Name:	
Klasse:	
Datum:	

3 Was man sonst noch brauchen kann...

JDBC bietet noch weitere Möglichkeiten.

3.1 Abfragen mit RowSet

Für Abfragen wurde bisher das `Resultset` verwendet. Dieses „blättert“ durch die Abfrage, speichert aber nicht den gesamten Inhalt in einem Container, sondern holt sich immer bei Bedarf die Zeilen über den Datenbanktreiber von der Datenbank.

Das `RowSet` (Interface) bindet das Interface `Resultset` ein, verhält sich diesbezüglich gleich, speichert aber die Daten in einem Container.

Dabei gibt es verschiedene weitere Interfaces mit entsprechenden Implementationen:

`JDBCRowSet`	dienst dazu, dass das `ResultSet` als JavaBean verwendet werden kann. Dabei muss immer eine Verbindung zur Datenbank bestehen. ☞
`CachedRowSet`	baut am Anfang eine Verbindung zur Datenbank auf und lädt die Daten gemäß der SQL-Anweisung. Danach ist keine Verbindung zur Datenbank nötig. Werden Daten geändert, so können diese später zurück gespielt werden. ☞
`WebRowSet`	erweitert `CachedRowSet`, wobei die Daten und Operationen in XML abgebildet werden, beispielsweise für Webservice. ☞
`FilteredRowSet`	erweitert `WebRowSet`, wobei Selektionen über Filter verwendet werden können. ☞
`JoinRowSet`	ist ein spezielles `WebRowSet` und verbindet (join) die Ergebnisse von verschiedenen `RowSets`. ☞

Über die Factory-Methode `newFactory()` der Klasse `RowSetProvider` lassen sich sehr einfach entsprechende Implementierungen erzeugen.

```
CachedRowSet crs = RowSetProvider.newFactory().createCachedRowSet();
...
```

Notiz:

3.1.1 CachedRowSet

Im nachfolgenden Beispiel wird ein `CachedRowSet` erzeugt. Über die Methode `setType()` wird die Einstellung wie bei der Klasse `Statement` vorgenommen. Mit `setCommand()` der SQL-Befehl definiert (auch Prepared Statements sind hier möglich). Ein kleiner Unterschied dabei ist, dass die Verbindung erst mit der Methode `execute()` übergeben wird. Danach kann das `CachedRowSet` wie ein normales `ResultSet` verwendet werden, aber mit zusätzlichen Funktionen.

```
33      try {
34         final Connection con = Util.getConnection("gm3");
35         System.out.println("... connected");
36
37         final CachedRowSet crs = RowSetProvider.newFactory()
38               .createCachedRowSet();
39         crs.setType(ResultSet.TYPE_SCROLL_INSENSITIVE);
40         crs.setCommand("SELECT id, name FROM mitarbeiter");
41         crs.execute(con);
42
43         crs.addRowSetListener(new ExampleListener());
44         while (crs.next()) {
45            System.out.print("id=" + crs.getString(1));
46            System.out.println(" name=" + crs.getString(2));
47         }
48
49         Util.close(con);
50      } catch (final Exception e) {
51         System.out.println("Fehler: " + e.getMessage());
52      }
```

Eine Erweiterung ist beispielsweise, das man einen Listener einbinden kann, der bei jeder Zeilenänderung aktiv wird und so auf die Änderungen reagieren kann.

```
13   private static class ExampleListener implements RowSetListener {
14
15      @Override
16      public void cursorMoved(final RowSetEvent event) {
17         System.out.println("# cursorMoved event");
18      }
19
20      @Override
21      public void rowChanged(final RowSetEvent event) {
22         System.out.println("# rowChanged event");
23      }
24
25      @Override
26      public void rowSetChanged(final RowSetEvent event) {
27         System.out.println("# rowSetChanged event");
28      }
29   }
```

Ausgabe:

```
... connected
# cursorMoved event
id=1 name=Lorenz
# cursorMoved event
id=2 name=Ritter
# cursorMoved event
id=3 name=Wolff
# cursorMoved event
id=4 name=Richter
# cursorMoved event
...
```

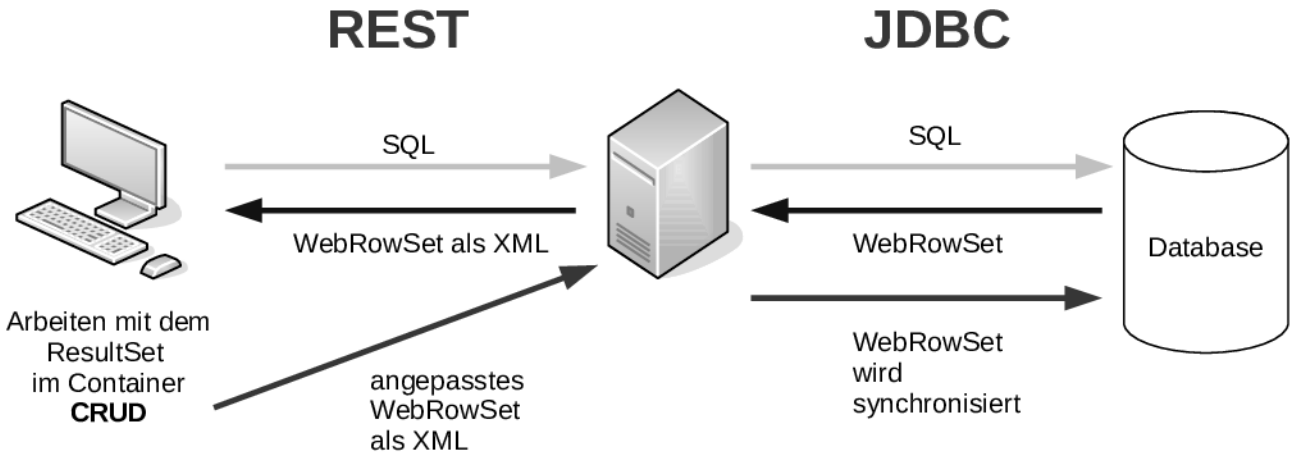

3.1.2 WebRowSet

Das `WebRowSet` bietet zusätzlich die Methoden `readXml()`, um Daten aus einer XML-Datei zu lesen und `writeXml()`, um Daten zu schreiben. Anstelle einer Datei können die Daten auch über eine Socket bei einer Client-Server-Kommunikation verwendet werden.

```
18          final WebRowSet wrs = RowSetProvider.newFactory()
19                .createWebRowSet();
20          wrs.setType(ResultSet.TYPE_SCROLL_INSENSITIVE);
21          wrs.setCommand("SELECT * FROM mitarbeiter");
22          wrs.execute(con);
23
24          wrs.writeXml(new FileOutputStream("target/rowset.xml"));
25          System.out.println("Datei erzeugt");
```

Anwendungsbeispiel

Somit ist es beispielweise möglich, eine SQL Abfrage per REST-Komminikation an einen Server zu senden, dieser schickt den SQL-Befehl per JDBC an das DBMS und sendet das WebRowSet per XML zurück an den Client. Somit kann der Client mit einem ResultSet arbeiten, auch wenn er keine direkte JDBC-Verbindung zur Datenbank hat.



XML-Datei:

```
<?xml version="1.0"?>
<webRowSet xmlns="http://java.sun.com/xml/ns/jdbc" xmlns:xsi="http://www.w3.
org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/jdbc http://java.sun.com/xml/ns
/jdbc/webrowset.xsd">
  <properties>
    <command>SELECT * FROM mitarbeiter</command>
    <concurrency>1008</concurrency>
    <datasource><null/></datasource>
    <escape-processing>true</escape-processing>
    <fetch-direction>1000</fetch-direction>
    <fetch-size>0</fetch-size>
    <isolation-level>2</isolation-level>
    <key-columns>
    </key-columns>
    <map>
    </map>
    <max-field-size>0</max-field-size>
    <max-rows>0</max-rows>
    <query-timeout>0</query-timeout>
    <read-only>true</read-only>
    <rowset-type>ResultSet.TYPE_SCROLL_INSENSITIVE</rowset-type>
    <show-deleted>false</show-deleted>
    <table-name>mitarbeiter</table-name>
    <url><null/></url>
```

```
    <sync-provider>
      <sync-provider-name>com.sun.rowset.providers.RIOptimisticProvider</sync-
      provider-name>
      <sync-provider-vendor>Oracle Corporation</sync-provider-vendor>
      <sync-provider-version>1.0</sync-provider-version>
      <sync-provider-grade>2</sync-provider-grade>
      <data-source-lock>1</data-source-lock>
    </sync-provider>
  </properties>
  <metadata>
    <column-count>9</column-count>
    <column-definition>
      <column-index>1</column-index>
      <auto-increment>true</auto-increment>
      <case-sensitive>true</case-sensitive>
      <currency>false</currency>
      <nullable>0</nullable>
      <signed>true</signed>
      <searchable>true</searchable>
      <column-display-size>11</column-display-size>
      <column-label>id</column-label>
      <column-name>id</column-name>
      <schema-name></schema-name>
      <column-precision>11</column-precision>
      <column-scale>0</column-scale>
      <table-name>mitarbeiter</table-name>
      <catalog-name>gm3</catalog-name>
      <column-type>4</column-type>
      <column-type-name>INTEGER</column-type-name>
    </column-definition>
...
  </metadata>
  <data>
    <currentRow>
      <columnValue>1</columnValue>
      <columnValue>Lorenz</columnValue>
      <columnValue>Sophia</columnValue>
      <columnValue>169945200000</columnValue>
      <columnValue>Hammer Str. 349</columnValue>
      <columnValue>7940</columnValue>
      <columnValue>1</columnValue>
      <columnValue>1</columnValue>
      <columnValue>946681200000</columnValue>
    </currentRow>
    <currentRow>
      <columnValue>2</columnValue>
      <columnValue>Ritter</columnValue>
      <columnValue>Tatjana</columnValue>
      <columnValue>279673200000</columnValue>
      <columnValue>Austermannstraße 75</columnValue>
      <columnValue>7945</columnValue>
      <columnValue>2</columnValue>
      <columnValue>5</columnValue>
      <columnValue>1078095600000</columnValue>
    </currentRow>
    <currentRow>
      <columnValue>3</columnValue>
      <columnValue>Wolff</columnValue>
      <columnValue>Theodor</columnValue>
      <columnValue>474073200000</columnValue>
      <columnValue>Telgenkamp 13</columnValue>
      <columnValue>4148</columnValue>
      <columnValue>3</columnValue>
      <columnValue>6</columnValue>
      <columnValue>1018821600000</columnValue>
```


```
    </currentRow>
...
    </data>
</webRowSet>
```

Notiz:

Aufgabe JDBC-03-1

1. Erstellen Sie ein Java-Programm, welches ein `WebRowSet` erzeugt (mit `SELECT * FROM mitarbeiter` aus DB `gm3`).
Speichern Sie das `WebRowSet` mit der Methode `writeXml()` in einer XML-Datei ab.
Zeigen Sie sodann den Inhalt mit der Methode `Util.printRs()` an.
Beispiel:

```
... connected
Datei erzeugt: rowset.xml
+---+---------------+--------------+----------+--------------------+-----+---+---+-----------+
|id |name           |vorname       |gebdat    |strasse             |oid  |aid|fid|eingestellt|
+---+---------------+--------------+----------+--------------------+-----+---+---+-----------+
|1  |Lorenz         |Sophia        |1975-05-22|Hammer Str. 349     |7940 |1  |1  |2000-01-01 |
|2  |Ritter         |Tatjana       |1978-11-12|Austermannstraße 75 |7945 |2  |5  |2004-03-01 |
|3  |Wolff          |Theodor       |1985-01-09|Telgenkamp 13       |4148 |3  |6  |2002-04-15 |
...
```

2. Erstellen Sie ein Java-Programm, welches ein `WebRowSet` erzeugt (ohne eine Verbindung zur Datenbank!).
Laden Sie mit `readXml()` den Inhalt aus der XML-Datei.
Zeigen Sie sodann den Inhalt mit der Methode `Util.printRs()` an.
Beispiel:

```
Datei gelesen: rowset.xml
+---+---------------+--------------+----------+--------------------+-----+---+---+-----------+
|id |name           |vorname       |gebdat    |strasse             |oid  |aid|fid|eingestellt|
+---+---------------+--------------+----------+--------------------+-----+---+---+-----------+
|1  |Lorenz         |Sophia        |1975-05-22|Hammer Str. 349     |7940 |1  |1  |2000-01-01 |
|2  |Ritter         |Tatjana       |1978-11-12|Austermannstraße 75 |7945 |2  |5  |2004-03-01 |
|3  |Wolff          |Theodor       |1985-01-09|Telgenkamp 13       |4148 |3  |6  |2002-04-15 |
...
```


3. Erstellen Sie ein Java-Programm, welches ein `WebRowSet` erzeugt (ohne eine Verbindung zur Datenbank!).
Laden Sie mit `readXml()` den Inhalt aus der XML-Datei.
Zeigen Sie sodann den Inhalt mit der Methode `Util.printRs()` an.
Nun fügen Sie einen Eintrag dem `WebRowSet` hinzu (siehe Einführungskapitel JDBC, `ResultSet` Daten hinzufügen) mit den Werten (ID=null, name=Kobold, vorname=Pumukel, ...).
Zeigen Sie sodann den geänderten Inhalt mit der Methode `Util.printRs()` an und speichern Sie das geänderte `WebRowSet` wieder in der XML-Datei.
Bei der Ausgabe fällt auf, dass „Pumukel“ noch keine ID hat, diese wird erst beim synchronisieren mit der Datenbank gesetzt.
Beispiel:

```
Datei gelesen: rowset.xml
+---+---------------+--------------+----------+--------------------+-----+---+---+-----------+
|id |name           |vorname       |gebdat    |strasse             |oid  |aid|fid|eingestellt|
+---+---------------+--------------+----------+--------------------+-----+---+---+-----------+
|1  |Lorenz         |Sophia        |1975-05-22|Hammer Str. 349     |7940 |1  |1  |2000-01-01 |
|2  |Ritter         |Tatjana       |1978-11-12|Austermannstraße 75 |7945 |2  |5  |2004-03-01 |
|3  |Wolff          |Theodor       |1985-01-09|Telgenkamp 13       |4148 |3  |6  |2002-04-15 |
...
|41 |Maier          |Peter         |1994-11-15|Bgm Huber Straße 2  |5277 |3  |1  |           |
+---+---------------+--------------+----------+--------------------+-----+---+---+-----------+


+---+---------------+--------------+----------+--------------------+-----+---+---+-----------+
|id |name           |vorname       |gebdat    |strasse             |oid  |aid|fid|eingestellt|
+---+---------------+--------------+----------+--------------------+-----+---+---+-----------+
|1  |Lorenz         |Sophia        |1975-05-22|Hammer Str. 349     |7940 |1  |1  |2000-01-01 |
|2  |Ritter         |Tatjana       |1978-11-12|Austermannstraße 75 |7945 |2  |5  |2004-03-01 |
|3  |Wolff          |Theodor       |1985-01-09|Telgenkamp 13       |4148 |3  |6  |2002-04-15 |
...
|41 |Maier          |Peter         |1994-11-15|Bgm Huber Straße 2  |5277 |3  |1  |           |
|   |Kobold         |Pumukel       |          |                    |     |   |   |           |
+---+---------------+--------------+----------+--------------------+-----+---+---+-----------+
Datei erzeugt: rowset.xml
```

4. Erstellen Sie ein Java-Programm, welches ein `WebRowSet` erzeugt (ohne eine Verbindung zur Datenbank!).
Laden Sie mit `readXml()` den Inhalt aus der XML-Datei.
Zeigen Sie sodann den Inhalt mit der Methode `Util.printRs()` an.
Übertragen Sie den Inhalt des `WebRowSet` mit der Methode `acceptChanges(con)` (als Parameter wird die DB-Connection angegeben) an die Datenbank.
Führen Sie sodann in der Datenbank eine SQL-Abfrage durch, die den neuen Datensatz anzeigt.
Beispiel:

```
Datei gelesen: rowset.xml
+---+---------------+--------------+----------+--------------------+-----+---+---+-----------+
|id |name           |vorname       |gebdat    |strasse             |oid  |aid|fid|eingestellt|
+---+---------------+--------------+----------+--------------------+-----+---+---+-----------+
|1  |Lorenz         |Sophia        |1975-05-22|Hammer Str. 349     |7940 |1  |1  |2000-01-01 |
|2  |Ritter         |Tatjana       |1978-11-12|Austermannstraße 75 |7945 |2  |5  |2004-03-01 |
|3  |Wolff          |Theodor       |1985-01-09|Telgenkamp 13       |4148 |3  |6  |2002-04-15 |
...
|41 |Maier          |Peter         |1994-11-15|Bgm Huber Straße 2  |5277 |3  |1  |           |
|   |Kobold         |Pumukel       |          |                    |     |   |   |           |
+---+---------------+--------------+----------+--------------------+-----+---+---+-----------+
in DB gespeichert...
```

DB:

```
MariaDB [gm3]> select * from mitarbeiter;
+-----+---------------+--------------+------------+---------------------+-------+------+------+-------------+
| id  | name          | vorname      | gebdat     | strasse             | oid   | aid  | fid  | eingestellt |
+-----+---------------+--------------+------------+---------------------+-------+------+------+-------------+
|   1 | Lorenz        | Sophia       | 1975-05-22 | Hammer Str. 349     |  7940 |    1 |    1 | 2000-01-01  |
|   2 | Ritter        | Tatjana      | 1978-11-12 | Austermannstraße 75 |  7945 |    2 |    5 | 2004-03-01  |
|   3 | Wolff         | Theodor      | 1985-01-09 | Telgenkamp 13       |  4148 |    3 |    6 | 2002-04-15  |
...
|  41 | Maier         | Peter        | 1994-11-15 | Bgm Huber Straße 2  |  5277 |    3 |    1 | NULL        |
|  42 | Kobold        | Pumukel      | NULL       | NULL                |  NULL | NULL | NULL | NULL        |
+-----+---------------+--------------+------------+---------------------+-------+------+------+-------------+
42 rows in set (0,001 sec)
```

3.2 SQL-Fehler auswerten

Bisher wurde ein geworfener SQL-Fehler einfach nur ausgegeben. Im nachfolgenden Beispiel wird ein Fehler geworfen, da die Tabelle nicht existiert.

```
11      try {
12         final Connection con = Util.getConnection("gm3");
13         System.out.println("... connected");
14
15         final Statement st = con.createStatement();
16         final ResultSet rs = st.executeQuery("SELECT * FROM GIBTESNICHT");
17
18         Util.close(con);
19      } catch (final Exception e) {
20         System.out.println("Fehler: " + e.getMessage());
21      }
```

Ausgabe

```
... connected
Fehler: (conn=3) Table 'gm3.gibtesnicht' doesn't exist
```

Die Klasse `SQLException` liefert folgende Informationen:

`getMessage()`	Die Beschreibung des Fehlers als Text.
`getSQLState()`	Ein fünfstelliger Code (genormt von ISO/ANSI und Open Group (X/Open)), der den Fehler beschreibt (fast alle DB-Hersteller halten sich an diesen Code).
`getErrorCode()`	Die Beschreibung des Fehlers als Zahl.
`getCause()`	Einen Grund für den Fehler. Dieser kann rekursiv mehrere `Throwable`-Objekte liefern.
`getNextException()`	Eine Referenz auf den nächsten verschachtelten Fehler.

Im JDBC-Tutorial findet man folgende Hilfsmethoden (hier etwas angepasst) für die Ausgabe des Fehlers.

```
5  public class SQLTool {
6
7     public static boolean ignoreSQLException(final String sqlState) {
8
9        if (sqlState == null) {
10          System.out.println("The SQL state is not defined!");
11          return false;
12       }
13
14       // X0Y32: Jar file already exists in schema
15       if (sqlState.equalsIgnoreCase("X0Y32")) {
16          return true;
17       }
18
19       // 42Y55: Table already exists in schema
20       if (sqlState.equalsIgnoreCase("42Y55")) {
21          return true;
22       }
23
24       return false;
25    }
26
27    public static void printSQLException(final SQLException ex) {
28
29       for (final Throwable e : ex) {
30          if (e instanceof SQLException) {
31             if (!ignoreSQLException(((SQLException) e).getSQLState())) {
```


```
32
33                  // e.printStackTrace(System.err);
34                  System.err.println("SQLState  : " + ((SQLException) e).
                    getSQLState());
35                  System.err.println("Error Code: " + ((SQLException) e).
                    getErrorCode());
36                  System.err.println("Message   : " + e.getMessage());
37
38                  Throwable t = ex.getCause();
39                  while (t != null) {
40                     System.out.println("Cause     : " + t);
41                     t = t.getCause();
42                  }
43               }
44            }
45         }
46      }
47   }
```

Im Beispiel kann dies dann wie folgt angewendet werden.

```
12       try {
13          final Connection con = Util.getConnection("gm3");
14          System.out.println("... connected");
15
16          final Statement st = con.createStatement();
17          final ResultSet rs = st.executeQuery("SELECT * FROM GIBTESNICHT");
18
19          Util.close(con);
20       } catch (final SQLException e) {
21          SQLTool.printSQLException(e);
22       }
```

Ausgabe

```
... connected
SQLState  : 42S02
Error Code: 1146
Message   : (conn=4) Table 'gm3.gibtesnicht' doesn't exist
```

Notiz:

3.3 Stored Procedures

Im Abschnitt **SQL – was sont noch gebraucht wird** wird gezeigt, wie man in SQL Prozeduren und Funtionen erstellt.

Erstellt man in SQL Prozeduren, so muss man hier immer den `DELIMITER` ändern, was ich persöhnlich immer etwas nervig finde. In Java ist dies nicht notwendig und somit deutlich einfacher.

3.3.1 Stored Procedures erstellen

Im nachfolgenden Beispiel wird eine Prozedure erstellt, die einen neuen Lieferanten erstellt. Als Parameter werden hier Name und E-Mail angegeben.

```
14          final String sql = """
15                CREATE OR REPLACE PROCEDURE insertLieferant
16                        (IN n VARCHAR(50), IN em VARCHAR(50))
17                   BEGIN
18                      INSERT INTO LIEFERANT (name,email) VALUES (n,em);
19                   END
20                """;
21
22          final Statement st = con.createStatement();
23          st.execute(sql);
24
25          System.out.println("Prozedure erstellt...");
```

In MariaDB kann man dann prüfen, ob die Prozedure auch entspechend erstellt worden ist.

```
SHOW CREATE PROCEDURE insertLieferant\G
*************************** 1. row ***************************
Procedure: insertLieferant
sql_mode: IGNORE_SPACE,STRICT_TRANS_TABLES,ERROR_FOR_DIVISION_BY_ZERO,NO_AUTO_CREATE_USER,
NO_ENGINE_SUBSTITUTION
Create Procedure: CREATE DEFINER=`test`@`localhost`
PROCEDURE `insertLieferant` (IN n VARCHAR(50), IN em VARCHAR(50))
BEGIN
      INSERT INTO LIEFERANT (name,email) VALUES (n,em);
END
...
```

3.3.2 Stored Procedures aufrufen

Der Aufruf sieht dann wie folgt aus.

```
16          final String sql = "{call insertLieferant(?, ?)}";
17          final CallableStatement cst = con.prepareCall(sql);
18          cst.setString(1, "Besondersschlau Susi");
19          cst.setString(2, "Susi@nix.de");
20          System.out.println("Prozedure ausgeführt...");
21          cst.execute();
22          Util.close(cst);
23
24          final Statement st = con.createStatement();
25          final ResultSet rs = st.executeQuery("SELECT * FROM LIEFERANT");
26          Util.printRs(rs);
```

Ausgabe

```
... connected
Prozedure ausgeführt...
+--+----------------------------+----------------------+----------+---------------------+-----+
|id|name                        |strasse               |telefon   |email                |oid  |
+--+----------------------------+----------------------+----------+---------------------+-----+
|1 |Augustiner Brauerei GmbH    |Landsberger Straße 31-35|089519940 |info@augustiner.de   |7889 |
|2 |Red Bull AG                 |Osterwaldstraße 10    |0891234567|bestellung@redbull.com|7908 |
|3 |Coca Cola Deutschland AG    |Kesslerweg 14         |08002223232|info@acocacola.de    |7939 |
|4 |Winzer Franke KG            |Blubajustrasse 35     |06713435667|Hans.Franke@gmail.com|7937 |
|5 |Adelholzener Alpenquellen GmbH|St.-Primus-Strasse 1-5|08662 620 |info@adelholzener.de |10735|
|6 |Besondersschlau Susi        |                      |          |Susi@nix.de          |     |
+--+----------------------------+----------------------+----------+---------------------+-----+
```


3.3.3 OUT-Parameter abfragen

Im nachfolgendem Beispiel wird der OUT-Paramter abgefragt. Alle Parameter, die nicht reine IN-Paramter sind, müssen entsprechend mit `registerOutParameter()` deklariert werden. Nach dem Aufruf der Prozedure kann der OUT-Parameter (hier `Integer`) über die Methode `cst.getInt("anzahl")` (bzw. `getString()`, ...) abgefragt werden.

```
16          // create procedure
17          String sql = """
18                CREATE OR REPLACE PROCEDURE countLieferantOrt
19                       (IN oname VARCHAR(50), OUT anzahl INT)
20                   BEGIN
21                       SELECT count(l.id) INTO anzahl
22                       FROM lieferant l, ort o
23                       WHERE o.id=l.oid AND o.name=oname;
24                   END
25                """;
26
27          final Statement st = con.createStatement();
28          st.execute(sql);
29          System.out.println("Prozedure erstellt...");
30
31          // call procedure
32          final String ort = "München";
33          sql = "{call countLieferantOrt(?, ?)}";
34          final CallableStatement cst = con.prepareCall(sql);
35          cst.registerOutParameter(2, Types.INTEGER);
36
37          // get OUT-parameter
38          cst.setString(1, ort);
39          System.out.println("Prozedure ausgeführt...");
40          cst.execute();
41          final Integer anzahl = cst.getInt("anzahl");
42          System.out.format("Es wurden %d Lieferanten in %s gefunden", anzahl, ort);
43
44          Util.close(cst);
45          Util.close(st);
46          Util.close(con);
```

Ausgabe

```
... connected
Prozedure erstellt...
Prozedure ausgeführt...
Es wurden 2 Lieferanten in München gefunden
```

Möchten Sie ein kompletes `ResultSet` abfragen, so hilft hier folgendes:

```
boolean hadResults = cst.execute();

if (hadResults) {
    ResultSet resultSet = statement.getResultSet();
   // ...
}
```

Aufgabe JDBC-03-2

Erstellen Sie die Beispiele eigenständig und probeiren diese aus.

3.4 Transaktionen

Transaktionen sorgen dafür, dass bei mehreren SQL-Befehlen, die in Folge abgearbeitet werden, die Datenintegrität gewährleistet wird. Führt ein Befehl zu einem Fehler, so werden alle SQL-Befehle, die in der Transaktionsgruppe sind, abgebrochen und es erfolgt keine Änderung in der Datenbank. Erst wenn alle Befehle einwandfrei ausgeführt worden sind, wird die Änderung in der Datenbank persitiert. Dadurch wird sichergestellt, dass die Daten auch bei Netzwerkproblemen, Softwarefehlern usw. konsistent bleiben.

Vorgehensweise

```
try {

    // Start der Transaction
    // Hierbei muss zwingend 'AutoCommit' ausgeschaltet werden.
    con.setAutoCommit(false);

    // execute SQL 1

    // execute SQL 2

    // execute SQL 3

    // Ende der Transaktion mit "commit"
    con.commit();

} catch (SQLException ex) {
    // Abbruch
    con.rollback();
}
// Transaktion wieder ausschalten
con.setAutoCommit(true);
```

savepoints

JDBC bietet die Methode `setSavepoint()` an, die einen Punkt markiert, zu dem die Transaktion zurückgesetzt werden kann. Die Methode `rollback(savepoint)` ermöglicht es, nur Änderungen nach dem Speicherpunkt rückgängig zu machen, falls etwas schief geht.

```
try {

    // Start der Transaction
    con.setAutoCommit(false);

    // execute SQL 1

    // execute SQL 2

   // Speicherpunkt
   Savepoint savepoint = con.setSavepoint();

    // execute SQL 3

    // execute SQL 4

     if (/* Fehler */) {
        // rollback zu 'savepoint'
        con.rollback(savepoint);
    }

    // execute SQL 5

    // Ende der Transaktion mit "commit"
    con.commit();
```


```
} catch (SQLException ex) {
    // Abbruch
    con.rollback();
}
// Transaktion wieder ausschalten
con.setAutoCommit(true);
```

Notiz: