Kapitel 9

Datenbankapplikationen

9.1 ODBC

Open Database Connectivity (ODBC) ist eine von Microsoft entwickelte standardisierte Anwendungs-Programmierungs-Schnittstelle (Application Programming Interface, API), über die Anwendungen auf Datenbanken unterschiedlicher Hersteller zugreifen können (Oracle, Informix, Microsoft, MySQL, ...). Hierzu wird auf dem Rechner des Anwenders unter Verwendung des für den jeweiligen Hersteller vorgeschriebenen Treibers eine sogenannte Datenquelle installiert, auf die dann das Anwendungsprogramm zugreifen kann, ohne die proprietären Netzwerkkommandos zu beherrschen.

Abbildung 9.1 zeigt das Hinzufügen einer ODBC-Datenquelle in der Systemsteuerung von Windows.

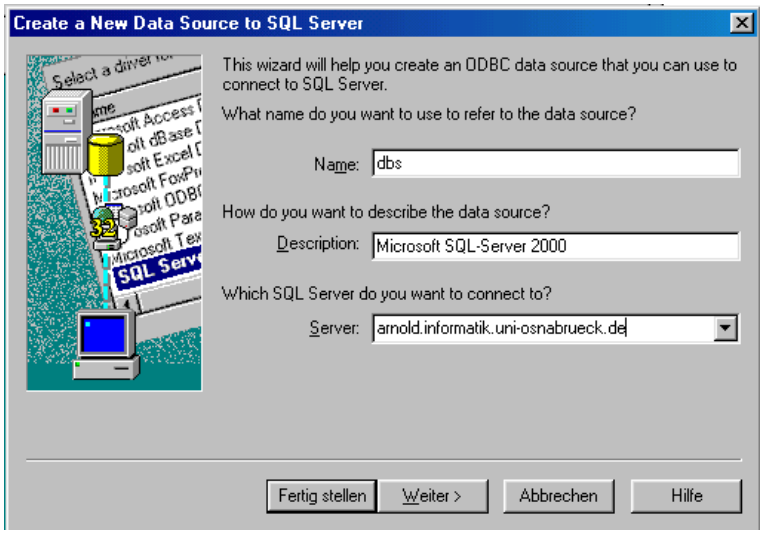


Abbildung 9.1: Hinzufügen einer ODBC-Datenquelle

9.2 Microsoft Visio

Microsoft Visio ist ein Vektorgrafikprogramm zum Erstellen von Grafiken mit vorgefertigten Schablonen. Über den Menupunkt *Database Reverse Engineering* kann das Datenbankschema einer ODBC-fähigen Datenbank eingelesen und grafisch aufbereitet werden (Abbildung 9.2).

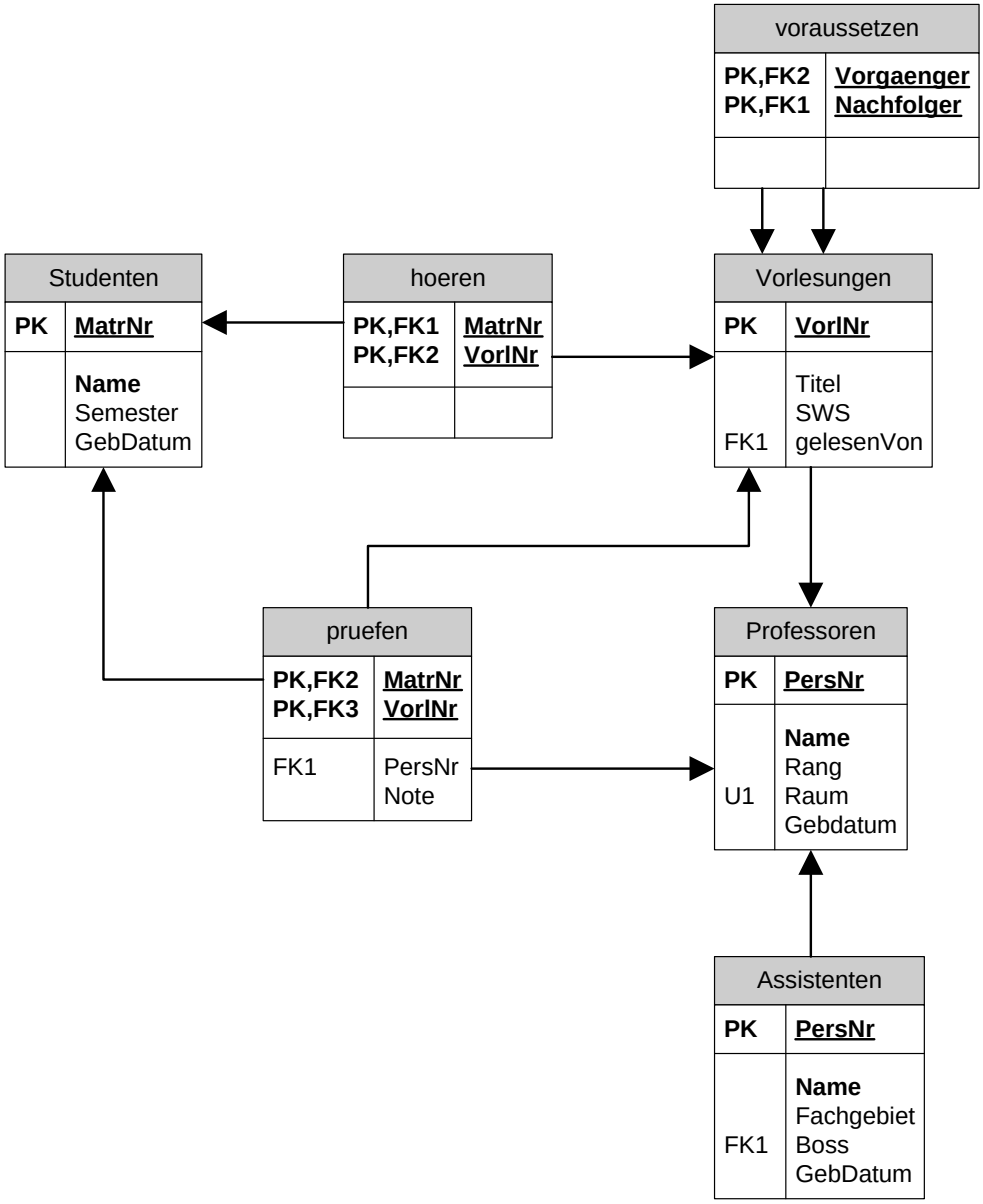


Abbildung 9.2: Universitätsschema, erzeugt von MS Visio

9.3 Microsoft Access

Access ist ein relationales Datenbanksystem der Firma *Microsoft*, welches als Einzel- und Mehrplatzsystem unter dem Betriebssystem Windows läuft. Seine wichtigsten Eigenschaften:

- Als **Frontend** eignet es sich für relationale Datenbanksysteme, die per ODBC-Schnittstelle angesprochen werden können. Hierzu wird in der Windows-Systemsteuerung der zum Microsoft-SQL-Server mitgelieferte ODBC (Open Data Base Connectivity) Treiber eingerichtet und eine User Data Source hinzugefügt, z.B. mit dem Namen `dbs`. Nun können die auf dem SQL-Server liegenden Tabellen verknüpft und manipuliert werden.
- Der **Schemaentwurf** geschieht menugesteuert i (Abbildung 9.3)
- Referenzen zwischen den Tabellen werden in Form von **Beziehungen** visualisiert.
- **Formulare** definieren Eingabemasken, die das Erfassen und Updaten von Tabellendaten vereinfachen (Abbildung 9.4).
- **Queries** können per SQL oder menugesteuert abgesetzt werden (Abbildung 9.5).
- **Berichte** fassen Tabelleninhalte und Query-Antworten in formatierter Form zusammen und können als Rich-Text-Format exportiert werden (Listing 9.1 + Abbildung 9.5).

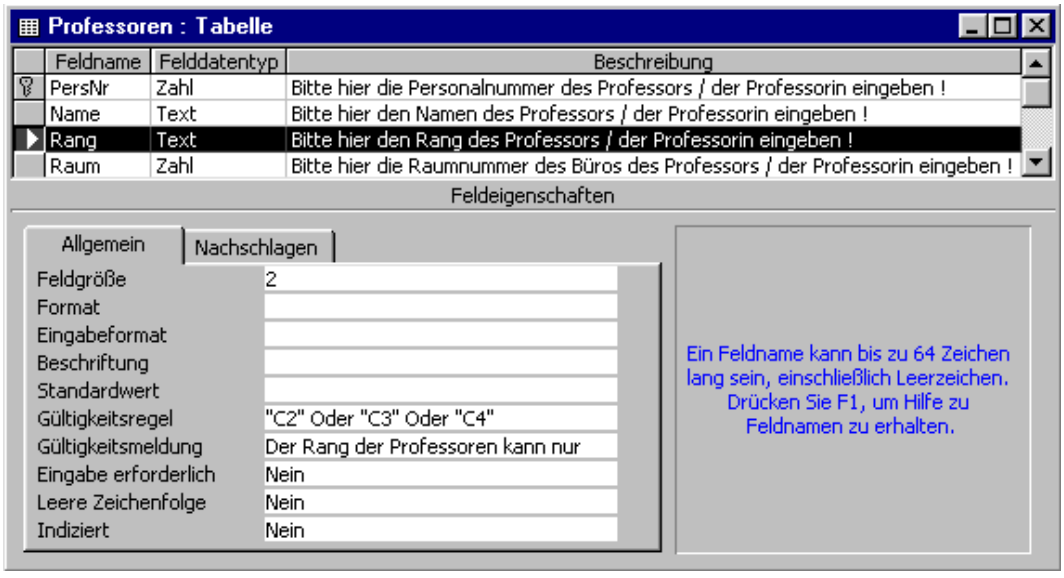


Abbildung 9.3: Schemadefinition in Microsoft Access

Listing 9.1 zeigt die Formulierung einer SQL-Abfrage, die zu jedem Professor seine Studenten ermittelt. Aus den Treffern dieser Query wird der Bericht in Abbildung 9.6 generiert.

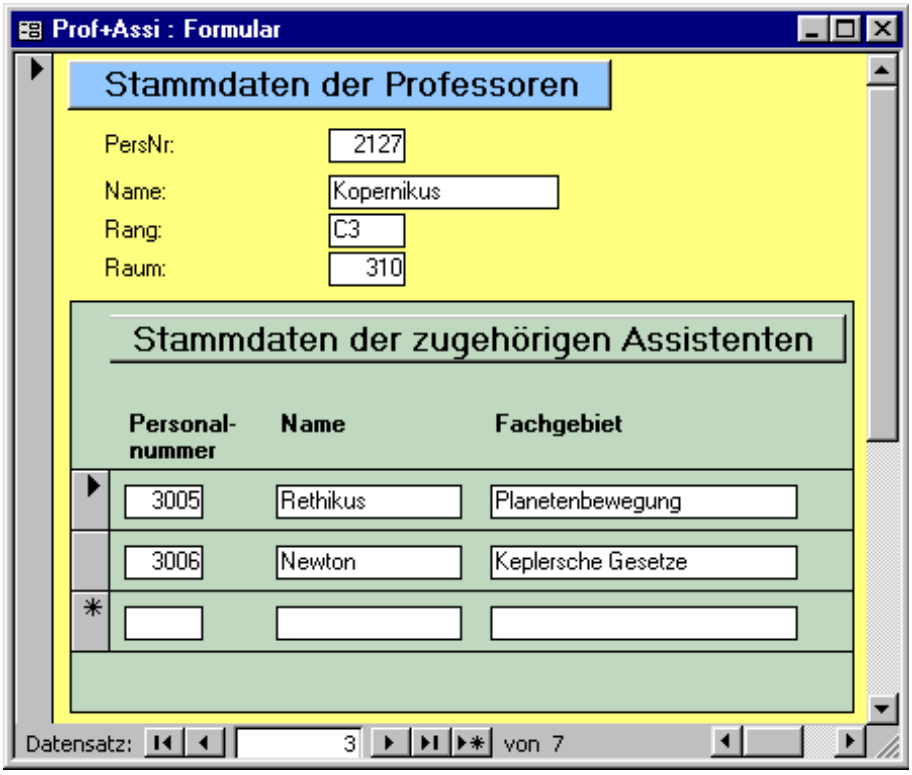


Abbildung 9.4: durch Microsoft Access-Formular erzeugte Eingabemaske

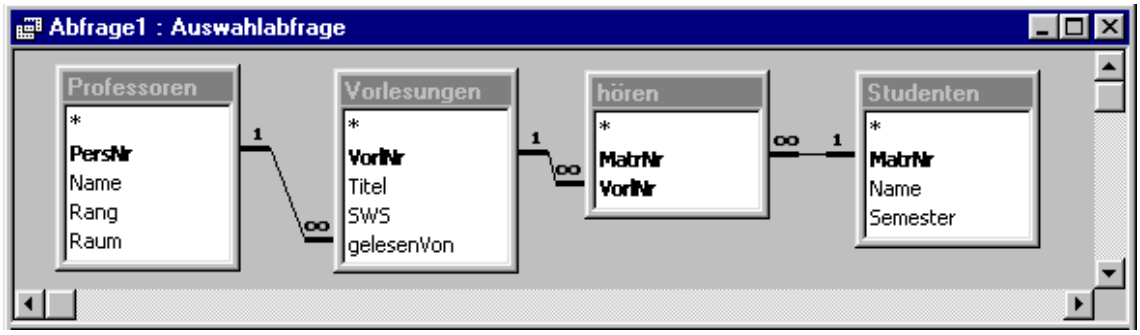


Abbildung 9.5: in Microsoft Access formulierte Abfrage

```
SELECT DISTINCT p.name AS Professor, s.name AS Student
FROM  professoren p, vorlesungen v, hoeren h, studenten s
WHERE v.gelesenvon = p.persnr
and   h.vorlnr     = v.vorlnr
and   h.matrnr     = s.matrnr
ORDER BY p.name, s.name
```

Listing 9.1: Abfrage für Bericht in Abbildung 9.6

Abbildung 9.6: Word-Ausgabe eines Microsoft Access-Berichts, basierend auf Listing 9.1

9.4 Embedded SQL

Unter *Embedded SQL* versteht man die Einbettung von SQL-Befehlen innerhalb eines Anwendungsprogramms. Das Anwendungsprogramm heißt *Host Programm*, die in ihm verwendete Sprache heißt *Host-Sprache*.

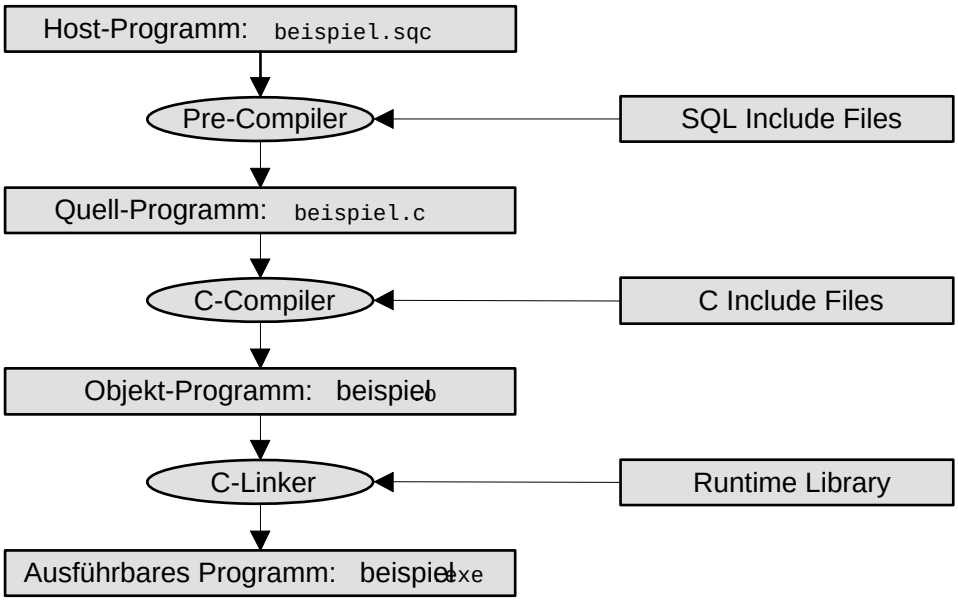


Abbildung 9.7: Vom Hostprogramm zum EXE-File

Der *Microsoft SQL-Server* unterstützt Embedded SQL im Zusammenspiel mit den Programmiersprachen C und C++ durch einen *Pre-Compiler*. Abbildung 9.7 zeigt den prinzipiellen Ablauf: Das mit eingebetteten SQL-Statements formulierte Host-Programm `beispiel.sqc` wird zunächst durch den

Pre-Compiler unter Verwendung von SQL-spezifischen Include-Dateien in ein ANSI-C-Programm `beispiel.c` überführt. Diese Datei übersetzt ein konventioneller C-Compiler unter Verwendung der üblichen C-Include-Files in ein Objekt-File `beispiel.o`. Unter Verwendung der Runtime Library wird daraus das ausführbare Programm `beispiel.exe` gebunden.

Eingebettete SQL-Statements werden durch ein vorangestelltes `EXEC SQL` gekennzeichnet und ähneln ansonsten ihrem interaktiven Gegenstück.

Die Kommunikation zwischen dem Host-Programm und der Datenbank geschieht über sogenannte *Host-Variablen*, die im C-Programm deklariert werden. Eine *Input-Host-Variable* überträgt Daten des Hostprogramms an die Datenbank, eine *Output-Host-Variable* überträgt Datenbankwerte und Statusinformationen an das Host-Programm. Hostvariablen werden innerhalb von SQL-Statements durch einen vorangestellten Doppelpunkt (:) gekennzeichnet.

Für Hostvariablen, die Datenbankattributen vom Typ VARCHAR entsprechen, empfiehlt sich eine Definition nach folgendem Beispiel:

```
char fachgebiet[20];
```

Mit einer Hostvariable kann eine optionale *Indikator-Variable* assoziiert sein, welche Null-Werte überträgt oder entdeckt. Folgende Zeilen definieren alle Hostvariablen zum Aufnehmen eines Datensatzes der Tabelle *Professoren* sowie eine Indikator-Variable `raum_ind` zum Aufnehmen von Status-Information zur Raumangabe.

```
int   persnr;
char  name [20];
char  rang [3];
int   raum;
short raum_ind;
```

Folgende Zeilen transferieren einen einzelnen Professoren-Datensatz in die Hostvariablen `persnr, name, rang, raum` und überprüfen mit Hilfe der Indikator-Variable `raum_ind`, ob eine Raumangabe vorhanden war.

```
EXEC SQL SELECT PersNr, Name, Rang, Raum
         INTO   :persnr, :name, :rang, :raum INDICATOR :raum_ind
         FROM   Professoren
         WHERE  PersNr = 2125;
if (raum_ind == -1) printf("PersNr %d ohne Raum \n", persnr);
```

Oft liefert eine SQL-Query kein skalares Objekt zurück, sondern eine Menge von Zeilen. Diese Treffer werden in einer sogenannten *private SQL area* verwaltet und mit Hilfe eines *Cursors* sequentiell verarbeitet.

```
EXEC SQL DECLARE C1 CURSOR FOR
  SELECT PersNr, Name, Rang, Raum
  FROM   Professoren
  WHERE  Rang='C4';
```

```
    EXEC SQL OPEN C1;
    EXEC SQL FETCH C1 into :persnr, :name, :rang, :raum

    while (SQLCODE ==0){
      printf("Verarbeite Personalnummer %d\n", persnr);
      EXEC SQL FETCH C1 into :persnr, :name, :rang, :raum
    }

    EXEC SQL CLOSE C1;
```

Listing 9.2 zeigt ein Embedded-SQL-Programm, die davon erzeugte Ausgabe zeigt Abb. 9.8.

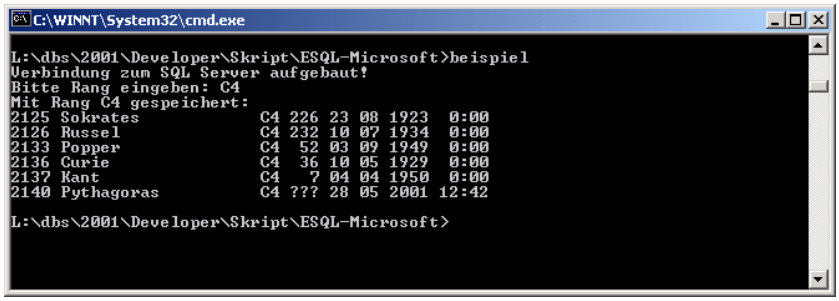


Abbildung 9.8: Ausgabe des Embedded-SQL-Programms von Listing 9.2

```
void ErrorHandler (void);

#include <stddef.h>                                    // Standardheader
#include <stdio.h>                                     // Standardheader

int main ( int argc, char** argv, char** envp) {
    EXEC SQL BEGIN DECLARE SECTION;                    // Deklarationen-Start

    char serverDatenbank[] = "arnold.uni";             // Server + DB
    char loginPasswort[] = "erika.mustermann";         // User + Passwort
                                                       // Hostvariablen
    int    persnr;                                     // Personalnummer
    char   name[20];                                   // Name
    char   rang[3];                                    // Rang
    int    raum;                                       // Raum
    char   gebdatum[17];                               // Geburtsdatum
    short  raum_ind;                                   // Raum-Indikator

    char eingaberang[3];                               // Eingabe vom User

    EXEC SQL END DECLARE SECTION;                      // Deklarationen-Ende
    EXEC SQL WHENEVER SQLERROR CALL ErrorHandler();    // Fehlermarke
    EXEC SQL CONNECT TO :serverDatenbank               // Verbindung aufbauen
                  USER :loginPasswort;

    if (SQLCODE == 0)                                  // bei Erfolg
```

```
        printf("Verbindung zum SQL Server aufgebaut!\n");
    else                                                // bei Misserfolg
    {
        printf("Fehler: Keine Verbindung zum SQL Server!\n");
        return (1);
    }

    printf("Bitte Rang eingeben: ");                    // gewuenschten Rang
    scanf("%s", eingaberang);                           // vom user holen
    printf("Mit Rang %s gespeichert:\n", eingaberang);

    EXEC SQL DECLARE C1 CURSOR FOR                      // Cursor vereinbaren
      SELECT PersNr, Name, Rang, Raum, Gebdatum         // SQL-Statement
      FROM Professoren
      WHERE Rang = :eingaberang;

    EXEC SQL OPEN C1;                                   // Cursor oeffnen
    EXEC SQL FETCH C1 INTO :persnr, :name, :rang,       // Versuche eine Zeile
             :raum INDICATOR :raum_ind, :gebdatum;      // zu lesen

    while (SQLCODE == 0)                                // SOLANGE erfolgreich
    {
      printf("%d %s %s", persnr, name, rang);           // Tupel ausgeben

      if(raum_ind == -1)                                // FALLS keine Raumnr
        printf(" ???");                                 // Platzhalter drucken
      else
        printf("%4d", raum);                            // SONST Raumnr

      printf(" %s\n", gebdatum);                        // letztes Attribut

      EXEC SQL FETCH C1 INTO :persnr, :name, :rang,     // naechste Zeile lesen
        :raum:raum_ind, :gebdatum;
    }

    EXEC SQL CLOSE C1;                                  // Cursor schliessen
    EXEC SQL DISCONNECT ALL;                            // Verbindung beenden
    return (0);
}

void ErrorHandler (void)
{
    printf("In Error Handler:\n");
    printf("    SQL Code = %li\n", SQLCODE);
    printf("    SQL Server Message %li: '%Fs'\n", SQLERRD1, SQLERRMC);
}
```

Listing 9.2: Quelltext von Embedded-SQL-Programm

9.5 JDBC

JDBC (Java Database Connectivity) ist ein Java-API (Application Programming Interface) zur Ausführung von SQL-Anweisungen innerhalb von Java-Applikationen und Java-Applets. Es besteht aus einer Menge von Klassen und Schnittstellen, die in der Programmiersprache Java geschrieben sind.

Ein JDBC-Programm läuft in drei Phasen ab:

1. Treiber laden und Verbindung zur Datenbank aufbauen,
2. SQL-Anweisungen absenden,
3. Ergebnisse verarbeiten.

Der folgende Quelltext zeigt ein einfaches Beispiel für diese Schritte:

```
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");           // mit Treiber
Connection con = DriverManager.getConnection              // Verbindung
                 ("jdbc:odbc:dbs","erika","mustermann");  // herstellen

Statement stmt= con.createStatement();                    // Query
ResultSet rs  = stmt.executeQuery                         // ausfuehren
                ("select * from Professoren");

while (rs.next()){                                        // Ergebnisse
  int x    = rs.getInt("persnr");                         // verarbeiten
  String s = rs.getString("name");
  System.out.println("Professor "+s+" hat die Personalnummer "+x);
}
```

Abbildung 9.9 zeigt die von Listing 9.3 erzeugte Ausgabe einer Java-Applikation auf der Konsole.

Abbildung 9.9: Ausgabe einer Java-Applikation

```
/* ShowJdbc.java (c) 1999 Stefan Rauch, 2001 Olaf Mueller */

import java.sql.*;                                      // Import der
                                                        // SQL-Klassen
public class ShowJdbc {

  public static void main(String args[]) {

    String url    = "jdbc:odbc:dbs";                    // Treiber-url
    String user   = "Erika";                            // User-Login
    String passwd = "Mustermann";                       // User-Passwort
    String query  = "select * from Professoren";        // User-Query

    Connection con;                                     // Verbindungs-Objekt
    Statement stmt;                                     // Query-Objekt

    try {                                               //versuche
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");    //jdbc-
    }catch(java.lang.ClassNotFoundException e) {        //odbc-
      System.err.print("ClassNotFoundException: ");     //Brueckentreiber
      System.err.println(e.getMessage());
    }

    try {
      con = DriverManager.getConnection(url,user,passwd); // Verbindung
      stmt = con.createStatement();                       // Statement
      ResultSet rs = stmt.executeQuery(query);            // ResultSet

      System.out.println
      ("Ausgabe der Professoren mit jeweiligem Rang: \n");

      while(rs.next()) {                            // Zeilenweise durch
        System.out.print("Der Professor ");         // Ergebnismenge laufen
        System.out.print(rs.getString("Name"));     // dabei Namen und Rang
        System.out.print(" hat den Rang ");         // formatiert ausgeben
        System.out.println(rs.getString("Rang"));
      }
      stmt.close();                                 // Statement schliessen
      con.close();                                  // Verbindung schliessen

    }catch(SQLException ex) {
      System.err.println("SQLException: " + ex.getMessage());
    }
  }
}
```

Listing 9.3: Quelltext der Java-Applikation ShowJdbc.java

Listing 9.4 zeigt den Quelltext einer HTML-Seite mit dem Aufruf eines Java-Applets. Abbildung 9.10 zeigt die Oberfläche des Applets. Listing 9.5 zeigt den Quelltext der im Applet verwendeten Javaklassen. Damit der Applet-Code Kontakt zur Datenbank aufnehmen kann, muß auf dem Rechner, auf dem der Browser läuft, der ODBC-Treiber samt Datenquelle eingerichtet sein. Daher eignet sich diese Lösung nur für Intra-Netze mit zentraler Systemadministration. Ferner darf das Applet nur eine Socket-Verbindung aufbauen zu Rechnern, von denen es gezogen wurde. Daher müssen Web-Server und Datenbankserver auf demselben Rechner liegen.

```
<HTML>
  <HEAD> <TITLE>JDBC Test-Applet</TITLE></HEAD>
  <BODY bgColor=Silver>
    <CENTER>
      <H2> Demo-Applet fr JDBC-Datenbankzugriff</H2>
      <APPLET
        codebase =.
        code     =JdbcApplet
        width    =700
        height   =400>
      </APPLET>
    </CENTER>
  </BODY>
</HTML>
```

Listing 9.4: Quelltext einer HTML-Seite zum Aufruf eines Applets

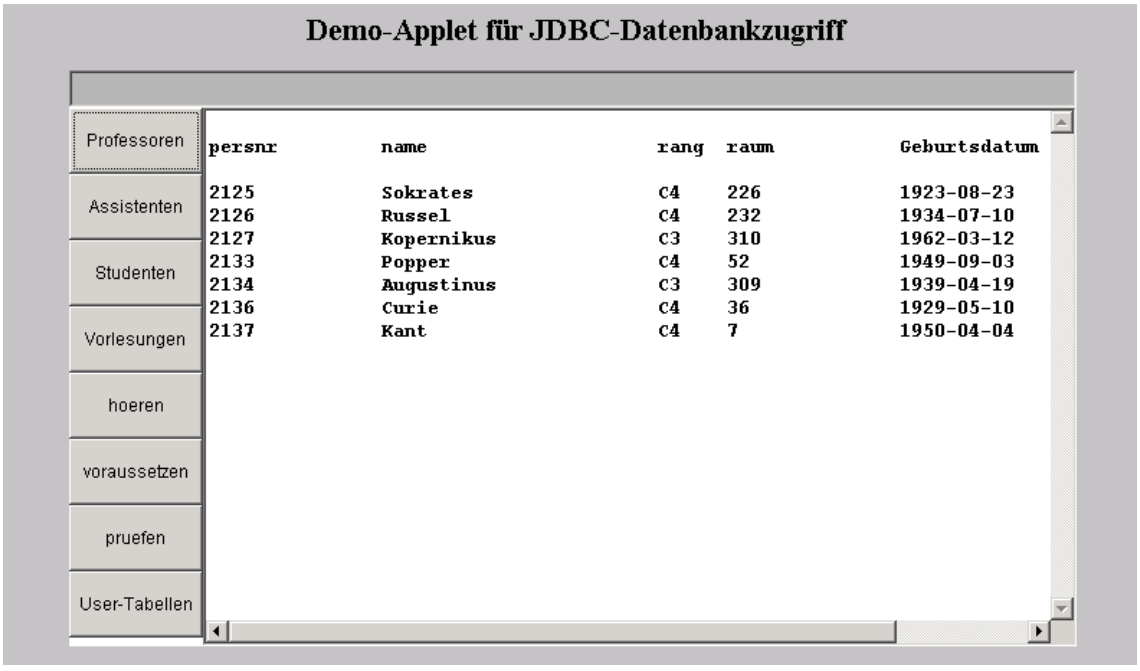


Abbildung 9.10: Java-Applet mit JDBC-Zugriff auf SQL-Server-Datenbank

```
/* JdbcApplet (c) Stefan Rauch 1999          */
/* Applet, das den Umgang mit dem JDBC zeigt */


import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.sql.*;


public class JdbcApplet extends Applet {


  BorderLayout layout = new BorderLayout();
  TextArea outputArea = new TextArea();
  TextField inputField = new TextField();
  Panel p;

  Button qu1 = new Button("Professoren");      /* Button fuer Professoren   */
  Button qu2 = new Button("Assistenten");      /* Button fuer Assistenten   */
  Button qu3 = new Button("Studenten");        /* Button fuer Studenten     */
  Button qu4 = new Button("Vorlesungen");      /* Button fuer Vorlesungen   */
  Button qu5 = new Button("hoeren");           /* Button fuer hoeren        */
  Button qu6 = new Button("voraussetzen");     /* Button fuer voraussetzen  */
  Button qu7 = new Button("pruefen");          /* Button fuer pruefen       */
  Button qu8 = new Button("User-Tabellen");    /* Button fuer User-Tabellen */



  Connection con;         /* Verbindungsobjekt zur Verbindung mit dem DBMS  */
  Statement stmt;            /* Statement-Objekt zur Kommunikation mit DBMS */


  public JdbcApplet(){}                          /* Konstruktor fuer Applet */


  public void init() {                          /* Das Applet initialisieren */
    try {
      this.setLayout(layout);
      this.setSize(700,400);
      inputField.setBackground(Color.gray);
      inputField.setFont(new Font("Serif",1,14));

      inputField.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {    /* ActionListener   */
          String q = inputField.getText();              /* fuer Eingabefeld */
          execQuery(q);                                 /* Gibt Query an    */
        }                                               /* execQuery weiter */
      });
      outputArea.setBackground(Color.white);
      outputArea.setEditable(false);
      outputArea.setFont(new Font("Monospaced",1,14));

      /* ActionListener fuer jeweiligen Button      */
      /* gibt Abfrage an Methode execQuery() weiter */
      qu1.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
```

```
    execQuery("select persnr,name,rang,raum," +
              "gebdatum as Geburtsdatum from Professoren");
  }
});

qu2.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    execQuery("select persnr, name, fachgebiet, boss," +
              "gebdatum as Geburtsdatum from Assistenten");
  }
});

qu3.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    execQuery("select matrnr, name, semester," +
              "gebdatum as Geburtsdatum from Studenten");
  }
});

qu4.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    execQuery("select * from Vorlesungen");
  }

});
qu5.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    execQuery("select * from hoeren");
  }
});

qu6.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    execQuery("select * from voraussetzen");
  }
});

qu7.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    execQuery("select * from pruefen");
  }
});

qu8.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    execQuery("select name from sysobjects where xtype='U'");
  }
});

this.add(outputArea,BorderLayout.CENTER); /* Hinzufuegen und Anordnen */
this.add(inputField,BorderLayout.NORTH);  /* der Elemente des Applets */
this.add(p= new Panel(new GridLayout(8,1)), BorderLayout.WEST);



p.add(qu1);
p.add(qu2);
```

```
      p.add(qu3);
      p.add(qu4);
      p.add(qu5);
      p.add(qu6);
      p.add(qu7);
      p.add(qu8);


    }
    catch (Exception e) {e.printStackTrace();
    }



    String url    = "jdbc:odbc:dbs";      /* Verbindungsaufbau mit dem DBMS */

    String user   = "erika";

    String passwd = "mustermann";


    try {                                              /* probiere           */

      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  /* Brueckentreiber    */

    }

    catch(java.lang.ClassNotFoundException ex) {

      System.err.print("ClassNotFoundException: ");
      System.err.println(ex.getMessage());

    }

    try {                                                    /* probiere    */
      con = DriverManager.getConnection(url, user, passwd);  /* Verbindung  */
    }

    catch(SQLException ex) {

      outputArea.setText("SQLException: " + ex.getMessage());

    }
  }

  void execQuery(String query) {       /* SQL-Query an DBMS uergeben        */
                                       /* und Ergebnis in TextArea anzeigen */
    try {
      stmt = con.createStatement();       /* Statement-Objekt instantiieren */
      ResultSet rs = stmt.executeQuery(query);           /* Resultset holen */

      int z = rs.getMetaData().getColumnCount();/* Zahl der Ergebnisspalten */
      outputArea.setText("\n");
      outputArea.setVisible(false);

      for (int i=1;i<=z;i++) {     /* alle Spaltennamen formatiert ausgeben */
        String lab=rs.getMetaData().getColumnLabel(i);
```

```
        outputArea.append(lab);
        int y = rs.getMetaData().getColumnDisplaySize(i)+4;
        for (int j=0;j<(y-lab.length());j++)
          outputArea.append(" ");
        }
        outputArea.append("\n");

        String arg;                          /* Inhalt der Ergebnismenge    */
        while(rs.next()) {                   /* wird zeilenweise formatiert */
          outputArea.append("\n");           /* in der TextArea ausgegeben  */
          for (int i=1;i<=z;i++) {
            arg=rs.getString(i);
            int len;
            if (arg != null) {
              len=arg.length();
      if(rs.getMetaData().getColumnName(i).equals("Geburtsdatum")) {
        outputArea.append(arg.substring(0,10));
      }
      else{
                outputArea.append(arg);
      }
            }
            else {
              len=4;
              outputArea.append("null");
            }
            int y = rs.getMetaData().getColumnDisplaySize(i)+4;
            for (int j=0;j<(y-len);j++) outputArea.append(" ");
          }
        }
        outputArea.setVisible(true);
        stmt.close();

      }catch(SQLException ex) {        /* Abfangen von SQL-Fehlermeldungen */
        outputArea.setText("SQLException: " + ex.getMessage());
    }
  }
}
```

Listing 9.5: Quelltext der Java-Klassen vom Java-Applet

9.6 Java-Servlets

Der Einsatz von JDBC auf dem Rechner des Clienten als Applet wird von den Sicherheitsbeschränkungen beeinträchtigt, unter denen Applets ausgeführt werden können. Als Alternative bieten sich Java-Servlets an, die auf dem Server des Datenbankanbieters laufen. Mit einem Formular auf einer HTML-Seite auf dem Web-Server des Anbieters werden die Argumente vom Anwender ermittelt und dann an das in Java formulierte Servlet übergeben. Per JDBC wird die Query an den Datenbankserver gerichtet und das Ergebnis mit Java-Befehlen zu einer dynamischen HTML-Seite aufbereitet.

Listing 9.6 zeigt eine HTML-Seite zum Erfassen der Anwendereingabe, Listing 9.7 zeigt den Quelltext des zugehörigen Java-Servlet, welches vom Rechner `atum` aus mit dem nativen Microsoft-Treiber den MS-SQL-Server kontaktiert. Die erzeugte Ausgabe findet sich in Abbildung 9.11.

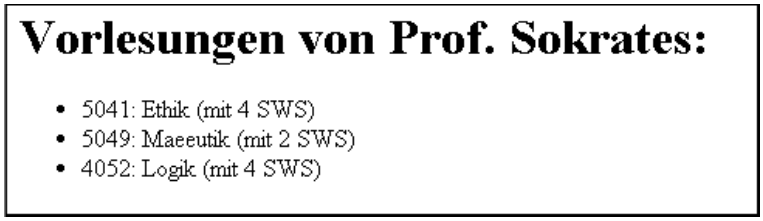


Abbildung 9.11: vom Java-Servlet erzeugte Ausgabe

```
<HTML>
  <HEAD><TITLE>Vorlesungsverzeichnis mittels Java-Servlet</TITLE></HEAD>
  <BODY>
    <FORM
     METHOD="GET"
     ACTION="http://atum.informatik.uni-osnabrueck.de:8080/examples/VrlVrz">
     Bitte geben Sie den Namen eines Professors ein:
     <P><INPUT NAME="professor_name" SIZE="40"><P>
     <INPUT TYPE="submit" VALUE="Vorlesungen ermitteln">
    </FORM>
  </BODY>
</HTML>
```

Listing 9.6: Quelltext einer HTML-Seite zur Versorgung des Servlets VrlVrz.java

```
import javax.servlet.*; import javax.servlet.http.*; import java.io.*;
import java.sql.*; import java.text.*;

public class VrlVrz extends HttpServlet {

  public void doGet (HttpServletRequest request,HttpServletResponse response)
                    throws ServletException, IOException {
    Connection con = null;
    Statement stmt = null;
    ResultSet rs   = null;
    response.setContentType("Text/html");
    PrintWriter out = response.getWriter();
    try {
      Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
      con = DriverManager.getConnection(
      "jdbc:microsoft:sqlserver://arnold.informatik.uni-osnabrueck.de:1433",
                                        "erika","mustermann");
      stmt = con.createStatement();
      String query = "select v.vorlnr, v.titel, v.sws " +
                     "from vorlesungen v, professoren p " +
                     "where v.gelesenvon = p.persnr and p.name =’" +
                      request.getParameter("professor_name") + "’";
      rs = stmt.executeQuery(query);
      out.println("<HTML>");
      out.println("<HEAD><TITLE>Java Servlet</TITLE></HEAD>");
      out.println("<BODY>");
      out.println("<H1>Vorlesungen von Prof. " +
                  request.getParameter("professor_name") +": </H1>");
      out.println("<UL>");
      while (rs.next())
        out.println("<LI>" +
          rs.getInt("VorlNr") + ": " + rs.getString("Titel") + " (mit " +
          rs.getInt("SWS") + " SWS)" + "</LI>");
      out.println("</UL>");
      out.println("<BODY></HTML>");
    }
    catch(ClassNotFoundException e) {
      out.println("Datenbanktreiber nicht gefunden: " + e.getMessage());
    }
    catch(SQLException e) {
      out.println("SQLException: " + e.getMessage());
    }
    finally {
      try { if (con != null ) con.close();
      } catch (SQLException ignorieren) {}
    }
  }
}
```

Listing 9.7: Quelltext des Servlets VrlVrz.java

9.7 Java Server Pages

Java-Servlets vermischen oft in verwirrender Weise HTML mit Java, d.h. Layout-Informationen und algorithmische Bestandteile. Abhilfe wird geschaffen durch sogenannte Java Server Pages, in denen eine bessere Trennung zwischen statischem Layout und den benötigten algorithmischen Bestandteilen ermöglicht wird.

Die wesentliche Inhalt einer Java-Server-Page besteht aus HTML-Vokabeln, die nun Methoden einer Java-Klasse aufrufen dürfen (gekennzeichnet durch spitze Klammern und Prozentzeichen).

Listing 9.8 zeigt die Java-Server-Page `vorlesungen.jsp`, welche zuständig ist für die Erfassung der Benutzereingabe und die Ausgabe der Datenbankantwort. Die Eingabe wird über ein HTML-Formular ermittelt und über den Feldnamen `profname` an die Java-Bean `VorlesungenBean` übermittelt. Die Ausgabe wird im wesentlichen bestritten durch einen String, der von der Methode `generiereVorlListe()` geliefert wird.

Verwendet wird hierbei der Tomcat-Server von Apache, der die Java-Server-Page in ein Servlet übersetzt und mit der zugehörigen Java-Bean verbindet. Die generierten Webseiten werden in Abbildungen 9.12 bzw. 9.13 gezeigt.

```
<%@ page import = "dbs.VorlesungenBean" %>
<jsp:useBean id="prg" class="dbs.VorlesungenBean" scope="request"/>
<jsp:setProperty name="prg" property="*"/>

<html>
  <% if (prg.getProfname() == null) { %>
    <head><title>Professoren-Namen erfassen</title></head>
    <body bgcolor="DDDDDD">
      <FORM METHOD="GET">
        Bitte geben Sie den Namen eines Professors ein:<P>
        <INPUT TYPE=TEXT NAME=profname><P>
        <INPUT TYPE=SUBMIT VALUE="Vorlesungen ermitteln!">
      </FORM>
    </body>
  <% } else { %>
    <head><title>Vorlesungen ausgeben</title></head>
    <body bgcolor="DDDDDD">
      Die Vorlesungen von <%= prg.getProfname() %> lauten: <P>
      <%= prg.generiereVorlListe() %>
    </body>
  <% } %>
</html>
```

Listing 9.8: Quelltext der Java-Server-Page vorlesungen.jsp

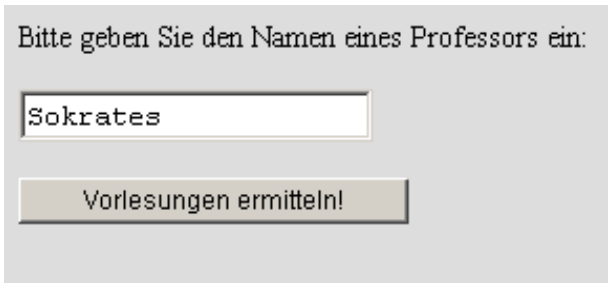


Abbildung 9.12: von der Java-Server-Page erzeugtes Eingabeformular

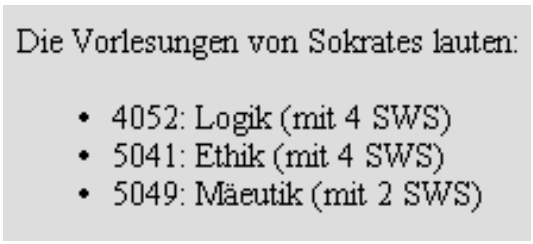


Abbildung 9.13: von der Java-Server-Page erzeugte Ausgabe

```
package dbs; import java.sql.*;

public class VorlesungenBean {
    Connection con  = null;
    String con_err  = null;
    String profname = null;

    public VorlesungenBean() {
      try {
      Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
      con = DriverManager.getConnection(
      "jdbc:microsoft:sqlserver://arnold.informatik.uni-osnabrueck.de:1433",
                                        "erika","mustermann");
      }
      catch(Exception e) { con_err = e.toString(); }
    }

  public void   setProfname(String name) { profname = name; }
  public String getProfname()            { return profname; }

  public String generiereVorlListe(){
    Statement stmt = null;
    ResultSet rs   = null;
    if (con==null) return ("Probleme mit der Datenbank: "+con_err + "<BR>");
    StringBuffer result = new StringBuffer();
    try{
      stmt = con.createStatement();
      String query =
        "select v.vorlnr, v.titel, v.sws from vorlesungen v, professoren p "+
        "where v.gelesenvon = p.persnr and p.name ='" + profname + "'";
      rs = stmt.executeQuery(query);
      result.append("<UL>");
      while (rs.next())
        result.append("<LI>"+rs.getInt("VorlNr")+": "+rs.getString("Titel")+
                      " (mit " + rs.getInt("SWS") + " SWS)" + "</LI>");
      result.append("</UL>");
    }
    catch(SQLException e) {
      result = new StringBuffer("Bei der Abfrage fuer " + profname +
               " trat ein Fehler auf: " + e.getMessage() + "</BR>");
    }
    return result.toString();
  }

  public void finalize () {
    try {if (con != null ) con.close();} catch (SQLException ignorieren) {}
  }
}
```

Listing 9.9: Quelltext der Java-Bean VorlesungenBean.java

9.8 Cold Fusion

Cold Fusion ist ein Anwendungsentwicklungssystem der Firma Macromedia für dynamische Web-Seiten. Eine ColdFusion-Anwendung besteht aus einer Sammlung von CFML-Seiten, die in der *Cold Fusion Markup Language* geschrieben sind. Die Syntax von CFML ist an HTML angelehnt und beschreibt die Anwendungslogik. In Abbildung 9.14 ist der grundsätzliche Ablauf dargestellt:

1. Wenn ein Benutzer eine Seite in einer Cold Fusion - Anwendung anfordert, sendet der Web-Browser des Benutzers eine HTTP-Anforderung an den Web-Server.
2. Der Web-Server übergibt die vom Clienten übermittelten Daten aufgrund der Dateiendung `cfm` an den Cold Fusion Application Server.
3. Cold Fusion liest die Daten vom Clienten und verarbeitet den auf der Seite verwendeten CFML-Code und führt die damit angeforderte Anwendungslogik aus.
4. Cold Fusion erzeugt dynamisch eine HTML-Seite und gibt sie an den Web-Server zurück.
5. Der Web-Server gibt die Seite an den Web-Browser zurück.

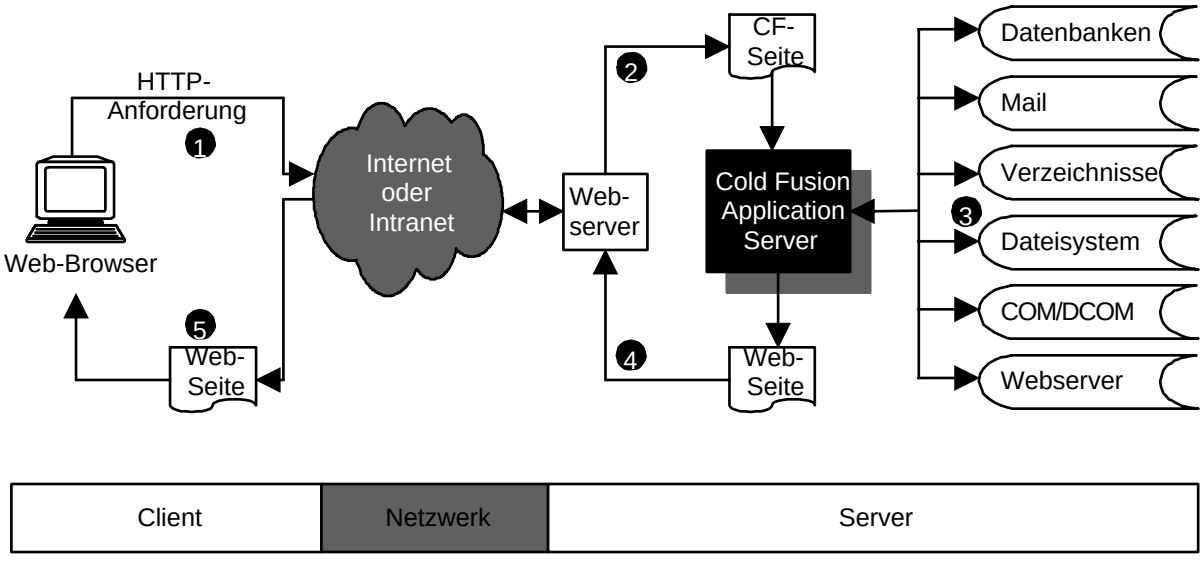


Abbildung 9.14: Arbeitsweise von Coldfusion

Von den zahlreichen Servertechnologien, mit denen Cold Fusion zusammenarbeiten kann, interessiert uns hier nur die Anbindung per ODBC an eine relationale Datenbank.

CF-Vorlesungsverzeichnis: `http://www.uos.de/vpv/sommer2003/index.cfm`

CF-Online-Dokumentation: `http://balrog.informatik.uni-osnabrueck.de/CFDOCS/dochome.ht`

Listing 9.10 zeigt eine unformatierte Ausgabe einer SQL-Query.

```
<CFQUERY NAME       = "Studentenliste"
         USERNAME   = "erika"
         PASSWORD   = "mustermann"
         DATASOURCE = "dbs"
         DBTYPE     = "ODBC">
  SELECT matrnr, name from studenten
</CFQUERY>

<HTML>
  <HEAD>
    <TITLE> Studentenliste </TITLE>
  </HEAD>
  <BODY>
    <H2> Studentenliste (unformatiert)</H2>
    <CFOUTPUT QUERY="Studentenliste">
      #name# #matrnr# <BR>
    </CFOUTPUT>
  </BODY>
</HTML>
```

Listing 9.10: Quelltext von studliste.cfm

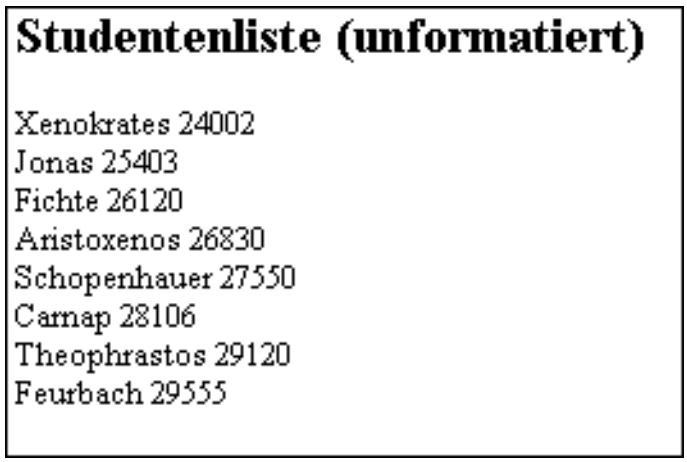


Abbildung 9.15: Screenshot von studliste.cfm

Listing 9.11 zeigt die formatierte Ausgabe einer SQL-Query unter Verwendung einer HTML-Tabelle.

```
<CFQUERY NAME       = "Studententabelle"
         USERNAME   = "erika"
         PASSWORD   = "mustermann"
         DATASOURCE = "dbs"
         DBTYPE     = "ODBC">
  SELECT matrnr, name, gebdatum as geburt from studenten
         WHERE  CURRENT_TIMESTAMP > DATEADD(year, 30, gebdatum)
</CFQUERY>

<HTML>
  <HEAD>
    <TITLE> Studententabelle </TITLE>
  </HEAD>
  <BODY>
    <H2> Alle Studenten, die &auml;lter als 30 Jahre sind,<BR>
         als HTML-Tabelle</H2>
    <TABLE BORDER>
    <TD>Matrikelnummer</TD> <TD> Nachname </TD> <TD>Geburtsdatum </TD></TR>
    <CFOUTPUT QUERY="Studententabelle">
      <TR><TD> #matrnr# </TD> <TD> #name# </TD> <TD> #geburt# </TR>
    </CFOUTPUT>
    </TABLE>
  </BODY>
</HTML>
```

Listing 9.11: Quelltext von studtabelle.cfm

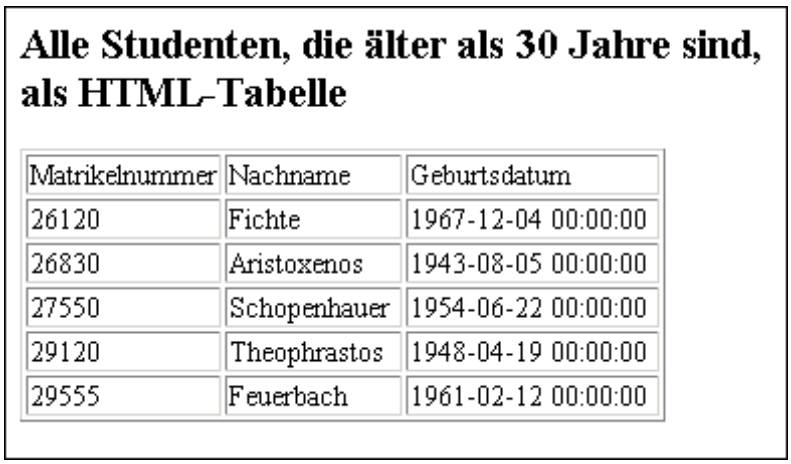


Abbildung 9.16: Screenshot von studtabelle.cfm

Listing 9.12. zeigt die Verwendung eines Formulars zum Eingeben eines Dozentennamens, der eine Suche anstößt.

```
<HTML>
  <HEAD>
    <TITLE> Studentenformular </TITLE>
  </HEAD>
  <BODY>
    <FORM ACTION="studsuche.cfm" METHOD="POST">
    Sie k&ouml;nnen nach den H&ouml;rern eines Dozenten suchen: <BR>
    Professorenname: <INPUT TYPE="text"   NAME="Profname">
    <INPUT TYPE="Submit" VALUE="Suchen">
    </FORM>
  </BODY>
</HTML>
```

Listing 9.12: Quelltext von studformular.cfm

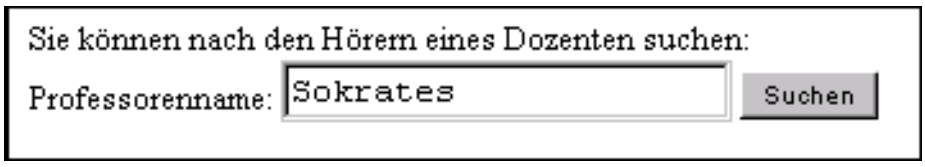


Abbildung 9.17: Screenshot von studformular.cfm

Der vom Formular `studformular.cfm` erfaßte Name wird übergeben an die Datei `studsuche.cfm`, welche im Listing 9.13 gezeigt wird.

```
<CFQUERY NAME       = "Studentensuche"
         USERNAME   = "erika"
         PASSWORD   = "mustermann"
         DATASOURCE = "dbs"
         DBTYPE     = "ODBC" >
  SELECT distinct s.matrnr, s.name
  FROM   professoren p, vorlesungen v, hoeren h, studenten s
  WHERE  s.matrnr     = h.matrnr
  AND    h.vorlnr     = v.vorlnr
  AND    v.gelesenvon = p.persnr
  AND    p.name       = '#FORM.Profname#'
</CFQUERY>

<HTML>
  <HEAD>
    <TITLE> Studenten eines Professors </TITLE>
  </HEAD>
  <BODY>
    <CFOUTPUT>
      Professor #FORM.Profname# hat folgende H&ouml;rer: <P>
    </CFOUTPUT>
    <CFOUTPUT QUERY="Studentensuche">
      #matrnr# #name#<BR>
    </CFOUTPUT>
  </BODY>
</HTML>
```

Listing 9.13: Quelltext von studsuche.cfm

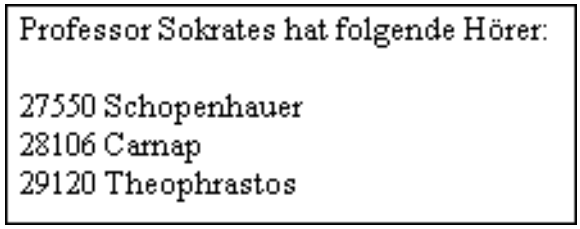


Abbildung 9.18: Screenshot von studsuche.cfm

Listing 9.14 zeigt eine HTML-Tabelle mit sensitiven Links für die Professoren.

```
<CFQUERY NAME       = "Vorlesungstabelle"
         USERNAME   = "erika"      PASSWORD = "mustermann"
         DATASOURCE = "dbs"        DBTYPE   = "ODBC">
  SELECT vorlnr, titel, name, persnr FROM vorlesungen, professoren
  where gelesenvon = persnr
</CFQUERY>
<HTML>
  <HEAD> <TITLE> Vorlesungstabelle </TITLE> </HEAD>
  <BODY>
    <H2> Vorlesungen mit sensitiven Links </H2>
    <TABLE BORDER>
    <TD>Vorlesungsnr</TD> <TD> Titel </TD> <TD>Dozent</TD> </TR>
    <CFOUTPUT QUERY="Vorlesungstabelle">
      <TR><TD>#vorlnr#</TD><TD>#Titel#</TD>
      <TD><A HREF="profinfo.cfm?profid=#persnr#">#name#</A></TD></TR>
    </CFOUTPUT>
    </TABLE>
  </BODY>
</HTML>
```

Listing 9.14: Quelltext von vorltabelle.cfm

Abbildung 9.19: Screenshot von vorltabelle.cfm

Die in Listing 9.14 ermittelte Personalnummer eines Professors wird in Form einer URL an die in Listing 9.15 gezeigte Datei `profinfo.cfm` übergeben und dort in einer Select-Anweisung verwendet. Die gefundenen Angaben zum Dozenten werden anschließend ausgegeben.

```
<CFQUERY NAME       = "Profinfo"
         USERNAME   = "erika"
         PASSWORD   = "mustermann"
         DATASOURCE = "dbs"
         DBTYPE     = "ODBC">
  SELECT * from Professoren
  WHERE persnr=#URL.profid#
</CFQUERY>

<HTML>
  <HEAD>
    <TITLE> Professoreninfo: </TITLE>
  </HEAD>
  <BODY>
   <H2> Professoren-Info</H2>
    <CFOUTPUT QUERY="Profinfo">
      Professor #name# hat die Personalnummer #persnr#. <BR>
      Er wird nach Rang #rang# besoldet. <BR>
      Sein Dienstzimmer ist Raum #Raum#.
    </CFOUTPUT>
    </TABLE>
  </BODY>
</HTML>
```

Listing 9.15: Quelltext von profinfo.cfm

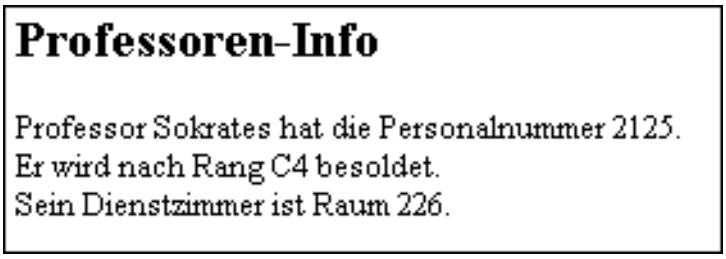


Abbildung 9.20: Screenshot von profinfo.cfm

Listing 9.16 zeigt ein Formular zum Einfügen eines Professors.

```
<HTML>
  <HEAD> <TITLE> Professoreneinf&uuml;geformular </TITLE> </HEAD>
  <BODY>
    <H2> Professoreneinf&uuml;geformular </H2>
    <FORM ACTION="profinsert.cfm" METHOD="POST"> <PRE>
    PersNr:   <INPUT SIZE= 4 TYPE="text" NAME="ProfPersnr">
              <INPUT TYPE="hidden" NAME="ProfPersnr_required"
              VALUE="PersNr erforderlich !">
              <INPUT TYPE="hidden" NAME="ProfPersnr_integer"
              VALUE="Personalnummer muss ganzzahlig sein !">
    Name:     <INPUT SIZE=15 TYPE="text"       NAME="ProfName">
              <INPUT TYPE="hidden" NAME="ProfName_required"
              VALUE="Name erforderlich !">
    Rang:     <SELECT NAME="ProfRang"> <OPTION>C2 <OPTION>C3 <OPTION>C4
              </SELECT>
    Raum:     <INPUT SIZE=4 TYPE="text" NAME="ProfRaum">
              <INPUT TYPE="Submit" VALUE="Einf&uuml;gen">
    </PRE></FORM>
  </BODY>
</HTML>
```

Listing 9.16: Quelltext von profinsertform.cfm

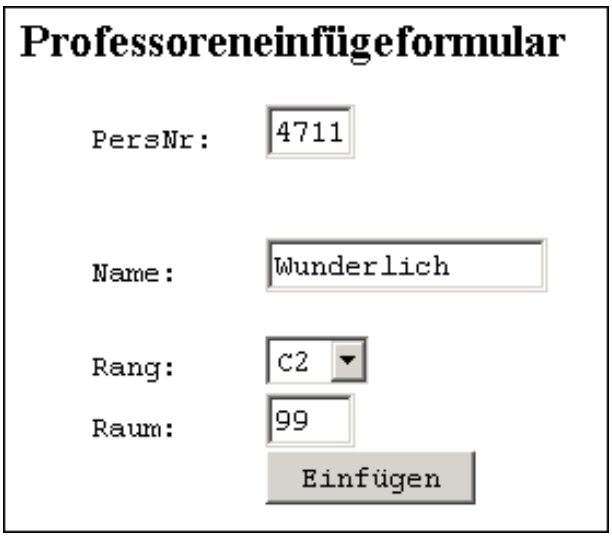


Abbildung 9.21: Screenshot von profinsertform.cfm

Die von Listing 9.16 übermittelten Daten werden in Listing 9.17 zum Einfügen verwendet. Anschließend erfolgt eine Bestätigung.

```
<CFQUERY NAME       = "Profinsert"
         USERNAME   = "erika"
         PASSWORD   = "mustermann"
         DATASOURCE = "dbs"
         DBTYPE     = "ODBC">

  INSERT INTO Professoren (PersNr, Name, Rang, Raum)
  VALUES ('#FORM.ProfPersnr#','#FORM.ProfName#','#FORM.ProfRang#',
          '#FORM.ProfRaum#')
</CFQUERY>

<HTML>
  <HEAD>
    <TITLE> Professoreneinf&uuml;gen </TITLE>
  </HEAD>
  <BODY>
    In die Tabelle der Professoren wurde eingef&uuml;gt: <P>
    <CFOUTPUT>
       <PRE>
       Persnr: #FORM.ProfPersnr#
       Name:   #FORM.ProfName#
       Rang:   #FORM.ProfRang#
       Raum:   #ProfRaum#
       </PRE>
    </CFOUTPUT>
  </BODY>
</HTML>
```

Listing 9.17: Quelltext von profinsert.cfm

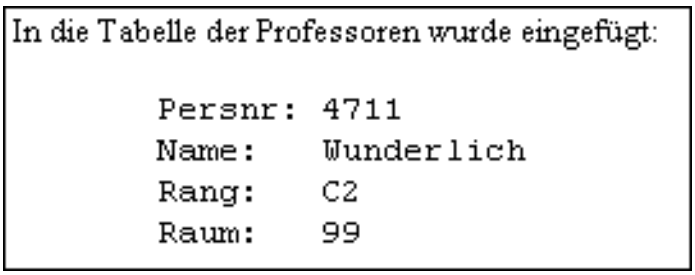


Abbildung 9.22: Screenshot von profinsert.cfm

Listing 9.18 zeigt eine Tabelle mit einem Formular zum Löschen eines Professors.

```
<CFQUERY NAME       = "Professorentabelle"
         USERNAME   = "erika"     PASSWORD   = "mustermann"
         DATASOURCE = "dbs"       DBTYPE     = "ODBC">
  SELECT * from professoren
</CFQUERY>
<HTML>
  <HEAD>
    <TITLE> Professorenformular zum L&oum;schen </TITLE>
  </HEAD>
  <BODY>
    <H2> Professorenformular zum L&ouml;schen</H2>
    <TABLE BORDER>
    <TD>PersNr</TD><TD>Name</TD><TD>Rang</TD><TD>Raum</TD></TR>
    <CFOUTPUT QUERY="Professorentabelle">
    <TR><TD>#persnr#</TD><TD>#name#</TD><TD>#rang#</TD><TD>#raum#</TD></TR>
    </CFOUTPUT>
    </TABLE>
    <FORM ACTION="profdelete.cfm" METHOD="POST">
    Personalnummer: <INPUT SIZE=4  TYPE="text" NAME="Persnr">
    <INPUT TYPE="Submit" VALUE="Datensatz l&ouml;schen">
    </FORM>
  </BODY>
</HTML>
```

Listing 9.18: Quelltext von profdeleteform.cfm

Abbildung 9.23: Screenshot von profdeleteform.cfm

Die in Listing 9.18 ermittelte Personalnummer eines Professors wird in Listing 9.19 zum Löschen verwendet. Anschließend erfolgt eine Bestätigung.

```
<CFQUERY NAME       = "Profdelete"
         USERNAME   = "erika"
         PASSWORD   = "mustermann"
         DATASOURCE = "dbs"
         DBTYPE     = "ODBC">
  delete from professoren
  where persnr = #FORM.persnr#
</CFQUERY>

<HTML>
  <HEAD> <TITLE> Professoren l&ouml;schen </TITLE> </HEAD>
  <BODY>
    <CFOUTPUT>
      Der Professor mit Personalnummer #FORM.persnr# wurde gel&ouml;scht
    </CFOUTPUT>
  </BODY>
</HTML>
```

Listing 9.19: Quelltext von profdelete.cfm

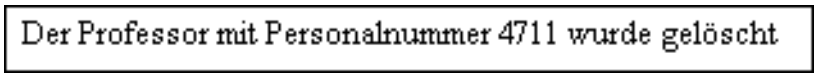


Abbildung 9.24: Screenshot von profdelete.cfm

Listing 9.20 zeigt ein Formular zum Suchen nach einem Professorendatensatz unter Verwendung des Wildcard-Zeichens %.

```
<CFAPPLICATION SESSIONMANAGEMENT="YES" NAME="Update">
<HTML>
  <HEAD> <TITLE> Professorenupdate </TITLE> </HEAD>
  <BODY>
    Bitte geben Sie Suchkriterien ein,
    um gezielt einen Professor zum UPDATE zu suchen.
    <P><FORM ACTION="profupdate.cfm" METHOD="POST">
    <TABLE>
    <TR><TD> Personalnummer: </TD><TD><INPUT TYPE="text" SIZE=4
             NAME="ProfPersnr">
    <INPUT TYPE="HIDDEN" NAME="ProfPersnr_integer"
     VALUE="Die Personalnummer muss ein ganzzahliger Wert sein"></TD></TR>
    <TR><TD> Nachname:</TD><TD><INPUT SIZE=15 TYPE="text"
    NAME="ProfName">   Wildcard <B>%</B>
                    kann genutzt werden.</TD></TR>
    <TR><TD> Gehaltsklasse: </TD>
    <TD>
    <SELECT NAME="ProfRang"><OPTION>C2<OPTION>C3<OPTION>C4
    </SELECT></TD></TR>
    <TR><TD> Raum: </TD><TD><INPUT SIZE=4 TYPE="text" NAME="ProfRaum">
    <INPUT TYPE="HIDDEN" NAME="ProfRaum_integer"
    VALUE="Die Raumnummer muss ein ganzzahliger Wert sein"></TD></TR>
    <INPUT TYPE="HIDDEN" NAME="i" VALUE="1">
    <TR><TD><INPUT TYPE="Submit" VALUE="Prof suchen"></TD><TD></TD></TR>
    </TABLE>
    </FORM>
</HTML>
```

Listing 9.20: Quelltext von profupdateformular.cfm

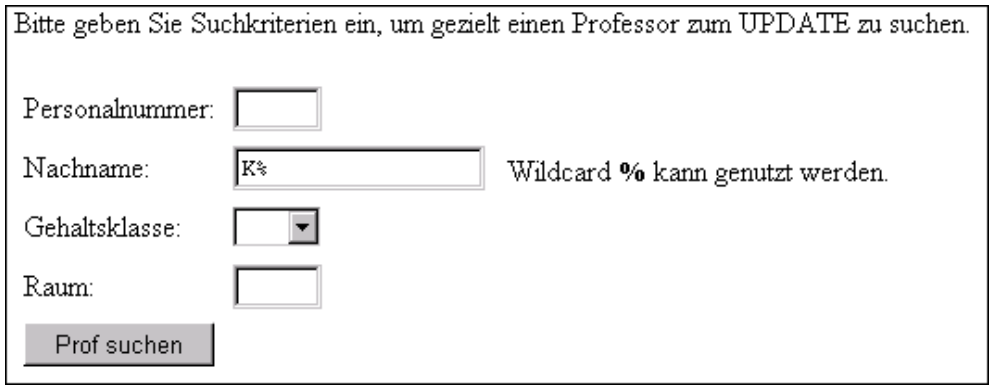


Abbildung 9.25: Screenshot von profupdateformular.cfm

Die in Listing 9.20 gefundenen Treffer können im Listing 9.21 durchlaufen werden und anschließend editiert werden.

```
<!--- erstellt von Ralf Kunze --->

<CFQUERY NAME       = "ProfAbfr"
         USERNAME   = "erika"
         PASSWORD   = "mustermann"
         DATASOURCE = "dbs"
         DBTYPE     = "ODBC">

        <!--- Where 0=0, um in jedem Fall eine
        korrekte Abfrage zu erhalten --->
  SELECT * FROM professoren where 0 = 0

        <!--- Weitere Statements gegebenenfalls anhaengen --->
<CFIF #ProfPersnr# is NOT "">
AND PersNr = #ProfPersnr#
</CFIF>

<CFIF #ProfName# is not "">
AND Name LIKE '#ProfName#'
</CFIF>

<CFIF #ProfRang# is not "">
AND Rang = '#ProfRang#'
</CFIF>

<CFIF #ProfRaum# is not "">
AND Raum = '#ProfRaum#'
</CFIF>

</CFQUERY>

<HTML>

  <HEAD>
    <TITLE> Professorenupdate </TITLE>
  </HEAD>

  <BODY>

    <!--- Falls keine Ergebnisse erzielt wurden, Fehlermeldung geben
          und den Rest der Seite mit CFABORT unterdruecken --->
    <CFIF #ProfAbfr.Recordcount# IS "0">
          Ihre Anfrage lieferte leider keine passenden Records.<BR>
          <A HREF="profupdateformular.cfm">New Search</A>
          <CFABORT>
    </CFIF>
    Bitte geben sie die gew&uuml;nschte &Auml;nderung ein
    bzw. w&auml;hlen sie den entsprechenden Datensatz aus:

    <!--- Ausgabe der Ergebnisse. Bei Record #i# starten
          und nur ein Record liefern --->
    <CFOUTPUT QUERY="ProfAbfr" STARTROW="#i#" MAXROWS="1">
    <FORM ACTION="update.cfm" METHOD="POST">
```

```
    <!--- Ausgabe der Werte in ein Formular zum aendern --->
    <TABLE>
      <TR><TD>Personalnummer: </TD>
          <TD><INPUT TYPE="text" SIZE=4 NAME="ProfPersnr" VALUE="#Persnr#">
              <INPUT TYPE="HIDDEN" NAME="ProfPersnr_integer"
                     VALUE="Personalnummer muss ganzzahlig sein"></TD></TR>
      <TR><TD>Nachname:</TD>
          <TD><INPUT SIZE=15 TYPE="text" NAME="ProfName"
                     VALUE="#Name#"></TD></TR>
      <TR><TD>Gehaltsklasse:</TD>
          <TD><SELECT NAME="ProfRang">
                <CFIF #Rang# IS "C2"><OPTION SELECTED><CFELSE><OPTION></CFIF>C2
                <CFIF #Rang# IS "C3"><OPTION SELECTED><CFELSE><OPTION></CFIF>C3
                <CFIF #Rang# IS "C4"><OPTION SELECTED><CFELSE><OPTION></CFIF>C4
      </SELECT></TD></TR>
      <TR><TD> Raum:</TD>
          <TD><INPUT SIZE=4 TYPE="text" NAME="ProfRaum" VALUE="#Raum#">
              <INPUT TYPE="HIDDEN" NAME="ProfRaum_integer"
                     VALUE="Raumnummer muss ganzzahlige sein"></TD></TR>
      <TR><TD><INPUT TYPE="Submit" VALUE="Update"></TD>
          <TD><INPUT TYPE="RESET"></TD></TR>
    </TABLE>
    </FORM>
    </CFOUTPUT>

    <!--- Den Zaehler setzen und entsprechend des
      Wertes weiteren Link anbieten oder nicht --->
    <CFIF #i# IS "1">
      <IMG SRC="Grayleft.gif" ALT="Back">
    <CFELSE>
      <CFSET iback=#i#-1>
      <CFOUTPUT>
        <A HREF="profupdate.cfm?i=#iback#&ProfPersnr=#ProfPersnr#
                 &Profname=#Profname#&ProfRang=#ProfRang#&ProfRaum=#ProfRaum#">
        <IMG SRC="redleft.gif" BORDER="0" ALT="back"></A>
      </CFOUTPUT>
    </CFIF>
    <A HREF="profupdateformular.cfm">New Search</A>
    <CFIF #i# LESS THAN #ProfAbfr.RecordCount#>
      <CFSET inext=#i#+1>
      <CFOUTPUT>
        <A HREF="profupdate.cfm?i=#inext#&ProfPersnr=#ProfPersnr#
                &Profname=#Profname#&ProfRang=#ProfRang#&ProfRaum=#ProfRaum#">
        <IMG SRC="redright.gif" ALIGN="Next Entry" BORDER="0"></A>
      </CFOUTPUT>
    <CFELSE>
      <IMG SRC="grayright.gif" ALT="Next">
    </CFIF>

    <!--- Ausgabe welcher Datensatz gezeigt wird
          und wieviele insgesamt vorhanden sind --->
    <CFOUTPUT>Eintrag #i# von #ProfAbfr.RecordCount#</CFOUTPUT><BR>
  </BODY>
</HTML>
```

Listing 9.21: Quelltext von profupdate.cfm

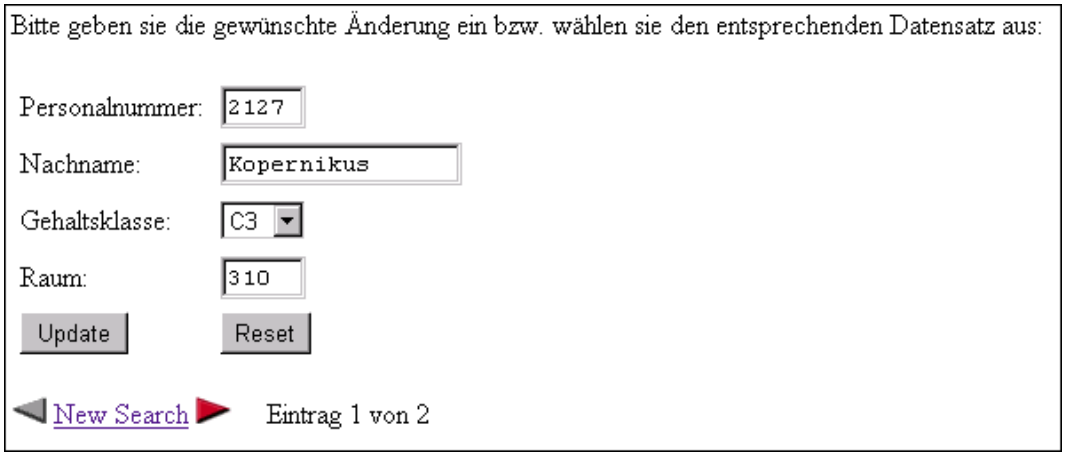


Abbildung 9.26: Screenshot von profupdate.cfm

Listing 9.22 zeigt die Durchführung der Update-Operation auf dem in Listing 9.21 ausgewählten Professorendatensatz.

```
<CFQUERY NAME       = "Profupdate"
         USERNAME   = "erika"
         PASSWORD   = "mustermann"
         DATASOURCE = "dbs"
         DBTYPE     = "ODBC">

  UPDATE professoren set
  name ='#FORM.ProfName#',
  rang ='#FORM.ProfRang#',
  raum ='#FORM.ProfRaum#'
  where persnr = #FORM.ProfPersnr#
</CFQUERY>

<HTML>
  <HEAD>
    <TITLE> Professorenupdate </TITLE>
  </HEAD>
  <BODY>
    In der Tabelle der Professoren wurde ein Datensatz modifiziert:
    <CFOUTPUT>
       <PRE>
       Persnr: #FORM.ProfPersnr#
       Name:   #FORM.ProfName#
       Rang:   #FORM.ProfRang#
       Raum:   #ProfRaum#
       </PRE>
    </CFOUTPUT>
    <A HREF="profupdateformular.cfm"> New Search </A>
  </BODY>
</HTML>
```

Listing 9.22: Quelltext von update.cfm

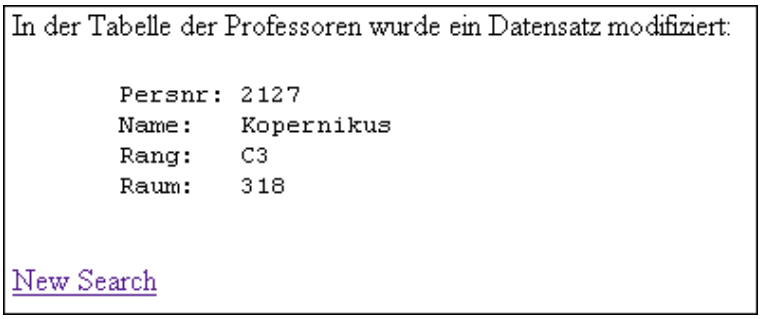


Abbildung 9.27: Screenshot von update.cfm

9.9 PHP

Eine Alternative zur ColdFusion-Technik stellt PHP dar, eine Server-basierte Skriptsprache, die in HTML-Seiten eingebettet werden kann. Der Name entstand ursprünglich aus der Abkürzung *Personal Home Page*. Eine HTML-Seite mit PHP-Aufrufen hat typischerweise die Endung `php`. Neben der Auswertung der HTML-Tags übernimmt ein entsprechend aufgerüsteter Web-Server die Interpretation der PHP-Scripte, welche für die Generierung dynamischer, nicht notwendigerweise datenbankgetriebener Inhalte sorgen.

Auf der Web-Seite `http://www.php.net` sind ausführliche Hinweise zu finden. Eine nützliche Seite ist `http://www.selfphp.info`, dort finden sich auch zahlreiche Funktionen der GD-Library, mit der dynamische Grafiken erzeugt werden können:
`http://www.selfphp.info/funktionsreferenz/image_funktionen`.

Unter Verwendung der GD-Library erzeugen die Routinen der JpGraph-Library komplexere Bilder, wie z.B. Tortengrafiken (`http://www.aditus.nu/jpgraph/index.php`).

Wir beschränken uns hier auf die Angabe von drei Beispielen, in denen gegen den *Microsoft SQL Server* eine SQL-Query abgesetzt und das Ergebnis tabellarisch bzw. graphisch angezeigt wird.

Eine Möglichkeit zur Übergabe der Query an die PHP-Seite besteht darin, die Query bei Aufruf der PHP-Seite an die URL der PHP-Seite zu hängen:

```
antwort.html?frage=select+*+from+professoren
```

Eine andere Möglichkeit besteht darin, die Query durch ein Formular in einer HTML-Seite zu ermitteln und von dort aus die PHP-Seite aufzurufen. PHP legt dann automatisch eine Variable mit den Namen der in dem Formular verwendeten Feldern an.

Listing 9.23 zeigt eine HTML-Seite mit einem Formular zur Erfassung einer SQL-Query. Die vom Benutzer eingegebene Frage wird übergeben an ein PHP-Script, welches im Listing 9.24 gezeigt wird. Das Ergebnis ist in Abbildung 9.28 zu sehen.

Listing 9.25 zeigt eine HTML-Seite, welche per PHP die Namen und Semesterzahlen der Studenten ermittelt und die Dauer ihres Studiums grafisch durch einen Balken visualisiert. Hierzu wird in dem PHP-Script `balken.php` (Listing 9.26) mithilfe der GD-Library ein blaues Rechteck gezeichnet mit einer Breite gemäß dem übergebenen Parameter `$zahl`. Das Ergebnis ist in Abbildung 9.29 zu sehen.

Listing 9.27 zeigt ein PHP-Script, welches die Lehrbelastung der Professoren ermittelt und die relativen Anteile durch eine Tortengraphik visualisiert. Hierzu werden die ermittelten Namen und Zahlenwerte an das PHP-Script `torte.php` (Listing 9.28) übergeben und dort mithilfe der jpgraph-Library visualisiert. Das Ergebnis ist in Abbildung 9.30 zu sehen.

```
<HTML>
  <HEAD><TITLE>Frage</TITLE></HEAD>
  <BODY>
    <FORM METHOD="POST" ACTION="antwort.php">
       Bitte geben Sie Ihre SQL-Query ein:
      <P><INPUT NAME="frage" SIZE="70"><P>
      <INPUT TYPE="submit" VALUE="Query absetzen">
    </FORM>
  </BODY>
</HTML>
```

Listing 9.23: Quelltext der HTML-Seite frage.html mit Formular zur Ermittlung einer Query

```
<HTML>
  <HEAD> <TITLE>Antwort auf Datenbank-Query</TITLE></HEAD>
  <BODY BGCOLOR="DDDDDD">
    <!-- der Querystring wurde in der Variablen $frage uebergeben -->
    <?php
      $con=mssql_connect("arnold",                    // verbinde mit arnold
                         "erika",                     // als user erika
                         "mustermann");               // mit Passwort mustermann
      mssql_select_db("uni",$con);                    // Datenbank waehlen
      $rs = mssql_query($frage, $con);                // Abfrage stellen
      $z  = mssql_num_rows($rs);                      // Zahl der Zeilen
      $s  = mssql_num_fields($rs);                    // Zahl der Spalten

      echo "Antwort ermittelt durch PHP-Script:<P>";
      echo "<table border cellpadding=3>\n";          // Tabelle beginnen
      echo "<tr>";                                    // Zeile beginnen
      for ($i=0 ; $i<$s ; ++$i) {                     // fuer jede Spalte
        $name = mssql_fetch_field($rs,$i);            // Namen besorgen
        echo "<th>$name->name</th>";                  // und ausgeben
      } echo "</tr>";                                 // Zeile beenden

      while ($tupel = mssql_fetch_array($rs)) {       // pro Tupel
        echo "<tr>";                                  // Zeile beginnen
        for ($i=0 ; $i<$s ; ++$i) {                   // pro Spalte
          echo "<td>$tupel[$i]</td>";                 // Inhalt ausgeben
        } echo "</tr>";                               // Zeile beenden
      }
      echo "</table>\n";                              // Tabelle beenden
      mssql_free_result($rs);                         // Ressource freigeben
      mssql_close($con);                              // Verbind. schliessen
    ?>
  </BODY>
</HTML>
```

Listing 9.24: Quelltext des PHP-Scripts antwort.php mit Berechnung der Antwort

```
<HTML>
  <HEAD>
    <TITLE>Zahl der Semester</TITLE>
  </HEAD>
  <BODY BGCOLOR=SILVER>
  <H2>Studenten und ihre Semesterzahlen</H2>
  <?php

                                                  // Verbindung herstellen
  $con=mssql_connect("arnold",                    // auf Server arnold
                     "erika",                     // als user erika
                     "mustermann");               // mit Passwort mustermann

  mssql_select_db("uni",$con);                    // Datenbank waehlen

  $frage = "select name, semester from studenten order by name";
  $rs = mssql_query($frage, $con);                          // Abfrage stellen

  echo "<table border=1>\n";                                // Tabelle beginnen
  echo "<TR><TH>Student</TH><TH>Studiendauer</TH></TR>";    // Ueberschrift
  while ($tupel = mssql_fetch_array($rs)) {                 // pro Tupel
    echo "<TR>";                                            // Zeile beginnen
    echo "<TD>$tupel[0]</TD>";                              // Namen schreiben
    echo "<TD><IMG SRC=balken.php?zahl=$tupel[1]></TD>";    // Balken malen
    echo "</TR>\n";                                         // Zeile beenden
  }
  echo "</table>\n";                                        // Tabelle beenden

  mssql_free_result($rs);                                   // freigeben
  mssql_close($con);                                        // schliessen
  ?>
</BODY>
</HTML>
```

Listing 9.25:Quelltext des PHP-Scripts semester.php zur Berechnung von dynamischen Grafiken

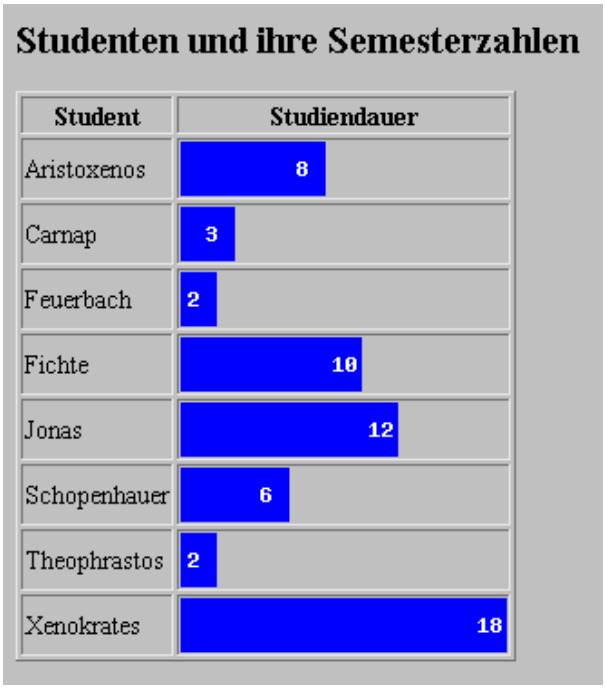
```
<?php
    $breite = $zahl*10;
    $hoehe  = 30;
    $bild = imagecreate($breite, $hoehe);
    $farbe_balken   = imagecolorallocate($bild,   0,   0, 255);
    $farbe_schrift  = imagecolorallocate($bild, 255, 255, 255);
    ImageString($bild,3,$breite-16,8,$zahl,$farbe_schrift);
    header("Content-Type: image/png");
    imagepng($bild);
?>
```

Listing 9.26: Quelltext des PHP-Script balken.php zur Berechnung eines dynamischen Balkens

Antwort ermittelt durch PHP-Script:

Student	Titel
Jonas	Glaube und Wissen
Fichte	Grundzüge
Schopenhauer	Logik
Schopenhauer	Die 3 Kritiken
Schopenhauer	Grundzüge
Schopenhauer	Ethik
Carnap	Logik
Carnap	Bioethik
Carnap	Der Wiener Kreis
Theophrastos	Ethik
Theophrastos	Mäeutik
Feuerbach	Grundzüge
Feuerbach	Glaube und Wissen

Abbildung 9.28: Ergebnis einer SQL-Query, ermittelt durch antwort.php



Studenten und ihre Semesterzahlen

Student	Studiendauer
Aristoxenos	8
Carnap	3
Feuerbach	2
Fichte	10
Jonas	12
Schopenhauer	6
Theophrastos	2
Xenokrates	18

Abbildung 9.29: Dynamische Grafiken erzeugt von balken.php, aufgerufen von semester.php

```
<HTML>
  <HEAD> <TITLE>Lehrbelastung der Professoren</TITLE> </HEAD>
  <BODY BGCOLOR=SILVER>
  <?php
                                                    // Verbindung herstellen
  $con=mssql_connect("arnold",                      // auf Server arnold
                     "erika",                       // als user erika
                     "mustermann");                 // mit Passwort mustermann

  mssql_select_db("uni",$con);                      // Datenbank waehlen

  $frage = "select name, sum(sws)
            from vorlesungen, professoren
            where persnr=gelesenvon group by name";

  $rs = mssql_query($frage, $con);                  // Resultset bilden
  $i=0;
  while ($tupel = mssql_fetch_array($rs)) {         // pro Treffertupel
    $parameter .="namen[$i]=".$tupel[0]."&";        // Namen konkatenieren
    $parameter .="daten[$i]=".$tupel[1]."&";        // SWS konkatenieren
    $i++;
  }

  echo "<IMG SRC=torte.php?$parameter>";           // Torte malen

  mssql_free_result($rs);                           // Ressource freigeben
  mssql_close($con);                                // Verbind. schliessen
  ?>
</BODY>
</HTML>
```

Listing 9.27: Quelltext des PHP-Scripts lehre.php zur Ermittlung der Lehrbelastung

```
<?php
include ("../jpgraph/src/jpgraph.php");
include ("../jpgraph/src/jpgraph_pie.php");

$graph = new PieGraph(600,400,"auto");
$graph->SetShadow();

$graph->title->Set("Lehrbelastung der Professoren");
$graph->title->SetFont(FF_FONT1,FS_BOLD);

$p1 = new PiePlot($daten);
$p1->SetLegends($namen);
$p1->SetCenter(0.4);

$graph->Add($p1);
$graph->Stroke();

?>
```

Listing 9.28: Quelltext des PHP-Script torte.php zur Berechnung einer dynamischen Tortengrafik

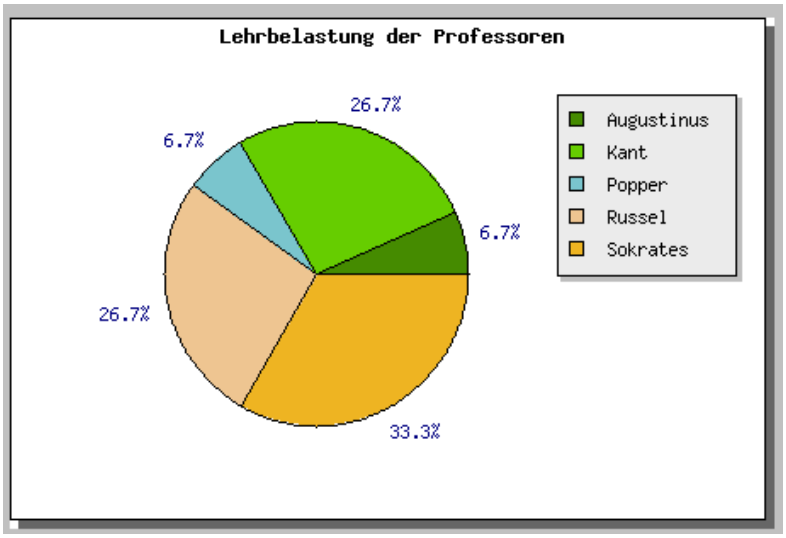


Abbildung 9.30: Dynamische Grafiken, erzeugt von torte.php, aufgerufen von lehre.php