Java Database Connectivity (JDBC)

Überblick

- Datenbankschnittstelle der Java-Plattform
- einheitliche Schnittstelle zu Datenbanken verschiedener Hersteller
- **Aufgaben**: Aufbau der Datenbankverbindung
  Weiterleitung von SQL-Anfragen an die Datenbank
  Umwandlung der Ergebnisse in eine für Java nutzbare Form
- eigene Treiber für jede spezifische Datenbank
- verschiedene Drivertypen hinsichtlich der Kommunikation mit der DB
- JDBC-Klassen im Java-Package java.sql
- Klassen:
  DriverManager: Laden von Treibern
  Connection: Datenbankverbindung
  Statement: Ausführung von SQL-Anweisungen über die Verbindung
  ResultSet: Verwaltung der Ergebnisse einer Anfrage,
  Zugriff auf einzelne Spalten

Datenbankanfragen mit JDBC

Phase 1: Connect
- **Datenbankverbindung herstellen**
- Driver registrieren (übersetzt JDBC Methodenaufrufe in herstellerspezifische Datenbank-Kommandos)
- Verbindung zur Datenbank

Phase 2: Query
- **Datenbankanfrage**
- Erzeuge ein Statement
- Abfrage auf Datenbank

Phase 3: Process Results
- **Verarbeiten der Ergebnisse**
- Durchlaufen der Ergebnisse
- Zuweisung der Ergebnisse an Java-Variablen

Phase 4: Close
- Schließe ResultSet
- Schließe Statement
- Schließe Connection

Phase 1: Connect

1. Registriere Driver

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

2. Connect zur Datenbank

```
Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@server:1521:SID",
                                              "userid","password");
```

Oracle JDBC Driver: **Thin Client Driver**

- Komplett in Java geschrieben, Typ 4 Driver
- Verwendet das Oracle Net Service Protokoll

Phase 2: Query

- Statement Objekt sendet SQL-Befehl zur Datenbank
- aktive Connection erforderlich
- Statement hat drei Methoden, um ein SQL-Statement zu erzeugen:
  `executeQuery()` für QUERY Statements
  `executeUpdate()`für INSERT, UPDATE, DELETE, oder DDL Statements
  `execute()` für beliebiges Statement

1. Erzeuge ein leeres Statement Objekt

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

2. Führe das Statement aus

```
ResultSet rset = stmt.executeQuery(statement);

int count = stmt.executeUpdate(statement);

boolean isquery = stmt.execute(statement);
```

Phase 2: Beispiele

Ausführen eines SELECT Statement

```
Statement stmt = conn.createStatement();
ResultSet rset = stmt.executeQuery("SELECT ANr, Bezeichnung FROM Artikel");
```

Ausführen eines DELETE Statement

```
Statement stmt = conn.createStatement();
int rowcount = stmt.executeUpdate("DELETE FROM Artikel WHERE ANr = 1234");
```

Phase 2: Transaktionssteuerung

- Schnittstellen zur Steuerung von Transaktionen: Connection-Objekt
- AutoCommit-Modus – jede SQL-Anweisungen ist separate Transaktionen
- Zusammenfassung von mehreren Änderungen zu einer Einheit → AutoCommit ausschalten

```
Connection conn = ...
...
conn.setAutoCommit(false);
```

- Transaktionen müssen dann explizit mit `commit()` oder `rollback()` abgeschlossen werden

```
Connection conn = ...;
...
if(ok)
   conn.commit();
else
   conn.rollback();
```

Phase 3: Process Results

- JDBC liefert die Ergebnisse einer Query in einem **`ResultSet`** Objekt
- ein **`ResultSet`** besitzt einen Cursor, der auf den aktuellen Datensatz zeigt
- Methode **`next()`** zum satzweisen Durchlaufen des `ResultSet`
- Methoden **`getString()`**, **`getInt()`** usw. für Wertzuweisung an Java-Variablen

1. Durchlaufen des Result Set

```
while (rset.next()) { ... }
```

2. Nutze getXXX() zum Lesen der Spaltenwerte

```
while (rset.next())
{
   int    anr   = rset.getInt("ANr");          //Spaltenname
   String bez   = rset.getString(2);           //Spaltennummer
   // Variablen verarbeiten
   System.out.println(anr+' '+bez);

}
```

Phase 3: Behandlung von SQL Nullwerten

- primitive Java Typen haben keine Nullwerte
- primitive Datentypen nicht verwenden, wenn Nullwerte erwartet werden
- Methode **`ResultSet.wasNull()`**, um zu bestimmen, ob eine Spalte einen Nullwert enthält

```
while (rset.next())
{
   String bezeichnung = rset.getString("Bezeichnung");

   if (rset.wasNull())
   {
       ... // Handle null value
   }
   ...
}
```

Phase 4: Close

1. Schließe ResultSet Objekt

```
set.close();
```

2. Schließe Statement Objekt

```
stmt.close();
```

3. Schließe Connection

```
conn.close();
```

Prepared Statement

- ist geeignet, wenn Statement mehr als einmal ausgeführt werden muss
- ein **`Prepared Statement`** Objekt enthält vorübersetzte SQL-Befehle
- ein **`Prepared Statement`** kann Variablen enthalten, die jedesmal bei Ausführung definiert werden

Erzeuge Prepared Statement, identifiziere Variablen mit ?

```
PreparedStatement pstmt = conn.prepareStatement("INSERT INTO Artikel
                                                 VALUES (?,?) ");
```

```
PreparedStatement pstmt = conn.prepareStatement("SELECT ANr, Bezeichnung "+
                                                "FROM Artikel "+
                                                "WHERE ANr BETWEEN ? AND ? ");
```

Ausführen eines Prepared Statement

1. Variablen-Werte übergeben

```
XXX paramValue;
pstmt.setXXX(paramIndex,paramValue);                    //paramIndex: 1,2,...
```

2. Statement ausführen

Methoden: `pstmt.executeQuery();`
`pstmt.executeUpdate();`

```
PreparedStatement pstmt = conn.prepareStatement("UPDATE Artikel "+
                                                "SET Bezeichnung = ? "+
                                                "WHERE ANr = ?");

pstmt.setString(1,"SCSI Kabel");
pstmt.setInt(2,101);

int count = pstmt.executeUpdate();
if(count>0)
   System.out.println("Änderung erfolgreich!");
```

Zugriff auf Metadaten des ResultSet

Klasse **`ResultSetMetaData`** enthält Informationen über das Ergebnis einer Anfrage:

- Anzahl Spalten
- Spaltenname und Datentyp
- Eigenschaften wie Zulässigkeit von Nullwerten

etc.

Beispiel:

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

ResultSet rset = stmt.executeQuery("SELECT * FROM Artikel "+
                                   "ORDER BY Bezeichnung");

ResultSetMetaData rsmetadata = rset.getMetaData();

int spalten = rsmetadata.getColumnCount();

for(int i=1;i<=spalten;i++)
{
   System.out.println("Spaltenname: "+rsmetadata.getColumnName(i));
   System.out.println("Datentyp:    "+rsmetadata.getColumnTypeName(i));
   System.out.println("Laenge:      "+rsmetadata.getPrecision(i));
   System.out.println("isNullable:  "+rsmetadata.isNullable(i));
}
```