Wicket – JDBC

TUTORIAL 6

DBMS 2013-2014 Fall

Nagehan Ilhan

Wicket – JDBC

- A major problem with the application as implemented so far is that the data the user enters do not persist.
- Every time, the application starts with an empty collection and added movies are lost when the application is shut down. In this chapter we will see how to store the data in a database.

SQL & **JDBC**

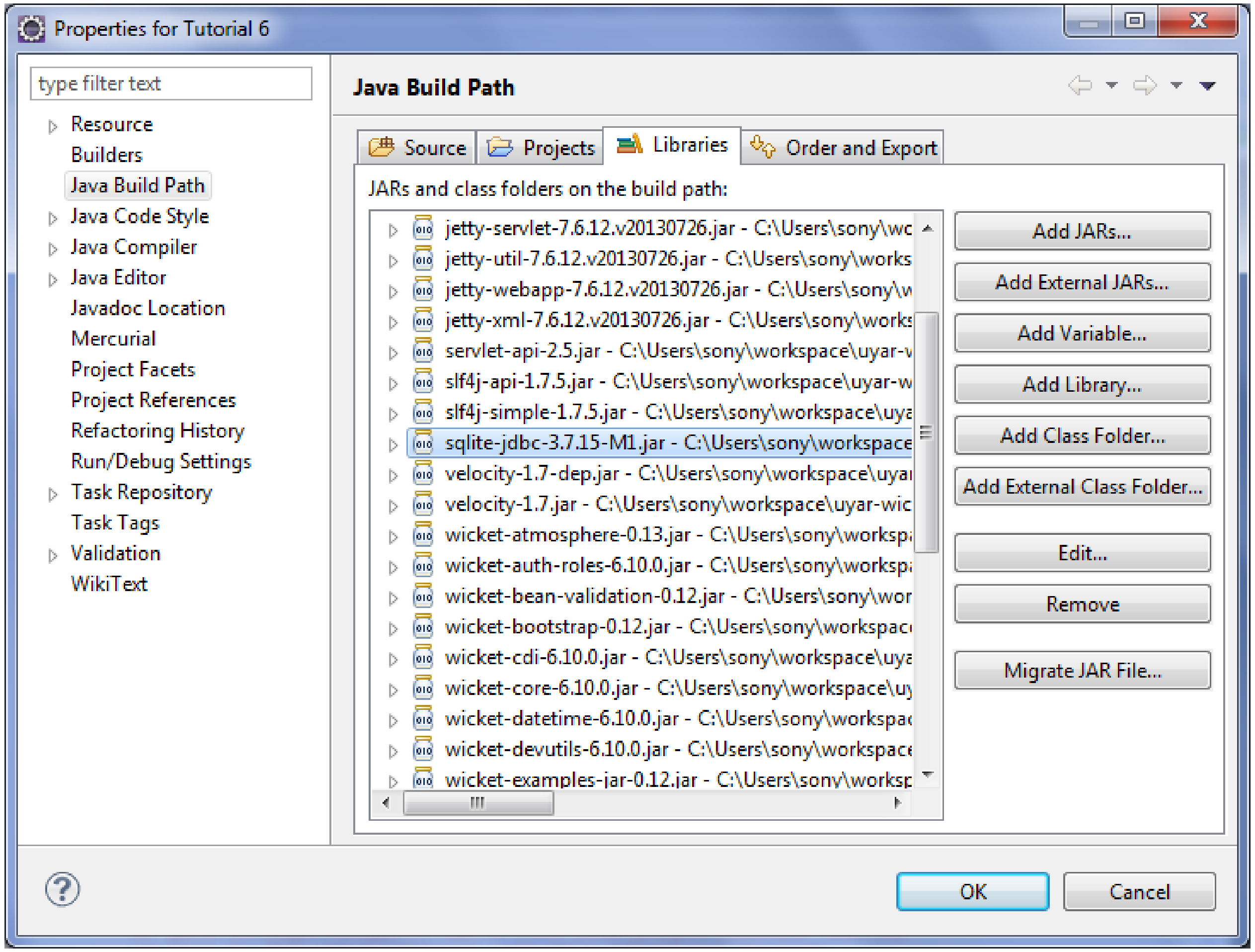
«
The Java Database Connectivity (JDBC) API is the industry standard for database-independent connectivity between the Java programming language and a wide range of databases SQL databases and other tabular data sources, such as spreadsheets or flat files. The JDBC API provides a call-level API for SQL-based database access.

JDBC technology allows you to use the Java programming language to exploit "Write Once, Run Anywhere" capabilities for applications that require access to enterprise data. With a JDBC technology-enabled driver, you can connect all corporate data even in a heterogeneous environment.
»

Wicket – JDBC

- uses SQLite for database => add SQLite JDBC driver.
- Download the JDBC driver JAR file for SQLite from the following page, put it in the lib directory, and add it to the project classpath:
- http://www.xerial.org/trac/Xerial/wiki/SQLiteJDBC

Wicket – JDBC


Properties for Tutorial 6

type filter text

- Resource
- Builders
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- Javadoc Location
- Mercurial
- Project Facets
- Project References
- Refactoring History
- Run/Debug Settings
- Task Repository
- Task Tags
- Validation
- WikiText

Java Build Path

Source | Projects | Libraries | Order and Export

JARs and class folders on the build path:

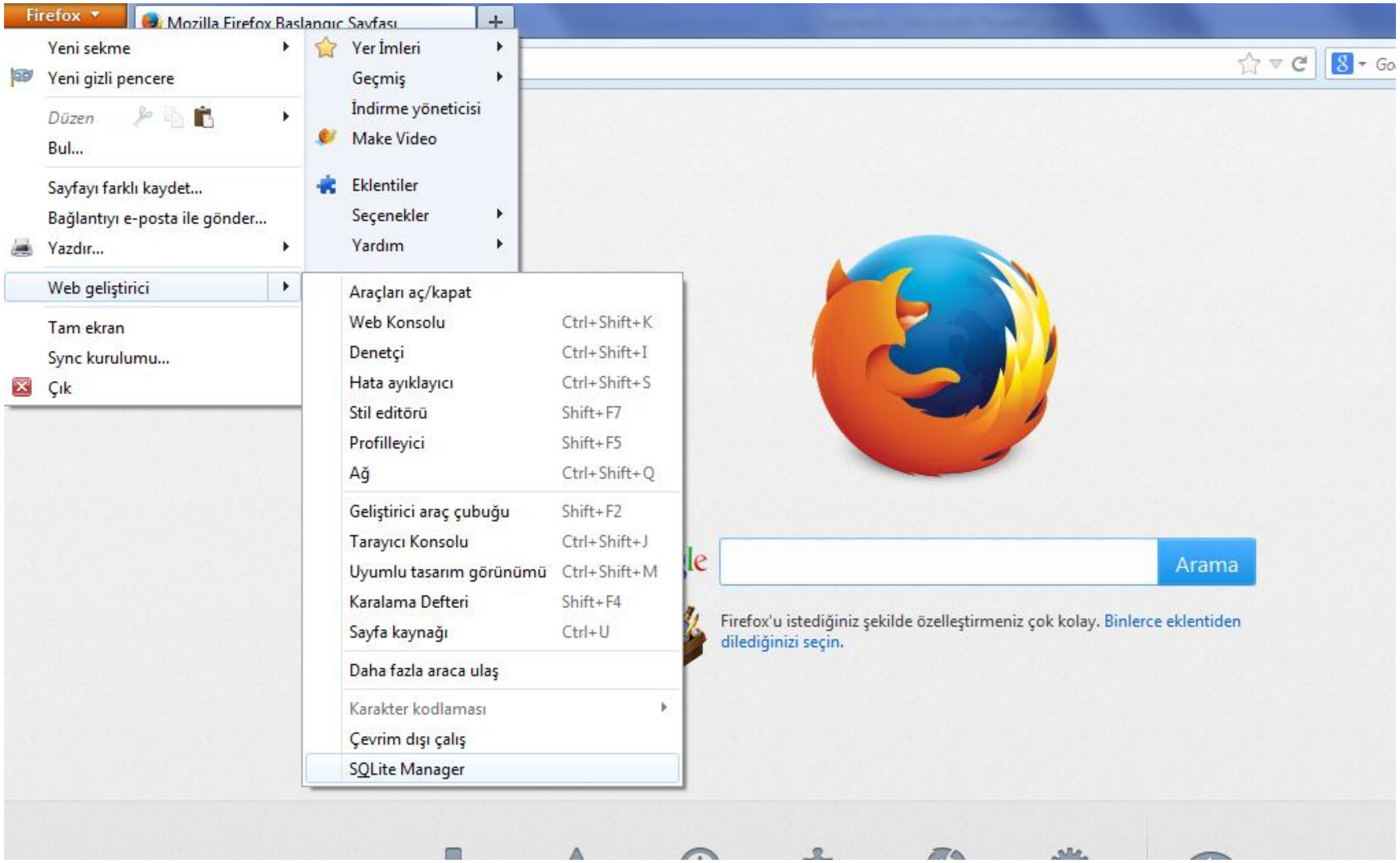
```
jetty-servlet-7.6.12.v20130726.jar - C:\Users\sony\wo
jetty-util-7.6.12.v20130726.jar - C:\Users\sony\works
jetty-webapp-7.6.12.v20130726.jar - C:\Users\sony\w
jetty-xml-7.6.12.v20130726.jar - C:\Users\sony\works
servlet-api-2.5.jar - C:\Users\sony\workspace\uyar-v
slf4j-api-1.7.5.jar - C:\Users\sony\workspace\uyar-w
slf4j-simple-1.7.5.jar - C:\Users\sony\workspace\uya
sqlite-jdbc-3.7.15-M1.jar - C:\Users\sony\workspace
velocity-1.7-dep.jar - C:\Users\sony\workspace\uyar
velocity-1.7.jar - C:\Users\sony\workspace\uyar-wic
wicket-atmosphere-0.13.jar - C:\Users\sony\worksp
wicket-auth-roles-6.10.0.jar - C:\Users\sony\worksp
wicket-bean-validation-0.12.jar - C:\Users\sony\wor
wicket-bootstrap-0.12.jar - C:\Users\sony\workspac
wicket-cdi-6.10.0.jar - C:\Users\sony\workspace\uya
wicket-core-6.10.0.jar - C:\Users\sony\workspace\uy
wicket-datetime-6.10.0.jar - C:\Users\sony\workspac
wicket-devutils-6.10.0.jar - C:\Users\sony\workspace
wicket-examples-jar-0.12.jar - C:\Users\sony\worksp
```

Wicket – JDBC

- You can use the SQLite Manager add-on for Firefox to manage SQLite databases:
- https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/
- Next, create a database with the name movies.sqlite in your home directory and create a table in it using the following SQL statement:

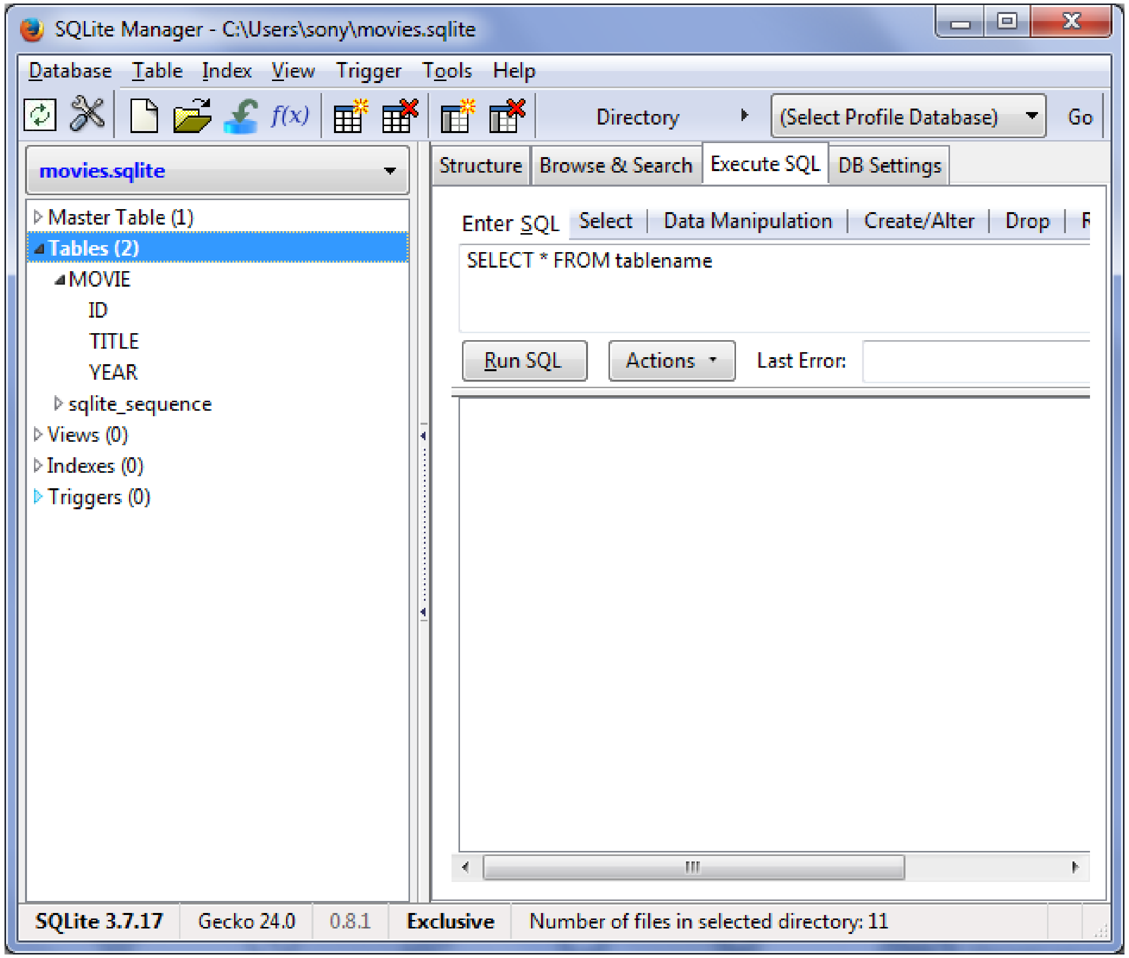
```
CREATE TABLE MOVIE
 ( ID INTEGER PRIMARY KEY AUTOINCREMENT,
   TITLE VARCHAR(80) NOT NULL,
   YEAR INTEGER )
```

Wicket – JDBC



Firefox'u istediğiniz şekilde özelleştirmeniz çok kolay. Binlerce eklentiden dilediğinizi seçin.

Wicket – JDBC



```
SELECT * FROM tablename
```

Wicket – JDBC

- Add a Java interface with the name IMovieCollection.
- The interface has to declare the methods.

```
package wicket.quickstart;
import java.util.List;

public interface IMovieCollection {
   public List<Movie> getMovies();
   public void addMovie(Movie movie);
   public void deleteMovie(Movie movie);
   public void updateMovie(Movie movie);
}
```

Wicket – JDBC

- The **MovieCollection** class based on lists already implements this interface; we just have to acknowledge this by changing the class definition:

```
public MovieCollection implements IMovieCollection
{
    …
}
```

Wicket – JDBC

- We also have to change the code pieces that refer to the **MovieCollection** class so that they will use the **IMovieCollection** interface.

```
public class WicketApplication extends WebApplication {
    private IMovieCollection _collection;

    @Override
    public Class<? extends WebPage> getHomePage() {
        return HomePage.class;
    }
  @Override
    public void init() {
        super.init();
        this._collection = new MovieCollection();
    }
  public IMovieCollection getCollection() {
        return this._collection;
    }
}
```

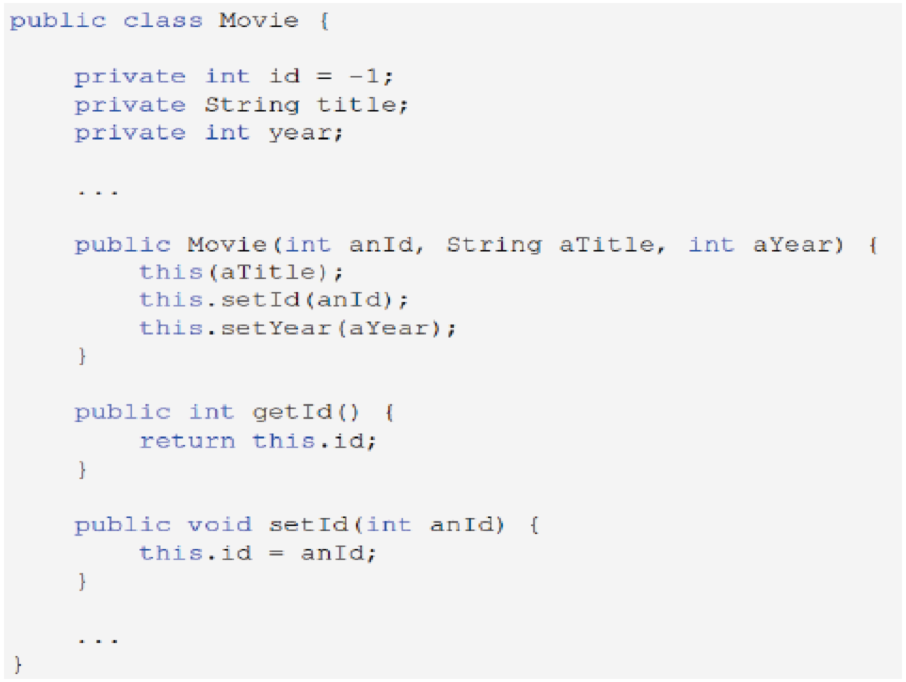
Wicket – JDBC

- Movie objects have identifiers. Getter and setter methods are also defined.
  - Movie.java

Id Property,

Constructor,

Getter & setter s



```
public class Movie {

    private int id = -1;
    private String title;
    private int year;

    ...

    public Movie(int anId, String aTitle, int aYear) {
        this(aTitle);
        this.setId(anId);
        this.setYear(aYear);
    }

    public int getId() {
        return this.id;
    }

    public void setId(int anId) {
        this.id = anId;
    }

    ...
}
```

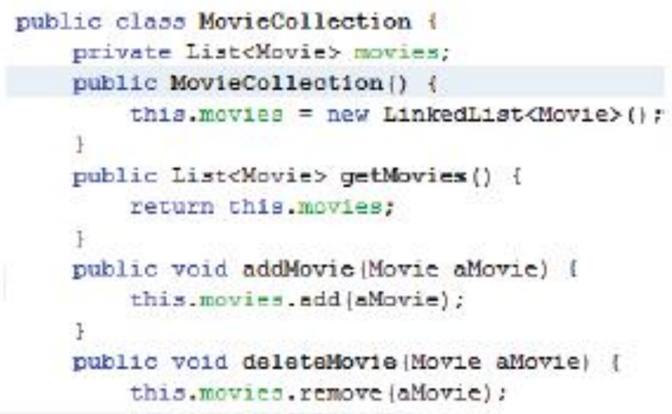
- MovieCollectionJDBC class

Movies were on the list, and add, delete operations were performed on list : Should be modified for DB

Sqlite jdbc driver

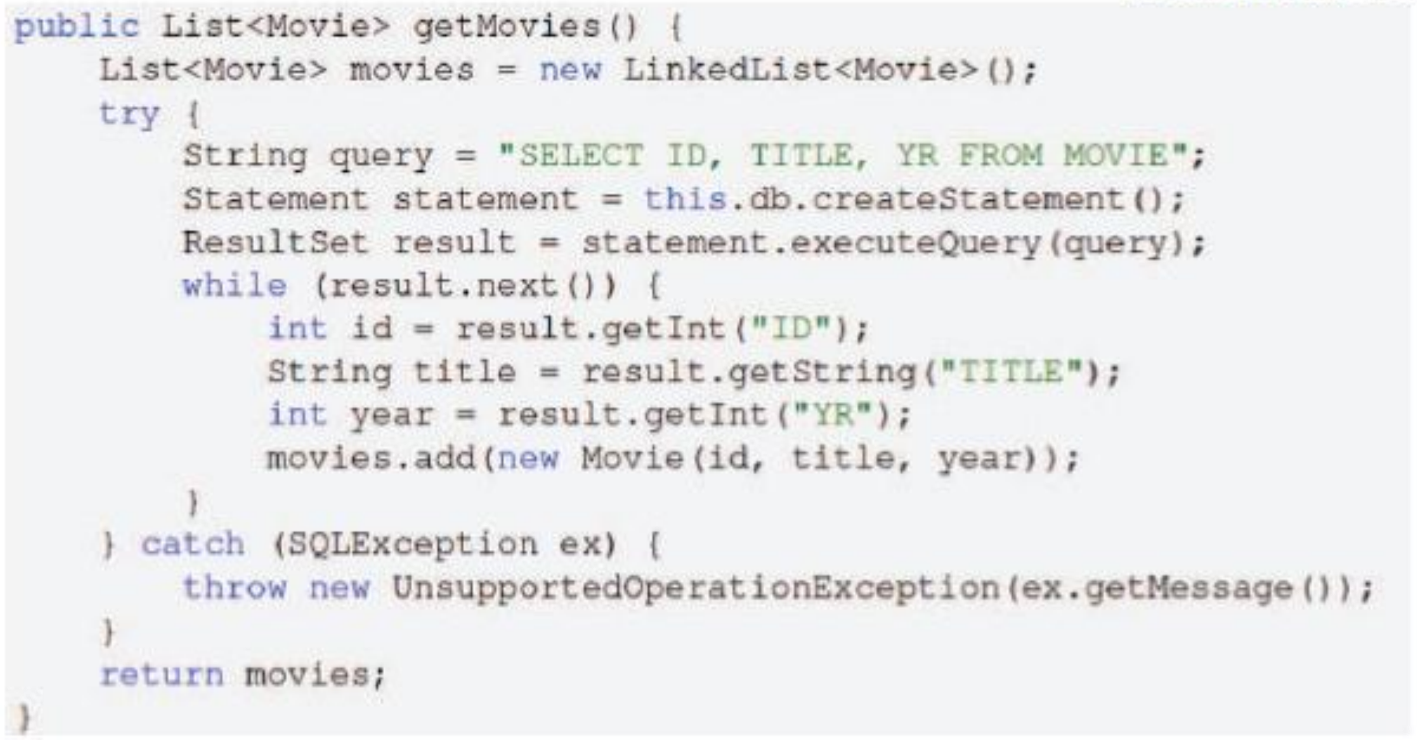
Connection is performed and kept in local variable **db**

```
public class MovieCollectionJDBC implements IMovieCollection {
   private Connection _db;

   public MovieCollectionJDBC(String dbFilePath) {
      try {
         Class.forName("org.sqlite.JDBC");
      } catch (ClassNotFoundException e) {
         throw new UnsupportedOperationException(e.getMessage());
      }
   try {
         String jdbcURL = "jdbc:sqlite:" + dbFilePath;
         this._db = DriverManager.getConnection(jdbcURL);
      } catch (SQLException ex) {
         throw new UnsupportedOperationException(ex.getMessage());
      }
   }

   public List<Movie> getMovies() { ...}

   public void addMovie(Movie movie) {  ... }

   public void deleteMovie(Movie movie) { … }

   public void updateMovie(Movie movie) {  ...  }
}
```



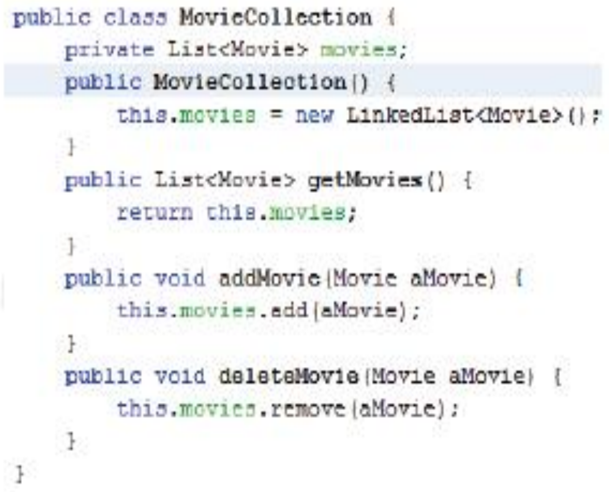
```java
public class MovieCollection {
    private List<Movie> movies;
    public MovieCollection() {
        this.movies = new LinkedList<Movie>();
    }
    public List<Movie> getMovies() {
        return this.movies;
    }
    public void addMovie(Movie aMovie) {
        this.movies.add(aMovie);
    }
    public void deleteMovie(Movie aMovie) {
        this.movies.remove(aMovie);
```

- MovieCollection.java – get Movies

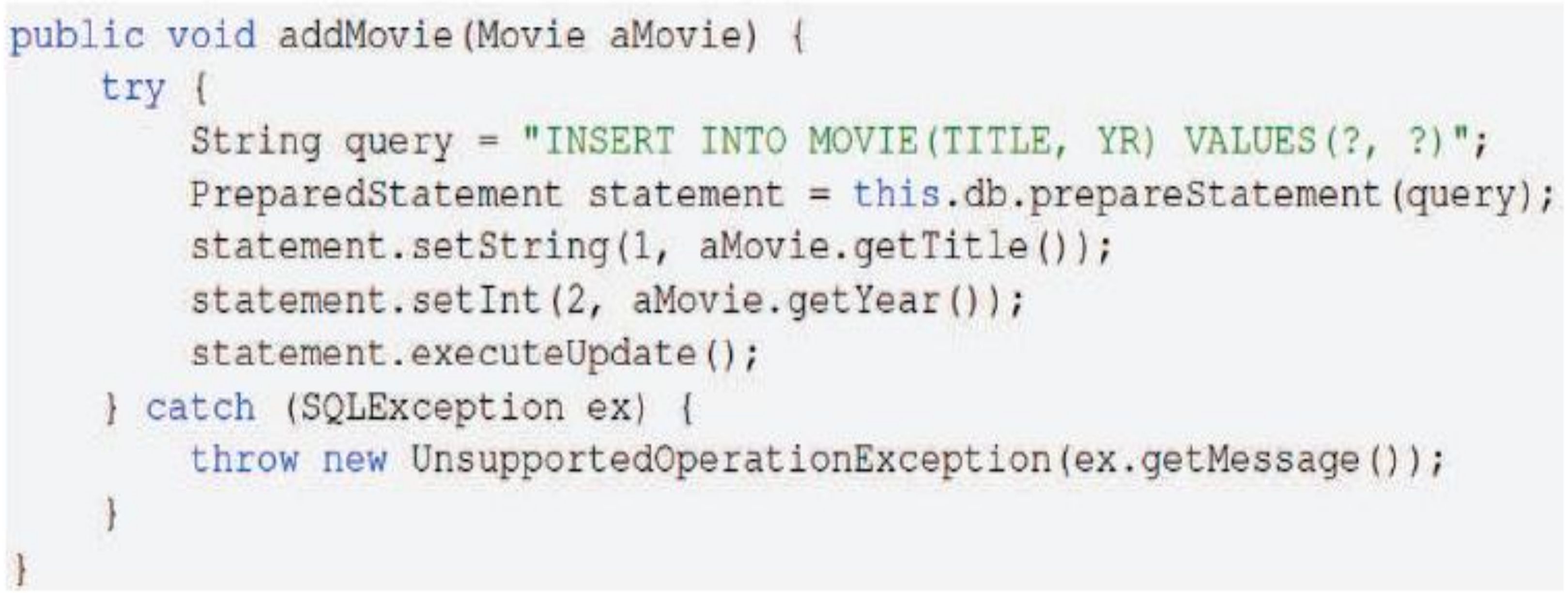


```java
public List<Movie> getMovies() {
    List<Movie> movies = new LinkedList<Movie>();
    try {
        String query = "SELECT ID, TITLE, YR FROM MOVIE";
        Statement statement = this.db.createStatement();
        ResultSet result = statement.executeQuery(query);
        while (result.next()) {
            int id = result.getInt("ID");
            String title = result.getString("TITLE");
            int year = result.getInt("YR");
            movies.add(new Movie(id, title, year));
        }
    } catch (SQLException ex) {
        throw new UnsupportedOperationException(ex.getMessage());
    }
    return movies;
}
```

- MovieCollection.java – add Movie

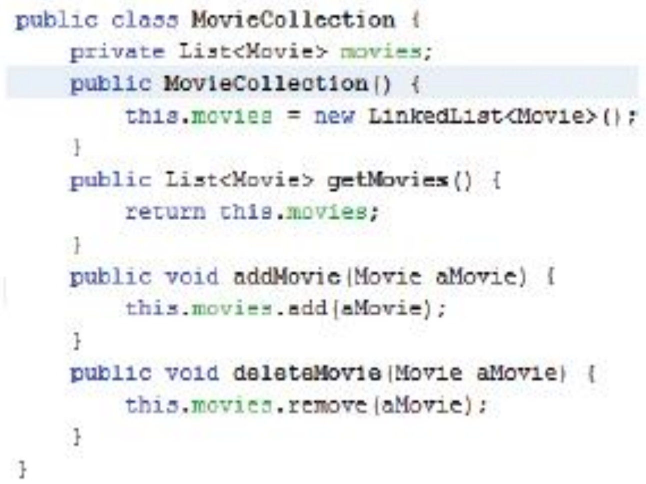


```
public class MovieCollection {
    private List<Movie> movies;
    public MovieCollection() {
        this.movies = new LinkedList<Movie>();
    }
    public List<Movie> getMovies() {
        return this.movies;
    }
    public void addMovie(Movie aMovie) {
        this.movies.add(aMovie);
    }
    public void deleteMovie(Movie aMovie) {
        this.movies.remove(aMovie);
    }
}
```

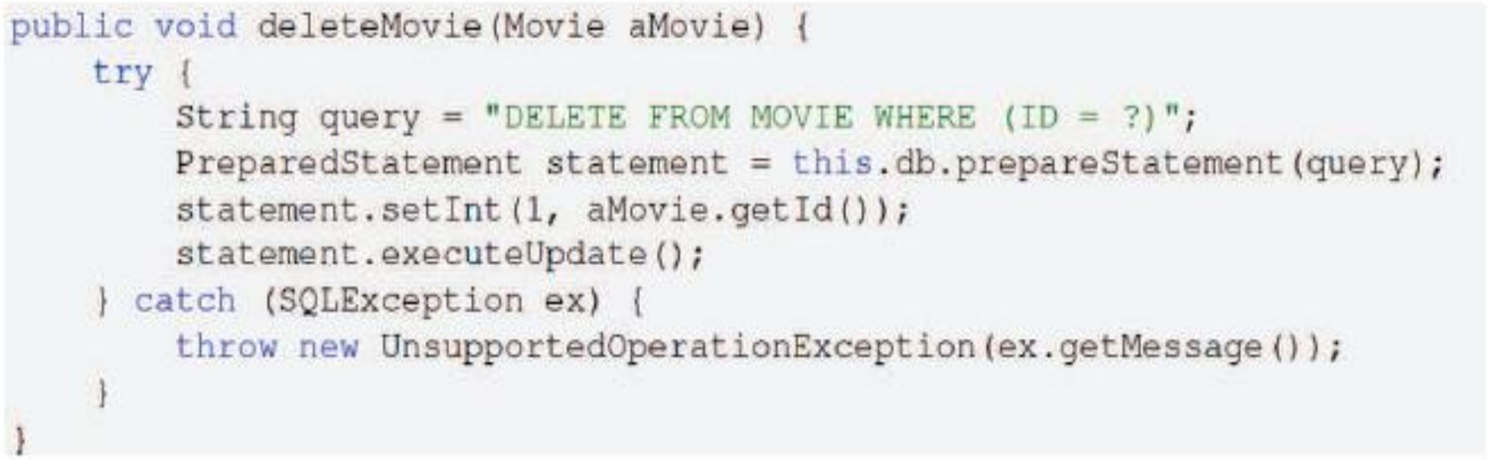


```
public void addMovie(Movie aMovie) {
    try {
        String query = "INSERT INTO MOVIE(TITLE, YR) VALUES(?, ?)";
        PreparedStatement statement = this.db.prepareStatement(query);
        statement.setString(1, aMovie.getTitle());
        statement.setInt(2, aMovie.getYear());
        statement.executeUpdate();
    } catch (SQLException ex) {
        throw new UnsupportedOperationException(ex.getMessage());
    }
}
```

- MovieCollection.java – delete Movie



```java
public class MovieCollection {
    private List<Movie> movies;
    public MovieCollection() {
        this.movies = new LinkedList<Movie>();
    }
    public List<Movie> getMovies() {
        return this.movies;
    }
    public void addMovie(Movie aMovie) {
        this.movies.add(aMovie);
    }
    public void deleteMovie(Movie aMovie) {
        this.movies.remove(aMovie);
    }
}
```



```java
public void deleteMovie(Movie aMovie) {
    try {
        String query = "DELETE FROM MOVIE WHERE (ID = ?)";
        PreparedStatement statement = this.db.prepareStatement(query);
        statement.setInt(1, aMovie.getId());
        statement.executeUpdate();
    } catch (SQLException ex) {
        throw new UnsupportedOperationException(ex.getMessage());
    }
}
```

- MovieCollection.java – update Movie

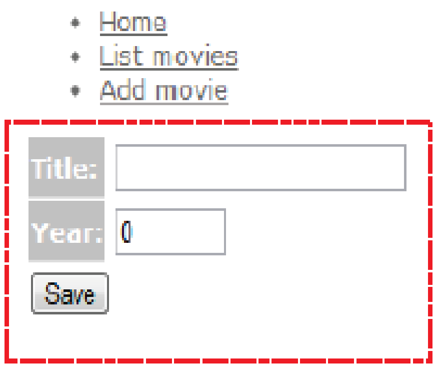
```
public class MovieCollection {
    private List<Movie> movies;
    public MovieCollection() {
        this.movies = new LinkedList<Movie>();
    }
    public List<Movie> getMovies() {
        return this.movies;
    }
    public void addMovie(Movie aMovie) {
        this.movies.add(aMovie);
    }
    public void deleteMovie(Movie aMovie) {
        this.movies.remove(aMovie);
    }
```

```
public void updateMovie(Movie aMovie) {
    try {
        String query = "UPDATE MOVIE SET TITLE=?, YR=? WHERE (ID=?)";
        PreparedStatement statement = this.db.prepareStatement(query);
        statement.setString(1, aMovie.getTitle());
        statement.setInt(2, aMovie.getYear());
        statement.setInt(3, aMovie.getId());
        statement.executeUpdate();
    } catch (SQLException ex) {
        throw new UnsupportedOperationException(ex.getMessage());
    }
}
```

★
Eskiden MovieEditForm da sadece yeni film ekliyorduk.
Varolan filmlerde update yapılmalı,
yeni film ekleme kısmı da kalmalı => sayfaya parametre gönder !

- Eskiden MovieEditForm da sadece yeni film ekliyorduk.
  Varolan filmlerde update yapılmalı,
  yeni film ekleme kısmı da kalmalı => sayfaya parametre gönder !

Add movies link cliked:

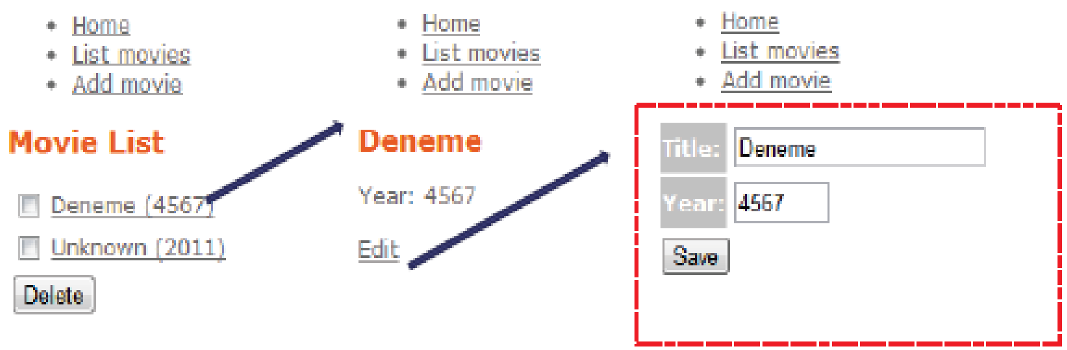


- Home
- List movies
- Add movie

Title:	
Year:	0

Save

List movies link cliked:



- Home
- List movies
- Add movie

Movie List

- [ ] Deneme (4567)
- [ ] Unknown (2011)

Delete

- Home
- List movies
- Add movie

Deneme

Year: 4567

Edit

- Home
- List movies
- Add movie

Title:	Deneme
Year:	4567

Save

★ HTML Changes

1. Change behavior of MoviePage : Add Edit link

2. Change behavior of «MovieEditPage»

HTML:

```
<p><a href="#“
wicket:id="edit_link">Edit</a>
</p>
```

JAVA:

```
public class MovieDisplayPage extends BasePage {
   private Movie _movie;

   public MovieDisplayPage(Movie movie) {
      this._movie = movie;

      this.add(new Label("title", movie.getTitle()));
      this.add(new Label("year", movie.getYear().toString()));

      Link editLink = new Link("edit_link") {
         @Override
         public void onClick() {
            MovieDisplayPage parent = (MovieDisplayPage) this.getParent();
            this.setResponsePage(new MovieEditPage(parent.getMovie(), false));
         }
      };
      this.add(editLink);
   }

   public Movie getMovie() {
      return this._movie;
   }
}
```

=Not a new movie

★ HTML Changes

1. Change behavior of MoviePage : Add Edit link
2. Change behavior of «MovieEditPage»

HTML : same ...

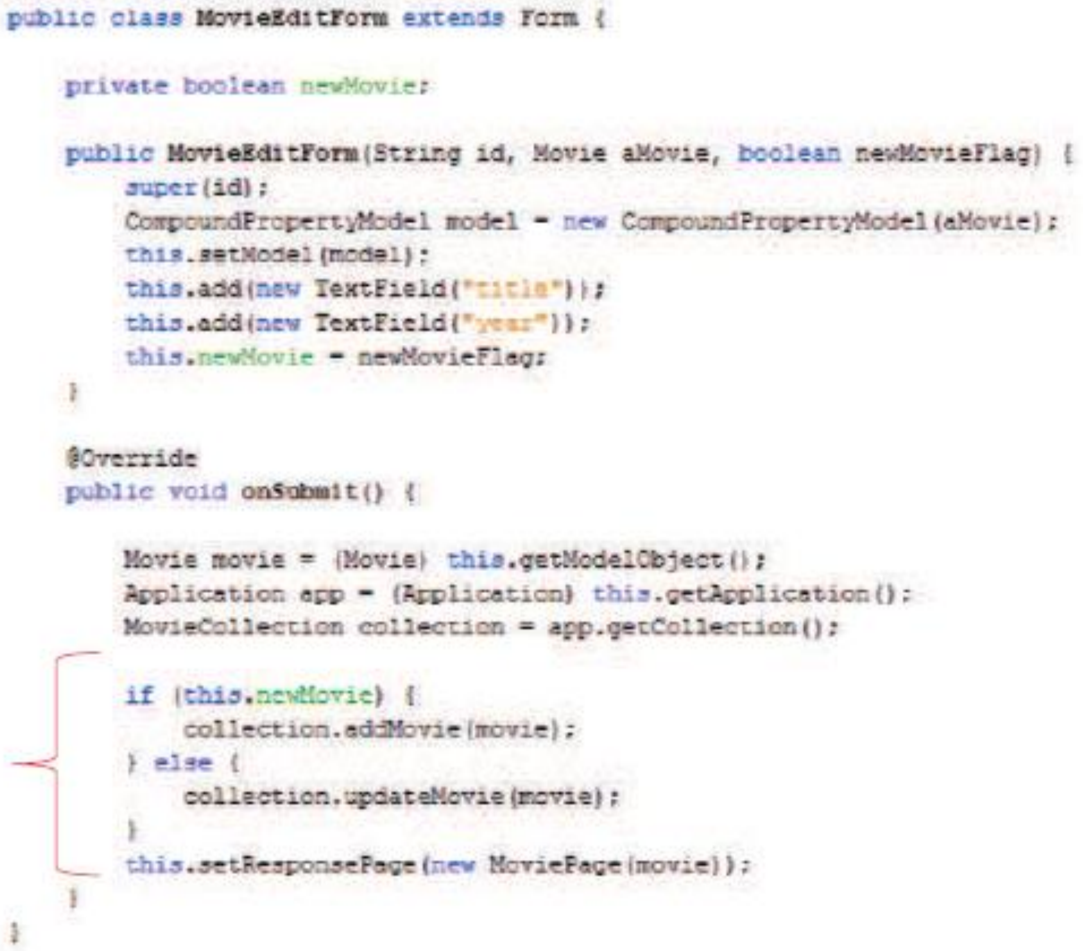
JAVA

```
public final class MovieEditPage extends BasePage {

    public MovieEditPage(Movie aMovie) {
        this.add(new MovieEditForm("movie_edit", aMovie,true));
    }

    public MovieEditPage(Movie aMovie, boolean newMovieFlag) {
        this.add(new MovieEditForm("movie_edit", aMovie, newMovieFlag));
    }
}
```

send parameter to Form Handler ..

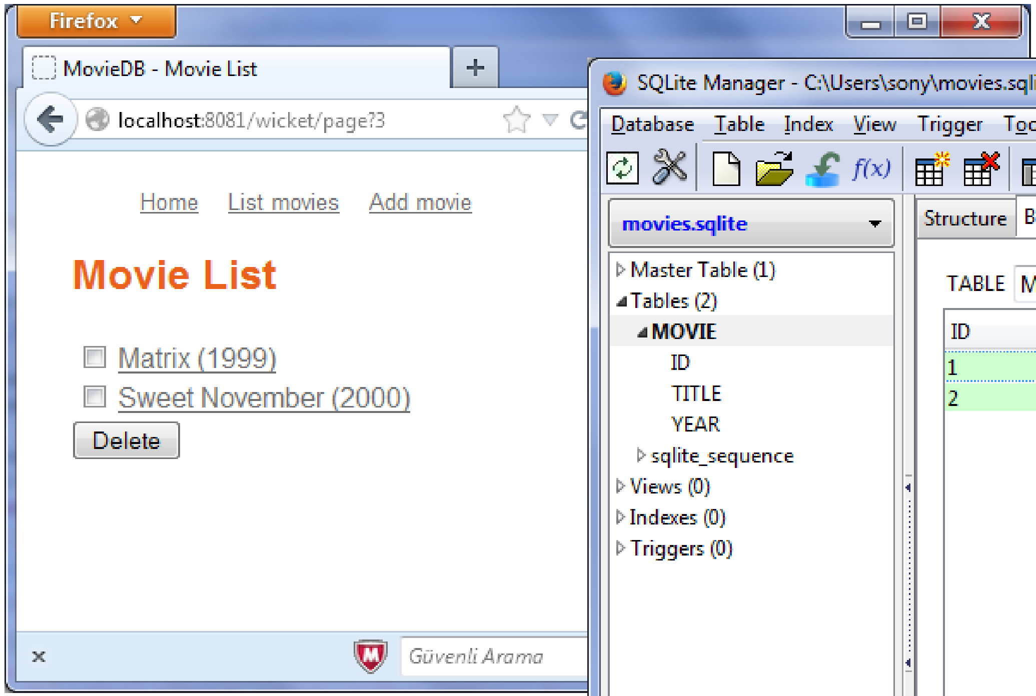
Note: take a look: it is not a PageParameters

★ Change behavior of «MovieEditForm»



```java
public class MovieEditForm extends Form {

    private boolean newMovie;

    public MovieEditForm(String id, Movie aMovie, boolean newMovieFlag) {
        super(id);
        CompoundPropertyModel model = new CompoundPropertyModel(aMovie);
        this.setModel(model);
        this.add(new TextField("title"));
        this.add(new TextField("year"));
        this.newMovie = newMovieFlag;
    }

    @Override
    public void onSubmit() {

        Movie movie = (Movie) this.getModelObject();
        Application app = (Application) this.getApplication();
        MovieCollection collection = app.getCollection();

        if (this.newMovie) {
            collection.addMovie(movie);
        } else {
            collection.updateMovie(movie);
        }
        this.setResponsePage(new MoviePage(movie));
    }
}
```

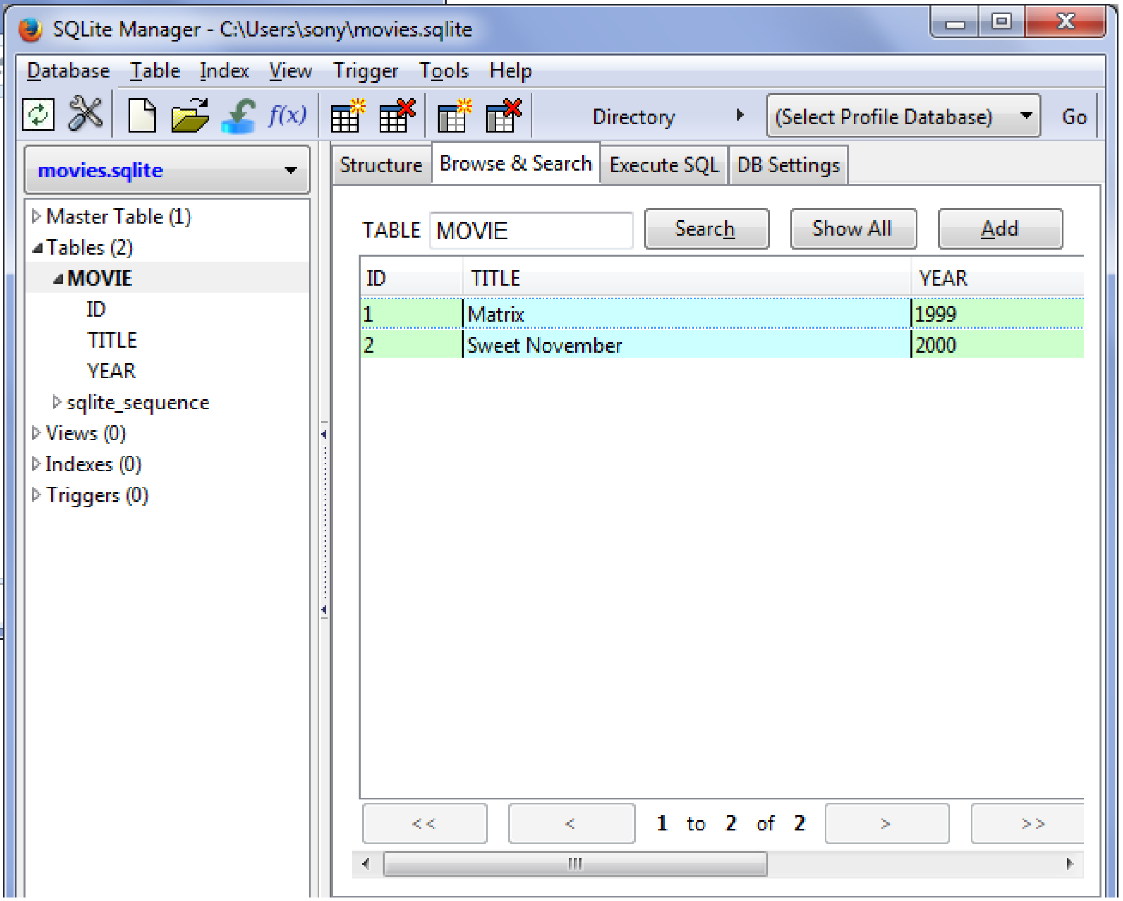
Sample run and DB check



Movie List

- [ ] Matrix (1999)
- [ ] Sweet November (2000)

Delete



ID	TITLE	YEAR
1	Matrix	1999
2	Sweet November	2000

MySQL Equivalent

- uses MySQL => add MySQL JDBC driver
- Download from: http://www.mysql.com/downloads/connector/j/
- Add jar file to your project's library
- Create data base movies.db in home directory and create a table

```
CREATE TABLE MOVIE (ID INTEGER PRIMARY KEY, TITLE VARCHAR(80), YR INTEGER)
```

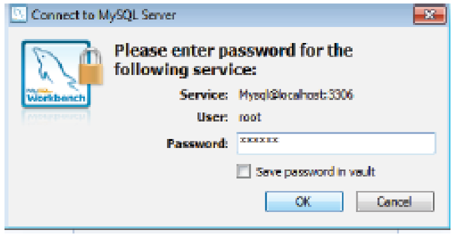
(Note: SQLite was automatically incrementing but MySQL command should aslso include AUTO _INCREMENT for ID field. http://www.sqlite.org/faq.html#q1
)

- Can use MySQL Workbench to manage MySQL databases: http://www.mysql.com/downloads/workbench/

  (do not forget to run WB as Administrator)

MySQL Equivalent

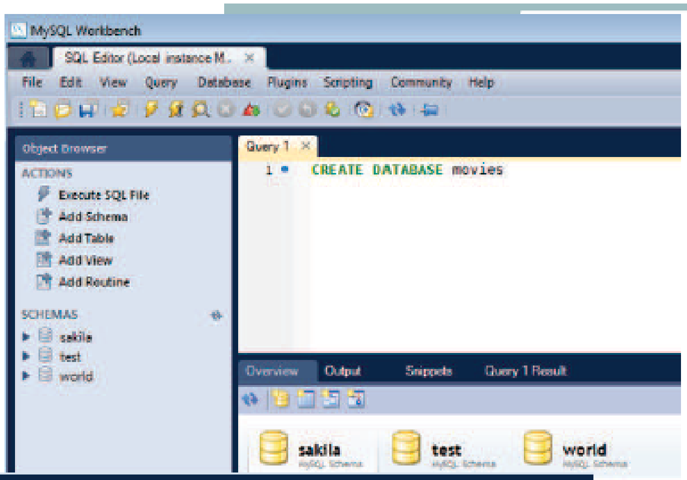
MySQL Equivalent


Connect to MySQL Server

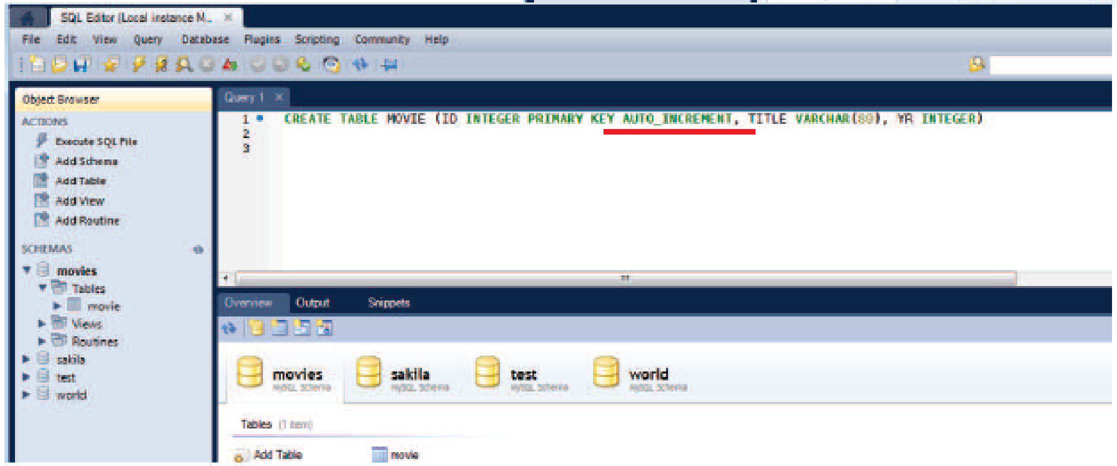
Please enter password for the following service:

Service:	Mysql@localhost:3306
User:	root
Password:	******

☐ Save password in vault



```
CREATE DATABASE movies
```



```
CREATE TABLE MOVIE (ID INTEGER PRIMARY KEY AUTO_INCREMENT, TITLE VARCHAR(80), YR INTEGER)
```

- MovieCollectionJDBC class

MovieCollectionJDBC.java

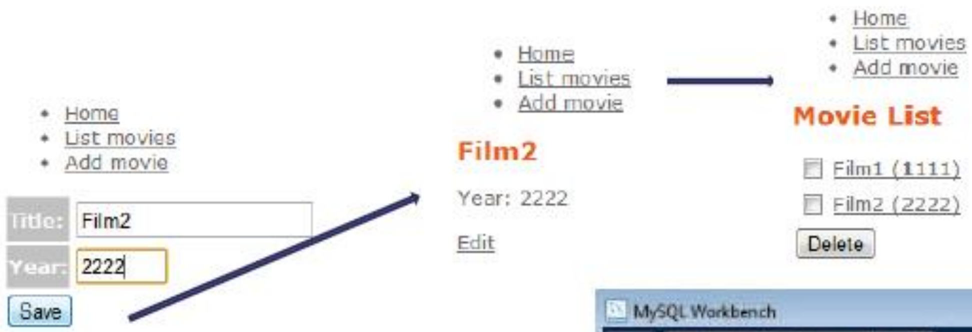
MySql jdbc driver

Connection is performed and kept in local variable **db**

```
public class MovieCollectionJDBC implements IMovieCollection {
   private Connection _db;

   public MovieCollectionJDBC(String dbFilePath) {
      try {
         Class.forName(“com.mysql.JDBC.driver");
      } catch (ClassNotFoundException e) {
         throw new UnsupportedOperationException(e.getMessage());
      }
  try {
         String userName=“root”;
         String password=“rootPass”;
         String url=“jdbc:mysql://localhost/movies”;
         this._db = DriverManager.getConnection(url,userName,password);
      } catch (SQLException ex) {
         throw new UnsupportedOperationException(ex.getMessage());
      }
   }

   public List<Movie> getMovies() { ...}

   public void addMovie(Movie movie) {  ... }

   public void deleteMovie(Movie movie) { ... }

  public void updateMovie(Movie movie) {  ...  }
}
```

MySQL Equivalent

Sampe Run and DB Check



- Home
- List movies
- Add movie

Title:	Film2
Year:	2222

Save

- Home
- List movies
- Add movie

Film2

Year: 2222

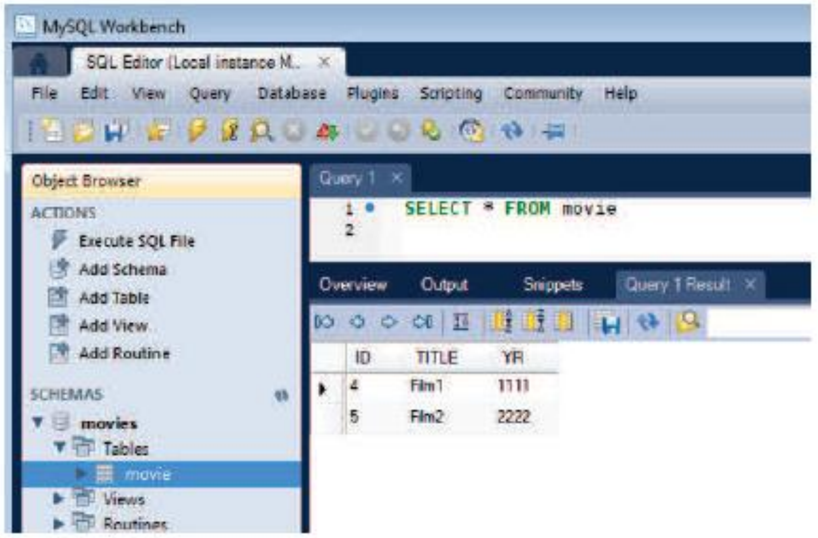
Edit

- Home
- List movies
- Add movie

Movie List

- [ ] Film1 (1111)
- [ ] Film2 (2222)

Delete



```
SELECT * FROM movie
```

ID	TITLE	YR
4	Film1	1111
5	Film2	2222

Note: snapshot is taken after deleting first 3 movies..