Client/Server-Programmierung

WS 2019/2020

Roland Wismüller
Betriebssysteme / verteilte Systeme
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: 17. Januar 2020

Client/Server-Programmierung

WS 2019/2020

18.10.2019

Roland Wismüller
Betriebssysteme / verteilte Systeme
roland.wismueller@uni-siegen.de
Tel.: 0271/740-4050, Büro: H-B 8404

Stand: 17. Januar 2020

Client/Server-Programmierung

WS 2019/2020

2 *Java Database Connectivity* (JDBC)

2.1 Überblick

- Java-API zum portablen Zugriff auf relationale Datenbank-Systeme
- Unabhängig von konkreter Datenbank-Implementierung
- Funktionen:
  - Verbindung zur Datenbank herstellen
  - Ausführung von SQL-Anweisungen
  - Zugriff auf Abfrage-Ergebnisse
- Vergleichbar mit ODBC, aber einfachere Schnittstelle
- Anschluß zur Datenbank über herstellerspezifische Treiber

2.2 Relationale Datenbanken und SQL

- Relationale Datenbank = Menge von Tabellen
  - jede Spalte hat Namen und Datentyp
  - jede Zeile enthält i.a. ein Feld, dessen Wert die Zeile eindeutig identifiziert (Primärschlüssel)
  - Aufbau festgelegt in Datenbank-Schema
- Beispiel:

ag_name

AG_ID	AG_NAME
1	BMW
2	Siemens
3	Thyssen

ag_data

ID	AG_ID	DAY	VALUE
7	3	9	102.30
9	1	9	99.10
12	2	8	30.45

Primär–schlüssel

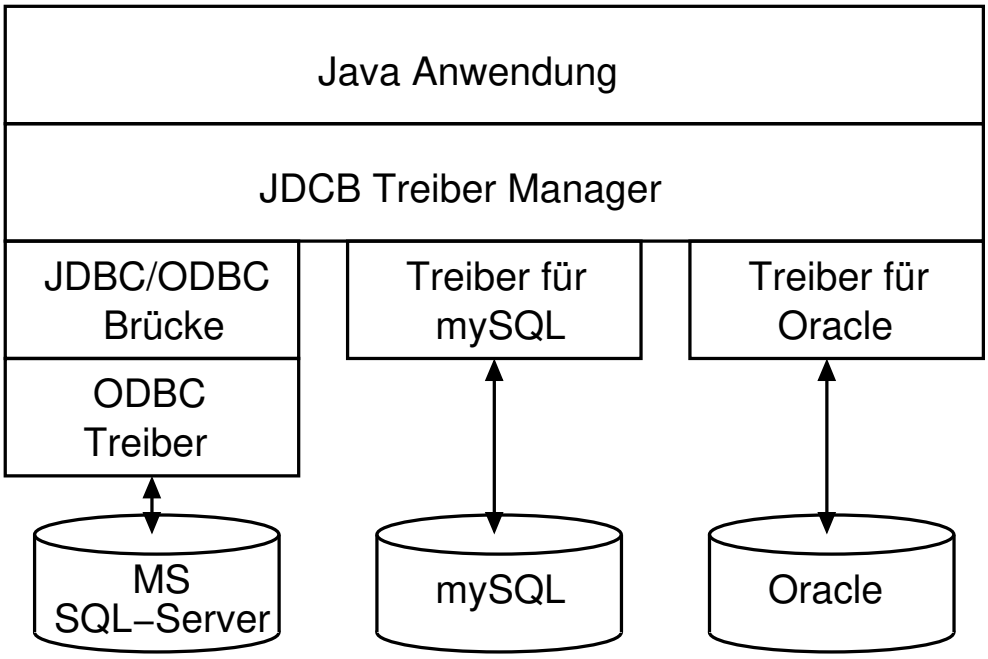
SQL

- Standardisierte Abfragesprache für relationale Datenbanken
- Erlaubt u.a.:
  - Abfrage von Daten (`SELECT`)
  - Erzeugung neuer Tabellen (`CREATE TABLE`)
  - Einfügen von Datensätzen (Zeilen) (`INSERT`)
  - Löschen von Datensätzen (`DELETE`)
  - Ändern von Datensätzen (`UPDATE`)
- Auswahl der Datensätze i.d.R. über deren Inhalt
  - häufig über Primärschlüssel

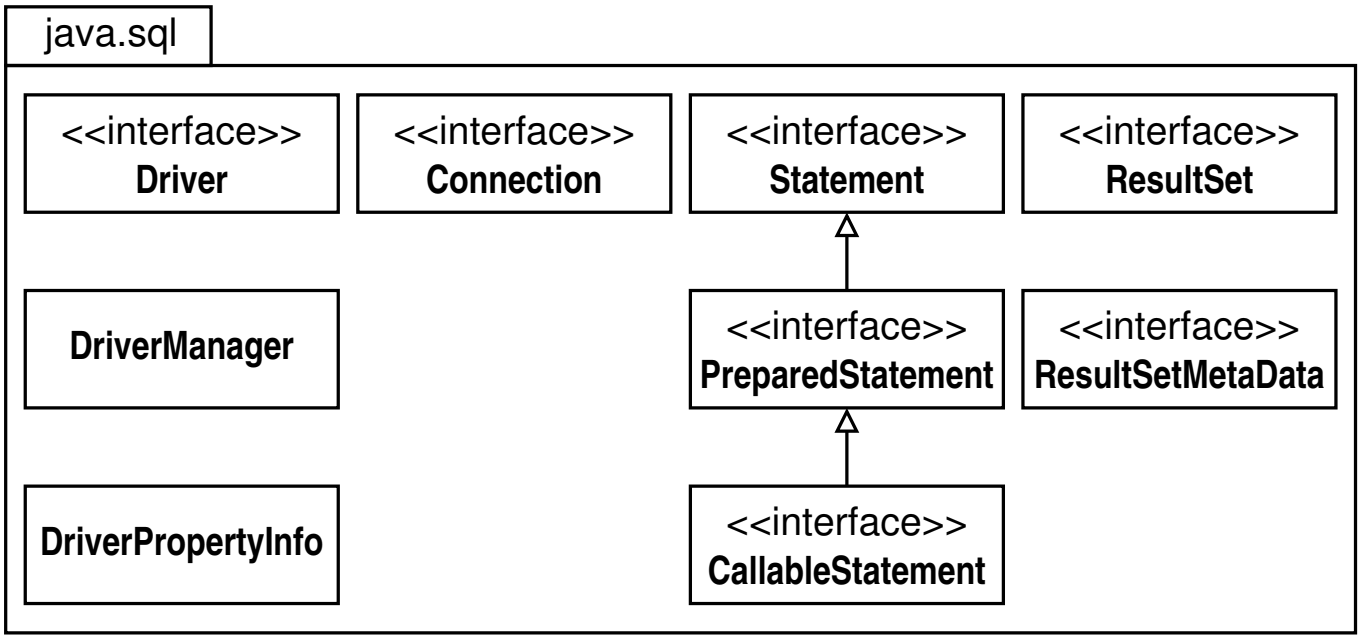
Beispiele für SQL-Anfragen

- ```
  SELECT AG_ID, AG_NAME
  FROM ag_name
  WHERE AG_NAME = 'Siemens'
  ```
  - liefert die Zeile für Siemens aus der `ag_name`-Tabelle
- ```
  SELECT ag_name.AG_NAME, ag_data.VALUE
  FROM ag_name, ag_data
  WHERE VALUE > 90 AND ag_name.AG_ID = ag_data.AG_ID
  ```
  - liefert Name und Kurs aller Aktien mit Kurs über 90
  - gibt Information aus zwei Tabellen zurück
    - Verbindung der Einträge über den Primärschlüssel (*Natural Join*)

2.3 Architektur von JDBC



Klassen des JDBC-Kerns



Klassen des JDBC-Kerns ...

- Interface `Driver`
  - Schnittstelle, die alle JDBC-Treiber implementieren müssen
  - neu geladener Treiber registriert sich bei `DriverManager`
- Klasse `DriverManager`
  - verwaltet `Driver`-Objekte
  - erzeugt Datenbank-Verbindungen (`Connection`)
- Klasse `DriverPropertyInfo`
  - erlaubt Definition spezieller Parameter beim Aufbau der Datenbank-Verbindung

Klassen des JDBC-Kerns ...

- Interface `Connection`
  - repräsentiert Sitzung mit ausgewählter Datenbank
  - erlaubt Erzeugung von `Statement`-Objekten
  - verwaltet Informationen zum Zustand der Datenbank
  - erlaubt Abfrage von Metadaten der Datenbank (Methode `getMetaData()`, Resultat: `DatabaseMetaData`)
    - z.B. unterstützte SQL-Versionen, Limitierungen des Datenbank-Systems, ...
- Interface `Statement`
  - zur Ausführung einer SQL-Anfrage
  - verwaltet auch Ergebnis der Anfrage (`ResultSet`)

Klassen des JDBC-Kerns ...

- Interface `PreparedStatement`
  - zur Ausführung einer vorkompilierten SQL-Anfrage
  - effizienter bei wiederholter Ausführung
- Interface `CallableStatement`
  - erlaubt Aufruf von *Stored Procedures*
    - SQL-Prozeduren, die in Datenbank selbst abgelegt sind
- Interface `ResultSet`
  - Ergebnis-Relation einer Datenbank-Anfrage
- Interface `ResultSetMetaData`
  - Metadaten zu den Spalten der Ergebnis-Relation
  - z.B. Name, Typ, vorzeichenbehaftet, ...

2.4 Ein Beispiel

```
import java.sql.*;
import java.lang.*;

public class Beispiel {
  public static void main(String[] args) {
    try {
      // Laden des JDBC-Treibers
      Class.forName("com.mysql.jdbc.Driver");
    }
    catch (ClassNotFoundException e) {
      System.out.println("Treiber nicht ladbar:" + e);
      return;
    }
```


2.4 Ein Beispiel ...

```
    try {
      // Verbindung zur Datenbank
      Connection con = DriverManager.getConnection(
        "jdbc:mysql://bslabserv01.lab.bvs/cspdb","","");

      // Erzeuge SQL-Anweisung und führe sie aus
      Statement stmt = con.createStatement();
      ResultSet rs = stmt.executeQuery(
        "SELECT AG_ID, AG_NAME FROM ag_name");

      // Ausgabe des Ergebnisses
      while (rs.next()) {
        System.out.println("" + rs.getInt("AG_ID") + ", "
                           + rs.getString("AG_NAME"));
      }
```

```
        // Alles schließen
        rs.close();
        stmt.close();
        con.close();
      }
      catch (SQLException e) {
        System.out.println("SQL Exception: "
                           + e.getMessage());
        e.printStackTrace(System.out);
      }
    }
}
```


2.5 Details zu JDBC

Laden der Treiber

- Vor Verwendung von JDBC müssen die Treiber geladen werden:
  - ```
    try {
      Class.forName("com.mysql.jdbc.Driver");
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    }
    catch (ClassNotFoundException e) { ... }
    ```
- Alternativ: Setzen der `jdbc.drivers` *Property*
  - ```
    java -Djdbc.drivers=com.mysql.jdbc.Driver:sun.jdbc.
        odbc.JdbcOdbcDriver ...
    ```

Verbindung zur Datenbank herstellen

- Verbindung wird durch `Connection`-Objekt repräsentiert
- Mehrere Datenbank-Verbindungen pro Anwendung möglich
- Erzeugung über
  - `Connection con = DriverManager.getConnection("url", "user", "password");`
- Variante von `getConnection()` erlaubt Definition von *Properties* für die Datenbank-Verbindung
- Wenn Verbindung nicht mehr benötigt wird:
  - explizites Schließen mit Methode `con.close()`
    - Freigabe von Netzwerk- und Speicherressourcen

Ausführung einer SQL-Anweisung

- Erzeugung eines `Statement`-Objekts
  - `Statement stmt = con.createStatement();`
- Ausführung der SQL-Anweisung
  - `ResultSet rs = stmt.executeQuery("SELECT ...");`
- Methoden `executeUpdate()` für Anfragen ohne Ergebnis, bzw. `execute()`, falls unbekannt, ob Ergebnis geliefert wird
  - `execute()` liefert `true`, falls Ergebnis vorhanden
- `Statement`-Objekt repräsentiert eine einzige SQL-Anfrage
  - `ResultSet` wird bei erneuter Anfrage über selbes `Statement`-Objekt ungültig
  - für simultane Anfragen: mehrere `Statement`-Objekte!

Zugriff auf die Resultate

- Ergebnis einer SQL-Anfrage (SELECT) ist eine Tabelle
- Struktur gekapselt in `ResultSet`-Objekt
- Methoden u.a.:
  - `next()`: setzt „Lesezeiger“ auf nächste Zeile
    - zu Begin steht Zeiger **vor** der ersten Zeile
    - Ergebnis `false`, falls keine Zeile mehr vorhanden
  - `get...(String name)` / `get...(int nr)`: liefert Inhalt des Feldes mit Spaltenname `name` bzw. Spaltennummer `nr`
    - mehrere Methoden für die verschiedenen Datentypen
    - `getString()` liefert immer String-Repräsentation
  - `close()`: Ressourcenfreigabe

SQL-Datentypen und Zugriffsmethoden (Auswahl)

SQL Typ(en)	Java Typ	Methode
CHAR, VARCHAR	`String`	`getString()`
NUMERIC, DECIMAL	`java.math.BigDecimal`	`getBigDecimal()`
BIT	`boolean`	`getBoolean()`
TINYINT	`byte`	`getByte()`
SMALLINT	`short`	`getShort()`
INTEGER	`int`	`getInt()`
BIGINT	`long`	`getLong()`
REAL	`float`	`getFloat()`
FLOAT, DOUBLE	`double`	`getDouble()`
BINARY, VARBINARY	`byte[]`	`getBytes()`
DATE	`java.sql.Date`	`getDate()`

Ausführung vorkompilierter SQL-Anfragen

- Für wiederkehrende, ähnliche Aufgaben sind vorkompilierte SQL-Anfragen (`PreparedStatement`) effizienter
  - die Anfragen sind auch parametrisierbar
- Auch Sicherheitsvorteil gegen SQL Injection
- Erzeugung eines `PreparedStatement`-Objekts
  - ```
    PreparedStatement stmt = con.prepareStatement(
        "INSERT INTO Employees (Name, Phone) (?, ?)");
    ```
  - `?` als Platzhalter für Parameter
- Ausführung der Anfrage mit konkreten Parametern
  - ```
    stmt.clearParameters();
    stmt.setString(1, "Jimmy Dean");    // erster Param.
    stmt.setString(2, "201 555-7685"); // zweiter Param.
    stmt.executeUpdate();               // kein Ergebnis
    ```


Ausführung von *Stored Procedures*

Oracle PL/SQL–Prozedur (Im Datenbanksystem gespeichert)

```
CREATE OR REPLACE
PROCEDURE sp_interest
(id IN INTEGER
bal IN OUT FLOAT) is
BEGIN
SELECT balance
INTO bal
FROM accounts
WHERE account_id = id;

bal = bal + bal * 0.03;

UPDATE accounts
SET balance = bal
WHERE account_id = id;

END;
```

Aufruf der Prozedur über JDBC

```
CallableStatement stmt
  = con.prepareCall(
    "{call sp_interest(?,?)}");
stmt.registerOutParameter(2,
                    Types.FLOAT);
stmt.setInt(1, accountID);
stmt.setFloat(2, 2343.23);
stmt.execute();
out.println("New Balance: "
           + stmt.getFloat(2));
```

Transaktionen

- Verantwortlich für Transaktionen: `Connection`-Objekt
- Methoden zur Steuerung von Transaktionen:
  - `setAutoCommit()` - automatisches Festschreiben?
    - Voreinstellung: jede SQL-Anweisung wird als individuelle Transaktion ausgeführt
  - `commit()` - Festschreiben der Transaktion
  - `rollback()` - Abbruch der Transaktion
  - `setTransactionIsolation()` - Isolationsebene festlegen
    - `TRANSACTION_NONE`, sowie die vier Isolations-Ebenen nach ANSI/ISO-SQL99 (☞ **VS, 7.4**):
      - *read uncommitted*, *read commited*, *repeatable read*, *serializable*
    - Voreinstellung ist vom Treiber abhängig

Transaktionen ...

```
try {
  // Höchste Isolationsebene
  con.setTransactionIsolation(TRANSACTION_SERIALIZABLE);
  // Transaktionen mit mehreren SQL—Anweisungen zulassen
  con.setAutoCommit(false);
  // SQL—Anweisungen
  stmt.executeUpdate("UPDATE inv SET onhand = 10 WHERE id = 5");
  stmt.executeUpdate("INSERT INTO shipping (qty) VALUES (5)");
  // Commit aller Aktionen seit letztem Commit/Rollback
  con.commit();
}
catch (SQLException e) {
  // Rückgängigmachen aller Änderungen
  con.rollback();
}
```

2.6 Zusammenfassung

- JDBC erlaubt portablen Zugriff auf relationale Datenbanken
  - Abfragen über SQL
- Grundsätzlicher Ablauf:
  - Laden des Treibers (`Class.forName()`)
  - Verbindung zur Datenbank herstellen (`Connection`)
  - SQL-Anweisung erzeugen (`Statement`)
  - SQL-Anweisung ausführen, Ergebnis auslesen (`ResultSet`)
- Daneben: Unterstützung für
  - vorkompilierte SQL-Anweisungen und *Stored Procedures*
  - Transaktionen