Работа с базами данных в Java

Тарасов Владимир
в.и.п. в отделе разработки
OPMS департамента КЦ

NAUMEN, 2018

Disclaimer

Весь материал данного доклада является личным мнением и опытом автора. Naumen и докладчик не несут никакой ответственности за применение содержимого доклада на практике

Кто я?

@vannoying

- Ведущий инженер-программист в Naumen
- Опыт разработки на Java около 8 лет
- Server-side разработчик
- Крупные проекты: Naumen CRM, Naumen CC OPMS

План

- О базах данных
- Проектирование в ER-модели
- Переход к реляционной модели
- Паттерны источника данных
- Java SQL Framework
- Паттерны ORM
- Java Persistence API

Не будет...

- Не реляционные базы данных
- Нормализация
- Индексирование
- Ограничения уникальности
- Транзакционность
- WEB разработка
- Паттерны бизнес слоя

Проект

- О кино (www.kinopoisk.ru)

  1. Просмотр списка фильмов

  2. Фильтрация по списку (название, год, жанр)

  3. Просмотр описания выбранного фильма

- Администрирование (просмотр/добавление/изменение/удаление бизнес-объектов)

Базы данных (Database)

- БД (DB) — это набор порций информации, существующей в течение длительного времени, под контролем СУБД
- СУБД (DBMS) — Система управления БД (Database Management System), Система БД (Database System)

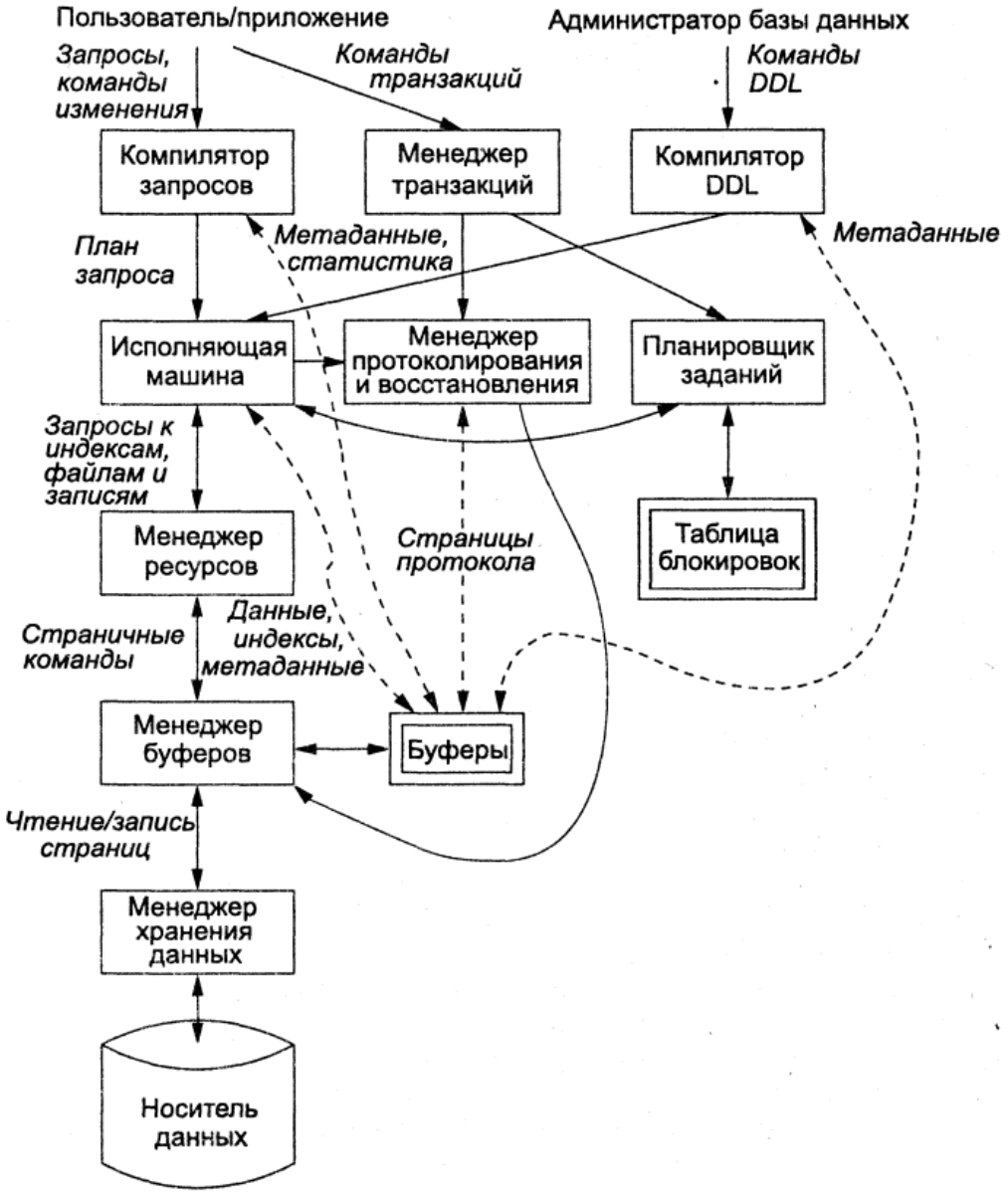
Функциональные возможности СУБД

- Средства постоянного хранения данных
- Интерфейс программирования
- Управление транзакциями (ACID):
  1. атомароность (atomicity)
  2. консистентность (consistence)
  3. изоляция (isolation)
  4. надежность (durability)

Требования к СУБД

- Создание схем (schema) баз данных (язык определения данных DDL)
- Выборка и модификация данных (язык манипулирования данными DML через запросы Query (SQL))
- Способность хранения больших объемов информации
- Обеспечение одновременного доступа

Структура СУБД

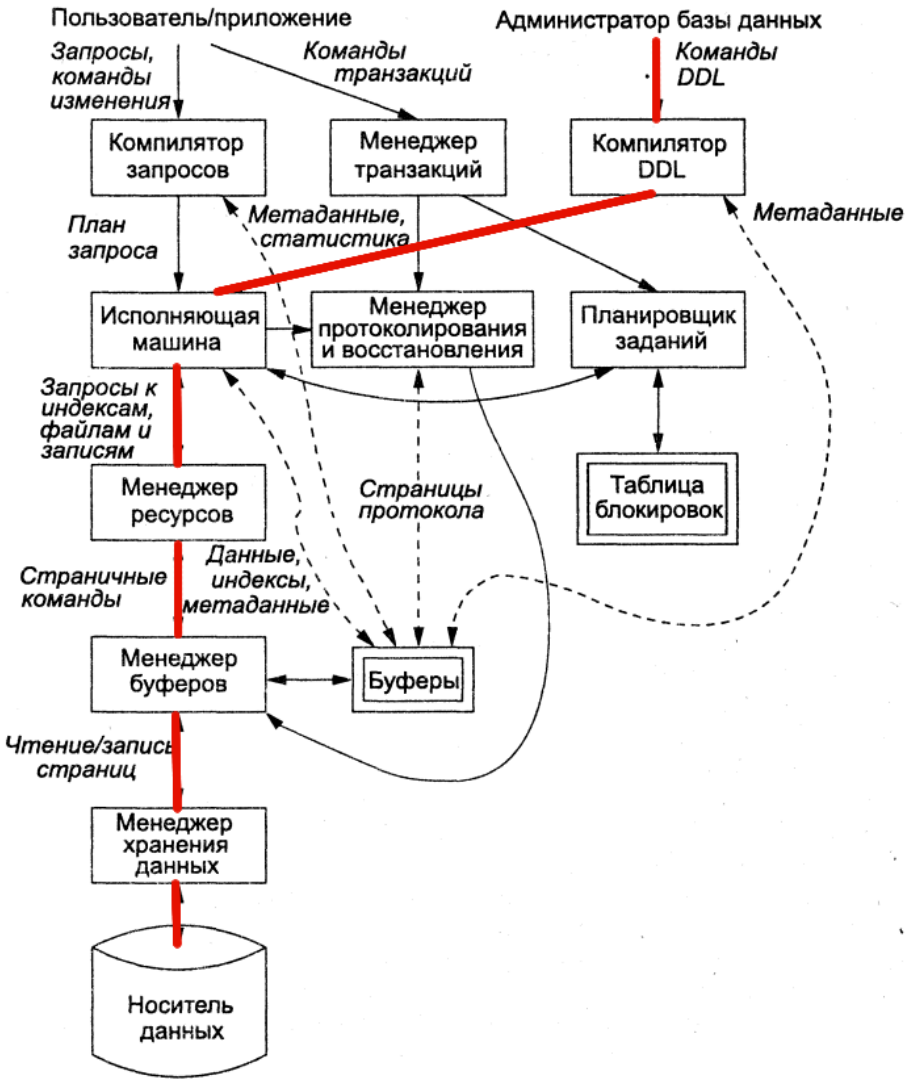


Два потока команд:

1. Пользователь — запросы и изменение данных (DML)

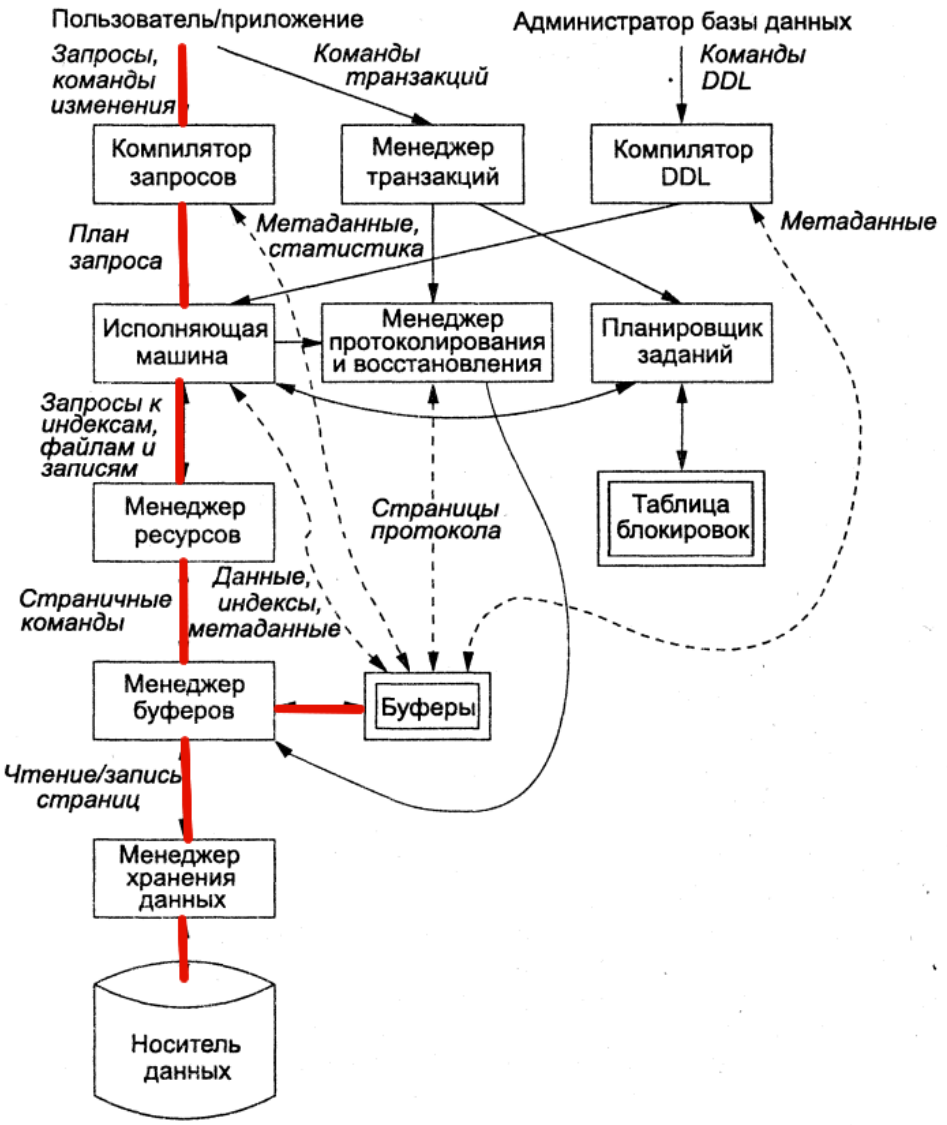
2. Администратор (DBA) — поддержка и развитие схемы (DDL)

Обработка команд DDL



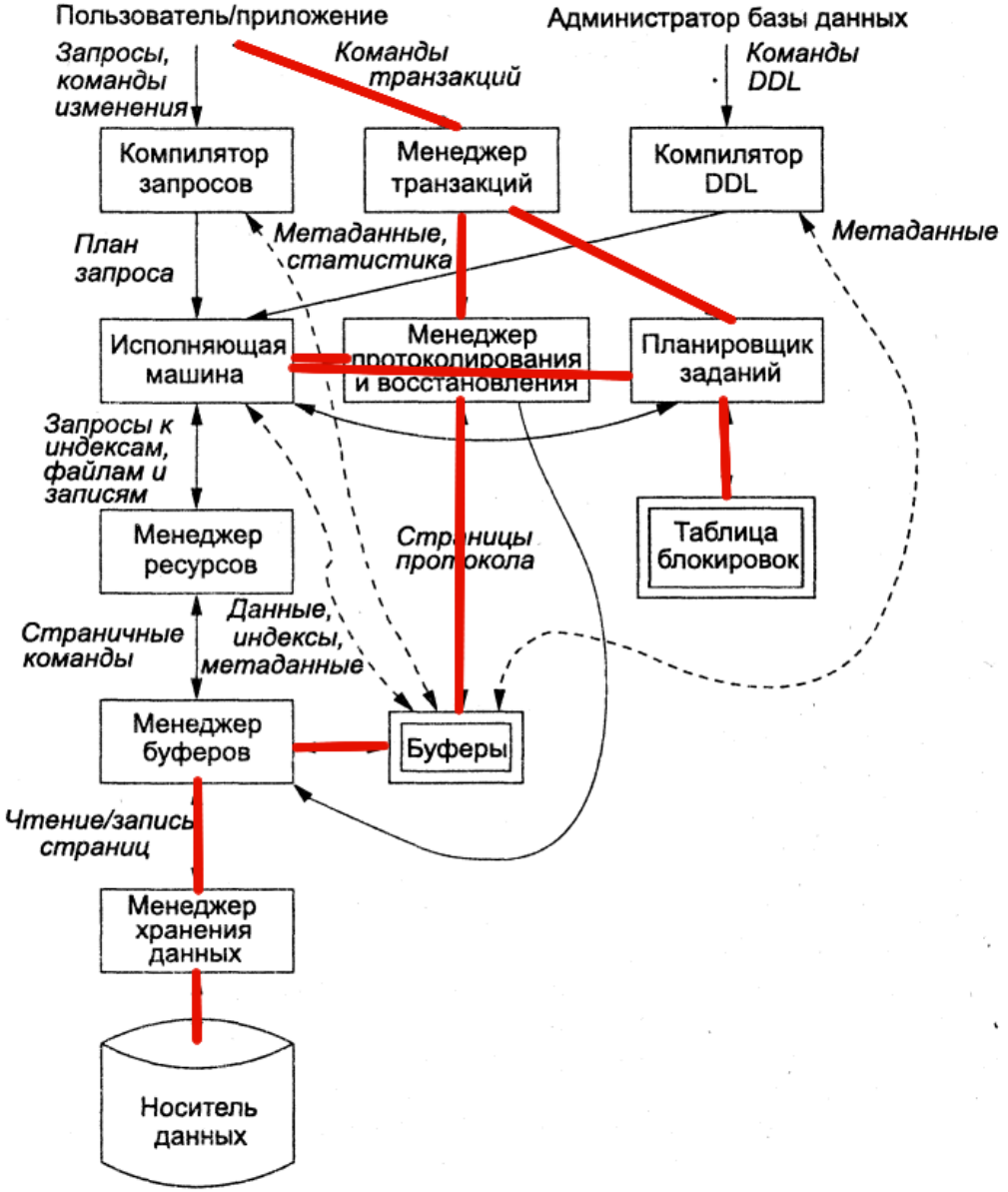
- Отправка команды
- Компиляция (DDL Compiler)
- Исполнение (Execution Engine)
- Менеджер ресурсов (Resource Manager)
- Изменение метаданных (Metadata, Scheme)

Обработка запросов DML



- Компиляция в план запроса (Query Plan)
- Оптимизация плана
- Исполнение
- Менеджер ресурсов
- Менеджер буферов (Buffer Manager)
- Менеджер хранения данных (Storage Manager)

Обработка транзакций



- Менеджер транзакций (Transaction Manager): разрешение блокировок (deadlock resoultion)
- Планировщик заданий (Scheduler)/Менеджер параллельных заданий (Concurrency Control Manager): атомарность, изолированность
- Менеджер протоколирования и восстановления (Logging and Recovery Manager): консистентность, надежность

Менеджер буферов и хранения данных

- Менеджер хранения данных — работает с блоками данных файлов на диске (disk blocks)
- Менеджер буферов — работает со страницами данных (pages), содержащих блоки данных, и кэширует:

  1. Данные
  2. Метаданные
  3. Статистика
  4. Индексы

Процессор запросов

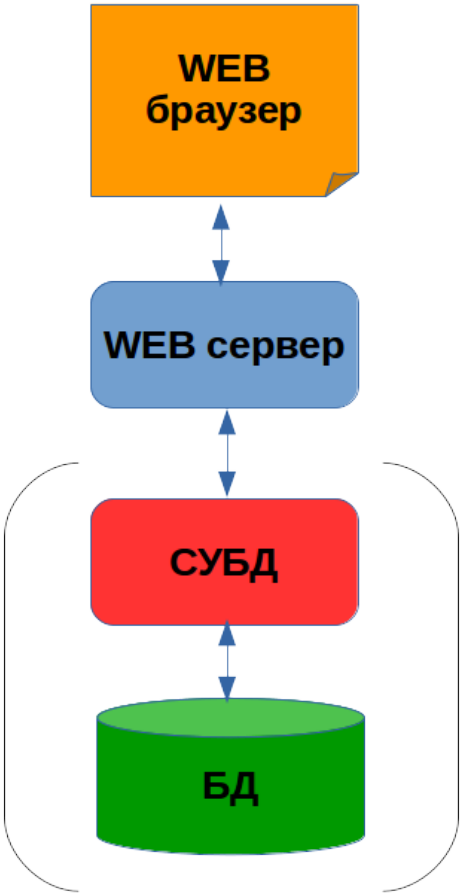
- Отвечает за производительность системы
- Состоит из:

  1. Компилятор запросов: синтаксический анализатор (query parser) — дерево, препроцессор запросов (query preprocessor) — семантика, оптимизатор запросов (query optimizer) — план запроса

  2. Исполняющая машина — исполняет план запросов

Системы клиент/сервер

- Наиболее популярная архитектура



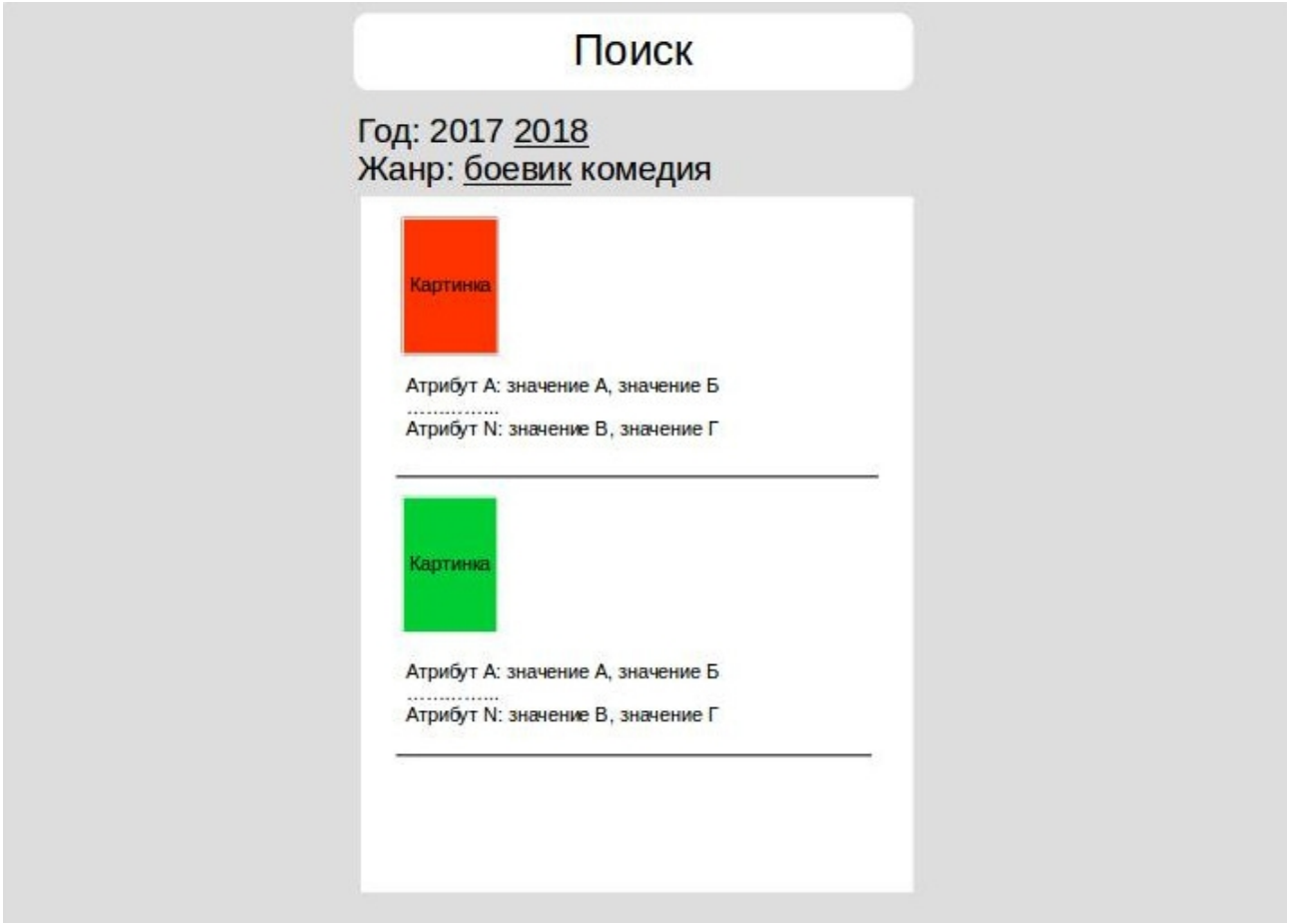
Проектирование

- Анализ предметной области
- Анализ требований
- Построение доменной модели (Domain Model)
- Проектирование бизнес-логики (Business-Logic)

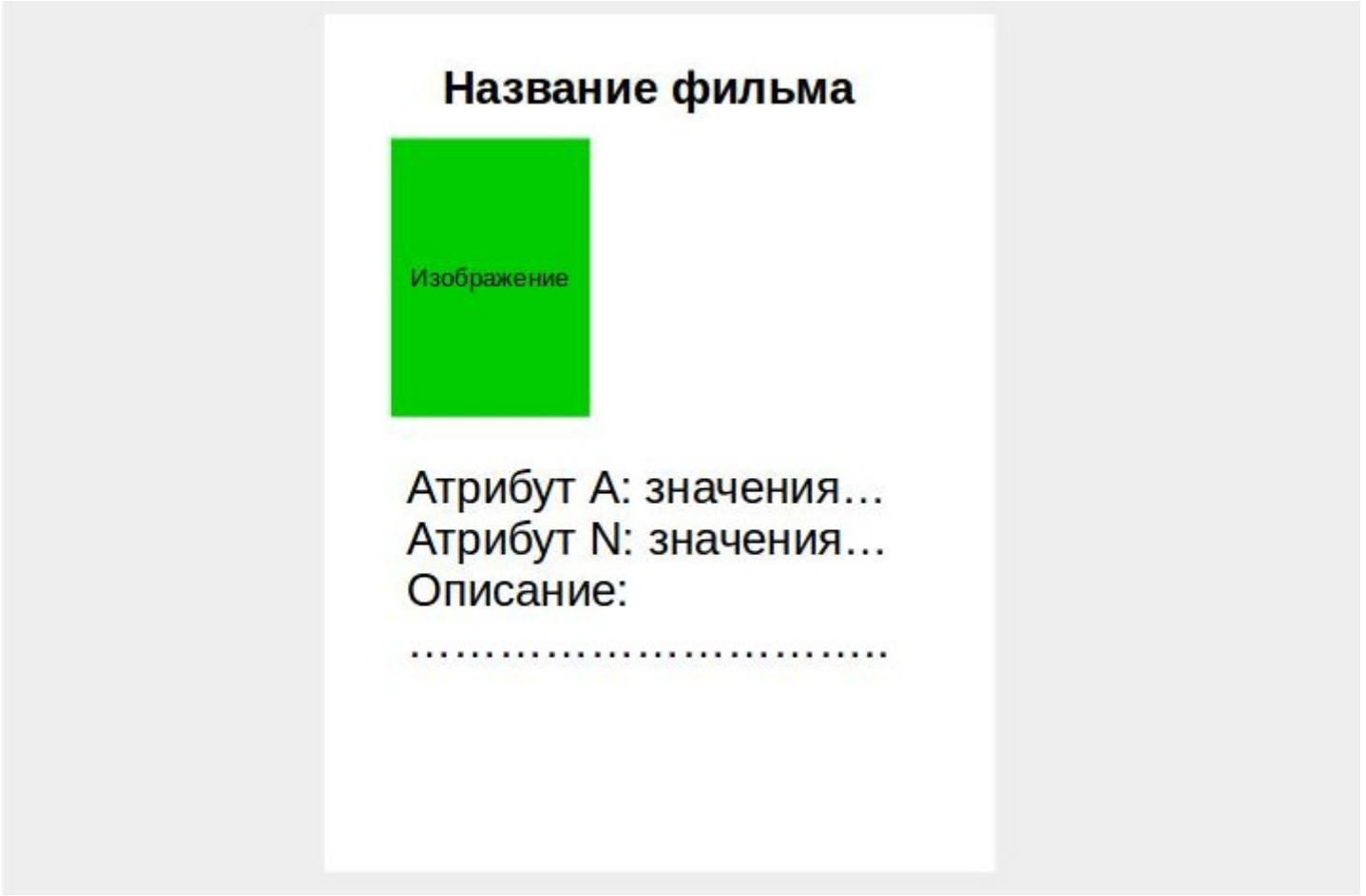
Анализ предметной области и анализ требований

- Сайт о просмотре информации о фильмах
- Главная страница со списком фильмов и фильтрацией по названию, годам и жанру
- Страница для каждого фильма с его описанием
- Страница администрирования для создания/изменения/удаления фильмов

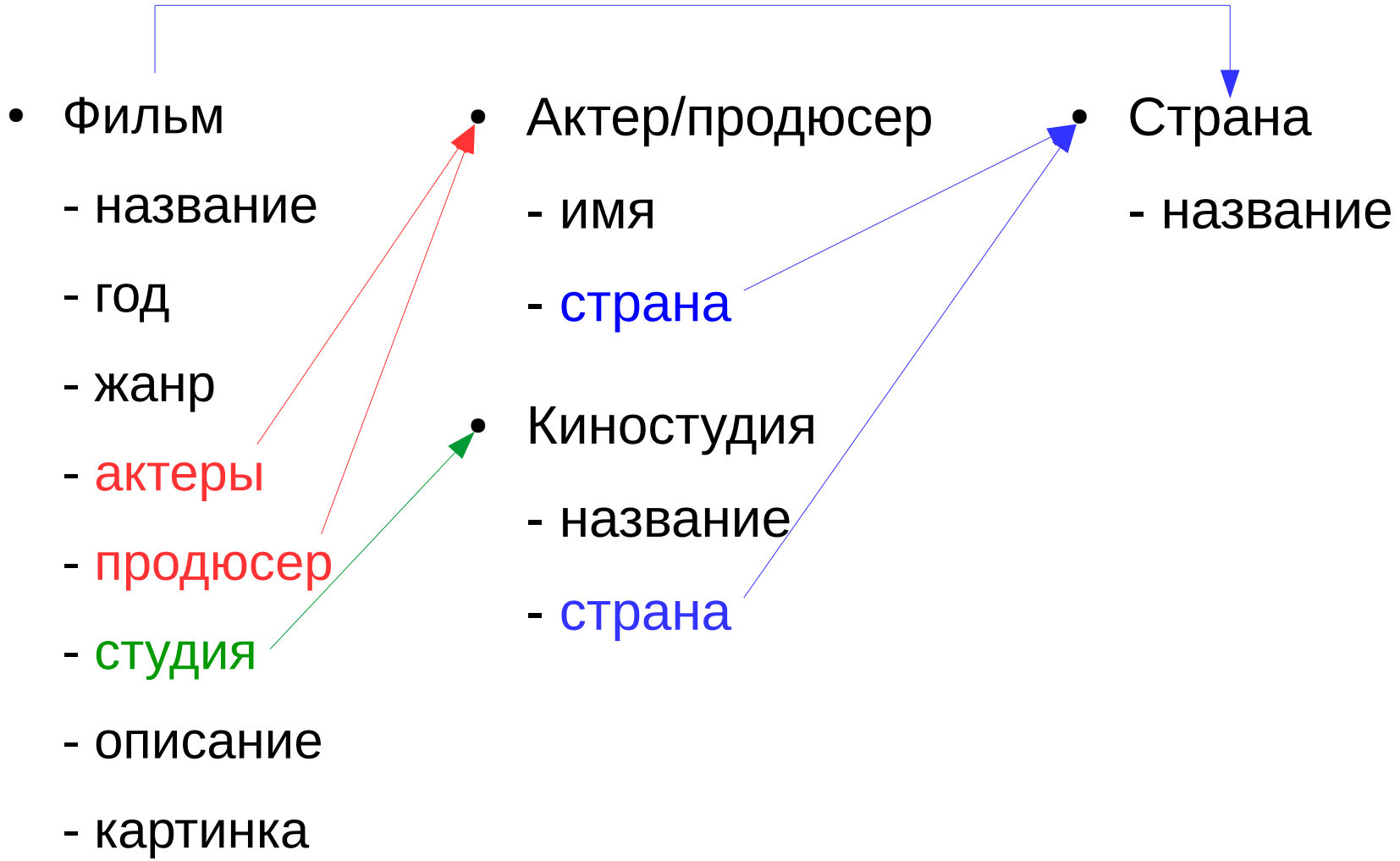
Главная страница



Страница для фильма



Построение доменной модели



Ограничения домена

- Название (строка 255 символов)
- Имя (строка 255 символов)
- Год (целое число 4 знака)
- Жанр (строка 255 символов)
- Изображение (байты без ограничений)
- Описание (текст без ограничений)

Проектирование бизнес-логики

- Сайт:

  1. Выборка из БД списка фильмов с актерами, продюсерами и странами

  2. Выборка из БД фильма с актерами, продюсером, страной и описанием

- Администрирование (CRUD-операции):

  1. Просмотр фильма/актера/продюсера/страны

  2. Добавление фильма/актера/продюсера/страны

  3. Изменение фильма/актера/продюсера/страны

  4. Удаление фильма/актера/продюсера/страны

Способы проектирования БД

- Specification first

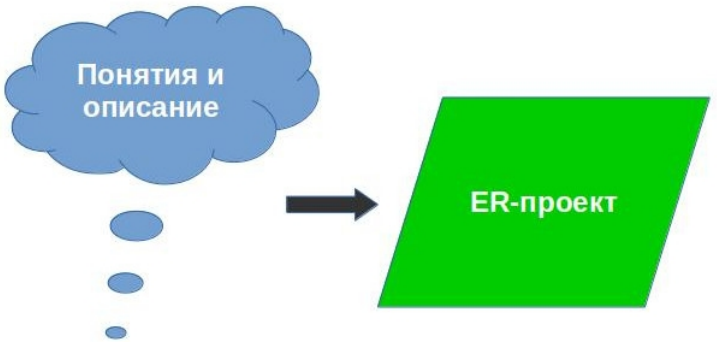
  1. OR-моделирование (Object-Relation Modeling)

  2. ER-моделирование (Entity-Relation Modeling)

- Implementation first

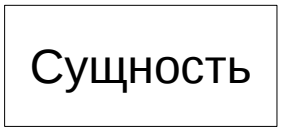
  Реляционное моделирование (Relation Modeling)

ER-моделирование

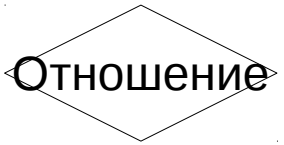


Элементы ER-модели:

- Множество сущностей

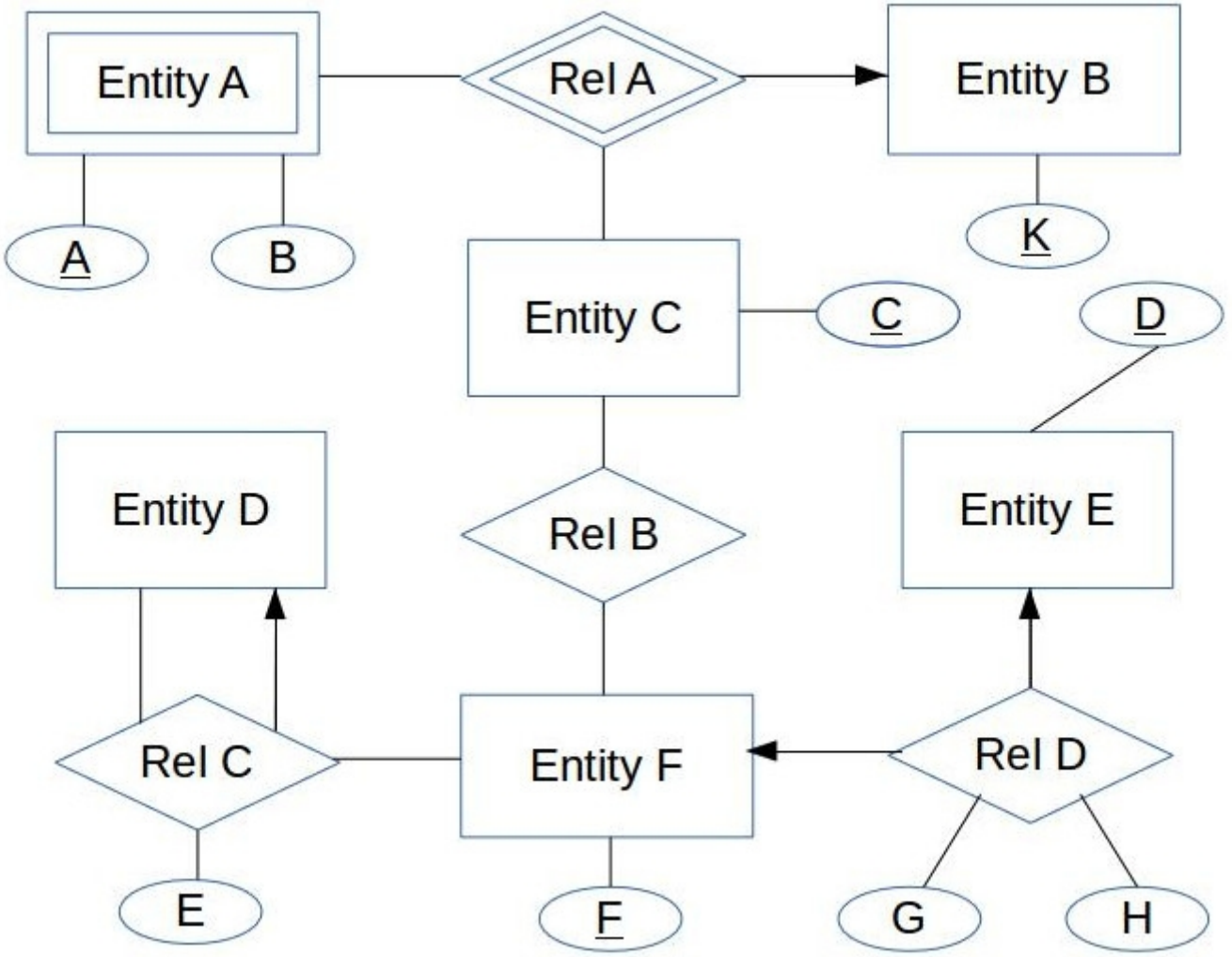


- Множество связей

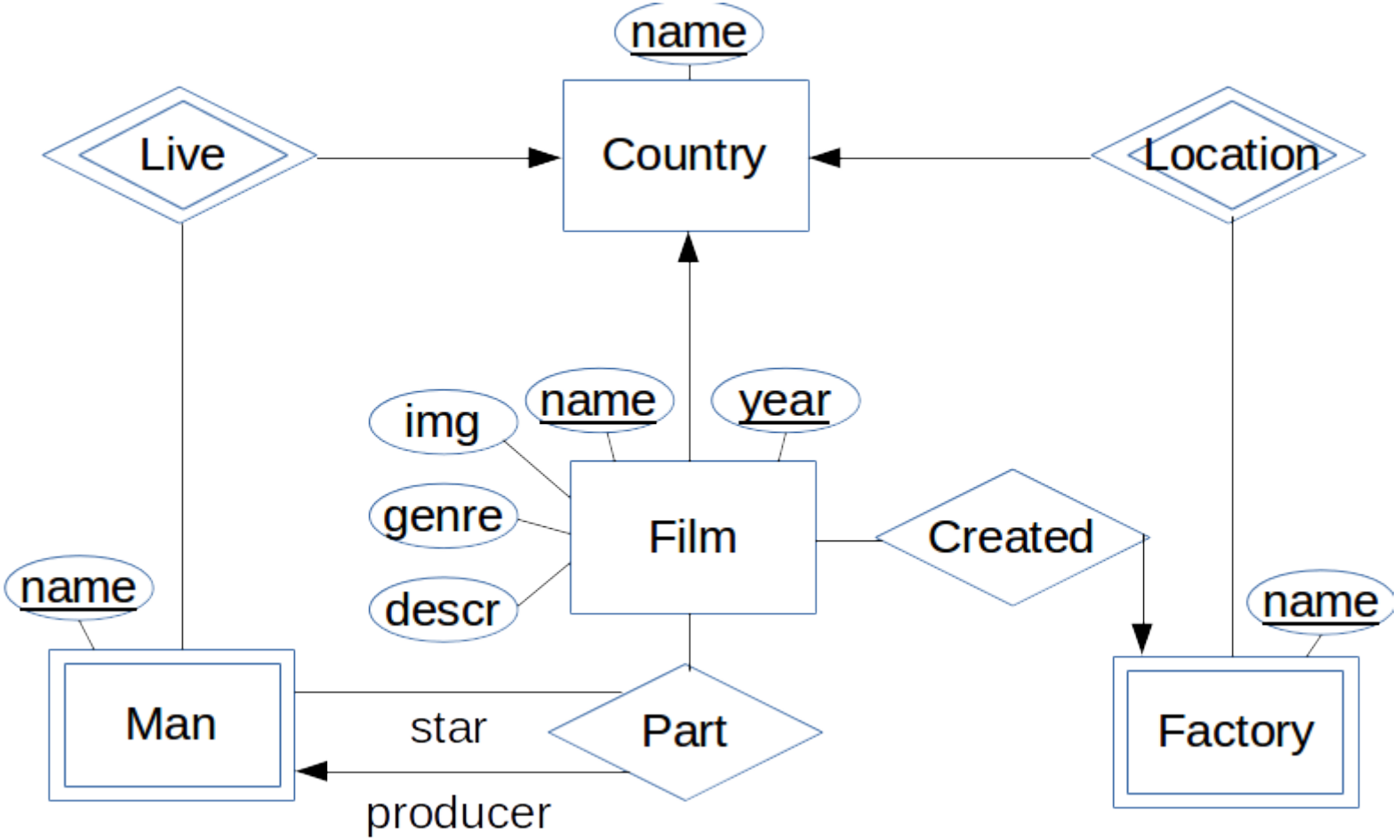


- Множество атрибуты

Пример ER-модели



ER-модель фильмов



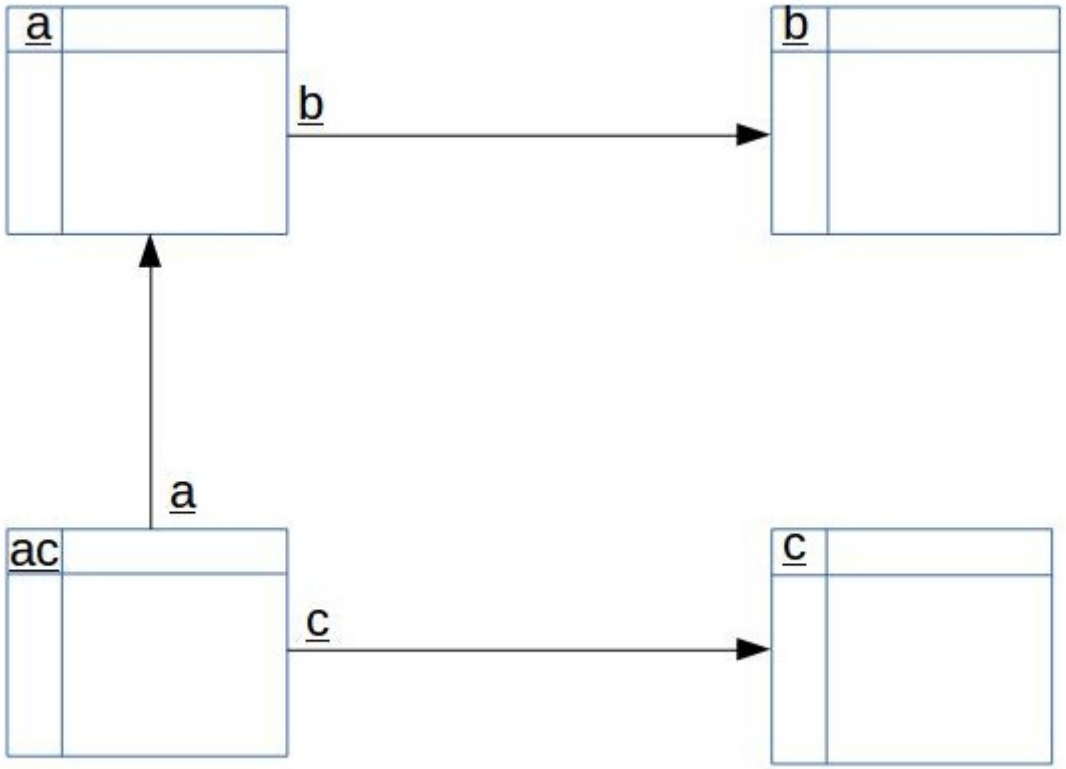
Реляционная модель

Элементы:

- Множество отношений
- Множество атрибутов
- Множество кортежей

a	b	c	d	e
a1	b1	c1	d1	e1
a2	null	null	d2	e2
aN	bN	null	d3	null

Пример реляционной модели



Переход от ER-модели к реляционной

- Сущности → отношения

  - атрибуты → атрибуты

  - ключи → ключи

- Связи

  - один к одному: pk сущностей → fk сущностей

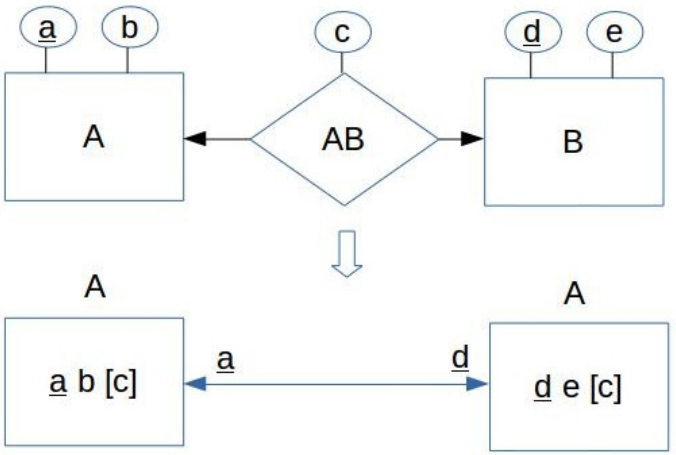
  - много к одному (A→B): pk сущности A → fk сущности B

  - много ко многим: связь → отношение, в котором fk = pk обеих сущностей

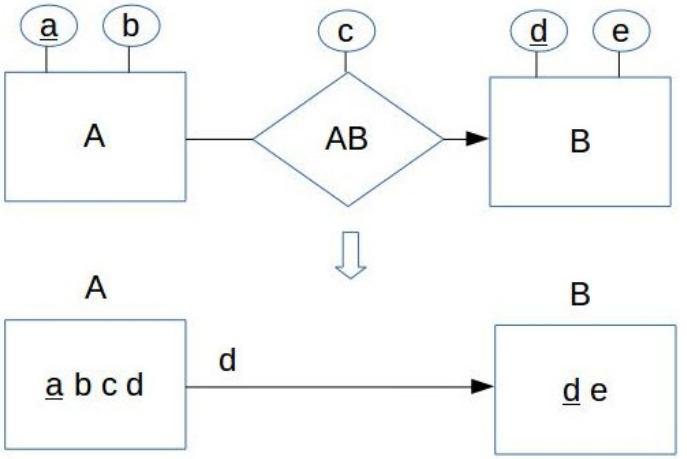
- Слабые сущности → отношения, в которых pk добавляется от сильной сущности

Переход от ER-модели к реляционной

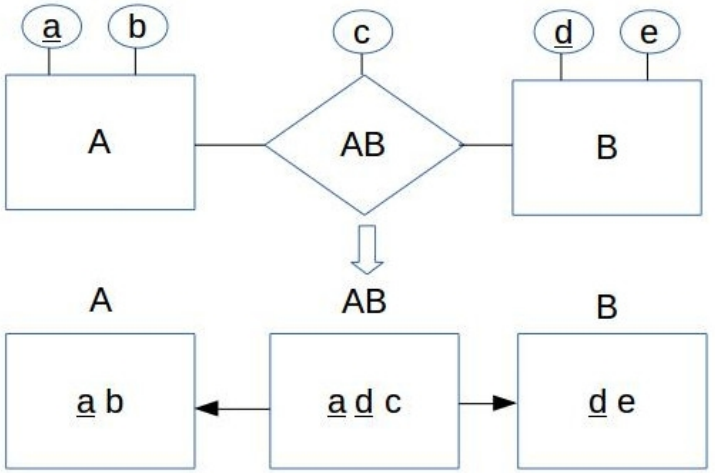
1 - 1



N - 1

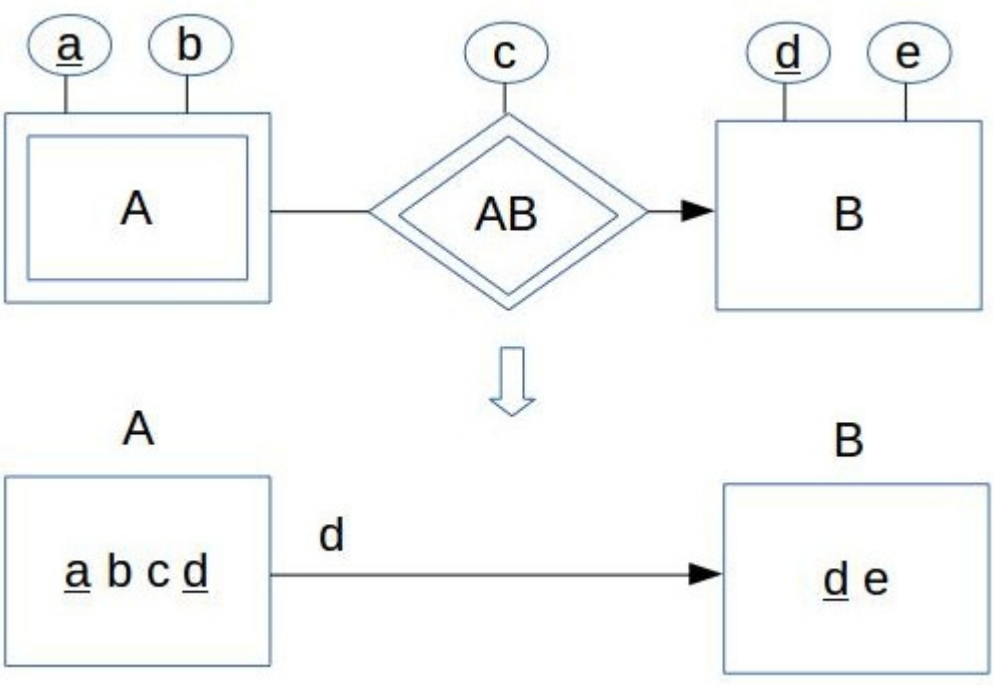


N - N

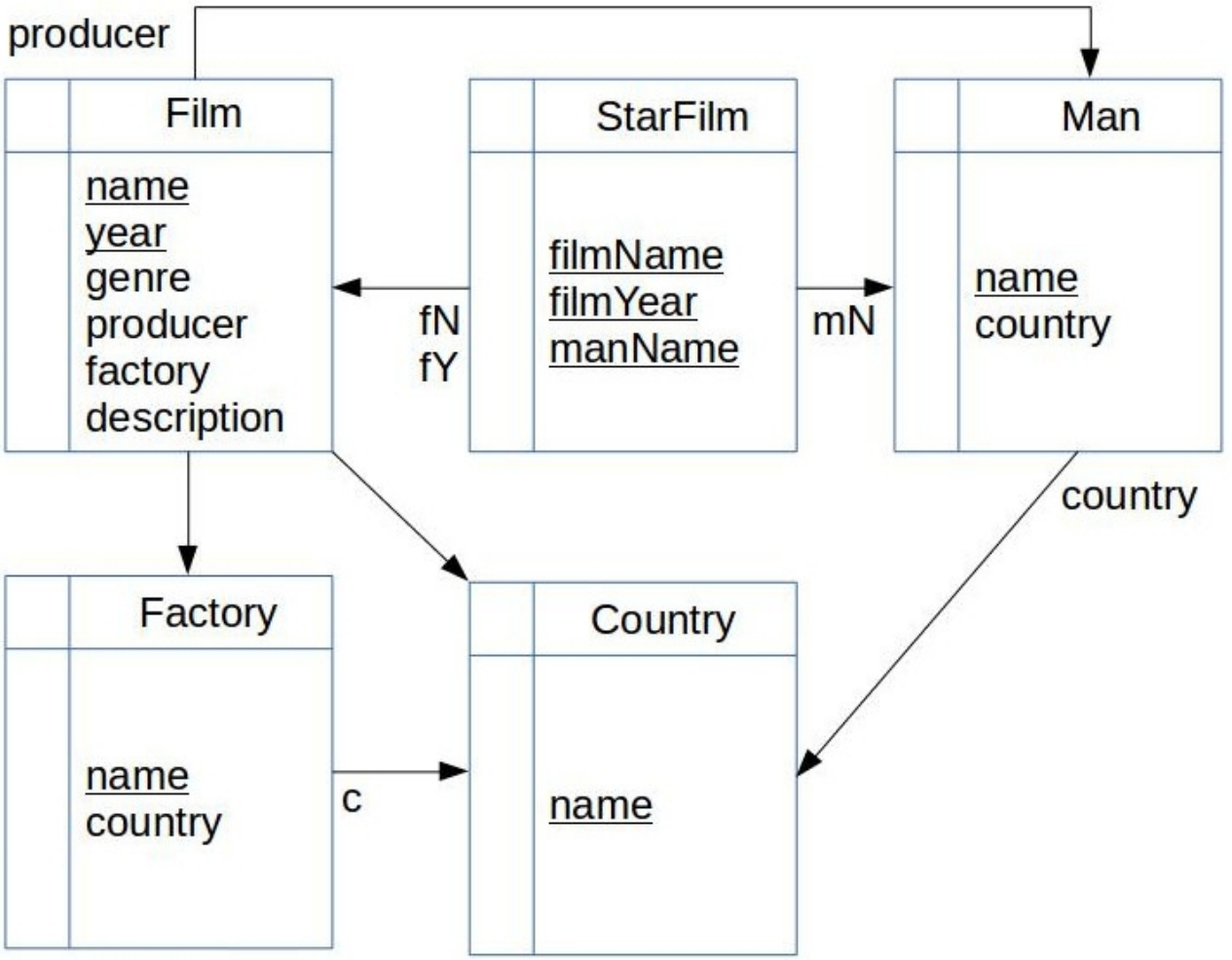


Переход от ER-модели к реляционной

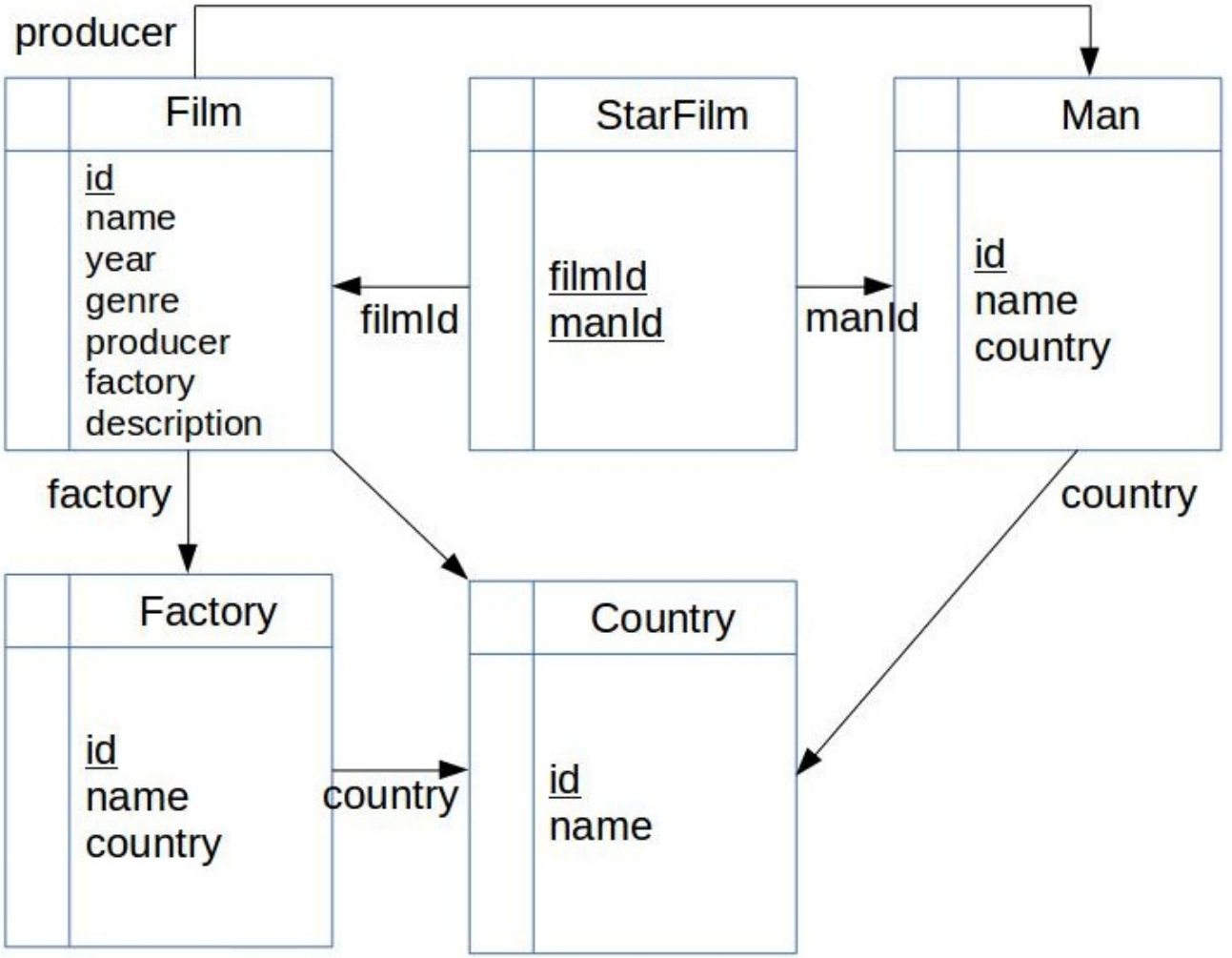
- Слабые сущности



Реляционная модель фильмов



Суррогатные ключи в реляционной модели фильмов



Генерация DDL-скрипта по реляционной модели

- На примере фильма (для остальных аналогично)

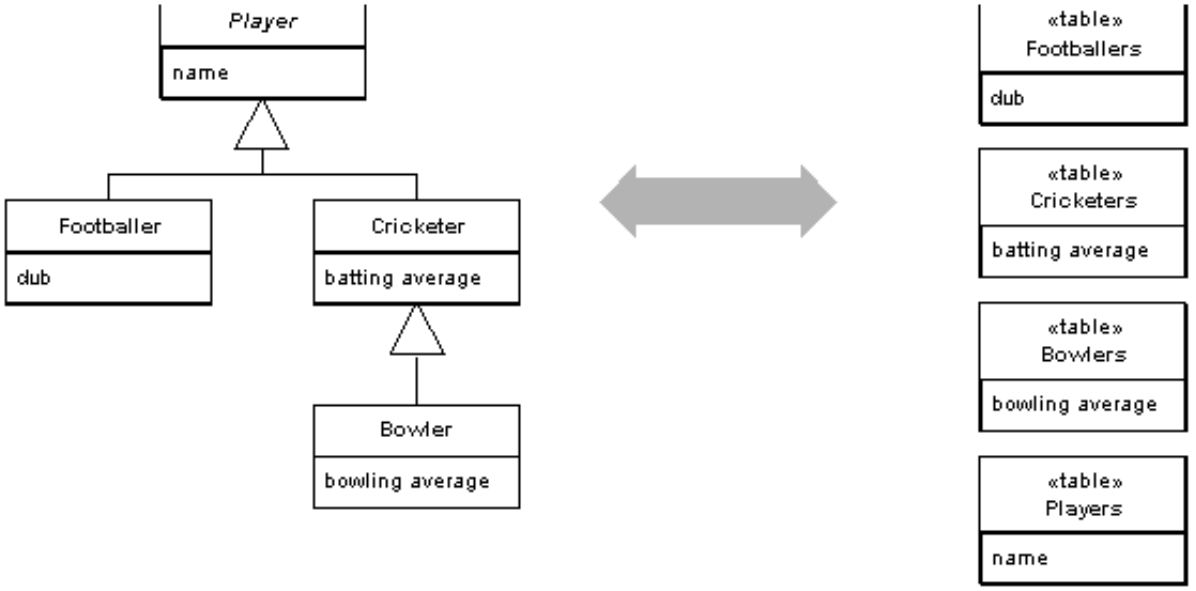
```
CREATE TABLE film
(
  id integer NOT NULL,
  name character varying(255) NOT NULL,
  genre character varying(255),
  year integer NOT NULL,
  producer integer,
  factory integer,
  description text,
  CONSTRAINT film_pkey PRIMARY KEY (id),
  CONSTRAINT film_factory_fkey FOREIGN KEY (factory)
      REFERENCES factory (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT film_producer_fkey FOREIGN KEY (producer)
      REFERENCES man (id) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT film_name_year_key UNIQUE (name, year)
)
WITH (
  OIDS=FALSE
);
ALTER TABLE film
  OWNER TO postgres;
```

Переход от реляционной модели к объектной

Паттерны

- Class Table Inheritance (Наследование с таблицами классов)
- Concrete Table Inheritance (Наследование с таблицами конечных классов)
- Single Table Inheritance (Наследование с единой таблицей)

Class Table Inheritance



+

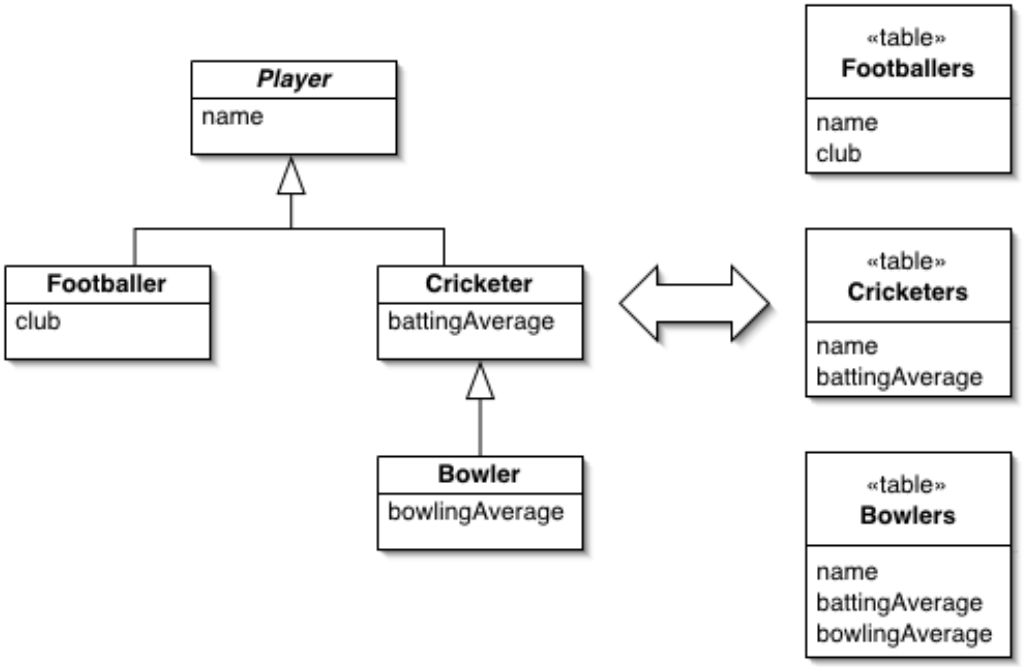
меньшие затраты памяти

-

связывание таблиц при загрузке объектов

связывание таблиц при загрузке списка разнотипных объектов

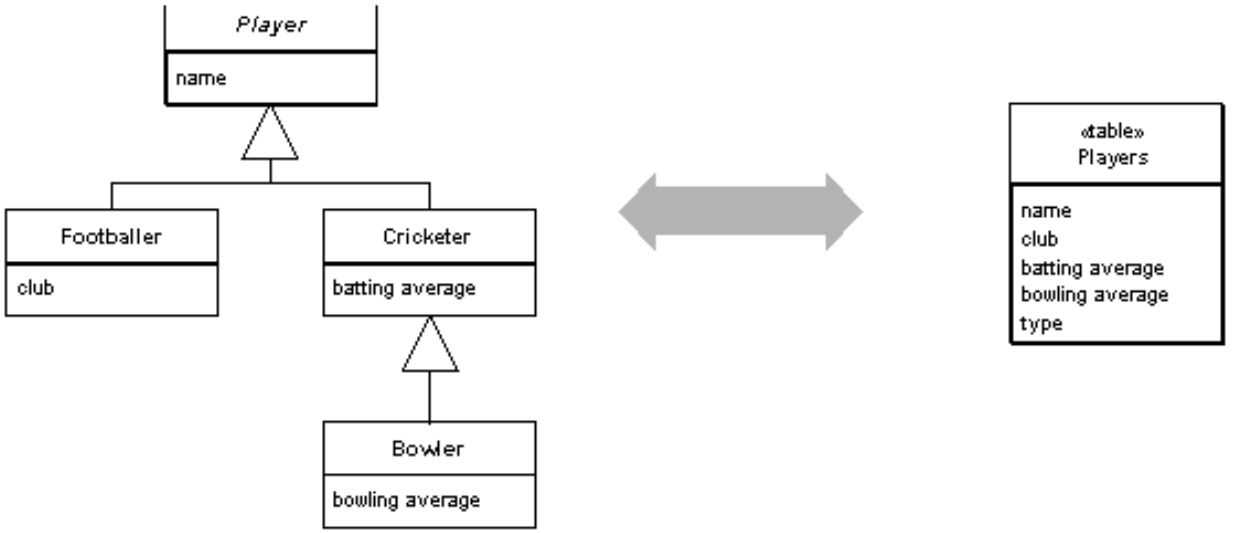
Concrete Table Inheritance



+
нет связывания таблиц при загрузке объектов

-
расходование памяти на метаинформацию
связывание таблиц при загрузке списка разнотипных объектов

Single Table Inheritance



+

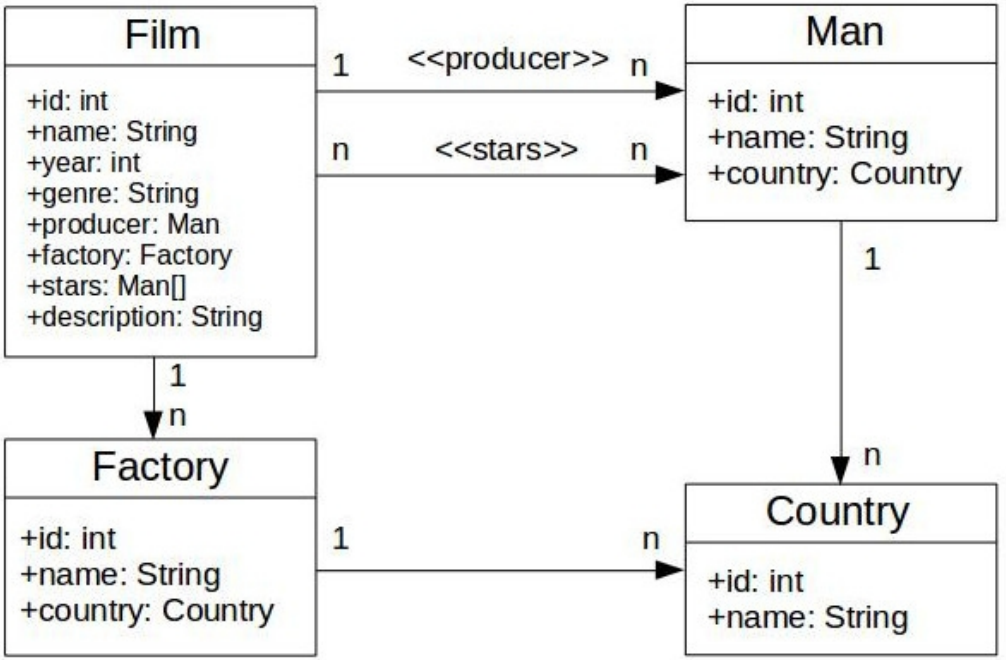
нет связывания таблиц

-

излишнее расходование памяти

Выбор объектной модели

- Так как нет наследования классов, то все модели будут выглядеть одинаково как Concrete Table Inheritance



Работа с базой данных в Java

- Доступ через Java SQL Framework

  \+ написание эффективного SQL

  \- сложность загрузки нескольких объектов

  \- зависимость от базы данных

- Доступ через Java ORM Framework (JPA + Hibernate)

  \+ SQL выполняет фреймворк за нас

  \- зачастую не эффективные запросы

Доступ через Java SQL Framework

- Java < 9 — пакеты javax.sql и java.sql
- Основные интерфейсы

  javax.sql.DataSource — фабрика соединений

  java.sql.Connection — соединение

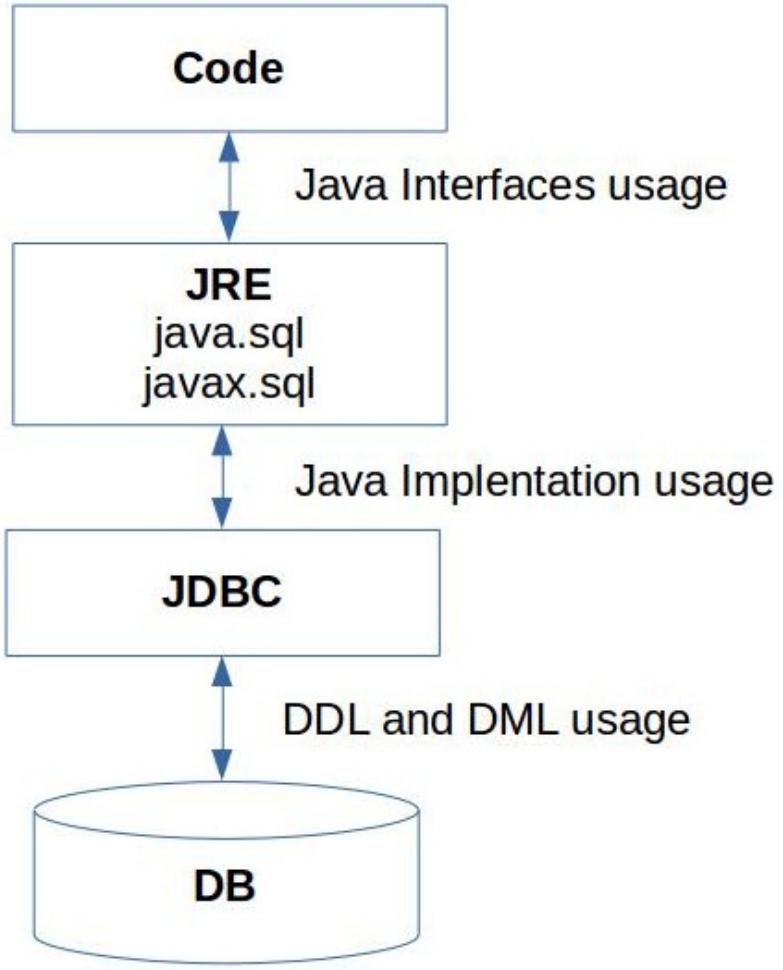
  java.sql.Statement — SQL-запрос

  java.sql.ResultSet — результат SQL-запроса
- В classpath должен лежать драйвер к используемой БД (JAR-файл)
- DataSource может быть создан

  - программно

  - зарегистрирован и получен через JNDI

Архитектура Java SQL Framework



Алгоритм работы с Java SQL Framefork

```
DataSource ds = /* Получение DataSource */

try (Connection con = ds.getConnection();
    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("select something from SOME_TABLE where something_else = whatever")) {

    while (rs.next()) {
        String something = rs.getString(1));

        /* Логика работы с полученным значением */
    }
}
```

+

непосредственный доступ к базе данных посредство JDBC-драйверов

-

много ручной работы
зависимость от драйверов
нативные SQL-запросы

Доступ через Java ORM Framework (JPA + Hibernate)

- Object Relation Mapping (ORM, паттерн DataMapper) — объектно-реляционное отображение
- Java Persistence API — обертка над ORM фреймворками для Java
- Hibernate — ORM фреймворк для Java

Доступ через Java ORM Framework (JPA + Hibernate)

- В classpath должны лежать библиотеки

  - org.hibernate.hibernate-core — ядро hibernate

  - org.hibernate.hibernate-entitymanager — реализация менеджера сущностей

  - jdbc-драйвер для используемой базы данных

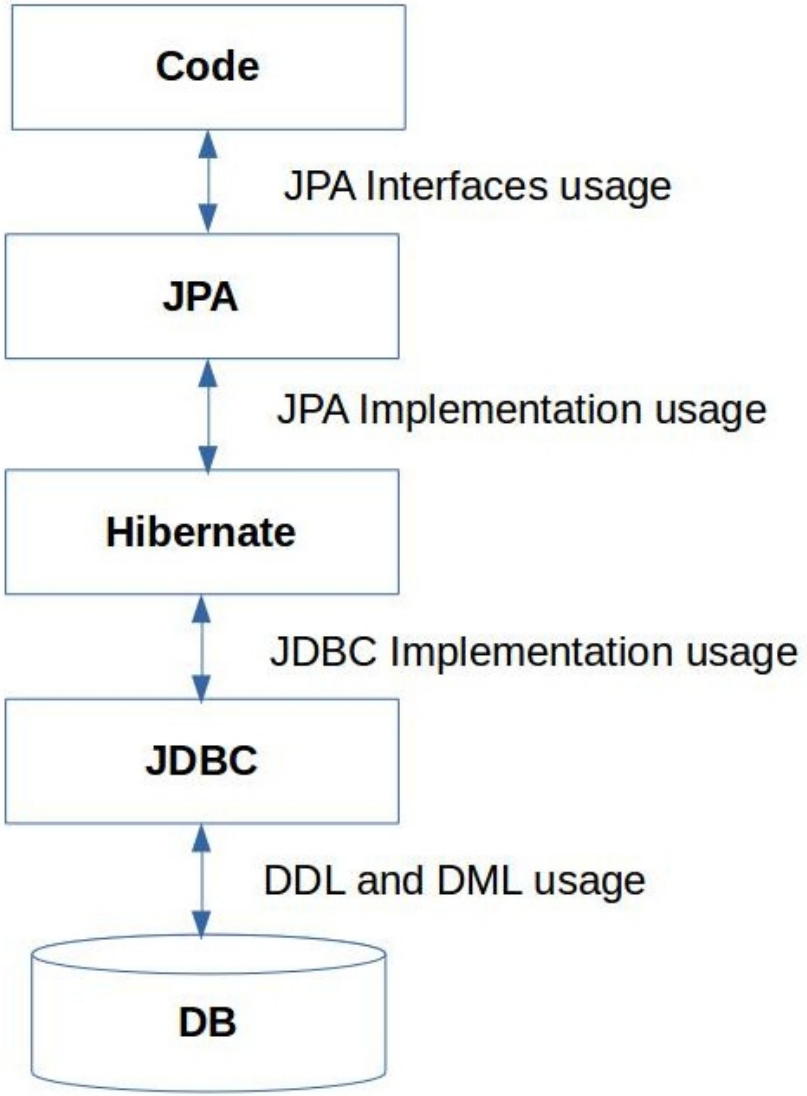
- Основные интерфейсы:

  javax.persistence.EntityManagerFactory — фаборика менеджера сущностей

  javax.persistence.EntityManager — менеджер сущностей

  javax.persistence.EntityTransaction — транзакция

Архитектура Java ORM Framework (JPA + Hibernate)



Алгоритм работы Java ORM Framework (JPA + Hibernate)

- Создание доменных объектов (POJO)
- Создание отображения на базу данных

  - persistence.xml

  - JPA-аннотации
- Алгоритм

```
EntityManagerFactory emf = /* Получение emf, например,
Persistence.createEntityManagerFactory("unit-name"); */
EntityManager manager = emf.createEntityManager();
EntityTransaction transaction = null;

try {
    transaction = manager.getTransaction();
    transaction.begin();

    MyPojo pojo = new MyPojo();
    pojo.setId(id);
    pojo.setName(name);

    manager.persist(pojo);
    transaction.commit();
} catch (Exception ex) {
    /* Обработка ошибки */
} finally {
    manager.close();
}
```

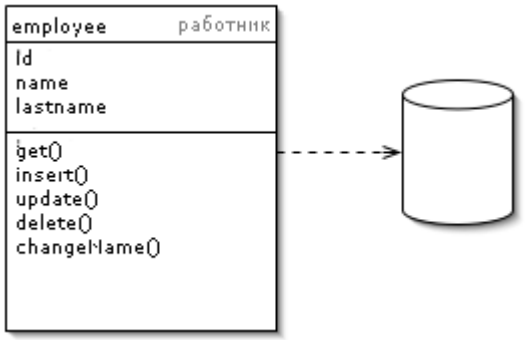
Пример аннотаций маппинга Java ORM Framework (JPA + Hibernate)

```
@Entity
public class MyPojo {

    @Id
    @SequenceGenerator(name="man_id_seq", sequenceName="man_id_seq")
    private int id;

    private String name;

    @ManyToOne
    @JoinColumn(name = "country")
    private Country country;

    /* Getter and setters */

}
```

Паттерны архитектуры источников данных

- Active Record (Активная запись)
- Table Data Gateway (Шлюз к данным таблицы)
- Row Data Gateway (Шлюз к данным записи)
- Data Mapper (Маппер данных)

Active Record (Активная запись)



- Объект сам работает с базой данных

+

короткие дистанции (модель и хранение в одном объекте)

инкапсуляция персистентности

-

нарушает Single of Responsibility

неудобство при написании логики и sql разными людьми

Active Record для фильмов

- На примере фильмов

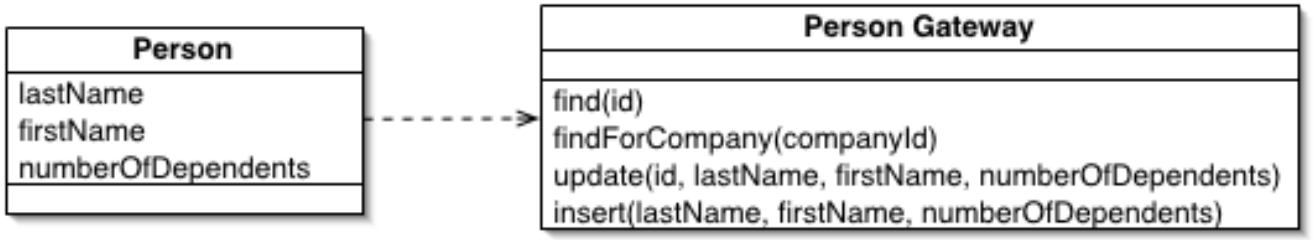
```
public class Film {
    private int id;
    private String name;
    private int year;
    private String genre;
    private Factory factory;
    private Man[] stars;
    private Man producer;
    private String description;

    /* Getters and Setters */

    public void save(DataSource ds) throws SQLException {
        try (Connection con = ds.getConnection();
            Statement stmt = con.createStatement()) {

            /* SQL-insert фильма (film) и его связи с актерами (starfilm) */
        }
    }

    public void update(DataSource ds) throws SQLException {
        try (Connection con = ds.getConnection();
            Statement stmt = con.createStatement()) {

            /* SQL-update фильма (film) и его связи с актерами (starfilm) */
        }
    }

    public void delete(DataSource ds) throws SQLException {
        try (Connection con = ds.getConnection();
            Statement stmt = con.createStatement()) {

            /* SQL-delete фильма (film) и его связи с актерами (starfilm) */
        }
    }

    public static Film load(DataSource ds, int id) throws SQLException {
        try (Connection con = ds.getConnection();
            Statement stmt = con.createStatement()) {

            /* SQL-select фильма (film), его связи с актерами (starfilm), продюссера, актеров и киностудии*/
        }
    }
}
```

Active Record для фильмов

- На примере фильмов

```
DataSource ds = /* Получение DataSource */

Film film = new Film();
film.setName("Rocky I");

film.save(ds);

Film film = film.load(ds, id);
film.setName("Rocky II");

film.update(ds);

film.delete(ds);
```

Table Data Gateway (Шлюз к данным таблицы)



+

логика персистентности отделена от бизнес-логики

-

бизнес-объект взаимодействует со шлюзом через DTO

нет инкапсуляции данных

Table Data Gateway для стран

- На примере стран

```
public class Country {
    private int id;
    private String name;

    /* Getters and Setters */
}
```

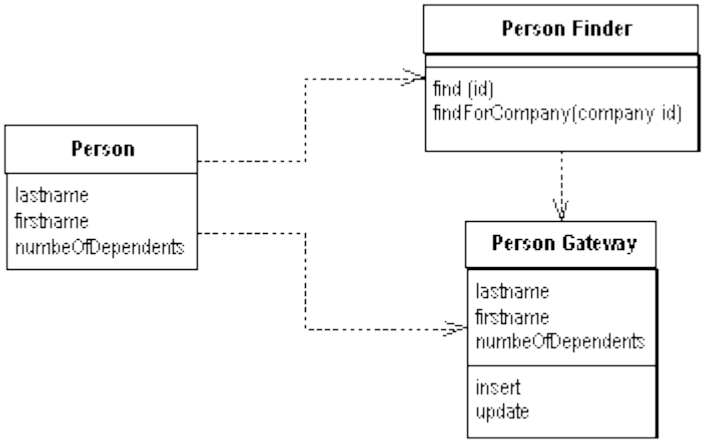
```
public class CountryGatewayImpl {
    private DataSourceFactory dsf;

    public void save(String name) throws Exception {
        try (Connection con = dsf.get().getConnection();
            Statement stmt = con.createStatement()) {

            /* SQL-insert */
        }
    }

    public void update(int id, String name) throws Exception {
        try (Connection con = dsf.get().getConnection();
            Statement stmt = con.createStatement()) {

            /* SQL-update */
        }
    }

    public void delete(int id) throws Exception {
        try (Connection con = dsf.get().getConnection();
            Statement stmt = con.createStatement()) {

            /* SQL-delete */
        }
    }
}
```

Table Data Gateway для стран

- На примере стран

```
CountryGatewayImpl gateway = new CountryGatewayImpl();
gateway.save("Russia");

Map<String, Object> map = gateway.load(id);

Country country = new Country(map);
country.setName("USA");

gateway.update(id, country.getName());

gateway.delete(id);
```

Row Data Gateway (Шлюз к данным записи)



+

логика персистентности отделена от бизнес-логики

-

бизнес-объект знает о шлюзе

Row Data Gateway для фильмов

- На примере человека

```
public class Man {
    private ManRowGateway gateway;

    public Man(ManRowGateway gateway) {
        this.gateway = gateway;
    }

    public ManRowGateway getGateway() {
        return gateway;
    }
}
```

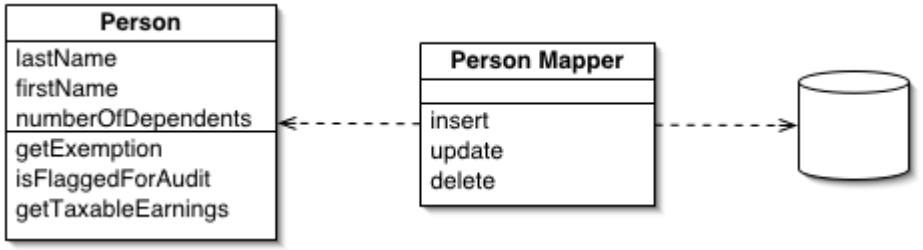
```
public class ManRowGatewayImpl {
    private DataSourceFactory dsf;

    private int id;
    private String name;
    private Country country;

    /* Getters and Setters */

    public void save() throws Exception {
        /* SQL-insert */
    }

    public void update() throws Exception {
        /* SQL-update */
    }

    public void delete() throws Exception {
        /* SQL-delete */
    }
}
```

Row Data Gateway для фильмов

- На примере человека

```
public class ManFinder {
    private DataSourceFactory dsf;

    @Override
    public ManRowGateway load(int id) throws Exception {
        /* SQL-select */
    }
}

ManFinder finder = new ManFinder();

Man man = new Man(new ManRowGatewayImpl());
man.setName("Stallone");

man.getGateway().save();

man = new Man(finder.load(id));
man.setName("Arnold");

man.getGateway().update();

man.getGateway().delete();
```

Data Mapper (Маппер данных)



+

отделяет бизнес-объект от персистентности

-

нарушает инкапсуляцию данных объекта

Data Mapper для фильмов

- На примере киностудии

```
public class Factory {
    private int id;
    private String name;
    private Country country;

    /* Getters and Setters */
}
```

```
public class FactoryMapper {
    private DataSourceFactory dsf;

    public void save(Factory f) {
        /* SQL-insert */
    }

    public void update(Factory f) {
        /* SQL-update */
    }

    public void delete(Factory f) {
        /* SQL-delete */
    }

    public Factory load(int id) {
        /* SQL-select */
    }
}
```

Data Mapper для фильмов

- На примере киностудии

```
FactoryMapper fm = new FactoryMapper();

Factory f = new Factory();
f.setName("Columbia");
f.setCountry(usa);

fm.save(f);

f = fm.load(id);
f.setName("Paramount");

fm.update(f);

fm.delete(f);
```

Использование Java ORM Framework для проекта фильмов

- ORM — реализация паттерна DataMapper
- Наиболее простой и быстрый способ обеспечения

  - персистентности

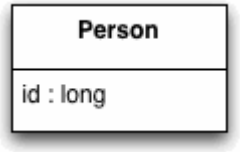
  - транзакционности

  - кэширования

Паттерны объектно-реляционного маппинга

- Identity Field (Поле первичного ключа)
- Foreign Key Mapping (Маппинг внешнего ключа)
- Association Table Mapping (Маппинг таблицы связи)
- Serialized LOB (Сериализованный LOB)
- Lazy Load (Ленивая загрузка)
- Identity Map (Карта присутствия)
- Unit of Work (Единица работы)

Identity Field (Поле первичного ключа)



+

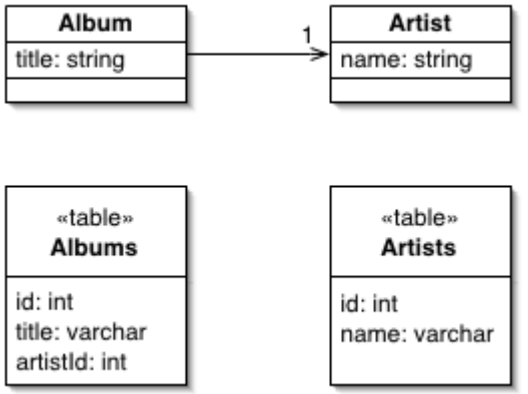
более быстрая выборка данных, чем при использовании композитного ключа

-

новое поле

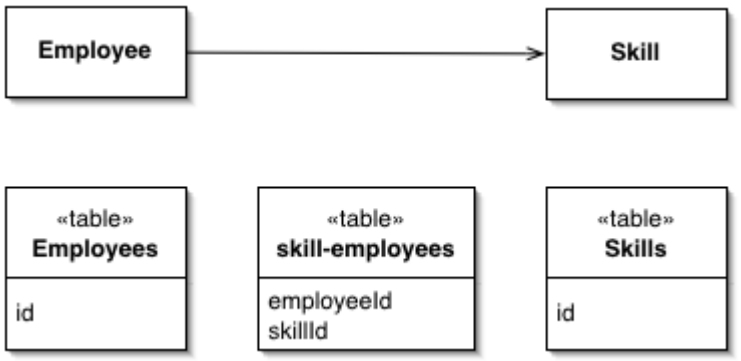
необходимость ограничечения уникальности для композитного ключа

Foreign Key Mapping (Маппинг внешнего ключа)



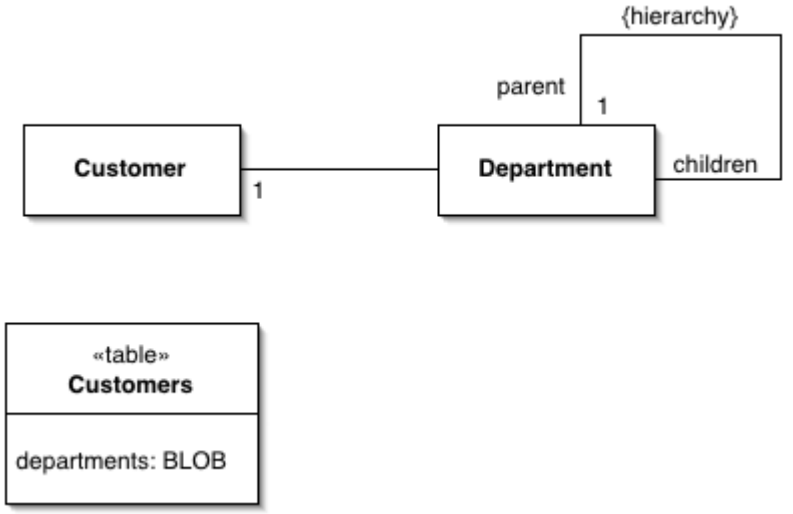
- Ссылка на объект → поле внешнего ключа

Association Table Mapping (Маппинг таблицы связи)



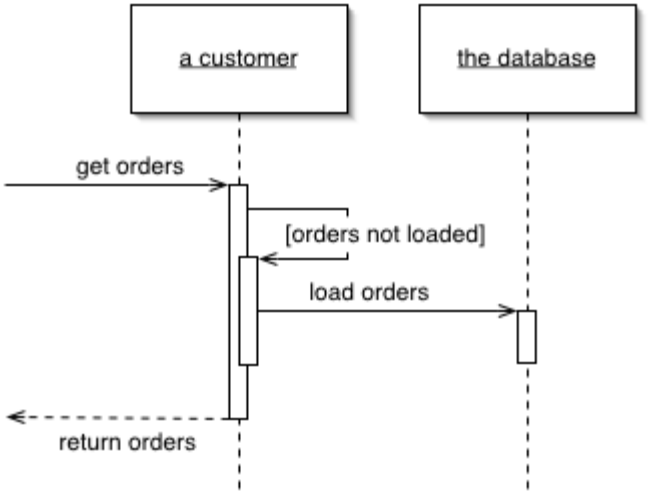
- Связь много ко многим → таблица связи

Serialized LOB (Сериализованный LOB)



- Хранение больших объектов как массив байт

Lazy Load (Ленивая загрузка)



- Загрузка тяжелых объектов происходит по требованию

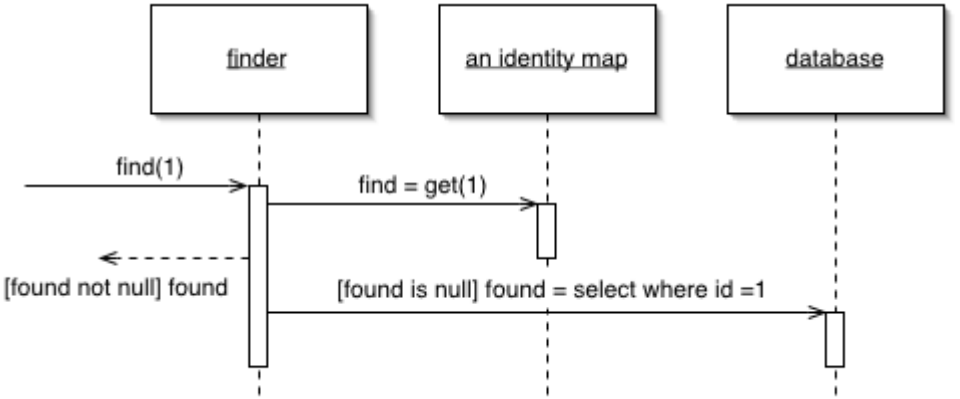
  +

  не нужно сразу загружать связанный объект

  -

  требуется дополнительный запрос

Identity Map (Карта присутствия)



- Кэширование персистентных объектов

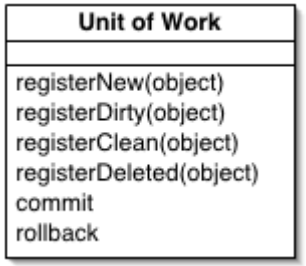
  +

  не нужно выполнять дополнительных запросов, если объект присуствует в кэше

  -

  усложняется логика работы в менеджере персистентности

Unit of Work (Единица работы)



- Поддержка работы с объектами в пределах транзакции

  +

  позволяет не делать лишних запросов

  -

  усложняется логика работы в менеджере персистентности

ORM в проекте фильмов

```
@Entity
public class Film {
    @Id
    @SequenceGenerator(name="film_id_seq", sequenceName="film_id_seq")
    private int id;

    private String name;
    private int year;
    private String genre;

    @ManyToOne
    @JoinColumn(name = "factory")
    private Factory factory;

    @ManyToMany
    @JoinTable(name = "filmstar", joinColumns = @JoinColumn(name = "film"), inverseJoinColumns = @JoinColumn(name =
"star"))
    private Set<Man> stars;

    @ManyToOne
    @JoinColumn(name = "producer")
    private Man producer;

    private String description;

    @Lob
    @Basic(fetch = LAZY)
    @Type(type="org.hibernate.type.BinaryType")
    private byte[] image;
}
```

Identity Field

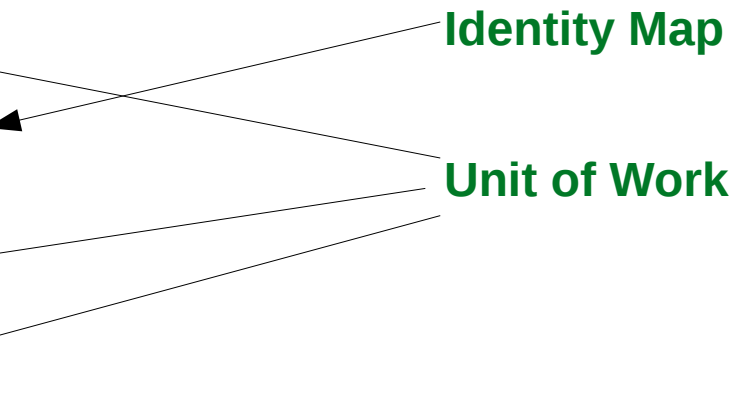
Foreign Key Mapping

Association Table Mapping

Serialized LOB
Lazy Load

ORM в проекте фильмов

```
EntityManager em = /* Получение EntityManager */

Film film = new Film();
/* Заполнение полей через Getters и Setters */

em.persist(film);

film = em.find(Film.class, id);
film.setName("Other name");

em.merge(film);

em.remove(film);
```



Итого

- Мы изучили предметную область и требования к сайту о кинофильмах
- Составили доменную модель
- Создали ER-проект
- Преобразовали его в реляционную модель
- Создали объектную модель Java
- Изучили способы работы с базой данных
- Изучили паттерны работы с базой данных
- Реализовали сайт о кинофильмах с использованием ORM

Спасибо за внимание!