In diesem Kursteil

- Einführung in JDBC
  - SQL
  - ODBC
  - Java und JDBC
  - JDBC 1.0
  - JDBC 2.0
- Ein Beispiel
  - Szenario
  - Kreieren einer Datenbank
  - Kreieren einer Tabelle
  - Einfügen von Daten in die Tabelle / DB
  - Abfragen der Datenbank
- Verbindung zur Datenbank herstellen
- Anweisungen, ResultSets und Datenbank
- Prepared Statements
- Java SQL Datentypen
- JDBC Exceptions
- Metadaten
- Stored Procedures
- Transaktionen
  - Commit
  - Rollback
  - Concurrency
- Batch Update
- Scrollable Result Sets
- Lobs

JDBC™ – J2EE Grundlagen und Praxis

.

1.1. Kursübersicht

In diesem Modul besprechen wir

- die Hintergründe, die zu JDBC führten.
- die grundlegenden Konzepte zum Thema Datenbanken und JDBC Applikationen.
- wie JDBC Klassen zusammenarbeiten, um mit Datenbanken zu kommunizieren.
- einige fortgeschrittene Konzepte zu JDBC und Datenbanken.
- wie man ausgewählte Möglichkeiten von JDBC 2.0 einsetzen kann.

1.1.1. Kursvoraussetzungen

Sie sollten vertraut sein mit den Konzepten der objektorientierten Programmierung im Allgemeinen und der Java Programmiersprache im Speziellen.

Die Übungen zu JDBC setzen voraus, dass Sie Java Programme verstehen, modifizieren und selber schreiben können. In der Regel benötigen Sie dazu mittlere Java Kenntnisse. So oder so finden Sie jeweils zu den Aufgaben eine Musterlösung, sozusagen als Leiter für eine eigene Lösung.

Von Vorteil wäre es, wenn Sie bereits Kenntnisse über Relationale Datenbank Managemenet Systeme hätten und die Structured Query Language (SQL) mindestens ansatzweise kennen würden. Im Abschnitt *SQL Primer* und *Ressourcen* des Appendix finden Sie weitere Informationen wie Links, Bücher und ähnliches.

1.1.1.1. Lernziele

Nach dem Durcharbeiten dieses Moduls sollten Sie in der Lage sein:

- Datenbank Tabellen zu kreieren und zu modifizieren.
- auf Informationen in einer Datenbank zugereifen können und die darin enthaltenen Daten modifizieren können.
- dynamisch Informationen über eine Datenbank und die darin enthaltenen Daten zu erhalten.
- die Fehlerbehandlung und Ausnahmen von JDBC und deren Einsatz zu kennen.
- prepared Statements einzusetzen.
- Transaktionen zu definieren und mehrere Operatione zu einer Transaktion zusammenzufassen.
- Batch Updates zu programmieren und Scrollable ResultSets einzusetzen.
- BLOBs (binary large objects) einzusetzen einzusetzen, um beispielsweise Bilder oder binäre Daten abzuspeichern.

1.1.1.2. Benötigte Software

Cloudscape™, eine 100% in Java™ geschriebene SQL DBMS, welche mit der Java™ 2 Platform, Enterprise Edition (J2EE™) von Sun heruntergeladen werden kann, kann für viele der in diesem Modul besprochenen Themen eingesetzt werden. Mit geringen Modifikationen können Sie alle Übungen genauso mit anderen JDBC konformen Datenbanken durchführen. Falls Sie weitere Informationen zu dieser Datenbank benötigen, finden Sie diese in den Appendices *Cloudscape Installation und Setup* und *Starten und Stoppen von Cloudscape*.

JDBC ist eine der wenigen Java Technologien , bei denen Sie auf externe Implementationen angewiesen sind. Daher werden Sie gelegentlich speziell kenennzeichnete Hinweise auf die Details der einen oder anderen Datenbank finden.

Cloudscape ist ein Markenname von Informix. DB2 ist ein Markenname von IBM.

1.1.2. Einführung in JDBC™

JDBC™ ist ein Java™ API (Application Programming Interface), eine Beschreibung eines Standard Frameworks für die Bearbeitung von tabellarischen und allgemeiner, präziser gesagt, relationalen Daten. JDBC 2.0 macht SQL dem Programmierer semi-transparent. SQL ist immer noch die *lingua franca* der Standard Datenbanken und stellt einen grossen Fortschritt dar, auf dem Weg der Trennung der Daten von den Programmen. Bevor wir mit dem eigentlichen Kurs anfangen, schauen wir noch einmal kurz zurück.

1.1.2.1. SQL

SQL ist eine standardisierte Sprache, um relationale Datenbanken zu kreieren, zu manipulieren, abzufragen und zu verwalten. In diesem Modul werden wir nicht sehr tief auf SQL eingehen. Einige Grundlagen werden aber wiederholt, so dass der Modul möglichst selbständig genutzt werden kann.

Folgende Begriffe sollten Sie mindestens grob kennen:

- eine *Datenbank* ist im Wesentlichen ein smarter Container für Tabellen.
- eine *Tabelle* ist ein Container, der aus Datensätzen, "Zeilen", besteht.
- eine *Zeile* ist (konzeptionell) ein Container bestehend aus Kolonnen.
- eine *Spalte* ist ein einzelnes Datenelement mit Namen, Datentyp und Wert.

Sie sollten sich mit diesen Begriffen vertraut machen und die Unterschiede kennen. Aber am Anfang reichen einfachste Kenntnisse: eine Datenbank entspricht grob einem Dateisystem; eine Tabelle entspricht einer Datei; eine Zeile (row) entspricht grob einem Datensatz; eine Spalte (column) endspricht einem Datenfeld oder einer Variable.

Im Zusammenhang mitdiesen Begriffen sollten Sie auch die *Input/Output (I/O) Operationen* von Java kennen.

Weil SQL eine anwendungsspezifische Sprache ist, kann eine einzelne SQL Anweisung zu komplexen Berechnungen und umfangreichen Datenmanipulationen führen, speziell beim Sortieren oder Einfügen neuer Daten. SQL wurde 1992 standardisiert. Damit besteht die prinzipielle Möglichkeit, Programme für mehrere Datenbank zu entwickeln und ohne Änderungen ausführen zu können. Um SQL einsetzen zu können, muss man aber mit einer SQL Datenbank verbunden sein und daher wird man in der Regel die jeweils speziellen Erweiterungen dieser Datenbank nutzen.

1.1.2.2. ODBC

ODBC (Open Database Connectivity) entspricht in etwa einem C-basierten Interfacezu SQL-basierten Datenbanken und stellt ein konsistentes Interface für die Kommunikation mit Datenbanken und den Zugriff auf deren *Metadaten* zur Verfügung. Die Metadaten beschreiben die Datenbank und deren Tabellen, Datentypen und beispielsweise Indices.

Datenbankhersteller stellen spezifische Treiber oder "Bridges" für verschiedene Datenbanken zur Verfügung. Damit kann man mit Hilfe von SQL und ODBC auf eine standardisierte Art und Weise auf Datenbanken zugreifen. ODBC wurde für PCs entwickelt, ist heute aber ein de facto Industriestandard.

Obschon SQL gut geeignetist, um Daten und Datenbanken zu manipulieren, ist SQL keine vollständige Programmiersprache. Die Sprache dient lediglich der Kommunikation mit der Datenbank. Sie benötigen also eine weitere, vollständige Programmiersprache, um die Verbindung mit der Datenbank aufzunehmen und die Ergebnisse aufzubereiten und eventuell zu visualisieren.

Falls Sie plattformunabhängig entwickeln wollen, ist Java eine gute Wahl, speziell für solche Datenbankanwendungen. Falls Sie lediglich eine Plattform benötigen, kann C++ genau so gut geeignet sein.

1.1.2.3. Die Java™ Programmier Sprache und JDBC

Ein Java Programm kann auf unterschiedlichen Plattformen ausgeführt werden. Das gestattet es Ihnen Datenbankprogramme zu schreiben, welche universell einsetzbar sind. Einzig das Package java.sql oder JDBC, auch als portable Version von ODBC angesehen, muss vorhanden sein.

Bemerkung 1

Obschon Sie portable Applikationen mit standardisierten Datenbankinterfaces schreiben können, müssen Sie beachten, dass nicht zuletzt aus Konkurrenzgründen, die Datenbanken selbst, inkompatibel bleiben, selbst auf der Stufe SQL! Sie müssen also versuchen, den kleinsten gemeinsamen Nenner der von Ihnen eingesetzten Datenbanken zu finden. Dieses Problem besteht unabhängig davon, ob Sie OBDC, JDBC und SQL mit Java oder prorietäre Protokolle verwenden.

Ein *JDBC Driver* ist eine Klasse, welche das JDBC Driver Interface implementiert. Zudem muss der Treiber die JDBC Aufrufe in datenbankspezifische Aufrufe umsetzen. Der Treiber muss also die gesamte Arbeit erledigen. Es sind mehrere Treiber in Java erhältlich, so dass Sie ausgehend davon eigene Treiber schreiben können.

Gemäss JDK existieren vier verschiedene Treiber Typen für JDBC. Diese sind in der aktuellen Beschreibung von JDBC enthalten. Viele Datenbankanbieter stellen heute JDBC Treiber für ihre Datenbanken zur Verfügung. Daneben existieren auch neutrale Treiber, die universell einsetzbar sind und in der Regel auf ODBC abbilden. Einige Hersteller haben sich auch auf die Entwicklung von Datenbanktreibern spezialisiert und bieten diese unabhängig von der Datenbank an.

Die vier JDBC Treibertypen sind:

1. *JDBC-ODBC bridge plus ODBC driver*
2. *Native-API partly-Java driver*
3. *JDBC-Net pure Java driver*
4. *Native-protocol pure Java driver.*

JDBC hat sich über die letzten Jahre signifikant weiter entwickelt. Die erste Version war bereits mit der ersten Version von JDK erhältlich. Die aktuellste Version finden Sie bei Sun: http://java.sun.com/products/jdbc/index.html

1.1.2.4. JDBC 1.0

Das JDBC 1.0 API stellte eine einfache Basis eines Frameworks zur Verfügung, um Daten abzufragen und SQL Anweisungen auszuführen. Dazu wurden Schnittstellen für folgende Funktionen zur Verfügung gestellt:

- Driver
- DriverManager
- Connection
- Statement
- PreparedStatement
- CallableStatement
- ResultSet
- DatabaseMetaData
- ResultSetMetaData
- Types

Wie Sie sehen werden, wird ein Treiber an den DriverManager übergeben. Dieser stellt dann eine Verbindung her, liefert also ein Connection Objekt. Falls Sie dann ein Statement, PreparedStatement oder CallableStatement erstellen, dann können Sie auf die Datenbank zugreifen und die Daten manipulieren.

Eine Abfrage liefert Ihnen Daten zurück, die Sie eventuell weiter aufbereiten müssen. Durch Zugriff auf die Metadaten und die ResultSet Metadaten erhalten Sie detailliertere Informationen über die Datenbank und die selektierten Daten Ihrer Abfrage.

1.1.2.5. JDBC 2.0

Die JDBC 2.0 API Dokumentation besteht aus zwei Teilen: dem *core* API, welches wir hier besprechen und das JDBC 2.0 Optional Package. Im Allgemeinen befasst sich das JDBC 2.0 core API primär mit Performance, Klassenerweiterungen und Funktionalitäten sowie SQL3 (SQL-99) Datentypen.

Neue Funktionalitäten im core API umfassen Scrollable Result Sets, Batch Updates, verbesserte Funktionalitäten zum einfügen, löschen und mutieren der Daten sowie der Internationalisierung, java.math.BigDecimal und verschiedene Zeitzonen.

- Das java.sql package ist das JDBC 2.0 core API. Es umfasst das ursprüngliche JDBC API, also JDBC 1.0 API, plus neue core APIs, welche später hinzukamen. Diese Teile sind Bestandteil des JDKs..
- Das *javax.sql* package ist das JDBC 2.0 Standard Extension API. Dieses Package ist völlig neu und als separater Download erhältlich oder als Teil von Java 2 Platform SDK, Enterprise Edition.

1.1.3. Ein vollständiges Beispiel

Als erstes Beispiel für den Einsatz von JDBC betrachten wir ein vollständiges Beispiel, inklusive Datenbank kreieren und dem Ersterfassen von Daten in Tabellen.Die grundlegenden Arbeiten beim Aufbau einer Datenbankanwendung sind:

- **Kreieren einer Datenbank**.
  eine Datenbank kann man in der Regel mit Tools kreieren, die der Datenbankanbieter mitliefert. Normalerweise besitzt eine Firma einen Datenbank Administrator, der dafür zuständig und berechtigt ist. Einige der JDBC Treiber unterstützen das Kreieren einer neuen Datenbank nicht. In der Regel müssen Sie dafür spezielle Definitionssprachen einsetzen, Data Definition Language (DDL). Diese Spezialitäten können wir nicht für jede erdenkliche Datenbank besprechen.

  Typischerweise existiert eine *CREATE DATABASE* Anweisung, wie Sie sie in vielen SQL Varianten auch finden. Aber diese Anweisungen sind Datenbank spezifisch, da die Datentypen je nach Datenbank unterschiedlich sind.

- **Verbindung zur Datenbank aufbauen**:
  Der JDBC Treiber muss als erstes eine Verbindung zur Datenbank aufbauen. Die grundlegende Information, die der Treiber an die Datenbank übermitteln muss ist die *Datenbank URL (Universal Resource Locator)*, eine *User ID* und ein *Password*. Je nach Treiber müssen auch noch weitere Argumente, Attribute oder Properties übermittelt werden.

  Hier die Links zur Dokumentation für zwei Beispiele:

  Cloudscape (Informix) Database Connection URL Attributes:
  http://www.cloudscape.com/docs/doc_36/doc/html/coredocs/develop3.htm#810326
  und
  IBM AS/400TM JDBC Properties.
  http://publib.boulder.ibm.com/pubs/html/as400/v4r5/ic2924/info/java/rzahh/jdbcprop.htm

  Wir werden konkrete Beispiele noch und noch sehen, so dass hier einfach ein Link angegeben wird.

- **Kreieren einer Tabelle**:
  Die Datenbank enthält Tabellen. In den Tabellen werden die Daten abgelegt, in Form von Zeilen und Spalten. Tabellen kreiert man mit Hilfe der DDL *CREATE TABLE* Anweisung. Diese Anweisung besitzt viele Optionen, welche vom Datenbankanbieter anhängen. Daher muss man auch hier die SQL Dokumentation der Anbieter anschauen.

- **Informationen in die Datenbank einfügen:**
  Daten kann man entweder mit einem Loader Programm oder mit SQL in eine Tabelle einfügen. Da wir uns mit JDBC beschäftigen wollen, werden wir die Daten mit SQL einfügen.

- **Selektives Abfragen der Informationen:**
  Für das Abfragen der Informationen in der Datenbank muss im Java Programm eine entsprechende SQL Abfrage eingegeben werden. JDBC liest die Informationen und speichert sie in den Variablen ab. Somit stehen sie dann zur weiteren Verarbeitung innerhalb des Programms zur Verfügung.

1.1.3.1. Beschreibung des Scenario

Als erstes müssen wir die Daten, die wir verwenden möchten irgendwie erfassen. In diesem Beispiel erfassen wir den Kaffee Konsum eines typischen Informatik Teams, vorallem Entwickler, bei denen Kaffee in den Adern fliesst. Die folgende Tabelle fasst den Kaffeeverbrauch während einer Woche pro Person und Tag zusammen.

Kaffee Konsum des XP Java Entwicklungsteams

"Koffein ist unser Haupternährungsmittel"

Eintrag	**Kunde**	**Wochentag**	**Anzahl Tassen**	**Typ**
1	Hans	Mo	1	JustJoe
2	JS	Mo	1	Cappuccino
3	Marie	Mo	2	CaffeMocha
4	Anne	Di	8	Cappuccino
5	Hubert	Di	2	MoJava
6	jDuke	Di	3	Cappuccino
7	Marie	Mi	4	Espresso
8	JS	Mi	4	Latte
9	Alex	Do	3	Cappuccino
10	James	Do	1	Cappuccino
11	jDuke	Do	4	JustJoe
12	**JS**	**Fr**	**9**	**Espresso**
13	Hans	Fr	3	Cappuccino
14	Berth	Fr	2	Cappuccino
15	jDuke	Fr	1	Latte

1.1.3.2. Kreieren der Datenbank

Wie bereits bemerkt, ist das Kreieren einer Datenbank DBMS (Datenbank Management System) spezifisch. Hier beschränken wir uns auf die Cloudscape Datenbank. An anderer Stelle finden Sie den Zugriff auf Excel, Access und Textdateien, die in ODBC definiert werden. Der Zugriff auf die Cloudscape Datenbank geschieht mittels eines URLs und setzen der relevanten Attribute (Benutzername, Passwort). Zudem muss man in Cloudscape ein Attribut 'create' auf true setzen: create=true. Der Name der Datenbank soll, wie oben erwähnt, XP_Team sein. Falls Sie die J2EE heruntergeladen haben, finden Sie bei den Unterlagen zur Enterprise Edition auch Hinweise zum Thema '*Cloudscape Installation and Setup'*. Das Installationsverzeichnis wird, falls Sie keine besonderen Verzeichnisse bei der Installation von J2EE gewählt haben, %J2EE_HOME%\Cloudscape. Falls Sie eine Datenbank kreieren wollen, die bereits existiert, erhalten Sie eine Fehlermeldung. Cloudscape kreiert aber auch in diesem Fall ein Connection Objekt.

Bemerkung 2

Beachten Sie, dass die im Folgenden verwendeten Methoden und Einstellungen datenbankabhängig sind. Falls Sie eine andere als die Cloudscape Datenbank verwenden, müssen Sie vermutlich auch andere Befehle verwenden. Falls Sie beispielsweise DB2/400 verwenden, wäre der erste Befehl STRSQL und anschliessend CREATE COLLECTION.

1.1.3.3. Verbindung zur Datenbank aufbauen

Die Verbindung zur Datenbank geschieht bei JDBC in zwei Schritten:

1. **Laden des JDBC Treibers:**
   Zuerst müssen Sie einen Treiber laden, mit dem die JDBC Klassen mit der Datenbank kommunizieren können.

   Im ersten Beispiel werden wir die Treiberklasse von Cloudscape, RmiJdbcDriver, verwenden:

   ```
   Class.forName( DriverClassName);
   ```

   Ein Standard *JDBC Compliant™* Treiber *sollte* auch eine Instanz der Treiberklasse kreieren. Dies trifft aber nicht immer zu. In unserem Beispiel verwenden wir stattdessen die Anweisung:

   ```
   Class.forName(DriverClassName).newInstance();
   ```

   Diese Anweisung wird ein zusätzliches Objekt kreieren. Falls Sie nun zwei Objekte haben, wird der Garbage Collector ein unreferenziertes Objekt über kurz oder lang aus dem System entfernen. Der Treibermanager registriert den Treiber nur einmal.

   Treiber kann man auch auf der Kommandozeile spezifizieren. Dies geschieht in diesem Fall mit der Systemeigenschaft jdbc.drivers. Diese Methode funktioniert aber nur, falls der Treiber, die Treiberklassen, im Klassenpfad enthalten sind.:

   ```
   java -Djdbc.drivers=DriverClassName AJavaApp
   ```

   Die angegebene Treiberklasse DriverClassName für diesen Modul ist im Falle von Cloudscape:

   ```
   COM.cloudscape.core.RmiJdbcDriver
   ```

2. **Verbindungsaufbau zu einer Datenquelle**.
   Der Treiber liefert Methoden, mittels derer eine Verbindung aufgebaut werden kann. Allerdings muss eine spezielle URL, eine JDBC Protokoll konforme, angegeben werden. Die allgemeine Form für so eine URL ist:

   jdbc:<subprotocol>:<subname>.

   In der JDK2SE Dokumentation finden Sie unter *Getting Started with the JDBC API* (im Internet unter http://java.sun.com/j2se/1.3/docs/guide/jdbc/getstart/GettingStartedTOC.fm.html) die Beschreibung dieser URLs: *URLs in General Use* und *JDBC URLs* .

Da Sie eine URL einsetzen, ist es naheliegend, aber in der Regel nicht mehr explizit erwähnt, dass Sie mit JDBC in einer verteilten Umgebung arbeiten.

In unserem Beispiel werden wir als erstes folgende URL verwenden:

jdbc:cloudscape:rmi:XP_Team;create=true

Mit Hilfe des DriverManagers können Sie eine Verbindung zur URL herstellen. Der DriverManager lädt den passenden Treiber, in unserem Fall also den Cloudscape Treiber.

In userer Umgebung sieht ein Verbindungsaufbau folgendermassen aus:

```
Connection con = DriverManager.getConnection(
  URL,
  Username,
  Password );
```

Diese Darstellung ist recht universell, selbst in den Fällen, in denen DerBenutzername und das Passwort leer bleiben, beispielsweise falls Sie auf Textdateien über ODBC zugreifen (in diesem Fall ist das Attribut Username bedeutungslos).

In unserem Cloudscape Beispiel würde zu diesem Zeitpunkt wegen unserem URL Attribut 'create=true' die Datenbank kreiert. Das Attribut würde bei erneutem Zugriff natürlich nicht mehr verwendet.

1.1.3.4. Kreien einer Tabelle

Mit dem Connection Objekt kommen wir aber nicht sehr weit. Die DDL (Data Manipulation Language) und die DML (Data Manipulation Language) benötigen für die SQL Ausführung ein Statement Objekt. Als nächstes müssen wir also ein Statement Objekt kreieren. Dieses steht aber mit der Datenbankverbindung, dem Connection Objekt, in Beziehung:

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

Damit wir Daten speichen können, müssen wir eine Tabelle in unserer Datenbank kreieren. Wir nennen unsere Tabelle *KaffeeListe*; sie befindet sich in der Datenbank *XP_Team*.

Die folgenden Anweisungen kreieren diese Tabelle mit den unterschiedlichen Datenfeldern, die wir zur Beschreibung unserer KaffeeListe benötigen.

Die SQL Schlüsselworte sind gross geschrieben, damit Sie sie besser erkennen können. Das Datenbankmanagementsystem verlangt dies jedoch nicht. Es gibt allerdings DBMS, welche zwischen Gross- und Kleinschreibung unterscheiden.

```
CREATE TABLE KaffeeListe (
   Eintrag              INTEGER      NOT NULL,
   Mitarbeiter          VARCHAR (20) NOT NULL,
   Wochentag            VARCHAR (2)  NOT NULL,
   Tassen               INTEGER      NOT NULL,
   Kaffeesorte          VARCHAR (10) NOT NULL,
   PRIMARY KEY( Eintrag )
)
```

In Java sieht diese Anweisung für Cloudscape folgendermassen aus:

```
  stmt.executeUpdate( "CREATE TABLE KaffeeListe ("  +
         "Eintrag       INTEGER      NOT NULL, "    +
         "Mitarbeiter   VARCHAR (20) NOT NULL, "    +
         "Wochentag     VARCHAR (2)  NOT NULL, "    +
         "Tassen        INTEGER      NOT NULL, "    +
         "Kaffeesorte   VARCHAR (10) NOT NULL,"     +
         "PRIMARY KEY( Entry )"                     +
       ")" );
```

Beachten Sie, dass die SQL Anweisung nicht durch ein Semikolon, einen Punkt oder was auch immer abgeschlossen wird. Unterschiedliche DBMS verlangen unterschiedliche Abschlusszeichen (Oracle in der Regel ein Semikolon am Ende einer SQL Anweisung).

Portabilität kann man nur erreichen, indem man das Abschlusszeichen weglässt, also überhaupt *kein* Trennzeichen verwendet. Die Aufgabe, ein Abschlusszeichen hinzuzufügen wird einfach an den Treiber delegiert.

In der obigen Definition der Tabelle sehen Sie auch, dass alle Spalten / Datenelemente nicht leer sein dürfen, NOT NULL. Damit vermeiden wir einige der typischen Probleme, die bei SQL Datenbanken auftreten können bzw. die Leute, die noch wenig mit Datenbanken zu tun hatten, eher irritieren.

Zudem haben wir einen eindeutigen Schlüssel, die laufende Nummer der Eintragung in der Liste, definiert.

1.1.3.5. Einfügen von Daten in eine Datenbank

Nachdem wir die Tabelle KaffeeListe in der Datenbank XP_Team kreiert haben, müssen wir noch Daten einfügen. Dies geschieht mit der SQL INSERT Anweisung:

```
INSERT INTO JJJJData VALUES ( 1, 'Hans',  'Mo', 1, 'JustJoe' )
INSERT INTO JJJJData VALUES ( 2, 'JS',    'Mo', 1, 'Cappuccino' )
INSERT INTO JJJJData VALUES ( 3, 'Marie', 'Mo', 2, 'CaffeMocha' )
...
```

Praktischer wäre es, die Daten in einem Array abzuspeichern oder aus einer Datei einzulesen:

```
"(1,  'Hans',   'Mo', 1, 'JustJoe')"
```

Damit würde unsere Programmanweisung oben, die SQL INSERT Anweisung, folgendermassen aussehen:

```
stmt.executeUpdate(
   "INSERT INTO KaffeeListe VALUES " + SQLData[i] );
```

1.1.3.6. Schritt für Schritt

Hier noch einmal kurz zusammengefasst, wie Sie vorzugehen haben:

1. laden des JDBC Treibers (in der Regel müssen Sie eine jar Datei im Klassenpfad aufführen).
2. kreieren einer URL mit dem jdbc Protokoll und allfälligen Attributen (bei Cloudscape kann gleichzeitig eine Datenbank kreiert werden, falls create=true angegeben wird).

   Jetzt kann das Programm eine Verbindung zur Datenbank aufbauen.

3. Das Connection Objekt liefert ein Statement Objekt.
4. Nun können SQL Anweisungen an die Datenbank gesandt werden (beispielsweise zum Kreieren oder Löschen einer Tabelle und der Datenmanipulation [INSERT, DELETE, UPDATE]).

Die erste Übung enthält den vollständigen Programmcode für eine kleine Anwendung, mit der die Datenbank XP_Team mit der Tabelle KaffeeListe kreiert und Daten eingefügt werden.

Selbsttestaufgabe 1Schauen Sie sich jetzt die Übung 1 an.

In dieser Übung lernen Sie an einem konkreten Beispiel und der Datenbank Cloudscape (als Teil der J2EE) wie

1. ein JDBC Datenbanktreiber geladen wird
2. die Verbindung zu einer Datenbank aufgebaut wird
3. ein Statement Objekt kreiert wird
4. die eigentliche Aufgabe des Programms (in Form von SQL Anweisungen) ausgeführt wird
5. das Statement Objekt geschlossen wird
6. die Verbindung zur Datenbank abgebaut wird.

Das Schema jedes JDBC Programms sieht immer etwa gleich aus.Diese erste Übung dient also als Beispiel und Sie selber brauchen nichts zu programmieren. Schauen Sie sich die Lösung genau an und versuchen Sie diese zu verstehen. In dem Text unten finden Sie Hinweise auf die Lösung.

Das Programm funktioniert nur, falls Sie eine Cloudscape Datenbank installiert haben, beispielsweise im Rahmen der J2EE Installation. Die Batch Datei verweist auf die Cloudescape Treiber. Das Programm ist etwas vereinfacht, weil die Treiber und die Datenbank- Informationen direkt im Programm festgelegt werden.

Aber falls Sie eine andere Datenbank verwenden möchten dürfte es Ihnen leicht fallen, diese Anpassungen zu machen.

1.1.3.7. Voraussetzungen

Keine

1.1.3.8. Arbeitsschritte

Importieren Sie das java.sql Package. Dies wird für das JDBC API benötigt.

Hier die Lösung:

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

Laden Sie den JDBC Treiber.

Hier die Lösung:

```
    String sDriver =
      "COM.cloudscape.core.RmiJdbcDriver";
    ...

    try   // Versuche den JDBC Treiber zu laden
    {     // mit newInstance
        Class.forName( sDriver ).newInstance();
    }
    catch( Exception e )  // Fehler
    {
      System.err.println(
        "Der Treiber konnte nicht geladen werden.");
      return;
    } // end catch
```

Wie oben erwähnt, sollte diese Form recht gut geeignet sein, universell einsetzbar zu sein. Falls Sie eine andere Datenbank einsetzen, sollten Sie bei der Dokumentation Ihrer Datenbank und speziell deren Treiber (ODBC oder JDBC) nachschauen, wie dies zu geschehen hat.

Kreieren Sie ein `Connection` *Objekt und ein dazugehörendes* `Statement` *Objekt mit* `jdbc:cloudscape:rmi:jGuru;create=true` *als* Datenbank URL *für Cloudscape*.

Hier die Lösung:

```
try {
      con = DriverManager.getConnection ( sURL, sUsername, sPassword);
      stmt = con.createStatement();
} catch ( Exception e) {
      System.err.println( "problems connecting to "+sURL + ":" );
      System.err.println( e.getMessage()
);

if(con != null) {
        try { con.close(); }
        catch( Exception e2 ) {}
}

   return;
} // end catch
```

Beachten Sie, dass im Programm die Verbindung zur Datenbank immer geschlossen werden sollte, sobald diese nicht mehr benötigt wird.

Falls bereits eine Kopie von der Tabelle KaffeeListe *in der Datenbank* XP_Team *vorhanden ist, dann löschen Sie diese zuerst.*

Hier die Lösung:

```
// damit das Programm mehr als einmal ausgeführt werden kann,
    // wird zuerst versucht, die (existierende) Tabelle zu löschen
    System.out.println("[CreateUndInsert]DROP TABLE KaffeeListe");
    try
    {
      stmt.executeUpdate( "DROP TABLE KaffeeListe" );
      System.out.println(
        "[CreateUndInsert]Tabelle KaffeeListe wurde entfernt.");
    }
    catch ( Exception e ) { /* Tabelle ist noch nicht vorhanden */ }
```

Diesen Aspekt der Datenbankanwendung haben wir im Text oben nicht besprochen. Aber falls Sie das Programm mehr als einmal ausführen wollen, müssen Sie vor dem Kreieren einer Tabelle sicher sein, dass diese in der Datenbank noch nicht vorhanden ist. Die `CREATE TABLE` Anweisung würde bei mehrmaliger Ausführung zum Programmabbruch führen.

Die `DROP TABLE` Anweisung löscht die betreffende Tabelle vollständig aus der Datenbank.

Kreieren Sie die KaffeeListe Tabelle und fügen Sie die Daten ein.

Hier die Lösung:

```
// SQL Anweisung ausführen
    // CREATE TABLE und INSERT
    System.out.println("[CreateUndInsert]CREATE TABLE KaffeeListe");
    try    {
      stmt.executeUpdate( "CREATE TABLE KaffeeListe (" +
             "Eintrag       INTEGER      NOT NULL, "   +
             "Mitarbeiter   VARCHAR (20) NOT NULL, "   +
             "Wochentag     VARCHAR (2)  NOT NULL, "   +
             "Tassen        INTEGER      NOT NULL, "   +
             "Kaffeesorte   VARCHAR (10) NOT NULL,"    +
             "PRIMARY KEY( Eintrag )"                  +
                                        ")" );
```

Mit der `CREATE TABLE` SQL Anweisung werden die Spalten und Datentypen sowie der Primärschlüssel definiert und die KaffeeListe Tabelle kreiert.

```
System.out.println(
        "[CreateUndInsert]Tabelle KaffeeListe wurde kreiert.");

      System.out.println("[CreateUndInsert]INSERT INTO KaffeListe VALUES");
      for (int i = 0; i < SQLData.length; i++)
      {
        iRowCount +=
        stmt.executeUpdate(
          "INSERT INTO KaffeeListe VALUES " +
           SQLData[i] );
      }

      System.out.println("[CreateUndInsert] "+ iRowCount +
         " Zeilen in die Tabelle KaffeeListe eingefuegt.");

    }
```

Die Daten selbst sind in einem Array SQLData definiert. Die Daten werden in der Zeile mit der *executeUpdate()* Anweisung in die Tabelle eingefügt.

Beachten Sie, dass das `Statement` Objekt für mehrere SQL Anweisungen verwendet werden kann. *executeUpdate()* liefert die Anzahl Zeilen, welche gefunden wurden. Damit kann man eine Schleife bauen und alle Daten herauslesen und eventuell Mittelwerte bilden.

```
catch ( Exception e )
    {
      System.err.println(
        "[CreateUndInsert]Problem beim Senden einer SQL Anweisung an " +
sURL + ":" );
      System.err.println( e.getMessage() );
    }
    finally
    {
      try { stmt.close(); }
      catch( Exception e ) {}

      try { con.close(); }
      catch( Exception e ) {}
    } // end finally clause

  } // end main

} // end class
```

Um sicherzustellen, dass `Statement` und `Connection` sauber geschlossen werden können, verwendet der try catch Block auch noch ein finally.

Und dies ist die Reihenfolge:
get `Connection`, create `Statement`, ... close `Statement`, close `Connection`.

1.1.3.9. Quellcode der Lösung

```
package createtableundinsert;
import java.sql.*;

/**
 * Title:       Create Und Insert
 * Description: kreiert eine Tabelle und fügt die Daten der Kaffeeliste ein
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */

public class CreateUndInsert {

  static String[] SQLData =   {
    "(1,  'Hans',   'Mo', 1, 'JustJoe')",
    "(2,  'JS',     'Mo', 1, 'Cappuccino')",
    "(3,  'Marie',  'Mo', 2, 'CaffeMocha')",
    "(4,  'Anne',   'Di', 8, 'Cappuccino')",
    "(5,  'Holley', 'Di', 2, 'MoJava')",
    "(6,  'jDuke',  'Di', 3, 'Cappuccino')",
    "(7,  'Marie',  'Mi', 4, 'Espresso')",
```

```
    "(8,  'JS',     'Mi', 4, 'Latte')",
    "(9,  'Alex',   'Do', 3, 'Cappuccino')",
    "(10, 'Peter',  'Do', 1, 'Cappuccino')",
    "(11, 'jDuke',  'Do', 4, 'JustJoe')",
    "(12, 'JS',     'Fr', 9, 'Espresso')",
    "(13, 'HAns',   'Fr', 3, 'Cappuccino')",
    "(14, 'Beth',   'Fr', 2, 'Cappuccino')",
    "(15, 'jDuke',  'Fr', 1, 'Latte')"
  };

  public static void main(String[] args) {
    System.out.println("[CreateUndInsert]Start");
    Connection con = null;
    int iRowCount = 0;
    Statement stmt = null;

    String sDriver =
      "COM.cloudscape.core.RmiJdbcDriver";

    // kreieren der DB mit create = true
    String sURL =
      "jdbc:cloudscape:rmi:XP_Team;create=true";
    String sUsername = "sa";
    String sPassword = "admin";

    System.out.println("[CreateUndInsert]laden des Treibers");
    try   // laden des JDBC Treibers
    {     // mit newInstance
       Class.forName( sDriver ).newInstance();
    }
    catch( Exception e )  // klappt so nicht
    {
      System.err.println(
       "[CreateUndInsert]Der Treiber konnte nicht geladen werden.");
      return;
    } // end catch

    System.out.println("[CreateUndInsert]Verbindungsaufbau");
    try {
      con = DriverManager.getConnection ( sURL,
                                  sUsername,
                                  sPassword);
      stmt = con.createStatement();
    }
    catch ( Exception e){
      System.err.println("[CreateUndInsert]Probleme beim Verbindungsaufbau mit " +
                         sURL + ":" );
      System.err.println( e.getMessage() );

      if( con != null)
      {
        try { con.close(); }
        catch( Exception e2 ) {}
      }

      return;
    } // end catch

    // damit das Programm mehr als einmal ausgeführt werden kann,
    // wird zuerst versucht, die (existierende) Tabelle zu löschen
    System.out.println("[CreateUndInsert]DROP TABLE KaffeeListe");
```

```
    try
    {
      stmt.executeUpdate( "DROP TABLE KaffeeListe" );
      System.out.println(
        "[CreateUndInsert]Tabelle KaffeeListe wurde entfernt.");
    }
    catch ( Exception e ) { /* Tabelle ist noch nicht vorhanden */ }


    // SQL Anweisung ausführen
    // CREATE TABLE und INSERT
    System.out.println("[CreateUndInsert]CREATE TABLE KaffeeListe");
    try    {
      stmt.executeUpdate( "CREATE TABLE KaffeeListe (" +
            "Eintrag       INTEGER      NOT NULL, "   +
            "Mitarbeiter   VARCHAR (20) NOT NULL, "   +
            "Wochentag     VARCHAR (2)  NOT NULL, "   +
            "Tassen        INTEGER      NOT NULL, "   +
            "Kaffeesorte   VARCHAR (10) NOT NULL,"    +
            "PRIMARY KEY( Eintrag )"                  +
                                    ")" );

      System.out.println(
        "[CreateUndInsert]Tabelle KaffeeListe wurde kreiert.");

      System.out.println("[CreateUndInsert]INSERT INTO KaffeListe VALUES");
      for (int i = 0; i < SQLData.length; i++)
      {
        iRowCount +=
        stmt.executeUpdate(
          "INSERT INTO KaffeeListe VALUES " +
           SQLData[i] );
      }

      System.out.println("[CreateUndInsert] "+ iRowCount +
         " Zeilen in die Tabelle KaffeeListe eingefuegt.");

    }
    catch ( Exception e )
    {
      System.err.println(
        "[CreateUndInsert]Problem beim Senden einer SQL Anweisung an " + sURL + ":" );
      System.err.println( e.getMessage() );
    }
    finally
    {
      try { stmt.close(); }
      catch( Exception e ) {}

      try { con.close(); }
      catch( Exception e ) {}
    } // end finally clause
  } // end main
} // end class
```

1.1.3.10. Demonstration

Falls Sie dieses Programm laufen lassen werden die Datenbank XP_Team und die Tabelle KaffeeListe kreiert und mit Daten bevölkert werden.

1.1.3.11. Daten aus der Datenbank lesen

Um Daten aus der Tabelle in der Datenbank zu lesen, müssen wir eine SELECT Anweisung an die Datenbank senden, mittels einer Statement.executeQuery() Anweisung. Diese liefert das Ergebnis zeilenweise aus dem ResultSet Objekt. Ein Standard ResultSet Objekt wird mit der Methode ResultSet.next() abgefragt, jeweils einfach die nächste Zeile. Mit getXXX() kann man auch individuelle Spaltendaten erhalten. Falls Sie beispielsweise die maximale Anzahl Kaffees pro Mitarbeiter bestimmen wollten, müssten wir die Daten temporär sortieren. Dies kann man aber mit SQL problemlos spezifizieren: ORDER BY sortiert die resultierenden Daten nach dem Ordnungsbegriff, der hinter dieser Anweisung steht.

Hier die SQL Anweisung für diesen Sortvorgang:

```
SELECT Eintrag, Mitarbeiter, Wochentag, Tassen, Kaffeesorte
  FROM KaffeeListe
  ORDER BY Tassen DESC
```

In einem JDBC Programm wird daraus:

```
ResultSet result = stmt.executeQuery(
"SELECT Eintrag, Mitarbeiter, Wochentag, Tassen, Kaffeesorte "+
"  FROM KaffeeListe"+
"  ORDER BY Tassen DESC");
```

1.1.3.12. Datennavigation

Die Methode ResultSet.next() liefert ein boolean: true, falls es eine nächste Zeile gibt, false sonst, also wenn man am Ende des ResultSets angelangt ist.

Intern wird ein Zeiger vor die erste Zeile im ResultSet gesetzt. Der Aufruf von next() verschiebt den Zeiger zur ersten Zeile, dann zur zweiten usw. Um die erste Zeile zu erhalten müssen wir abfragen, ob überhaupt eine Zeile vorhanden ist:

```
if( result.next() )
```

Nachher können wir mit einer Schleife die jeweils nächste Zeile aus dem ResultSet lesen:

```
while(result.next())
```

1.1.3.13. Datenextraktion

Nachdem wir den Zeiger positioniert haben, muss das Anwendungsprogramm die Daten übernehmen. Dies geschieht mit den getXXX() Methoden des ResultSet Objekts.

Hier ein Beispiel für unsere KaffeeListe:

```
iEintrag        = result.getInt("Eintrag");
Mitarbeiter     = result.getString("Mitarbeiter");
Wochentag       = result.getString("Wochentag");
Tassen          = result.getInt("Tassen");
TotalTassen     += Tassen;  // inkrement TotalTassen
KaffeeSorte     = result.getString("KaffeeSorte");
```

In unserem Demo Programm verwenden wir System.out als Standard Ausgabestrom: System.out.println().

Falls alles korrekt abläuft sollten Sie eine Ausgabe wie unten erhalten:

```
[KaffeeListenAuswertung]Start
[KaffeeListenAuswertung]Laden des JDBC Treibers
[KaffeeListenAuswertung]Verbindungsaufbau zur Datenbank
[KaffeeListenAuswertung]executeQuery("SELECT ..."
Am Fr konsumierte der XP_Team Mitarbeiter JS am meisten Kaffee.
      Anzahl Tassen: 9
      Kaffeesorte: Espresso.

4,    Anne, Di,   8,    Cappuccino
11,   jDuke,      Do,   4,    JustJoe
8,    JS,   Mi,   4,    Latte
7,    Marie,      Mi,   4,    Espresso
13,   Hans, Fr,   3,    Cappuccino
9,    Alex, Do,   3,    Cappuccino
6,    jDuke,      Di,   3,    Cappuccino
14,   Beth, Fr,   2,    Cappuccino
5,    Holley,     Di,   2,    MoJava
3,    Marie,      Mo,   2,    CaffeMocha
15,   jDuke,      Fr,   1,    Latte
10,   Peter,      Do,   1,    Cappuccino
2,    JS,   Mo,   1,    Cappuccino
1,    Hans, Mo,   1,    JustJoe

Das XP Team konsumierte diese Woche total : 48 Tassen Kaffee.
[KaffeeListenAuswertung]Ende
```

Beachten Sie, dass das ResultSet sortiert ist, nach Anzahl Tassen, die der Mitarbeiter pro Tag getrunken hat.

Auch dieses Beispiel finden Sie als Übung auf dem Server / der CD.

1.1.3.14. Übung

In dieser Übung lesen wir die Daten aus der Tabelle KaffeeListe in der Datenbank XP_Team und geben Sie einigermassen sortiert aus. Der Programmcode lehnt sich an den Aufbau aus dem ersten Beispiel an. Diese Übung dient als Einarbeit. Sie können also getrost direkt zur Musterlösung gehen und sich diese genau anschauen.

Auch dieses Beispiel verwendet die Cloudscape Datenbank aus der J2EE, wie im ersten Beispiel.

1.1.3.14.1. Voraussetzungen

- Sie sollten die erste Übung erfolgreich abgeschlossen haben, da wir auf die dort in die Tabelle eingefügten Daten zugreifen.

1.1.3.14.2. Aufgabe 1

Fragen Sie die Daten der Tabelle KaffeeListe (in der Datenbank XP_Team) ab, wobei Sie die SQL Anweisung SELECT verwenden. Die Daten sollen absteigend sortiert sein, mit der Anzahl Tassen als Sortierkriterium.

1.1.3.14.2.1. Lösungshinweise 1

```
ResultSet result = stmt.executeQuery(
        "SELECT Eintrag, Mitarbeiter, Wochentag, Tassen, Kaffeesorte " +
        "FROM KaffeeListe " +
        "ORDER BY Tassen DESC");
```

1.1.3.14.3. Aufgabe 2

Bestimmen Sie, ob überhaupt eine Zeile gefunden wurde.
Verwenden Sie die ResultSet.getXXX() Methoden um die Information auszugeben, wer an welchem Wochentag am meisten Kaffetassen von welcher Sorte getrunken hat, wie oben in der Musterausgabe.

1.1.3.14.3.1. Lösungshinweis 2

```
if( result.next() ) {   // erste Zeile
                        // falls Daten vorhanden sind
        sMitarbeiter = result.getString("Mitarbeiter");
        iTassen = result.getInt("Tassen");
        System.out.println(
          "Am " + result.getString("Wochentag") +
         " konsumierte der XP_Team Mitarbeiter " + sMitarbeiter +
         " am meisten Kaffee." +
         "\n\tAnzahl Tassen: " + iTassen +
         "\n\tKaffeesorte: " + result.getString("Kaffeesorte") +
         ".\n");

      iTotalTassen = iTassen;  // Total initialisieren
```

1.1.3.14.4. Aufgabe 3

Lesen Sie die weiteren Daten aus dem ResultSet. Verwenden Sie dazu die ResultSet.getXXX() Methode, um die Ergebnisse in Programmvariablen zu speichern.

1.1.3.14.4.1. Lösungshinweis 3

```
      while(result.next()) {  // Daten zeilenweise lesen
                iEintrag = result.getInt("Eintrag");
                sMitarbeiter = result.getString("Mitarbeiter");
                sWochentag = result.getString("Wochentag");
                iTassen = result.getInt("Tassen");
                iTotalTassen += iTassen;  // Total erhöhen
                sKaffeesorte = result.getString("Kaffeesorte");

                // Ausgabe der KaffeeListe
                System.out.println( iEintrag    + ",\t" +
                                    sMitarbeiter + ",\t" +
                                    sWochentag      + ",\t" +
                                    iTassen     + ",\t" +
                                    sKaffeesorte );
              }
```

1.1.3.14.5. Aufgabe 4

Geben Sie die Anzahl Tassen aus, welche insgesamt konsumiert wurden und schliessen Sie anschliessend die Verbindung zur Datenbank.

1.1.3.14.5.1. Lösungshinweis 4

```
// Total
        System.out.println(
          "\nDas XP Team konsumierte diese Woche total : " +
             iTotalTassen + " Tassen Kaffee.");

      } // end if( result.next() )

    } // end try
    catch (Exception e) { e.printStackTrace(); }
    finally
    {
      try { stmt.close(); }
      catch( Exception e ) {}

      try { con.close(); }
      catch( Exception e ) {}
    } // end finally clause

    System.out.println("[KaffeeListenAuswertung]Ende");
  } // end main

} // end class KaffeeListenAuswertung
```

Das ResultSet wird automatisch geschlossen, falls Sie das Statement schliessen. Dies geschieht auch dann, wenn eine neue Abfrage mit Statement.execute(...) ausgeführt wird. Trotzdem sollten Sie sich angewöhnen, jeweils alle nicht mehr benötigten Verbindungen zur Datenbank zu schliessen.

1.1.3.15. Musterlösung

```
package kaffeelisteauswerten;

import java.sql.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * Company:      Joller-Voss GmbH
 * @author J.M.Joller
 * @version 1.0
 */

public class KaffeeListenAuswertung
{
  public static void main (String args[])  {

    System.out.println("[KaffeeListenAuswertung]Start");
    Connection con = null;
    int iTassen, iTotalTassen, iEintrag;
    Statement stmt = null;

    String sDriver =
      "COM.cloudscape.core.RmiJdbcDriver";
    String sURL =
      "jdbc:cloudscape:rmi:XP_Team";
//      "jdbc:rmi:jdbc:cloudscape:XP_Team;create=true";
    String sUsername = "sa";
    String sPassword = "admin";

    String sMitarbeiter   = null,
           sWochentag     = null,
           sKaffeesorte    = null;
    System.out.println("[KaffeeListenAuswertung]Laden des JDBC Treibers");
    try  {  //Laden des JDBC Treibersr
            //mit newInstance
        Class.forName( sDriver ).newInstance();
    } catch( Exception e ) { // Fehler
      System.err.println(
        "[KaffeeListenAuswertung]Der JDBC Treiber konnte nicht geladen
werden.");
      return;
    } // end catch

    try   {
      System.out.println("[KaffeeListenAuswertung]Verbindungsaufbau zur
Datenbank");
      con = DriverManager.getConnection ( sURL,
                                          sUsername,
                                          sPassword);
      stmt = con.createStatement();
    }    catch ( Exception e)    {
      System.err.println( "[KaffeeListenAuswertung]Der Verbindungsaufbau zu
" +
                          sURL + " schlug fehl:" );
      System.err.println( e.getMessage() );

      if( con != null)      {
        try { con.close(); }
        catch( Exception e2 ) {}
      }

      return;
    } // end catch

    try    {
      System.out.println("[KaffeeListenAuswertung]executeQuery(\"SELECT
...\" ");
      ResultSet result = stmt.executeQuery(
```

```
        "SELECT Eintrag, Mitarbeiter, Wochentag, Tassen, Kaffeesorte " +
        "FROM KaffeeListe " +
        "ORDER BY Tassen DESC");

      if( result.next() ) { // erste Zeile
                            // falls Daten vorhanden sind
        sMitarbeiter = result.getString("Mitarbeiter");
        iTassen = result.getInt("Tassen");
        System.out.println(
          "Am " + result.getString("Wochentag") +
         " konsumierte der XP_Team Mitarbeiter " + sMitarbeiter +
         " am meisten Kaffee." +
         "\n\tAnzahl Tassen: " + iTassen +
         "\n\tKaffeesorte: " + result.getString("Kaffeesorte") +
         ".\n");

        iTotalTassen = iTassen;  // Total initialisieren

        System.out.println("[KaffeeListenAuswertung]while(result.next())
");
        while(result.next()) {  // Daten zeilenweise lesen
          iEintrag = result.getInt("Eintrag");
          sMitarbeiter = result.getString("Mitarbeiter");
          sWochentag = result.getString("Wochentag");
          iTassen = result.getInt("Tassen");
          iTotalTassen += iTassen;  // Total erhöhen
          sKaffeesorte = result.getString("Kaffeesorte");

          // Ausgabe der KaffeeListe
          System.out.println( iEintrag    + ",\t" +
                              sMitarbeiter + ",\t" +
                              sWochentag      + ",\t" +
                              iTassen     + ",\t" +
                              sKaffeesorte );
        }
        // Total
        System.out.println(
          "\nDas XP Team konsumierte diese Woche total : " +
             iTotalTassen + " Tassen Kaffee.");

      } // end if( result.next() )

    } // end try
    catch (Exception e) { e.printStackTrace(); }
    finally
    {
      try { stmt.close(); }
      catch( Exception e ) {}

      try { con.close(); }
      catch( Exception e ) {}
    } // end finally clause

    System.out.println("[KaffeeListenAuswertung]Ende");
  } // end main

} // end class KaffeeListenAuswertung
```

1.1.3.16. Demonstration

Falls Sie das Programm ausführen, sollten Sie eine Auswertung analog wie oben erhalten.

1.1.3.17. Zusammenfassung

Was haben wir in diesem Abschnitt gelernt?

1. **JDBC ist portabel.**
   In unseren Beispielen haben wir den Treiber und alle anderen Attribute fest einprogrammiert. Es ist aber keine grosse Sache, die Programme universell zu gestalten.
2. **Alle Programme laufen mit JDK 1.1 bis zu den aktuellsten Versionen**
   Obschon zum Erstellen aller Programme JDK 1.3 und JDK1.4 und JDBC 1.2 verwendet wurden, sind die Programme auch mit älteren Versionen lauffähig.

1.1.4. Verbindungsaufbau mit einer Datenbank

Ein Connection Objekt repräentiert und kontrolliert den Datenbankzugriff. Grundsätzlich kennen wir diese Objekte bereits aus dem vorhergehenden Abschnitt. In diesem Abschnitt wollen wir lediglich einige Begriffe klären und vertiefen und einige weitere Übungen durchspielen, diesmal Sie!

Im Allgemeinen hängt in JDBC alles von den Fähigkeiten des JDBC Treibers ab. Aber es ist jederzeit möglich, mehr als eine Verbindung zur Datenbank aufzubauen und / oder gleichzeitig zu einer und / oder mehreren Datenbanken. Der Treibermanager ist für die Treiberregistrierung verantwortlich und stellt Methoden zur Verfügung, mit deren Hilfe auf die Datenbanken zugegriffen werden kann, also ein Connection Objekt erhalten werden kann. Die Datenbanktreiber besitzen statische Methoden, so dass Sie also keine Instanzen bilden müssen.

Einer der ersten Schritte, um eine Verbindung zu erreichen, besteht in der Regel darin, die Datenbak URL zu bestimmen. Im Prinzip sieht diese sehr einfach aus:

jdbc:<subprotocol>:<subname>

wobei das <subprotocol> die Maschine oder den Server repräsentiert und <subname> im Wesentlichen die Datenbank identifiziert.

In der Praxis hängt die Datenbank-URL vom spezifischen Treiber ab und kann wesentlich anders aussehen. Schauen wir uns den Pfad für Cloudscape an:

```
jdbc:cloudscape:rmi:XP_Team
```

wobei wir also folgende Zuordnungen erreichen:

```
jdbc:     <subprotocol>:       <subname>
jdbc:      cloudscape:rmi:      XP_Team
```

In diesem Fall sieht das Ganze sehr einfach aus, vorallem weil sich alles auf einem Rechner abspielt. Falls Sie Treiber verwenden müssen, welche nicht in Java geschrieen sind, werden Sie unter Umständen CLASSPATH Angaben und Ähnliches machen müssen.

Da der Zugriff oft mittels TCP / IP erfolgt - auch im Falle von Cloudscape - werden Sie über bestimmte Ports kommunizieren. Im Falle von Cloudscape ist dies, wie Sie durch Starten von netstat (in einem DOS Fenster) sehen können, der Port 1099.

Aber jedes DMBS kann die Kommunikationsprotokolle selber wählen. Beispielsweise im Falle von SQLBase: NetBEUI, oder IPX / SPX oder TCP/IP; im Falle der DB/2 von IBM ist dies in der Regel APPC (Advanced Program to Program Communication).

Falls eine Applikation versucht auf einen Netzwerk oder Internet Server zuzugreifen, muss die entsprechende Lokationsinformation plus Benutzerkennung eingegeben werden. In JDBC geschieht dies mittels //host:port/subname, wobei Host ein Name oder eine IP Adresse sein kann. Jeder Treiber kann diesen Teil sehr individuell gestalten! Sie müssen also bei Ihrem Treiberhersteller die benötigte Verbindungsinformatione erfragen oder die Anleitung lesen.

Im Falle von Cloudscape sieht dieser Teil der Verbindung wie bereits bekannt, folgendermassen aus:

```
jdbc:cloudscape:rmi:XP_Team;create=true
```

Der *;create=true* Teil ist ein Attribut der Cloudscape Syntax.

In Kurzfassung:
Sie müssen die Dokumentation Ihres DBMS Anbieters plus des Treiberanbieters durchlesen, um die exakte Syntax herauszufinden.

Eine Bemerkung zu den Verbindungsobjekten: diese werden zwar vom Garbage Collector automatisch aus dem Speicher entfernt. Es gehört aber zu gutem Programmierstil, selber diese Objekte zu schliessen, um zu signalisieren, dass das Objekt nicht mehr gebraucht wird und vorallem, dass allfällige Locks auf der Datenbank gelöscht werden können.

Das Gleiche gilt für die Statement Objekte!

Connection, wie viele andere Teile ds JDBC APIs, ist ein Interface. Viele Programmierer wundern sich, warum ein Interface überhaupt zu einem Objekt werden kann, da ja keine Instanzen von Interfaces gebildet werden können.

Die kurze Antwort darauf ist: JDBC Treiber implementieren das Interface und liefern das Objekt.

Dies ist auch der Grund für ein mögliches Problem bei der Entwicklung von JDBC Applikationen: die Übersetzung zeigte keinerlei Probleme, beim Ausführen traten diese aber plötzlich auf.

1.1.4.1. Was wird vom Connection Interface geliefert?

Bisher haben wir uns immer mit den Treibern und den Statement Objekten befasst. Die Treiber Methode getConnection() liefert ein Connection Objekt, welches folgende Aufgaben hat:

- es ist verantwortlich für das Kreieren der Objekte:
  Statement,
  PreparedStatement und
  CallableStatement (welches man im Zusammenhang mit Stored Procedures benötigt).
- es liefert die DatabaseMetadata Objekte.
- es kontrolliert Transaktionen mittels der commit()und der rollback() Methode.
- das Connection Objekt wird eingesetzt, um Isolation Levels in den Transaktionen zu setzen.

In JDBC wurden auch Methoden definiert, um die spezifischen SQL Anweisungen für eine bestimmte DBMS zu bestimmen. Wir kommen noch darauf zurück.

Im JDBC Version 2.0 Optional Package wurde eine neue DataSource Klasse eingeführt. Diese sieht fast so aus, als wollte man die alte Treiber gesteuerte Verbindung zur Datenbank auf eine neue Grundlage stellen. Sie sollten diese Entwicklung genauer verfolgen, falls Sie produktive Datenbankanwendungen entwickeln wollen. Im Moment , und speziell im J2EE Umfeld, ist die klassische Methode des Verbindungsaufbaus über Treiber die einzig sinnvolle.

1.1.4.2. Generalisieren von Verbindungsinformationen

Auf Grund der obigen Diskussion sollte Ihnen klar sein, dass es kaum Sinn macht den Verbindungsaufbau fix zu programmieren! Die folgende Übung verwendet ein ResourceBundle, in Java im Bereich Internationalisierung massiv verwendet, um Detailinformationen zu erhalten und die Werte der Datenbank plus deren Attribute (Benutzernamen, Passwort) zu setzen.

Noch ein Hinweis betreffend Cloudscape:
Benutzer : sa
Passwort : admin
sind rein willkürlich, da Cloudscape keine Benutzerüberprüfung durchführt.

1.1.4.3. Übung - Batch Connect

Die ersten Übungsbeispiele dienten dem Vertrautwerden mit JDBC. Deswegen spielte die variable Eingabe von Datenbankattributen (Treiber, Benutzerinfo, Passwort) keinerlei Rolle, und half vorallem, die Programme übersichtlich zu gestalten.

Die generelle Technik mit diesem Problem umzugehen in Java ist der Einsatz von RessourceBundle und damit der Einsatz von Property Dateien oder ListResourceBundle.

"Batch," ist ein Begriff, den man aus den Mainframe Umgebungen kennt. "ein Programm läuft ohne Benutzerinteraktion", im Gegensatz zu interaktiven Programmen, bei denen der Benutzer Zusatzinfos eingeben muss.

1.1.4.3.1. Vorbedingungen

Sie sollten die Übung : Kreieren einer Tabelle und Daten einfügen abgeschlossen haben.

1.1.4.3.2. Rumpfprogramme

BatchConnect.properties

```
#PropertiesResourceBundle für den JDBC Verbindungsaufbau
CSDriver=COM.cloudscape.core.RmiJdbcDriver
CSURL=jdbc:cloudscape:rmi:XP_Team
CSUserID=sa
CSPassword=admin
```

BatchJDBCConnect.java

```
import java.sql.*;
import java.util.*;

public class ConnectU
{
  Connection con;
  ResourceBundle rbConnect;
  ResultSet rs;
  ResultSetMetaData rsmd;
  Statement stmt;

// hier müssen Sie noch String Variablen definieren



// Ende String Variablen


  public ConnectU()
  {
```

```
// Zugriff auf das ResourceBundle
// Bestimmen des Treibernamens und der Datenbank-URL, UserID und Passwort


// Ende ResourceBundle

    try   // Laden des JDBC Treibers
    {     // mit newInstance

// Treibercode einfügen

// Ende Treibercode

    }
    catch( Exception e )  // Fehler
    {
      System.err.println(
                  "Der Treiber konnte nicht geladen werden.");
      return;
    } // end catch

    try
    {

// ConnectionCode einfügen
// Treiberinfos und DBinfos bestimmen und ausgeben
// Statementobjekt kreieren



// Ende Connection, Statement Objekte

    }
    catch ( SQLException SQLe)
    {
      System.err.println( "Verbindungsaufbau zu " +
                          sURL + " schlug fehl:" );
      System.err.println( SQLe.getMessage() );
      System.err.println( "SQL State: " +
                          SQLe.getSQLState() );

      if( con != null)
      {
        try { con.close(); }
        catch( Exception e ) {}
      }

      return;
    } // end catch


    try
    {

// Abfragecode einfügen
// und Metadaten abfragen


// Abfragecode


    } // end try
    catch (Exception e) { e.printStackTrace(); }
    finally
    {
      try { stmt.close(); }
      catch( Exception e ) {}

      try { con.close(); }
      catch( Exception e ) {}
```

```
    } // end finally clause

  } // end constructor


  public static void main (String args[])
  {
    new ConnectU();
  } // end main

} // end class BatchJDBCConnect
```

1.1.4.3.3. Aufgabe 1

Bei gegebener Property Datei (BatchConnect.properties oben), versuchen Sie die String Variablen in Ihrem Java Programm so aufzusetzen, zu definieren, dass diese Schlüsselwerte (CS...) im Programm definiert sind und aus der Datei gelesen werden können, wie bei der Intenationalisierung.

Definieren Sie zudem Zeichenketten, welche
1) den Namen der Property-Datei enthalten (ConnectU)
2) eine SELECT Anweisung enthalten, um die Zeile aus der Tabelle herauszulesen, welche "MoJava" als 'Kaffeesorte' enthält.

1.1.4.3.4. Lösung

Zur Lösung Ihrer Aufgabe müssen Sie folgenden Programnmcode einfügen:

```
String sDriver,
 sDriverKey = "CSDriver",
 sPassword,
 sPasswordKey ="CSPassword",
 sQuery =
  "SELECT * FROM KaffeeListe " +
  "WHERE Type = 'MoJava'",
 srbName = "ConnectU",
 sURL,
 sURLKey="CSURL",
 sUserID,
 sUserIDKey = "CSUserID";
```

1.1.4.3.5. Aufgabe 2

Greifen Sie auf das ResourceBundle zu und bestimmen Sie den Namen des Treibers, die Datenbank URL und Benutzername / Passwort.

1.1.4.3.6. Lösung

```
try {
      // PropertyResourceBundle
      rbConnect = ResourceBundle.getBundle( srbName );
      sDriver = rbConnect.getString( sDriverKey );
      sPassword = rbConnect.getString( sPasswordKey );
      sURL = rbConnect.getString( sURLKey );
      sUserID = rbConnect.getString( sUserIDKey );
} catch( MissingResourceException mre ) {
      System.err.println( "ResourceBundle Problem bei " + srbName + ",
             Programm wird abgebrochen." );
      System.err.println("Fehler: " + mre.getMessage() );
      return;
      // Fehlerhafter Abschluss
}
```

1.1.4.3.7. Aufgabe 3

Laden Sie den Datenbanktreiber.

1.1.4.3.8. Lösung

```
Class.forName( sDriver ).newInstance();
```

1.1.4.3.9. Aufgabe 4

Kreieren Sie eine Verbindung zur Datenbank und bestimmen Sie deren Metadaten mittels des DatabaseMetaData Objekts. Geben Sie den Datenbanknamen, den Treibernamen und dessen Version aus und kreieren Sie ein Statement Objekt.

1.1.4.3.10. Lösung

```
con = DriverManager.getConnection ( sURL,
                                    sUserID,
                                    sPassword);

      DatabaseMetaData dbmd = con.getMetaData();
      System.out.println(
        "DBMS: " +
         dbmd.getDatabaseProductName()  + ", " +
         dbmd.getDatabaseProductVersion() );

      System.out.println(
        "Driver: " +
         dbmd.getDriverName()  + ", " +
         dbmd.getDriverVersion() );

      stmt = con.createStatement();
```

1.1.4.3.11. Aufgabe 5

Führen Sie die Abfrage aus. Falls Daten zurück gegeben werden, bestimmen Sie die Metadaten des ResultSet ColumnName, ColumnTypeName und ColumnClassName für alle Spalten des ResultSets.

1.1.4.3.12. Lösung

```
      rs = stmt.executeQuery( sQuery );
      if( rs.next() ) {
            // erste Zeile
            // sofern Daten vorliegen
            rsmd = rs.getMetaData();
            System.out.println();
            int i = rsmd.getColumnCount();
            for( int ndx = 1; ndx <= i; ndx++ ) {
                  System.out.println( "Column Name: " + rsmd.getColumnName(
                                                        ndx ) + "." );
                  System.out.println( "Column SQL Type: " +
                                      rsmd.getColumnTypeName( ndx ) + "." );
                  System.out.println( "Column Java Class Equivalent: " +
                                     rsmd.getColumnClassName( ndx ) + ".\n" );
            } // end for...
      } // end if( result.next() )
```

1.1.4.3.13. Vollständige Lösung

Sie finden eine voll funktionsfähige Lösung auf dem Server / der CD.

1.1.4.3.13.1. Property Datei

BatchConnect.properties

```
#PropertiesResourceBundle für Connection Properties
CSDriver=COM.cloudscape.core.RmiJdbcDriver
CSURL=jdbc:cloudscape:rmi:XP_Team
CSUserID=sa
CSPassword=admin
```

1.1.4.3.13.2. BatchJDBCConnect Java Datei

```
package batchconnect;

import java.sql.*;
import java.util.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */
public class BatchJDBCConnect {
  Connection con;
  ResourceBundle rbConnect;
  ResultSet rs;
  ResultSetMetaData rsmd;
  Statement stmt;
  String sDriver,
         sDriverKey = "CSDriver",
         sPassword,
         sPasswordKey ="CSPassword",
         sQuery =
          "SELECT * FROM KaffeeListe " +
          "WHERE Kaffeesorte = 'MoJava'",
         srbName = "BatchConnect",
         sURL,
         sURLKey="CSURL",
         sUserID,
         sUserIDKey = "CSUserID";


  public BatchJDBCConnect()
  {
    try  // PropertyResourceBundle
    {
      rbConnect = ResourceBundle.getBundle( srbName );

      sDriver   = rbConnect.getString( sDriverKey );
      sPassword = rbConnect.getString( sPasswordKey );
      sURL      = rbConnect.getString( sURLKey );
      sUserID = rbConnect.getString( sUserIDKey );
    }
    catch( MissingResourceException mre )
    {
      System.err.println(
                  "ResourceBundle Problem " +
                   srbName + ", Programm wird abgebrochen." );
      System.err.println("Fehler: " + mre.getMessage() );
      return; // exit on error
    }

    try   //  JDBC Driver laden
    {     //  mit newInstance
```

```
    Class.forName( sDriver ).newInstance();
}
catch( Exception e )  // error
{
  System.err.println(
              "Datenbanktreiber konnte nicht geladen werden.");
  return;
} // end catch

try  {
  con = DriverManager.getConnection ( sURL,
                                      sUserID,
                                      sPassword);

  DatabaseMetaData dbmd = con.getMetaData();
  System.out.println(
    "DBMS: " +
     dbmd.getDatabaseProductName()  + ", " +
     dbmd.getDatabaseProductVersion() );

  System.out.println(
    "Driver: " +
     dbmd.getDriverName()  + ", " +
     dbmd.getDriverVersion() );

  stmt = con.createStatement();
} catch ( SQLException SQLe)  {
  System.err.println( "Der Verbindungsaufbau zu " +
                       sURL + " schlug fehl:" );
  System.err.println( SQLe.getMessage() );
  System.err.println( "SQL State: " +
                       SQLe.getSQLState() );

  if( con != null){
    try { con.close(); }
    catch( Exception e ) {}
  }

  return;
} // end catch

try {
  rs = stmt.executeQuery( sQuery );

  if( rs.next() ) // erste Zeile
  {               // Falls DAten vorliegen
    rsmd = rs.getMetaData();
    System.out.println();

    int i = rsmd.getColumnCount();

    for( int ndx = 1; ndx <= i; ndx++ )
    {
      System.out.println(
       "Column Name: " +
        rsmd.getColumnName( ndx ) + "." );
      System.out.println(
       "Column SQL Type: " +
        rsmd.getColumnTypeName( ndx ) + "." );
      System.out.println(
       "Column Java Class Equivalent: " +
        rsmd.getColumnClassName( ndx ) + ".\n" );
    }

  } // end if( result.next() )

} // end try
catch (Exception e) { e.printStackTrace(); }
finally
{
```

```
      try { stmt.close(); }
      catch( Exception e ) {}

      try { con.close(); }
      catch( Exception e ) {}
    } // end finally clause

  } // end constructor
  public static void main (String args[]) {
    new BatchJDBCConnect();
  } // end main
} // end class ConnectU
```

1.1.4.3.14. Demonstration

Als erstes müssen Sie die Datenbank starten, sonst läuft gar nichts!
Falls Sie dann die Musterlösung beispielsweise mit der Batch Datei starten, erhalten Sie folgendes Ergebnis:

```
DBMS: DBMS:cloudscape, 3.0.4
Driver: Cloudscape Embedded JDBC Driver, 3.0

Column Name: EINTRAG.
Column SQL Type: INT.
Column Java Class Equivalent: java.lang.Integer.

Column Name: MITARBEITER.
Column SQL Type: VARCHAR.
Column Java Class Equivalent: java.lang.String.

Column Name: WOCHENTAG.
Column SQL Type: VARCHAR.
Column Java Class Equivalent: java.lang.String.

Column Name: TASSEN.
Column SQL Type: INT.
Column Java Class Equivalent: java.lang.Integer.

Column Name: KAFFEESORTE.
Column SQL Type: VARCHAR.
Column Java Class Equivalent: java.lang.String.
```

1.1.4.4. Übung - Interactive Connect

In dieser Übung implementieren wir das Beispiel von vorhin noch einmal, wollen aber die Tabelleninformationen dynamisch mit einem GUI anzeigen. Dafür verwenden wir Swing, aber lediglich einfachste Konstrukte.

Dieses Beispiel dient lediglich der Demonstration der Möglichkeiten, bei vielen Daten ist es kaum sinnvoll einsetzbar!

1.1.4.4.1. Vorbedingungen

Sie sollten die Datenbank kreiert und Daten eingefügt haben (Übung 1 oben). Von Vorteil ist es zudem, wenn Sie die Batch Übung bereits gemacht haben, da wir hier lediglich diese Lösung modifizieren.

1.1.4.4.2. Rumpfprogramme

InteractiveJDBCConnect.java:

```
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;
import javax.swing.*;

public class InteractiveJDBCConnect extends JFrame
                      implements ActionListener,
                                 WindowListener
{
  Connection con;
  int        iColumnCount;

  JButton         jb = new JButton("Connect");
  JLabel          jlUserID = new JLabel("UserID:"),
                  jlPassword = new JLabel(
                    "Password:"),
                  jlTable = new JLabel("Table:");
  JPanel          jpCenter = new JPanel(),
                  jpNorth = new JPanel(),
                  jpSouth = new JPanel();
  JPasswordField  jpfPassword =
                     new JPasswordField( 10 );
  JTextArea       jta = new JTextArea( 10, 30 );
  JTextField      jtUserID = new JTextField( 10 ),
                  jtTable = new JTextField( "JJJJData", 10 );
  JScrollPane     jsp = new JScrollPane( jta );

  ResourceBundle rbConnect;
  ResultSet rs;
  ResultSetMetaData rsmd;
  Statement stmt;

// String Variablen einfügen



// Ende String Variablen



  public InteractiveJDBCConnect()
  {
    super("InteractiveJDBCConnect");

    try  // get PropertyResourceBundle
    {
```

```
// Zugriff auf ResourceBundle für
// Treiber und DB Infos.


// ResourceBundle

    }
    catch( MissingResourceException mre )
    {
      System.err.println(
                 "ResourceBundle Problem mit " +
                  srbName + ", Programm wird abgebrochen." );
      System.err.println("Programmfehler: " +
                         mre.getMessage() );
      endApp(); // exit on error
    }

    jb.addActionListener( this );

    jpNorth.add( jlUserID );
    jpNorth.add( jtUserID );
    jpNorth.add( jlPassword );
    jpNorth.add( jpfPassword );

    jpCenter.add( jb );
    jpCenter.add( jlTable );
    jpCenter.add( jtTable );
    jpSouth.add( jsp );

    Container cp = getContentPane();
    cp.add( jpNorth,  BorderLayout.NORTH );
    cp.add( jpCenter, BorderLayout.CENTER );
    cp.add( jpSouth,  BorderLayout.SOUTH );

    addWindowListener( this );
    pack();
    show();

  } // end constructor


  public void doConnect()
  {
    try // Attempt to load the JDBC driver
    {   // with newInstance
        Class.forName( sDriver ).newInstance();
    }
    catch( Exception e )  // Fehler
    {
      jta.setText("Der Treiber konnte nicht geladen werden.");
      return;
    } // end catch

    try
    {
      con = DriverManager.getConnection ( sURL,
                                          sUserID,
                                          sPassword);
      stmt = con.createStatement();
    }
    catch ( SQLException SQLe)
    {
      reportSQLError( SQLe,
        "Probleme beim Verbindungsaufbau zu " +
         sURL + ":" );

      if( con != null)
      {
        try { con.close(); }
```

```
        catch( Exception e ) {}
      }

      return;
    } // end catch

    try
    {
// Query einfügen


// Ende Query


      boolean bFirst = true;
      StringBuffer sb = new StringBuffer(30);
      while( rs.next() )  // falls Daten vorliegen
      {
        if( bFirst )
        {
// Abfrage der Metadaten
// Lesen und Anzeige von ColumnCount
// ColumnLabels
// Reset StringBuffer,
// bFirst to false


// Ende Logik 1
        } // end if( bFirst )

// Bestimme ColumnCount
// mit getObject,
// Append zu jta und reset StringBuffer.


// Ende Logik 2

      } // end while( result.next() )
      jta.setCaretPosition( 0 );

    } // end try
    catch ( SQLException SQLe)
    {
      String s = null;

      try
      {
        s = "Native SQL war: " +
           con.nativeSQL( sQuery + sTable );
      }
      catch( Exception e ) { /* was soll's */ }

      reportSQLError( SQLe, s );
    }
    finally
    {
      try { stmt.close(); }
      catch( Exception e ) {}

      try { con.close(); }
      catch( Exception e ) {}
    } // end finally clause

  } // end doConnect


  public void reportSQLError( SQLException SQLe,
                              String       s )
  {
      jta.setText( s + "\n" );
```

```
      jta.append( SQLe.getMessage() + "\n" );
      jta.append( "SQL State: " +
         SQLe.getSQLState() + "\n"  );
  } // end reportSQLError


  public void endApp()
  {
    dispose();
    System.exit(0);
  }


  // ActionListener
  public void actionPerformed(ActionEvent e)
  {
// Lies Benutzereingabe
// Lösche textarea und rufe doConnect() auf



// Ende Logik 3
    sTable = jtTable.getText();
    if( sTable.equals("") )
    {
      jta.setText( "Die Tabelle muss Daten enthalten." );
      return;
    }
    sUserID = jtUserID.getText();
    sPassword = jpfPassword.getText();
    jta.setText( "" );

    doConnect();
  } // end actionPerformed


// Window Listener Implementation
    public void windowOpened(WindowEvent e) {}

     public void windowClosing(WindowEvent e)
     {
       endApp();
     }

    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
// End Window Listener Implementation

  public static void main (String args[])
  {
    new InteractiveJDBCConnect();
  } // end main

} // end class InteractiveJDBCConnect
```

und die Property Datei:

```
#PropertiesResourceBundle für Connection Properties
CSDriver=COM.cloudscape.core.RmiJdbcDriver
CSURL=jdbc:cloudscape:rmi:XP_Team
```

1.1.4.4.3. Aufgabe 1 - Strings

Ergänzen Sie das Rumpfprogramm so, dass die Property Datei gelesen werden kann. Definieren Sie auch eine String Variable, welche den Namen der Property Datei enthält. In einer weiteren String Variable soll eine SQL Anweisung enthalten sein, welche keinen Tabellennamen enthält, da dieser erst noch interaktiv eingegeben werden soll.

1.1.4.4.4. Lösung

Ergänzen Sie das Rumpfprogramm durch den folgenden Programmcode:

```
  String sDriver,
         sDriverKey = "CSDriver",
         sPassword,
         sQuery =
          "SELECT * FROM ",
         srbName = "InteractiveJDBCConnect",
         sTable,
         sURL,
         sURLKey="CSURL",
         sUserID;
```

1.1.4.4.5. Aufgabe 2 - ResourceBundle

Greifen Sie auf das ResourceBundle zu und bestimmen Sie den Namen des Datenbanktreibers und der Datenbank URL.

1.1.4.4.6. Lösung

```
rbConnect = ResourceBundle.getBundle( srbName );

sDriver   = rbConnect.getString( sDriverKey );
sURL      = rbConnect.getString( sURLKey );
```

1.1.4.4.7. Aufgabe 3 - GUI

Zeigen Sie ein GUI Formular an, mit dem eine BenutzerID, Passwort und ein Tabellenname eingegeben werden kann und einen 'Connect' Knopf besitzt und eine TextArea.

Schauen Sie sich das Programmfragment an: das meiste ist bereits für Sie erledigt!
Falls Sie den 'Connect' Knopf betätigen, wird in actionPerformed() der Tabellenname verifiziert (jtTable), Benutzername und Passwort bestimmt (jtUserID, jtPassword) und die TextArea (jta) gelöscht und schliesslich noch die Methode doConnect() ausgeführt.

1.1.4.4.8. Lösung

```
sTable = jtTable.getText();
if( sTable.equals("") ) {
      jta.setText( "Die Tabelle muss Daten enthalten." );
      return;
}
sUserID = jtUserID.getText();
sPassword = jpfPassword.getText();
jta.setText( "" );
doConnect();
```

1.1.4.4.9. Aufgabe 4 - Query

In der Methode doConnect() wird der Treiber geladen, ein Connection Objekt erhalten und ein Statement Objekt kreiert.

Ihre Aufgabe ist es, den Abfrage-String aufzubauen (Tabellennamen hinzufügen) und die Abfrage auszuführen.

1.1.4.4.10. Lösung

```
rs = stmt.executeQuery( sQuery + sTable );
```

1.1.4.4.11. Aufgabe 5 - Metadaten

In der Methode doConnect() befindet sich eine Abfrageschlaufe mit while(rs.next())

Ihre Aufgabe:
bestimmen Sie das ResultSetMetadata Objekt und zeigen Sie die Anzahl Spalten (ColumnCount) und Anzahl Spaltenlabels (ColumnLabels) als eine Zeichenkette in der TextArea an. Dies soll jedoch innerhalb der while Schleife nur einmal geschehen, also nicht dauernd überschrieben werden.

1.1.4.4.12. Lösung

```
rsmd = rs.getMetaData();
iColumnCount = rsmd.getColumnCount();
for( int i = 1; i <= iColumnCount; i++ ) {
      sb.append( rsmd.getColumnLabel( i ) + "\t" );
}
jta.append( sb.toString() + "\n");
sb.replace( 0, sb.length(), "" );
bFirst = false;
```

1.1.4.4.13. Aufgabe 6 - Ausgabe

Bestimmen Sie in der Methode doConnect() in der while Schlaufe die ColumnCount Zahl der Spalten für die Zeile, mittels ResultSet.getObject() und verknüpfen Sie diese zu einer einzigen Zeichenkette. Fügen Sie diese zur TextArea hinzu.

1.1.4.4.14. Lösung

```
for( int i = 1; i <= iColumnCount; i++ ) {
      sb.append( rs.getObject( i ) + "\t" );
}
jta.append( sb.toString() + "\n");
sb.replace( 0, sb.length(), "" );
```

1.1.4.4.15. Musterlösung

1.1.4.4.15.1. Die Property Datei

```
#PropertiesResourceBundle für Connection Properties
CSDriver=COM.cloudscape.core.RmiJdbcDriver
CSURL=jdbc:cloudscape:rmi:XP_Team
```

1.1.4.4.15.2. Der InteractiveJDBCConnection Programmcode

```
package interactiveconnect;

import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;
import javax.swing.*;
/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * Company:      Joller-Voss GmbH
 * @author J.M.Joller
 * @version 1.0
 */

public class InteractiveJDBCConnect extends JFrame
                     implements ActionListener,
                                WindowListener
{
  Connection con;
  int        iColumnCount;

  JButton         jb = new JButton("Connect");
  JLabel          jlUserID = new JLabel("UserID:"),
                  jlPassword = new JLabel(
                    "Password:"),
                  jlTable = new JLabel("Table:");
  JPanel          jpCenter = new JPanel(),
                  jpNorth = new JPanel(),
                  jpSouth = new JPanel();
  JPasswordField  jpfPassword =
                     new JPasswordField( 10 );
  JTextArea       jta = new JTextArea( 10, 30 );
  JTextField      jtUserID = new JTextField( 10 ),
                  jtTable = new JTextField( "KaffeeListe", 10 );
  JScrollPane     jsp = new JScrollPane( jta );

  ResourceBundle rbConnect;
  ResultSet rs;
  ResultSetMetaData rsmd;
  Statement stmt;
  String sDriver,
         sDriverKey = "CSDriver",
         sPassword,
         sQuery =
          "SELECT * FROM ",
         srbName = "InteractiveConnect",
         sTable,
         sURL,
         sURLKey="CSURL",
         sUserID;


  public InteractiveJDBCConnect()
  {
    super("InteractiveJDBCConnect");

    try  // PropertyResourceBundle
```

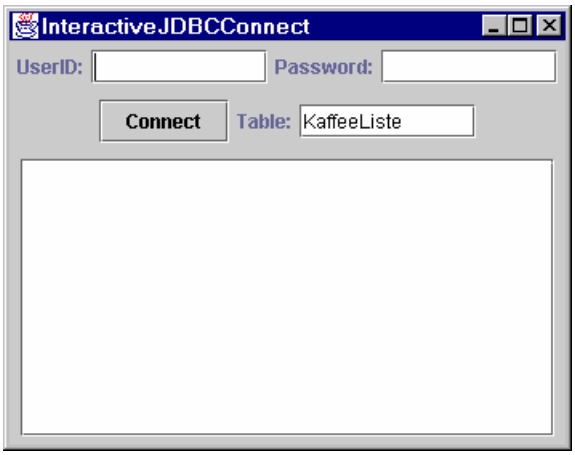
```
    {
      rbConnect = ResourceBundle.getBundle( srbName );

      sDriver   = rbConnect.getString( sDriverKey );
      sURL      = rbConnect.getString( sURLKey );
    }
    catch( MissingResourceException mre )
    {
      System.err.println(
                "ResourceBundle Problem bei " +
                 srbName + ", Programm wird beendet." );
      System.err.println("Programmfehler: " +
                         mre.getMessage() );
      endApp(); // exit on error
    }

    jb.addActionListener( this );

    jpNorth.add( jlUserID );
    jpNorth.add( jtUserID );
    jpNorth.add( jlPassword );
    jpNorth.add( jpfPassword );

    jpCenter.add( jb );
    jpCenter.add( jlTable );
    jpCenter.add( jtTable );
    jpSouth.add( jsp );

    Container cp = getContentPane();
    cp.add( jpNorth,  BorderLayout.NORTH );
    cp.add( jpCenter, BorderLayout.CENTER );
    cp.add( jpSouth,  BorderLayout.SOUTH );

    addWindowListener( this );
    pack();
    show();

  } // end constructor


  public void doConnect()
  {
    try // JDBC driver laden
    {   // mit newInstance
        Class.forName( sDriver ).newInstance();
    }
    catch( Exception e )  // error
    {
      jta.setText("Fehler beim Laden des Datenbanktreibers.");
      return;
    } // end catch

    try
    {
      con = DriverManager.getConnection ( sURL,
                                          sUserID,
                                          sPassword);
      stmt = con.createStatement();
    }
    catch ( SQLException SQLe)
    {
      reportSQLError( SQLe,
        "Probleme beim Verbindungsaufbau zu " +
         sURL + ":" );

      if( con != null)
      {
        try { con.close(); }
        catch( Exception e ) {}
      }
```

```
      return;
    } // end catch

    try
    {
      rs = stmt.executeQuery( sQuery + sTable );

      boolean bFirst = true;
      StringBuffer sb = new StringBuffer(30);
      while( rs.next() )  // falls Daten vorliegen
      {
        if( bFirst )
        {
          rsmd = rs.getMetaData();
          iColumnCount = rsmd.getColumnCount();
          for( int i = 1; i <= iColumnCount; i++ )
          {
            sb.append(
               rsmd.getColumnLabel( i ) + "\t" );
          }

          jta.append( sb.toString() + "\n");
          sb.replace( 0, sb.length(), "" );
          bFirst = false;
        } // end if( bFirst )

        for( int i = 1; i <= iColumnCount; i++ )
        {
          sb.append( rs.getObject( i ) + "\t" );
        }

        jta.append( sb.toString() + "\n");
        sb.replace( 0, sb.length(), "" );

      } // end while( result.next() )
      jta.setCaretPosition( 0 );

    } // end try
    catch ( SQLException SQLe)
    {
      String s = null;

      try
      {
        s = "Native SQL war: " +
           con.nativeSQL( sQuery + sTable );
      }
      catch( Exception e ) { /* was soll's */ }

      reportSQLError( SQLe, s );
    }
    finally
    {
      try { stmt.close(); }
      catch( Exception e ) {}

      try { con.close(); }
      catch( Exception e ) {}
    } // end finally clause

} // end doConnect


public void reportSQLError( SQLException SQLe,
                            String       s )
{
    jta.setText( s + "\n" );
    jta.append( SQLe.getMessage() + "\n" );
    jta.append( "SQL State: " +
```

```
          SQLe.getSQLState() + "\n"  );
    } // end reportSQLError


    public void endApp()
    {
      dispose();
      System.exit(0);
    }


    // ActionListener implementation
    public void actionPerformed(ActionEvent e)
    {
      sTable = jtTable.getText();
      if( sTable.equals("") )
      {
        jta.setText( "Die Tabelle muss Daten enthalten." );
        return;
      }
      sUserID = jtUserID.getText();
      sPassword = jpfPassword.getText();
      jta.setText( "" );

      doConnect();
    } // end actionPerformed


// Window Listener Implementation
    public void windowOpened(WindowEvent e) {}

     public void windowClosing(WindowEvent e)
     {
       endApp();
     }

    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
// End Window Listener Implementation

    public static void main (String args[])
    {
      new InteractiveJDBCConnect();
    } // end main

} // end class InteractiveJDBCConnect
```

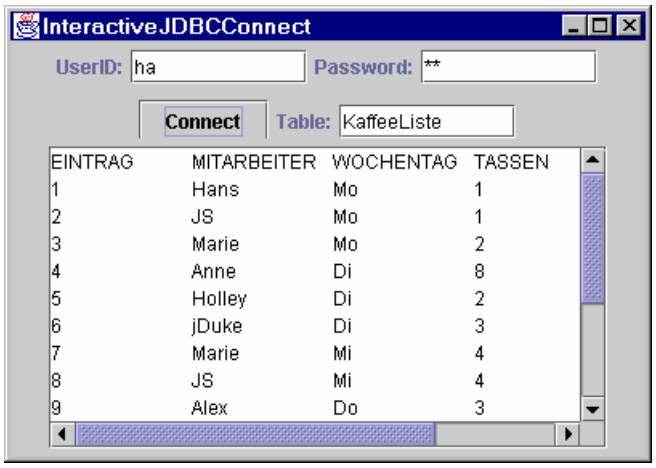
1.1.4.4.16. Demonstration

Die Musterlösung baut einen einfachen Bildschirm auf, und zeigt den Eingabebildschirm an:

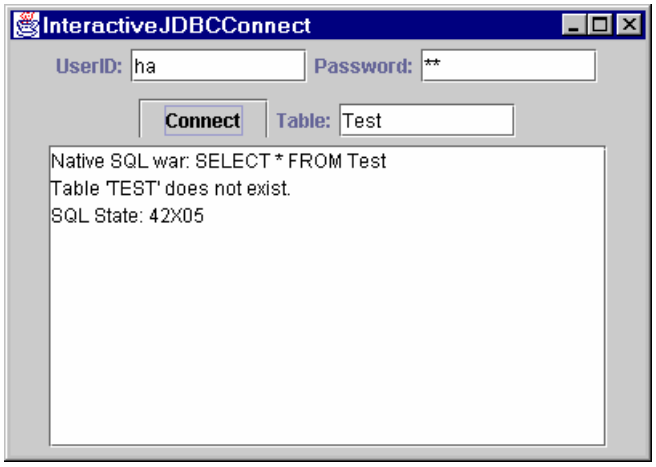


Nachdem Sie Benutzername und Passwort eingegeben haben, wird eine Verbindung zur Datenbank hergestellt und die Daten passend aufbereitet:

und Sie erhalten die Kaffeliste schön formattiert angezeigt. Natürlich können Sie diese noch nicht ausdrucken und interaktiv modifizieren....



Falls Sie den Tabellennamen ändern, versucht das Programm die Daten zu diesem Programm herunterzuladen:



1.1.4.5. Statements, ResultSets und Interaktion mit der Datenbank

Ein `Statement` Objekt ist ein Container oder ein Transport Mechanismus, mit dem man (normalerweise) SQL Anweisungen senden / ausführen kann. Die Ergebnisse werden über die zum Statement gehörende `Connection` dem Client zugänglich gemacht. Wie dies genau geschieht, ist im Abschnitt über das `Connection` Interface beschrieben.

Wichtig zu wissen ist, dass es drei Typen von SQL Anweisungen gibt:
1) Statement
2) PreparedStatement
3) CallableStatement

Die zwei letzteren sind Unterklassen der ersten.

Ein `Statement` Objekt wird nicht mit dem Konstruktor kreiert sondern im Zusammenhang mit einer Datenbankverbindung, einem `Connection` Objekt.

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

Die wichtigsten Methoden sind die *executeXXX()* Methoden des `Statement` Objekts:

- *executeQuery()*
  führt eine SQL Anweisung aus und liefert ein einfaches *ResultSet* Objekt.

- *executeUpdate()*
  wird eingesetzt, um SQL Anweisungen auszuführen, welche eine ganze Tabelle modifizieren, also mehrere Spalten und Zeilen.

- *execute()*
  damit können beliebige SQL Anweisungen ausgeführt werden. Eigentlich ist die Methode für jene Fälle, in denen mehrere Werte zurückgegeben werden. Wir werden darauf nicht näher eingehen.

Um möglichst flexibelzu bleiben, definiert JDBC keinerlei Einschränkungen zum Einsatz von SQL. Wichtig ist lediglich, dass die Datenquelle etwas mit der Anweisung anfangen kann. Dies trifft sogar auf Anweisungen zu, die nicht in SQL beschrieben werden. Allerdings muss ein Treiber, damit er als *JDBC Compliant* bezeichnet werden kann, mindestens den ANSI SQL-92 Entry Level unterstützen.

Ein `Statement` Objekt sollte automatisch geschlossen werden, sobald es nicht mehr benötigt wird, selbst wenn der Garbage Collector unbenötigte Objekte mit der Zeit wegräumt. Die JDBC Empfehlung lautet: schliessen Sie nicht mehr benötigte `Statement` Objekte selber!

1.1.4.6. Modifizieren von Daten

Der Begriff *Update* ist in der Informatik kaum missverständlich. SQL verwendet den Begriff als Befehl zum Modifizieren von Daten in einer Tabelle, wobei die Modifikationen selber aus einer oder mehreren anderen Tabellen stammen können.

In JDBC existiert die Methode executeUpdate(), allerdings mit einer erweiterten Bedeutung: damit kann ein DML Befehle ausgeführt werden DML (INSERT, UPDATE und DELETE), aber auch DDL Anweisungen, wie etwa CREATE TABLE, DROP TABLE und ALTER

TABLE. Damit kann man sagen, dass die Methode immer dann eingesetzt werden kann, falls kein `ResultSet` generiert werden soll oder muss.

JDBC definiert Datentypen, welche den SQL Datentypen entsprechen.

Die Methode `executeUpdate()` liefert eine Integer Zahl, welche anzeigt, wieviele Zeilen in der Tabelle von der Update Anweisung betroffen waren, also auch 0, falls schlicht keine Information zurück geliefert wird, wie etwa in den DDL Anweisungen.

1.1.4.7. Übung - executeUpdate()

Diese Übung zeigt Ihnen die flexiblen Möglichkeiten der Methode `executeUpdate()` anhand eines einfachen Beispiels, welches später noch ergänzt wird.

1.1.4.7.1. Lernziele

Nach dem Durcharbeiten dieser Übung sollten Sie :

- verstehen, wie `executeUpdate()` in DML Anweisungen eingesetzt werden kann.
- einige Ideen haben, wie ein JDBC Datenpflegeprogramm aussehen könnte.

1.1.4.7.2. Szenario

Die Firma XP_Team kommt zum Schluss, dass das Verkaufen oder eventuell auch Verschenken von T-Shirts ein geschickter Marketing Move wäre. Nun soll die Datenbank durch eine Tabelle erweitert werden. Diese soll Informationen über die T-Shirts und den Kunden enthalten:

```
CREATE TABLE XP_TEE (
   Eintrag      INTEGER      NOT NULL,
   Kunde        VARCHAR (20) NOT NULL,
   TGroesse     VARCHAR (10) NOT NULL,
   TFarbe       VARCHAR (10) NOT NULL,
   PRIMARY KEY( Eintrag )
)
```

Geplant ist eine Umfrage unter den XP_Team Kunden, um die gewünschte Grösse und Farbpräferenzen zu erfragen. Dies ist nötig, um die Bestellung platzieren zu können. Da die Daten wahrscheinlich öfters mutiert werden müssen, möchte das XP_Team eine Lösung, welche ein Wartungsprogramm enthält, mit `INSERT`, `UPDATE` oder `DELETE` Anweisungen.

Nach einigem Nachdenken und teuren Rechnungen für konzeptionelles Arbeiten kommt der Software Anbieter zum Schluss, dass die Methode `executeUpdate()` bei der Problemlösung hilfreich sein könnte. Der Rest der konzeptionellen Arbeit ist fast banal: man muss nur noch die Daten, die in der Anweisung benutzt werden von zu einem GUI verschieben.

Weil wir wie die meisten Informatiker zu faul sind alles auszuprogrammieren, beschliessen wir für diese Übung die SQL Anweisungen einfach in eine Property Datei, ein ResourceBundle, zu schreiben. Diese wird im Programm gelesen. Wir können damit den SQL Teil relativ flexibel gestalten, brauchen aber keine Eingabemaske dafür.

Der Aufbau dieser SQL Anweisungsdateien ist denkbar einfach: vorne steht ein Schlüssel, eine ganze Zahl, hinten folgt die SQL Anweisung:

```
1=INSERT INTO XP_Tee VALUES ( ... )
2=INSERT INTO XP_Tee VALUES (... )
```

Die folgenden ResourceBundle stehen Ihnen bereits zur Verfügung. Sie sollten die reihenfolge bei der Ausführung beachten!

- `XPRunInsert.properties`
  Ersteintrag
- `XPRunDelete.properties`
  Später werden die Kunden 20 (Wavy) und 24 (Ralph) aus der Tabelle gelöscht.
- `XPRunUpdate.properties`
  Nach dem Löschen entscheidet das Management, dass lediglich eine reduzierte Anzahl Farben erlaubt wird.
- `XPRunMixed.properties`
  Nach der Bereinigung sind weitere Datenmutationen nötig. Diese umfassen INSERTs, UPDATEs und DELETEs.

`XPRun` benötigt genau diese Property Dateien als Eingabeargumente. Das Programm wird mit:

`java XPRun <propertiesDateiName>` (ohne Extension `.properties`)

Vorgängig muss mit dem Programm `CreateXP_Tee` die Tabelle kreiert werden. Diese Programm steht Ihnen einfach zur Verfügung, da die entsprechende Aufgabe in einer früheren Übung bereits gelöst wurde.

Die Aufgaben betreffen also lediglich `XPRun.java`.

Mit den `ConnectXXX` Programmen können Sie jeweils nach einen Lauf die Daten in der Tabelle nachsehen und überprüfen, ob alles korrekt abgelaufen ist.

Beachten Sie zwei Punkte:

1) *das Programm XPRun.java stellt einen generellen Mechanismus zum Mutieren von Tabellen zur Verfügung.*

   Allerdings wird für jede Änderung eine vollständige und korrekte SQL Anweisung benötigt.

   Das kann didaktisch sinnvoll sein; für die Praxis würde man ein anderes Vorgehen wählen!

2) *Die Applikation birgt die Gefahr, dass einfach alles, was übergeben wird ausgeführt wird, ohne grosse Prüfungen!*

   In der Praxis würde man Prüfungsroutinen einfügen, um die Ausführung sicherer gestalten zu können.

1.1.4.7.3. Voraussetzungen

Sie sollten vor dieser Übung bereits die Datenbank korrekt aufgesetzt haben und Daten aus der Datenbank lesen können, also Data Retrieval und natürlich Verbindungsaufbau zu einer Datenbank!

1.1.4.7.4. Rahmenprogramm

- XPRun.java
- XPRunDelete.properties
- XPRunInsert.properties
- XPRunMixed.properties
- XPRunUpdate.properties

1.1.4.7.5. Aufgaben

1) Bestimmen Sie in `doUpdate()` das RessourceBundle, welches als Parameter beim Aufruf verwendet wurde und in `sargRBName steht.` In einer `Enumeration` werden die `ResourceBundle` Schlüssel gespeichert; deren Anzahl steht in der Variable `int iCount`.

2) Programmieren Sie in `doUpdate()` eine Schleife, in der alle SQL Anweisungen aus dem `ResourceBundle` ausgeführt werden. Die Anzahl Durchläufe steht wie unter 1) erwähnt, in der Variable `iCount`. Bei jedem Durchlauf wird der Zugriffsschlüssel für das `ResourceBundle` um eines erhöht.
   Das gelesene SQL Statement wird in `executeUpdate()` ausgeführt und das Total der betroffenen Zeilen in `int iProcessed` abgespeichert.

3) Fügen Sie nun noch in `doUpdate()` Programmcode ein, mit dem Sie ausgeben, wieviele Zeilen modifiziert wurden.

1.1.4.7.6. Lösungshinweise

```
1)    // PropertyResourceBundle
      // SQL update statements
         rb = ResourceBundle.getBundle( sargRBName );
         e = rb.getKeys();
         // Keys - starten bei 1.
         for( ; e.hasMoreElements(); iCount++ )
         {
           (e.nextElement());
         };


   2) // <= weil die Keys mit 1 starten
      for( int i = 1; i <= iCount; i++ ){
        sKey = "" + i;
        sUpdate = rb.getString( sKey );
        iProcessed += stmt.executeUpdate( sUpdate );
      }


   3) System.out.println( iProcessed + " rows processed." );
```

1.1.4.7.7. Musterlösung

CreateXP_TEE.java:

```
package dbmod_executeupdate;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */
import java.sql.*;

public class CreateXP_Tee{

  public static void main(String[] args)  {
    Connection con = null;
    Statement stmt = null;

   // die Properties lasse ich mal weg!
    String sDriver =  "COM.cloudscape.core.RmiJdbcDriver";
    String sURL =    "jdbc:cloudscape:rmi:XP_Team";
    String sUsername = "sa";
    String sPassword = "admin";

    try   { // Laden des JDBC driver
                // mit newInstance
        Class.forName( sDriver ).newInstance();
    }  catch( Exception e )    {    // Fehler
      System.err.println( "Der Datenbanktreiber konnte nicht geladen
werden..");
      return;
    } // end catch

    try    {
      con = DriverManager.getConnection ( sURL,
                                          sUsername,
                                          sPassword);
      stmt = con.createStatement();
    }   catch ( Exception e)    {
      System.err.println( "Problem beim Verbindungsaufbau zu " +
                          sURL + ":" );
      System.err.println( e.getMessage() );
      if( con != null)      {
        try { con.close(); }
        catch( Exception e2 ) {}
      }

      return;
    } // end catch

    // damit das Programm mehrfach gestartet werden kann
    // wird zuerst versucht, die Tabelle zu löschen
    try    {
      //stmt.executeUpdate( "DROP TABLE XP_Tee" );
      stmt.executeUpdate( "DROP TABLE XP_TEE" );
      System.out.println( "Tabelle XP_TEE wurde geloescht");
    }   catch ( Exception e ) { System.out.println("Tabelle XP_TEE ist noch
nicht vorhanden."); }
    try    {
      //stmt.executeUpdate( "DROP TABLE XP_Tee" );
      stmt.executeUpdate( "DROP TABLE XP_Tee" );
      System.out.println( "Tabelle XP_Tee wurde geloescht");
    }   catch ( Exception e ) { System.out.println("Tabelle XP_Tee ist noch
nicht vorhanden."); }
```

```
    //  SQL
    //  CREATE TABLE
    try    {
      stmt.executeUpdate( "CREATE TABLE XP_TEE (" +
             "Eintrag         INTEGER      NOT NULL, "  +
             "Kunde     VARCHAR (20) NOT NULL, "  +
             "TGroesse        VARCHAR (10) NOT NULL, " +
             "TFarbe          VARCHAR (10) NOT NULL, " +
             "PRIMARY KEY( Eintrag )"                 +
                                              ")" );

      System.out.println( "Die Tabelle XP_TEE wurde kreiert.");
    }    catch ( Exception e )    {
      System.err.println( "Problem mit SQL und der Datenbank " + sURL + ":"
);
      System.err.println( e.getMessage() );
    }    finally    {
      try { stmt.close(); }
      catch( Exception e ) {}

      try { con.close(); }
      catch( Exception e ) {}
    } // end finally clause

  } // end main

} // end class CreateXP_Tee
```

RunXP.java:

```
package dbmod_executeupdate;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */
import java.sql.*;
import java.util.*;

public class XPRun
{
  Connection con;
  ResourceBundle rb;
  Statement stmt;
  String sDriver,
         sDriverKey = "CSDriver",
         sKey,
         sPassword,
         sPasswordKey ="CSPassword",
         sUpdate,
         srbName = "ConnectXP",
         srbUpdate,
         sURL,
         sURLKey="CSURL",
         sUserID,
         sUserIDKey = "CSUserID";


  public XPRun() throws MissingResourceException,
                        ClassNotFoundException,
                        InstantiationException,
                        IllegalAccessException
  {
    // PropertyResourceBundle
    rb = ResourceBundle.getBundle( srbName );

    sDriver   = rb.getString( sDriverKey );
    sPassword = rb.getString( sPasswordKey );
    sURL      = rb.getString( sURLKey );
    sUserID   = rb.getString( sUserIDKey );

    // Laden des JDBC Treibers
    // mit newInstance
    Class.forName( sDriver ).newInstance();

  } // end Konstruktor


  public void doUpdate( String sargRBName )
              throws MissingResourceException,
                     SQLException
  {
    try // Connection und Statement
    {
      con = DriverManager.getConnection ( sURL,
                                          sUserID,
                                          sPassword);
      stmt = con.createStatement();
    }
    catch ( SQLException SQLe)
    {
      reportSQLError( SQLe,
        "Probleme beim Verbindungsaufbau zu " +
```

```
      sURL + ":" );

    if( con != null)
    {
      try { con.close(); }
      catch( Exception e ) {}
    }
    // rethrow Exception
    throw SQLe;

  } // end catch

  try
  {
    Enumeration e = null;
    int iCount = 0;

    // PropertyResourceBundle
    // für SQL Update Statements
    rb = ResourceBundle.getBundle( sargRBName );
    e = rb.getKeys();

    // Zähler
    for( ; e.hasMoreElements(); iCount++ )
    {
      (e.nextElement());
    };

    int iProcessed = 0;

    // Achtung: Keys starten bei 1; also  <=
    for( int i = 1; i <= iCount; i++ )
    {
      sKey = "" + i;
      sUpdate = rb.getString( sKey );

      iProcessed += stmt.executeUpdate( sUpdate );
    }

    System.out.println( iProcessed +
                        " Zeilen wurden veraendert." );

  } // end try
  catch( MissingResourceException mre )
  {
    System.err.println(
               "ResourceBundle Problem mit " +
                sargRBName + ", Programmabbruch." );
    System.err.println("Programmfehler: " +
                       mre.getMessage() );
    throw mre;
  }
  catch ( SQLException SQLe)
  {
    // SQL und Fehler ausgeben
    System.err.println( con.nativeSQL( sUpdate ) );
    reportSQLError( SQLe,
      "Probleme mit executeUpdate:" );
    // rethrow Exception
    throw SQLe;
  }
  finally
  {
    try { stmt.close(); }
    catch( Exception e ) {}

    try { con.close(); }
    catch( Exception e ) {}

  } // end finally
```

```
  } // end doUpdate


  public void reportSQLError( SQLException SQLe,
                              String       s )
  {
    System.err.println( s );
    System.err.println( SQLe.getMessage() );
    System.err.println( "SQL State: " +
                        SQLe.getSQLState() );
  } // end reportSQLError


  public static void main (String args[])
  {
    boolean bContinue = true;
    XPRun xpApp = null;

    if( args.length != 1 )
    {
      System.err.println("Usage: " +
        "java XPRun <SQLUpdateResourceBundleName>" );
      return;
    }

    try
    {
      xpApp = new XPRun();
    }
    catch( Exception e )
    {
      System.err.println("Konstruktor Exception: " +
                         e.getMessage() );
      bContinue = false;
    }
    if( bContinue ){
      try      {
        xpApp.doUpdate( args[0] );
      }  catch( Exception e ) {}
    }
  } // end main

} // end class XPRun
```

ConnectXP.properties:

```
#PropertiesResourceBundle für VDatenbankverbindungen
CSDriver=COM.cloudscape.core.RmiJdbcDriver
CSURL=jdbc:cloudscape:rmi:XP_Team
CSUserID=sa
CSPassword=admin
```

XPRunInsert.properties:

```
#PropertiesResourceBundle für XPRun Properties
1=INSERT  INTO XP_TEE VALUES (9,   'Alex',    'Large',     'Beige')
2=INSERT  INTO XP_TEE VALUES (4,   'Anne',    'Small',     'Pink')
3=INSERT  INTO XP_TEE VALUES (5,   'Holley',  'ExtraHuge', 'Hawaiian')
4=INSERT  INTO XP_TEE VALUES (2,   'JS',      'Medium',    'LightBlue')
5=INSERT  INTO XP_TEE VALUES (10,  'James',   'Large',     'Javan')
6=INSERT  INTO XP_TEE VALUES (1,   'Hans',    'ExtraHuge', 'Green')
7=INSERT  INTO XP_TEE VALUES (6,   'jDuke',   'Petite',    'White')
8=INSERT  INTO XP_TEE VALUES (20,  'Wavy',    'Medium',    'Rainbow')
9=INSERT  INTO XP_TEE VALUES (21,  'Adam',    'Large',     'Red')
10=INSERT INTO XP_TEE VALUES (22,  'Shonica', 'Medium',    'Yellow')
11=INSERT INTO XP_TEE VALUES (23,  'Elton',   'Large',     'Purple')
12=INSERT INTO XP_TEE VALUES (24,  'Ralph',   'Large',     'Teal')
```

XPRunUpdate.properties:

```
#PropertiesResourceBundle für XPRun Properties
1=UPDATE XP_TEE SET TFarbe = 'Yellow' WHERE Eintrag =  4
2=UPDATE XP_TEE SET TFarbe = 'Red'    WHERE Eintrag =  5
3=UPDATE XP_TEE SET TFarbe = 'Blue'   WHERE Eintrag =  2
4=UPDATE XP_TEE SET TFarbe = 'Black'  WHERE Eintrag = 10
5=UPDATE XP_TEE SET TFarbe = 'Yellow' WHERE Eintrag = 22
6=UPDATE XP_TEE SET TFarbe = 'Red'    WHERE Eintrag = 23
```

XPRunDelete.properties:

```
#PropertiesResourceBundle für XPRun Properties
1=DELETE FROM XP_Tee WHERE Entry =  20
2=DELETE FROM XP_Tee WHERE Entry =  24
```

XPRunMixed.properties:

```
#PropertiesResourceBundle für XPRun Properties
1=DELETE FROM XP_TEE WHERE Eintrag =  23
2=INSERT INTO XP_TEE VALUES (25,  'Rosa',  'Petite',  'Blue')
3=UPDATE XP_TEE SET TFarbe = 'Black'   WHERE TFarbe =  'Red'
4=UPDATE XP_TEE SET TGroesse = 'Small'   WHERE TGroesse =  'Petite'
5=UPDATE XP_TEE SET TGroesse = 'XLarge'   WHERE TGroesse =  'ExtraHuge'
```

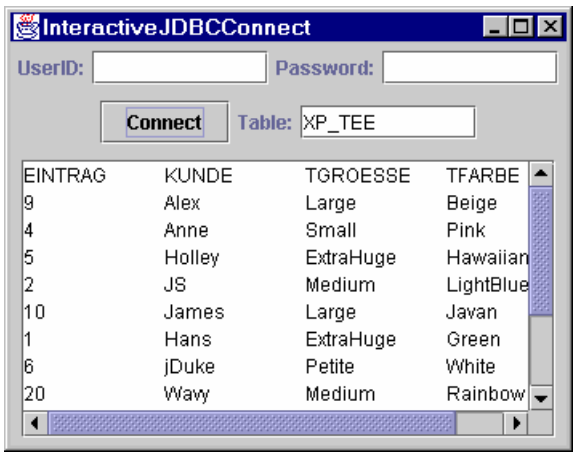
1.1.4.7.8. Demonstration

Auf dem Server / der CD finden Sie Batch Dateien, mit denen Sie alle Programme und die verschiedenen Optionen / Property Dateien / Parameter direkt starten können.

Die Reihenfolge der Ausführung muss beachtet werden, sonst kann es zu Problemen führen. Deswegen wurden die Batch Dateien durchnummeriert.

Damit Sie die jeweiligen Änderungen verfolgen können, wurde auch noch eine Batch Datei für die interaktive Abfrage geschrieben. Diese zeigt Ihnen in einem Fenster die aktuellen Werte in der Tabelle an. Sie müssen lediglich den Tabellennamen von KaffeeListe auf XP_TEE ändern (alles Grossbuchstaben).

Hier ein Beispiel ScreenShot:



Benutzername und Passwort spielt in diesem Fall keine Rolle.

1.1.4.8. Datenbankabfragen

Mit der `executeQuery()` Methode des `Statement` Objekts kann man Result-Sets abfragen, also im Prinzip einfach SQL SELECT Anweisungen ausführen.

Die Methode `executeQuery()` liefert ein Standard `ResultSet` Objekt durch das man mit einem Cursor durchscrollen kann, indem man mit `next()` den Cursor eine Position im Result-Setnach vorne schiebt.

Beachten muss man, dass *immer* ein `ResultSet` Objekt generiert wird, selbst wenn keine Daten darin stehen! Sie dürfen also nicht dieses Objekt auf null überprüfen, um festzustellen ob Daten vorliegen. Die Methode `next()` liefert `true`, falls weitere Daten vorliegen. Sie können also sehr einfach das Result-Set abfragen.

```
int iCount = 0;
while( myResultSet.next() )
{
  // Lesen der Daten
  // Verarbeiten der Daten
  iCount++;
}
if( iCount == 0 )
{
  System.out.println("Im ResultSet waren keine Daten.");
}else if( bNoErrorsOrExceptionsOrEarlyTerminations ){
  System.out.println("Es wurden alle Daten aus dem ResultSet
                        verarbeitet.");
}
```

Die Spalten werden von links nach rechts, in der gleichen Reihenfolge wie in der SELECT Anweisung gelesen. Praktisch ist die Verwendung eines Index (1,2,3 .... nicht 0,1,2,...); aber die Verwendung der Spaltennamen macht das Programm vermutlich lesbarer.
Mit *getXXX()* Methoden des ResultSet Objekts können Sie die Daten herauslesen.
JDBC definiert Datentypen, die den SQL Datentypen entsprechen. pro Datentyp existiert eine *getXXX()* Methode.

Pro `Statement` Objekt existiert jeweils nur ein `ResultSet` Objekt. Das gleiche Objekt kann mehrfach wiederverwendet werden. Sie müssen also darauf achten alle Daten herauszulesen, bevor die neuen Daten alle alten überschreiben, falls Sie mehrere Abfragen staren müssen.

Sobald Sie ein `ResultSet` Objekt nicht mehr benötigen sollten Sie dieses schliessen, mit der *close()* Methode. Dies geschieht auch implizit, sobald Sie neue Daten abfragen.

Ein `ResultSet` Objekt liefertIhnen auch *Metadaten*, also Informationen über das `ResultSet` Objekt.

1.1.4.9. Übung - Selektieren und Präsentieren von Informationen

In dieser Übung geht es darum, Daten auszuwählen. Falls man die Datensätze, / Rows / Zeilen einer Tabelle jeweils nur genau einmal im ResultSet haben will, kann man im SQL Befehl das Schlüsselwort DISTINCT verwenden. Zudem wollen wir ein Beispiel anschauen, bei dem die Daten grafisch aufbereitet werden, mit Hilfe einer Java Bean aus dem Internet, gratis von IBM.

1.1.4.9.1. Lernziele

Nach dem Durcharbeiten dieses Übung sollten Sie in der Lage sein:

- einige Details zum SELECT Befehl von SQL zu verstehen und anzuwenden.
- Daten aus Tabellen zu lesen und grafisch aufzubereiten

1.1.4.9.2. Szenario

Das Management Team vom XP_Team möchte gerne eine Auswertung des Kaffeekonsums, weil sie sich Gedanken über die Gesundheit der Mitarbeiter machen. Eine externe Firma beschliesset nach gründlicher Analyse der Anforderungen, dass mit dem Lightweight Chart Beans (http://www.alphaworks.ibm.com/ab.nsf/techmain/4822D3386110D441882567540059C8D5?OpenDocument) Package von alphaWorks (http://www.alphaworks.ibm.com) eine kostengünstige Lösung erstellt werden kann.

Bemerkung: in dieser Übung gehen wir nicht auf die Details der Java Beans ein. Der Download enthält einiges an Dokumentation und Sie werden am konkreten Beispiel sehen, wie man mit JavaBeans unzugehen hat.

Einzig die Installation ist wirklich wesentlich:
1) laden Sie die ZIP Datei herunter oder kopieren Sie die Datei vom Server / der CD.
2) kopieren Sie das Archiv s11_chart.jar in **JAVA_HOME/jre/lib/ext**.

1.1.4.9.3. Voraussetzungen

Sie sollten bereits die Übung zum interaktiven Zugriff auf Tabellen und das Lesen von Daten aus einer Tabelle verstehen und die entsprechenden Programme kennen.

1.1.4.9.4. Rahmenprogramme

Ihnen stehen zwei Dateien zur Verfügung:
1) die Property Datei für den Verbindungsaufbau

```
#PropertiesResourceBundle für den Verbindungsaufbau
CSDriver=COM.cloudscape.core.RmiJdbcDriver
CSURL=jdbc:cloudscape:rmi:XP_Team
```

2) ein Rahmenprogramm in Java

```
import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;
import javax.swing.*;

import com.ibm.eou.swingchart.*;  // chart import


public class ChartXP extends JFrame
                     implements ActionListener,
```

```
                                WindowListener
{
  boolean bFirstPass = true;
  Connection con;
  int i, ndx;

  JButton         jb = new JButton("Connect");
  JLabel          jlUserID = new JLabel("UserID:"),
                  jlPassword = new JLabel(
                    "Passwort:");
  JPanel          jpCenter = new JPanel(),
                  jpNorth = new JPanel(),
                  jpSouth = new JPanel();
  JPasswordField  jpfPassword =
                     new JPasswordField( 10 );
  JTextArea       jta = new JTextArea(
                    "Programmfehler.", 2, 30 );
  JTextField      jtUserID = new JTextField( 10 );

  ResourceBundle rbConnect;
  ResultSet rs;
  ResultSetMetaData rsmd;
  Statement stmt;
  String sDriver,
         sDriverKey = "CSDriver",
         sPassword,
         sQuery =
// TODO
// OORDER BY Query einfügen

// ENDTODO

         sQueryD =
// TODO
// DISTINCT Query einfügen

// ENDTODO

         srbName = "ConnectXP",
         sURL,
         sURLKey="CSURL",
         sUserID;

// KaffeeListe Daten / Variablen
  int iCups;

// alphaworks Chart Variablen
  Color[] cuc = {
            Color.blue,    Color.cyan, Color.green,
            Color.magenta, Color.red,  Color.yellow
                };
  double[][] ducValues;
  String[]   sucLegends;
  UniversalChart ucChart = new UniversalChart();



  public ChartXP() {
    super("ChartXP - Kaffee Verbrauch");

    // lies das PropertyResourceBundle
    try  {
      rbConnect = ResourceBundle.getBundle( srbName );

      sDriver   = rbConnect.getString( sDriverKey );
      sURL      = rbConnect.getString( sURLKey );
    }
    catch( MissingResourceException mre ) {
      System.err.println(
                 "ResourceBundle Problem: " +
```

```
                    srbName + ", Programm wird beendet." );
      System.err.println("Programmfehler: " +
                         mre.getMessage() );
      endApp(); // exit on error
    }

    jb.addActionListener( this );

    jpNorth.add( jlUserID );
    jpNorth.add( jtUserID );
    jpNorth.add( jlPassword );
    jpNorth.add( jpfPassword );

    jpCenter.add( jb );
    jpCenter.add( jta );

// Position und Chart Komponenten
    ucChart.setChartType( Chart.PARALLEL );
    ucChart.setShowGridLines( false );
    jpSouth.add( ucChart );

    Container cp = getContentPane();
    cp.add( jpNorth,  BorderLayout.NORTH );
    cp.add( jpCenter, BorderLayout.CENTER );
    cp.add( jpSouth,  BorderLayout.SOUTH );

    addWindowListener( this );
    pack();

    ucChart.setVisible( false );

    show();

  } // end Konstruktor
```

```
public void doConnect() {
  String sSaveValue = null,
         sValue = null;
  //Laden des JDBC Treibers
  // mit newInstance
  try  {
      Class.forName( sDriver ).newInstance();
  }
  catch( Exception e )   {
    jta.setText("Treiber konnte nicht geladen werden.");
    return;
  } // end catch

  try    {
    con = DriverManager.getConnection ( sURL,
                                        sUserID,
                                        sPassword);
    stmt = con.createStatement();
  }
  catch ( SQLException SQLe)    {
    jta.setText( "Verbindungsaufbau zu " +
                 sURL + " schlug fehl:" );
    jta.append( SQLe.getMessage() );
    jta.append( "SQL State: " + SQLe.getSQLState() );

    if( con != null)      {
      try { con.close(); } catch( Exception e ) {}
    }

    return;
  } // end catch

  try    {
    bFirstPass = true;
    i = ndx = 0;

    // TODO
    // Bestimmen der Anzahl unterschiedlicher Kaffeesorten
    // fügen Sie den Query ein und bestimmen Sie auch das Total

    // ENDTODO

    // TODO
    // Definieren Sie ein Array für die Daten

    // ENDTODO

    // TODO
    // bestimmen Sie die Daten (ORDER BY Query)

    // ENDTODO

    while( rs.next() ) { // falls Daten vorhanden
      iCups = rs.getInt( 1 );
      sValue = rs.getString( 2 );

      // TODO
      // Programmcode für erste Zeile einfügen

      // ENDTODO

      // TODO
      // fügen Sie den Abschluss Code ein

      // ENDTODO

      // TODO
      // Totale berechnen und in
      // ducValues Array abspeichern
```

```
        // ENDTODO
      } // end while

      ucChart.setColors( cuc );
      ucChart.setLegends( sucLegends );
      ucChart.setValues( ducValues );
      if( !ucChart.isVisible() )
      { ucChart.setVisible( true ); }

    } catch ( SQLException SQLe) {
      jta.append( SQLe.getMessage() );
      jta.append( "SQL State: " + SQLe.getSQLState() );
      SQLe.printStackTrace();
    } finally    {
      try { stmt.close(); } catch( Exception e ) {}

      try { con.close(); }  catch( Exception e ) {}
    } // end finally clause

  } // end doConnect


  public void endApp()  {
    dispose();
    System.exit(0);
  }
  // ActionListener implementation
  public void actionPerformed(ActionEvent e)  {
    sUserID = jtUserID.getText();
    sPassword = jpfPassword.getText();
    jta.setText( "Es trat kein Fehler auf." );

    doConnect();
  } // end actionPerformed


  // Window Listener Implementation
    public void windowOpened(WindowEvent e) {}

     public void windowClosing(WindowEvent e)
     {
       endApp();
     }

    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
  // End Window Listener Implementation

  public static void main (String args[]) {
    new ChartXP();
  } // end main

} // end class ChartXP
```

1.1.4.9.5. Aufgaben

1) Das Entwicklerteam möchte eine Auswertung erstellen, bei der die Kaffeesorten ausgewertet werden. Dies wäre manuell sehr einfach, maschinell müssen wir einen Mechanismus haben, mit dem man die Tabelle lesen kann, aber nur unterschiedliche Kaffeesorten auswählt (keine Mehrfachnennungen der Kaffeesorte).
SQL stellt dafür das Schlüsselwort DISTINCT zur Verfügung. Dies hat auch noch den Vorteil, dass man die Anzahl Kaffeesorten zählen kann. Dies ist eine Voraussetzung für die Erstellung einer grafischen Auswertung.

   SELECT DISTINCT Kaffeesorte FROM KaffeeListe
   und

   SELECT Tassen, Kaffeesorte FROM KaffeeListe
   ORDER BY Kaffeesorte

   Ihre Aufgabe:
   Definieren Sie String Variablen, welche die SQL Anweisungen aufnehmen können.

2) Das interaktive Abfrageprogramm wird als Basis verwendet. Im Konstruktor werden einige Änderungen vorgenommen. Beispielsweise wird in der TextArea lediglich die Fehlermeldung ausgegeben, da wir die Daten visualisieren möchten. Die Visualisierung benötigt die Daten in einem bestimmten Format, also auch hier einige Arbeit.

   Weitere Änderungen geschehen in der `doConnect()` Methode, nachdem wir die Verbindung zur Datenbank aufgebaut haben und beim Kreieren des `Statement`. Kontrollvariablen werden initialisiert, `bFirstPass` wird auf `true` gesetzt und `i` und `ndx` werden auf 0 gesetzt.

   Ihre Aufgabe:
   Schreiben Sie ein Abfrage, welche die Kaffeesorten bestimmt (`DISTINCT Kaffeesorte)` und die Anzahl in `ndx` abspeichert.

3) Nun müssen wir die Konstruktion der grafischen Auswertung vorbereiten. Dzu verwenden wir Arrays mit den entsprechenden Werten und Legenden `ducValues` und `sucLegends`.

4) Fügen Sie Programmzeilen ein, welche die `ORDER BY` Abfrage ausführen.

5) In der Leseschleife werden die Daten `Tassen` und `Kaffeesorte` gelesen, mit Hilfe der *getXXX(int)* Methode, nicht mit Hilfe der Spaltennamen . Dies ist leicht effizienter als die Spaltennamen einzusetzen.

   Ihre Aufgabe:
   Lesen Sie die `Kaffeesorte` aus der ersten Zeile und speichern Sie diese in der Variable `sSaveValue` für spätere Vergleiche.
   Speichern Sie diesen Wert auch als erstes Element im Array `sucLegends`.

6) Wie bereits früher erwähnt, kann man mit ORDER BY die Daten sortieren. Wir wollen die einzelnen Kaffeesorten bestimmen, also bietet es sich an, die Datensätze zuerst zu sortieren, damit der Vergleich alteSorte / neueSorte leichter wird.
In einer Mehrbenutzerdatenbank müssten wir spezielle Vorkehrungen treffen, da die Daten sich sehr schnell ändern können. Um `DISTINCT` zu illustrieren, verwenden wir zwei Queries. Aber im schlimmsen Fall würde während der Abfrage eine Kaffeesorte eingefügt und dies würde zu einer OutOfBounds Exception führen, da wir zuerst zuwenig Kaffeesorten gefunden haben.

   Ihre Aufgabe:
   Überlegen Sie sich, wie diese Änderungen abgefangen werden könnten und der Benutzer darüber informiert werden könnte.

7) Kumulieren Sie die Anzahl Kaffeetassen im `ducValues` Array als `[i][0]` beim i-ten Durchlauf .

8) Die grafische Darstellung wird Ihnen abgenommen.

1.1.4.9.6. Hilfestellungen

1) Lösungshinweis

```
sQueryD:
"SELECT DISTINCT Kaffeesorte " +
 "FROM KaffeeListe"

sQuery:
 "SELECT Tassen, Kaffeesorte " +
 "FROM Kaffeeliste " +
"ORDER BY Kaffeesorte"
```

2) Lösungshinweis

```
rs = stmt.executeQuery( sQueryD );
 while( rs.next() )  {
   ndx++;
}
```

3) Lösungshinweis

```
ducValues = new double[ndx][1];
sucLegends = new String[ndx];
```

4) Lösungshinweis

```
rs = stmt.executeQuery( sQuery );
```

5) Lösungshinweis

```
if(  bFirstPass ){
   sSaveValue = sValue;
   sucLegends[i] = sValue;
   bFirstPass = false;
}
```

6) Lösungshinweis

```
if( !sSaveValue.equals( sValue ) )  { // Wert ändert
i++;
// Grenzen beachten
if( i >= ndx )  {
 jta.append( "Es trat in der Zwischenzeit eine Datenmutation auf " +
               "Starten Sie neu." );
            break;
          }

         sSaveValue = sValue;
         sucLegends[i] = sValue;
       }
```

7) Lösungshinweis

```
ducValues[i][0] += iCups;
```

1.1.4.9.7. Musterlösung

0) Die Installation der JavaBean:
   Sie können wie üblich die jar Datei ins ext Verzeichnis kopieren und im JBuilder als Library definieren.
   Das ZIP enthält zwei jar's: 101 und 11... also zwei Versionen! Sie benötigen die neuere 11, sonst wird ein Swing Fehler resultieren (früher war Swing com.sun...., jetzt einfach java oder javax....)

1) Die Properties:

```
PropertiesResourceBundle für den Verbindungsaufbau
CSDriver=COM.cloudscape.core.RmiJdbcDriver
CSURL=jdbc:cloudscape:rmi:XP_Team
```

2) ChartXP.java:

```
package chartxp;

import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;
import javax.swing.*;

import com.ibm.eou.swingchart.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */

public class ChartXP extends JFrame
                     implements ActionListener,
                                WindowListener
{
  boolean bFirstPass = true;
  Connection con;
  int i, iCount, ndx;
  JButton         jb = new JButton("Connect");
  JLabel          jlUserID = new JLabel("UserID:"),
                  jlPassword = new JLabel(
                    "Passwort:");
  JPanel          jpCenter = new JPanel(),
                  jpNorth = new JPanel(),
                  jpSouth = new JPanel();
```

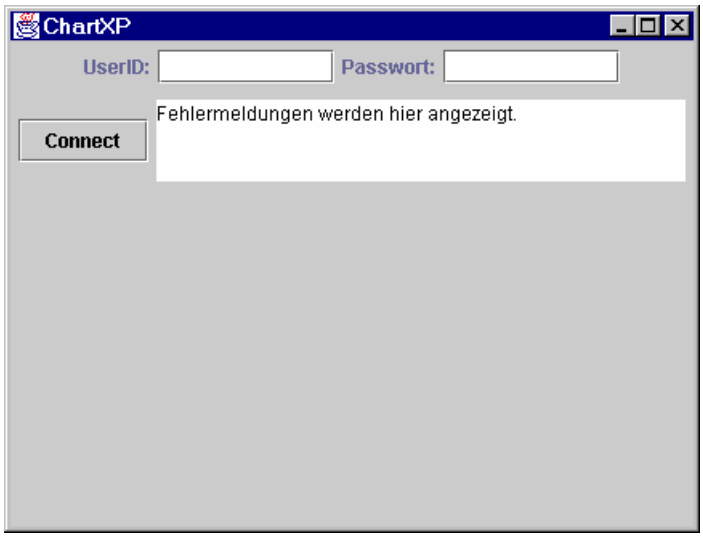
```
JPasswordField  jpfPassword =
                   new JPasswordField( 10 );
JTextArea       jta = new JTextArea(
                  "Fehlermeldungen werden hier angezeigt.", 3, 30 );
JTextField      jtUserID = new JTextField( 10 );

ResourceBundle rbConnect;
ResultSet rs;
ResultSetMetaData rsmd;
Statement stmt;
String sDriver,
       sDriverKey = "CSDriver",
       sPassword,
       sQuery =
        "SELECT Tassen, Kaffeesorte " +
        "FROM KaffeeListe " +
        "ORDER BY Kaffeesorte",
       sQueryD =
        "SELECT DISTINCT Kaffeesorte " +
        "FROM Kaffeeliste",
       srbName = "ConnectXP",
       sURL,
       sURLKey="CSURL",
       sUserID;

// Daten Variablen
int iCups;

// alphaworks Chart Variablen
Color[] cuc = {
          Color.blue,    Color.cyan, Color.green,
          Color.magenta, Color.red,  Color.yellow
              };
double[][] ducValues;
String[]   sucLegends;
UniversalChart ucChart = new UniversalChart();

public ChartXP(){
  super("ChartXP");

  // PropertyResourceBundle
  try  {
    rbConnect = ResourceBundle.getBundle( srbName );

    sDriver   = rbConnect.getString( sDriverKey );
    sURL      = rbConnect.getString( sURLKey );
  } catch( MissingResourceException mre )    {
    System.err.println(
               "ResourceBundle Problem bei " +
                srbName + ", Programm wird abgebrochen." );
    System.err.println("Programmfehler: " +
                       mre.getMessage() );
    endApp(); // exit on error
  }

  jb.addActionListener( this );

  jpNorth.add( jlUserID );
  jpNorth.add( jtUserID );
  jpNorth.add( jlPassword );
  jpNorth.add( jpfPassword );

  jpCenter.add( jb );
  jpCenter.add( jta );

  // Position und chart Komponente
  ucChart.setChartType( Chart.PARALLEL );
  ucChart.setShowGridLines( false );
  jpSouth.add( ucChart );
```

```
    Container cp = getContentPane();
    cp.add( jpNorth,  BorderLayout.NORTH );
    cp.add( jpCenter, BorderLayout.CENTER );
    cp.add( jpSouth,  BorderLayout.SOUTH );

    addWindowListener( this );
    pack();

    ucChart.setVisible( false );

    show();

  } // end Konstruktor


  public void doConnect()  {
    String sSaveValue = null,
           sValue = null;
    // JDBC Treiber laden
    // mit newInstance
    try     {
        Class.forName( sDriver ).newInstance();
    }  catch( Exception e )     {
      jta.setText("Fehler beim Laden des JDBC Treibers.");
      return;
    } // end catch

    try    {
      con = DriverManager.getConnection ( sURL,
                                          sUserID,
                                          sPassword);
      stmt = con.createStatement();
    }    catch ( SQLException SQLe)    {
      jta.setText( "Probleme beim Verbindungsaufbau zu " +
                    sURL + ":" );
      jta.append( SQLe.getMessage() );
      jta.append( "SQL State: " + SQLe.getSQLState() );

      if( con != null)    {
        try { con.close(); } catch( Exception e ) {}
      }

      return;
    } // end catch

    try    {
      bFirstPass = true;
      i = iCount = ndx = 0;

      // bestimme die Anzahl unterschiedlicher Werte
      rs = stmt.executeQuery( sQueryD );
      while( rs.next() )      {
        ndx++;
      }
      // kreiere passende Arrays
      ducValues = new double[ndx] [1];
      sucLegends = new String[ndx];

      // lies die Werte
      rs = stmt.executeQuery( sQuery );

      while( rs.next() )  { // falls Daten vorliegen

        iCups = rs.getInt( 1 );
        sValue = rs.getString( 2 );
        if(  bFirstPass )        {
          sSaveValue = sValue;
          sucLegends[i] = sValue;
          bFirstPass = false;
        }
```

```
        if( !sSaveValue.equals( sValue ) )  { // falls Mutation auftrat
          i++;
          // Mutation während der Arbeit
          if( i >= ndx )   {
            jta.append( "Waehrend der Abfrage trat eine Mutation auf. "
+
               "Starten Sie neu." );
            break;
          }

          sSaveValue = sValue;
          iCount = 0;
          sucLegends[i] = sValue;
        }

        ducValues[i][0] += iCups;

      } // end while

      ucChart.setColors( cuc );
      ucChart.setLegends( sucLegends );
      ucChart.setValues( ducValues );
      if( !ucChart.isVisible() ) { ucChart.setVisible( true ); }

    }  catch ( SQLException SQLe)   {
      jta.append( SQLe.getMessage() );
      jta.append( "SQL State: " + SQLe.getSQLState() );
      SQLe.printStackTrace();
    }    finally {
      try { stmt.close(); }   catch( Exception e ) {}

      try { con.close(); }   catch( Exception e ) {}
    } // end finally clause

  } // end doConnect


  public void endApp()  {
    dispose();
    System.exit(0);
  }


  // ActionListener implementieren
  public void actionPerformed(ActionEvent e)  {
    sUserID = jtUserID.getText();
    sPassword = jpfPassword.getText();
    jta.setText( "" );
    doConnect();
  } // end actionPerformed

// Window Listener Implementation
    public void windowOpened(WindowEvent e) {}
     public void windowClosing(WindowEvent e)  {
       endApp();
     }

    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
// End Window Listener Implementation

  public static void main (String args[])  {
    new ChartXP();
  } // end main
} // end class ChartXP
```

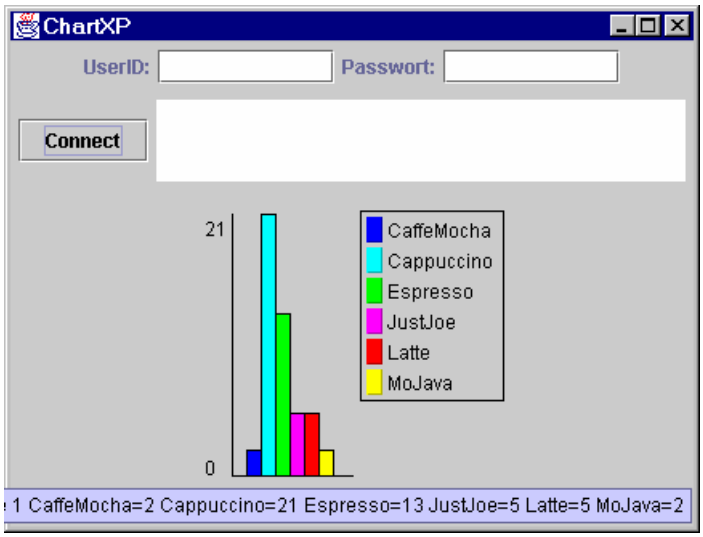
1.1.4.9.8. Demonstration

Wenn Sie das Programm nach Aufstarten von Cloudscape starten, sollten Sie folgenden Bildschirm sehen:



Nachdem Sie die connect Taste gedrückt haben, werden die Daten aufbereitet und die Grafik erscheint:

die Legende unten erscheint sobald Sie mit der Maus über die Grafik fahren, mit den Cursor auf die Grafik zeigen.



1.1.5. Vorbereitete Datenbank Anweisungen - PreparedStatement

Ein PreparedStatement ist ein Subinterface von Statement, mit zusätzlichen Eigenschaften:

- Eine normale SQL Anweisung wird an die Datenbank gesandt vorher übersetzt und vorbereitet.
  Nachdem die Anweisung einmal gesandt wurde, muss sie nicht immer wieder neu übersetzt und vorbereitet werden.
  Beim dynamischen SQL ausführen müssen die Schritte 'übersetzen' und 'vorbereiten' jedesmal durchgeführt werden.
  Je nach DBMS könnte ein PrepareStatement im Datenbank Cache abgespeichert werden und bei Bedarf schnell ausgeführt werden, also PreparedStatement wiederverwendet werden. Damit wird dem Treiber viel Arbeit abgenommen: die Hauptarbeit leistet die DBMS.
- Ein PreparedStatement kann IN Parameter aufnehmen. Diese sind Aufrufparameter und entsprechen den Spaltenwerten.
- PreparedStatements bemühen sich auch, die Datenkonversionen durchzuführen.

Bemerkung: Die SQL3 Datentypen gehen davon aus, dass man PreparedStatement für DML (Data Manipulation Language) einsetzt.

Hier sehen Sie in zwei Beispielen, wie ein PreparedStatement aufgesetzt wird:

```
pstmtU = con.prepareStatement(
  "UPDATE meineTabelle SET meineStringSpalte = ? " +
  "WHERE meineIntSpalte = ?"  );

pstmtQ = con.prepareStatement(
  "SELECT meineStringSpalte FROM meineTabelle " +
  "WHERE meineIntSpalte = ? ");
```

Beachten Sie: das Interface heisst PreparedStatement, die Methode oben in der Anweisung heisst prepareStatement (ohne d bei 'prepare').

Das Fragezeichen markiert die Position der Werte, die in der Anwendung eingefügrt, an die Anweisung übergeben werden müssen. Allgemein bezeichnet man diese als *Parameter Markers*. Diese werden einfach durchnummeriert, falls es mehrere sind: die erste Variable ientspricht der 1, die zweite der 2 usw.

Die Methode *setXXX()* des PrepareStatement wird eingesetzt, um die Werte der IN Parameter zu setzen. Diese bleiben dann unverändert, bis sie neu gesetzt werden.

Hier ein Beispiel zu den vorigen Anweisungen:

```
// Update
pstmtU.setString( 1, "meinString" );
pstmtU.setInt( 2, 1024 );
pstmtU.executeUpdate();

// Query
pstmtQ.setInt( 1, 1024 );
pstmtQ.executeQuery();
```

Sie können auch SQL Anweisungen als PreparedStatement definieren, welche keine Parameter besitzen.

Ein PreparedStatement macht immer dann Sinn, wenn eine Anweisung mehrfach ausgeführt werden muss, also beispielsweise Daten in eine Tabelle eingefügt werden sollen. Die Variable erhält dann der Reihe nach die Werte, die Sie einfügen wollen.

1.1.5.1. Übung

In dieser Übung modifizieren Sie die Datenbanktabelle KaffeeListe und mutieren mehrere Spalten mit Hilfe eines `PreparedStatement`, um Ihnen eine Idee zu geben, wie eine solche Anweisung sinnvoll eingesetzt werden kann.

1.1.5.1.1. Lernziele

Nach dem Durcharbeiten dieser Übung sollten Sie

- das Konzept des `PreparedStatement` verstehen und einsetzen können.
- wissen, wie man eine Tabelle mit der `ALTER TABLE` SQL Anweisung verändern kann.

Das XP_Team ist von Ihrer Arbeit bisher recht begeistert. Allerdings möchte das Management an Stelle der Kurzzeichen "Mo", "Di", ... den vollen Namen des Wochentages sehen, in der Auswertung!

Das XP_Team möchte allerdings den alten Bericht genau so belassen und keinerlei Änderungen sehen oder neue Layouts verstehen lernen.

Zudem wurde einstimmig beschlossen, dass der Name "JustJoe" in "Jus'Joe" abgeändert werden soll.

Ihnen glänzen die Augen, weil Sie mal endlich wieder ein Projekt haben, bei dem Sie wissen, um was es geht!

Beachten Sie, dass in diesem Programm die Tabelle KaffeeListe verändert wird, also im schlimmsten Fall die alten Programme nicht mehr laufen! Allerdings haben wir uns bemüht, genau auf diesen Punkt sehr genau zu achten.

Behauptung: die Programme funktinieren weiterhin;
Aber: die neuen Daten erscheinen in keiner der alten Auswertungen, weil die SELECT und UPDATE Anweisungen nicht modifiziert wurden.

1.1.5.1.2. Voraussetzungen

Sie sollten bereits Tabellen kreieren können und Updates ausführen können. Falls Sie nicht mehr wissen, schauen Sie einfach weiter vorne nach, eventuell die Musterlösung, falls Sie die Aufgabenstellung nicht so ganz verstehen (das würde ich noch verstehen: die Aufgabenstellung war bisher komplizierter als die Erarbeitung der Lösung).

1.1.5.1.3. Rahmenprogramme

Inhalt der Property Datei:

```
#PropertiesResourceBundle für Connection Properties
CSDriver=COM.cloudscape.core.RmiJdbcDriver
CSURL=jdbc:cloudscape:rmi:XP_Team
CSUserID=sa
CSPassword=admin
```

PrepareAlter.java:

```
import java.sql.*;
import java.util.*;

public class PrepareAlter
{
  static String[] asWochentag =
  {
    "Mo",
    "Di",
    "Mi",
    "Do",
    "Fr",
    "Sa",
    "So"
  };

  static String[] asVollerWochentag =
  {
    "Montag",
    "Dienstag",
    "Mittwoch",
    "Donnerstag",
    "Freitag",
    "Samstag",
    "Sonntag"
  };

  Connection con;
  int ndx,
      iRowCount;
  PreparedStatement pstmt1 = null,
                    pstmt2 = null;
  ResourceBundle rbConnect;
  ResultSet rs;
  Statement stmt;
  String sDriver,
         sDriverKey = "CSDriver",
         sPassword,
         sPasswordKey ="CSPassword",
// TODO Abfrage
         //sQuery =

// ENDTODO

// TODO erster Update
         //sUpdate1 =

// ENDTODO

// TODO zweiter Update
         //sUpdate2 =

// ENDTODO

         srbName = "ConnectU";
```

```
  // Spalten
  int iTassen,
      iEintraege;

  String sMitarbeiter     = null,
         sWochentag       = null,
         sVollerWochentag = null,
         sKaffeesorte     = null;


  public PrepareAlter() {
    // PropertyResourceBundle lesen
    try  {
      rbConnect = ResourceBundle.getBundle( srbName );

      sDriver   = rbConnect.getString( sDriverKey );
      sPassword = rbConnect.getString( sPasswordKey );
      sURL      = rbConnect.getString( sURLKey );
      sUserID = rbConnect.getString( sUserIDKey );
    }  catch( MissingResourceException mre )  {
      System.err.println("ResourceBundle Problem beim Lesen von " +
                         srbName + ", Programmabbruch." );
      System.err.println("Programmfehler: "+ mre.getMessage() );
      return; // exit on error
    }
    // versuche den JDBC Treiber zu laden mit newInstance
    try  {
        Class.forName( sDriver ).newInstance();
    }  catch( Exception e ) {
        System.err.println( "Der Treiber konnte nicht geladen werden.");
      return;
    } // end catch

    try  {
      con = DriverManager.getConnection ( sURL,
                                          sUserID,
                                          sPassword);
      stmt = con.createStatement();
    }  catch ( SQLException SQLe) {
      ReportSQLError( SQLe );
      if( con != null)  {
        try { con.close(); } catch( Exception e ) {}
      }
      return;
    } // end catch

   // eine neue VollerWochentag Spalte hinzufügen
    try  {

// TODO ALTER TABLE und Ausgabe

// ENDTODO

      // prepareStatements

// TODO prepareStatement

// ENDTODO

      // Lade VolleWochentage
// TODO Laden der Daten für die vollen Wochentagnamen

// ENDTODO

      // Update "JustJoe" auf "Jus'Joe"
// TODO  Update


// ENDTODO
```

```
      // Ausgabe der Tabelle

// TODO Auswertung für KaffeeListe

// ENDTODO


    } catch ( SQLException SQLe) {
      ReportSQLError( SQLe );
    } catch (Exception e) { e.printStackTrace(); }
      finally {
        try { stmt.close(); } catch( Exception e ) {}

      if( pstmt1 != null) {
        try { pstmt1.close(); } catch( Exception e ) {}
      }

      if( pstmt2 != null) {
          try { pstmt2.close(); }  catch( Exception e ) {}
      }

      try { con.close(); } catch( Exception e ) {}
    } // end finally clause

  } // end Konstruktor

  public void ReportSQLError( SQLException SQLe ) {
      System.err.println( "Problem:" );
      System.err.println( SQLe.getMessage() );
      System.err.println( "SQL State: " +
                          SQLe.getSQLState() );

  }  // end ReportSQLError


  public static void main (String args[])  {
    new PrepareAlter();
  } // end main
} // end class PrepareAlter
```

1.1.5.1.4. Aufgaben

1) Bei dieser Applikation kann das alte ResourceBundle eingesetzt werden.
   Es sollen zwei Arrays angelegt werden, je eines für die Kurznamen und eines für die vollen Wochentagnamen. Zudem werden zwei PreparedStatement eingesetzt werden.
   Alle diese Definitionen finden Sie bereits im Rahmenprogramm.

   Ihre Aufgabe:
   Definieren Sie eine Zeichenkette, welche alle Spalten der Tabelle auswählt und ausgeben kann.
   In zwei weiteren Zeichenketten müssen Sie die Update Anweisungen angeben.
   In der ersten wird zu jedem Kurzzeichen ein voller Wochentagname eingefügt, also : zu jedem Wert des einen Arrays oben, wird ein Wert das zweiten Arrays beigestellt.
   In zweiten zweiten wird die Kaffeesorte von "JustJoe" auf "Jus'Joe" abgeändert.

2) Im Rahmenprogramm sehen Sie weitere Anweisungen, um mit der Datenbank eine Verbindung aufzubauen (Datenbank URL, Benutzername, Passwort) sowie das Laden des Treibers und das Kreieren eines Statement.

   Ihre Aufgabe:
   fügen Sie eine neue Spalte in die Tabelle KaffeeListe ein:
   ALTER TABLE KaffeeListe
   ADD COLUMN VollerWochentag VARCHAR(12)
   (alle Wochentage müssen Platz haben, sonst wird eine SQLException geworfen)
   und geben Sie bei erfolgreicher Mutation eine kurze Meldung aus.

3) Bereiten Sie beide PreparedStatement vor

4) Nun müssen Sie die Daten in die neue Spalte einfügen.
   Im Programm sind dafür zwei Spalten vorgesehen:
   `asWochentag und`
   `asVollerWochentag`
   Zudem finden Sie einen Index für diese zwei Arrays: `int idx;`

   Ihre Aufgabe:
   Schreiben Sie eine Schleife, damit die Elemente des Arrays asVollerWochentag passend eingefügt wird, mit pstm1.

   Der relevante Teil der UPDATE Anweisung sieht folgendermassen aus:
   `SET VollerWochentag = ? WHERE Wochentag = ?`

   Geben Sie zudem aus, wieviele Zeilen Sie verändert haben.

5) Nun müssen wir die Mutation der Kaffeesorte noch durchführen.
   Dazu verwenden wir pstm2. Der relevante Teil der UPDATE Anweisung sieht folgendermassen aus:
   SET Kaffeesorte = ? WHERE Kaffeesorte = ?
   Geben Sie auch hier die Anzahl betroffener Zeilen in der Tabelle aus.

6) Erstellen Sie nun einen Bericht, der alle Daten der Tabelle enthält, insbesondere alle neu eingefügten.

1.1.5.1.5. Lösungshinweise

1) Die Zeichenketten:

```
 "SELECT * FROM KaffeeListe",
 "UPDATE KaffeeListe SET VollerWochentag= ? WHERE Wochentag= ?",
"UPDATE KaffeeListe SET Kaffeesorte = ? WHERE Kaffeesorte = ?",
```

2) Das Programmfragment:

```
stmt.executeUpdate( "ALTER TABLE KaffeeListe "  +
     "ADD COLUMN LDOW VARCHAR (12)" );
System.out.println("Spalte VollerWochentag wurde der Tabelle KaffeeListe
hinzugefuegt");
```

3) Programmfragment:

```
pstmt1 = con.prepareStatement( sUpdate1 );
pstmt2 = con.prepareStatement( sUpdate2 );
for ( ; ndx < asDOW.length; ndx++ )      {
   pstmt1.setString( 1, asLDOW[ndx] );
   pstmt1.setString( 2, asDOW[ndx] );
   iRowCount += pstmt1.executeUpdate();
}
System.out.println( iRowCount + " Zeilen wurden mutiert.");
```

4) Programmfragment:

```
pstmt2.setString( 1, "Jus'Joe" );
pstmt2.setString( 2, "JustJoe" );
iRowCount = pstmt2.executeUpdate();
System.out.println( iRowCount + " Zeilen auf Jus'Joe mutiert.");
```

5) Programmfragment:

```
rs = stmt.executeQuery( sQuery );
while(rs.next())     {
   iEintrag = rs.getInt("Eintrag");
   sMitarbeiter = rs.getString("Mitarbeiter");
   sWochentag = rs.getString("Wochentag");
   iTassen = rs.getInt("Tassen");
   sKaffeesorte = rs.getString("Kaffeesorte");
   sVollerWochentag = rs.getString("VollerWochentag");
   // Mitarbeiterauswertung
   System.out.println( iEintrag    + ",\t" +
               sMitarbeiter + ",\t" +
                     sWochentag      + ",\t" +
               iTassen     + ",\t" +
               sKaffeesorte     + ",\t" +
               sVollerWochentag);
 } // end while
```

1.1.5.1.6. Musterlösung

ConnectXP.properties:

```
#PropertiesResourceBundle für Connection Properties
CSDriver=COM.cloudscape.core.RmiJdbcDriver
CSURL=jdbc:cloudscape:rmi:XP_Team
CSUserID=sa
CSPassword=admin
```

PrepareAlter.java:

```
package tabellenmutationen;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */

import java.sql.*;
import java.util.*;

public class PrepareAlter
{
  static String[] asWochentag =
  {
    "Mo",
    "Di",
    "Mi",
    "Do",
    "Fr",
    "Sa",
    "So"
  };

  static String[] asVollerWochentag =
  {
    "Montag",
    "Dienstag",
    "Mittwoch",
    "Donnerstag",
    "Freitag",
    "Samstag",
    "Sonntag"
  };

  Connection con;
  int ndx,
      iRowCount;
  PreparedStatement pstmt1 = null,
                    pstmt2 = null;
  ResourceBundle rbConnect;
  ResultSet rs;
  Statement stmt;
  String sDriver,
         sDriverKey = "CSDriver",
         sPassword,
         sPasswordKey ="CSPassword",
         sQuery = "SELECT * FROM KaffeeListe",
         srbName = "ConnectXP",
         sUpdate1="UPDATE KaffeeListe SET VollerWochentag = ? WHERE
                                                    Wochentag = ?",
         sUpdate2="UPDATE KaffeeListe SET Kaffeesorte = ? WHERE Kaffeesorte
                                                      = ?",
         sURL,
         sURLKey="CSURL",
```

```
        sUserID,
        sUserIDKey = "CSUserID";

  // Spalten
  int iTassen,
      iEintrag;

  String sMitarbeiter     = null,
         sWochentag       = null,
         sVollerWochentag = null,
         sKaffeesorte     = null;


  public PrepareAlter() {
    // PropertyResourceBundle lesen
    try  {
      rbConnect = ResourceBundle.getBundle( srbName );

      sDriver   = rbConnect.getString( sDriverKey );
      sPassword = rbConnect.getString( sPasswordKey );
      sURL      = rbConnect.getString( sURLKey );
      sUserID = rbConnect.getString( sUserIDKey );
    }  catch( MissingResourceException mre )  {
      System.err.println("ResourceBundle Problem beim Lesen von " +
                         srbName + ", Programmabbruch." );
      System.err.println("Programmfehler: "+ mre.getMessage() );
      return; // exit on error
    }
    // versuche den JDBC Treiber zu laden mit newInstance
    try  {
        Class.forName( sDriver ).newInstance();
    }  catch( Exception e ) {
        System.err.println( "Der Treiber konnte nicht geladen werden.");
      return;
    } // end catch

    try  {
      con = DriverManager.getConnection ( sURL,
                                          sUserID,
                                          sPassword);
      stmt = con.createStatement();
    }  catch ( SQLException SQLe) {
      ReportSQLError( SQLe );
      if( con != null)  {
        try { con.close(); } catch( Exception e ) {}
      }
      return;
    } // end catch

   // eine neue VollerWochentag Spalte hinzufügen
    try  {
      stmt.executeUpdate( "ALTER TABLE KaffeeListe "  +
                          "ADD COLUMN VollerWochentag VARCHAR (12)" );
      System.out.println("Spalte VollerWochentag wurde in die Tabelle
                              KaffeeListe eingefuegt.");

      // prepareStatement
      pstmt1 = con.prepareStatement( sUpdate1 );
      pstmt2 = con.prepareStatement( sUpdate2 );

      // Spalte VollerWochentag ergänzen
      for ( ; ndx < asWochentag.length; ndx++ )   {
        pstmt1.setString( 1, asVollerWochentag[ndx] );
        pstmt1.setString( 2, asWochentag[ndx] );
        iRowCount += pstmt1.executeUpdate();
      }

      System.out.println( iRowCount + " Zeilen wurden in die Spalte
                  VollerWochentag in der Tabelle KaffeeListe eingefuegt.");
```

```
      // Mutation von "JustJoe" Werten auf "Jus'Joe"
      pstmt2.setString( 1, "Jus'Joe" );
      pstmt2.setString( 2, "JustJoe" );

      iRowCount = pstmt2.executeUpdate();
      System.out.println("In "+ iRowCount +" Zeilen der Tabelle KaffeeListe
                             wurde JustJoe durch Jus'Joe ersetzt.");

      // Tabellen Daten ausgeben
      rs = stmt.executeQuery( sQuery );

      // für jede Datenzeile
      while(rs.next())     {
        iEintrag = rs.getInt("Eintrag");
        sMitarbeiter = rs.getString("Mitarbeiter");
        sWochentag = rs.getString("Wochentag");
        iTassen = rs.getInt("Tassen");
        sKaffeesorte = rs.getString("Kaffeesorte");
        sVollerWochentag = rs.getString("VollerWochentag");

        // Auswertung pro Mitarbeiter
        System.out.println( iEintrag    + ",\t" +
                            sMitarbeiter + ",\t" +
                            sWochentag      + ",\t" +
                            iTassen     + ",\t" +
                            sKaffeesorte     + ",\t" +
                            sVollerWochentag );
      } // end while
    } catch ( SQLException SQLe) {
      ReportSQLError( SQLe );
    } catch (Exception e) { e.printStackTrace(); }
      finally {
        try { stmt.close(); } catch( Exception e ) {}

      if( pstmt1 != null) {
        try { pstmt1.close(); } catch( Exception e ) {}
      }

      if( pstmt2 != null) {
          try { pstmt2.close(); }  catch( Exception e ) {}
      }

      try { con.close(); } catch( Exception e ) {}
    } // end finally clause

  } // end Konstruktor

  public void ReportSQLError( SQLException SQLe ) {
      System.err.println( "Problem:" );
      System.err.println( SQLe.getMessage() );
      System.err.println( "SQL State: " +
                           SQLe.getSQLState() );

  }  // end ReportSQLError


  public static void main (String args[])  {
    new PrepareAlter();
  } // end main
} // end class PrepareAlter
```

1.1.5.1.7. Demonstration

Falls Sie noch einmal sehen wollen, welche Daten die Tabelle KaffeeListe enthält, dann sollten Sie eine der bereits gelösten Aufgaben nehemn und diese Tabelle ausgeben.

```
Spalte VollerWochentag wurde in die Tabelle KaffeeListe eingefuegt

15 Zeilen wurden in die Spalte VollerWochentag in der Tabelle KaffeeListe
eingefuegt.

In 2 Zeilen der Tabelle KaffeeListe wurde JustJoe durch Jus'Joe ersetzt.

1,    Hans,       Mo,   1,    Jus'Joe,    Montag
2,    JS,         Mo,   1,    Cappuccino, Montag
3,    Marie,      Mo,   2,    CaffeMocha, Montag
4,    Anne,       Di,   8,    Cappuccino, Dienstag
5,    Holley,     Di,   2,    MoJava,     Dienstag
6,    jDuke,      Di,   3,    Cappuccino, Dienstag
7,    Marie,      Mi,   4,    Espresso,   Mittwoch
8,    JS,         Mi,   4,    Latte,      Mittwoch
9,    Alex,       Do,   3,    Cappuccino, Donnerstag
10,   Peter,      Do,   1,    Cappuccino, Donnerstag
11,   jDuke,      Do,   4,    Jus'Joe,    Donnerstag
12,   JS,         Fr,   9,    Espresso,   Freitag
13,   Hans,       Fr,   3,    Cappuccino, Freitag
14,   Beth,       Fr,   2,    Cappuccino, Freitag
15,   jDuke,      Fr,   1,    Latte,      Freitag
```

1.1.6. Java-SQL Typen Äquivalenz

JDBC definiert Datentypen, welche auf generische SQL Datentypen ebgebildet werden. In der Regel ist es sehr einfach die Java Datentypen aus den SQL Datentypen herzuleiten. Die folgende Tabelle zeigt die normalen ResultSet Methoden, mit denen die entsprechenden SQL Datentypen gelesen werden können. Die setXXX() Methoden sind analog aufgebaut:

SQL Datentyp	Java Methode
BIGINT	getLong()
BINARY	getBytes()
BIT	getBoolean()
CHAR	getString()
DATE	getDate()
DECIMAL	getBigDecimal()
DOUBLE	getDouble()
FLOAT	getDouble()
INTEGER	getInt()
LONGVARBINARY	getBytes()
LONGVARCHAR	getString()
NUMERIC	getBigDecimal()
OTHER	getObject()
REAL	getFloat()
SMALLINT	getShort()
TIME	getTime()
TIMESTAMP	getTimestamp()
TINYINT	getByte()
VARBINARY	getBytes()
VARCHAR	getString()

Um Daten auch problemlos darstellen zu können, kann man auch immer die `ResultSet.getString()` Methode einsetzen.

SQL-3 Datentyp	Java Methode
ARRAY	getArray()
BLOB	getBlob()
CLOB	getClob()
DISTINCT	get*`UnderlyingType`*()
REF	getRef()
STRUCT	*`(castToStruct)`*getObject()
JAVA_OBJECT	*`(castToObjectType)`*getObject()

`ResultSet.getObject()` kann ebenfalls in jedem Fall eingesetzt werden.

1.1.7. JDBC Exception Typen und Exception Handling

Exceptions sind ein wichtiges Konzept in Java. Auch JDBC unterstützt Exceptions und es lohnt sich diese zu kennen. Sie finden in vielen Foren im Internet Fragen zu diesem Thema.

Wichtig ist, dass Sie sich beim Entwickeln von Anwendungen, speziell Datenbankanwendungen, vergewissern, dass die Qualität stimmt. Oder möchten Sie, dass Ihre Buchungen auf dem Bankkonto nicht korrekt sind?

Hier geht es um drei Level :
- SQL Exceptions
- SQL Warnings
- Data Truncations, also Fälle, in denen die Daten im vorgesehenen Datentyp oder dessen Länge (Anzahl Zeichen) nicht Platz finden.

1.1.7.1. SQL Exceptions

Viele der im Package java.sql Package vorhandenen Methoden werfen eine SQLException. Diese müssen mit `try{ } catch() { }` Blöchen abgefangen werden, also genauso wie irgend eine andere Exception.

Damit werden beispielsweise SQL Fehler oder Treiber Fehler erkannt und gemeldet.

Neben der `getMessage()` Methode aus der Klasse Throwable kennt SQLException weitere Methoden, welche weitere Informationen liefern:

- `getSQLState()` liefert eine Zustandskennzeichnung gemäss der X/Open SQL Spezifikation. In den DBMS Manuals stehen normalerweise Erklärungen, was das Werfen einer dieser Exceptions im konkreten Fall / DBMS bedeutet.
- `getErrorCode()` liefert einen DBMS Anbieter spezifischen Fehlercode.
- `getNextException()` zeigt die nächste SQLException oder null falls es keine weiteren mehr gibt.
  Da bei einem Datenbankzugriff sehr viel schief laufen kann, können Sie damit eine umfassende Liste der möglichen Fehler erstellen.
- `setNextException()` gestattet dem Programmierer weitere SQLExceptions einer Exception Kette hinzuzufügen.

Diese Methoden funktionieren genau so wie Sie erwarten können. Typischerweise hätten Sie folgende Codefragmente in Ihren Programmen:

```
try    {
   //       DB Code
} catch (  SQLException SQLe)    {
  while( SQLe != null)      {
    // doHandling
    SQLe = SQLe.getNextException();
  }
} // end catch
```

Tip: falls Sie Wert darauf legen bei SQL Fehlern die SQL Anweisung sehen zu können, sollten Sie jeweils die Anweisung `Connection.nativeSQL(ihrQueryString)` in die Behandlung der Exception (catch) einbauen.

1.1.7.2. SQL Warnings

Eine SQLWarning ist eine Unterklasse der SQLException Klasse. Sie wirft aber keine Exception, sondern es liegt am Programmierer diese Warnungen explizit abzufragen und anzuzeigen.

`Connections, Statements` und `ResultSets` kennen alle eine `getWarnings()` Methode. Es gibt auch eine `clearWarnings()`Methode, um ein mehrfaches Lesen der gleichen Warnung zu vermeiden. Die `SQLWarning` Klasse selber kennt zwei Methoden `getNextWarning()` und `setNextWarning()`.

Eine SQLWarning gleicht einer traditionellen Compiler Warnung: irgend etwas stimmt nicht, aber der Effekt ist nicht so, dass deswegen das Programm nichtlauffähig wäre. Es hängt schlicht von Kontext ab, ob man etwas unternehmen muss oder einfach darüber hinweg sehen kann.

Statements löschen die Warnungen automatisch, sobald sie erneut ausgeführt werden. ResultSets löschen die Warnung immer dann, wenn eine neue Zeile gelesen wird. Connections können sich so oder so verhalten. Die Dokumentation sagt nichts darüber aus. Sicherer sind Sie, falls Sie in diesem Fall `clearWarnings()` Methode anwenden. sobald Sie die Warnung, die Sie erhalten, analysiert haben.

Ein typisches Codefragment könnte folgendermassen aussehen:

```
try    {
  ...

  stmt = con.createStatement();
  sqlw = con.getWarnings();
  while( sqlw != null) {
    // handleSQLWarnings

    sqlw = sqlw.getNextWarning();
  }
  con.clearWarnings();

  stmt.executeUpdate( sUpdate );
  sqlw = stmt.getWarnings();
  while( sqlw != null) {
    // handleSQLWarnings

    sqlw = sqlw.getNextWarning();
  }
} // end try
catch ( SQLException SQLe)  {
  ...
} // end catch
```

1.1.7.3. Data Truncation

DataTruncation entspricht in etwa einer SQL Warnung. Falls eine solche Ausnahme beim Lesen auftritt, wird eine Warnung gesetzt. Falls sie in einem write/update Vorgang auftritt, wird eine Exception geworfen mit SQLState of 01004.

Data Truncation bedeutet, dass weniger Information gelesen oder geschrieben wird, als verlangt wird. *Einige* Datenbanktreiber akzeptieren Daten, welche grösser sind, als der vorhandene Platz in der entsprechenden Spalte, im entsprechenden Zielfeld. Aber diese zu grossen Daten müssen irgendwie gekürzt werden, um abgespeichert zu werden.

Sie können mit den folgenden Methoden Informationen über die Daten und Data Truncation erfahren: `getDataSize(), getIndex(), getParameter(), getRead()` und `getTransferSize()`.

1.1.7.4. Einfache Beispielausgaben

Sie können sehr leicht Warnings oder Fehler in Ihren Programmen generieren, vermutlich haben Sie dies bereits mehrfach.

Versuchen Sie folgende Fälle:

1) Löschen von Daten, die es nicht gibt.
2) Einfügen von Datensätzen mit einem Schlüssel, der bereits vergeben ist.
3) UPDATE einer Zeile die nicht vorhanden ist.
4) DROP einer Tabelle, welche nicht vorhanden ist
5) UPDATE einer Zeile, mit beispielsweise einer Zeichenkette, die länger ist als bei der Definition der Tabelle angegeben.
6) SQL Syntax Fehler (DBMS spezifisch)

1.1.7.5. Übung - Behandlung von SQLExceptions und SQLWarnings

In dieser Übung erweitern wir das Programm aus der ersten Übung und benutzen es, um einige SQLExceptions und Warnings zu produzieren.

1.1.7.6. Lernziele

Nach dem durcharbeiten dieser Übung sollten Sie in der Lage sein:

- einige wichtige Techniken der Fehlerbehandlung und Fehlercodes von JDBC zu kennen.
- JDBC Fehler systematisch zu suchen und zu eliminieren.

1.1.7.7. Szenario

Das Entwicklerteam stellte in Tests fest, dass gelegentlich weniger Zeilen bearbeitet wurden als eigentlich bearbeitet werden sollten.

Im Sinne einer proaktiven, kundenfreundlichen Massnamhe, beschliesst das Entwicklerteam die Fehlerbehandlung zu verbessern und ein Fehler- und Warnungs- System einzubauen.

Um eine Art reproduzierbare Tests zur Verfügung zu haben, werden mehrere SQL Anweisungen als Properties abgespeichert:

```
#PropertiesResourceBundle für XPRunTest Properties
#Test auf ungültige Einträge
#Die einzelnen Testfälle müssen noch jeweils aktiviert werden.
#1=DELETE FROM XP_TEE WHERE Eintrag = 97
#1=INSERT INTO XP_TEE VALUES (25, 'Rosa', 'Petite', 'Blue')
#1=UPDATE XP_TEE SET TFarbe = 'Black' WHERE TFarbe = 'Appetite'
#1=DROP TABLE XPExistiertNicht
#1=UPDATE XP_TEE SET TGroesse = 'Small Doppelganger' WHERE TGroesse = 'Small'
#1=UPDATE XP_TEE SET TGroesse = 'Small ' WHERE TGroesse = 'Small'
#1=DROP TSBLE BadSQL )
```

Die neue Version des Programms soll die Fehlerfälle testen. Gestartet wird das Programm als:

```
java XPRunTest XPRunTests
```

Das Programm enthält zwei zusätzliche Methoden:

```
  public void handleSQLExceptions( SQLException SQLe,
                                   String       s,
                                   String       sSQL )
```

und

```
  public void handleSQLWarnings( SQLWarning SQLw,
                                 String        s,
                                 String     sSQL )
```

Zusätzlich werden wir die bereits existierende Methode:

```
  public void reportSQLError( SQLException SQLe,
                              String       s )
```

umbenennen und ein zusätzliches Argument übergeben:

```
  public void reportSQLExceptions( SQLException SQLe,
                                   String       s,
```

```
                                String       sSQL )
```

Dieser Level der Überprüfung reicht im Allgemeinen auch bei produktiven Systemen. Weitergehende Tests eines produktiven Systems sind denkbar, aber von der jeweiligen Applikation abhängig.

1.1.7.8. Voraussetzungen

Sie haben die Übung zum Thema Updates abgeschlossen oder kennen die Musterlösung

1.1.7.9. Rahmenprogramm

Für diese Übung verwenden wir das übliche ResourceBundle für den Verbindungsaufbau.
ConnectXP.properties:

```
#PropertiesResourceBundle für Connection Properties
CSDriver=COM.cloudscape.core.RmiJdbcDriver
CSURL=jdbc:cloudscape:rmi:XP_Team
CSUserID=sa
CSPassword=admin
```

Daneben verwenden wir eine Testsuite, wie oben besprochen, in Form eines ResourceBundles.
XPRunTests.properties:

```
#PropertiesResourceBundle für XPRun2 Properties
#Test auf ungültige Einträge
#Die einzelnen Testfälle müssen noch jeweils aktiviert werden.
#1=DELETE FROM XP_TEE WHERE Eintrag = 97
#1=INSERT INTO XP_TEE VALUES (25, 'Rosa', 'Petite', 'Blue')
#1=UPDATE XP_TEE SET TFarbe = 'Black' WHERE TFarbe = 'Appetite'
#1=DROP TABLE XPExistiertNicht
#1=UPDATE XP_TEE SET TGroesse = 'Small Doppelganger' WHERE TGroesse =
'Small'
#1=UPDATE XP_TEE SET TGroesse = 'Small ' WHERE TGroesse = 'Small'
#1=DROP TSBLE BadSQL )
```

Und schliesslich steht Ihnen ein Java Rahmenprogramm zur Verfügung.
XPRunTest.java:

```
import java.sql.*;
import java.util.*;

public class XPRunTest
{
  Connection con;
  ResourceBundle rb;
  SQLWarning sqlw;
  Statement stmt;
  String sDriver,
         sDriverKey = "CSDriver",
         sKey,
         sPassword,
         sPasswordKey ="CSPassword",
         sUpdate,
         srbName = "ConnectXP",
         srbUpdate,
         sURL,
         sURLKey="CSURL",
         sUserID,
         sUserIDKey = "CSUserID";



  public XPRunTest() throws MissingResourceException,
                            ClassNotFoundException,
```

```
                        InstantiationException,
                        IllegalAccessException {

    // PropertyResourceBundle für den Verbindungsaufbau lesen
    rb = ResourceBundle.getBundle( srbName );

    sDriver   = rb.getString( sDriverKey );
    sPassword = rb.getString( sPasswordKey );
    sURL      = rb.getString( sURLKey );
    sUserID   = rb.getString( sUserIDKey );

    // JDBC Treiber laden
    // mit newInstance
    Class.forName( sDriver ).newInstance();

  } // end Konstruktor


  public void doUpdate( String sargRBName )
              throws MissingResourceException,
                     SQLException {
    int iProcessed = 0,
        iProcessedCount =0;

    // bestimme Connection und Statement Objekte
    try     {
      con = DriverManager.getConnection ( sURL,
                                          sUserID,
                                          sPassword);
      stmt = con.createStatement();

// TODO Einfügen des Warnungsaufrufes

// ENDTODO


    } // end try
    catch ( SQLException SQLe)
    {

// TODO Aufruf von handleSQLExceptions()

// ENDTODO

      if( con != null)
      {
        try { con.close(); }
        catch( Exception e ) {}
      }
      // rethrow Exception
      throw SQLe;

    } // end catch

    try
    {
      Enumeration e = null;
      int iCount = 0;

      // PropertyResourceBundle mit Tests lesen
      // für die SQL Update Anweisungen
      rb = ResourceBundle.getBundle( sargRBName );
      e = rb.getKeys();
```

```
      // Count Keys - Keys starten bei 1.
      for( ; e.hasMoreElements(); iCount++ )
      {
        (e.nextElement());
      };

      // verwende <= weil Keys mit 1 starten
      for( int i = 1; i <= iCount; i++ )
      {
        sKey = "" + i;
        sUpdate = rb.getString( sKey );

        iProcessed = stmt.executeUpdate( sUpdate );

// TODO EInfügen von Code if iProcessed==0 und else

// ENDTODO


// TODO EInfügen von stmtWarnungen

// ENDTODO

      } // end for

    } // end try
    catch( MissingResourceException mre )
    {
      System.err.println(
                "ResourceBundle Problem wegen " +
                 sargRBName + ", Programm wird abgebrochen." );
      System.err.println("Programmfehler: " +
                         mre.getMessage() );
      throw mre;
    }    catch ( SQLException SQLe)    {

// TOD Aufruf von handleSQLExceptions()

// ENDTODO

      // rethrow Exception
      throw SQLe;
    }    finally    {

// TODO Ausgabe der verarbeiteten Datensätze / Zeilen

// ENDTODO
      try { stmt.close(); }  catch( Exception e ) {}
      try { con.close(); }  catch( Exception e ) {}
    } // end finally clause

  } // end doUpdate

  public void handleSQLExceptions( SQLException SQLe,
                                   String       s,
                                   String       sSQL )
  {
    boolean bFirstPass = true;

// TODO Aufruf von reportSQLExceptions()
// für alle Exceptions

// ENDTODO

  } // end handleSQLExceptions
```

```
  public void handleSQLWarnings( SQLWarning SQLw,
                                 String        s,
                                 String     sSQL )
  {
    boolean bFirstPass = true;

    if( s == null ) { s = "SQLWarning:";  }

// TODO AUfruf von reportSQLExceptions()
// für alle Warnungen

// ENDTODO


  } // end handleSQLWarnings


  public void reportSQLExceptions( SQLException SQLe,
                                   String       s,
                                   String       sSQL )
  {
    String sSQLState = null;

// TODO Reporting
// inclusive DataTruncation

// ENDTODO


  } // end reportSQLExceptions


  public static void main (String args[])
  {
    boolean bContinue = true;
    XPRunTest xpApp = null;

    if( args.length != 1 )
    {
      System.err.println("Usage: " +
        "java XPRunTest <SQLUpdateResourceBundleName>" );
      return;
    }

    try
    {
      xpApp = new XPRunTest();
    }
    catch( Exception e )
    {
      System.err.println("Constructor Exception: " +
                         e.getMessage() );
      bContinue = false;
    }

    if( bContinue )
    {
      try
      {
        xpApp.doUpdate( args[0] );
      }
      catch( Exception e ) {}
    }

  } // end main

} // end class XPRunTest
```

1.1.7.10. Aufgaben

1) Ergänzen Sie die Methode `doUpdate()` durch Programmcode, mit welchem Sie allfällige Probleme beim Verbindungsaufbau anzeigen können.
   Verwenden Sie dazu die Variable sqlw und

   ```
   stmt = con.createStatement();
   ```

   Falls Warnungen auftraten, rufen Sie die `handleSQLWarnings()` Methode auf. Löschen Sie anschliessend die Warnung.

2) Ergänzen Sie die Methode `doUpdate()` im ersten `catch( SQLException SQLe)` Block. Fügen Sie Programmcode ein, mit dem Sie handleSQLExceptions() verwenden können.

3) In der `doUpdate()` Methode nach der Anweisung

   ```
   iProcessed = stmt.executeUpdate( sUpdate )
   ```

   geben Sie aus, wieviele Zeilen bearbeitet wurden und fügen Sie eine spezielle Meldung hinzu für den Fall, dass keine Zeile bearbeitet wurde.

   `iProcessed` enthält die Information darüber, wieviele Zeilen verarbeitet wurden, `iProcessedCount` enthält das Total.

4) In der Methode `doUpdate()` nach dem obigen Code, sollten Sie das programm ergänzen, um Warnungen abzufangen, auch hier mit Hilfe der SQLW Variable.

5) Im anschliessenden `catch( SQLException SQLe)` Block muss die Methode `handleSQLExceptions()` aufgerufen werden.

6) Ergänzen Sie das Programm im anschliessenden `finally` Block durch eine Ausgabe der insgesamt verarbeiteten Zeilen.

7) Ergänzen Sie das Programm in `handleSQLExceptions()`, so dass für jede Exception die Methode `reportSQLExceptions()` aufgerufen wird.

   ```
   while( SQLe != null )
   ```
   ...
   und
   ```
   SQLe = SQLe.getNextException().
   ```
   Nach dem ersten Aufruf sollte nur noch `SQLe` oder `null` an `reportSQLExceptions()` übergeben werden.

8) Ergänzen Sie `handleSQLWarnings()` so dass `reportSQLExceptions()` bei jeder gelesenen Warnung ausgegeben wird, mit

   ```
   while( SQLw != null )
   ```
   und
   ```
   SQLw = SQLw.getNextWarning().
   ```

   Nach dem ersten Aufruf sollte nur noch SQLw und null an `reportSQLExceptions()`übergeben werden.

9) Fügen Sie in `reportSQLExceptions()` Programmcode ein, um einen Report zu generieren.
   Falls `sSQL` nicht `null` ist, dann geben Sie auch native SQL aus.
   Falls der vom Programm an die DBMS gesandte String / Text nicht leer war, dann geben Sie auch diesen aus:

   `SQLState`
   und alle verfügbaren `SQLException` Informationen.
   Falls SQLState "01004" ist, dann sollen auch die DataTruncation Informationen ausgegeben werden.

1.1.7.11. Lösungshinweise

Zu Aufgabe 1:

```
if( sqlw != null )  {
        handleSQLWarnings( sqlw,
                           "Warnung beim Verbindungsaufbau: ",
                            null );
        con.clearWarnings();
      }
```

Zu Aufgabe 2:

```
handleSQLExceptions(SQLe,"Probleme beim Verbindungsaufbau zu " + sURL + :",
            null);
```

Zu Aufgabe 3:

```
if( iProcessed == 0 ){
   System.err.println( "XPRunTest konnte einen Test nicht durchführen:\n" +
                sUpdate + "." );
   System.err.println( "Ignorieren Sie die obige Meldung " +
               "falls Sie lediglich DDL Anweisungen ausführten.\n" );
}  else {
   iProcessedCount += iProcessed;
}
```

Zu Aufgabe 4:

```
sqlw = stmt.getWarnings();
if( sqlw != null ){
   handleSQLWarnings( sqlw, "Warnung wegen einer Anweisung: ",
                                sUpdate );
}
```

Zu Aufgabe 5:

```
handleSQLExceptions(SQLe, "Probleme mit executeUpdate:", sUpdate);
```

Zu Aufgabe 6:

```
System.out.println( iProcessedCount +" Zeilen verarbeitet." );
```

Zu Aufgabe 7:

```
while( SQLe != null) {
   reportSQLExceptions( SQLe, s, sSQL );
   if( bFirstPass )   {
        s = sSQL = null;
        bFirstPass = false;
   }
   SQLe = SQLe.getNextException();
}
```

Zu Aufgabe 8:

```
while( SQLe != null) {
   reportSQLExceptions( SQLe, s, sSQL );
   if( bFirstPass ) {
      s = sSQL = null;
      bFirstPass = false;
   }

   SQLe = SQLe.getNextException();
}
```

Zu Aufgabe 9:

```
if( sSQL != null ) {
        // Report des Problem SQL
        try {
           System.err.println(
           con.nativeSQL( sUpdate ) );
        } catch( Exception e) { /* alles klar? */ }
}

if( s != null ) { // Report Programm Text
        System.err.println( s );
}
sSQLState = SQLe.getSQLState();

// Report Error Information
System.err.println( SQLe.getMessage() );
System.err.println( "SQL State: " + sSQLState );
System.err.println( "Vendor Error Code: " + SQLe.getErrorCode() );

// check Data Truncation
if( sSQLState.equals( "01004") )  {
        DataTruncation dt = (DataTruncation)(SQLe);
        System.err.println( "Data Size: " +dt.getDataSize() );
        System.err.println( "Transfer Size: " + dt.getTransferSize() );
        System.err.println( "Index des  " +
                ( dt.getParameter()
                                ? "Parameter "
                                : "Spalte " ) +
                           "war " +
                            dt.getIndex() + "." );

}  // end if Data Truncation
```

1.1.7.12. Musterlösung

Property Dateien: siehe oben

XPRunTest.java:

```
package sqlerrorundwarning;

import java.sql.*;
import java.util.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */
public class XPRunTest
{
  Connection con;
  ResourceBundle rb;
  SQLWarning sqlw;
  Statement stmt;
  String sDriver,
         sDriverKey = "CSDriver",
         sKey,
         sPassword,
         sPasswordKey ="CSPassword",
         sUpdate,
         srbName = "ConnectXP",
         srbUpdate,
         sURL,
         sURLKey="CSURL",
         sUserID,
         sUserIDKey = "CSUserID";


  public XPRunTest() throws MissingResourceException,
                            ClassNotFoundException,
                            InstantiationException,
                            IllegalAccessException {

    // PropertyResourceBundle für den Verbindungsaufbau lesen
    rb = ResourceBundle.getBundle( srbName );

    sDriver   = rb.getString( sDriverKey );
    sPassword = rb.getString( sPasswordKey );
    sURL      = rb.getString( sURLKey );
    sUserID   = rb.getString( sUserIDKey );

    // JDBC Treiber laden
    // mit newInstance
    Class.forName( sDriver ).newInstance();

  } // end Konstruktor


  public void doUpdate( String sargRBName )
              throws MissingResourceException,
                     SQLException {
    int iProcessed = 0,
        iProcessedCount =0;

    // bestimme Connection und Statement Objekte
    try     {
      con = DriverManager.getConnection ( sURL,
                                          sUserID,
                                          sPassword);
      stmt = con.createStatement();
      sqlw = con.getWarnings();
```

```
  if( sqlw != null )  {
    handleSQLWarnings( sqlw,
                       "Warnung beim Verbindungsaufbau: ",
                        null );
    con.clearWarnings();
  }

} // end try
catch ( SQLException SQLe)
{
  handleSQLExceptions(
     SQLe,
    "Probleme beim Verbindungsaufbau zu " + sURL + ":",
     null            );

  if( con != null)
  {
    try { con.close(); }
    catch( Exception e ) {}
  }
  // rethrow Exception
  throw SQLe;

} // end catch

try
{
  Enumeration e = null;
  int iCount = 0;

  // PropertyResourceBundle mit Tests lesen
  // für die SQL Update Anweisungen
  rb = ResourceBundle.getBundle( sargRBName );
  e = rb.getKeys();

  // Count Keys - Keys starten bei 1.
  for( ; e.hasMoreElements(); iCount++ )
  {
    (e.nextElement());
  };

  // verwende <= weil Keys mit 1 starten
  for( int i = 1; i <= iCount; i++ )
  {
    sKey = "" + i;
    sUpdate = rb.getString( sKey );

    iProcessed = stmt.executeUpdate( sUpdate );
    if( iProcessed == 0 )
    {
      System.err.println(
        "XPRunTest konnte einen Test nicht durchführen:\n" +
         sUpdate + "." );
      System.err.println( "Ignorieren Sie die obige Meldung " +
        "falls Sie lediglich DDL Anweisungen ausführten.\n" );
    }
    else
    {
      iProcessedCount += iProcessed;
    }
    sqlw = stmt.getWarnings();
    if( sqlw != null )
    {
      handleSQLWarnings( sqlw,
                         "Warnung wegen einer Anweisung: ",
                          sUpdate );
    }
  } // end for

} // end try
```

```
    catch( MissingResourceException mre )
    {
      System.err.println(
                 "ResourceBundle Problem wegen " +
                  sargRBName + ", Programm wird abgebrochen." );
      System.err.println("Programmfehler: " +
                         mre.getMessage() );
      throw mre;
    }
    catch ( SQLException SQLe)
    {
      handleSQLExceptions(
         SQLe,
        "Probleme mit executeUpdate:",
         sUpdate         );

      // rethrow Exception
      throw SQLe;
    }
    finally
    {
      System.out.println( iProcessedCount +
                        " Zeilen verarbeitet." );

      try { stmt.close(); }
      catch( Exception e ) {}

      try { con.close(); }
      catch( Exception e ) {}

    } // end finally

  } // end doUpdate


  public void handleSQLExceptions( SQLException SQLe,
                                   String       s,
                                   String       sSQL )
  {
    boolean bFirstPass = true;

    while( SQLe != null)
    {
      reportSQLExceptions( SQLe, s, sSQL );

      if( bFirstPass )
      {
        s = sSQL = null;
        bFirstPass = false;
      }

      SQLe = SQLe.getNextException();
    }

  } // end handleSQLExceptions


  public void handleSQLWarnings( SQLWarning SQLw,
                                 String        s,
                                 String     sSQL )
  {
    boolean bFirstPass = true;

    if( s == null ) { s = "SQLWarning:";  }
```

```
  while( SQLw != null)    {
    reportSQLExceptions( SQLw, s, sSQL );

    if( bFirstPass )      {
      s = sSQL = null;
      bFirstPass = false;
    }

    SQLw = SQLw.getNextWarning();
  }

} // end handleSQLWarnings


public void reportSQLExceptions( SQLException SQLe,
                                 String       s,
                                 String       sSQL )
{
  String sSQLState = null;

  if( sSQL != null )
  { // Report des Problem SQL
    try
    {
      System.err.println(
         con.nativeSQL( sUpdate ) );
    }
    catch( Exception e) { /* alles klar? */ }
  }

  if( s != null )
  { // Report Programm Text
    System.err.println( s );
  }

  sSQLState = SQLe.getSQLState();

  // Report Error Information
  System.err.println( SQLe.getMessage() );
  System.err.println( "SQL State: " +
                      sSQLState );
  System.err.println( "Vendor Error Code: " +
                      SQLe.getErrorCode() );
  // check Data Truncation
  if( sSQLState.equals( "01004") )
  {
    DataTruncation dt = (DataTruncation)(SQLe);

    System.err.println( "Data Size: " +
                        dt.getDataSize() );
    System.err.println( "Transfer Size: " +
                       dt.getTransferSize() );
    System.err.println( "Index des  " +
                        ( dt.getParameter()
                           ? "Parameter "
                           : "Spalte " ) +
                       "war " +
                        dt.getIndex() + "." );

  }  // end if Data Truncation
} // end reportSQLExceptions
```

```
  public static void main (String args[])  {
    boolean bContinue = true;
    XPRunTest xpApp = null;

    if( args.length != 1 )
    {
      System.err.println("Usage: " +
        "java XPRunTest <SQLUpdateResourceBundleName>
                        (ohne properties Extension!)" );
      return;
    }

    try    {
      xpApp = new XPRunTest();
    }    catch( Exception e )    {
      System.err.println("Constructor Exception: " +
                         e.getMessage() );
      bContinue = false;
    }

    if( bContinue )
    {
      try
      {
        xpApp.doUpdate( args[0] );
      }
      catch( Exception e ) {}
    }

  } // end main

} // end class XPRunTest
```

1.1.7.13. Demonstration

Falls Sie dieses Programm starten, die Lösung befindet sich auf dem Server / der CD, dann sollten Sie eine vollständige Fehlermeldung und Warnungen erhalten, sowohl für DDL als auch für DML:

- SQLExceptions,
- SQLWarnings,
- und alle Anweisungen, mit denen keine Zeile bearbeitet wurde.

Beachten Sie, dass ein DDL durchaus keine Zeilen beeinflussen kann. Aber da das Programm nicht weiss von welchem Typ die Anweisung ist, ist es klug eine Warnung auszugeben.

Die wilde Mischung der Ausgabe (DBMS in Englisch und die eigenen in Deutsch zeigt, dass man gescheiter alles in Eglisch schreiben würde!).

Beispiel:

```
DROP TSBLE BadSQL )
Probleme mit executeUpdate:
Syntax error: Encountered "TSBLE" at line 1, column 6.
SQL State: 42X01
Vendor Error Code: 20000
0 Zeilen verarbeitet.

DROP TABLE XPExistiertNicht
Probleme mit executeUpdate:
Table 'XPEXISTIERTNICHT' does not exist.
SQL State: 42X05
Vendor Error Code: 20000
```

```
0 Zeilen verarbeitet.
```

1.1.8. Metadata

Metadaten sind Dazen oder Informationen über andere Daten. JDBC gestattet es dem Programmierer sehr viel Informationen über die Datenbank oder die Tabellen oder ein ResultSet abzufragen. Dies geschieht mit Hilfe der Metadaten Klassen..

1.1.8.1. Datenbank Metadaten

Damit Sie Daten über die Datenbank erfragen können, müssen Sie ein `DatabaseMetaData` Objekt kreieren. Dies geschieht mit folgender Anweisung, vorausgesetzt, die sind mit einer Datenbank verbunden:

```
DatabaseMetaData dbmd = con.getMetaData();
```

Jetzt haben Sie nur noch eine passende Methode zu finden, umdie Daten zu erhalten. Der Haken an der Geschichte ist, dass:

- es ungefährt 150 Methoden in der `DatabaseMetaData` Klasse gibt. Diese alle auch nur einigermassen zu kennen, ist kaum wünschenswert. Aber Sie sollten einfach einmal nachschauen, für was es so Methoden gibt.
- viele dieser Methoden liefern Ihnen ein `ResultSet`.Aus diesem muss dann die gewünschte Information herausgeholt werden.
- einige der Methoden, welche Informationen über die Datenbank oder die Tabellen liefern, verwenden recht eigenartige Namensmuster. Es kann auch sein, dass Sie die Informationen nur erhalten, falls Sie die Gross- / Kleinschreibung beachten., je nach DBMS.

Die wichtigsten Informationen erhalten Sie jedoch in der Regel sehr einfach. Dies umfasst beispielsweise:

- Namen der Datenbank
- Name des Treibers
- Version
- maximale Anzahl möglicher gleichzeitiger Datenbankverbindungen
- SQL Konformität

Viele Programme benötigen überhaupt keine Metadaten. Es kann auch sein, dass eine bestimmte Datenbank eine gewünschte Information nicht liefert, nicht liefern kann.

1.1.8.2. Übungen

Um Übungen zu diesem Thema brauchen wir uns keine Gedanken zu machen: in den bisherigen Übungen hatten wir bereits mehrfach Metadaten abgefragt.

1.1.8.3. ResultSet Metadaten

Um Informationen über ein gegebenes ResultSet zu erhalten, müssen wir ein `ResultSetMetaData` Objekt erhalten. Dies lässt sich auf folgende Art und Weise gewinnen, vorausgesetzt Sie haben eine Vervbindung zur Datenbank und bereits ein `ResultSet` Objekt:

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

Ein `ResultSetMetaData` Objekt ist handlicher als ein `DatabaseMetaData` Objekt: es kennt lediglich 25 Methoden.

Mit einem solchen Objekt können Sie beispielsweise die Anzahl Spalten oder die Grösse des Anzeigefeldes für eine Spalte, Spaltennamen, Spaltentypus und so weiter, bestimmen.
Auch hier kann es sein, dass Ihnen eine bestimmte Datenbank diese Informationen nicht oder aber zusätzliche liefert.

1.1.8.4. Übungen

Auch zu diesem Abschnitt haben wir bereits viele Beispiele in den bisherigen Übungen verwendet, und weitere werden Sie noch kennen lernen.

1.1.9. Escape Syntax und Skalare Funktionen

Viele Datenbanken stellen skalare Funktionen (auch als *eingebaute Funktionen* bezeichnet) zur Verfügung. Diese erleichren das Arbeiten mit der Datenbank. Gemäss JDBC sollten die Funktionen unterstützt werden, welche in der X/Open Spezifikation des Call Level Interface (CLI) angegeben sind. Auch die Namen sollten mit jenen aus X/Open übereinstimmen, obschon dies nicht immer der Fall ist.

JDBC stellt Methoden zur Verfügung, um feststellen zu können, welche Funktionen zur Verfügung stehen:

- `getNumericFunctions()`,
- `getStringFunctions()`,
- `getSystemFunctions()`,
- `getTimeDateFunctions()` und zwei Versionen für
- `supportsConvert()`.

Die `getXXXFunctions()` Methoden liefern Funktionsnamen als durch Komma getrennte Zeichenkette. Weil die unterschiedlichen Datenbankanbieter eine unterschiedliche Syntax für Funktionsaufrufe verwenden, normiert JDBC diese Aufrufe mittel einer sogenannten Escape Syntax. Der JDBC Treiber sollte diese Syntax verstehen und die Funktionsaufrufe auf die DBMS spezifischen Aufrufe abbilden.

Diese Escape Notation ist beispielsweise fn für function, der aktuelle Funktionsname folgt in Klammern:

```
fn<scalareFunktion()>
```

In der Regel setzen Sie solche Funktionen mit Spaltenwerten ein. Hier einBeispiel:

```
UPDATE meineTabelle
SET bogenWert = bogen * { fn PI() }
...
```

oder

```
SELECT { fn concat( string, "bean" ) }
...
```

Auch hier müssen Sie im Handbuch der DBMS nachschauen, welche Funktionen nun konkret unterstützt werden.

1.1.9.1. Übung - bestimmen Sie die skalaren Funktionen

In dieser Übung bestimmen wir die verfügbaren skalaren Funktionen einer DBMS und zeigen einzelne Möglichkeiten auf, wie man diese einsetzen könnte.

1.1.9.2. Lernziele

Nach dem Durcharbeiten dieser Übung sollten Sie

- verstehen, wie man skalare Funktionen abfragt und einsetzen kann.
- eine Idee haben, wie die Escape Notation funktioniert

1.1.9.3. Szenario

Das proaktive Entwicklerteam überlegt sich, dass bei den wachsenden Anforderungen des XP_Teams sicher bald weitere Anforderungen eintreffen werden und dann der Einsatz von eingebauten / skalaren Funktionen wichtig werden könnte.

Um sicherzustellen, dass sie die Escape Notation verstehen, bauen Sies ich eine einfache Testapplikation, ohne Verrechnung! In dieser fragen sie die Numeric, Date und Time, String und System skalaren Funktionen ab, welche vom DBMS unterstützt werden.

Da diese Funktionen im Zusammenhang mit Abfragen eingesetzt werden, entscheidet das Entwicklungsteam, eine einfache Abfrage fest einzuprogrammieren. Diese bestimmt einfach Daten aus der Tabelle KaffeeListe in der Datenbank XP_Team und wendet auf die Daten die Funktionen an, falls dies möglich ist.

1.1.9.4. Voraussetzungen

- Sie wissen, wie man eine Tabelle kreiert und Daten einfügt.
- Sie können interaktiv auf Tabellen zugreifen
- Sie können im Batch Modus auf Tabellen zugreifen

1.1.9.5. Rahmenprogramm

Wir verwenden die üblichen ConnectionXP.properties.
Zudem steht Ihnen folgendes Rahmenprogramm zur Verfügung.

SkalareFunktionen.java:

```
package skalarefunktionen;

import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;
import javax.swing.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */
```

```
public class SkalareFunktionen extends JFrame
                     implements ActionListener,
                                WindowListener
{
  Connection con = null;
  DatabaseMetaData dbmd;
  DefaultComboBoxModel dcbm =
     new DefaultComboBoxModel();

  JButton          jbConnect = new JButton("Connect"),
                   jbFN = new JButton(
                     "Funktionen bestimmen" ),
                   jbDoFn = new JButton(
                     "Bestimme CURDATE" );
  JComboBox        jcb = new JComboBox( dcbm );
  JLabel           jlUserID = new JLabel("UserID:"),
                   jlPassword = new JLabel(
                     "Passwort:");
  JPanel           jpCenter = new JPanel(
                     new BorderLayout() ),
                   jpCenterEast = new JPanel(),
                   jpCenterWest = new JPanel(
                     new BorderLayout() ),
                   jpCenterWestNorth = new JPanel(
                     new GridLayout( 3,2 ) ),
                   jpCenterWestSouth = new JPanel(
                     new GridLayout( 2,1 ) ),
                   jpNorth = new JPanel(
                     new BorderLayout() ),
                   jpNorthNorth = new JPanel(),
                   jpNorthSouth = new JPanel();
  JPasswordField   jpfPassword =
                     new JPasswordField( 10 );
  JRadioButton     jrbNumeric = new JRadioButton(
                     "Numerisch" ),
                   jrbString = new JRadioButton(
                     "String" ),
                   jrbTD = new JRadioButton(
                     "Datum/Uhrzeit" ),
                   jrbSystem = new JRadioButton(
                     "System" );
  JTextArea        jta = new JTextArea(
                     "Fehlermeldungen.", 4, 30 );
  JTextField       jtUserID = new JTextField( 10 );

  ResultSet rs = null;
  Statement stmt = null;
  ResourceBundle rbConnect;
  String sDriver,
         sDriverKey = "CSDriver",
         sPassword,
         srbName = "ConnectXP",
         sURL,
         sURLKey="CSURL",
         sUserID;


  public SkalareFunktionen()
  {
    super("SkalareFunktionen");

    try  // get the PropertyResourceBundle
    {
      rbConnect = ResourceBundle.getBundle( srbName );

      sDriver   = rbConnect.getString( sDriverKey );
      sURL      = rbConnect.getString( sURLKey );
    }
```

```
    catch( MissingResourceException mre )
    {
      System.err.println(
                 "Probleme beim lesen des ResourceBundle " +
                  srbName + ", das Programm wird abgebrochen." );
      System.err.println("Programmfehler: " +
                          mre.getMessage() );
      endApp(); // exit on error
    }

    jbConnect.addActionListener( this );
    jbDoFn.addActionListener( this );
    jbFN.addActionListener( this );

    jrbTD.setSelected( true );
    ButtonGroup bg = new ButtonGroup();
    bg.add( jrbNumeric );
    bg.add( jrbString );
    bg.add( jrbTD );
    bg.add( jrbSystem );

    jpCenterWestNorth.add( jrbNumeric );
    jpCenterWestNorth.add( jrbString );
    jpCenterWestNorth.add( jrbTD );
    jpCenterWestNorth.add( jrbSystem );


    jbFN.setEnabled( false );
    jbDoFn.setEnabled( false );
    jpCenterWestSouth.add( jbFN );
    jpCenterWestSouth.add( jbDoFn );

    jpCenterWest.add( jpCenterWestNorth,
       BorderLayout.NORTH );
    jpCenterWest.add( jpCenterWestSouth,
       BorderLayout.SOUTH );

    jcb.setMaximumRowCount( 5 );
    jpCenterEast.add( jcb );
    jpCenter.add( jpCenterWest, BorderLayout.WEST );
    jpCenter.add( jpCenterEast, BorderLayout.EAST );


    jpNorthNorth.add( jlUserID );
    jpNorthNorth.add( jtUserID );
    jpNorthNorth.add( jlPassword );
    jpNorthNorth.add( jpfPassword );

    jpNorthSouth.add( jbConnect );

    JScrollPane jsp = new JScrollPane( jta );
    jpNorthSouth.add( jsp );

    jpNorth.add( jpNorthNorth, BorderLayout.NORTH );
    jpNorth.add( jpNorthSouth, BorderLayout.SOUTH );

    Container cp = getContentPane();
    cp.add( jpNorth,  BorderLayout.NORTH );
    cp.add( jpCenter, BorderLayout.CENTER );

    addWindowListener( this );
    pack();
    show();

  } // end constructor
```

```
  public void doConnect()
  {
    try // Versuche den JDBC Treiber zu laden,
    {   // mit newInstance
        Class.forName( sDriver ).newInstance();
    }
    catch( Exception e )  // error
    {
      jta.setText("Der JDBC Treiber konnte nicht geladen werden.");
      return;
    } // end catch

    try
    {
      con = DriverManager.getConnection ( sURL,
                                          sUserID,
                                          sPassword);
      stmt = con.createStatement();

//TODO bestimmen Sie die DatabaseMetaData


// ENDTODO
      jbConnect.setEnabled( false );

      LoadFunctions();
      jbFN.setEnabled( true );
    }
    catch ( SQLException SQLe)
    {
      reportSQLError( SQLe,
        "Probleme in DoConnect():" );

      if( con != null )
      {
        try { con.close(); }
        catch( Exception e ) {}
        stmt = null;
      }

      return;
    } // end catch
  } // end doConnect


  public void doInvokeFn()
  {
    String sTemp1 = null,
           sTemp2 = null,
           sTemp3 = null;

    if( jrbNumeric.isSelected() )
    {
      sTemp1 = "PI ist: ";
      sTemp2 = "PI()";
    }
    else
    if( jrbString.isSelected() )
    {
      sTemp1 = "LTRIM Resultat ist: ";
      sTemp2 = "LTRIM('     Right')";
    }
    else
    if( jrbTD.isSelected() )
    {
      sTemp1 = "CURDATE ist: ";
      sTemp2 = "CURDATE()";
```

```
    }
    else
    if( jrbSystem.isSelected() )
    {
      sTemp1 = "USER ist: ";
      sTemp2 = "USER()";
    }

    try
    {

// TODO definieren Sie eine einfache Abfrage
// mit Hilfe der Escape Syntax
// geben Sie den Rückgabewert aus und
// schliessen Sie das ResultSet
// ENDTODO

      jta.setText( sTemp1 + sTemp3 );

    } // end try
    catch ( SQLException SQLe)
    {
      reportSQLError( SQLe,
        "Probleme in doInvokeFn():" );
      SQLe.printStackTrace();
    }
  } // end doInvokeFn


  public void LoadFunctions()
  {
    String sFunctions = null;
    StringTokenizer st = null;

    dcbm.removeAllElements();
    try
    {

// TODO
//    bestimmen Sie alle Funktionen zur aktuellen Auswahl
// ENDTODO

      jbDoFn.setEnabled( false );
      if( sFunctions.equals( "" ) )
      {
        dcbm.addElement( "Es wurden keine Ergebnisse zurück geliefert." );
        return;
      }
      else
      {

// TODO
//    bestimmen Sie die Funktionsnamen
// ENDTODO

      }
```

```
      if( jrbNumeric.isSelected() )
      {
        jbDoFn.setText( "Bestimme PI" );
        if( dcbm.getIndexOf( "PI" ) >= 0 )
        {
          jbDoFn.setEnabled( true );
        }
      }

      if( jrbString.isSelected() )
      {
        jbDoFn.setText( "Bestimme LTRIM" );
        if( dcbm.getIndexOf( "LTRIM" ) >= 0 )
        {
          jbDoFn.setEnabled( true );
        }
      }

      if( jrbTD.isSelected() )
      {
        jbDoFn.setText( "Bestimme CURDATE" );
        if( dcbm.getIndexOf( "CURDATE" ) >= 0 )
        {
          jbDoFn.setEnabled( true );
        }
      }

      if( jrbSystem.isSelected() )
      {
        jbDoFn.setText( "Bestimme USER" );
        if( dcbm.getIndexOf( "USER" ) >= 0 )
        {
          jbDoFn.setEnabled( true );
        }
      }
    } // end try
    catch ( SQLException SQLe)
    {
      reportSQLError( SQLe,
        "Probleme in doInvokeFn():" );
      SQLe.printStackTrace();
    }
  } // end LoadFunctions


  public void endApp()
  {
    if( stmt != null)
    {
      try { stmt.close(); }
      catch( Exception e ) {}
    }

    if( con != null)
    {
      try { con.close(); }
      catch( Exception e ) {}
    }

    dispose();
    System.exit(0);
  } // end endApp
```

```
  // ActionListener implementation
  public void actionPerformed(ActionEvent e)
  {
    Object oSource = e.getSource();

    jta.setText( "Keine Fehler." );
    if( oSource == jbDoFn )
    {
      doInvokeFn();
      return;
    }

    if( oSource == jbFN )
    {
      LoadFunctions();
      return;
    }

    if( oSource == jbConnect )
    {
      sUserID = jtUserID.getText();
      sPassword = jpfPassword.getText();
      doConnect();
      return;
    }

  } // end actionPerformed


  public void reportSQLError( SQLException SQLe,
                              String       s )
  {
      jta.setText( s + "\n" );
      jta.append( SQLe.getMessage() + "\n" );
      jta.append( "SQL State: " +
         SQLe.getSQLState() + "\n"  );
  } // end reportSQLError


// Window Listener Implementation
    public void windowOpened(WindowEvent e) {}

     public void windowClosing(WindowEvent e)
     {
       endApp();
     }

    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
// End Window Listener Implementation

  public static void main (String args[])
  {
    new SkalareFunktionen();
  } // end main

} // end class SkalareFunktionen
```

1.1.9.6. Aufgaben

1) In der Methode `doConnect()` nach dem Verbindungsaufbau:
   bestimmen Sie ein `Statement` Objekt und die `DataBaseMetaData` in `dbmd`.

2) In der doInvokeFn() Methode, innerhalb eines try Blocks:
   fügen Sie Programmcode ein, um eine Abfrage durchzuführen. Hier ist diese SQL Anweisung:

```
SELECT <skalare Funktion>
FROM KaffeeListe
WHERE Eintrag = 10
```

   Der Rückgabewert soll in `sTemp3` abgespeichert werden und anschliessend das `ResultSet rs` geschlossen werden.

3) In der Methode loadFunctions() finden Sie folgenden Programmcode:

```
if( jrbNumeric.isSelected() ) {       }
else if( jrbString.isSelected() ) {   }
else if( jrbTD.isSelected() ) {       }
else if( jrbSystem.isSelected() ) { }
```

   um die entsprechende Gruppe von skalaren Funktionen zu bestimmen. Die Namen dieser Funktionen stehen in `sFunctions`.
   Ihre Aufgabe:
   bestimmen Sie für `jrbNumeric` die numerischen Funktionen
   für `jrbString` die String Funktionen,
   für `jrbTD` die Time und Date Funktionen und
   für `jrbSystem` die System Funktionen.

4) In `LoadFunctions()` müssen Sie jeden Funktionsnamen aus `sFunctions` mit einem `StringTokenizer st` bestimmen und dann diesen Namen in das Modell `dcbm` mit `dcbm.addElement()` laden.
   Damit das Programm nicht zu sehr DBMS abhängig ist, sollten Sie ausschliesslich Gross- oder Kleinbuchstaben verwenden. Grossbuchstaben sind sicherer!

1.1.9.7. Lösungshinweise

Aufgabe 1:

```
dbmd = con.getMetaData();
```

Aufgabe 2:

```
rs = stmt.executeQuery( "SELECT " +
           "{ fn " + sTemp2 + "} " +
           "FROM KaffeeListe " +
           "WHERE Eintrag = 10" );

        if( rs.next() )
        {
          sTemp3 = rs.getObject( 1 ).toString();
        }
        rs.close();
```

Aufgabe 3:

```
        if( jrbNumeric.isSelected() )
        {
          sFunctions = dbmd.getNumericFunctions();
        }
        else
        if( jrbString.isSelected() )
        {
          sFunctions = dbmd.getStringFunctions();
        }
        else
        if( jrbTD.isSelected() )
        {
          sFunctions = dbmd.getTimeDateFunctions();
        }
        else
        if( jrbSystem.isSelected() )
        {
          sFunctions = dbmd.getSystemFunctions();
        }
```

Aufgabe 4:

```
          st = new StringTokenizer( sFunctions, "," );
          while( st.hasMoreTokens() )
          {
            dcbm.addElement(
               st.nextToken().toUpperCase() );
```

1.1.9.8. Musterlösung

ConnectionXP.properties:

```
#PropertiesResourceBundle für Connection Properties
CSDriver=COM.cloudscape.core.RmiJdbcDriver
CSURL=jdbc:cloudscape:rmi:XP_Team
```

SkalareFunktionen.java:

```
package skalarefunktionen;

import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;
import javax.swing.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */

public class SkalareFunktionen extends JFrame
                     implements ActionListener,
                                WindowListener
{
  Connection con = null;
  DatabaseMetaData dbmd;
  DefaultComboBoxModel dcbm =
     new DefaultComboBoxModel();

  JButton          jbConnect = new JButton("Connect"),
                   jbFN = new JButton(
                     "Funktionen bestimmen" ),
                   jbDoFn = new JButton(
                     "Bestimme CURDATE" );
  JComboBox        jcb = new JComboBox( dcbm );
  JLabel           jlUserID = new JLabel("UserID:"),
                   jlPassword = new JLabel(
                     "Passwort:");
  JPanel           jpCenter = new JPanel(
                      new BorderLayout() ),
                   jpCenterEast = new JPanel(),
                   jpCenterWest = new JPanel(
                      new BorderLayout() ),
                   jpCenterWestNorth = new JPanel(
                      new GridLayout( 3,2 ) ),
                   jpCenterWestSouth = new JPanel(
                      new GridLayout( 2,1 ) ),
                   jpNorth = new JPanel(
                      new BorderLayout() ),
                   jpNorthNorth = new JPanel(),
                   jpNorthSouth = new JPanel();
  JPasswordField   jpfPassword =
                      new JPasswordField( 10 );
  JRadioButton     jrbNumeric = new JRadioButton(
                     "Numerisch" ),
                   jrbString = new JRadioButton(
                     "String" ),
                   jrbTD = new JRadioButton(
                     "Datum/Uhrzeit" ),
                   jrbSystem = new JRadioButton(
                     "System" );
  JTextArea        jta = new JTextArea(
                     "Fehlermeldungen.", 4, 30 );
```

```
JTextField      jtUserID = new JTextField( 10 );

ResultSet rs = null;
Statement stmt = null;
ResourceBundle rbConnect;
String sDriver,
       sDriverKey = "CSDriver",
       sPassword,
       srbName = "ConnectXP",
       sURL,
       sURLKey="CSURL",
       sUserID;


public SkalareFunktionen()
{
  super("SkalareFunktionen");

  try  // get the PropertyResourceBundle
  {
    rbConnect = ResourceBundle.getBundle( srbName );

    sDriver   = rbConnect.getString( sDriverKey );
    sURL      = rbConnect.getString( sURLKey );
  }
  catch( MissingResourceException mre )
  {
    System.err.println(
               "Probleme beim lesen des ResourceBundle " +
                srbName + ", das Programm wird abgebrochen." );
    System.err.println("Programmfehler: " +
                       mre.getMessage() );
    endApp(); // exit on error
  }

  jbConnect.addActionListener( this );
  jbDoFn.addActionListener( this );
  jbFN.addActionListener( this );

  jrbTD.setSelected( true );
  ButtonGroup bg = new ButtonGroup();
  bg.add( jrbNumeric );
  bg.add( jrbString );
  bg.add( jrbTD );
  bg.add( jrbSystem );

  jpCenterWestNorth.add( jrbNumeric );
  jpCenterWestNorth.add( jrbString );
  jpCenterWestNorth.add( jrbTD );
  jpCenterWestNorth.add( jrbSystem );


  jbFN.setEnabled( false );
  jbDoFn.setEnabled( false );
  jpCenterWestSouth.add( jbFN );
  jpCenterWestSouth.add( jbDoFn );

  jpCenterWest.add( jpCenterWestNorth,
     BorderLayout.NORTH );
  jpCenterWest.add( jpCenterWestSouth,
     BorderLayout.SOUTH );

  jcb.setMaximumRowCount( 5 );
  jpCenterEast.add( jcb );
  jpCenter.add( jpCenterWest, BorderLayout.WEST );
  jpCenter.add( jpCenterEast, BorderLayout.EAST );


  jpNorthNorth.add( jlUserID );
  jpNorthNorth.add( jtUserID );
```

```
    jpNorthNorth.add( jlPassword );
    jpNorthNorth.add( jpfPassword );

    jpNorthSouth.add( jbConnect );

    JScrollPane jsp = new JScrollPane( jta );
    jpNorthSouth.add( jsp );

    jpNorth.add( jpNorthNorth, BorderLayout.NORTH );
    jpNorth.add( jpNorthSouth, BorderLayout.SOUTH );

    Container cp = getContentPane();
    cp.add( jpNorth,  BorderLayout.NORTH );
    cp.add( jpCenter, BorderLayout.CENTER );

    addWindowListener( this );
    pack();
    show();

  } // end constructor


  public void doConnect()
  {
    try // Versuche den JDBC Treiber zu laden,
    {   // mit newInstance
        Class.forName( sDriver ).newInstance();
    }
    catch( Exception e )  // error
    {
      jta.setText("Der JDBC Treiber konnte nicht geladen werden.");
      return;
    } // end catch

    try
    {
      con = DriverManager.getConnection ( sURL,
                                          sUserID,
                                          sPassword);
      stmt = con.createStatement();
      dbmd = con.getMetaData();

      jbConnect.setEnabled( false );

      LoadFunctions();
      jbFN.setEnabled( true );
    }
    catch ( SQLException SQLe)
    {
      reportSQLError( SQLe,
        "Probleme in DoConnect():" );

      if( con != null )
      {
        try { con.close(); }
        catch( Exception e ) {}
        stmt = null;
      }

      return;
    } // end catch
  } // end doConnect


  public void doInvokeFn()
  {
    String sTemp1 = null,
           sTemp2 = null,
           sTemp3 = null;
```

```
    if( jrbNumeric.isSelected() )
    {
      sTemp1 = "PI ist: ";
      sTemp2 = "PI()";
    }
    else
    if( jrbString.isSelected() )
    {
      sTemp1 = "LTRIM Resultat ist: ";
      sTemp2 = "LTRIM('     Right')";
    }
    else
    if( jrbTD.isSelected() )
    {
      sTemp1 = "CURDATE ist: ";
      sTemp2 = "CURDATE()";
    }
    else
    if( jrbSystem.isSelected() )
    {
      sTemp1 = "USER ist: ";
      sTemp2 = "USER()";
    }

    try
    {
      rs = stmt.executeQuery( "SELECT " +
           "{ fn " + sTemp2 + "} " +
           "FROM KaffeeListe " +
           "WHERE Eintrag = 10" );

      if( rs.next() )
      {
        sTemp3 = rs.getObject( 1 ).toString();
      }
      rs.close();

      jta.setText( sTemp1 + sTemp3 );

    } // end try
    catch ( SQLException SQLe)
    {
      reportSQLError( SQLe,
        "Probleme in doInvokeFn():" );
      SQLe.printStackTrace();
    }
  } // end doInvokeFn


  public void LoadFunctions()
  {
    String sFunctions = null;
    StringTokenizer st = null;

    dcbm.removeAllElements();
    try
    {
      if( jrbNumeric.isSelected() )
      {
        sFunctions = dbmd.getNumericFunctions();
      }
      else
      if( jrbString.isSelected() )
      {
        sFunctions = dbmd.getStringFunctions();
      }
      else
      if( jrbTD.isSelected() )
      {
        sFunctions = dbmd.getTimeDateFunctions();
```

```
    }
    else
    if( jrbSystem.isSelected() )
    {
      sFunctions = dbmd.getSystemFunctions();
    }

    jbDoFn.setEnabled( false );
    if( sFunctions.equals( "" ) )
    {
      dcbm.addElement( "Es wurden keine Ergebnisse zurück geliefert." );
      return;
    }
    else
    {
      st = new StringTokenizer( sFunctions, "," );
      while( st.hasMoreTokens() )
      {
        dcbm.addElement(
           st.nextToken().toUpperCase() );
      }
    }

    if( jrbNumeric.isSelected() )
    {
      jbDoFn.setText( "Bestimme PI" );
      if( dcbm.getIndexOf( "PI" ) >= 0 )
      {
        jbDoFn.setEnabled( true );
      }
    }

    if( jrbString.isSelected() )
    {
      jbDoFn.setText( "Bestimme LTRIM" );
      if( dcbm.getIndexOf( "LTRIM" ) >= 0 )
      {
        jbDoFn.setEnabled( true );
      }
    }

    if( jrbTD.isSelected() )
    {
      jbDoFn.setText( "Bestimme CURDATE" );
      if( dcbm.getIndexOf( "CURDATE" ) >= 0 )
      {
        jbDoFn.setEnabled( true );
      }
    }

    if( jrbSystem.isSelected() )
    {
      jbDoFn.setText( "Bestimme USER" );
      if( dcbm.getIndexOf( "USER" ) >= 0 )
      {
        jbDoFn.setEnabled( true );
      }
    }
  } // end try
  catch ( SQLException SQLe)
  {
    reportSQLError( SQLe,
      "Probleme in doInvokeFn():" );
    SQLe.printStackTrace();
  }
} // end LoadFunctions


public void endApp()
{
```

```
    if( stmt != null)    {
      try { stmt.close(); } catch( Exception e ) {}
    }
    if( con != null){
      try { con.close(); } catch( Exception e ) {}
    }

    dispose();
    System.exit(0);
  } // end endApp

  // ActionListener implementation
  public void actionPerformed(ActionEvent e)  {
    Object oSource = e.getSource();

    jta.setText( "Keine Fehler." );
    if( oSource == jbDoFn )    {
      doInvokeFn();
      return;
    }

    if( oSource == jbFN )    {
      LoadFunctions();
      return;
    }

    if( oSource == jbConnect )    {
      sUserID = jtUserID.getText();
      sPassword = jpfPassword.getText();
      doConnect();
      return;
    }

  } // end actionPerformed


  public void reportSQLError( SQLException SQLe,
                              String       s )
  {
      jta.setText( s + "\n" );
      jta.append( SQLe.getMessage() + "\n" );
      jta.append( "SQL State: " +
         SQLe.getSQLState() + "\n"  );
  } // end reportSQLError


// Window Listener Implementation
    public void windowOpened(WindowEvent e) {}

     public void windowClosing(WindowEvent e)
     {
       endApp();
     }

    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
// End Window Listener Implementation

  public static void main (String args[])
  {
    new SkalareFunktionen();
  } // end main

} // end class SkalareFunktionen
```

1.1.10. Demonstration

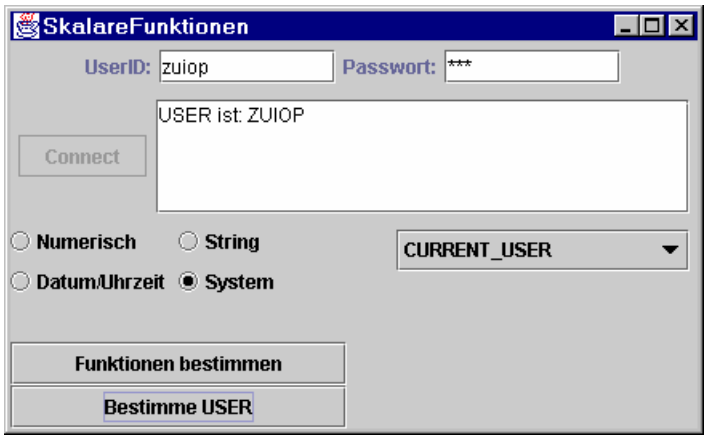
Das programm kann mit der Batch Datei auf dem Server / der CD gestartet werden. Das GUI ist fast selbsterklärend.

Sie müssen als erstes eine Verbindung zur Datenbank aufbauen. Falls Sie keinen Benutzernamen eintippen, können Sie später auch keinen abfragen.

Dann wählen Sie die Kategorie Funktionen, die Sie interessiert, beim Beispiel unten war die "System".

Nun müssem die Funktionen mittels der einfachen Abfrage der metadatan ebestimmte und ins GUI übertragen werden.

Die Anzeige passt sich an und zeigt Ihnen nun die oberste Funktion zum Testen an. Sie können also nicht alle Funktionen testen, wie Sie leicht aus dem Programmcode erkennen können - nett wäre dies aber schon. Vielleicht versuchen Sie dies mal selber.



1.1.11. Stored Procedures

Stored Procedures sind benutzergenerierte Funktionen oder Prozeduren, welche von Client Anwendungen aufgerufen aber auf dem Server gespeichert und ausgeführt werden.

Leider gibt es keinerlei Standardisierung, weder von der Syntax noch vom Konzept her. Beispielsweise in Cloudscape kann man einfach alle Java Methoden verwenden, da Cloudscape in Java geschrieben wurde. In Access gibt es meines Wissens keine Stored Procedures, wohl aber auf dem SQLServer, Oracle und vielen anderen Datenbanken.

Damit sind wir in der glücklichen Lagem kein Beispiel zeigen zu müssen!

Im Folgenden soll trotzdem kurz das Konzept vorgestellt werden.

1.1.11.1. MetaData Support

Sie können mittels Metadaten abfragen, ob Ihre Datenbank überhaupt Stored procedures unterstützt oder nicht. Dafür stehen einfach Methoden zur Verfügung:

- `supportsStoredProcedures()`
  bestimmt, ob Ihre DBMS über JDBC Stored Procedures mittels Escape Syntax untersützt.
- `getProcedures()`
  liefert Ihnen eine Liste der vorhandenen Stored Procedures,
  während
  `getProcedureColumns()`
  die Parameter und Rückgabewerte beschreibt.
- `getProcedureTerm()`
  informiert den Programmierer über spezielle Standardnotationen, die vom DBMS unterstützt werden.

1.1.11.2. Parameter INs und OUTs

Wenn Sie eine Methode oder eine Prozedur aufrufen, besitzt diese in der Regel irgendwelche Argumente oder Parameter : die IN Parameter.

Zurück liefert die Methode dann ein ResultSet, einen Zähler, oder keinen, einen oder mehrere OUT Parameter.

Eine Stored Procedure kann auch INOUT Parameter besitzen, also Parameter, welche sowohl IN als auch OUT Parameter sind.

1.1.11.3. Escape Syntax

Wie bereits erwähnt, verwendet JDBC die sogenannte Escape Syntax. Der Treiber muss dann für die nötige Umsetzung in normale Datenbankaufrufe sorgen.

Analog dazu existiert eine JDBC Version der Escape Notation für Stored Procedures:

call sp_name oder
? = call sp_name mit optionalen Parametern in Klammern.

Dabei treten verschiedene Formen auf, die wir im Folgenden betrachten werden:

A - ohne Parameter und Rückgabewert oer ein ResultSet oder ein Zeilenzähler:
```
{ call sp_A }
```
B - ein einziger Parameter und ein Ergebnisparameter
Der Resultatparameter sei int , der IN Parameter ein String:
```
{ ? = call sp_B( ? ) }
```
C - mehrere Parameter und keine Rückgabe, ein ResultSet oderr ein Zeilenzähler.
Annahme int IN, OUT und INOUT Parameter:
```
{ call sp_C( ? ? ? ) }
```

1.1.11.4. CallableStatement

Um eine Stored Procedure aufzurufen, müssen Sie ein CallableStatement definieren. Diese erweitern die PreparedStatement Klasse.

A -
```
CallableStatement cstmt =
   con.prepareCall( "{ call sp_A }" );
```
B -
```
CallableStatement cstmt =
   con.prepareCall( "{ ? = call sp_B( ? ) }" );
```
C -
```
CallableStatement cstmt =
   con.prepareCall( "{ call sp_C( ? ? ? ) }" );
```

1.1.11.5. Setup, Invocation und Value Retrieval

Bevor Sie eineStored procedure aufrufen und benutzen können, müssen Sie passende Parameter definieren.

- IN werden mit setXXX() Methoden aus der Klasse PreparedStatement verwendet.
- OUT Parameter müssen registriert werden, mittels einer der CallableStatement Methoden:
  CallableStatement.registerOutParameter().
- INOUT müssen gesetzt und registriert werden.

Der aktuelle Aufruf geschieht wie gehabt mittels executeQuery(), executeUpdate() oder execute() je nach erwartetem Ergebnis.

A -
```
CallableStatement cstmt = con.prepareCall( "{ call sp_A }" );
```

Falls nichts zurück gegeben wird:
```
cstmt.execute(); // oder executeUpdate()
```

Falls ein ResultSet zurück gegeben wird:
```
ResultSet rs = cstmt.executeQuery();
```

Falls ein Zähler mutiert wird:
```
int iUC = cstmt.executeUpdate();
```

B -
```
CallableStatement cstmt=con.prepareCall( "{ ? = call sp_B( ? ) }" );
// int Resultat Parameter
cstmt.registerOutParameter( 1, Types.INTEGER );
// String IN Parameter
cstmt.setString( 2, "M-O-O-N" );

cstmt.execute(); // oder executeUpdate()
int iRP = cstmt.getInt( 1 );
```

C -
```
CallableStatement cstmt=con.prepareCall( "{ call sp_C( ? ? ? ) }" );
```

Setup:
```
// set int IN Parameter
cstmt.setInt( 1, 333 );
// register int OUT Parameter
cstmt.registerOutParameter( 2, Types.INTEGER );

// set int INOUT Parameter
cstmt.setInt( 3, 666 );
// register int INOUT Parameter
cstmt.registerOutParameter( 3, Types.INTEGER );
```

Ohne Rückgabe (ausser OUT und INOUT:)
```
cstmt.execute(); // oder executeUpdate()
// get int OUT und INOUT
int iOUT = cstmt.getInt( 2 );
int iINOUT = cstmt.getInt( 3 );
```

Mit ResultSet:
```
ResultSet rs = cstmt.executeQuery();
// get int OUT und INOUT
int iOUT = cstmt.getInt( 2 );
int iINOUT = cstmt.getInt( 3 );
```

Falls ein Zähler existiert:
```
int iUC = cstmt.executeUpdate();
// get int OUT und INOUT
int iOUT = cstmt.getInt( 2 );
int iINOUT = cstmt.getInt( 3 );
```

Beachten Sie, dass oben lediglich das Schema aufgezeigt wird. Die Realität sieht viel komplexer aus, in der Regel wenigstens. Aber Ihr DBMS Anbieter liefert normalerweise Beispiele mit, an denen Sie sich orientieren können und müssen, da die Dokumentation genauso schlecht ist wie die Heterogenität der Aufrufsyntax.

1.1.12. Transaktion

In SQL Terminologie besteht eine Transaktion aus einer logischen Einheit von Aktivitäten (logic unit of work LUW). Im einfachsten Fall ist also alles eine Transaktion?

Transaktionen haben bestimmte Eigenschaften, die Sie sonst nicht immer antreffen, beispielsweise die "Alles oder Nichts" Eigenschaft: eine Transaktion wird entweder als Ganzes oder eben überhaupt nicht ausgeführt.

Das klassische Beispiel einer Transaktion ist eine Banktransaktion: sie heben Geld ab, ganz oder nicht. Aber Sie legen Wert darauf, dass beim Abbrechen einer Banktransaktion nicht ein Teil Ihres Geldes veschwindet.

Genau so sollten die Datenbankoperationen INSERT, UPDATE und DELETE funktionieren! Es existieren SQL Dialekte, bei denen mittels begin... end. Konstrukten die Atomizität einer Transaktion formal beschrieben wird. Schliesslich wird die Transaktion committed, also letzlich ausgeführt.

JDBC verwendet ein ähnliches Modell, mit Commit. Man kann bei vielen Datenbanken auch auf Autocommit umschalten, wobei aber auch dies nicht immer sinnvoll ist!

Bemerkung: im autocommit Mode wird eine Anweisung nach deren Abschluss committed.

Falls ein ResultSet zurückgeliefert wird, ist die Ausführung der Anweisung erst dann abgeschlossen, wenn die letzte Zeile gelesen wurde oder das ResultSet geschlossen wurde.

Sie können mit der Methode `setAutoCommit(boolean autocommit)` der Klasse `Connection` den AutoCommit Modus setzen. Damit wird jede Transaktion automatisch comitted. Mit `Connection.setAutoCommit(false)` wird diese Einstellung wieder rückgängig gemacht. Sie können dann mit `Connection.commit()` und `Connection.rollback()` die Transaktionen kontrollieren.

1.1.12.1. Commit

Falls AutoCommit auf `false` gesetzt wurde, gelten alle Datenbanktransaktionen lediglich temporär, bis Sie diese bestätigen, mit `Connection.commit()` Methode.
Falls Sie diese Methode aufrufen, werden alle Transaktionen seit dem letzten Commit gültig. Sonst werden sie rückgängig gemacht (*rollback*).

Wichtig in diesem Zusammenhang ist noch, dass Daten solange Sie diese nicht bestätigen, also eine Transaktion committen, nicht verwendet werden können. Im Schlimmsten Fall werden auch Locks gesetzt, welche den Zugriff auf Daten oder Tabellen oder sogar eine ganze Datenbank sperren!

1.1.12.2. Rollback

Die Methode `Connection.rollback()` kann eingesetzt werden, um Änderungen seit dem letzten Commit zu annulieren.

Diese Methode können Sie beispielsweise einsetzen, falls Fehler aufreten und Sie nicht sicher sein können, dass eine Transaktion korrekt abgewickelt wurde.

1.1.12.3. Concurrency / Nebenläufigkeit

Die meisten DBMS gestatten den nebenläufigen Zugriff auf Daten in der Datenbank, in einem klar definierten Rahmen. Je nach Level der Nebenläufigkeit können gravierende Konsequenzen resultieren, beispielsweise der Verlust von Daten oder Verklemmungen bzw. Performance Einbussen.

JDBC kennt folgende Transaction Isolation Levels, mit deren Hilfe die Concurrency kontrolliert werden kann:

- TRANSACTION_NONE
- TRANSACTION_READ_COMMITTED
- TRANSACTION_READ_UNCOMMITTED
- TRANSACTION_REPEATABLE_READ
- TRANSACTION_SERIALIZABLE

Sie können in JDBC Programmen mit den Methoden `getTransactionIsolation()` und `setTransactionIsolation (int level)` Methoden der Connection Klasse bestimmen, welche Isolation Level möglich und gesetzt sind.

Ein JDBC Treiber definiert einen Standard Isolationslevel für die zu Grunde liegende DBMS. Aber nicht alle Datenbanken unterstützen Isolationslevel. Die sorgfältige Planung der Isolationslevel ist eine sehr wichtige Aufbage bei einem Datenbank-Design speziell der Zugriffslogik! Hinweise auf optimale Zugriffssteuerung finden Sie in der Regel in den Handbüchern der DBMS Anbieter, beispielsweise im Tuning Guide.

1.1.12.4. Typische Transaction Code

Hier einige Beispiele für typische Transaktionscodes:

```
con.setAutoCommit( false );
...
bError = false;
try {
      for( ... ) {// validiere die Daten. Im Fehlerfall Rollback
        if( bError ) {  break;  }
        stmt.executeUpdate( ... );
      }
      if( bError )  {  con.rollback(); }
      else { con.commit();  }

} catch ( SQLException SQLe) {
      con.rollback();
      ...
} catch ( Exception e) {
      con.rollback();
      ...
}
```

1.1.12.5. Übung - Transaktionen

In dieser Übung modifizieren wir einmal mehr ein bestehendes Programm. Wir haben bei der Besprechung der Warnings und Exceptions bereits einiges eingebaut, was wir hier wieder benutzen können. Nun fügen wir noch Transaktionen hinzu.

1.1.12.6. Lernziele

Nach dem Durcharbeiten dieser Übung sollten Sie in der Lage sein:

- einige der Möglichkeiten von Transaktionen zu demonstrieren
- Isolationslevel und andere Metadaten, welche bei Transaktionen eine Rolle spielen, zu bestimmen und eventuell anzupassen.
- Transaktionsprogramme zu schreiben und zu testen.

1.1.12.7. Szenario

Um die Datenbank Integrität zu garantieren, beschliesst das Entwicklungsteam sich mit Transaktionen zu befassen und das bestehende Programm sicherer zu machen.

Ziel ist es, die Daten nur dann abzuspeichern, wenn alle Anweisungen in der transaktion erfolgreich abgeschlossen werden können.

Um Tests durchführen zu können,werden weitere Tabellen definiert und eingebaut: `KreiereKaffeeTabellen.properties` beschreibt die SQL Anweisung zum Generieren dieser Tabelle.

Mit diesem ResourceBundle werden die Tabellen KaffeeVal, KaffeeComp und KaffeeAnbieter kreiert.

Im ResourceBundle `LadeKaffeeAnbieter.properties` stehen die Daten.

Nach Abschluss der Tests können Sie die Tabellen mit dem ResourceBundle `LoescheKaffeeTabellen.properties` alles wieder löschen.

Nun müssen wir nur noch die Anwendungsprogramme oder das Demobeispiel programmieren.

1.1.12.8. Voraussetzungem

Sie haben das Beispiel zu den SQL Warnings und Exceptions durchgearbeitet.

1.1.12.9. Rahmenprogramm

Die ConnectionXP.properties gestatten Ihnen wie immer den Zugang zur Datenbank.
Die anderen Property Dateien sehen folgendermassen aus:

KreiereKaffeeTabellen.properties:

```
#PropertiesResourceBundle für Transaktionen Properties
# Kreiere Kaffee Tabellen
1=CREATE TABLE KaffeeVal ( Sorte VARCHAR (25) NOT NULL, Gewicht INTEGER NOT
NULL, Preis NUMERIC(5, 2) NOT NULL )
2=CREATE TABLE KaffeeAnbieter ( Name VARCHAR (25) NOT NULL, ID INTEGER NOT
NULL, PRIMARY KEY ( ID ) )
3=CREATE TABLE KaffeeComp ( ID INTEGER NOT NULL, Sorte VARCHAR (25) NOT
NULL, Gewicht INTEGER NOT NULL, Preis NUMERIC(5, 2) NOT NULL, PRIMARY KEY (
ID, Sorte, Gewicht ) )
```

LadeKaffeeAnbieter.properties:

```
#PropertiesResourceBundle für Transaktionen Properties
# Laden der Anbieter Table
1=INSERT INTO KaffeeAnbieter VALUES ('BeanWhackers', 10 )
2=INSERT INTO KaffeeAnbieter VALUES ('JoeBeans', 20 )
3=INSERT INTO KaffeeAnbieter VALUES ('StandardBeans', 30 )
4=INSERT INTO KaffeeAnbieter VALUES ('ExoticaBeans', 40 )
5=INSERT INTO KaffeeAnbieter VALUES ('EthiopianPlus', 50 )
6=INSERT INTO KaffeeAnbieter VALUES ('NavyBeans', 60 )
7=INSERT INTO KaffeeAnbieter VALUES ('BeanHomeLately', 70 )
8=INSERT INTO KaffeeAnbieter VALUES ('Beanies', 80 )
9=INSERT INTO KaffeeAnbieter VALUES ('Jones, Smith & Beanette', 90 )
10=INSERT INTO KaffeeAnbieter VALUES ('BeanBag', 100 )
```

LoescheKaffeeTabellen.properties:

```
#PropertiesResourceBundle für Kaffee Properties
# Drop Kaffee Tabellen
1=DROP TABLE KaffeeVal
2=DROP TABLE KaffeeComp
3=DROP TABLE KaffeeAnbieter
```

Und schliesslich das Java Rahmenprogramm (aus der Musterlösung generiert):

```
package transaktionen;

import java.sql.*;
import java.util.*;


public class XPTransaktionen
{
  boolean bError = false;
  Connection con;
  ResourceBundle rb;
  SQLWarning sqlw;
  Statement stmt;
  String sDriver,
         sDriverKey = "CSDriver",
         sKey,
         sPassword,
         sPasswordKey ="CSPassword",
         sUpdate,
         srbName = "ConnectXP",
         srbUpdate,
         sURL,
         sURLKey="CSURL",
         sUserID,
         sUserIDKey = "CSUserID";
```

```
  public XPTransaktionen() throws MissingResourceException,
                          ClassNotFoundException,
                          InstantiationException,
                          IllegalAccessException
  {
    // lies das PropertyResourceBundle
    rb = ResourceBundle.getBundle( srbName );

    sDriver   = rb.getString( sDriverKey );
    sPassword = rb.getString( sPasswordKey );
    sURL      = rb.getString( sURLKey );
    sUserID   = rb.getString( sUserIDKey );

    // versuche den JDBC Treiber zu laden
    // mit newInstance
    Class.forName( sDriver ).newInstance();

  } // end Konstruktor


  public void doUpdate( String sargRBName )
              throws MissingResourceException,
                     SQLException
  {
    int iProcessed = 0,
        iProcessedCount =0;

    try // bestimme das Connection und Statement Objekt
    {
      con = DriverManager.getConnection ( sURL,
                                          sUserID,
                                          sPassword);

// TODO
//    Schalten Sie autocommit aus
// ENDTODO

      stmt = con.createStatement();
      sqlw = con.getWarnings();
      if( sqlw != null )
      {
        handleSQLWarnings( sqlw,
                           "Warnung beim Verbindungsaufbau: ",
                            null );
        con.clearWarnings();
      }

    } // end try
    catch ( SQLException SQLe)
    {
      handleSQLExceptions(
         SQLe,
        "Probleme beim Verbindungsaufbau zu " + sURL + ":",
         null            );

      if( con != null)
      {
        try
        {
          con.commit();
          con.close();
        }
        catch( Exception e ) {}
      }
      // rethrow Exception
      throw SQLe;

    } // end catch
```

```
    try
    {
      Enumeration e = null;
      int iCount = 0;

      // lies das PropertyResourceBundle
      // für SQL Update Statements
      rb = ResourceBundle.getBundle( sargRBName );
      e = rb.getKeys();

      // Count keys - Keys starten mit 1.
      for( ; e.hasMoreElements(); iCount++ )
      {
        (e.nextElement());
      };

      // verwende <= weil Keys bei 1 anfangen
      for( int i = 1; i <= iCount; i++ )
      {
        sKey = "" + i;
        sUpdate = rb.getString( sKey );

        iProcessed = stmt.executeUpdate( sUpdate );
        if( iProcessed == 0 )
        {
          System.err.println(
            "XPTransaktionen konnte den Eintrag nichtverarbeiten:\n" +
             sUpdate + "." );

// TODO
//    geben Sie die Rollback Mitteilung aus
//    reset iProcessedCount,
//    set bError und
//    exit loop
//ENDTODO

        }
        else
        {
          iProcessedCount += iProcessed;
        }
        sqlw = stmt.getWarnings();
        if( sqlw != null )
        {
          handleSQLWarnings( sqlw,
                             "Statement Warnungen: ",
                              sUpdate );
        }
      } // end for

    } // end try
    catch( MissingResourceException mre )
    {
      System.err.println(
                 "ResourceBundle Problem mit " +
                  sargRBName + ", Programmabbruch." );
      System.err.println("Programmabbruch: " +
                         mre.getMessage() );
      throw mre;
    }
    catch ( SQLException SQLe)
    {

// TODO
//    fügen Sie Code ein um
//    iProcessedCount,
//    bError
//    zu setzen
// ENDTODO
      handleSQLExceptions(
```

```
          SQLe,
         "Probleme mit executeUpdate:",
          sUpdate         );

      // rethrow Exception
      throw SQLe;
    }
    catch ( RuntimeException e )
    {

    // TODO
//    iProcessedCount,
//    bError
//    setzen
// ENDTODO
      // rethrow Exception
      throw e;
    }
    finally
    {
      System.out.println( iProcessedCount +
                          " Zeilen wurden bearbeitet." );

      try { stmt.close(); }
      catch( Exception e ) {}

      try
      {

// TODO
//    falls bError
//    dann Rollback
//    sonst commit
// ENDTODO

        con.close();
      }
      catch( Exception e ) {}
    } // end finally clause

  } // end doUpdate


  public void handleSQLExceptions( SQLException SQLe,
                                   String       s,
                                   String       sSQL )
  {
    boolean bFirstPass = true;

    while( SQLe != null)
    {
      reportSQLExceptions( SQLe, s, sSQL );

      if( bFirstPass )
      {
        s = sSQL = null;
        bFirstPass = false;
      }

      SQLe = SQLe.getNextException();
    }

  } // end handleSQLExceptions
```

```
public void handleSQLWarnings( SQLWarning SQLw,
                               String        s,
                               String     sSQL )
{
  boolean bFirstPass = true;

  if( s == null ) { s = "SQLWarning:";  }

  while( SQLw != null)
  {
    reportSQLExceptions( SQLw, s, sSQL );

    if( bFirstPass )
    {
      s = sSQL = null;
      bFirstPass = false;
    }
    SQLw = SQLw.getNextWarning();
  }

} // end handleSQLWarnings


public void reportSQLExceptions( SQLException SQLe,
                                 String       s,
                                 String       sSQL )
{
  String sSQLState = null;

  if( sSQL != null )
  { // Report SQL Problem
    try
    {
      System.err.println(
         con.nativeSQL( sUpdate ) );
    }      catch( Exception e) { /* was nun? */ }
  }

  if( s != null )    { // Report Programmtext
    System.err.println( s );
  }

  sSQLState = SQLe.getSQLState();

  // Report Error Informationen
  System.err.println( SQLe.getMessage() );
  System.err.println( "SQL State: " +
                      sSQLState );
  System.err.println( "Vendor Error Code: " +
                      SQLe.getErrorCode() );
  // check for Data Truncation
  if( sSQLState.equals( "01004") )
  {
    DataTruncation dt = (DataTruncation)(SQLe);

    System.err.println( "Data Size: " +
                        dt.getDataSize() );
    System.err.println( "Transfer Size: " +
                       dt.getTransferSize() );
    System.err.println( "Index der/des " +
                        ( dt.getParameter()
                           ? "Parameters "
                           : "Spalte " ) +
                       "war " +
                        dt.getIndex() + "." );

  }  // end if Data Truncation
} // end reportSQLExceptions
```

```
  public static void main (String args[])
  {
    boolean bContinue = true;
    XPTransaktionen xpApp = null;

    if( args.length != 1 )
    {
      System.err.println("Usage: " +
        "java XPTransaktionen <SQLUpdateResourceBundleName>" );
      System.err.println("       " +
        "                                    (ohne Extension)" );
      return;
    }

    try
    {
      xpApp = new XPTransaktionen();
    }
    catch( Exception e )
    {
      System.err.println("Konstruktor Exception: " +
                         e.getMessage() );
      bContinue = false;
    }

    if( bContinue )
    {
      try
      {
        xpApp.doUpdate( args[0] );
      }
      catch( Exception e ) {}
    }

  } // end main

} // end class XPTransaktionen
```

1.1.12.10. Aufgaben

Aufgabe 1:
Fügen Sie in der `doUpdate()` Methode Programmcode ein, so dass `autocommit` ausgeschaltet wird.

Aufgabe 2:
In `doUpdate()` :
falls die Anzahl Zeilen, welche verändert wurden 0 ist,

- gib eine Meldung an System.err aus, dass die vorgängigen Operationen rückgängig gemacht werden.
- setzen Sie `iProcessedCount` auf 0 (Anzahl bearbeiteter Zeilen)
- setzen Sie das Error Flag `bError` auf `true` und verlassen Sie die Bearbeitungsschleife.

Aufgabe 3:
In `doUpdate()`:
fügen Sie in den letzten zwei `catch` Blöcken Programmcode ein, so dass wie oben die Anzahl bearbeiteter Zeilen in `iProcessedCount` steht und das Error Flag `bError` `true` gesetzt wird.

Aufgabe 4:
In `doUpdate()` :
prüfen Sie im `finally` Block das Error Flag `bError`.
Falls es `true` ist, führen Sie ein Rollback durch und drucken Sie eine Meldung aus.
Falls es `false` ist, commit die Statements.

1.1.12.11. Lösungshinweise

Zu Aufgabe 1:

```
con.setAutoCommit( false );
```

Zu Aufgabe 2:

```
System.err.println(
      "XPTransaktionen konnte den Eintrag nichtverarbeiten:\n" +
            sUpdate + "." );
System.err.println( "Bemerkung: XPTransaktionen wird " +
            "rueckgaengig gemacht (rollback) : " +
             sargRBName + " falls irgendein Statement, " +
            "inclusive DDLs 0 zurueck liefert.\n"
                        );
iProcessedCount = 0;
bError = true;
break;
```

Zu Aufgabe 3:

```
iProcessedCount = 0;
bError = true;
```

Zu Aufgabe 4:

```
if( bError )
{
      con.rollback();
      System.err.println( "ALLE Statements für " +
      sargRBName + " wurden rolled back.\n" );
}
else { con.commit(); }
```

1.1.12.12. Musterlösung

Die Property Dateien haben wir weiter oben bereits gezeigt.

Die Java Lösung sieht folgendermassen aus:

```
package transaktionen;

import java.sql.*;
import java.util.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */

public class XPTransaktionen
{
  boolean bError = false;
  Connection con;
  ResourceBundle rb;
  SQLWarning sqlw;
  Statement stmt;
  String sDriver,
         sDriverKey = "CSDriver",
         sKey,
         sPassword,
         sPasswordKey ="CSPassword",
         sUpdate,
         srbName = "ConnectXP",
         srbUpdate,
         sURL,
         sURLKey="CSURL",
         sUserID,
         sUserIDKey = "CSUserID";


  public XPTransaktionen() throws MissingResourceException,
                          ClassNotFoundException,
                          InstantiationException,
                          IllegalAccessException
  {
    // lies das PropertyResourceBundle
    rb = ResourceBundle.getBundle( srbName );

    sDriver   = rb.getString( sDriverKey );
    sPassword = rb.getString( sPasswordKey );
    sURL      = rb.getString( sURLKey );
    sUserID   = rb.getString( sUserIDKey );

    // versuche den JDBC Treiber zu laden
    // mit newInstance
    Class.forName( sDriver ).newInstance();

  } // end Konstruktor
```

```
public void doUpdate( String sargRBName )
            throws MissingResourceException,
                   SQLException
{
  int iProcessed = 0,
      iProcessedCount =0;

  try // bestimme das Connection und Statement Objekt
  {
    con = DriverManager.getConnection ( sURL,
                                        sUserID,
                                        sPassword);
    con.setAutoCommit( false );

    stmt = con.createStatement();
    sqlw = con.getWarnings();
    if( sqlw != null )
    {
      handleSQLWarnings( sqlw,
                         "Warnung beim Verbindungsaufbau: ",
                          null );
      con.clearWarnings();
    }

  } // end try
  catch ( SQLException SQLe)
  {
    handleSQLExceptions(
       SQLe,
      "Probleme beim Verbindungsaufbau zu " + sURL + ":",
       null            );

    if( con != null)
    {
      try
      {
        con.commit();
        con.close();
      }
      catch( Exception e ) {}
    }
    // rethrow Exception
    throw SQLe;

  } // end catch

  try
  {
    Enumeration e = null;
    int iCount = 0;

    // lies das PropertyResourceBundle
    // für SQL Update Statements
    rb = ResourceBundle.getBundle( sargRBName );
    e = rb.getKeys();

    // Count keys - Keys starten mit 1.
    for( ; e.hasMoreElements(); iCount++ )
    {
      (e.nextElement());
    };

    // verwende <= weil Keys bei 1 anfangen
    for( int i = 1; i <= iCount; i++ )
    {
      sKey = "" + i;
      sUpdate = rb.getString( sKey );

      iProcessed = stmt.executeUpdate( sUpdate );
```

```
      if( iProcessed == 0 )
      {
        System.err.println(
          "XPTransaktionen konnte den Eintrag nichtverarbeiten:\n" +
           sUpdate + "." );
        System.err.println( "Bemerkung: XPTransaktionen wird " +
          "rueckgaengig gemacht (rollback) : " +
           sargRBName + " falls irgendein Statement, " +
          "inclusive DDLs 0 zurueck liefert.\n"
                            );
        iProcessedCount = 0;
        bError = true;
        break;
      }
      else
      {
        iProcessedCount += iProcessed;
      }
      sqlw = stmt.getWarnings();
      if( sqlw != null )
      {
        handleSQLWarnings( sqlw,
                           "Statement Warnungen: ",
                            sUpdate );
      }
    } // end for

  } // end try
  catch( MissingResourceException mre )
  {
    System.err.println(
                "ResourceBundle Problem mit " +
                 sargRBName + ", Programmabbruch." );
    System.err.println("Programmabbruch: " +
                        mre.getMessage() );
    throw mre;
  }
  catch ( SQLException SQLe)
  {
    iProcessedCount = 0;
    bError = true;
    handleSQLExceptions(
       SQLe,
      "Probleme mit executeUpdate:",
       sUpdate         );

    // rethrow Exception
    throw SQLe;
  }
  catch ( RuntimeException e )
  {
    iProcessedCount = 0;
    bError = true;
    // rethrow Exception
    throw e;
  }
  finally
  {
    System.out.println( iProcessedCount +
                       " Zeilen wurden bearbeitet." );

    try { stmt.close(); }
    catch( Exception e ) {}
```

```
    try
    {
      if( bError )
      {
        con.rollback();
        System.err.println( "ALLE Statements für " +
           sargRBName + " wurden rolled back.\n" );
      }
      else { con.commit(); }

      con.close();
    }
    catch( Exception e ) {}
  } // end finally clause

} // end doUpdate


public void handleSQLExceptions( SQLException SQLe,
                                 String       s,
                                 String       sSQL )
{
  boolean bFirstPass = true;

  while( SQLe != null)
  {
    reportSQLExceptions( SQLe, s, sSQL );

    if( bFirstPass )
    {
      s = sSQL = null;
      bFirstPass = false;
    }

    SQLe = SQLe.getNextException();
  }

} // end handleSQLExceptions


public void handleSQLWarnings( SQLWarning SQLw,
                               String        s,
                               String     sSQL )
{
  boolean bFirstPass = true;

  if( s == null ) { s = "SQLWarning:";  }

  while( SQLw != null)
  {
    reportSQLExceptions( SQLw, s, sSQL );

    if( bFirstPass )
    {
      s = sSQL = null;
      bFirstPass = false;
    }

    SQLw = SQLw.getNextWarning();
  }

} // end handleSQLWarnings
```

```
public void reportSQLExceptions( SQLException SQLe,
                                 String       s,
                                 String       sSQL )
{
  String sSQLState = null;

  if( sSQL != null )
  { // Report SQL Problem
    try
    {
      System.err.println(
         con.nativeSQL( sUpdate ) );
    }
    catch( Exception e) { /* was nun? */ }
  }

  if( s != null )
  { // Report Programmtext
    System.err.println( s );
  }

  sSQLState = SQLe.getSQLState();

  // Report Error Informationen
  System.err.println( SQLe.getMessage() );
  System.err.println( "SQL State: " +
                      sSQLState );
  System.err.println( "Vendor Error Code: " +
                      SQLe.getErrorCode() );
  // check for Data Truncation
  if( sSQLState.equals( "01004") )
  {
    DataTruncation dt = (DataTruncation)(SQLe);

    System.err.println( "Data Size: " +
                        dt.getDataSize() );
    System.err.println( "Transfer Size: " +
                       dt.getTransferSize() );
    System.err.println( "Index der/des " +
                        ( dt.getParameter()
                           ? "Parameters "
                           : "Spalte " ) +
                        "war " +
                        dt.getIndex() + "." );

  }  // end if Data Truncation
} // end reportSQLExceptions


public static void main (String args[])
{
  boolean bContinue = true;
  XPTransaktionen xpApp = null;

  if( args.length != 1 )
  {
    System.err.println("Usage: " +
      "java XPTransaktionen <SQLUpdateResourceBundleName>" );
    System.err.println("       " +
      "                                (ohne Extension)" );
    return;
  }
```

```
    try
    {
      xpApp = new XPTransaktionen();
    }
    catch( Exception e )
    {
      System.err.println("Konstruktor Exception: " +
                         e.getMessage() );
      bContinue = false;
    }

    if( bContinue )
    {
      try
      {
        xpApp.doUpdate( args[0] );
      }
      catch( Exception e ) {}
    }

  } // end main

} // end class XPTransaktionen
```

1.1.12.13. Demonstration

Das Beispielprogramm wird beim Fehlerfall ein Rollback ausführen und eine entsprechende Meldung ausgeben.

Nachdem Sie das Programm übersetzt haben, oder einfach die Musterlösung auf dem Server / der CD verwenden, sollten Sie in folgender Reihenfolge vorgehen (die Batch Dateien sind entsprechend nummeriert):

1. Starten Sie
   ```
   java XPRun KreiereKaffeeTabellen
   ```
   um die Tabellen `KaffeeVal`,`KaffeeComp` und `KaffeeAnbieter` zu kreieren.

2. Starten Sie das Transaktions Programm `XPTransaktionen`, welches Sie in dieser Übung kreiert haben:
   ```
   java XPTransaktionen LadeKaffeeAnbieter
   ```
   Die Tabelle KaffeeAnbieter wird zehn Zeilen enthalten.

3. Um das Transaktionsverhalten zu testen, und die Fehlerbehandlung zu prüfen, machen Sie einfach eine Kopie von `LadeKaffeeAnbieter.properties` und ändern irgend eine Zeile darin so, dass ein Fehler auftreten sollte (siehe unten):
   `LadeFalschenKaffee.properties` enthält ein Beispiel

4. Damit die Daten wieder unverändert zur Verfügung stehen, löschen Sie nach den Tests einfach die Tabellen wieder:
   ```
   java XPRun LoescheKaffeeTabellen
   ```
   Dann können Sie wieder wie oben weiterfahren.

Und so könnte die Ausgabe aussehen:

1) Kreieren der Tabellen:

```
XPRun konnte einen Test nicht durchführen:
CREATE TABLE KaffeeVal ( Sorte VARCHAR (25) NOT NULL, Gewicht INTEGER NOT
NULL, Preis NUMERIC(5, 2) NOT NULL ).
Ignorieren Sie die obige Meldung falls Sie lediglich DDL Anweisungen
ausführten.

XPRun konnte einen Test nicht durchführen:
CREATE TABLE KaffeeAnbieter ( Name VARCHAR (25) NOT NULL, ID INTEGER NOT
NULL, PRIMARY KEY( ID ) ).
Ignorieren Sie die obige Meldung falls Sie lediglich DDL Anweisungen
ausführten.

XPRun konnte einen Test nicht durchführen:
CREATE TABLE KaffeeComp ( ID INTEGER NOT NULL, Sorte VARCHAR (25) NOT NULL,
Gewicht INTEGER NOT NULL, Preis NUMERIC(5, 2) NOT NULL, PRIMARY KEY ( ID,
Sorte, Gewicht ) ).
Ignorieren Sie die obige Meldung falls Sie lediglich DDL Anweisungen
ausführten.
```

Und hier die Ausgabe nach dem Ausführen der Transaktionen (korrektes Laden der Daten in die Tabellen):

```
10 Zeilen wurden bearbeitet.
```

Im Fehlerfall oder beim wiederholten Einfügen der selben Daten entsteht folgende Ausgabe:

```
INSERT INTO KaffeeAnbieter VALUES ('BeanWhackers', 10 )
Probleme mit executeUpdate:
The statement was aborted because it would have caused a duplicate key
value in a unique or primary key constraint.
SQL State: 23500
Vendor Error Code: 20000
0 Zeilen wurden bearbeitet.
ALLE Statements für LadeFalschenKaffee wurden rolled back.
```

Auch hier sehen Sie die wilde Mischung der Sprachen. Eine rein englische version drängt sich auf, weil in unserem Beispiel die DBMS Cloudscape keine deutschen Meldungen generieren kann.

Und hier auch noch die (unbedeutende) Meldung beim Löschen der Tabellen:

```
XPRun konnte einen Test nicht durchführen:
DROP TABLE KaffeeVal.
Ignorieren Sie die obige Meldung falls Sie lediglich DDL Anweisungen
ausführten.

XPRun konnte einen Test nicht durchführen:
DROP TABLE KaffeeComp.
Ignorieren Sie die obige Meldung falls Sie lediglich DDL Anweisungen
ausführten.

XPRun konnte einen Test nicht durchführen:
DROP TABLE KaffeeAnbieter.
Ignorieren Sie die obige Meldung falls Sie lediglich DDL Anweisungen
ausführten.

0 Zeilen verarbeitet.
```

Weil keine Zeilen bearbeitet wurde entstand die Warnung.

1.1.13. Batch Update Facility

Das Batch Update Facility ist neu ab JDBC 2.0. Damit wird es möglich, mehrere SQL Anweisungen gemeinsam an die Datenbank zu senden und damit die Performance zu steigern. Die Treiber müssen allerdings dieses Features nicht implementieren. Auch ist die Implementation nicht fest vorgegeben. Es könnte also sein, dass eine Implementation sich völlig anders als eine andere verhält.

Sie können über die Metadaten prüfen, ob die einzelnen Features unterstützt werden:
`DatabaseMetaData.supportsBatchUpdates()`

Batch Anweisungen werden durch die Methoden:
`addBatch()`, `clearBatch()` und `executeBatch()`

angepasst, ergänzt oder ausgeführt. Dabei werden auch Fehlercodes oder Warnings generiert. Die JDBC Empfehlung für das Ausführen von Batch Anweisungen ist Autocommit auf false zu setzen.

Die Rückgabe kann folgendermassen aussehen:

- **-3**--Operation error.
  Ein Treiber stoppt nach dem ersten Fehler und wirft eine `BatchUpdateException` oder zeigt einen Fehler an und fährt fort.
- **-2**--Operation war erfolgreich
  Die Anzahl betroffener Zeilen ist aber nicht bekannt.
- **0**--DDL Statement oder kein Effekt auf die Zeilen der Tabellen.
- **grösser als 0**--Operation war erfolgreich
  Die Rückgabe zeigt an, wieviele Zeilen von der Änderung betroffen waren.

1.1.13.1. Typical Batch Update Programmcode

Hier Skizzen für Programme, welche Batch Update Funktionen verwenden:

```
try
{
    con.setAutoCommit( false );
    ...
    bError = false;
    stmt.clearBatch();

    // SQL Statements hinzufügen
    stmt.addBatch( sUpdate1 );
    stmt.addBatch( sUpdate2 );
    stmt.addBatch( sUpdate3 );

    // Batch Statement ausführen
    aiupdateCounts = stmt.executeBatch();

} // end try

// catch blocks
...

finally
{
    // Resultat bestimmen
    for (int i = 0; i < aiupdateCounts.length; i++)
    {
      iProcessed = aiupdateCounts[i];
      if( iProcessed > 0 ||
          iProcessed == -2
        )
      {
        // Statement war erfolgreich
        ...
      }
      else
      {
        // Fehler
        bError = true;
        break;
      }
    } // end for

    if( bError )
    {
      con.rollback();
    }
    else
    {
      con.commit();
    }
} // end finally
```

1.1.13.2. behandlung von `BatchUpdateException`

Beim Methodenaufruf von `Statement.executeBatch()` kann eine `BatchUpdateException` geworfen werden. Diese gehört zu einer Unterklasse von `SQLException`.

Die Klasse enthält eine einzige zusätzliche Methode `getUpdateCounts()`, welche es dem Programmierer gestattet, Zähler zu bestimmen, die etwas über die Zahl der veränderten Zeilen in den Tabellen aussagen.

1.1.13.2.1. Typischer BatchUpdateException Handler

Hier ein Beispiel für die behandlung von Batch Update Exceptions:

```
catch( BatchUpdateException bue )
  {
    bError = true;
    aiupdateCounts = bue.getUpdateCounts();

    SQLException SQLe = bue;
    while( SQLe != null)
    {
      // Exception Code

      SQLe = SQLe.getNextException();
    }
  } // end BatchUpdateException catch
  catch( SQLException SQLe )
  {
    ...

  } // end SQLException catch
```

1.1.13.3. Übung

In dieser Übung untersuchen wir einige Eigenschaften eines Batch Updates. Diese Funktion steht nur bedingt zur Verfügung, beispielsweise nicht für DB2 NT.

Zudem werden wir die Exceptions des BatchUpdates kennen lernen und einsetzen.

1.1.13.4. Lernziele

Nach dem Durcharbeiten dieser Übung sollten Sie in der Lage sein

- das Batch Update Facility einzusetzen
- BatchUpdateException Handlings zu definieren und zu interpretieren

1.1.13.5. Vorkenntnisse

Sie sollten Sie Transaktionsübung abgeschlossen haben oder die Musterlösung kennen, da wir die dortige Lösung weiterverwenden, inklusive der darin benutzten Tabellen.

1.1.13.6. Rahmenprogramme

Sie können die folgenden Property Dateien weiterverwenden:

- ConnectXP.properties
- LadeKaffeeAnbieter.properties
- KreiereKaffeeTabellen.properties

Neue Properties sind
LadeKaffeeVal.properties:

```
#PropertiesResourceBundle für XPRun Properties
#Kaffees,Gewicht und Preise
1=INSERT INTO KaffeeVal VALUES ('Colombian Supremo', 50, 135.00)
2=INSERT INTO KaffeeVal VALUES ('Colombian Supremo', 100, 245.00)
3=INSERT INTO KaffeeVal VALUES ('Ethiopian Sidamo', 50, 165.00)
4=INSERT INTO KaffeeVal VALUES ('Ethiopian Sidamo', 100, 280.00)
5=INSERT INTO KaffeeVal VALUES ('Ethiopian Yirgacheffe', 50, 175.00)
6=INSERT INTO KaffeeVal VALUES ('Ethiopian Yirgacheffe', 100, 310.00)
7=INSERT INTO KaffeeVal VALUES ('Haitian Bleu', 50, 145.00)
8=INSERT INTO KaffeeVal VALUES ('Haitian Bleu', 100, 260.00)
9=INSERT INTO KaffeeVal VALUES ('India Monsoon Malabar', 50, 155.00)
10=INSERT INTO KaffeeVal VALUES ('India Monsoon Malabar', 100, 310.00)
11=INSERT INTO KaffeeVal VALUES ('Jamaica Blue', 50, 142.00)
12=INSERT INTO KaffeeVal VALUES ('Jamaica Blue', 100, 275.00)
13=INSERT INTO KaffeeVal VALUES ('Kona Coffee', 50, 160.00)
14=INSERT INTO KaffeeVal VALUES ('Kona Coffee', 100, 290.00)
15=INSERT INTO KaffeeVal VALUES ('Mocha Java', 50, 158.00)
16=INSERT INTO KaffeeVal VALUES ('Mocha Java', 100, 305.00)
17=INSERT INTO KaffeeVal VALUES ('Peru Vill Rica', 50, 138.00)
18=INSERT INTO KaffeeVal VALUES ('Peru Vill Rica', 100, 260.00)
19=INSERT INTO KaffeeVal VALUES ('Sumatra Mandheling', 50, 165.00)
20=INSERT INTO KaffeeVal VALUES ('Sumatra Mandheling', 100, 295.00)
21=INSERT INTO KaffeeVal VALUES ('Timor Organic', 50, 185.00)
22=INSERT INTO KaffeeVal VALUES ('Timor Organic', 100, 325.00)
23=INSERT INTO KaffeeVal VALUES ('Yemen Mocha Mattari', 50, 177.00)
24=INSERT INTO KaffeeVal VALUES ('Yemen Mocha Mattari', 100, 328.00)
```

LadeKaffeeComp.properties:

```
#PropertiesResourceBundle für XPRun Properties
# Laden der Vergleichstabelle
1=INSERT INTO KaffeeComp ( ID, Kaffeesorte, Gewicht, Preis ) SELECT ID,
Kaffeesorte, Gewicht, Preis FROM KaffeeVal, KaffeeAnbieter
2=UPDATE KaffeeComp SET Preis = Preis * .98 WHERE ID = 10
3=UPDATE KaffeeComp SET Preis = Preis * .97 WHERE ID = 20
4=UPDATE KaffeeComp SET Preis = Preis * 1.00 WHERE ID = 30
5=UPDATE KaffeeComp SET Preis = Preis * 1.23 WHERE ID = 40
6=UPDATE KaffeeComp SET Preis = Preis * 1.05 WHERE ID = 50
7=UPDATE KaffeeComp SET Preis = Preis * .92 WHERE ID = 60
8=UPDATE KaffeeComp SET Preis = Preis * .95 WHERE ID = 70
9=UPDATE KaffeeComp SET Preis = Preis * .90 WHERE ID = 80
10=UPDATE KaffeeComp SET Preis = Preis * 1.55 WHERE ID = 90
11=UPDATE KaffeeComp SET Preis = Preis * .99 WHERE ID = 100
```

LadeKaffeeAnbieter.properties:

```
#PropertiesResourceBundle für Transaktionen Properties
# Laden der Anbieter Tabelle
1=INSERT INTO KaffeeAnbieter VALUES ('BeanWhackers', 10 )
2=INSERT INTO KaffeeAnbieter VALUES ('JoeBeans', 20 )
3=INSERT INTO KaffeeAnbieter VALUES ('StandardBeans', 30 )
4=INSERT INTO KaffeeAnbieter VALUES ('ExoticaBeans', 40 )
5=INSERT INTO KaffeeAnbieter VALUES ('EthiopianPlus', 50 )
6=INSERT INTO KaffeeAnbieter VALUES ('NavyBeans', 60 )
7=INSERT INTO KaffeeAnbieter VALUES ('BeanHomeLately', 70 )
8=INSERT INTO KaffeeAnbieter VALUES ('Beanies', 80 )
9=INSERT INTO KaffeeAnbieter VALUES ('Jones, Smith & Beanette', 90 )
10=INSERT INTO KaffeeAnbieter VALUES ('BeanBag', 100 )
```

Zudem können folgende Java Programme angepasst werden:

- Transaktionen.java
- XPRun.java

und das zu ergänzende XPBatchUpdate Programm:

```
package batchupdatefacility;

import java.sql.*;
import java.util.*;


public class XPBatchUpdate
{
  boolean bError = false;
  Connection con;
  DatabaseMetaData dbmd;
  int [] aiupdateCounts;
  ResourceBundle rb;
  SQLWarning sqlw;
  Statement stmt;
  String sDriver,
         sDriverKey = "CSDriver",
         sKey,
         sPassword,
         sPasswordKey ="CSPassword",
         sUpdate,
         srbName = "ConnectXP",
         srbUpdate,
         sURL,
         sURLKey="CSURL",
         sUserID,
         sUserIDKey = "CSUserID";
```

```
  public XPBatchUpdate() throws MissingResourceException,
                        ClassNotFoundException,
                        InstantiationException,
                        IllegalAccessException
  {
    // lesen des PropertyResourceBundle
    rb = ResourceBundle.getBundle( srbName );

    sDriver   = rb.getString( sDriverKey );
    sPassword = rb.getString( sPasswordKey );
    sURL      = rb.getString( sURLKey );
    sUserID   = rb.getString( sUserIDKey );

    // versuche den JDBC Treiber zu laden
    // mit newInstance
    Class.forName( sDriver ).newInstance();

  } // end Konstruktor


  public void doUpdate( String sargRBName )
              throws MissingResourceException,
                     SQLException
  {
    int    iCount = 0,
           iProcessed = 0,
           iProcessedCount = 0;

    try // bestimme Connection und Statement
    {
      con = DriverManager.getConnection ( sURL,
                                          sUserID,
                                          sPassword);

// TODO
//    prüfen Sie, ob Batch Update unterstützt wird
// ENDTODO
      con.setAutoCommit( false );

      stmt = con.createStatement();
      sqlw = con.getWarnings();
      if( sqlw != null )
      {
        handleSQLWarnings( sqlw,
                           "Conection Warnung: ",
                            null );
        con.clearWarnings();
      }

    } // end try
    catch ( SQLException SQLe)    {
      handleSQLExceptions(
         SQLe,
        "Probleme beim Verbindungsaufbau zu " + sURL + ":",
         null            );

      if( con != null)
      {
        try
        {
          con.commit();
          con.close();
        }
        catch( Exception e ) {}
      }
      // rethrow Exception
      throw SQLe;

    } // end catch
```

```
    try
    {
      Enumeration e = null;

      // lies das PropertyResourceBundle
      // für SQL Update Statements
      rb = ResourceBundle.getBundle( sargRBName );
      e = rb.getKeys();

      // Zähle die Keys - Keys starten mit 1.
      for( ; e.hasMoreElements(); iCount++ )
      {
        (e.nextElement());
      };

      // verwende <= weil die Keys mit 1 beginnen
      System.out.println("[XPBatchUpdate]Anweisungen:");
      for( int i = 1; i <= iCount; i++ )
      {
        sKey = "" + i;
        sUpdate = rb.getString( sKey );

// TODO
//      addBatch Code einfügen
// ENDTODO

      } // end for

// TODO
//    Programmcode für BatchUpdate Methode
// ENDTODO


      sqlw = stmt.getWarnings();
      if( sqlw != null )
      {
      System.err.println("Aus SQLw - " );
        handleSQLWarnings( sqlw,
                           " Statement Warnung: ",
                            sUpdate );
      }
    } // end try
    catch( MissingResourceException mre )
    {
      System.err.println(
                 "ResourceBundle Problem mit " +
                  sargRBName + ", Programm wird beendet." );
      System.err.println("Specific error: " +
                         mre.getMessage() );
      throw mre;
    }
// TODO
//    Programmcode für BatchUpdateException Handling
// ENDTODO
```

```
    catch ( SQLException SQLe)
    {
      bError = true;
      handleSQLExceptions(
         SQLe,
        "Beim Verarbeiten einer SQL Anweisung sind Probleme
aufgetreten...",
         null            );

      // rethrow Exception
      throw SQLe;
    }
    catch ( RuntimeException e )
    {
      bError = true;
      throw e; // rethrow Exception
    }
    finally
    {

// TODO
//    fügen Sie Programmcode ein, mit dem
//    angezeigt wird, welche Statements
//    submitted wurden
//    und welche Ergebnisse erzielt wurden
// ENDTODO

      try { stmt.close(); }
      catch( Exception e ) {}

      try
      {
        if( bError )
        {
          con.rollback();
          System.err.println( "\nALLE Statements fuer " +
             sargRBName + " wurden rolled back wegen " +
            "Fehlerbedingungen." );
        }
        else { con.commit(); }

        con.close();
      }
      catch( Exception e ) {}
    } // end finally clause

  } // end doUpdate


  public void handleSQLExceptions( SQLException SQLe,
                                   String       s,
                                   String       sSQL )
  {
    boolean bFirstPass = true;

    while( SQLe != null)
    {
      reportSQLExceptions( SQLe, s, sSQL );

      if( bFirstPass )
      {
        s = sSQL = null;
        bFirstPass = false;
      }

      SQLe = SQLe.getNextException();
    }
  } // end handleSQLExceptions
```

```
public void handleSQLWarnings( SQLWarning SQLw,
                               String        s,
                               String     sSQL )
{
  boolean bFirstPass = true;

  if( s == null ) { s = "SQLWarning:";  }

  while( SQLw != null)
  {
    reportSQLExceptions( SQLw, s, sSQL );
    if( bFirstPass )
    {
      s = sSQL = null;
      bFirstPass = false;
    }
    SQLw = SQLw.getNextWarning();
  }
} // end handleSQLWarnings


public void reportSQLExceptions( SQLException SQLe,
                                 String       s,
                                 String       sSQL )
{
  String sSQLState = null;

  if( sSQL != null )
  { // SQL Problem anzeigen
    try
    {
      System.err.println(
         con.nativeSQL( sUpdate ) );
    }
    catch( Exception e) { /* na sowas */ }
  }

  if( s != null )
  { // Programmtext ausgeben, falls möglich
    System.err.println( s );
  }

  sSQLState = SQLe.getSQLState();

  // Report error information
  System.err.println( SQLe.getMessage() );
  System.err.println( "SQL State: " +
                      sSQLState );
  System.err.println( "Vendor Error Code: " +
                      SQLe.getErrorCode() );
  // check for Data Truncation
  if( sSQLState.equals( "01004") )
  {
    DataTruncation dt = (DataTruncation)(SQLe);

    System.err.println( "Data Size: " +
                        dt.getDataSize() );
    System.err.println( "Transfer Size: " +
                       dt.getTransferSize() );
    System.err.println( "Der Index von " +
                        ( dt.getParameter()
                           ? "Parameter "
                           : "Spalte " ) +
                        "war " +
                        dt.getIndex() + "." );

  }  // end if Data Truncation
} // end reportSQLExceptions
```

```
  public static void main (String args[])
  {
    boolean bContinue = true;
    XPBatchUpdate xpApp = null;

    if( args.length != 1 )
    {
      System.err.println("Usage: " +
        "java XPBatchUpdate <SQLUpdateResourceBundleName>" );
      return;
    }

    try
    {
      xpApp = new XPBatchUpdate();
    }
    catch( Exception e )
    {
      System.err.println("Konstruktor Exception: " +
                         e.getMessage() );
      bContinue = false;
    }

    if( bContinue )
    {
      try
      {
        xpApp.doUpdate( args[0] );
      }
      catch( Exception e ) {}
    }

  } // end main

} // end class XPBatchUpdate
```

1.1.13.7. Aufgaben

Sie finden weiter unten die Lösungshinweise / Lösungen zu den Aufgaben:

1) in der `doUpdate()` Methode :
   bestimmen Sie die `DatabaseMetaData` und speichern Sie diese in `dbmd`;
   bestimmen Sie damit, ob Batch Updates unterstützt werden.
   Falls nicht, dass zeigen Sie dies an und beenden Sie das Programm. Vergessen Sie nicht alle offenen Verbindungen zu schliessen!

2) in der `doUpdate()` Methode :
   ergänzen Sie das Programm durch Programmcode, der das SQL Statement `sUpdate` der Stamenetliste hinzufügt.

3) in der `doUpdate()` Methode :
   führen Sie den Batch Update aus! Die Anzahl betroffener Zeilen sollten in der Variable `aiupdateCounts` abgespeichert werden.

4) in der `doUpdate()` Methode :
   fügen Sie einen `catch` Block hinzu, mit dem die `BatchUpdateException` behandelt wird.
   Das Ergebnis des Methodenaufrufes
   `BatchUpdateException.UpdateCounts()` wird in `aiupdateCounts` abgespeichert.

5) in der `doUpdate()` Methode :
   fügen Sie einen `finally` Block hinzu, mit dem das Ergebnis des Batch Updates angezeigt wird.
   Zeigen Sie alle Ergebnisse für jedes Element aus dem `aiupdateCounts` Array an.

1.1.13.8. Lösungshinweise

Zu den einzelnen Aufgaben oben finden Sie die fehlenden Programmzeilen:

```
1) dbmd = con.getMetaData();
   if( !dbmd.supportsBatchUpdates() ) {
      System.err.println( "XPBatchUpdate Fehler: Treiber " +
                          "unterstuetzt Batch Updates nicht." );
      System.err.println( "Programm wird beendet..." );
      try { con.close(); }catch( Exception e ) {}
      return;
   }

2) System.out.println("[XPBatchUpdate]"+sUpdate);
   stmt.addBatch( sUpdate );

3) aiupdateCounts = stmt.executeBatch();

4) catch( BatchUpdateException bue )  {
      bError = true;
      aiupdateCounts = bue.getUpdateCounts();
      handleSQLExceptions(bue, "BatchUpdateException:", null);
      throw bue; // rethrow Exception
   }

5) System.err.println( "Anzahl Updates: " + iCount +
                       " Statements wurden submitted.");

   System.err.println( "Fuer Statement Nummer: ");
   for (int i = 0; i < aiupdateCounts.length; i++)      {
     iProcessed = aiupdateCounts[i];
     if( iProcessed == -3 )        {
       System.err.println( ( i+1 ) + " war fehlerhaft." );
       bError = true; // sollte true sein, zur Sicherheit neu setzen
       continue;
     }
     if( iProcessed == -2 )        {
       System.err.println( ( i+1 ) +  " war erfolgreich, aber " +
         "die Anzahl betroffener Zeilen ist unbekannt." );
       continue;
     }
     if( iProcessed == 0 )  {
       System.err.println( ( i+1 ) +  " XPBatchUpdate Rollback, " +
         "die Anzahl betroffener Zeilen ist unbekannt." );
       bError = true;
       continue;
     }
     if( iProcessed > 0 )        {
       System.err.println( ( i+1 ) +
         " war erfolgreich, Anzahl betroffener Zeilen: " +
          iProcessed );
       iProcessedCount += iProcessed;
       continue;
     }
     System.err.println( ( i+1 ) +
       " Resultat ist unbekannt, die Anzahl betroffener Zeilen ist: " +
        iProcessed + ", (diese werden im Total nicht beruecksichtig)." );
   } // end for

   System.err.println( aiupdateCounts.length +
     " Statements verarbeitete, " + iProcessedCount +
     " betroffene Zeilen. ");
```

1.1.13.9. Musterlösung

Die Property Dateien haben wir bereits gesehen.
Einzig `BatchUpdateTest.properties` fehlt noch:

```
#PropertiesResourceBundle für Batch Updates Properties
# Löschen und neues Laden der Anbieter Table
1=DELETE FROM KaffeeAnbieter
2=INSERT INTO KaffeeAnbieter VALUES ('JoeBeans', 20 )
3=INSERT INTO KaffeeAnbieter VALUES ('StandardBeans', 30 )
4=INSERT INTO KaffeeAnbieter VALUES ('ExoticaBeans', 40 )
5=INSERT INTO KaffeeAnbieter VALUES ('EthiopianPlus', 50 )
6=INSERT INTO KaffeeAnbieter VALUES ('NavyBeans', 60 )
7=INSERT INTO KaffeeAnbieter VALUES ('BeanHomeLately', 70 )
8=INSERT INTO KaffeeAnbieter VALUES ('Beanies', 80 )
9=INSERT INTO KaffeeAnbieter VALUES ('Jones, Smith & Beanette', 90 )
10=INSERT INTO KaffeeAnbieter VALUES ('BeanBag', 100 )
11=INSERT INTO KaffeeAnbieter VALUES ('BeanWhackers', 10 )
```

Dies beschreibt eine einfache Batch Update Funktion, bei der zuerst die Daten gelöscht, und anschliessend wieder eingefügt werden.

Eine fehlerhafte Update Spezifikation könnte folgendermassen aussehen:
FehlerhafterBatchUpdateTest.properties

```
#PropertiesResourceBundle für Batch Updates Properties
# Löschen und neues Laden der Anbieter Table
1=DELETE FROM KaffeeAnbieter
2=INSERT INTO KaffeeAnbieter VALUES ('JoeBeans', 20 )
3=INSERT INTO KaffeeAnbieter VALUES ('StandardBeans', 30 )
4=INSERT INTO KaffeeAnbieter VALUES ('ExoticaBeans', 40 )
5=INSERT INTO KaffeeAnbieter VALUES ('EthiopianPlus', 50 )
6=INSERT INTO KaffeeAnbieter VALUES ('NavyBeans', 60 )
7=INSERT INTO KaffeeAnbieter VALUES ('BeanHomeLately', 70 )
8=INSERT INTO KaffeeAnbieter VALUES ('Beanies', 80 )
9=INSERT INTO KaffeeAnbieter VALUES ('Jones, Smith & Beanette', 90 )
10=INSERT INTO KaffeeAnbieter VALUES ('BeanBag', 100 )
# fehlerhafte Zeile (Zwei Zeichenketten)
11=INSERT INTO KaffeeAnbieter VALUES ('ABC', 'BeanWhackers', 10 )
```

1.1.13.10. Demonstration

Die Reihenfolge der einzelnen Schritte ist durch die Batch Dateien auf dem Server / der CD gegeben. Sie sollten diese strikt einhalten, da die Tabelle KaffeeComp mit Daten aus den anderen Tabellen gefüllt wird. Falls diese nicht vorhanden oder leer sind, kann die Demo nicht funktionieren.

Hier die einzelnen Batch Jobs und Beispielausgaben

1) Starten von Cloudscape
2) Kreieren der Tabellen und einfügen von Daten:
   2KreierenDerTestKaffeeTabellen.bat
   Drop Table ...
   Create Table ...
   Insert into KaffeeVal ....
   Insert into KaffeeAnbieter ...
   ... erst dann
   Insert into KaffeeComp

2) BatchUpdate
   oder
   FehlerhafterBatchUpdate

Protokolle für die einzelnen Schritte im zweiten Batch Job.
Schritt 1:

```
Loeschen der eventuell schon vorhandenen Hilfs-Tabellen
[XPRun]Anzahl SQLAnweisungen : 3
[XPRun]SQLAnweisung : DROP TABLE KaffeeVal
DROP TABLE KaffeeVal
Probleme mit executeUpdate:
Table 'KAFFEEVAL' does not exist.
SQL State: 42X05
Vendor Error Code: 20000
```

Dieses Programm könnte man verbessern: eventuell sollte nach dem ersten Fehler weiter gefahren werden. Aber darüber kann man geteilter Meinung sein.

Schritt 2:

```
[XPRun]Anzahl SQLAnweisungen : 3
[XPRun]SQLAnweisung : CREATE TABLE KaffeeVal ( Kaffeesorte VARCHAR (25) NOT NULL, Gewicht
INTEGER NOT NULL, Preis NUMERIC(5, 2) NOT NULL )
XPRun konnte einen Test nicht durchfuehren:
CREATE TABLE KaffeeVal ( Kaffeesorte VARCHAR (25) NOT NULL, Gewicht INTEGER NOT NULL, Prei
s NUMERIC(5, 2) NOT NULL ).
Ignorieren Sie die obige Meldung falls Sie lediglich DDL Anweisungen ausfuehrten.

[XPRun]SQLAnweisung : CREATE TABLE KaffeeAnbieter ( Name VARCHAR (25) NOT NULL, ID INTEGER
 NOT NULL, PRIMARY KEY ( ID ) )
XPRun konnte einen Test nicht durchfuehren:
CREATE TABLE KaffeeAnbieter ( Name VARCHAR (25) NOT NULL, ID INTEGER NOT NULL, PRIMARY KEY
 ( ID ) ).
Ignorieren Sie die obige Meldung falls Sie lediglich DDL Anweisungen ausfuehrten.

[XPRun]SQLAnweisung : CREATE TABLE KaffeeComp ( ID INTEGER NOT NULL, Kaffeesorte VARCHAR (
25) NOT NULL, Gewicht INTEGER NOT NULL, Preis NUMERIC(5, 2) NOT NULL, PRIMARY KEY ( ID, Ka
ffeesorte, Gewicht ) )
XPRun konnte einen Test nicht durchfuehren:
CREATE TABLE KaffeeComp ( ID INTEGER NOT NULL, Kaffeesorte VARCHAR (25) NOT NULL, Gewicht
INTEGER NOT NULL, Preis NUMERIC(5, 2) NOT NULL, PRIMARY KEY ( ID, Kaffeesorte, Gewicht ) ).
Ignorieren Sie die obige Meldung falls Sie lediglich DDL Anweisungen ausfuehrten.

0 Zeilen verarbeitet.
```

Schritt 3:

```
[XPRun]Anzahl SQLAnweisungen : 24
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Colombian Supremo', 50, 135.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Colombian Supremo', 100, 245.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Ethiopian Sidamo', 50, 165.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Ethiopian Sidamo', 100, 280.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Ethiopian Yirgacheffe', 50, 175.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Ethiopian Yirgacheffe', 100, 310.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Haitian Bleu', 50, 145.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Haitian Bleu', 100, 260.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('India Monsoon Malabar', 50, 155.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('India Monsoon Malabar', 100, 310.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Jamaica Blue', 50, 142.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Jamaica Blue', 100, 275.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Kona Coffee', 50, 160.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Kona Coffee', 100, 290.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Mocha Java', 50, 158.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Mocha Java', 100, 305.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Peru Vill Rica', 50, 138.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Peru Vill Rica', 100, 260.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Sumatra Mandheling', 50, 165.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Sumatra Mandheling', 100, 295.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Timor Organic', 50, 185.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Timor Organic', 100, 325.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Yemen Mocha Mattari', 50, 177.00)
[XPRun]SQLAnweisung : INSERT INTO KaffeeVal VALUES ('Yemen Mocha Mattari', 100, 328.00)
24 Zeilen verarbeitet.
```

Schritt 4:

```
[XPRun]Anzahl SQLAnweisungen : 10
[XPRun]SQLAnweisung : INSERT INTO KaffeeAnbieter VALUES ('BeanWhackers', 10 )
[XPRun]SQLAnweisung : INSERT INTO KaffeeAnbieter VALUES ('JoeBeans', 20 )
[XPRun]SQLAnweisung : INSERT INTO KaffeeAnbieter VALUES ('StandardBeans', 30 )
[XPRun]SQLAnweisung : INSERT INTO KaffeeAnbieter VALUES ('ExoticaBeans', 40 )
[XPRun]SQLAnweisung : INSERT INTO KaffeeAnbieter VALUES ('EthiopianPlus', 50 )
[XPRun]SQLAnweisung : INSERT INTO KaffeeAnbieter VALUES ('NavyBeans', 60 )
[XPRun]SQLAnweisung : INSERT INTO KaffeeAnbieter VALUES ('BeanHomeLately', 70 )
[XPRun]SQLAnweisung : INSERT INTO KaffeeAnbieter VALUES ('Beanies', 80 )
[XPRun]SQLAnweisung : INSERT INTO KaffeeAnbieter VALUES ('Jones, Smith & Beanette', 90 )
[XPRun]SQLAnweisung : INSERT INTO KaffeeAnbieter VALUES ('BeanBag', 100 )
10 Zeilen verarbeitet.
```

Schritt 5:

```
[XPRun]Anzahl SQLAnweisungen : 11
[XPRun]SQLAnweisung : INSERT INTO KaffeeComp ( ID, Kaffeesorte, Gewicht, Preis ) SELECT ID
, Kaffeesorte, Gewicht, Preis FROM KaffeeVal, KaffeeAnbieter
[XPRun]SQLAnweisung : UPDATE KaffeeComp SET Preis = Preis * .98 WHERE ID = 10
[XPRun]SQLAnweisung : UPDATE KaffeeComp SET Preis = Preis * .97 WHERE ID = 20
[XPRun]SQLAnweisung : UPDATE KaffeeComp SET Preis = Preis * 1.00 WHERE ID = 30
[XPRun]SQLAnweisung : UPDATE KaffeeComp SET Preis = Preis * 1.23 WHERE ID = 40
[XPRun]SQLAnweisung : UPDATE KaffeeComp SET Preis = Preis * 1.05 WHERE ID = 50
[XPRun]SQLAnweisung : UPDATE KaffeeComp SET Preis = Preis * .92 WHERE ID = 60
[XPRun]SQLAnweisung : UPDATE KaffeeComp SET Preis = Preis * .95 WHERE ID = 70
[XPRun]SQLAnweisung : UPDATE KaffeeComp SET Preis = Preis * .90 WHERE ID = 80
[XPRun]SQLAnweisung : UPDATE KaffeeComp SET Preis = Preis * 1.55 WHERE ID = 90
[XPRun]SQLAnweisung : UPDATE KaffeeComp SET Preis = Preis * .99 WHERE ID = 100
480 Zeilen verarbeitet.
```

Protokoll des Batch Updates

```
[XPBatchUpdate]Anweisungen:
[XPBatchUpdate]DELETE FROM KaffeeAnbieter
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('JoeBeans', 20 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('StandardBeans', 30 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('ExoticaBeans', 40 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('EthiopianPlus', 50 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('NavyBeans', 60 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('BeanHomeLately', 70 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('Beanies', 80 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('Jones, Smith & Beanette', 90 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('BeanBag', 100 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('BeanWhackers', 10 )
Anzahl Updates: 11 Statements wurden submitted.
Fuer Statement Nummer:
1 war erfolgreich, Anzahl betroffener Zeilen: 10
2 war erfolgreich, Anzahl betroffener Zeilen: 1
3 war erfolgreich, Anzahl betroffener Zeilen: 1
4 war erfolgreich, Anzahl betroffener Zeilen: 1
5 war erfolgreich, Anzahl betroffener Zeilen: 1
6 war erfolgreich, Anzahl betroffener Zeilen: 1
7 war erfolgreich, Anzahl betroffener Zeilen: 1
8 war erfolgreich, Anzahl betroffener Zeilen: 1
9 war erfolgreich, Anzahl betroffener Zeilen: 1
10 war erfolgreich, Anzahl betroffener Zeilen: 1
11 war erfolgreich, Anzahl betroffener Zeilen: 1
11 Statements verarbeitete, 20 betroffene Zeilen.
```

Protokoll eines fehlerhaften Batch Updates:

```
[XPBatchUpdate]Anweisungen:
[XPBatchUpdate]DELETE FROM KaffeeAnbieter
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('JoeBeans', 20 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('StandardBeans', 30 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('ExoticaBeans', 40 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('EthiopianPlus', 50 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('NavyBeans', 60 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('BeanHomeLately', 70 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('Beanies', 80 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('Jones, Smith & Beanette', 90 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('BeanBag', 100 )
[XPBatchUpdate]INSERT INTO KaffeeAnbieter VALUES ('ABC', 'BeanWhackers', 10 )
BatchUpdateException:
Too many result columns specified for table 'APP.KAFFEEANBIETER'.
SQL State: 42X06
Vendor Error Code: 20000
Too many result columns specified for table 'APP.KAFFEEANBIETER'.
SQL State: 42X06
Vendor Error Code: 20000
Anzahl Updates: 11 Statements wurden submitted.
Fuer Statement Nummer:
1 war erfolgreich, Anzahl betroffener Zeilen: 10
2 war erfolgreich, Anzahl betroffener Zeilen: 1
3 war erfolgreich, Anzahl betroffener Zeilen: 1
4 war erfolgreich, Anzahl betroffener Zeilen: 1
5 war erfolgreich, Anzahl betroffener Zeilen: 1
6 war erfolgreich, Anzahl betroffener Zeilen: 1
7 war erfolgreich, Anzahl betroffener Zeilen: 1
8 war erfolgreich, Anzahl betroffener Zeilen: 1
9 war erfolgreich, Anzahl betroffener Zeilen: 1
10 war erfolgreich, Anzahl betroffener Zeilen: 1
10 Statements verarbeitete, 19 betroffene Zeilen.

ALLE Statements für FehlerhafterBatchUpdateTest wurden rolledback wegen Fehlerbedingungen.
```

1.1.14. Scrollable Result Sets

Bisher haben wir die ResultSets sequentiell benutzt, also eine Zeile nach der andern gelesen, mit `ResultSet.next()`. ResultSets haben wir bereits vielfach angetroffen, in fast jedem Programm! Bisher haben wir das ResultSet auf folgende Art und Weise gewonnen:

```
stmt = con.createStatement();
```

Dies war die einzige Methode in JDBC 1.0. In JDBC 2.0 wurden neue Methoden definiert, mit der scrollable und oder mutierbare ResultSets definiert werden können:

```
createStatement(
   int resultSetType,
   int resultSetConcurrency )
```

resultSetType kann sein:

- ResultSet.TYPE_FORWARD_ONLY
  Dies ist der Standardwert aus JDBC 1.0. Man kann nur vorwärts scrollen und die Daten lediglich einmal lesen. Sobald `ResultSet.next() false` liefert sind die Daten aus dem ResultSet nicht mehr erhältlich und wird in der Regel automatisch geschlossen.

- ResultSet.TYPE_SCROLL_INSENSITIVE
  gestattet das Kreieren von ResultSets, in denen man den Cursor, mit dem die aktuelle Position markiert wird, vorwärts und rückwärts oder zufällig bewegen.
  Die Daten sind statisch, d.h. dass jede Änderung in der Datenbank während dem bearbeiten des ResultSets nicht berücksichtigt wird.
  Das ResultSet ist insensitive gegenüber Datenmodifikationen.

- ResultSet.TYPE_SCROLL_SENSITIVE
  gestattet das Kreieren von ResultSets, in denen man den Cursor vorwärts, rückwärts oder in eine beliebige Rictung bewegen kann.
  Die Daten sind dynamisch, d.h. dass jede Änderung in der Datenbank während dem bearbeiten des ResultSets auch im ResultSet sichtbar wird.
  Das ResultSet ist sensitive gegenüber Datenmodifikationen.

resultSetConcurrency kann sein

- ResultSet.CONCUR_READ_ONLY
  Dies ist der Standardwert aus der Version JDBC 1.0

- ResultSet.CONCUR_UPDATABLE
  gestattet die Daten mittels Methoden zum ResultSet zu verändern.

Updatable ResultSets haben Vorteile und Nachteile. Wir werden nicht weiter darauf eingehen, da wir uns möglichst auf die universellen Teile des JDBC beschränen wollen.

Der Typus des ResultSets wird von einigen Treibern nicht zurückgegeben, Sie müssten also in der Anbieterdokumentation nachsehen, ob der Treiber diese ResulSets unterstützt oder nicht. `DatabaseMetaData.supportsResultSetType()` sollte Ihnen die Typen ResultSets liefern, die vomTreiber unterstützt werden;

`ResultSet.getType()` liefert den aktuellen Typus des vorliegenden ResultSets.

Sie erhalten ResultSets mittels `Statement.executeQuery()`. Falls es sich um ein ScrollableResultSet andelt, stehen Ihnen zusätzlich folgende Methoden zur Verfügung:

- absolute()
- afterLast()
- beforeFirst()
- first()
- getRow()
- isAfterLast()
- isBeforeFirst()
- isFirst()
- isLast()
- last()
- moveToCurrentRow()
  steht nur in mutierbaren ResultSets zur Verfügung.
- moveToInsertRow()
  steht nur in mutierbaren ResultSets zur Verfügung.
- previous()
- relative()

1.1.14.1. Einsatzmöglichkeiten und Hinweise

Die Funktionalität der einzelnen Treiber unterscheidet sich teils gewaltig. Sie müssen in der Regel die Dokumentation des Anbieters ansehen, damit Sie die Details kennen lernen können.

Folgende Faktoren sind zu beachten:

- der Cursor steht vor der Bearbeitung des ResultSets vor dem ersten Datensatz, genau wie bei normalen ResultSets.
- falls alle Zeilen aus dem ResultSet gelesen wurden, können einige Treiber ein commit ausführen, falls Autocommit auf `true` gesetzt wurde. Der Treiber schliesst das ResultSet und eine `SQLException` wird beim nächsten Leseversuch geworfen.
  Sie sollten in diesem Fall Autocommit auf `false` setzen.
- `ResultSet.getRow()` liefert unter Umständen null.
  Dies hat unter anderem zur Folge, dass `ResultSet.last(), ResultSet.getRow()` keine sichere Methode ist, um die letzte Zeile zu lesen.
- `ResultSet.absolute()` wirft eine `SQLException` falls Position 0 angegeben wird.
- `ResultSet.relative()` ändert die Position des Cursors nicht, falls Sie als Position 0 eingeben. Aber bei einigen DBMS Anbietern wird `ResultSet.absolute()` mittels `ResultSet.relative()` implementiert, ohne Überprüfung auf 0. Das kann zu unerwarteten Abbrüchen führen.

1.1.14.2. Übung - Paging mit Scrollable ResultSets

Diese Übung implementiert eine neue Version des TabellenSCrollers aus einer der ersten Übungen. Die Applikation stellt die Daten in Tabellenform dar und gestattet das Scrollen vorwärts und rückwärts. In einem separaten Fenster werden die Metadaten der Tabelle und des ResultSets angezeigt.

1.1.14.3. Lernziele

Nach dem Durcharbeiten dieser Übung sollten Sie

- verstehen, um was es bei Scrollable ResultSets geht und wie man ein ResultSet kreiert.
- dynamisch Daten formattieren können.
- zusätzliche Möglichkeiten der Metadaten kennen und auswerten können.

1.1.14.4. Szenario

Aus vielen Gründen hat das Entwicklunsteam beschlossen, das Viewer Programm neu zu implementieren. Weil JTable aus Swing zu starr ist und zu komplex, sollen die Daten einzeln gelesen und angezeigt werden, jeweils eine Seite. Der Treiber liefert in der Regel die Daten blockweise, wobei jeder Treiber eine eigene Blocklänge definieren kann.

Optional soll die Tabelle sortiert sein, also ein ORDER BY Feld auf dem Bildschirm angezeigt werden, womit die Sortierreihenfolge und das Sortierfeld angegeben werden kann. Auch der Tabellenname kann eingegeben werden. Der Einfachheit halber wird eine Tabelle im Programm voreingestellt (besser wäre es, dies über ein ResourceBundle zu regeln).

1.1.14.5. Voraussetzungen

Sie sollten die Übung zum interaktiven Zugriff auf Tabellendaten gelesen oder durchgearbeitet haben.

1.1.14.6. Rahmenprogramm

Sie kennen bereits die ConnectXP.properties.
Zusätzlich steht Ihnen folgendes Rahmenprogramm zur Verfügung:
ScrollResult.java:

```
package scrollresult;

import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;
import javax.swing.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */
```

```
 public class ScrollResult extends JFrame
                           implements ActionListener,
                                      WindowListener
{
  static int[] aiValidTypes =
               {
                 Types.BIGINT,
                 Types.BIT,
                 Types.CHAR,
                 Types.DATE,
                 Types.DECIMAL,
                 Types.DOUBLE,
                 Types.FLOAT,
                 Types.INTEGER,
                 Types.NUMERIC,
                 Types.REAL,
                 Types.SMALLINT,
                 Types.TIME,
                 Types.TIMESTAMP,
                 Types.VARCHAR
               };

  boolean[]   abValid;
  char[]      achHeader;
  char[]      achOutput;
  char[]      achUnderline;
  int[]       aiColumnLengths;
  String[]    asColumnLabels;

  static final int iPAGE_SIZE = 10;
  static final int iSET_PREV = iPAGE_SIZE * 2;

  int iPAGE_COUNT,
      iROW_WIDTH;

  boolean bFirst = true,
          bHasNextPage = false, bHasPrevPage = false;
  Connection con = null;
  DatabaseMetaData dbmd;
  int ndx,      ndx2;

  JButton          jbConnect = new JButton("Verbinde"),
                   jbNext = new JButton("Nächster"),
                   jbPrev = new JButton("Vorheriger"),
                   jbSelect = new JButton("Selektiere");
  JFrame           jf;
  JLabel           jlOrderBy = new JLabel("Sortiere nach:"),
                   jlUserID = new JLabel("UserID:"),
                   jlPassword = new JLabel(
                     "Passwort:"),
                   jlTable = new JLabel("Tabelle:");
  JPanel           jpCenter = new JPanel(),
                   jpNorth = new JPanel(),
                   jpNorthCenter = new JPanel(
                     new GridLayout( 3, 2) ),
                   jpSouth = new JPanel(
                      new BorderLayout() ),
                   jpSouthSouth = new JPanel();
  JPasswordField   jpfPassword =
                      new JPasswordField( 10 );
  JScrollPane      jsp,
                   jspMD;
  JTextArea        jta = new JTextArea( iPAGE_SIZE + 3,
                                        30 ),
                   jtaMD = new JTextArea( 10, 45 );
  JTextField       jtOrderBy = new JTextField( 10 ),
                   jtTable = new JTextField(
                    "KaffeeListe", 10 ),
                   jtUserID = new JTextField( 10 );
```

```
ResourceBundle rbConnect;
ResultSet rsMetaData,
          rsScroll;
ResultSetMetaData rsmd;
Statement stmt = null;
String sDriver,
       sDriverKey = "CSDriver",
       sOrderBy,
       sPassword,
       sQuery =
        "SELECT * FROM ",
       srbName = "ConnectXP",
       sTable,
       sTemp,
       sURL,
       sURLKey="CSURL",
       sUserID;


public ScrollResult()
{
  super("ScrollResult");
  addWindowListener( this );

  try  // lesen des PropertyResourceBundle
  {
    rbConnect = ResourceBundle.getBundle( srbName );

    sDriver   = rbConnect.getString( sDriverKey );
    sURL      = rbConnect.getString( sURLKey );
  }
  catch( MissingResourceException mre )
  {
    System.err.println(
               "ResourceBundle Problem bei " +
                srbName + ", Programmabbruch." );
    System.err.println("Programmfehler: " +
                       mre.getMessage() );
    endApp(); // exit on error
  }
  setLocation( new Point( 50, 100 ) );

  jf = new JFrame( "Login" );
  jf.addWindowListener( this );

  jbConnect.addActionListener( this );
  jpNorthCenter.add( jlUserID );
  jpNorthCenter.add( jtUserID );
  jpNorthCenter.add( jlPassword );
  jpNorthCenter.add( jpfPassword );
  jpNorthCenter.add( new JLabel( "" ) );
  jpNorthCenter.add( jbConnect );

  jpNorth.add( jpNorthCenter );
  Container cp = jf.getContentPane();
  cp.add( jpNorth,  BorderLayout.NORTH );
  jf.pack();

  jbNext.addActionListener( this );
  jbNext.setEnabled( false );
  jbPrev.addActionListener( this );
  jbPrev.setEnabled( false );
  jbSelect.addActionListener( this );
  jbSelect.setEnabled( false );

  jpCenter.add( jbSelect );
  jpCenter.add( jlTable );
  jpCenter.add( jtTable );
  jpCenter.add( jlOrderBy );
```

```
    jpCenter.add( jtOrderBy );

    Font font = jta.getFont();
    font = new Font( "Monospaced",
                     font.getStyle(),
                     font.getSize()
                   );
    jta.setFont( font );
    jta.setEditable( false );
    jsp = new JScrollPane( jta,
      JScrollPane.VERTICAL_SCROLLBAR_NEVER,
      JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS );

    jpSouthSouth.add( jbNext );
    jpSouthSouth.add( jbPrev );

    jpSouth.add( jsp, BorderLayout.CENTER );
    jpSouth.add( jpSouthSouth, BorderLayout.SOUTH );

    cp = getContentPane();
    cp.add( jpCenter, BorderLayout.CENTER );
    cp.add( jpSouth,  BorderLayout.SOUTH );
    pack();

    Rectangle r = getBounds();
    jf.setLocation( r.x + r.width + 20, 100 );

    show();
    jf.show();
  } // end constructor


  public void doConnect()
  {
    try // versuche den JDBC Treiber
    {   // mit newInstance zu laden
        Class.forName( sDriver ).newInstance();
    }
    catch( Exception e )  // error
    {
      jta.setText("Der Treiber konnte nicht geladen werden.");
      return;
    } // end catch
```

```
    try
    {
      con = DriverManager.getConnection ( sURL,
                                          sUserID,
                                          sPassword);
// TODO
//  speichern Sie die Metaddaten ab
//  kreieren Sie das Scrollable ResultSet
// ENDTODO

    }
    catch ( SQLException SQLe)
    {
      reportSQLError( SQLe,
        "Probleme in doConnect():" );

      if( con != null )
      {
        try
        {
          con.commit();
          con.close();
        }
        catch( Exception e ) {}
        stmt = null;
      }

      return;
    } // end catch

    jf.setVisible( false );
    jbSelect.setEnabled( true );

    jf.removeWindowListener( this );
    jf.setTitle( "MetaDaten" );

    jtaMD.setEditable( false );
    jspMD = new JScrollPane( jtaMD );
    jpNorth.remove( jpNorthCenter );
    jpNorth.add( jspMD );

    jf.pack();
    jf.show();

  } // end doConnect
```

```
  public void doRetrieval()
  {
    int iCount = 0,
        iLength = 0;

    try
    {

// TODO
//    bestimme Primärschlüssel
//    append an die TextArea
//    schliesse rsMetadata
// ENDTODO

// TODO
//    führen Sie die Abfrage aus
//    wurde mindestens eine Zeile geliefert?
//    bestimmen Sie den Scrolltypus
//    Zeigen Sie die Infos an
// ENDTODO

// TODO
//    bestimmen Sie die Metaddaten des ResultSets
//    zeigen Sie Catalog, SChema und Tabellennamen an
// ENDTODO

      iCount = rsmd.getColumnCount();
      abValid = new boolean[ iCount ];
      aiColumnLengths = new int[ iCount ];
      asColumnLabels = new String[ iCount ];

      // lies Namen, Längen, Typen
      for( ndx = 1; ndx <= iCount; ndx++ )
      {
        // bestimme das Spalten Label
        asColumnLabels[ ndx - 1 ] =
           rsmd.getColumnLabel( ndx );

        aiColumnLengths[ ndx - 1 ] =
           rsmd.getColumnDisplaySize( ndx );
        // benutze MAX(length, label_length)
        iLength = asColumnLabels[ ndx - 1 ].length();
        if( iLength > aiColumnLengths[ ndx - 1 ] )
        {
          aiColumnLengths[ ndx - 1 ] = iLength;
        }

        abValid[ ndx - 1 ] = false;
        // bestimmen des sql.Type
        int iType = rsmd.getColumnType( ndx );
        // soll die Spalte angezeigt werden?
        for( ndx2 = 0;
             ndx2 < aiValidTypes.length;
             ndx2++ )
        {
          if( iType == aiValidTypes[ ndx2 ] )
          {
            abValid[ ndx - 1 ] = true;
            break;
          }
        } // end for ndx2...
```

```
        jtaMD.append(
           rsmd.getColumnName( ndx ) +
           ( abValid[ ndx - 1 ]
             ? ""
             : " ( nicht angezeigt, )" ) +
         " ist ein Typ " +
           rsmd.getColumnType( ndx ) + " - " +
           rsmd.getColumnClassName( ndx ) +
         ", Anzeigelänge ist " +
           rsmd.getColumnDisplaySize( ndx ) + ".\n" );

      } // end for ndx...

      // bestimme Zeilengrösse
      for( ndx = 0, iROW_WIDTH = 0;
           ndx < aiColumnLengths.length;
           ndx++ )
      {
        if( !abValid[ndx] ) { continue; }
        iROW_WIDTH += aiColumnLengths[ ndx ] + 1;
      }
      iROW_WIDTH -= 1;
      achHeader = new char[iROW_WIDTH];
      achOutput = new char[iROW_WIDTH];
      achUnderline = new char[iROW_WIDTH];

      blankFill( achHeader );

      // bestimme Header
      for( ndx = ndx2 = 0;
           ndx < asColumnLabels.length;
           ndx++ )
      {
        if( !abValid[ndx] ) { continue; }
        sTemp = asColumnLabels[ ndx ];
        sTemp.getChars( 0,
                        asColumnLabels[ ndx ].length(),
                        achHeader,
                        ndx2
                      );
        ndx2 += aiColumnLengths[ ndx ] + 1;
      }
      // bestimme Underline
      for( ndx = 0; ndx < iROW_WIDTH; ndx++ )
      {
        achUnderline[ndx] = '-';
      }
      for( ndx = ndx2 = 0;
           ndx < aiColumnLengths.length - 1;
           ndx++ )
      {
        if( !abValid[ndx] ) { continue; }
        ndx2 += aiColumnLengths[ ndx ];
        // falls wir Spalten ausgelassen haben
        if( ndx2 >= achUnderline.length ) { break; }
        achUnderline[ndx2] = ' ';
        ndx2++;
      }

      rsScroll.beforeFirst();
      doNext(); // laden der ersten Seite
    } // end try
    catch ( SQLException SQLe)
    {
      reportSQLError( SQLe,
        "Probleme in DoRetrieve():" );
    }

  } // end doRetrieval
```

```
  public void doNext()
  {
    jta.setText( new String( achHeader ) +"\n" );
    jta.append( new String( achUnderline ) +"\n" );

    try
    {

// TODO
//    enablen / disablen des Rückwärts / Prev Buttons
//    bestimme und setze IPAGE_SIZE Zeilen
//    disasble / enable den Nächste / Next / Forward Button
//    positioniere den Cursor
// ENDTODO

    } // end try
    catch ( SQLException SQLe)
    {
      reportSQLError( SQLe,
        "Probleme in DoNext():" );
    } // end catch

  } // end doNext


  public void doPrev()
  {
    int i = 0;

    try
    {

// TODO
//    setze den Cursor zum Lesen der nächsten Daten
// ENDTODO

      doNext();
    } // end try
    catch ( SQLException SQLe)
    {
      reportSQLError( SQLe,
        "Probleme in DoPrev():" );
    } // end catch

  } // end doPrev


  public void loadRow() throws SQLException
  {
    blankFill( achOutput );

    for( ndx = 1, ndx2 = 0;
         ndx <= asColumnLabels.length;
         ndx++ )
    {
      if( !abValid[ ndx - 1 ] ) { continue; }
      sTemp = rsScroll.getObject( ndx ).toString();
      sTemp.getChars( 0,
                      sTemp.length(),
                      achOutput,
                      ndx2
                    );
      ndx2 += aiColumnLengths[ ndx - 1 ] + 1;
    } // end for
    jta.append( new String( achOutput ) +"\n" );

  } // end loadRow
```

```
public void blankFill( char[] ach )
{
  for( int i = 0; i < ach.length; i++ )
  { // blank init
    ach[ i ] = ' ';
  }
} // end blankFill


public void endApp()
{
  if( stmt != null)
  {
    try { stmt.close(); }
    catch( Exception e ) {}
  }
  if( con != null)
  {
    try
    {
      con.commit();
      con.close();
    }
    catch( Exception e ) {}
  }
  jf.dispose();
  dispose();
  System.exit(0);
} // end endApp
```

```
// ActionListener implementation
public void actionPerformed(ActionEvent e)
{
  Object oSource = e.getSource();

  if( oSource == jbNext )
  {
    doNext();
    return;
  }

  if( oSource == jbPrev )
  {
    doPrev();
    return;
  }

  if( oSource == jbSelect )
  {
    jta.setText( "" );
    sTable = jtTable.getText();
    sOrderBy = jtOrderBy.getText();
    if( sOrderBy.length() > 0 )
    {
      sOrderBy = "ORDER BY " + sOrderBy;
    }

    bHasNextPage = false;
    jbNext.setEnabled( false );
    bHasPrevPage = false;
    jbPrev.setEnabled( false );
    doRetrieval();
    return;
  }

  if( oSource == jbConnect )
  {
    jta.setText( "Keine Fehler." );
    sUserID = jtUserID.getText();
    sPassword = jpfPassword.getText();
    doConnect();
    return;
  }

} // end actionPerformed
```

```
  public void reportSQLError( SQLException SQLe,
                              String       s )
  {
    boolean bFirst = true;
    while( SQLe != null)
    {
      if( bFirst )
      {
        SQLe.printStackTrace();
        jta.setText( s + "\n" );
        bFirst = false;
      }

      jta.append( SQLe.getMessage() + "\n" );
      jta.append( "SQL State: " +
         SQLe.getSQLState() + "\n"  );
      jta.append( "Vendor Error Code: " +
         SQLe.getErrorCode() + "\n"  );

      SQLe = SQLe.getNextException();
    }

    bHasNextPage = false;
    jbNext.setEnabled( false );
    bHasPrevPage = false;
    jbPrev.setEnabled( false );

  } // end reportSQLError


// Window Listener Implementation
    public void windowOpened(WindowEvent e) {}

     public void windowClosing(WindowEvent e)
     {
       endApp();
     }

    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
// End Window Listener Implementation

  public static void main (String args[])
  {
    new ScrollResult();
  } // end main

} // end class ScrollResult
```

1.1.14.7. Aufgaben

Versuchen Sie folgende Aufgaben (im obigen Rahmenprogramm) zu lösen. Hilfestellungen bzw. der gesuchte Programmcode folgt weiter unten inklusive der vollständigen Lösung.

3) in `doConnect()`
bestimmen Sie, nachdem Sie die Verbindung zur Datenbank aufgebaut haben, die `DatabaseMetaData` und speichern Sie diese in `dbmd`;
kreieren Sie ein scrollable `ResultSet` mit `ResultSet.TYPE_SCROLL_INSENSITIVE` und `ResultSet.CONCUR_READ_ONLY`;
fügen Sie zudem Programmcode ein, so dass das ResultSet nicht ungeschickterweise gelöscht wird; setzen Sie Autocommit auf off / false.

4) in `doRetrieval()`
benutzen Sie das `rsMetaData ResultSet`, um Informationen über den Primärschlüssel der Tabelle `sTable` zu erhalten;
zeigen Sie die Information in der `JTextArea jtaMD` an (`append`) und schliessen Sie das `ResultSet`.

5) in `doRetrieval()`
ergänzen Sie das Programm so, dass Sie die Abfrage ausführen können:
`sQuery`, `sTable` und `sOrderBy`; lesen Sie das `ResultSet rsScroll`;
bestimmen Sie die letzte zurückgegebene Zeile;
falls Probleme auftauchen, informieren Sie den Benutzer und schliessen Sie `rsScroll`;
falls der Typus `ResultSet.TYPE_FORWARD_ONLY` ist, schliessen Sie alles und informieren Sie den Benutzer, dass das Programm nicht weiterfahren kann;
zeigen Sie den Bildschirm erneut an.

6) in `doRetrieval()`
ergänzen Sie das Programm so, dass die `ResultSetMetadata` in `rsmd` abgespeichert werden und zeigen Sie
`Catalog, Schema` und `Table` Namen an.
Danach werden die Daten mit Hilfe der Metadaten formattiert. Schauen Sie sich genau an, wie dies geschieht.

7) in `doNext()`
ergänzen Sie das Programm so, dass festgestellt werden kann, ob das rückwärts Scrollen erlaubt sein soll oder nicht (sind Sie auf der ersten Seite?);
setzen Sie den Page Tracker `bHasPrevPage`;
bestimmen Sie `iPAGE_SIZE` Zeilen;
ergänzen Sie das Programm so, dass festgestellt werden kann, ob das vorwärts Scrollen erlaubt sein soll oder nicht (sind Sie auf der letzten Seite?);
setzen Sie den Page Tracker `bHasNextPage`;
stellen Sie sicher, dass der Cursor an der richtigen Position steht, also neue Daten liefern wird.

8) in `doPrev()`
ergänzen Sie das Programm, so dass die Cursor Position korrekt gesetzt wird..

1.1.14.8. Hilfestellungen

Zu den einzelnen Aufgaben :

1)
```
dbmd = con.getMetaData();
stmt = con.createStatement(
            ResultSet.TYPE_SCROLL_INSENSITIVE,
            ResultSet.CONCUR_READ_ONLY );
con.setAutoCommit( false );
```

2)
```
rsMetaData = dbmd.getPrimaryKeys( null, null,
                        sTable.toUpperCase() );
bFirst = true;
while( rsMetaData.next() ) {
      if( bFirst ){
            jtaMD.setText( "Primärschlüssel: \n" );
            bFirst = false;
      }
      jtaMD.append( rsMetaData.getString(4) + ", Sequenz: " +
                        rsMetaData.getShort(5) + "\n" );
} // end while

if( bFirst ) {
      jtaMD.setText( "Es wurde kein Primärschlüssel " +
                  "gefunden, \nnehme an, es gibt keinen.\n" );
      bFirst = false;
}
rsMetaData.close();
```

3)
```
rsScroll = stmt.executeQuery( sQuery+sTable + " "+sOrderBy );
// prüfen, ob wenigstens eine Zeile zurück gegeben wurde
if( !rsScroll.next() ){
      jta.append( "Es wurden keine Daten gefunden.\n" );
      rsScroll.close();
      con.commit();
      return;
}
ndx = rsScroll.getType();
switch( ndx ) {
      case ResultSet.TYPE_FORWARD_ONLY:
            jtaMD.append("\nResultSet.TYPE_FORWARD_ONLY\n" );
            jtaMD.append("\ndas Programm wird abgebrochen.\n" );
            rsScroll.close();
            stmt.close();
            con.commit();
            con.close();
            return;
      case ResultSet.TYPE_SCROLL_INSENSITIVE:
            jtaMD.append("\nResultSet.TYPE_SCROLL_INSENSITIVE\n" );
            break;
      default:
            jtaMD.append("\nEin anderer ResultSet Typus.\n" );
            break;
}
```

4)
```
rsmd = rsScroll.getMetaData();
jtaMD.append( "\nKatalog: " + rsmd.getCatalogName(1) + "\n" );
jtaMD.append( "Schema: "  + rsmd.getSchemaName(1) + "\n" );
jtaMD.append( "Tabelle: " + rsmd.getTableName(1) + "\n" );
jtaMD.append( "\nSpalten: \n" );
```

5)
```
if( rsScroll.isBeforeFirst() || rsScroll.isFirst()  ) {
      bHasPrevPage = false;
      jbPrev.setEnabled( false );
} else {
      bHasPrevPage = true;
      jbPrev.setEnabled( true );
}
for( iPAGE_COUNT = 0;iPAGE_COUNT < iPAGE_SIZE;iPAGE_COUNT++) {
      if( rsScroll.next() ){
            loadRow();
      } else { break; }
} // end for

jbNext.setEnabled( false );
bHasNextPage = false;

if( iPAGE_COUNT == iPAGE_SIZE ) {
      if( rsScroll.next() ){
            jbNext.setEnabled( true );
            bHasNextPage = true;
      }
}

rsScroll.previous();
```

6)
```
if( !bHasNextPage ) {   // auf letzter Seite
      if( iPAGE_COUNT < iPAGE_SIZE )  {
            rsScroll.relative( -( iPAGE_SIZE + iPAGE_COUNT ) );
      } else  {// erste oder andere Seite
            rsScroll.relative( -(iSET_PREV) );
      }
} // end if !bHasNextPage
else   { // erste Seite
      i = rsScroll.getRow();

      if( i != 0 && ( i - iSET_PREV ) < 1 )  {
            rsScroll.beforeFirst();
      } else {
            rsScroll.relative( -(iSET_PREV) );
      }
} // end else
```

1.1.14.9. Musterlösung

ScrollResult.java:

```
package scrollresult;

import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;
import javax.swing.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */

 public class ScrollResult extends JFrame
                           implements ActionListener,
                                      WindowListener
{
  static int[] aiValidTypes =
               {
                 Types.BIGINT,
                 Types.BIT,
                 Types.CHAR,
                 Types.DATE,
                 Types.DECIMAL,
                 Types.DOUBLE,
                 Types.FLOAT,
                 Types.INTEGER,
                 Types.NUMERIC,
                 Types.REAL,
                 Types.SMALLINT,
                 Types.TIME,
                 Types.TIMESTAMP,
                 Types.VARCHAR
               };

  boolean[]   abValid;
  char[]      achHeader;
  char[]      achOutput;
  char[]      achUnderline;
  int[]       aiColumnLengths;
  String[]    asColumnLabels;

  static final int iPAGE_SIZE = 10;
  static final int iSET_PREV = iPAGE_SIZE * 2;

  int iPAGE_COUNT,
      iROW_WIDTH;

  boolean bFirst = true,
          bHasNextPage = false,
          bHasPrevPage = false;
  Connection con = null;
  DatabaseMetaData dbmd;
  int ndx,
      ndx2;

  JButton         jbConnect = new JButton("Verbinde"),
                  jbNext = new JButton("Nächster"),
                  jbPrev = new JButton("Vorheriger"),
                  jbSelect = new JButton("Selektiere");
  JFrame          jf;
```

```
JLabel          jlOrderBy = new JLabel("Sortiere nach:"),
                 jlUserID = new JLabel("UserID:"),
                 jlPassword = new JLabel(
                   "Passwort:"),
                 jlTable = new JLabel("Tabelle:");
 JPanel          jpCenter = new JPanel(),
                 jpNorth = new JPanel(),
                 jpNorthCenter = new JPanel(
                   new GridLayout( 3, 2) ),
                 jpSouth = new JPanel(
                    new BorderLayout() ),
                 jpSouthSouth = new JPanel();
 JPasswordField  jpfPassword =
                    new JPasswordField( 10 );
 JScrollPane     jsp,
                 jspMD;
 JTextArea       jta = new JTextArea( iPAGE_SIZE + 3,
                                      30 ),
                 jtaMD = new JTextArea( 10, 45 );
 JTextField      jtOrderBy = new JTextField( 10 ),
                 jtTable = new JTextField(
                  "KaffeeListe", 10 ),
                 jtUserID = new JTextField( 10 );

 ResourceBundle rbConnect;
 ResultSet rsMetaData,
           rsScroll;
 ResultSetMetaData rsmd;
 Statement stmt = null;
 String sDriver,
        sDriverKey = "CSDriver",
        sOrderBy,
        sPassword,
        sQuery =
         "SELECT * FROM ",
        srbName = "ConnectXP",
        sTable,
        sTemp,
        sURL,
        sURLKey="CSURL",
        sUserID;


 public ScrollResult()
 {
   super("ScrollResult");
   addWindowListener( this );

   try  // lesen des PropertyResourceBundle
   {
     rbConnect = ResourceBundle.getBundle( srbName );

     sDriver   = rbConnect.getString( sDriverKey );
     sURL      = rbConnect.getString( sURLKey );
   }
   catch( MissingResourceException mre )
   {
     System.err.println(
                "ResourceBundle Problem bei " +
                 srbName + ", Programmabbruch." );
     System.err.println("Programmfehler: " +
                        mre.getMessage() );
     endApp(); // exit on error
   }
   setLocation( new Point( 50, 100 ) );

   jf = new JFrame( "Login" );
   jf.addWindowListener( this );

   jbConnect.addActionListener( this );
```

```
    jpNorthCenter.add( jlUserID );
    jpNorthCenter.add( jtUserID );
    jpNorthCenter.add( jlPassword );
    jpNorthCenter.add( jpfPassword );
    jpNorthCenter.add( new JLabel( "" ) );
    jpNorthCenter.add( jbConnect );

    jpNorth.add( jpNorthCenter );
    Container cp = jf.getContentPane();
    cp.add( jpNorth,  BorderLayout.NORTH );
    jf.pack();

    jbNext.addActionListener( this );
    jbNext.setEnabled( false );
    jbPrev.addActionListener( this );
    jbPrev.setEnabled( false );
    jbSelect.addActionListener( this );
    jbSelect.setEnabled( false );

    jpCenter.add( jbSelect );
    jpCenter.add( jlTable );
    jpCenter.add( jtTable );
    jpCenter.add( jlOrderBy );
    jpCenter.add( jtOrderBy );

    Font font = jta.getFont();
    font = new Font( "Monospaced",
                     font.getStyle(),
                     font.getSize()
                   );
    jta.setFont( font );
    jta.setEditable( false );
    jsp = new JScrollPane( jta,
      JScrollPane.VERTICAL_SCROLLBAR_NEVER,
      JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS );

    jpSouthSouth.add( jbNext );
    jpSouthSouth.add( jbPrev );

    jpSouth.add( jsp, BorderLayout.CENTER );
    jpSouth.add( jpSouthSouth, BorderLayout.SOUTH );

    cp = getContentPane();
    cp.add( jpCenter, BorderLayout.CENTER );
    cp.add( jpSouth,  BorderLayout.SOUTH );
    pack();

    Rectangle r = getBounds();
    jf.setLocation( r.x + r.width + 20, 100 );

    show();
    jf.show();
  } // end constructor
```

```
public void doConnect()
{
  try // versuche den JDBC Treiber
  {   // mit newInstance zu laden
      Class.forName( sDriver ).newInstance();
  }
  catch( Exception e )  // error
  {
    jta.setText("Der Treiber konnte nicht geladen werden.");
    return;
  } // end catch

  try
  {
    con = DriverManager.getConnection ( sURL,
                                        sUserID,
                                        sPassword);
    dbmd = con.getMetaData();

    stmt = con.createStatement(
      ResultSet.TYPE_SCROLL_INSENSITIVE,
      ResultSet.CONCUR_READ_ONLY );
    con.setAutoCommit( false );
  }
  catch ( SQLException SQLe)
  {
    reportSQLError( SQLe,
      "Probleme in doConnect():" );

    if( con != null )
    {
      try
      {
        con.commit();
        con.close();
      }
      catch( Exception e ) {}
      stmt = null;
    }

    return;
  } // end catch

  jf.setVisible( false );
  jbSelect.setEnabled( true );

  jf.removeWindowListener( this );
  jf.setTitle( "MetaDaten" );

  jtaMD.setEditable( false );
  jspMD = new JScrollPane( jtaMD );
  jpNorth.remove( jpNorthCenter );
  jpNorth.add( jspMD );

  jf.pack();
  jf.show();

} // end doConnect
```

```
public void doRetrieval()
{
  int iCount = 0,
      iLength = 0;

  try
  {
    rsMetaData = dbmd.getPrimaryKeys( null, null,
       sTable.toUpperCase() );

    bFirst = true;
    while( rsMetaData.next() )
    {
      if( bFirst )
      {
        jtaMD.setText( "Primärschlüssel: \n" );
        bFirst = false;
      }
      jtaMD.append( rsMetaData.getString(4) +
        ", Sequenz: " + rsMetaData.getShort(5) +
        "\n" );
    } // end while

    if( bFirst )
    {
      jtaMD.setText( "Es wurde kein Primärschlüssel " +
        "gefunden, \nnehme an, es gibt keinen.\n" );
      bFirst = false;
    }
    rsMetaData.close();

    rsScroll = stmt.executeQuery( sQuery +
                                  sTable + " " +
                                  sOrderBy );

    // prüfen, ob wenigstens eine Zeile zurück gegeben wurde
    if( !rsScroll.next() )
    {
      jta.append( "Es wurden keine Daten gefunden.\n" );
      rsScroll.close();
      con.commit();
      return;
    }

    ndx = rsScroll.getType();
    switch( ndx )
    {
      case ResultSet.TYPE_FORWARD_ONLY:
           jtaMD.append(
          "\nResultSet.TYPE_FORWARD_ONLY\n" );
           jtaMD.append(
          "\ndas Programm wird abgebrochen.\n" );
            rsScroll.close();
            stmt.close();
            con.commit();
            con.close();
            return;

      case ResultSet.TYPE_SCROLL_INSENSITIVE:
           jtaMD.append(
          "\nResultSet.TYPE_SCROLL_INSENSITIVE\n" );
           break;
      default:
           jtaMD.append(
          "\nEin anderer ResultSet Typus.\n" );
           break;
    }

    rsmd = rsScroll.getMetaData();
```

```
jtaMD.append( "\nKatalog: " +
   rsmd.getCatalogName(1) + "\n" );
jtaMD.append( "Schema: "  +
   rsmd.getSchemaName(1) + "\n" );
jtaMD.append( "Tabelle: " +
   rsmd.getTableName(1) + "\n" );
jtaMD.append( "\nSpalten: \n" );

iCount = rsmd.getColumnCount();
abValid = new boolean[ iCount ];
aiColumnLengths = new int[ iCount ];
asColumnLabels = new String[ iCount ];

// lies Namen, Längen, Typen
for( ndx = 1; ndx <= iCount; ndx++ )
{
  // bestimme das Spalten Label
  asColumnLabels[ ndx - 1 ] =
     rsmd.getColumnLabel( ndx );

  aiColumnLengths[ ndx - 1 ] =
     rsmd.getColumnDisplaySize( ndx );
  // benutze MAX(length, label_length)
  iLength = asColumnLabels[ ndx - 1 ].length();
  if( iLength > aiColumnLengths[ ndx - 1 ] )
  {
    aiColumnLengths[ ndx - 1 ] = iLength;
  }

  abValid[ ndx - 1 ] = false;
  // bestimmen des sql.Type
  int iType = rsmd.getColumnType( ndx );
  // soll die Spalte angezeigt werden?
  for( ndx2 = 0;
       ndx2 < aiValidTypes.length;
       ndx2++ )
  {
    if( iType == aiValidTypes[ ndx2 ] )
    {
      abValid[ ndx - 1 ] = true;
      break;
    }
  } // end for ndx2...

  jtaMD.append(
     rsmd.getColumnName( ndx ) +
     ( abValid[ ndx - 1 ]
       ? ""
       : " ( nicht angezeigt, )" ) +
    " ist ein Typ " +
     rsmd.getColumnType( ndx ) + " - " +
     rsmd.getColumnClassName( ndx ) +
    ", Anzeigelänge ist " +
     rsmd.getColumnDisplaySize( ndx ) + ".\n" );

} // end for ndx...

// bestimme Zeilengrösse
for( ndx = 0, iROW_WIDTH = 0;
     ndx < aiColumnLengths.length;
     ndx++ )
{
  if( !abValid[ndx] ) { continue; }
  iROW_WIDTH += aiColumnLengths[ ndx ] + 1;
}
iROW_WIDTH -= 1;
achHeader = new char[iROW_WIDTH];
achOutput = new char[iROW_WIDTH];
achUnderline = new char[iROW_WIDTH];
```

```
      blankFill( achHeader );

      // bestimme Header
      for( ndx = ndx2 = 0;
           ndx < asColumnLabels.length;
           ndx++ )
      {
        if( !abValid[ndx] ) { continue; }
        sTemp = asColumnLabels[ ndx ];
        sTemp.getChars( 0,
                        asColumnLabels[ ndx ].length(),
                        achHeader,
                        ndx2
                      );
        ndx2 += aiColumnLengths[ ndx ] + 1;
      }
      // bestimme Underline
      for( ndx = 0; ndx < iROW_WIDTH; ndx++ )
      {
        achUnderline[ndx] = '-';
      }
      for( ndx = ndx2 = 0;
           ndx < aiColumnLengths.length - 1;
           ndx++ )
      {
        if( !abValid[ndx] ) { continue; }
        ndx2 += aiColumnLengths[ ndx ];
        // falls wir Spalten ausgelassen haben
        if( ndx2 >= achUnderline.length ) { break; }
        achUnderline[ndx2] = ' ';
        ndx2++;
      }

      rsScroll.beforeFirst();
      doNext(); // laden der ersten Seite
    } // end try
    catch ( SQLException SQLe)
    {
      reportSQLError( SQLe,
        "Probleme in DoRetrieve():" );
    }

  } // end doRetrieval
```

```
public void doNext()
{
  jta.setText( new String( achHeader ) +"\n" );
  jta.append( new String( achUnderline ) +"\n" );

  try
  {
    if( rsScroll.isBeforeFirst() ||
        rsScroll.isFirst()  )
    {
      bHasPrevPage = false;
      jbPrev.setEnabled( false );
    }
    else
    {
      bHasPrevPage = true;
      jbPrev.setEnabled( true );
    }

    for( iPAGE_COUNT = 0;
         iPAGE_COUNT < iPAGE_SIZE;
         iPAGE_COUNT++
       )
    {
      if( rsScroll.next() )
      {
        loadRow();
      }
      else { break; }
    } // end for

    jbNext.setEnabled( false );
    bHasNextPage = false;
    if( iPAGE_COUNT == iPAGE_SIZE )
    {
      if( rsScroll.next() )
      {
        jbNext.setEnabled( true );
        bHasNextPage = true;
      }
    }

    rsScroll.previous();

  } // end try
  catch ( SQLException SQLe)
  {
    reportSQLError( SQLe,
      "Probleme in DoNext():" );
  } // end catch

} // end doNext
```

```
public void doPrev()
{
  int i = 0;

  try
  {
    if( !bHasNextPage ) // auf der letzten Seite
    {
      if( iPAGE_COUNT < iPAGE_SIZE )
      {
        rsScroll.relative(
           -( iPAGE_SIZE + iPAGE_COUNT ) );
      }
      else // erste oder andere Seite
      {
        rsScroll.relative( -(iSET_PREV) );
      }

    } // end if !bHasNextPage
    else // erste Seite
    {
      i = rsScroll.getRow();

      if( i != 0 && ( i - iSET_PREV ) < 1 )
      {
        rsScroll.beforeFirst();
      }
      else
      {
        rsScroll.relative( -(iSET_PREV) );
      }
    } // end else

    doNext();
  } // end try
  catch ( SQLException SQLe)
  {
    reportSQLError( SQLe,
      "Probleme in DoPrev():" );
  } // end catch

} // end doPrev
```

```
public void loadRow() throws SQLException
{
  blankFill( achOutput );

  for( ndx = 1, ndx2 = 0;
       ndx <= asColumnLabels.length;
       ndx++ )
  {
    if( !abValid[ ndx - 1 ] ) { continue; }
    sTemp = rsScroll.getObject( ndx ).toString();
    sTemp.getChars( 0,
                    sTemp.length(),
                    achOutput,
                    ndx2
                  );
    ndx2 += aiColumnLengths[ ndx - 1 ] + 1;
  } // end for
  jta.append( new String( achOutput ) +"\n" );

} // end loadRow


public void blankFill( char[] ach )
{
  for( int i = 0; i < ach.length; i++ )
  { // blank init
    ach[ i ] = ' ';
  }
} // end blankFill


public void endApp()
{
  if( stmt != null)
  {
    try { stmt.close(); }
    catch( Exception e ) {}
  }
  if( con != null)
  {
    try
    {
      con.commit();
      con.close();
    }
    catch( Exception e ) {}
  }
  jf.dispose();
  dispose();
  System.exit(0);
} // end endApp


// ActionListener implementation
public void actionPerformed(ActionEvent e)
{
  Object oSource = e.getSource();

  if( oSource == jbNext )
  {
    doNext();
    return;
  }

  if( oSource == jbPrev )
  {
    doPrev();
    return;
  }
```

```
    if( oSource == jbSelect )    {
      jta.setText( "" );
      sTable = jtTable.getText();
      sOrderBy = jtOrderBy.getText();
      if( sOrderBy.length() > 0 ){
        sOrderBy = "ORDER BY " + sOrderBy;
      }

      bHasNextPage = false;
      jbNext.setEnabled( false );
      bHasPrevPage = false;
      jbPrev.setEnabled( false );
      doRetrieval();
      return;
    }

    if( oSource == jbConnect )    {
      jta.setText( "Keine Fehler." );
      sUserID = jtUserID.getText();
      sPassword = jpfPassword.getText();
      doConnect();
      return;
    }

  } // end actionPerformed


  public void reportSQLError( SQLException SQLe, String       s )  {
    boolean bFirst = true;
    while( SQLe != null)    {
      if( bFirst )      {
        SQLe.printStackTrace();
        jta.setText( s + "\n" );
        bFirst = false;
      }

      jta.append( SQLe.getMessage() + "\n" );
      jta.append( "SQL State: " +
         SQLe.getSQLState() + "\n"  );
      jta.append( "Vendor Error Code: " +
         SQLe.getErrorCode() + "\n"  );

      SQLe = SQLe.getNextException();
    }
    bHasNextPage = false;
    jbNext.setEnabled( false );
    bHasPrevPage = false;
    jbPrev.setEnabled( false );
  } // end reportSQLError

// Window Listener Implementation
    public void windowOpened(WindowEvent e) {}

     public void windowClosing(WindowEvent e)
     {
       endApp();
     }
    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
// End Window Listener Implementation

  public static void main (String args[])  {
    new ScrollResult();
  } // end main
} // end class ScrollResult
```

1.1.14.10. Demonstration

In diesem GUI sehen Sie als erstes zwei Bildschirme, einen zum Anmelden und einen, um die Daten zu präsentieren:

da Cloudscape keinerlei Benutzervalidation durchführt, können Sie irgend einen Benutzernamen und ein Passwort eingeben.

Gleichzeitig wird eine leerer Maske angezeigt, in die das Ergebnis der Abfrage geschrieben wird. Damit Sie nicht zu lange nach einer Tabelle suchen müssen, steht eine im Textfeld.

Selektieren können Sie erst, nachdem Sie mit der Datenbank verbunden sind.

Sobald dies geschehen ist, wechselt der Login Bildschirm und zeigt die Metadaten der Datenbank, der Tabelle und des ResultSets an.

Beispielbildschirme mit Daten können Sie leicht selber produzieren. Starten Sie einfach die Musterlösung auf dem Server / der CD, nachdem Sie alles zu sich kopiert haben (Cloudscape muss auch gestartet werden!).

1.1.15. LOBs

JDBC 2.0 definiert neue Klassen, um die SQL3 Datentypen handlen zu können.
Diese umfassen *LOBs* oder Large OBjects.

Es wurden zwei Typen von LOBs definiert:
1) BLOBs--Binary Large OBjects und
2) CLOBs--Character Large OBjects.

Aus Sicht der relationalen Datenbanken bestehen Clob aus einer Menge Zeichen und ein Blob ist eigentlich kein Datentyp. Das einzige was man weiss ist, dass ein BLOB viele Bytes enthält. Diese können irgend etwas darstellen. Allerdings besteht eine Gefahr beim Verwenden der CLOB speziell der BLOBs, weil die Datenunabhängigkeit sicher nicht mehr gegeben ist. Ein BLOB kann beispielsweise eine Grafik oder eine binäre Datei, ein programm sein, oder auch Java Bytecode. Aber Sie können auch Audio Dateien abspeichern und diese für Bilder ausgeben, irgendwann wird's jemand merken, wenn er auch nicht erfreut sein dürfte.

1.1.15.1. Locators

Ein SQL *Locator* Typ entspricht in etwa einem Pointer oder irgend einer anderen Information, welche auf eine Entität zeigt. In der Regel werden Sie in JDBC nichts mit solchen Grössen zu tun haben (wollen). Aber es gibt Fälle, in denen der Programmierer froh ist, ein solches Konzept einsetzen zu können. Im Prinzip kann man eine Locator einsetzen, um zu beschreiben, was der Treiber in einem Array, BLOB oder CLOB finden wird. Die Daten werden nicht in ein ResultSet geladen, Sie sparen sich also viel Zeit und Ärger.

Sie verlangen LOB Daten genau dann, wenn Sie diese benötigen, also immer dann wenn Sie die Daten *materialisieren* müssen.

Das ist viel effizienter, als Daten herunterzuladen, die Sie nie benötigen werden.

1.1.15.2. Clob

Ein `Clob` ist ein JDBC Interface, welches einem SQL CLOB entspricht. Ein Clob kann mit einer `getClob()` Methode aus einem `ResultSet` oder einem `CallableStatement` erhalten werden. Ein Clob besitztMethoden, um Substrings zu bestimmen, die Länge der Daten oder die Position eines anderen Clob oder Strings in diesem Clob.

Diese Methoden können Sie einsetzen, ohne den Clob materialisieren zu müssen.

Die Meterialisierung geschieht mit der `getAsciiStream()` oder `getCharacterStream()` (Unicode) Methode. Daraus lassen sich dann brauchbare Objekte konstruieren.

Um Clobs abzuspeichern kann man `setClob()` für ein `PreparedStatement` oder `updateObject()` bei einem updatable ResultSet einsetzen.

In der Regel reichen diese Methoden, um Clobs innerhalb der Datenbank oder einer Tabelle zu verschieben. Um Clobs in eine Datenbank oder eine Tabelle einzufügen, kann man `getAsciiStream()` and `getCharacterStream()` Methoden einsetzen : `PreparedStatement`'s kennt die Methoden `setAsciiStream()` und `setCharacterStream()`.

1.1.15.3. Blob

Ein Bob ist ein JDBC Interface, welches einem SQL BLOB entspricht. Blobs kann man mit der Methode `getBlob()` erhalten, einer Methode des ResultSets oder CallableStatements.

Ein Blob selber verfügt über Methoden, beispielsweise die Anzahl Bytes, den Startpunkt eines anderen Blobs ... zu bestimmen.

Auch in diesem Fall funktionieren die Methoden wie bei den Clobs ohne Materialisierung.

Mit den Methoden `getBinaryStream()` oder `getBytes()` (für Teile eines Blobs) kann man ein Blob materialisieren und dann brauchbare Objekte daraus konstruieren, beispielsweise aus den Bytearrays.

Zum Speichern von Blobs kann man `setBlob()` des PreparedStatements oder `updateObject()` des ResultSets verwenden.

Zum Laden eines Blobs in die Tabelle kann man `getBinaryStream()` und `getBytes()` einsetzen:
`PreparedStatement.setBinaryStream()` oder
`PreparedStatement.setBytes()`.

Cloudscape unterstützt keine BLOBS, noch nicht.

Falls Sie die Übung machen oder die Programme testen wollen, benötigen Sie entweder ORACLE oder DB2/NT (oder eine andere Version von IBM DB2).

1.1.15.4. Übung - Speicher eines Bildes in einem Blob

In diesem Beispiel speichern wir ein Bild in einer Blob Spalte. Dazu verwenden wir die `PreparedStatement`'s `setBinaryStream` Methode.

1.1.15.5. Lernziele

Nach dem Durcharbeiten dieser Übung sollten Sie

- ein vertieftes Verständnis über Blobs haben
- grob verstehen, wie Blobs mit binären Daten geladen werden

Hinweis: Dieses Beispiel benötigt eine Datenbank, welche Blob Datentypen unterstützt. Die Cloudscape Datenbank aus der J2EE kann dies nicht.

Das folgende Beispiel wurde mit DB2 getestet. Sie können DB2 für verschiedene Plattformen vom IBM Web Site herunter laden.

Der CREATE Befehl kann von DBMS zu DBMS leicht unterschiedlich sein. Falls ich Zeit habe, werde ich auch mit Oracle Tests durchführen und die nötigen Änderungen noch in den Text einfügen.

Wichtig ist, dass die DBMS über einen JDBC 2.0 kompatiblen Treiber verfügt.

1.1.15.6. Szenario

Das XP_Team möchte neben Textdaten auch Bilder der T-Shirts abspeichern.
Das Entwicklungsteam beschliesst, die T-Shirts zu fotografieren und die Bilder in die Datenbank einzufügen, also Blobs.

Dafür wird eine Tabelle `TeeColor` definiert (und kreiert), welche eine Spalte mit der Bezeichnung derFarbe und eine Spalte mit einem Blob enthält.

Als erstes muss nun die Tabelle kreiert werden!

1.1.15.7. Vorausetzungen

Sie wissen, wie man eine normale Tabelle anlegt und Daten einfügt.

1.1.15.8. Programmrahmen

Ihnen stehen nicht allzugrosse JPEG Bilder als Platzhalter zur Verfügung. JPEG wurde gewählt, weil man diese in Java darstellen kann und dies ein im Internet gängiges Format ist.

- BeigeXP.jpeg
- BlackXP.jpeg
- BlueXP.jpeg
- GreenXP.jpeg
- WhiteXP.jpeg
- YellowXP.jpeg

1.1.15.9. Aufgaben

1) Treiber und Benutzernamen
   Das Programm wurde so ausgelegt, dass ein Array die Farbnamen und Bilddaten aufnehmen kann. Die Bilder stehen bereits im Projektverzeichnis. Ihre erste Aufgabe ist es, sich mit der Datenbank vertraut zu machen, die Sie zum Speichern der Blobs verwenden möchten, insbesondere Benutzernamen und Treiber feststellen und im Programm eingeben.

2) CREATE TABLE
   Kreieren Sie nun eine Tabelle , die beispielsweise in DB2 folgendermassen definiert ist:

```
CREATE TABLE TeeFarbe (
   TFarbe      VARCHAR (10)  NOT NULL,
   TFBlob      BLOB    (64K) NOT NULL,
 PRIMARY KEY( TFarbe )
)
```

   Damit können wir Blobs mit binären Daten und bis zu 64 KB speichern.

3) INSERT INTO
   Schreiben Sie ein `PreparedStatement pstmt`, mit folgender DML:

```
INSERT INTO TeeColor VALUES( ?, ? )
```

   um die Spalten in die Tabelle einfügen zu können.

4) Laden der Daten
   Schreiben Sie nun Programmcode zum Lesen der Daten und einfügen in die Tabelle. Nehmen Sie an die Daten, die Bilder, seien im selben Verzeichnis (verschieben Sie die Bilder dorthin) und definieren Sie im Programm
   - ein `File`.
   - ein `FileInputstream` für jedes Bild.
   Der erste Parameter in Ihrem PreparedStatement aus Aufgabe 3 ist die Farbe, der zweite ist ein `BinaryStream` vom `FileInputstream` mit `File.length()`.
   Die Gesamtzahl eingefügter Zeilen soll in `iRowCount` stehen.

1.1.15.10. Lösungshinweise

Zu den einzelnen Aufgaben kann man lediglich Hinweise geben sofern man die Datenbank genau kennt. Im folgenden wird angenommen, es handle sich um die DB2.

1)
```
String sDriver =   "COM.ibm.db2.jdbc.net.DB2Driver";
String sURL = "jdbc:db2://localhost:6789/XP_Team";
String sUsername = "System";
String sPassword = "Manager";
```

2)
```
CREATE TABLE TeeFarbe (
    TFarbe      VARCHAR (10)  NOT NULL,
    TFBlob      BLOB    (64K) NOT NULL,
 PRIMARY KEY( TFarbe )
)
```
Je nach DBMS, hier für DB2:
```
stmt.executeUpdate( "CREATE TABLE TeeFarbe (" +
          "TFarbe      VARCHAR (10)  NOT NULL, " +
          "TFBlob      BLOB    (64K) NOT NULL, " +
          "PRIMARY KEY( TFarbe )"               +
          ")" );
```

3)
```
pstmt = con.prepareStatement( "INSERT INTO TeeColor VALUES( ?, ? )" );
```

4)
```
for (int i = 0; i < asColors.length; i++)      {
    pstmt.setString( 1, asColors[i] );
    fImage = new File( asImages[i] );
    isImage = new FileInputStream( fImage );
    pstmt.setBinaryStream( 2, isImage, (int)( fImage.length() ) );
    iRowCount +=  pstmt.executeUpdate();
}
```

1.1.15.11. Musterlösung

CreateXPTeeFarbe.java:

```
package einlesenderblobs;

import java.io.*;
import java.sql.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */

public class CreateXPTeeFarbe {

  static String[] asColors =   {
    "Beige",
    "Black",
    "Blue",
    "Green",
    "White",
    "Yellow"
  };

  static String[] asImages =  {
    "BeigeXP.jpeg",
    "BlackXP.jpeg",
    "BlueXP.jpeg",
    "GreenXP.jpeg",
    "WhiteXP.jpeg",
    "YellowXP.jpeg"
  };

  public static void main(String[] args)  {
    Connection con = null;
    File fImage;
    InputStream isImage;
    int iRowCount = 0;
    PreparedStatement pstmt = null;
    Statement stmt = null;

// DB Treiber und Verbindungsinfo (UserID, Passwort)
    String sDriver =
      "";
    String sURL =
      "";
    String sUsername = "";
    String sPassword = "";
    //  DB2
    /*
    sDriver =   "COM.ibm.db2.jdbc.net.DB2Driver";
    sURL = "jdbc:db2://localhost:6789/XP_Team";
    sUsername = "System";
    sPassword = "Manager";
    */

    // Cloudscape : keine BLOB Unterstützung
    /*
    sDriver="COM.cloudscape.core.RmiJdbcDriver";
    sURL="jdbc:cloudscape:rmi:XP_Team";
    sUsername="System";
    sPassword="Manager";
    */
```

```
    // Oracle : folgt später
    /*
    sDriver="COM.cloudscape.core.RmiJdbcDriver";
    sURL="jdbc:cloudscape:rmi:XP_Team";
    sUsername="System";
    sPassword="Manager";
    */

// END DB Treiber und Verbindungsinfo

    try   // Versuche den JDBC Treiber
    {     // mit newInstance zu laden
        Class.forName( sDriver ).newInstance();
    }
    catch( Exception e )  // error
    {
      System.err.println(
        "Der Treiber kann nicht geladen werden.");
      return;
    } // end catch

    try {

      con = DriverManager.getConnection ( sURL,
                                          sUsername,
                                          sPassword);
      stmt = con.createStatement();
    }  catch ( Exception e) {
      System.err.println( "Probleme beim Verbindungsaufbau zu " +
                           sURL + ":" );
      System.err.println( e.getMessage() );

      if( con != null)   {
        try { con.close(); } catch( Exception e2 ) {}
      }

      return;
    } // end catch

    // Das Programm soll mehrfach gestartet werden dürfen
    // Daher werden allfällige Daten aus der DB gelöscht
    try    {
      stmt.executeUpdate( "DROP TABLE TeeFarbe" );
      System.out.println(
        "Tabelle TeeFarbe wurde entfernt.");
    } catch ( Exception e ) { /* don't care */ }


    // Ausführen der SQL Anweisung
    // um eine Tabelle zu kreieren
    try    {

// DBMS spezifisch
      stmt.executeUpdate( "CREATE TABLE TeeFarbe (" +
             "TFarbe      VARCHAR (10)  NOT NULL, " +
             "TFBlob      BLOB    (64K) NOT NULL, " +
             "PRIMARY KEY( TFarbe )"                +
                                              ")" );
// END DBMS spezifisch
      System.out.println( "Tabelle TeeFarbe wurde kreiert.");

      try { stmt.close(); } catch( Exception e ) {}
      stmt = null;

      pstmt = con.prepareStatement(
        "INSERT INTO TeeFarbe VALUES( ?, ? )" );
```

```
      for (int i = 0; i < asColors.length; i++)  {
        pstmt.setString( 1, asColors[i] );

        fImage = new File( asImages[i] );
        isImage = new FileInputStream( fImage );
        pstmt.setBinaryStream( 2, isImage, (int)( fImage.length() ) );

        iRowCount += pstmt.executeUpdate();
      }

      System.out.println( iRowCount +
            " Zeilen wurden in die Tabelle TeeFarbe eingefuegt.");
    }
    catch ( Exception e ) {
      System.err.println(
        "Probleme mit der SQL Anweisung, welche an " +
                                    sURL+" gesandt wurde:" );
      System.     err.println( e.getMessage() );
    }
    finally     {
      if( stmt != null)  {
        try { stmt.close(); } catch( Exception e2 ) {}
      }
      try { pstmt.close(); }  catch( Exception e ) {}
      try { con.close(); }  catch( Exception e ) {}
    } // end finally clause
  } // end main
} // end class CreateXPTeeFarbe
```

1.1.15.12. Demonstration

Falls alles okay läuft, werden die Bilder in Ihre BLOB fähige Tabelle eingefügt.

1.1.15.13. Lesen und Anzeigen von Bildern aus einem Blob

Diese Übung zeigt, wie Sie die gespeicherten Bilder wieder aus der DBMS herauslesen und darstellen können.

1.1.15.14. Lernziele

Nach dem Durcharbeiten dieser Übung sollten Sie

- wissen, wie Blob Daten gelesen werden können
- Bilddaten, welche in einem Blob abgespeichert sind, anzeigen können.

Bemerkung: Auch diese Übung setzt voraus, dass Sie eine Blob fähige DBMS haben, beispielsweise Oracle, DB2, ...

Definitiv nicht funktionieren die Programme mit Access, Excel oder Cloudscape.Sie benötigen Kenntnisse der Treiber und einen DB Zugang (UserID, Passwort).

1.1.15.15. Szenario

Das Entwicklungsteam entscheidet sich, ein Swing GUI zu entwickeln. Der Aufbau ist analog zu den bisherigen GUIs. Sie können mit der DBMS eine Verbindung aufbauen und die Daten aus dem Blob lesen und anzeigen.

Sie wählen die Farbe aus und das Programm sucht das Bild aus der DBMS.

1.1.15.16. Voraussetzungen

Sie können Daten aus einer Tabelle in einer Datenbank lesen und darstellen.

1.1.15.17. Programmrahmen

Ihnen steht ein Rahmenprogramm auf dem Server / der CD zur Verfügung. Allerdings müssen Sie einiges zu Ihrer DBMS selber herausfinden:

- JDBC Treiber
- Benutzername
- Passwort
- DBMS URL

1.1.15.18. Aufgaben

Versuchen Sie folgende Aufgaben zu lösen. Lösungshinweise finden Sie gleich anschliessend.

1) Sie benötigen zwei Query Strings: einen zur Auswahl aller unterschiedlichen Farben in der Tabelle und einen zweiten für die Blob Auswahl im PreparedStatement.s

   `QueryD`:

   ```
   SELECT DISTINCT TFarbe
       FROM TeeFarbe
   ```

   `sQueryP`:

   ```
   SELECT TFBlob
       FROM TeeFarbe
       WHERE TFarbe = ?
   ```

   Bauen Sie diese Strings in das Programm ein oder schauen Sie nach, wie dies geschieht.

2) Sehen Sie in `doConnect()` nach. Nachdem die Verbindung aufgebaut wurde und ein `Statement` Objekt kreiert wurde muss die `DISTINCT` Abfrage ausgeführt werden. Lesen Sie die Zeilen und laden Sie die Farben in die Combo Box oder schauen Sie nach, wie dies gemacht wird.

3) Nachdem Sie mit der DBMS verbunden sind können Sie das `Statement` Objekt schliessen und die Bildschirme aufbauen und das `prepareStatement` ausführen.
   Dies alles geschieht in der Methode `doConnect()`.

4) In der `doRetrieve()` Methode müssen Sie einen `String` setzen und das `PreparedStatement` ausführen;
   lesen Sie das `Blob` und seine Länge;
   materialieren Sie die `Blob` Daten mit `Blob.getBytes()` und benutzen Sie das resultierende ByteArray um ein `ImageIcon` mit `ImageIcon( byte[] )` zu kreieren.
   (Hinweis: in `Blob.getBytes()` starten die binären Daten ab Position 1).
   Benutzen Sie `JLabel.setIcon()` um das Bild in JLabel zu laden

5) In `EndApp()`:
   fügen Sie Programmcode ein, um die Verbindung zu schliessen.

1.1.15.19. Lösungshinweise

Die Lösungshinweise enthalten mögliche Programmfragmente. Diese können je nach DBMS unterschiedlich sein.

1)
```
sQueryD =  "SELECT DISTINCT TFarbe " +
           "FROM TeeFarbe",

sQueryP =  "SELECT TFBlob " +
           "FROM TeeFarbe" +
           "WHERE TFarbe = ?",
```

2)
```
rs = stmt.executeQuery( sQueryD );
 while( rs.next() )  {
     jcb.addItem( rs.getString( 1 ) );
}
```

3)
```
pstmt = con.prepareStatement( sQueryP );
```

4)
```
pstmt.setString( 1,  (String)(jcb.getSelectedItem()) );
rs = pstmt.executeQuery();
if( rs.next() )  {
    blob = rs.getBlob( 1 );
    iLength = (int)(blob.length());
    ii = new ImageIcon(blob.getBytes( 1, iLength ));
    jlImage.setIcon( ii );
}
```

5)
```
if( con != null)   {
    try { con.close(); } catch( Exception e ) {}
}
```

1.1.15.20. Musterlösung

```
package bloblesen;

import java.awt.*;
import java.awt.event.*;
import java.sql.*;
import java.util.*;
import javax.swing.*;

/**
 * Title:
 * Description:
 * Copyright:    Copyright (c) J.M.Joller
 * @author J.M.Joller
 * @version 1.0
 */

public class TeeFarbenXP extends JFrame
                     implements ActionListener,
                                WindowListener
{
  Blob blob;
  Connection con = null;
  int iLength;

  ImageIcon ii;

  JButton         jbConnect = new JButton("Verbinde"),
                  jbShow = new JButton("Anzeigen!");
  JComboBox       jcb = new JComboBox();
  JLabel          jlImage = new JLabel(),
                  jlUserID = new JLabel("UserID:"),
                  jlPassword = new JLabel(
                    "Passwort:");
  JPanel          jpCenter = new JPanel(),
                  jpNorth = new JPanel(),
                  jpSouth = new JPanel(
                     new BorderLayout() ),
                  jpSouthEast = new JPanel(),
                  jpSouthWest = new JPanel();
  JPasswordField  jpfPassword =
                     new JPasswordField( 10 );
  JTextArea       jta = new JTextArea(
                    "Fehlermeldungen.", 2, 30 );
  JTextField      jtUserID = new JTextField( 10 );

  PreparedStatement pstmt = null;
  ResourceBundle rbConnect;
  ResultSet rs;
  ResultSetMetaData rsmd;
  Statement stmt = null;
  String sDriver,
         sDriverKey = "CSDriver",
         sPassword,
         sQueryD =
          "SELECT DISTINCT TFarbe " +
          "FROM TeeFarbe",
         sQueryP =
          "SELECT TFBlob " +
          "FROM TeeFarbe " +
          "WHERE TFarbe = ?",
         srbName = "ConnectXP",
         sURL,
         sURLKey="CSURL",
         sUserID;
```

```
public TeeFarbenXP()
{
  super("TeeFarbenXP - Erhältliche Tee's");

  try  // Lesen des PropertyResourceBundle
  {
    rbConnect = ResourceBundle.getBundle( srbName );

    sDriver   = rbConnect.getString( sDriverKey );
    sURL      = rbConnect.getString( sURLKey );
  }
  catch( MissingResourceException mre )
  {
    System.err.println(
               "Beim Lesen des ResourceBundle " +
                srbName + " trat ein Problem auf, Programmabbruch." );
    System.err.println("Programmfehler: " +
                       mre.getMessage() );
    endApp(); // exit on error
  }

  jbConnect.addActionListener( this );
  jbShow.addActionListener( this );

  jpNorth.add( jlUserID );
  jpNorth.add( jtUserID );
  jpNorth.add( jlPassword );
  jpNorth.add( jpfPassword );

  jpCenter.add( jbConnect );
  jpCenter.add( jta );

  jpSouthWest.add( jcb );
  jpSouthWest.add( jbShow );
  jlImage.setPreferredSize(
     new Dimension( 100, 75 ) );
  jpSouthEast.add( jlImage );
  jpSouth.add( jpSouthEast, BorderLayout.EAST );
  jpSouth.add( jpSouthWest, BorderLayout.WEST );

  Container cp = getContentPane();
  cp.add( jpNorth,  BorderLayout.NORTH );
  cp.add( jpCenter, BorderLayout.CENTER );
  cp.add( jpSouth,  BorderLayout.SOUTH );

  addWindowListener( this );
  pack();

  jcb.setVisible( false );
  jbShow.setVisible( false );

  show();

} // end constructor


public void doConnect()
{
  try // Versuche den JDBC driver
  {   // mit newInstance zu laden
      Class.forName( sDriver ).newInstance();
  }
  catch( Exception e )  // error
  {
    jta.setText("Der JDBC Treiber konnte nicht geladen werden.");
    return;
  } // end catch

  try
  {
```

```
      con = DriverManager.getConnection ( sURL,
                                          sUserID,
                                          sPassword);
      stmt = con.createStatement();
      rs = stmt.executeQuery( sQueryD );
      while( rs.next() )
      {
        jcb.addItem( rs.getString( 1 ) );
      }

      try { stmt.close(); }
      catch( Exception e ) {}
      rs = null;

      jcb.setVisible( true );
      jbShow.setVisible( true );

      pstmt = con.prepareStatement( sQueryP );
      jbConnect.setEnabled( false );
    }
    catch ( SQLException SQLe)
    {
      reportSQLError( SQLe,
        "Probleme in DoConnect():" );

      if( con != null )
      {
        try { con.close(); }
        catch( Exception e ) {}
        stmt = null;
      }

      return;
    } // end catch
    finally
    {
      if( stmt != null )
      {
        try { stmt.close(); }
        catch( Exception e ) {}
      }
      rs = null;
    }
  } // end doConnect


  public void doRetrieve()
  {
    try
    {
      pstmt.setString( 1,
        (String)(jcb.getSelectedItem()) );
      rs = pstmt.executeQuery();
      if( rs.next() )
      {
        blob = rs.getBlob( 1 );

        iLength = (int)(blob.length());
        ii = new ImageIcon(
           blob.getBytes( 1, iLength )
                        );
        jlImage.setIcon( ii );
        pack();
      }
    } // end try
    catch ( SQLException SQLe)
    {
      reportSQLError( SQLe,
        "Probleme in DoRetrieve():" );
      SQLe.printStackTrace();
```

```
    }
    finally
    {
      try { rs.close(); }
      catch( Exception e ) {}
      rs = null;
    } // end finally

  } // end doRetrieve


  public void endApp()
  {
    if( con != null)
    {
      try { con.close(); }
      catch( Exception e ) {}
    }
    dispose();
    System.exit(0);
  }
  // ActionListener implementation
  public void actionPerformed(ActionEvent e)
  {
    Object oSource = e.getSource();
    jta.setText( "Keine Fehler." );
    if( oSource == jbShow )
    {
      doRetrieve();
      return;
    }

    if( oSource == jbConnect )
    {
      sUserID = jtUserID.getText();
      sPassword = jpfPassword.getText();
      doConnect();
      return;
    }

  } // end actionPerformed


  public void reportSQLError( SQLException SQLe,
                              String       s )
  {
      jta.setText( s + "\n" );
      jta.append( SQLe.getMessage() + "\n" );
      jta.append( "SQL State: " +
         SQLe.getSQLState() + "\n"  );
  } // end reportSQLError

// Window Listener Implementation
    public void windowOpened(WindowEvent e) {}

     public void windowClosing(WindowEvent e)
     {
       endApp();
     }

    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
// End Window Listener Implementation

  public static void main (String args[])  {
    new TeeFarbenXP();
  } // end main
```

```
} // end class TeeFarbenXP
```

1.1.15.21. Demonstration

Falls Sie in der glücklichen Lage sind eine DBMS mit Blobs zu haben, sehen Sie zuerst den Anmeldebildschirm und nach Auswahl einer Farbe das entsprechende Bild

In einem Combo können Sie die Farbe aussuchen. Falls Sie dann den Anzeigebutton anklicken, wird das entsprechend T-Shirt (besser: das JPEG) angezeigt.

1.1.16. SQL Konformität

Die grundlegende Anforderung an einen JDBC konformen Treiber ist, dass er ANSI SQL-92 Entry Level unterstützen muss. Dieser Level ist in etwa kompatibel zu Level 2 von SQL-89.

Neben den grundlegenden Operationen SELECT, INSERT, UPDATE und DELETE umfasst dies:

- mehrere Tabellen im FROM
- Datentypen: characterType, decimalType, integerType, smallintType, floatType, realType, doublePrecisionType und numericType.
- einfache SQL Ausdrücke: and, or, not, like, =, <>, arithmetische Funktionen, Joins, group by, having, order by und aggregierte Funktionen (wie sum, count, max, min)
- einfache Tabellen und Spaltenbeschreibungen: tableName, columnName.
- Unique und Primary Key in der Tabellenbeschreibung.
- korrelierte Subqueries und EXISTS Subqueries.
- voller Support von Distinct in Funktionen.
- Union wird unterstützt.

Details zu diesen Anforderungen finden Sie bei: FIPS PUB 127-2: The Standard for Database Language SQL (http://www.itl.nist.gov/fipspubs/fip127-2.htm).

Die DatabaseMetaData Methoden unterstützen ANSI92EntryLevelSQL(), und ANSI92IntermediateSQL() und ANSI92FullSQL() . Ein JDBC geprüfter Treiber muss true liefern, falls Sie die Methode supportsANSI92EntryLevelSQL() ausführen.

Neben dem ODBC definierten SQL Set können Sie auch mit den DatabaseMetaData Methoden `supportsMinimumSQLGrammar()`, `supportsCoreSQLGrammar()` und `supportsExtendedSQLGrammar()` die Konformität überprüfen.

ODBC ist kein reiner Microsoft Standard. ODBC SQL Konformität wird in AcuODBC SQL Conformance (http://www.cs.uleth.ca/libcommon/docs50/acuodbc/odb00071.htm) beschrieben. Weitere ODBC Informationen finden Sie unter ODBC Version 3.51 (http://msdn.microsoft.com/library/psdk/dasdk/odbc4vcn.htm).

Sie finden auf dem Sun Web Site Konformitätstests für JDBC.

1.1.17. Das JDBC 2.0 Optional Package und J2EE

In JDBC 2.0 sind zwei Packages enthalten. Wir haben lediglich die Kernklassen besprochen. Ein weiteres Package, das JDBC 2.0 Optional Package (javax.sql) finden Sie unter http://java.sun.com/products/jdbc/jdbc20.stdext.javadoc . Es umfasst ein *DataSource interface*, *Connection pooling*, *Distributed Transactions* und *Rowsets*.

Da bereits die Version 3.0 als Draft erhältlich ist, werden diese Teile vermutlich in einem weiteren Tutorial behandelt werden.

JDBC und das JDBC 2.0 Optional Package sind integraler Bestandteil der Java 2 Platform, Enterprise Edition (J2EE) http://www.javasoft.com/products/OV_enterpriseProduct.html , welche die Technologien umfasst, die man zum Aufbau von unternehmensweiten Applikatione benötigt:

- RMI
- IDL / CORBA
- Java Messaging Services
- Java Naming & Directory Services
- Enterprise Java Beans
- ...

1.1.18. Cloudscape Installation und Setup

Die Cloudscape Datenbank wird, wenigstens die Laufzeitversion, mit der Java™ 2 SDK Enterprise Edition (J2SE™) ausgeliefert. Unsere Beispiele haben (fast ausschliesslich) diese Version von Cloudscape benutzt.

Mir der J2EE kommt auch eine Anleitung und Dokumentation unter anderem über die unterstützten SQL Befehle und vieles mehr.

Es sind mindestens folgende Schritte nötig:

- die Variable J2EE_HOME muss gesetzt werden.
  Beispiel:
  set J2EE_HOME=c:\j2sdkee1.2.1
- die Variable JAVA_HOME muss gesetzt sein.
  Beispiel:
  JAVA_HOME=D:\j2se
  oder
  JAVA_HOME = c:\JBuilder4\jdk1.3

Sie sollten auf Grund der Beispiele herausfinden, wie diese Variablen auf Ihrem PC gesetzt werden müssen.

Um CLASSPATH Konflikte zu vermeiden, sollten Sie:

- die jars von J2EE_HOME/lib/cloudscape nach JAVA_HOME/jre/lib/ext verschieben:
  - cloudscape.jar
  - client.jar *rename* auf cs_client.jar
  - RmiJdbc.jar *rename* auf cs_RmiJdbc.jar
  - tools.jar *rename* auf cs_tools.jar

Das Unbenennen ist nicht zwingend. Aber Sie vermeiden damit Konflikte wegen Namensgleichkeit. Auf meinem PC habe ich die obigen Schritte weggelassen.

Weitere Dokumentation finden zur *Cloudscape DBMS* finden Sie unter J2EE_HOME/doc/cloudscape/index.html. Allerdings kann es passieren, dass die Laufzeitversion des J2EE einige der Features in der Dokumentation nicht unterstützt.

1.1.18.1. Starting und Stopping von Cloudscape

Starten und Stoppen von Coudscape geschieht auf Command Prompt Ebene. Deswegen finden Sie auf dem Server / der CD spezielle Batch Dateien, eine zum Starten und eine zum Stoppen.

- CTRL C kann zu Problemen führen. Sie sollten also jeweils die DBMS korrekt stoppen:
  J2EE_HOME/bin/cloudscape -stop
  oder die Batch Datei
- das Starten geschieht analog
  J2EE_HOME/bin/cloudscape -start

1.1.19. SQL Primer

In diesem Abschnitt finden Sie einige der verwendeten SQL Befehle, die in den Übungen verwendet wurden.

1.1.19.1. Kreieren von Tabellen

Mit der CREATE TABLE Anweisung kann man neue Tabellen kreieren. Dieser Befehl ist auch weitestgehend einheitlich implementiert, ausser bei ODBC Zugriff auf Textdateien.

```
CREATE TABLE <table name>
  (<column element> [, <column element>]...)
```

Ein <`column element`> ist:

```
<column name> <data type>
   [DEFAULT <expression>]
   [<column constraint> [, <column constraint>]...]
```

Ein <`column constraint`> ist:

```
NOT NULL |
  UNIQUE |
  PRIMARY KEY
```

Beispiel:

```
CREATE TABLE java (
   version_name varchar (30),
   major_version int,
   minor_version int,
   release_date date);
```

Mit DROP TABLE wird eine Tabelle gelöscht. Wie bei CREATE TABLE ist dieser Befehl weitestgehend einheitlich implementiert.

```
DROP TABLE <table name>
```

1.1.19.2. Zugriff auf Spalten

Mit dem SELECT Befehl können Sie auf einzelne Werte einer Spalte zugreifen. Die Daten, die Sie selektieren, können auch aus mehreren Tabellen stammen. Die meisten Befehle sind ziemlich einheitlich definiert. Aber es gibt immer einige Sonderfälle.

```
SELECT [ALL | DISTINCT] <select list>
   FROM <table reference list>
   WHERE <search condition list>
   [ORDER BY <column designator> [ASC | DESC]
          [, <column designator> [ASC | DESC]]...]
```

Die <`select list`> enthält normalerweise eine Komma-getrennte Liste von Spalten oder ein '*' , falls Sie alle Spalten lesen wollen.

```
SELECT version_name, release_date from java;
```

Falls Ihr Treiber JDBC konform ist, unterstützt er auch GROUP BY, HAVING und UNION Klauseln im SELECT.

Verknüpfungen mehrerer Tabellen werden normalerweise mit der WHERE Klausel realisiert.

```
SELECT employee_id, employee_name,
       department_table.department_id, department_name
FROM   employee_table, department_table
WHERE  employee_table.department_id =
       department_table.department_id;
```

Falls Sie in der obigen Darstellung des JOIN zuviele Tabellen sehen, dann können Sie auch Abkürzungen einführen (Alias):

```
SELECT employee_id, employee_name,
       d.department_id, department_name
FROM   employee_table e, department_table d
WHERE  e.department_id =
       d.department_id;
```

1.1.19.3. Informationen speichern

Mit der INSERT Anweisung können Sie Daten in eine Spalte schreiben.

```
INSERT INTO <table name>
  [(<column name> [, <column name>]...)]
  VALUES (<expression> [, <expression>]...)
```

Beispiel

```
INSERT INTO java VALUES
  ('2.0Beta', 2, 0, 'Aug-1-1997');
```

Sie können die Daten auch mit einer SELECT Anweisung aus einer oder mehreren anderen Tabellen selektieren.

Mit der UPDATE Anweisung können Sie eine oder mehrere Zeilen mutieren.

```
UPDATE <table name>
   SET <column name = {<expression> | NULL}
    [, <column name = {<expression> | NULL}]...
   WHERE <search condition>
```

Falls Sie Daten löschen müssen, steht Ihnen die DELETE Anweisung zur Verfügung.

```
DELETE FROM <table name>
   WHERE <search condition>
```

1.1.20. Ressourcen

Hier eine kurze Liste möglicher weiterführender Informationsquellen, die teilweise auch zum Schreiben dieses Tutorials benutzt wurden.

1.1.20.1. Spezifische Informationen

- Liste der JDBC Treiber:
  http://java.sun.com/products/jdbc
- Generelle Informationen über JDBC:
  selber Link
- User Group:
  selber Link

1.1.20.2. Dokumentation und Spezifikation

Auf dem Java Site von Sun finden sie die jeweils aktuellste Version der JDBC Dokumentation, inklsuive Treiberanbieter und vielen weiteren Informationen.

1.1.20.3. Bücher

Im Englischen gilt das Buch

- *JDBC API Tutorial and Reference, Second Edition* von Maydene Fisher, Dr. Rick Cattell, Graham Hamilton, Seth White and Mark Hapner (Addison Wesley ISBN 0201433281) als Standardwerk
- *Database Programming with JDBC and Java, Second Edition* von George Reese (O'Reilly & Associates ISBN 1565926161) ist etwas veraltet
- deutsche Bücher finden Sie beim Hanser Verlag und neu beim dPunkt Verlag. Das Buch im Hanser Verlag ist didaktisch sehr gut, geht aber nicht so weit wie das Buch im dPunkt Verlag.

1.1.20.4. SQL Ressourcen

1.1.20.4.1. Web Sites

- FIPS PUB 127-2: The Standard for Database Language SQL
  http://www.itl.nist.gov/fipspubs/fip127-2.htm
- Introduction to Structured Query Language
  http://w3.one.net/~jhoffman/sqltut.htm
- DB2 Home Page
  http://www-4.ibm.com/software/data/db2
- Oracle Home Page
  http://www.oracle.com

1.1.20.4.2. Bücher

- *SQL The Complete Reference* von James R. Groff und Paul N. Weinberg (McGraw-Hill ISBN 0072118458)
- *Joe Celko's SQL for Smarties : Advanced SQL Programming, Second Edition* von Joe Celko (Morgan Kaufmann ISBN 1558605762)