Kapitel 9

Datenbankapplikationen

9.1 MS-Access

MS-Access ist ein relationales Datenbanksystem der Firma *Microsoft*, welches als Einzel- und Mehrplatzsystem unter Windows 95/98 und Windows NT 4.0 läuft. Seine wichtigsten Eigenschaften:

- Der **Schemaentwurf** geschieht menugesteuert (Abbildung 9.2))
- Referenzen zwischen den Tabellen werden in Form von **Beziehungen** visualisiert.
- **Queries** können per SQL oder menugesteuert abgesetzt werden (Abbildung 9.1).
- **Formulare** definieren Eingabemasken, die das Erfassen und Updaten von Tabellendaten vereinfachen (Abbildung 9.3)
- **Berichte** fassen Tabelleninhalte und Query-Antworten in formatierter Form zusammen und können als Rich-Text-Format exportiert werden (Listing 9.1 + Abbildung 9.4).
- Als **Frontend** eignet es sich für relationale Datenbanksysteme, die per ODBC-Schnittstelle angesprochen werden können.

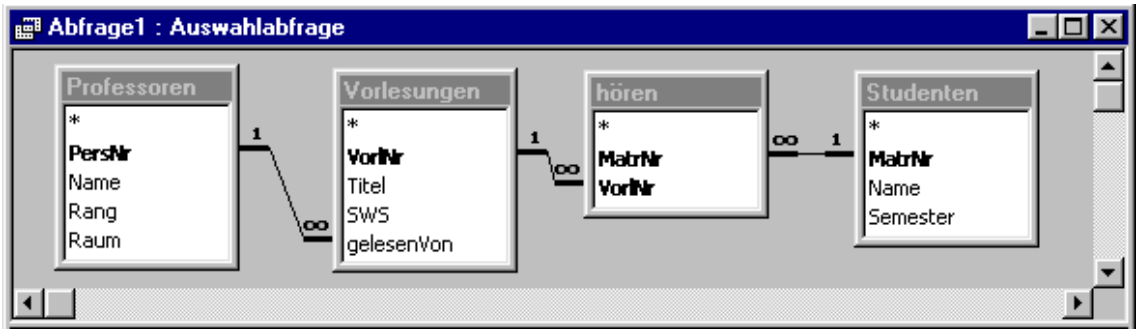


Abbildung 9.1: in MS-Access formulierte Abfrage

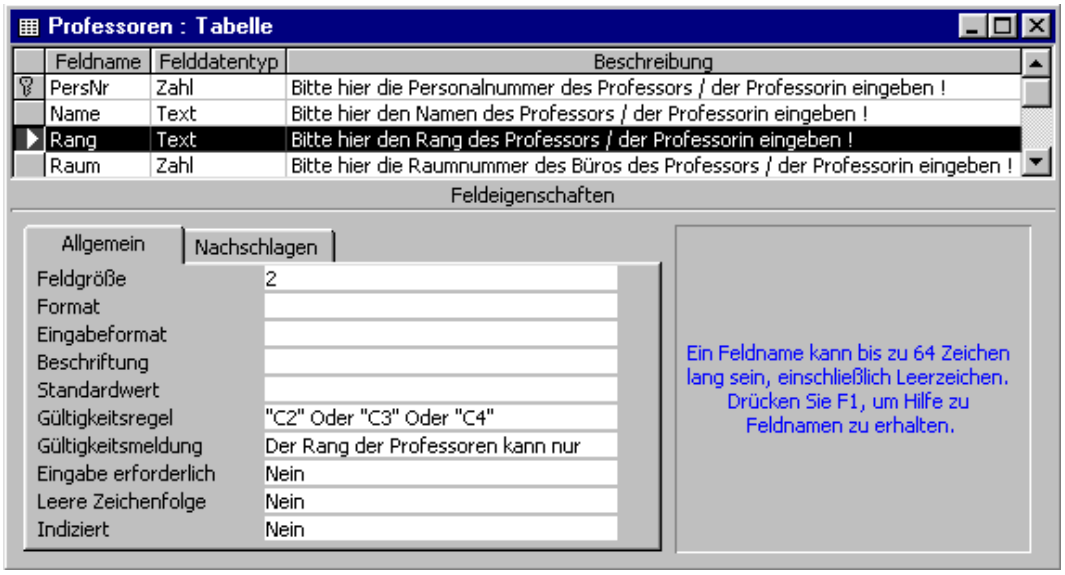


Abbildung 9.2: Schemadefinition in MS-Access

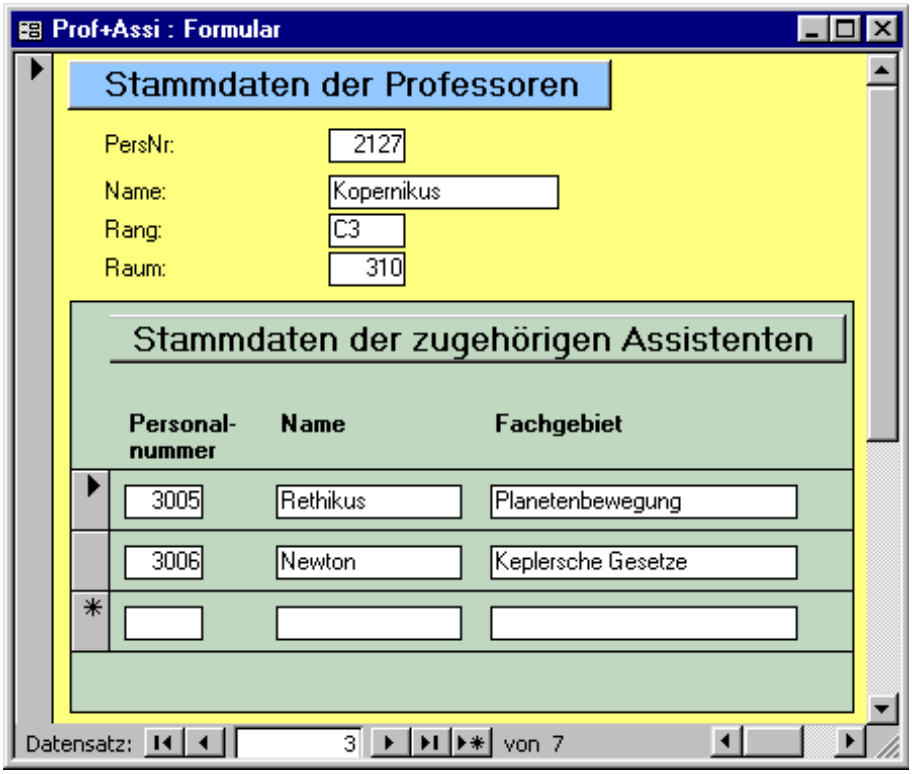
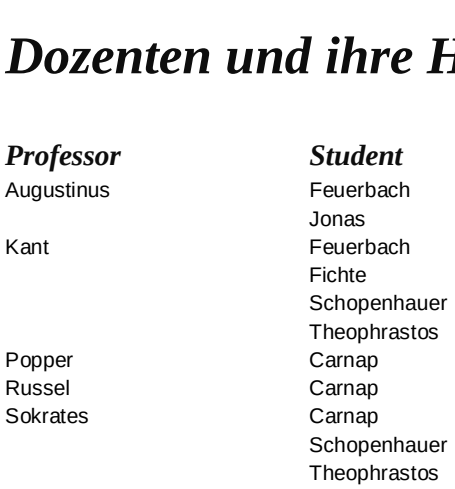


Abbildung 9.3: durch MS-Access-Formular erzeugte Eingabemaske

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 9.4 generiert.

```
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 9.4



Dozenten und ihre Hörer

Professor	Student
Augustinus	Feuerbach
	Jonas
Kant	Feuerbach
	Fichte
	Schopenhauer
	Theophrastos
Popper	Carnap
Russel	Carnap
Sokrates	Carnap
	Schopenhauer
	Theophrastos

Abbildung 9.4: Word-Ausgabe eines MS-Access-Berichts, basierend auf Listing 9.1

9.2 PL/SQL

Das Oracle-Datenbanksystem bietet eine prozedurale Erweiterung von SQL an, genannt *PL/SQL*. Hiermit können SQL-Statements zu namenlosen Blöcken, Prozeduren oder Funktionen zusammengefaßt und ihr Ablauf durch Kontrollstrukturen gesteuert werden.

Sei eine Tabelle *konto* mit Kontonummern und Kontoständen angelegt durch

```
create table konto (nr int, stand int);
```

Listing 9.1 zeigt einen namenlosen PL/SQL-Block, der 50 Konten mit durchlaufender Numerierung einrichtet und alle Kontostände mit 0 initialisiert.

```
declare
  i int;
begin
  for i in 1..50 loop
    insert into konto
    values (i, 0);
  end loop;
end;
```

Listing 9.1: Namenloser PL/SQL-Block

Listing 9.2 zeigt eine benannte PL/SQL-Prozedur, welche versucht, innerhalb der Tabelle *konto* eine Überweisung durchzuführen und danach das Ergebnis in zwei Tabellen

```
create table gebucht   (datum DATE, nr_1 int, nr_2 int, betrag int);
create table abgelehnt (datum DATE, nr_1 int, nr_2 int, betrag int);
```

in Form einer Erfolgs- bzw. Mißerfolgsmeldung festhält.

```
create or replace procedure ueberweisen     -- lege Prozedur an
( x int,                                    -- Konto-Nr. zum Belasten
  y int,                                    -- Konto-Nr. fuer Gutschrift
  betrag int)                               -- Ueberweisungsbetrag
IS
s konto.stand%TYPE;                         -- lokale Variable vom Typ konto.stand
begin
  SELECT stand INTO s FROM konto            -- hole Kontostand nach s
  WHERE nr = x FOR UPDATE OF stand;         -- von Konto-Nr. x und sperre Konto
  IF s < betrag THEN                        -- falls Konto ueberzogen wuerde
    INSERT INTO abgelehnt                   -- notiere den Fehlschlag
    VALUES (SYSDATE, x, y, betrag);         -- in der Tabelle abgelehnt
  ELSE
    UPDATE konto                            -- setze in der Tabelle konto
    SET stand = stand-betrag WHERE nr = x;  -- neuen Kontostand bei Konto x ein
    UPDATE konto                            -- setze in der Tabelle konto
    SET stand = stand+betrag WHERE nr = y;  -- neuen Kontostand bei Konto y ein
    INSERT INTO gebucht                     -- notiere die Ueberweisung
    VALUES (SYSDATE, x, y, betrag);         -- in der Tabelle gebucht
  END IF;
  COMMIT;
end;
```

Listing 9.2: PL/SQL-Prozedur

Im Gegensatz zu einem konventionellen Benutzerprogramm wird eine PL/SQL-Prozedur in der Datenbank gespeichert. Sie wird aufgerufen und (später) wieder entfernt durch

```
execute ueberweisung (42,37,50);
drop procedure ueberweisung;
```

In Listing 9.3 wird eine Funktion `f2c` definiert, die eine übergebene Zahl als Temperatur in Fahrenheit auffaßt und den Wert nach Celsius umrechnet.

```
create or replace function f2c                -- definiere eine Funktion f2c
  (fahrenheit IN number)                       -- Eingangsparameter vom Typ number
  return number                                -- Ausgangsparameter vom Typ number
is
  celsius number;                              -- lokale Variable
  begin                                        -- Beginn des Funktionsrumpfes
    celsius := (5.0/9.0)*(fahrenheit-32);      -- Umrechnung nach Celsius
    return celsius;                            -- Rueckgabe des Funktionswertes
  end f2c;                                     -- Ende der Funktion
```

Listing 9.3: PL/SQL-Funktion

Der Aufruf der Funktion erfolgt innerhalb einer SQL-Abfrage:

```
select nr, f2c(nr) from daten;
```

9.3 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*.

Oracle unterstützt Embedded SQL im Zusammenspiel mit den Programmiersprachen C und C++ durch den *Pre-Compiler Pro*C/C++*. Abbildung 9.5 zeigt den prinzipiellen Ablauf: Das mit eingebetteten SQL-Statements formulierte Host-Programm `hallo.pc` wird zunächst durch den Pre-Compiler unter Verwendung von SQL-spezifischen Include-Dateien in ein ANSI-C-Programm `hallo.c` überführt. Diese Datei übersetzt ein konventioneller C-Compiler unter Verwendung der üblichen C-Include-Files in ein Objekt-File `hallo.o`. Unter Verwendung der Oracle Runtime Library wird daraus das ausführbare Programm `hallo.exe` gebunden.

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

```
EXEC SQL COMMIT;
```

die seit Transaktionsbeginn durchgeführten Änderungen permanent gemacht.

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

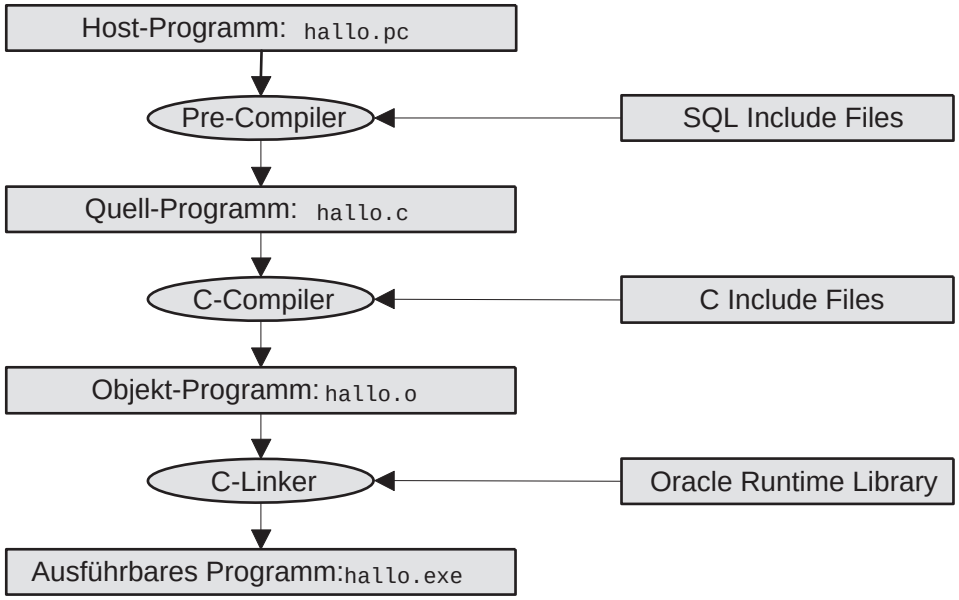


Abbildung 9.5: Vom Hostprogramm zum EXE-File

Datenbankwerte und Statusinformationen an das Host-Programm. Hostvariablen werden innerhalb von SQL-Statements durch einen vorangestellten Doppelpunkt (:) gekennzeichnet.

Für Hostvariablen, die Datenbankattributen vom Typ VARCHAR2 entsprechen, empfiehlt sich eine Definition nach folgendem Beispiel:

```
VARCHAR fachgebiet[20];
```

Daraus generiert der Pre-Compiler eine Struktur mit einem Character-Array und einer Längen-Komponente:

```
struct
{
    unsigned short len;
    unsigned char  arr[20];
} fachgebiet;
```

Bevor diese Struktur als Null-terminierter String genutzt werden kann, muß das Null-Byte explizit gesetzt werden:

```
fachgebiet.arr[fachgebiet.len] = '\0';
```

Mit einer Hostvariable kann eine optionale *Indikator-Variable* assoziiert sein, welche Null-Werte überträgt oder entdeckt. Folgende Zeilen definieren unter Verwendung des Oracle-spezifischen Datentyps `VARCHAR` eine Struktur `prof_rec` zum Aufnehmen eines Datensatzes der Tabelle *Professoren* sowie eine Indikator-Struktur `prof_ind` zum Aufnehmen von Status-Information.

```
struct{
  int     persnr;
  VARCHAR name [15];
  char    rang [2];
  int     raum;
} prof_rec;

struct {
  short persnr_ind;
  short name_ind;
  short rang_ind;
  short raum_ind;
} prof_ind;
```

Folgende Zeilen transferieren einen einzelnen Professoren-Datensatz in die Hostvariable `prof_rec` und überprüfen mit Hilfe der Indikator-Variable `prof_ind`, ob eine Raumangabe vorhanden war.

```
EXEC SQL SELECT PersNr, Name, Rang, Raum
         INTO   :prof_rec INDICATOR :prof_ind
         FROM   Professoren
         WHERE  PersNr = 2125;

if (prof_ind.raum_ind == -1)
   printf("Personalnummer %d hat keine Raumngabe \n", prof_rec.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 prof_cursor CURSOR FOR
  SELECT PersNr, Name, Rang, Raum
  FROM   Professoren
  WHERE  Rang='C4';

EXEC SQL OPEN prof_cursor;
EXEC SQL WHENEVER NOT FOUND DO break;

for (;;)
{
  EXEC SQL FETCH prof_cursor INTO :prof_rec;
  printf("Verarbeite Personalnummer %d\n", prof_rec.persnr);
}

EXEC SQL CLOSE prof_cursor;
```

Listing 9.4. zeigt ein Embedded-SQL-Programm, die davon erzeugte Ausgabe zeigt Abb. 9.6.

```
#include <stdio.h>                                      /* C-Header fuer I/O         */
#include <sqlca.h>                                      /* SQL-Communication Area    */

EXEC SQL BEGIN DECLARE SECTION;                         /* Beginn der Deklarationen  */

  char * username = "erika";                            /* Benutzername              */
  char * passwort = "mustermann";                       /* Benutzerpasswort          */
  char * dbdienst = "dbs99";                            /* Datenbankdienst           */


  struct{                                               /* Daten-Record              */
    int      persnr;                                    /* Personalnummer            */
    VARCHAR  name[15];                                  /* Name                      */
    char     rang[2];                                   /* Rang                      */
    int      raum;                                      /* Raum                      */
  } prof_rec;

  struct {                                              /* Indikator-Record          */
    short persnr_ind;                                   /* Indikator fuer PersNr     */
    short name_ind;                                     /* Indikator fuer Name       */
    short rang_ind;                                     /* Indikator fuer Rang       */
    short raum_ind;                                     /* Indikator fuer Raum       */
  } prof_ind;

  char eingaberang[3];                                  /* lokale Variable           */

EXEC SQL END DECLARE SECTION;                           /* Ende der Deklarationen    */

void main()
{
    EXEC SQL WHENEVER SQLERROR GOTO fehler;             /* ggf. zur Fehlermarke      */
    EXEC SQL DECLARE dbname DATABASE;                   /* deklariere Datenbankname  */

    EXEC SQL CONNECT :username                          /* mit Benutzername          */
    IDENTIFIED BY    :passwort                          /* mit Passwort              */
    AT dbname USING  :dbdienst;                         /* mit Datenbankdienst       */

    printf("Bitte Rang eingeben: ");                    /* Aufforderung zur Eingabe  */
    scanf("%s", eingaberang);                           /* Einlesen der Eingabe      */
    printf("Mit Rang %s gespeichert:\n",eingaberang);   /* Ausgabeankuendigung       */

    EXEC SQL at dbname DECLARE prof_cursor CURSOR FOR   /* oeffne Cursor             */
    SELECT PersNr, Name, Rang, Raum                     /* fuer alle Attibute von    */
    FROM   oliver.Professoren                           /* oliver's Professoren      */
    WHERE  Rang = :eingaberang;                         /* mit vorgegebenem Rang     */

    EXEC SQL OPEN prof_cursor;                          /* oeffne den Cursor         */
    EXEC SQL WHENEVER NOT FOUND DO break;               /* falls keine Records:      */
                                                        /* hinter die Schleife       */
    for (;;)
    {
      EXEC SQL FETCH prof_cursor                        /* arbeite Cursor ab         */
               INTO :prof_rec INDICATOR :prof_ind;      /* incl. Indikatorvariable   */
```

```
      prof_rec.name.arr[prof_rec.name.len]='\0';        /* setze Nullbyte           */
      printf("%d %-10s %s",                             /* gib formatiert aus       */
             prof_rec.persnr,                           /* die Personalnummer       */
             prof_rec.name.arr,                         /* den Professorennamen     */
             prof_rec.rang);                            /* den Professorenrang      */
      if (prof_ind.raum_ind == -1) printf(" ???\n");    /* falls kein Null-Wert     */
                else printf(" %d\n",prof_rec.raum);     /* gib Raumnummer aus       */
    }

    EXEC SQL CLOSE prof_cursor;                         /* schliesse den Cursor     */

    EXEC SQL at dbname COMMIT RELEASE; return;          /* beende Verbindung        */

    fehler:
       EXEC SQL WHENEVER SQLERROR CONTINUE;             /* im Falle eines Fehlers   */
       printf("Fehler: %70s\n",sqlca.sqlerrm.sqlerrmc); /* gib Fehlermeldung aus    */
       EXEC SQL at dbname ROLLBACK RELEASE;             /* Rollback und beenden     */
       return;
}
```

Listing 9.4: Quelltext von Embedded-SQL-Programm

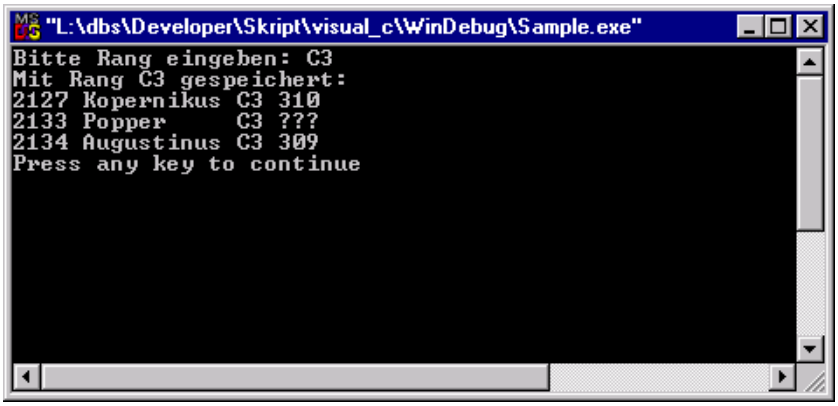


Abbildung 9.6: Ausgabe des Embedded-SQL-Programms von Listing 9.4

9.4 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.

JDBC ermöglicht drei Dinge:

1. eine Verbindung zur Datenbank aufzubauen,
2. SQL-Anweisungen abzusenden,
3. die Ergebnisse zu verarbeiten.

Der folgende Quelltext zeigt ein einfaches Beispiel für diese drei Schritte:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");        // Treiber laden
Connection con  = DriverManager.getConnection         // Verbindung herst.
                  ("jdbc:odbc:dbserv.dsn","erika","Mustermann");
Statement  stmt = con.createStatement();
ResultSet  rs   = stmt.executeQuery("select persnr, name from Professoren");
while (rs.next()){
  int x    = rs.getInt("persnr");
  String s = rs.getString("name");
  System.out.println("Professor "+s" hat die Personalnummer "+x);
}
```

Abbildung 9.7 zeigt die von Listing 9.5 erzeugte Ausgabe einer Java-Applikation auf der Konsole.

Abbildung 9.7: Ausgabe einer Java-Applikation

```
/* ShowJdbc.java (c) Stefan Rauch 1999 */

import java.sql.*;                                  // Import der SQL-Klassen

public class ShowJdbc {

  public static void main(String args[]) {

    String url = "jdbc:odbc:dbserv.dsn";           // Treiber-url f. Verbindung
    Connection con;                                // Verbindungs-Objekt
    Statement stmt;                                // Instanz der Verbindung
                                                   // sendet query und liefert
                                                   // Ergebnis (ResultSet)
    String user   = "Erika";
    String passwd = "Mustermann";
    String query  = "select * from Professoren";

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

    try {
      con = DriverManager.getConnection(url, user, passwd); // Erst. der Verbindung
      stmt = con.createStatement();                // Instantiierung des Statement
      ResultSet rs = stmt.executeQuery(query);     // Ergebnis in 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();                                // Schliessen des Statements
      con.close();                                 // Schliessen der Verbindung

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

Listing 9.5: Quelltext der Java-Applikation ShowJdbc.java

Listing 9.6 zeigt den Quelltext einer HTML-Seite mit dem Aufruf eines Java-Applets. Abbildung 9.8 zeigt die Oberfläche des Applets. Listing 9.7 zeigt den Quelltext des Applets.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<!--last modified on Thursday, June 03, 1999 11:37 PM -->
<HTML>
  <HEAD>
    <META HTTP-EQUIV="Content-Type" CONTENT="text/html;CHARSET=iso-8859-1">
    <META NAME="Author" Content="Stefan Rauch">
    <TITLE>JDBC Test-Applet </TITLE>
  </HEAD>
  <BODY BGCOLOR="#d3e2cf">
  <P ALIGN="CENTER"><FONT SIZE="5">
  <B>Demo-Applet f&uuml;r JDBC-Datenbankzugriff </B></FONT></P>
  <CENTER>
  <P> <FONT SIZE="4">
    <B>SQL-Abfrage an dbserv als user Erika mit Passwort Mustermann</B> </FONT>
  </P>
  <P><APPLET CODEBASE="../skript/Applets" CODE="JdbcApplet" WIDTH="700" HEIGHT="400" ALIGN="BOTTOM">
  </APPLET>
  </CENTER>
  </BODY>
</HTML>
```

Listing 9.6: Quelltext einer HTML-Seite zum Aufruf eines Applets

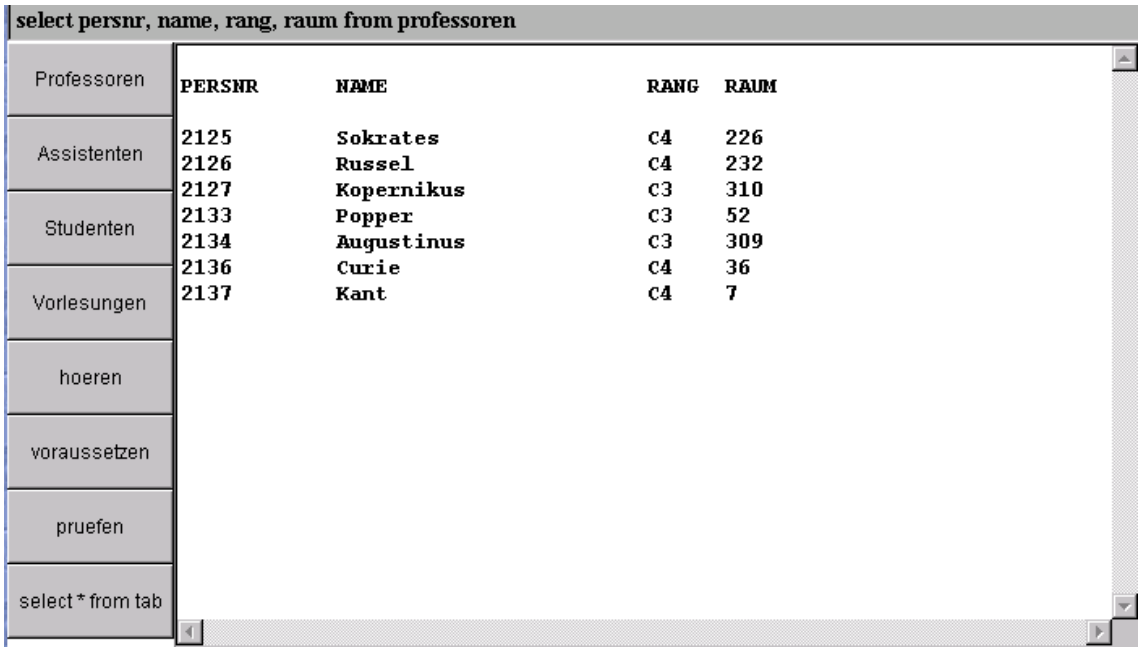


Abbildung 9.8: Java-Applet mit JDBC-Zugriff auf Oracle-Datenbank

```
/* 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;

/* Erstellen der Buttons, die den Inhalt der benannten Tabellen komplett ausgeben */
  Button qu1 = new Button("Professoren");
  Button qu2 = new Button("Assistenten");
  Button qu3 = new Button("Studenten");
  Button qu4 = new Button("Vorlesungen");
  Button qu5 = new Button("hoeren");
  Button qu6 = new Button("voraussetzen");
  Button qu7 = new Button("pruefen");
  Button qu8 = new Button("select * from tab");

/* Verbindungsobjekt um die Verbindung zum DBMS aufzubauen */
  Connection con;

/* Statement-Objekt zur Kommunikation mit DBMS */
  Statement stmt;


  public JdbcApplet() {
  }

/* Das Applet initialisieren */

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

/* ActionListener fuer das Eingabefeld         */
/* gibt den Text an Methode execQuery() weiter */
      inputField.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
          String q = inputField.getText();
          execQuery(q);
        }
```

```
      });
      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," +
                    "to_char(gebdatum,'dd:mm:YYYY') as Geburtsdatum from Professoren");
        }
      });
      qu2.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
          execQuery("select persnr, name, fachgebiet, boss," +
                    "to_char(gebdatum,'dd:mm:YYYY') as Geburtsdatum from Assistenten");
        }
      });
      qu3.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
          execQuery("select matrnr, name, semester," +
                    "to_char(gebdatum,'dd:mm:YYYY') 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 * from tab");
        }
      });

/* Hinzufuegen und Anordnen der einzelnen Elemente des Applets */
      this.add(outputArea, BorderLayout.CENTER);
```

```
      this.add(inputField, BorderLayout.NORTH);
      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();
    }

/* Verbindungsaufbau mit dem DBMS */
    String url    = "jdbc:odbc:dbserv.dsn";
    String user   = "Erika";
    String passwd = "Mustermann";

/* JDBC-ODBC Brueckentreiber wird genutzt und vom DriverManager registriert */
    try {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch(java.lang.ClassNotFoundException ex) {
      System.err.print("ClassNotFoundException: ");
      System.err.println(ex.getMessage());
    }

/* Verbindung zum DBMS wird geoeffnet */
    try {
      con = DriverManager.getConnection(url, user, passwd);
    }
    catch(SQLException ex) {
      outputArea.setText("SQLException: " + ex.getMessage());
    }
  }


/* Methode, die die jeweiligen SQL-Querys an das DBMS weiterleitet und die */
/* in einem ResultSet empfangenen Ergebnisse in der TextArea darstellt     */

  void execQuery(String query) {

    try {

/* Fuer jede Abfrage muss ein Statement-Objekt instantiiert werden */
      stmt = con.createStatement();

/* Ausfuehren der Abfrage und Speichern der Ergebnisse im ResultSet rs */
      ResultSet rs = stmt.executeQuery(query);

/* z := Anzahl der Spalten des Ergebnisses */
```

```
      int z = rs.getMetaData().getColumnCount();
      outputArea.setText("\n");
      outputArea.setVisible(false);

/* Fuer jede Spalte wird zunaechst der Spaltenname formatiert ausgegeben */
      for (int i=1;i<=z;i++) {
        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");

/* Der Inhalt der Ergebnismenge wird zeilenweise formatiert in der TextArea ausgegeben */
        String arg;
        while(rs.next()) {
          outputArea.append("\n");
          for (int i=1;i<=z;i++) {
            arg=rs.getString(i);
            int len;
            if (arg != null) {
              len=arg.length();
              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();

/* Abfangen von etwaigen SQL-Fehlermeldungen und Ausgabe derer in der TextArea */
      }catch(SQLException ex) {
        outputArea.setText("SQLException: " + ex.getMessage());
      }

  }

}
```

Listing 9.7: Quelltext von Java-Applet

9.5 Cold Fusion

Cold Fusion ist ein Anwendungsentwicklungssystem der Firma Allaire 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 Abbilung 9-9 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 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.

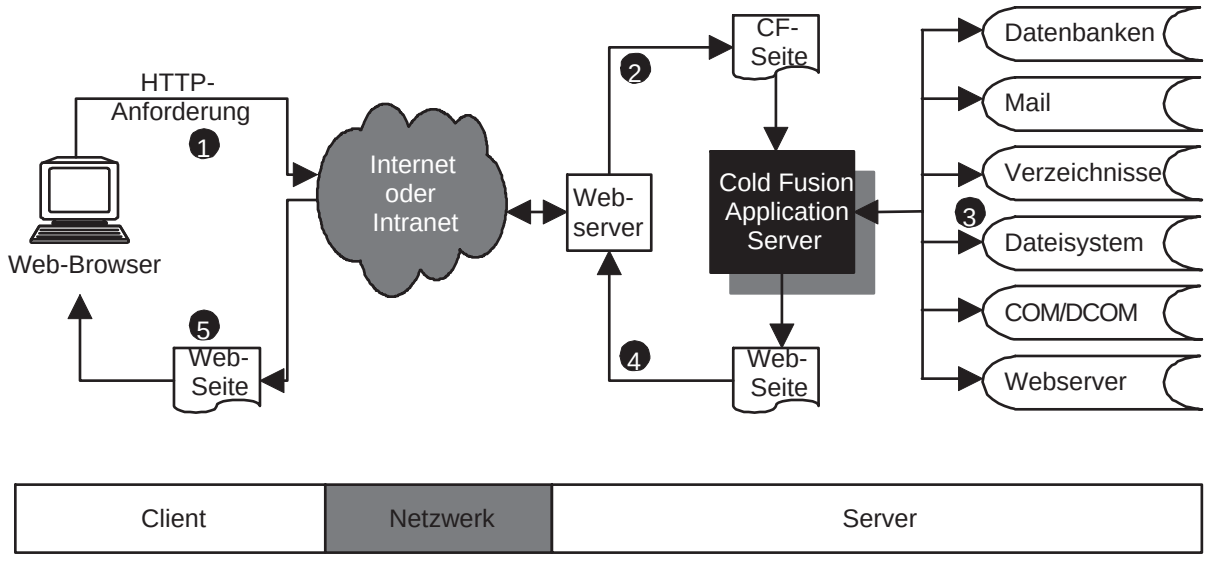


Abbildung 9.9: Arbeitsweise von Coldfusion

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://iuk-www.vdv.uni-osnabrueck.de/vpv/sommer99/index.cfm`

CF-Online-Dokumentation: `http://cfserv.rz.uni-osnabrueck.de/cfdocs` .

Listing 9.8 zeigt eine unformatierte Ausgabe einer SQL-Query.

```
<CFQUERY NAME       = "Studentenliste"
         USERNAME   = "erika"
         PASSWORD   = "mustermann"
         DATASOURCE = "dbserv.dsn"
         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 9.8: Quelltext von studliste.cfm

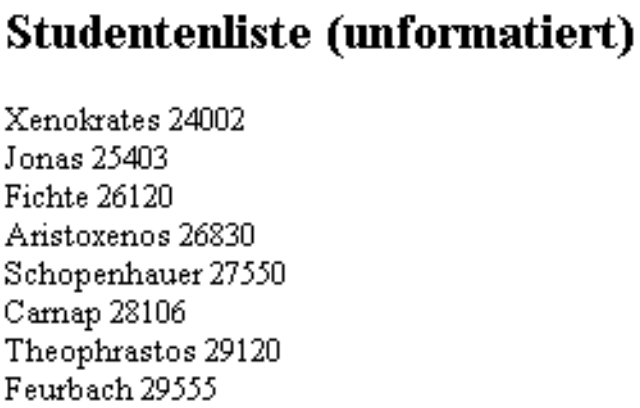


Abbildung 9.10: Screenshot von studliste.cfm

Listing 9.9 zeigt die formatierte Ausgabe einer SQL-Query unter Verwendung einer HTML-Tabelle.

```
<CFQUERY NAME       = "Studententabelle"
         USERNAME   = "erika"
         PASSWORD   = "mustermann"
         DATASOURCE = "dbserv.dsn"
         DBTYPE     = "ODBC">
  SELECT matrnr, name, to_char(gebdatum,'DD.MM.YYYY') as geburt from studenten
  WHERE  (sysdate-gebdatum)/365 > 30
</CFQUERY>

<HTML>
  <HEAD>
    <TITLE> Studententabelle </TITLE>
  </HEAD>
  <BODY>
    <H2> Studenten 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 9.9: Quelltext von studtabelle.cfm

Studenten als HTML-Tabelle

Matrikelnummer	Nachname	Geburtsdatum
26120	Fichte	04.12.1967
26830	Aristoxenos	05.08.1943
27550	Schopenhauer	22.06.1954
29120	Theophrastos	19.04.1948
29555	Feurbach	12.02.1961

Abbildung 9.11: Screenshot von studtabelle.cfm

Listing 9.10 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 9.10: Quelltext von studformular.cfm

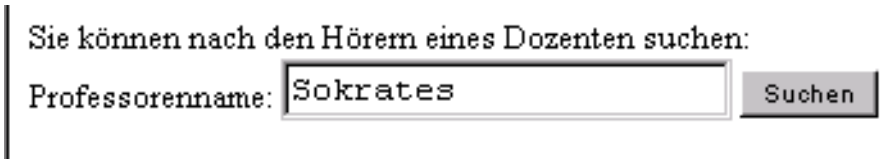


Abbildung 9.12: Screenshot von studformular.cfm

Der vom Formular `studformular.cfm` erfaßte Name wird übergeben an die Datei `studsuche.cfm`, welche im Listing 9.11 gezeigt wird.

```
<CFQUERY NAME       = "Studentensuche"
         USERNAME   = "erika"
         PASSWORD   = "mustermann"
         DATASOURCE = "dbserv.dsn"
         DBTYPE     = "ODBC" >
  SELECT unique 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 9.11: Quelltext von studsuche.cfm

Professor Sokrates hat folgende Hörer:

27550 Schopenhauer
28106 Carnap
29120 Theophrastos

Abbildung 9.13: Screenshot von studsuche.cfm

Listing 9.12 zeigt eine HTML-Tabelle mit sensitiven Links für die Professoren.

```
<CFQUERY NAME       = "Vorlesungstabelle"
         USERNAME   = "erika"      PASSWORD = "mustermann"
         DATASOURCE = "dbserv.dsn" 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 9.12: Quelltext von vorltabelle.cfm

Vorlesungen mit sensitiven Links

Vorlesungsnr	Titel	Dozent
5001	Grundzuge	Kant
5041	Ethik	Sokrates
5043	Erkenntnistheorie	Russel
5049	Maeutik	Sokrates
4052	Logik	Sokrates
5052	Wissenschaftstheorie	Russel
5216	Bioethik	Russel
5259	Der Wiener Kreis	Popper
5022	Glaube und Wissen	Augustinus
4630	Die 3 Kritiken	Kant

Abbildung 9.14: Screenshot von vorltabelle.cfm

Die in Listing 9.12 ermittelte Personalnummer eines Professors wird in Form einer URL an die in Listing 9.13 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 = "dbserv.dsn"
         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 9.13: Quelltext von profinfo.cfm

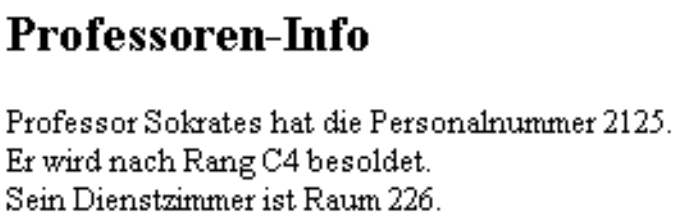


Abbildung 9.15: Screenshot von profinfo.cfm

Listing 9.14 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 9.14: Quelltext von profinsertform.cfm

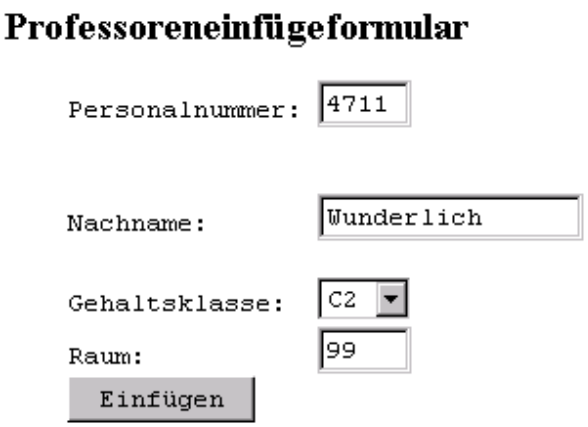


Abbildung 9.16: Screenshot von profinsertform.cfm

Die von Listing 9.14 übermittelten Daten werden in Listing 9.15 zum Einfügen verwendet. Anschließend erfolgt eine Bestätigung.

```
<CFQUERY NAME       = "Profinsert"
         USERNAME   = "erika"
         PASSWORD   = "mustermann"
         DATASOURCE = "dbserv.dsn"
         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 9.15: Quelltext von profinsert.cfm

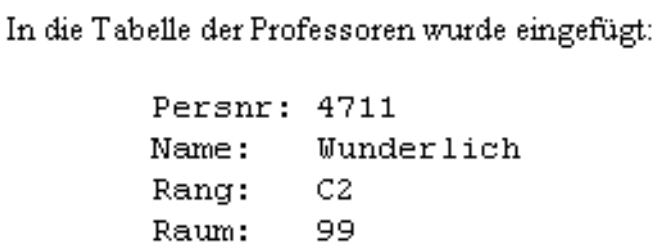


Abbildung 9.17: Screenshot von profinsert.cfm

Listing 9.16 zeigt eine Tabelle mit einer Form zum Löschen eines Professors.

```
<CFQUERY NAME       = "Professorentabelle"
         USERNAME   = "erika" PASSWORD    = "mustermann"
         DATASOURCE = "dbserv.dsn" 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 9.16: Quelltext von profdeleteform.cfm

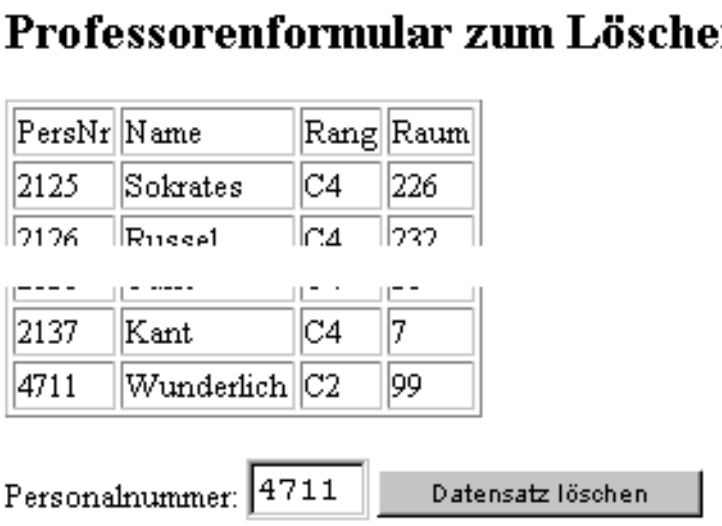


Abbildung 9.18: Screenshot von profdeleteform.cfm

Die in Listing 9.16 ermittelte Personalnummer eines Professors wird in Listing 9.17 zum Löschen verwendet. Anschließend erfolgt eine Bestätigung.

```
<CFQUERY NAME       = "Profdelete"
         USERNAME   = "erika"
         PASSWORD   = "mustermann"
         DATASOURCE = "dbserv.dsn"
         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 9.17: Quelltext von profdelete.cfm

Abbildung 9.19: Screenshot von profdelete.cfm

Listing 9.18 zeigt ein Formular zum Suchen nach einem Professorendatensatz unter Verwendung des Wildcard-Zeichens %.

```
<HTML>
  <HEAD>
    <TITLE> Professorenupdate </TITLE>
  </HEAD>

  <BODY>
    Bitte geben Sie Suchkriterien ein,
    um gezielt einen Professor zum UPDATE zu suchen.<BR>
    <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="Personalnummer muss ganzzahlig 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>
                      <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 ganzzahlig sein"></TD></TR>

             <!--- Hiddenfield zur spaeteren Steuerung --->
             <INPUT TYPE="HIDDEN" NAME="i" VALUE="1">
    <TR><TD><INPUT TYPE="Submit" VALUE="Prof suchen"></TD><TD></TD></TR>
    </TABLE>
    </FORM>
  </BODY>
</HTML>
```

Listing 9.18: Quelltext von profupdateformular.cfm

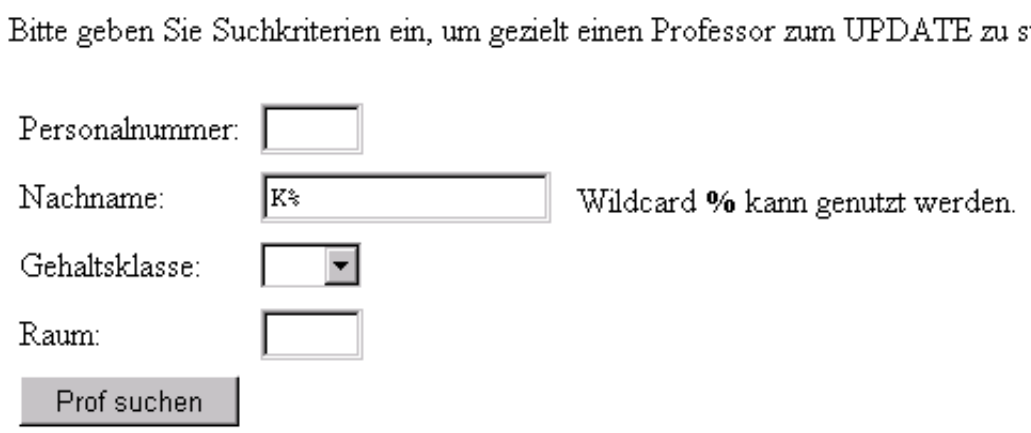


Abbildung 9.20: Screenshot von profupdateformular.cfm

Die in Listing 9.18 gefundenen Treffer können im Listing 9.19 durchlaufen werden und anschließend editiert werden.

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

<CFQUERY NAME       = "ProfAbfr"
         USERNAME   = "erika"
         PASSWORD   = "mustermann"
         DATASOURCE = "dbserv.dsn"
         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 9.19: Quelltext von profupdate.cfm

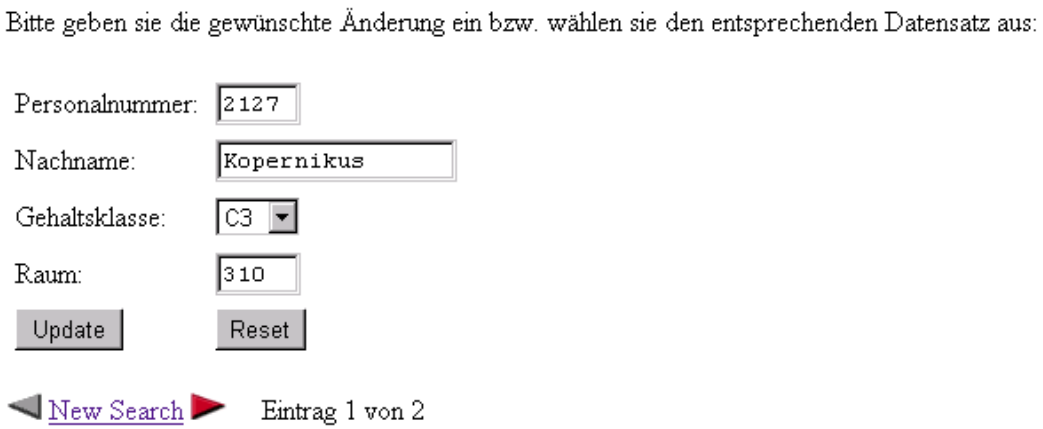


Abbildung 9.21: Screenshot von profupdate.cfm

Listing 9.20 zeigt die Durchführung der Update-Operation auf dem in Listing 9.19 ausgewählten Professorendatensatz.

```
<!--- erstellt von Ralf Kunze --->
<CFQUERY NAME       = "Profupdate"
         USERNAME   = "erika"
         PASSWORD   = "mustermann"
         DATASOURCE = "dbserv.dsn"
         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:   #Form.ProfRaum#
      </PRE>
    </CFOUTPUT>
    <A HREF="profupdateformular.cfm">New Search</A>
  </BODY>
</HTML>
```

Listing 9.20: Quelltext von update.cfm

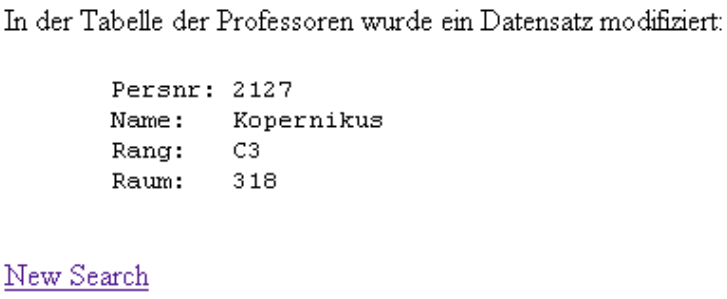


Abbildung 9.22: Screenshot von update.cfm

Kapitel 10

Relationale Entwurfstheorie

10.1 Funktionale Abhängigkeiten

Gegeben sei ein Relationenschema $\mathcal{R}$ mit einer Ausprägung R. Eine *funktionale Abhängigkeit* (engl. *functional dependency*) stellt eine Bedingung an die möglichen gültigen Ausprägungen des Datenbankschemas dar. Eine funktionale Abhängigkeit, oft abgekürzt als FD, wird dargestellt als

$$\alpha \rightarrow \beta$$

Die griechischen Buchstaben α und β repräsentieren Mengen von Attributen. Es sind nur solche Ausprägungen zulässig, für die gilt:

$$\vee\ r, t \in R:\ r.\alpha = t.\alpha \Rightarrow r.\beta = t.\beta$$

D. h., wenn zwei Tupel gleiche Werte für alle Attribute in α haben, dann müssen auch ihre β-Werte übereinstimmen. Anders ausgedrückt: Die α-Werte bestimmen eindeutig die β-Werte; die β-Werte sind funktional abhängig von den α-Werten.

Die nächste Tabelle zeigt ein Relationenschema $\mathcal{R}$ über der Attributmenge $\{A,\ B,\ C,\ D\}$.

R			
A	B	C	D
a_4	b_2	c_4	d_3
a_1	b_1	c_1	d_1
a_1	b_1	c_1	d_2
a_2	b_2	c_3	d_2
a_3	b_2	c_4	d_3

Aus der momentanen Ausprägung lassen sich z. B. die funktionalen Abhängigkeiten $\{A\} \rightarrow \{B\}, \{A\} \rightarrow \{C\}, \{C, D\} \rightarrow \{B\}$ erkennen, hingegen gilt nicht $\{B\} \rightarrow \{C\}$.

Ob diese Abhängigkeiten vom Designer der Relation als semantische Konsistenzbedingung verlangt wurden, läßt sich durch Inspektion der Tabelle allerdings nicht feststellen.

Statt $\{C, D\} \to \{B\}$ schreiben wir auch $CD \to B$. Statt $\alpha \cup \beta$ schreiben wir auch $\alpha\beta$.

Ein einfacher Algorithmus zum Überprüfen einer (vermuteten) funktionalen Abhängigkeit $\alpha \to \beta$ in der Relation R lautet:

1. sortiere R nach α-Werten
2. falls alle Gruppen bestehend aus Tupeln mit gleichen α-Werten auch gleiche β-Werte aufweisen, dann gilt $\alpha \to \beta$, sonst nicht.

10.2 Schlüssel

In dem Relationenschema $\mathcal{R}$ ist $\alpha \subseteq \mathcal{R}$ ein *Superschlüssel*, falls gilt

$$\alpha \to \mathcal{R}$$

Der Begriff Superschlüssel besagt, daß alle Attribute von α abhängen aber noch nichts darüber bekannt ist, ob α eine minimale Menge von Attributen enthält.

Wir sagen: β ist *voll funktional abhängig* von α, in Zeichen $\alpha \dot{\to} \beta$, falls gilt

1. $\alpha \to \beta$
2. $\forall A \in \alpha : \alpha \Leftrightarrow \{A\} \not\to \beta$

In diesem Falle heißt α *Schlüsselkandidat*. Einer der Schlüsselkandidaten wird als *Primärschlüssel* ausgezeichnet.

Folgende Tabelle zeigt die Relation *Städte*:

Städte			
Name	BLand	Vorwahl	EW
Frankfurt	Hessen	069	650000
Frankfurt	Brandenburg	0335	84000
München	Bayern	089	1200000
Passau	Bayern	0851	50000
...	...	...	...

Offenbar gibt es zwei Schlüsselkandidaten:

1. {Name, BLand}
2. {Name, Vorwahl}

10.3 Bestimmung funktionaler Abhängigkeiten

Wir betrachten folgendes Relationenschema:

ProfessorenAdr : {[PersNr, Name, Rang, Raum,
Ort, Straße, PLZ, Vorwahl, BLand, Landesregierung]}

Hierbei sei *Ort* der eindeutige Erstwohnsitz des Professors, die *Landesregierung* sei die eindeutige Partei des Ministerpräsidenten, *BLand* sei der Name des Bundeslandes, eine Postleitzahl (*PLZ*) ändere sich nicht innerhalb einer Straße, Städte und Straßen gehen nicht über Bundesgrenzen hinweg.

Folgende Abhängigkeiten gelten:

1.	{PersNr}	→	{PersNr, Name, Rang, Raum, Ort, Straße, PLZ, Vorwahl, BLand, EW, Landesregierung}
2.	{Ort, BLand}	→	{Vorwahl}
3.	{PLZ}	→	{BLand, Ort}
4.	{Ort, BLand, Straße}	→	{PLZ}
5.	{BLand}	→	{Landesregierung}
6.	{Raum}	→	{PersNr}

Hieraus können weitere Abhängigkeiten abgeleitet werden:

7.	{Raum}	→	{PersNr, Name, Rang, Raum, Ort, Straße, PLZ, Vorwahl, BLand, Landesregierung}
8.	{PLZ}	→	{Landesregierung}

Bei einer gegebenen Menge F von funktionalen Abhängigkeiten über der Attributmenge U interessiert uns die Menge F^+ aller aus F ableitbaren funktionalen Abhängigkeiten, auch genannt die *Hülle* (engl. *closure*) von F.

Zur Bestimmung der Hülle reichen folgende *Inferenzregeln*, genannt *Armstrong Axiome*, aus:

- Reflexivität: Aus $\beta \subseteq \alpha$ folgt: $\alpha \to \beta$
- Verstärkung: Aus $\alpha \to \beta$ folgt: $\alpha\gamma \to \beta\gamma$ für $\gamma \subseteq U$
- Transitivität: Aus $\alpha \to \beta$ und $\beta \to \gamma$ folgt: $\alpha \to \gamma$

Die Armstrong-Axiome sind *sound* (korrekt) und *complete* (vollständig). Korrekt bedeutet, daß nur solche FDs abgeleitet werden, die von jeder Ausprägung erfüllt sind, für die F erfüllt ist. Vollständig bedeutet, daß sich alle Abhängigkeiten ableiten lassen, die durch F logisch impliziert werden.

Weitere Axiome lassen sich ableiten:

- Vereinigung: Aus $\alpha \to \beta$ und $\alpha \to \gamma$ folgt: $\alpha \to \beta\gamma$
- Dekomposition: Aus $\alpha \to \beta\gamma$ folgt: $\alpha \to \beta$ und $\alpha \to \gamma$
- Pseudotransitivität: Aus $\alpha \to \beta$ und $\gamma\beta \to \delta$ folgt $\alpha\gamma \to \delta$

Wir wollen zeigen: $\{PLZ\} \rightarrow \{Landesregierung\}$ läßt sich aus den FDs 1-6 für das Relationenschema *ProfessorenAdr* herleiten:

- {PLZ} → {BLand} (Dekomposition von FD Nr.3)
- {BLand} → {Landesregierung} (FD Nr.6)
- {PLZ} → {Landesregierung} (Transitivität)

Oft ist man an der Menge von Attributen α^+ interessiert, die von α gemäß der Menge F von FDs funktional bestimmt werden:

$$\alpha^+ := \{\beta \subseteq U \mid \alpha \rightarrow \beta \in F^+\}$$

Es gilt der Satz:

$$\alpha \rightarrow \beta \text{ folgt aus Armstrongaxiomen genau dann wenn } \beta \in \alpha^+.$$

Die Menge α^+ kann aus einer Menge F von FDs und einer Menge von Attributen α wie folgt bestimmt werden:

$$X^0 := \alpha$$

$$X^{i+1} := X^i \cup \gamma \quad falls\ \beta \rightarrow \gamma \in F \wedge \beta \subseteq X^i$$

D. h. von einer Abhängigkeit $\beta \rightarrow \gamma$, deren linke Seite schon in der Lösungsmenge enthalten ist, wird die rechte Seite hinzugefügt. Der Algorithmus wird beendet, wenn keine Veränderung mehr zu erzielen ist, d. h. wenn gilt: $X^{i+1} = X^i$.

Beispiel :

Sei	U	$=$	$\{A, B, C, D, E, G\}$
Sei	F	$=$	$\{AB \rightarrow C, C \rightarrow A, BC \rightarrow D, ACD \rightarrow B,$
			$D \rightarrow EG, BE \rightarrow C, CG \rightarrow BD, CE \rightarrow AG\}$
Sei	X	$=$	$\{B, D\}$
	X^0	$=$	BD
	X^1	$=$	$BDEG$
	X^2	$=$	$BCDEG$
	X^3	$=$	$ABCDEG = X^4$, Abbruch.
Also:	$(BD)^+$	$=$	$ABCDEG$

Zwei Mengen F und G von funktionalen Abhängigkeiten heißen genau dann *äquivalent* (in Zeichen $F \equiv G$), wenn ihre Hüllen gleich sind:

$$F \equiv G \Leftrightarrow F^+ = G^+$$

Zum Testen, ob $F^+ = G^+$, muß für jede Abhängigkeit $\alpha \rightarrow \beta \in F$ überprüft werden, ob gilt: $\alpha \rightarrow \beta \in G^+$, d. h. $\beta \subseteq \alpha^+$. Analog muß für die Abhängigkeiten $\gamma \rightarrow \delta \in G$ verfahren werden.

Zu einer gegebenen Menge F von FDs interessiert oft eine kleinstmögliche äquivalente Menge von FDs.

Eine Menge von funktionalen Abhängigkeiten heißt minimal $\Leftrightarrow$

1. Jede rechte Seite hat nur ein Attribut.
2. Weglassen einer Abhängigkeit aus F verändert F^+.
3. Weglassen eines Attributs in der linken Seite verändert F^+.

Konstruktion der minimalen Abhängigkeitsmenge geschieht durch Aufsplitten der rechten Seiten und durch probeweises Entfernen von Regeln bzw. von Attributen auf der linken Seite.

Beispiel :

$$\text{Sei } U = \{A, B, C, D, E, G\}$$

$$\text{Sei } F = \{\ AB \to C,\ D \to EG;\ C \to A,\ BE \to C;\ BC \to D,\ CG \to BD;\ ACD \to B,\ CE \to AG\ \}$$

Aufspalten der rechten Seiten liefert

$$\begin{array}{lcl} AB & \to & C \\ C & \to & A \\ BC & \to & D \\ ACD & \to & B \\ D & \to & E \\ D & \to & G \\ BE & \to & C \\ CG & \to & B \\ CG & \to & D \\ CE & \to & A \\ CE & \to & G \end{array}$$

Regel	$CE \to A$	ist redundant wegen	C	$\to$	A	
	$CG \to B$	ist redundant wegen	CG	$\to$	D	
			C	$\to$	A	
			ACD	$\to$	B	
Regel	$ACD \to B$	kann gekürzt werden zu	CD	$\to$	B, wegen $C \to A$	

10.4 Schlechte Relationenschemata

Als Beispiel für einen schlechten Entwurf zeigen wir die Relation *ProfVorl*:

ProfVorl						
PersNr	Name	Rang	Raum	VorlNr	Titel	SWS
2125	Sokrates	C4	226	5041	Ethik	4
2125	Sokrates	C4	226	5049	Mäutik	2
2125	Sokrates	C4	226	4052	Logik	4
...	...	...	...	...	...	...
2132	Popper	C3	52	5259	Der Wiener Kreis	2
2137	Kant	C4	7	4630	Die 3 Kritiken	4

Folgende Anomalien treten auf:

- Update-Anomalie:
  Angaben zu den Räumen eines Professors müssen mehrfach gehalten werden.
- Insert-Anomalie:
  Ein Professor kann nur mit Vorlesung eingetragen werden (oder es entstehen NULL-Werte).
- Delete-Anomalie:
  Das Entfernen der letzten Vorlesung eines Professors entfernt auch den Professor (oder es müssen NULL-Werte gesetzt werden).

10.5 Zerlegung von Relationen

Unter *Normalisierung* verstehen wir die Zerlegung eines Relationenschemas $\mathcal{R}$ in die Relationenschemata $\mathcal{R}_1, \mathcal{R}_2, \ldots R_n$, die jeweils nur eine Teilmenge der Attribute von $\mathcal{R}$ aufweisen, d. h. $\mathcal{R}_i \subseteq \mathcal{R}$. Verlangt werden hierbei

- Verlustlosigkeit:
  Die in der ursprünglichen Ausprägung R des Schemas $\mathcal{R}$ enthaltenen Informationen müssen aus den Ausprägungen $R_1, \ldots, R_n$ der neuen Schemata $\mathcal{R}_1, \mathcal{R}_2, \ldots \mathcal{R}_n$ rekonstruierbar sein.
- Abhängigkeitserhaltung: Die für $\mathcal{R}$ geltenden funktionalen Abhängigkeiten müssen auf die Schemata $\mathcal{R}_1, \ldots, \mathcal{R}_n$ übertragbar sein.

Wir betrachten die Zerlegung in zwei Relationenschemata. Dafür muß gelten $\mathcal{R} = \mathcal{R}_1 \cup \mathcal{R}_2$. Für eine Ausprägung R von $\mathcal{R}$ definieren wir die Ausprägung R_1 von $\mathcal{R}_1$ und R_2 von $\mathcal{R}_2$ wie folgt:

$$R_1 := \Pi_{\mathcal{R}_1}(R)$$

$$R_2 := \Pi_{\mathcal{R}_2}(R)$$

Eine Zerlegung von $\mathcal{R}$ in $\mathcal{R}_1$ und $\mathcal{R}_2$ heißt *verlustlos*, falls für jede gültige Ausprägung R von $\mathcal{R}$ gilt:

$$R = R_1 \bowtie R_2$$

Es folgt eine Relation *Biertrinker*, die in zwei Tabellen zerlegt wurde. Der aus den Zerlegungen gebildete natürliche Verbund weicht vom Original ab. Die zusätzlichen Tupel (kursiv gesetzt) verursachen einen Informationsverlust.

Biertrinker		
Kneipe	Gast	Bier
Kowalski	Kemper	Pils
Kowalski	Eickler	Hefeweizen
Innsteg	Kemper	Hefeweizen

Besucht	
Kneipe	Gast
Kowalski	Kemper
Kowalski	Eickler
Innsteg	Kemper

Trinkt	
Gast	Bier
Kemper	Pils
Eickler	Hefeweizen
Kemper	Hefeweizen

Besucht ⋈ Trinkt		
Kneipe	Gast	Pils
Kowalski	Kemper	Pils
Kowalski	*Kemper*	*Hefeweizen*
Kowalski	Eickler	Hefeweizen
Innsteg	*Kemper*	*Pils*
Innsteg	Kemper	Hefeweizen

Eine Zerlegung von $\mathcal{R}$ in $\mathcal{R}_1, \ldots, \mathcal{R}_n$ heißt *abhängigkeitsbewahrend* (auch genannt *hüllentreu*) falls die Menge der ursprünglichen funktionalen Abhängigkeiten äquivalent ist zur Vereinigung der funktionalen Abhängigkeiten jeweils eingeschränkt auf eine Zerlegungsrelation, d. h.

- $F_{\mathcal{R}} \equiv (F_{\mathcal{R}_1} \cup \ldots \cup F_{\mathcal{R}_n})$ bzw.
- $F_{\mathcal{R}}^{+} = (F_{\mathcal{R}_1} \cup \ldots \cup F_{\mathcal{R}_n})^{+}$

Es folgt eine Relation *PLZverzeichnis*, die in zwei Tabellen zerlegt wurde. Fettgedruckt sind die jeweiligen Schlüssel.

PLZverzeichnis			
Ort	**BLand**	**Straße**	PLZ
Frankfurt	Hessen	Goethestraße	60313
Frankfurt	Hessen	Galgenstraße	60437
Frankfurt	Brandenburg	Goethestraße	15234

Straßen	
PLZ	**Straße**
15234	Goethestraße
60313	Gorthestraße
60437	Glagenstraße

Orte		
Ort	BLand	**PLZ**
Frankfurt	Hessen	60313
Frankfurt	Hessen	60437
Frankfurt	Brandenburg	15234

Es sollen die folgenden funktionalen Abhängigkeiten gelten:

- {PLZ} → {Ort, BLand}
- {Straße, Ort, BLand} → {PLZ}

Die Zerlegung ist verlustlos, da PLZ das einzige gemeinsame Attribut ist und {PLZ} → {Ort, BLand} gilt.

Die funktionale Abhängigkeit {Straße, Ort, BLand} → {PLZ} ist jedoch keiner der beiden Relationen zuzuordnen, so daß diese Zerlegung nicht abhängigkeitserhaltend ist.

Folgende Auswirkung ergibt sich: Der Schlüssel von *Straßen* ist {PLZ, Straße} und erlaubt das Hinzufügen des Tupels [15235, Goethestraße].

Der Schlüssel von *Orte* ist {PLZ} und erlaubt das Hinzufügen des Tupels [Frankfurt, Brandenburg, 15235]. Beide Relationen sind lokal konsistent, aber nach einem Join wird die Verletzung der Bedingung {Straße, Ort, BLand} → {PLZ} entdeckt.

10.6 Erste Normalform

Ein Relationenschema $\mathcal{R}$ ist in erster Normalform, wenn alle Attribute atomare Wertebereiche haben. Verboten sind daher zusammengesetzte oder mengenwertige Domänen.

Zum Beispiel müßte die Relation

Eltern		
Vater	Mutter	Kinder
Johann	Martha	{Else, Lucia}
Johann	Maria	{Theo, Josef}
Heinz	Martha	{Cleo}

„flachgeklopft“ werden zur Relation

Eltern		
Vater	Mutter	Kind
Johann	Martha	Else
Johann	Martha	Lucia
Johann	Maria	Theo
Johann	Maria	Josef
Heinz	Martha	Cleo

10.7 Zweite Normalform

Ein Attribut heißt *Primärattribut*, wenn es in mindestens einem Schlüsselkandidaten vorkommt, andernfalls heißt es Nichtprimärattribut.

Ein Relationenschema $\mathcal{R}$ ist in zweiter Normalform falls gilt:

- $\mathcal{R}$ ist in der ersten Normalform
- Jedes Nichtprimär-Attribut $A \in \mathcal{R}$ ist voll funktional abhängig von jedem Schlüsselkandidaten.

Seien also $\kappa_1, \ldots, \kappa_n$ die Schlüsselkandidaten in einer Menge F von FDs. Sei $A \in \mathcal{R} \Leftrightarrow (\kappa_1 \cup \ldots \cup \kappa_n)$ ein *Nichtprimärattribut*. Dann muß für $1 \leq j \leq n$ gelten:

$$\kappa_j \dot{\rightarrow} A \in F^+$$

Folgende Tabelle verletzt offenbar diese Bedingung:

StudentenBelegung			
MatrNr	**VorlNr**	Name	Semester
26120	5001	Fichte	10
27550	5001	Schopenhauer	6
27550	4052	Schopenhauer	6
28106	5041	Carnap	3
28106	5052	Carnap	3
28106	5216	Carnap	3
28106	5259	Carnap	3
...	...	...	...

Abbildung 10.1 zeigt die funktionalen Abhängigkeiten der Relation *StudentenBelegung*. Offenbar ist diese Relation nicht in der zweiten Normalform, denn *Name* ist nicht voll funktional abhängig vom Schlüsselkandidaten {*MatrNr, VorlNr*}, weil der Name alleine von der Matrikelnummer abhängt.

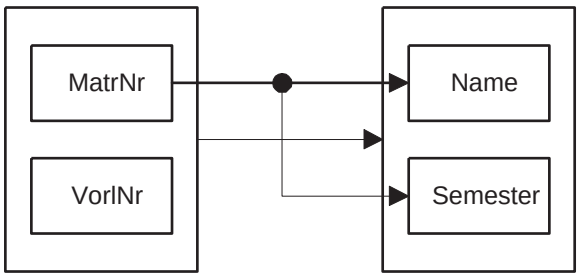


Abbildung 10.1: Graphische Darstellung der funktionalen Abhängigkeiten von StudentenBelegung

Als weiteres Beispiel betrachten wir die Relation

Hörsaal : { [Vorlesung, Dozent, Termin, Raum] }

Eine mögliche Ausprägung könnte sein:

Vorlesung	Dozent	Termin	Raum
Backen ohne Fett	Kant	Mo, 10:15	32/102
Selber Atmen	Sokrates	Mo, 14:15	31/449
Selber Atmen	Sokrates	Di, 14:15	31/449
Schneller Beten	Sokrates	Fr, 10:15	31/449

Die Schlüsselkandidaten lauten:

- {Vorlesung, Termin}
- {Dozent, Termin}
- {Raum, Termin}

Alle Attribute kommen in mindestens einem Schlüsselkandidaten vor. Also gibt es keine Nichtprimärattribute, also ist die Relation in zweiter Normalform.

10.8 Dritte Normalform

Wir betrachten die Relation

Student : {[MatrNr, Name, Fachbereich, Dekan]}

Eine mögliche Ausprägung könnte sein:

MatrNr	Name	Fachbereich	Dekan
29555	Feuerbach	6	Matthies
27550	Schopenhauer	6	Matthies
26120	Fichte	4	Kapphan
25403	Jonas	6	Matthies
28106	Carnap	7	Weingarten

Offenbar ist *Student* in der zweiten Normalform, denn die Nichtprimärattribute *Name*, *Fachbereich* und *Dekan* hängen voll funktional vom einzigen Schlüsselkandidat *MatrNr* ab.

Allerdings bestehen unschöne Abhängigkeiten zwischen den Nichtprimärattributen, z. B. hängt *Dekan* vom *Fachbereich* ab. Dies bedeutet, daß bei einem Dekanswechsel mehrere Tupel geändert werden müssen.

Seien X, Y, Z Mengen von Attributen eines Relationenschemas $\mathcal{R}$ mit Attributmenge U.
Z heißt *transitiv abhängig* von X, falls gilt

$$X \cap Z = \emptyset$$
$$\exists\, Y \subset U : X \cap Y = \emptyset, Y \cap Z = \emptyset$$
$$X \to Y \to Z, Y \not\to X$$

Zum Beispiel ist in der Relation *Student* das Attribut *Dekan* transitiv abhängig von *MatrNr*:

$$\text{MatrNr} \overset{\not\leftarrow}{\rightarrow} \mathit{Fachbereich} \rightarrow \mathit{Dekan}$$

Ein Relationenschema $\mathcal{R}$ ist in dritter Normalform falls gilt

- $\mathcal{R}$ ist in zweiter Normalform
- Jedes Nichtprimärattribut ist nicht-transitiv abhängig von jedem Schlüsselkandidaten.

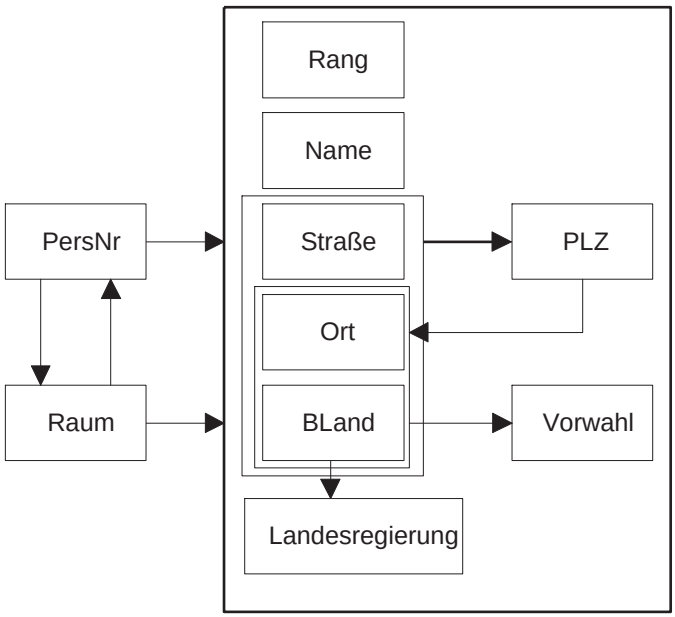


Abbildung 10.2: Graphische Darstellung der funktionalen Abhängigkeiten von ProfessorenAdr

Als Beispiel betrachten wir die bereits bekannte Relation

$$\begin{aligned}\text{ProfessorenAdr} : \{[&\text{PersNr, Name, Rang, Raum, Ort, Straße}\\ &\text{PLZ, Vorwahl, BLand, Landesregierung}]\}\end{aligned}$$

Abbildung 10.2 zeigt die funktionalen Abhängigkeiten in der graphischen Darstellung. Offenbar ist die Relation nicht in der dritten Normalform, da das Nichtprimärattribut *Vorwahl* nicht-transitiv-abhängig vom Schlüsselkandidaten *PersNr* ist:

$$\mathit{PersNr} \overset{\not\leftarrow}{\rightarrow} \{\mathit{Ort},\ \mathit{BLand}\} \rightarrow \mathit{Vorwahl}$$

Um Relationen in dritter Normalform zu erhalten, ist häufig eine starke Aufsplittung erforderlich. Dies führt natürlich zu erhöhtem Aufwand bei Queries, da ggf. mehrere Verbundoperationen erforderlich werden.