Datenbankanwendungen (JDBC)

Hierarchie:

Connection

Transaction

Statement

Connection

Aufbau (klassisch):

Registrierung des JDBC Driver beim DriverManager:

Class.forName(JDBC Driver);

Eigentlicher Verbindungsaufbau über DriverManager:

Connection conn =

Driver Manager get Connection (DB URL, Username, Password);

Inhalt der DB URL:
(z.B. jdbc:oracle:thin:@itlab34.ba-mannheim.de:1521:test17)

JDBC Driver (jdbc:oracle:thin)
Rechnername oder IP-Adresse
Port, auf dem DB Connections erwartet werden (Oracle Listener: 1521)
i.a. Datenbank, für Oracle die SID (Instanz ID)

Ende:

conn.close();

Connection via DataSource Interface:

DataSource wird mittels JNDI (Java Naming and Directory Interface) bekannt gemacht (z.B. in den Deployment Descriptoren eines J2EE Servers)

DataSource

- **wird (in der Applikation) über den definierten Namen angesprochen**
- **liefert eine Connection**

```
Context Ctx = new InitialContext();

DataSource ds = (DataSource)
Ctx.lookup("java:/comp/env/jdbc/oracleds");

Connection conn = ds.getConnection();
```

Vorteile:

- **Transparenz**
- **Unterstützung von Connection Pooling**
- **Unterstützung verteilter Transaktionen**

JDBC Driver

Typ	Name	Client Software	Server Software
1	JDBC/ODBC Bridge	JDBC Driver, ODBC Driver, DB Client	
2	Application Driver	JDBC Driver, DB Client	
3	Net Driver	JDBC Driver	JDBC “Server” DB Client / DBMS
4	Thin Driver	JDBC Driver	DBMS

Tendenz auch bei Tier 3 Architekturen => Typ 4

Fehlerbehandlung

Behandlung von SQLExceptions:

```
try {
  // Datenbankzugriffe
catch (SQLException SQLe) {
  while (SQLe != null) {
    // Fehlerbehandlung
    SQLe = SQLe.getNextException();
  }
}
```

Zur Auswertung der SQL Exception stehen folgende Methoden zur Verfügung:

- **getSQLState() (liefert Fehlercode gemäß SQL Standard)**
- **getErrorCode() (liefert herstellerspezifischen Fehlercode)**

SQL Warnings müssen explizit behandelt werden:

```
try
  {
  …
  stmt.executeUpdate( … );
  sqlw = stmt.getWarnings();
  while( sqlw != null)
    {
      // Behandlung
      sqlw = sqlw.getNextWarning();
    }
  …
  }
catch ( SQLException SQLe)
  {
    …
  }
```

Transaction

Steuerung über Connection

AutoCommit Modus

conn.setAutoCommit (true/false);

AutoCommit true: 1 Transaktion besteht aus 1 Statement

(automatisches Commit nach Statement Ende)

=> echte Transaktionen nur für AutoCommit false

Steuerung:

conn.commit();

conn.rollback();

Savepoints (Strukturierung von TX)

Savepoint savepoint1 = conn.setSavepoint ();

…

conn.rollback(savepoint1);

Isolation Level

conn.setIsolationLevel (…);

Isolation Level bestimmt Isolation (und damit, welche Sperren gesetzt bzw. beachtet werden) für lesende Zugriffe

Schreibende Zugriffe erfordern immer exklusive Sperren bis zum TX Ende (unverträglich mit beliebigen Sperren)!

JDBC Isolation Level

None	**Transaktionen werden nicht unterstützt**
read uncommitted	**Bereitschaft, nicht bestätigte Daten zu lesen**
read committed	**Nur bestätigte Daten werden gelesen**
repeatable read	**Wiederholtes Lesen einer Zeile innerhalb einer TX führt zum gleichen Ergebnis**
Serializable	**Wiederholtes Ausführen einer Abfrage innerhalb einer TX führt zum gleichen Ergebnis (keine Phantom Inserts!)**

Es gibt DBS, die nicht alle Isolation Level implementieren (z.B. Oracle)

Die Implementierung der Isolation Level ist von DBS zu DBS unterschiedlich!

Statement

Hierarchie:

Statement

PreparedStatement

CallableStatement (hier nicht behandelt)

Statement Interface

Erzeugen

Statement stmt = conn.createStatement();

Ausführen

Falls keine Ergebnismenge geliefert wird

int rowcount = stmt.executeUpdate(“update…”);

Falls eine Ergebnismenge (ResultSet) geliefert wird (select)

ResultSet rs = stmt.executeQuery("select….“);

Generell

boolean hasResultSet = stmt.execute("...“);

Zugriff auf Rowcount: stmt.getRowCount();

Zugriff auf Resultset: stmt.getResultSet ();

Ressourcen freigeben

stmt.close();

PreparedStatement Interface

Bsp.: Lesen der Daten zu einem Mitarbeiter (empno, ename, sal)

```
select  empno, ename, sal
from emp
where empno = WERT
```

Erzeugen

```
PreparedStatement pstmt =
        conn.prepareStatement
          (“select  empno, ename, sal from emp where empno = ?”);
```

Ausführen

z.B. Daten zu Mitarbeiter 1000

Parameter setzen:

```
//Syntax set<Datentyp> (Parameternummer, Parameter)
pstmt.setInt(1,1000);
```

Ausführen:

```
ResultSet rs = pstmt.executeQuery();
```

Ressourcen freigeben

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

Für PreparedStatements ist die Wiederverwendung des QEP möglich

(QEP = Query Execution Plan)

Was bedeutet Wiederverwendung eines QEP?

Serverseitige Verarbeitung von SQL-Anweisungen umfaßt

- **Parsen**
- **Query Rewrite (nötig, wenn eine View involviert ist, ggf. sinnvoll, wenn intern in eine bessere Formulierung umgeformt wird)**
- **Optimierung**
- **Ausführung**

Bei der wiederholten Ausführung von PreparedStatements entfallen Parsen, Query Rewrite und Optimierung, falls der QEP wiederverwendet wird.

Wie stellt das DBS fest, ob ein QEP wiederverwendet werden kann?

Oracle verwendet einen Hash für die Map Statement Text => QEP

Hash (Statement Text 1)	**QEP 1**
Hash (Statement Text 2)	**QEP 2**

Es gibt auch weniger effiziente Verfahren!

ResultSet

JDBC 1: ResultSets sind forward only und read only

Typischer Zugriff:

```
Connection conn = ....;
Statement stmt = conn.createStatement();
resultSet rs = stmt.executeQuery
  (“select empno, ename from emp where…”);
while(rs.next())
{
        // datentypverträglicher Zugriff über Spaltenposition,
        // beginnend mit 1!
        System.out.print(rs.getInt(1);
        …
        // Zugriff über Spaltenname
        System.out.println(rs.getString(“ename”);
}
rs.close();
stmt.close();
conn.close();
```

ResultSetMetaData

liefern die Struktur einer Ergebnismenge

```
ResultSetMetaData rsmd = rs.getMetaData();
```

Interessante Metadaten:

- Anzahl der Spalten (getColumnCount)
- Pro Spalte :
  - Name
  - Datentypinfo
  - ...

Result Sets in JDBC 2 und JDBC 3

Navigation:

forward only (Vorwärtsbewegung über die Methode next())

scrollable (direkter Zugriff):

insensitiv

sensitiv (konkurrierende Änderungen werden gesehen)

Concurrency:

read only

updatable

(update, delete, insert via ResultSet Interface möglich)

Hold ResultSet

wird nicht bei Transaktionsende automatisch geschlossen

JDBC:

createStatement

(Navigation, Concurrency, Hold Spezifikation)

prepareStatement

(Statement, Navigation, Concurrency, Hold Spezifikation)

ResultSets und optimistisches / pessimistisches Locking für Read for Update Situationen

Optimistisches Locking

(keine Lesesperren, Konfliktbehandlung in der Anwendung)

Read Only ResultSet

keine Lesesperren bzw. Sofortige Freigabe nach dem Lesen (automatisch für Oracle)

Pessimistisches Locking

(Serialisierung der lesenden Zugriffe durch Update Locks)

Updatable ResultSet Voraussetzung

genügt bei Oracle nicht, das Select muß die ROWID in der Select Liste umfassen und FOR UPDATE deklariert sein:

select ROWID, ... FOR UPDATE

...

Batch Update

Ein Statement Objekt kann assoziiert werden mit mehreren

SQL-Anweisungen, die einen RowCount liefern

(insert, update, delete)

Hinzufügen einer SQL-Anweisung:

stmt.addBatch(...)

Ausführung aller SQL-Anweisungen zu dem Statement Objekt:

int RowCount [] = stmt.executeBatch()

Löschen der Liste der SQL-Anweisungen:

stmt.clearBatch(...)

Vorteil

Reduktion der Kommunikation Client-Server

Ohne Batch Update pro SQL-Statement eine Kommunikation

Mit Batch Update eine Kommunikation für mehrere SQL-Anweisungen

Hinweis

Prepared Statement

reduziert Aufwand der Planerstellung

Batch Update

reduziert die Kommunikation

Man kann bzw. muß (Oracle im JDBC Modus) beide Konzepte kombinieren!

Weitere Aspekte zur Performancesteigerung durch Reduktion der Kommunikation:

Kommunikationspakete für lesende Zugriffe

stmt.setFetchSize(*Zahl der Zeilen*)

Reduktion der Zahl der Datenbankzugriffe

Weniger Selects, die jeweils mehr Ergebniszeilen liefern

Verwendung von Stored Procedures und Triggern zur Verlagerung von Verarbeitung

Nicht behandelte JDBC Konzepte

DatabaseMetaData

Large Objects

Stored Procedures

Connection / Statement Pooling

Distributed Transactions

Rowsets