBENE PARAMETER
*****
*
*   RECHNUNGSDATUM ABFRAGEN
*
  DISPLAY "Bitte Rechnungsdatum (TT.MM.JJJJ) eingeben:"
  ACCEPT DAT
*   ACCEPT DAT FROM DATE
  MOVE DAT TO DATUMV
  MOVE CORR DAT TO DATUM
*
*   ANMELDUNG AN DB2
*
  EXEC SQL
    CONNECT TO MARINA
  END-EXEC
  IF SQLCODE NOT = 0
    PERFORM SQL-FEHLER
  END-IF
*
*   LESEN BASIS-DATEN
*
  EXEC SQL
    SELECT NAME_1, NAME_2, ANSCHRIFT_1, ANSCHRIFT_2,
           TELEFON, TELEFAX, W_JAHR,
           METER_KOST, ZUSCHLAG, MWST_SATZ, RECHNR
    INTO   :NAME1, :NAME2, :ANSCHRIFT1, :ANSCHRIFT2,
           :TELEFON, :TELEFAX, :WJAHR,
           :KOSTENSATZ, :ZUSCHLAG:ZIND, :MWST, :RECHNR
    FROM   WIRTJAHR
    WHERE  W_JAHR = (SELECT MAX(W_JAHR) FROM WIRTJAHR)
  END-EXEC
  IF SQLCODE NOT = 0
    PERFORM SQL-FEHLER
  END-IF
  MOVE NAME1      TO NAME1K
  MOVE NAME2      TO NAME2K
  MOVE ANSCHRIFT1 TO ANSCHRIFT1K
  MOVE ANSCHRIFT2 TO ANSCHRIFT2K
  MOVE TELEFON    TO TELEFONK
  MOVE TELEFAX    TO TELEFAXK

```

```
      IF ZIND = -1
*     ZUSCHLAG IST >NULL<
          MOVE ZERO TO ZUSCHLAG
      END-IF
*
*     CURSOR EROEFFNEN
*
      EXEC SQL
          OPEN CR001
      END-EXEC
      IF SQLCODE NOT = 0
          PERFORM SQL-FEHLER
      END-IF
*
*     AUSGABE EROEFFNEN
*
      OPEN OUTPUT AUSGABE
      .
Z.  EXIT.

* -----

      NACHLAUF SECTION.

*****
*     NACHARBEITEN
*****

*     CURSOR SCHLIESSEN

      EXEC SQL CLOSE CR001
      END-EXEC
      EXEC SQL
          UPDATE WIRTJAHR
              SET RECHNR = :RECHNR
              WHERE W_JAHR = :WJAHR
      END-EXEC
      EXEC SQL COMMIT WORK
      END-EXEC

*     AUSGABE SCHLIESSEN

      CLOSE AUSGABE
```

```
.
Z.  EXIT.

* -----

RECHNUNG SECTION.

*****
* AUSFUEHRUNG DER RECHNUNGSSCHREIBUNG: LESEN DER BELEGUNGEN MIT
* YACHT- UND LIEGEPLATZDATEN, NACHLESEN DER EIGNERDATEN MIT 1.
* ANSCHRIFT ALS RECHNUNGSANSCHRIFT
*****
      PERFORM UNTIL SQLCODE = EOF
        EXEC SQL
          FETCH CR001 INTO :YNAME,
                          :LAENGE,
                          :LPNR,
                          :ZEITRAUM,
                          :STROM,
                          :WASSER,
                          :PNR

          END-EXEC
        IF SQLCODE = 0
          PERFORM GET-EIGNER
          PERFORM COMPUTE-GEB
          PERFORM PRINT-RECHNUNG
        ELSE IF SQLCODE NOT = EOF
          PERFORM SQL-FEHLER
        END-IF
      END-PERFORM
.
Z.  EXIT.

* -----

GET-EIGNER SECTION.

*****
* EIGNERDATEN ZUR YACHT LESEN (1. ANSCHRIFT = RECHNUNGSANSCHRIFT)
*****
```

```

EXEC SQL
  SELECT NAME,
         VORNAME,
         NKZ,
         POSTLZ,
         ORT,
         STRASSE
  INTO  :PNAME,
        :VORNAME,
        :NKZ,
        :PLZ,
        :ORT,
        :STRASSE:STRIND
  FROM PERSON P, ADRESSE A
  WHERE A.PNR = P.PNR
        AND ADR_NR = (SELECT MIN(ADR_NR) FROM ADRESSE
                      WHERE PNR = :PNR)

END-EXEC
IF SQLCODE NOT = 0
  PERFORM SQL-FEHLER
END-IF
IF STRIND = -1
  STRASSE IST >NULL<
  MOVE SPACES TO STRASSE
END-IF
.
Z.  EXIT.

* -----

COMPUTE-GEB SECTION.

*****
*           GEBÜHREN ERRECHNEN           *
*****

MOVE ZERO TO SUMME1, ZUSCHLAG1, ZUSCHLAG2, SUMME2
COMPUTE LGEBUEHR = ZEITRAUM * KOSTENSATZ * LAENGE
IF STROM = "JA"
  COMPUTE ZUSCHLAG1 = ZEITRAUM * ZUSCHLAG
END-IF
IF WASSER = "JA"
  COMPUTE ZUSCHLAG2 = ZEITRAUM * ZUSCHLAG

```

```

END-IF
ADD LGEUEHR, ZUSCHLAG1, ZUSCHLAG2 TO SUMME1
COMPUTE MWSTB = SUMME1 * MWST / 100
ADD SUMME1, MWSTB TO SUMME2
ADD 1 TO RECHNR
.
Z. EXIT.

```

* -----

PRINT-RECHNUNG SECTION.

```

*****
*      AUSGABE DER RECHNUNG      *
*****
      MOVE RECHNR      TO RECH-NR
      MOVE LPNR        TO LIEG-NR
      MOVE LAENGE      TO LAENGEA
      MOVE ZEITRAUM    TO TAGE
      STRING PNAME, ", ", VORNAME DELIMITED BY SPACE
                          INTO NAME-VORN
      MOVE STRASSE     TO STRASSEA
      MOVE NKZ         TO NKZA
      MOVE PLZ         TO PLZA
      MOVE ORT         TO ORTA
      MOVE KOSTENSATZ  TO KOSTENSATZA
      MOVE LGEUEHR     TO LIEGEKOSTEN
      MOVE ZUSCHLAG2   TO WZUSCHLAG
      MOVE ZUSCHLAG1   TO SZUSCHLAG
      MOVE SUMME1      TO SUMME-1
      MOVE SUMME2      TO SUMME-2
      MOVE MWST        TO MWST-SATZ
      MOVE MWSTB       TO MWST-BETRAG
      WRITE ZEILE FROM KOPF1 AFTER PAGE
      WRITE ZEILE FROM KOPF2 AFTER 1
      WRITE ZEILE FROM KOPF3 AFTER 1
      WRITE ZEILE FROM KOPF4 AFTER 1
      WRITE ZEILE FROM KOPF5 AFTER 1
      WRITE ZEILE FROM KOPF6 AFTER 1
      WRITE ZEILE FROM AZEILE1 AFTER 8
      WRITE ZEILE FROM AZEILE2 AFTER 1
      WRITE ZEILE FROM AZEILE3 AFTER 2

```

```

WRITE ZEILE FROM DATZEILE AFTER 4
WRITE ZEILE FROM BETREFF1 AFTER 4
WRITE ZEILE FROM BETREFF2 AFTER 2
WRITE ZEILE FROM STZEILE AFTER 4
WRITE ZEILE FROM RZEILE1 AFTER 2
WRITE ZEILE FROM RZEILE2 AFTER 2
WRITE ZEILE FROM RZEILE3 AFTER 2
WRITE ZEILE FROM STZEILE AFTER 2
WRITE ZEILE FROM RZEILE4 AFTER 1
WRITE ZEILE FROM RZEILE5 AFTER 1
WRITE ZEILE FROM STZEILE AFTER 1
WRITE ZEILE FROM RZEILE6 AFTER 1
.
Z.  EXIT.
* -----
FEHLER SECTION.

*****
*          BEARBEITEN SQL-FEHLER          *
*****

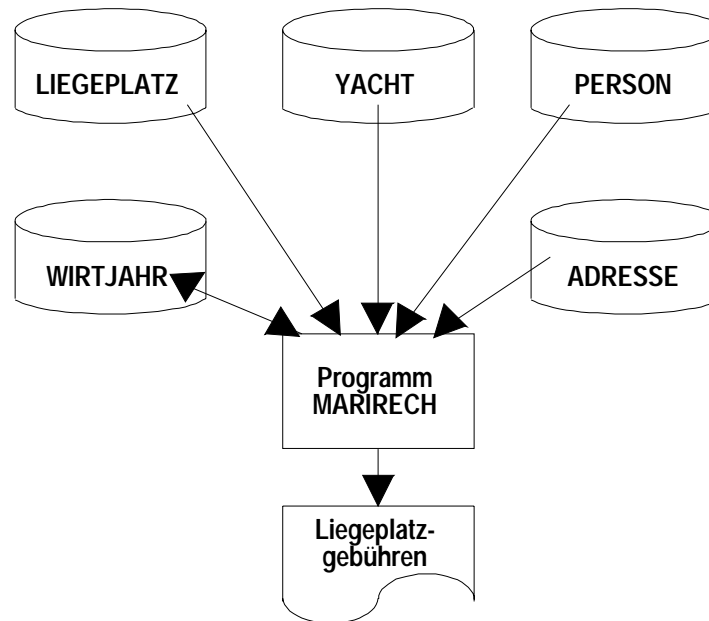
MOVE SQLCODE TO FEHLER
DISPLAY "DATENBANK-FEHLER: ", FEHLER

EXEC SQL ROLLBACK WORK
END-EXEC
.
Z.  STOP RUN.

```

Ich lese in dem Programm zunächst die Basisdaten aus der Tabelle `WIRTJAHR` mit einem einzelsatzorientierten `SELECT`-Befehl. Dann lese ich in einer Schleife über den Cursor `CR001` die abzurechnenden Belegungsdaten zusammen mit den rechnungsrelevanten Daten der Tabellen `LIEGEPLATZ` und `YACHT`. Dazu hole ich mir die Daten zur Eigner-Anschrift aus den Tabellen `PERSON` und `ADRESSE`. Sind alle Rechnungen erstellt, wird die letzte Rechnungsnummer in der Tabelle `WIRTJAHR` mit einem `UPDATE`-Befehl gesichert.

Bild 4-1:
Tabellen in
MARIRECH



Ich hätte natürlich die Daten der Belegung samt Liegeplatz-, Yacht-, Eigner- und Adreßdaten mit einem Join über fünf Tabellen in einem SELECT lesen können, doch sollte der SELECT übersichtlich bleiben und nur das Prinzip der Anwendungsentwicklung mit ESQL gezeigt werden. Ich hoffe daher, mit den beiden SELECTs mit kleineren Joins einen einfacheren Weg gefunden zu haben – auch, wenn man über die Performance dieser Lösung diskutieren kann.

NULL-Wert

Das Problem im Zusammenspiel von relationalen Datenbanken und klassischen Programmiersprachen bei der Behandlung NULL-Werten habe ich schon weiter vorn erwähnt. NULL-Werte müssen mit Hilfe von Indikatoren abgefragt werden:

In unserem Beispiel kann nur in den Feldern ZUSCHLAG und STRASSE der Wert NULL vorkommen. Dafür haben wir die Indikatoren ZIND und STRIND definiert und auf -1 abgefragt. Generell müssen Sie in allen Tabellenspalten, die nicht mit NOT NULL oder mit NOT NULL WITH DEFAULT definiert wurden, mit NULL-Werten rechnen und für sie eine entsprechende Behandlung mit Indikatoren in Ihren Programmen vorsehen.

Wenn Sie bisher hauptsächlich in der Mainframe-Welt tätig waren, wird Ihnen in dem Beispiel-Programm folgendes aufgefallen sein:

- Der COBOL-Datentyp COMP-5 ist nicht auf Mainframes üblich. Er entspricht auf dem PC der ganzzahligen Binärzahl, INTEGER beziehungsweise SMALLINT in DB2. Ihre interne Darstellung weicht auf einigen Hardware-Plattformen insofern von der vertrauten Darstellung ab, daß das linke Byte einer Halb-, Voll- oder Doppelwortzahl die weniger signifikanten Stellen als das rechts davon liegende Byte enthält. Man sagt, daß diese Reihenfolge in erster Linie bei Prozessoren vorkommt, die aus der 8-Bit-Welt in die 16- und 32-Bit-Welt empor gewachsen sind. In der originären 32-Bit-Welt ist es dagegen üblich, daß die höherwertigen Bytes links von den niederwertigen liegen – wie es ja bei den Bits innerhalb der Bytes immer der Fall ist. Zu den Vertretern dieser Technologie gehören alle bekannten Mainframes.

Das Beispiel-Programm wurde – als Teil einer Fallstudie „Yachthafen“ – auch vergleichsweise mit ORACLE und anderen relationalen Datenbank-Managementsystemen realisiert. Ich habe mich bei der Codierung des Programms am SQL-Standard orientiert und habe Erweiterungen individueller SQL-Dialekte nicht berücksichtigt. So nutze ich in diesem Beispiel außer dem SQLCODE auch nicht die Felder des SQL-Kommunikationsbereich SQLCA, der üblicherweise in vielen Programmen genutzt wird. Dieser hat folgenden Aufbau:

SQL-Kommunikationsbereich
 SQLCA

```
* SQL Communication Area - SQLCA
01 SQLCA SYNC.
   05 SQLCAID PIC X(8) VALUE "SQLCA ".
   05 SQLCABC PIC S9(9) COMP-5 VALUE 136.
   05 SQLCODE PIC S9(9) COMP-5.
   05 SQLERRM.
       49 SQLERRML PIC S9(4) COMP-5.
       49 SQLERRMC PIC X(70).
   05 SQLERRP PIC X(8).
   05 SQLERRD OCCURS 6 TIMES PIC S9(9) COMP-5.
   05 SQLWARN.
       10 SQLWARN0 PIC X.
       10 SQLWARN1 PIC X.
       10 SQLWARN2 PIC X.
       10 SQLWARN3 PIC X.
       10 SQLWARN4 PIC X.
       10 SQLWARN5 PIC X.
       10 SQLWARN6 PIC X.
       10 SQLWARN7 PIC X.
```

```
10 SQLWARN8 PIC X.  
10 SQLWARN9 PIC X.  
10 SQLWARNA PIC X.  
05 SQLSTATE PIC X(5).
```

Wer seine Programme nur im Gültigkeitsbereich von IBMs SAA einsetzt, sollte die SQLCA für eine ausführliche Fehlerbehandlung nutzen. Auch die Hersteller anderer relationaler Systeme, z.B. ORACLE, unterstützen einen solchen Bereich mit sehr ähnlichem Aufbau.

Precompiler
PREP

Der Precompiler PREP erwartet, daß die angesprochenen Datenbank-Objekte zur Übersetzungszeit angelegt sind und prüft Programm-Angaben gegen den Katalog. So entdeckt er Fehler, die sonst erst bei den Testläufen offenkundig würden. Denken Sie daran: je früher Fehler entdeckt werden, desto kostengünstiger ist ihre Beseitigung.



Im Gegensatz zu DB2 für OS/390, dessen Precompiler üblicherweise nicht auf den System-Katalog zugreift, kennt DB2 keinen DECLARE TABLE-Befehl, aus dem die Tabellenstruktur ersichtlich ist. Dies erhöht die Datenunabhängigkeit der Programme gegenüber DB2 unter OS/390.

PREP erzeugt neben einigen Meldungen eine Datei für den BIND-Lauf und eine Datei mit den vorübersetzten SQL-Befehlen für den COBOL-Compiler. Es werden neben den üblichen ANSI-COBOL-Compilern ihrer Betriebssystem-Umgebung auch die COBOL-Compiler von Micro Focus (MF) und IBM unterstützt. Mit dem Parameter TARGET können Sie den gewünschten Compiler explizit vorgeben, zum Beispiel IBMCOB für IBMs 32-Bit-Compiler oder MFCOB16 für den 16-Bit-Compiler von Micro Focus unter OS/2.

PREP erzeugt standardmäßig in einem integrierten BIND-Lauf ein Zugriffspaket (package) in der Datenbank. Mit dem Parameter BINDFILE erhalten Sie ein bindefähiges Modul (proname.BND), mit SYNTAX können Sie eine reine Syntax-Prüfung durchführen.

Zugriffspaket
(package)

Ein Zugriffspaket (package) enthält die Zugriffe, die der Optimizer aufgrund der physischen Gegebenheiten in der Datenbank für die SQL-Befehle eines Programms ausgewählt hat. Ändern sich die physischen Gegebenheiten der Datenbank, zum Beispiel durch einen neuen Index oder durch andere Statistikwerte im DB2-Katalog, kann ein erneutes Binden der Zugriffspakete notwendig oder sinnvoll sein.

Ein Zugriffspaket unterteilt sich in Abschnitte (section), die die Zugriffe für einen SQL-Befehl aufnehmen. Je Zugriffspaket wird eine Zeile mit allgemeinen Informationen in der Katalog-Tabelle `SYSIIBM.SYSPLAN` und je Abschnitt mindestens eine Zeile mit den verschlüsselten Zugriffsdaten in der Katalog-Tabelle `SYSIIBM.SYSSECTION` gespeichert. Die Abhängigkeiten des Zugriffspakets von Datenbank-Objekten wie Tabellen oder Indizes werden in `SYSIIBM.SYSPLANDEP` festgehalten, Berechtigungen in `SYSIIBM.SYPLANAUTH`. Die zugehörigen SQL-Befehle werden in der Tabelle `SYSIIBM.SYSSTMT` gespeichert.

Die Aufrufe von COBOL-Compiler und Linker (linkage editor) sind abhängig von den Gegebenheiten Ihrer Installation.

Das fertige Programm `MARIRECH` ist, wie Sie sicher schon erkannt haben, kein Programm mit Schnittstellen zu einer grafischen Benutzerschnittstelle. Es kann im Befehlszeilenmodus aufgerufen. Die Rechnungen werden in die Datei `RECHNUNG.LST` geschrieben und können mit dem `PRINT`-Befehl ausgedruckt werden.

Hier ein Beispiel einer Liegeplatz-Abrechnung vom 10.1.2003 für einen Dauerlieger:

	Marina Neuwismar Der Hafenmeister Am Yachthafen 1 D-29999 Neuwismar Telefon 0333/12-0 Telefax 0333/12-556
Blöd,Hein Im Wrack 1	
B - 100 Bärenklippe	
	10.01.2003
Rechnung Nr.: 9336	
Liegeplatz Nr.: 3 - Schiffslänge: 5,0 Meter	

355 Tage zu €/m 02,50	4437,50
Zuschlag für Wasseranschluß	00,00
Zuschlag für Stromanschluß	00,00

Summe Gebühren	4437,50
Mehrwertsteuer 16%	710,00

Gesamtsumme	5147,50

5. ESQL-Befehle

Es gilt zwar der Grundsatz, daß alle SQL-Befehle auch aus Programmen heraus absetzbar sind, dennoch sind einige Befehle oder Befehlsformen speziell auf die Belange der Anwendungsprogrammierung zugeschnitten. In diesem Abschnitt gebe ich Ihnen einen kurzen Überblick über die statischen SQL-Befehle beziehungsweise -Befehlsformen, die nur in Anwendungsprogrammen benutzt werden können. Dabei beschränke ich mich bewußt auf die gebräuchlichsten Parameter. Insbesondere Befehlsvarianten für die Programmierung von Stored Procedures lasse ich außer acht. Für eine ausführliche Darstellung dieser Befehle mit allen Parametern verweise ich Sie nachdrücklich auf das SQL-Referenzhandbuch Ihrer aktuellen DB2-Version.

Die folgenden Befehle werde ich Ihnen kurz vorstellen:

- BEGIN DECLARE SECTION (siehe Kapitel 5.1, Seite 5-2)
- END DECLARE SECTION (siehe Kapitel 5.2, Seite 5-2)
- INCLUDE (siehe Kapitel 5.3, Seite 5-3)
- DECLARE CURSOR (siehe Kapitel 5.4, Seite 5-4)
- OPEN (siehe Kapitel 5.5, Seite 5-7)
- FETCH (siehe Kapitel 5.6, Seite 5-9)
- CLOSE (siehe Kapitel 5.7, Seite 5-10)
- DELETE (siehe Kapitel 5.8, Seite 5-10)
- INSERT (siehe Kapitel 5.9, Seite 5-11)
- UPDATE (siehe Kapitel 5.10, Seite 5-13)
- SELECT INTO (siehe Kapitel 5.11, Seite 5-15)

- CALL (siehe Kapitel 5.12, Seite 5-17)
- FREE LOCATOR (siehe Kapitel 5.13, Seite 5-19)
- VALUES INTO (siehe Kapitel 5.14, Seite 5-19)
- Compound SQL (siehe Kapitel 5.15, Seite 5-20)
- WHENEVER (siehe Kapitel 5.16, Seite 5-22)

5.1 BEGIN DECLARE SECTION

▶▶ — BEGIN DECLARE SECTION ————— |

Mit BEGIN DECLARE SECTION leiten Sie den Deklarationsteil für die Programm-Variablen ein, die in SQL-Befehlen benutzt werden.

SQL-Befehle sind in der DECLARE SECTION nicht erlaubt.

Die Befehle BEGIN DECLARE SECTION und END DECLARE SECTION müssen paarweise verwendet und dürfen nicht ineinander verschachtelt werden.

Der Befehl darf in Programmen überall dort benutzt werden, wo Variablen-Deklarationen in der jeweiligen Programmiersprache erlaubt sind.

5.2 END DECLARE SECTION

▶▶ — END DECLARE SECTION ————— |

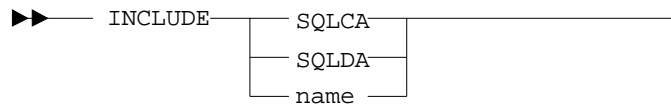
Mit END DECLARE SECTION beenden Sie den Deklarationsteil für die Programm-Variablen, die in SQL-Befehlen benutzt werden.

Die Befehle BEGIN DECLARE SECTION und END DECLARE SECTION müssen paarweise verwendet und dürfen nicht ineinander verschachtelt werden.

SQL-Befehle sind in der DECLARE SECTION nicht erlaubt.

Der Befehl darf in Programmen überall dort benutzt werden, wo Variablen-Deklarationen in der jeweiligen Programmiersprache erlaubt sind.

5.3 INCLUDE



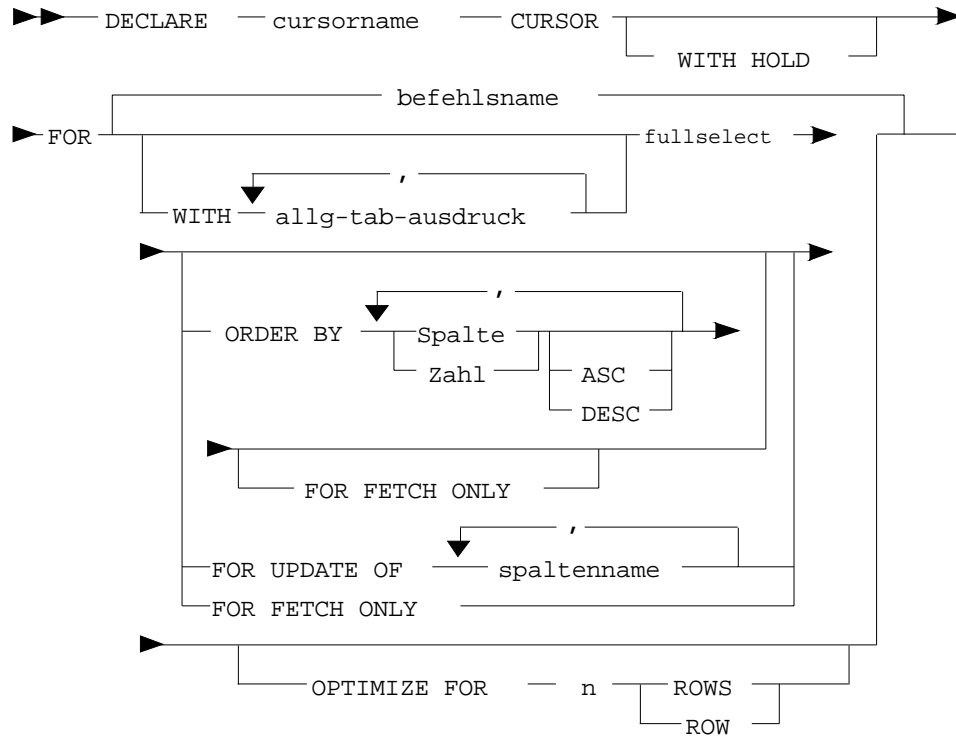
Mit INCLUDE kopieren Sie Deklarationen in ein Quellprogramm. Der Precompiler ersetzt diese Anweisung durch die referenzierten Quellenweisungen.

INCLUDE kann nur an den Stellen des Quellprogramms angegeben werden, wo die eingefügten Quellenweisungen erlaubt sind.

INCLUDE-Anweisungen dürfen ineinander verschachtelt (nested) werden, zyklische Aufrufe sind nicht erlaubt.

Parameter	
SQLCA	Die Beschreibung des SQL-Kommunikationsbereichs SQLCA (SQL communication area) wird in das Programm eingefügt.
SQLDA	Die Beschreibung des SQL-Deskriptorbereichs SQLDA (SQL deskriptor area) wird in das Programm eingefügt.
name	Der Text in der Datei <i>name</i> wird in das Programm eingefügt. <i>name</i> kann ein SQL-Identifizier oder eine Textkonstante in Hochkommata (') eingeschlossen sein. Bei einem SQL-Identifizier wird die Dateinamenextension unterstellt, die zum übersetzten Quellprogramm gehört. Der Text in der angegebenen Datei muß den Regeln der verwendeten Programmiersprache gehorchen. Dieser Parameter ist in COBOL-Programmen nicht erlaubt.

5.4 DECLARE CURSOR



DECLARE CURSOR definiert einen Cursor zur satzweisen Bearbeitung der Ergebnistabelle des darin enthaltenen SELECT-Befehls. Der Cursor referenziert die Ergebnistabelle und zeigt auf eine Zeile darin. Diese Anweisung ist kein ausführbarer Befehl.

Unterprogramme können keine Cursor benutzen, die im aufrufenden Programm definiert wurden.

Der enthaltene SELECT-Befehl wird beim Öffnen des Cursor ausgeführt (siehe *OPEN*, Seite #). Daher weisen die im SELECT enthaltenen System-Variablen CURRENT DATE, CURRENT TIME oder CURRENT TIMESTAMP denselben Wert bei jedem FETCH-Befehl auf.

Dieser Befehl kann nicht mit PREPARE dynamisch übersetzt werden.

Ein Cursor ist nur zum Lesen bestimmt, wenn

- das Ergebnis des SELECT nicht änderungsfähig ist. Dies ist der Fall, wenn
 - die erste FROM-Angabe mehr als eine Tabelle enthält,
 - die erste FROM-Angabe eine nicht änderungsfähige Sicht enthält,
 - die SELECT-Angabe DISTINCT oder eine Spaltenfunktion enthält,
 - die äußere Abfrage GROUP BY oder HAVING enthält,
 - die Abfrage eine Unterabfrage auf dieselbe Tabelle enthält, die in der FROM-Angabe der Hauptabfrage enthalten ist,
 - die äußere Abfrage einen Mengenoperator enthält,
 - die Abfrage eine ORDER BY-Angabe enthält,
- die Cursor-Deklaration FOR FETCH ONLY¹ enthält,
- die Cursor-Deklaration *keine* FOR UPDATE OF-Angabe enthält und das Programm mit der Precompiler-Option LANGLEVEL SAA1 übersetzt wird, keinen DELETE-Befehl mit Bezug auf den Cursor enthält und der Cursor nicht in ein Zugriffspaket mit dynamischen SQL-Befehlen zusammengebunden ist.

Ein Cursor ist zum Ändern bestimmt, wenn

- die Cursor-Deklaration FOR UPDATE OF enthält,
- das Programm mit der Precompiler-Option LANGLEVEL MIA oder SQL92E übersetzt wird und der Cursor änderungsfähig ist,
- ein DELETE-Befehl Bezug auf den Cursor nimmt.

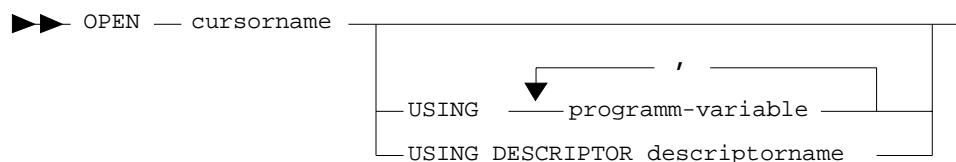
Ein Cursor ist zweideutig (ambiguous), wenn keine der vorgenannten Bedingungen zutrifft und das Programm dynamische SQL-Befehle übersetzt und ausführt. Zweideutige Cursor sind nur dann änderungsfähig, wenn das Programm mit der Precompiler-Option BLOCKING NO oder UNAMBIG übersetzt wurde.

¹ statt FOR FETCH ONLY kann auch FOR READ ONLY benutzt werden

Parameter	
cursorname	identifiziert den Cursor und darf nicht mit dem Namen eines anderen Cursor, der im selben Programm definiert wurde, übereinstimmen.
WITH HOLD	<p>erhält benötigte Ressourcen über Transaktionsgrenzen hinweg. Dies bedeutet im Falle eines COMMIT-Befehls, daß</p> <ul style="list-style-type: none"> • eröffnete Cursor, die mit WITH HOLD definiert wurden, geöffnet bleiben und ihre Positionierung in der Ergebnismenge behalten, • bei folgendem DISCONNECT der Cursor zuvor explizit geschlossen werden muß, damit der DISCONNECT nicht scheitert, • alle übersetzten SQL-Befehle (siehe auch PREPARE auf Seite #) erhalten bleiben, die eröffnete Cursor referenzieren, die mit WITH HOLD definiert wurden, • alle Sperren freigegeben werden, außer Tabellensperren für eröffnete Cursor, die mit WITH HOLD definiert wurden. • LOB-Lokatoren freigegeben werden. • Gültige Befehle mit Cursors, die mit WITH HOLD definiert wurden, unmittelbar nach einem COMMIT sind: FETCH, CLOSE. <p>Dies bedeutet im Falle eines ROLLBACK-Befehls, daß</p> <ul style="list-style-type: none"> • alle eröffneten Cursor geschlossen werden, • alle Sperren, die in der Transaktion gesetzt wurden, freigegeben werden, • alle übersetzten Befehle gelöscht werden, • LOB-Lokatoren freigegeben werden.
befehlsname	verweist auf einen SELECT-Befehl, der unter diesem Namen übersetzt wird, bevor der Cursor geöffnet wird. Der Befehlsname darf nicht mit einem anderen Befehlsnamen übereinstimmen, der im selben Programm in einem anderen DECLARE CURSOR angegeben wurde.

Parameter	
fullselect	darf Programm-Variable enthalten, die zuvor im Quellprogramm als solche deklariert wurden.
allg-tab-ausdruck	ermöglicht die Definition einer Ergebnistabelle unter einem Namen, der als Tabellename in einer FROM Klausel eines folgenden fullselect benutzt werden kann.
ORDER BY	sortiert die Ergebnis-Tabelle aufsteigend (ASC) oder absteigend (DESC) nach den angegebenen Spalten. Sie können statt des Spaltennamens auch die relative Position der Spalten in der Liste angeben (nützlich bei arithmetischen Ausdrücken ohne Namen).
FOR UPDATE OF	meldet die Änderungsabsicht an.
FOR FETCH ONLY	definiert den Cursor als reinen Lese-Cursor (read only). Es kann auch FOR READ ONLY verwendet werden.
OPTIMIZE FOR	beeinflusst die Zugriffsoptimierung und gibt die Größe des Kommunikationspuffers für geblockte Cursor vor.

5.5 OPEN



OPEN öffnet einen deklarierten Cursor und führt das zugehörige SELECT aus. Dabei werden die aktuellen Werte der Programm-Variablen benutzt, die im SELECT angegeben wurden. Der Cursor wird vor den ersten Satz der Ergebnis-Tabelle positioniert. Ist die Ergebnis-Tabelle leer, verhält sich der Cursor wie nach der letzten gelesenen Zeile.

Für das Lesen der Zeilen aus der Ergebnis-Tabelle müssen Sie FETCH verwenden. Um den SELECT erneut auszuführen, müssen Sie den Cursor schließen (CLOSE) und nochmals öffnen.

DB2 entscheidet, ob das Ergebnis in einer temporären Tabelle zwischengespeichert oder dynamisch erstellt wird. In Abhängigkeit davon können ändernde Befehle (INSERT, DELETE, UPDATE) das Ergebnis nicht oder sehr wohl beeinflussen. Eine Beeinflussung der Ergebnis-Tabelle ist **mit** einer temporären Tabelle *nicht möglich*, **ohne** diese temporäre Tabelle *möglich*, aber nicht immer in der Auswirkung eindeutig vorhersehbar.

Weitere Informationen dazu finden Sie im *Application Development Guide* Ihrer DB2-Version.

Der Befehl darf in Programmen überall dort benutzt werden, wo ausführbare Anweisungen in der jeweiligen Programmiersprache erlaubt sind.

Dieser Befehl kann nicht mit PREPARE dynamisch übersetzt werden.

Parameter	
cursorname	verweist auf den Cursor, der unter diesem Namen mit DECLARE CURSOR zuvor definiert wurde.
USING	leitet eine Liste von Programm-Variablen ein, die bei dynamischen SELECT-Befehlen die Parameter-Platzhalter ersetzen. Die Ersetzung erfolgt der Reihenfolge nach.
USING DESCRIPTOR	verweist auf die SQLDA (SQL dynamic area), die eine Beschreibung der Programm-Variablen enthält. Vor Ausführung des OPEN muß der Benutzer die SQLDA mit den benötigten Angaben versorgen.

5.7 CLOSE

```
▶▶ CLOSE — cursorname ————— |
                                   |
                                   | WITH RELEASE |
                                   |
                                   |—————|
```

CLOSE schließt den Cursor und gibt die Ergebnis-Tabelle frei. Wurde von DB2 eine temporäre Tabelle für die Ergebnis-Tabelle benutzt, wird diese gelöscht.

CLOSE beinhaltet **kein** Transaktionsende (COMMIT oder ROLLBACK).

Der Befehl darf in Programmen überall dort benutzt werden, wo ausführbare Anweisungen in der jeweiligen Programmiersprache erlaubt sind.

Dieser Befehl kann nicht mit PREPARE dynamisch übersetzt werden.

Parameter	
cursorname	verweist auf den Cursor, der unter diesem Namen mit DECLARE CURSOR zuvor definiert wurde.
WITH RELEASE	gibt an, daß alle Lese-Sperren des Cursors freigegeben werden. Diese Angabe ist nur für die Benutzertrennungen RS und RR von Bedeutung und hebt deren spezifische Vorteile auf.

5.8 DELETE

```
▶▶ DELETE FROM — tabelle — WHERE CURRENT OF — cursorname — |
```

Diese Form des DELETE löscht die aktuelle Zeile, auf die der angegebene Cursor zeigt. Der Cursor muß auf einer Zeile positioniert sein. Nach Ausführung des DELETE ist der Cursor vor der nächsten Zeile der Ergebnis-Tabelle positioniert. Wurde die letzte Zeile gelöscht, so ist er hinter der letzten Zeile positioniert.

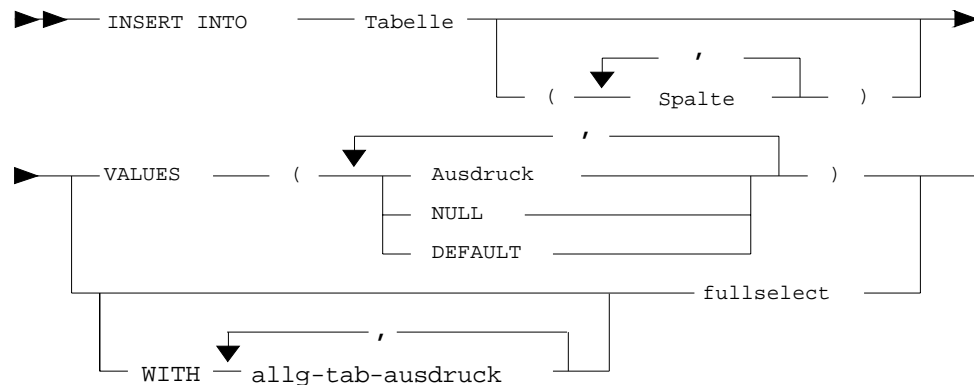
Die Regeln zur Erhaltung der referentiellen Integrität werden entsprechend den Angaben in den Tabellen-Definitionen angewendet.

In SQLERRD(3) der SQLCA sehen Sie die Anzahl der gelöschten Zeilen ohne die Anzahl der Zeilen in anderen Tabellen, die über CASCADE-Regeln gelöscht wurden. Bei diesem Befehl beträgt die Anzahl immer 1. Die von den Regeln der referentiellen Integrität und ausgelösten Triggern betroffenen Zeilen sehen Sie in SQLERRD(5). Die Anzahl umfaßt Zeilen mit zutreffenden CASCADE- oder SET NULL-Angaben.

Der Befehl darf in Programmen überall dort benutzt werden, wo ausführbare Anweisungen in der jeweiligen Programmiersprache erlaubt sind.

Parameter	
tabelle	gibt die Tabelle oder Sicht an, aus der die Zeile gelöscht werden soll. Sie muß mit der Tabellen-Angabe im DECLARE CURSOR übereinstimmen.
cursorname	verweist auf den Cursor, der unter diesem Namen mit DECLARE CURSOR zuvor definiert wurde.

5.9 INSERT



Der Befehl darf in Programmen überall dort benutzt werden, wo ausführbare Anweisungen in der jeweiligen Programmiersprache erlaubt sind.

Zur Unabhängigkeit der Programme von den Tabellendefinitionen sollten Sie immer eine Spaltenliste im Befehl angeben.

In SQLERRD(3) der SQLCA sehen Sie die Anzahl der eingefügten Zeilen.

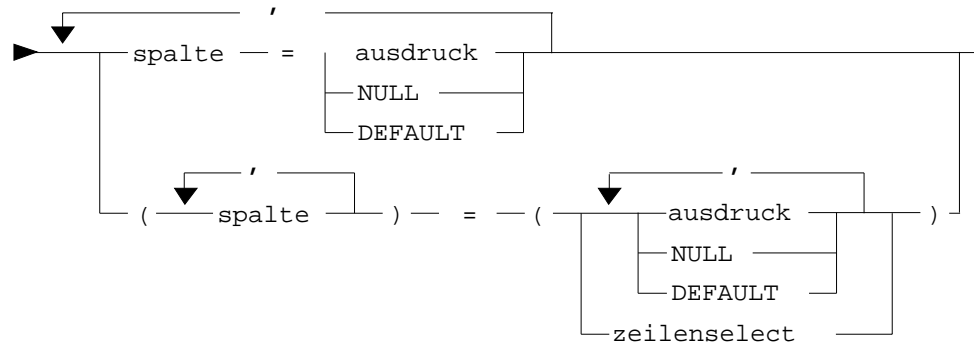
In SQLERRD(5) der SQLCA sehen Sie die Anzahl der Zeilen, die von allen ausgelösten Triggern eingefügt, gelöscht oder verändert wurden.

Parameter	
tabelle	gibt die Tabelle oder Sicht an, in die Zeilen eingefügt werden sollen. Eine angegebene Sicht muß änderungsfähig sein.
(spalte, ..)	gibt eine Liste von Spalten vor, zu denen Datenwerte in der VALUES-Klausel angegeben werden. Die Zuordnung erfolgt der Reihenfolge nach.
VALUES	leitet eine Liste von Datenwerten ein, die eingefügt werden sollen.
fullselect	gibt eine Tabelle als Ergebnis eines vollständigen SELECT-Befehls vor. Ist die Ergebnistabelle leer, wird der SQLCODE +100 gesetzt.
allg-tab- ausdruck	ermöglicht die Definition einer Ergebnistabelle unter einem Namen, der als Tabellename in einer FROM Klausel eines folgenden fullselect benutzt werden kann.

5.10 UPDATE

►► UPDATE — tabelle — SET — | zuordnungs-anw | ►
 ► WHERE CURRENT OF — cursorname — |

zuordnungs-anw:



Diese Form des UPDATE-Befehls ändert die aktuelle Zeile, auf die der angegebene Cursor zeigt. Der Cursor muß auf einer Zeile positioniert sein.

Die Integritätsregeln werden entsprechend den Angaben in den Tabellen- oder Sicht-Definitionen angewendet.

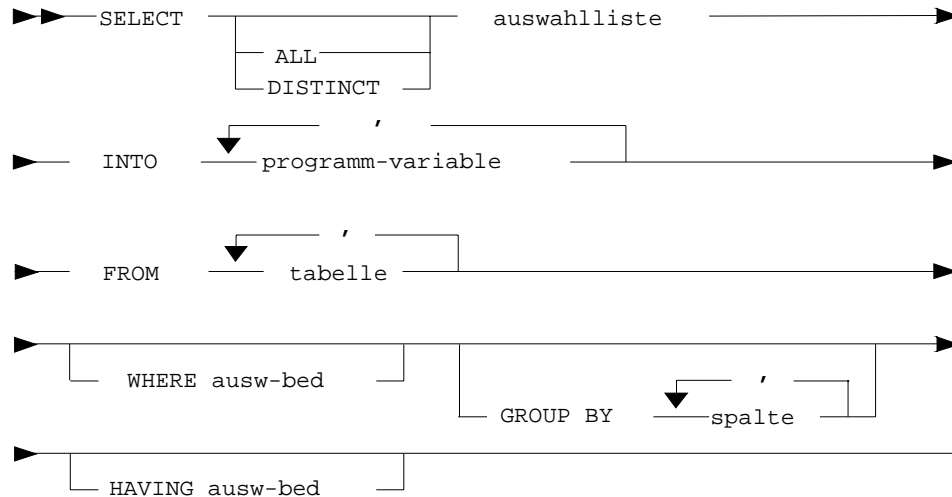
Der Befehl darf in Programmen überall dort benutzt werden, wo ausführbare Anweisungen in der jeweiligen Programmiersprache erlaubt sind.

In SQLERRD(3) der SQLCA sehen Sie die Anzahl der geänderten Zeilen. Bei diesem Befehl beträgt die Anzahl immer 1.

In SQLERRD(5) der SQLCA sehen Sie die Anzahl der Zeilen, die von allen ausgelösten Triggern eingefügt, gelöscht oder verändert wurden.

Parameter	
tabelle	gibt die Tabelle oder Sicht an, aus der die Zeile gelöscht werden soll. Sie muß mit der Tabellen-Angabe im DECLARE CURSOR übereinstimmen und änderungsfähig sein.
SET	<p>Wurde die FOR UPDATE-Klausel nicht spezifiziert, aber das Programm mit Precompiler-Option LANGLEVEL MIA oder SQL92E übersetzt, können in der Zuweisungsliste beliebige Spalten der Tabelle oder Sicht angegeben werden.</p> <p>Wurde die FOR UPDATE-Klausel nicht spezifiziert, und das Programm mit Precompiler-Option LANGLEVEL SAA1 übersetzt, dürfen keine Spalten verändert werden.</p> <p>Ein SELECT in der Zuweisung darf nur eine Zeile zum Ergebnis haben.</p>
cursorname	verweist auf den Cursor, der unter diesem Namen mit DECLARE CURSOR zuvor definiert wurde.
zeilenselect	SELECT-Befehl, der eine Zeile als Ergebnis hat. Kann keine Zeile als Ergebnis ermittelt werden, werden NULLs zugewiesen.

5.11 SELECT INTO



`SELECT INTO` führt eine Abfrage aus und weist das Ergebnis den angegebenen Programm-Variablen zu. Das Ergebnis der Abfrage darf nur aus einer Zeile bestehen. Ist die Ergebnis-Tabelle leer, wird der `SQLCODE +100` gesetzt.

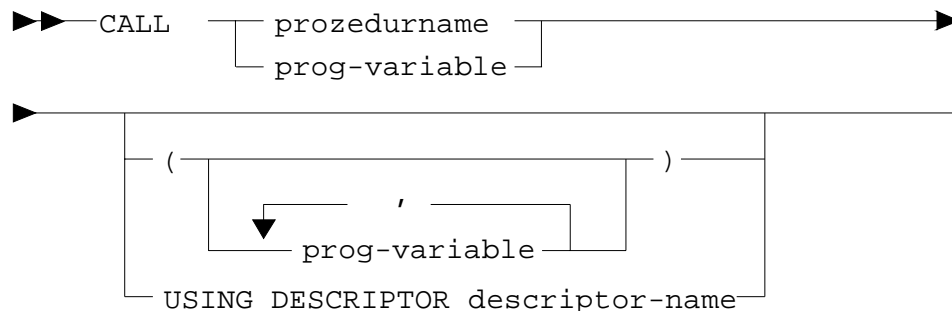
Besteht das Ergebnis der Abfrage aus mehr als einer Zeile, erhalten Sie einen Fehlercode. Die Programm-Variablen können dann Werte enthalten. Es ist aber nicht definiert, zu welcher Zeile diese gehören.

Dieser Befehl kann nicht mit `PREPARE` dynamisch übersetzt werden.

Parameter	
auswahlliste	<p>Die Auswahlliste der gewünschten Spalten (Projektion) kann enthalten:</p> <ul style="list-style-type: none"> * (Stern) bewirkt die Auswahl aller Spalten in der Reihenfolge ihrer Definition Name.* bewirkt die Auswahl aller Spalten in der Reihenfolge ihrer Definition von der Tabelle, deren Name oder Alias dem * vorangestellt wurde. Spaltennamen Die Spalten werden entsprechend ihrer Reihenfolge in der Aufzählung ausgegeben. Der Name muß eindeutig sein, gegebenenfalls muß er durch Tabellename oder Tabellen-Alias qualifiziert werden. Ausdrücke mit Spalten, Konstanten und Funktionen, ausgegeben wird das Ergebnis des Ausdrucks. Konstanten
INTO	<p>leitet eine Liste von Programm-Variablen ein, in die die Spalten der Zeile aus der Ergebnis-Tabelle übertragen werden. Die Zuweisung erfolgt der Reihenfolge nach. Ist die Anzahl der Spalten kleiner als die der angegebenen Variablen, erhalten Sie den Wert "W" in dem Feld SQLWARN3 der SQLCA zurück. Die Datentypen von Spalten und Programm-Variablen müssen kompatibel sein. Um NULL-Werte der Datenbank erkennen zu können, müssen Sie Indikator-Variablen angeben.</p>
FROM	<p>gibt die Tabelle oder Sicht an, die durchsucht werden soll.</p>
WHERE ausw-bed	<p>Es werden nur die Zeilen ausgewählt, die die Kriterien der Auswahlbedingung erfüllen (Selektion).</p>

Parameter	
GROUP BY spalte	GROUP BY verdichtet die Ergebnisse des SELECT-Befehls. Die Spalten, die in dieser Klausel angegeben sind, werden auf Zeilen mit gleichen Werten geprüft und bei Gleichheit zu einer Zeile zusammengefaßt. Die Spalten in der GROUP BY-Angabe müssen auch in der <i>auswahlliste</i> enthalten sein.
HAVING ausw-bed	HAVING erlaubt eine nochmalige Zeilenauswahl nach Verdichtung der Tabelle gemäß der GROUP BY-Angabe. Fehlt die GROUP BY-Angabe, werden alle Zeilen der Tabelle als eine Gruppe angesehen.

5.12 CALL



CALL ruft eine (entfernt) gespeicherte Prozedur (stored procedure) auf. Eine solche Prozedur wird auf dem Server der Datenbank ausgeführt und liefert dem aufrufenden Client Daten zurück. Sie läuft innerhalb derselben Transaktion wie die aufrufende Anwendung ab.

Der Befehl darf in Programmen überall dort benutzt werden, wo ausführbare Anweisungen in der jeweiligen Programmiersprache erlaubt sind.

Er kann nicht mit PREPARE dynamisch übersetzt werden. Allerdings darf der Name der Prozedur über eine Programm-Variable erst zur Laufzeit bestimmt werden.

Parameter	
CALL prozedurname oder prog-variable	gibt den Namen der Prozedur direkt oder als Inhalt der Programm-Variablen an. Die maximale Länge des Prozedurnamens beträgt 254 Bytes. Der Namen muß den Konventionen des Betriebssystems des Datenbank-Servers entsprechen.
prog-variable	Jede Programm-Variable ist ein Parameter des Aufrufs. Indikator-Variablen helfen den Ein- und Ausgabe-Parameter zu identifizieren: -1 beim Aufruf zeigt an, daß keine Eingabe für diese Variable vorgesehen ist. -128 beim Rücksprung zeigt an, daß für diese Variable keine Ausgabe vorgesehen ist. Die Parameter von Prozeduren unter DB2 UDB für C/S müssen auf Client- und Server-Seite vom passenden Datentyp sein. Bei Prozeduren unter DB2 für OS/390 und DB2 für AS/400 sind Typ-Konvertierungen möglich.
USING DESCRIPTOR	verweist auf die SQLDA (SQL dynamic area), die eine Beschreibung der Programm-Variablen enthält. Vor Ausführung des CALL muß der Benutzer die SQLDA mit den benötigten Angaben versorgen.

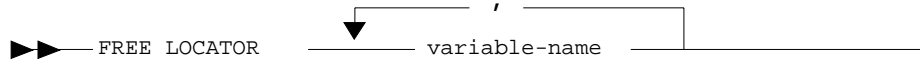
Besondere
Berechtigungen

Die Berechtigungen orientieren sich am Server, auf dem die Prozedur gespeichert ist. Für DB2 UDB für C/S brauchen Sie mindestens eine der folgenden:

- EXECUTE-Berechtigung für das Zugriffspaket (package)
- CONTROL-Berechtigung für das Zugriffspaket (package)
- SYSADM oder DBADM.

Für den Aufruf von Prozeduren auf Servern mit DB2 für OS/390 bzw. z/OS oder DB2 für AS/400 bzw. i-Serie verweise ich Sie auf die zugehörigen Handbücher.

5.13 FREE LOCATOR



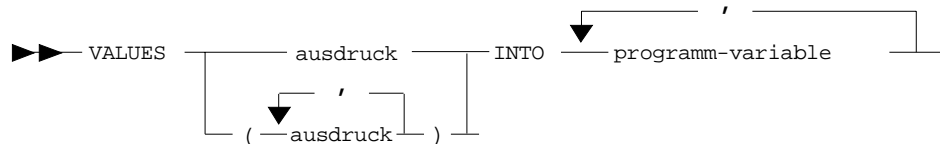
FREE LOCATOR trennt die Verbindung zwischen der Lokator-Variable und ihrem Inhalt.

Der Befehl darf in Programmen überall dort benutzt werden, wo ausführbare Anweisungen in der jeweiligen Programmiersprache erlaubt sind.

Er kann nicht mit PREPARE dynamisch übersetzt werden.

Parameter	
variable-name	identifiziert eine Variable, der ein Lokator (durch FETCH oder SELECT INTO) zugewiesen ist.

5.14 VALUES INTO



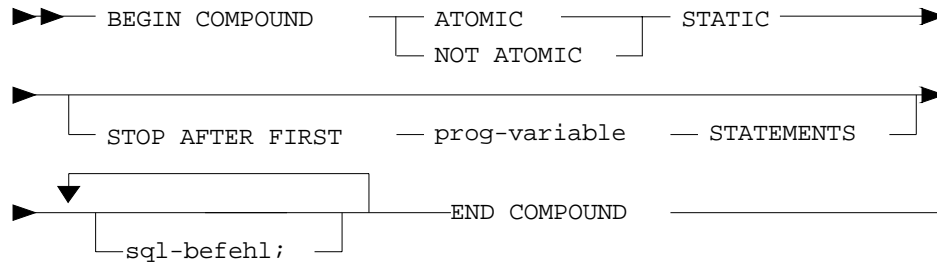
VALUES INTO erzeugt eine Ergebnis-Tabelle mit meistens einer Zeile, die den Programm-Variablen zugewiesen wird.

Der Befehl darf in Programmen überall dort benutzt werden, wo ausführbare Anweisungen in der jeweiligen Programmiersprache erlaubt sind.

Er kann nicht mit PREPARE dynamisch übersetzt werden.

Parameter	
VALUES ausdruck	gibt eine oder mehrere Spalten der Ergebnis-Tabelle an.
INTO programm-variable	gibt eine oder mehrere Programm-Variablen an, denen die Spalten der Ergebnis-Tabelle zugewiesen werden. Werden weniger Programm-Variable angegeben als Spalten, so wird dem Feld SQLWARN3 der SQLCA der Wert "W" zugewiesen.

5.15 Compound SQL



Compound SQL faßt einen oder mehrere SQL-Befehle (Unterbefehle) in einem ausführbaren Block zusammen. Es sind dazwischen keine Befehle in der Gastsprache erlaubt. Compound SQL-Befehle können nicht geschachtelt werden.

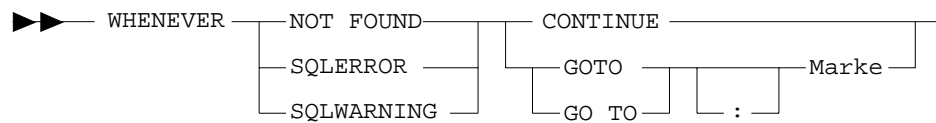
Es wird nur eine SQLCA benutzt. Ihre Werte geben im allgemeinen die Werte nach der Ausführung des letzten Unterbefehls wieder. Die SQLWARN-Felder enthalten allerdings die Akkumulation über alle Unterbefehle. SQLSTATE '02000' (No data found) erhält aber Vorrang vor anderen Warnungen und kann auch von weiter vorne liegenden Unterbefehle gesetzt worden sein.

Der Befehl darf nur in Programmen benutzt werden. Der gesamte Compound SQL-Befehl ist eine ausführbare Anweisung, die nicht dynamisch übersetzt werden kann.

Parameter	
ATOMIC	gibt vor, daß im Falle eines Fehlers eines enthaltenen SQL-Befehls alle Veränderungen, auch die erfolgreich durchgeführter Unterbefehle, des Compound SQL zurückgesetzt werden (nicht erlaubt unter DDCS).
NOT ATOMIC	gibt vor, daß im Falle eines Fehlers eines enthaltenen SQL-Befehls keine Veränderungen anderer Unterbefehle zurückgesetzt werden.
STATIC	gibt an, daß alle Programm-Variablen als Eingabe-Parameter ihren Übergabewert für alle Unterbefehle behalten – selbst wenn sie zwischenzeitlich in Unterbefehlen überschrieben wurden. Dynamisches Verhalten (not static) wird zur Zeit nicht unterstützt, das heißt zwischen Unterbefehlen können keine Abhängigkeiten bestehen, und die Verarbeitungsreihenfolge kann nicht als sequentiell angesehen werden.
STOP AFTER FIRST prog-variable STATEMENTS	gibt vor, daß nur die in <i>prog-variable</i> enthaltene Anzahl von Befehlen ausgeführt wird.
sql-befehl	alle SQL-Befehle sind als Unterbefehle erlaubt außer : CALL, FETCH, CLOSE, OPEN, CONNECT, PREPARE, Compound SQL, RELEASE, DESCRIBE, ROLLBACK, SAVEPOINT, RELEASE SAVEPOINT, DISCONNECT, SET CONNECTION, EXECUTE IMMEDIATE Ein COMMIT darf nur als letzter Unterbefehl enthalten sein. Als solcher wird er auch nach einer STOP AFTER FIRST-Klausel ausgeführt, wenn er außerhalb der vorgegebenen Spanne liegt. SAVEPOINT-Befehle dürfen nicht enthalten sein.

Berechtigungen Für den Compound SQL-Befehl selber brauchen Sie keine Berechtigungen. Sie benötigen aber die entsprechenden Berechtigungen für die SQL-Befehle, die darin enthalten sind.

5.16 WHENEVER



WHENEVER gibt die Aktion vor, die beim Auftreten von Ausnahmezuständen (exceptions conditions) durchgeführt werden soll.

Diese Anweisung ist kein ausführbarer Befehl, sondern eine Anweisung an den Precompiler, der die ausführbaren Befehle zur Durchführung der Aktion generiert. Von der Anweisung betroffen sind daher alle ausführbaren SQL-Befehle in der Reihenfolge ihres Erscheinens im Quellcode nach WHENEVER, nicht aber in der Reihenfolge ihrer Ausführung.

Ein folgendes WHENEVER ändert ein vorheriges WHENEVER zur gleichen Ausnahmebedingung ab seinem Platz im Quellcode.

Geben Sie kein WHENEVER an, so gilt für alle Ausnahmezustände CONTINUE..

Parameter	
NOT FOUND	bedeutet SQLCODE = +100 oder SQLSTATE = '02000'.
SQLERROR	bedeutet SQLCODE < 0.
SQLWARNING	bedeutet SQLCODE > 0 aber <> +100 oder SQLWARN0 = 'W'.
CONTINUE	bringt den nächsten Befehl im Quellcode zur Ausführung.
GOTO oder GO TO	springt die angegebene Marke an.

6. Dynamic SQL-Befehle

Die folgenden SQL-Befehle dienen der dynamischen Übersetzung und Ausführung von SQL-Befehlen aus Programmen heraus.

- DESCRIBE (siehe Kapitel 6.1, Seite 6-1)
- PREPARE (siehe Kapitel 6.2, Seite 6-3)
- EXECUTE (siehe Kapitel 6.3, Seite 6-5)
- EXECUTE IMMEDIATE (siehe Kapitel 6.4, Seite 6-6)

6.1 DESCRIBE

```
▶▶ — DESCRIBE — befehlsname — INTO — deskriptor —————|
```

DESCRIBE stellt Informationen über den übersetzten SQL-Befehl *befehlsname* in den Bereich *deskriptor*.

Dieser Befehl kann nicht mit PREPARE dynamisch übersetzt werden.

Vor dem Aufruf von DESCRIBE muß folgende Variable gesetzt werden:

SQLDA	
SQLN	Anzahl der Variablen, die der Bereich SQLVAR enthält. SQLN muß größer oder gleich 0 sein. Ist SQLN gleich 0, wird in SQLD die Anzahl der Spalten des Ergebnisses von <i>befehlsname</i> zurückgegeben.

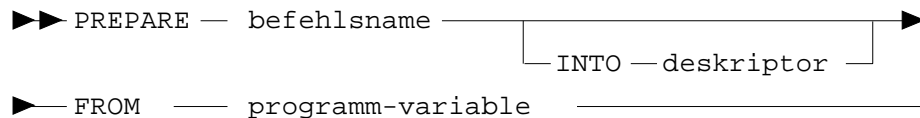
DB2 versorgt folgende Variablen bei Befehlsausführung:

SQLDA	
SQLDAID	8 Bytes langes Feld, beginnend mit "SQLDA " Das 7. Byte enthält "2", wenn 2 SQLVAR-Einträge für ein Spalte benutzt werden. Das 8. Byte enthält immer ein Leerzeichen.
SQLDABC	Länge der SQLDA
SQLD	Anzahl der Spalten bei einem übersetzten SELECT-Befehl, sonst 0. Ist SQLD 0 oder größer als SQLN, enthält der Bereich SQLVAR keine Daten.
SQLVAR	Für $0 < \text{SQLD} \leq \text{SQLN}$ enthält SQLVAR die Beschreibungen der Spalten der Ergebnis-Tabelle in der Reihenfolge dieser Tabelle. Die Beschreibung besteht jeweils aus SQLTYPE, SQLLEN und SQLNAME und ggf. aus SQLLONGLEN und SQLDATATYPE_NAME. SQLTYPE Codierter Datentyp für die Spalte, Wertebereich 384 bis 501, wobei gerade Zahlen angeben, daß NULL erlaubt ist, ungerade, daß NULL nicht erlaubt ist. Die Werte 804 bis 813 stehen für LOB-Dateireferenzen und 960 bis 969 für LOB-Lokatoren. SQLLEN Länge in Abhängigkeit vom Datentyp der Spalte, 0 für LOB-Spalten SQLNAME Unqualifizierter Spaltenname SQLLONGLEN Länge der LOB-Spalte SQLDATATYPE_NAME voll qualifizierter Name des Datentyps

Parameter	
befehlsname	verweist auf einen Befehl, der unter diesem Namen zuvor übersetzt wurde.
INTO deskriptor	verweist auf die SQLDA, die unter <i>deskriptor</i> angelegt wurde.

Die Strukturdefinitionen der SQLDA für Cobol und für C finden Sie in Kapitel 8, *Anhang*.

6.2 PREPARE



PREPARE übersetzt in einem Anwendungsprogramm zu dessen Ausführungszeit einen SQL-Befehl, der als Zeichenkette vorliegt, und speichert ihn. Enthält der SQL-Befehl als Zeichenkette Fehler, wird er nicht in übersetzter Form abgelegt; die SQLCA enthält dann den Fehlercode.

Übersetzte Befehle können mit EXECUTE beliebig oft ausgeführt werden, wenn sie keine SELECT-Befehle sind. SELECT-Befehle müssen als Cursor deklariert und mit OPEN ausgeführt werden. Wenn Befehle nur einmal ausgeführt werden sollen, ist es effizienter, EXECUTE IMMEDIATE zu benutzen.

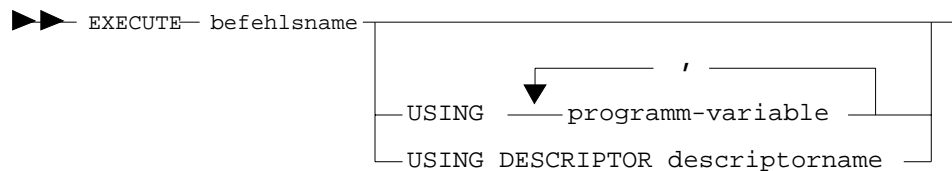
Übersetzte Befehle werden gecached und bleiben über Transaktionsgrenzen erhalten.

PREPARE selbst kann nicht mit PREPARE dynamisch übersetzt werden.

Parameter	
befehlsname	benennt den übersetzten Befehl. Wenn der Name bereits einen übersetzten Befehl identifiziert, wird dieser zerstört. Der Name darf keinen übersetzten SELECT-Befehl identifizieren, dessen Cursor geöffnet ist.
INTO deskriptor	verweist auf die SQLDA, die unter <i>deskriptor</i> angelegt wurde. Der DESCRIBE-Befehl kann stattdessen verwendet werden.
FROM programm- variable	<p>verweist auf zuvor als solche deklarierte Programm-Variable, die den SQL-Befehl als Zeichenkette enthält. Die Variable muß einen der folgenden SQL-Befehle enthalten:</p> <p>ALTER TABLE, COMMENT ON, COMMIT, CREATE, DECLARE GLOBAL TEMPORARY TABLE, DELETE, DROP, EXPLAIN, FLUSH EVENT MONITOR, GRANT, INSERT, LOCK TABLE, REFRESH TABLE, RELEASE SAVEPOINT, RENAME ... ; REVOKE, ROLLBACK, SAVEPOINT, SELECT, SET INTEGRITY, SET CURRENT ... , SET EVENT MONITOR STATE, SET PASSTHRU, SET PATH, SET SCHEMA, SET SERVER OPTION, UPDATE.</p> <p>Die Zeichenkette darf nicht enthalten:</p> <ul style="list-style-type: none"> • SELECT INTO • EXEC SQL und Befehlsbegrenzer wie END-EXEC oder ; • Referenzen auf Programm-Variablen • Kommentare. <p>Statt der Programm-Variablen kann die Zeichenkette Parameter-Platzhalter "?" (auch mit CAST-Angabe) enthalten. Diese werden durch die Werte von Programm-Variablen ersetzt, wenn der Befehl ausgeführt wird.</p>

Besondere Berechtigungen Bei Ausführung des Befehls werden für DML-Befehle die Berechtigungen geprüft. Sie müssen über die notwendigen Berechtigungen für die Ausführung des zu übersetzenden Befehls verfügen. Bei anderen Befehlen (DDL, DCL) werden die Berechtigungen zu deren Ausführung erst zur Ausführungszeit geprüft. Dann benötigen Sie keine Berechtigungen, um PREPARE aufzurufen.

6.3 EXECUTE



EXECUTE führt einen zuvor übersetzten SQL-Befehl aus.

Dieser Befehl kann nicht mit PREPARE dynamisch übersetzt werden.

Parameter	
befehlsname	verweist auf den SQL-Befehl, der unter diesem Namen mit PREPARE zuvor in derselben Transaktion übersetzt wurde. Der Befehl darf kein SELECT sein.
USING	leitet eine Liste von Programm-Variablen ein, die die Parameter-Platzhalter ersetzen. Die Ersetzung erfolgt der Reihenfolge nach.
USING DESCRIPTOR	verweist auf die SQLDA (SQL dynamic area), die eine Beschreibung der Programm-Variablen enthält. Vor Ausführung des OPEN muß der Benutzer die SQLDA mit den benötigten Angaben versorgen.

Besondere Berechtigungen Für Befehle, bei denen zur Ausführungszeit die Berechtigung geprüft wird, müssen Sie über die Berechtigungen verfügen, die zur Ausführung des auszuführenden Befehls notwendig sind. Für Befehle, bei denen die Berechtigung bereits bei der Übersetzung geprüft wurde, benötigen Sie keine Berechtigungen, um EXECUTE aufzurufen.

6.4 EXECUTE IMMEDIATE

▶▶ EXECUTE IMMEDIATE — programm-variable —|

EXECUTE IMMEDIATE übersetzt in einem Anwendungsprogramm zu dessen Ausführungszeit einen SQL-Befehl, der als Zeichenkette vorliegt, führt ihn aus. Der übersetzte Befehl bleibt solange im DB2-Cache, bis ein erneuter EXECUTE IMMEDIATE mit einem anderen SQL-Befehl ihn überschreibt. Enthält der SQL-Befehl als Zeichenkette Fehler, wird er nicht ausgeführt; die SQLCA enthält dann den Fehlercode.

Führen Sie denselben Befehl mehrmals aus, ist es effizienter, ihn mit PREPARE einmal zu übersetzen und dann mit EXECUTE mehrmals auszuführen.

Parameter	
programm-variable	<p>verweist auf zuvor als solche deklarierte Programm-Variable, die den SQL-Befehl als Zeichenkette enthält. Die Variable muß einen der folgenden SQL-Befehle enthalten:</p> <p>ALTER TABLE, COMMENT ON, COMMIT, CREATE, DECLARE GLOBAL TEMPORARY TABLE, DELETE, DROP, EXPLAIN, FLUSH EVENT MONITOR, GRANT, INSERT, LOCK TABLE, REFRESH TABLE, RELEASE SAVEPOINT, RENAME ...; REVOKE, ROLLBACK, SAVEPOINT, SELECT, SET INTEGRITY, SET CURRENT ... , SET EVENT MONITOR STATE, SET PASSTHRU, SET PATH, SET SCHEMA, SET SERVER OPTION, UPDATE.</p> <p>Die Zeichenkette darf nicht enthalten:</p> <ul style="list-style-type: none"> • SELECT INTO • EXEC SQL und Befehlsbegrenzer wie END-EXEC oder ; • Parameter-Platzhalter oder Referenzen auf Programm-Variablen • Kommentare.

Dieser Befehl kann nicht mit PREPARE dynamisch übersetzt werden.

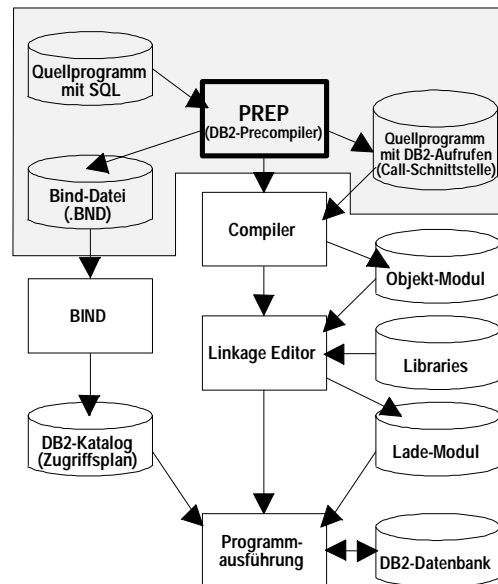
Berechtigungen Zur Ausführung dieses Befehls müssen Sie über die Berechtigungen verfügen, die Sie zur Ausführung des angegebenen SQL-Befehls benötigen.

7. Kommandos zum Übersetzen und Binden

7.1 PREP (PRECOMPILE PROGRAM)

Das DB2-Kommando PREP übersetzt die Quelldatei eines Anwendungsprogramms in eine erweiterte Quelldatei und erzeugt eine .BND-Datei, die für den Binde-Lauf benötigt wird.

Bild 7-1:
Quellprogramm
mit *PREP*
vorübersetzen



Beim Binden werden die SQL-Befehle, die in der .BND-Datei enthalten sind, in ein Zugriffspaket (package) umgesetzt und in den Tabellen SYSIBM.SYSPLAN und SYSIBM.SYSSECTION im DB2-Katalog gespeichert. Das Binden kann von PREP übernommen oder abgetrennt und als eigenständiger Schritt mit dem BIND-Kommando durchgeführt werden.

Ein Zugriffspaket unterteilt sich je SQL-Befehl in Abschnitte (sections). Es kann maximal 400 Abschnitte besitzen.

Sie müssen mit der Datenbank verbunden sein, für die die Anwendung übersetzt und gebunden werden soll.

Der BIND wird abgebrochen, wenn ein schwerer Fehler (fatal error) oder mehr als 100 Fehler aufgetreten sind. Die Fehlermeldungen werden in der Reihenfolge ihres Auftretens ausgegeben.

Beim Aufruf müssen Sie mindestens den Namen des zu übersetzenden Programms und die zugehörige Datenbank angeben.

Das Programm muß in einer unterstützten Programmiersprache geschrieben sein. Unterstützt werden zur Zeit C, C++, COBOL und FORTRAN¹.

Das Ergebnis des PREP-Laufs sind eine erweiterte Quelldatei als Eingabe für den Compiler und eine .BND-Datei mit den SQL-Befehlen und zusätzlichen Angaben für DB2.

PREP entnimmt der Erweiterung (extension) des Dateinamens die Programmiersprache.

gültige Dateinamen-Erweiterungen	
.sqc	C
.sqx	C++ unter Windows oder OS/2
.sqC	C++ unter UNIX
.sqb	COBOL
.sqf	FORTRAN

¹ JAVA-Programme mit SQLJ-Schnittstelle werden von einem speziellen Dienstprogramm übersetzt.

Wenn Sie keine weiteren Angaben machen, unterstellt PREP folgende Standards:

- Für die ausgegebene erweiterte Quelldatei:

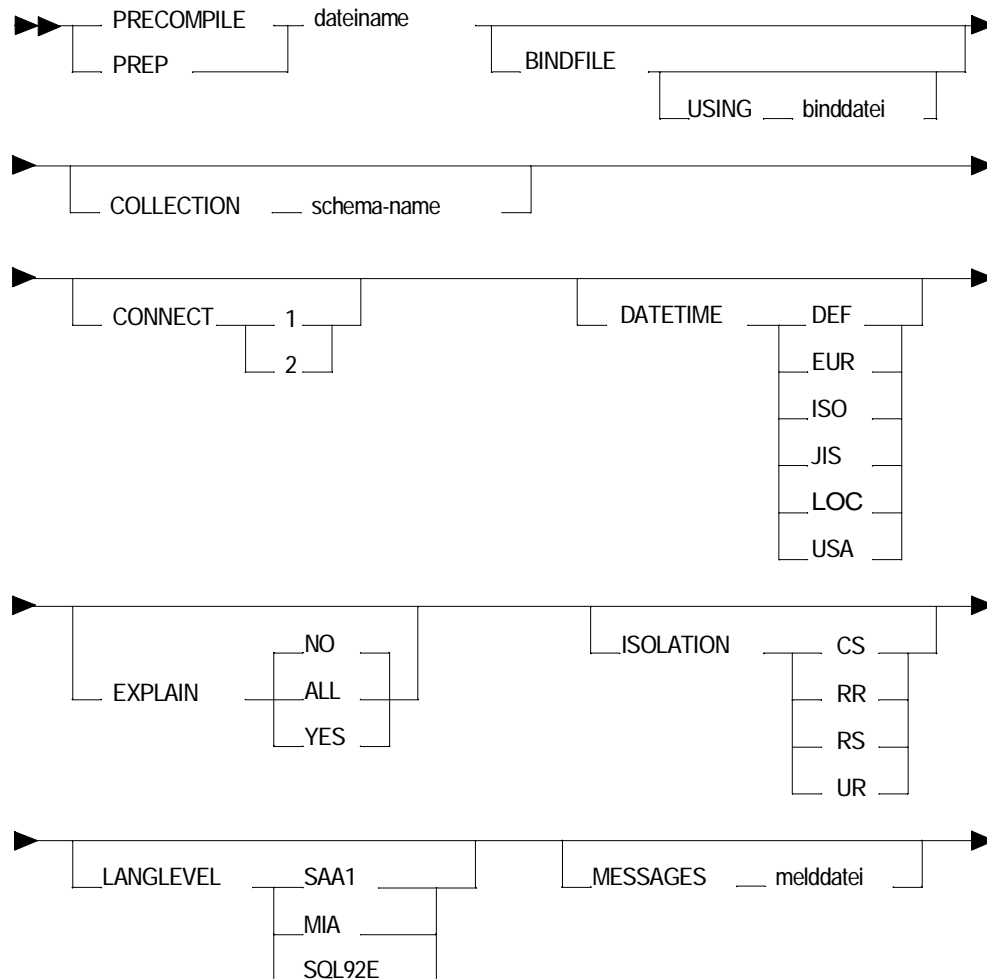
Dateinamen-Standards	
Programm-Name.c	C-Programme
Programm-Name.cxx	C++-Programme unter Windows oder OS/2
Programm-Name.C	C++-Programme unter UNIX
Programm-Name.cbl	COBOL-Programme
Programm-Name.for	FORTRAN-Programme unter Windows oder OS/2
Programm-Name.f	FORTRAN-Programme unter UNIX

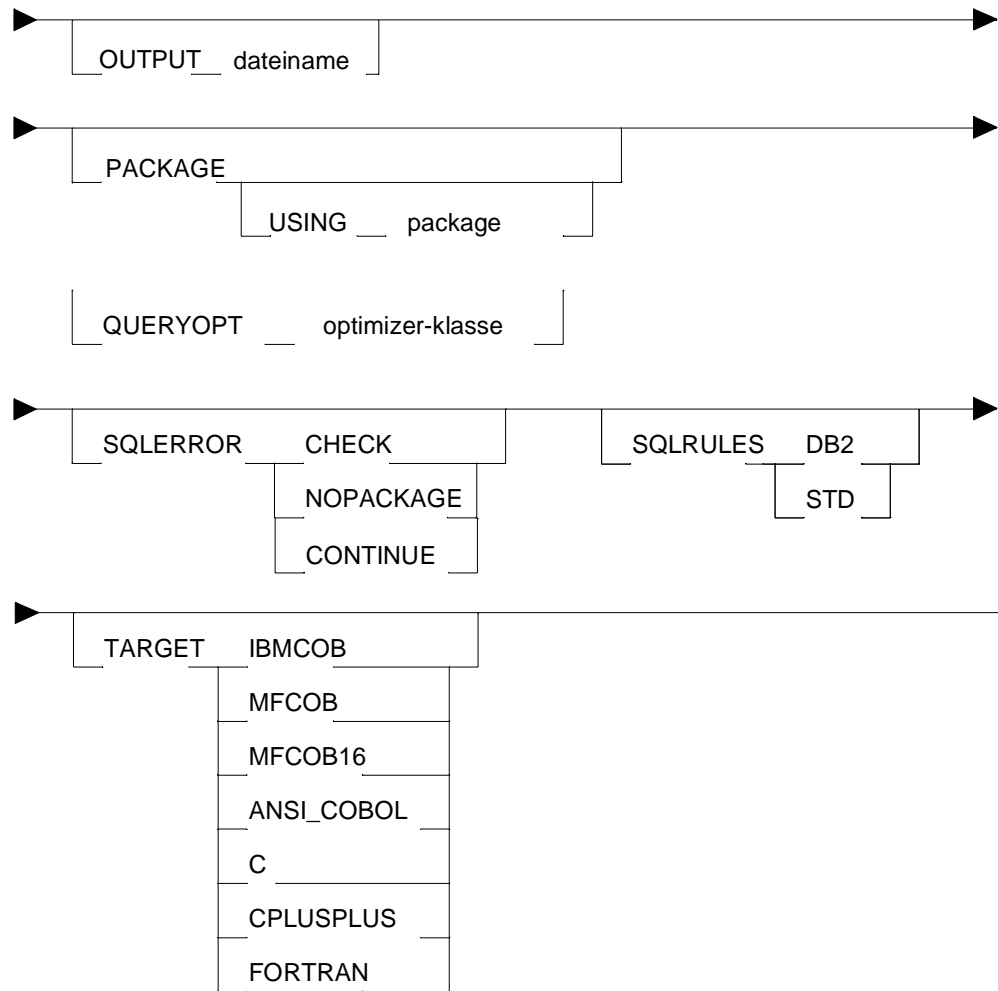
- Für das Zugriffspaket (package) den Namen des Programms (ohne Erweiterung (extension))
- Für die Binde-Datei² *Programm-Name.BND*. Eine existierende Datei wird überschrieben.

PREP speichert die erweiterte Quelldatei und die .BND-Datei in demselben Verzeichnis wie das Programm, wenn Sie keinen anderen Pfad angeben.

Hier folgt nun die Syntax mit den wichtigsten Parametern:

² Bei DB2 für OS/390 heißt diese Datei DBRM (DataBase Resource Module)





Für die weiteren Parameter und für Übersetzungen auf DRDA-Servern, für die teilweise andere Parameter gelten, sollten Sie bei Bedarf im DB2 Command Reference Handbuch nachschlagen.

Parameter

BINDFILE

BINDFILE erzeugt eine Binde-Datei. Ein Zugriffspaket (package) wird nur erzeugt, wenn Sie die PACKAGE-Option gleichzeitig angeben. SQLCODES wegen fehlender Objekte oder Berechtigung werden nur als Warnungen behandelt. Dadurch können Sie auch dann eine Binde-Datei erzeugen, wenn die Datenbank nicht alle Objekte besitzt. Mit dieser Datei können Sie später dann ein Zugriffspaket erzeugen (BIND).

USING binddatei

Mit USING geben Sie den Namen der Binde-Datei vor. Geben Sie keinen Namen an, wird eine Datei mit dem Namen *programm.BND* erzeugt. Machen Sie keine vollständige Pfadangabe, benutzt PREP das aktuelle Verzeichnis.

COLLECTION schema-name

Mit COLLECTION geben Sie einen 8-stelligen Schema-Namen vor. Standardmäßig wird die Berechtigungsidentifikation des Benutzers benutzt, der das Zugriffspaket (package) ausführt.

CONNECT

Parameter	
1	ein CONNECT-Befehl wird als Typ 1 behandelt. Mit einem CONNECT Typ 1 kann ein Programm je Transaktion nur mit einer Datenbank arbeiten. Die aktuelle Transaktion muß erst abgeschlossen werden, bevor eine Verbindung zu einer anderen Datenbank aufgebaut werden kann.
2	ein CONNECT-Befehl wird als Typ 2 behandelt. Mit einem CONNECT Typ 2 kann ein Programm in einer Transaktion mit mehreren Datenbanken arbeiten, d.h. die aktuelle Transaktion muß nicht abgeschlossen werden, wenn eine Verbindung zu einer anderen Datenbank aufgebaut wird.

DATETIME

DATETIME legt das Format für Datum und Uhrzeit fest für Felder der entsprechenden Datentypen, die Zeichenketten-Darstellungen zugeordnet werden. Wenn Sie kein Format angeben, benutzt BIND DEF.

gültige Angaben	
DEF	Format, das zum Ländercode der Datenbank gehört.
USA	USA-Format nach IBM Standard Datum: mm/tt/jjjj Uhrzeit: hh:mm AM oder PM
EUR	Europa-Format nach IBM-Standard Datum: tt.mm.jjjj Uhrzeit: hh.mm.ss
ISO	International genormtes Format Datum: jjjj-mm-tt Uhrzeit: hh.mm.ss
JIS	Japanischer Industriestandard Datum: jjjj-mm-tt Uhrzeit: hh:mm:ss
LOC	Spezielles Format in Abhängigkeit vom Ländercode der Datenbank

EXPLAIN

Mit EXPLAIN können Sie Informationen über die Zugriffswege für jeden SQL-Befehl in die EXPLAIN-Tabellen abspeichern lassen.

Parameter	
NO	keine Informationen werden gespeichert.
YES	Informationen werden gespeichert.
ALL	Informationen für jeden statischen SQL-Befehl werden gespeichert. Zusätzlich werden zur Laufzeit Informationen über dynamische SQL-Befehle gesammelt, selbst wenn die Umgebungs-Variable CURRENT EXPLAIN SNAPSHOT auf NO gesetzt ist.

ISOLATION

Mit ISOLATION bestimmen Sie die Ebene der Benutzertrennung.

gültige Angaben	
CS	Cursor Stability
RR	Repeatable Read
RS	Read Stability
UR	Uncommitted Read

Der Standard für die Benutzertrennung ist die Angabe beim Aufruf des Precompilers. CS ist Standard, wenn auch dort nichts explizit vorgegeben wurde.

LANGLEVEL

LANGLEVEL gibt die Ebene der Kompatibilität vor:

Parameter	
SAA1	erfordert Kompatibilität mit SAA Level 1 Database CPI. Dies bedeutet u.a., daß für änderbare Cursor die FOR UPDATE OF-Angabe für alle veränderten Spalten erwartet wird, daß für den SQLCODE die SQLCA deklariert werden muß und daß abgeschnittene Zeichenketten in C nicht mit Binär-0 (\0) beendet werden.
MIA	verlangt Kompatibilität mit MIA (multi-vendor integrated architecture). Dies bedeutet u.a., daß die FOR UPDATE OF-Angabe in Cursor-Deklarationen optional ist, daß für den SQLCODE die SQLCA deklariert werden muß und daß Zeichenketten in C immer, auch abgeschnittene, mit Binär-0 (\0) beendet werden.
SQL92E	verlangt SQL92-Kompatibilität. Dies bedeutet u.a., daß die FOR UPDATE OF-Angabe in Cursor-Deklarationen optional ist, daß SQLCODE oder SQLSTATE als Variablen (ohne SQLCA) deklariert werden können und daß Zeichenketten in C immer, auch abgeschnittene, mit Binär-0 (\0) beendet werden.

MESSAGES melddatei

Mit MESSAGES geben Sie an, wohin die Fehler-, Warn- und Beendigungsmeldungen ausgegeben werden sollen. Die Meldedatei wird angelegt unabhängig davon, ob der BIND erfolgreich ist oder nicht. Eine schon bestehende Datei wird überschrieben. Ohne explizite Angabe erfolgt die Ausgabe auf die Standard-Ausgabe des Betriebssystems.

OUTPUT dateiname

Mit OUTPUT überschreiben Sie den Standardnamen des Compilers für das vorübersetzte Quellprogramm. Die Angabe kann einen Pfad enthalten.

PACKAGE

Mit PACKAGE erstellen Sie ein Zugriffspaket.

USING package

Mit USING geben Sie den Namen vor, unter dem das Zugriffspaket abgelegt wird. Geben Sie keinen Namen vor, übernimmt PREP den Namen der Quelldatei (maximal 8-stellig).

QUERYOPT

Mit QUERYOPT geben Sie eine gewünschte Optimizer-Klasse vor. Der Standardwert ist 5.

SQLERROR

Mit SQLERROR bestimmen Sie, ob bei einem Fehler ein Zugriffspaket (package) oder eine Binde-Datei erstellt werden sollen.

Parameter	
CHECK	Das Zielsystem führt alle Syntax- und Semantik-Prüfungen der SQL-Befehle durch. Ein Zugriffspaket wird nicht erstellt.
NOPACKAGE	kein Zugriffspaket oder Binde-Datei im Fehlerfalle
CONTINUE	erzeugt ein Zugriffspaket (package) auch bei Fehlern

SQLRULES

SQLRULES gibt an, ob ein CONNECT Type 2 nach den DB2-Regeln oder entsprechend dem SQL-ISO/ANS-Standard von 1992 durchgeführt wird.

Parameter	
DB2	erlaubt es, mit dem SQL CONNECT von der aktuellen Verbindung zu einer aufgebauten, ruhenden Verbindung umzuschalten.
STD	erlaubt nur, eine neue Verbindung aufzubauen. Zum Umschalten auf eine ruhende Verbindung muß der Befehl SET CONNECTION benutzt werden.

TARGET

Mit TARGET geben Sie vor, für welchen Compiler PREP Code generieren soll.

Parameter	
IBMCOB	IBM COBOL unter AIX und OS/2
MFCOB	Micro Focus COBOL (32-Bit)
MFCOB16	Micro Focus COBOL Version 3 unter OS/2 (noch 16 BIT)
ANSI_COBOL	COBOL nach Standard ANS X3.23-1985
C	C-Compiler, der von DB2 unterstützt wird
CPLUSPLUS	C++-Compiler, der von DB2 unterstützt wird
FORTRAN	FORTRAN-Compiler, der von DB2 unterstützt wird

Berechtigungen Um einen PREP-Lauf durchführen zu können, benötigen Sie mindestens eine der folgenden Berechtigungen:

- SYSADM oder DBADM
- BINDADD
- BIND.

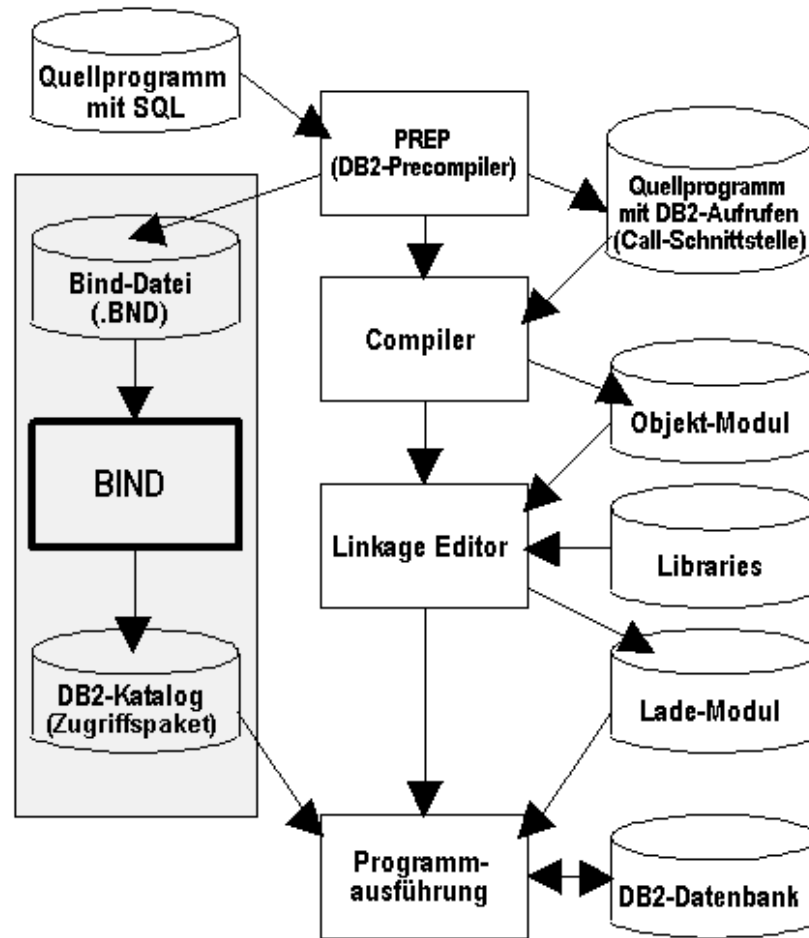
Sie benötigen außerdem alle Berechtigungen, die zur Übersetzung jedes statischen SQL-Befehls nötig sind.

7.2 BIND

Das DB2-Kommando BIND setzt die Ausgabe des Precompilers mit den SQL-Befehlen in ein Zugriffspaket (package) um, das im DB2-Katalog in den Tabellen SYSIBM.SYSPPLAN und SYSIBM.SYSSECTION gespeichert wird.

Dabei werden die SQL-Befehle vom DB2-Optimizer übersetzt und die Zugriffspfade auf die angesprochenen DB2-Objekte festgelegt.

Bild 7-2:
Zugriffspaket mit
BIND erstellen



Ein Zugriffspaket unterteilt sich je SQL-Befehl in Abschnitte (sections). Es kann maximal 400 Abschnitte besitzen.

BIND bricht die Verarbeitung ab, wenn ein schwerer Fehler (fatal error) oder mehr als 100 Fehler aufgetreten sind. Die Fehlermeldungen werden in der Reihenfolge ihres Auftretens in die Meldungsdatei geschrieben.

Bricht BIND bei einem schweren Fehler die Verarbeitung ab, versucht es alle Dateien zu schließen, und das Zugriffspaket in der Datenbank zu löschen.

Der Precompiler kann ebenfalls den Binde-Lauf durchführen, wenn er eine Datei übersetzt.

Der Standard-Name für das Zugriffspaket steht in der .BND-Datei und beruht auf dem Namen der Quelldatei, aus der die .BND-Datei erzeugt wurde. Die Namen der .BND-Datei und des Zugriffspakets können Sie beim Übersetzen überschreiben.

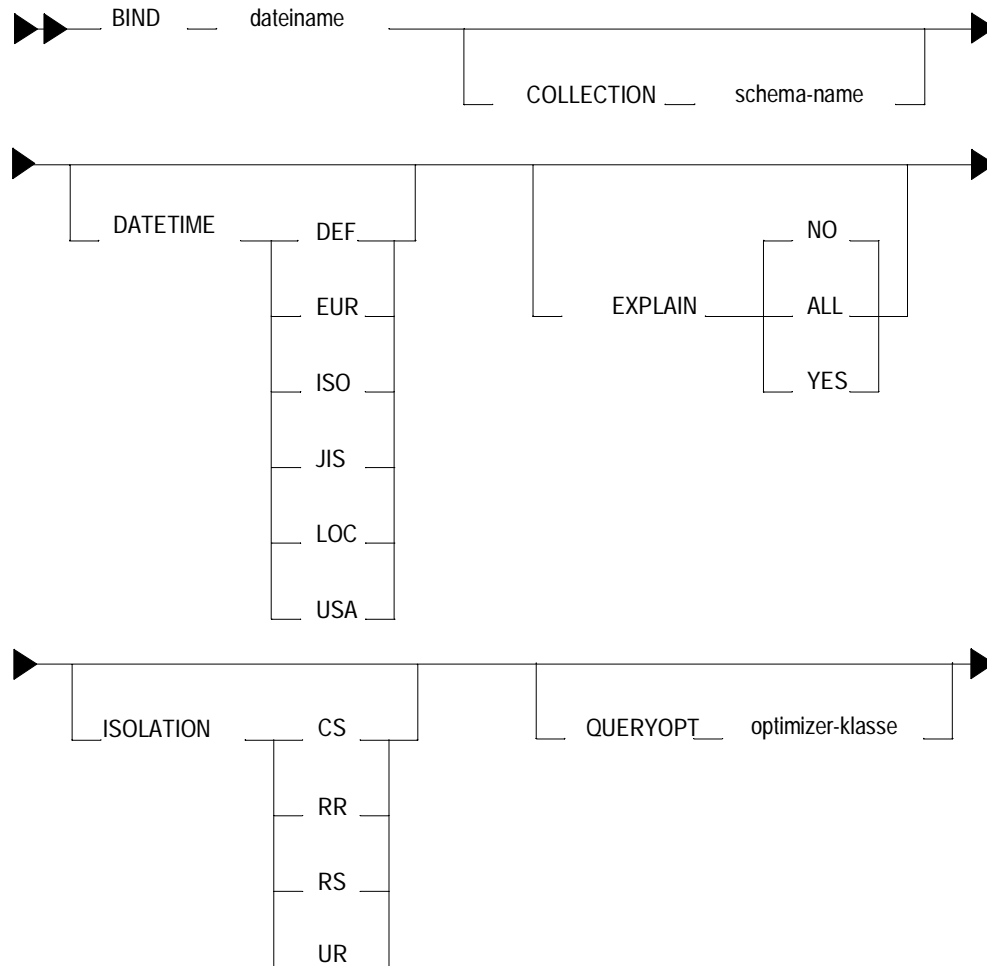
Statt des Namens der zu bindenden Datei können Sie auch den Namen einer Listendatei angeben, in der Sie mehrere zu bindende Dateien benennen können. Der Name der Listendatei muß mit einem "@" beginnen. Die Dateinamen der Aufzählung müssen mit einem Plus-Zeichen (+) miteinander verbunden sein, zum Beispiel

test_r.bnd+test_i.bnd.

Sie müssen mit der Datenbank verbunden sein, für die die Anwendung gebunden werden soll. Bei implizitem Verbindungsaufbau wird eine Anmeldung an der als Standard definierten Datenbank durchgeführt.

BIND arbeitet unter der Transaktion, die Sie bereits gestartet haben. Es beendet diese durch COMMIT oder ROLLBACK.

Hier folgt nun die Syntax von BIND mit nur den wichtigsten Parametern:



Für die weiteren Parameter und für BINDs auf DRDA-Servern, für die teilweise andere Parameter gelten, sollten Sie bei Bedarf im DB2 Command Reference Handbuch nachschlagen.

Wichtige
Parameter

COLLECTION schema-name

Mit COLLECTION geben Sie einen 8-stelligen Schema-Namen vor. Standardmäßig wird die Berechtigungsidentifikation des Benutzers benutzt, der das Zugriffspaket (package) ausführt.

DATETIME

DATETIME legt das Format für Datum und Uhrzeit fest für Felder der entsprechenden Datentypen, die Zeichenketten-Darstellungen zugeordnet werden. Wenn Sie kein Format angeben, benutzt BIND DEF.

gültige Angaben	
DEF	Format, das zum Ländercode der Datenbank gehört.
USA	USA-Format nach IBM Standard Datum: mm/tt/jjjj Uhrzeit: hh:mm AM oder PM
EUR	Europa-Format nach IBM-Standard Datum: tt.mm.jjjj Uhrzeit: hh.mm.ss
ISO	International genormtes Format Datum: jjjj-mm-tt Uhrzeit: hh.mm.ss
JIS	Japanischer Industriestandard Datum: jjjj-mm-tt Uhrzeit: hh:mm:ss
LOC	Spezielles Format in Abhängigkeit vom Ländercode der Datenbank

EXPLAIN

Mit EXPLAIN können Sie Informationen über die Zugriffswege für jeden SQL-Befehl in die EXPLAIN-Tabellen abspeichern lassen.

gültige Angaben	
NO	keine Informationen werden gespeichert.
YES	Informationen werden gespeichert.
ALL	Informationen für jeden statischen SQL-Befehl werden gespeichert. Zusätzlich werden zur Laufzeit Informationen über dynamische SQL-Befehle gesammelt, selbst wenn die Umgebungs-Variable CURRENT EXPLAIN SNAPSHOT auf NO gesetzt ist.

ISOLATION

Mit ISOLATION bestimmen Sie die Ebene der Benutzertrennung.

gültige Angaben	
CS	Cursor Stability
RR	Repeatable Read
RS	Read Stability
UR	Uncommitted Read

Der Standard für die Benutzertrennung ist die Angabe beim Aufruf des Precompilers. CS ist Standard, wenn auch dort nichts explizit vorgegeben wurde.

QUERYOPT

Mit QUERYOPT geben Sie eine gewünschte Optimizer-Klasse vor. Der Standardwert ist 5.

- Berechtigungen Um einen Binde-Lauf durchführen zu können, benötigen Sie mindestens eine der folgenden Berechtigungen:
- SYSADM oder DBADM
 - BINDADD
 - BIND.

8. Anhang

8.1 SQLDA als COBOL-COPY

```
*****
*      * Source File Name = SQLDA.CBL
*      * (C) Copyright IBM Corp. 1987, 1995
* All Rights Reserved
* Licensed Material - Program Property of IBM
* US Government Users Restricted Rights - Use, duplication or
* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
*      * Function = Copy File defining:
*          SQLDA
*****

*   SQL Descriptor Area - Variable descriptor

*                               SQL Descriptor Area - SQLDA
01 SQLDA SYNC.
   05 SQLDAID PIC X(8) VALUE "SQLDA ".
*                               Eye catcher = 'SQLDA  '
   05 SQLDABC PIC S9(9) COMP-5.
*                               SQLDA size in bytes = 16+44*SQLN
   05 SQLN PIC S9(4) COMP-5.
*                               Number of SQLVAR elements
```

```

05 SQLD PIC S9(4) COMP-5.
*           # of used SQLVAR elements
05 SQLVAR-ENTRIES OCCURS 0 TO 1489 TIMES
   DEPENDING ON SQLN OF SQLDA.
10 SQLVAR.
   15 SQLTYPE PIC S9(4) COMP-5.
*           Variable data type
   15 SQLLEN PIC S9(4) COMP-5.
*           Variable data length
   15 SQLDATA USAGE POINTER.
*           Pointer to variable data value
   15 SQLLIND USAGE POINTER.
*           Pointer to Null indicator
   15 SQLNAME.
*           Variable Name
   20 SQLNAMEL PIC S9(4) COMP-5.
*           Name length varies from 1 to 30
   20 SQLNAMEC PIC X(30).
*           Variable or Column name
10 SQLVAR2 REDEFINES SQLVAR.
   15 SQLVAR2-RESERVED-1 PIC S9(9) COMP-5.
*           Reserved for future use
   15 SQLLONGLEN REDEFINES SQLVAR2-RESERVED-1
   PIC S9(9) COMP-5.
*           LOB variable data length
   15 SQLVAR2-RESERVED-2 PIC S9(9) COMP-5.
*           Reserved for future use
   15 SQLDATALEN USAGE POINTER.
*           Pointer to LOB variable data length
   15 SQLDATATYPE-NAME.
*           Variable Name
   20 SQLDATATYPE-NAMEL PIC S9(4) COMP-5.
*           Name length varies from 1 to 27
   20 SQLDATATYPE-NAMEC PIC X(27).
*           Variable or Column name
   20 FILLER PIC X(3).

```

8.2 SQLDA für C

```
/******  
*  
* Source File Name = SQLDA.H  
*  
* (C) COPYRIGHT International Business Machines Corp. 1987, 1997  
* All Rights Reserved  
* Licensed Materials - Property of IBM  
*  
* US Government Users Restricted Rights - Use, duplication or  
* disclosure restricted by GSA ADP Schedule Contract with IBM Corp.  
*  
* Function = Include File defining:  
*           SQL Descriptor Area  
*  
* Operating System = Common C Include File  
*  
*****/  
  
#ifndef  SQLDASIZE          /* Permit duplicate Includes          */  
  
#include "sqlsystem.h"    /* System-Specific Include          */  
/* _SQLLOLDCHAR defaults to 'char'. See sqlsystem.h for details. */  
  
#if (defined(DB2OS2) || defined(DB2NT))  
#pragma pack(4)  
#elif (defined(DB2WIN))  
#pragma pack(2)  
#elif (defined(DB2MAC))  
#if (defined(DB2PPC))  
#pragma align 4  
#elif (defined(DB268K))  
#pragma align 2  
#endif  
#elif (defined(DB2HP) || defined(DB2SNI))
```

```

#elif (defined(DB2SUN) && defined(__xlc__))
#pragma options align=natural
#elif (defined(DB2AIX) && defined(__64BIT__))
#pragma options align=natural
#elif (defined(DB2AIX))
#pragma options align=power
#endif

/* SQLDA: SQL Descriptor Area - Variable descriptor */

SQL_STRUCTURE sqlname /* Variable Name */
{
    short length; /* Name length [1..30] */
    _SQLOLDCHAR data[30]; /* Variable or Column name */
};

SQL_STRUCTURE sqldistinct_type /* Name of user-defined type */
{
    short length; /* Name length [1..27] */
    char data[27]; /* Name of user-defined type */
    char reserved1[3]; /* reserved */
};

/* Structure for any user-defined types: */
/* Distinct types, structured types, and reference types */
typedef SQL_STRUCTURE sqldistinct_type sqluser_def_type;

SQL_STRUCTURE sqlvar /* Variable Description */
{
    short sqltype; /* Variable data type */
    short sqllen; /* Variable data length */
    _SQLOLDCHAR *SQL_POINTER sqldata; /* Pointer to variable data value */
    short *SQL_POINTER sqlind; /* Pointer to Null indicator */
    struct sqlname sqlname; /* Variable name */
};

```

```
SQL_STRUCTURE  sqlda
{
    _SQLOLDCHAR  sqldaid[8];          /* Eye catcher = 'SQLDA  '      */
    /******
    /* The 7th byte has special meaning.  If it is a '2', this means there
    /* are twice as many sqlvars as there are host variables or columns.
    /* This is typically used when Large Objects (LOBs) or User Defined
    /* Types (UDFs) are present in the SQLDA.  The first N entries use the
    /* sqlvar structure and the second N entries use the sqlvar2 structure.
    /******
    sqlint32     sqldabc;              /* SQLDA size in bytes=16+44*SQLN */
    short        sqln;                /* Number of SQLVAR elements      */
    short        sqld;                /* # of columns or host vars.     */
    struct sqlvar sqlvar[1];          /* first SQLVAR element           */
};

/* macro for allocating SQLDA */
#define SQLDASIZE(n) (offsetof(struct sqlda, sqlvar) + \
    (n) * sizeof(struct sqlvar))

/******
/* Because we may need to support 8 byte lengths in the future,
/* sql8bytelen is defined using 8 bytes.  Currently, however, we have only
/* four byte integers.
/******
union sql8bytelen
{
    sqlint32     reservel[2];         /* reserved for future 8 byte lengths. */
    sqlint32     sqllonglen;         /* this is what is currently used      */
};

union sql4bytelen
{
    unsigned char reservel[4];       /* reserved                            */
    sqlint32     sqllonglen;         /* this is what is currently used      */
};
```

```

/*****/
/* The sqlvar2 structure maps the second "N" sqlvar entries. The */
/* second "N" entries are used to hold the length of LOB columns */
/* and host variables. The second "N" entries may also be used to */
/* hold the SQLDATALEN field for LOB columns on a FETCH USING */
/* DESCRIPTOR request. */
/* */
/* To set or retrieve these fields, use the macros provided below. */
/*****/

#ifdef db2Is64bit
#define SQLVAR2_PAD 8
#else
#define SQLVAR2_PAD 0
#endif

SQL_STRUCTURE sqlvar2 /* Variable Description */
{
    union sql4bytelen len; /* Four byte length. */
    unsigned char reserve2[3+SQLVAR2_PAD]; /* Reserved */
    unsigned char sqlflag4; /* Indicates type of Var (see flag values */
                                /* below) */
    char *SQL_POINTER sqldataalen; /* Pointer to four (4) byte */
                                    /* length buffer. This may be */
                                    /* used to hold the length for */
                                    /* lob data types. */
    sqluser_def_type sqldatatype_name; /* User-defined type name */
};

/*****/
/* Flag values for sqlflag4 field of sqlvar2. */
/*****/

#define SQLFLAG4_BASE 0x00 /* Type is base type or distinct type */
#define SQLFLAG4_REF 0x01 /* Reference type */
#define SQLFLAG4_STRUCT 0x12 /* Structured type */

```

```

/*****/
/* Macros for using the sqlvar2 fields. */
/*****/

/*****/
/* A '2' in the 7th byte of sqldaid indicates a doubled amount of sqlvars. */
/*****/
#define SQLDOUBLED '2'
#define SQLSINGLED ' '

/*****/
/* GETSQLDOUBLED(daptr) will return 1 if the SQLDA pointed to by daptr */
/* has been doubled, or 0 if it has not been doubled. */
/*****/
#define GETSQLDOUBLED(daptr) (((daptr)->sqldaid[6] == ( char) SQLDOUBLED) ? \
    (1) : \
    (0) )

/*****/
/* SETSQLDOUBLED(daptr, SQLDOUBLED) will make the 7th byte of sqldaid to */
/* be a '2'. */
/* SETSQLDOUBLED(daptr, SQLSINGLED) will make the 7th byte of sqldaid to */
/* be a ' '. */
/*****/
#define SETSQLDOUBLED(daptr, newvalue) (((daptr)->sqldaid[6] = (newvalue)))

/*****/
/* GETSQLDALONGLEN(daptr,n) will return the data length of the nth entry */
/* in the sqlda pointed to by daptr. Use this only if the sqlda was */
/* doubled and the nth SQLVAR entry has a LOB datatype. */
/*****/
#define GETSQLDALONGLEN(daptr,n) \
    (((struct sqlvar2 *) &((daptr)->sqlvar[(n) + ((daptr)->sqlid)])->len.sqllong-
glen)

```

```

/*****
/* SETSQLDALONGLEN(daptr,n,len) will set the sqllonglen field of the      */
/* sqllda pointed to by daptr to len for the nth entry. Use this only if  */
/* the sqllda was doubled and the nth SQLVAR entry has a LOB datatype.   */
/*****
#define SETSQLDALONGLEN(daptr,n,length) { \
    struct sqlvar2      *var2ptr; \
    var2ptr = (struct sqlvar2 *) &((daptr)->sqlvar[(n) + ((daptr)->sqllda)]); \
    var2ptr->len.sqllonglen = (sqlint32) (length); \
}

/*****
/* GETSQLDALENPTR(daptr,n) will return a pointer to the data length for    */
/* the nth entry in the sqllda pointed to by daptr. Unlike the inline     */
/* value (union sql8bytelen len), which is 8 bytes, the sqldataalen pointer */
/* field returns a pointer to a sqlint32 (4 byte) integer.                 */
/* If the SQLDATALEN pointer is zero, a NULL pointer will be returned.     */
/*                                                                           */
/* NOTE: Use this only if the sqllda has been doubled.                     */
/*****
#define GETSQLDALENPTR(daptr,n) \
    ((sqlint32 *) ((struct sqlvar2 *) &((daptr)->sqlvar[(n) + (daptr)->sqllda])->sql-
dataalen )

/*****
/* SETSQLDALENPTR(daptr,n,ptr) will set a pointer to the data length for   */
/* the nth entry in the sqllda pointed to by daptr.                       */
/* Use this only if the sqllda has been doubled.                         */
/*****
#define SETSQLDALENPTR(daptr,n,ptr) { \
    struct sqlvar2 *var2ptr; \
    var2ptr = (struct sqlvar2 *) &((daptr)->sqlvar[(n) + ((daptr)->sqllda)]; \
    var2ptr->sqldataalen = (char *) ptr; \
}

```

```

/*****
/* GETSQLDAFLAG4(daptr,n) will return the type character flag (sqlflag4) */
/* from the nth entry in the sqlda pointed to by daptr. */
/* Use this only if the sqlda was doubled. */
/*****

#define GETSQLDAFLAG4(daptr,n) ( \
    (char) (((struct sqlvar2 *) &((daptr)->sqlvar[(n) + \
        ((daptr)->sqld)]))->sqlflag4) )

/*****
/* SETSQLDAFLAG4(daptr,n,flag) will set the type character flag (sqlflag4) */
/* from the nth entry in the sqlda pointed to by daptr. */
/* Use this only if the sqlda was doubled. */
/*****

#define SETSQLDAFLAG4(daptr,n,flag) { \
    struct sqlvar2 *var2ptr; \
    var2ptr = (struct sqlvar2 *) &((daptr)->sqlvar[(n) + ((daptr)->sqld)]; \
    var2ptr->sqlflag4 = (char) (flag); \
}

#if (defined(DB2OS2) || defined(DB2NT) || defined(DB2WIN))
#pragma pack()
#elif (defined(DB2MAC))
#pragma align
#elif (defined(DB2HP) || defined(DB2SNI))

#elif (defined(DB2AIX) || (defined(DB2SUN) && defined(__xlc__)))
#pragma options align=reset
#endif

#endif /* SQLDASIZE */
```

9. Stichwortverzeichnis

Symbole

.BND 1-3
.BND-Datei 7-13
 erzeugen 7-1
.C 7-3
.c 7-3
.cbl 7-3
.cxx 7-3
.f 7-3
.for 7-3
.sqb 7-2
.sqC 7-2
.sqc 7-2
.sqf 7-2
.sqx 7-2

- A -

Abfrage
 ausführen 5-15
Anwendungsprogrammierung
 1-1
 COBOL 4-1
API 1-2
application programming inter-
 face 1-2
ausführbares Lade-Modul 1-3
ausführbares Programm 1-3
Ausführung
 dynamische von SQL-Befehl
 6-1
Ausnahmebedingung 5-22
Ausnahmезustand 5-22

- B -

BEGIN DECLARE SECTION
 5-2
Beispiel-Programm
 Liegeplatz-Gebühren kassie-

ren 4-1
 MARIRECH 4-1
Benutzertrennung
 festlegen 7-8, 7-16
BIND 1-3
 Zugriffspaket erstellen 7-11
Binde-Datei
 erzeugen 7-6
binden 7-2
 Kommandos 7-1
BINDFILE (Parameter) 3-8,
 4-15

- C -

C 1-1
CALL 5-17
Call-Level-Interface 1-2
Call-Schnittstelle 1-2
CLI 1-2
CLOSE 5-10
COBOL 1-1
 Anwendungsprogrammie-
 rung 4-1
COBOL-Compiler
 aufrufen 3-8, 4-16
 IBM 32-Bit-Compiler 4-15
COBOL-Datentyp 4-14
COMP-5 4-14
Compiler 1-3
 COBOL- aufrufen 3-8, 4-16
 für PREP-Code vorgeben
 7-11
Compound SQL 5-20
CONNECT
 Durchführung 7-10

Cursor 2-1
 definieren 5-4
 öffnen 5-7
 positionieren 5-9
 schließen 5-10
 zweideutiger 5-5

- D -

Darstellung
 Datum festlegen 7-7, 7-15
Datentyp
 COBOL- 4-14
Datum
 Darstellung festlegen 7-7,
 7-15
DECLARE CURSOR 5-4
Deklaration
 kopieren 5-3
Deklarationsteil
 benden 5-2
 einleiten 5-2
DELETE 5-10
DESCRIBE 6-1
DSQL 1-1
dynamic SQL-Befehle 6-1
dynamische Ausführung
 SQL-Befehl 6-1
dynamische Übersetzung
 SQL-Befehl 6-1
dynamisches SQL 1-1

- E -

embedded SQL 1-1
END DECLARE SECTION 5-2
Ergebnis-Tabelle
 bearbeiten 5-4
 erzeugen 5-19
 Programm-Variablen zuwei-
 sen 5-19
ESQL 1-1

ESQL-Befehl 5-1
EXECUTE 6-5
EXECUTE IMMEDIATE 6-6
EXPLAIN-Tabelle 7-8, 7-16

- F -

Fallstudie 4-1
FETCH 5-9
FREE LOCATOR 5-19

- I -

IBM 32-Bit COBOL-Compiler
4-15
INCLUDE 5-3
Indikator-Variable 2-2, 4-13
INSERT 5-11

- J -

JAVA -vii

- K -

Katalog-Tabelle
SYSIBM.SYPLANAUTH
3-8, 4-16
SYSIBM.SYSPLAN 3-8,
4-16
SYSIBM.SYSPLANDEP
3-8, 4-16
SYSIBM.SYSSECTION 3-8,
4-16
SYSIBM.SYSSTMT 3-8,
4-16

Kommunikationsbereich
SQL- 3-5, 4-14
Kompatibilität
festlegen 7-9
kopieren
Deklaration 5-3

- L -

Lade-Modul
ausführbares 1-3
Liegeplatz-Gebührenkassieren
(Beispiel-Programm) 4-1
Linkage Editor 1-3
aufrufen 3-8, 4-16
Linker 1-3
aufrufen 3-8, 4-16
Listendatei 7-13
Lokator-Variable
Verbindung trennen 5-19
löschen
Zeile 5-10

- M -

MARIRECH (Beispiel-Pro-
gramm) 4-1
Meldungen
ausgeben 7-9

- N -

NULL-Indikator 2-2, 4-13
NULL-Wert 2-2, 4-13

- O -

ODBC 1-2
OPEN 5-7
Optimizer 1-1
Optimizer-Klasse
vorgeben 7-10, 7-16

- P -

Precompiler 1-1, 7-1
Ausgabe umsetzen 7-11
Precompiler PREP 3-8, 4-15
PREP 7-1
Compiler für -Code vorgeben
7-11
PREP (Precompiler) 3-8, 4-15
PREPARE 6-3

Programm

ausführbares 1-3
Programmierung 1-1
Programm-Variable
Deklarationsteil benden 5-2
Deklarationsteil einleiten 5-2
Ergebnis-Tabelle zuweisen
5-19
Prozedur
aufrufen 5-17

- Q -

Quelldatei
vorübersetzen 7-1
Quellprogramm
bearbeiten 1-3
vorübersetztes -, Name 7-9

- R -

REXX 1-2

- S -

Schema-Name
vorgeben 7-6, 7-15
Schnittstelle
Call- 1-2
SQL- 1-1
SELECT
ausführen 5-7
SELECT INTO 5-15
SELECT-Befehl 2-1
Sicherheit 1-1
Sicht
Zeile einfügen 5-11
SQL
compound 5-20
dynamisches 1-1
statisches ESQL 1-1
SQL-Befehl
ausführen 6-5 bis 6-6
dynamische Ausführung 6-1
dynamische Übersetzung

- 6-1
 - statischer 5-1
 - übersetzen 1-3, 6-3, 6-6
 - übersetzter 6-1
 - zusammenfassen 5-20
 - SQLBIND 1-3
 - SQLCA 3-5, 4-14
 - SQLD (Variable) 6-2
 - SQLDA
 - als COBOL-COPY 8-1
 - für C 8-3
 - SQLDABC (Variable) 6-2
 - SQLDAID (Variable) 6-2
 - SQLDATATYPE_NAME (Variable) 6-2
 - SQL-Kommunikationsbereich 3-5, 4-14
 - SQLLEN (Variable) 6-2
 - SQLLONGLEN (Variable) 6-2
 - SQLNAME (Variable) 6-2
 - SQLPREP 1-3
 - SQL-Schnittstelle 1-1
 - SQLTYPE (Variable) 6-2
 - SQLVAR (Variable) 6-2
 - statischer SQL-Befehl 5-1
 - statisches ESQL 1-1
 - SYNTAX (Parameter) 3-8, 4-15
 - Syntax-Prüfung 3-8, 4-15
 - SYSIBM.SYPLANAUTH 3-8, 4-16
 - SYSIBM.SYSPLAN 3-8, 4-16, 7-2, 7-11
 - SYSIBM.SYSPLANDEP 3-8, 4-16
 - SYSIBM.SYSSECTION 3-8, 4-16, 7-2, 7-11
 - SYSIBM.SYSSTMT 3-8, 4-16
- T -**
- Tabelle
 - bearbeiten 5-4
 - Zeile einfügen 5-11
- TARGET (Parameter) 3-8, 4-15
- U -**
- übersetzen
 - Kommandos 7-1
 - SQL-Befehl 1-3, 6-3, 6-6
 - übersetzter SQL-Befehl 6-1
 - Übersetzung
 - dynamische von SQL-Befehl 6-1
 - Uhrzeit
 - Darstellung festlegen 7-7, 7-15
 - UPDATE 5-13
- V -**
- VALUES INTO 5-19
 - vorübersetzen
 - Quelldatei 7-1
 - Quellprogramm 1-3
 - vorübersetztes Quellprogramm
 - Name 7-9
- W -**
- WHENEVER 5-22
- X -**
- X/Open CLI 1-2
- Y -**
- Yachthafen
 - Fallstudie 4-1
- Z -**
- Zeile
 - ändern 2-1, 5-13
 - einfügen 5-11
 - lesen 2-1
 - löschen 2-1, 5-10
 - Zugriffspaket 3-8, 4-15
 - erstellen 7-9
 - mit BIND erstellen 7-11
 - Zugriffspfad 1-1
 - Zugriffsplan
 - erstellen 7-11
 - Zugriffsweg
 - abspeichern 7-8, 7-16
 - zweideutiger Cursor 5-5