

Datenbanksysteme

Programmieren von Datenbankzugriffen mit JDBC

Burkhardt Renz

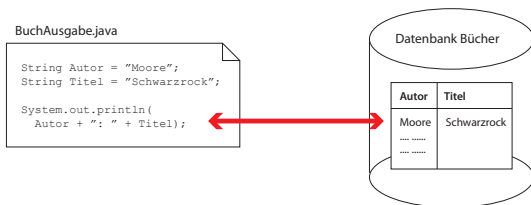
Fachbereich MNI
Technische Hochschule Mittelhessen

Sommersemester 2018

Übersicht

- Architektur von JDBC
 - Ziele von JDBC
 - Grundstruktur eines JDBC-Programms
 - Grundlegendes Beispiel
 - Überblick über das Package `java.sql`
- Datenretrieval mit JDBC
 - Parametrisierte Anweisungen
 - Metadaten
- Datenmodifikation mit JDBC
 - Ändernde Anweisungen
 - Änderungen über einen Cursor
- Transaktionen mit JDBC
 - Arbeiten mit Transaktionen
 - Einstellen des Isolationslevels

Fragestellung



- Wie kommen Werte aus der Datenbank in die Variablen unserer Anwendungen?
- Wie können wir Werte in unserem Programm in der Datenbank speichern?

Varianten der Zugriffstechnik

SLI – Statement Level Interface

Einbettung von SQL-Anweisungen in den Programmcode

Verarbeitung durch einen Präprozessor

Beispiele: embedded SQL in C (eSQL/C), SQLJ

CLI – Call Level Interface

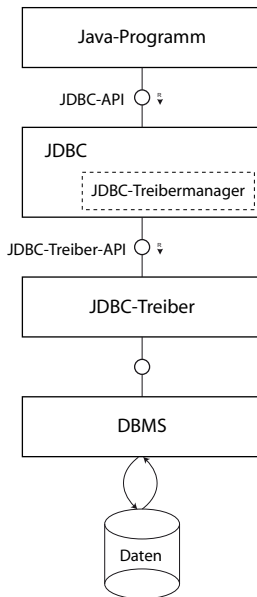
Bibliothek mit Funktionen bzw. Klassen und Methoden für den Zugriff auf das DBMS

Beispiele: ODBC (C/C++), JDBC (Java), ADO.NET (C#)

Ziele von JDBC

- SQL als Sprache für den Datenbankzugriff
- Gleichzeitiger Zugriff auf mehrere DBMS bzw. Datenbanken
- „Adaptives“ Programmiermodell
- Einfachheit („Keep it simple“)
- Robustheit, Verfügbarkeit, Skalierbarkeit
- Grundlage für andere Zugriffstechniken wie SQLJ oder JPA (Java Persistence API)

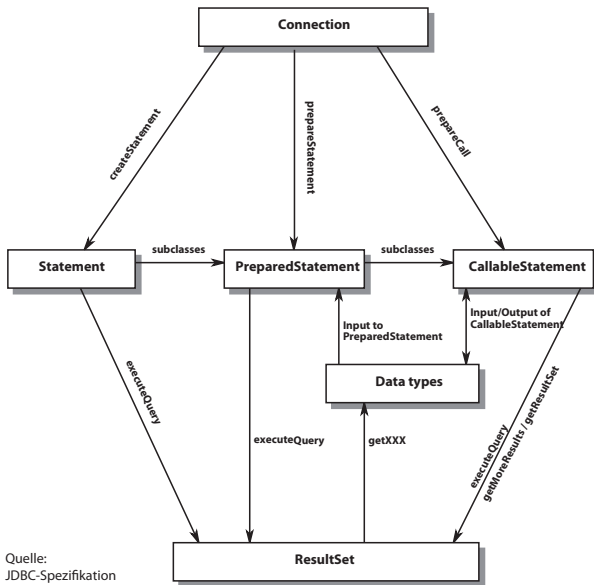
Architektur



Grundlegendes Beispiel

- Beispiel BuchJDBC.java in IntelliJ
- Einbinden des JDBC-Treibers, hier: JDBC-Treiber für PostgreSQL
- Schritt 1: JDBC-Treiber laden
- Schritt 2: Verbindung zum DBMS und zur Datenbank herstellen
- Schritt 3: Objekt für SQL-Anweisung erzeugen
- Schritt 4: DBMS Anweisung direkt ausführen lassen
- Schritt 5: Ergebnis anzeigen durch Iteration über den Cursor auf die Ergebnismenge
- Schritt 6: Fehler abfangen und Ressourcen freigeben

Interfaces und Klassen von JDBC



Quelle:
JDBC-Spezifikation

Übersicht

- Architektur von JDBC
 - Ziele von JDBC
 - Grundstruktur eines JDBC-Programms
 - Grundlegendes Beispiel
 - Überblick über das Package `java.sql`
- Datenretrieval mit JDBC
 - Parametrisierte Anweisungen
 - Metadaten
- Datenmodifikation mit JDBC
 - Ändernde Anweisungen
 - Änderungen über einen Cursor
- Transaktionen mit JDBC
 - Arbeiten mit Transaktionen
 - Einstellen des Isolationslevels

Arten von Statements

- `Statement`
wird vom DBMS übersetzt, optimiert und ausgeführt
- `PreparedStatement`
zweistufiges Verfahren
Schritt 1: DBMS übersetzt und optimiert
Schritt 2: Anweisung kann mehrfach mit immer neuen Parametern ausgeführt werden
- `CallableStatement`
zum Aufruf von Stored Procedures

Beispiel einer parametrisierten Anweisung

- Wir möchten Bücher bestimmter Autoren suchen, siehe BuchSuche.java
- Parametrisierte Anweisungen verwenden *Platzhalter*
- Zuerst wird im DBMS der Zugriffsplan erstellt
- In der Schleife wird derselbe Zugriffsplan immer wieder aufgerufen

SQL-Injection

- Implementierung der Suche nach Büchern ohne Platzhalter, siehe BuchInjection.java
- Auf den ersten Blick erfüllt es dieselbe Funktionalität
- Aber: der Inhalt der Benutzereingabe wird vom DBMS interpretiert
- Deshalb ist SQL-Injection möglich
- Demo:
Neue Tabelle demo mit einem Feld msg anlegen
Einen Datensatz einfügen
- Angriff: Der Angreifer möchte die Tabelle demo löschen
Was muss er eingeben?
- Diskussion

Ermitteln der Metadaten zu einer Ergebnismenge

- Interface `ResultSetMetaData` mit den Methoden:
- `getColumnCount()`
- `getColumnName(int column)`
- `getColumnType(int column)`

DatabaseMetaData

- Informationen über das DBMS und die Datenbank erhält man via das Interface DatabaseMetaData, zum Beispiel:
- `getDatabaseProductName()`
- `getDriverName()`
- `getTables(...)`
- `getColumns(...)`
- `supportsANSI92FullSQL()`
- ...
unzählige Informationsfunktionen

Übersicht

- Architektur von JDBC
 - Ziele von JDBC
 - Grundstruktur eines JDBC-Programms
 - Grundlegendes Beispiel
 - Überblick über das Package `java.sql`
- Datenretrieval mit JDBC
 - Parametrisierte Anweisungen
 - Metadaten
- Datenmodifikation mit JDBC
 - Ändernde Anweisungen
 - Änderungen über einen Cursor
- Transaktionen mit JDBC
 - Arbeiten mit Transaktionen
 - Einstellen des Isolationslevels

Ändernde Anweisungen

- `executeQuery` für „select ...“
gibt ein Objekt vom Typ `ResultSet` zurück
- `executeUpdate` Methode von `Statement` für
„update ...“ oder
„insert ...“
gibt die Zahl der betroffenen Zeilen zurück
- Was tun, wenn man den Typ der Anweisung zur Compile-Zeit nicht kennt?
- `execute` Methode von `Statement` für beliebige Anweisungen
gibt einen booleschen Wert zurück:
`true` bedeutet, dass eine Ergebnismenge erstellt wurde, kann man abholen mit `getResultSet`
`false` bedeutet, dass Daten geändert wurden, die Zahl der geänderten Zeilen kann man abholen mit `getUpdateCount`

Arten von ResultSets

- Art der Bewegung des Cursors
 - TYPE_FORWARD_ONLY
 - TYPE_SCROLL_INSENSITIVE
 - TYPE_SCROLL_SENSITIVE
- Lesender oder ändernder Cursor
 - CONCUR_READ_ONLY
 - CONCUR_UPDATABLE

Navigieren in ResultSets

- `next()`
- `previous()`
- `first()`
- `last()`
- `beforeFirst()`
- `afterLast()`
- `relative(int rows)`
- `absolute(int r)`

Cursor verwenden in ResultSets

- Werte lesen
 - `rs.getString(1)`
 - `rs.getString("author")`
- Werte ändern
 - auf den entsprechenden Datensatz navigieren
 - `rs.updateString("author", "Geänderter Autor")`
 - `rs.updateRow()`
- Datensätze einfügen
 - `rs.moveToInsertRow()`
 - `rs.UpdateString("author", "Neuer Autor")`
 - `rs.UpdateString("title", "Neuer Titel")`
 - ...
 - `rs.insertRow()`

Übersicht

- Architektur von JDBC
 - Ziele von JDBC
 - Grundstruktur eines JDBC-Programms
 - Grundlegendes Beispiel
 - Überblick über das Package `java.sql`
- Datenretrieval mit JDBC
 - Parametrisierte Anweisungen
 - Metadaten
- Datenmodifikation mit JDBC
 - Ändernde Anweisungen
 - Änderungen über einen Cursor
- Transaktionen mit JDBC
 - Arbeiten mit Transaktionen
 - Einstellen des Isolationslevels

Auto-Commit-Modus

- Auto-Commit-Modus = Jede einzelne SQL-Anweisung wird automatisch in einer Transaktion durchgeführt, also automatisch bestätigt
- Für welche Art von Anwendungen ist der Auto-Commit-Modus *nicht* geeignet?
- Ausschalten des Auto-Commit-Modus:
`con.setAutoCommit(false)`
- Nun muss man im Programm das Transaktionsende bestätigen oder ein „Rollback“ veranlassen:
`con.commit()`
`con.rollback()`

Blaupause für Transaktionen

```
boolean autoCommit = con.getAutoCommit();
Statement stmt;
try {
    con.setAutoCommit( false );
    stmt = con.createStatement();
    stmt.execute(...);
    stmt.execute(...);
    stmt.execute(...);
    ...
    con.commit();
} catch(SQLException sqle) {
    con.rollback();
} finally {
    stmt.close();
    con.setAutoCommit( autoCommit );
}
```

Isolationslevel in JDBC

- Im Interface `Connection` werden die Isolationslevel definiert:

`TRANSACTION_NONE`

`TRANSACTION_READ_UNCOMMITTED`

`TRANSACTION_READ_COMMITTED` (Default in JDBC)

`TRANSACTION_REPEATABLE_READ`

`TRANSACTION_SERIALIZABLE`

- Einstellen durch

`con.setTransactionIsolation(int Level)`

- Das eingestellte Level gilt dann für alle folgenden Transaktionen, bis es umgestellt wird