3 Embedded SQL

3.1 Theorie

3.1.1 Was ist Embedded SQL?

Nachdem bisher alle SQL-Anweisungen über den CLP abgewickelt wurden, sollen diese Anweisungen nun in eine Host-Sprache eingebettet werden. Wir werden die Programmiersprache C verwenden. Abbildung 6 zeigt die prinzipielle Vorgehensweise.

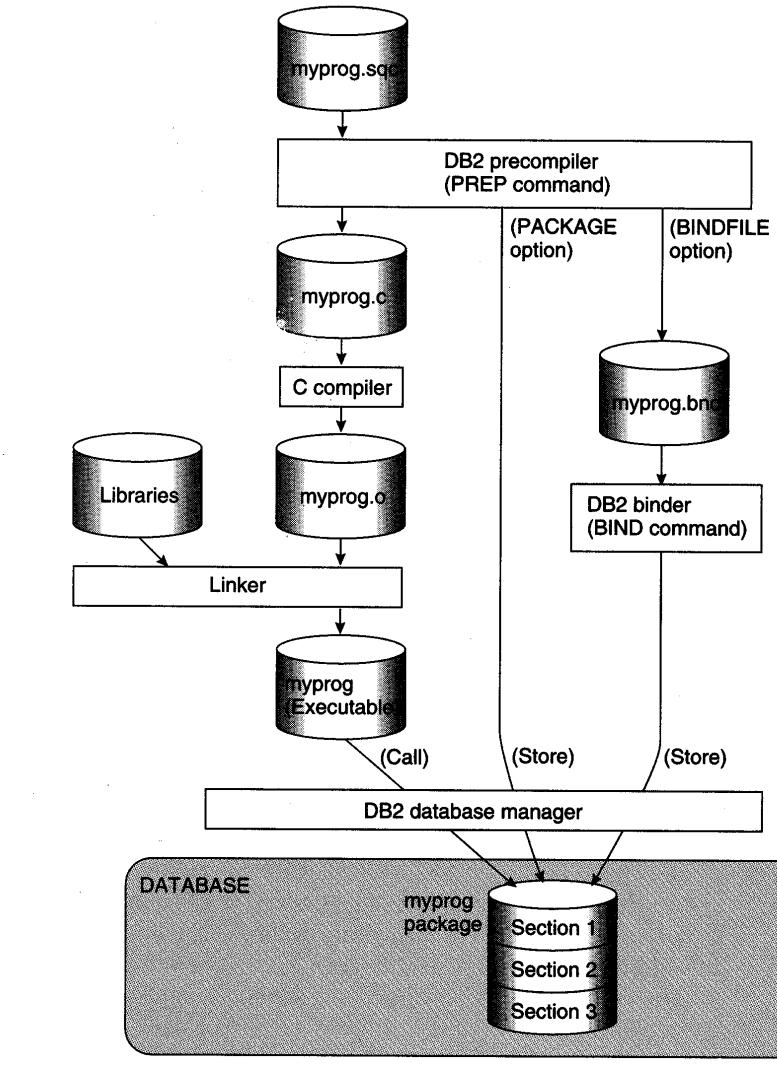

Figure 1-6: The Process of Binding an Application Program

Abbildung 6: Vorgehensweise für Einbettung von SQL-Anweisungen

Ein C-Programm mit eingebetteten SQL-Anweisungen (auch SQC-Datei genannt) wird zuerst von einem SQL-Präprozessor (Aufruf mit PREP) verarbeitet. Alle SQL-Anweisungen im Programm müssen mit EXEC SQL eingeleitet werden, damit der SQL-Präprozessor die Befehle erkennt. Jede SQL-Anweisung muß mit einem Semikolon enden, darf sich aber über mehrere Zeilen erstrecken. Der SQL-Präprozessor durchsucht das Programm nach den SQL-Anweisungen und generiert für jede Anweisung einen optimierten Ausführungsplan. Die Ausführungspläne werden in der Datenbank in einer sogenannten *Package* abgelegt. Das Generieren einer Package für ein Programm wird Binden (englisch: binding) des Programms genannt. Während des Bindens wird auch die syntaktische Korrektheit des Programms überprüft. Bei Fehlern gibt das Datenbanksystem eine Fehlermeldung zurück. Nach erfolgreichem Binden können Packages mit dem Befehl LIST PACKAGES angezeigt (und mit dem Befehl DROP PACKAGE <Packagename> auch wieder gelöscht) werden. Außerdem ersetzt der SQL-Präprozessor die SQL-Anweisungen durch reinen C-Code, der die entsprechenden Ausführungspläne in der zugehörigen Package aufruft. Danach kann das Programm mit jedem gewöhnlichen C-Compiler übersetzt und gelinkt werden.

Der SQL-Präprozessor kann mit verschiedenen Parametern aufgerufen werden. Mit keinem Parameter oder nur dem Parameter PACKAGE wird eine Package im Datenbanksystem erzeugt und ein C-Programm aus der SQC-Datei generiert. Bei Angabe des Parameters BINDFILE wird eine sogenannte Bind-Datei (mit der Endung .bnd), aber keine Package erzeugt (dafür muß zusätzlich der Parameter PACKAGE angegeben werden). Was ist der Nutzen einer Bind-Datei? Wenn sich in der Datenbank etwas ändert (z.B. ein Index wird hinzugefügt), bekommt die Anwendung davon erst einmal nichts mit, d.h. die Ausführungspläne werden nicht an die neue Situation angepaßt. Bleiben wir bei dem Beispiel mit dem hinzugefügten Index. Die Anwendung wird beim Aufruf immer noch die alten (langsameren) Ausführungspläne ausführen, die den neuen Index nicht berücksichtigen. Mit Hilfe des REBIND Befehls und der Bind-Datei können eine neue Package und damit neue optimierte Ausführungspläne erzeugt werden, ohne daß die Anwendung komplett neu übersetzt werden muß. In einigen Fällen, in denen die Anwendung ohne REBIND nicht mehr lauffähig wäre, kann das Datenbanksystem dies erkennen und implizit das Programm neu binden. Kommerzielle Anwendungen profitieren ebenfalls von Bind-Dateien. Mit Hilfe des BIND Befehls und der Bind-Datei kann ohne Sourcecode eine Package erzeugt werden, d.h. es müssen lediglich eine ausführbare Datei und die Bind-Datei ausgeliefert werden.

3.1.2 Ein einführendes Beispiel

In diesem Abschnitt wird anhand eines kleinen Beispielprogramms erläutert, wie SQL in C eingebettet wird.

C-Includes für Ein-/Ausgabe und Stringverarbeitung

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

Hier sind Strukturen, Funktionen und Konstanten enthalten, die jedes C-Programm braucht, um mit der Datenbank zu interagieren.

```
#include <sqlenv.h>
```

Deklariert eine Struktur für Fehler- und Rückmeldecodes

```
EXEC SQL INCLUDE SQLCA;

main()
{
```

Der SQL Deklarationsabschnitt enthält alle Hostsprachendeklarationen der Hostvariablen, die später in den SQL-Anweisungen verwendet werden.

```
   EXEC SQL BEGIN DECLARE SECTION;
    char name[30];
    char fachgebiet[30];
   EXEC SQL END DECLARE SECTION;
```

Hier wird ein Cursor für die Anfrage deklariert die später ausgeführt werden soll.

```
   EXEC SQL DECLARE C1 CURSOR FOR
    SELECT name, fachgebiet
    FROM   assistenten;
```

Wir bauen eine Verbindung zur Datenbank *unidb* auf.

```
   EXEC SQL CONNECT TO unidb;
```

Der Cursor wird geöffnet.

```
   EXEC SQL OPEN C1;
```

Solange kein Fehler vorliegt....

```
   while(strncmp(sqlca.sqlstate, "02000", 5))
   {
```

Schreibe Werte des nächsten Tupels in die Hostvariablen. Allen Hostvariablen wird ein Semikolon vorangestellt.

```
      EXEC SQL FETCH C1
        INTO :name, :fachgebiet;
```

Wenn kein Fehler beim Holen des Tupels aufgetaucht ist, dann Tupelwerte ausgeben.

```
      if(strncmp(sqlca.sqlstate, "02000", 5))
      {
         printf("%s %s\n", name, fachgebiet);
      }
   }
```

Cursor schließen und Verbindung zur Datenbank abbauen.

```
   EXEC SQL CLOSE C1;

   EXEC SQL DISCONNECT CURRENT;

   return;
}
```

3.1.3 Was gibt es noch zu sagen?

In diesem Abschnitt sind wichtige Kleinigkeiten aufgeführt, die den Umgang mit Embedded SQL erleichtern.

Hostvariablen In SQL-Ausdrücken kann hinter jeder Hostvariablen eine Indikatorvariable angegeben werden. Beide diese Variablen müssen mit einem Semikolon beginnen, wenn sie in SQL-Ausdrücken verwendet werden (im normalen C-Code werden sie selbstverständlich ohne Semikolon angesprochen). Die Indikatorvariable wird benutzt, um NULL-Werte darzustellen. Eine negativer Wert einer Indikatorvariable steht für einen NULL-Wert in der entsprechenden Hostvariable. Ein Wert größer gleich 0 markiert einen gültigen Hostvariablenwert. (Siehe Abbildung 7 für ein Syntaxdiagramm.)

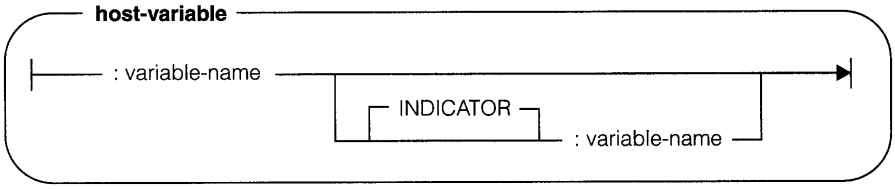

Abbildung 7: Host- und Indikatorvariablen

SQL Declare Section Alle Hostvariablen, die später in SQL-Ausdrücken verwendet werden sollen, müssen in diesem Abschnitt deklariert werden. Auf diese Weise werden sie dem SQL-Präprozessor bekanntgemacht. Die Deklarationen sind im allgemeinen in der Hostsprache formuliert. Für eine Entsprechung zwischen C-Datentypen und SQL-Datentypen siehe Abbildung 8.

TABLE 2-5: C Datatypes Corresponding to Basic SQL Datatypes

SQL Datatype	C Datatype
Smallint and indicator variables	`short`
Integer	`long`
Decimal(p,s)	(no C equivalent)
Double	`double`
Char(n)	`char[n+1]` (null-terminated)

Varchar(n)	`char[n+1]` (null-terminated)
	or
	`struct`
	`{`
	`short length;`
	`char data[n];`
	`}`

Date	`char[11]`
Time	`char[9]`
Timestamp	`char[27]`
Graphic(n)	`wchar_t[n+1]` (null-terminated)

Vargraphic(n)	`wchar_t[n+1]` (null-terminated)
	or
	`struct`
	`{`
	`short length;`
	`wchar_t data[n];`
	`}`

Abbildung 8: Entsprechungen zwischen C- und SQL-Datentypen

Der Decimal Datentyp muß in C in einen String oder in eine Fließkommazahl umgewandelt werden. Dazu können die eingebauten Funktionen CHAR und DECIMAL bzw. DOUBLE und DECIMAL (in den SQL-Anweisungen) verwendet werden.

Cursor INSERT, DELETE und UPDATE Operationen sind relativ einfach in C einzubetten. Sie werden ausgeführt, ändern womöglich den Inhalt der Datenbank und lie-

fern eine SQLCA Struktur zurück, die Erfolg oder Mißerfolg anzeigt. Anfragen sind etwas schwieriger zu formulieren, da nicht von vornherein feststeht, wieviele Ergebnistupel zurückgeliefert werden. Zu diesem Zweck wird ein Cursor benötigt, der das Ergebnis Tupel für Tupel zurückliefert. Ein Cursor kann am ehesten mit einem Namen (oder einem Handle) für eine Anfrage verglichen werden. Beim Deklarieren wird der Cursor an eine Anfrage gekoppelt. Mit Hilfe der drei Befehle OPEN, FETCH und CLOSE kann mit einem Cursor gearbeitet werden. Siehe Abbildung 9 für ein Syntaxdiagramm.

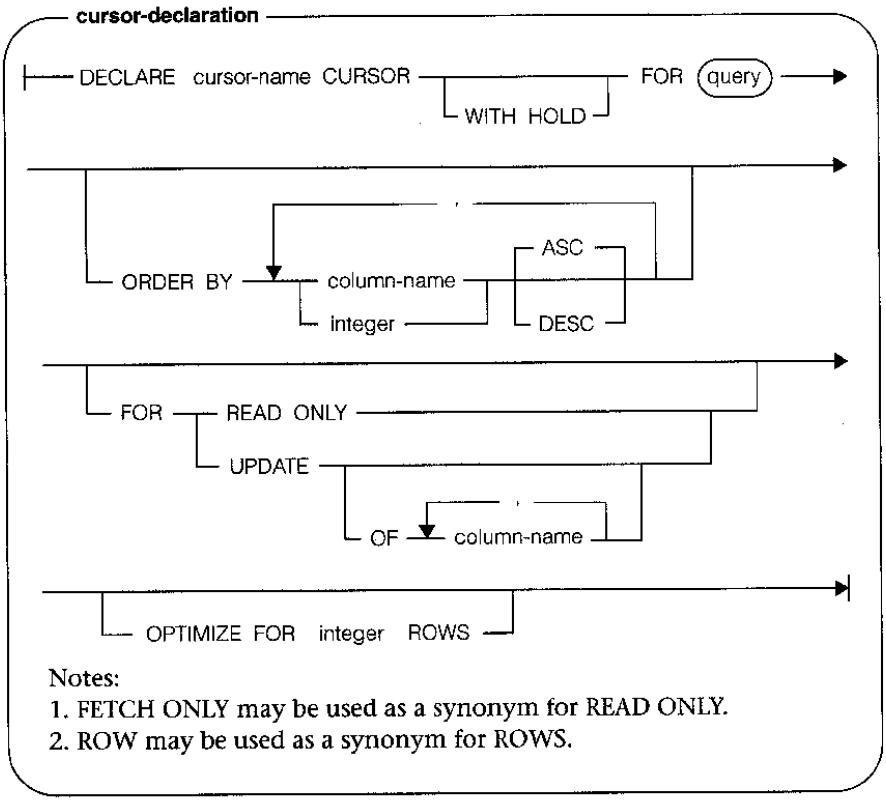

Abbildung 9: Cursor Deklaration

OPEN Eine OPEN-Anweisung öffnet einen Cursor und bereitet ihn für eine Anfrage vor. Der Cursor wird vor das erste Tupel gesetzt. (Syntaxdiagramm siehe Abbildung 10.)

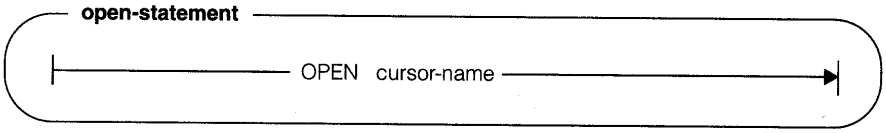

Abbildung 10: OPEN-Anweisung

FETCH Eine FETCH-Anweisung holt das nächste Tupel und schreibt die Werte in die entsprechenden Hostvariablen. Wenn es kein Tupel mehr nach dem aktuellen gibt, dann wird der SQLCODE +100 (SQLSTATE 02000) zurückgegeben. (Syntaxdiagramm siehe Abbildung 11.)

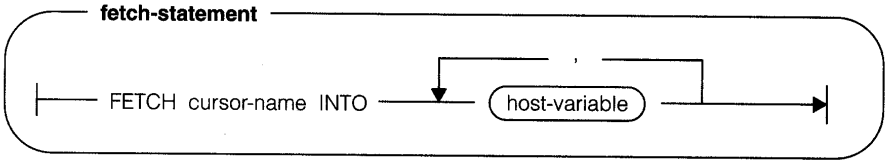

Abbildung 11: FETCH-Anweisung

CLOSE Eine CLOSE-Anweisung schließt einen Cursor und gibt belegte Ressourcen wieder frei. Wenn der Cursor später wieder geöffnet wird, dann wird die Anfrage frisch ausgeführt, d.h. der Cursor ist nicht mehr an der alten Position. (Syntaxdiagramm siehe Abbildung 12.)

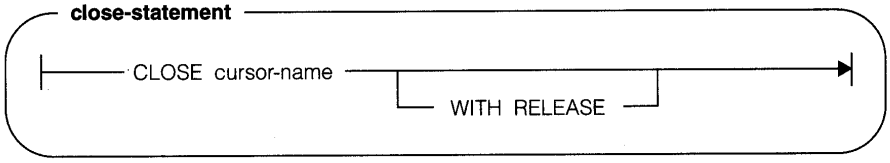

Abbildung 12: CLOSE-Anweisung

Fehlermeldungen Die Fehlermeldungen die in einer SQLCA structure stehen sind nicht besonders lesbar. Eine aussagekräftige Übersetzung der Fehlermeldung erhält man durch einen Aufruf der Funktion `sqlaintp`. Diese Funktion bekommt als Argument eine SQLCA structure und liefert einen String mit der detailierten Fehlermeldung zurück.

```
int sqlaintp (
   char*  buffer,              /* Puffer fuer die Meldung */
   short  buff_size,           /* Groesse des Puffers */
   short  line_width,          /* erwuenschte Zeilenlaenge */
   struct sqlca* sqlca         /* soll dekodiert werden */
);
```

Der erste Parameter ist ein Zeiger auf den Puffer in dem die dekodierte Meldung gespeichert werden soll. Der zweite Parameter gibt die Größe des Puffers an (die meisten Meldungen passen in einen Puffer von 512 byte). Wenn `sqlaintp` einen negativen Wert zurückgibt, gibt an, daß der Puffer nicht geändert wurde (es gab keine Fehlermeldung). Ein positiver Wert gibt die Länge der Fehlermeldung an.

3.2 Versuch2: Embedded SQL

3.2.1 Laufenlassen eines einführenden Beispiels

In Ihrem Verzeichnis/auf der Webseite finden Sie die Datei `beispiel.sqc`. Lassen Sie das Programm zunächst durch den SQL-Präprozessor laufen und übersetzen Sie es danach mit einem C-Compiler. (Hinweise dazu finden Sie in der Datei `READMEBSP`.)

3.2.2 Eine kleine Applikation

1. Schreiben Sie ein C-Programm, das als Eingabe zwei Städtenamen erhält und die Entfernung (Luftlinie) zwischen diesen Städten berechnet.

   Nehmen Sie für diese Aufgabe an, daß die Erde eine Kugel ist (um das ganze zu vereinfachen). Rechnen Sie zunächst die Längen-/ Breitengradangaben mit Hilfe folgender Formeln in kartesische Koordinaten (x,y,z) um:

   $$\begin{aligned} x &= \cos(\text{Breite}) * \cos(\text{Länge}) \\ y &= \cos(\text{Breite}) * \sin(\text{Länge}) \\ z &= \sin(\text{Breite}) \end{aligned}$$

   (Achten Sie darauf, daß die Winkelfunktionen in C Angaben in Bogenmaß erwarten!)

   Die Entfernung zwischen zwei Städten kann jetzt mit Hilfe des Skalarprodukts berechnet werden (Erdradius = 6371.032 km):

   $$\text{Distanz} = \arccos(x_1 * x_2 + y_1 * y_2 + z_1 * z_2) * \text{Erdradius}$$

2. Erweitern Sie das Programm um einen Eingabeteil der nichtvorhandene Städte einfügt (INSERT) bzw. nichtvorhandene Koordinatenangaben ändert (UPDATE).