Universität Leipzig, Institut für Informatik
Abteilung Datenbanken
Prof. Dr. E. Rahm, A. Thor, D. Aumüller

Ausgabe: 29. 04. 2008

Datenbanksysteme II

SS 2008 – 1. Praktische Übung - JDBC und SQLJ

In dieser praktischen Übung erstellen Sie einfache Java-Programme, welche mittels JDBC bzw. SQLJ auf eine Datenbank zugreifen. Das grundlegende Vorgehen ist u.a. im Skript zur Vorlesung (Kapitel 2) illustriert.

Nach dem Durcharbeiten dieser praktischen Übungen sollten Sie in der Lage sein, Java-Programme zu schreiben, die

- eine Datenbankverbindung auf- und wieder abbauen und
- SQL-Anweisungen zum Erstellen und Verarbeiten von Tabellen und deren Datensätze an die Datenbank senden

können. Sofern Sie keine Zwischenklausur schreiben, werden diese Fähigkeiten zusammen mit dem Inhalt der zweiten praktischen Übung im Rahmen einer praktischen Klausur getestet, deren erfolgreiches Bestehen (neben der erfolgreichen Bearbeitung der LOTS-Übungsblätter) Voraussetzung für die Teilnahme an der Abschlussklausur DBS2 ist.

Allgemeine Hinweise zu Java: Falls Sie noch keine Erfahrung mit Java besitzen, bietet das "Java Buch" (www.javabuch.de) einen sehr guten Einstieg. Das komplette Buch kann als HTML-Version kostenfrei heruntergeladen werden. Als "Schnelleinstieg" beschreibt Kapitel 2 alle notwendigen Schritte zum ersten Java-Programm (u.a. Installation benötigter Software, Übersetzung des Java-Quellcodes, Ausführung des Java-Bytecodes). In den Rechner-Pools der Informatik (Johannisgasse 26, 3. Etage) ist bereits sämtliche benötigte Software zur Java-Programmierung (inklusive mächtiger Entwicklungsumgebungen wie Eclipse) installiert.

Allgemeine Hinweise zu Datenbankverbindungen mit Java: Die für die Verbindung mit der Datenbank benötigten JDBC-Treiber sowie (für Aufgabe 3) SQLJ-Precompiler laden Sie sich (als ZIP-File) von der DBS2-Website (http://dbs.uni-leipzig.de/stud/ss2008/dbs2) herunter. Entpacken und speichern Sie alle Dateien (zwei JAR-Dateien für den JDBC-Treiber sowie eine ZIP-Datei für SQLJ) und geben Sie diese bei sämtlichen Java-Aufrufen als Parameter mit an (Option `-cp`). Bei Verwendung einer Java-Entwicklungsumgebung (z.B. Eclipse) genügt i.A. das Hinzufügen der Dateien in den "Library Path".

1. Aufgabe (Auf- und Abbau einer JDBC-Verbindung)

Schreiben Sie unter Verwendung von JDBC ein Java-Programm, dass eine Datenbankverbindung auf- und wieder abbaut. Benutzen Sie dazu folgende Verbindungsparameter:

- Name des Treibers: `com.ibm.db2.jcc.DB2Driver`
- Hostname des Datenbankservers: `leutzsch.informatik.uni-leipzig.de`
- Portnummer: `50001`
- Name der Datenbank: `db2ueb08`

Hinweis: Nutzernamen und Passwort erhalten Sie, indem Sie eine E-Mail an juseks@informatik.uni-leipzig.de mit dem Betreff "DBS2 Praktische Übung <Matrikel>" schreiben, wobei Sie <Matrikel> durch Ihre Matrikelnummer ersetzen.

2. Aufgabe (Schema- und Datenmanipulation)

In dieser Aufgabe erweitern Sie Ihr Programm aus Aufgabe 1 um typische Funktionen einer datenbankgestützten Anwendung (Anlegen und Löschen einer Tabelle sowie Einspielen, Verändern und Auslesen von Datensätzen). Verändern Sie Ihr Programm derart, dass sämtliche nachfolgend aufgeführten Aktionen automatisch durch Ihr Programm ausführt werden. Es soll keinerlei Benutzerinteraktion geben.

a) Erstellen Sie unter Verwendung von `CREATE TABLE` eine Tabelle `Mitarbeiter` mit folgenden Spalten:

- `Name` mit Typ `VARCHAR(50)` als Primärschlüssel
- `Gehalt` mit Typ `INT`, das nicht `NULL` sein darf

Hinweise:

- Erstellen Sie die Tabelle in Ihrem Schema, das den gleichen Namen wie Ihr Datenbank-Nutzername hat. Der Tabellenname ist dabei `<Schema>.<Tabelle>`, also z.B. `db2ueb01.Mitarbeiter`.

- Um eine wiederholte Ausführung Ihrer Programms zu gewährleisten, löschen Sie die Tabelle vor dem Anlegen mittels `DROP TABLE <Schema>.<Tabelle>`. Eine etwaige Fehlermeldung (falls die Tabelle nicht vorhanden sein sollte) können Sie durch einen `try-catch`-Anweisung in Java abfangen (`SQLException`) und ignorieren.

b) Befüllen Sie die Tabelle `Mitarbeiter` durch entsprechende `INSERT`-Anweisungen, so dass sie folgende Datensätze enthält.

- (Schulz, 3000)
- (Maier, 4000)

c) Lesen Sie alle Datensätze der Tabelle `Mitarbeiter` aus. Führen Sie dazu eine entsprechende `SELECT`-Anweisung aus und durchlaufen Sie das zurückgelieferte `ResultSet` vollständig mit einer Schleife, wobei in jedem Schleifendurchlauf mittels Name und Gehalt der jeweiligen Mitarbeiter ausgegeben werden.

d) Erweitern Sie das unter c) beschriebene Auslesen der Datensätze so, dass auch die Spaltennamen ausgegeben werden. Geben Sie dabei die Namen der Spalten nicht fest an, sondern lesen Sie die Werte aus dem `ResultSet` aus.

> *Hinweis*: Verwenden Sie dazu die Methoden `ResultSet.getMetaData()` sowie `ResultSetMetaData.getColumnCount()` und `ResultSetMetaData getColumnName(int column)`.

e) Ändern Sie das Gehalt des Mitarbeiters Schulz auf 4500 unter Verwendung eines `UPDATE`-Statements. Zur Überprüfung der korrekten Ausführung lassen Sie erneut alle Datensätze wie unter d) beschrieben ausgeben.

3. Aufgabe (SQLJ)

Wiederholen Sie die in Aufgabe 1 und 2 beschriebenen Schritte mit SQLJ, d.h. schreiben Sie ein Java-Programm, dass mittels SQLJ auf die Datenbank zugreift und die in Aufgabe 2 genannten Operationen ausführt.

Hinweise zur Durchführung: Speichern Sie dazu Ihr SQLJ-Java-Programm mit der Dateiendung `.sqlj` ab. Rufen Sie anschließend den Precompiler mit

```
java -cp .:/<LIB>/sqlj.zip sqlj.tools.Sqlj
     -C-classpath=.:/<LIB>/sqlj.zip <PROGRAMM>.sqlj
```

wobei `<LIB>` durch den Verzeichnisnamen, in dem die Datei `sqlj.zip` liegt, und `<PROGRAMM>` durch den Dateinamen Ihres Programms ersetzt wird. Compilieren und Starten Sie das dadurch erzeugte Programm `<PROGRAMM>.java` wie gewohnt. Geben Sie dabei (neben den JAR-Dateien für den JDBC-Treiber) auch stets die Datei `sqlj.zip` im Classpath (Option `-cp`) an.

Zusatzaufgabe (Transaktionen, "Dirty Read")

In dieser Aufgabe wenden Sie Ihre in Aufgabe 2 gewonnen Fähigkeiten an und erstellen *zwei* Java-Programme, welche mittels je einer JDBC-Verbindung *gleichzeitig* auf die Datenbank zugreifen. Dabei sehen Sie, dass die Ausführung der Programme je nach verwendetem Isolation Level unterschiedliche Ergebnisse hervorbringen kann. Eine Übersicht zu Isolation Levels und wie sie in einer JDBC-Verbindung eingestellt werden, finden Sie unter http://java.sun.com/j2se/1.4.2/docs/api/java/sql/Connection.html#setTransactionIsolation(int). Zur Verwendung der Isolation Levels ist es weiterhin notwendig, nach dem Verbindungsaufbau das (standardmäßig aktivierte) Auto-Commit auszustellen. Ihre Programme beginnen daher nach dem Aufbau der Verbindung `con` stets mit

```
con.setTransactionIsolation(isoLevel);
con.setAutoCommit(false);
```

wobei `isoLevel` durch den entsprechenden Isolation Level zu ersetzen ist.

Im Folgenden sind zwei Programme tabellarisch in Pseudo-Code skizziert. Erstellen Sie damit zwei Java-Programme A und B, die jeweils eine Datenbankverbindung herstellen (und bei Programmende wieder abbauen) und die im Pseudo-Code definierten Operationen in der definierten Reihenfolge ausführen. Beachten Sie, dass in Programm A eine Pause eingebaut ist, so dass Programm A "wartet", dass Programm B etwas ausführt. Das Warten der Programme kann mittels einer einfacher Nutzereingabe realisier werden. So "wartet" der Java-Befehl `System.in.read()` solange, bis der Nutzer die Enter-Taste betätigt hat.

Programm A	Programm B

`UPDATE <Schema>.Mitarbeiter`
`SET Gehalt = Gehalt + 100`
`WHERE Name = 'Schulz'`	

`<PAUSE>`	*`Geh`* `= SELECT Gehalt`
	`FROM <Schema>.Mitarbeiter`
	`WHERE Name = 'Schulz'`

	`Geh = Geh + 50`

	`UPDATE <Schema>.Mitarbeiter`
	`SET Gehalt =` *`Geh`*
	`WHERE Name = 'Maier'`

	`con.commit()`
`con.rollback()`	

Erläuterung: Zunächst wird Programm A gestartet, dass ein `UPDATE` ausführt. Anschließend wartet Programm A (auf eine Nutzereingabe). Währenddessen wird Programm B gestartet, dass mittels `SELECT` eine Java-Variable *`Geh`* vom Typ *`int`* belegt und anschließend verarbeitet ("plus 50") . Anschließend wird dieser Wert in einer `UPDATE`-Anweisung verwendet und die Transaktion mit *`COMMIT`* bestätigt. Programm B ist somit beendet und Programm A läuft nun (nach Nutzereingabe) weiter, wobei es seine Transaktion mittels *`ROLLBACK`* zurücksetzt.

a) Führen Sie beide Programme zunächst im Isolation Level "`READ UNCOMMITTED`" aus. Beachten Sie, dass Sie vor dem Starten die Tabelle `Mitarbeiter` wieder in den Ursprungszustand zurücksetzen, d.h. sie muss (ausschließlich) die in Aufgabe 2b) genannten Daten enthalten.

b) Führen Sie beide Programme zunächst im Isolation Level "`READ COMMITTED`" aus. Beachten Sie auch hier, dass sie zunächst die Tabelle `Mitarbeiter` wieder in den Ursprungszustand zurücksetzen.

c) Vergleichen Sie den Inhalt der Tabelle `Mitarbeiter` nach dem ersten Durchlauf (a) mit dem Inhalt nach dem zweiten Durchlauf (b). Was stellen Sie fest? Begründen Sie Ihre Beobachtung.