3 Datenbankanbindung

Grundstruktur von Java – JDBC

JDBC (Java Database Connectivity) beschreibt eine universelle Programmschnittstelle (› API) zu allen SQL-fähigen Datenbanken. Für alle gängigen Datenbankverwaltungssysteme existiert ein spezielles, den JDBC-Konventionen genügendes Paket von Kommunikationsklassen (Treiber), das der Nutzer nur noch anschließen muss; die Kommunikation nutzt dann für alle Datenbanken die gleiche Klassenstruktur.

› API: application programming interface

Obwohl das Paket für komplexere Aufgaben sehr viele Klassen (mit vielen Methoden) bereitstellt, ist die Nutzung für „normale“ Aufgaben sehr einfach. Die Klassen Connection (verwaltet die Verbindung) und Statement (verwaltet das Absenden von Anweisungen an die Datenbank) bilden zusammen die Entsprechung für die in Kapitel 15 beschriebene Klasse DATENBANKVERBINDUNG.

15

Die Klasse ResultSet stellt die Fähigkeiten der Klasse ERGEBNISTABELLE zur Verfügung.

Aufbau der Verbindung

Zuerst muss der Treiber für die gewünschte Datenbank angebunden werden. Das geschieht mit der Klassenmethode *forName* der Klasse Class. So wird zum Beispiel der Treiber für eine MySQL-Datenbank angebunden:

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
```

Die Methode *newInstance* registriert das erzeugte Treiberobjekt in der Treiberverwaltung. Anschließend kann die Verbindung geöffnet werden. Die Klassenmethode *getConnection* der Treiberverwaltung baut die gewünschte Verbindung auf und liefert eine Referenz auf ein Objekt der Klasse Connection, mit dem die Verbindung verwaltet wird:

```
Connection conn = DriverManager.getConnection("jdbc:mysql:
                  //localhost/test?user=hugo&password=abcd");
```

Die Angabe der Datenbank erfolgt in der Form einer URL. Die Protokollangabe (vor „//“) bezeichnet eine Verbindung über JDBC zu einer MySQL-Datenbank. Der Pfad („localhost/test“) referenziert die Datenbank „test“ auf dem gleichen Rechner, auf dem das Programm ausgeführt wird. Statt „localhost“ kann jeder beliebige, gültige Rechnerbezeichner stehen. Nach „?“ folgen die Wertangaben für die Parameter user und password, um das Programm bei der Benutzerverwaltung von MySQL zu authentifizieren.

Um die Verbindung zu einer anderen Datenbank herzustellen, müssen nur die beiden oben angegebenen Anweisungen angepasst werden, z. B. für eine ODBC-Schnittstelle:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
Connection conn = DriverManager.getConnection("jdbc:odbc:test");
```

Ausführen von Anweisungen

JDBC erlaubt die parallele Ausführung mehrerer Anweisungen an die Datenbank. Daher ist die Verwaltungsklasse für Anweisungen (Statement) getrennt von der Klasse Connection für die Verwaltung der Verbindung. Ein neues Anweisungsobjekt erhält man mit

```
Statement stat = conn.createStatement();
```

Damit können nun beliebig viele Anweisungen an die Datenbank gesendet werden. Für Datenmanipulationsanweisungen nutzt man normalerweise die Methode *executeUpdate*:

```
int anzahl = stat.executeUpdate("DELETE FROM konto WHERE KontoNummer=27");
```

Das Ergebnis gibt an, wie viele Zeilen der Tabelle betroffen waren; insbesondere bei Sammeländerungen ist diese Information nützlich.
Für Abfragen wird die Methode *executeQuery* genutzt:

```
RestultSet res = stat.executeQuery("SELECT kontonummer, kontostand FROM
                        konto WHERE eigentuemer='Kunde1'");
```

Das Ergebnisobjekt der Klasse ResultSet verwaltet die auf dem Server vorgehaltene Ergebnistabelle. Daten werden nur übertragen, wenn sie gelesen werden.

Auswerten einer Ergebnistabelle

Die typische Auswertung der Ergebnistabelle geschieht in einer Wiederholung über alle Datensätze dieser Ergebnistabelle. Die Klasse ResultSet stellt dazu die Methode *next* zur Verfügung, die die Leseposition von einem Datensatz auf den nächsten weiterschaltet (positioniert). *next* meldet einen Wahrheitswert zurück, der angibt, ob der neue Datensatz existiert. Zum Lesen der Felder des aktuellen Datensatzes dienen Methoden der Namen *getInt*, *getFloat*, *getString* usw., deren Rückgabewert der angegebene Datentyp ist; Parameter ist der Feldname (als String) oder die laufende Nummer des Feldes, gezählt ab 1.

```
while res.next()
{
    System.out.println("Kontonummer: " + res.getInt("kontonummer") +
                        ", Stand: " + res.getFloat("kontostand");
}
```

Da nach dem Ausführen der Abfrage noch kein Datensatz verfügbar ist, muss erst *res.next()* aufgerufen werden, um den ersten Datensatz verfügbar zu machen. Das macht Wiederholungen einfacher und sorgt dafür, dass auch leere Ergebnistabellen korrekt verarbeitet werden.

Abschlussarbeiten

Ist eine Abfrage ausgewertet, so wird die Beabeitung mit der Methode *close* abgeschlossen. Dabei werden alle Ressourcen freigegeben, insbesondere die auf dem Server vorgehaltene Ergebnistabelle.
Auch die Klassen Statement und Connection stellen je eine Methode *close* bereit, um die Ressourcen freizugeben, die von Objekten dieser Klassen benötigt werden.

Übersicht über wichtige Methoden

Klasse Connection	
Statement createStatement()	Erzeugt ein neues Anweisungsobjekt.
void close()	Schließt die Verbindung.
Klasse Statement	
int executeUpdate(String Anweisung)	Führt die angegebene SQL-Anweisung aus.
ResultSet executeQuery(String Abfrage)	Führt die angegebene SQL-Abfrage aus.
void close()	Schließt die Anweisung.
Klasse ResultSet	
boolean next()	Positioniert auf den nächsten Datensatz der Ergebnistabelle.
boolean prev()	Positioniert auf den vorhergehenden Datensatz der Ergebnistabelle.

int getInt(String Feldname)	Liest den Wert des Feldes mit dem gegebenen Namen als ganze Zahl.
int getInt(int Feldnummer)	Liest den Wert des Feldes mit der gegebenen Nummer als ganze Zahl.
float getFloat(String Feldname)	Liest den Wert des Feldes mit dem gegebenen Namen als Kommazahl.
float getFloat(int Feldnummer)	Liest den Wert des Feldes mit der gegebenen Nummer als Kommazahl.
String getString(String Feldname)	Liest den Wert des Feldes mit dem gegebenen Namen als Text.
String getString(int Feldnummer)	Liest den Wert des Feldes mit der gegebenen Nummer als Text.
void close()	Schließt die Ergebnistabelle

Datenbankverbindung mit Python

Die Verwendung einer Datenbankverbindung mit der Sprache Python ist ähnlich wie bei Java. Hier muss der jeweils benötigte Treiber in das Anwenderprogramm importiert werden, dann lässt sich die Verbindung öffnen.

```
import MySQLdb
conn = MySQLdb.connect(host = "localhost", user = "hugo",
                       passwd = "abcd", db = "test")
```

Um Anweisungen an die Datenbank absetzen zu können, wird ähnlich wie bei Java ein Objekt einer eigenen Klasse benötigt, das auch die Ergebnistabelle verwaltet. Mit diesem Objekt, z. B. einem der Klasse Cursor, können dann SQL-Anweisungen ausgeführt werden:

```
cursor = conn.cursor()
cursor.execute("DELETE FROM konto WHERE KontoNummer = 27")
anzahl = cursor.rowcount
cursor.execute("SELECT kontonummer, kontostand FROM konto WHERE
               eigentuemer = 'Kunde1'")
```

Bei Abfragen holt man sich das Ergebnis als Menge von Tupeln, die dann abgearbeitet werden:

```
rows = cursor.fetchall()
for row in rows:
    print "Kontonummer: %d, Stand: %s" % (row[0], row[1])
```

Um auf die Spalten über Namen zugreifen zu können, müsste ein Objekt einer anderen Cursor-Klasse angelegt werden.
Werden das Cursorobjekt bzw. die Verbindung nicht mehr benötigt, werden sie geschlossen. Beim Schließen muss dafür gesorgt werden, dass die durchgeführten Veränderungen durch Aufruf der Methode *commit* bestätigt werden und damit endgültig persistent sind:

```
cursor.close()
conn.commit()
conn.close()
```