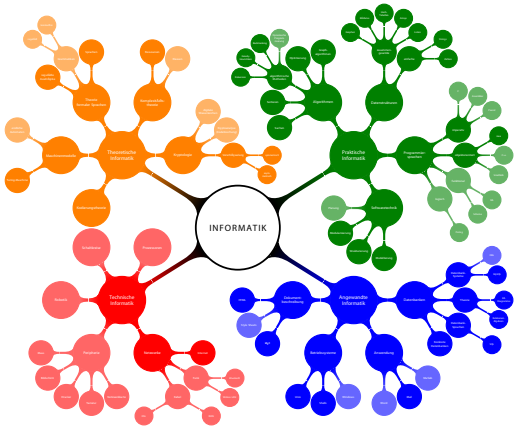
UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR
THEORETISCHE INFORMATIK

Kapitel 40

SQL – Anbindung an Java

Eine Kaffeeklatsch zum Thema Datenbanken

Vorlesung Einführung in die Informatik 1 vom 17. Juni 2014 von Till Tantau



Lernziele von Kapitel 40

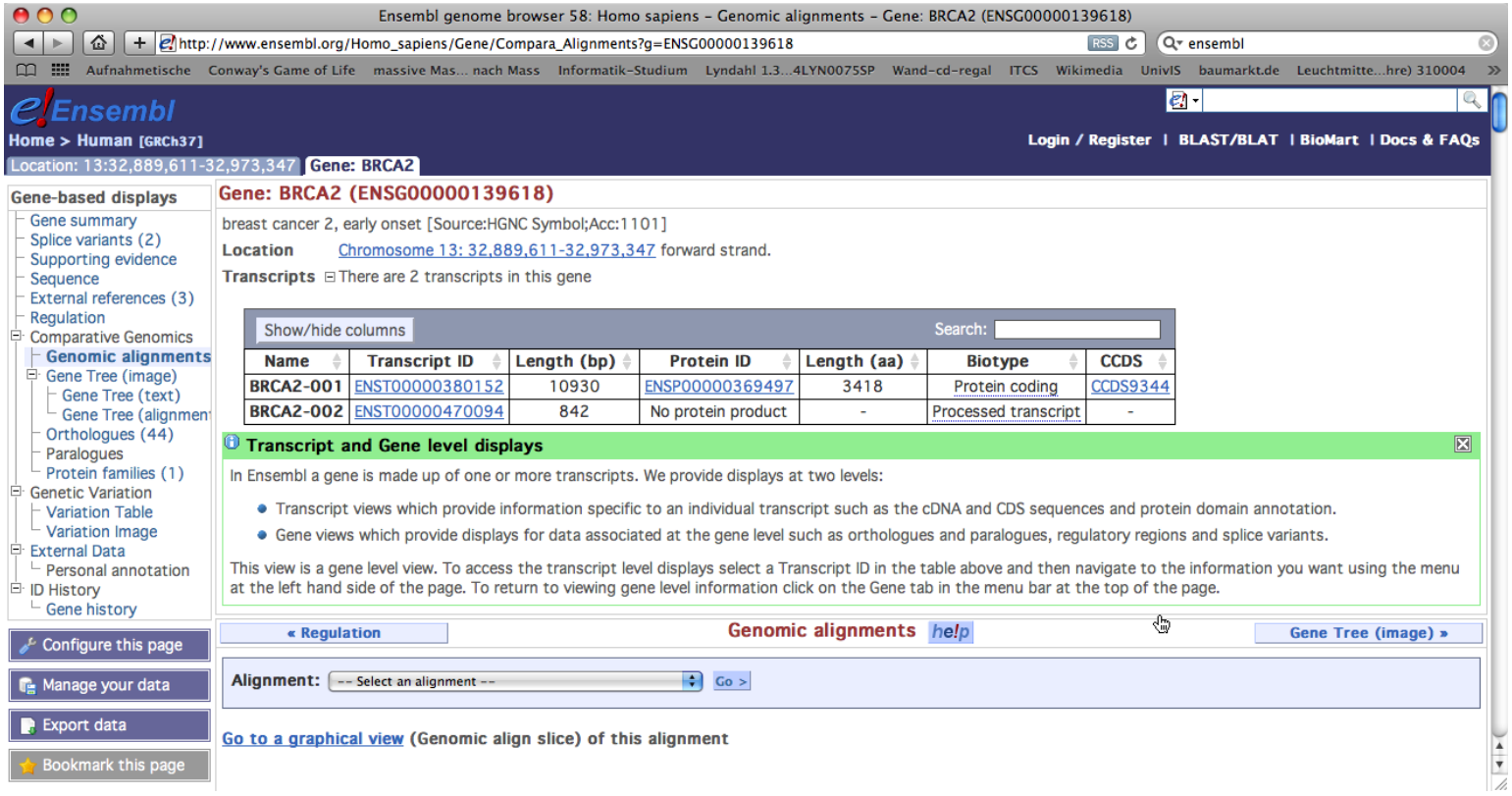
1. JDBC kennen und seine Aufgabe verstehen
2. Verbindungen mit Datenbanken aus Java-Programmen heraus aufbauen können
3. Daten mittels JDBC lesen können
4. Daten mittels JDBC schreiben können

Gliederung von Kapitel 40

Menschen wollen nicht mit Datenbanken reden

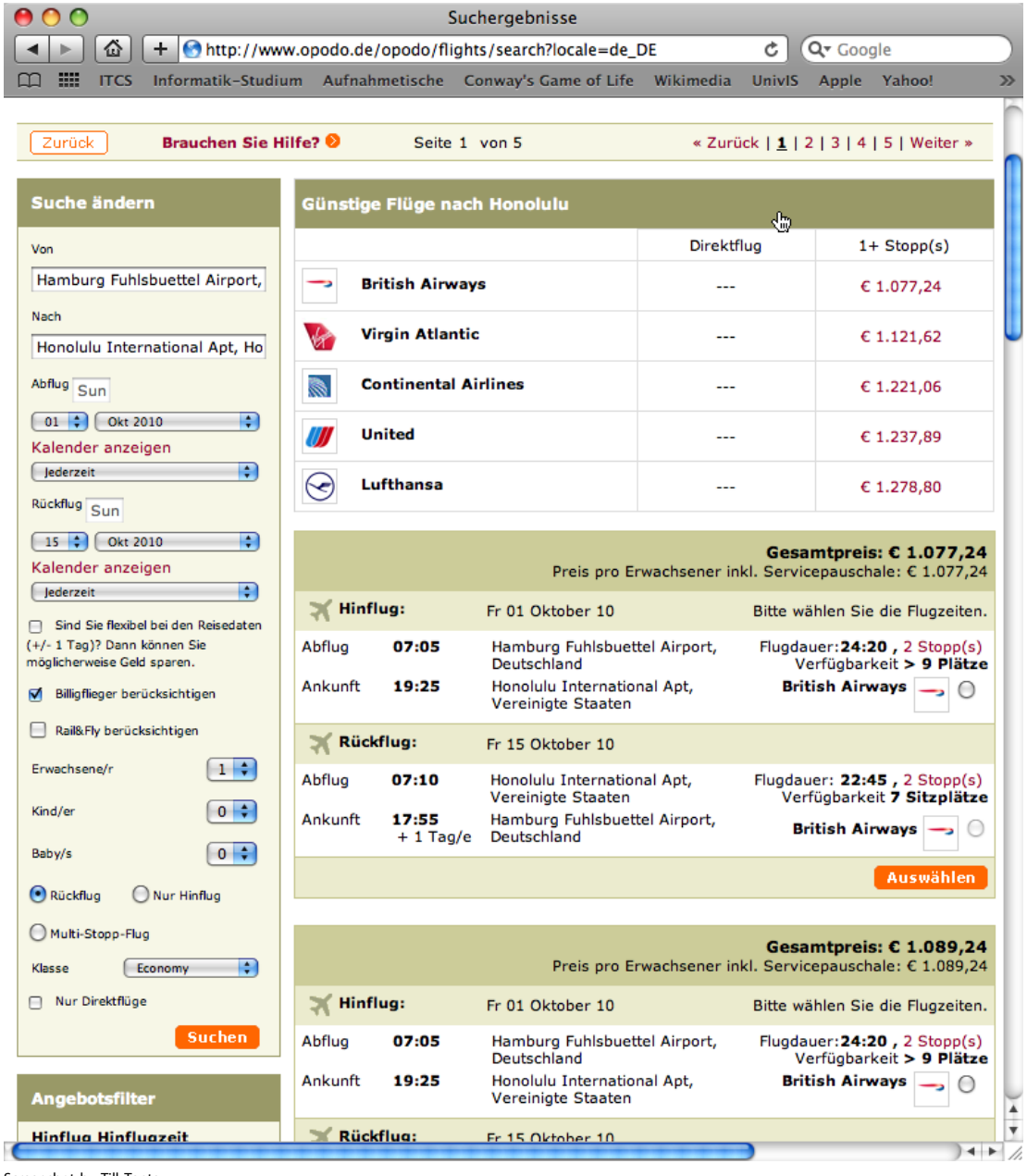
- Als Mensch möchte man auf eine Datenbank *nicht über SQL-Befehle* zugreifen.
- Vielmehr möchte man *eine graphische Oberfläche* oder eine *Webseite* nutzen.
- Dann muss aber *an Stelle des Menschen ein Programm* mit der Datenbank kommunizieren.

Menschen wollen nicht mit Datenbanken reden



Screenshot by Till Tantau

Menschen wollen nicht mit Datenbanken reden



Screenshot by Till Tantau

Was die Java DataBase Connection (JDBC) bietet

Nehmen wir an, wir haben bereits:

1. Einen laufenden Datenbank-Server und
2. eine Datenbank auf dem Server.

In einem Java-Programm kann man nun Klassen aus dem Paket `java.sql` benutzen:

- Objekte zu diesen Klassen repräsentieren dann Dinge wie »Verbindungen« oder »SQL-Statements«.
- Da es sich um ganz normale Java-Objekte handelt, kann man den Objekten Nachrichten schicken wie »Verbinde dich mit dem Server« oder »Schaue in der Schafs-Tabelle nach«.
- Was der Benutzer davon sieht, kann sich das Programm dann aussuchen – *oft merkt der Benutzer gar nicht, dass intern eine Datenbank angefragt wurde*.

Die Idee hinter Verbindungsobjekten.

- Damit ein Java-Programm aus einer Datenbank etwas lesen kann, muss es zunächst eine *Verbindung* mit dem Datenbank-Server aufnehmen.
- Eine solche Verbindung wir durch ein Objekt der Klasse `Connection` repräsentiert.
  (Die Klasse `Connection` findet sich, wie alle im Folgenden vorgestellten Klassen, im Paket `java.sql`.)
- Ein Programm kann mit mehreren Datenbanken gleichzeitig kommunizieren – dann gibt es pro Verbindung ein `Connection`-Objekt.

Warum JDBC Treiber braucht.

Man könnte erwarten, dass man Verbindungen wie folgt erzeugt:

```
Connection con = new Connection ("Server-Name",
    "User-Name", "Passwort");
```

Es gibt aber Gründe, weshalb das so nicht gemacht wird:

- Es gibt viele unterschiedliche SQL-Datenbank-Systeme (zum Beispiel MySQL, Oracle, IBM und andere).
- Für Java-Programme soll es aber *egal sein*, welches System dies genau ist.
- Aus diesem Grund gibt es wie bei Betriebssystemen einen *Treiber* für jede Art von Datenbank-System.
- Deshalb erzeugt man Verbindungen zu Datenbank-Systemen nicht einfach mittels `new`, sondern bittet die Treiber-Verwaltung, eine solche Verbindung herzustellen.

Beispiel-Code zur Erzeugung eines Verbindungsobjekts.


```
import java.sql.*;
...
Class.forName("com.mysql.jdbc.Driver");
Connection connection = DriverManager.getConnection
  ("jdbc:mysql://localhost/molecular_sheep", "root",
      "*******");
```

Was hier passiert:

- Mittels `Class.forName(...)` wird der Treiber geladen.
- Die Variable `connection` hat den Typ `Connection`. Sie wird einen Verweis auf das Verbindungsobjekt speichern.
- Die Klasse `DriverManager` hat eine (statische) Methode namens `getConnection`, die folgende Parameter hat:
  1. Ein String, der die Datenbank lokalisiert.
  2. Name eines in der Datenbank registrierten Benutzers.
  3. Das Passwort dieses Benutzers.

  Die Methode liefert ein Verbindungsobjekt zurück.

Aufbau des Lokalisierungs-Strings.

- Beim Aufbau einer Verbindung gibt man mittels eines Strings an, mit welcher Datenbank auf welchem Server man sich verbinden möchte.
- Dieser String beginnt mit `jdbc:`.
- Danach kommt der Name des Treibers für den Datenbank-Server, beispielsweise `mysql:`.
- Was danach kommt, hängt vom Treiber ab. Bei MySQL kommen:
  1. Zwei Schrägstriche, dann die Internet-Adresse des Servers.
  2. Ein Schrägstrich, dann der Name der Datenbank auf diesem Server.

Beispiele

- `jdbc:mysql://localhost/molecular_sheep`
- `jdbc:mysql://ensembldb.ensembl.org/homo_sapiens_variation_46_36h`
- `jdbc:mysql://ensembldb.ensembl.org:5306/homo_sapiens_variation_49_36k`

Direktes Ausführen von SQL-Statements.

Ist erstmal eine Verbindung mit einer Datenbank aufgebaut, so kann man *Statements* benutzen, um *mit der Datenbank zu reden*:

1. Zunächst erzeugt man wie folgt ein Objekt der Klasse `Statement`:

   ```
   Statement statement = connection.createStatement();
   ```

2. Dann bittet man das so erzeugte Objekt, etwas für uns zu tun: Erstmal eine Tabelle zu erzeugen. . .

   ```
   statement.execute( "create␣table␣schaf␣("+
                      "id␣integer␣primary␣key," +
                      "name␣char␣(20)," +
                      "farbe␣char␣(20)␣)");
   ```

3. . . . und dann etwas hineinschreiben:

   ```
   statement.execute( "insert␣into␣schaf␣" +
       "␣values␣(1,␣\"dolly\",␣\"blau\")");
   ```

Daten liest man mittels Antwort-Mengen.

- Zum *Lesen* von Daten aus der Datenbank benutzt man *auch Statements*.
- Jedoch gibt es nun als *Rückgabewerte* die gefundenen Zeilen in der Tabelle.
- *Man könnte erwarten,* dass man einen Array zurückbekommen sollte, der die zum Statement passenden Zeilen enthält.
- In Wirklichkeit bekommt man jedoch nur einen »Cursor-Objekt«, das immer auf eine der gefundenen Zeilen zeigt.
- Es gibt dann Methoden, die die Werte der Attribute der Zeile auslesen, auf der der Cursor ist, und Methoden, die den Cursor bewegen.

Beispiel: Ausgabe aller schwarzen Schafe.


```
Statement statement = connection.createStatement();

statement.executeQuery
  ("select␣*␣from␣schaf␣where␣farbe␣=␣\"schwarz\"");

ResultSet rs = statement.getResultSet ();

while (rs.next()) {
  System.out.println( rs.getString("name"));
}
```

Was hier passiert:

- Mittels `statement.executeQuery` werden die schwarzen Schafe ausgesucht.
- Mittels `statement.getResultSet` wird dann ein Objekt erzeugt, das ein Cursor ist. Es zeigt anfänglich auf die Position »vor der ersten ausgesuchten Zeile«.
- Mittels `rs.next` wird der Cursor »weitergeschoben«. Gibt diese Methode `false` zurück, so gibt es keinen weiteren Zeilen mehr.
- Mittels `rs.getString` kann dann der Wert eines Attributs in der aktuellen Zeile ausgesucht werden.

Einfügen eines neuen Schafs.


```
public static void newSheep () throws Exception {

  System.out.println ("Bitte␣geben␣Sie␣nacheinander␣den␣
      Namen,␣die␣Farbe␣und␣die␣Nummer␣des␣Schafs␣ein:");
  String name  = SimpleIO.readLine();
  String farbe = SimpleIO.readLine();
  int id       = Integer.parseInt(SimpleIO.readLine());

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

  Connection con = DriverManager.getConnection
    ("jdbc:mysql://localhost/molecular_sheep", "root",
        "");

  Statement stmt = con.createStatement();
  stmt.execute
    ("insert␣into␣schaf␣values␣(" + id + ",␣\"" + name +
        "\",␣\"" + farbe + "\")");
}
```

Ausgabe aller Schafe einer Farbe.


```
public static void printColoredSheep () throws Exception {

  System.out.println ("Bitte␣geben␣Sie␣eine␣Farbe␣ein:");
  String farbe = SimpleIO.readLine();

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

  Connection con = DriverManager.getConnection
    ("jdbc:mysql://localhost/molecular_sheep", "root",
        "");

  Statement stmt = con.createStatement();
  stmt.executeQuery
    ("select␣*␣from␣schaf␣where␣farbe␣=␣\"" + farbe +
        "\"");

  System.out.println ("Folgende␣Schafe␣sind␣" + farbe +
      ":");
  ResultSet rs = stmt.getResultSet ();
  while (rs.next()) {
    System.out.println(rs.getString("name"));
  }
}
```

Code aus einem echten Bioinformatik-Projekt.

- In einem unserer Forschungsprojekte geht es um *Haplotypisierung*.
- Dabei versucht man mittels algorithmischer Methoden, aus *Genotyp-Daten* die *Haplotypen* von Individuen zu rekonstruieren.
- Die Genotyp-Daten stehen in der Ensembl-Datenbank zur Verfügung.
- Unsere Programme holen sich mittels JDBC die Daten aus Ensembl.

Schritt 1: Erstellen der Verbindung


```
public class InputEnsembl {

  Connection conn;

  String DATABASE_REV = "49_36k";
  String DATABASE =
    "jdbc:mysql://ensembldb.ensembl.org:5306/" +
    "homo_sapiens_variation_" + DATABASE_REV;

  String USER = "anonymous";
  String PASSWORD = "";

  void connectDB() throws ClassNotFoundException,
      SQLException, UnknownHostException {
    Class.forName("com.mysql.jdbc.Driver");
    conn = DriverManager.getConnection(DATABASE, USER,
        PASSWORD);
  }

  ...
}
```

Schritt 2: Genotypen aufgrund eines Gen-Names auswählen


```
public GenotypeMatrix getGenotypesByGeneName
    (String geneName, int population) {
  connectDB();

  String sqlQuery = "SELECT␣c.sample_id,␣c.seq_region_start,␣
      c.genotypes,␣c.seq_region_id,␣g.seq_region_start,␣
      g.seq_region_end␣"
  + "FROM␣homo_sapiens_core_" + DATABASE_REV + ".gene␣AS␣g,␣
      homo_sapiens_core_" + DATABASE_REV + ".xref␣AS␣x,␣"
  + "compressed_genotype_single_bp␣AS␣c,␣individual_population␣AS␣
      ip␣"
  + "WHERE␣x.display_label␣=␣\"" + geneName + "\"" // desired
      gene by name
  + "AND␣g.display_xref_id␣=␣x.xref_id␣" // desired gene by
      xref_id
  + "AND␣c.seq_region_id␣=␣g.seq_region_id␣"
  + "AND␣(c.seq_region_start␣>=␣g.seq_region_start␣OR␣
      c.seq_region_end␣>=␣g.seq_region_start)␣"
  + "AND␣(c.seq_region_end␣<=␣g.seq_region_end␣OR␣
      c.seq_region_start␣<=␣g.seq_region_end)␣"
  + "AND␣ip.population_sample_id␣=␣\"" + population + "\"␣AND␣
      ip.individual_sample_id␣=␣c.sample_id␣order␣by␣sample_id";

  Statement stmt = conn.createStatement();
  ResultSet rSet = stmt.executeQuery(sqlQuery);
  ...
}
```

Installation eines Treibers.

- Wie schon erwähnt, braucht Java einen *Treiber*, um auf eine Datenbank zugreifen zu können.
- Diesen Treiber erhält man vom Hersteller der Datenbank.
- Typischerweise ist er in einer so genannten jar-Datei (dies steht für Java ARchive) verpackt.
- Um ihn zu benutzen, muss man zwei Dinge tun:
  1. Man muss BlueJ, Eclipse oder dem Programm `java` mitteilen, dass es die jar-Datei »beachten« soll. Dies geschieht bei BlueJ und Eclipse über den Menüpunkt »Einstellungen«, bei `java` mittels der Option `-cp`.
  2. Man muss den Treiber im laufenden Programm laden mittels

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

Exceptions: Wenn etwas schief geht.

- Bei der Kommunikation mit einer Datenbank kann *»beliebig viel schiefgehen«*.
- Beispielsweise kann die Verbindung abbrechen oder die Datenbank ist nicht vorhanden oder. . .
- Um mit solchen Situationen umzugehen, benutzt Java *Exceptions* also »Ausnahmezustände«.

Die Idee hinter Exceptions

- Nehmen wir an, eine Methode merkt, dass etwas Unvorhergesehenes geschehen ist.
- Sie erzeugt dann ein Objekt vom Typ `Exception` und *wirft* es mittels **`throw`**.
- Nun wird nach einem so genannten *Try-Catch-Block* gesucht, der diese Exception *fangen* kann.
- In dem **`catch`**-Code wird dann (hoffentlich) alles wieder in Ordnung gebracht.
- Die Berechnung geht *hinter dem Try-Catch-Block* weiter.

Das Abfangen einer Exception.


```
try {
  connectDB();

  String sqlQuery = "SELECT c.sample_id," ...;

  Statement stmt = conn.createStatement();
  ResultSet rSet = stmt.executeQuery(sqlQuery);
  ...
}
catch (SQLException e) {
  System.out.println("Error while fetching genotype
      data.");
  return;
}
catch (ClassNotFoundException e) {
  System.out.println("Error while loading mySQL-JDBC
      driver.");
  return;
}
catch (UnknownHostException e) {
  System.out.println("Could not connect to database.");
  return;
}
```

Zusammenfassung

1. JDBC erlaubt es Java-Programmen, mit Datenbank-Servern zu kommunizieren.
2. Ein `Connection`-Objekt repräsentiert eine Verbindung zu einer Datenbank auf einem bestimmten Server.
3. Ein `Statement`-Objekt repräsentiert eine Anfrage (Query), die einfach als String angegeben wird.
4. Ein `ResultSet`-Objekt repräsentiert einen Cursor, der immer auf eine Zeile einer Ergebnismenge zeigt.