TRILIX

We listen, understand and deliver

JavaCro'17

SQL and Java in 21st century

Luka Banožić

SQL and Java

- writing SQL in Java has never been easy
- lot of time spent on solving SQL/Java infrastructure problems

Introducing...

jOOQ

„The easiest way to write SQL in Java”

www.jooq.org

jOOQ

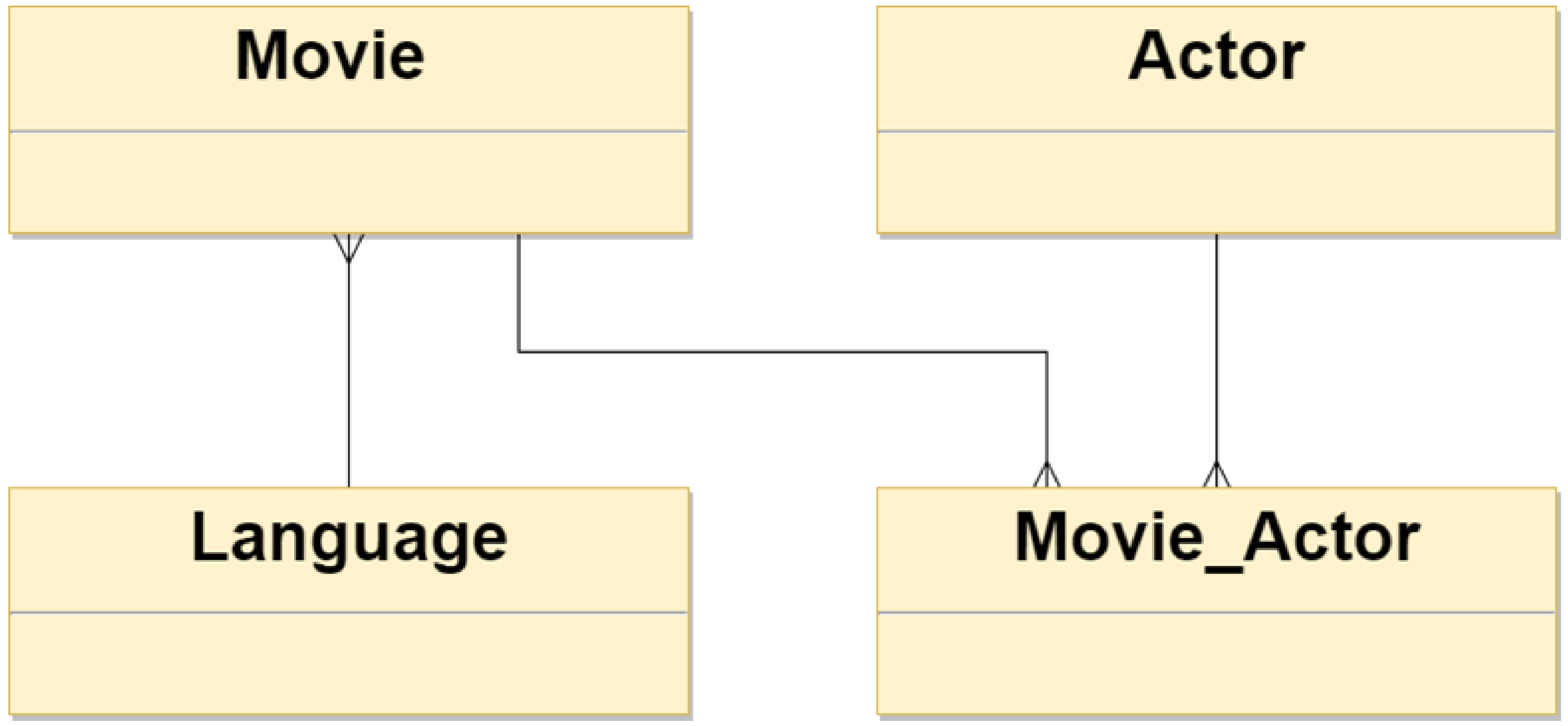
- generates Java code from database
- typesafe SQL in Java
- fluent API
- pricing:
  - free for open source databases
  - not free for commercial databases

jOOQ

- You can use jOOQ for:
  - code generation
  - SQL building
  - SQL execution

Movies database

Database model



Setup
(using Spring Tool Suite)

Domain classes

```java
public class Movie {

   private Integer id;
   private String title;
   private Short year;
   private String plot;
   private Short duration;
   private BigDecimal imdbRating;
   private Integer boxOffice;
   private Integer languageId;

   //getters and setters omitted
}
```

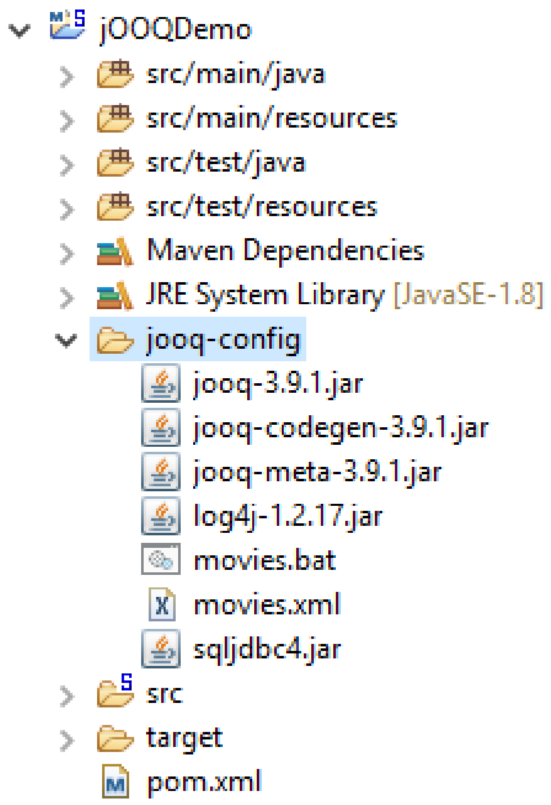
```java
public class Language {

   private Integer id;
   private String name;

   //getters and setters omitted
}
```

```java
public class Actor {

   private Integer id;
   private String firstName;
   private String lastName;
   private Date dateOfBirth;

   //getters and setters omitted
}
```

pom.xml

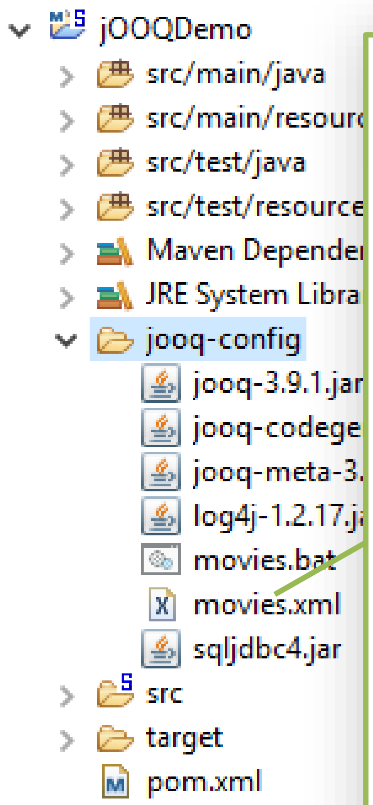
```
<!-- jOOQ -->
<dependency>
   <groupId>org.jooq.trial</groupId>
   <artifactId>jooq</artifactId>
   <version>3.9.1</version>
</dependency>
<dependency>
   <groupId>org.jooq.trial</groupId>
   <artifactId>jooq-meta</artifactId>
   <version>3.9.1</version>
</dependency>
<dependency>
   <groupId>org.jooq.trial</groupId>
   <artifactId>jooq-codegen</artifactId>
   <version>3.9.1</version>
</dependency>
```

jOOQ config



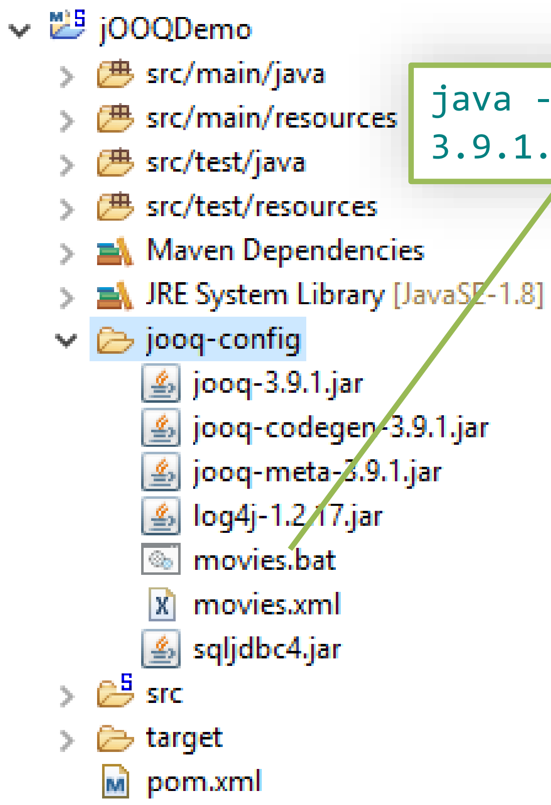
```
jOOQDemo
  src/main/java
  src/main/resources
  src/test/java
  src/test/resources
  Maven Dependencies
  JRE System Library [JavaSE-1.8]
  jooq-config
    jooq-3.9.1.jar
    jooq-codegen-3.9.1.jar
    jooq-meta-3.9.1.jar
    log4j-1.2.17.jar
    movies.bat
    movies.xml
    sqljdbc4.jar
  src
  target
  pom.xml
```

jOOQ config - movies.xml



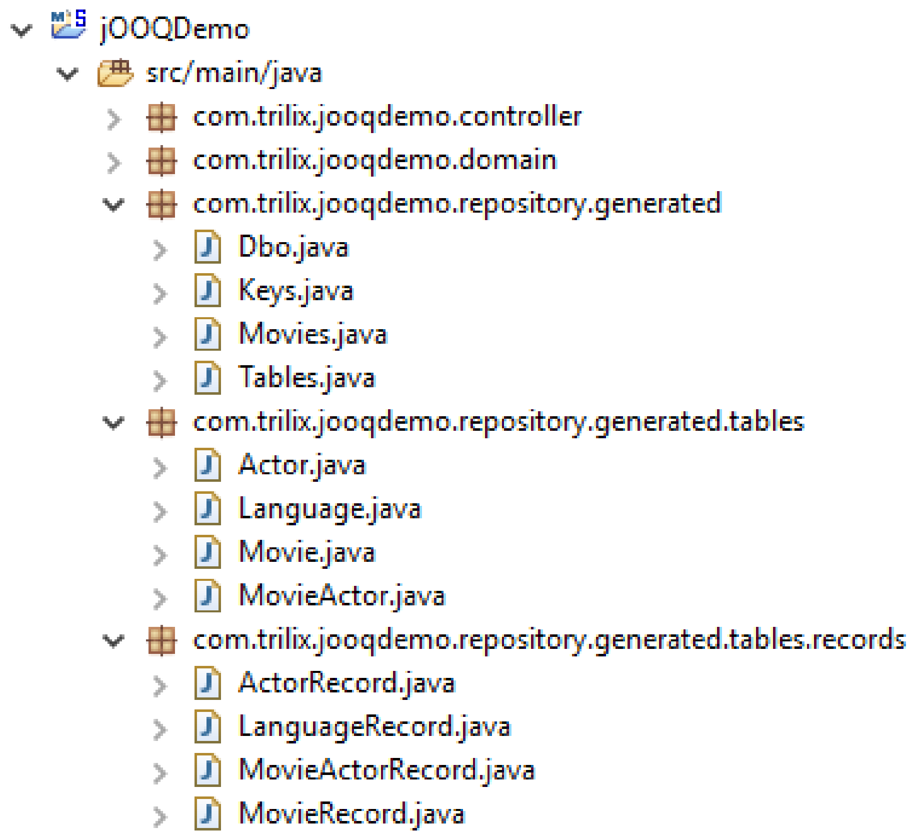
```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE xml>
<configuration>
   <jdbc>
      <driver>com.microsoft.sqlserver.jdbc.SQLServerDriver</driver>
      <url>jdbc:sqlserver://localhost\MSSQLSERVER2016;
            integratedSecurity=true;
            databaseName=Movies</url>
   </jdbc>
   <generator>
      <database>
         <name>org.jooq.util.sqlserver.SQLServerDatabase</name>
         <includes>.*</includes>
         <excludes></excludes>
         <inputCatalog>Movies</inputCatalog>
         <inputSchema>dbo</inputSchema>
      </database>

      <target>
         <packageName>com.trilix.jooqdemo.repository.generated</packageName>
         <directory>../src/main/java</directory>
      </target>
   </generator>
</configuration>
```

jOOQ config - movies.bat



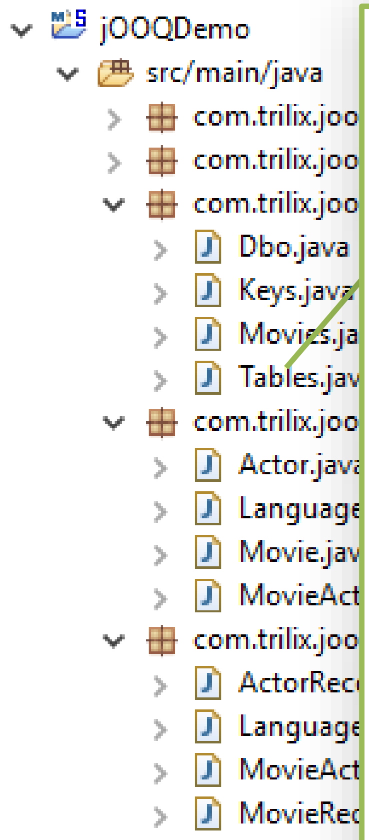
```
java -classpath jooq-3.9.1.jar;jooq-meta-3.9.1.jar;jooq-codegen-
3.9.1.jar;sqljdbc4.jar;. org.jooq.util.GenerationTool movies.xml
```

jOOQ config - generated classes



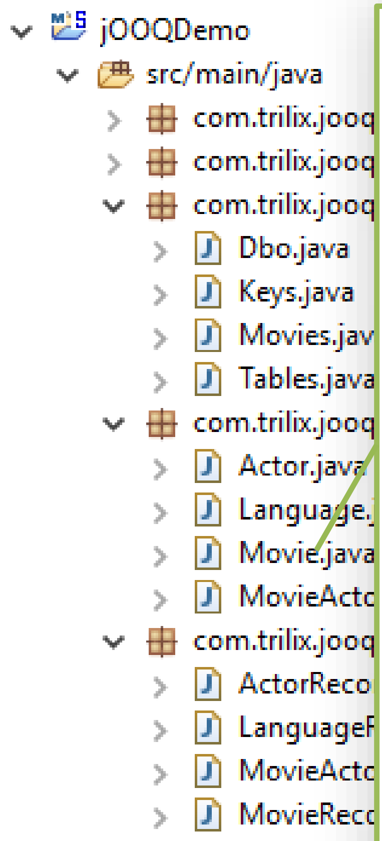
```
jOOQDemo
  src/main/java
    com.trilix.jooqdemo.controller
    com.trilix.jooqdemo.domain
    com.trilix.jooqdemo.repository.generated
      Dbo.java
      Keys.java
      Movies.java
      Tables.java
    com.trilix.jooqdemo.repository.generated.tables
      Actor.java
      Language.java
      Movie.java
      MovieActor.java
    com.trilix.jooqdemo.repository.generated.tables.records
      ActorRecord.java
      LanguageRecord.java
      MovieActorRecord.java
      MovieRecord.java
```

jOOQ config - generated classes - Tables



```
...
/**
 * The table <code>Movies.dbo.Actor</code>.
 */
public static final Actor ACTOR =
       com.trilix.jooqdemo.repository.generated.tables.Actor.ACTOR;

/**
 * The table <code>Movies.dbo.Language</code>.
 */
public static final Language LANGUAGE =
       com.trilix.jooqdemo.repository.generated.tables.Language.LANGUAGE;

/**
 * The table <code>Movies.dbo.Movie</code>.
 */
public static final Movie MOVIE =
       com.trilix.jooqdemo.repository.generated.tables.Movie.MOVIE;

/**
 * The table <code>Movies.dbo.Movie_Actor</code>.
 */
public static final MovieActor MOVIE_ACTOR =
       com.trilix.jooqdemo.repository.generated.tables.MovieActor.MOVIE_ACTOR;
...
```

jOOQ config - generated classes - Movie



```
...
/**
* The column <code>Movies.dbo.Movie.id</code>.
*/
public final TableField<MovieRecord, Integer> ID = createField("id",
   org.jooq.impl.SQLDataType.INTEGER.nullable(false), this, "");

/**
* The column <code>Movies.dbo.Movie.title</code>.
*/
public final TableField<MovieRecord, String> TITLE = createField("title",
   org.jooq.impl.SQLDataType.NVARCHAR.length(50).nullable(false), this, "");

/**
 The column <code>Movies.dbo.Movie.year</code>.
*/
public final TableField<MovieRecord, Short> YEAR = createField("year",
   org.jooq.impl.SQLDataType.SMALLINT.nullable(false), this, "");

/**
* The column <code>Movies.dbo.Movie.plot</code>.
*/
public final TableField<MovieRecord, String> PLOT = createField("plot",
   org.jooq.impl.SQLDataType.VARCHAR.length(400), this, "");
...
```

Basic CRUD

First things first - DSLContext bean

```
<bean id="dslContext" class="org.jooq.impl.DefaultDSLContext">
   <constructor-arg ref="dataSource" />
   <constructor-arg type="org.jooq.SQLDialect">
      <value>SQLSERVER</value>
   </constructor-arg>
</bean>
```

First things first - import and DSLContext

```
import static org.jooq.impl.DSL.*;
import static com.trilix.jooqdemo.repository.generated.Tables.*;

...

@Autowired
private DSLContext create;
```

Get all movies

SQL:

```
SELECT  id,
        title,
        [year],
        plot,
        duration,
        imdb_rating,
        box_office,
        language_id
FROM    dbo.Movie;
```

jOOQ:

```
create.select (MOVIE.ID,
               MOVIE.TITLE,
               MOVIE.YEAR,
               MOVIE.PLOT,
               MOVIE.DURATION,
               MOVIE.IMDB_RATING,
               MOVIE.BOX_OFFICE,
               MOVIE.LANGUAGE_ID)
      .from   (MOVIE)
```

Get all movies

SQL:

```
SELECT   id,
         title,
         [year],
         plot,
         duration,
         imdb_rating,
         box_office,
         language_id
FROM     dbo.Movie;
```

jOOQ:

```
List<Movie> movies = create.select (MOVIE.ID,
                                    MOVIE.TITLE,
                                    MOVIE.YEAR,
                                    MOVIE.PLOT,
                                    MOVIE.DURATION,
                                    MOVIE.IMDB_RATING,
                                    MOVIE.BOX_OFFICE,
                                    MOVIE.LANGUAGE_ID)
                           .from   (MOVIE)
                           .fetchInto(Movie.class);
```

Get movie by id

SQL:

```
SELECT  id,
        title,
        [year],
        plot,
        duration,
        imdb_rating,
        box_office,
        language_id
FROM    dbo.Movie
WHERE id = 3;
```

jOOQ:

```
create.select (MOVIE.ID,
               MOVIE.TITLE,
               MOVIE.YEAR,
               MOVIE.PLOT,
               MOVIE.DURATION,
               MOVIE.IMDB_RATING,
               MOVIE.BOX_OFFICE,
               MOVIE.LANGUAGE_ID)
      .from    (MOVIE)
      .where   (MOVIE.ID.eq(3))
```

Get movie by id

SQL:

```
SELECT  id,
        title,
        [year],
        plot,
        duration,
        imdb_rating,
        box_office,
        language_id
FROM    dbo.Movie
WHERE id = 3;
```

jOOQ:

```
Movie movie = create.select (MOVIE.ID,
                             MOVIE.TITLE,
                             MOVIE.YEAR,
                             MOVIE.PLOT,
                             MOVIE.DURATION,
                             MOVIE.IMDB_RATING,
                             MOVIE.BOX_OFFICE,
                             MOVIE.LANGUAGE_ID)
                    .from   (MOVIE)
                    .where  (MOVIE.ID.eq(3))
                    .fetchOneInto(Movie.class);
```

Create new movie

SQL:

```
INSERT  INTO dbo.Movie
        (
         title,
         [year],
         plot,
         duration,
         imdb_rating,
         box_office,
         language_id
        )
VALUES  (
         @title,
         @year,
         @plot,
         @duration,
         @imdb_rating,
         @box_office,
         @language_id
        );
```

jOOQ:

```
create.insertInto(MOVIE,
         MOVIE.TITLE,
         MOVIE.YEAR,
         MOVIE.PLOT,
         MOVIE.DURATION,
         MOVIE.IMDB_RATING,
         MOVIE.BOX_OFFICE,
         MOVIE.LANGUAGE_ID
        )
      .values
        (
         movie.getTitle(),
         movie.getYear(),
         movie.getPlot(),
         movie.getDuration(),
         movie.getImdbRating(),
         movie.getBoxOffice(),
         movie.getLanguageId()
        )
      .execute();
```

Create new movie

SQL:

```
INSERT  INTO dbo.Movie
        (
         title,
         [year],
         plot,
         duration,
         imdb_rating,
         box_office,
         language_id
        )
VALUES  (
         @title,
         @year,
         @plot,
         @duration,
         @imdb_rating,
         @box_office,
         @language_id
        );
```

jOOQ:

```
create.insertInto(MOVIE,
         MOVIE.TITLE,
         MOVIE.YEAR,
         MOVIE.PLOT,
         MOVIE.DURATION,
         MOVIE.IMDB_RATING,
         MOVIE.BOX_OFFICE,
         MOVIE.LANGUAGE_ID
        )
      .values
        (
         movie.getTitle(),
         movie.getYear(),
         movie.getPlot(),
         movie.getDuration(),
         movie.getBoxOffice(),
         movie.getImdbRating(),
         movie.getLanguageId()
        )
      .execute();
```

TRILIX

We listen, understand and deliver

Create new movie

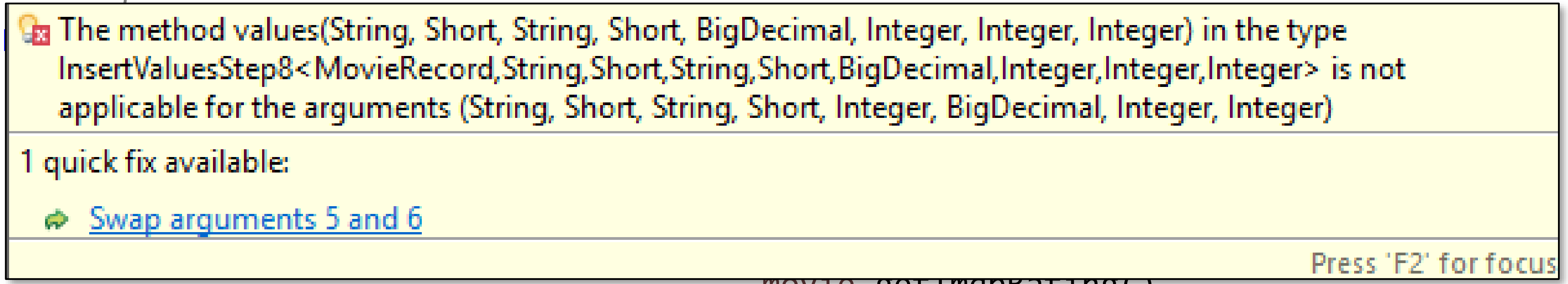
SQL:

```
INSERT  INTO dbo.Movie
        (
         title,
         [year],
         plot,
         duration,
         imdb_rating,
         box_office,
         language_id
        )
VA
```

```
         @box_office,
         @language_id
        );
```

jOOQ:

```
create.insertInto(MOVIE,
         MOVIE.TITLE,
         MOVIE.YEAR,
         MOVIE.PLOT,
         MOVIE.DURATION,
         MOVIE.IMDB_RATING,
         MOVIE.BOX_OFFICE,
         MOVIE.LANGUAGE_ID
        )
      .values
```



The method values(String, Short, String, Short, BigDecimal, Integer, Integer, Integer) in the type InsertValuesStep8<MovieRecord,String,Short,String,Short,BigDecimal,Integer,Integer,Integer> is not applicable for the arguments (String, Short, String, Short, Integer, BigDecimal, Integer, Integer)

1 quick fix available:

Swap arguments 5 and 6

Press 'F2' for focus

```
         movie.getImdbRating(),
         movie.getLanguageId()
        )
      .execute();
```

Update movie

SQL:

```
UPDATE   dbo.Movie
SET      title = @title,
         [year] = @year,
         plot = @plot,
         duration = @duration,
         imdb_rating = @imdb_rating,
         box_office = @box_office,
         language_id = @language_id
WHERE    id = @id;
```

jOOQ:

```
create.update(MOVIE)
      .set    (MOVIE.TITLE, movie.getTitle())
      .set    (MOVIE.YEAR, movie.getYear())
      .set    (MOVIE.PLOT, movie.getPlot())
      .set    (MOVIE.DURATION, movie.getDuration())
      .set    (MOVIE.IMDB_RATING, movie.getImdbRating())
      .set    (MOVIE.BOX_OFFICE, movie.getBoxOffice())
      .set    (MOVIE.LANGUAGE_ID, movie.getLanguageId())
      .where (MOVIE.ID.eq(movie.getId()))
      .execute();
```

Update movie

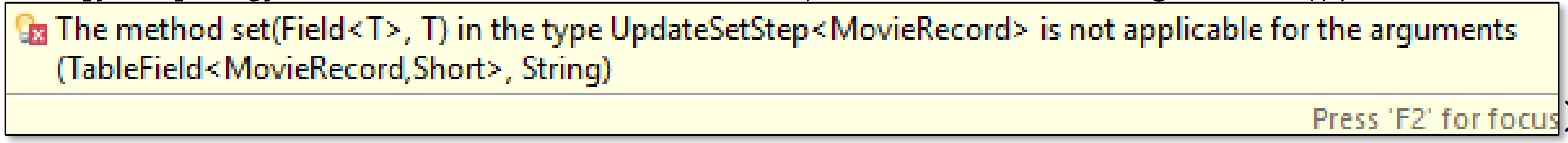
SQL:

```
UPDATE   dbo.Movie
SET      title = @title,
         [year] = @year,
```

```
         box_office = @box_office,
         language_id = @language_id
WHERE    id = @id;
```

jOOQ:

```
create.update(MOVIE)
      .set    (MOVIE.TITLE, movie.getTitle())
      .set    (MOVIE.YEAR, movie.getTitle())
```


The method set(Field<T>, T) in the type UpdateSetStep<MovieRecord> is not applicable for the arguments (TableField<MovieRecord,Short>, String)

Press 'F2' for focus

```
                                                                        )
      .set    (MOVIE.BOX_OFFICE, movie.getBoxOffice())
      .set    (MOVIE.LANGUAGE_ID, movie.getLanguageId())
      .where  (MOVIE.ID.eq(movie.getId()))
      .execute();
```

Delete movie

SQL:

```
DELETE dbo.Movie
WHERE id = 6;
```

jOOQ:

```
create.delete (MOVIE)
      .where  (MOVIE.ID.eq(6))
      .execute();
```

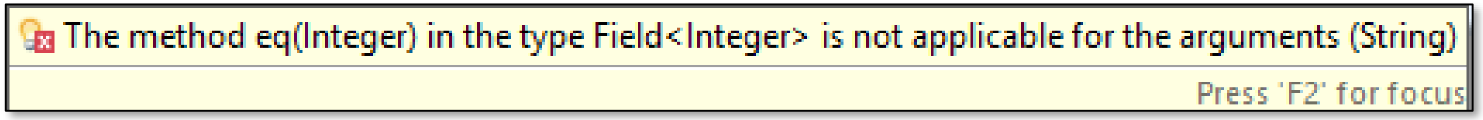
Delete movie

SQL:

```
DELETE dbo.Movie
WHERE id = 6;
```

jOOQ:

```
create.delete (MOVIE)
      .where  (MOVIE.ID.eq("6"))
```



The method eq(Integer) in the type Field<Integer> is not applicable for the arguments (String)

Press 'F2' for focus

The good stuff

Get all movies with language name - SQL

```
SELECT  m.title,
        m.[year],
        m.plot,
        m.duration,
        m.imdb_rating,
        m.box_office,
        la.[name] AS 'language',
FROM    dbo.Movie m
INNER JOIN dbo.[Language] la ON la.id = m.language_id;
```

Get all movies with language name - jOOQ

```
List<Movie> movies = create.select(MOVIE.ID,
                                   MOVIE.TITLE,
                                   MOVIE.YEAR,
                                   MOVIE.PLOT,
                                   MOVIE.DURATION,
                                   MOVIE.IMDB_RATING,
                                   MOVIE.BOX_OFFICE,
                                   LANGUAGE.NAME.as("language"))
                           .from  (MOVIE)
                           .innerJoin(LANGUAGE).on(LANGUAGE.ID.eq(MOVIE.LANGUAGE_ID))
                           .fetchInto(Movie.class);
```

Get all movies with language name - jOOQ - aliased

```
com.trilix.jooqdemo.repository.generated.tables.Movie m = MOVIE.as("m");
com.trilix.jooqdemo.repository.generated.tables.Language la = LANGUAGE.as("la");

List<Movie> movies = create.select(m.ID,
                                   m.TITLE,
                                   m.YEAR,
                                   m.PLOT,
                                   m.DURATION,
                                   m.IMDB_RATING,
                                   m.BOX_OFFICE,
                                   la.NAME.as("language"))
                           .from  (m)
                           .innerJoin(la).on(la.ID.eq(m.LANGUAGE_ID))
                           .fetchInto(Movie.class);
```

Get movie average IMDB rating

SQL:

```sql
SELECT   AVG(imdb_rating)
FROM     dbo.Movie;
```

jOOQ:

```java
BigDecimal averageImdbRating =
    create.select (avg(MOVIE.IMDB_RATING))
            .from   (MOVIE)
            .fetchOneInto(BigDecimal.class);
```

Get highest grossing movie

```
com.trilix.jooqdemo.repository.generated.tables.Movie m = MOVIE.as("m");

Movie movie = create.select (m.ID,
                             m.TITLE,
                             m.YEAR,
                             m.PLOT,
                             m.DURATION,
                             m.IMDB_RATING,
                             m.BOX_OFFICE,
                             m.LANGUAGE_ID)
                     .from   (m)
                     .orderBy(m.BOX_OFFICE.desc())
                     .limit(1)
                     .fetchOneInto(Movie.class);
```

Get actors with more than one movie appearance

```
com.trilix.jooqdemo.repository.generated.tables.Actor a = ACTOR.as("a");
com.trilix.jooqdemo.repository.generated.tables.MovieActor ma = MOVIE_ACTOR.as("ma");

List<Actor> actor = create.select(a.ID,
                                  a.FIRST_NAME,
                                  a.LAST_NAME,
                                  a.DATE_OF_BIRTH,
                                  count().as("movieAppearances"))
                          .from  (a)
                          .innerJoin(ma).on(a.ID.eq(ma.ACTOR_ID))
                          .groupBy(a.ID,
                                   a.FIRST_NAME,
                                   a.LAST_NAME,
                                   a.DATE_OF_BIRTH)
                          .having(count().gt(1))
                          .fetchInto(Actor.class);
```

Get actors that are not assigned to any movie (NOT EXISTS)

```
com.trilix.jooqdemo.repository.generated.tables.Actor a = ACTOR.as("a");
com.trilix.jooqdemo.repository.generated.tables.MovieActor ma = MOVIE_ACTOR.as("ma");

List<Actor> actors = create.select(a.ID,
                                   a.FIRST_NAME,
                                   a.LAST_NAME)
                            .from  (a)
                            .where(notExists(create.select(ma.ID)
                                                   .from  (ma)
                                                   .where (ma.ACTOR_ID.eq(a.ID))))
                            .fetchInto(Actor.class);
```

Get actors that are not assigned to any movie (LEFT JOIN)

```
com.trilix.jooqdemo.repository.generated.tables.Actor a = ACTOR.as("a");
com.trilix.jooqdemo.repository.generated.tables.MovieActor ma = MOVIE_ACTOR.as("ma");

List<Actor> actors = create.select(a.ID,
                                   a.FIRST_NAME,
                                   a.LAST_NAME)
                            .from  (a)
                            .leftJoin(ma).on(a.ID.eq(ma.ACTOR_ID))
                            .where (ma.ACTOR_ID.isNull())
                            .fetchInto(Actor.class);
```

Get actors that are not assigned to any movie (dynamic SQL)

```
com.trilix.jooqdemo.repository.generated.tables.Actor a = ACTOR.as("a");
com.trilix.jooqdemo.repository.generated.tables.MovieActor ma = MOVIE_ACTOR.as("ma");

Select<?> notExists = create.select(ma.ID)
                            .from  (ma)
                            .where (ma.ACTOR_ID.eq(a.ID));

List<Actor> actors = create.select(a.ID,
                                   a.FIRST_NAME,
                                   a.LAST_NAME)
                           .from  (a)
                           .where (notExists(notExists))
                           .fetchInto(Actor.class);
```

Get all movies with additional duration info

```
com.trilix.jooqdemo.repository.generated.tables.Movie m = MOVIE.as("m");

List<Movie> movies =
    create.select(m.ID,
                  m.TITLE,
                  m.YEAR,
                  m.PLOT,
                  m.DURATION,
                  m.IMDB_RATING,
                  m.BOX_OFFICE,
                  DSL.when(m.DURATION.between((short) 0, (short)99), "short")
                      .when(m.DURATION.between((short)100, (short)140), "average")
                      .otherwise("long").as("durationCategory"))
         .from  (m)
         .fetchInto(Movie.class);
```

Get all movies with their languages

```
com.trilix.jooqdemo.repository.generated.tables.Movie m = MOVIE.as("m");
com.trilix.jooqdemo.repository.generated.tables.Language la = LANGUAGE.as("la");

create.select(m.ID,
              m.TITLE,
              m.YEAR,
              la.ID,
              la.NAME)
      .from   (m)
      .innerJoin(la).on(la.ID.eq(m.LANGUAGE_ID))
      .fetch();
```

Get all movies with their languages

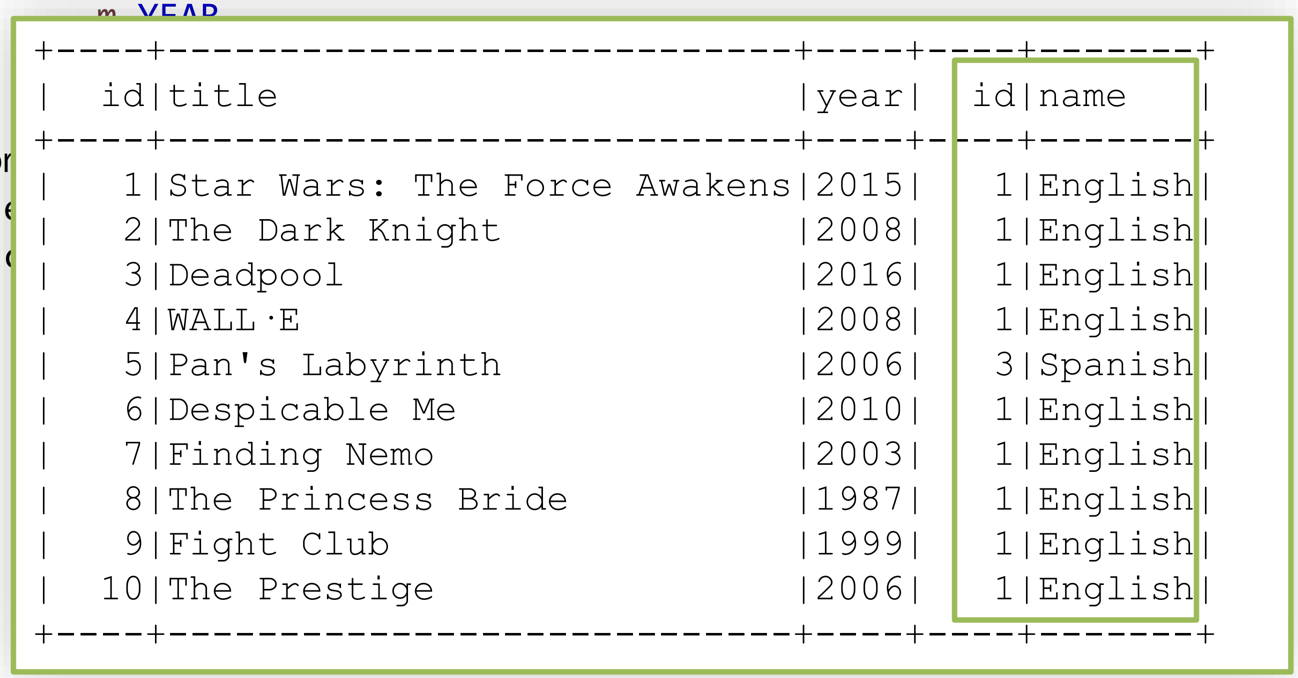
```
com.trilix.jooqdemo.repository.generated.tables.Movie m = MOVIE.as("m");
com.trilix.jooqdemo.repository.generated.tables.Language la = LANGUAGE.as("la");

create.select(m.ID,
              m.TITLE,
              m.YEAR




      .fro
      .inn
      .fet
```

```
+----+------------------------------+----+----+-------+
|  id|title                         |year|  id|name   |
+----+------------------------------+----+----+-------+
|   1|Star Wars: The Force Awakens|2015|   1|English|
|   2|The Dark Knight               |2008|   1|English|
|   3|Deadpool                      |2016|   1|English|
|   4|WALL·E                        |2008|   1|English|
|   5|Pan's Labyrinth               |2006|   3|Spanish|
|   6|Despicable Me                 |2010|   1|English|
|   7|Finding Nemo                  |2003|   1|English|
|   8|The Princess Bride            |1987|   1|English|
|   9|Fight Club                    |1999|   1|English|
|  10|The Prestige                  |2006|   1|English|
+----+------------------------------+----+----+-------+
```

We listen, understand and deliver

Get all movies with their languages

```
com.trilix.jooqdemo.repository.generated.tables.Movie m = MOVIE.as("m");
com.trilix.jooqdemo.repository.generated.tables.Language la = LANGUAGE.as("la");

create.select(m.ID,
              m.TITLE,
              m.YEAR



      .fro
      .inne
      .fetc
```



```
+----+------------------------------+----+----+--------+
|  id|title                         |year|  id|name    |
+----+------------------------------+----+----+--------+
|   1|Star Wars: The Force Awakens|2015|   1|English|
|   2|The Dark Knight               |2008|   1|English|
|   3|Deadpool                      |2016|   1|English|
|   4|WALL·E                        |2008|   1|English|
|   5|Pan's Labyrinth               |2006|   3|Spanish|
|   6|Despicable Me                 |2010|   1|English|
|   7|Finding Nemo                  |2003|   1|English|
|   8|The Princess Bride            |1987|   1|English|
|   9|Fight Club                    |1999|   1|English|
|  10|The Prestige                  |2006|   1|English|
+----+------------------------------+----+----+--------+
```

Get all movies with their languages

```
public class Movie {

   private Integer id;
   private String title;
   private Short year;
   private String plot;
   private Short duration;
   private BigDecimal imdbRating;
   private Integer boxOffice;
   private Language language;

   //getters and setters omitted
}

public class Language {

   private Integer id;
   private String name;

   //getters and setters omitted
}
```

Get all movies with their languages

```
com.trilix.jooqdemo.repository.generated.tables.Movie m = MOVIE.as("m");
com.trilix.jooqdemo.repository.generated.tables.Language la = LANGUAGE.as("la");

List<Movie> movies = new ArrayList<>();

create.select(m.ID,
              m.TITLE,
              m.YEAR,
              la.ID,
              la.NAME)
     .from   (m)
     .innerJoin(la).on(la.ID.eq(m.LANGUAGE_ID))
     .fetch()
     .forEach(record -> {

         MovieRecord mr = record.into(m);
         LanguageRecord lr = record.into(la);

         Movie movie = mr.into(Movie.class);
         Language language = lr.into(Language.class);

         movie.setLanguage(language);
         movies.add(movie);
      });
```

Get all movies and their actors

```
com.trilix.jooqdemo.repository.generated.tables.Movie m = MOVIE.as("m");
com.trilix.jooqdemo.repository.generated.tables.MovieActor ma = MOVIE_ACTOR.as("ma");
com.trilix.jooqdemo.repository.generated.tables.Actor a = ACTOR.as("a");

create.select()
       .from  (m)
       .innerJoin(ma).on(m.ID.eq(ma.MOVIE_ID))
       .innerJoin(a).on(a.ID.eq(ma.ACTOR_ID))
       .fetch();
```

Get all movies and their actors

```
co
co                                                                      a");
co

cr
```

```
+----+----------------------------+----+----+------------+--------------+
|  id|title                       |year|  id|first_name  |last_name     |
+----+----------------------------+----+----+------------+--------------+
|   1|Star Wars: The Force Awakens|2015|   1|Harrison    |Ford          |
|   1|Star Wars: The Force Awakens|2015|   2|Mark        |Hamill        |
|   1|Star Wars: The Force Awakens|2015|   3|Carrie      |Fisher        |
|   1|Star Wars: The Force Awakens|2015|   4|Adam        |Driver        |
|   1|Star Wars: The Force Awakens|2015|   5|Daisy       |Ridley        |
|   2|The Dark Knight             |2008|   6|Christian   |Bale          |
|   2|The Dark Knight             |2008|   7|Heath       |Ledger        |
|   2|The Dark Knight             |2008|   8|Michael     |Caine         |
|   2|The Dark Knight             |2008|   9|Maggie      |Gyllenhaal    |
|   2|The Dark Knight             |2008|  10|Gary        |Oldman        |
|   2|The Dark Knight             |2008|  11|Morgan      |Freeman       |
|   3|Deadpool                    |2016|  12|Ryan        |Reynolds      |
|   3|Deadpool                    |2016|  13|Morena      |Baccarin      |
|   3|Deadpool                    |2016|  14|Ed          |Skrein        |
|   4|WALL·E                      |2008|  15|Benjamin A. |Burtt         |
|   4|WALL·E                      |2008|  16|Elissa      |Knight        |
|   4|WALL·E                      |2008|  17|Jeff        |Garlin        |
|   4|WALL·E                      |2008|  18|Sigourney   |Weaver        |
| ...|...                         |... | ...|...         |...           |
+----+----------------------------+----+----+------------+--------------+
```

Get all movies and their actors

```
co
co                                                                a");
co

cr
```

```
+----+----------------------------+----+----+-----------+--------------+
|  id|title                       |year|  id|first_name |last_name     |
+----+----------------------------+----+----+-----------+--------------+
|   1|Star Wars: The Force Awakens|2015|   1|Harrison   |Ford          |
|    |                            |    |   2|Mark       |Hamill        |
|    |                            |    |   3|Carrie     |Fisher        |
|    |                            |    |   4|Adam       |Driver        |
|    |                            |    |   5|Daisy      |Ridley        |
|   2|The Dark Knight             |2008|   6|Christian  |Bale          |
|    |                            |    |   7|Heath      |Ledger        |
|    |                            |    |   8|Michael    |Caine         |
|    |                            |    |   9|Maggie     |Gyllenhaal    |
|    |                            |    |  10|Gary       |Oldman        |
|    |                            |    |  11|Morgan     |Freeman       |
|   3|Deadpool                    |2016|  12|Ryan       |Reynolds      |
|    |                            |    |  13|Morena     |Baccarin      |
|    |                            |    |  14|Ed         |Skrein        |
|   4|WALL·E                      |2008|  15|Benjamin A.|Burtt         |
|    |                            |    |  16|Elissa     |Knight        |
|    |                            |    |  17|Jeff       |Garlin        |
|    |                            |    |  18|Sigourney  |Weaver        |
| ...|...                         |... | ...|...        |...           |
+----+----------------------------+----+----+-----------+--------------+
```

Get all movies and their actors

```
public class Movie {

    private Integer id;
    private String title;
    private Short year;
    private String plot;
    private Short duration;
    private BigDecimal imdbRating;
    private Integer boxOffice;
    private Language language;
    private List<Actor> actors;

    //getters and setters omitted
}
public class Actor {

    private Integer id;
    private String firstName;
    private String lastName;
    private Date dateOfBirth;

    //getters and setters omitted
}
```

Get all movies and their actors

```
com.trilix.jooqdemo.repository.generated.tables.Movie m = MOVIE.as("m");
com.trilix.jooqdemo.repository.generated.tables.MovieActor ma = MOVIE_ACTOR.as("ma");
com.trilix.jooqdemo.repository.generated.tables.Actor a = ACTOR.as("a");

Map<Record, Result<Record>> result = create.select()
                                          .from  (m)
                                          .innerJoin(ma).on(m.ID.eq(ma.MOVIE_ID))
                                          .innerJoin(a).on(a.ID.eq(ma.ACTOR_ID))
                                          .fetch().intoGroups(MOVIE.fields());
```

```
List<Movie> movies = new ArrayList<>();

for (Entry<Record, Result<Record>> entry : result.entrySet()) {
    Record mr = entry.getKey();
    Result<Record> actors = entry.getValue();

    Movie movie = mr.into(Movie.class);
    movie.setActors(actors.into(Actor.class));
    movies.add(movie);
}
```

More jOOQ good stuff

Transactions

```
create.transaction(configuration -> {

    DSLContext dslContext = DSL.using(configuration);

    MovieRecord movieRecord = dslContext.insertInto(MOVIE,
                MOVIE.TITLE, MOVIE.YEAR, MOVIE.PLOT, MOVIE.DURATION,
                MOVIE.IMDB_RATING, MOVIE.BOX_OFFICE, MOVIE.LANGUAGE_ID)
        .values(movie.getTitle(), movie.getYear(), movie.getPlot(), movie.getDuration(),
                movie.getImdbRating(), movie.getBoxOffice(), movie.getLanguageId())
        .returning()
        .fetchOne();

    BatchBindStep batch = dslContext.batch(create.insertInto(MOVIE_ACTOR,
                    MOVIE_ACTOR.MOVIE_ID, MOVIE_ACTOR.ACTOR_ID)
            .values (Integer) null,       (Integer) null));

    Integer movieId = movieRecord.getId();

    for (Integer actorId : actorIds) {
        batch.bind(movieId, actorId);
    }

    batch.execute();
});
```

Stored procedures

```
CREATE PROCEDURE Actor_Get_By_Movie @movie_id INT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT  a.id,
            a.first_name,
            a.last_name,
            a.date_of_birth
    FROM    dbo.Actor a
    INNER JOIN dbo.Movie_Actor ma ON a.id = ma.actor_id
    WHERE   ma.movie_id = @movie_id;

END;
```

Stored procedures

```
ActorGetByMovie procedure = new ActorGetByMovie();
procedure.setMovieId(movieId);
procedure.execute(create.configuration());

Result<Record> results = procedure.getResults().get(0);
List<Actor> actors = new ArrayList<>();

for (Record r : results) {
    Actor a = new Actor();
    a.setId((Integer) r.getValue("id"));
    a.setFirstName((String) r.getValue("first_name"));
    a.setLastName((String) r.getValue("last_name"));
    a.setDateOfBirth((Date) r.getValue("date_of_birth"));
    actors.add(a);
}
```

onKey()

```
com.trilix.jooqdemo.repository.generated.tables.Movie m = MOVIE.as("m");
com.trilix.jooqdemo.repository.generated.tables.MovieActor ma = MOVIE_ACTOR.as("ma");
com.trilix.jooqdemo.repository.generated.tables.Actor a = ACTOR.as("a");
com.trilix.jooqdemo.repository.generated.tables.Language la = LANGUAGE.as("la");


create.select(m.TITLE,
              m.YEAR,
              la.NAME,
              a.FIRST_NAME,
              a.LAST_NAME)
      .from  (m)
      .innerJoin(ma).on(ma.MOVIE_ID.eq(m.ID))
      .innerJoin(a).on(a.ID.eq(ma.ACTOR_ID))
      .innerJoin(la).on(la.ID.eq(m.LANGUAGE_ID))
      .fetch();
```

onKey()

```
com.trilix.jooqdemo.repository.generated.tables.Movie m = MOVIE.as("m");
com.trilix.jooqdemo.repository.generated.tables.MovieActor ma = MOVIE_ACTOR.as("ma");
com.trilix.jooqdemo.repository.generated.tables.Actor a = ACTOR.as("a");
com.trilix.jooqdemo.repository.generated.tables.Language la = LANGUAGE.as("la");


create.select(m.TITLE,
              m.YEAR,
              la.NAME,
              a.FIRST_NAME,
              a.LAST_NAME)
      .from  (m)
      .innerJoin(ma).onKey()
      .innerJoin(a).onKey()
      .innerJoin(la).onKey()
      .fetch();
```

Conclusion

- jOOQ:
  - „The easiest way to write SQL in Java”
  - typesafe queries
  - has lot of powerful SQL stuff
  - not trying to hide SQL
  - if you know SQL, then you already know jOOQ
  - doesn’t have to be replacement for what you are already using
  - free alternatives to jOOQ for commercial databases?

    Querydsl

Questions?

Contact: Luka.Banozic@trilix.eu

Thank you!

☺

Batch

```
Integer[] actorIds = { 11, 17, 37, 31 };
Integer movieId = 9;

BatchBindStep batch =
create.batch(create.insertInto(MOVIE_ACTOR,
                                MOVIE_ACTOR.MOVIE_ID,
                                MOVIE_ACTOR.ACTOR_ID
                               )
                               .values
                               (
                                (Integer) null,
                                (Integer) null
                               ));

for (Integer actorId : actorIds) {
    batch.bind(movieId, actorId);
}

batch.execute();
```

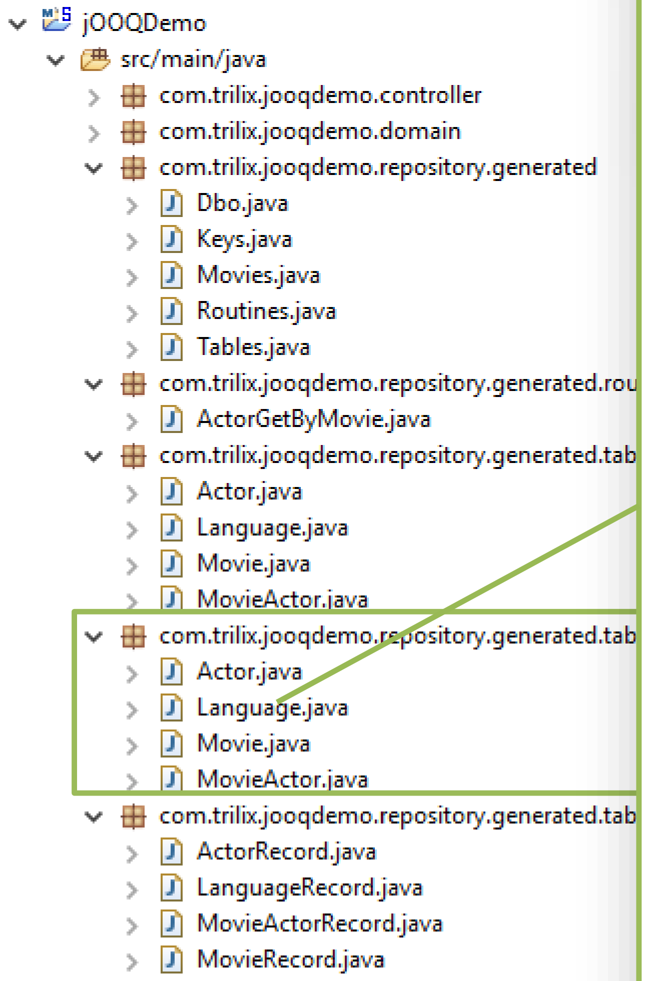
DDL (Data Definition Language)

```
create.alterTable(MOVIE)
      .alter(MOVIE.PLOT).set(SQLDataType.VARCHAR.length(500))
      .execute();
```

POJO generation

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE xml>
<configuration>
    <jdbc>
        <driver>com.microsoft.sqlserver.jdbc.SQLServerDriver</driver>
        <url>jdbc:sqlserver://localhost\MSSQLSERVER2016;
                integratedSecurity=true;
                databaseName=Movies</url>
    </jdbc>
    <generator>
        <database>
            <name>org.jooq.util.sqlserver.SQLServerDatabase</name>
            <includes>.*</includes>
            <excludes></excludes>
            <inputCatalog>Movies</inputCatalog>
            <inputSchema>dbo</inputSchema>
        </database>
      <generate>
            <pojos>true</pojos>
      </generate>
        <target>
            <packageName>com.trilix.jooqdemo.repository.generated</packageName>
            <directory>../src/main/java</directory>
        </target>
    </generator>
</configuration>
```

POJO generation



- jOOQDemo
  - src/main/java
    - com.trilix.jooqdemo.controller
    - com.trilix.jooqdemo.domain
    - com.trilix.jooqdemo.repository.generated
      - Dbo.java
      - Keys.java
      - Movies.java
      - Routines.java
      - Tables.java
    - com.trilix.jooqdemo.repository.generated.rou
      - ActorGetByMovie.java
    - com.trilix.jooqdemo.repository.generated.tab
      - Actor.java
      - Language.java
      - Movie.java
      - MovieActor.java
    - com.trilix.jooqdemo.repository.generated.tab
      - Actor.java
      - Language.java
      - Movie.java
      - MovieActor.java
    - com.trilix.jooqdemo.repository.generated.tab
      - ActorRecord.java
      - LanguageRecord.java
      - MovieActorRecord.java
      - MovieRecord.java

```
/**
 * This class is generated by jOOQ.
 */
@Generated(value = { "http://www.jooq.org", "jOOQ
version:3.9.1" }, comments = "This class is generated by
jOOQ")
@SuppressWarnings({ "all", "unchecked", "rawtypes" })
public class Language implements Serializable {

    private static final long serialVersionUID =
                                    959054493;

    private Integer id;
    private String name;

    public Language() {
    }

    public Language(Language value) {
        this.id = value.id;
        this.name = value.name;
    }

    public Language(Integer id, String name) {
        this.id = id;
        this.name = name;
    }

    //getters and setters omitted
}
```