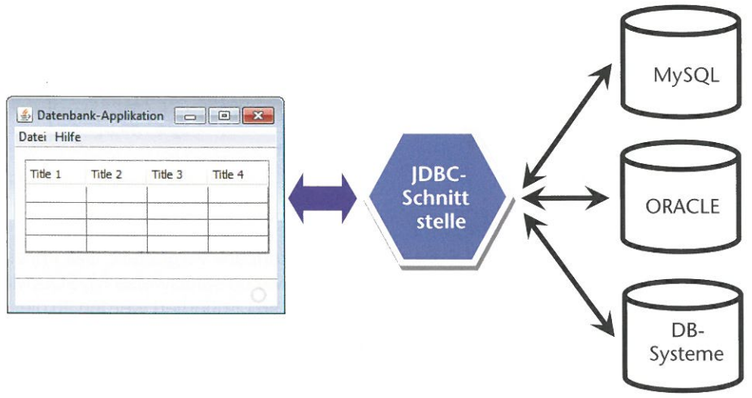
9 Datenbankzugriff mit Java

9.1 Datenbankzugriff mit Java

Das Speichern von Daten ist eine elementare Aufgabe von Anwendungsprogrammen. Natürlich kann eine Anwendung das Speichern von Daten mithilfe von Dateioperationen selbst durchführen. Für wenige Daten ist das wahrscheinlich auch die beste Wahl bei der Entwicklung einer Anwendung, weil sie damit relativ unabhängig ist. Wenn allerdings viele Daten (oder Datensätze) zu speichern sind und die Daten zusätzlich einen komplizierten Aufbau haben, dann ist die Speicherung in einer Datenbank in Betracht zu ziehen. Der große Vorteil bei einer Datenbankanbindung ist die Unabhängigkeit der Anwendung von der technischen Umsetzung der Datenspeicherung. Das erledigt die Datenbank im Hintergrund. Auch das Ändern oder Löschen von Daten ist bequem durch die entsprechenden Datenbankbefehle zu realisieren. Die bereits ausführlich behandelte Abfragesprache **SQL** spielt hierbei eine wichtige Rolle.

9.1.1 Datenbankanbindung mit JDBC

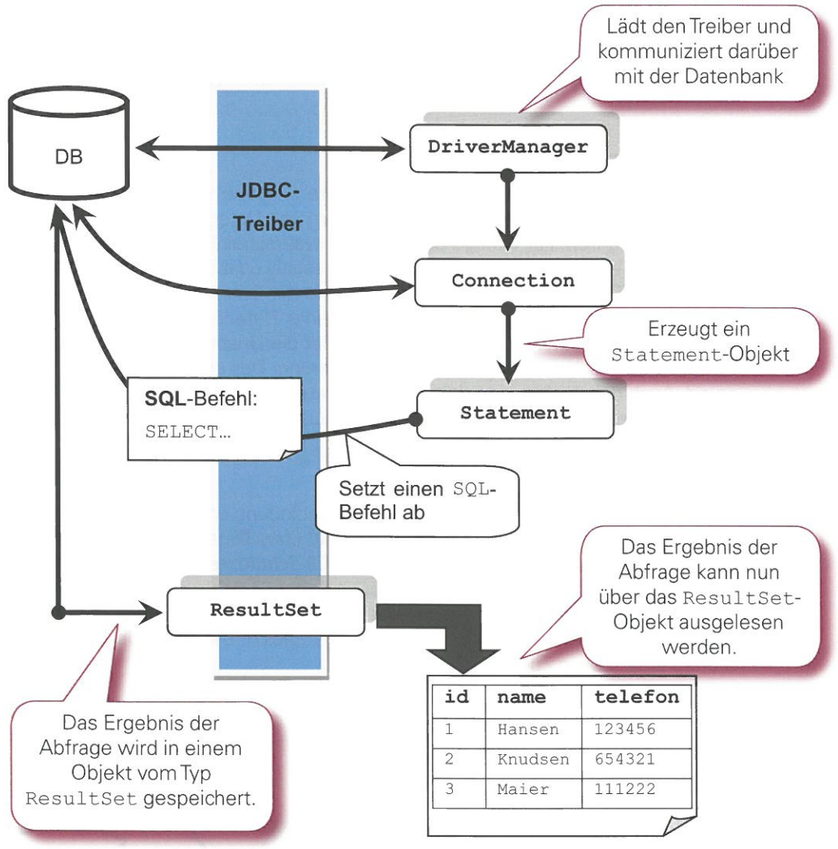
Java bietet eine Vielzahl von Klassen, um die Anbindung an eine Datenbank zu realisieren. Diese Klassen sind unter dem Oberbegriff **JDBC** (**J**ava **D**ata**b**ase **C**onnectivity) gesammelt. Für die meisten Datenbanken existieren ***JDCB-Schnittstellen***, die den Datenbankzugriff aus einer Java-Anwendung in die entsprechenden Befehle der Datenbank umwandeln. Damit ist es für den Java-Programmierer im Prinzip egal, mit welchem Datenbanksystem im Hintergrund gearbeitet wird – der Zugriff ist gleich. Die folgende Abbildung zeigt das Grundprinzip dieses Zugriffs:



9.1.2 JDBC-Treiber laden und eine Verbindung aufbauen

Um eine Verbindung zu einer Datenbank aufzubauen, muss der entsprechende Treiber vorliegen. Das Laden des Treibers in den Speicher kann dann mit einer Methode der Klasse **`DriverManager`** erfolgen. Diese Klasse ist Bestandteil des **`java.sql`**-Paketes, welches in das Java-Projekt importiert werden sollte. Nach dem erfolgreichen Laden des Treibers kann dann eine Verbindung zur Datenbank aufgebaut werden. Das geschieht mithilfe der Klasse **`Connection`**, die über den **`DriverManager`** eine Verbindung zur Datenbank erhält. Je nach Datenbank sind Nutzername und Passwort anzugeben. Über ein **`Statement`**-Objekt kann dann eine Abfrage gestartet und mit einem **`ResultSet`**-Objekt ausgelesen werden.

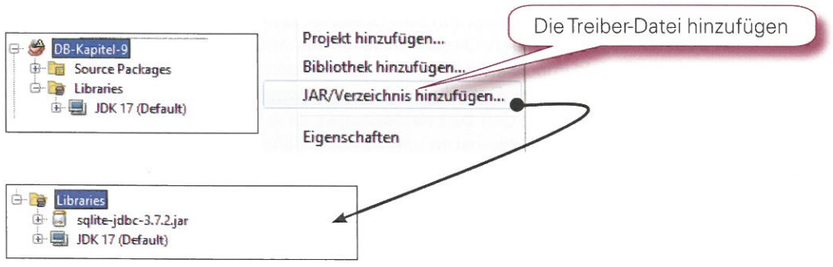
Die folgende Abbildung soll den Zusammenhang noch einmal darstellen:



9.1.3 Zugriff auf eine SQLite-Datenbank

Im Folgenden wird der Zugriff auf eine SQLite-Datenbank vorgestellt. Das Prinzip ist aber übertragbar auf andere relationale Datenbanken wie beispielsweise MySQL-/Maria- oder Oracle-Datenbanken. Dabei muss das Paket **`java.sql`** importiert werden. In diesem Paket sind alle relevanten Klassen, um auf eine Datenbank zuzugreifen.

In den folgenden Beispielen wird die Entwicklungsumgebung **NetBeans** verwendet. Das Prinzip ist aber auf andere Umgebungen (wie **Eclipse**) übertragbar. Nach dem Download des gewünschten Treibers (beispielsweise *sqlite-jdbc-XXX.jar*) wird die Datei vom Typ Java-Archive in das Projekt integriert:



Nach dem erfolgreichen Hinzufügen des Treibers kann die Verbindung mit der Klasse **`Class`** erzeugt werden:

```
String datenbank = "jdbc:sqlite:/Pfad/Datenbank";
```

Den Pfad und die Datenbankdatei angeben.

```
Class.forName("org.sqlite.JDBC");
```

Den Treiber laden.

Die Verbindung herstellen

```
Connection verbindung;
verbindung = DriverManager.getConnection(datenbank,"","");
```

Tipp:

ID	Name
Filter	Filter
1	Maier
2	Knudsen
3	Kaiser
4	Franzen
5	Knobloch
6	Laufer

Die Datenbank SQLite ist eine kostenfreie und portable Datenbank, die eine komplette Datenbanklogik und die Daten selbst in einer Datei kapselt. Damit ist die Weitergabe von Java-Programmen mit einer eigenen Datenbank möglich. Für kleine Projekte mit relativ wenig Datenvolumen ist es eine hervorragende Alternative zu den großen Datenbanken wie Oracle oder auch MySQL / MariaDB. Mit kostenfreien Tools wie dem „DB-Browser für SQLite" können die Datenbanken einfach administriert werden.

Die zugrunde liegende Datenbank *Kunden.sqlite* liegt für das folgende Beispiel in dem Ordner *"C:\temp"*. Sie verfügt über eine Beispieltabelle `Kunden` mit den Attributen `ID` (Typ `Zahl`) und `Name` (Typ `VARCHAR`):

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

public class DBZugriff {
    public static void main(String[] args) {
        try {
```

Den Verbindungsstring mit Treiberangabe und der Datenquelle festlegen.

```
            String datenbank = "jdbc:sqlite:/c:/temp/kunden.sqlite";
```

Den Treiber laden.

```
            Class.forName("org.sqlite.JDBC");
```

Ein Verbindungsobjekt mithilfe der statischen Methode `getConnection` anfordern.

```
            Connection verbindung =
                DriverManager.getConnection(datenbank,"","");
```

Über das Verbindungsobjekt wird ein Objekt für den SQL-Befehl erstellt.

```
            Statement sqlBefehl = verbindung.createStatement();
```

Die Methode `executeQuery` führt die SQL-Abfrage (bzw. den *SELECT*-Befehl) aus und gibt das Ergebnis als Objekt vom Typ `ResultSet` zurück.

```
            ResultSet ergebnis =
                sqlBefehl.executeQuery("SELECT * FROM Kunden;");
```

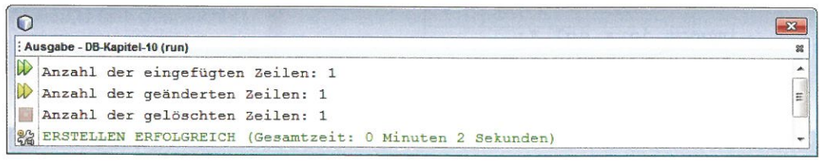
Das Ergebnisobjekt bietet Methoden an, um die Ergebnistabelle abzufragen. Die Methode `next` zeigt an, ob weitere Einträge vorhanden sind und die Methode `getString` liest den nächsten Eintrag aus der gewünschten Spalte (hier *Name*).

```
            while (ergebnis.next() == true) {
                System.out.println("Name: " + ergebnis.
                                   getString("Name"));
            }
            verbindung.close();
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

WICHTIG: Datenbankverbindungen sollten wieder geschlossen werden.

ACHTUNG: Bei der Abfrage von Datenbanken ist es besonders wichtig, dass die Ausnahmebehandlung eingesetzt wird!

Nach dem Starten erscheint dann die folgende Bildschirmausgabe:



Die Datenbank wurde einwandfrei abgefragt und alle Namen der Kunden ausgelesen und angezeigt.

Hinweis:

Der Zugriff auf die Spaltenwerte einer Tabelle mit dem Ergebnisobjekt erfolgt in Abhängigkeit vom jeweiligen Datentyp. Für jeden Datentyp steht eine geeignete Methode zu Verfügung, die entweder den Spaltenindex oder den Spaltennamen übernimmt:

- `getString( int spaltenindex )`
- `getString ( String spaltenname )`
- `getDouble( int spaltenindex )`
- `getDouble ( String spaltenname )`
- `getInt( int spaltenindex )`
- `getInt ( String spaltenname )`
- ... weitere Typen

Beispielsweise könnte die erste Spalte der Kunden-Tabelle mit der Methode `getInt` ausgelesen werden, da es sich um einen ganzzahligen numerischen Typen handelt:

```
while (ergebnis.next() == true){
 System.out.println("ID: " +
        ergebnis.getInt(0));
}
```

9.1.4 Nicht-Select-Befehle absetzen

Das Auslesen einer beliebigen Tabelle kann mithilfe der oben beschriebenen Anweisungen erfolgen. Möchte man hingegen nicht selektieren, sondern einfügen, ändern oder löschen, so kann ein so genannter **executeUpdate**-Befehl abgesetzt werden. Vorher sollte der gewünschte SQL-Befehl in einer Zeichenkette erstellt werden. In dem folgenden Beispiel wird eine neue Zeile in die Kundentabelle eingefügt, eine bestehende Zeile geändert und eine Zeile gelöscht:

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

public class DBZugriff {
    public static void main(String[] args) {
      try {
            String datenbank = "jdbc:sqlite:/c:/temp/kunden.sqlite";
            Class.forName("org.sqlite.JDBC");
            Connection verbindung =
                 DriverManager.getConnection(datenbank,"","");
            Statement sqlBefehl = verbindung.createStatement();
```

SQL-Befehl absetzen und die Anzahl der betroffenen Zeilen zurückerhalten.

Der SQL-Befehl, um eine Zeile einzufügen.

```
            int anzahl = sqlBefehl.executeUpdate("INSERT INTO
                                                  Kunden VALUES
                                                  (7,'Koenig');");

            System.out.println("Anzahl der eingefügten Zeilen: "
                               + anzahl);
```

Ein `UPDATE`-Befehl

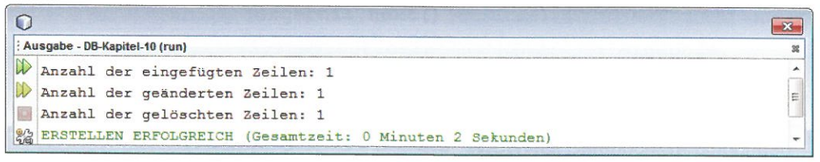
```
            anzahl = sqlBefehl.executeUpdate("UPDATE Kunden SET
                                              Name = 'Knoblauch'
                                              WHERE Name =
                                              'Knobloch';");

            System.out.println("Anzahl der geänderten Zeilen: "
                               + anzahl);
```

Ein `DELETE`-Befehl

```
            anzahl = sqlBefehl.executeUpdate("DELETE FROM Kunden
                                              WHERE Name =
                                              'Knudsen';");
```

```
            System.out.println("Anzahl der gelöschten Zeilen: "
                               + anzahl);
            verbindung.close();
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Nach dem Starten werden die drei *Nicht-Select-SQL-Befehle* abgesetzt und die Anzahl der betroffenen Zeilen ausgegeben:



Zum Vergleich: Die Kundentabelle vor und nach den SQL-Befehlen:

Vorher:

ID	Name
Filter	Filter
1	Maier
2	Knudsen
3	Kaiser
4	Franzen
5	Knobloch
6	Laufer

Nachher:

ID	Name
Filter	Filter
1	Maier
2	Kaiser
3	Franzen
4	Knobloch
5	Laufer
6	Koenig

9.1.5 Metadaten ermitteln

Für einen flexiblen Datenbankzugriff ist es oftmals wichtig, Informationen zur Datenbank, zum Treiber und auch zu den Tabellen der Datenbank zu erhalten. Diese Informationen können mithilfe von Datenbank-Metadatenklassen ermittelt werden. Aufgrund dieser Informationen kann das Programm dann weitere Entscheidungen treffen. Beispielsweise könnte es sein, dass eine Tabelle der Datenbank ständig neue Spalten erhält. Damit muss der Zugriff auf diese Tabelle flexibel gestaltet werden, sonst führt er zu einem Fehler oder zu einem Abbruch. Für das Auslesen der Metadaten sind zwei Klassen wichtig: **`DatabaseMetaData`** und **`ResultSetMetaData`**

Das folgende Beispiel zeigt die Verwendung der beiden Klassen, um die SQLite-Datenbank aus dem obigen Beispiel auszulesen:

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

public class DBZugriff {
    public static void main(String[] args) {
      try {
          String datenbank = "jdbc:sqlite:/c:/temp/kunden.sqlite";
```



```
          Class.forName("org.sqlite.JDBC");
          Connection verbindung =
                  DriverManager.getConnection(datenbank,"","");
```

Über das `Connection`-Objekt werden die Metadaten der Datenbank abgefragt.

```
          //Datenbank Metadaten
          DatabaseMetaData dbinfos = verbindung.getMetaData();

          System.out.println("Metadaten der Datenbank:");
          System.out.println("Name der Datenbank: "
                             + dbinfos.getDatabaseProductName());
          System.out.println("Name des Treibers : "
                             + dbinfos.getDriverName());
          System.out.println();

          //Tabellen Metadaten
          Statement sqlBefehl = verbindung.createStatement();
          ResultSet ergebnis =
                  sqlBefehl.executeQuery("SELECT * FROM
                                          Kunden;");
```

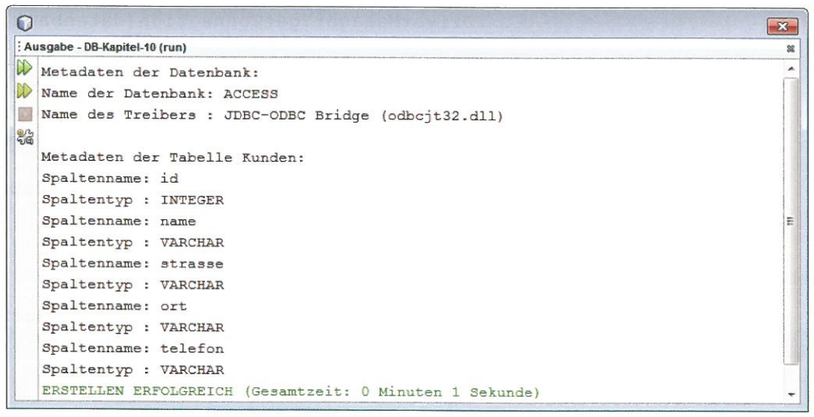
Über das `ResultSet`-Objekt werden die Metadaten der Tabelle abgefragt.

```
          ResultSetMetaData tbinfos = ergebnis.getMetaData();

          System.out.println("Metadaten der Tabelle Kunden:");
          for ( int i = 1; i <= tbinfos.getColumnCount(); i++ )
                                                          {
```

Anzahl der Spalten erfragen.
ACHTUNG: Index startet mit 1!

```
              System.out.println("Spaltenname: " +
                                 tbinfos.getColumnLabel(i));
              System.out.println("Spaltentyp : " +
                                 tbinfos.getColumnTypeName(i));
          }
          verbindung.close();
      }
      catch (Exception e) {
          System.out.println(e.getMessage());
      }
  }
```

Nach dem Starten unter ***NetBeans*** erscheint dann die folgende Bildschirmausgabe:

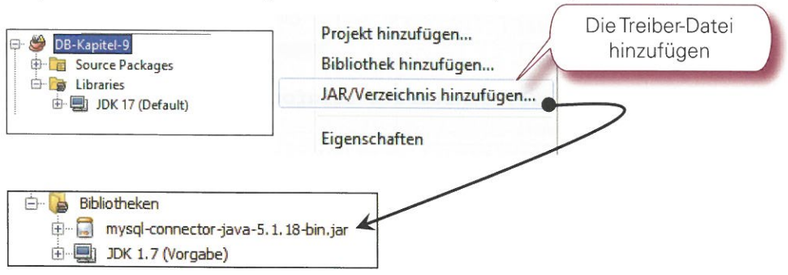


9.2 Weitere Datenbanken ansprechen

Für die meisten Datenbanken existieren Treiber, so dass mit Java und JDBC darauf zugegriffen werden kann. In der Regel muss der Treiber nur heruntergeladen und dem Projekt hinzugefügt werden. Danach kann der Treiber wie gewohnt registriert werden.

9.2.1 Einen Treiber hinzufügen

Nach dem Download des gewünschten Treibers (beispielsweise *mysql-connector-java-XXX-bin.jar*) wird die Datei vom Typ Java-Archive in das Projekt integriert:



Nach dem erfolgreichen Hinzufügen des MySQL-Treibers kann die Verbindung mit der Klasse `Class` erzeugt werden:

```
String datenbank = "jdbc:mysql://Servername/Datenbank";
```

Den Namen des Servers der MySQL-/Maria-DB-Datenbank angeben. Bei lokaler Installation einfach `localhost` eintragen.

Den Datenbanknamen angeben.

```
Class.forName("com.mysql.jdbc.Driver");
```

Den Treiber laden.

Eine Verbindung zur Datenbank herstellen (mit Angabe von Benutzer und Passwort).

```
Connection verbindung;
verbindung = DriverManager.getConnection(datenbank,"Benutzer","Pwd");
```


9.2.2 Weitere Datenbanktreiber

Die folgende Tabelle zeigt eine Übersicht gängiger Datenbanken und den zugehörigen Java-Treibernamen. Die entsprechende Treiberdatei muss wie oben beschrieben von der Web-Seite des Anbieters heruntergeladen werden oder auf anderem Wege vorhanden sein.

Datenbank	Java-Treibername
Firebird (freie DB, Nachfolger von Borland Interbase)	`org.firebirdsql.jdbc`
DB2 (IBM)	`db2jcc4.jar`
Informix (IBM)	`com.informix.jdbc.IfxDriver`
Microsoft **SQL-Server**	`com.microsoft.sqlserver.jdbc.SQLServerDriver`
MySQL	`com.mysql.jdbc.Driver`
MariaDB	`mariadb-java-client-2.6.0-sources.jar`
Oracle	`oracle.jdbc.OracleDriver`
SQLite	`org.sqlite.JDBC`

Hinweis:

Vor dem Einbinden einer Datenbank bleibt in der Regel keine Alternative zu einer umfassenden Internetrecherche oder der Sichtung entsprechender Fachliteratur zu der Datenbank.

9.3 Aufgaben zu Kapitel 9

Aufgabe 1

In einer Firma werden die Provisionen der Vertriebsmitarbeiter in einer einfachen Datenbank (in diesem Beispiel einer *SQLite*-Datenbank) gespeichert. Erstellen Sie eine solche Datenbank mit einer Tabelle und dem entsprechenden Inhalt. Anschließend sollen folgende statistische Kenndaten aus der Tabelle ausgelesen werden:

- Den Vertriebsmitarbeiter mit der höchsten Provision
- Den Vertriebsmitarbeiter mit der geringsten Provision
- Die Summe aller Provisionen
- Den Durchschnitt aller Provisionen

Die Tabelle in der *SQLite*-Datenbank sieht so aus:

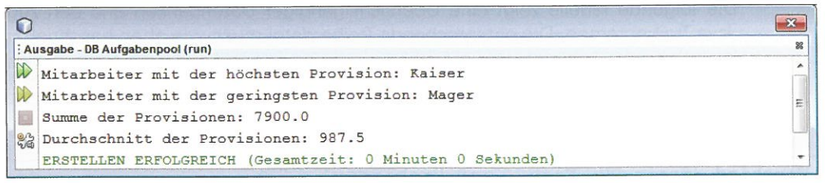
Name	Provision
Filter	Filter
Maier	1200.0
Knudsen	800.0
Laufer	600.0
Kaufhold	1400.0
Mager	350.0
Kaiser	1900.0
Lehmberg	950.0
Katernberg	700.0

Die Verwaltung einer SQLite-Datenbank kann komfortabel über ein kostenfreies Tools wie „DB-Browser für SQLite“ erfolgen.

Hinweis:

Die Berechnung der Kenndaten kann entweder mit den entsprechenden SQL-Funktionen (wie `SUM`, `AVG`, `MIN` und `MAX`) geschehen oder mithilfe von Java-Programmlogik umgesetzt werden.

Nach dem Starten könnte die Bildschirmausgabe so aussehen:



Aufgabe 2

Ausgangssituation:

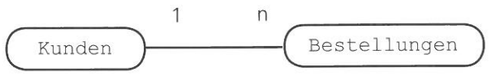
In einer Firma sind die Bestelldaten der Kunden in zwei Datenbanktabellen (beispielsweise mit *SQLite*) abgelegt. Für die Mitarbeiter soll eine einfache Java-GUI-Anwendung geschrieben werden, mit der die Bestelldaten eines Kunden übersichtlich dargestellt werden können.

Die zugrunde liegenden Tabellen sehen so aus:

Kundentabelle:

ID	Name
Filter	Filter
1	Maier
2	Knudsen
3	Kaiser
4	Franzen
5	Knobloch
6	Laufer

Beziehung der Tabellen:



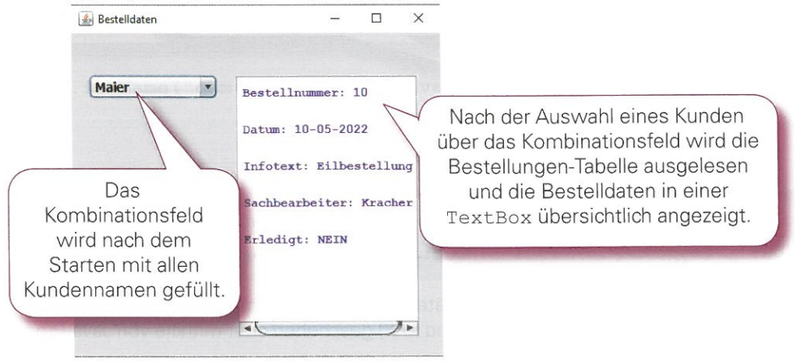
Bestellungen-Tabelle:

Kunden_ID	Bestellnummer	Datum	Infotext	Sachbearbeiter	Erledigt
Filter	Filter	Filter	Filter	Filter	Filter
1	10	10-05-2022	Eilbestellung	Kracher	true
3	11	10-06-2022	gefährliche Fr...	klauber	true
4	12	10-07-2022	guter Kunde	Hütter	false

Die Bestellungen-Tabelle hat einen Fremdschüssel `Kunden_ID`, der die "1:n" – Beziehung der beiden Tabellen umsetzt.

Aufgabenstellung:

Legen Sie die beiden Tabellen in einer geeigneten Datenbank an (beispielsweise *SQLite*) und füllen Sie die Tabellen mit den entsprechenden Werten. Implementieren Sie dann eine Java-Anwendung, die auf die Datenbank zugreift und die Tabellen ausliest. Die Oberfläche der Anwendung sollte so aussehen:



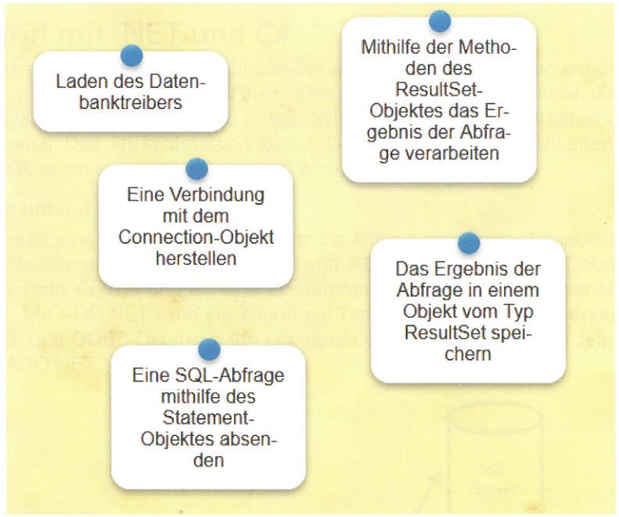
9.4 Digitale Inhalte zu Kapitel 9:

Hinweis:

Um die Aufgaben online zu bearbeiten, bitte den QR-Code scannen oder den Link eingeben.

Aufgabe 1: Den Ablauf einer Datenbank-Verbindung rekonstruieren

https://vel.plus/hvoX



Aufgabe 2: JDBC-Begriffe finden

https://vel.plus/BV5T

T	Y	M	I	U	E	M	Y	A	Ö	H	D	T	H	T	R	W	L	C	T	M	M	M	Ö	Q	I
H	S	R	T	C	U	E	N	C	Y	K	H	J	S	Y	A	U	Ü	O	U	L	Ü	A	T	A	S
E	K	W	N	I	D	C	F	O	W	R	S	T	A	T	E	M	E	N	T	S	Ö	K	O	Ä	M
B	N	F	V	P	I	Y	N	Q	V	D	R	I	V	E	R	M	A	N	A	G	E	R	P	Z	I
N	Ö	B	Q	W	R	E	X	E	E	A	Ü	Y	P	L	D	I	R	E	S	U	L	T	S	E	T
T	D	N	G	J	Q	G	Ü	C	S	F	O	T	L	L	X	V	F	C	Ä	O	Ü	Q	Ü	J	Ü
J	A	V	A	D	A	T	A	B	A	S	E	C	O	N	N	E	C	T	I	V	I	T	Y	C	Ö
F	Ü	U	W	K	V	L	C	D	N	T	R	Ä	K	V	W	I	E	I	D	U	I	K	D	J	F
T	V	Y	P	D	I	H	H	V	F	R	L	T	Ö	Ü	M	U	P	O	Ö	Z	S	X	N	U	U
H	T	Ö	D	U	P	T	G	S	E	Z	G	R	Z	B	Q	M	Q	N	J	Ü	Z	K	H	U	G