4 Dynamic SQL

SQL kann in vielen verschiedenen Spielarten benutzt werden. Bisher hatten wir interaktives SQL (CLP) und in Anwendungsprogramme eingebettetes (statisches) SQL. Was ist, wenn beim Übersetzen der Applikation die SQL-Anfragen, die später an die Datenbank gestellt werden sollen, noch nicht feststehen? Wir brauchen ein Programm, das SQL-Anweisungen während der Laufzeit generieren und ausführen kann. Hier kommt Dynamic SQL ins Spiel. In DB2 existieren mehrere verschiedene Arten des Dynamic SQL: das Call Level Interface (CLI) und Embedded Dynamic SQL. Wir beschäftigen uns mit dem CLI. Für CLI, ursprünglich von Microsoft, X/Open und der SQL Access Group entworfen, exisistiert inzwischen ein internationaler Standard (ISO/IEC 9075-3: SQL Call-Level-Interface). DB2 unterstützt ODBC (Open Database Connectivity, die Microsoft Version des CLI) Level 1 voll und Level 2 zu großen Teilen. Aus diesem Grund können CLI-Applikationen relativ einfach auf verschiedene Datenbanksysteme portiert werden. Eine weitere standardisierte Schnittstelle stellt JDBC dar, das Gegenstück von ODBC für die Programmiersprache Java. Im Gegensatz zu ODBC ist JDBC objektorientiert entworfen und implementiert worden, d.h. es ist um einiges einfacher handzuhaben.

Wir werden zunächst auf ODBC eingehen, danach JDBC besprechen und zum Schluß noch auf die Integration von Datenbanken und Webservern zu sprechen kommen.

4.1 ODBC

In Abbildung 13 ist schematisch dargestellt, wie eine Anwendung, die dynamisches SQL (mit ODBC) verwendet, prinzipiell abläuft. In den weißen Bereichen der Kästchen sind die verwendeten Funktionen der CLI-Schnittstelle aufgeführt. Die Abbildung kann in drei Teile eingeteilt werden: die Initialisierungsphase, die Verarbeitungsphase (das gestrichelte Kästchen) und die Terminierungsphase. Die Verarbeitungsphase, in der die eigentliche Ausführung und Auswertung eines SQL-Ausdrucks stattfindet ist in Abbildung 14 detailierter dargestellt. Auch hier sind wieder die relevanten Funktionen der CLI-Schnittstelle in den weißen Bereichen aufgeführt.

4.1.1 Ein einführendes Beispiel

Handles Um CLI-Programme schreiben (und lesen) zu können, ist es wichtig das Konzept eines Handles zu verstehen. Ein Handle ist eine Referenz auf Informationen die "hinter den Kulissen" verwaltet wird. In C liegen Handles in der Form von Variablen vom Typ `long` vor. Es existieren drei verschiedene Arten von Handles:

Environment Handle: Ein Environmental Handle verwaltet den globalen Zustand der Applikation und wird zu Anfang allokiert und am Ende wieder freigegeben. Wir werden diesen Handle sonst aber kaum benötigen.

Connection Handle: Ein Connection Handle repräsentiert die Verbindung der Applikation zu einer bestimmten Datenbank. Ein CLI-Programm kann gleichzeitig mit mehreren Datenbanken oder mehrmals mit der gleichen Datenbank in Verbindung

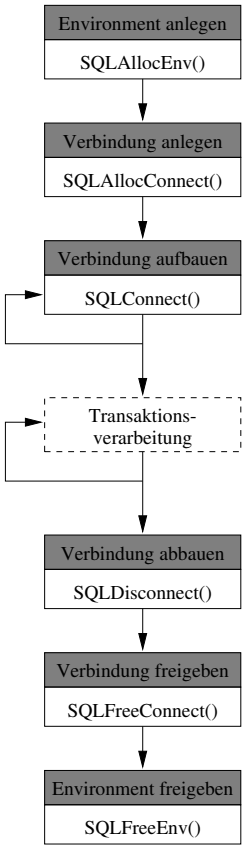

Abbildung 13: Schematische Darstellung einer CLI-Anwendung

stehen. Für jede dieser Verbindungen wird ein Connection Handle benötigt. Das erfolgreiche Abschließen (commit) oder Zurücksetzen (roll back) einer Transaktion wird ebenfalls über Connection Handles gesteuert.

Statement Handle: Hier wird der Ausführungszustand einer SQL-Anweisung festgehalten. Für jede SQL-Anweisung muß ein Statement Handle angelegt werden. Im Gegensatz zu Embedded SQL müssen Cursor, Fehler- und Rückmeldecodes nicht direkt verwaltet werden, dies wird alles von einem Statement Handle erledigt.

Nach dieser kurzen Einführung in Handles jetzt zum eigentlichen Beispielprogramm:

C-Includes für Ein-/Ausgabe und Stringverarbeitung

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

Um Portabilität zu gewährleisten, sollten die Datentypen aus diesem Includefile (anstelle der C-Datentypen) benutzt werden.

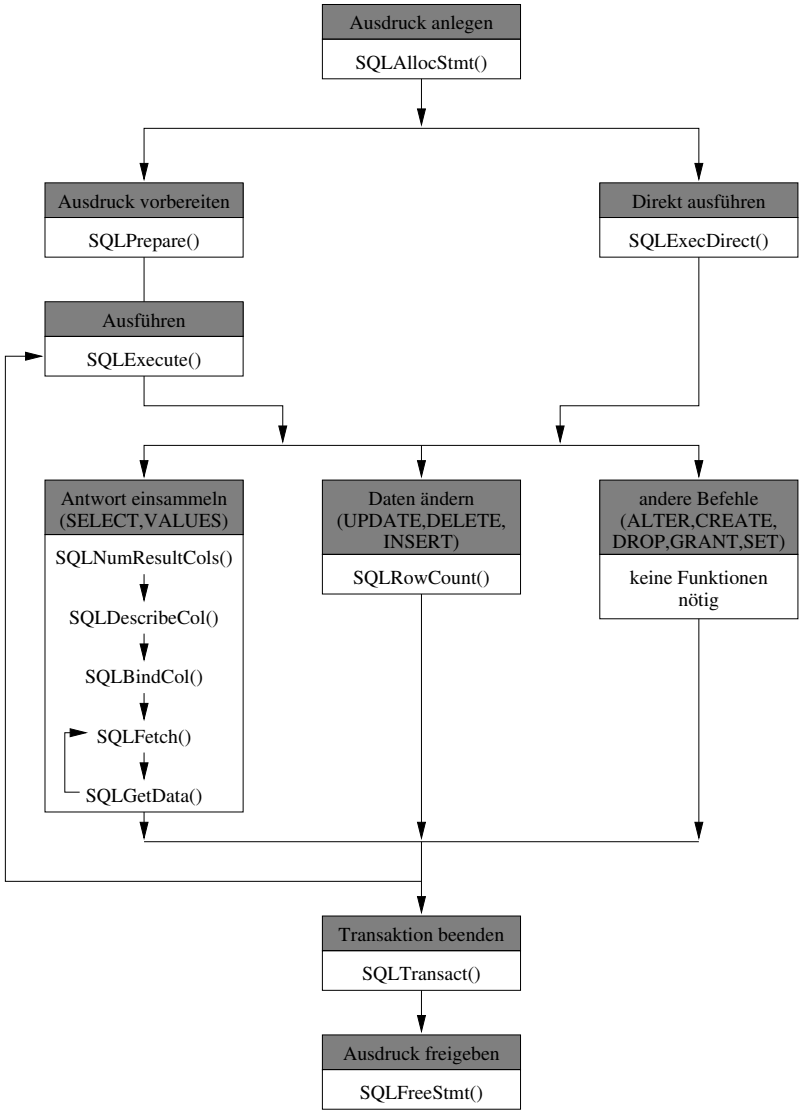

Abbildung 14: Schematische Darstellung der Verarbeitungsphase

```
#include <sqlcli1.h>
```

Vorwärtsdeklaration einer Fehlerbehandlungsroutine (die Routine selbst steht am Ende).

```
void errorExit(SQLHENV henv, SQLHDBC hdbc, SQLHSTMT hstmt, char* place);


int main()
{
```

Deklarieren der Handles.

```
SQLHENV henv;
SQLHDBC hdbc;
SQLHSTMT hstmt;
```

Deklarieren der Variablen, die später für die Anfrage benötigt werden. Es ist keine SQL Declare Section nötig, da das Programm nicht von einem SQL-Präprozessor bearbeitet wird. Hier könnten auch C-Datentypen stehen, aus Portabilitätsgründen werden aber die Typen aus `sqlcli1` verwendet.

```
SQLCHAR dbname[] = "unidb";
char qstring[80] = "select matrnr from studenten";
char tablename[19];

SQLSMALLINT ncols;
SQLCHAR colname[SQL_MAX_ID_LENGTH + 1];
SQLSMALLINT colnamelen;
SQLSMALLINT coltype;

SQLINTEGER answer;

SQLINTEGER nullindicator;

SQLRETURN rc;
```

Allokieren des Environment und Connection Handles.

```
SQLAllocEnv(&henv);
SQLAllocConnect(henv, &hdbc);
SQLSetConnectOption(hdbc, SQL_AUTOCOMMIT, SQL_AUTOCOMMIT_OFF);

printf("handles allocated\n");
```

Aufbauen der Verbindung zur Datenbank.

```
rc = SQLConnect(hdbc, dbname, SQL_NTS,
                NULL, SQL_NTS,
                NULL, SQL_NTS);
if(rc != SQL_SUCCESS)
{
    errorExit(henv, hdbc, SQL_NULL_HSTMT, "Connection");
}

printf("connected to database\n");
```

Allokieren des Statement Handles und Ausführen der Anfrage.

```
SQLAllocStmt(hdbc, &hstmt);

rc = SQLExecDirect(hstmt, (SQLCHAR*)qstring, SQL_NTS);

if(rc == SQL_ERROR)
{
    errorExit(henv, hdbc, hstmt, "query");
}
```

Hole Information über die Spalten der Antwort.

```
rc = SQLNumResultCols(hstmt, &ncols);

printf("number of columns: %d\n", ncols);

rc = SQLDescribeCol(hstmt, 1, colname, SQL_MAX_ID_LENGTH,
                    &colnamelen, &coltype, NULL, NULL, NULL);

printf("%s\n", colname);
printf("-----------\n");
```

Hier wird CLI mitgeteilt in welche Variable die Attributwerte geschrieben werden sollen.

```
SQLBindCol(hstmt, 1, coltype, &answer, 0, &nullindicator);
```

Hole solange Werte bis keine weiteren mehr gefunden werden.

```
while((rc = SQLFetch(hstmt)) != SQL_NO_DATA_FOUND)
{
    printf("%d\n", answer);
}
```

Beende Transaktion

```
rc = SQLTransact(henv, hdbc, SQL_COMMIT);
if(rc != SQL_SUCCESS)
{
    errorExit(henv, hdbc, SQL_NULL_HSTMT, "Commit");
}
```

Baue Verbindung ab und gebe Handles frei.

```
    SQLFreeStmt(hstmt, SQL_DROP);
    SQLDisconnect(hdbc);
    SQLFreeConnect(hdbc);
    SQLFreeEnv(henv);

    printf("cleaned up\n");

    return;
}



void errorExit(SQLHENV henv, SQLHDBC hdbc, SQLHSTMT hstmt, char* place)
{
    SQLCHAR sqlstate[SQL_SQLSTATE_SIZE + 1];
    SQLINTEGER sqlcode;
    SQLSMALLINT msglength;
    SQLCHAR msgbuffer[SQL_MAX_MESSAGE_LENGTH + 1];

    printf("\nSQL error at %s, transaction rolled back.\n", place);

    while(SQLError(henv, hdbc, hstmt, sqlstate, &sqlcode,
                   msgbuffer, SQL_MAX_MESSAGE_LENGTH + 1, &msglength)
          == SQL_SUCCESS)
    {
        printf("   SQLCODE = %d, SQLSTATE = %s\n", sqlcode, sqlstate);
        printf("   MESSAGE:  %s\n", msgbuffer);
    }

    SQLTransact(henv, hdbc, SQL_ROLLBACK);
    SQLDisconnect(hdbc);
    SQLFreeEnv(henv);
    exit(-2);
}
```

4.1.2 Die wichtigsten CLI Funktionen

Verbindung zu einer Datenbank

`SQLAllocEnv()` holt sich einen Environment Handle. Ein Environment Handle kann für eine oder mehrere Verbindungen benutzt werden.

`SQLAllocConnect()` holt sich einen Connection Handle.

`SQLConnect()` stellt unter Angabe eines Datenbanknamens, einer UserID und eines Passworts eine Verbindung zu einer Datenbank her.

`SQLSetConnectOption()` setzt Attribute einer Verbindung.

Vorbereiten eines SQL-Ausdrucks

`SQLAllocStmt()` holt sich einen Statement Handle.

`SQLPrepare()` bereitet einen SQL-Ausdruck für eine spätere Ausführung vor.

Ausführen eines SQL-Ausdrucks

`SQLExecute()` führt einen vorher vorbereiteten SQL-Ausdruck aus.

`SQLExecDirect()` führt einen SQL-Ausdruck aus.

Holen des Ergebnisses

`SQLRowCount()` gibt die Anzahl der beeinflussten Tupel bei einem Insert-, Update- oder Delete-Ausdruck an.

`SQLNumResultCols()` gibt die Anzahl der Attribute der Ergebnisrelation zurück.

`SQLDescribeCol()` gibt eine Beschreibung eines Attributs der Ergebnisrelation zurück.

`SQLBindCol()` weist einem Attribut Speicherplatz zu und spezifiziert den Typ.

`SQLFetch()` holt das nächste Tupel aus der Ergebnisrelation.

`SQLGetData()` holt einen Teil oder den ganzen Wert eines Attributs des aktuellen Tupels.

`SQLError()` gibt im Falle eines Fehlers detailierte Fehlerbeschreibungen zurück.

Terminierung eines SQL-Ausdrucks

`SQLFreeStmt()` beendet die Bearbeitung des SQL-Ausdrucks, schließt den entsprechenden Cursor und gibt den Statement Handle wieder frei.

`SQLTransact()` schließt eine Transaktion erfolgreich ab (Commit) oder bricht sie ab (Rollback).

Verbindungsabbau

`SQLDisconnect()` schließt die Verbindung zu einer Datenbank.

`SQLFreeConnect()` gibt den Connection Handle wieder frei.

`SQLFreeEnv` gibt den Environment Handle wieder frei.

4.2 JDBC

Ziel von JDBC war es, einen portablen Standard für das Schreiben von Datenbankanwendungen zu schaffen. ODBC ist aufgrund der verwendeten Programmiersprache C mit einigen Problemen behaftet: C-Programme sind nicht so einfach auf andere Plattformen zu portieren wie Java, es fehlt eine (standardisierte) Infrastruktur in C für verschiedene Komponenten, wie z.B. eine graphische Benutzeroberfläche, und durch den prozeduralen Ansatz (von C) ist ODBC komplizierter handzuhaben als ein objektorientierter Ansatz.

4.2.1 Ein einführendes Beispiel

Auch an dieser Stelle gibt es zunächst ein einführendes Beispiel. Das folgende JDBC Progamm hat diesselbe Funktionalität wie das entsprechende ODBC Programm in Abschnitt 4.1.1.

Importieren des JDBC Packages, jedes JDBC-Programm muß dieses Package importieren

```
import java.sql.*;

class Beispiel {
```

Versuche den JDBC-Treiber von DB2 zu laden.

```
    static {
        try {
            Class.forName("COM.ibm.db2.jdbc.app.DB2Driver").newInstance();
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println(e);
        }
    }

    public static void main(String argv[]) {
```

Deklariere Verbindungsobjekt, setze die URL-Variable für die Datenbank und versuche eine Verbindung aufzubauen

```
        Connection con = null;
        String url = "jdbc:db2:unidb";
        try {
            con = DriverManager.getConnection(url);
        } catch (Exception e) {
            System.out.println("Error while connecting");
            e.printStackTrace();
```

```
            System.out.println(e);
        }

        System.out.println("connected to database");
```

Allokiere ein Statement-Objekt, führe Anfrage aus und weise Antwort einem Ergebnismengenobjekt zu.

```
        String query = "select matrnr from Studenten";
        try {
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery(query);
```

Hole Informationen über die Spalten der Antwort. Dazu muß eine Metadatenobjekt angelegt werden. Die Spalten sind von 1 bis n durchnumeriert (nicht von 0 bis $n-1$).

```
            ResultSetMetaData rsmd = rs.getMetaData();
            int noOfCols = rsmd.getColumnCount();

            System.out.println("number of columns " + noOfCols);

            System.out.println(rsmd.getColumnLabel(1));
            System.out.println("----------");
```

Hole solange Werte bis keine mehr gefunden werden und gib sie aus. In JDBC können alle Attributwerte in Form eines Strings abgerufen werden, was die reine Ausgabe erleichtert. Falls die Daten weiterverarbeitet werden müssen, können die Daten mit entsprechenden get-Funktionen auch in anderer Form geholt werden.

```
            while (rs.next()) {
                String a = rs.getString(1);
                System.out.println(a);
            }
```

Schließe Ergebnismengen- und Statementobjekt.

```
            rs.close();
            stmt.close();
```

Baue Verbindung ab.

```
            con.close();
        } catch( Exception e ) {
            e.printStackTrace();
            System.out.println(e);
        }
    }
}
```

4.2.2 Die wichtigsten JDBC-Klassen und -Methoden

Die folgende Übersicht stellt nur einen kleinen Aussschnitt aus der vollständigen Liste der Klassen vor. Den gesamten Überblick bekommt man auf der folgenden Seite (unter Package java.sql):

http://java.sun.com/j2se/1.4.2/docs/api/

Ebenfalls auf den Seiten von Sun ist ein kurzes Tutorial zu JDBC zu finden:

http://developer.java.sun.com/developer/onlineTraining/Database/JDBC20Intro/JDBC20.html

DriverManager stellt die Basisfunktionalität zur Verwaltung von JDBC-Treibern zur Verfügung. Folgende Methoden sind nützlich in diesem Zusammenhang:

`Class.forName()` lädt explizit einen bestimmten Treiber.

`getConnection()` ist zuständig für den Aufbau einer Verbindung zu einer Datenbank und existiert in zwei Varianten: mit Angabe von Benutzername und Passwort und ohne diese Angabe.

Connection ist für die Verwaltung einer Datenbankverbindung zuständig. Hier existieren z.B. folgende Methoden:

`createStatement()` erzeugt ein Objekt mit dessen Hilfe man SQL-Kommandos an die Datenbank schicken kann.

`close()` schließt die Verbindung.

`getMetaData` holt Schemainformation über die aktuelle Datenbank mit der man verbunden ist.

Außerdem existieren noch weitere Methoden zur Transaktionsverwaltung, wie Setzen des Isolation Levels, Einstellen von Autocommit, Committen und Abbrechen einer Transaktion.

Statement übernimmt die Verwaltung von Anfragen (Schicken an die Datenbank, usw.). Hier gibt es u.a. folgende Methoden:

`executeQuery()` führt eine Anfrage aus (select statement) aus.

`executeUpdate()` führt einen Update aus (insert, delete, update oder create statement).

`close()` gibt die Ressourcen eines Statementobjekts wieder frei.

Klasse ResultSet ist für die Repräsentation des Ergebnisses einer Anfrage zuständig. Mit den hier bereitgestellten Methoden kann auf das Ergebnis zugegriffen werden:

`getString()` holt den Wert eines Attributs in Form eines Strings (funktioniert mit jedem Typ).

`getX()` holt den (entsprechend typisierten) Wert eines Attributs. X steht dabei für den entsprechenden Datentyp (für einen kompletten Überblick siehe Webseite).

`getMetaData()` gibt eine Beschreibung eines Attributs der Ergebnismenge zurück.

`next()` holt das nächste Tupel aus der Ergebnisrelation.

`close()` gibt die belegten Ressourcen der Ergebnismenge wieder frei.

Klasse ResultSetMetaData verwaltet die Information über die Attribute einer Ergebnismenge. Wichtigste Methoden:

`getColumnCount()` gibt an, wieviele Spalten das Ergebnis hat.

`getColumnLabel()` liefert den vorgeschlagenen Titel eines Attributs (für die Überschrift der Spalte).

`getColumnType()` liefert den Typ eines Spalte zurück (für die Kodierung siehe Webseite).

4.3 Anbindung von Datenbanken an das Web

Die Integration von Websites und Datenbanken ist gerade im Bereich E-commerce von großer Bedeutung. Dort gibt es zahlreiche datenintensive Anwendungen, wie z.B. Online-Shops, Auktionshäuser, Webinformationssysteme und Online-Banking. Die Webseiten mit denen hier umgegangen wird sind hauptsächlich dynamische Seiten. Ihr Inhalt variiert je nach Benutzereingabe und oft müssen Daten von externen Quellen (wie z.B. Datenbanken) geholt serden. Wir bezeichnen solche Seiten als datenbankbasierte Webseiten. Im Gegensatz dazu existieren statische Webseiten, deren Inhalt bereits beim Erstellen festgelegt wird. Datenbankbasierte Webseiten sollten auch von Seiten unterschieden werden, die auf Clientseite mit Skriptsprachen wie JavaScript oder VBScript arbeiten, die lediglich eine animierte Darstellung von statischen Daten erlauben.

Datenbankbasierte Seiten können auf verschiedene Arten erzeugt werden, einmal durch serverseitige Verarbeitung, zum anderen durch clientseitige Verarbeitung (es ist natürlich auch möglich die beiden Ansätze zu mischen):

Serverseitige Verarbeitung: der Webserver bekommt eine Anforderung für eine Seite, führt die nötige Verarbeitung zur Erzeugung der Seite durch und schickt sie an den Client (zur Anzeige in einem Browser).

Clientseitige Verarbeitung: bei diesem Ansatz läuft ein Programm auf dem Rechner des Clients, das direkt mit der jeweiligen Datenbank kommuniziert

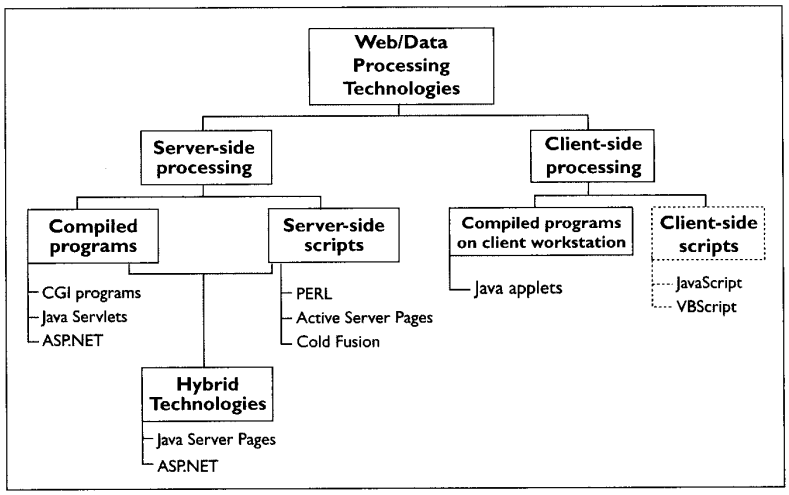

Abbildung 15: Überblick über verschiedene Verfahren

Abbildung 15 zeigt einen Überblick über die gebräuchlichsten Techniken. (Clientseitige Skripte sind gestrichelt markiert, um anzudeuten, daß diese Techniken nicht in der Lage sind, direkt mit einer Datenbank zu interagieren.) Wir werden im folgenden die verschiedenen Techniken ansprechen.

4.3.1 Serverseitige Verarbeitung

Vor der Verarbeitung auf dem Server werden häufig HTML-Formulare (HTML forms) eingesetzt, um die Eingaben des Benutzers einzusammeln und an den Server zu schicken. Auf dem Server läuft ein Verarbeitungsprogramm, das die Eingaben auswertet und die angeforderte Webseite erstellt.

Fertig übersetzte Serverprogramme (Compiled Programs) In diesem Fall liegt das Verarbeitungsprogramm in Form einer ausführbaren Datei vor. (Diese Datei ist unabhängig von anderen Dienstprogrammen des Webservers, wie z.B. einem Listener oder Administrationswerkzeugen.) Das Verarbeitungsprogramm holt sich die Eingaben, verarbeitet sie, legt das Ergebnis an einer bestimmten Stelle des Servers ab und beendet sich. Der Webserver liest das Ergebnis aus und schickt es an den Client.

Java, Visual Basic und C++ sind übliche Sprachen für fertig übersetzte Verarbeitungsprogramme. Es kann aber jede Sprache verwendet werden die über eine Anbindung an einen Webserver verfügt, d.h. die Funktionen bereitstellt, um mit einem Webserver zu kommunizieren. 1993 wurde das erste Protokoll verabschiedet, daß eine auf HTML-Formularen basierende Schnittstelle zwischen Webserver und Verarbeitungsprogrammen definierte: Common Gateway Interface (CGI). Es wird heutzutage noch vielfach eingesetzt, hat aber einen entscheidenden Nachteil. Jedes HTML-Formular, das an den Server geschickt wird, startet eine Kopie des Programms als eigenständigen Prozeß. Bei großem Anfragevolumen kann es passieren, daß dem Webserver der Speicher ausgeht.

Aus diesem Grund wurden Methoden entwickelt, die mit einer einzigen Kopie des Verarbeitungsprogramms (mit Hilfe mehrerer Threads) mehrere Benutzer gleichzeitig bedienen können. Prominente Vertreter sind Servlets und (eine Form der) Application Server Pages (ASP.NET). Es existieren noch weitere Vorteile gegenüber CGI-Programmen: eine direkte Kommunikation mit dem Webserver ist möglich, Daten können gemeinsam genutzt werden (data sharing) und Mechanismen zum Session Tracking sind verfügbar (wie z.B. Verwaltung von Cookies).

Serverseitige Skripte Die Architektur zur Anfrageverarbeitung ist die gleiche wie bei übersetzten Programmen. Der einzige Unterschied ist, daß auf dem Server Skripte interpretiert statt fertig übersetzte Programme ausgeführt werden. Skripte die das CGI-Protokoll nutzen werden häufig in PERL geschrieben (Practical Extraction and Report Language). PERL stellt ein großes Repertoire an Befehlen zur Verarbeitung von Zeichenketten zur Verfügung, verleitet aber durch die Knappheit der Ausdrücke zu einem schwer verständlichen Programmierstil. Abbildung 16 zeigt ein Beispiel für ein PERL-Skript, das mit einem HTML-Formular interagiert. Auch hier gibt es das Problem mit CGI, daß für jede Anfrage eine eigene Kopie des Skripts (samt zugehörigem Interpreter) gestartet wird.

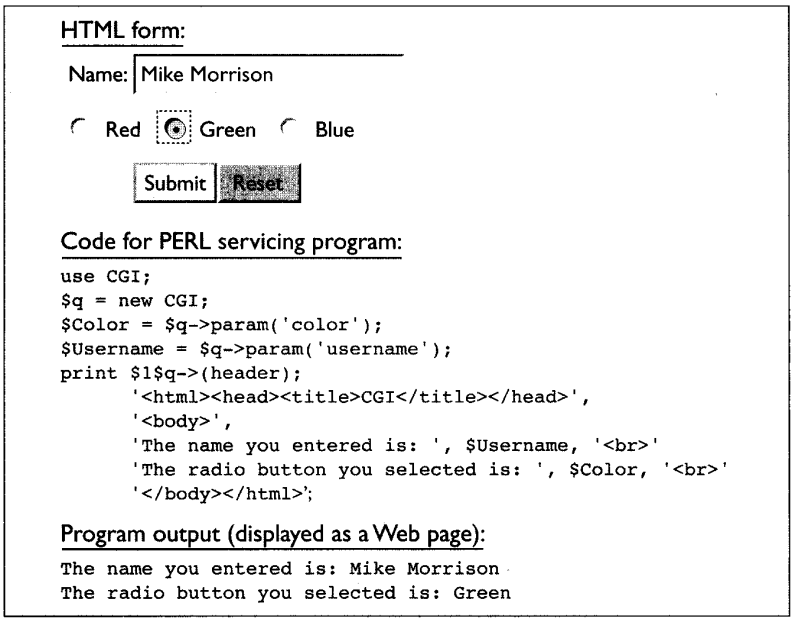

Abbildung 16: Beispiel für ein PERL Skript

Es wurden eine Reihe weiterer Technologien entwickelt, die dieses Problem beheben. Namhafte Vertreter sind Cold Fusion, Active Server Pages (ASP) und PHP:Hypertext Preprocessor (PHP). Hier kann ein einziges laufendes Skript mehrere Anfragen bearbeiten. Außerdem wird ein anderer Ansatz als z.B. in PERL verfolgt. Anstatt den HTML-Code in die Ausgabeanweisungen des Skripts zu integrieren, wird Code in HTML-Seiten eingebettet. Das erleichtert die Erstellung von Seiten, da die Struktur der Seite nicht in

Ausgabeanweisungen über das ganze Skript verteilt und versteckt ist. Abbildung 17 zeigen Beispiele für Cold Fusion, ASP und PHP.

```
Cold Fusion Example
<HTML><HEAD><TITLE>Cold Fusion</TITLE></HEAD><BODY>
The name you entered is: #form.username#
The radio button you selected is: #form.color#
</BODY></HTML>

ASP Example
<HTML><HEAD><TITLE>ASP</TITLE></HEAD><BODY>
The name you entered is: <%=request.querystring('username') %>
The radio button you selected is: <%=request.querystring('color') %>
</BODY></HTML>

PHP Example
<HTML><HEAD><TITLE>PHP</TITLE></HEAD><BODY>
The name you entered is: <?php $username; ?>
The radio button you selected is: <?php $color; ?>
</BODY></HTML>
```

Abbildung 17: Beispiele für Cold Fusion, ASP und PHP

Hybride serverseitige Verarbeitung Fertig übersetzte Programme haben zwei Vorteile. Da sie schon in maschinenlesbarem Format abgelegt sind, laufen sie normalerweise schneller als Skripte. Außerdem werden sie (normalerweise) in Entwicklungsumgebungen mit Debuggingwerkzeugen geschrieben, die das Auffinden von Fehlern erleichtern. Der Vorteil von Skripten liegt darin, daß sie schnell und unkompliziert mit einem einfachen Texteditor modifiziert werden können.

Bei den hybriden Ansätzen versucht man die Vorteile von übersetzten Programmen und Skripten zu kombininieren. Die Verarbeitungsprogramme werden wie Skripte geschrieben, aber zunächst nicht übersetzt. Das erste Mal, wenn ein Nutzer auf das Programm zugreift, wird es übersetzt und für spätere Aufrufe abgespeichert. Nach einer Änderung am Programm muß bei einem Aufruf entsprechend neu übersetzt werden.

```
JSP Example
<%@ page import="login.inputHandler" %>
<jsp:useBean id="abean" scope="page" class="login.inputHandler" />
<HTML><HEAD><TITLE>CG13</TITLE></HEAD><BODY>
The name you entered is:
<jsp:getProperty name="abean" property="username"/>
The radio button you selected is:
<jsp:getProperty name="abean" property="color"/>
</BODY></HTML>
```

Abbildung 18: Beispiel für JSP

Bekannte Technologien hier sind Java Server Pages (JSP) und ASP.NET (das auch in einem hybriden Modus betrieben werden kann). Für ein JSP-Beispiel siehe Abbildung 18.

Wie wählt man aus? Da übersetzte Programm schneller als Skripte sind, sollten stark frequentierte Webserver diese Variante vorziehen. Die Stärke von Skripten ist bei kleineren Anwendungen zu sehen, da ihre Erstellung nicht so zeitaufwendig ist.

CGI hat insofern noch eine Daseinsberechtigung da es auf vielen Betriebssystemen unter vielen verschiedenen Programmiersprachen unterstützt wird. Auch in bezug auf die Performanceprobleme hat sich einiges getan. So vermeiden die Ansätze PersistentCGI und FastCGI das Starten von Kopien eines Programms bei der Abarbeitung einer Anfrage.

ASP.NET bietet umfassende Möglichkeiten bei der Anfrageverarbeitung, ist aber um einiges komplexer und dadurch schwerer zu erlernen.

4.3.2 Clientseitige Verarbeitung

Clientseitige Verarbeitung kann prinzipiell auf zweierlei Arten realisiert werden: 1) der Client lädt ein fertig übersetztes Programm vom Server herunter, installiert es und führt es aus, 2) in den HTML-Code werden Skripte eingebettet, die vom Browser des Client ausgeführt werden.

Ausführung übersetzter Programme Bei dieser Variante muß der Browser des Client in der Lage sein ein übersetztes Programm auszuführen. Dieses Programm interagiert mit dem Nutzer und schickt und empfängt Daten an/von einem Datenbankserver. Die bekannteste Technik in diesem Bereich sind Java Applets. Java Applets sind Programme, die in der Laufzeitumgebung eines Browsers laufen. Aus Sicherheitsgründen können Applets nur Daten mit einem Webserver austauschen, der Zugriff auf lokale Daten des Benutzers bleibt ihnen verwehrt.

Die Absicht von Java Applets war es, einen plattformunabhängigen Mechanismus zur Verfügung zu stellen. Die Inkonsistenzen bei der Implementierung der Laufzeitumgebungen in Browsern machen es allerdings schwierig, Applets zu entwickeln, die überall das gleiche Verhalten zeigen.

Clientseitige Skripte Es ist auch möglich, Skripte in HTML-Code einzubetten, die auf der Clientseite ausgeführt werden. Obwohl clientseitige Skripte nicht dazu benutzt werden können, um von außerhalb auf einen Datenbankserver zuzugreifen, können sie dennoch die Eingaben prüfen oder bieten erweiterte Darstellungsmöglichkeiten.

JavaScript ist die gebräuchlichste Sprache, um clientseitige Skripte zu schreiben. (In der Microsoftwelt existiert VBScript, welches aber nicht auf allen Browsern lauffähig ist.)

4.4 Versuch3: Dynamic SQL über das Web

In diesem Versuch wird die Terra-Datenbank mit Hilfe von Dynamic SQL und einer Webschnittstelle im Netz zugänglich gemacht. Dies wird über JDBC und Servlets realisiert (alternativ kann auch die Variante ueber ODBC und CGI implementiert werden). Bevor wir mit der Versuchsbeschreibung beginnen, gibt es noch eine kurz Einführung in HTTP, Servlets und CGI. (Außerdem werden wieder einige Beispielprogramme zur Verfügung gestellt.)

4.4.1 HTTP

Das HyperText Transfer Protocol (HTTP) ist das am weitesten verbreitete Kommunikationsprotokoll für das Web. HTTP ist zustandslos und folgt einem einfachen Modell: ein Client (wie z.B. ein Webbrowser) schickt eine Anfrage (request) an einen Webserver und dieser antwortet darauf (response).

Request Ein Request besteht aus folgenden Elementen:

- einer HTTP-Methode (wie z.B. GET oder POST) die dem Webserver sagt, was gemacht werden soll
- einer URL (Uniform Resource Locator), die angibt welches Dokument gewünscht wird
- einer Versionsnummer
- einer Vielzahl weiterer Informationen (auf die hier aber nicht eingegangen werden soll)

Ein Beispiel für eine mögliche Anfrage ist hier zu sehen:

```
GET /intro.html HTTP/1.0
User-Agent: Mozilla/4.0 (compatible; NETSCAPE 4.0; Windows 95)
Accept: image/gif, image/jpeg, text/*, */*
```

Response Je nachdem, ob eine Anfrage erfolgreich bearbeitet werden konnte, werden Daten oder eine Fehlermeldung zurückgeschickt. Vor den eigentlichen Daten steht ein Kopf (Header) mit Informationen wie Status, Beschreibung des Typs der geschickten Daten, Datenmenge, usw. Auch hier wieder ein Beispiel:

```
HTTP/1.0 200 OK
Date: Saturday, 23-May-98 03:25:12 GMT
Server: JavaWebServer/1.1.1
MIME-Version: 1.0
Content-type: text/html
Content-length: 1029
Last-modified: Thursday, 7-May-98 12:15:35 GMT
...
```

Methoden Die zwei gebräuchlichsten Methoden sind GET und POST. Mit GET werden Daten vom Server angefordert, mit POST Daten zum Server geschickt. Diese zwei Methoden können mit Parametern aufgerufen werden, die zusätzliche Informationen an den Webserver übermitteln. HTML-Formulare (HTML forms) sind z.B. eine Art Parameter zu übermitteln. Die Parameter werden dabei mit in die URL kodiert. Das folgende Beispiel zeigt den HTML-Code für das Formular aus Abbildung 16:

```
<html> <head>
<title>Example</title>
</head>
<h1>HTML form</h1>

<form method=GET action=http://URL/cgi-bin/skript.perl>
<table>
<tr><td>
Name: <input type=text name=name>
</td></tr>
<tr><td>
<input type=radio name=color value=red> Red
<input type=radio name=color value=green> Green
<input type=radio name=color value=blue> Blue
</td></tr>
<tr><td>
<input type=submit>
<input type=reset>
</td></tr>
</table>
</form>

</body>
</html>
```

Die vollständige URL, die beim Abschicken des Formulars aus Abbildung 16 angegeben wird, sieht so aus (Leerzeichen werden durch + ersetzt, da sie in URLs nicht erlaubt sind):

```
http://URL/cgi-bin/skript.perl?name=Mike+Morrison&color=green
```

4.4.2 Servlets

Ein Servlet ist eine Javaklasse, die von einem Webserver dynamisch geladen und ausgeführt werden kann. Auf diese Weise kann die Funktionalität eines Servers beliebig erweitert werden.

Servlets haben keine main-Methode, sind also keine eigenständigen Applikationen, sondern werden vom Server aus angesteuert. Die wichtigsten Methoden sind die `init()`-, die `service()`- und die `destroy()`-Methode. Die `init()`-Methode wird vor der Bearbeitung der ersten Anfrage ausgeführt und initialisiert das Servlet. Die `service()`-Methode nimmt eine Anfrage an, bearbeitet sie und schickt ein Ergebnis zurück. Diese Methode kann vom Webserver beliebig oft aufgerufen werden. Die `destroy()`-Methode wird beim Entladen des Servlets aufgerufen und gibt angeforderte Ressourcen wieder frei.

Im Zusammenhang mit Servlets gibt es zwei wichtige Packages, nämlich `javax.servlet` und `javax.servlet.http`:

- `javax.servlet` enthält die allgemeinen Interfaces und Klassen für generische Servlets
- `javax.servlet.http` ist für die Entwicklung von Servlets vorgesehen, die HTTP benutzen

Wir werden HTTP-Servlets benutzen und dabei sind zwei Methoden besonders wichtig (neben der `init()`- und der `destroy()`-Methode), nämlich die `doGet()`- und die `doPost()`-Methoden. Diese beiden Methoden werden zur Abarbeitung der HTTP-Methoden GET und POST benötigt.

Details zu den Servlet-Klassen sind auf folgender Webseite zu finden:

http://java.sun.com/products/servlet/2.2/javadoc/

4.4.3 CGI

Die zwei wichtigsten Fragen im Zusammenhang mit dem Schreiben eines CGI-Programms sind: wie bekomme ich die Informationen vom Server und wie schicke ich meine Daten zum Client?

Daten einsammeln Die Daten werden nicht über Kommandozeilenparameter (command line options) an das Programm übergeben (also z.B. mit argv in C). Vielmehr setzt der Webserver Umgebungsvariablen (environment variables) vor dem Aufrufen des CGI-Programms. Diese müssen dann vom Programm ausgelesen werden (in C z.B. mit der Funktion `getenv()`). Die wichtigste Umgebungsvariable ist `QUERY_STRING`. Dort wird alles was in der Aufruf-URL hinter dem Fragezeichen steht abgelegt. Die Leerzeichen werden durch + ersetzt, die Sonderzeichen durch % gefolgt von ihrem Hexadezimalcode.

Daten zurückschicken Das Zurückschicken von Daten ist wesentlich einfacher. Alles was vom Programm über die Standard-Ausgabe (stdout) ausgegeben wird, wird an den Client geschickt. Man muß nur darauf achten, daß die Antwort auch verstanden wird. Dafür muß ein kurzer Kopf (Header) angegeben werden, in dem vermerkt ist, in welchem Format die Antwort sein wird. Bei HTML sieht der Kopf so aus:

```
Content-type: text/html

<HTML><HEAD>
...
```

(Achtung: Die Leerzeile zwischen dem Kopf und den eigentlichen Daten ist nicht optional!)

4.4.4 Laufenlassen der einführenden Beispiele

Für diesen Versuch gibt es mehrere Beispielprogramme und zwar jeweils zwei für reines Dynamic SQL über die Standard-Ein-/Ausgabe (einmal C und ODBC und einmal Java und JDBC) und zwei für eine kleine Webanbindung (einmal ein Servlet und einmal ein CGI-Programm). Für die Webanbindung gibt es außerdem noch zwei HTML-Formulare. Ihre erste Aufgabe besteht darin, diese Beispielprogramme zum Laufen zu bringen.

4.4.5 Eine kleine Applikation

1. Implementieren Sie eine Webanfrageschnittstelle für DB2 (ähnlich der Schnittstelle die für die Vorlesung DBS I eingesetzt wird).

2. Falls noch nicht geschehen, erweitern Sie die Schnittstelle so, daß auch Insert-, Update- und Delete-Ausdrücke verarbeitet werden können. Geben Sie dabei als Ausgabe die Anzahl der veränderten bzw. eingefügten/gelöschten Tupel an.