Prof. Dr. V. Linnemann
Christoph Reinke
Universität zu Lübeck
Institut für Informationssysteme

Lübeck, den 1. Februar 2010

Nonstandard Datenbanken

Wintersemester 2009/2010

13. Übungsblatt: Probe-Klausur

Hinweise:

- Bitte versuchen Sie, die Aufgaben zum Test in der in der Klausur vorgegebenen Zeit (90 Minuten) zu lösen.
- Die Lösungen dieses Aufgabenblattes können am Montag, den 8. Februar 2010 in der Vorlesung abgegeben werden, sie werden jedoch nicht bepunktet.

Aufgabe 1: Java Server Pages

a) Beschreiben Sie kurz die Motivation für die Verwendung von Java Server Pages. Welche Nachteile von Java Servlets werden damit behoben?

b) Skizzieren und beschreiben Sie die Übersetzung und die Ausführung einer Java Server Page. Welche ungefähre Struktur hat dabei die Java Server Page? Was ist das Resultat der Übersetzung?

c) Wie können in diesem Zusammenhang *Java Beans* verwendet werden?

d) Beschreiben Sie die Funktion der *'scope'*-Angabe in einer Java Server Page (z.B. scope='session'). Welche weiteren scope-Angaben kennen Sie?

Aufgabe 2: Java Servlets

In dieser Aufgabe ist ein Servlet zu entwickeln, das zu einem Produktnamen den zugehörigen Preis liefert. Das zugehörige HTML-Formular ist wie folgt aufgebaut:

```
<form action="servlet/GetPreis" method="GET">
   Produktname: <input type="TEXT" name="pname">
   <input type="SUBMIT" value="OK">
</form>
```

Die an die JDBC-Schnittstelle angebundene Datenbank enthält eine Tabelle *produkte*:

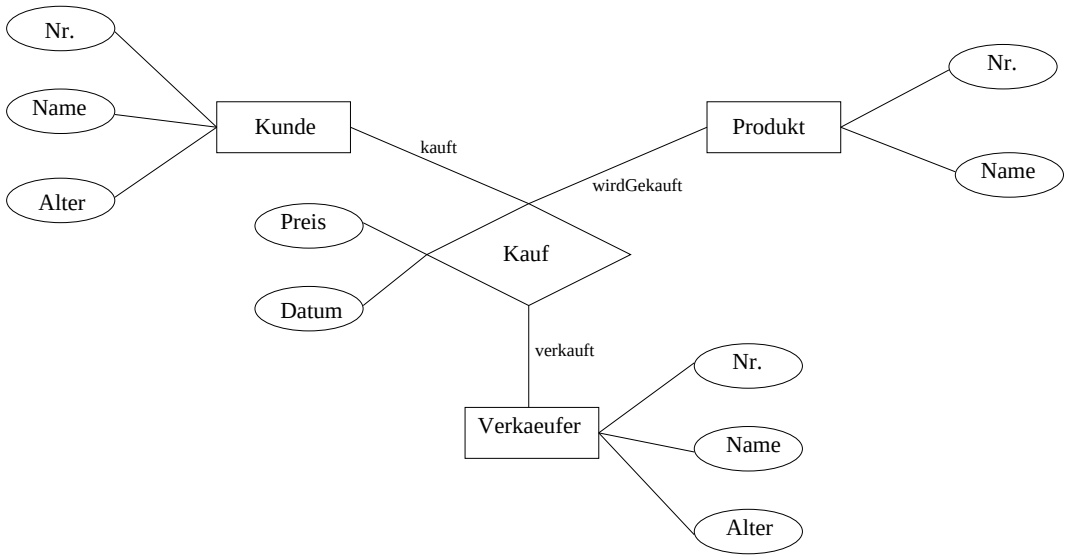
```
produkte: +----------+-----------+
          | pname    | preis     |
          +----------+-----------+
          |          |           |
```

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

public class (1)__________________ extends
                                   (2)__________________{
 public void (3)_____________________ (HttpServletRequest request,
                                       HttpServletResponse response)
  throws ServletException,IOException {
  response.setContentType('text/plain');
  PrintWriter out = response.getWriter();
  String pname = (4)_____________________________
   if (pname == null || pname.length() == 0)
    out.println('␣Fehler:␣keinen␣Produktnamen␣angegeben␣');
   else {
    Connection con = null;
    Statement stmt = null;
    ResultSet rs = null;
    try {
     Class.forName('org.hsqldb.JdbcDriver');
     con = DriverManager.getConnection(
      'jdbc:hsqldb:hsql://localhost', 'sa','');
     stmt = (5)_____________________________________
      rs = (6)_____________________________________
      boolean found = false;
     out.println("<HTML><HEAD><TITLE>");
     out.println("Produktpreis:");
     out.println("</TITLE></HEAD><BODY>");
     out.println("<UL>");
     while ((7)_____________________________) {
      out.println((8)_____________________________);
     }
     out.println("</UL>");
     out.println("</BODY></HTML>");
    } catch (Exception x) {
     x.printStackTrace(out);
    } finally {
     con.close();
    }
   }
 }
}
```

Füllen Sie bitte die 8 Lücken im Quellcode aus.

Aufgabe 3: Relationale und objektorientierte Modellierung



a) Formulieren Sie zu dem ER-Modell in der Abbildung ein relationales Modell. Geben Sie dabei Primär- und Fremdschlüssel an. Benutzen Sie dabei die Schreibweise: *NameDerRelation(Attribut1, Attribut 2, Attribut3->NameEinerAnderenRelation)*, wobei unterstrichene Attribute den Primärschlüssel kennzeichnen. In diesem Fall ist Attribut 3 ein Fremdschlüssel.

b) Geben Sie in SQL die Namen der Produkte an, die gekauft wurden.

c) Geben Sie eine Anfrage an, die die Namen der Kunden eines Verkäufers 'Meier' ausgibt.

d) Geben Sie die Namen der Produkte an, die ein Verkäufer namens 'Meier' einem Kunden namens 'Schulze' verkauft hat.

e) Modellieren Sie den dargestellten Sachverhalt objektorientiert mit Hilfe von Java-Klassen. Dabei soll es zusätzlich eine Klasse '*Person*' geben, von der *Verkäufer* und *Kunde* erben. Die gemeinsamen Attribute sollen nur in der Klasse *Person* vorkommen. Dabei sollen auftretende Referenzen zwischen den Klassen jeweils in beide Richtungen realisiert werden.

f) Formulieren Sie die 3 Anfragen in b), c) und d) jeweils in der OQL oder in der Syntax des *db4o*-DBMS.

g) Schätzen Sie ganz grob die Komplexität der Anfrage aus c) für die SQL und die OQL-Version ab.

Aufgabe 4: XML

```
<PLAY>
  <TITLE> The Tragedy of Hamlet, Prince of Denmark</TITLE>

  <PERSON NAME='Claudius' DESCRIPTION='King of Denmark'>
  <PERSON NAME='Hamlet' DESCRIPTION='Nephew to zhe present king'>
  <PERSON NAME='Polonius' DESCRIPTION='Lord Chamberlain'>
  <PERSON NAME='Ophelia' DESCRIPTION='Daughter to Polonius'>
    .
    .
  <SCENE>
    <SPEECH SPEAKER='Hamlet'>
      <LINE><LOUD>To be, or not to be</LOUD>:
                    that is the question</LINE>
      <LINE> Whether 'tis nobler in the mind to suffer </LINE>
        .
        .
      <LINE>  The fair Ophelia! Nymph, in the orisons</LINE>
      <LINE> Be all my sins remember'd.</LINE>
    </SPEECH>
    <SPEECH SPEAKER='Ophelia'>
      <LINE> <LOUD>Good my lord,</LOUD></LINE>
      <LINE> How does your honour for this many a day?</LINE>
    </SPEECH>
    <SPEECH SPEAKER='Hamlet'>
      <LINE> I humbly thank you; well, well, well.</LINE>
    </SPEECH>
      .
      .
  </SCENE>
</PLAY>
```

a) Geben Sie eine geeignete DTD für das obige XML-Dokument an. Achten Sie dabei auf die sogenannte *'Generalisierungsfähigkeit'* Ihrer DTD. D.h. die DTD sollte nach Möglichkeit auch andere zu erwartende Dokumente dieser Art beschreiben. Erläutern Sie genau, wenn es hierbei Zweifelsfälle bezüglich der DTD-Entwicklung geben kann.

b) Geben Sie einen XPATH-Ausdruck an, der alle Textstellen zurückliefert, die laut ('loud') gesprochen werden.

c) Geben Sie einen XPATH-Ausdruck an, der alle Textpassagen ('speech') zurückgibt, die von Hamlet gesprochen werden.

d) Stellen Sie sich vor, es wäre Ihre Aufgabe, ein Programm zu schreiben, das Aufgaben wie in Teil a) automatisch löst. Auf welche Probleme kann das Programm dabei stoßen (ein Problemfall genügt)?