Datenbankprogrammierung mit Java

Einführung

Viele, wenn nicht sogar die meisten, aller Programme sind Datenbankanwendungen. In eigentlich allen Programmen werden Daten gespeichert und das geht besonders gut in Datenbanken. Im Gegensatz zum Speichern der Daten in Excel-Dateien kann man in Datenbanken die Daten sortieren, in ihnen suchen, sie strukturiert ablegen und vieles mehr. Java wäre nicht sinnvoll einsetzbar, wenn es nicht auch ein Modul zur Datenbankprogrammierung gäbe, die *JDBC* (**J**ava **D**ata**B**ase **C**onnectivity). Mit ihr ist der Zugriff auf Datenbanken möglich

JDBC

Die *JDBC* umfasst eine Sammlung von Klassen und Schnittstellen (Interfaces) und zusätzlich dazu einen Treiber-Manager un einen Treiber um auf die Datenbank zugreifen zu können. Bei diesem Treiber handelt es sich um den *ODBC*-Treiber (**O**pen **D**ata**B**ase **C**onnectivity), der in allen Windows-Versionen bereits integriert ist und auf eine Vielzahl an Datenbankarten (z.B. Access, dBase, FoxPro) zugreifen kann.

Wie bei allen anderen Aspekten von Java war auch diesmal wieder der universelle Einsatz des Datenbankzugriffs für die Java-Architekten wichtig. Der Treiber, der sich um den eigentlichen Zugriff kümmert liegt im Betriebssystem, er ist beliebig erweiterbar (um weitere Datenbankarten) und an die Besonderheiten der Datenbank angepasst. Durch die "Abgabe der Verantwortung" für den Datenbankzugriff ist der Java-Programmierer sehr flexibel; er braucht sich nur um die individuelle Verarbeitung der Daten aus der Datenbank kümmern. Mit wenigen Handgriffen lässt sich die Grundlage (die Datenbank) schnell ändern, ohne gleich das ganze Programm neu programmieren zu müssen. Auch die JDBC erfordert nicht sehr viel Konfigurations- und Programmierarbeit.

Es ist also für den Java-Programmierer praktisch völlig egal, welche Datenbank benutzt wird. Da automatisch von der JDBC der erforderlich Treiber verwendet wird.

Datenbankzugriff auf eine Access-Datenbank

Nun wollen wir den Datenbankzugriff mal ausprobieren. Dafür erstellen wir eine Access-Datei ("db1.mdb", 1 Tabelle "tab", mit 3 Feldern: Name: Text, 50; Vorname: Text, 50; Geburtstag: Datum).

Nun müssen wir etwas im System einstellen. Gehe in die ***Systemsteuerung*** und wähle dort ***ODBC-Datenquellen (32 Bit)*** in dem sich daraufhin öffnenden ***ODBC-Datenquellen-Administrator*** drücke auf den ***Hinzufügen-Button***, wähle im dauf aufpoppenden Dialog den ***MS-Access-Treiber*** aus. Trage bei Datenquellenname "***daten***" ein, weiter unten gibt es einen eingerahmten Bereich Datenbank, dort gibt es den Knopf ***Auswählen***. Nun muss du die Datei ***db1.mdb***, die du soeben erstellen (oder downgeloadet) hast, auswählen.

Das waren alle Vorarbeiten zum Datenbankzugriff. Dieser Teil war die Konfiguration des ODBC-Treibers.

Einfacher Datenbankzugriff mittels Konsolenprogramm

Grundsätzlich muss als erstes der passende Datenbanktreiber geladen werden. Wir verwenden die *JDBC-ODBC-Bridge* die bei der JDK mitgeliefert wird.

Der Datenbanktreiber ist eine Klasse. Seine Aufgabe ist es, ein Objekt dieser Klasse zu erzeugen und beim Treibermanager anzumelden. Dies geschieht mittels *Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");*

So dies wollen wir nun in einem kleinen Beispiel einsetzen, bei dem wir danach auch noch gleich die Verbindung zur Datenbank herstellen.

```
import java.sql.*;
public class Datenbank
{
  public static void main(String argv[])
  {
    Connection dbVerbindung=null;
    try
    {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      System.out.println("Treiber erfolgreich geladen...");
    }
    catch(ClassNotFoundException e)
    {
      System.out.println("Fehler beim Laden des Treibers"+e);
      System.exit(0);
    }
    try
    {
      dbVerbindung = DriverMana-
      ger.getConnection("jdbc:odbc:daten","","");
      System.out.println("Verbindung erfolgreich...");
      dbVerbindung.close();
      System.out.println("Verbindung geschlossen...");
    }
    catch(SQLException e)
    {
      System.out.println("DB-Verbindungsfehler: "+e);
      System.exit(0);
    }
  }
}
```

Neu ist die Variable *dbVerbindung* der Klasse *Connection*. Eine Verbindung mit einer Datenbank wird immer über ein Objekt der Klasse *Connection* hergestellt. Das Objekt erhält man durch den Aufruf der statischen Methode:
DriverManager.getConnetion(url,user,password)

url stellt eine Datenquelle dar, die so aufgebaut ist:
jdbc:Subprotokoll:Datenquellennamen

Da wir ODBC einsetzen, ist das Subprotokoll *odbc* und bei den Vorarbeiten zu diesem Beispiel haben wir unserer Datenquelle den Namen *daten* gegeben.

SQL-Zugriff

Jetzt könnte man sagen: Eh, super die Datenbank läuft. Aber toll, was bringt mir das! Da hat man recht! Noch passiert nichts. Wir müssen mit einem SQL-Zugriff Daten aus der Datenbank auslesen. Dies wollen wir nun tun.

Doch bevor wir Daten auslesen können, müssen wir erst einmal welche eingeben. Geben wir also vier bis fünf Datensätze ein (erfinden Sie einfach was!)
Bei SQL (Structured Query Language) handelt es sich um eine Abfragesprache, die den Inhalt einer Datenbank strukturiert ausgeben kann.
Hier erstmal der modifizierte Quelltext:

```
import java.sql.*;
public class Datenbank
{
  public static void main(String argv[])
  {
    Connection dbVerbindung=null;
    Statement sqlStatement=null;
    String sqlString="SELECT * FROM tab";
    ResultSet resultSet=null;
    try
    {
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      System.out.println("Treiber erfolgreich geladen...");
    }
    catch(ClassNotFoundException e)
    {
      System.out.println("Fehler beim Laden des Treibers"+e);
      System.exit(0);
    }
    try
    {
      dbVerbindung = DriverManager.getConnection("jdbc:odbc:daten","","");
      System.out.println("Verbindung erfolgreich...");
    }
    catch(SQLException e)
    {
      System.out.println("DB-Verbindungsfehler: "+e);
      System.exit(0);
    }
    try
    {
      sqlStatement = dbVerbindung.createStatement();
      System.out.println("SQL-Statement erzeugt...");
      resultSet = sqlStatement.executeQuery(sqlString);
      while(resultSet.next())
      Sys-
      tem.out.println(resultSet.getString(1)+"\t"+resultSet.getString(2)+"\t"+res
      ultSet.getDate(3));
      resultSet.close();
      System.out.println("resultSet-Objekt geschlossen...");
      sqlStatement.close();
      System.out.println("sqlStatement-Objekt geschlossen...");
      dbVerbindung.close();
```

```
        System.out.println("Verbindung geschlossen...");
      }
      catch(SQLException e)
      {
        System.out.println("Fehler beim DB-Zugriff!"+e);
        System.exit(0);
      }
    }
  }
```

SQL-Anweisungen werden über ein Objekt der Klasse *Statement* an die Datenbank geschickt. Die Objekte der Klasse *Statement* werden mit der Methode *createStatement()* der Klasse *Connection* erzeugt. Die SQL-Anweisungen werden in einer *String*-Variable zwischengespeichert. Dieser String wird über *sqlStatement* mit der Methode *executeQuery* an die Datenbank übergeben. Als Antwortobjekt dieser SQL-Anweisung erhält man ein Objekt der Klasse *ResultSet*.
Mit diesem Antwortobjekt kann man nun arbeiten. Wir geben die einzelnen Spalten der Tabelle aus. Für jeden Datentyp gibt es eine eine entsprechende *get*-Methode, die als Parameter die Spaltennummer hat. In der *while*-Schleife wird mit *next()* solange weitergesprungen bis das Ende der Datenbank erreicht ist (dann gibt die Methode *false* zurück und die Schleife kann nicht mehr wiederholt werden).

Abschluss

Das war ein ersten und einfacher Einstieg in die Datenbankprogrammierung mit Java. Es gibt noch sehr viele Ausbaumöglichkeiten. So ist es zum Beispiel unerlässlich die Oberfläche der Datenbank grafisch ansprechend zu gestalten.