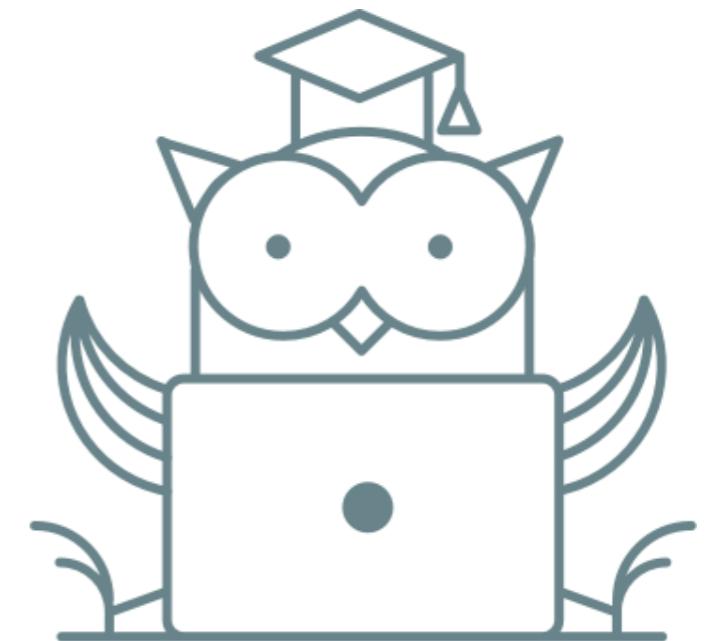


O ·Τ· U S

ОНЛАЙН-ОБРАЗОВАНИЕ

06 – DAO на Spring JDBC

Дворжецкий Юрий



Как меня слышно && видно?



Если нет – напишите, если слышите – смайлик в чат.



Цели:

- Реляционные БД, SQL, JDBC, паттерн DAO (наконец-то)
- Познакомимся с Embedded RDBMS
- Spring JDBC
- Spring Boot Starter JDBC



Цели:

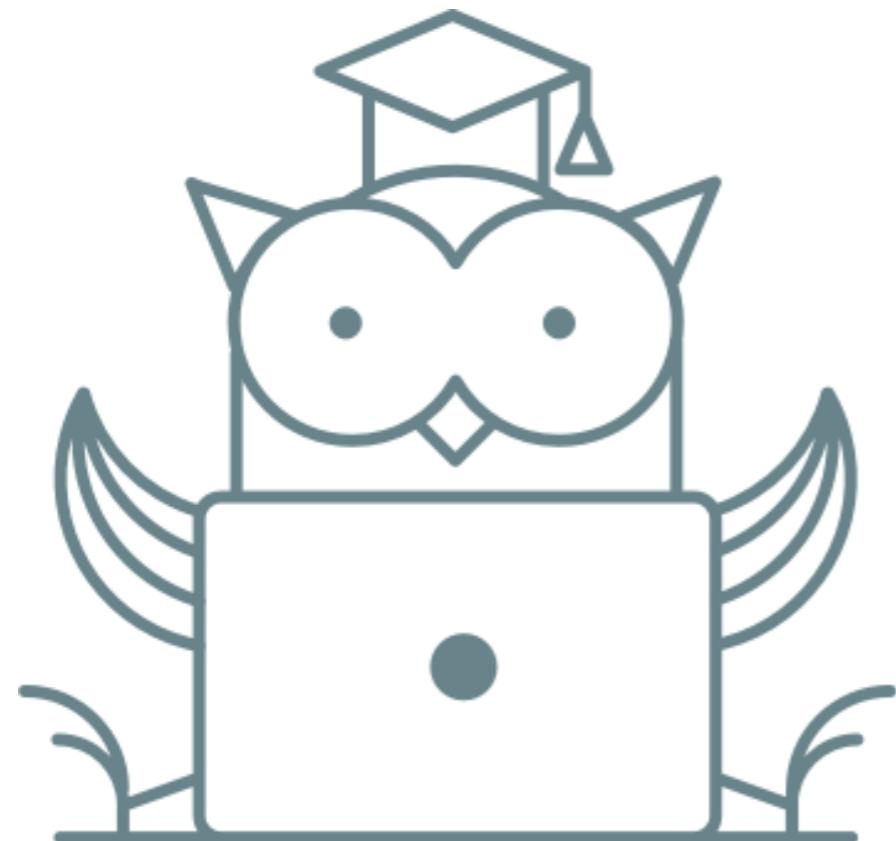
- Сегодня не очень жесть.
- Но домашка крутая
- Материал у Вас останется
- Подробности можете пересмотреть и в документации
- Но основное лучше помнить



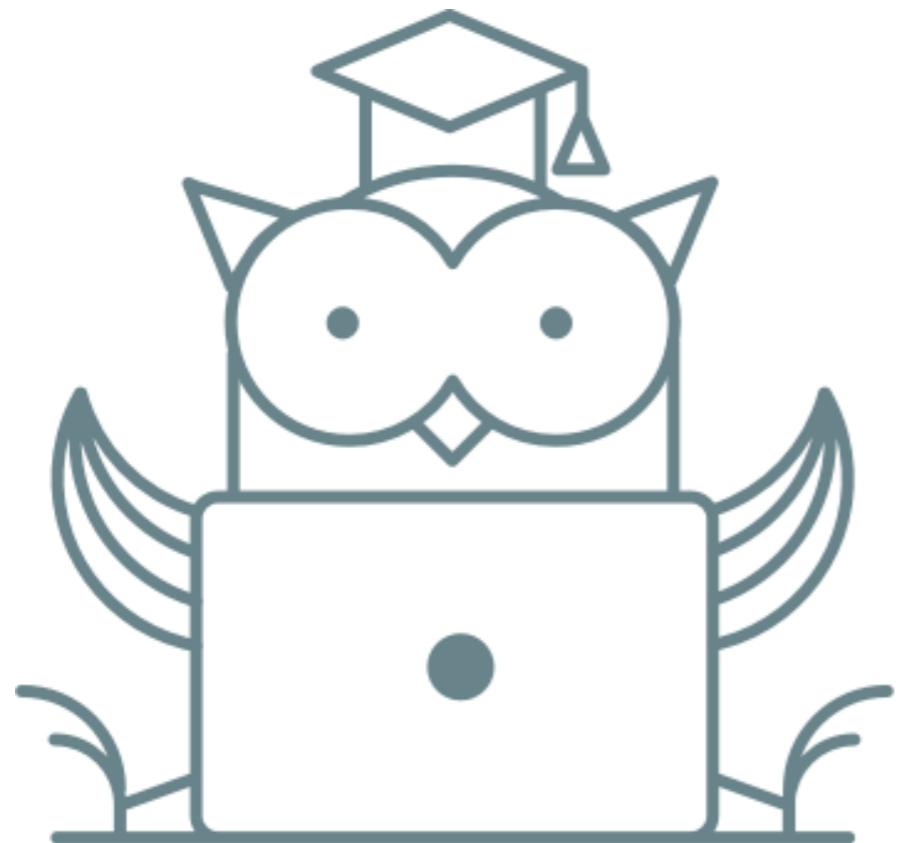
Совсем чуть-чуть орг.вопросов:

- Ваши работы нравятся всё больше и больше

- Чуть-чуть задержка, но по срокам успеваем.
- Сегодня новое ДЗ – ждать не надо.
- Ветку можете создавать от ветки, как и PR
- Со всех отзыв



Поехали?



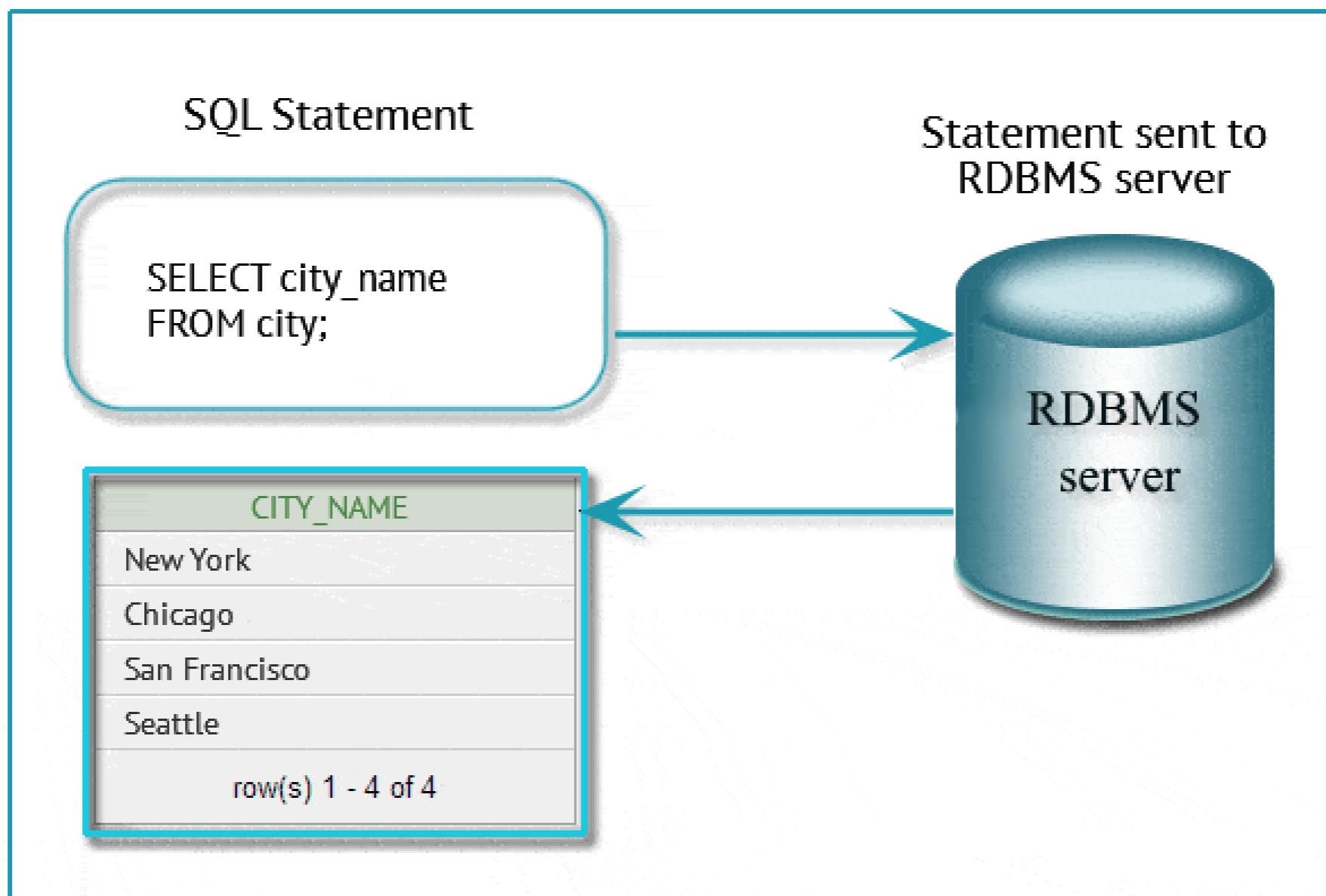
Реляционные Бд
SQL
JDBC

Релационные БД (rdb, rdbms)

- Релационные БД хранят данные в виде таблиц.
- Каждая запись – строчка
- Поле таблицы - ячейка
- Таблицы ещё друг-с-другом связаны Primary Key, Foreign Key и другие навороты



Реляционные БД и SQL



SQL

- Раньше он был SEQUEL – Structured English Query Lanugage. Теперь он просто SQL но остался английским.
- Изначально предполагалось, что любой человек (менеджер), знающий английский – может написать запрос (нет)
- Для каждой БД – свой диалект SQL – они иногда очень сильно различаются.
- Есть общая стандартизированная часть – ANSI SQL, но даже она не во всех БД работает.

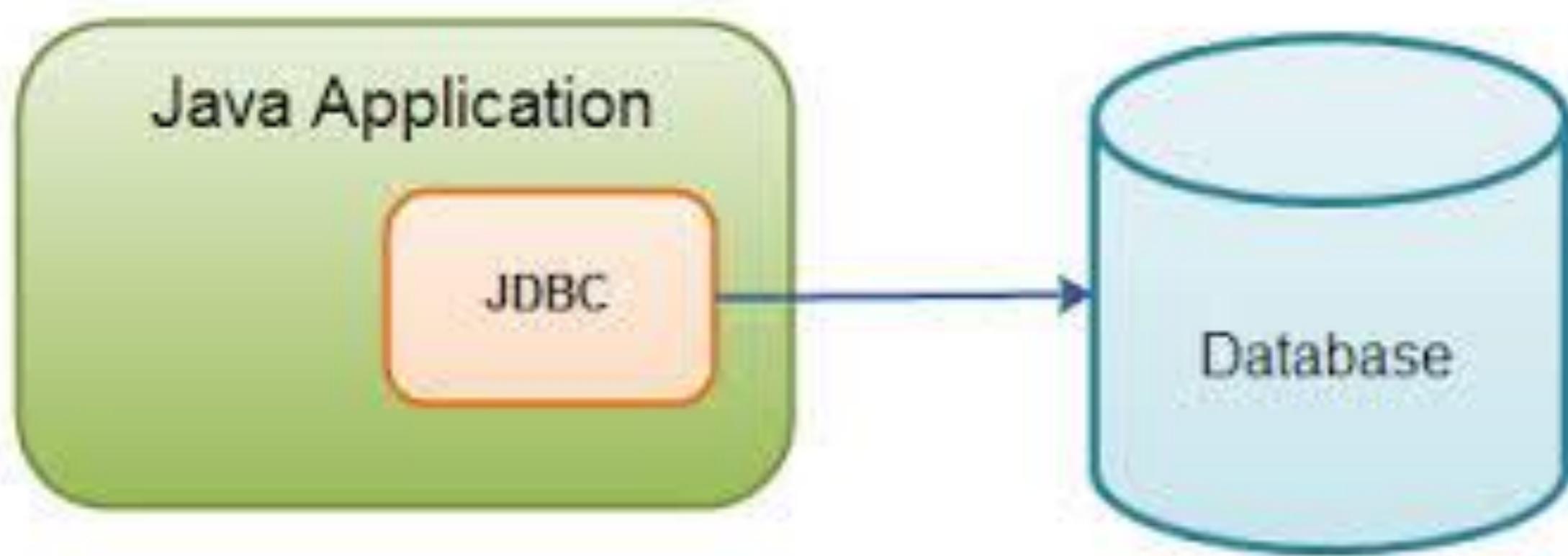
SQL

- **DDL** (Data definition language) – CREATE TABLE
- DML (Data manipulation language) – SELECT, INSERT, DELETE,...
- DCL (Data control language) – DENY, GRANT,...
- DTL (Data transaction language) – COMMIT,...
- PL/SQL (procedural language) – IF, LOOP, WHILE

SQL

- SQL Вы можете писать в клиенте БД
- А можете в приложении
- Эти запросы отличаются. Обычно в приложении пишут запросы с параметрами (и их как-то нужно ещё передать БД)
- Ну и в Java-приложении нужна какая-то технология
- Она называется JDBC

JDBC

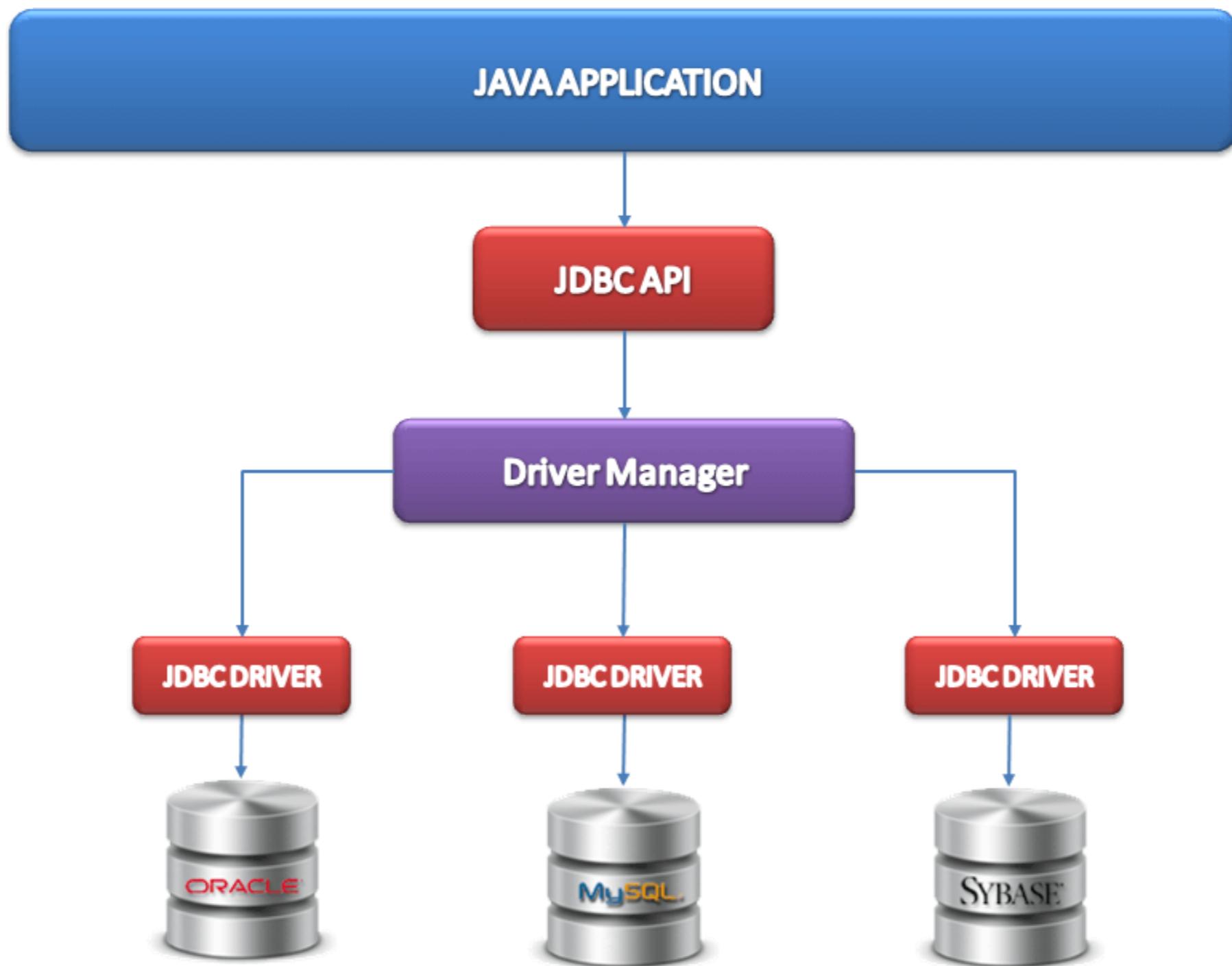


JDBC

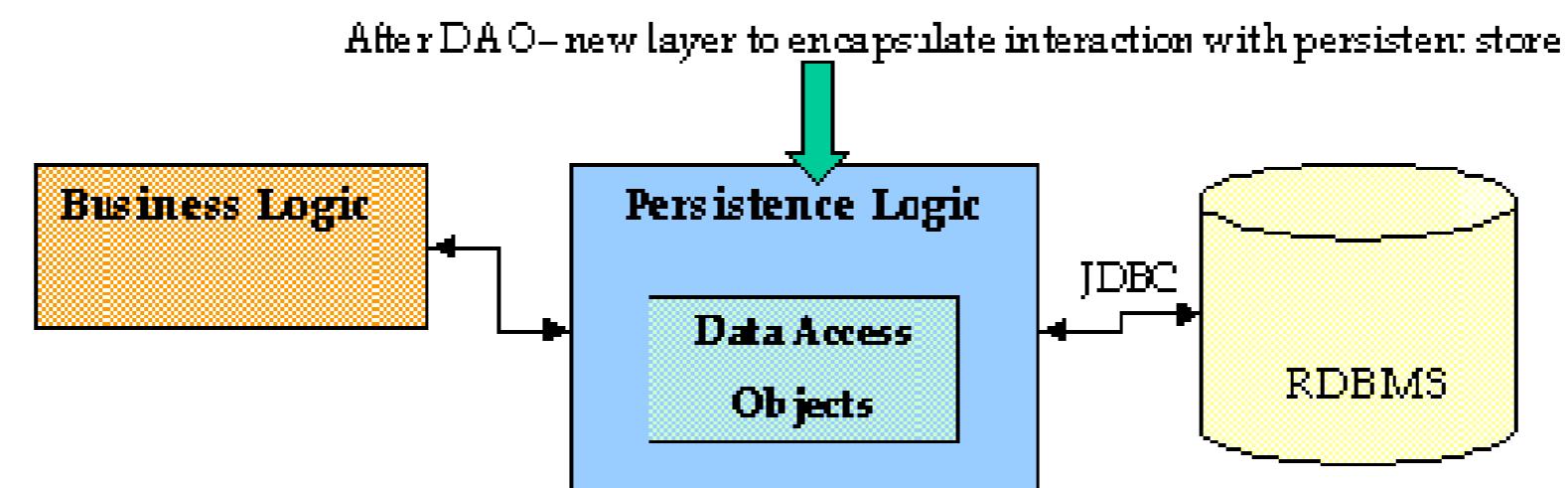
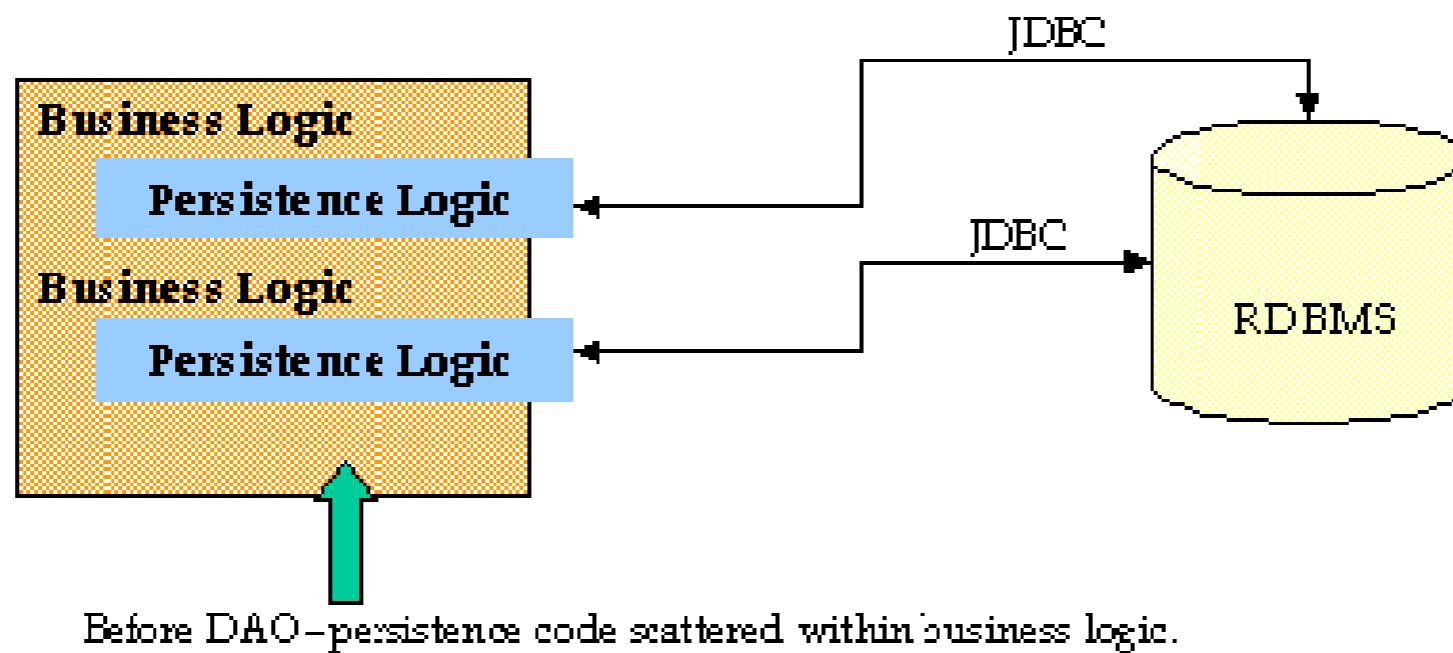
- JDBC хитрее, чем кажется
- Т.к. эта общая оболочка – то можно абстрагироваться от БД
- Для этого используется такое понятие как драйвер, драйвер выпускают разработчики БД
- Но SQL для каждой БД – свой

P.S. Код JDBC мы писать не будем.

JDBC



DAO



DAO

- DAO – Data access object
- Доп.прослойка между бизнесом и БД
- Обычно пишется для каждой БД отдельно
- Ну и соответствует бизнес-сущностям
- В терминах ORM её называют репозиторием.

DAO

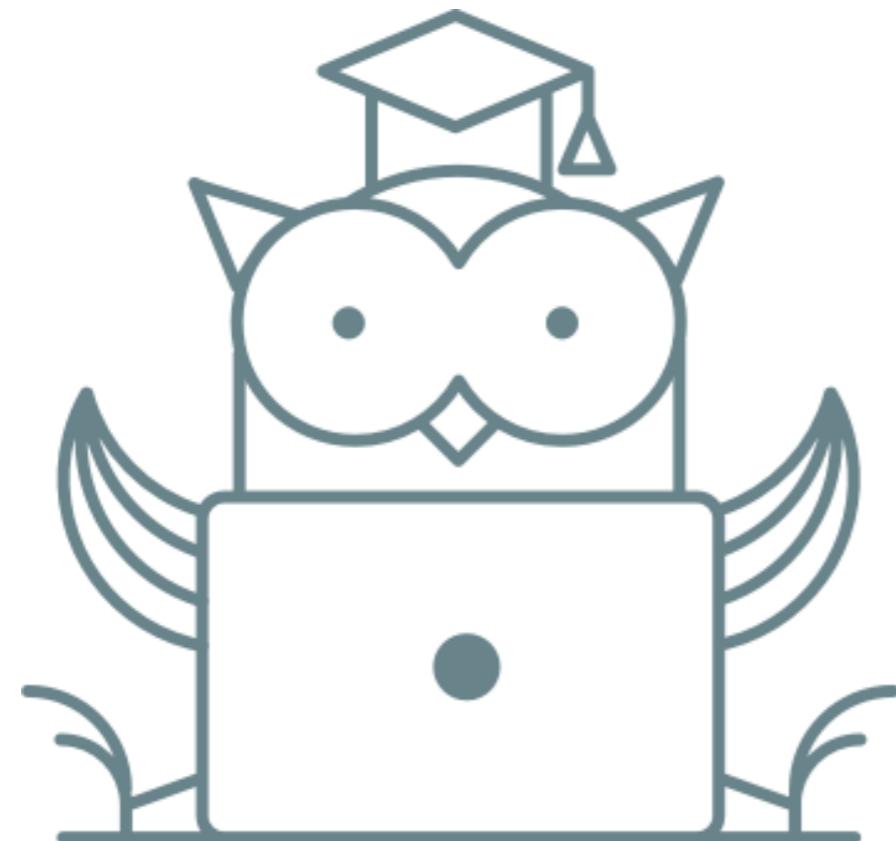
```
public interface PersonDao {  
    int count();  
    void insert(Person person);  
    Person getById(int id);  
    List<Person> getAll();  
}
```

План подключения к БД

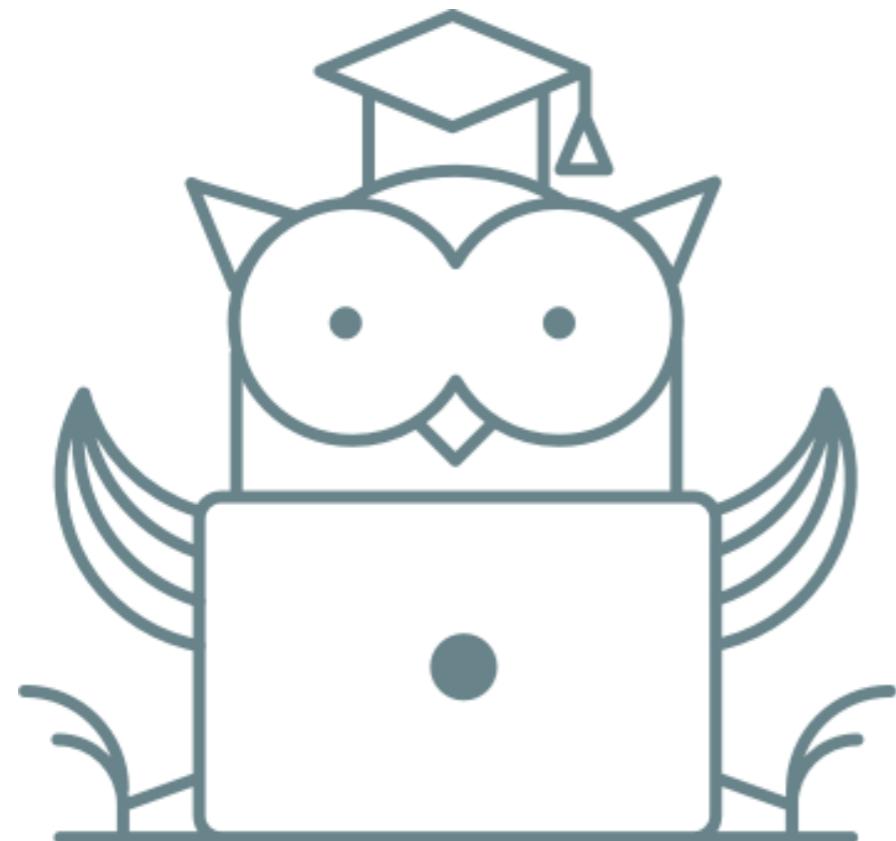
- Чтобы подключиться к БД, необходимо:
- Иметь доступ к БД (url, login/pass)
- JDBC и так есть в Java
- А вот драйвер добавить надо в зависимости
- Пишем DAO, с SQL запросами в строчку
- И написать код JDBC, подключающийся к БД и отправляющий данный запрос

ИТОГО

- Прошли что такое RDB и RDBMS
- Надеюсь узнали что-то новое про SQL
- Узнали что есть JDBC
- Как устроено JDBC API



Вопросы?



Embedded RDB

О Бд

- Бд – ооочень сложное ПО
- Ставить локально нужно
- И в действительности занимают не один сервер, а несколько – кластер
- А что делать если не хочу, нет ресурсов, не удобно?

Embedded DB

Есть 3 популярных embedded DB для Java:

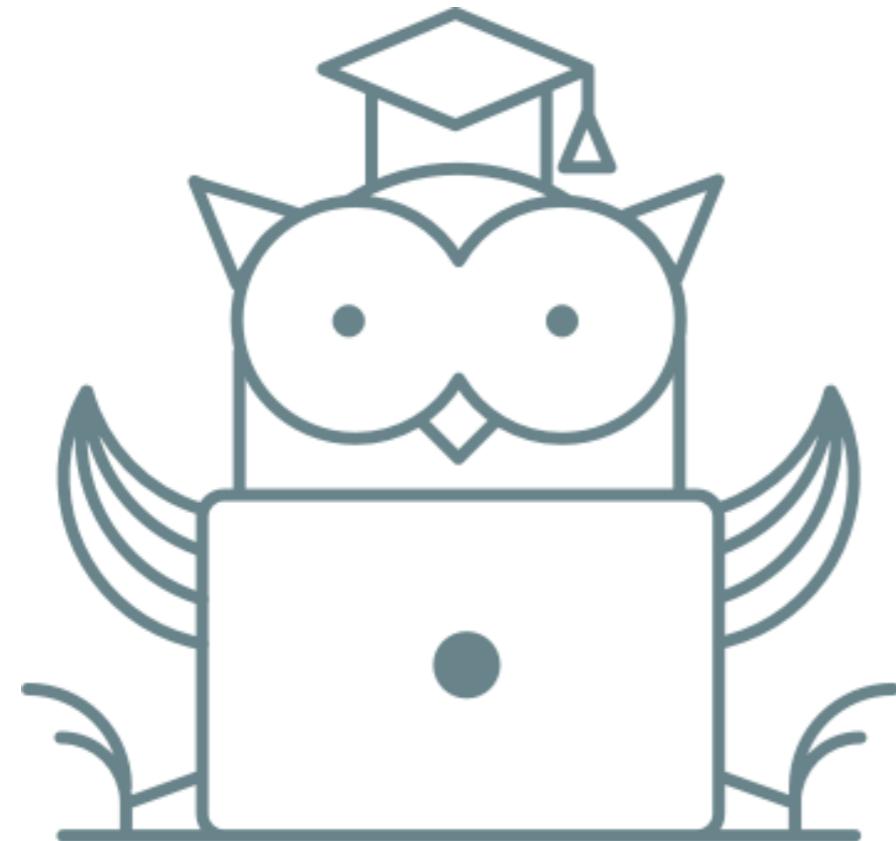
- H2 – клёвая Web-консоль
- HSQLDB – клёвая Swing-консоль
- Derby – гвоорят просто клёвая

Упражнение 1

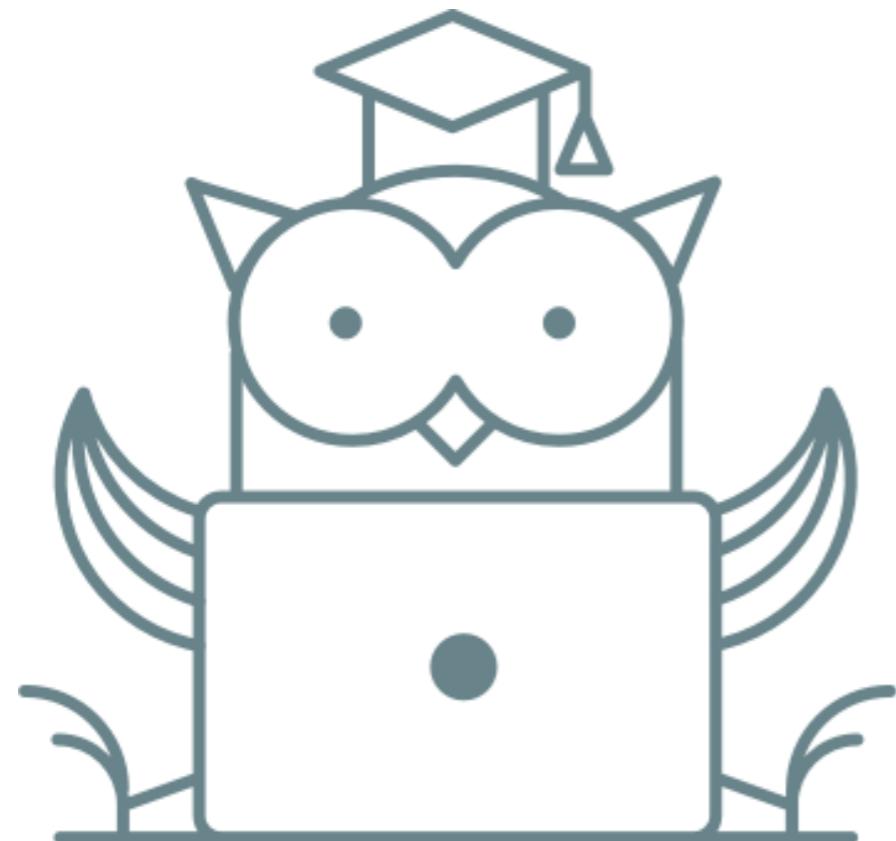
- Пробуем h2
- <https://github.com/ydvorzhetskiy/spring-framework-06>

Упражнение 2

- Создайте таблицу TEST
- Добавьте туда запись
- Сделаёте select *
- Подсказка: есть ссылка внизу.



Вопросы?



Spring JDBC

Plain JDBC

- Чистый JDBC – достаточно старая технология
- И неудобная
- Пользоваться ей напрямую – целый ад

Plain JDBC vs. Spring JDBC

```
final Connection connection = ds.getConnection();
try {
    final Statement statement = connection.createStatement();
    try {
        final ResultSet resultSet = statement.executeQuery("SELECT COUNT(*) FROM Orders");
        try {
            resultSet.next();
            final int c = resultSet.getInt(1);
        } finally {
            resultSet.close();
        }
    } finally {
        statement.close();
    }
} finally {
    connection.close();
}
```

```
int c = new JdbcTemplate(ds).queryForInt("SELECT COUNT(*) FROM Orders");
```

Минусы Plain JDBC

- SQL Exception – checked и одно на все ошибки
- Нужно открывать/закрывать
- Дурацкий PreparedStatement

Spring JDBC

- **javax.sql.DataSource** – управляет подключениями;
- **JdbcTemplate** – центральный класс для выполнения запросов;
- **RowMapper** – маппит строчку БД на объект;
- **JdbcDaoSupport** – Немного упрощает конфигурирование;

DataSource

- Часть спецификации JDBC
- Подключаться к БД можно не только через DataSource
- Но Spring подключается через DataSource
- DataSource позволяет абстрагироваться от connection-ов и пулов
- Объект DataSource требуется создать самостоятельно или получить откуда-нибудь (JNDI, от Spring Boot)

DataSource

```
@Bean
public DataSource dataSource(){
    DriverManagerDataSource ds = new DriverManagerDataSource();
    ds.setDriverClassName("com.mysql.jdbc.Driver");
    ds.setUrl("jdbc:mysql://localhost:3306/gene");
    ds.setUsername("root");
    ds.setPassword("root");
    return ds;
}
```

JdbcTemplate

JdbcTemplate – это главный класс в org.springframework.jdbc.core:

- Выполняет SQL-запросы;
- Итерирует по результатам;
- Ловит JDBC исключения;

Для работы ему необходимы:

- DataSource;
- RowMapper;
- Собственно, SQL-запрос;

JdbcTemplate

- JdbcTemplate - threadsafe;
- Может быть сконфигурирован однажды и использоваться несколькими DAO;
- А можно создавать несколько на один DataSource
- DataSource требуется для создания JdbcTemplate;
- Обычно DataSource передаётся DAO а потом в JdbcTemplate, но проще тестировать, когда передаётся сразу JdbcTemplate;
- JdbcOperations – интерфейс для JdbcTemplate – и лучше использовать его

JdbcTemplate

```
12  @Repository
13  public class PersonDaoJdbc implements PersonDao {
14
15      ... private final JdbcOperations jdbc;
16
17      public PersonDaoJdbc(JdbcOperations jdbcOperations) {
18          ... jdbc = jdbcOperations;
19      }
20
21      @Override
22      public int count() {
23          ... return jdbc.queryForObject(S: "select count(*) from persons", Integer.class);
24      }
25  }
```

Упражнение

Добавить в PersonDaoJdbc JdbcOperations

Метод count в PersonDAO



query*

```
    @Override
    public Person getById(int id) {
        return jdbc.queryForObject("select * from persons where id = ?",
            new Object[] {id}, new PersonMapper());
    }

    @Override
    public List<Person> getAll() {
        return jdbc.queryForList("select * from persons",
            Person.class, new PersonMapper());
    }
```

JdbcTemplate: insert, update, delete

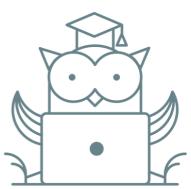
```
@Override  
public void insert(Person person) {  
    jdbc.update( s: "insert into persons (id, `name`) values (?, ?)" ,  
                person.getId(), person.getName() );  
}
```

JdbcTemplate: другие SQL запросы

```
public PersonDaoJdbc (JdbcOperations jdbcTemplate) {  
    . . .  
    jdbcTemplate = jdbcTemplate;  
    . . .  
    jdbcTemplate.execute (sql: "CREATE TABLE PERSONS (ID INT PRIMARY KEY, NAME VARCHAR(255))");  
}
```

Упражнение

Метод `insert` в `PersonDAO`



RowMapper

```
private static class PersonMapper implements RowMapper<Person> {  
  
    @Override  
    public Person mapRow(ResultSet resultSet, int i) throws SOLEException {  
        int id = resultSet.getInt( columnLabel: "id" );  
        String name = resultSet.getString( columnLabel: "name" );  
        return new Person(id, name);  
    }  
}
```

RowMapper

- Интерфейс из org.springframework.jdbc.core;
- Он описывает маппинг строчек ResultSet в конкретные объекты;
- Используется в методе query() JdbcTemplate-а или в результате вызова хранимой процедуры;
- Обычно сохраняется в поле DAO, если он stateless

Что будет 0, 1, 2 person-ах?

```
    @Override
    public Person getById(int id) {
        return jdbc.queryForObject("select * from persons where id = ?",
        new Object[] {id}, new PersonMapper());
    }

    @Override
    public List<Person> getAll() {
        return jdbc.queryForList("select * from persons",
        Person.class, new PersonMapper());
    }
```

Упражнение

RowMapper

Метод getById в PersonDAOJdbc

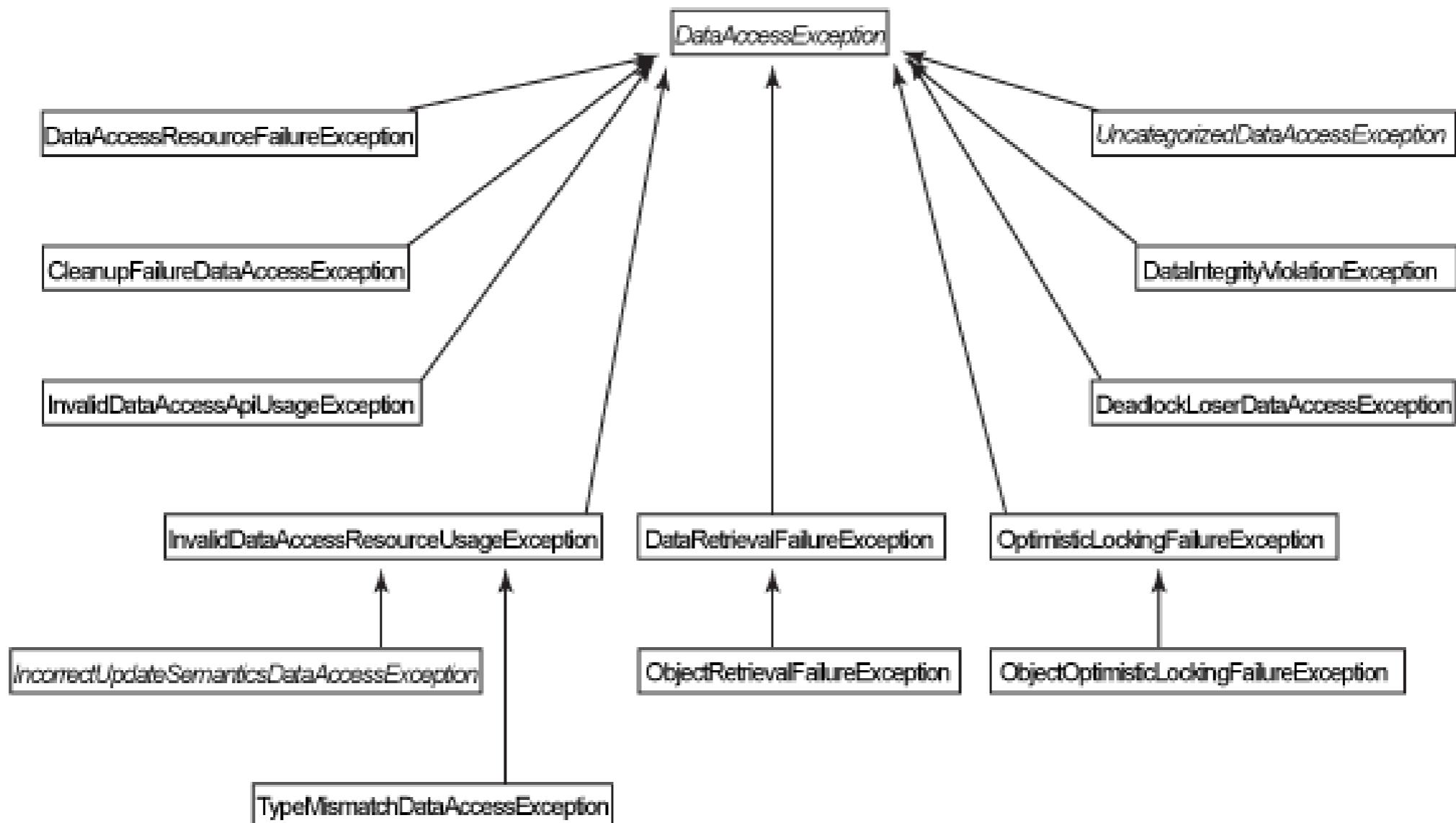
Метод getAll в PersonDAOJdbc



Обработка исключений

- Spring преобразует исключения, зависящие от платформы, такие как SQLException в иерархию собственных исключений, DataAccessException – корень этой иерархии;
- Это runtime-исключения (!)

Иерархия исключений



NamedParameterJdbcTemplate

```
@Repository  
public class PersonDaoJdbc implements PersonDao {  
  
    private final NamedParameterJdbcTemplate jdbcTemplate;  
  
    public PersonDaoJdbc(NamedParameterJdbcTemplate jdbcTemplate) {  
        this.jdbcTemplate = jdbcTemplate;  
    }  
}
```

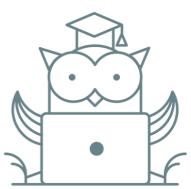
```
@Override  
public Person getById(int id) {  
    final HashMap<String, Object> params = new HashMap<>( initialCapacity: 1 );  
    params.put("id", id);  
    return jdbcTemplate.queryForObject(  
        sql: "select * from persons where id = :id",  
        params, new PersonMapper()  
    );  
}
```

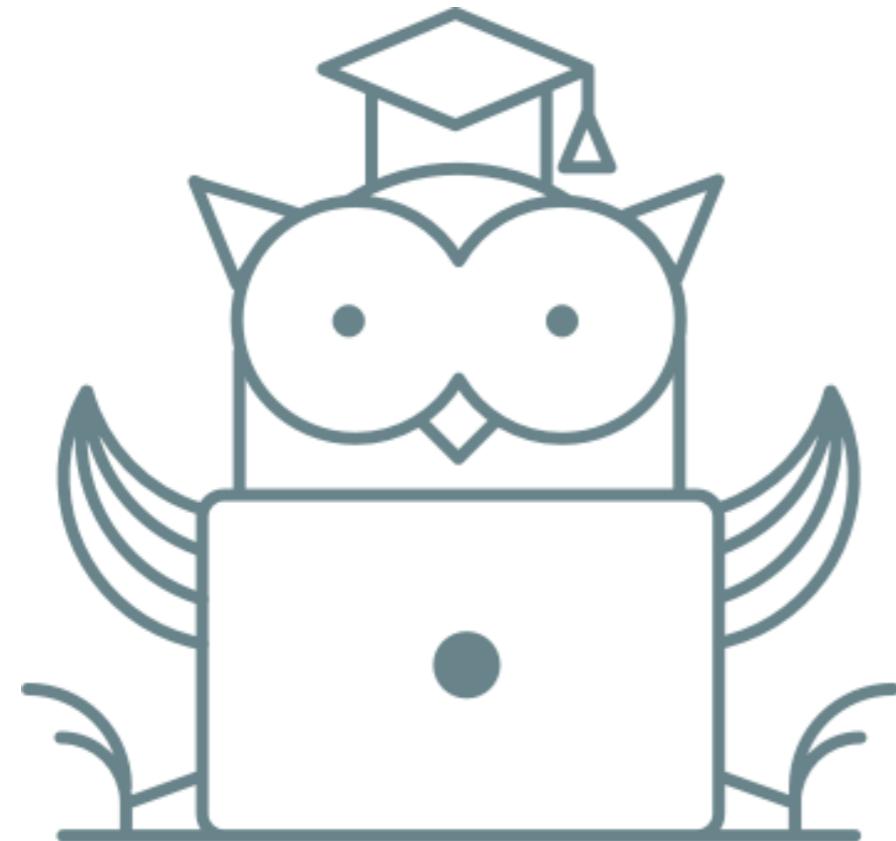
NamedParameterJdbcTemplate

- Использовать только так
- Дополнительный параметр –развязка
- NamedParameterOperations – интерфейс – используйте его (можно мокать)

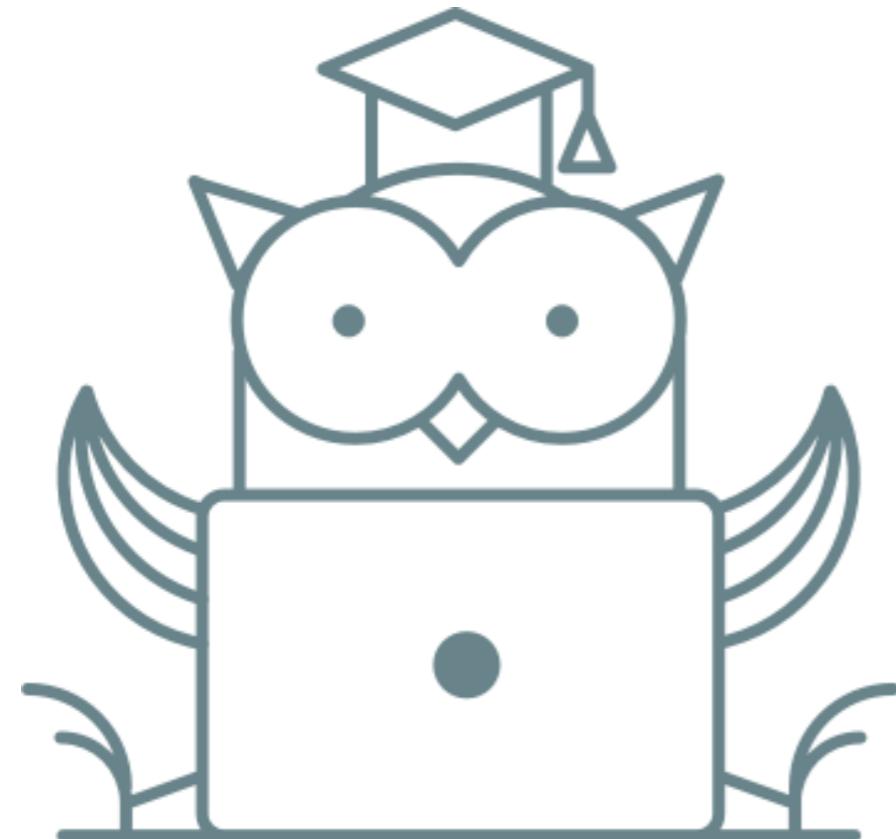
Упражнение*

Методы `getById` и `delete` на named-





Вопросы?



Spring Boot Starter JDBC

DataSource

```
spring.datasource.url=jdbc:mysql://localhost/test  
spring.datasource.username=dbuser  
spring.datasource.password=dbpass  
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
```

DataSource

- Если используется H2, HSQLDB или Derby, то ничего писать совсем не надо
- А ещё создаёт JdbcTemplate
- И JdbcOperationsTemplate

Spring Boot Starter Jdbc

- Подключает транзакционность
- Автоматически загружает файлы schema.sql, data.sql , testdata.sql
- Упражнени: - смотрим в ресурсы



Вопросы?

Домашнее задание

Создать приложение хранящее информацию о книгах в библиотеке

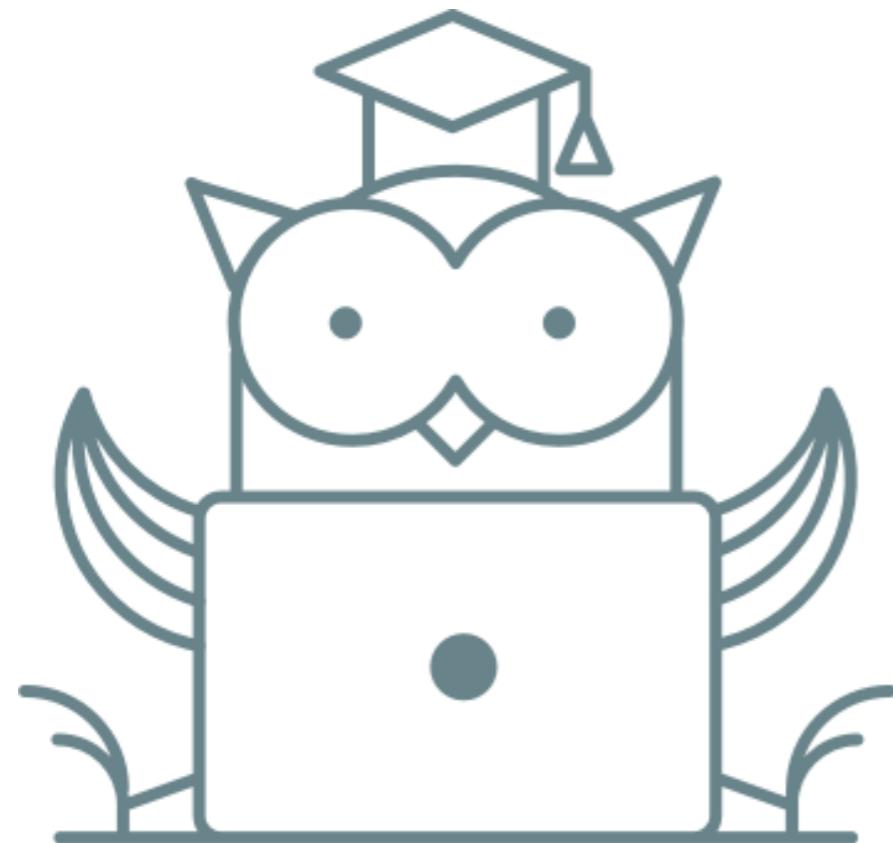
Использовать Spring JDBC и реляционную базу.

Опционально использовать настоящую реляционную БД, но можно использовать H2.

Предусмотреть таблицы авторов, книг и жанров.

Покрыть тестами, насколько это возможно.

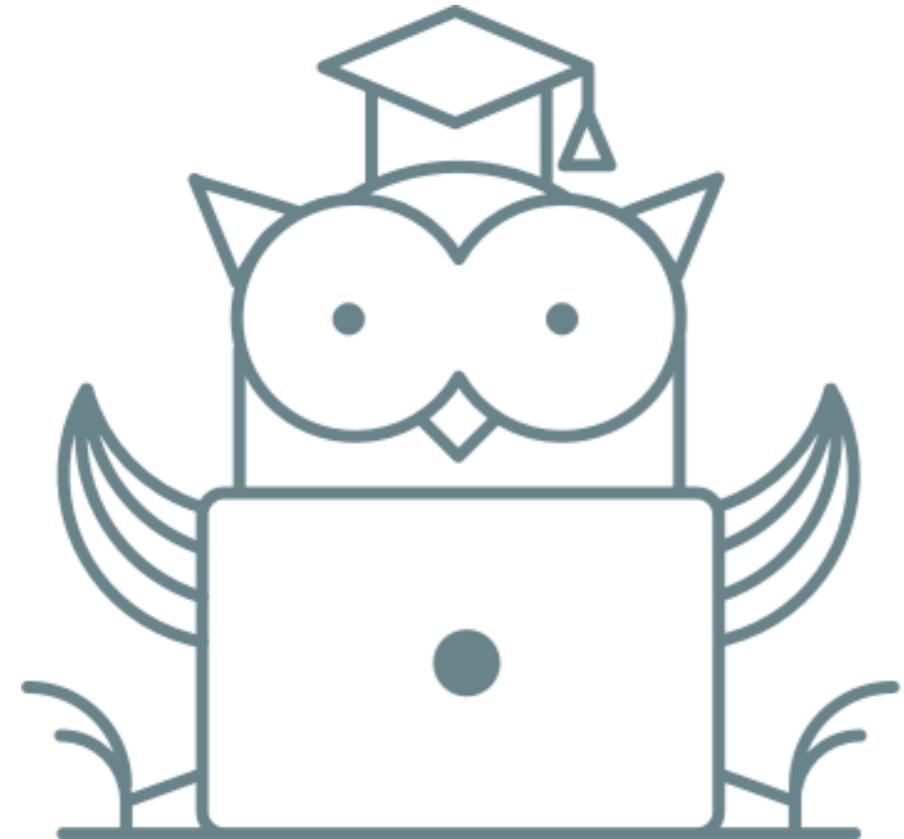




Вопросы?

Пожалуйста, пройдите опрос

<https://otus.ru/polls/1514/>



Спасибо
за внимание!

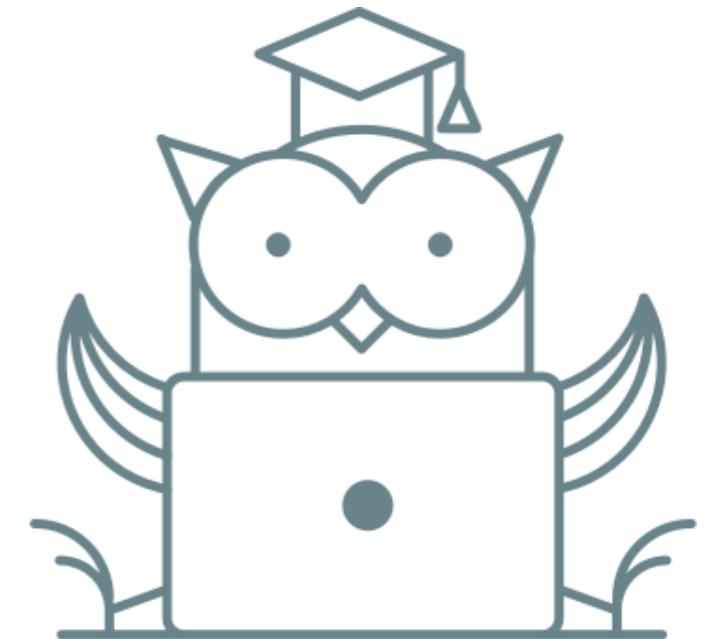
СпрыгбутА Вам!

O ·Τ· U S

ОНЛАЙН-ОБРАЗОВАНИЕ

06 – DAO на Spring JDBC

Дворжецкий Юрий



Как меня слышно && видно?



Если нет – напишите, если слышите – смайлик в чат.



Цели:

- Идеология ORM
- JPA, место Hibernate и *Batis в этом мире



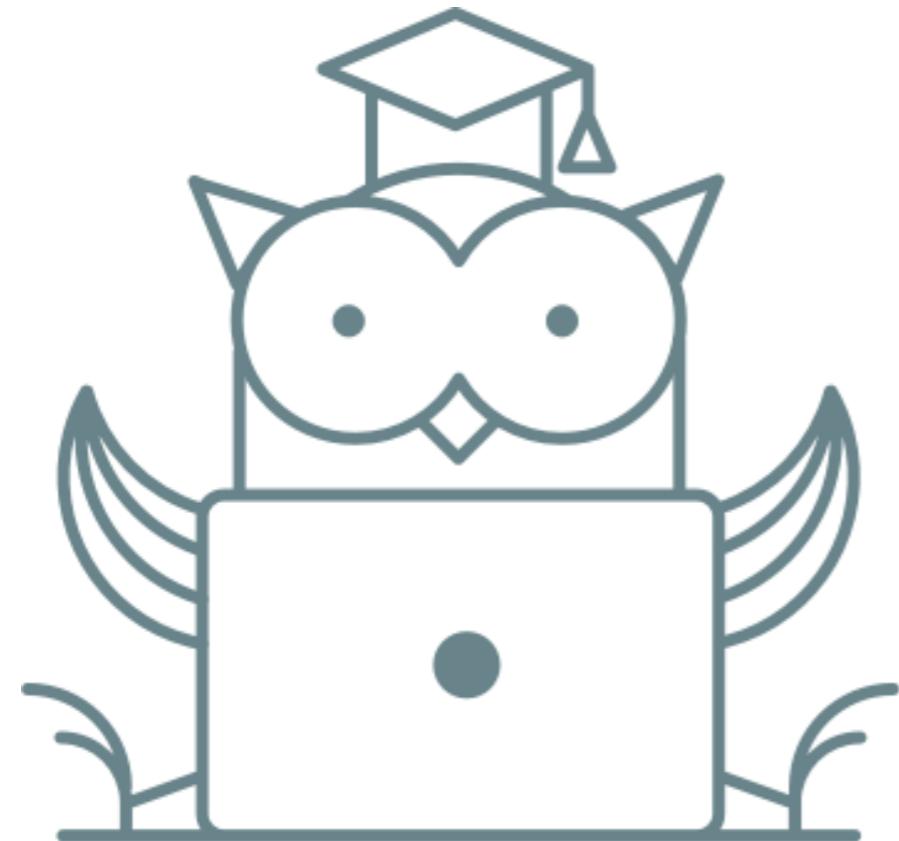
Цели:

- Сегодня не очень жесть, больше теория. Где Spring – узнаем на следующем занятии
- JPA пройдём только на достаточноном уровне
- Часто это всё пишется с документацией, рекомендую знакомиться
- Домашки нет – спокойно доделываем, готовимся к жести.



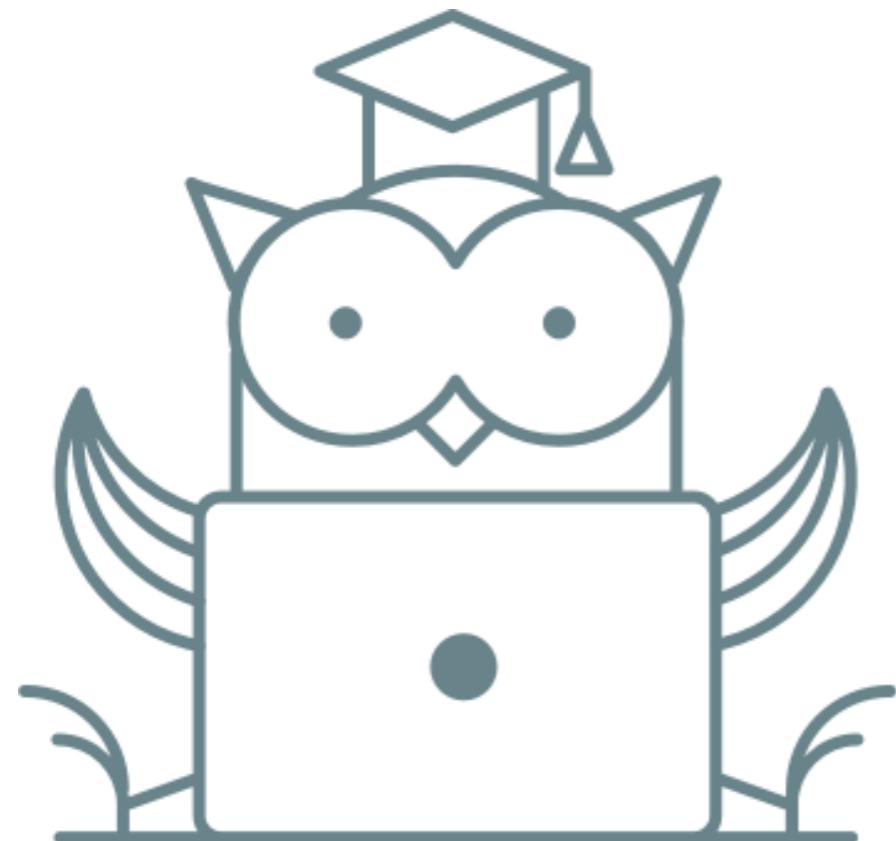
Совсем чуть-чуть орг.вопросов:

- Срок проверки – три дня, стараемся быстрее
- Вопросы задавайте. В сложных ситуациях помогаем. Но не всегда сразу)
- Ветку можете создавать от ветки, как и PR
- Со всех отзыв!



Поехали?

O T U S



ORM

Реляционные БД

- В БД есть таблицы – хранят сущности
- В таблицах есть строчки – сами объекты
- В строчках есть ячейки – типа «поля»

Объекты в Java

- В программе есть классы сущностей
- Есть объекты сущностей
- В конкретном объекте есть реальные поля, хранящие данные

ORM

Хочется автоматическим образом сmapить:

- Классы на таблицы
- Объекты на строчки
- Ячейки на поля

И работать в привычной ООП парадигме

Мы писали RowMapper

```
private static class PersonMapper implements RowMapper<Person> {  
  
    @Override  
    public Person mapRow(ResultSet resultSet, int i) throws SOLEException {  
        int id = resultSet.getInt(columnLabel: "id");  
        String name = resultSet.getString(columnLabel: "name");  
        return new Person(id, name);  
    }  
}
```

ORM

- ORM (Object Relational Mapping) подход как раз про этот маппинг
- ORM-фреймворки как раз этим и занимаются
- Hibernate, *Batis, Eclipse Link – ORM-фреймворки
- JPA – «прослойка», язык маппинга, который поддерживают фреймворки.
- Фреймворки имеют и свой язык маппинга

Пример маппинга

```
7
8     @Entity
9     public class Person {
10
11         @Id
12         @GeneratedValue
13         private int id;
14         private String name;
15
16         @OneToOne
17         private Email email;
18
19         public Person(String name, Email email) {
20             this.name = name;
21             this.email = email;
22         }
23
24         public int getId() {
25             return id;
26         }
27
28         public void setId(int id) {
29             this.id = id;
30         }
31
32         public String getName() {
```

Пример репозитория

```
8
9  @Repository
10 public class PersonRepositoryJdbc implements PersonRepository {
11
12     .... @PersistenceContext
13     .... private EntityManager em;
14
15     .... @Override
16     public Person getById(int id) {
17         .... return em.find(Person.class, id);
18     }  
19 }  
20 |
```

Impedance mismatch

- это такое смешное слово для обозначения несоответствия парадигм
- Классы только кажутся таблицами, а объекты строчками
- Объекты – это, скорее, граф в математическом понимании
- Проблемы несоответствия известны и, собственно, решаются ORM-фреймворком

Impedance mismatch

- Гранулярность (Granularity)
- Наследование (Inheritance)
- Индивидуальность (Identity)
- Ассоциации (Associations)
- Получение данных (Data navigation)

Гранулярность

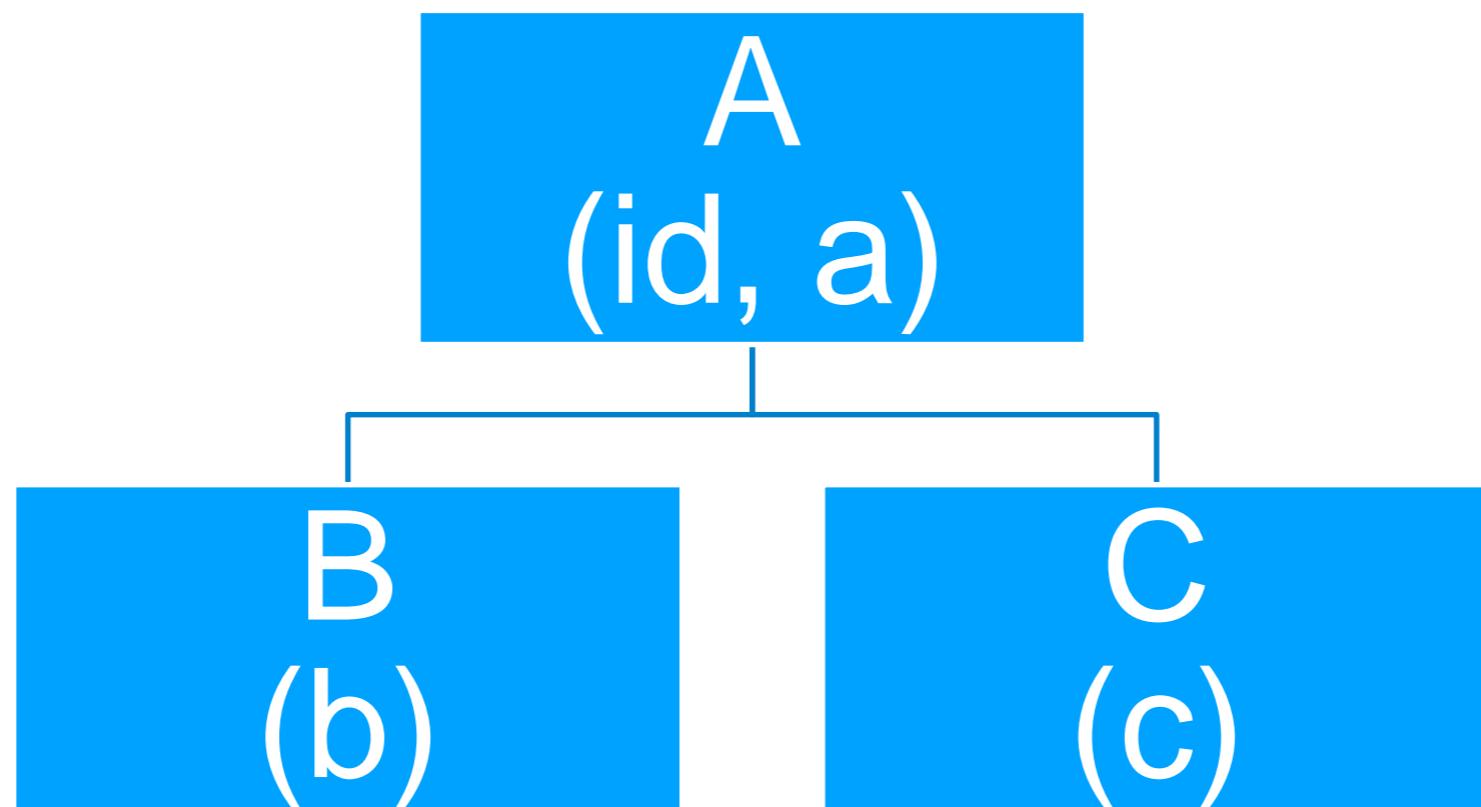
Чем будут поля в таблице?

Сколько таблиц и классов?

```
@Entity  
@Table(name = "users")  
public class User {  
    @Id  
    @GeneratedValue  
    private int id;  
  
    private Email login;  
  
    private String password;  
  
    private Address address;
```

Наследование

Как это реализовать в Бд? (я знаю 5 способов)



Ассоциация

Как это реализовать в БД?

(это основной функционал ORMF)

```
public class Company {  
    ...  
    private List<Person> employees;
```

Получение данных

```
public class Company {  
    private Person director;  
  
    public String getDirectorName () {  
        return director.getName ();  
    }  
}
```

DAO на Spring JDBC

Business Service

Dao

Daolmpl

Spring JDBC

JDBC

DB

DAO ha ORM

Repository

Entity + Mapping

ORM framework (Hibernate)

JDBC

DB

DAO ha ORM + JPA

Repository

Entity + JPA

JPA

Hibernate as JPA Vendor

JDBC

DB

DAO на Spring Data JPA

Repository

Entity + JPA

JPA

Hibernate as JPA Vendor

JDBC

DB

```
package com.concretepage.entity;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
@Entity
@Table(name="farmer")
public class Farmer {
    @Id
    @Column(name="id")
    private int id;
    @Column(name="name")
    private String name;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

Где здесь
SQL ?

DAO ha ORM

Repository

Entity + Mapping

ORM framework (Hibernate)

DB Dialect

JDBC

DB

ORM

- ORM – по сути дела весь SQL – в маппинге Entity
- Entity – обычно на слое домена (бизнеса)
- Поэтому DAO тоже на слое бизнес
- И DAO теперь принято называть репозиторием

Плюсы ORM

- ORM теперь позволяет оперировать бизнес-понятиями и ООП, поля вместо столбцов, объекты вместо таблиц (хотя они могут быть размазаны по разным таблицам)
- ORM теперь позволяет не знать SQL 😊 (нет 😞)
- ORM позволяет абстрагироваться от диалекта SQL (!!!!!)
- А с JPA и от конкретного провайдера JPA. В идеале его можно заменить в Maven.

Плюсы ORM

- Со Spring Boot и Spring Data можно вытворять нереальные вещи
- Тестировать – одно удовольствие, можно тестировать на H2 и выше где угодно
- Тупо меньше кода, со Spring Data -
- Естественная транзакционность

Минусы ORM

- Это магия
- Как и за любую магию приходится платить производительностью
- Производительность – странная штука*
- Нет нормального доступа к нативному SQL и плюшкам Бд

ORM vs JDBC

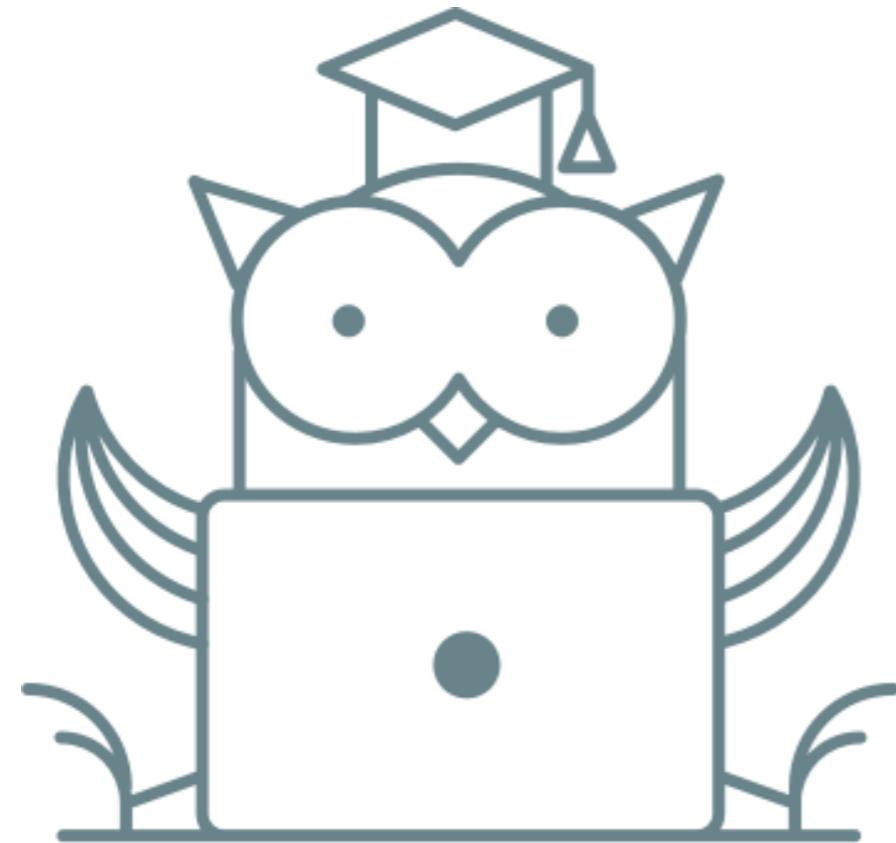
- Нужно решить, что проекту нужно, может это прототип.
- Какая нагрузка (кто говорит, что у них будет большая нагрузка – скорее всего не правы).
- Сильно зависит от числа бизнес-объектов.
- Больше 10-20 взаимосвязанных бизнес объектов – разрабатывать на SQL становится сильно тяжело.
(Связей – квадрат от числа entity)

ORM vs JDBC

- Если приложение действительно (!) под нагрузкой, то лучше использовать чистый JDBC. SQL не такой страшный
- Если много бизнес-объектов – ORM незаменим.
- Из одной парадигмы в другую перейти можно.
- Периодически встречается комбинация

ИТОГО

- Прошли что такое ORM
- Узнали про место Hibernate
- Узнали что есть JPA
- Примерно понятно где выбрать ORM или Plain JDBC (нет)



Вопросы?

O T U S



JPA

План

- Что такое JPA
- @Entity
- @Table
- @Id
- @Column
- @OneToMany, @Many***

JPA

- JPA – прослойка между Вашим кодом и ORM фреймворком
- Позволяет не привязываться к вендору JPA
- Какой-никакой стандарт (со сложной судьбой правда)
- Помните, расширения языка в центре луковицы (но не SQL 😞)
- Содержит аннотации маппинга и кое-какие классы

Пример маппинга

```
7
8     @Entity
9     public class Person {
10
11         @Id
12         @GeneratedValue
13         private int id;
14         private String name;
15
16         @OneToOne
17         private Email email;
18
19         public Person(String name, Email email) {
20             this.name = name;
21             this.email = email;
22         }
23
24         public int getId() {
25             return id;
26         }
27
28         public void setId(int id) {
29             this.id = id;
30         }
31
32         public String getName() {
```

Пример репозитория

```
8
9  @Repository
10 public class PersonRepositoryJdbc implements PersonRepository {
11
12     .... @PersistenceContext
13     .... private EntityManager em;
14
15     .... @Override
16     public Person getById(int id) {
17         .... return em.find(Person.class, id);
18     }  
19 }  
20 |
```

Упражнение 1

- <https://github.com/ydvorzhetskiy/spring-framework-07>
- Смотрим на Person
- Смотрим в h2 console (откуда это?)

@Entity

- Должны быть помечены аннотацией @Entity
- Должны содержать правильные getters|setters|constructors (Proxy)
- Аннотация @Entity – настраиваемая (что там?)
- У каждой @Entity должен быть @Id

@Table

- Откуда взято имя таблицы?
- С @Entity можно применять @Table
- В @Table можно настраивать имя таблицы соответствующей @Entity + дофига координат

@Table

```
  @Entity
  @Table("persons")
public class Person {

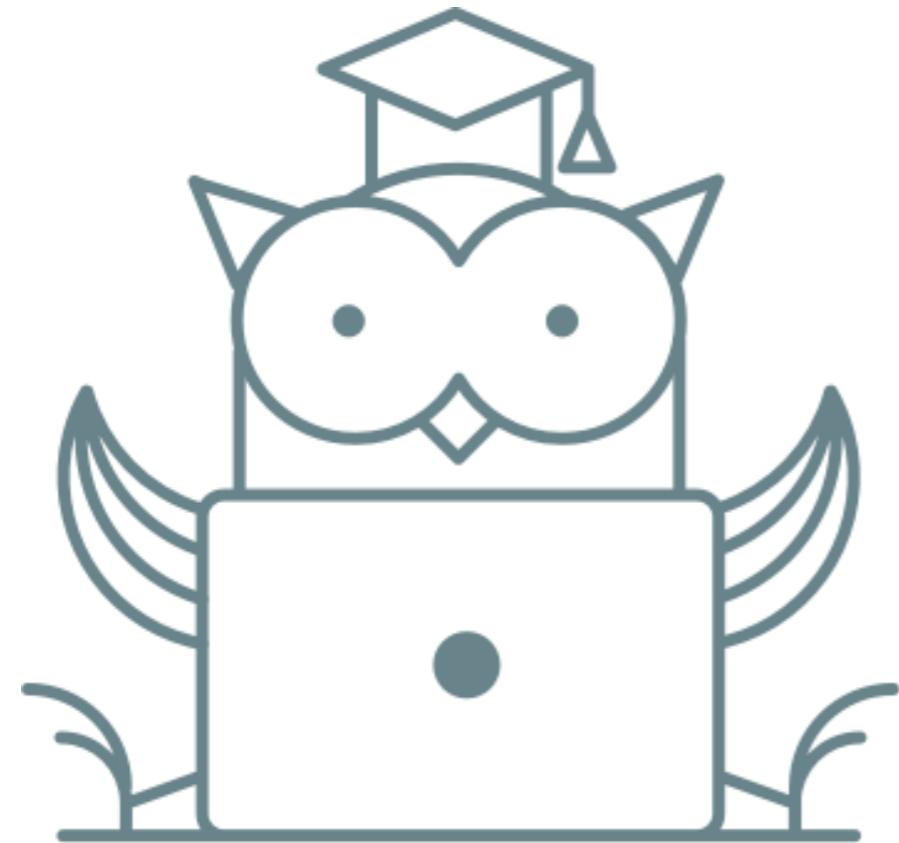
    @Id
    @GeneratedValue
    private int id;

    private String name;

    public Person(String name) {
        this.name = name;
    }
}
```

Упражнение 1.1

- Настроить имя таблицы “persons”



Вопросы?

@Column

- Откуда взято имя колонок?
- На полях можно ставить аннотацию @Column
- Там тьма всего (смотрим, зачем это?)

@Column

```
5  @Entity
6  public class Person {
7
8      @Id
9      @GeneratedValue
10     @Column("person_id")
11     private int id;
12     private String name;
13
14    public Person(String name) {
15        this.name = name;
16    }
17
18    public int getId() {
19        return id;
```

@Column

```
@Entity
public class Person {
    private int id;
    private String name;

    public Person(String name) {
        this.name = name;
    }

    @Id
    @GeneratedValue
    @Column("person_id")
    public int getId() {
        return id;
    }
}
```

Атрибуты на полях/методах

- Всё будет прокси (наследником, скорее всего)
- Если Вы хотите, чтобы реально вызывался метод – ставьте на геттер/сеттер
- Если нет – на поля
- Если аннотации стоят на полях и на сеттерах, то поведение не определено

Упражнение 1.2

- Настроить имя колонок “`person_id`”, например

@Id

- У любого @Entity должен быть @Id
- Это поле может быть @Embeddable

@GeneratedValue

- @Id может генерироваться
- Эту обязанность можно переложить на Hibernate, sequence в БД или таблицу-генераторов
- По умолчанию – стратегию выбирает ORM провайдер (AUTO), а он по своему усмотрению может переложить на БД

@GeneratedValue (sequence)

```
@Entity
public class Person {

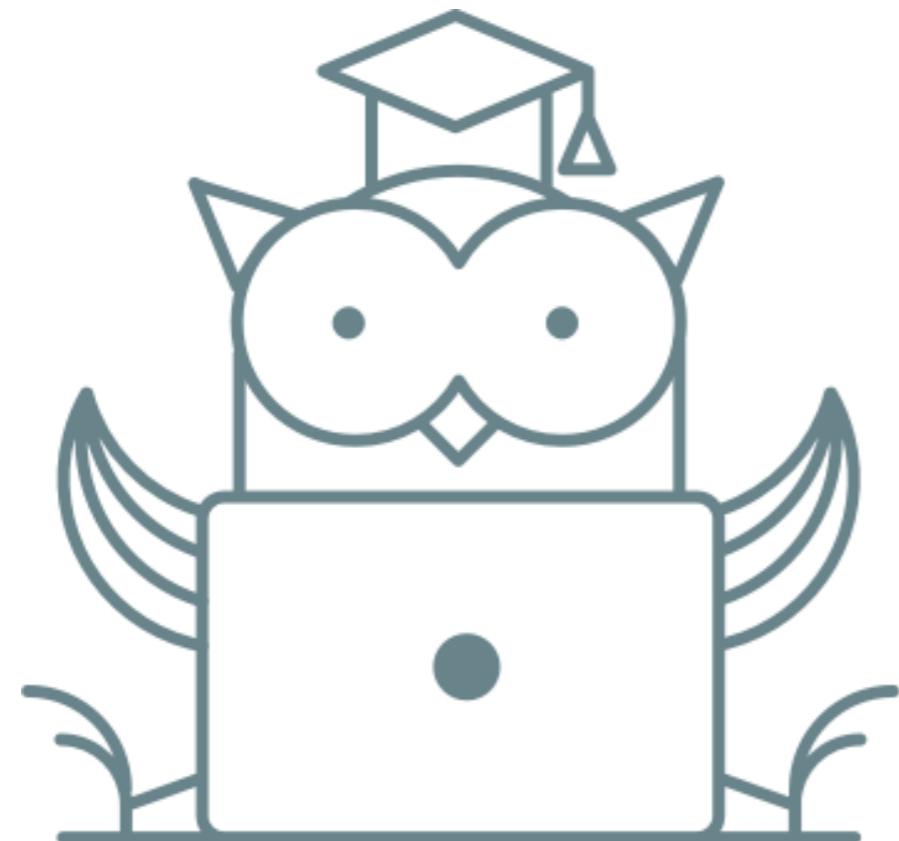
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private int id;
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public int getId() {
        return id;
    }
}
```

Упражнение 2

- Сделайте Email @Entity
- Запустить, посмотреть Бд



Вопросы?

@OneToOne

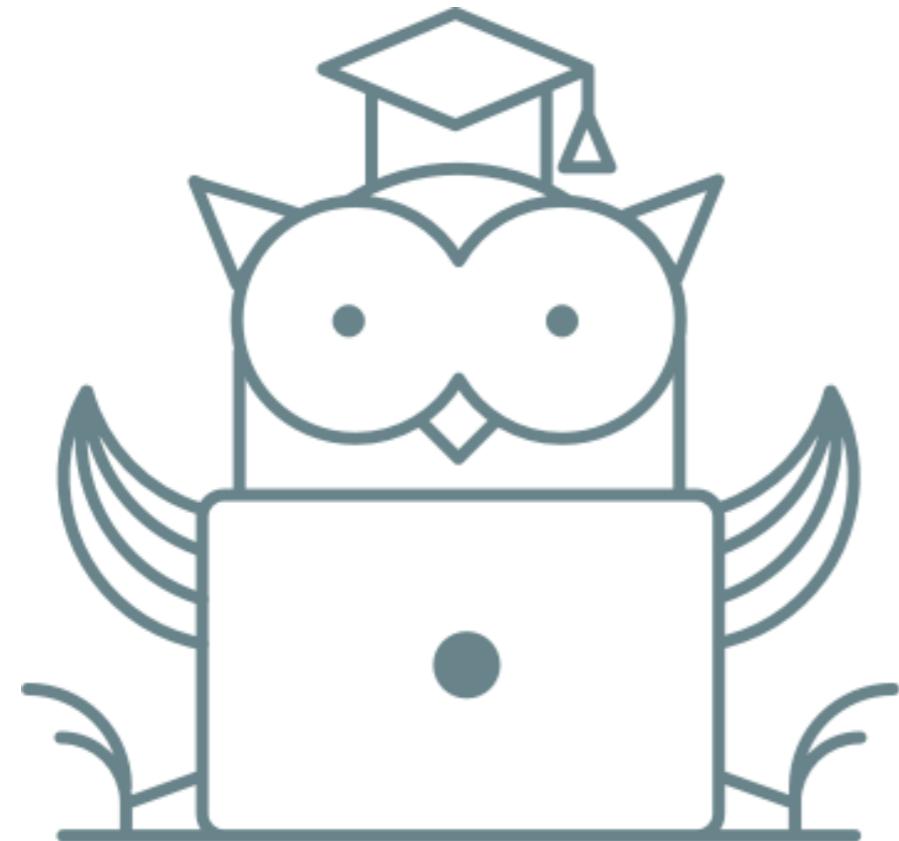
```
3 import javax.persistence.*;
4
5 @Entity
6 public class Person {
7
8     @Id
9     @GeneratedValue
10    private int id;
11    private String name;
12
13    @OneToOne
14    private Email email;
15
16    public Person(String name, Email email) {
17        this.name = name;
18        this.email = email;
19    }
20}
```

@ManyToOne

```
3 import javax.persistence.*;
4
5 @Entity
6 public class Person {
7
8     @Id
9     @GeneratedValue
10    private int id;
11    private String name;
12
13    @ManyToOne
14    private Email email;
15
16    public Person(String name, Email email) {
17        this.name = name;
18        this.email = email;
19    }
20}
```

Упражнение 3

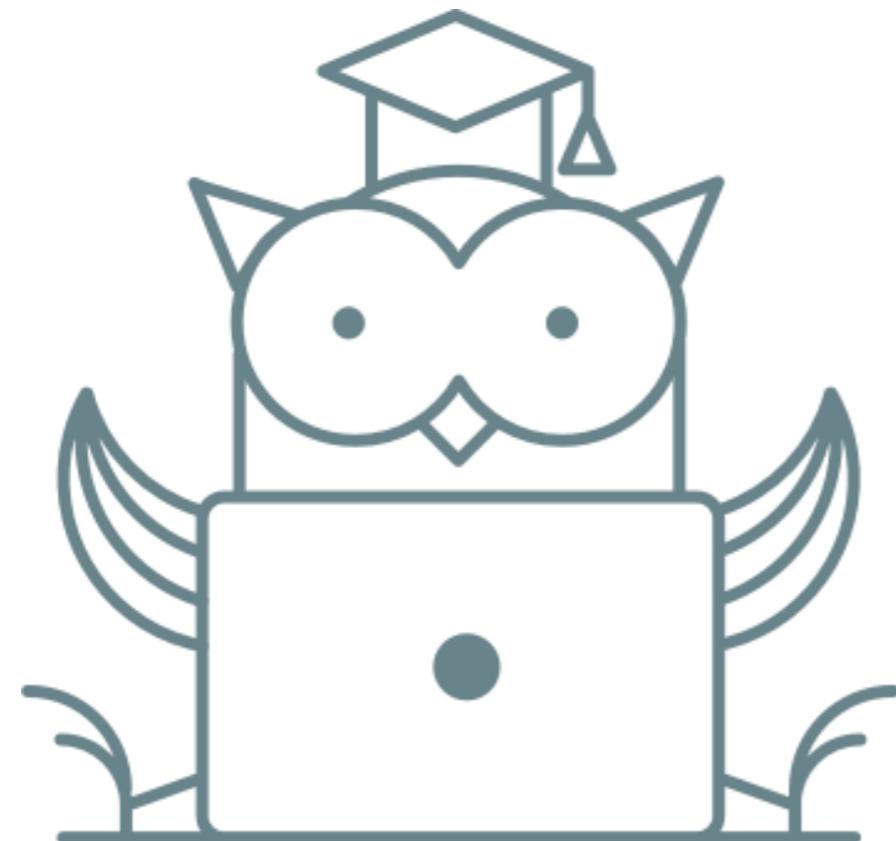
- Связать Email и Person @OneToOne



Вопросы?

Упражнение 4*

- Связать Email и Person @ManyToOne



Вопросы?

ИТОГО

- Прошли что такое JPA
- `@Entity`
- `@Table`
- `@Id`
- `@Column`
- `@OneToMany`, `@Many***`

О чём опционально можно прочитать

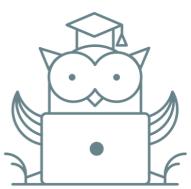
- @Embedded – во всех проявлениях
- Варианты @Join*, @ManyToMany
- Converters
- Collections
- Реализации наследования в JPA

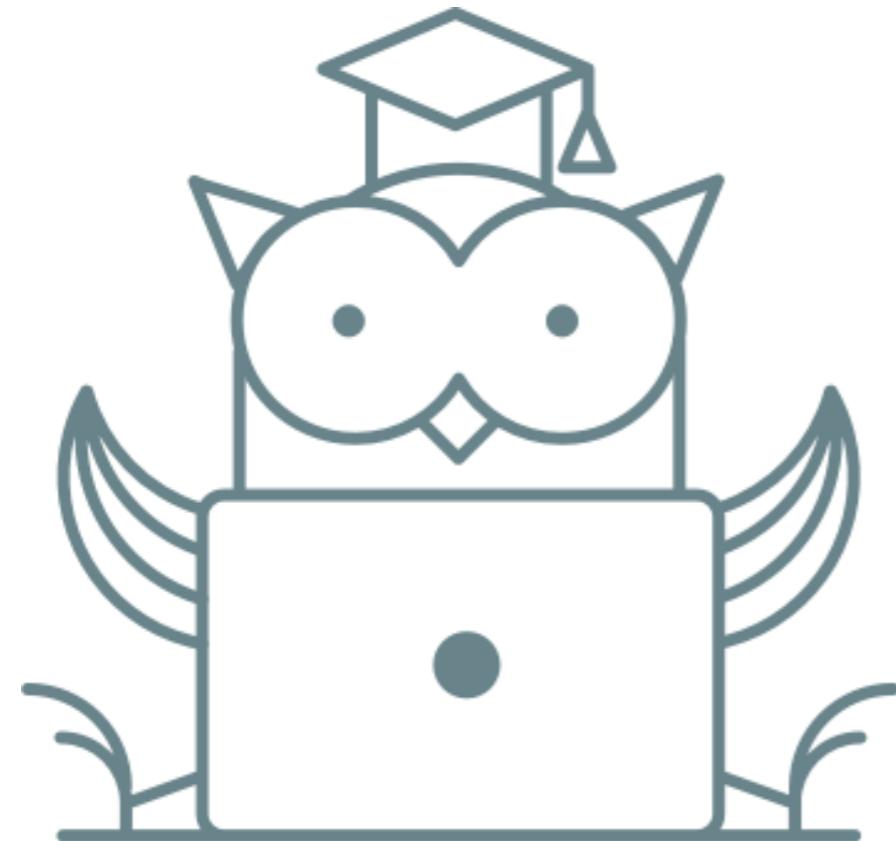


Вопросы?

Домашнее задание

нет

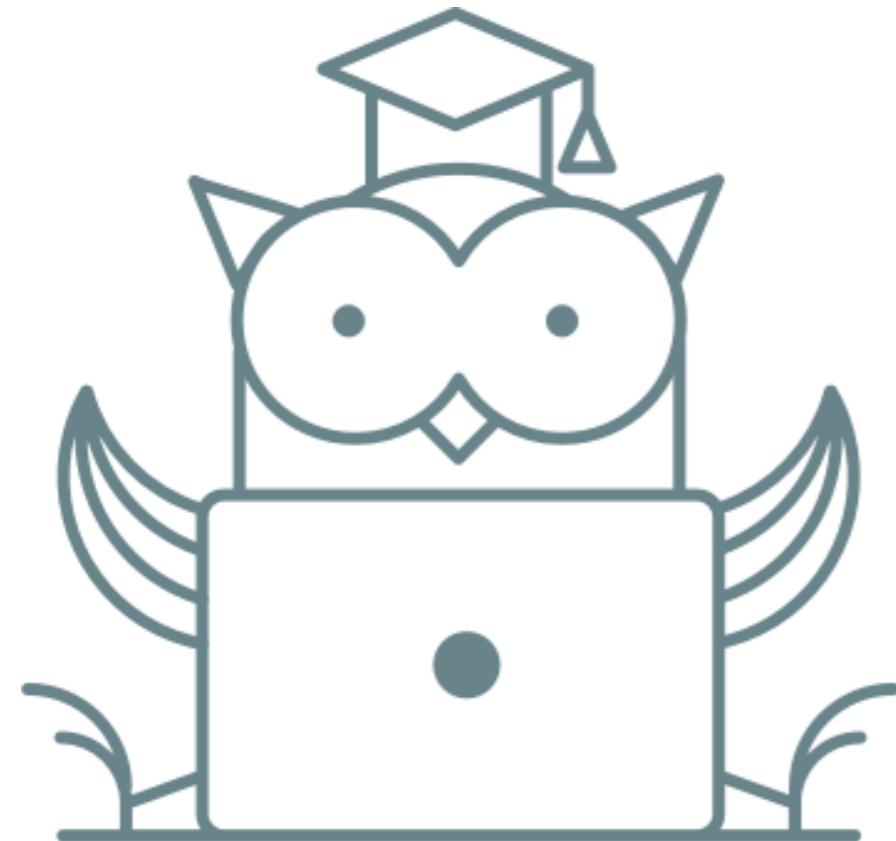




Вопросы?

Пожалуйста, пройдите опрос

<https://otus.ru/polls/1529/>



Спасибо
за внимание!

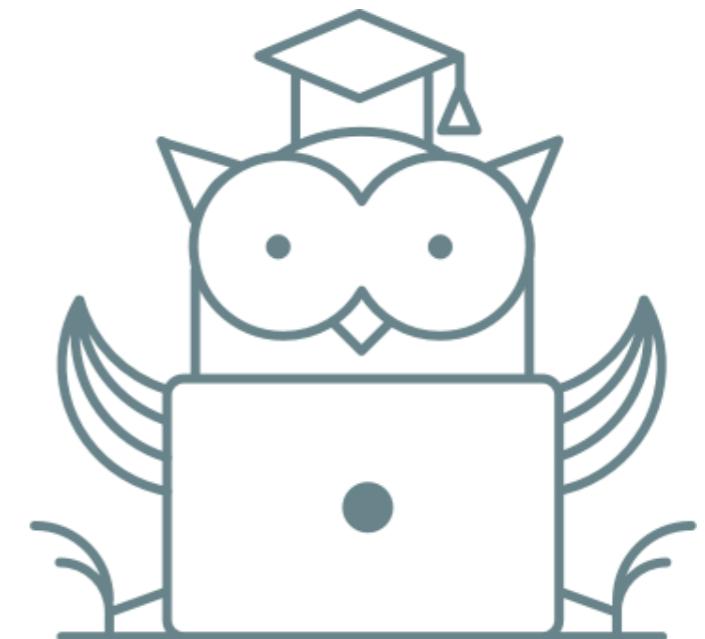
JPA Вам!

O ·Τ· U S

ОНЛАЙН-ОБРАЗОВАНИЕ

08 - JPQL, Spring ORM, DAO на основе Spring ORM + JPA

Дворжецкий Юрий



Как меня слышно && видно?



Если нет – напишите, если слышите – смайлик в чат.



Цели:

- API JPA, что делает Spring ORM
DAO на JPA
DAO на Spring ORM + JPA
- API JPA, JPQL
- Тестирование



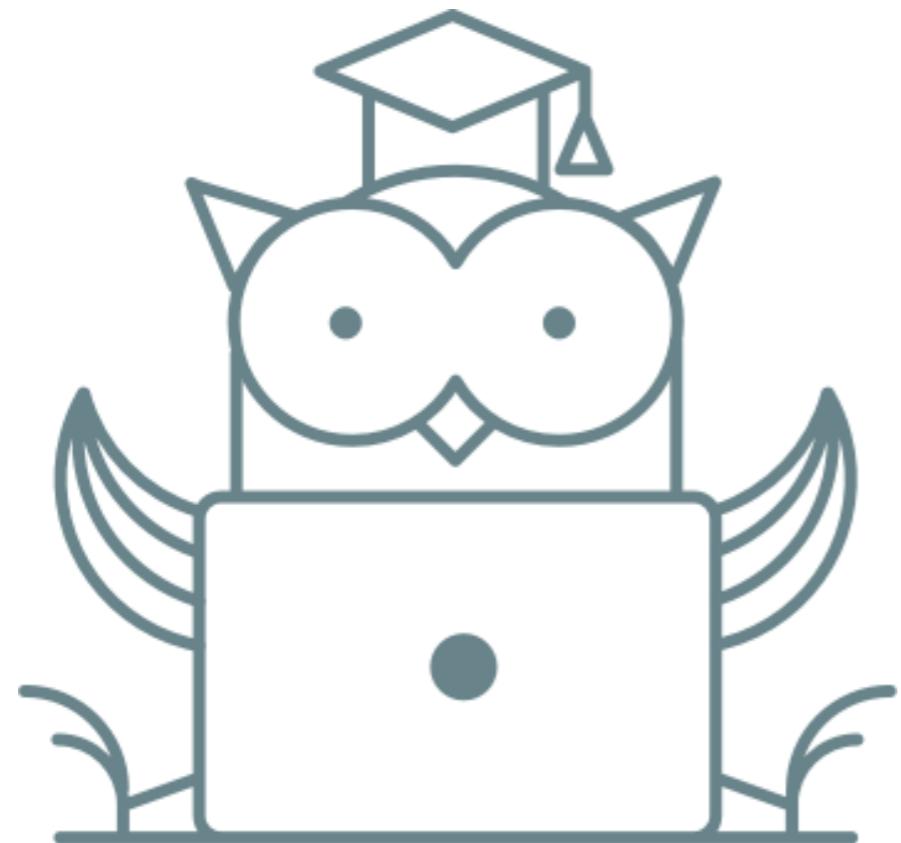
Цели:

- Сегодня, думаю, жесть)
- Наконец-то Spring
- Со Spring Boot всё так и будет
- Домашка нааконец-то есть.



Совсем чуть-чуть орг.вопросов:

- Срок проверки – три дня, стараемся быстрее
- Был завал – исправились.
- Вопросы задавайте. В сложных ситуациях помогаем. И давайте помогать друг-другу тоже)
- Со всех отзыв!



Поехали?



API JPA

JPA

- JPA – прослойка между Вашим кодом и ORM фреймворком
- JPA – это аннотации маппинга
- JPA – это кое-какие классы (API)
- Вот API сейчас и будем рассматривать

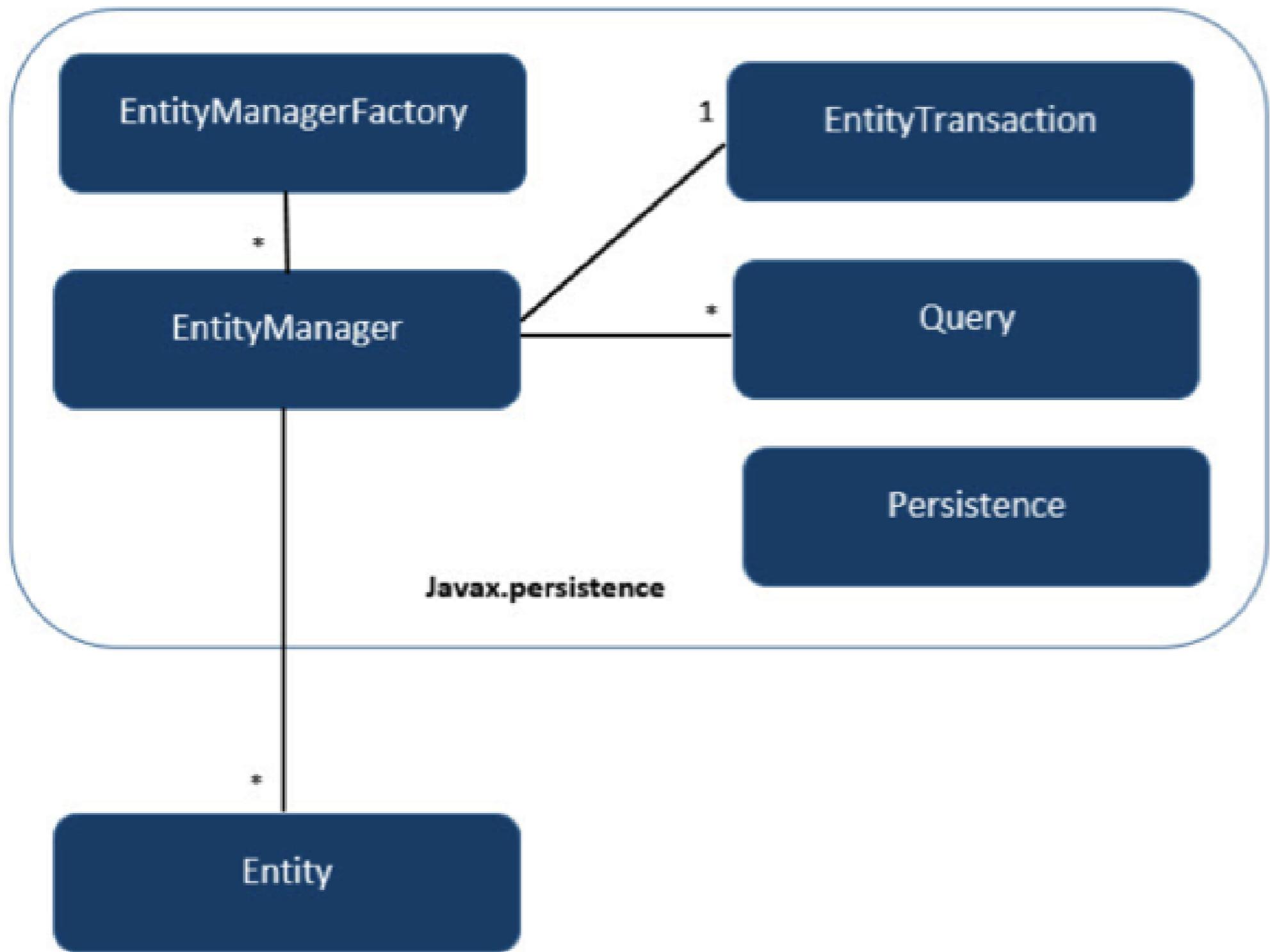
Пример маппинга

```
7
8     @Entity
9     public class Person {
10
11         @Id
12         @GeneratedValue
13         private int id;
14         private String name;
15
16         @OneToOne
17         private Email email;
18
19         public Person(String name, Email email) {
20             this.name = name;
21             this.email = email;
22         }
23
24         public int getId() {
25             return id;
26         }
27
28         public void setId(int id) {
29             this.id = id;
30         }
31
32         public String getName() {
```

Пример репозитория

```
8
9  @Repository
10 public class PersonRepositoryJdbc implements PersonRepository {
11
12     .... @PersistenceContext
13     .... private EntityManager em;
14
15     .... @Override
16     public Person getById(int id) {
17         .... return em.find(Person.class, id);
18     }  
19 }  
20 |
```

Классы



Немножко про старое JPA

- JPA – стационарный стандарт
- По стандарту по идее предполагалось наличие META-INF/persistence.xml

Он содержал

- persistence-unit-ы (БД + набор entity)* их может быть много
- Список entity
- И настройки JPA Vendor-a

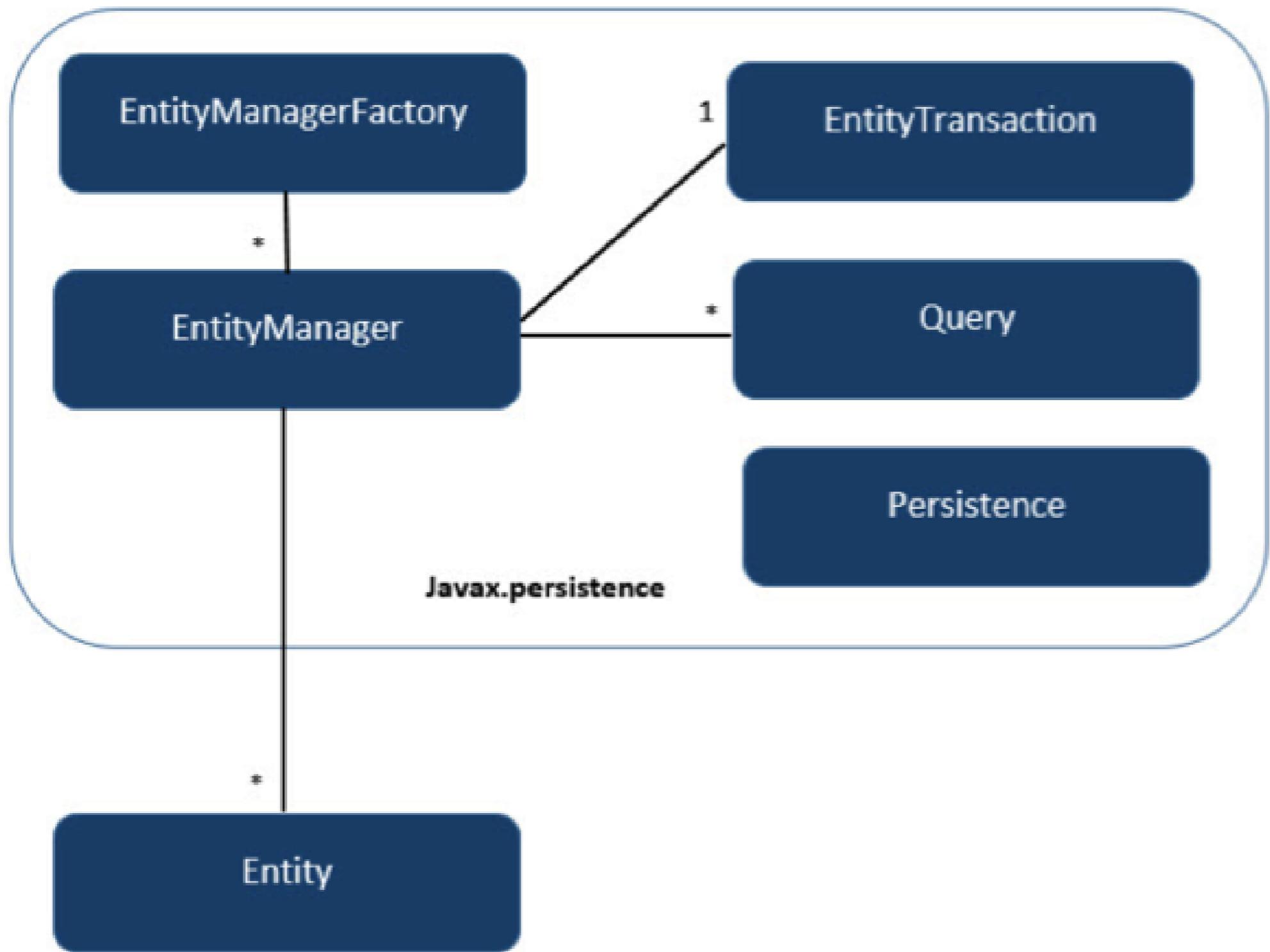
persistence.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
  http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">

  <persistence-unit name="my-pu">
    <description>My Persistence Unit</description>
    <provider>com.objectdb.jpa.Provider</provider>
    <mapping-file>META-INF/mappingFile.xml</mapping-file>
    <jar-file>packedEntity.jar</jar-file>
    <class>sample.MyEntity1</class>
    <class>sample.MyEntity2</class>
    <properties>
      <property name="javax.persistence.jdbc.url"
               value="objectdb://localhost/my.odb"/>
      <property name="javax.persistence.jdbc.user" value="admin"/>
      <property name="javax.persistence.jdbc.password" value="admin"/>
    </properties>
  </persistence-unit>

</persistence>
```

Классы



Классы

- Persistence (соответствует persistence.xml) – главный контекст для получения EntityManagerFactory
- EntityManagerFactory (соответствует Persistence unit) – создёт EntityManager + отвечает за метаданные
- EntityManager (PersistenceContext) – главный класс с которым происходит работа
- В один момент времени EntityManager соответствует ровно одна EntityTransaction
- Из EntityManager создаётся запрос (Query)

Пример репозитория на классическом JPA

```
@Repository
public class PersonRepositoryJpa implements PersonRepository {

    private EntityManagerFactory emf;

    @PersistenceUnit(unitName = "my-pu")
    public void setEmf(EntityManagerFactory emf) {
        this.emf = emf;
    }

    @Override
    public Person getById(int id) {
        EntityManager em = emf.createEntityManager();
        em.getTransaction().begin();
        Person p = em.find(Person.class, id);
        em.getTransaction().rollback(); // или .commit
        return p;
    }
}
```

Классический JPA

- @PersistenceUnit с помощью Spring ORM инжектит EntityManagerFactory
- Внутри метода создаём EntityManager
- Открываем транзакцию
- Получаем энтити
- Закрываем транзакцию

Хочется такого

```
@Repository
public class PersonRepositoryJpa implements PersonRepository {

    @PersistenceContext
    private EntityManager em;

    @Override
    public Person getById(int id) {
        em.getTransaction().begin();
        Person p = em.find(Person.class, id);
        em.getTransaction().rollback(); // или .commit()
        return p;
    }
}
```

Классический JPA

- @PersistenceContext инжектит EntityManager
- Открываем транзакцию
- Получаем энтити
- Закрываем транзакцию
- СТОП! Что здесь не так????

А со spring-ом это thread-safe 😊

Как он это делает?

```
@Repository
public class PersonRepositoryJpa implements PersonRepository {

    @PersistenceContext
    private EntityManager em;

    @Override
    public Person getById(int id) {
        em.getTransaction().begin();
        Person p = em.find(Person.class, id);
        em.getTransaction().rollback(); // или .commit
        return p;
    }
}
```

Spring ORM

- Spring ORM содержит реализацию EntityManagerFactory (LocalContainerEntityManagerFactoryBean)
- Он инжектит прокси EntityManager для @Repository
- Thread-safe
- LocalContainerEntityManagerFactoryBean – ещё заменяет persistence.xml – его не нужно писать.

Хочется такого

```
@Repository
public class PersonRepositoryJpa implements PersonRepository {

    @PersistenceContext
    private EntityManager em;

    @Override
    public Person getById(int id) {
        return em.find(Person.class, id);
    }
}
```

Spring Tx

- Содержит TransactionManager
- Управляет JPA транзакциями
- Реопзитории теперь можно смело писать просто
- spring-boot-starter-data-jpa содержит и spring-orm и spring-tx

DAO ha ORM

Repository

Entity + Mapping

ORM framework (Hibernate)

JDBC

DB

DAO на JPA

Repository

Entity + JPA

JPA

Hibernate as JPA Vendor

JDBC

DB

DAO và Spring ORM + JPA

Repository

Entity + JPA

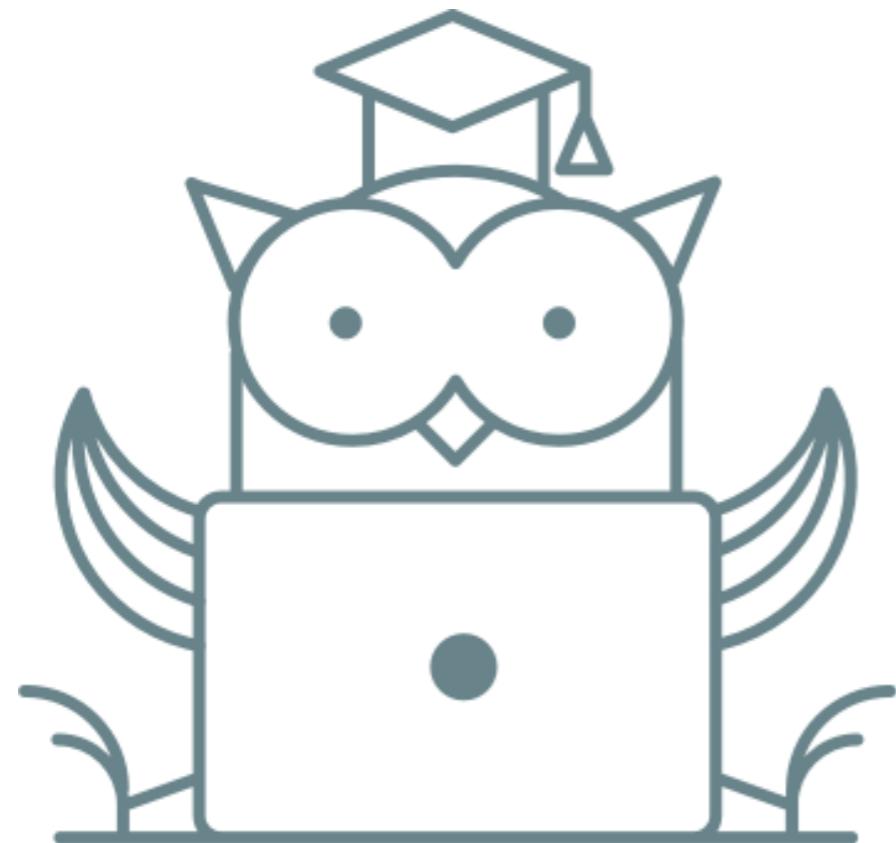
JPA

Spring ORM

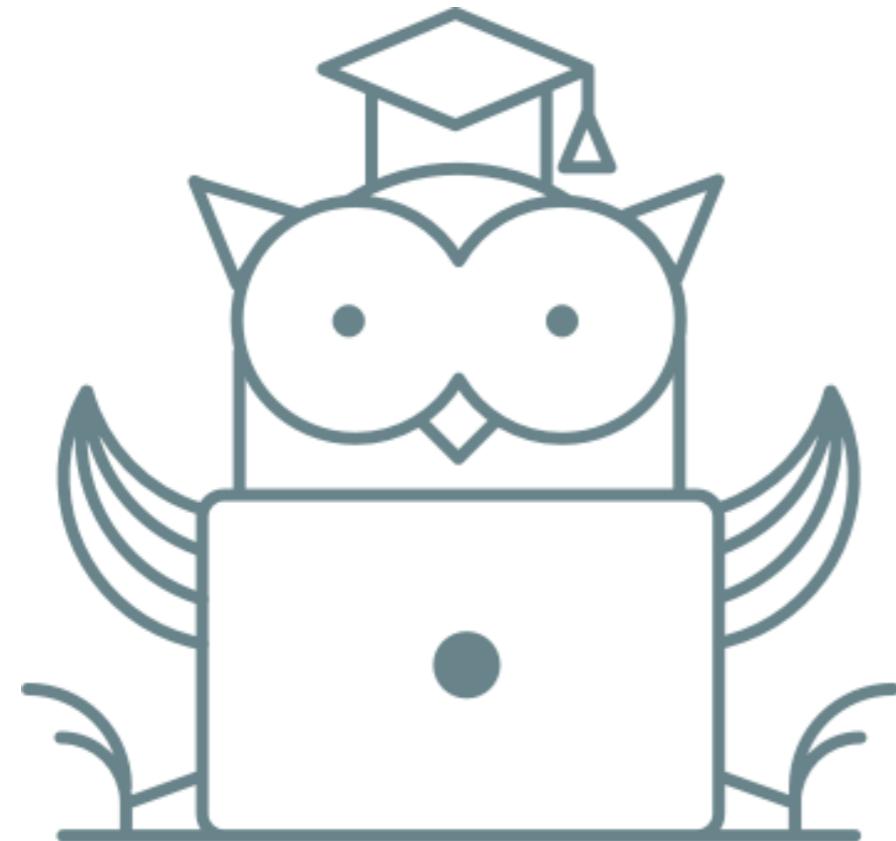
Hibernate as JPA Vendor

JDBC

DB



Вопросы?



API JPA, JPQL

State Entity

- Когда речь идёт про ORM принято говорить о состоянии
- Есть состояние текущего объекта (state)
- А есть сохранённое состояние (persist state)
- И по идее обязанность фреймворка синхронизировать state и persist state

Entity manager

- persist() - сохранить
- merge() – смержить state в persist state
- remove() - удалить entity
- find() – найти
- lock() – прям лок
- refresh() – persist state -> state
- detach() – отключить state от persistence context

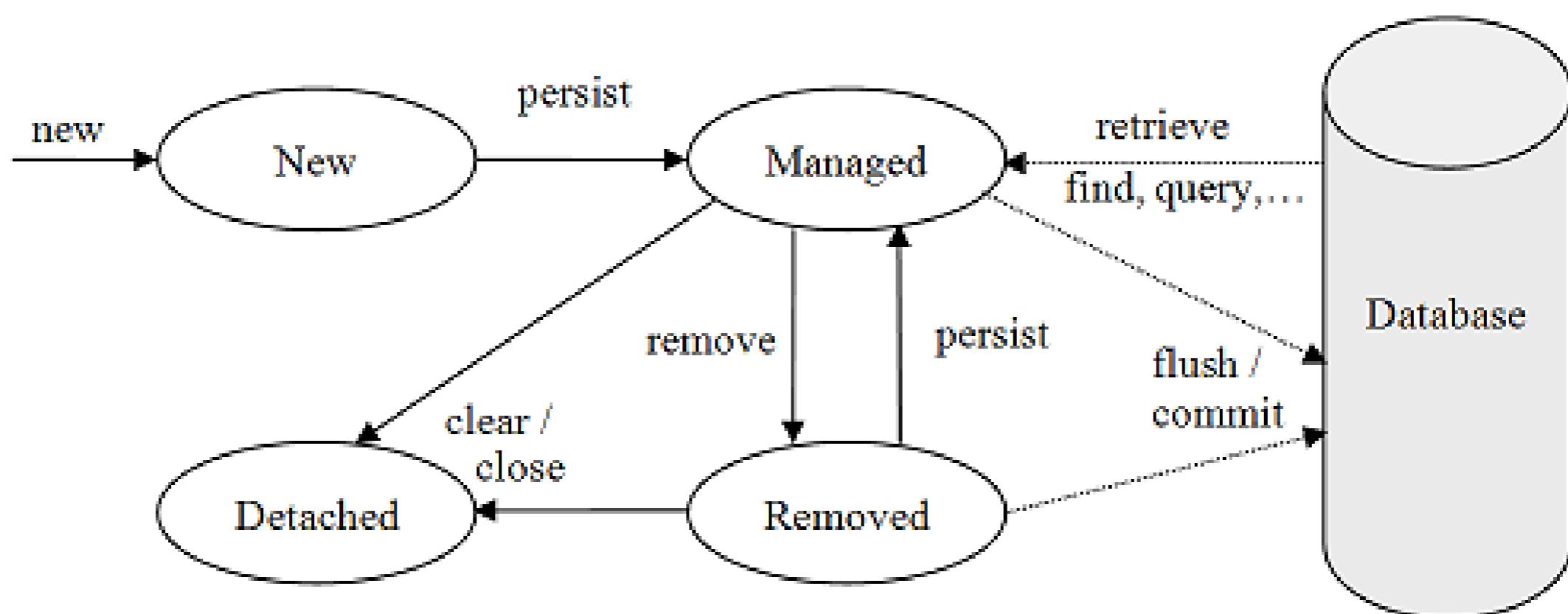
Пример

```
@Repository
public class PersonRepositoryJpa implements PersonRepository {

    @PersistenceContext
    private EntityManager em;

    @Override
    public void insert(Person p) {
        em.persist(p);
    }
}
```

State entity



State entity

- Ваше entity переходит из состояния в состояние
- Переходы нужно учить, как они переходят после методов
- Не забываем, что есть detach

Упражнение

- <https://github.com/ydvorzhetskiy/spring-framework-08>
- insert

Query

```
@Override  
public Person getFirst() {  
    TypedQuery<Person> query = em.createQuery(  
        s: "select e from Employee e where e.id = 1",  
        Person.class);  
    return query.getSingleResult();  
}  
  
@Override  
public List<Person> getAll() {  
    TypedQuery<Person> query = em.createQuery(  
        s: "select e from Employee e",  
        Person.class);  
    return query.getResultList();  
}
```

Query

- Создаётся в EntityManager
- Используется язык запросов JPQL (бизенс-объекты)
- Query и TypeQuery<> (используйте typed)
- getSingleResult и getMultipleResult

Упражнение

- <https://github.com/ydvorzhetskiy/spring-framework-08>
- getFirst (id = 1)
- getAll

JPQL

переведите

```
// Запрос для списка объектов.  
Query query = em.createQuery("select e FROM Employee e WHERE e.salary > 100000")  
List<Employee> result = query.getResultList();  
  
// Запрос для единственного объекта.  
Query query = em.createQuery("select e FROM Employee e WHERE e.id = :id");  
query.setParameter("id", id);  
Employee result2 = (Employee)query.getSingleResult();  
  
// Запрос для единственного элемента данных.  
Query query = em.createQuery("select MAX(e.salary) FROM Employee e");  
BigDecimal result3 = (BigDecimal)query.getSingleResult();  
  
// Запрос для списка элементов данных.  
Query query = em.createQuery("Select e.firstName FROM Employee e");  
List<String> result4 = query.getResultList();  
  
// Запрос для списка массивов элемента.  
Query query = em.createQuery("Select e.firstName, e.lastName FROM Employee e");  
List<Object[]> result5 = query.getResultList();
```

JPQL

переведите

```
SELECT COUNT(e) FROM Employee e
```

```
SELECT MAX(e.salary) FROM Employee e
```

```
SELECT e, a FROM Employee e, MailingAddress a WHERE e.address = a.address
```

```
SELECT e FROM Employee e JOIN e.projects p JOIN e.projects p2 WHERE p.name = :p1 and p2.name = :p2
```

```
SELECT AVG(e.salary), e.address.city FROM Employee e GROUP BY e.address.city
```

```
SELECT AVG(e.salary), e.address.city FROM Employee e GROUP BY e.address.city ORDER BY AVG(e.salary)
```

```
SELECT e, COUNT(p) FROM Employee e LEFT JOIN e.projects p GROUP BY e
```

```
SELECT AVG(e.salary), e.address.city FROM Employee e GROUP BY e.address.city HAVING AVG(e.salary) > 100000
```

JPQL

смотрим и радуемся

```
e.firstName IN (:name1, :name2, :name3)
e.firstName IN (:name1)
e.firstName IN :names
e.firstName IN (SELECT e2.firstName FROM Employee e2 WHERE e2.lastName = 'smith')
```

```
e.firstName = (SELECT e2.firstName FROM Employee e2 WHERE e2.id = :id)
e.salary < (SELECT e2.salary FROM Employee e2 WHERE e2.id = :id)
e.firstName = ANY (SELECT e2.firstName FROM Employee e2 WHERE e.id <> e.id)
e.salary <= ALL (SELECT e2.salary FROM Employee e2)
```

JPQL

СМОТРИМ

```
Query query = em.createQuery("UPDATE Employee e SET e.salary = 60000 WHERE e.salary = 50000");
int rowCount = query.executeUpdate();
```

```
Query query = em.createQuery("DELETE FROM Employee e WHERE e.department IS NULL");
int rowCount = query.executeUpdate();
```

Query params

```
@Override  
public Person getByName(String name) {  
    TypedQuery<Person> query = em.createQuery(  
        "select e from Employee e where e.name = :name",  
        Person.class);  
    query.setParameter("name", name);  
    return query.getSingleResult();  
}
```

Упражнение

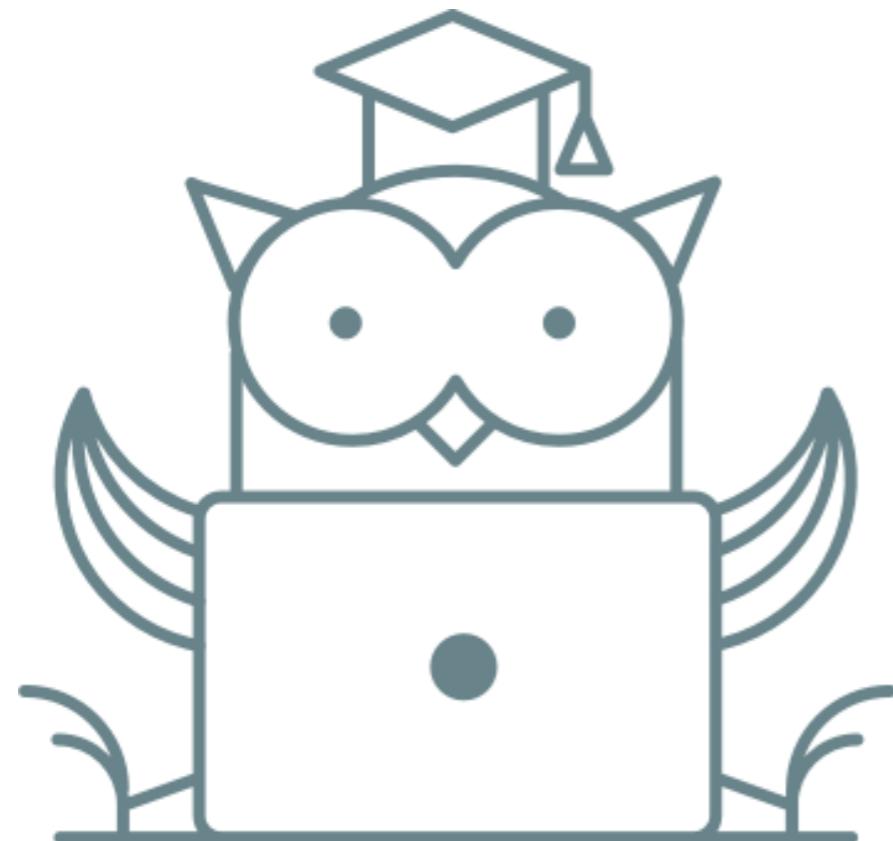
- <https://github.com/ydvorzhetskiy/spring-framework-08>
- `getByName`

О чём опционально можно прочитать

- Criteria API
- @NamedQuery
- State-ы



Вопросы?



Тестирование

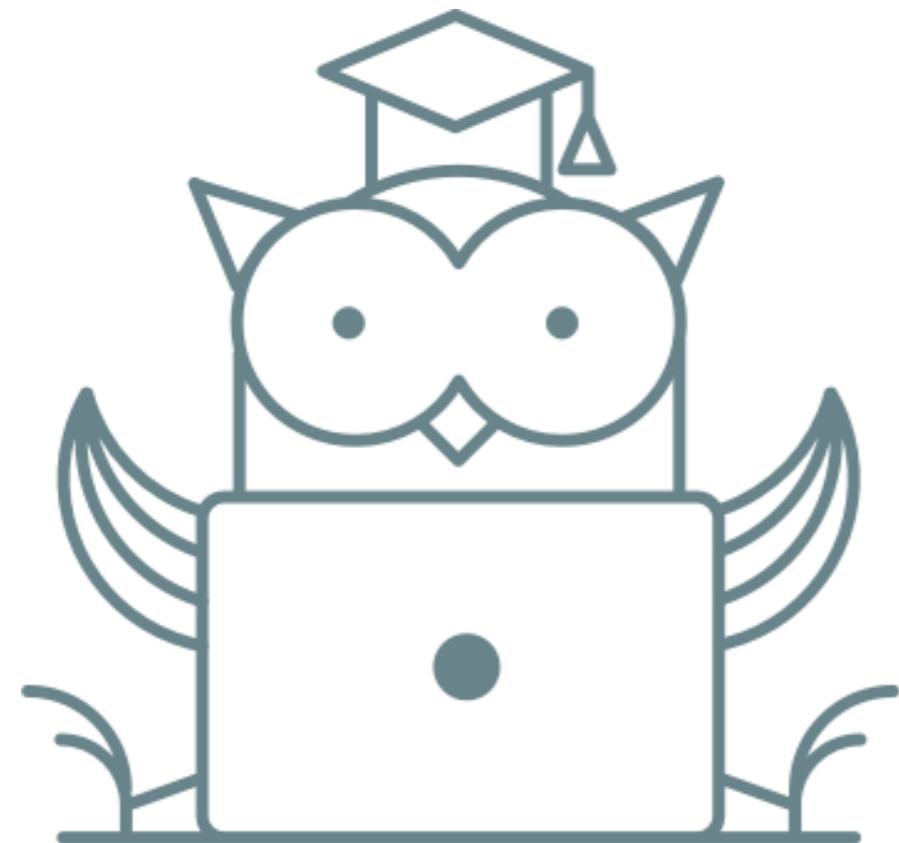
Аннотации и классы

- `@EntitiesScan`
- `TestEntityManager.`

Пример МОКОВ

```
public class EmploymentDaoJpaTest {  
  
    private EntityManager entityManager;  
    private EntityTransaction transaction;  
    private Query query;  
  
    @Before  
    public void setUp(){  
        entityManager = mock(EntityManager.class);  
        transaction = mock(EntityTransaction.class);  
        query = mock(Query.class);  
    }  
}
```

```
1  @Test
2  public void whenFindByName_thenReturnEmployee() {
3      // given
4      Employee alex = new Employee("alex");
5      entityManager.persist(alex);
6      entityManager.flush();
7
8      // when
9      Employee found = employeeRepository.findByName(alex.getName());
10
11     // then
12     assertThat(found.getName())
13         .isEqualTo(alex.getName());
14 }
```



Вопросы?

Домашнее задание

Использовать JPA, Hibernate только в качестве JPA-провайдера.

Добавить комментарии к книгам, и высокоуровневые сервисы, оставляющие комментарии к книгам.

Покрыть DAO тестами используя H2 базу данных и соответствующий H2 Hibernate-диалект

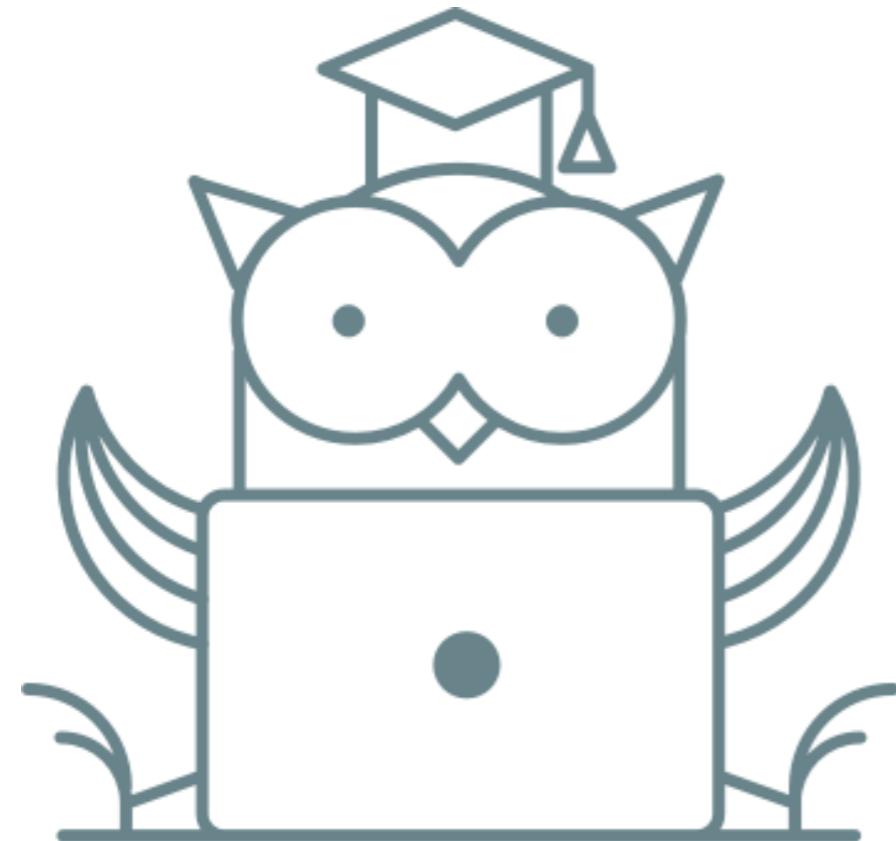




Вопросы?

Пожалуйста, пройдите опрос

<https://otus.ru/polls/1544/>



Спасибо
за внимание!

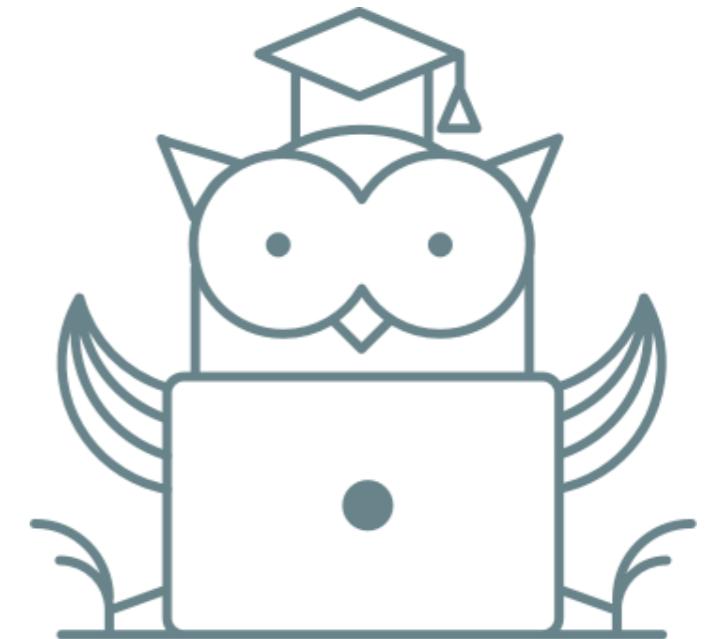
JPA Вам!

O ·Τ· U S

ОНЛАЙН-ОБРАЗОВАНИЕ

09 – Транзакции, Spring Tx

Дворжецкий Юрий



Как меня слышно && видно?



Если нет – напишите, если слышите – смайлик в чат.



Цели:

- Теория транзакций в реляционных БД
- Spring Tx на основе Spring JDBC



Планы:

- Теория – на самом деле не жесть, жесть на практике.
- Только теория. Spring-а немнogo.
- Но это очень важно, Spring Boot Вас, кстати, не спасёт.
- Домашки нет.



Совсем чуть-чуть орг.вопросов:

- Срок проверки – три дня, стараемся быстрее
- Мне всё больше и больше нравятся Ваши ДЗ
😊 Фантазии наконец-то даём волю 😊
- Вопросы задавайте в чат. У нас же коммьюнити 😊
- Сегодня новинка.
- Со всех отзыв!

Новинка

- Ввожу раздел «Наши звёздочки»
- Буду показывать идеи и кусочки кода.
- Код, только с Вашего позволения.
- Обязательно с указанием авторства

«Наши звёздочки»



Andrey Dzhura



- Между домашними JDBC и JPA в рамках домашней JDBC реализовал на основе Spring JDBC

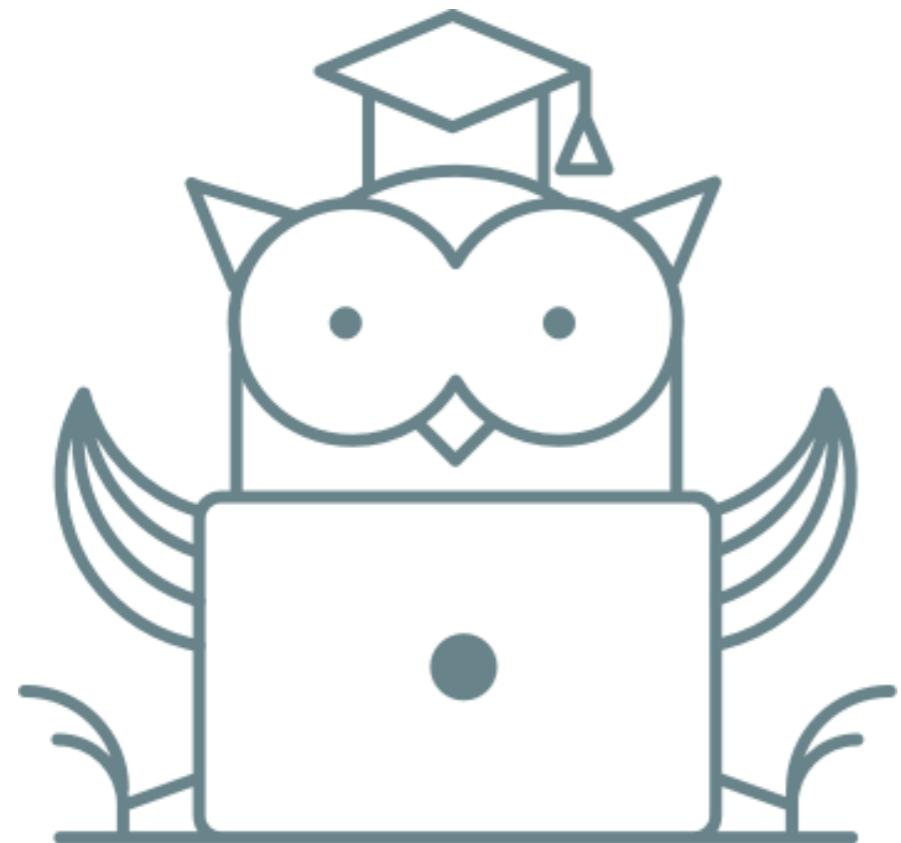
ORM-фреймворк (!!?)

«Реализующий» маппинг некоторых аннотаций JPA

Поздравляем!!!



Поехали?



Теория транзакций

Упражнение

- Что по Вашему такое транзакция?

ACID

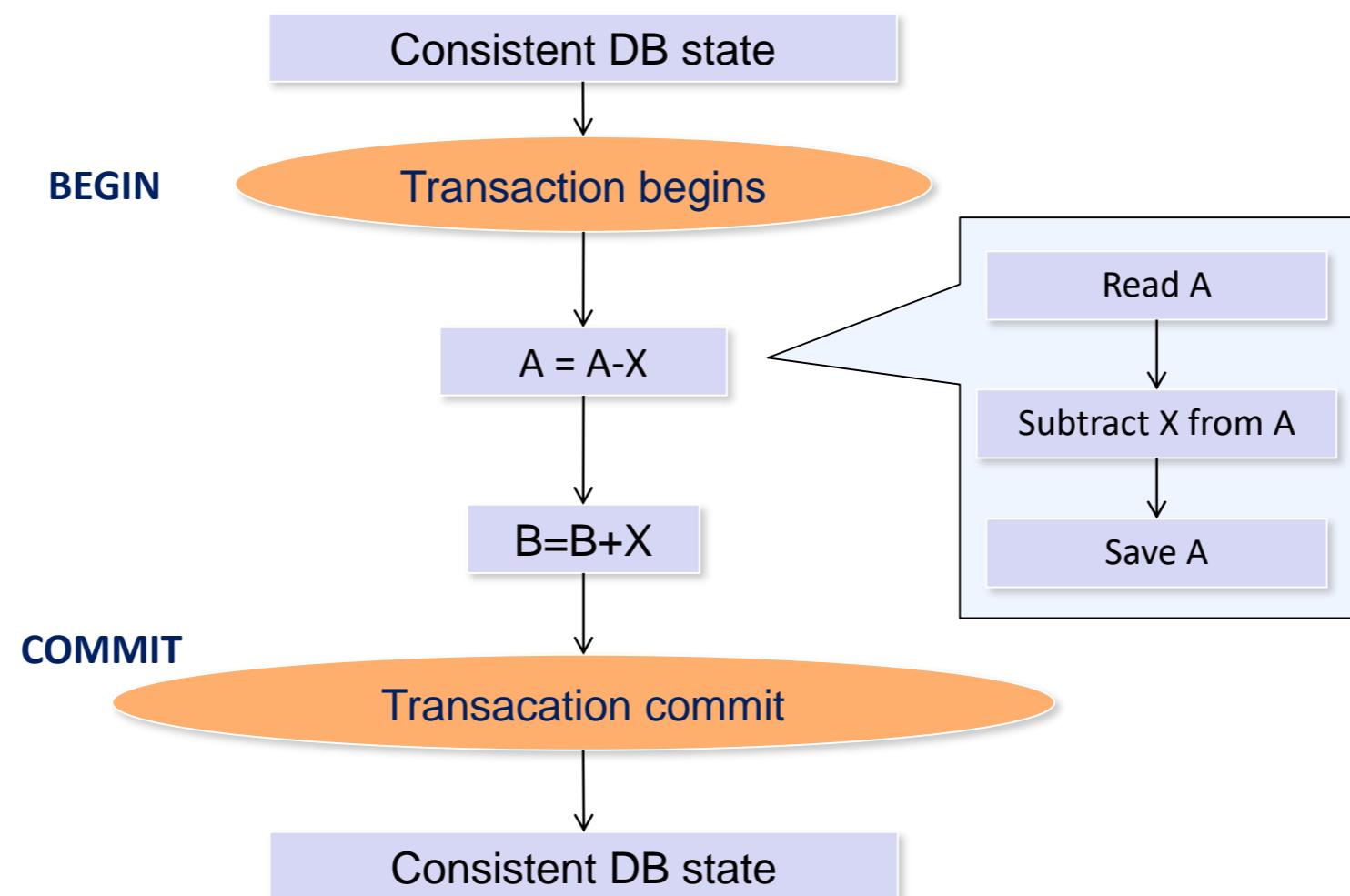
- Atomicity - атомарность
- Consistency - консистентность
- Isolation - изоляция
- Durability - долговечность

ACID

- ACID – очень важные
- Часто в БД «транзакции» – это не транзакции.
- В NoSQL базах данных – это совсем мАркенговая уловка
- Как ни странно и в

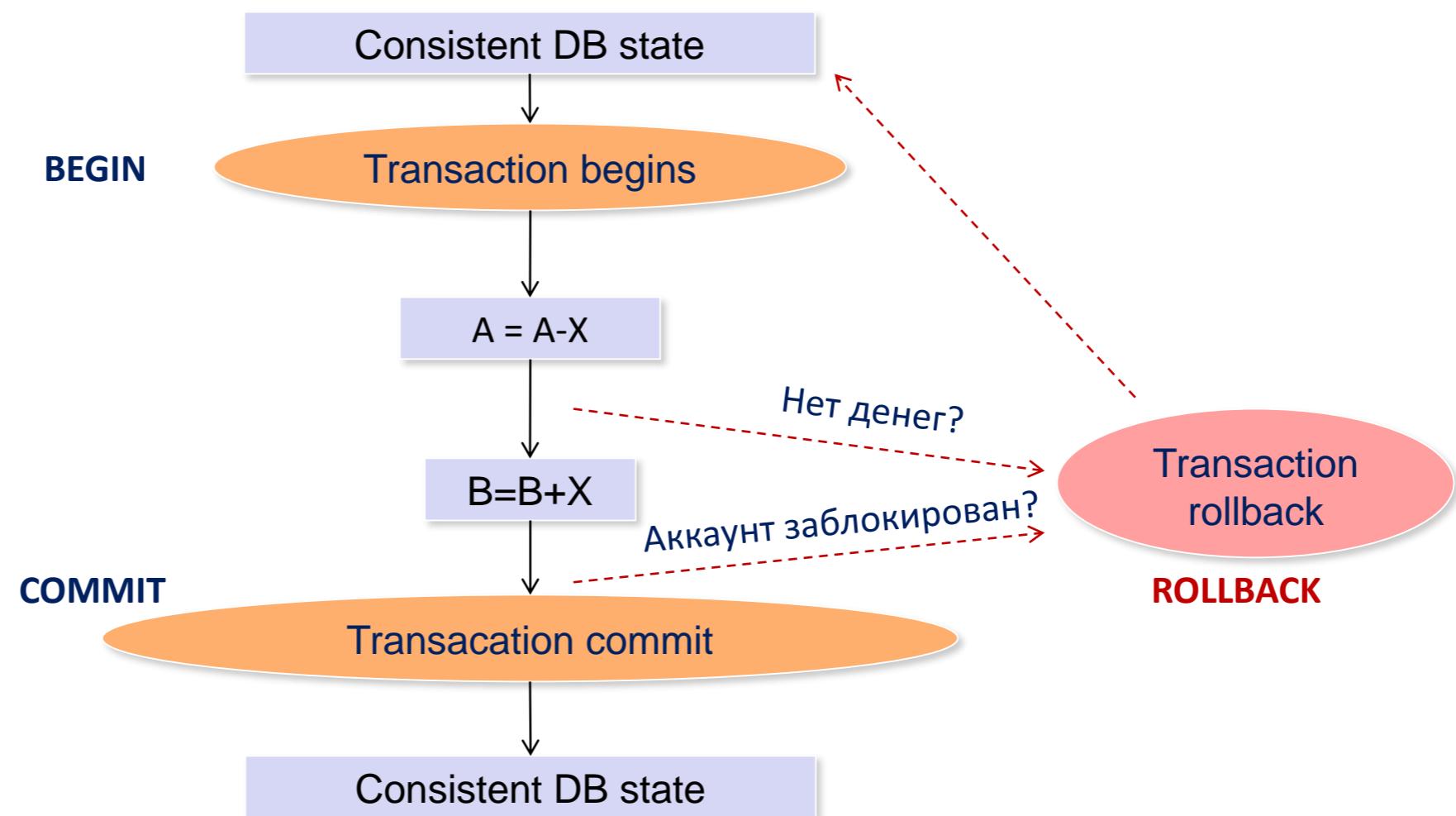
Commit

Перевод X долларов с аккаунта А на аккаунт В



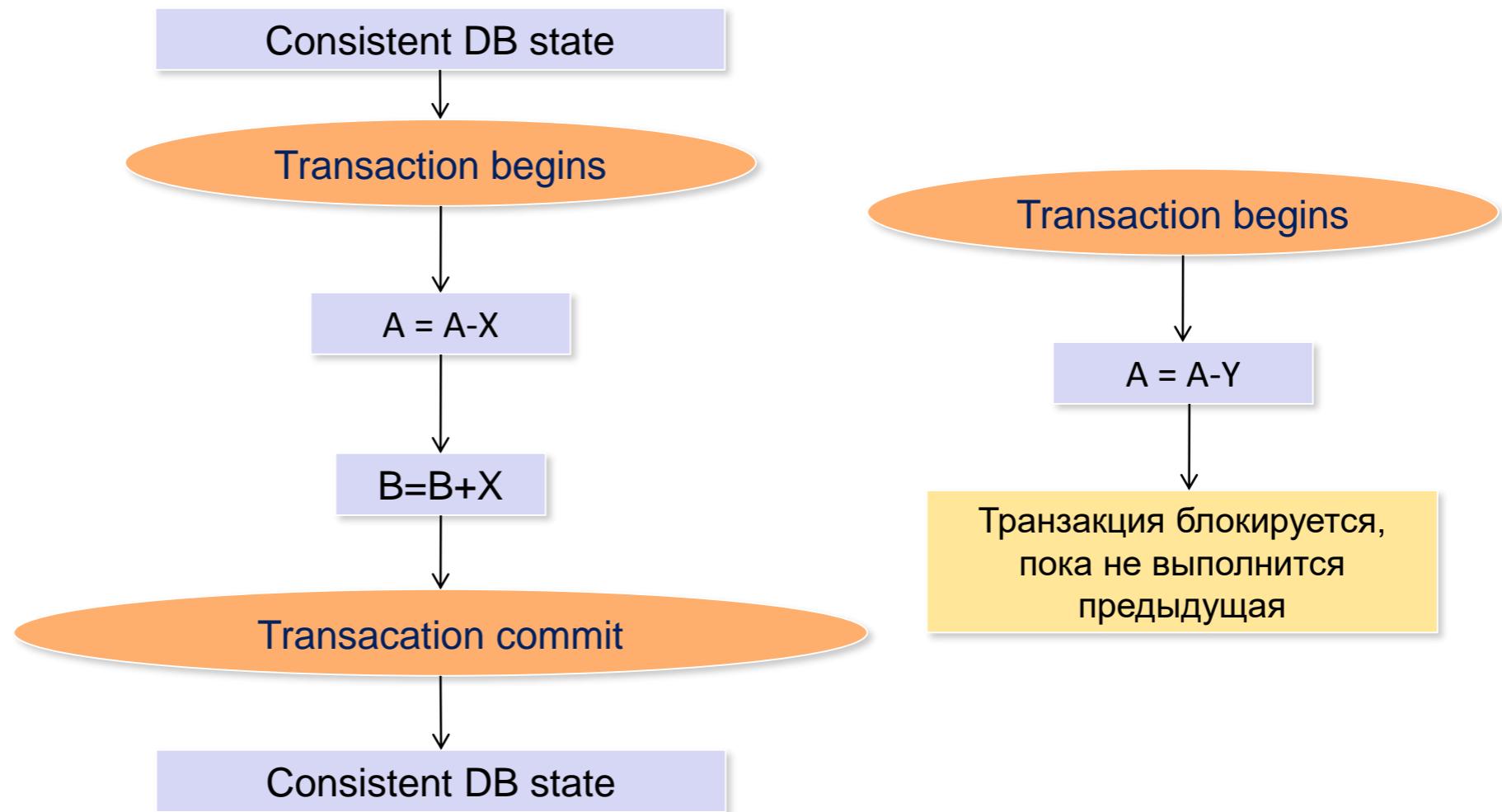
Rollback

Перевод X долларов с аккаунта А на аккаунт В



Параллельные транзакции

Перевод X долларов с аккаунта А на аккаунт В

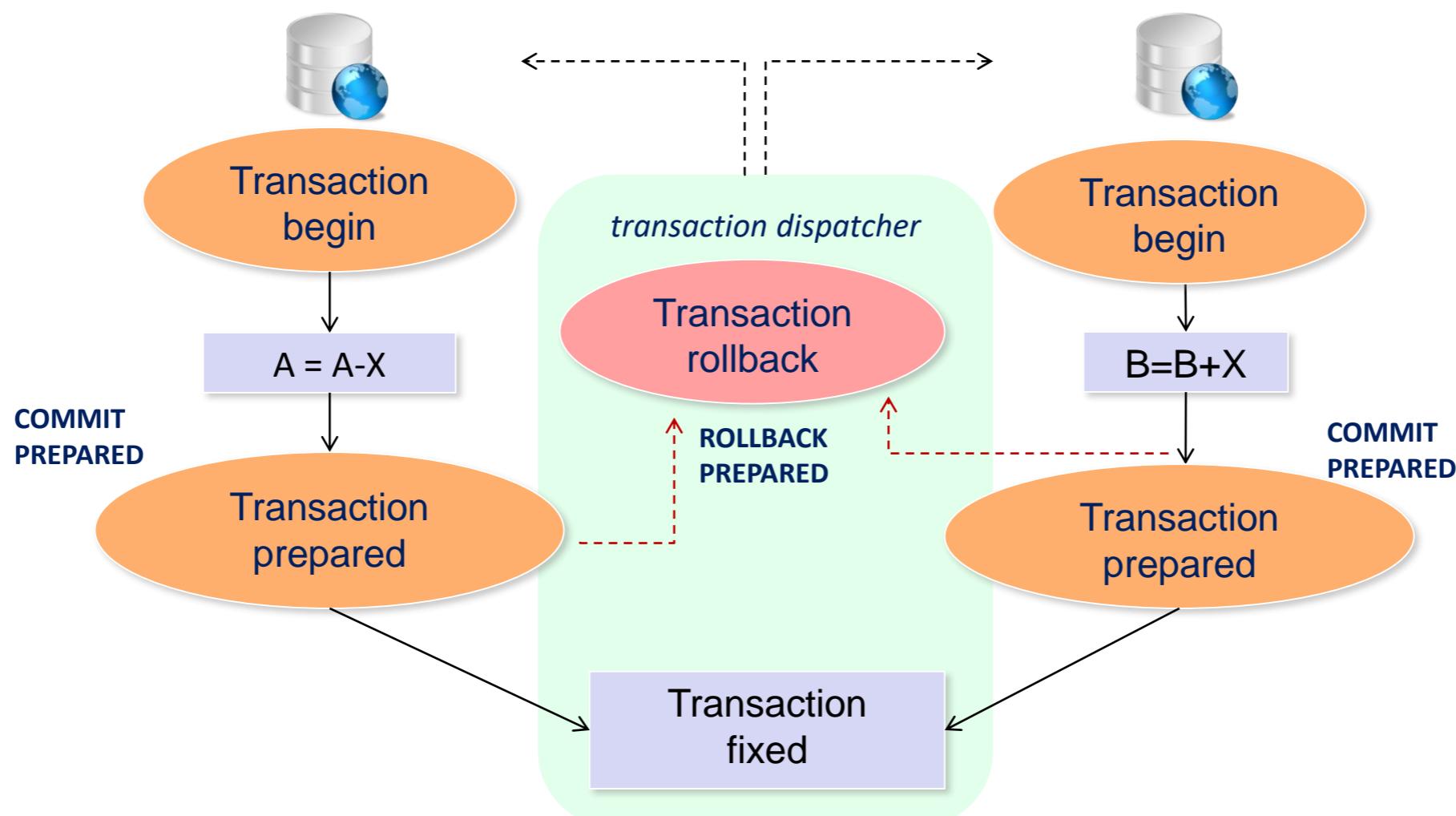


Типы транзакций

- Local – транзакции в одном DataSource
- Global – глобальные (распределённые), в нескольких DataSource-ах.
- Обычно управляются J2EE контейнером, используя JTA (Java Transaction API).
- Spring Boot позволяет просто подключать глобальные транзакции
- Но это совсем не просто...

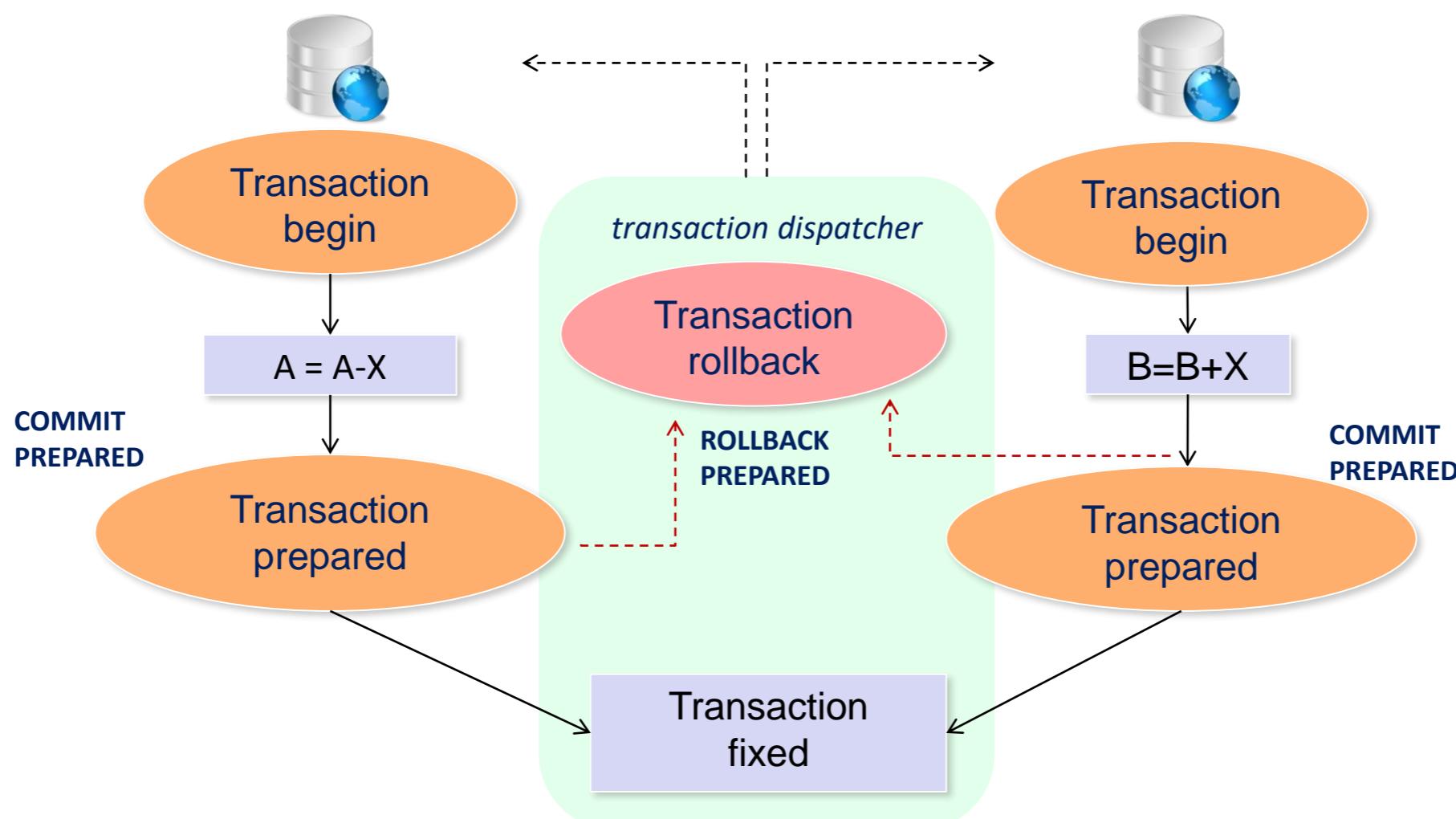
Распределённые транзакции

Перевод X долларов с аккаунта А в одном банке на В в другом банке



Какие проблемы?

Перевод X долларов с аккаунта А в одном банке на В в другом банке



Транзакции

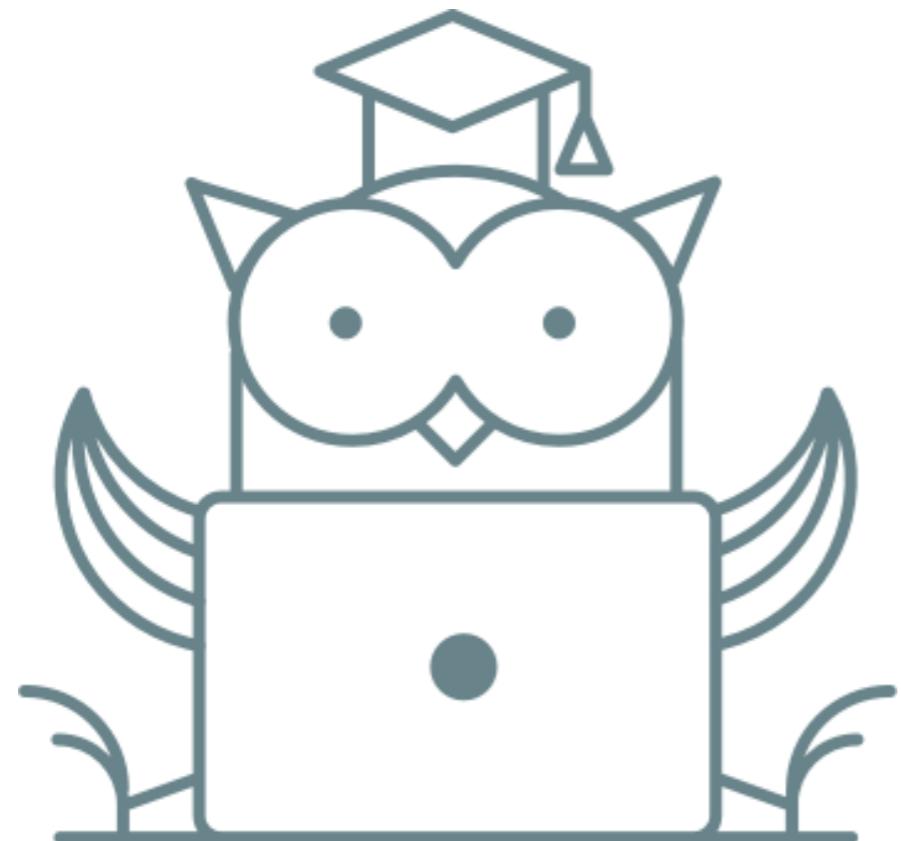
- В действительности транзакции бывают не только у БД
- Есть транзакции и JMS
- Как думаете для чего?

Пара слов о транзакциях

- Транзакционность очень тяжело тестируется
- Дефекты иногда невозможно вспоминать*
- Баг с транзакциями может искааться полгода*
- Unit-тестами почти не покрывается
- Но это очень важно в реальных приложениях



Вопросы?



Spring Tx

Транзакции в Spring

- Spring Tx – модуль для управления транзакциями (spring-tx, часть spring-jdbc)
- Поддерживает транзакции для разных технологий – JDBS, JPA и т.д.
- TransactionManager управляет транзакциями

Transaction Manager

- DataSourceTransactionManager
- HibernateTransactionManager
- JmsTransactionManager
- JmsTransactionManager102
- JpaTransactionManager

Spring Tx

- Декларативное управление с помощью аннотации
@Transactional
(Часто незаслуженно не любят за «магию»)
- Есть возможность и императивного управления –
TransactionTemplate
(Вполне объективно не любят за Большее количество
кода)

@Transactional

```
[-] @Transactional(readOnly = true)
public class DefaultFooService implements FooService {
    [-] ... public Person getPerson(String name) {
        ... // do something
    }
    [-] ... @Transactional(readOnly = false, propagation = Propagation.REQUIRES_NEW)
    ... public void update(Person person) {
        ... // do something
    }
}
```

@Transactional

- org.springframework.transaction.annotation.Transactional
- Не путать с javax.transaction.Transactional
- Хотя поведение схоже
- Опишите как она реализуется в Spring? Мы уже знаем.

@Transactional

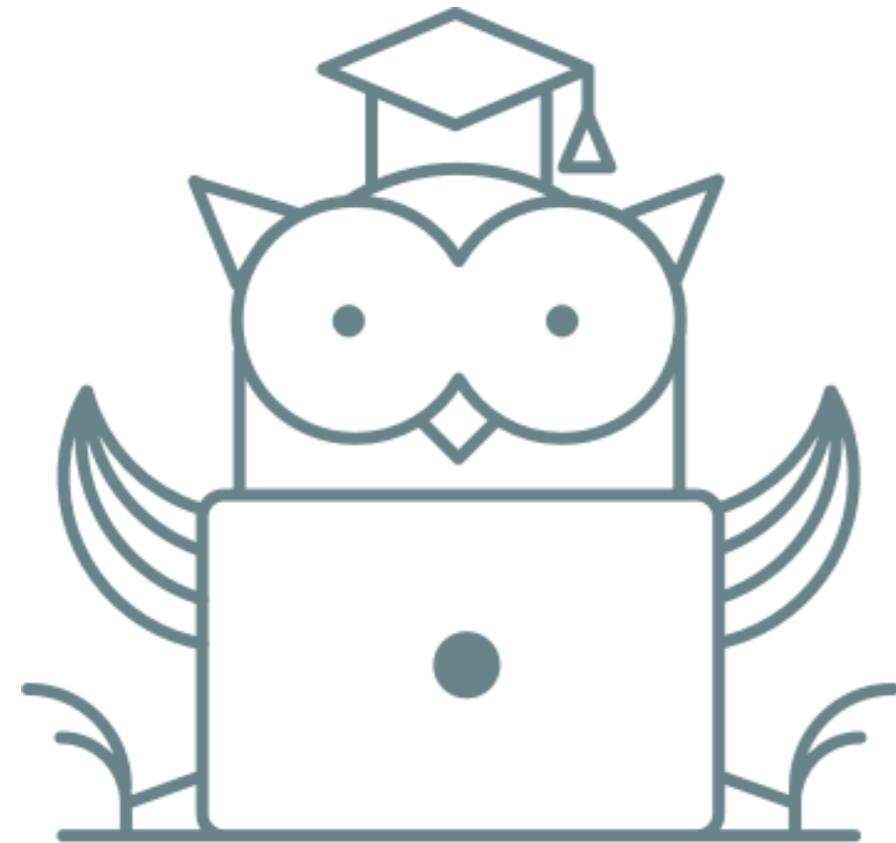
Параметры аннотации @Transactional:

- isolation – уровень изоляции транзакции
- propagation – как вкладываются транзакции
- timeout
- readOnly
- И другие

@Transactional

@Transactional применяется к:

- Интерфейсам
- Классам
- Методам интерфейса
- public методам класса (!)
- Считается хорошим тоном применять @Transactional на конкретные класс и методы, вместо интерфейсов



Вопросы?

TransactionManager

- Чтобы это заработало нужен тот самый TransactionManager
- Для разных технологий – он разный
- spring-boot-starter-tx включает TransactionManager для каждой транзакции

Isolation

- Транзакции очень дорогие для БД.
- Чтобы выполнить их корректно, по-хорошему их нужно выполнять последовательно
- Никакая БД не выполняет последовательно, иначе её никто не купит.
- Приходится идти на мАркетинговые уловки

*Тот самый @Transactional(isolation=)

Уровни изоляции транзакций

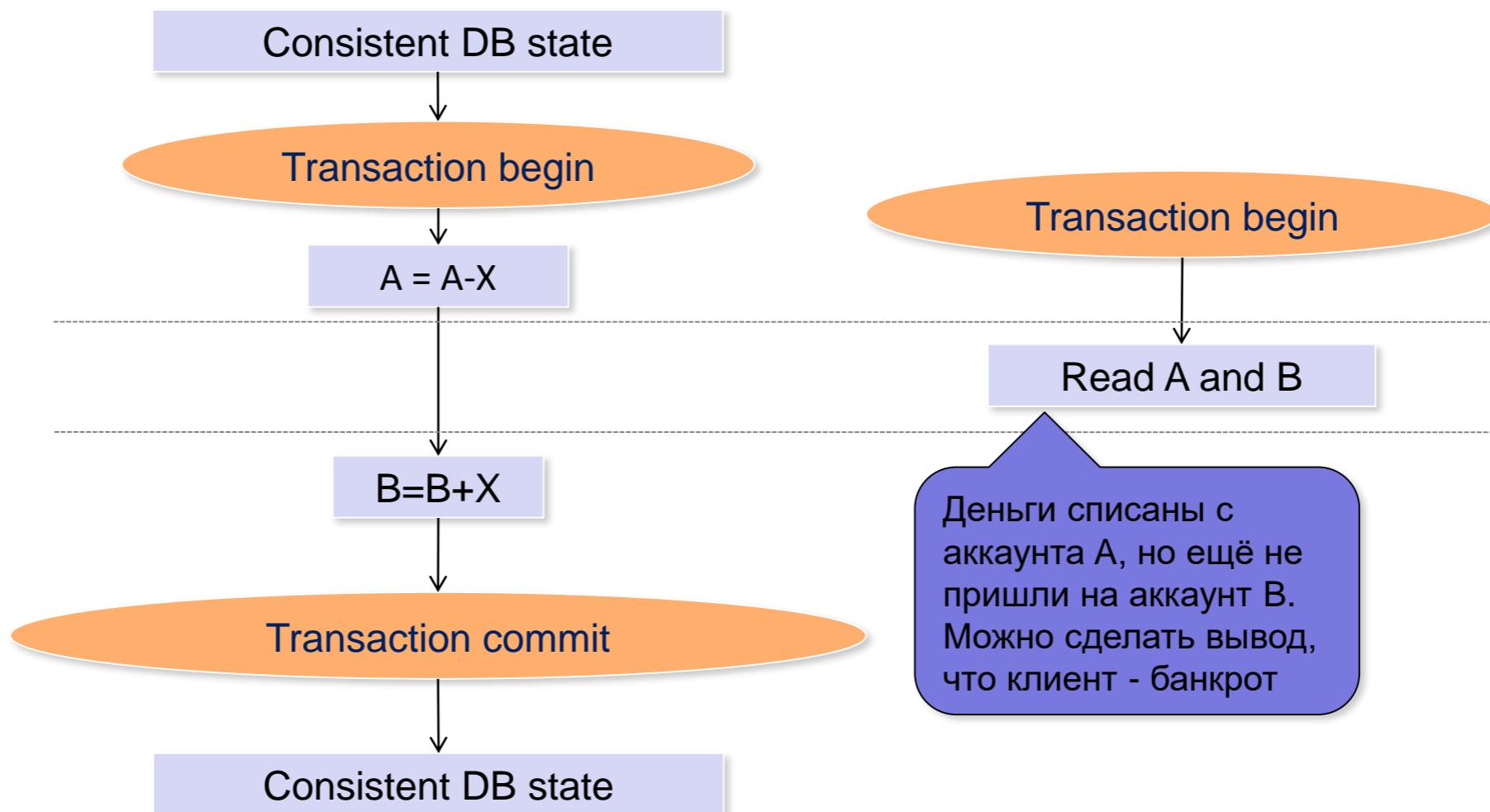
- Уровни изоляции транзакций – это как раз та уловка
- В действительности они не изолированы
- Но появляются какие-то side-эффекты, с которыми можно мириться в некоторых запросах
- За них Бд получает плюшки, а от них – скорость.
- Бд работает на своём уровне

Уровни изоляции транзакций

Уровень изоляции	Phantom reading	Unrepeatable read	«Dirty» read	Lost update
ISOLATION_READ_UNCOMMITTED				
READ UNCOMMITTED	—	—	—	+
ISOLATION_READ_COMMITTED				
READ COMMITTED	—	—	+	+
ISOLATION_REPEATABLE_READ				
REPEATABLE READ	—	+	+	+
ISOLATION_SERIALIZABLE				
SERIALIZABLE	+	+	+	+

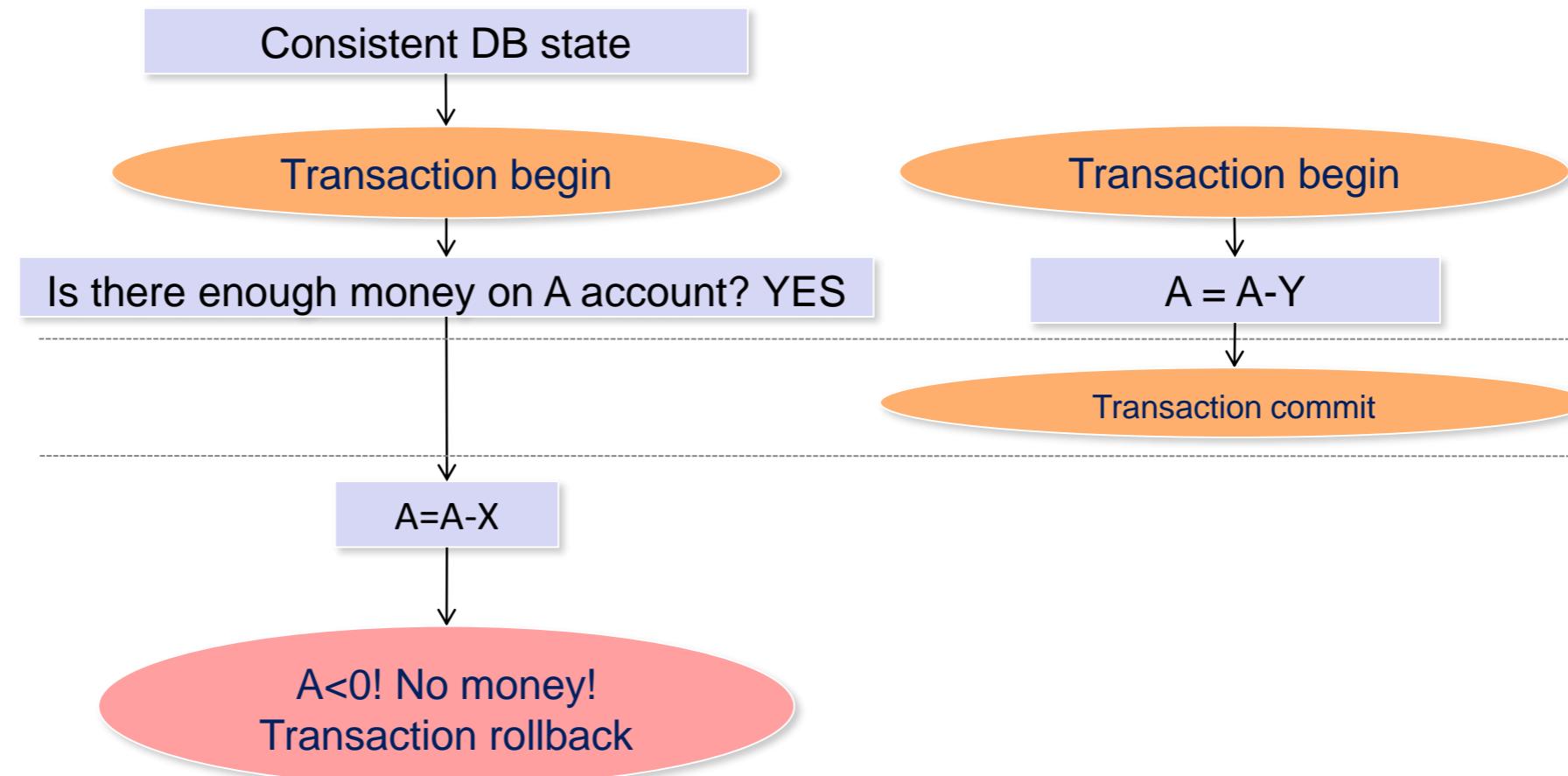
Dirty read (грязное чтение)

Перевод X долларов с аккаунта А на В



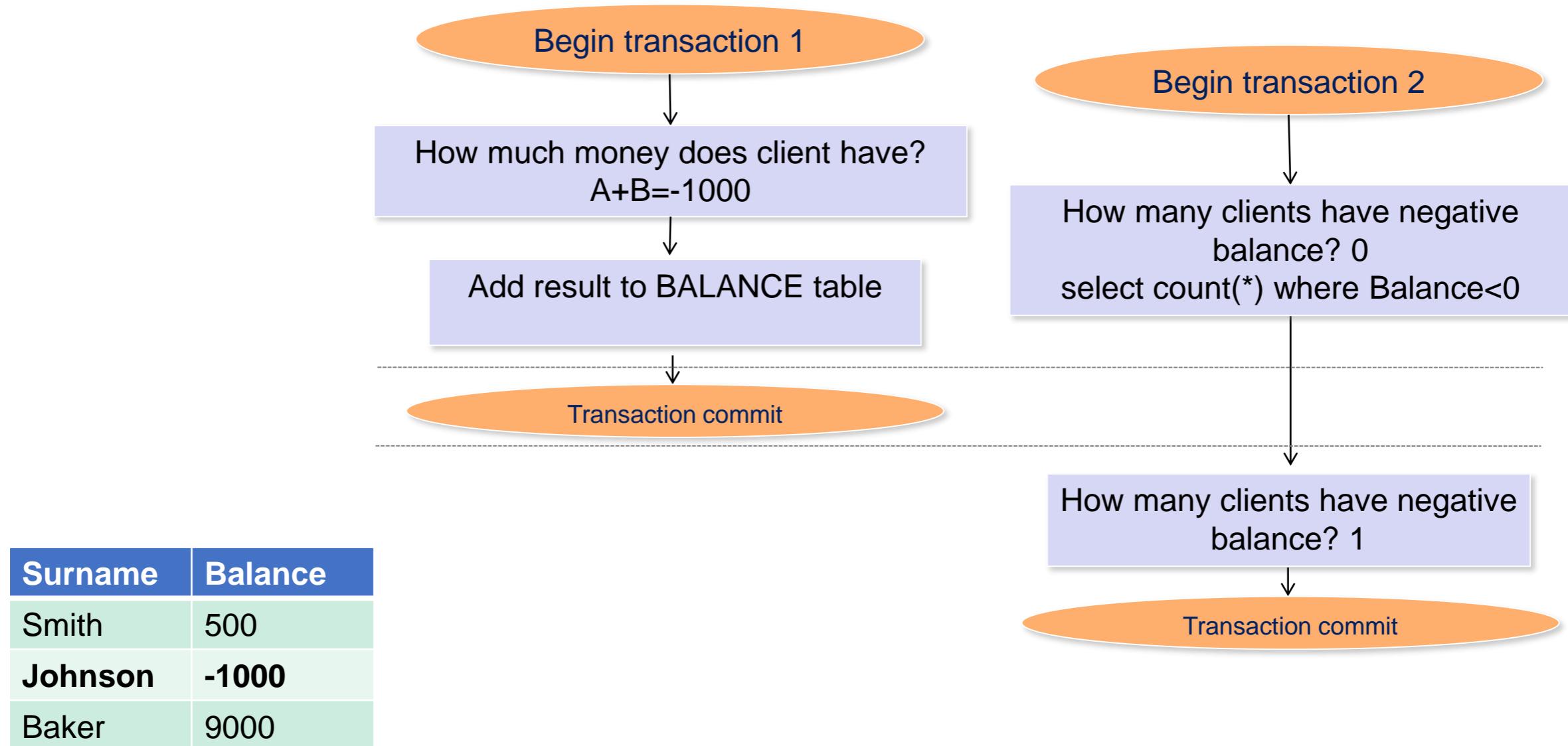
Unrepeatable read (неповторяющееся чтение)

Снятие денег с аккаунта А двумя параллельными транзакциями:



Phantom read (фантомное чтение)

Во время второй транзакции появляются фантомные записи

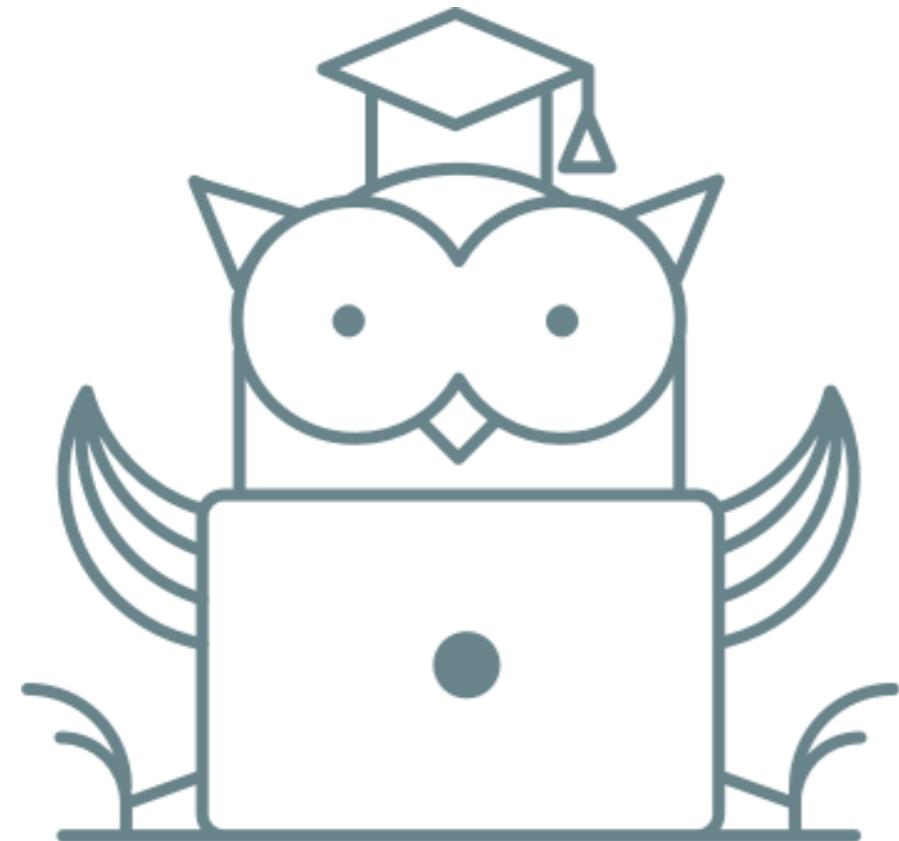


Уровни изоляции транзакций

Уровень изоляции	Phantom reading	Unrepeatable read	«Dirty» read	Lost update
ISOLATION_READ_UNCOMMITTED				
READ UNCOMMITTED	—	—	—	+
ISOLATION_READ_COMMITTED				
READ COMMITTED	—	—	+	+
ISOLATION_REPEATABLE_READ				
REPEATABLE READ	—	+	+	+
ISOLATION_SERIALIZABLE				
SERIALIZABLE	+	+	+	+

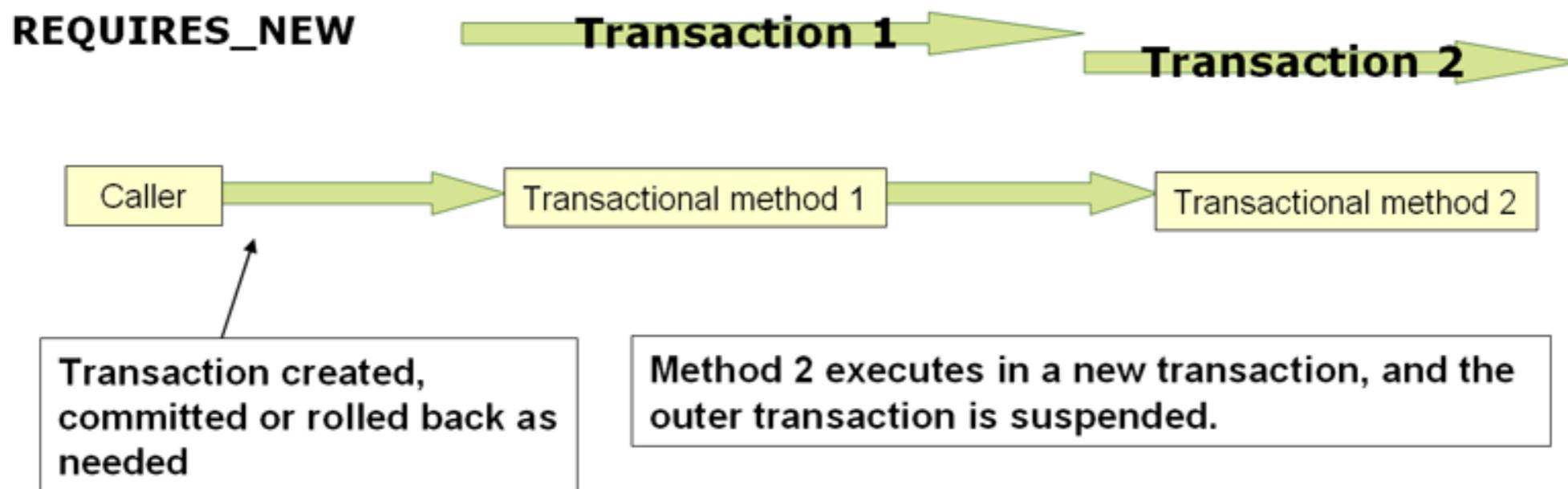
Для каких запросов? Где работает Ваша БД?

Уровень изоляции	Phantom reading	Unrepeatable read	«Dirty» read	Lost update
ISOLATION_READ_UNCOMMITTED				
READ UNCOMMITTED	—	—	—	+
ISOLATION_READ_COMMITTED				
READ COMMITTED	—	—	+	+
ISOLATION_REPEATABLE_READ				
REPEATABLE READ	—	+	+	+
ISOLATION_SERIALIZABLE				
SERIALIZABLE	+	+	+	+

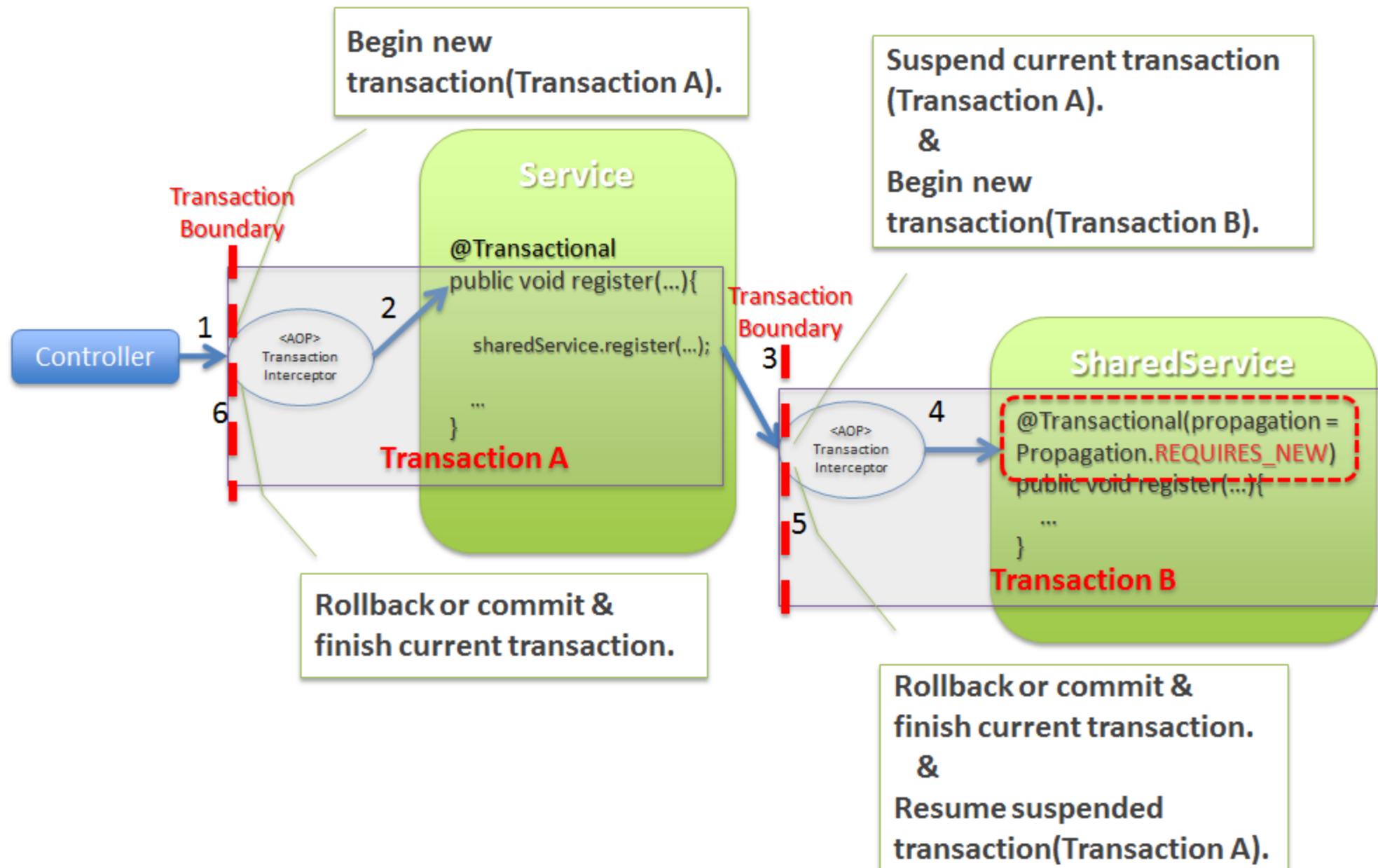


Вопросы?

Propagation (как вкладываются)



Propagation



Propagation

- Аспекты Spring применяются только при вызове из другого бина
- Вызов private, да другого public не приведёт к созданию транзакций
- Cglib может, но Spring не использует его возможность.

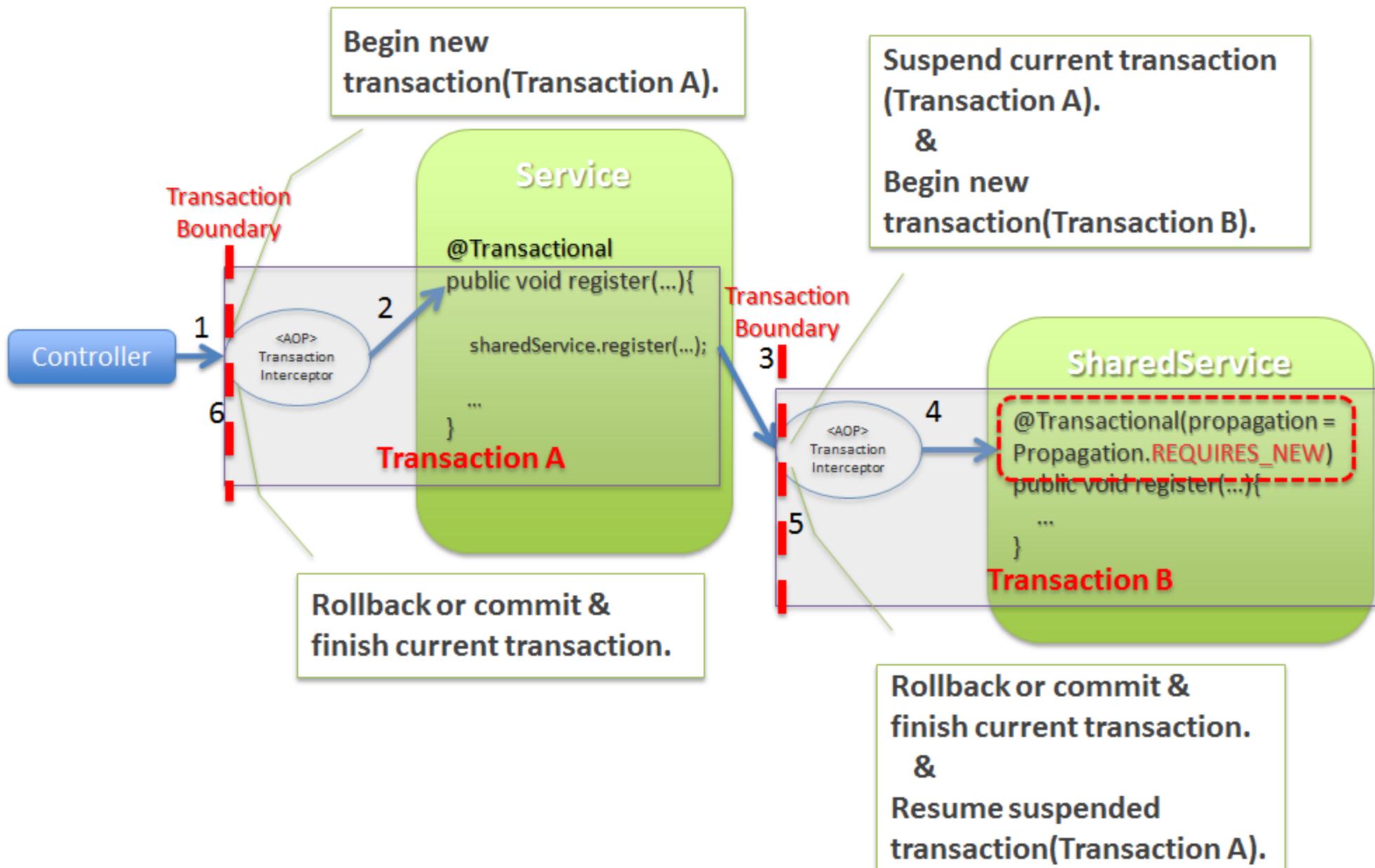
Propagation

- Required
- RequiresNew
- Mandatory
- NotSupported
- Supports
- Never
- Nested

Propagation (как вкладываются)

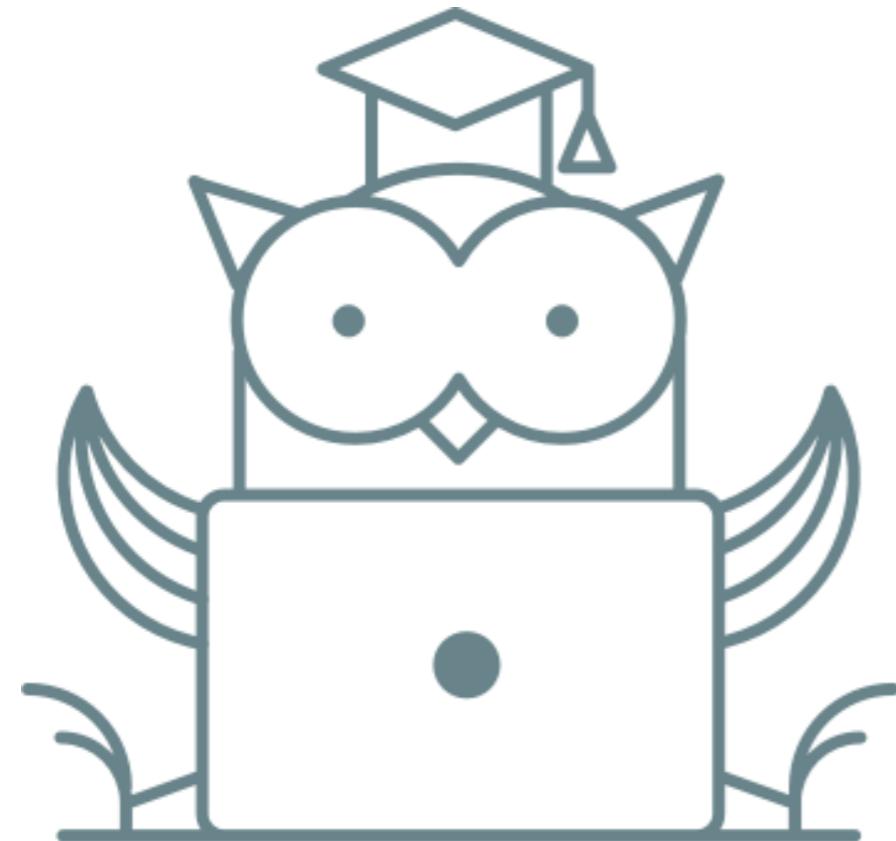
Propagation в @Transacitonal	Вызывающий метод	Вызываемый метод
Required – если транзакция начата – использует её. Если нет – начинает новую	No	T2
	T1	T1
RequiresNew – если метод вызван в транзакции – приостанавливает её и начинает новую, как только метод заканчивается, то исходная транзакция возобновляется.	No	T2
	T1	T2
Mandatory – если нет транзакции, то бросается exception.	No	Error
	T1	T1
NotSupported – приостанавливает текущую транзакцию, если была начата.	No	No
	T1	No
Supports – использует текущую транзакцию, если не была начата, то не открывает новую.	No	No
	T1	T1
Never – если запускается в транзакции, то бросается исключение.	No	No
	T1	Error
Nested – запускается во внутренней транзакции, если текущая существует,	No	T2
	T1	T2 (внутр.)

REQUIRES_NEW



Для чего нужны?

Propagation в @Transacitonal	Вызывающий метод	Вызываемый метод
Required – если транзакция начата – использует её. Если нет – начинает новую	No	T2
	T1	T1
RequiresNew – если метод вызван в транзакции – приостанавливает её и начинает новую, как только метод заканчивается, то исходная транзакция возобновляется.	No	T2
	T1	T2
Mandatory – если нет транзакции, то бросается exception.	No	Error
	T1	T1
NotSupported – приостанавливает текущую транзакцию, если была начата.	No	No
	T1	No
Supports – использует текущую транзакцию, если не была начата, то не открывает новую.	No	No
	T1	T1
Never – если запускается в транзакции, то бросается исключение.	No	No
	T1	Error
Nested – запускается во внутренней транзакции, если текущая существует,	No	T2
	T1	T2 (внутр.)



Вопросы?

Rollback транзакций

```
@Service
public class OrderServiceImpl implements OrderService {

    @Autowired
    private Dao<OrderItem> dao;

    @Override
    @Transactional(rollbackFor = InvalidOrderItemException.class)
    public void persistOrders(List<OrderItem> orderItems) throws InvalidOrderItemException {
        for (OrderItem orderItem : orderItems) {
            if (orderItem.getQty() > 100) {
                throw new InvalidOrderItemException(
                    "Order quantity cannot be more than 100, found: "
                    + orderItem.getQty());
            }
            long id = dao.save(orderItem);
            System.out.println("id generated: " + id);
        }
    }
}
```

```
1  @Service
2  public class StoreService {
3      @PersistenceContext
4      EntityManager entityManager;
5
6      @Transactional(propagation = Propagation.REQUIRED)
7      public void createPerson() throws Exception {
8          Personal personal = new Personal();
9          personal.setName("required");
10         entityManager.persist(personal);
11         addNewPerson();
12
13     }
14
15     @Transactional(propagation = Propagation.REQUIRES_NEW, rollbackFor =
Exception.class)
16     public void addNewPerson() throws Exception{
17         Personal personal = new Personal();
18         personal.setName("name_required_new");
19         entityManager.persist(personal);
20         throw new Exception();
21     }
22 }
23 }
```

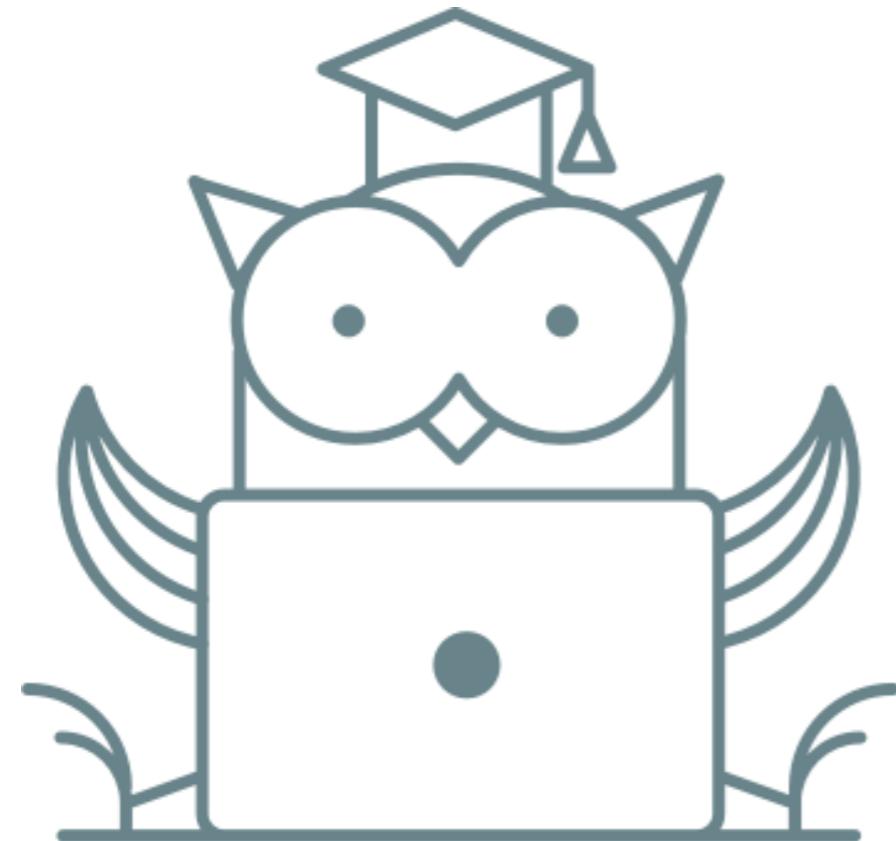
Почему при вызове метода `addNewPerson()` из метода `createPerson()` не создается новая транзакция, несмотря на `Propagation.REQUIRES_NEW`.

И даже при наличии `rollbackFor = Exception.class` в базу коммитится две сущности `Personal`

TransactionTemplate

```
1 @Override  
2 public final void updateGreet(final Greeter g, final String newTarget) {  
3     transactionTemplate.execute(new TransactionCallbackWithoutResult() {  
4         @Override  
5         protected void doInTransactionWithoutResult(TransactionStatus status) {  
6             Greeter greet = em.merge(g);  
7             greet.setTarget(newTarget);  
8  
9             status.setRollbackOnly();  
10        }  
11    });  
12 }
```

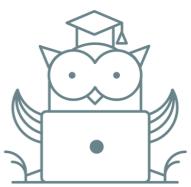
```
1 @Override  
2 public final List<Greeter> getGreetings() {  
3     TransactionTemplate tx = new TransactionTemplate(transactionManager);  
4     tx.setReadOnly(true);  
5     return tx.execute(status -> em.createQuery("from Greeter", Greeter.class)  
6           .getResultList());  
7 }
```

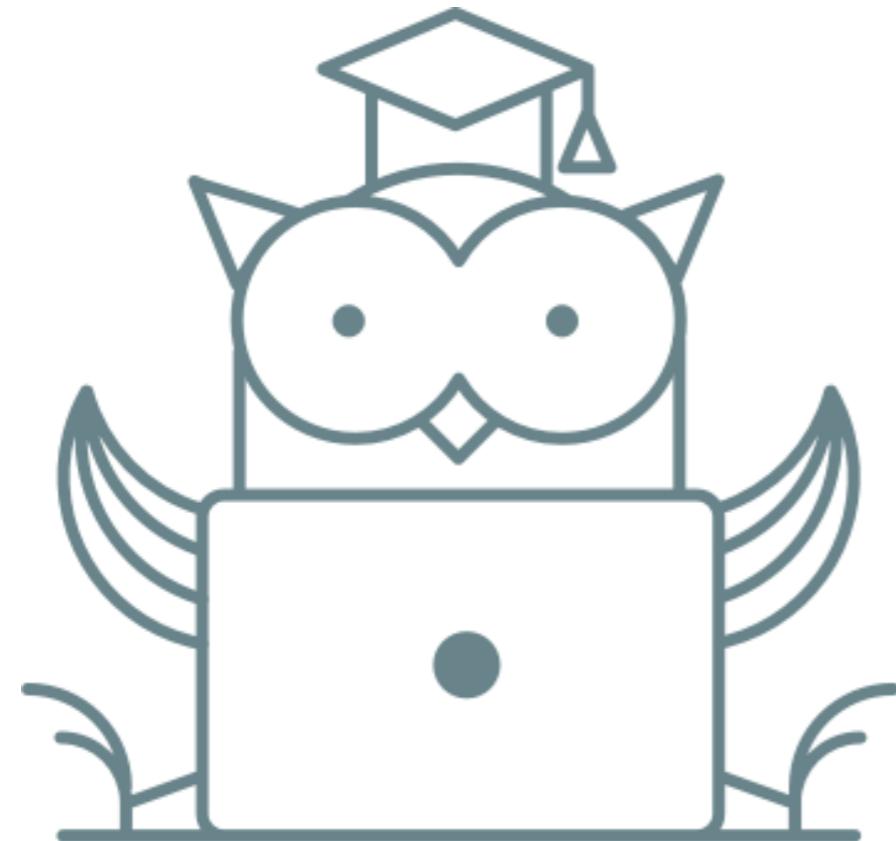


Вопросы?

Домашнее задание

Нет

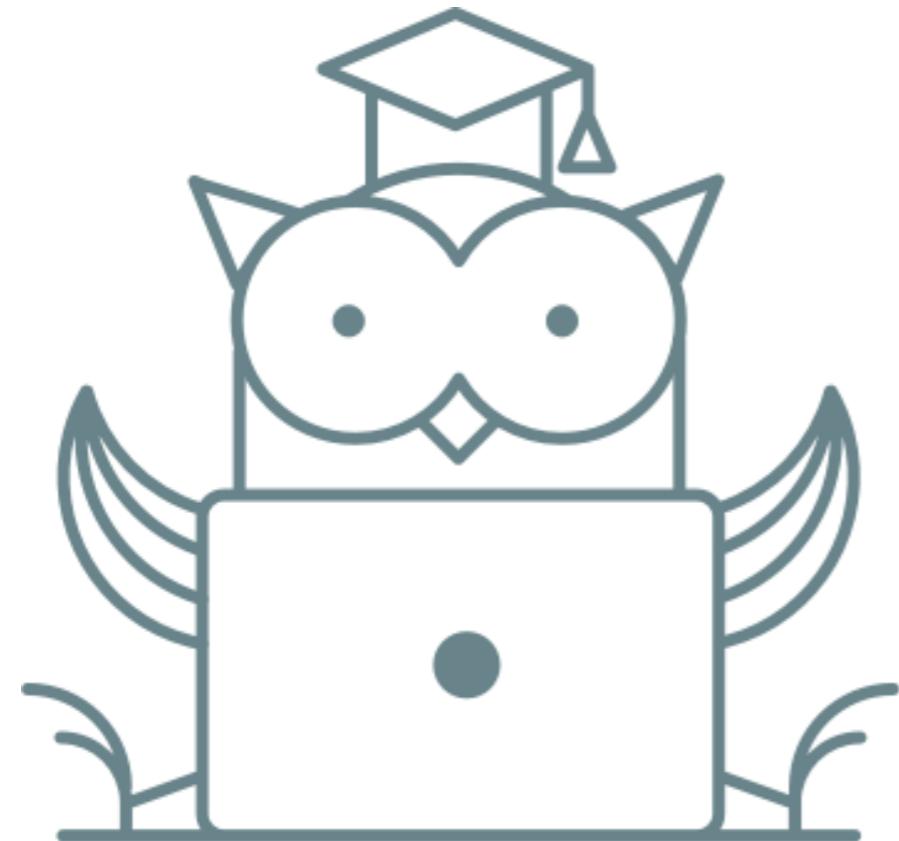




Вопросы?

Пожалуйста, пройдите опрос

<https://otus.ru/polls/1558/>



Спасибо
за внимание!

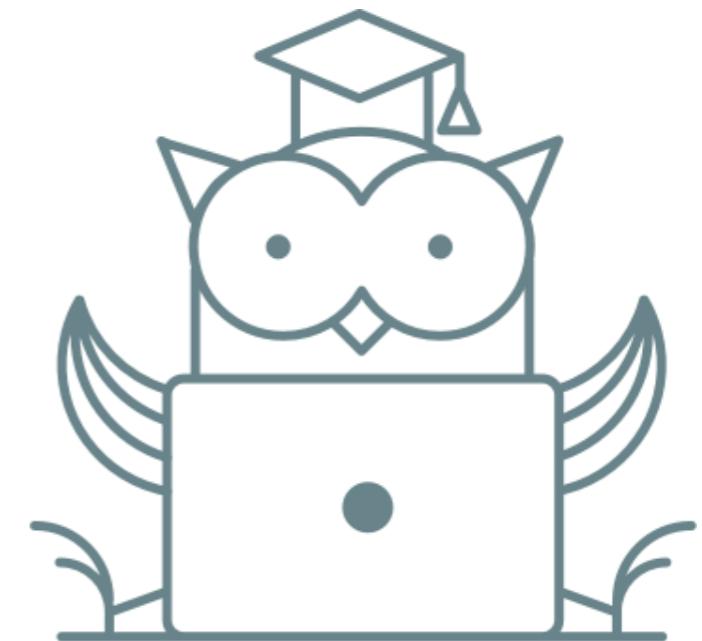
JPA Вам!

O ·Τ· U S

ОНЛАЙН-ОБРАЗОВАНИЕ

10 – Белая магия Spring Data: Spring Data JPA

Дворжецкий Юрий



Как меня слышно && видно?



Если нет – напишите, если слышите – смайлик в чат.



Цели:

- Познакомиться немного с проектом Spring Data
- Познакомиться плотно с подпроектом Spring Data JPA
- Немного попрактиковаться.



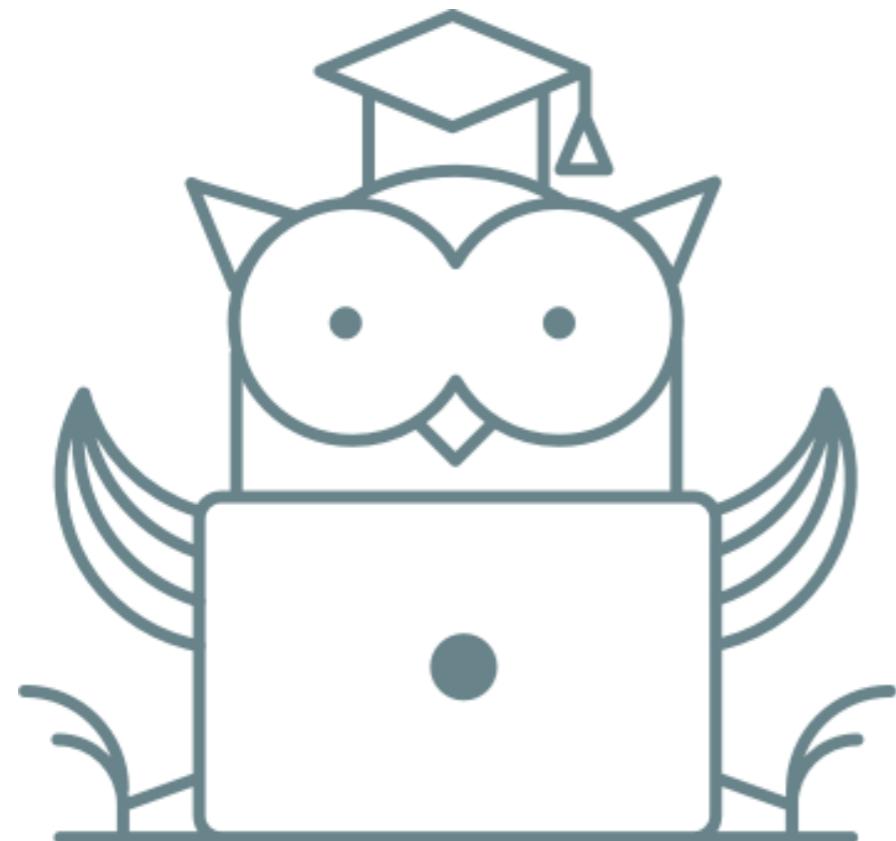
Планы:

- Теория – не жесть, практика (как и дальнейшая жизнь) – просто курорт.
- Считаем, что JPA теперь знаем.
- Проходим Spring.
- Домашняя работа есть, эта ДЗ не засчитывает предыдущую.

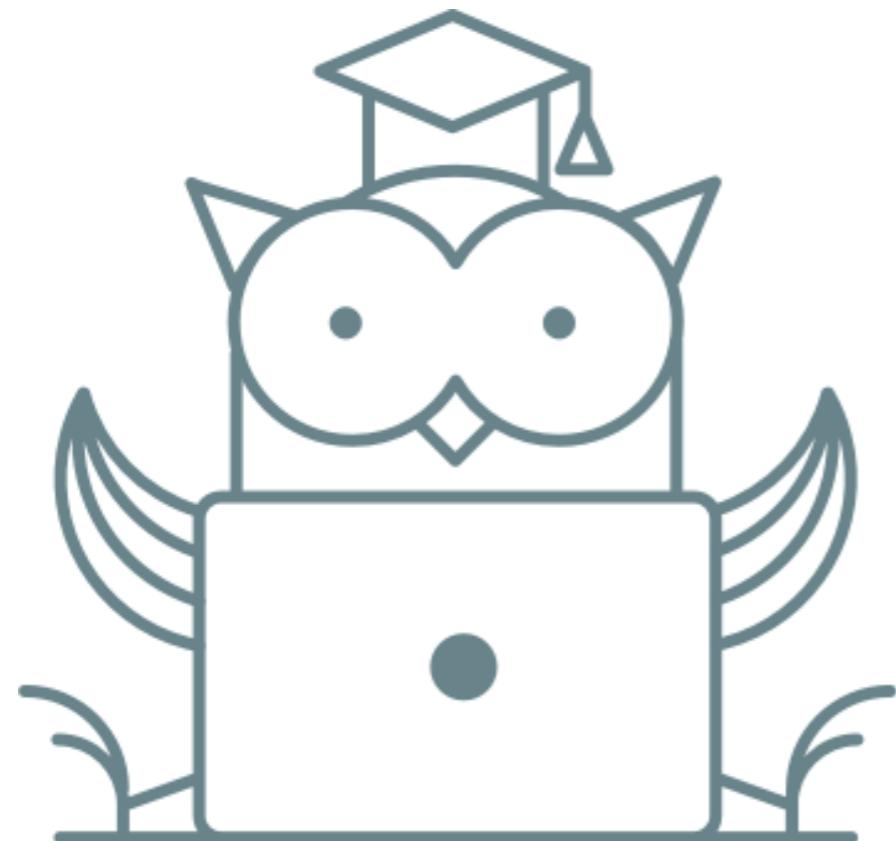


Совсем чуть-чуть орг.вопросов:

- Искренне благодарен за поддержку на время свадьбы.
- Небольшой завал мы разгребли.
- У нас уже коммьюнити 😊
- Со всех отзыв!



Поехали?



Spring Data

Abstract DAO

- DAO – слой DataSource, когда используется JDBC
- DAO похожи между собой, кто-то пробовал сделать.
- В действительности, особенно, когда используются особенности БД – нет.
- Поэтому, как ни странно, лучше не писать Abstract DAO и не городить наследования сервисов.

Abstract Repository

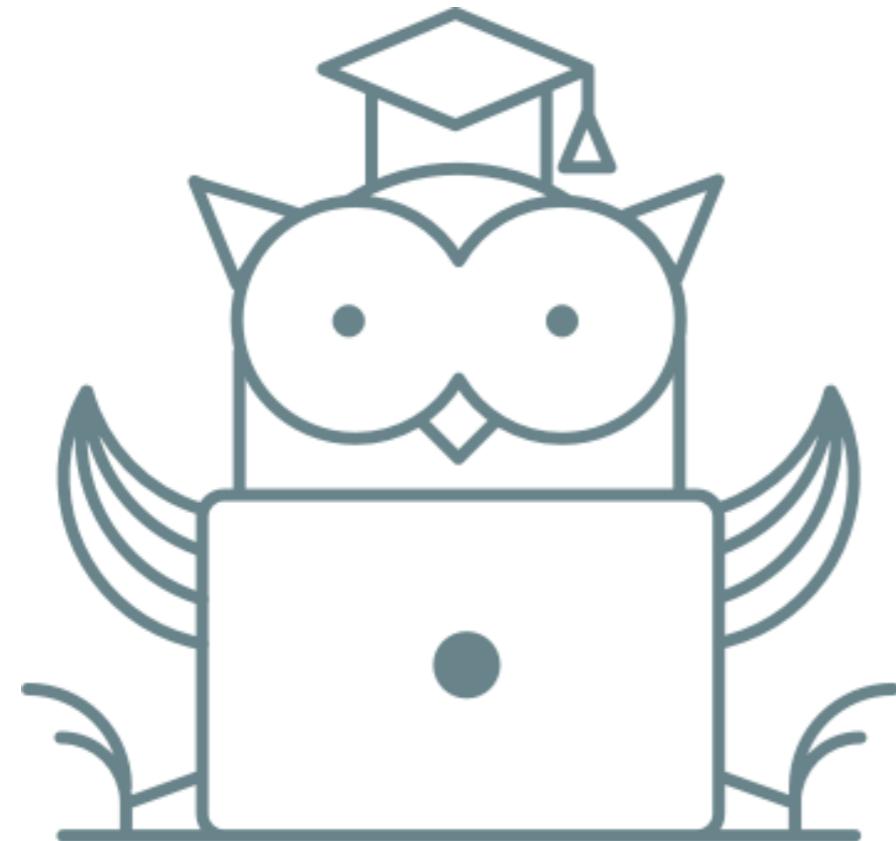
- Repository – слой Business, когда используется ORM
- С JPA совсем всё получается похожим
- И это действительно так.
- Можно писать AbstractRepository
- А можно всё сделать на интерфейсах и на кодогенерации

Spring Data

- Spring Data – это целый проект Spring, состоит из других подпроектов Spring
- Spring Data как раз и вводит понятие абстрактного репозитория
- Spring Data Commons – базовое понятие репозитория
- Spring Data JPA – реализация репозиториев на JPA
- Spring Data JDBC - JDBC-based repositories

Spring Data

- Spring Data KeyValue – для Map-based repositories
- Spring Data MongoDB – MongoDB
- REST, Redis, Cassandra, ElasticSearch, Hazelcast и куча других



Вопросы?



Spring Data JPA

Магия spring-data

- Позволяет писать минимум кода
- Нереально крутая вещь для проектов
- Особенно, когда Вы интегрируетесь с NoSQL БД
- Пока будем владеть магией spring-data-jpa

Шаг 1:

Магия подключается так

- spring-data-* для модуля *
- spring-boot-starter-data-*

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>2.0.9.RELEASE</version>
</dependency>
```

Шаг 2:

Пишем entity

```
@Entity
public class Employee {

    private @Id @GeneratedValue Long id;
    private String firstName, lastName, description;

    private Employee() {}

    public Employee(String firstName, String lastName, String description) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.description = description;
    }
}
```

Шаг 3:

Пишем интерфейс репозитория

```
public interface EmployeeRepository extends CrudRepository<Employee, Long> {  
  
    Employee findByFirstName(String firstName);  
  
    List<Employee> findByLastName(String lastName);  
}
```

Шаг 4: Добавляем аннотацию

```
@SpringBootApplication  
public class MyApp {  
  
    public static void main(String[] args) {  
        SpringApplication.run(MyApp.class, args);  
    }  
}
```

`@EnableJpaRepositories`

Шаг 5:

Используем!

```
    @Autowired  
    private PersonRepository repository;  
  
    @PostConstruct  
    public void init() {  
        repository.save(new Person( name: "Pushkin" ));  
    }  
}
```

Упражнение

- Где код?

```
public interface EmployeeRepository extends CrudRepository<Employee, Long> {  
  
    Employee findByFirstName(String firstName);  
  
    List<Employee> findByLastName(String lastName);  
}
```

Часть методов CrudRepository

```
package org.springframework.data.repository;

import java.util.Optional;

/**
 * @NoRepositoryBean
 */
public interface CrudRepository<T, ID> extends Repository<T, ID> {

    /**
     * <S extends T>. S save(S entity);

    /**
     * <S extends T>. Iterable<S>. saveAll(Iterable<S>. entities);

    /**
     * Optional<T>. findById(ID id);

    /**
     * boolean existsById(ID id);

    /**
     * Iterable<T>. findAll();

    /**
     * Iterable<T>. findAllById(Iterable<ID>. ids);
}
```

CrudRepository

- Методы CrudRepository перегружать можно, но по правилам:

`Iterable<T> findAll(); -> List<Person> findAll();`

- Если хотите написать только Ваши методы (например без save), то наследуйтесь от Repository
- Если хотите зачем-то использовать интерфейс и не реализовывать JPA репозиторий, то можно написать `@NoDataRepository`

Генерация методов

- Остальные методы генерируются по имени
- Правила генерации хитрые
- Генерируются по имени в JPQL (!)
- Компилируются в рантайме

Язык методов («таблица глаголов»)

Лексема	Пример	JPQL
And	findByNameAndEmail	where x.name = ?1 and x.email = ?2
Or	findByNameOrEmail	where x.name = ?1 or x.email = ?2
Is, Equals	findByName, findByNameIs	where x.name = ?1
Between	findByPriceBetween	where x.price between ?1 and ?2
LessThan	findByPriceLessThan	where x.price < ?1
GreaterTh an	findByPriceGreaterThan	where x.price > ?1
Like	findByNameLike	where x.name like ?1
IsNull	findByPeriodIsNull	where x.period is null
After	findByStartDateAfter	where x.startDate > ?1

Упражнение

- <https://github.com/ydvorzhetskiy/spring-framework-10>
- Сделайте магию по имени

PagingAndSortingRepository

```
package org.springframework.data.repository;

import ...
/*
 * ...
 */
@NoRepositoryBean
public interface PagingAndSortingRepository<T, ID> extends CrudRepository<T, ID> {

    /*
     * ...
     */
    Iterable<T>.findAll(Sort sort);

    /*
     * ...
     */
    Page<T>.findAll(Pageable pageable);
}
```

PagingAndSortingRepository

```
repository.findAll(  
    Sort.by(Sort.Direction.ASC, ...properties: "name", "id"));  
  
repository.findAll(PageRequest.of( page: 2, size: 20));
```

PageRepository

- Методы CrudRepository перегружать можно, но по правилам:

`Iterable<T> findAll(); -> List<Person> findAll();`

- Если хотите написать только Ваши методы (например без save), то наследуйтесь от Repository

Упражнение

- <https://github.com/ydvorzhetskiy/spring-framework-10>
- Сделайте PagingRepository для Email

Собственные методы

```
public interface AccountRepository
    extends JpaRepository<Account, Long>, AccountRepositoryCustom { ... }
```

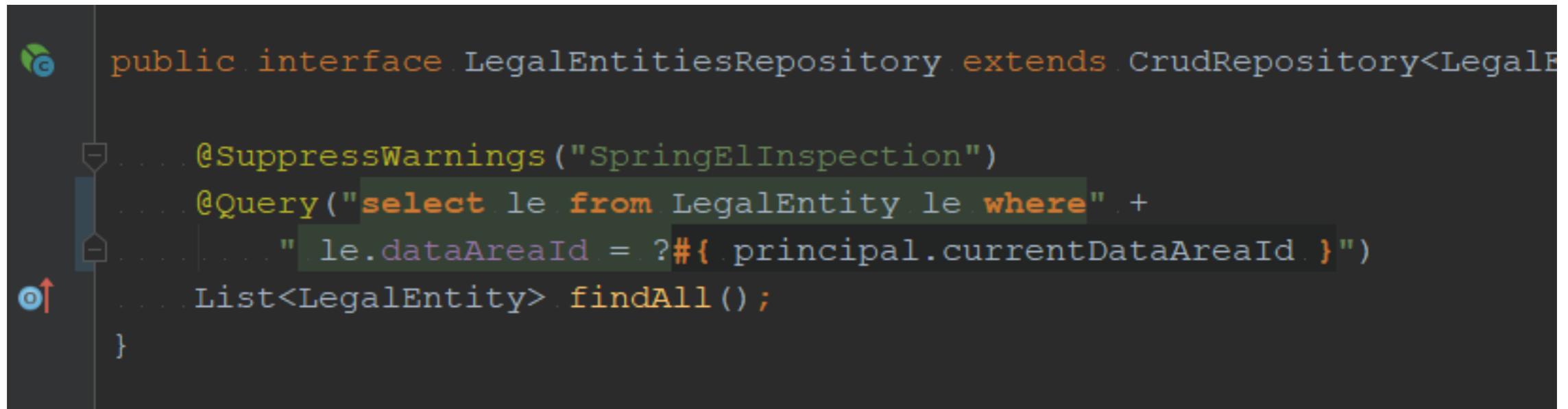
```
public interface AccountRepositoryCustom {
    public void customMethod();
}
```

```
public class AccountRepositoryImpl implements AccountRepositoryCustom {

    @Autowired
    AccountRepository accountRepository; /* Optional - if you need it */

    public void customMethod() { ... }
}
```

@Query



```
public interface LegalEntitiesRepository extends CrudRepository<LegalEntity, Long> {

    @SuppressWarnings("SpringElInspection")
    @Query("select le from LegalEntity le where" +
           " le.dataAreaId = ?#{ principal.currentDataAreaId }")
    List<LegalEntity> findAll();
}
```

@DataJpaTest

```
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;
import org.springframework.test.context.junit4.SpringRunner;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;

@RunWith(SpringRunner.class)
@DataJpaTest
@Transactional(propagation = Propagation.NOT_SUPPORTED)
public class ExampleNonTransactionalTests {

}
```

@DataJpaTest

```
import org.junit.*;
import org.junit.runner.*;
import org.springframework.boot.test.autoconfigure.orm.jpa.*;

import static org.assertj.core.api.Assertions.*;

@RunWith(SpringRunner.class)
@DataJpaTest
public class ExampleRepositoryTests {

    @Autowired
    private TestEntityManager entityManager;

    @Autowired
    private UserRepository repository;

    @Test
    public void testExample() throws Exception {
        this.entityManager.persist(new User("sboot", "1234"));
        User user = this.repository.findByUsername("sboot");
        assertThat(user.getUsername()).isEqualTo("sboot");
        assertThat(user.getVin()).isEqualTo("1234");
    }

}
```

@DataJpaTest (можно h2 настроить)

```
@RunWith(SpringRunner.class)
@DataJpaTest
@AutoConfigureTestDatabase(replace=Replace.NONE)
public class ExampleRepositoryTests {
    // ...
}
```

Какие тесты добавлять?

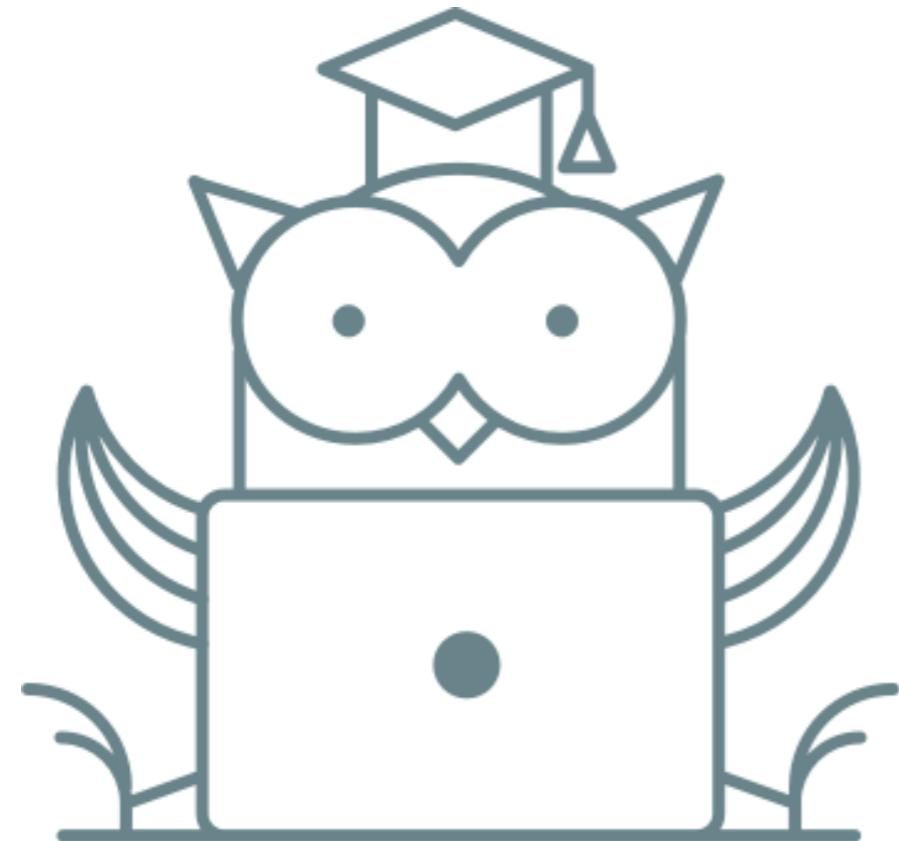
- Вставить и прочитать (можно с обоими методами JPA)
 - большинство ошибок маппинга отсюда уйдут
- Сложные JPQL Query
- Ваши собственные реализации методов

Упражнение

- <https://github.com/ydvorzhetskiy/spring-framework-10>
- Сделайте тест для Person (не забудьте spring-boot-test)

Про что умолчал

- Query DSL



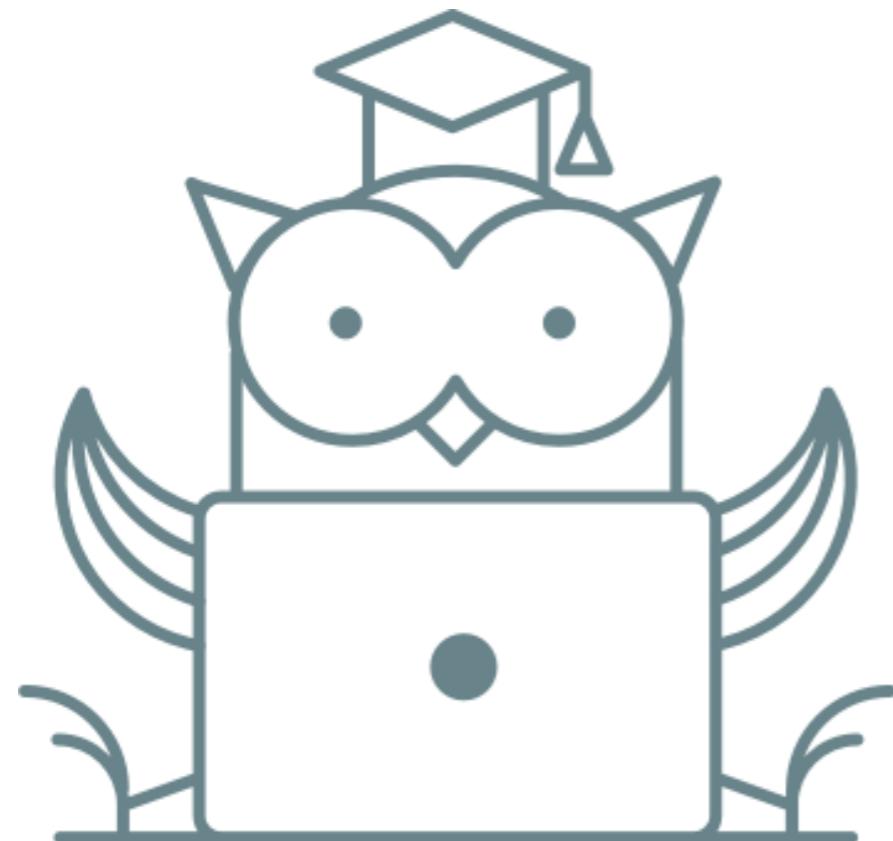
Вопросы?

Домашнее задание

Библиотеку на Spring Data JPA

Реализовать весь функционал работы с БД в приложении книг с использованием spring-data-jpa репозиториев.

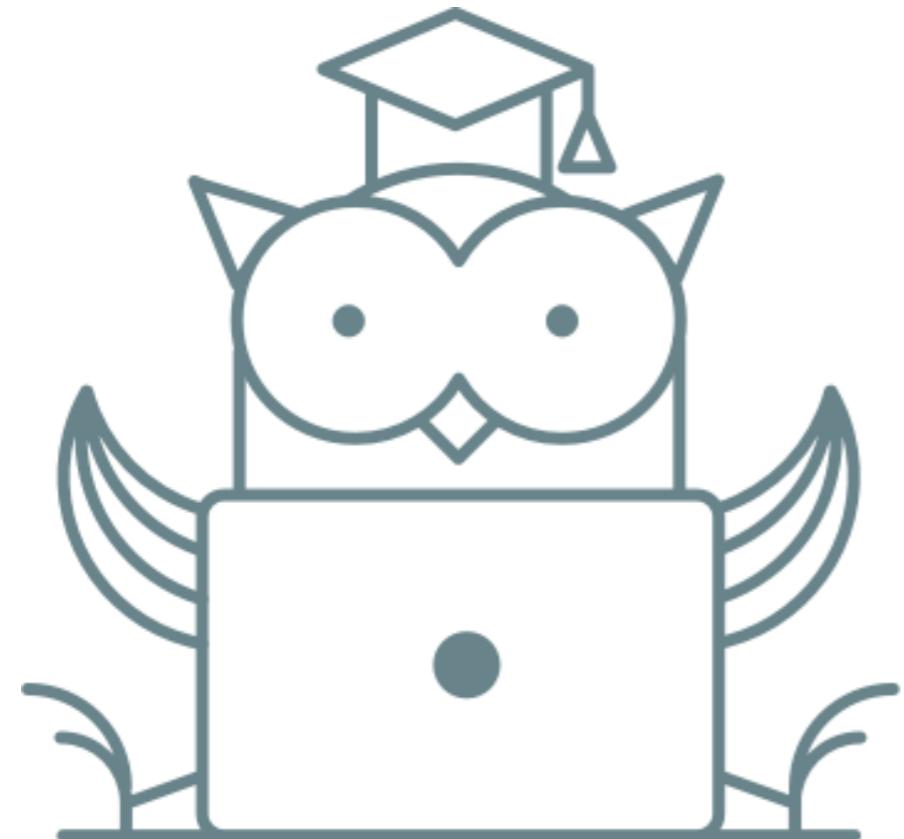




Вопросы?

Пожалуйста, пройдите опрос

<https://otus.ru/polls/1575/>



Спасибо
за внимание!

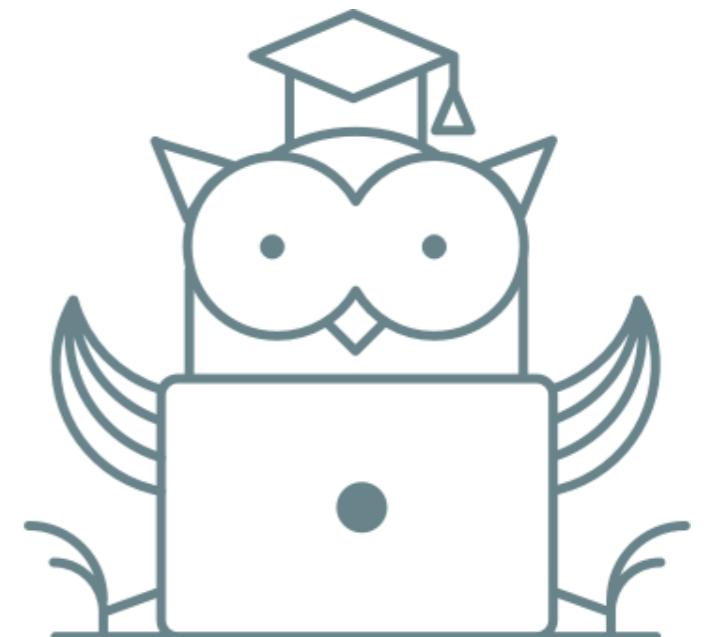
Spring Data Вам!

O ·Τ· U S

ОНЛАЙН-ОБРАЗОВАНИЕ

11 – SQL и NoSQL базы данных

Дворжецкий Юрий



Как меня слышно && видно?



Если нет – напишите, если слышите – смайлик в чат.



Цели:

- Познакомиться с NoSQL базами данных
- Ну и немного теории.
- А ещё покажу монгу, она прелесть)))



Планы:

- Теория – достаточно обзорная.
- Несмотря на кажущуюся простоту – это очень важно. Если Вам кажется, что очень тяжело – не переживайте, это просто 😊
- Самым шиком будет использовать БД в Вашем проекте и использовать правильно.
- ДЗ нет.



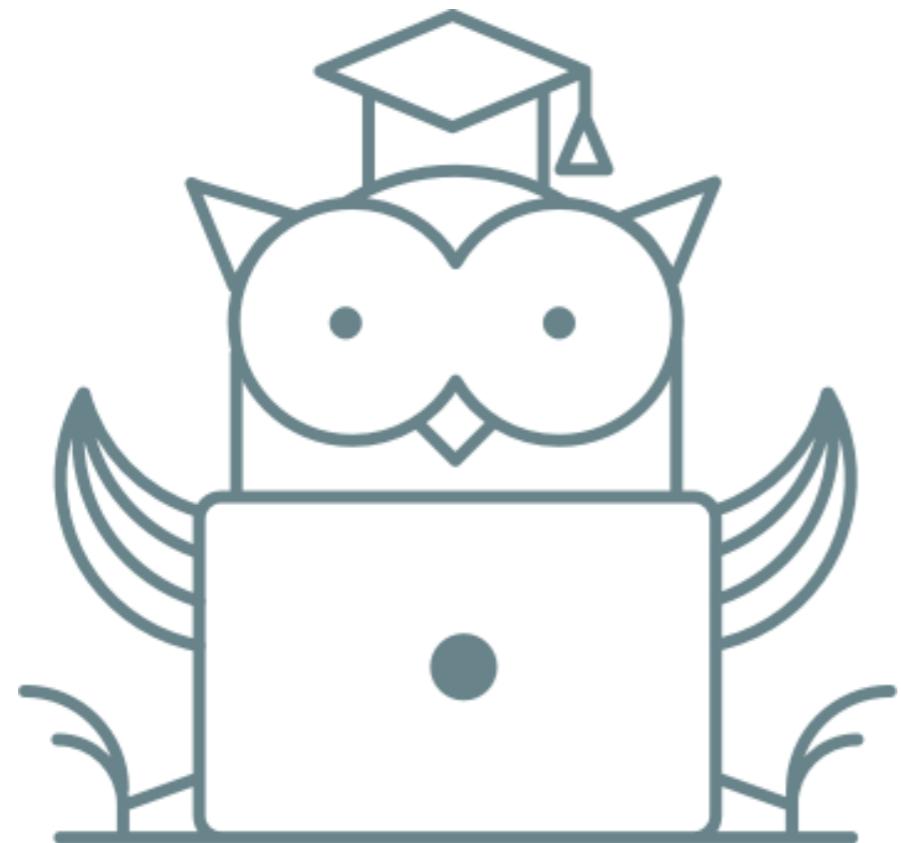
Планы:

- Сразу предупрежу, что я не проектировщик Бд.
- Честно, с некоторыми Бд, про которые говорю – не работал.
- Но важные вещи я Вам расскажу.

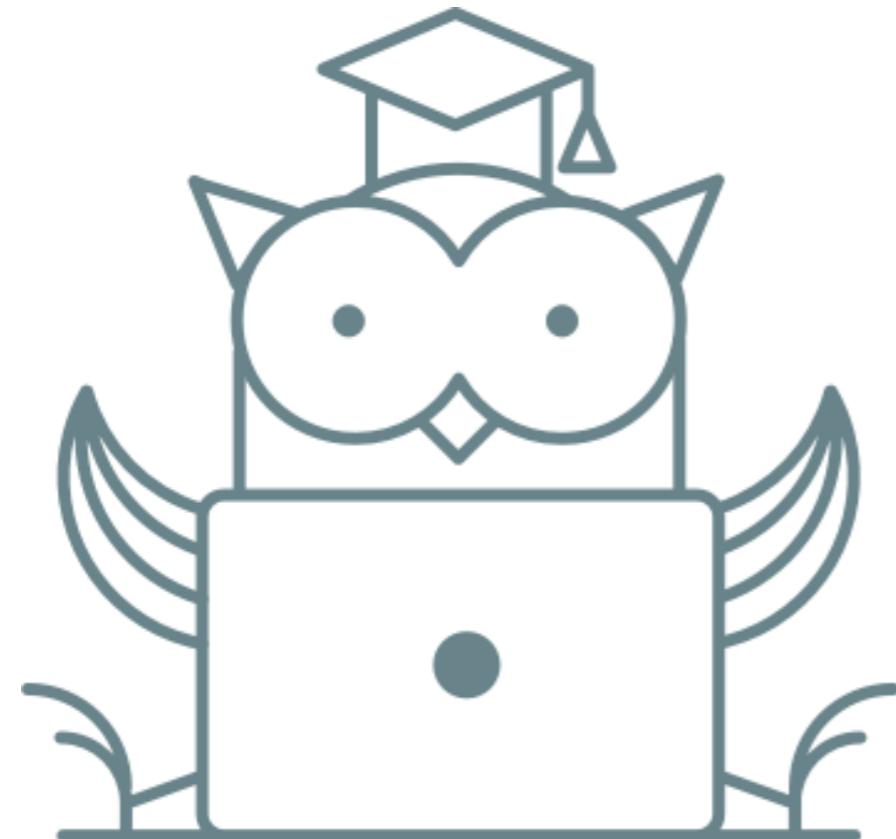


Совсем чуть-чуть орг.вопросов:

- Небольшая задержка – сегодня/завтра планировал разгрести.
- Пишите в общий чат ☺ Не стесняйтесь глупых вопросов ☺ И если Вы задали глупый вопрос – тоже не стесняйтесь, это нормально ☺
- Со всех отзыв!

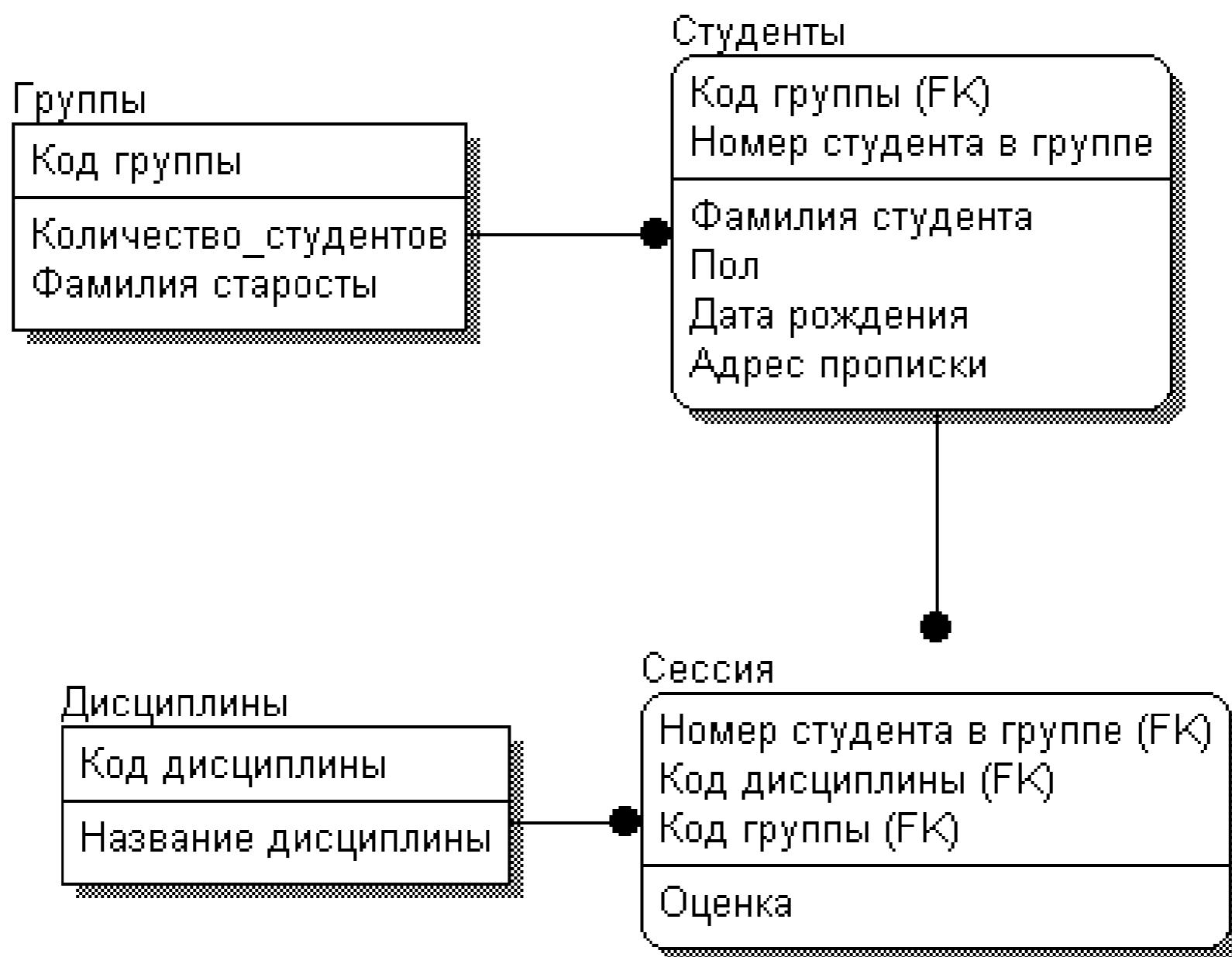


Поехали?



SQL и NoSQL БД

SQL DBs (RDB)



SQL DBs (RDB)

- Под SQL базами понимают реляционные базы данных, потому что запросы к ним пишутся на SQL языке
- Данные строго структурированы в виде таблиц
- У них есть ACID-транзакции
- Часто они развернуты в кластере (горизонтальное масштабирование) обеспечивающим большую надёжность (хотя для них достаточно тяжеловато)
- Вертикально масштабируются прекрасно.

SQL DBs (RDB)

- SQL DB появились давно, как и реляционные БД
- Поэтому все БД и используют SQL
- Они очень популярны (хотя когда я был в детском саду, уже тогда были нереляционные БД)
- Многие технологии выстроены вокруг них (*DBC, ORM)
- SQL – это просто интерфейс, Hadoop, кстати, имеет SQL интерфейс (Hive)

SQL DBs (RDB)

- MySQL, MariaDB
- SQLite, H2, Derby
- PostgreSQL
- Oracle
- MS SQL

Где используются SQL

- Если я скажу что «SQL базы данных используются для хранения табличных данных» то это не означает, что табличные данные нельзя больше нигде хранить.
- И в обратную сторону, если я скажу, что «JSON»-ы лучше хранить в MongoDB, то это не означает, что их нельзя хранить в других БД.
- Кстати, в PostgreSQL прекрасно хранятся JSON, причём по производительности уделяет монгу.

Где используются SQL

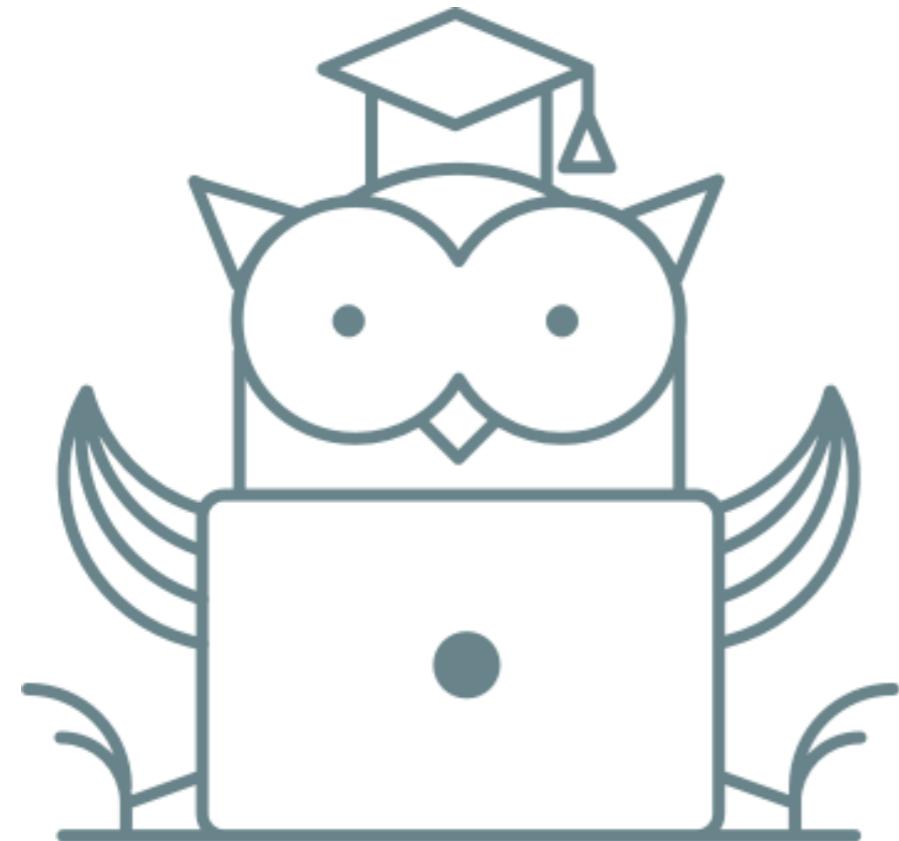
- Там где действительно важны транзакции. Банковские операции и т.д.
- Если данные реально представляют собой таблицы, а структура меняется редко.

Где используются SQL

- Как ни странно, SQL БД часто спокойно используются, там где сильны другие БД.
- JSON-ы (неструктурированные данные) можно прекрасно хранить во всех современных SQL DB и делать запросы в полям.
- Кэши (!), и даже для хранения графовых данных.

Где используются SQL

- Просто так реально удобнее
- Добавление в проект новой базы данных обычно накладно, да и приводит к некоторым последствиям.
- Хотя иногда старт быстрее с NoSQL БД (с правильно выбранной)
- А вот когда есть конкретная задача, и эта задача становится критичной, то тогда появляются..... NoSQL



Вопросы?

HOW TO WRITE A CV



Leverage the NoSQL boom

NoSQL

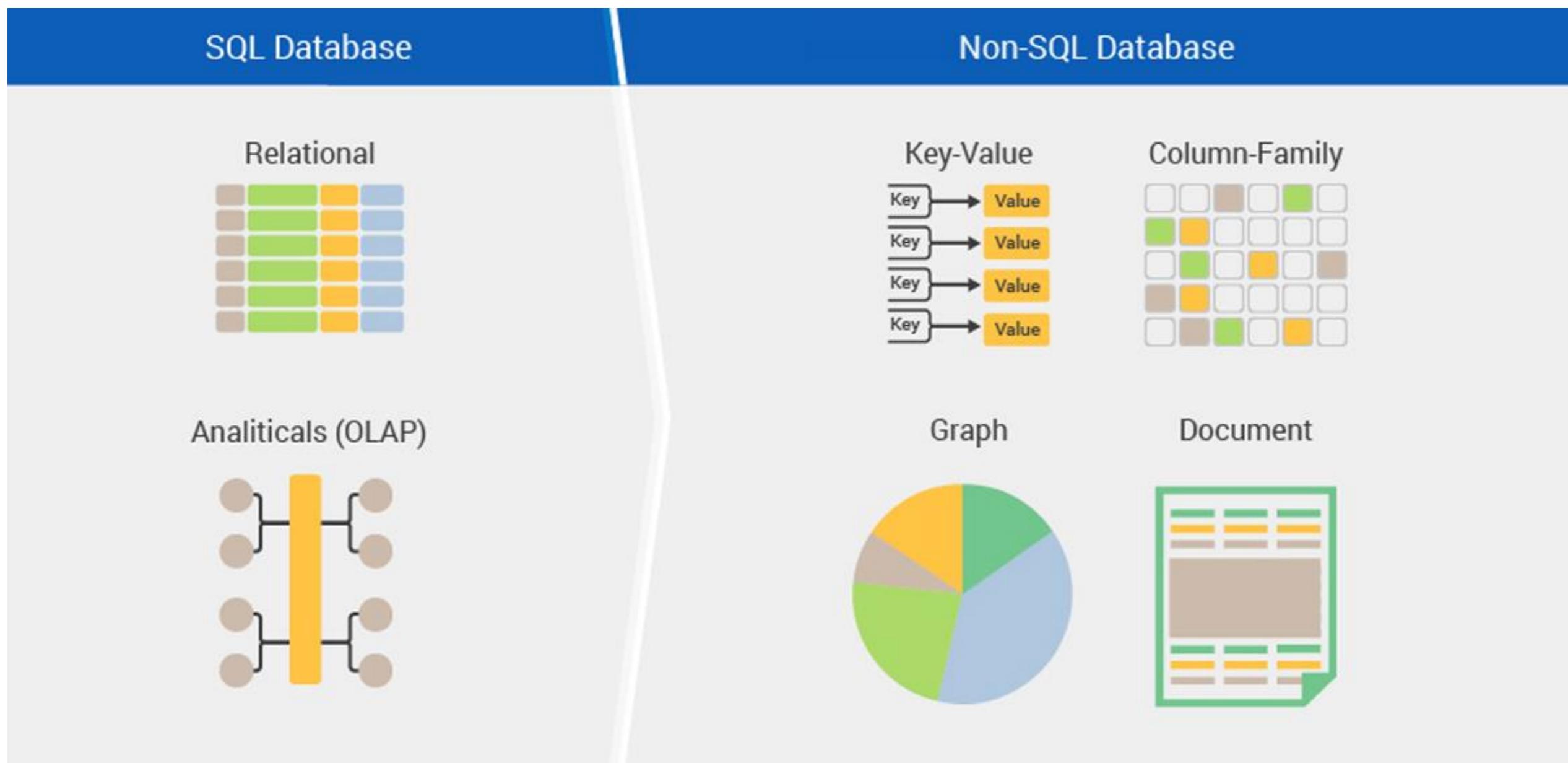
NoSQL

- NoSQL – это «buzzword» 😊 общее название для баз данных, не использующих реляционную модель
- Но мы то знаем, что SQL и реляционная модель – это такие разные вещи)
- Они были давно, но использовались ну в оооочень специфичных ситуациях.
- Получили широкое распространение с популярностью BigData
- Ну и сейчас в NoSQL БД записали, то что не является БД как таковыми – например, распределённые кэши.

Принципы хранения баз данных

- Каждая БД выбирает свои принципы хранения данных.
- Часто даже одни и те же принципы хранения разительно отличаются в разных БД.
- Ну и принципы одно – а интерфейс доступа к БД – другое.

Принципы хранения данных



Key-Value

- Key-value – одну такую Вы точно знаете, она называется HashMap 😊
- Это очень популярная структура, как и такие Бд
- Собственно так и выглядит Бд – по ключу Вы кладёте значение и получаете. Никаких связей
- Подобная простота как раз и оказывается предпочтительной для некоторых операций

Key-Value

- Они ёщё делятся на RAM, можно выделить eventually consistented и ordered
- таких баз реально много, есть хранилища, а есть кэши
- Кэши часто хранят данные только в памяти
- В зависимости от цели, они дико отличаются в реализации

Key-Value

- Redis
- MemcacheDB
- Aerospike
- Apache Ignite
- Hazelcast
- Riak

Когда применяются

- Когда нужен большой словарь ооочень большой.
- Или хранилище
- Когда нет связей, только get и put, типа кэш (HasMap)
- Ordered позволяют организовать что-то вроде очереди

Семейства столбцов

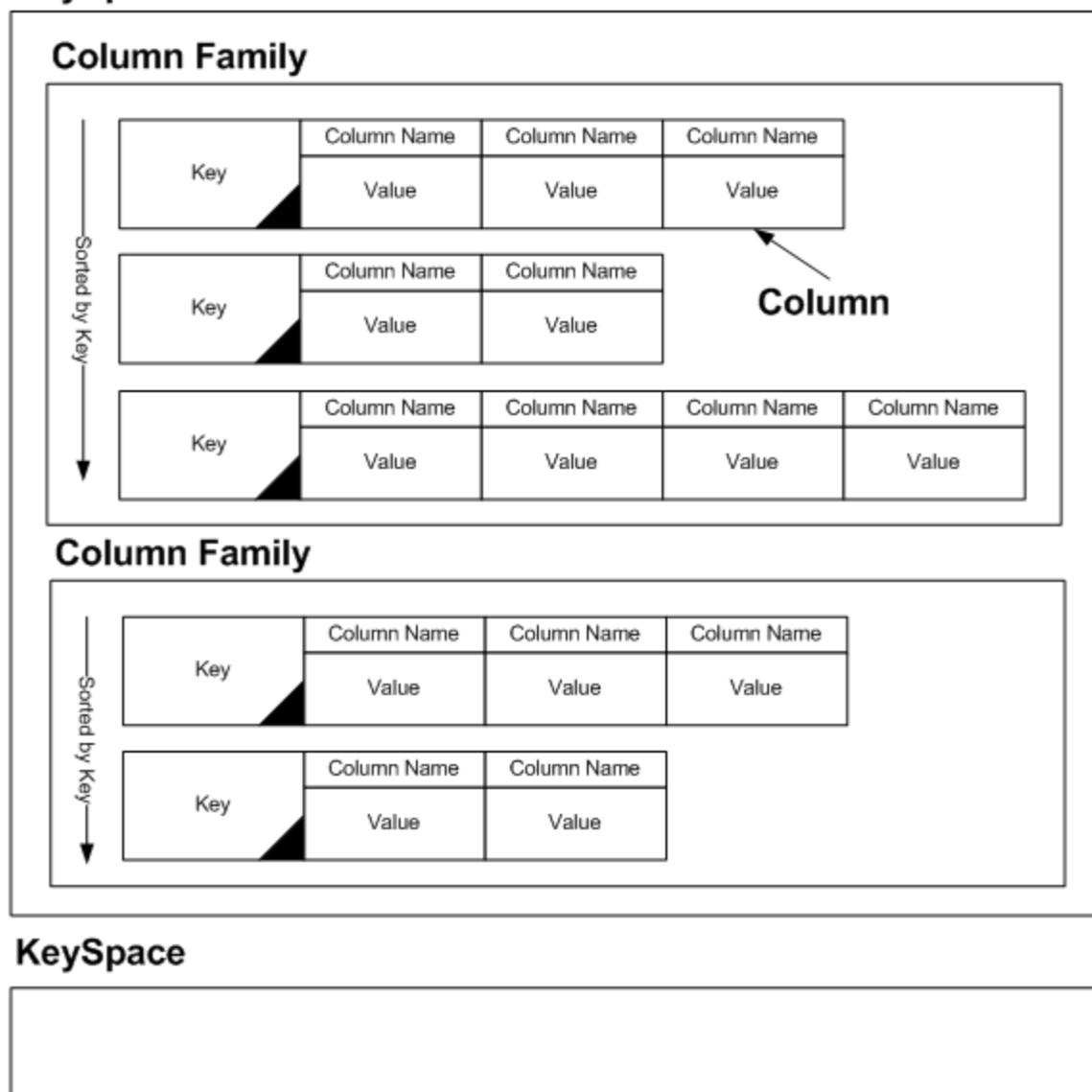
- Это табличные БД (оцените юмор)
- Скорее столбчатые)
- Можно представлять как таблицу строчек переменной длины в столбцах
- А семейство столбцов не определено
- Типа отметок в школьном журнале

Семейства столбцов

- Cassandra (создавалась для хранения временных данных)
- Очень чувствительна к времени (там это ключевое понятие).
- Рекомендую с ней познакомиться (одна из немногих NoSQL БД сертифицированная ФСТЭК)

Cassandra

KeySpace



Семейства столбцов

- HBase - хранилище для Hadoop
- Hadoop (да и окружение) огромная система для работы с BigData.
- Hbase это база Hadoop.

Где применяются

- Хранение (долговременное) больших наборов данных.
- И там где реально требуется масштабирование.

Документ-ориентированные

- У Вас есть структурированный документ (JSON)
- От записи к записи он может сильно различаться (описание товаров в интернет-магазине)*
- Документы могут содержать ссылки на другие документы
- А самое страшное, что они могут редактироваться и запросы могут делаться по полям (опять же. Которые могут отсутствовать)

Документ-ориентированные

- Couchbase
- CouchDB
- MongoDB – сейчас покажу (основа MEAN-stack)
- ElasticSearch – основная фишка – полнотекстовый поиск
- Informix – прав для документов
- Solr
- PostgreSQL ☺

DEMO

- MongoDB и Compass

Граф-ориентированные

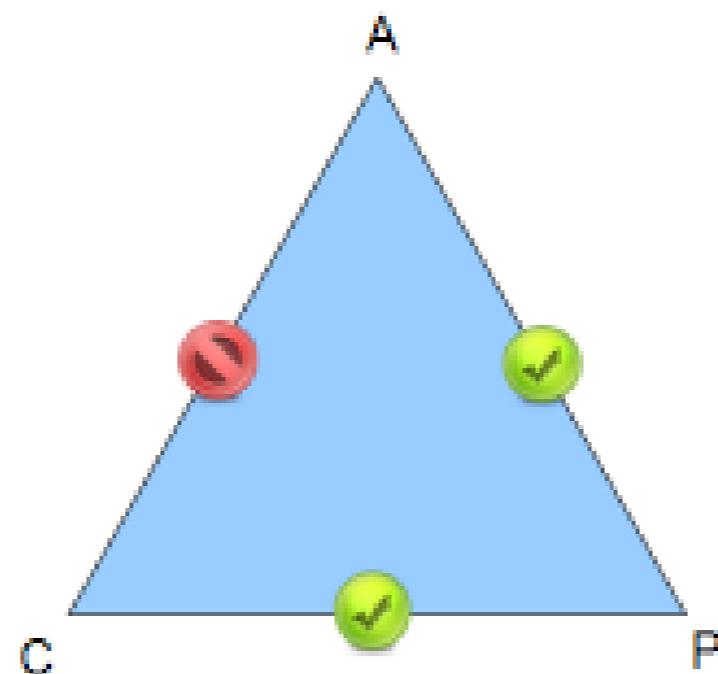
- Прямо граф – связей больше чем объектов (помним что квадрат)
- Neo4J – я с ней не работал, но многие говорят, что она эталонная.
- Рекомендую с ней познакомиться

Принципы построения

- Базы данных жертвуют не только структурой, но и принципами)
- Да, данные могут теряться)
- CAP-теорема

CAP-теорема

- <https://habr.com/post/136305/>



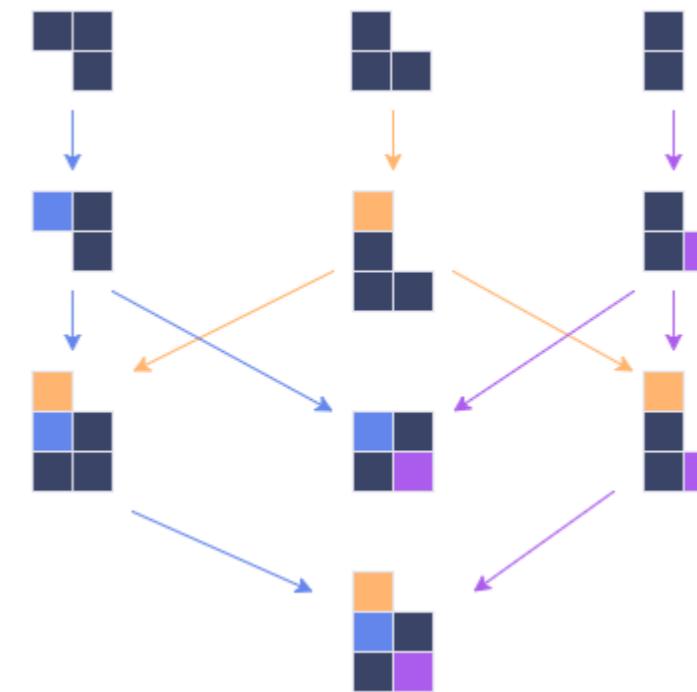
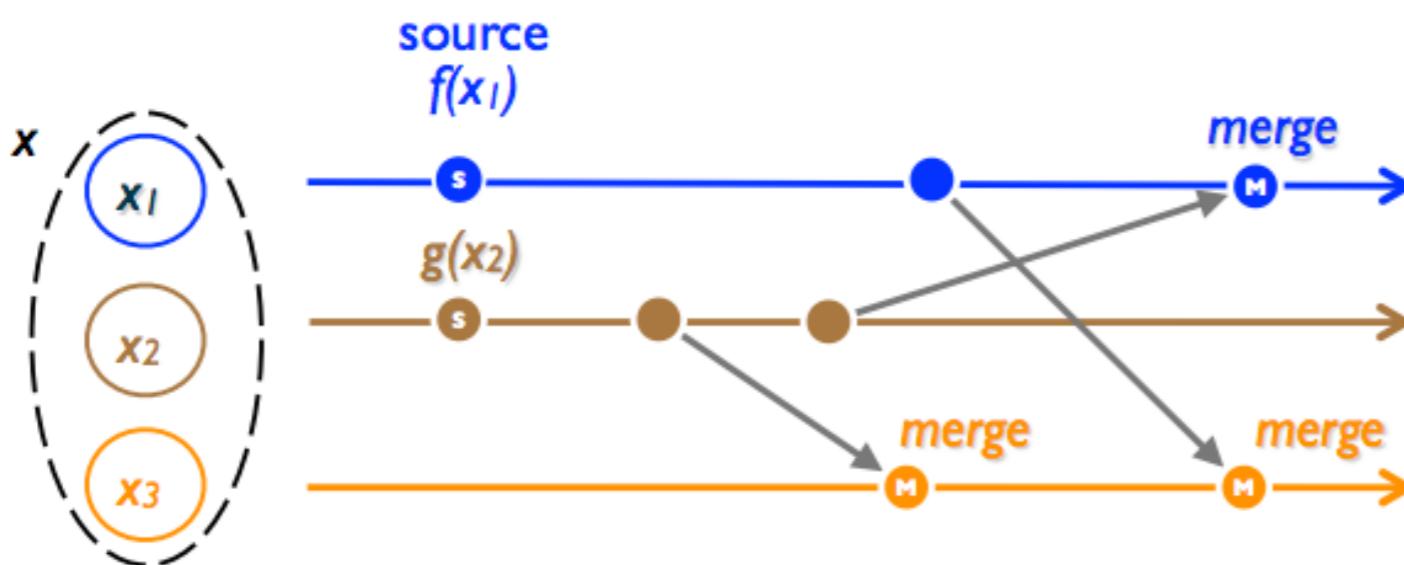
CAP

- **Consistency (Согласованность)**. Как только мы успешно записали данные в наше распределенное хранилище, любой клиент при запросе получит эти последние данные.
- **Availability (Доступность)**. В любой момент клиент может получить данные из нашего хранилища, или получить ответ об их отсутствии, если их никто еще не сохранял.
- **Partition Tolerance (Устойчивость к разделению системы)**. Потеря сообщений между компонентами системы (возможно даже потеря всех сообщений) не влияет на работоспособность системы. Здесь очень важный момент состоит в том, что если какие-то компоненты выходят из строя, то это тоже подпадает под этот случай, так как можно считать, что данные компоненты просто теряют связь со всей остальной системой.

Пример из жизни

- Отключили ноду кассандры
- Не очистили БД
- Когда подключили в кластер – пошли дубликаты данных

CRDT

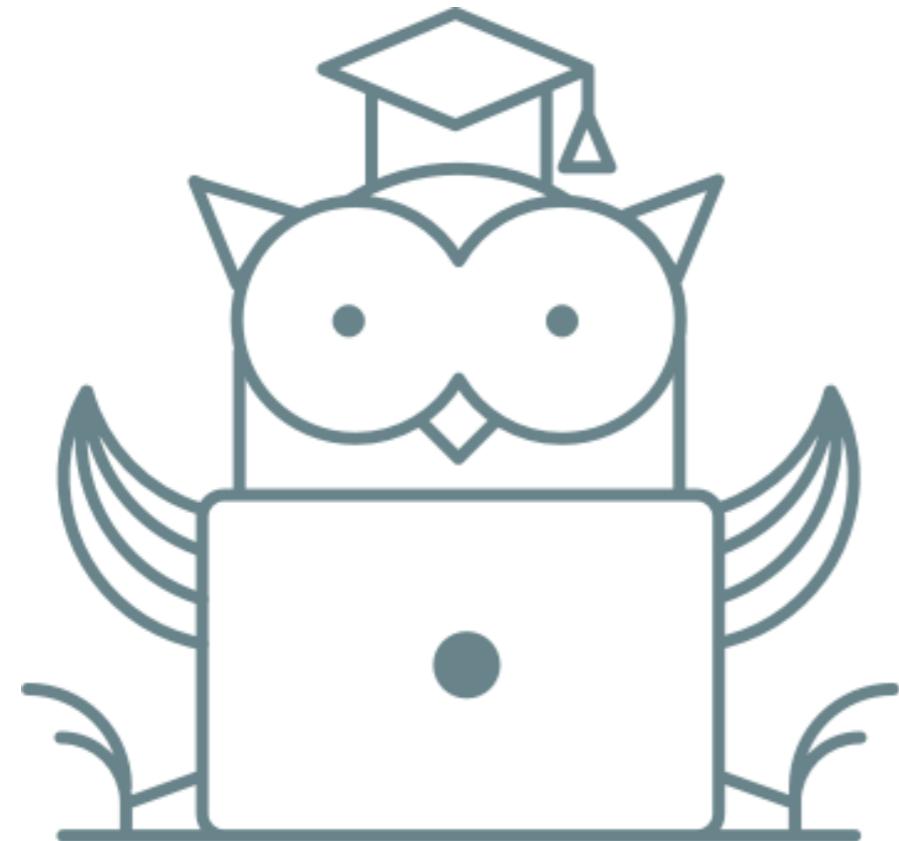


Украденная табличка

Тип хранилища данных	Сценарий использования	Пример	Рекомендации
Хранилище типа ключ-значение	Подходит для простых приложений, с одним типом объектов, в ситуациях, когда поиск объектов выполняют лишь по одному атрибуту.	Интерактивное обновление домашней страницы пользователя в Facebook.	Рекомендовано знакомство с технологией <code>memcached</code> . Если приходится искать объекты по нескольким атрибутам, рассмотрите вариант перехода к хранилищу, ориентированному на документы.
Хранилище, ориентированное на документы	Подходит для хранения объектов различных типов.	Транспортное приложение, оперирующее данными о водителях и автомобилях, работая с которым надо искать объекты по разным полям, например — имя или дата рождения водителя, номер прав, транспортное средство, которым он владеет.	Подходит для приложений, в ходе работы с которыми допускается реализация принципа «согласованность в конечном счёте» с ограниченными атомарностью и изоляцией. Рекомендуется применять механизм кворумного чтения для обеспечения своевременной атомарной непротиворечивости.

Украденная табличка

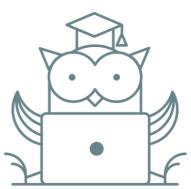
Система хранения данных с расширяемыми записями	Более высокая пропускная способность и лучшие возможности параллельной обработки данных ценой слегка более высокой сложности, нежели у хранилищ, ориентированных на документы.	Приложения, похожие на eBay. Вертикальное и горизонтальное разделение данных для хранения информации клиентов.	Для упрощения разделения данных используются HBase или Hypertable.
Масштабируемая RDBMS	Использование семантики ACID освобождает программистов от необходимости работать на достаточно низком уровне, а именно, отвечать за блокировки и непротиворечивость данных, обрабатывать устаревшие данные, коллизии.	Приложения, которым не требуются обновления или слияния данных, охватывающие множество узлов.	Стоит обратить внимание на такие системы, как MySQL Cluster, VoltDB, Clustrix, ориентированные на улучшенное масштабирование.

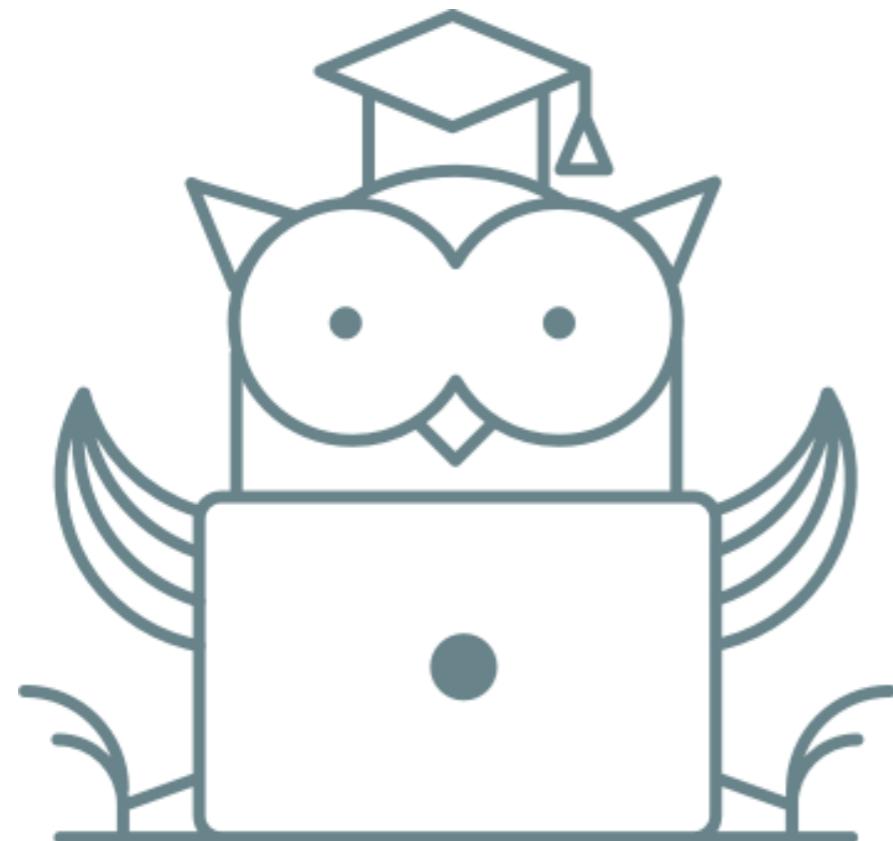


Вопросы?

Домашнее задание

Нет





Вопросы?

Пожалуйста, пройдите опрос

<https://otus.ru/polls/1611/>



Спасибо
за внимание!

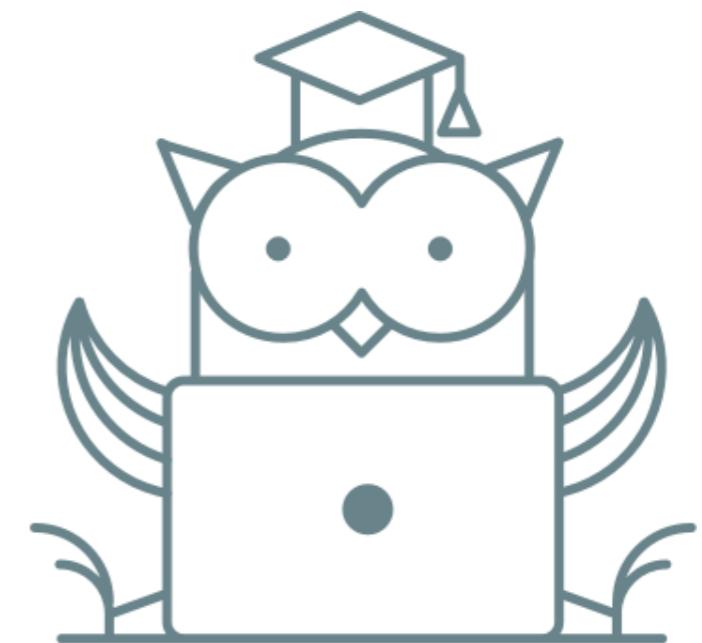
Spring Data Вам!

O ·Τ· U S

ОНЛАЙН-ОБРАЗОВАНИЕ

12 – Spring Data для подключения к NoSQL базам данных

Дворжецкий Юрий



Как меня слышно && видно?



Если нет – напишите, если слышите – смайлик в чат.



Цели:

- Познакомиться с Spring Data Key Value
- MongoDB
- Spring Data для MongoDb



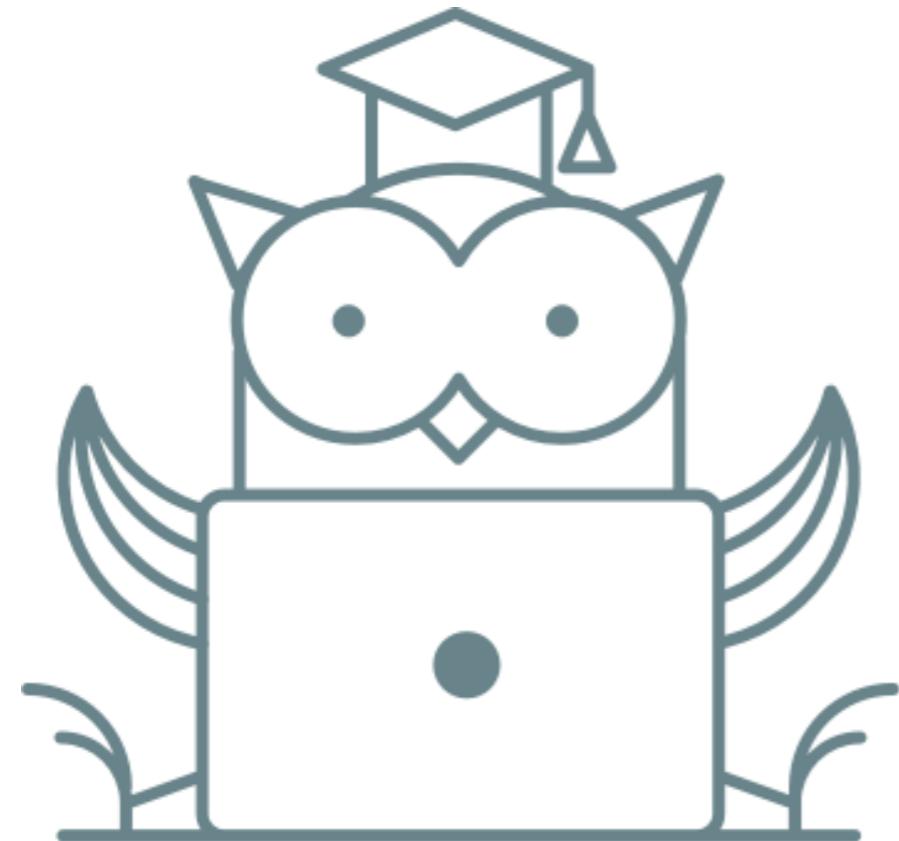
Планы:

- Пройдём то, что будем использовать на практике.
- Самый шик будет использовать это в проектах.
- ДЗ наконец-то есть.

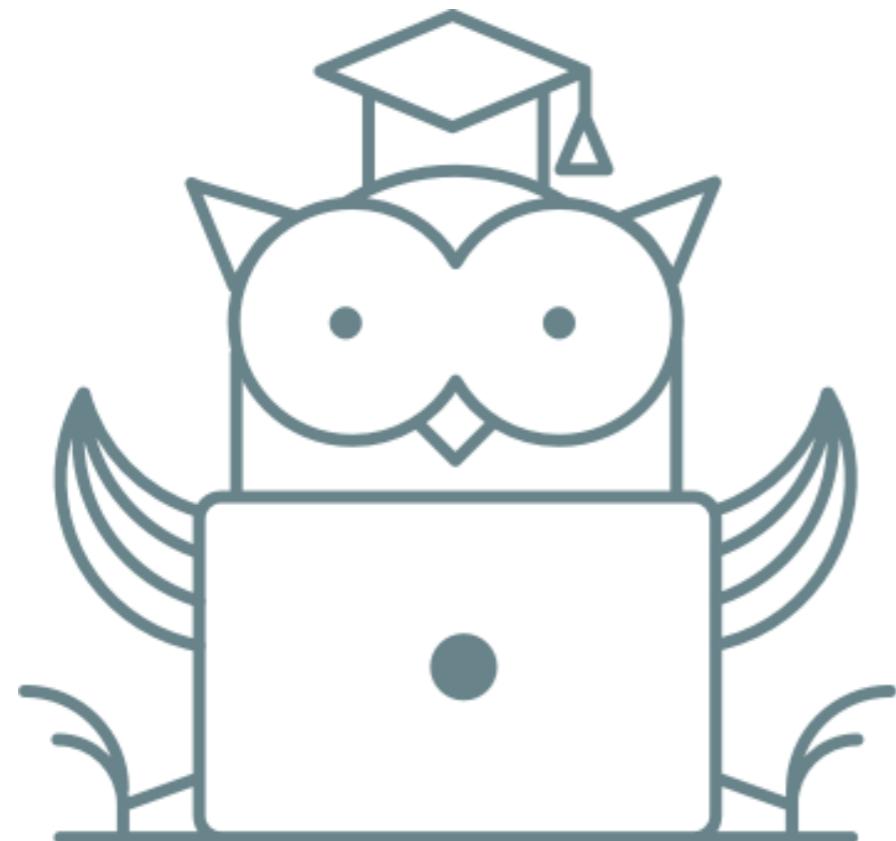


Совсем чуть-чуть орг.вопросов:

- Небольшая задержка – сегодня/завтра планировал разгрести.
- Если Вы хотите что-то (побольше ДЗ, поменьше ДЗ) или т.д. – пишите в личку.
- А так же у нас есть чат ☺
- Со всех отзыв!

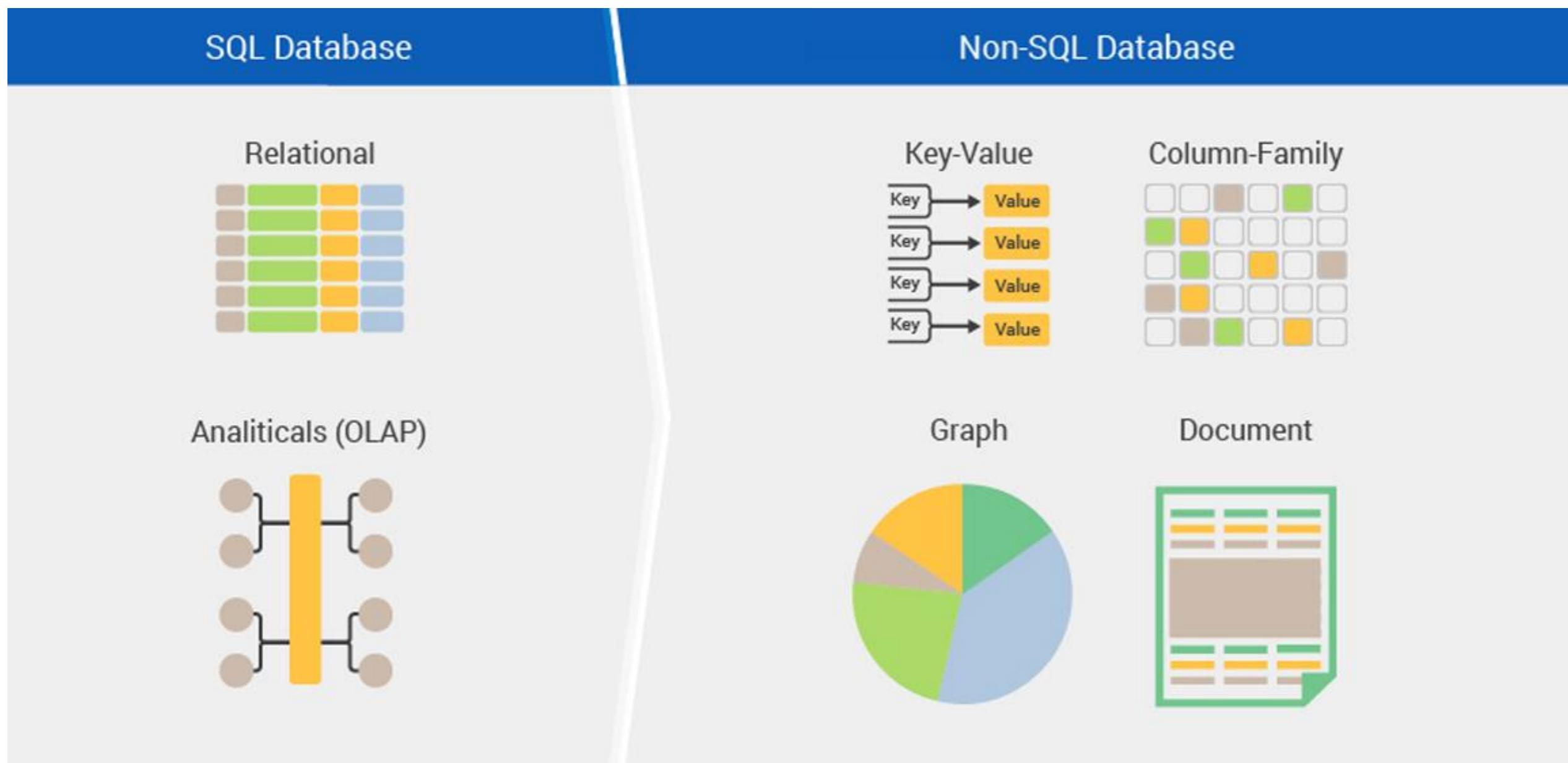


Поехали?



Вспомним

Принципы хранения данных



Key-Value

- Key-value – одну такую Вы точно знаете, она называется HashMap 😊
- Это очень популярная структура, как и такие Бд
- Собственно так и выглядит Бд – по ключу Вы кладёте значение и получаете. Никаких связей
- Подобная простота как раз и оказывается предпочтительной для некоторых операций

Документ-ориентированные

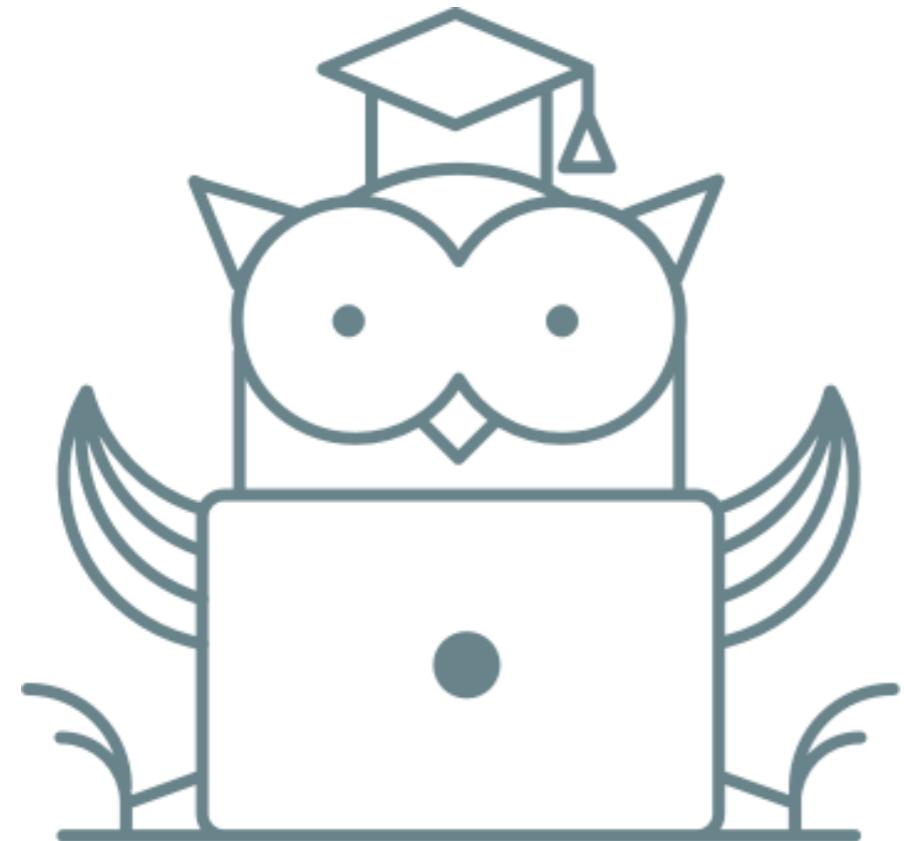
- У Вас есть структурированный документ (JSON)
- Документы могут содержать ссылки на другие документы
- И они могут редактироваться и запросы могут делаться по полям

Упражнение

- Поставить себе MongoDB (можете и Compass)

Онлайн-консоли

- https://www.tutorialspoint.com/redis_terminal_online.php
- https://www.tutorialspoint.com/memcached_terminal_online.php
- https://www.tutorialspoint.com/mongodb_terminal_online.php



Spring Data Key Value

Spring Data Key-Value

- Абстракция для Key-value баз данных
- Содержит дефолтную реализацию на основе WeakMap)
- Если Вы ещё не определились в выбором key-Value БД, то можете использовать Spring Data Key-Value
- Но модули для работы с другими БД предоставляют более специфичные методы и Tempalte
- Прекрасна для тестов

Maven

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-keyvalue</artifactId>
    <version>${version}</version>
</dependency>
```

@EnableMapRepositories

```
@Configuration  
@EnableMapRepositories("com.acme.repositories")  
class AppConfig { ... }
```

```
@EnableMapRepositories(mapType = WeakHashMap.class)
```

Entity

```
@KeySpace("user")
class User {
    @Id String uuid;
    String firstname;
}
```

KeySpace это

- collections в MongoDB
- collections в Elasticsearch
- cores в Solr
- таблицы в JPA.

Repository

```
public interface UserRepository extends CrudRepository<User, String> {  
    List<String> findByLastname(String lastname);  
}
```

- Paging and sorting не поддерживается в общем виде
- P.S. У меня почему-то заработало

Упражнение

- <https://github.com/ydvorzhetskiy/spring-framework-12>
- Пробуем?

KeyValueTemplate

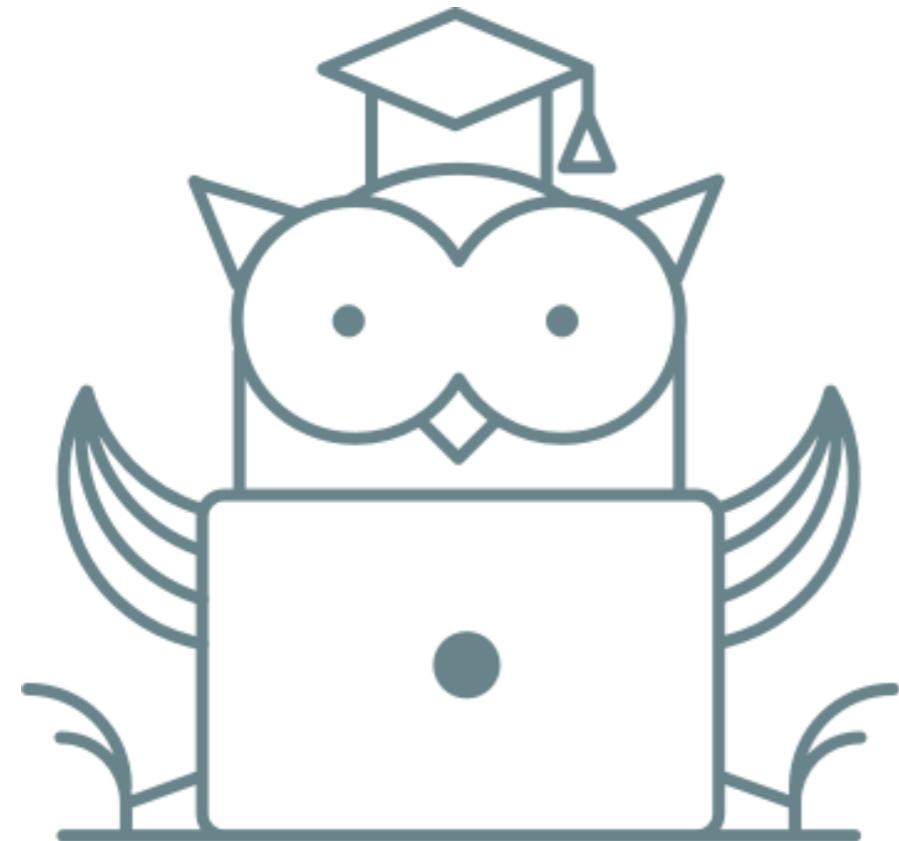
```
@Bean  
public KeyValueOperations keyValueTemplate() {  
    return new KeyValueTemplate(keyValueAdapter());  
}
```

```
@Bean  
public KeyValueAdapter keyValueAdapter() {  
    return new MapKeyValueAdapter(WeakHashMap.class);  
}
```

```
Employee employee = new Employee(1, "Mile", "IT", "5000");  
keyValueTemplate.insert(employee);
```

Упражнение

- <https://github.com/ydvorzhetskiy/spring-framework-12>
- Пробуем?
- * Попробуйте то же сделать с Email и через
KeyValueOperations

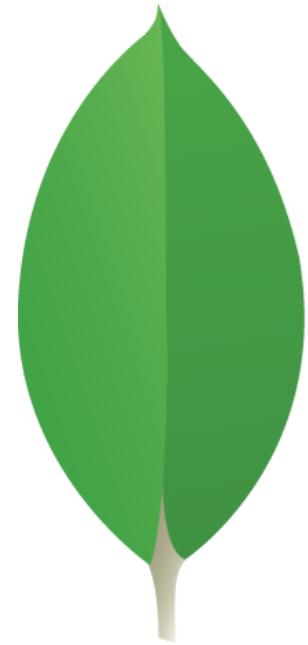


Вопросы?

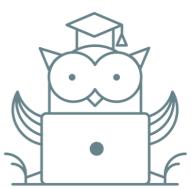


MongoDb

O T U S



mongoDB®



M - MongoDB

1. Масштабируемая.
2. Высокопроизводительная
3. Интерфейс на JS (!)
4. Простая в использовании
5. Эффективно работает с большими данными
6. Хранит документы
7. Никаких JOIN-ов



Объекты MongoDB

```
Public class Employee
{
    private int empld;
    private String name;
    private String department;
    ...
}
```

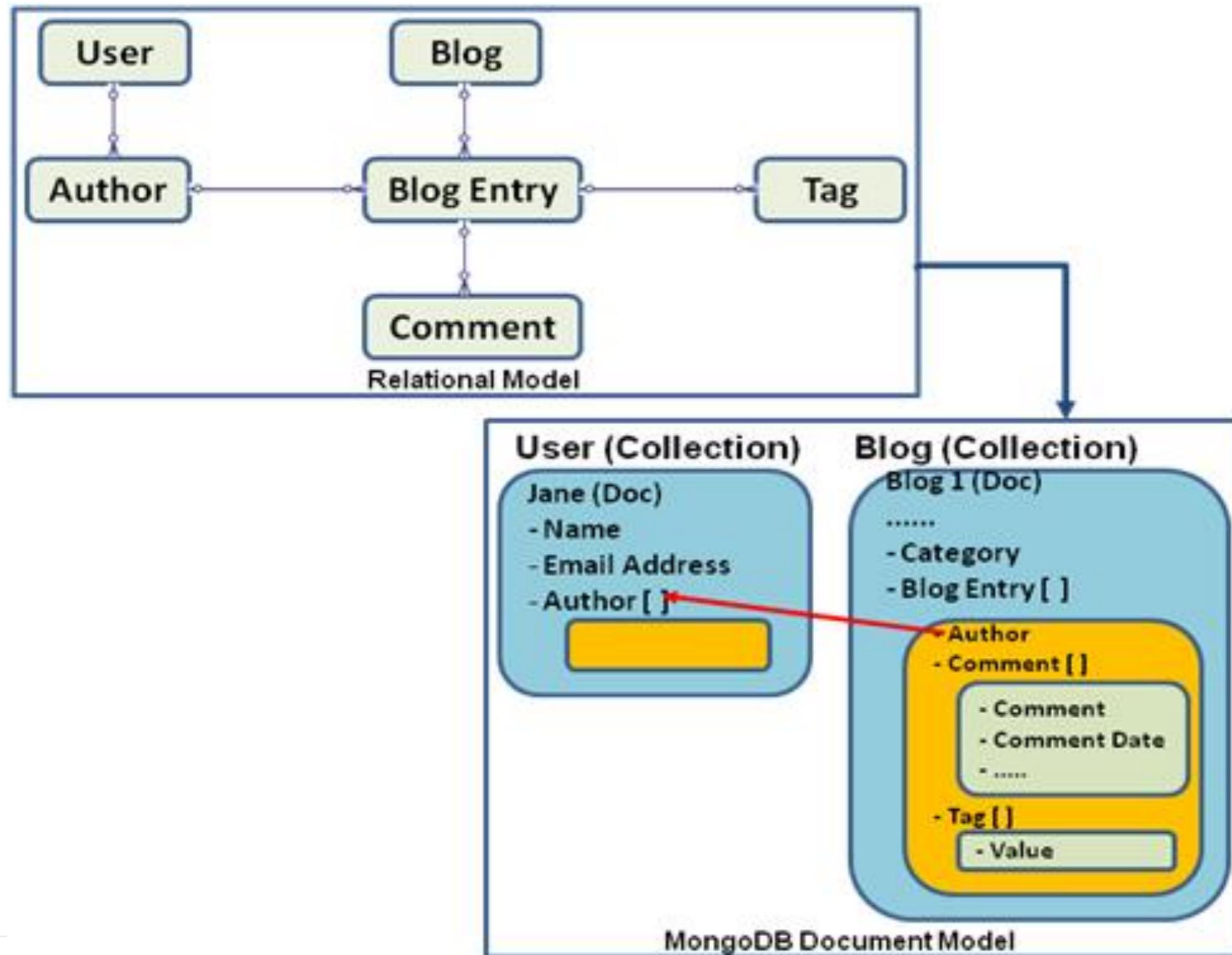
EMPLOYEE	
PK	EMPID
	NAME
	DEPARTMENT
	CITY

No Impedance Mismatch

```
// your application code
class Foo { int x; string [] tags; }

// mongo document for Foo
{ x: 1, tags: ['abc','xyz'] }
```





Database (MongoDB) – Database (SQL)

1. Database – база данных
2. Состоит из коллекций
3. Сама создаётся, если к ней обращаются



Collection (MongoDB) – Table (SQL)

1. Collection – коллекция
2. Состоит из документов
3. Индексируется причём хитрыми индексами



Document (MongoDB) – Row (SQL)

1. Document - документ
2. По сути дела – просто JSON
3. Хранится в формате JSONB
4. Есть `_id` – вроде primary key
5. Поддерживает отношения с другим документами – по ссылки или embedded



// Пробуем?

```
db.products.insert(  
  { _id: 10, item: "box", qty: 20 }  
)
```

// Ещё?



// Пробуем?

```
db.products.find({ _id: 10 })
```



// Пробуем?

```
db.products.find({ _id: 10 })
```

```
db.products.find({ item: "box" })
```

// дадада

```
db.products.find({ _id: { $gt: 5 } })
```

// проекция

```
db.products.find(  
  { _id: { $gt: 5 } }, { item: 1, qty: 1 }  
)
```



// Пробуем?

```
db.products.find({ _id: 10 })
```

```
db.products.find({ _id: { $gt: 5 } })
```

```
db.products.find({ item: "box" })
```

```
db.products.find(  
  { _id: { $gt: 5 } }, { item: 1, qty: 1 }  
)
```



// Пробуем?

```
db.products.find().sort({ name: 1 }).limit( 5 )
```

```
db.products.find().limit(5).sort( { name: 1 } )
```



// Пробуем?

```
db.products.update(  
  { id: { $gte: 100 } },  
  { $set: { "weight": 100 } },  
  {  
    multi: true,  
    $addToSet: { items: "box" }  
  }  
)
```

\$set, \$unset, \$push, \$pull, \$pop, \$addToSet,
\$incr, \$decr ...

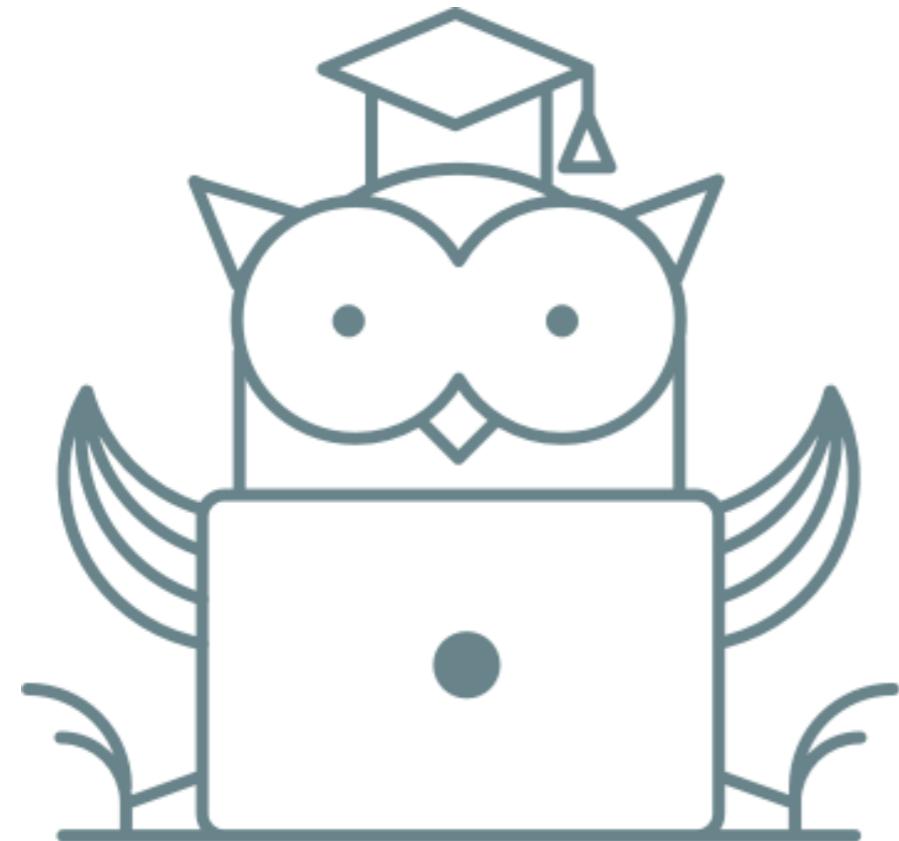


```
db.collection.findAndModify({  
    query: <document>,  
    sort: <document>,  
    remove: <boolean>,  
    update: <document>,  
    new: <boolean>,  
    fields: <document>,  
    upsert: <boolean>,  
    bypassDocumentValidation: <boolean>,  
    writeConcern: <document>,  
    collation: <document>,  
    arrayFilters: [ <filterdocument1>, ... ]  
});
```





Вопросы?



Spring Data MongoDB

Spring Data MongoDB

- Руками и через spring-boot-starter-mongodb

```
@Bean
public Mongo mongo() throws Exception {
    return new Mongo("localhost");
}

@Override
public String getDatabaseName() {
    return "database";
}

@Override
public String getMappingBasePackage() {
    return "com.bigbank.domain";
}
```

```
#Local MongoDB config
spring.data.mongodb.authentication-database=admin
spring.data.mongodb.username=root
spring.data.mongodb.password=root
spring.data.mongodb.database=user_db
spring.data.mongodb.port=27017
spring.data.mongodb.host=localhost
```

Entity

```
@Document  
public class User {  
    //  
}  
  
@Id  
private String id;
```

```
@QueryEntity  
@Document  
@CompoundIndexes({  
    @CompoundIndex(name = "email_age", def = "{'email.id' : 1, 'age': 1}")  
})  
public class User {  
    //  
}
```

Entity

- `@Document`
- `@Field`
- `@Id`
- `@DBRef`
- `@Transient`
- `@PersistenceConstructor`
- `@Indexed`

Spring Data MongoDB

- То же самое – или репозитории или темплейт

```
@EnableMongoRepositories(basePackages = "org.baeldung.repository")
```

```
public interface UserRepository extends MongoRepository<User, String> {  
    //  
}
```

Spring Data MongoDB

- То же самое – или репозитории или темплейт

```
@Configuration
public class SimpleMongoConfig {

    @Bean
    public Mongo mongo() throws Exception {
        return new MongoClient("localhost");
    }

    @Bean
    public MongoTemplate mongoTemplate() throws Exception {
        return new MongoTemplate(mongo(), "test");
    }
}
```

MongoTemplate

```
Query query = new Query();
query.addCriteria(Criteria.where("name").is("Alex"));
Update update = new Update();
update.set("name", "James");
mongoTemplate.updateFirst(query, update, User.class);
```

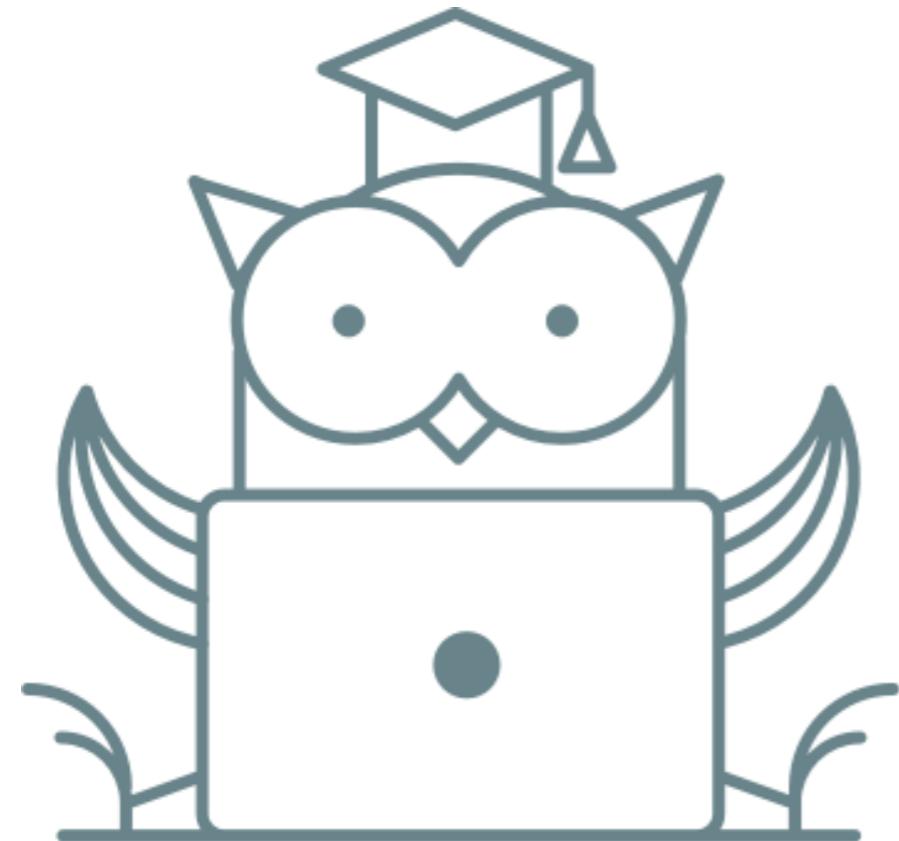
Embedded-mongodb

```
<dependency>
    <groupId>de.flapdoodle.embed</groupId>
    <artifactId>de.flapdoodle.embed.mongo</artifactId>
    <scope>test</scope>
</dependency>
```

```
IMongodConfig mongodConfig = new MongodConfigBuilder().version(Version.MA
    .net(new Net(ip, port, Network.localhostIsIPv6())))
    .build();

MongodStarter starter = MongodStarter.getDefaultInstance();
mongodExecutable = starter.prepare(mongodConfig);
mongodExecutable.start();
mongoTemplate = new MongoTemplate(new MongoClient(ip, port), "test");
```

<https://github.com/flapdoodle-oss/de.flapdoodle.embed.mongo>



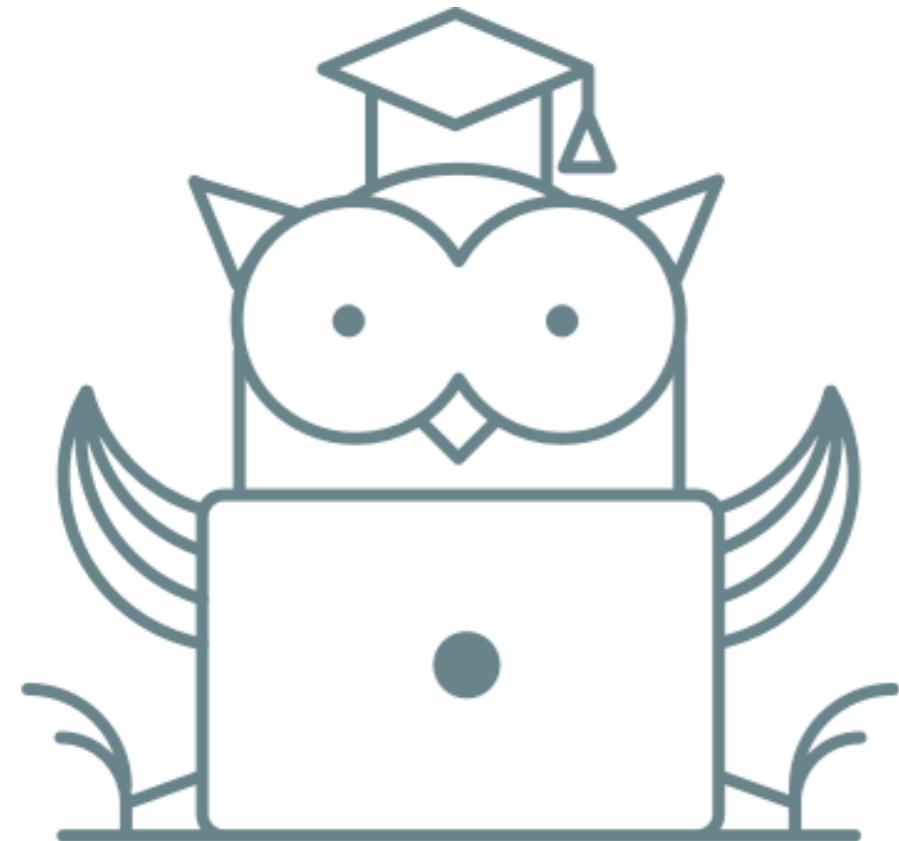
Вопросы?

Домашнее задание

Использовать MongoDB и spring-data для хранения информации о книгах

Тесты можно реализовать с помощью spring-boot-starter-embedded-mongodb

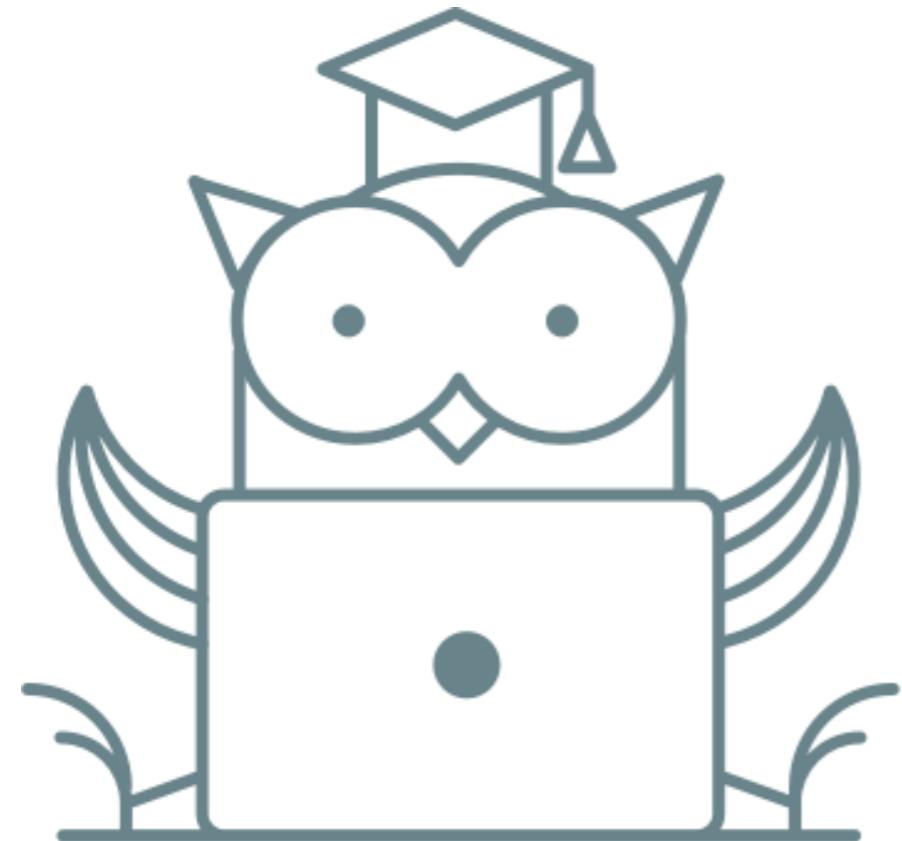




Вопросы?

Пожалуйста, пройдите опрос

<https://otus.ru/polls/1626/>



Спасибо
за внимание!

Spring Data Вам!