Praktikum Datenbanken / DB2
Woche 8: Transaktionen, JDBC

Betreuer: Sascha Kriewel, Tobias Tuttas
Raum: LF 230
Bearbeitung: 27., 29. Juni, 2. Juli 2007

Datum	
Team (Account)	
Vorbereitung	
Präsenz	

Aktuelle Informationen, Ansprechpartner und Material unter:
`http://www.is.inf.uni-due.de/courses/dbp_ss07/index.html`

Wochenziele

In dieser Woche testen wir das Transaktionskonzept und bereiten die Kommunikation mit der graphischen Benutzeroberfläche (GUI) über JDBC vor.

Transaktionen

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

A tomicity (Atomarität),

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 wird durch die Option `-c` (*auto commit*) beim Aufruf des *command line processors* (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 anwender IN EXCLUSIVE MODE;

CONNECT TO filme 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 wird. 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 1.4.2. Herausfinden, welche Java-Version benutzt wird, kann man mit dem Befehl `java -version` aus einer Konsole heraus.

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.

Beispiel:

```
java -classpath db2jcc.jar:db2jcc_license_cu.jar:. Sample
```

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: `db2jcc.jar` und `db2jcc_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.

Verbindungsaufbau

Wichtig: Damit von nicht–lokalen Rechnern eine Verbindung über Benutzername und Passwort hergestellt werden kann, muss die Authentifizierungsmethode der Datenbankinstanz von *Client* auf *Server* gestellt werden. Man ändere dazu die Konfiguration des Datenbankmanagers entsprechend. Der folgende Befehl bewirkt diese Änderung:

```
db2 update dbm cfg using AUTHENTICATION SERVER
```

Anschließend muss man den Datenbankmanager mit den Befehlen `db2stop` und `db2start` auf `salz` (!) neu starten.

Bei der Methode *Client* ist der lokale Rechner für die Authentifizierung zuständig. Werden die Zugangsdaten jedoch in einem Programm hinterlegt, welches zwar auf dem lokalen Rechner (Client) läuft, sich aber direkt mit der Datenbank verbinden will, dann kann der Client diese Zugangsdaten nicht mehr überprüfen, d.h. diese Überprüfung muss auf Server-Seite geschehen. Daher muss man die Authentifizierungsmethode auf *Server* ändern, sobald man mit einem externen Programm auf die Datenbank zugreifen will.

Falls es dabei zu Problemen kommt, weil noch Datenbanken geöffnet sind und von Applikationen benutzt werden, bekommt man über die folgenden Befehle eine Liste der offenen Datenbanken bzw. der noch offenen Verbindungen:

- `db2 list active databases`

- `db2 list applications`

Mit Hilfe des so erhaltenen *Application Handle* kann man einzelne bestehende Verbindungen zur Datenbank trennen:

- `db2 force application` *handle, ...*
- `db2 force application all`

Mit `db2stop force` kann man auch den Datenbankmanager ohne Rücksichtnahme auf offene Anwendungen herunterfahren. Das sollte man allerdings nur tun, wenn sicher ist, dass dadurch keine wichtigen Verbindungen unterbrochen werden.

Die eigentliche Verbindung wird über die Klasse `DriverManager` aufgebaut, spezifiziert durch das Interface `Connection` (Listing 1, Zeile 20). 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
- `getMetadata()`: ein Objekt zurück, welches das Interface `DatabaseMetaData` implementiert, hierüber können Informationen über den Aufbau der Datenbank ausgelesen werden (z.B. die Namen existierender Tabellen über die Methode `getTables()`)
- `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 1, Zeile 25). 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 1, Zeile 29) 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 Programm baut eine Verbindung zu einer entfernten Datenbank auf. Dann setzt es eine SQL–Anfrage ab und gibt zwei Spalten der Ergebnisrelation aus.

Listing 1: Sample.java

```
1  import java.sql.Connection;
2  import java.sql.DriverManager;
3  import java.sql.ResultSet;
4  import java.sql.SQLException;
5  import java.sql.Statement;
6
7  public class Sample {
8
9      public static void main(String[] args) {
10         try {
11             // DB2-Treiber laden
12             Class.forName("com.ibm.db2.jcc.DB2Driver");
13
14             String dbserver = "salz.is.inf.uni-due.de";
15             String dbport   = "50038";
16             String database = "video38";
17
18             // Verbindung zur DB aufbauen
19             // Passwort wird in der Praktikumssitzung ausgegeben
20             Connection db2Conn = DriverManager.getConnection
21                 ("jdbc:db2://"+dbserver+":"+dbport+"/"+database,
22                  "dbp0738","mediavid");
23
24             // Statement, um Daten aus einer Tabelle zu holen
25             Statement st = db2Conn.createStatement();
26             String myQuery = "SELECT␣*␣FROM␣Anwender";
27
28             // Ausfuehren der Anfrage
```

```
29              ResultSet resultSet = st.executeQuery(myQuery);
30
31              // Iterieren durch ResultSet und Ausgabe
32              while (resultSet.next()) {
33                  String name = resultSet.getString("name");
34                  String vorname = resultSet.getString("vorname");
35                  System.out.println("Name: " + name);
36                  System.out.println("Vorname: " + vorname);
37                  System.out.println("--------------------------------");
38              }
39
40              // Aufraeumen
41              resultSet.close();
42              st.close();
43              db2Conn.close();
44          }
45          catch (ClassNotFoundException e) {
46              e.printStackTrace();
47          }
48          catch (SQLException e) {
49              e.printStackTrace();
50          }
51      }
52  }
```

Das Beispielprogramm `Sample.java` findet Ihr auch im Verzeichnis `/home/dbprak/exercises`.

Übersicht über die neuen (und einige alte) Befehle

(db2) change isolation to *level*	Setzen des Isolationsgrades vor Aufbau einer Datenbankverbindung
(db2) lock table *name* in *modetype* mode	Sperren einer Tabelle in einem bestimmten Modus (`exclusive` oder `share`)
(db2) connect to *name* in *modetype* mode user *username*	Verbinden mit einer Datenbank unter Angabe des Sperrmodus
(db2) update dbm cfg using *attr value*	Setzen eines Konfigurationsattributes auf einen neuen Wert
db2start	Starten der Datenbankinstanz
db2stop	Herunterfahren der Datenbankinstanz
db2stop force	Erzwungenes Herunterfahren der Datenbankinstanz
(db2) list active databases	Liste der in Benutzung befindlichen Datenbanken einer Instanz
(db2) list applications	Liste der derzeit verbundenen Anwendungen
(db2) force application *handle*	Verbindung zu einer Anwendung erzwungen beenden
(db2) force application all	Alle Verbindungen zu Anwendungen erzwungen beenden

Aufgaben

Vorbereitung (Hausaufgaben)

V1: Interaktion zwischen Anwender und System

Schreibt Euch für die folgenden Anwendungsfälle (*Use Cases*) möglichst detailliert die Einzelschritte auf, die der Anwender oder das System dafür erledigen müssen. Anfragen an die Datenbank gelten als einzelne Aktionen. Nennt dabei auch die Tabellen, die bei der Anfrage angesprochen werden.

(a) Einen neuen Benutzer anmelden

(b) Ein neues Medium (DVD oder Videokassette) mit Inhalt (Filme, Serienfolgen) eintragen

(c) Alle Medien eines Benutzers auflisten

(d) Ein Medium ausleihen

(e) Ein Medium zurückgeben

Beispiel:

Der Anwendungsfall „Anmelden“ kann in folgenden Einzelschritten umgesetzt werden. Aktionen des Anwenders sind mit **A** markiert, Aktionen des Systems mit **S**.

A Startet die graphische Benutzeroberfläche

S Zeigt ein Anmeldeformular an

A Gibt Nick und Passwort ein

S Prüft die Anmeldedaten

- Holt zu dem angegebenen Nick das Passwort und die BenutzerID des Anwenders aus der Tabelle `Anwender`, eventuell noch weitere Daten.
- Vergleicht das ermittelte Passwort mit dem eingegebenen.
- Falls das Passwort korrekt ist,
  - gilt der Anwender als angemeldet,
  - das System merkt sich die BenutzerID (und eventuell weitere Daten)
  - und gibt dem Anmelder eine Erfolgsmeldung zurück.
- Falls das Passwort nicht korrekt ist,
  - verwirft das System die aus der Datenbank geholten Anwenderdaten,
  - gibt dem Anwender eine Fehlermeldung zurück
  - und zeigt wieder das Anmeldeformular an.

V2: Transaktionen

Welche der folgenden Aktionen in der „dezentralen Videothek“ sollten als Transaktionen ausgeführt werden? Begründet Eure Antwort jeweils.

(a) Einen neuen Benutzer anmelden

(b) Ein neues Medium (DVD oder Videokassette) mit Inhalt (Filme, Serienfolgen) eintragen

(c) Alle Medien eines Benutzers auflisten

(d) Ein Medium ausleihen

(e) Ein Medium zurückgeben

Präsenz

P1: Rollback von Transaktionen

Ruft den CLP mit `db2 +c -t` auf. Stellt eine Verbindung zu Eurer Datenbank her. Alle Veränderungen auf der Datenbank werden nun erst nach einem ausdrücklichen `commit` durchgeschrieben. Probiert das aus.

Wenn Ihr mutig seid, löscht eine Tabelle mit `drop table name`. Gebt die Liste der Tabellen aus. Auf der aktuellen Datenbank ist die Änderung bereits zu sehen. Nun nehmt alle Aktionen seit dem letzten `commit` zurück. Die Tabelle erscheint wieder in der Auflistung. Wer weniger mutig ist, soll dies mit einer anderen Veränderung testen.

Achtung: Automatisches Commit ist bei DB2 die Voreinstellung. Sobald Ihr `db2` einmal ohne +c aufruft, werden alle bisher gemachten Veränderungen durchgeschrieben und können nicht mehr durch ein Rollback zurückgenommen werden.

P2: Konfiguration Eurer Datenbank-Manager-Instanz

Für diese Aufgabe müsst Ihr Euch über `ssh` auf `salz` einloggen. Informiert Euch mithilfe des Befehls `db2 get dbm cfg` über die aktuelle Konfiguration Eures Datenbankmanagers (Instanz). Richtet diese derart ein, dass ein Verbindungsaufbau von einem entfernten Rechner über JDBC möglich ist (`AUTHENTCATION SERVER`).

P3: JDBC und Transaktionen

(a) Kopiert Euch das Beispiel–Programm und passt es für Eure Datenbankinstanz an. Kompiliert es und testet es.

(b) Erweitert das Programm, so dass es nun auch eine Liste aller Tabellen in Eurer Datenbank ausgibt.

(c) Erweitert das Programm so, dass es

- eine Transaktion startet,

- eine neue Tabelle `Test` anlegt,
- die Liste von Tabellen ausgibt,
- die Transaktion *abbricht* und
- wieder die Liste der Tabellen ausgibt.

Hinweis:

Verwendet beim Verbindungsaufbau die Methode `setAutoCommit` der Klasse `Connection`.

Zum Schluss ...
stoppt bitte Eure Datenbank-Manager-Instanz auf `salz`,
loggt Euch von `salz` und vom lokalen Rechner aus,
aber schaltet den Rechner nicht ab.