

# Работа с базами данных в Java

Щербаков Максим

Naumen, 2020

# Цель лекции

- Научиться использовать СУБД из программы на Java

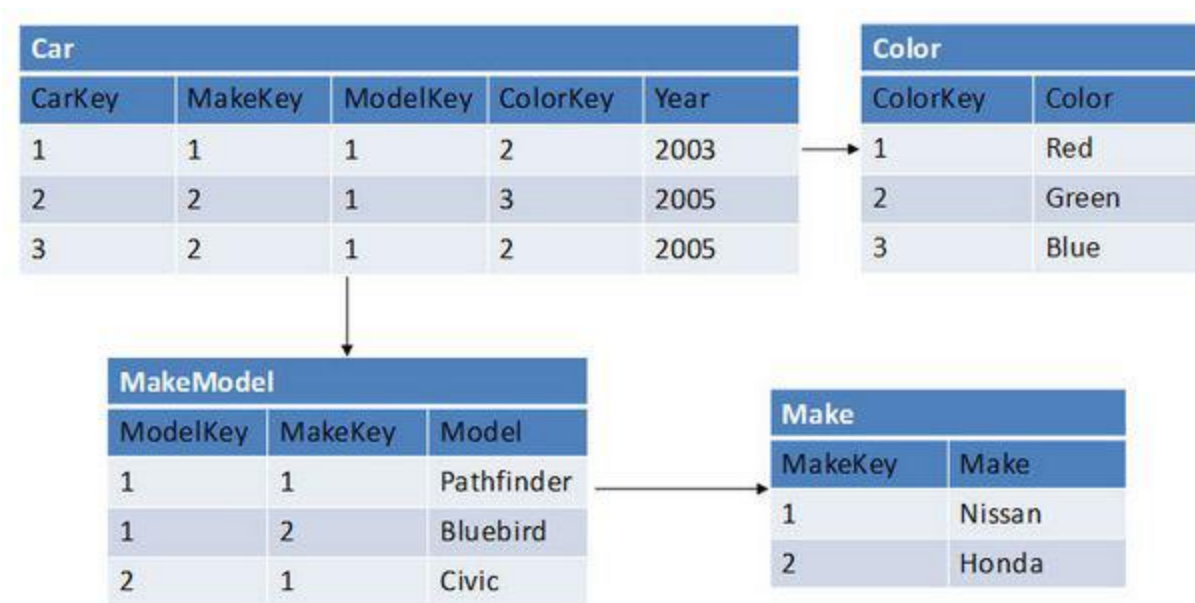
# План лекции

- Что такое СУБД?
- Начнем с создания проекта...
- Зачем нам вообще использовать СУБД?
- Использование Hibernate для работы с СУБД
- Первый блин комом или зачем нужны транзакции
- Пользователи жалуются на слишком долгую работу сайта
- Теперь все хорошо, но что делать дальше?
- Итог

# Что такое СУБД?

- СУБД - это программа, которая занимается управлением вашими данными (например, PostgreSQL, MS SQL, MySQL, OracleDB).
- БД - это ваши данные в каком-то структурированном виде (например в json'e или просто строки в текстовом файле).

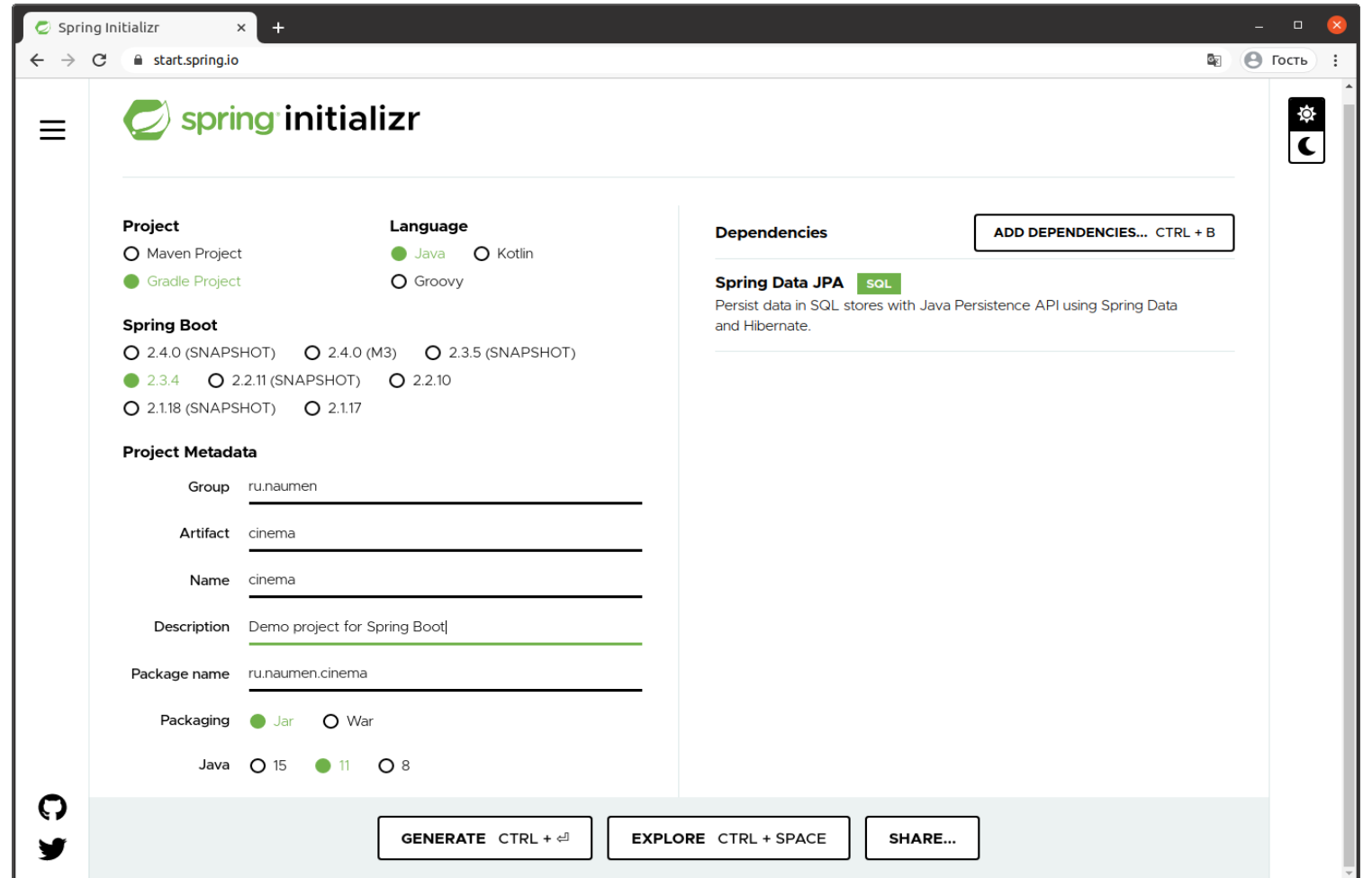
# Реляционные СУБД



Пример типичной реляционной модели данных

# Создание проекта

<https://start.spring.io>

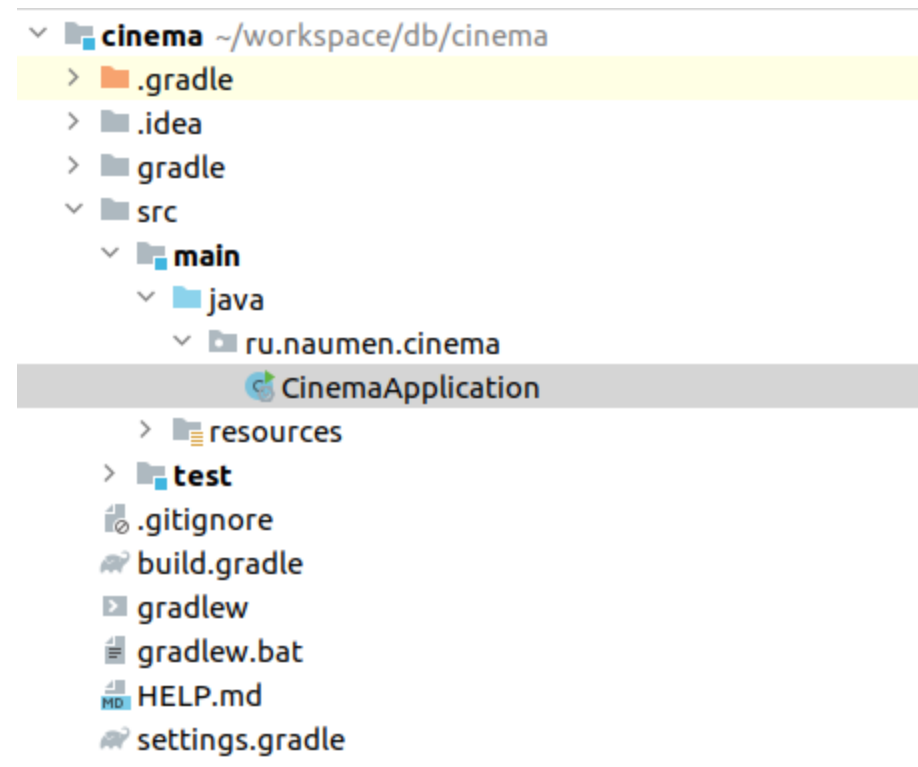


The screenshot shows the Spring Initializr web application interface. The browser address bar displays "start.spring.io". The page features a navigation menu on the left and a main content area with the following sections:

- Project:** Radio buttons for "Maven Project" and "Gradle Project" (selected).
- Language:** Radio buttons for "Java" (selected), "Kotlin", and "Groovy".
- Spring Boot:** Radio buttons for versions: "2.4.0 (SNAPSHOT)", "2.4.0 (M3)", "2.3.5 (SNAPSHOT)", "2.3.4" (selected), "2.2.11 (SNAPSHOT)", "2.2.10", "2.1.18 (SNAPSHOT)", and "2.1.17".
- Project Metadata:** Input fields for "Group" (ru.naumen), "Artifact" (cinema), "Name" (cinema), "Description" (Demo project for Spring Boot), and "Package name" (ru.naumen.cinema).
- Packaging:** Radio buttons for "Jar" (selected) and "War".
- Java:** Radio buttons for versions "15", "11" (selected), and "8".
- Dependencies:** A section with a button "ADD DEPENDENCIES... CTRL + B" and a dependency "Spring Data JPA" with a "SQL" tag.

At the bottom of the interface, there are three buttons: "GENERATE CTRL + ⌘", "EXPLORE CTRL + SPACE", and "SHARE...".

# Создание проекта



# Функции приложения

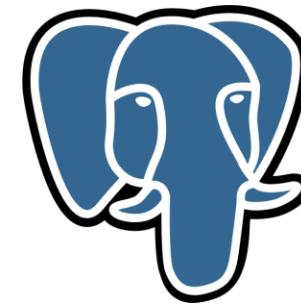
- Будем делать аналог кинопоиска (kinopoisk.ru)
- Можно добавлять, удалять и изменять фильмы из библиотеки
- Можно выставлять рейтинги фильмам



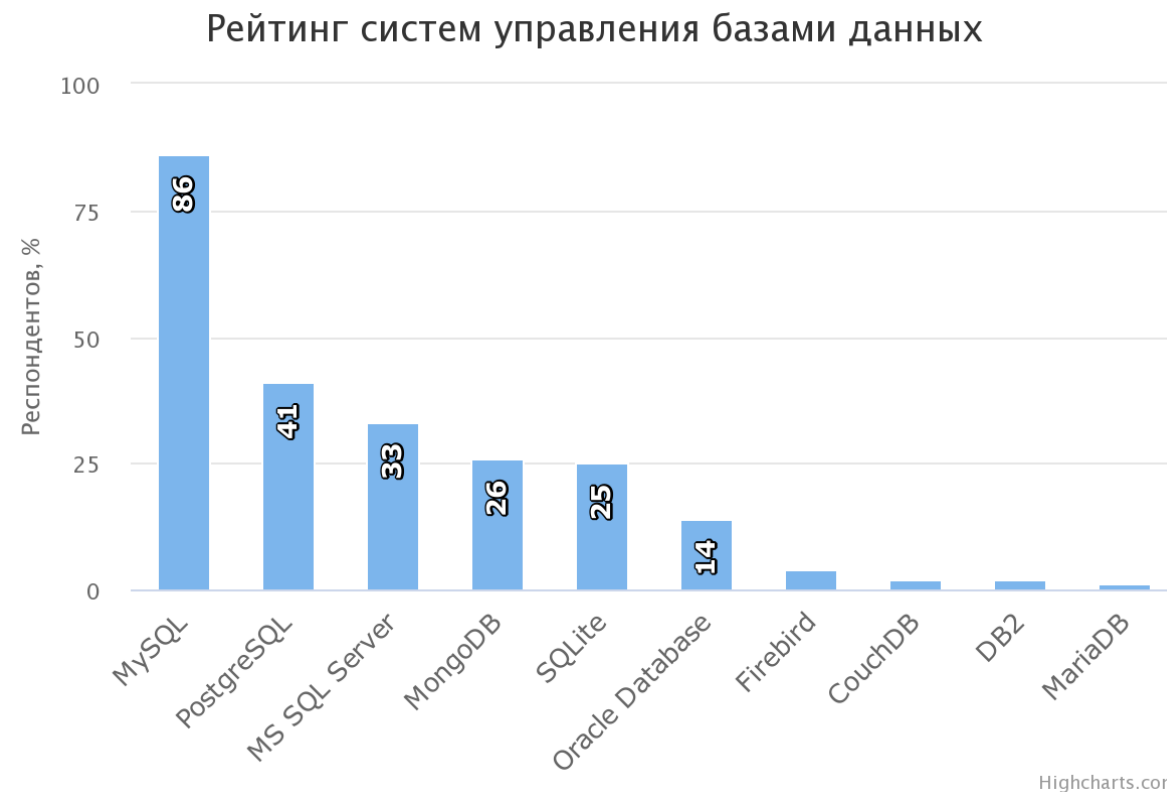
# Преимущества СУБД

- Многопользовательский доступ
- Стойкость к сбоям
- Удобный API (SQL)
- Широкие возможности для прокачки производительности
- Простота резервирования
- Возможности для поддержки согласованности данных

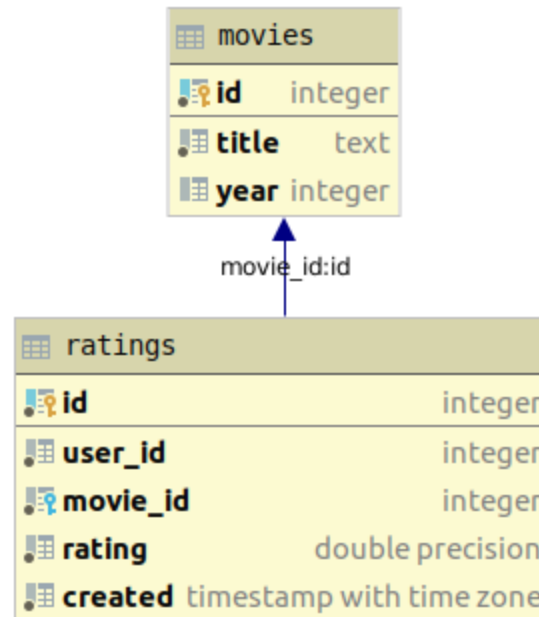
# PostgreSQL



- Open source
- Одна из самых популярных СУБД
- Самая популярная СУБД для Java
- Большое сообщество
- Понятная документация



# Схема нашей БД



# Отображение схемы в Java

```
@Entity
@Table(name = "movies")
public class Movie {

    @Id
    private int id;

    @Column(name = "title")
    private String title;

    @Column(name = "year")
    private int year;

}
```

# Отображение схемы в Java

```
@Entity
@Table(name = "ratings")
public class Rating {

    @Id
    private int id;

    @ManyToOne
    @JoinColumn(name = "movie_id")
    private Movie movie;

    @Column(name = "user_id")
    private int userId;

    @Column(name = "created")
    private Instant created;

    @Column(name = "rating")
    private double rating;
}
```

# Отображение схемы в Java

```
@Entity
@Table(name = "movies")
public class Movie {

    @Id
    private int id;

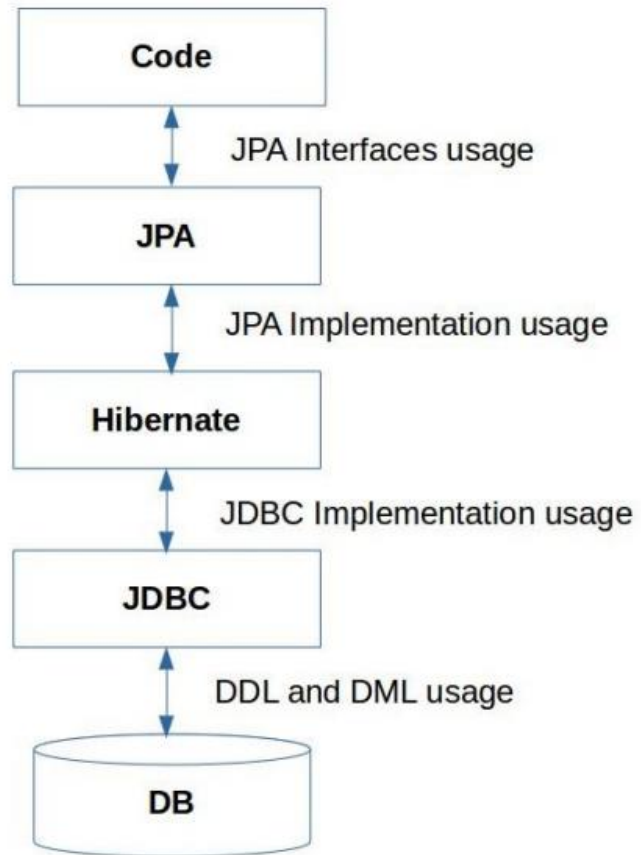
    @Column(name = "title")
    private String title;

    @Column(name = "year")
    private int year;

    @OneToMany(mappedBy = "movie")
    private List<Rating> ratings;

}
```

# Структура Spring Data



# Репозиторий Spring Data

```
package ru.naumen.cinema.repos;
```

```
import org.springframework.data.jpa.repository.JpaRepository;
```

```
import ru.naumen.cinema.db.Movie;
```

```
public interface MovieRepository extends JpaRepository<Movie, Integer> {
```

```
}
```



# Репозиторий Spring Data

@NoRepositoryBean

```
public interface JpaRepository<T, ID> extends PagingAndSortingRepository<T, ID>, QueryByExampleExecutor<T> {
```

@Override

```
List<T> findAll();
```

@Override

```
List<T> findAll(Sort sort);
```

@Override

```
List<T> findAllById(Iterable<ID> ids);
```

@Override

```
<S extends T> List<S> saveAll(Iterable<S> entities);
```

# Репозиторий Spring Data

```
public void updateMovieTitle(int movieId, String newTitle) {  
    var movie = movieRepository.getOne(movieId);  
    movie.setTitle(newTitle);  
    movieRepository.save(movie);  
}
```

# Свой запрос в Spring Data

```
public interface RatingRepository extends JpaRepository<Rating, Integer> {  
  
    @Query("select avg(rating) from Rating where movie = :movie")  
    double avgRatingForMovie(Movie movie);  
}
```

# Свой запрос в Spring Data

```
public interface RatingRepository extends JpaRepository<Rating, Integer> {  
  
    @Query("select avg(rating) from Rating where movie = :movie")  
    double avgRatingForMovie(Movie movie);  
  
    List<Rating> findByCreatedAfter(Instant date);  
}
```

# Реализация логики

```
public void addRatingToMovie(int movieId, int userId, double rating) {  
    var movie = movieRepository.getOne(movieId);  
  
    var newRating = new Rating();  
    newRating.setMovie(movie);  
    newRating.setUserId(userId);  
    newRating.setRating(rating);  
    newRating.setCreated(Instant.now());  
  
    ratingRepository.save(newRating);  
}
```

```
public void removeMovieIfBadRating(int movieId) {  
    var movie = movieRepository.getOne(movieId);  
    var avgRating = ratingRepository.avgRatingForMovie(movie);  
    if (avgRating < 1) {  
        movieRepository.delete(movie);  
    }  
}
```

# Реализация логики

```
ratingService.addRatingToMovie(movieId, userId, rating);  
ratingService.removeMovieIfBadRating(movieId);
```

# Реализация логики

```
@Transactional
public void addRatingToMovie(int movieId, int userId, double rating) {
    var movie = movieRepository.getOne(movieId);

    var newRating = new Rating();
    newRating.setMovie(movie);
    newRating.setUserId(userId);
    newRating.setRating(rating);
    newRating.setCreated(Instant.now());

    ratingRepository.save(newRating);
}
```

```
@Transactional
public void removeMovieIfBadRating(int movieId) {
    var movie = movieRepository.getOne(movieId);
    var avgRating = ratingRepository.avgRatingForMovie(movie);
    if (avgRating < 1) {
        movieRepository.delete(movie);
    }
}
```

# Реализация логики

`@Transactional`

```
public void addRatingToMovie(int movieId, int userId, double rating) {  
    var movie = movieRepository.getOne(movieId);  
  
    createNewRating(movie, userId, rating);  
    removeMovieIfBadRating(movie);  
}
```

```
private void createNewRating(Movie movie, int userId, double rating) {  
    var newRating = new Rating();  
    newRating.setMovie(movie);  
    newRating.setUserId(userId);  
    newRating.setRating(rating);  
    newRating.setCreated(Instant.now());  
  
    ratingRepository.save(newRating);  
}
```

```
private void removeMovieIfBadRating(Movie movie) {  
    var avgRating = ratingRepository.avgRatingForMovie(movie);  
    if (avgRating < 1) {  
        movieRepository.delete(movie);  
    }  
}
```



# Оптимизация запросов

```
public interface RatingRepository extends JpaRepository<Rating, Integer> {  
  
    @Query("select avg(rating) from Rating where movie = :movie")  
    double avgRatingForMovie(Movie movie);  
}
```

# Оптимизация запросов

```
spring.jpa.show-sql=true  
spring.jpa.hibernate.format_sql=true
```

# Оптимизация запросов

Hibernate:

```
select
  avg(rating0_.rating) as col_0_0_
from
  ratings rating0_
where
  rating0_.movie_id=?
```

# Оптимизация запросов

```
explain analyze  
select avg(rating0_.rating) as col_0_0_  
from ratings rating0_  
where rating0_.movie_id=?
```

# Оптимизация запросов

QUERY PLAN	
1	Finalize Aggregate (cost=349623.13..349623.14 rows=1 width=8) (actual time=547.495..548.865 rows=1 loops=1)
2	→ Gather (cost=349622.91..349623.12 rows=2 width=32) (actual time=547.435..548.858 rows=3 loops=1)
3	Workers Planned: 2
4	Workers Launched: 2
5	→ Partial Aggregate (cost=348622.91..348622.92 rows=1 width=32) (actual time=523.296..523.297 rows=1 loops=3)
6	→ Parallel Seq Scan on ratings rating0_ (cost=0.00..348619.21 rows=1480 width=8) (actual time=343.728..523.282 rows=1 loops=3)
7	Filter: (movie_id = 193887)
8	Rows Removed by Filter: 9251149
9	Planning Time: 0.081 ms
10	JIT:
11	Functions: 17
12	Options: Inlining false, Optimization false, Expressions true, Deforming true
13	Timing: Generation 4.318 ms, Inlining 0.000 ms, Optimization 1.108 ms, Emission 18.608 ms, Total 24.034 ms
14	Execution Time: 550.976 ms

# Оптимизация запросов

```
private List<Row> rows;  
  
private List<Row> findAllRows(int movieId) {  
    var resultedRows = new ArrayList<Row>();  
  
    for (Row row : rows) {  
        if (row.movieId == movieId) {  
            resultedRows.add(row);  
        }  
    }  
  
    return resultedRows;  
}
```

# Оптимизация запросов

```
private List<Row> rows;  
  
private List<Row> findAllRows(int movieId) {  
    var resultRows = new ArrayList<Row>();  
  
    for (Row row : rows) {  
        if (row.movieId == movieId) {  
            resultRows.add(row);  
        }  
    }  
  
    return resultRows;  
}
```



```
private List<Row> rows;  
private Map<Integer, List<Row>> rowsByMovieId;  
  
private List<Row> findAllRows(int movieId) {  
    var resultRows = rowsByMovieId.get(movieId);  
  
    return resultRows;  
}
```

# Оптимизация запросов

```
create index idx_movie_id on ratings (movie_id)
```



# Оптимизация запросов

`create index idx_movie_id on ratings (movie_id)` – 8 секунд блокировки!

# Оптимизация запросов

`create index idx_movie_id on ratings (movie_id)` – 8 секунд блокировки!

`create index concurrently idx_movie_id on ratings (movie_id)`

# Оптимизация запросов

```
62
63 ✓ explain analyse
64 select avg(rating0_.rating) as col_0_0
65 from ratings rating0_
66 where rating0_.movie_id = 193887;
67
```

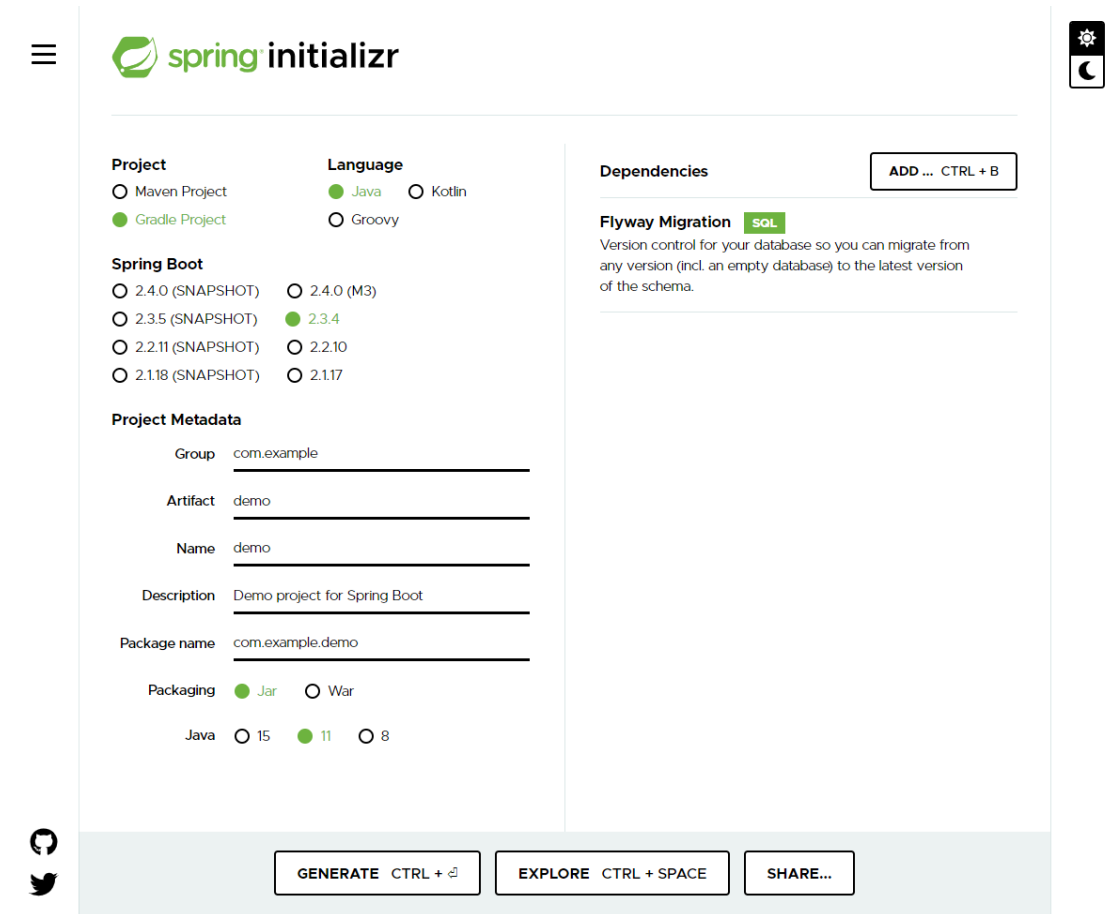
Output Result 23 x

8 rows v > >| ↺ ■ ↻ ↗

QUERY PLAN

- 1 Aggregate (cost=12797.16..12797.17 rows=1 width=8) (actual time=0.052..0.053 rows=1 loops=1)
- 2 → Bitmap Heap Scan on ratings rating0\_ (cost=43.97..12788.28 rows=3552 width=8) (actual time=0.031..0.032 rows=3 loops=1)
- 3 Recheck Cond: (movie\_id = 193887)
- 4 Heap Blocks: exact=1
- 5 → Bitmap Index Scan on idx\_movie\_id (cost=0.00..43.08 rows=3552 width=0) (actual time=0.028..0.028 rows=3 loops=1)
- 6 Index Cond: (movie\_id = 193887)
- 7 Planning Time: 0.186 ms
- 8 Execution Time: 0.079 ms

# Новые требования – новые проблемы



The image shows the Spring Initializr web interface. It features a sidebar with a hamburger menu, the Spring logo, and a Twitter icon. The main content area is divided into several sections: 'Project' with radio buttons for Maven and Gradle; 'Language' with radio buttons for Java, Kotlin, and Groovy; 'Spring Boot' with radio buttons for various versions (2.4.0, 2.3.5, 2.2.11, 2.1.18); 'Project Metadata' with input fields for Group, Artifact, Name, Description, and Package name; and 'Packaging' with radio buttons for Jar and War, and a 'Java' section with radio buttons for versions 15, 11, and 8. A 'Dependencies' section includes a search box and a 'Flyway Migration' section with a 'SQL' button. At the bottom, there are buttons for 'GENERATE', 'EXPLORE', and 'SHARE...'. A settings icon and a moon icon are visible in the top right corner.

spring initializr

**Project**

Maven Project  Gradle Project

**Language**

Java  Kotlin  Groovy

**Spring Boot**

2.4.0 (SNAPSHOT)  2.4.0 (M3)  2.3.5 (SNAPSHOT)  2.2.11 (SNAPSHOT)  2.2.10  2.1.18 (SNAPSHOT)  2.1.17

**Project Metadata**

Group

Artifact

Name

Description

Package name

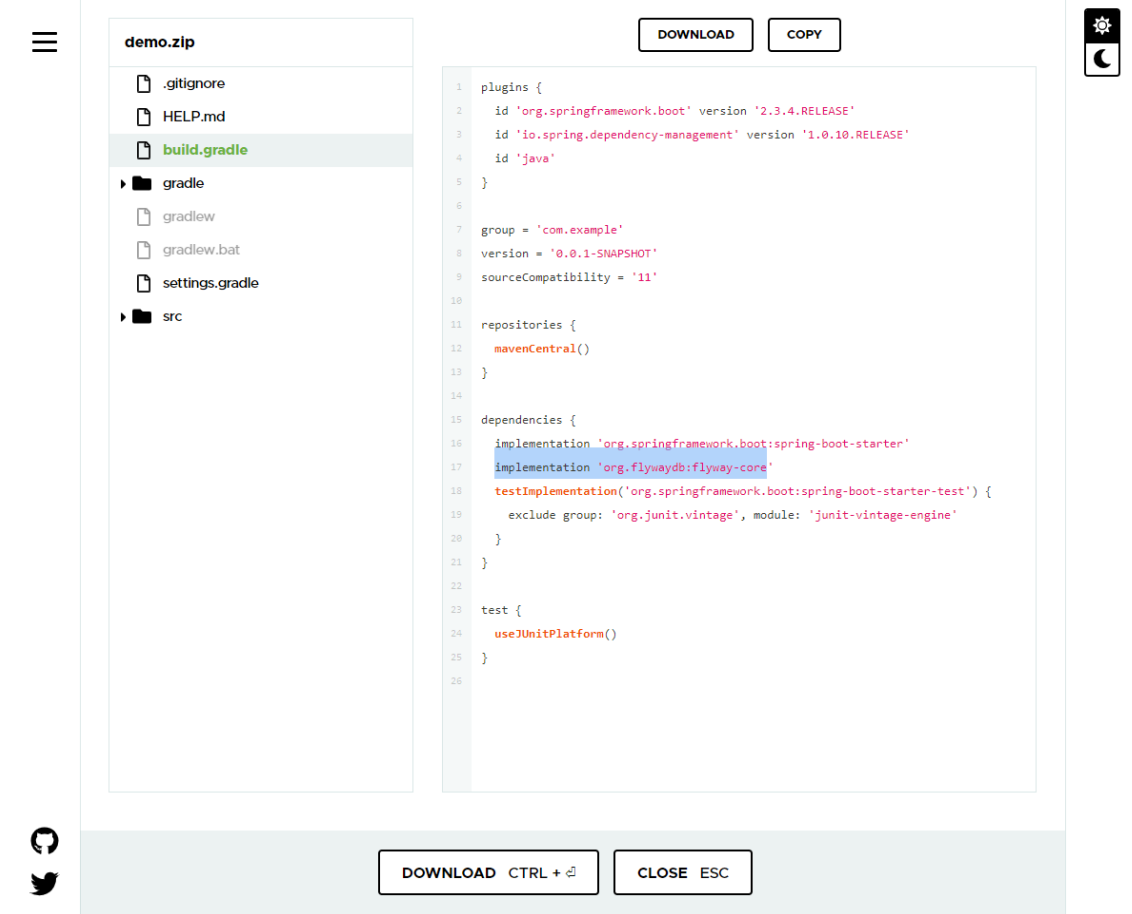
Packaging  Jar  War

Java  15  11  8

**Dependencies**

**Flyway Migration**

Version control for your database so you can migrate from any version (incl. an empty database) to the latest version of the schema.



The image shows the Spring Initializr interface displaying a 'demo.zip' file. The file explorer on the left shows the contents of the zip file: .gitignore, HELP.md, build.gradle, gradle, gradlew, gradlew.bat, settings.gradle, and src. The code editor on the right shows the contents of the build.gradle file. At the top right of the code editor, there are 'DOWNLOAD' and 'COPY' buttons. At the bottom, there are 'DOWNLOAD' and 'CLOSE' buttons. A settings icon and a moon icon are visible in the top right corner.

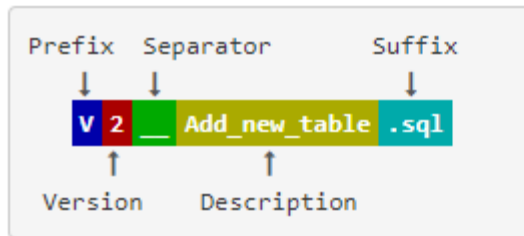
demo.zip

- .gitignore
- HELP.md
- build.gradle
- gradle
  - gradlew
  - gradlew.bat
  - settings.gradle
- src

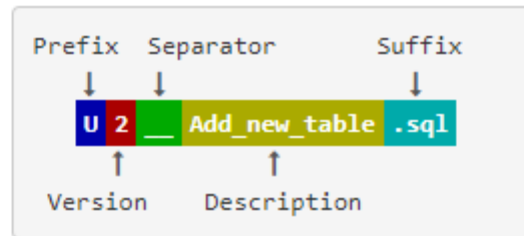
```
1 plugins {
2   id 'org.springframework.boot' version '2.3.4.RELEASE'
3   id 'io.spring.dependency-management' version '1.0.10.RELEASE'
4   id 'java'
5 }
6
7 group = 'com.example'
8 version = '0.0.1-SNAPSHOT'
9 sourceCompatibility = '11'
10
11 repositories {
12   mavenCentral()
13 }
14
15 dependencies {
16   implementation 'org.springframework.boot:spring-boot-starter'
17   implementation 'org.flywaydb:flyway-core'
18   testImplementation('org.springframework.boot:spring-boot-starter-test') {
19     exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'
20   }
21 }
22
23 test {
24   useJUnitPlatform()
25 }
26
```

# Новые требования – новые проблемы

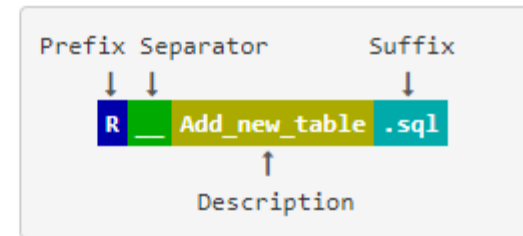
Versioned Migrations



Undo Migrations

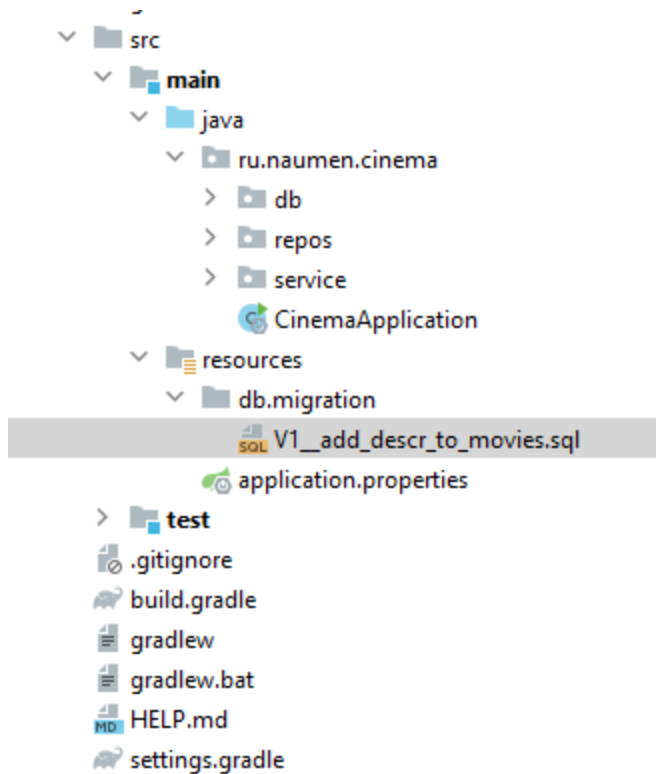


Repeatable Migrations



# Новые требования – новые проблемы

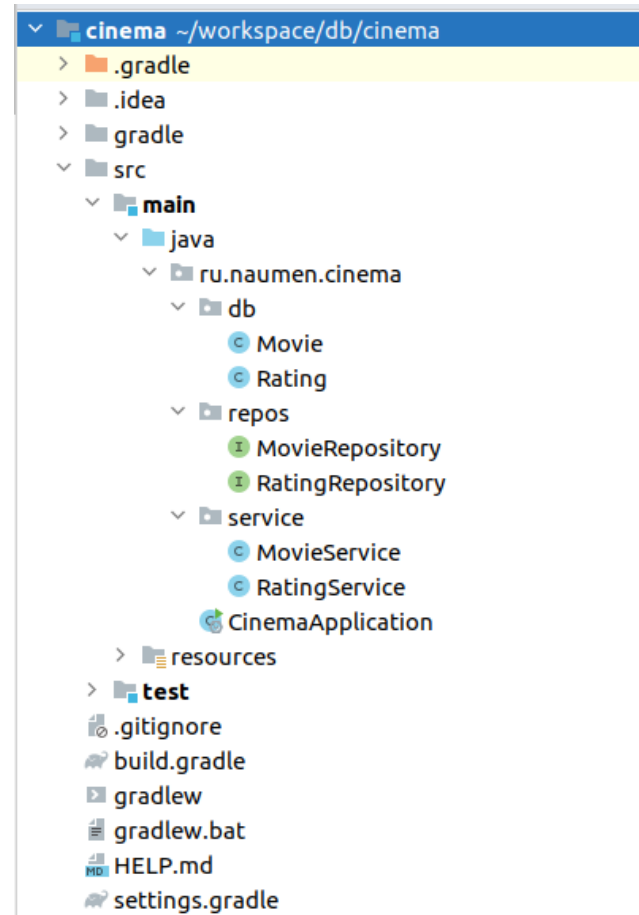
alter table if exists movies add column if not exists description text;



# Новые требования – новые проблемы

`spring.jpa.hibernate.ddl-auto=update`

# ИТОГ





# ИТОГ

- Вспомнили что такое СУБД и чем отличается от БД
- Поняли зачем нужно использовать СУБД
- Увидели как можно быстро создавать проект на Spring Boot'e
- Научились отображать реляционную схему на простые классы
- Научились сохранять данные в бд и эффективно читать их оттуда
- Поняли как атомарно вносить несколько изменений в бд
- Научились писать миграции