Praktikum Datenbanken / DB2
Woche 9: Transaktionskonzept, Java-Anbindung über JDBC

Raum: LF 230 Nächste Sitzung: 8./11. Dezember 2003

Die Dokumentation zu DB2 steht online zur Verfügung. Eine lokale Installation der Dokumentation findet sich unter der Adresse `http://salz.is.informatik.uni-duisburg.de/db2doc/de_DE/index.htm`.
Die englischsprachigen Handbücher für IBM DB2 V8.1 liegen als PDF Dateien lokal im Verzeichnis `/usr/projects/db2doc/`. Diese Handbücher sind sehr umfangreich und sollten nur zum Betrachten am Bildschirm herangezogen und nicht ausgedruckt werden. Die Referenzen in diesem Arbeitsblatt beziehen sich auf diese Handbücher. Die gleichen Informationen sind jedoch auch in der Online-Dokumentation zu finden.

Die Klassendokumentation zu Java finden sich auf der Webseite von Sun (`http://java.sun.com/j2se/1.4.2/docs/api/index.html`), zudem gibt es zahlreiche gute Bücher zur Einführung in Java, einige davon frei im Web verfügbar. Entsprechende Literaturhinweise finden sich auf dem ersten Arbeitsblatt. Die Dokumentation zu JDBC und einige Tutorien finden sich unter dieser URL: `http://xl.in.tum.de/java-doc/guide/jdbc/index.html`. Sun hat ebenfalls ein sehr gutes Tutorium zu JDBC, bei dem man eigentlich alles lernt, was man im Rahmen des Praktikums benötigt: `http://java.sun.com/docs/books/tutorial/jdbc/basics/`.

Transaktionen

Transaktionen in Datenbanken sind charakterisiert durch die sogenannten **ACID**-Eigenschaften. Das bedeutet, eine Transaktion soll den Eigenschaften

A tomicity (Atomaritär),

C onstistency (Konsistenz),

I solation, und

D urability (Dauerhaftigkeit)

genügen. Das bedeutet, eine Transaktion soll entweder als Ganzes oder gar nicht wirksam werden; die korrekte Ausführung einer Transaktion überführt die Datenbank von einem konsistenten Zustand in einen anderen; jede Transaktion soll unbeeinflußt von anderen Transaktionen ablaufen; und die Änderungen einer wirksam gewordenen Transaktion dürfen nicht mehr verloren gehen.

In SQL gibt es zur Unterstützung von Transaktionen verschiedene sprachliche Konstrukte. Im Allgemeinen haben wir DB2 bisher so benutzt, dass alle Anweisungen automatisch implizit als Transaktionen behandelt wirden. Dies haben wir durch die Option `-c` (*auto commit*) beim Aufruf des CLPs erreicht. Alle Änderungen eines jedes SQL-Statements werden sofort durchgeschrieben und wirksam.

Ruft man den CLP mit `+c` auf, dann wird das *auto commit* deaktiviert. Nun gilt Folgendes:

- Jeder lesende oder schreibende Zugriff auf die Datenbank beginnt implizit eine neue Transaktion.
- Alle folgenden Lese- oder Schreibvorgänge gehören zu dieser Transaktion.
- Alle eventuellen Änderungen innerhalb dieser Transaktion gelten zunächst einmal nur vorläufig.

- Der Befehl `commit` schreibt die Änderungen in die Datenbank durch. Zu diesem Zeitpunkt erst müssen Integritätsbedingungen eingehalten werden. Zwischendurch kann also auch eine Integritätsbedingung verletzt werden.
- Commit macht die von den folgenden Befehlen getätigten Änderungen wirksam: ALTER, COMMENT ON, CREATE, DELETE, DROP, GRANT, INSERT, LOCK TABLE, REVOKE, SET INTEGRITY, SET transition variable und UPDATE.
- Solange diese Transaktion nicht durchgeschrieben wurde, nimmt der Befehl `rollback` alle Änderungen seit dem letzten **commit** als Ganzes zurück.

Im Mehrbenutzerbetrieb auf einer Datenbank kann es durch die Konkurrenz zweier Transaktionen zu Problemen oder Anomalien kommen. Typisches Beispiel sind gleichzeitige Änderungen an einer Tabelle durch zwei unabhängige Benutzer:

Phantomtupel: Innerhalb einer Transaktion erhält man auf die gleiche Anfrage bei der zweiten Ausführung zusätzliche Tupel.

Nichtwiederholbarkeit: Das Lesen eines Tupels liefert innerhalb einer Transaktion unterschiedliche Ergebnisse.

"Schmutziges" Lesen: Lesen eines noch nicht durchgeschriebenen Tupels.

Verlorene Änderungen: Bei zwei gleichzeitigen Änderungen wird eine durch die andere überschrieben und geht verloren.

Daher kann man in DB2-SQL zum einen den Isolationsgrad setzen, mit dem man arbeitet, indem man **vor** Aufnahme der Verbindung mit einer Datenbank den Befehl CHANGE ISOLATION benutzt. Es gibt vier Isolationsgrade, die unterschiedlich gegen die genannten Anomalien schützen:

	RR	RS	CS	UR
Phantomtupel	nein	ja	ja	ja
Nichtwiederholb.	nein	nein	ja	ja
Schmutziges Lesen	nein	nein	nein	ja
Verlorenes Update	nein	nein	nein	nein

Außerdem lassen sich explizit Tabellen oder eine gesamte Datenbank sperren, indem man den Befehl LOCK TABLE benutzt, bzw. die Verbindung mit der Datenbank unter Angabe des Sperrmodus herstellt. SHARE-Sperren sind der Standard und bedeuten, dass niemand anderes eine EXCLUSIVE-Sperre anfordern kann. Beispiele:

```
LOCK TABLE autor IN EXCLUSIVE MODE;

CONNECT TO imdb23 IN SHARE MODE USER username;
```

Mini-Einführung in Java und JDBC

Java ist eine objektorientierte und plattformunabhängige Programmiersprache, die im Rahmen dieses Praktikums zur Entwicklung einer Datenbankanwendung benutzt werden soll. Java wurde von der Firma Sun entwickelt und hat eine C/C++ ähnliche Syntax. Einer der größten Vorteile Javas als Sprache zur Anwendungsentwicklung ist die sehr umfangreiche Klassenbibliothek. Zur Lösung der meisten häufigen Probleme existieren bereits fertige Klassen, die sich leicht in eigene Anwendungen einbinden lassen.

Man unterscheidet grundsätzlich zwischen Java-Applikationen und Java-Applets. Applets werden in Webseiten eingebunden und von einem Webbrowser ausgeführt. Sie werden von der Klasse Applet abgeleitet und unterliegen einigen Restriktionen. Wir wollen uns aber auf Java-Applikationen beschränken. Dies sind vollwertige Programme, die lokal laufen, oder als Client-Server-Applikation über ein Netzwerk, oder als Server-Programm (Servlets) auf einem Webserver.

Im Rahmen des Praktikums arbeiten wir standardmäßig mit der Java-Version 1.3.1. Herausfinden, welche Java-Version benutzt wird, kann man mit dem Befehl `java -version` oder mit `which java` aus einer Konsole heraus. Aktuellere Versionen des JDK sind installiert und können benutzt werden, wenn man zuvor in einer Konsole den Befehl `source /usr/sw/java/new/.bashenv` (J2SDK 1.4.1) bzw. `source /usr/sw/java/verynew/.bashenv` (J2SDK 1.4.2) ausführt. Möchte man diese Versionen standardmäßig in seine Umgebung aufnehmen, so kann man den Befehl in der Datei `~/.bashrc` eintragen.

Zur Entwicklung von Java-Programmen stehen die Entwicklungsumgebungen WebSphere und Eclipse (Start aus der Konsole mit dem Kommando `eclipse`) zur Verfügung. Theoretisch kann man aber auch jeden beliebigen Texteditor benutzen, um eine Java-Anwendung zu schreiben.

Bevor ein fertiges Java-Programm ausgeführt werden kann, muß es kompiliert werden. Wenn keine Entwicklungsumgebung benutzt wird, so kann das in der Konsole durch den Aufruf von `javac Programm.java` geschehen. Dabei ist darauf zu achten, dass der sogenannte Klassenpfad korrekt gesetzt ist, über den der Java-Compiler einzubindende Klassen findet. Den aktuellen Klassenpfad ausgeben lassen kann man sich mit `echo $CLASSPATH`, zusätzliche Klassen lassen sich beim Aufruf des Compilers über die Option `-classpath` übergeben. Gestartet wird ein fertiges Programm dann mit `java -cp KLASSENPFAD Programm`.

Webanwendung

Im Rahmen des Praktikum soll eine auf Java-Servlets und JDBC basierende Webanwendung geschrieben werden. Euch steht frei, diese zu implementieren, wie Ihr wollt, aber zur Vorführung soll sie auf unserem lokalen Server laufen. Aus Lizenzgründen ist der Zugriff auf die Webanwendungen passwortgeschützt. Benutzername und Passwort werden in der Praktikumssitzung genannt.

Auf dem Datenbankserver `salz` existiert ein Verzeichnis `/usr/projects/dbprak/jakarta-tomcat-4.1.18/` mit einer laufenden Tomcat-Version. Unterhalb des Installationsverzeichnisses gibt es ein Unterverzeichnis `webapps` mit Arbeitsverzeichnissen für jede Gruppe. In dieses Arbeitsverzeichnis gehören die Webseiten Eurer Webanwendung, sowie die folgende Unterverzeichnisstruktur:

```
./WEB-INF/
./WEB-INF/classes/
./WEB-INF/lib/
./WEB-INF/web.xml
```

In das Verzeichnis `WEB-INF/classes/` kommen Eure kompilierten Java-Klassen, in das Verzeichnis `WEB-INF/lib/` alle eventuell benötigten Bibliotheken. Die Datei `web.xml` beschreibt Eure Webanwendung.

Listing 1: web.xml

```
1 <?xml version="1.0" encoding="ISO-8859-1"?>
2
```

```
3  <!DOCTYPE web-app
4      PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
5      "http://java.sun.com/dtd/web-app_2_3.dtd">
6
7  <web-app>
8    <servlet>
9      <servlet-name>IMDB</servlet-name>
10     <servlet-class>IMDBServlet</servlet-class>
11   </servlet>
12
13   <servlet-mapping>
14     <servlet-name>IMDB</servlet-name>
15     <url-pattern>/IMDB</url-pattern>
16   </servlet-mapping>
17 </web-app>
```

Ihr könnt das obige Beispiel als Grundlage für eine eigene Beschreibung nutzen. `servlet-class` sollte der Klassenname Eures Servlets sein, `url-pattern` definiert, wie das Servlet aus einem Browser aufgerufen werden kann.

Wenn Eure Anwendung erfolgreich in diesem Verzeichnis als Servlet installiert wurde und eine Startseite mit Namen `index.html` existiert, kann man unter der Adresse `http://salz.is.informatik.uni-duisburg.de:1977/dbpw03XX/index.html` darauf zugreifen. Über das Managementinterface von Tomcat (`http://salz.is.informatik.uni-duisburg.de:1977/manager/html`) kann eine Anwendung neu geladen werden, um Änderungen zu aktualisieren. Das Passwort zum Zugriff auf dieses Interface gibt es ebenfalls in der Sitzung.

Wer daheim sein Programm testen möchte, ohne es erst auf `salz` zu installieren, kann Tomcat auch selber herunterladen und installieren: `http://jakarta.apache.org/tomcat/`.

JDBC

Zum Zugriff auf die DB2 Datenbank benutzen wir JDBC (*Java DataBase Connectivity*). Dies ist eine einheitliche Datenbankschnittstelle für Java, die es für den Anwendungsentwickler transparent macht, ob hinter dieser Schnittstelle auf eine DB2, Oracle, mysql, oder sonst eine Datenbank zugegriffen wird. Die Verbindung wird durch einen Treiber hergestellt, der erst zur Laufzeit in das Programm eingebunden wird. Wir benutzen die von IBM zur Verfügung gestellten JDBC-Treiber, die Ihr in Eurem Homeverzeichnis unter `~/sqllib/java/` findet: `db2cc.jar` und `db2cc_license_cu.jar`.

Unter JDBC greift man über eine URL auf die Datenbank zu, deren Aufbau folgendermassen aussieht: `jdbc:subprotocol://host:port/database`. In unserem Fall ist 'subprotocol' `db2`, 'host' `salz.is.informatik.uni-duisburg.de`, 'port' wie bisher `500xx` und 'database' ist der Name Eurer Datenbank. Man schaue sich hierzu auch das untenstehende Beispiel an.

Die eigentliche Verbindung wird über die Klasse `DriverManager` aufgebaut, spezifiziert durch das Interface `Connection` (Listing 2, Zeile 94). Ein Programm kann dabei auch Verbindungen zu mehreren Datenbanken offenhalten, wobei jede Verbindung durch ein Objekt realisiert wird, welches das Interface `Connection` implementiert. Dieses bietet unter anderem folgende Methoden:

- `createStatement()`: erzeugt Objekte, die das Interface `Statement` implementieren

- `commit()`: schreibt Transaktionen durch
- `rollback()`: nimmt Transaktionen zurück
- `setAutoCommit(boolean)`: setzt AutoCommit auf wahr oder falsch
- `close()`: schließt eine offene Verbindung

SQL-Anweisungen werden über das Interface `Statement` benutzt (Listing 2, Zeile 99). Das Interface definiert eine Reihe von Methoden, die es erlauben, Anweisungen auf der Datenbank auszuführen. Man kann Objekte, die dieses Interface implementieren, z.B. durch die Methode `createStatement()` auf Objekten vom Typ `Connection` erzeugen lassen. Das Interface bietet u.a. folgende Methoden:

- `executeQuery(String)`: führt ein SQL-Statement aus und liefert ein Ergebnis zurück
- `executeUpdate(String)`: führt INSERT, UPDATE, DELETE aus, dabei ist die Rückgabe die Anzahl der betroffenen Tupel oder nichts

Im Interface `ResultSet` (Listing 2, Zeile 100) werden die Funktionalitäten zur Weiterverarbeitung von Anfrageergebnissen definiert. Auf Objekten, die dieses Interface implementieren, stehen u.a. folgende Methoden zur Verfügung:

- `next()`: geht zum ersten bzw. nächsten Tupel des Ergebnisses
- `getMetaData()`: liefert ein Objekt zurück, welches das Interface `ResultSetMetaData` implementiert, hierüber können Informationen über den Aufbau des Ergebnisses ausgelesen werden
- `findColumn(String)`: findet zu einem Attributnamen die Position im Ergebnistupel
- `get<Typ>(int)` oder `get<Typ>(String)`: lesen die (benannte) Spalte des Ergebnistupels aus und liefern ein Objekt vom angegeben Typ zurück

Die verschiedenen in SQL bekannten Datentypen werden als Java-Klassen nachgebildet:

SQL-Typ	**Java-Typ**
CHAR, VARCHARm LONG VARCHAR	String
DECIMAL, NUMERIC	java.math.BigDecimal
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
DOUBLE, FLOAT	double
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.TimeStamp

Beispiel

Das folgende kleine Java-Servlet erwartet einen Parameter **request**. Dieser soll eine syntaktisch korrekte und zum Schema unserer Datenbank passende Anfrage sein, z.B. `SELECT * FROM film WHERE jahr=2003 FETCH FIRST 25 ROWS ONLY`. Aufgerufen wird das Servlet über ein HTML-Formular.

Listing 2: MovieDBServlet.java

```
1
2  /*
3   * Created on Nov 19, 2003
4   *
5   */
6  import java.util.*;
7  import java.io.*;
8  import java.sql.*;
9  import java.net.*;
10 import javax.servlet.*;
11 import javax.servlet.http.*;
12
13 /**
14  * @author kriewel
15  *
16  * This is a minimalistic example of a Java servlet that connects
17  * to a DB2 database, sends an SQL statement and prints out
18  * the result
19  *
20  */
21 public class IMDBServlet extends HttpServlet {
22
23   // defaults for connecting to the database
24   String group = "23";
25   String database = "IMDB23";
26   String userid = "dbpw0323";
27   String url = "";
28   String passwd = "";
29   String request = "";
30
31   // we select the database driver at runtime
32   static {
33     try {
34       Class.forName("com.ibm.db2.jcc.DB2Driver");
35     } catch (Exception e) {
36       e.printStackTrace();
37     }
38   }
39
40   // this method
41   void getParameters (HttpServletRequest req) {
42
43     request = req.getParameter("request");
44
45     // replace line breaks with spaces
46     request = request.replace('\n',' ');
47     request = request.replace('\r',' ');
48
49     url = "jdbc:db2://salz.is.informatik.uni-duisburg.de:500"
50           +group+"/"+database;
51
52     // in an actual script one would have to
53     //  create a password dialog
54     passwd = "********";
55
56   }
57
58   // this method handles GET requests
59   public void doGet(HttpServletRequest req,
60                     HttpServletResponse res)
61                 throws IOException, ServletException {
```

```
62
63      res.setContentType("text/html");
64      getParameters(req);
65      doRequest(new PrintWriter(new
66                      BufferedWriter(res.getWriter())));
67    }
68
69    // this method handles POST requests
70    public void doPost(HttpServletRequest req,
71                       HttpServletResponse res)
72                      throws IOException, ServletException {
73
74      res.setContentType("text/html");
75      getParameters(req);
76      doRequest(new PrintWriter(new
77                      BufferedWriter(res.getWriter())));
78    }
79
80    // stubs
81    public void doSelectPersons (PrintWriter out) {
82    }
83
84    public void doSelectProductions (PrintWriter out) {
85    }
86
87    // simple handler for requests
88    public void doRequest(PrintWriter out) {
89
90      // container for results
91      Vector lines = new Vector();
92      try {
93        // open a connection
94        Connection con =
95           DriverManager.getConnection(url,userid,passwd);
96
97        // create statement, send statement to
98        // Database and get results
99        Statement stmt = con.createStatement();
100       ResultSet rs = stmt.executeQuery(request);
101       ResultSetMetaData rsmd = rs.getMetaData();
102
103       // how many result columns do we have
104       int cols = rsmd.getColumnCount();
105
106       // iterate through result and build web output
107       while (rs.next()) {
108         String s = "<li>";
109         // we don't care for the datatype of the output
110         // and convert everything to strings
111         for (int i=1; i < cols; i++)
112         s += rs.getString(i) + ",␣";
113         s += rs.getString(cols) + "</li>";
114         lines.addElement(s);
115       }
116       rs.close();
117       stmt.close();
118       con.close();
119
120     } catch (Exception e)  {
121       e.printStackTrace();
122     }
123
```

```
124     out.println("<html><body><ul>");
125
126     // dump result into webpage,
127     // one line per tupel
128     Iterator i = lines.iterator();
129     while (i.hasNext() ){
130       out.println(i.next());
131     }
132     out.println("</ul></body></html>");
133
134     // flush out the print buffer !
135     out.flush();
136   }
137 }
```

Listing 3: search.html

```
1  <html>
2    <head>
3      <title>Very simple Web-Interface to the movie database</title>
4    </head>
5    <body bgcolor="#ffffff">
6      <h1>Very simple Web-Interface to the movie database</h1>
7      <form action="IMDB" method=GET>
8        <input type="submit" value="Submit">
9        <table>
10         <tbody>
11           <tr>
12             <td>SQL Statement:</td>
13             <td><textarea type="text" name="request" cols="100"
14             rows="10" maxlength="500"></textarea></td>
15           </tr>
16         </tbody>
17       </table>
18     </form>
19
20     <h3>Beispiele:</h3>
21
22     <p>SELECT DISTINCT art FROM film</p>
23
24     <p>SELECT * FROM serie WHERE titel LIKE '2%'</p>
25
26     <p>SELECT name
27     FROM spielt_in
28     WHERE produktion = 'Buffy the Vampire Slayer'<br>
29     INTERSECT<br>
30     SELECT name FROM spielt_in
31     WHERE produktion = 'Angel'</p>
32
33     <p>SELECT produktion2
34     FROM beziehung
35     WHERE produktion1='Friday the 13th'
36       AND art='followed by'</p>
37
38     <p>SELECT count(*) FROM spielt_in
39     WHERE produktion='Carolina' AND jahr=2003</p>
40
41     <p><b>A very simple recursion (counter):</b><br>
42     WITH r(c) AS (<br>
43     SELECT * FROM (VALUES (1)) AS t<br>
44     UNION ALL<br>
45     SELECT c+1 FROM r WHERE c <4 <br>
```

```
46      )<br>SELECT * FROM r</p>
47    </body>
48  </html>
```

Aufgaben

Schreibt ein Java-Servlet zum Zugriff auf Eure Datenbank, das auf unter dem Verzeichnis Eurer Gruppe auf unserem Tomcat-Server läuft. Hierzu vervollständigt zunächst einmal fehlende Dinge aus dem letzten Wochen. Insbesondere sollten alle Tabelle existieren und mit sinnvollen Daten gefüllt werden. Erstellt auch Indexe zum schnellen Zugriff auf die Attribute, welche Ihr für Eure Anfragen benötigt. Einige der Anforderungen sind leichter zu erfüllen, wenn die im Laufe des Praktikums entwickelten Views oder UDFs existieren. Testet Eure Anwendung gründlich.

Das Servlet sollte folgende Anforderungen erfüllen:

(a) Es muß dem Benutzer möglich sein, über Auswahlboxen eine Anfrage nach Produktionen bzw. Personen zu stellen. Das Servlet soll diese Parameter dann zu einer sinnvollen Anfrage (SELECT) zusammenbauen und an die Datenbank schicken.

(b) Folgende Anfragen sollten möglich sein:
- Suche nach Personen mit einem gegebenen Vornamen und/oder Nachnamen
- Suche nach Regisseuren, Schauspieler(innen), Komponisten
- Suche nach Personen mit einem gegebenem Geburts- oder Todesjahr
- Suche nach Personen, die in einem näherungsweise gegebenen Film mitgewirkt haben
- Kombinationen davon
- Suche nach Produktionen eines näherungsweise gegebenen Titels
- Suche nach Produktionen eines gegebenen Genres
- Suche nach Produktionen eines gegebenen Jahres
- Suche nach Miniserien, normalen Serien, Kinofilmen, Fernsehspielen
- Suche nach deutschen oder amerikanischen Produktionen
- Suche nach Produktionen, in der eine näherungsweise gegebene Person mitgewirkt hat
- Kombinationen davon

(c) Das Ergebnis der Anfrage sollte auf eine sinnvolle Anzahl von Resultaten gekürzt werden (z.B. 25) und als Webseite ausgegeben werden.

(d) Ist das Ergebnis eine Liste von Personen, so sollte jede Person ein Link sein, der eine neue Anfrage nach den Produktionen stellt, in denen die Person mitgewirkt hat.

(e) Ist das Ergebnis eine Liste von Produktionen, so sollte jede Produktion ein Link sein, der eine neue Anfrage nach den Personen stellt, die in dieser Produktion mitgewirkt haben.

(f) Es soll eine Möglichkeit geben, neue Filme in die Datenbank einzutragen.

(g) Führt mit Hilfe eines Tiggers ein Protokoll über Einfügungen, und macht dieses über Eure Webanwendung abfragbar. Das Protokoll sollte mindestens den Titel des eingefügten Films und den Zeitpunkt der Einfügung festhalten.

Für die Bearbeitung dieser Aufgabe ist Zeit bis zum 13. Januar 2004 (bzw. bis zum 17. Januar 2004 für die Teilnehmer der Donnerstagsgruppe). Dann müssen die Ergebnisse den Betreuern vorgeführt werden. Zwischen dem 19. Dezember und dem 12. Januar sind keine weiteren Praktikumstermine.