

Anleitung zur Programmierung von WatchAndChill

Alexander Thelen
alexander.thelen@uni-duesseldorf.de

8. März 2017

0 Vorbereitung

WatchAndChill ist ein Programm zur Verwaltung einer Film-Datenbank. Die Programmierung basiert auf dem Framework „ApplicationKit“.

Das gesamte Projekt steht schon als ZIP-Archiv zur Verfügung. Es muss nur noch entpackt und in Eclipse importiert werden. Gegebenenfalls müssen Libraries (JRE und JDBC) neu eingebunden werden.

Wir werden hauptsächlich nur in dem Package `com.watchandchill`, das all unsere programmspezifischen Dateien enthält, arbeiten. Dateien im `ApplicationKit`¹ werden nicht bearbeitet. `ApplicationKit` ist ein Framework, das den Umgang mit JavaFX vereinfacht und gleichzeitig eine einfache Anbindung zu einer SQLite-Datenbank bereitstellt. Es folgt dabei nahezu komplett dem MVC-Muster. Nähere Informationen zu diesem Framework findet man in der Dokumentation².

1 Datenbank anlegen

Da persistente Datenhaltung erforderlich ist, müssen wir eine Datenbank in SQLite3³ erzeugen. Voraussetzung dafür ist, dass das Konsolenprogramm⁴ für SQLite3 auf dem Rechner installiert ist und gestartet werden kann.

Die DDL- und ggf. DML-Befehle werden per Datei eingelesen. Dazu legen wir eine Datei `watchandchill.sql` im Projektverzeichnis an, die alle Befehle zum Initialisieren der Datenbank enthält. Vor den eigentlichen Befehlen müssen die Anweisungen aus Codeblock 1 eingefügt werden.

```
1 PRAGMA encoding = 'UTF-8';  
2 PRAGMA foreign_keys = ON;  
3 PRAGMA auto_vacuum = 1;  
4 PRAGMA automatic_index = ON;
```

¹`com.alexanderthelen.applicationkit`

²siehe Ordner „doc“ im Projekt

³<https://www.sqlite.org>

⁴<https://www.sqlite.org/download.html>

Codeblock 1: PRAGMA-Statements

Diese PRAGMA-Anweisungen sind wichtig, um volle Funktionalität (z.B. Fremdschlüsselintegrität) zu gewährleisten. Ausführliche Informationen dazu bietet die Dokumentation⁵ zu SQLite3.

Es ist absolut notwendig, dass man sich beim Erstellen der Datenbank auf folgende von ApplicationKit unterstützte Datentypen beschränkt:

- INTEGER
- BOOLEAN
- REAL
- NUMERIC
- TEXT
- VARCHAR
- BLOB

Gibt man einen anderen Datentyp an, so treten eventuell Fehler auf. Ein wichtiger Hinweis: Will man in bestimmten Spalten der Datenbanktabellen auch Zeilenumbrüche speichern, so ist dies nur durch den Datentyp TEXT möglich, da erst bei diesem TextArea-Instanzen in der Zeilenansicht (siehe unten) erstellt werden.

Nachdem nun diese Datei angelegt wurde, öffnen wir die Konsole, wechseln in das Projektverzeichnis und starten das SQLite3-Konsolenprogramm (meistens mit dem Befehl `sqlite3`). Eine Eingabeaufforderung erscheint, die nicht nur SQL-Befehle sondern auch andere Anweisungen entgegennimmt. Durch die Anweisung `.open watchandchill.db` wird eine leere Datenbank erstellt, sofern die Datei `watchandchill.db` in dem Verzeichnis, in dem SQLite3 ausgeführt wird, nicht existiert; ansonsten wird versucht, die entsprechende Datei zu öffnen. Der Befehl `.read watchandchill.sql` liest die eben erstellten Befehle ein und gibt gegebenenfalls Fehlermeldungen aus. Danach beenden wir die Eingabeaufforderung mit `.exit`.

Sollten beim Einlesen Fehlermeldungen auftauchen, muss man die angelegte Datenbank `watchandchill.db` löschen und die oben erwähnten Schritte, nachdem die Datei `watchandchill.sql` auf Fehler überprüft wurde, wiederholen.

2 Verbindung einrichten und verwalten

Damit das Programm weiß, welche Datenbank es verwalten soll, muss zuerst die Verbindung zur Datenbank gesetzt bzw. angepasst werden. Dies geschieht in der `start()`-Methode der Klasse `com.watchandchill.Application`: Mit der Anweisung

⁵<https://www.sqlite.org/pragma.html>

setConnection(path) am Anfang des Methodenrumpfes, wobei path vom Datentyp Path ist und den Pfad zur Datenbank angibt, kann man die Verbindung initialisieren. Codeblock 2 dient als Beispiel dazu und setzt voraus, dass die Datenbankdatei watchandchill.db in dem Arbeitsverzeichnis liegt.

```
1 setConnection(new Connection(Paths.get("watchandchill.db")));
```

Codeblock 2: Setzen der Datenbankverbindung

Durch Application.getInstance().getConnection() kann nun programmweit, sofern die Klasse com.watchandchill.Application in der Java-Datei importiert wird, auf die Verbindung zugegriffen werden. Auf der Verbindung stellen die Methoden

- createStatement() und
- preparedStatement(sql)

eine Möglichkeit bereit, SQL-Anweisungen zu erstellen, wobei sql die Zeichenkette des SQL-Befehls ist. Der Rückgabewert dieser Methoden ist eine Statement⁶- bzw. PreparedStatement⁷-Instanz der entsprechenden Klasse in Java. Auf diesen Instanzen kann man dann die entsprechenden Befehle ausführen. Codeblock 3 dient als einfaches Beispiel.

```
1 String sql = "SELECT * FROM table";
2 Statement statement = Application.getInstance().getConnection().
  createStatement();
3 ResultSet resultSet = statement.executeQuery(sql);
4 while (resultSet.next()) {
5     System.out.println(resultSet.getObject("nameOfFirstColumn"));
6 }
```

Codeblock 3: Einfache SELECT-Anfrage in Java

Für weitere Informationen und Beispiele (insbesondere für PreparedStatement-Instanzen) sei auf das Internet und die Dokumentation zu Java verwiesen.

3 Authentifizierung ermöglichen

Sobald wir das Programm starten, soll eine Anmeldung erscheinen. Da das Programm aber nicht nur eine Anmeldung sondern auch eine Registrierung ermöglichen muss, eignet sich am besten eine Unterklasse des AuthenticationViewController des ApplicationKits. Diese Unterklasse besitzt den gleichen Namen und befindet sich im Package com.watchandchill.gui. Sie bettet schon eine LoginViewController-Instanz und eine RegistrationViewController-Instanz in sich ein.

⁶<https://docs.oracle.com/javase/8/docs/api/java/sql/Statement.html>

⁷<https://docs.oracle.com/javase/8/docs/api/java/sql/PreparedStatement.html>

WatchAndChill
Anmeldung

Benutzername:

Passwort:

Neu? [Dann registrier dich hier!](#)

Abbildung 1: View des `com.watchandchill.gui.LoginViewController`

WatchAndChill
Registrierung

Account:
Benutzername:

E-Mail:

Passwort:

Schauspieler: ☐ Ja ☒ Nein

Vorname:

Nachname:

Künstlername:

Geburtsdatum:

Geburtsort:

Premium: ☐ Ja ☒ Nein

Abbildung 2: View des `com.watchandchill.gui.RegistrationViewController`

Unsere Aufgabe ist es jetzt, die Methoden

- `loginUser(Data data)` und
- `registerUser(Data data)`

zu implementieren. Beide Methoden funktionieren nach folgendem Prinzip: In der jeweiligen Methode wird versucht, den Nutzer anzumelden bzw. zu registrieren. Der Parameter `data` vom Datentyp `Data` stellt dabei die Informationen

bereit, die in den Eingabefeldern eingegeben wurden. Der Zugriff auf eine Information funktioniert wie bei einer HashMap mit der Instanzmethode `get(key)`, wobei `key` der Schlüssel der entsprechenden Information ist. Ist dieser Schlüssel nicht vorhanden, so wird eine `SQLException`-Instanz geworfen.

Zum Beispiel gibt ein Aufruf von `data.get("password")` das eingegebene Passwort zurück. Ein einfacher Aufruf von `System.out.println(data)` gibt Klarheit über die sich in `data` befindenden Informationen. Sollte die Anmeldung bzw. Registrierung aus irgendeinem Grund fehlschlagen, muss eine `SQLException`-Instanz geworfen werden.

Der Methodenrumpf der Methode `registerUser(Data data)` soll einen neuen Benutzer in den entsprechenden Tabellen anlegen. Sollte der Benutzername schon existieren, ist die Registrierung ungültig und es muss eine `SQLException`-Instanz geworfen werden.

Der Methodenrumpf der Methode `loginUser(Data data)` soll überprüfen, ob die angegebene Benutzername-Passwort-Kombination in der Datenbank enthalten ist. Ist dies nicht der Fall, muss eine `SQLException`-Instanz geworfen werden. Ist die Anmeldung jedoch erfolgreich, dann bietet es sich an, den Benutzernamen und andere Informationen (wie Rechte, Name, E-Mail, ...) aus der Datenbank programmweit zu speichern. Beispielsweise kann durch Codeblock 4 der zugehörige Inhalt des Schlüssels `email` verwaltet werden.

```
1 Application.getInstance().getData().put("email", "max@mustermann.de");  
2 Application.getInstance().getData().get("email");
```

Codeblock 4: Verwaltung von programmweiten Daten

Sollte man bei `Data`-Instanzen auf Schlüssel zugreifen wollen, die nicht existieren, so wird ein neues `SQLException`-Objekt geworfen, die dieses Problem mit „Schlüssel x nicht vorhanden.“ bestätigt, wobei `x` der entsprechende Schlüsselname ist.

4 Tabellen anzeigen und verwalten

Sobald sich ein Nutzer angemeldet hat, erscheint eine `MasterDetailViewController`-Instanz des `ApplicationKits`. Der View ist zweigeteilt: Links sieht man den auf das Programm angepassten `MasterViewController` und rechts den entsprechenden `DetailViewController`. Klickt man links im `MasterViewController` auf einen Eintrag, so wird der `DetailViewController` entsprechend neu gesetzt. Auf unser Projekt bezogen heißt das Folgendes: Klickt man links auf einen Tabellentitel, wird rechts die entsprechende Tabelle in einer `TableViewController`-Instanz angezeigt. Eingerückte Tabellentitel sind als Beziehung oder untergeordnete Entitäten zu dem übergeordneten Tabellentitel zu interpretieren. Zum Beispiel ist „Videos“ unter „Watchlists“ eingerückt und sollte demnach als „Videos zu Watchlists“ gelesen werden. In dieser Tabelle „Videos“ sollen folglich nicht alle Informationen zu einem gewissen Video dargestellt werden, sondern nur die Tatsache, welches Video zu welcher Watchlist gehört, angezeigt werden. Die Informationen zu einem Video findet man dann entweder in „Filme“, in „Trailer zu Filmen“

oder in „Folgen zu Serien“. Manchmal allerdings ist dies nicht möglich und alle Informationen müssen in der eingerückten bzw. untergeordneten Tabelle dargestellt werden: Beispielsweise gibt es die übergeordnete Tabelle „Folgen“ nicht. Demnach müssen in „Folgen zu Serien“ alle Informationen einer Folge (zusätzlich zur Zuordnung zu einer Serie) dargestellt werden.

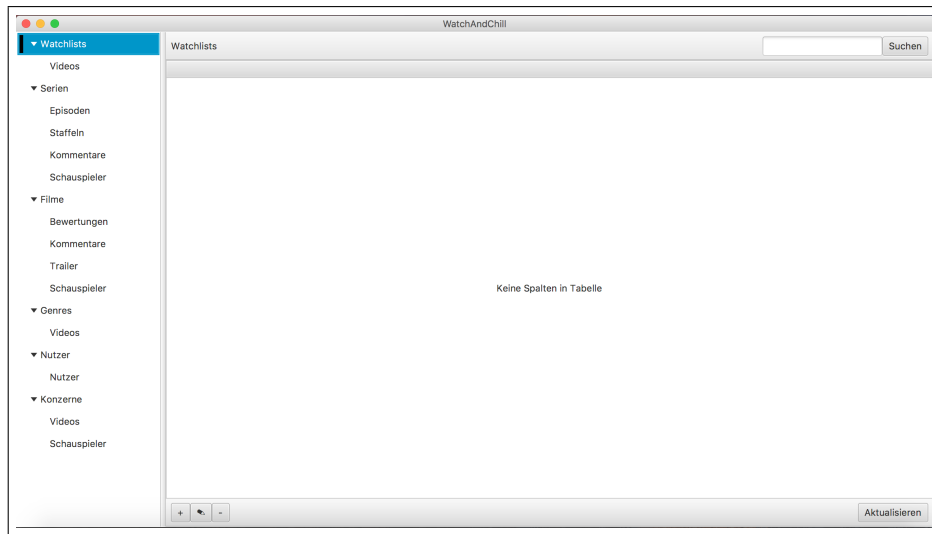


Abbildung 3: View des eigentlichen Programms (könnte leicht anders aussehen)

Zu jedem übergeordneten Tabellentitel gibt es in `com.watchandchill.table` ein entsprechendes Package, in dem alle zu implementierenden Tabellenklassen enthalten sind.

Jede Tabellenklasse in `com.musicous.table` erbt von der abstrakten `Table`-Klasse des `ApplicationKits`. Dadurch sind die Methoden

- `getSelectQueryForTableWithFilter(String filter)`,
- `getSelectQueryForRowWithData(Data data)`,
- `insertRowWithData(Data data)`,
- `updateRowWithData(Data oldData, Data newData)` und
- `deleteRowWithData(Data data)`

zu implementieren. Wir gehen diese Methoden und deren Bedeutung für die Tabelle in den folgenden Abschnitten anhand (unserer Version) der Tabelle „Watchlists“ Schritt für Schritt durch.

4.1 Tabellenansicht aufbauen und füllen

Wenn wir auf einen Tabellentitel klicken, wollen wir rechts die entsprechende Tabelle anzeigen lassen. Welche Spalten und Zeilen sollen in der Tabelle angezeigt werden, wenn wir „Watchlists“ anklicken?

Die Methode `getSelectQueryForTableWithFilter(String filter)` bestimmt diese Aspekte, indem der Rückgabewert ein SELECT-Befehl ist. In dieser SELECT-Anweisung legt die Projektion die folgenden anzuzeigenden Spalten fest:

- ID,
- Titel,
- Privat? und
- Nutzer.

Der Parameter `filter` beinhaltet die Eingabe, die man als letztes im Suchfeld über der Tabelle bestätigt hat. Will man in der Tabelle „Watchlists“ nach Nutzern filtern, so muss dem SELECT-Befehl noch die entsprechende WHERE-Bedingung angehängen werden. Die Implementierung dieser Methode in der Klasse `Filme` sieht folgendermaßen aus:

```
1 @Override
2 public String getSelectQueryForTableWithFilter(String filter)
3     throws SQLException {
4     String selectQuery = "SELECT ID, Bezeichnung AS Titel, Privat
5     AS \"Privat?\", Premiumnutzer AS Nutzer FROM Watchlist";
6     if (filter != null && !filter.isEmpty()) {
7         selectQuery += " WHERE Premiumnutzer LIKE '%" + filter + "
8         %'";
9     }
10    return selectQuery;
```

Codeblock 5: `getSelectQueryForTableWithFilter(String filter)` für Watchlists

Es können beliebige Tabellen der Datenbank miteinander in Verbund treten, sodass man beliebige Informationen in der Tabelle darstellen kann.

4.2 Zeilenansicht aufbauen und füllen

Wenn wir auf eine Zeile der Tabelle „Watchlists“ doppelklicken, wollen wir eine Detailansicht dieser Zeile (kurz: Zeilenansicht) anzeigen lassen, die wir gegebenenfalls bearbeiten können. Manchmal ist es notwendig, in der Tabelle selbst weniger bzw. andere Spalten anzuzeigen als in der Zeilenansicht. Wie legen wir diesen Sachverhalt fest? Wie erreichen wir überhaupt, dass die richtige Zeile in der Zeilenansicht angezeigt wird?

Die Methode `getSelectQueryForRowWithData(Data data)` entscheidet, welche Informationen in dieser Ansicht angezeigt werden. Dazu gibt die Methode eine SELECT-Anweisung zurück. In diesem Befehl legt die Projektion wie auch bei der vorherigen Methode die anzuzeigenden Spalten fest. Dieses Mal werden aber andere Spalten angezeigt:

- ID von Watchlist,
- Bezeichnung,
- Privat und
- Premiumnutzer.

Der Parameter `data` ist vom Datentyp `Data`, der schon weiter oben eingeführt wurde. Dieser stellt nun die Informationen bereit, die in der markierten Zeile der Tabelle stehen. Dabei bestehen die Schlüssel dieses Parameters aus dem Tabellennamen in der Datenbank, einem Punkt und dem Spaltennamen. Die Schlüssel werden also von dem Rückgabewert der Methode `getSelectQueryForTableWithFilter` (String filter) bestimmt und sind folglich:

- "Watchlist.ID",
- "Watchlist.Titel",
- "Watchlist.Privat?" und
- "Watchlist.Nutzer".

Die Methode lässt sich folgendermaßen implementieren:

```

1 @Override
2 public String getSelectQueryForRowWithData(Data data) throws
   SQLException {
3     String selectQuery = "SELECT ID AS \"ID von Watchlist\",
   Bezeichnung, Privat, Premiumnutzer FROM Watchlist WHERE ID = "
   + data.get("Watchlist.ID");
4     return selectQuery;
5 }

```

Codeblock 6: `getSelectQueryForRowWithData(Data data)` für Watchlists

Wie man erkennt, ist es absolut notwendig in der Tabellenansicht den Primärschlüssel in Spalten anzeigen zu lassen, da man in der Methode ansonsten nicht eindeutig auf die markierte Zeile zugreifen kann. In diesem Beispiel ist die Menge {ID} der Primärschlüssel, auf deren Element „ID“ man per `data.get("Watchlist.ID")` zugreifen kann.

Abbildung 4: Erfolgreiche Implementation von `getSelectQueryForRowWithData(Data data)` für Watchlists

Die Zeilenansicht dient nicht nur der genaueren Betrachtung und Bearbeitung der in der Tabelle ausgewählten Zeile, sondern wird auch angezeigt, wenn man auf den „+“-Button klickt.

4.3 Zeile hinzufügen

Wie erwähnt, wird die Zeilenansicht auch angezeigt, wenn man den „+“-Button betätigt. Sobald der Nutzer in der Zeilenansicht auf den „Speichern“-Button klickt, sollen die eingegebenen Informationen, sofern diese gültig sind, gespeichert werden. Wie erreichen wir das?

In der Methode `insertRowWithData(Data data)` werden sämtliche SQL-Anweisungen ausgeführt, die zum Einfügen der in `data` stehenden Informationen in die Datenbank notwendig sind. Diese Informationen sind genau diejenigen, die in der Zeilenansicht in die entsprechenden Eingabefelder eingegeben wurden. Der Parameter `data` beinhaltet diese Daten, die wieder per Schlüssel angesprochen werden. Diese bestehen wieder aus dem Tabellennamen in der Datenbank, einem Punkt und dem Spaltennamen. Allerdings werden die Schlüssel nun von dem `SELECT`-Befehl der Methode `getSelectQueryForRowWithData(Data data)` bestimmt, da dieser ja erst den Aufbau der Zeilenansicht bestimmt. Bevor wir weitere Hinweise zu der Methode geben, geben wir den Beispielcode zu der Tabelle „Watchlists“ an (siehe Codeblock 7).

```

1  @Override
2  public void insertRowWithData(Data data) throws SQLException {
3      if ((Integer) Application.getInstance().getData().get("
4          permission") >= 1) {
5          throw new SQLException("Nicht die notwendigen Rechte.");
6      }
7  }
```

```

6
7   PreparedStatement preparedStatement = Application.getInstance()
   .getConnection().prepareStatement("INSERT INTO Watchlist(
   Bezeichnung, Privat, Premiumnutzer) VALUES (?, ?, ?)");
8   preparedStatement.setObject(1, data.get("Watchlist.Bezeichnung"
   ));
9   preparedStatement.setObject(2, data.get("Watchlist.Privat"));
10  preparedStatement.setObject(3, Application.getInstance().
   getData().get("username"));
11  preparedStatement.executeUpdate();
12  /*preparedStatement.getGeneratedKeys().next();
13  int index = preparedStatement.getGeneratedKeys().getInt(1);*/
14 }

```


Codeblock 7: insertRowWithData(Data data) für Watchlists

Es fällt auf, dass, obwohl wir in der Zeilenansicht sowohl ID als auch Benutzername eintragen und dementsprechend auf die Eingaben per `data.get("Watchlist.ID von Watchlist")` bzw. `data.get("Watchlist.Premiumnutzer")` zugreifen können, diese Informationen nicht verwendet werden, da

- die Spalte ID in der ersten INSERT-Anweisung nicht aufgelistet wird, weil diese in unserer Beispieldatenbank automatisch inkrementiert, und
- die Spalte Premiumnutzer mit dem Benutzernamen unseres angemeldeten Nutzers gefüllt wird, weil jede angelegte Watchlist automatisch vom eingeloggteten Nutzer und von keinem anderen erstellt wird.

Eine weitere Besonderheit ist die `if`-Abfrage in Zeile 3 bis 5. Bei der Anmeldung haben wir nicht nur geprüft, ob die Kombination aus Benutzername und Passwort in der Datenbank vorhanden ist, sondern auch herausbekommen, welche Rechte dieser Nutzer hat, sodass wir diese programmweit zur Verfügung stellen konnten. Die `permission 1` steht in unserem Beispiel für Premiumrechte. Sind diese nicht gegeben, so wird eine neue `SQLException`-Instanz geworfen und alle in der Methode darauf folgenden Anweisungen werden nicht mehr ausgeführt. Generell muss bei implementierungsspezifischen Fehlern eine `SQLException`-Instanz geworfen werden.

4.4 Zeile bearbeiten

Lässt man sich eine Zeile der Tabelle in der Zeilenansicht genauer anzeigen (entweder mit Doppelklick auf diese Zeile oder per Klick auf den „“-Button), so kann man jederzeit auf den „Speichern“-Button klicken, sodass unter Umständen die Informationen einer Zeile in der Datenbank aktualisiert werden.

Die Methode `updateRowWithData(Data oldData, Data newData)` wird ausgeführt, sobald dieser Button betätigt wurde. Sie stellt zwei Parameter zur Verfügung:

- `oldData` beinhaltet die in den Eingabefeldern stehenden Informationen vor Bearbeitung und
- `newData` enthält die in den Eingabefeldern stehenden Informationen nach Bearbeitung.

Der Zugriff auf die entsprechenden Informationen erfolgt wieder durch Schlüssel gemäß der Spaltenauflistung im SELECT-Befehl von `getSelectQueryForRowWithData` (`Data data`). Auch hier gilt: Bei implementierungsspezifischen Fehlern muss eine `SQLException`-Instanz geworfen werden (wie zum Beispiel bei den beiden `if`-Abfragen im Codeblock 8).

```

1  @Override
2  public void updateRowWithData(Data oldData, Data newData) throws
    SQLException {
3      if (((Integer) Application.getInstance().getData().get("
        permission") >= 1) {
4          throw new SQLException("Nicht die notwendigen Rechte.");
5      } else if (!Application.getInstance().getData().get("username")
        .equals(oldData.get("Watchlist.Premiumnutzer"))) {
6          throw new SQLException("Nicht der gleiche Nutzer.");
7      }
8
9      PreparedStatement preparedStatement = Application.getInstance()
        .getConnection().prepareStatement("UPDATE Watchlist SET
        Bezeichnung = ?, Privat = ? WHERE ID = ?");
10     preparedStatement.setObject(1, newData.get("Watchlist.
        Bezeichnung"));
11     preparedStatement.setObject(2, newData.get("Watchlist.Privat"));
12     ;
13     preparedStatement.setObject(3, oldData.get("Watchlist.ID von
        Watchlist"));
14     preparedStatement.executeUpdate();
15 }

```

Codeblock 8: `updateRowWithData(Data oldData, Data newData)` für Watchlists

4.5 Zeile löschen

Sobald man in der Tabelle eine Zeile markiert, kann man durch Klicken auf den „Entfernen“-Button direkt diese Zeile löschen.

Die Methode `deleteRowWithData(Data data)` wird ausgeführt, sobald dieser Button betätigt wurde. Da das Löschen anhand von Informationen aus der Tabelle geschieht, bauen sich die Schlüssel des Parameters `data` anhand des SELECT-Befehls der Methode `getSelectQueryForTableWithFilter(String filter)` auf. Der Beispielcode zu „Watchlists“ ist in Codeblock 9 zu sehen.

```

1  @Override
2  public void deleteRowWithData(Data data) throws SQLException {
3      if (((Integer) Application.getInstance().getData().get("
        permission") >= 2) {
4          throw new SQLException("Nicht die notwendigen Rechte.");
5      } else if (!Application.getInstance().getData().get("username")
        .equals(data.get("Watchlist.Premiumnutzer"))) {
6          throw new SQLException("Nicht der gleiche Nutzer.");
7      }
8
9      PreparedStatement preparedStatement = Application.getInstance()
        .getConnection().prepareStatement("DELETE FROM Watchlist WHERE
        ID = ?");

```

```
10 |     preparedStatement.setObject(1, data.get("Watchlist.ID"));
11 |     preparedStatement.executeUpdate();
12 | }
```

Codeblock 9: deleteRowWithData(Data data) für Watchlists