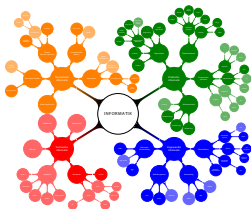


# Kapitel 40

## 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

- ▶ 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.

Ensembl genome browser 58: Homo sapiens - Genomic alignments - Gene: BRCA2 (ENSG00000139618)

http://www.ensembl.org/Homo\_sapiens/Gene/Comparative\_Alignments?g=ENSG00000139618

Ensembl  
Home > Human (GRCh37)  
Location: 13:32,889,611-32,973,347 Gene: BRCA2

Gene-based displays  
- Gene summary  
- Splice variants (2)  
- Supporting evidence  
- Sequence  
- External references (3)  
- Regulation  
- Comparative Genomics  
- **Genomic alignments**  
- Gene Tree (image)  
- Gene Tree (text)  
- Gene Tree (alignment)  
- Orthologues (44)  
- Paralogues  
- Protein families (1)  
- Genetic Variation  
- Variation Table  
- Variation Image  
- External Data  
- Personal annotation  
- ID History  
- Gene history

Genomic alignments  
breast cancer 2, early onset [Source:HGNC Symbol;Acc:1101]  
Location [Chromosome 13: 32,889,611-32,973,347](#) forward strand.  
Transcripts There are 2 transcripts in this gene

Name	Transcript ID	Length (bp)	Protein ID	Length (aa)	Biotype	CCDS
BRCA2-001	ENST00000380152	10930	ENSP00000369497	3418	Protein coding	CCDS9344
BRCA2-002	ENST00000470094	842	No protein product	-	Processed transcript	-

**Transcript and Gene level displays**

In Ensembl a gene is made up of one or more transcripts. We provide displays at two levels:

- Transcript views which provide information specific to an individual transcript such as the cDNA and CDS sequences and protein domain annotation.
- Gene views which provide displays for data associated at the gene level such as orthologues and paralogues, regulatory regions and splice variants.

This view is a gene level view. To access the transcript level displays select a Transcript ID in the table above and then navigate to the information you want using the menu at the left hand side of the page. To return to viewing gene level information click on the Gene tab in the menu bar at the top of the page.

Regulation Genomic alignments [help](#) Gene Tree (image)

Alignment: -- Select an alignment -- Go

[Go to a graphical view](#) (Genomic align slice) of this alignment

# Menschen wollen nicht mit Datenbanken reden

Suchergebnisse

http://www.opodo.de/opodo/flights/search?locale=de\_DE

Zurück Brauchen Sie Hilfe? Seite 1 von 5 Zurück 1 | 2 | 3 | 4 | 5 | Weiter »

### Suche ändern

Von: Hamburg Fuhlsbuettel Airport,  
Nach: Honolulu International Apt, Ho  
Abflug: Sun 01. Okt 2010  
Kalender anzeigen  
Rückflug: Sun 15. Okt 2010  
Kalender anzeigen  
☐ Sind Sie flexibel bei den Reisedaten (+/- 1 Tag)? Dann können Sie möglicherweise Geld sparen.  
☒ Billigflieger berücksichtigen  
☐ Rail&Fly berücksichtigen  
Erwachsene/r: 1  
Kind/er: 0  
Baby/s: 0  
☒ Rückflug ☐ Nur Hinflug  
☐ Multi-Stopp-Flug  
Klasse: Economy  
☐ Nur Direktflüge  
**Suchen**

### Günstige Flüge nach Honolulu

	Direktflug	1+ Stopp(s)
<b>British Airways</b>	---	€ 1.077,24
<b>Virgin Atlantic</b>	---	€ 1.121,62
<b>Continental Airlines</b>	---	€ 1.221,06
<b>United</b>	---	€ 1.237,89
<b>Lufthansa</b>	---	€ 1.278,80

**Gesamtpreis: € 1.077,24**  
Preis pro Erwachsener inkl. Servicepauschale: € 1.077,24

**Hinflug:** Fr 01 Oktober 10 Bitte wählen Sie die Flugzeiten.

Abflug: <b>07:05</b>	Hamburg Fuhlsbuettel Airport, Deutschland	Flugdauer: <b>24:20</b> , 2 Stopp(s)
Ankunft: <b>19:25</b>	Honolulu International Apt, Vereinigte Staaten	Verfügbarkeit > 9 Plätze <b>British Airways</b>

**Rückflug:** Fr 15 Oktober 10

Abflug: <b>07:10</b>	Honolulu International Apt, Vereinigte Staaten	Flugdauer: <b>22:45</b> , 2 Stopp(s)
Ankunft: <b>17:55</b> + 1 Tag/e	Hamburg Fuhlsbuettel Airport, Deutschland	Verfügbarkeit: 7 Sitzplätze <b>British Airways</b>

**Auswählen**

**Gesamtpreis: € 1.089,24**  
Preis pro Erwachsener inkl. Servicepauschale: € 1.089,24

**Hinflug:** Fr 01 Oktober 10 Bitte wählen Sie die Flugzeiten.

Abflug: <b>07:05</b>	Hamburg Fuhlsbuettel Airport, Deutschland	Flugdauer: <b>24:20</b> , 2 Stopp(s)
Ankunft: <b>19:25</b>	Honolulu International Apt, Vereinigte Staaten	Verfügbarkeit > 9 Plätze <b>British Airways</b>

**Rückflug:** Fr 15 Oktober 10

### Angebotsfilter

Hinflug Hinflugzeit

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.*

- ▶ Damit ein Java-Programm aus einer Datenbank etwas lesen kann, muss es zunächst eine *Verbindung* mit dem Datenbank-Server aufnehmen.
- ▶ Eine solche Verbindung wird 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.



*Man könnte erwarten*, dass man Verbindungen wie folgt erzeugt:

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

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.

```
import java.sql.*;

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

40-8

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.

- ▶ 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://ensembl.db.ensembl.org/  
homo_sapiens_variation_46_36h`
- ▶ `jdbc:mysql://ensembl.db.ensembl.org:5306/  
homo_sapiens_variation_49_36k`

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

40-10

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\"")");
```

- ▶ 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"));
}
```

40-12

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.

```
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 + "\"");
}
```

# Angabe 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"));  
    }  
}
```



- ▶ 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://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);
    ...
}
```

- ▶ 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");
```

- ▶ 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«.

40-19

### 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.

```
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;
}
```

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.