Programmieren II

Dr. Klaus Höppner

Hochschule Darmstadt – Sommersemester 2010

Datenbanken

SQL

JDBC

Relationale Datenbanken

Ein Datenbanksystem ist ein System zur Speicherung von (großen) Datenmengen:

- effizient,
- widerspruchsfrei,
- dauerhaft

Benötigte Teilmengen in unterschiedlichen, bedarfsgerechten Darstellungsformen werden für Benutzer und Anwendungsprogramme bereit gestellt.

Hierbei sind relationale Datenbankmanagementsysteme (RDBMS) weit verbreitet.

Beispiele für RDBMS

- Oracle (kommerziell)
- DB 2 (IBM, kommerziell)
- MySQL (Open Source, früher MySQL AB, nun Sun), weit verbreitet auf Webservern, Teil von LAMP (Linux, Apache, MySQL, PHP).
- PostgreSQL (Open Source)
- Access (kommerziell, Teil von Microsoft Office)
- HSQLDB (Open Source), in Java implementiert, wird auch in OpenOffice benutzt.

Organisation von RDBMS

Eine relationale Datenbank ist eine eine Sammlung von

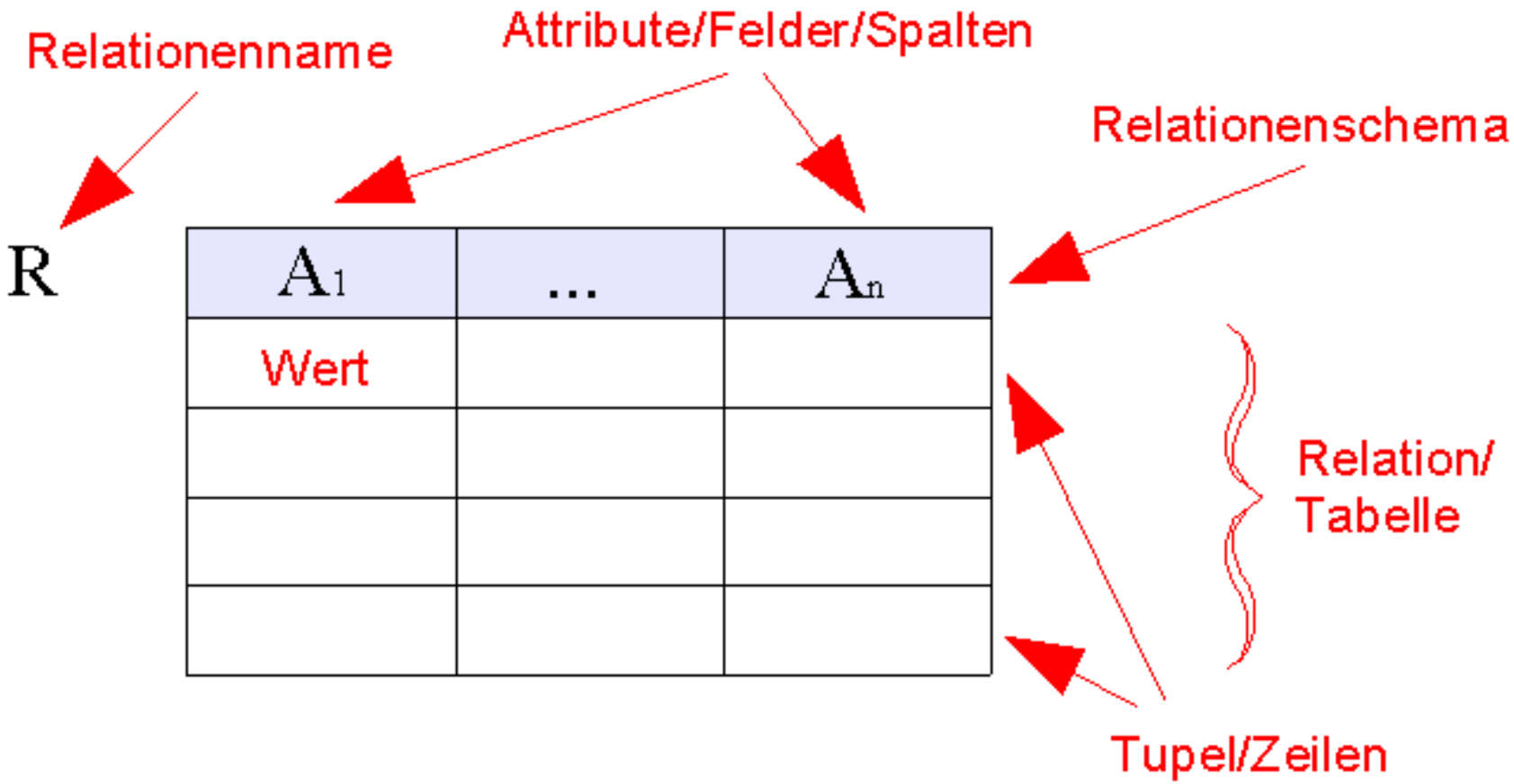
Tabellen (Relationen),

in welchen Datensätze abgespeichert sind.

Jeder Datensatz (*record*) ist eine Zeile (Tupel) in einer Tabelle. Jedes Tupel besteht aus einer Menge von Attributwerten, den Spalten der Tabelle.

In praktischen Anwendungen besteht eine Datenbank aus mehreren Relationen, die miteinander verknüpft sind.

Schaubild



Quelle: Wikipedia

Eigenschaften von Datenbanken

Moderne Datenbanken arbeiten in einem Transaktionsmodell, das die mit dem Acronym *ACID* beschriebenen Eigenschaften erfüllt:

Atomarität (*atomicity*) Eine Aktion wird entweder ganz oder gar nicht durchgeführt. Nicht zu Ende geführte Aktionen erzwingen automatisch einen *Rollback* $\rightarrow$ Zustand vor Start der Aktion.

Konsistenz (consistency) Jede Transaktion hinterlässt die Datenbank in einem konstistenten Zustand.

Isolation Transaktionen beeinflussen sich gegenseitig nicht!

Dauerhaftigkeit (*durability*) Transaktionen sind nach Abschluss dauerhaft, auch bei Systemabstürzen.

SQL

Die meisten RDBMS werden über

SQL (*structured query language*)

gesteuert. Die Funktionalität geht hierbei über den Namen hinaus:

- Erzeugen/Entfernen von Tabellen,
- Einfügen, Ändern, Löschen von Datensätzen,
- Abfragen von Datensätzen
- Definition von Indizes, Constraints, . . .

Leider verwenden die einzelnen RDBMS leicht unterschiedliche SQL-Dialekte.

Wichtige SQL-Befehle

- **CREATE TABLE** <table> <columns>
  Erzeugt eine neue Tabelle mit den angegebenen Spaltendefinitionen.
- **INSERT INTO** <table> (<columns>) **values** (<values>)
  Fügt einen neuen Datensatz in die angebene Tabelle ein.
- **UPDATE** <table> **SET** <column>=<value> **where** <constraint>
  ändert in den vorhandenen, über die Bedingung ausgewählten Datensätzen die angegebene(n) Spalte(n).
- **DELETE FROM** <table> **where** <constraint>
  entfernt die betreffenden Datensätze.
- **SELECT** <column> **FROM** <table>
  fragt Datensätze ab, optional eingeschränkt mit **WHERE** <constraint> und geordnet gemäß **ORDER BY** <column>

Beispieltabelle

Im folgenden Abschnitt wird eine einfache Beispieltabelle für Java-Anwendungen benutzt:

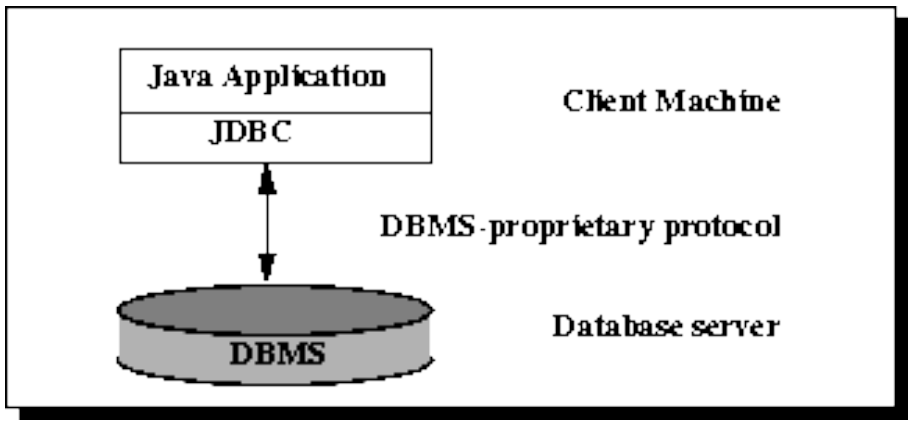
```
CREATE TABLE person (
  nachname varchar(40) NOT NULL,
  vorname varchar(40) NOT NULL,
  firma varchar(20)
)
```

Datenbank-Konnektierung in Java

Eine Standardmethode zur Konnektierung mit RDBMS in Java stellt

JDBC (*Java Database Connectivity*)

dar.



JDBC stellt hierbei eine standardisierte Zugriffs-Schnittstelle dar, über die auf eine Vielzahl von RDBMS zugegriffen werden kann. Hiermit entspricht JDBC in etwa ODBC für Windowsanwendungen und DBI für Perl.

Grundsätzliches Vorgehen

Für ein erfolgreiches Arbeiten mit einer Datenbank über JDBC sind folgende Schritte notwendig:

1. Datenbanktreiber laden
   Dieser sorgt für die Anbindung an die eigentliche Datenbank.
2. `Connection`-Objekt erzeugen
   Hiebei sind i. A. Typ, Name der Datenbank und Username, Passwort notwendig.
3. Einen SQL-Befehl über ein `Statement` absetzen.
4. Gegebenenfalls den `ResultSet` auswerten.

Beispiel: Mit der Datenbank verbinden

```
import java.sql.*;
public class Main {
    public static void main(String[] args) {
        try {
            Class.forName("org.hsqldb.jdbcDriver");
        } catch (ClassNotFoundException e) {
            System.exit(1);
        }

        try {
            Connection con = DriverManager.getConnection(
                "jdbc:hsqldb:file:data/mydb;shutdown=true", "SA",
            ...
            con.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Analyse

Im vorigen Beispiel soll eine HSQLDB-Datenbank konnektiert werden.

Hierzu muss zunächst der JDBC-Treiber für HSQLDB geladen werden. Dies geschieht mit dem Befehl `Class.forName("org.hsqldb.jdbcDriver")`, der die entsprechende Klasse sucht und in die Laufzeitumgebung lädt.
Dafür muss sich die Datei `hsqldb.jar` im `CLASSPATH` befinden.

Danach wird das `Connection`-Objekt erzeugt:

```
DriverManager.getConnection("jdbc:hsqldb:file:mydb", user, passwd)
```

(die Syntax zur Angabe von Server und Name der Datenbank variiert leider zwischen den Treibern).

Liste von JDBC-Treibern

DB	JDBC-Treiber
HSQLDB	org.hsqldb.jdbcDriver
MySQL	com.mysql.jdbc.Driver
Oracle	oracle.jdbc.driver.OracleDriver

Einfügen eines Datensatzes

Im Folgenden wird nun nach Aufbau der Verbindung ein Datensatz in die Tabelle eingefügt:

```
Statement st = con.createStatement();
st.executeUpdate("insert into person (nachname,vorname,firma)
            values ('Meier','Kurt','GSI')");
st.close();
```

Hierzu wird ein `Statement`-Objekt erzeugt. Dessen Methode `executeUpdate(...)` darf die Befehle `INSERT`, `UPDATE`, und `DELETE` enthalten.

Abfragen von Datensätzen

Nun werden die Datensätze und zeilenweise ausgegeben:

```
Statement st = con.createStatement();
ResultSet rs = st.executeQuery("select * from person");
ResultSetMetaData meta = rs.getMetaData();
while (rs.next()) {
    System.out.println("Zeile: "+rs.getRow());
    for (int i=1; i<=meta.getColumnCount(); i++) {
        System.out.println("Spalte "+meta.getColumnLabel(i)
                +": "+rs.getString(i));
    }
}
rs.close();
st.close();
```

Analyse

- Die Methode `executeQuery` dient Abfragen, die einen `ResultSet` zurück geben.
- Hierbei sind über die Methode `getMetaData()` die Metadaten des `ResultSet` zugänglich (insbes. Spaltenzahl und -namen).
- Über den `ResultSet` kann iteriert werden, solange `rs.next()` wahr ist.
- Die Werte zu den einzelnen Spalten sind auf zwei Wege zugänglich:
  - Über die Spaltennr. mit `rs.getString(index)`,
  - Über den Spaltennamen mit `rs.getString(name)`,
- Zeilen- und Spaltennummern beginnen bei 1!

Das Interface `Connection`

Das Interface `Connection` bietet im Wesentlichen folgende Methoden:

`Statement createStatement()` erzeugt ein `Statement`-Object.

`void commit()` schließt die Transaktion ab.

`void rollback()` macht alle Aktionen der Transaktion rückgängig.

`void setAutoCommit(boolean)` bestimmt, ob nach jedem Statement automatisch ein Commit durchgeführt wird.

`void close()` beendet die Verbindung.

`PreparedStatement prepareStatement(String sql)` bereitet eine SQL-Anweisung vor.

Das Interface `ResultSet`

Mit den Methoden von `ResultSet` kann man sich zeilenweise durch die Ergebnisse eines SQL-Statement durcharbeiten. Hierbei sind relative Bewegungen (`next()` und `previous()` und absolute Bewegungen (`first()`, `last()`, `absolute(int row)`) möglich. Achtung, nicht jeder JDBC-Treiber unterstützt die komplette Funktionalität!

Zum Lesen der Daten stehen die Methode `getXXX(int index)` und `getXXX(String label)` zur Verfügung. Diese bilden die SQL-Datentypen auf die Java-Datentypen ab.

Datentypen

Methode	SQL-Typ	Java-Typ
getInt()	INTEGER	int
getLong()	BIG INT	long
getFloat()	REAL	float
getDouble()	FLOAT	double
getBoolean()	BIT	boolean
getString()	VARCHAR	String
getString()	CHAR	String
getDate()	DATE	java.sql.Date
getTime()	TIME	java.sql.Time
getTimestamp()	TIME STAMP	java.sql.Timestamp