

# Базы данных и Java

Юрий Литвинов  
yurii.litvinov@gmail.com

13.02.2019г

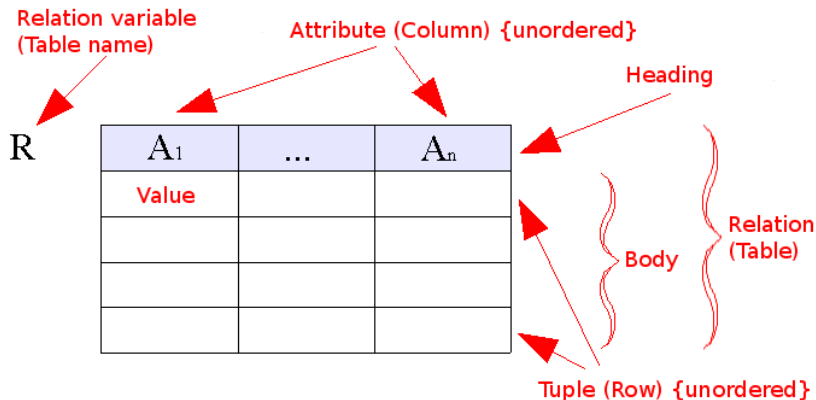
# СУБД

- ▶ Реляционные
  - ▶ Отношения
  - ▶ Операции
- ▶ Объектно-ориентированные
  - ▶ Сериализованные объекты
- ▶ Иерархические
- ▶ ...

# Реляционные vs ОО-СУБД

- ▶ Реляционные
  - ▶ Сложность интеграции с ОО-кодом
    - ▶ ORM (Microsoft Entity Framework, Hibernate, MyBatis, ...)
  - ▶ Эффективные и выразительные запросы
- ▶ Объектно-ориентированные
  - ▶ Проще, легче, не требуют ORM
  - ▶ “Бедный” язык запросов
  - ▶ Часто не умеют того, что для реляционных СУБД естественно (например, транзакций)

# Реляционная модель данных



© Wikipedia

# Пример таблицы

CustomerID	TaxID	Name	Address
1234567890	555-5512222	Munmun	323 Broadway
2223344556	555-5523232	Wile E.	1200 Main Street
3334445563	555-5533323	Ekta	871 1st Street
423242432	555-5325523	E.F. Codd	123 It Way

# Ключи

- ▶ Первичные (primary)
  - ▶ Естественные
  - ▶ Составные
  - ▶ Суррогатные
- ▶ Внешние (foreign)

CITY

ID	Name
1	Москва
2	Санкт-Петербург
3	Владивосток

STREET

ID	Name	ID_CITY
181	Малая Бронная	1
182	Тверской Бульвар	1
183	Невский проспект	2
184	Пушкинская	2
185	Светланская	3
186	Пушкинская	3

# Ограничения

- ▶ PRIMARY KEY
- ▶ FOREIGN KEY
- ▶ NOT NULL
- ▶ UNIQUE
- ▶ ...

# SQL SELECT

Таблица «Т»	Запрос	Результат												
<table><tr><th>C1</th><th>C2</th></tr><tr><td>1</td><td>a</td></tr><tr><td>2</td><td>b</td></tr></table>	C1	C2	1	a	2	b	<pre>SELECT * FROM T;</pre>	<table><tr><th>C1</th><th>C2</th></tr><tr><td>1</td><td>a</td></tr><tr><td>2</td><td>b</td></tr></table>	C1	C2	1	a	2	b
C1	C2													
1	a													
2	b													
C1	C2													
1	a													
2	b													
<table><tr><th>C1</th><th>C2</th></tr><tr><td>1</td><td>a</td></tr><tr><td>2</td><td>b</td></tr></table>	C1	C2	1	a	2	b	<pre>SELECT C1 FROM T;</pre>	<table><tr><th>C1</th></tr><tr><td>1</td></tr><tr><td>2</td></tr></table>	C1	1	2			
C1	C2													
1	a													
2	b													
C1														
1														
2														
<table><tr><th>C1</th><th>C2</th></tr><tr><td>1</td><td>a</td></tr><tr><td>2</td><td>b</td></tr></table>	C1	C2	1	a	2	b	<pre>SELECT * FROM T WHERE C1 = 1;</pre>	<table><tr><th>C1</th><th>C2</th></tr><tr><td>1</td><td>a</td></tr></table>	C1	C2	1	a		
C1	C2													
1	a													
2	b													
C1	C2													
1	a													
<table><tr><th>C1</th><th>C2</th></tr><tr><td>1</td><td>a</td></tr><tr><td>2</td><td>b</td></tr></table>	C1	C2	1	a	2	b	<pre>SELECT * FROM T ORDER BY C1 DESC;</pre>	<table><tr><th>C1</th><th>C2</th></tr><tr><td>2</td><td>b</td></tr><tr><td>1</td><td>a</td></tr></table>	C1	C2	2	b	1	a
C1	C2													
1	a													
2	b													
C1	C2													
2	b													
1	a													

© Wikipedia



# SELECT, вложенные запросы

```
SELECT isbn,  
        title,  
        price  
FROM Book  
WHERE price < (SELECT AVG(price) FROM Book)  
ORDER BY title;
```

# INNER JOIN

**City (Города)**

<u>Id</u>	Name
1	Москва
2	Санкт-Петербург
3	Казань

**Person (Люди)**

<u>Name</u>	CityId
Андрей	1
Леонид	2
Сергей	1
Григорий	4

```
SELECT *  
FROM  
  Person  
  INNER JOIN  
  City  
  ON Person.CityId = City.Id
```

Person.Name	Person.CityId	City.Id	City.Name
Андрей	1	1	Москва
Леонид	2	2	Санкт-Петербург
Сергей	1	1	Москва

© Wikipedia

# OUTER JOIN

**City (Города)**

<u>Id</u>	Name
1	Москва
2	Санкт-Петербург
3	Казань

**Person (Люди)**

<u>Name</u>	CityId
Андрей	1
Леонид	2
Сергей	1
Григорий	4

```
SELECT *  
FROM  
    Person  
LEFT OUTER JOIN  
    City  
ON Person.CityId = City.Id
```

Person.Name	Person.CityId	City.Id	City.Name
Андрей	1	1	Москва
Леонид	2	2	Санкт-Петербург
Сергей	1	1	Москва
Григорий	4	NULL	NULL

© Wikipedia

# CROSS JOIN

**City (Города)**

<u>Id</u>	Name
1	Москва
2	Санкт-Петербург
3	Казань

**Person (Люди)**

<u>Name</u>	CityId
Андрей	1
Леонид	2
Сергей	1
Григорий	4



```
SELECT *  
FROM  
  Person,  
  City
```



Person.Name	Person.CityId	City.Id	City.Name
Андрей	1	1	Москва
Андрей	1	2	Санкт-Петербург
Андрей	1	3	Казань
Леонид	2	1	Москва
Леонид	2	2	Санкт-Петербург
Леонид	2	3	Казань
Сергей	1	1	Москва
Сергей	1	2	Санкт-Петербург
Сергей	1	3	Казань
Григорий	4	1	Москва
Григорий	4	2	Санкт-Петербург
Григорий	4	3	Казань

© Wikipedia

# Таблицы-развязки

Author

ID	Name
1	Терехов
2	Брыксин
3	Литвинов

AuthorArticle

AuthorId	ArticleId
1	1
1	2
2	1
2	3
3	1
3	3

Article

ID	Title
1	Архитектура среды визуального моделирования QReal
2	Технология программирования
3	Среда визуального программирования роботов QReal:Robots

# Примеры запросов

```
SELECT Article.Title FROM Author, AuthorArticle, Article  
WHERE Author.Id = AuthorArticle.AuthorId  
AND AuthorArticle.ArticleId = Article.Id  
AND Author.Name = 'Терехов'
```

```
SELECT Author.Name FROM Author, AuthorArticle, Article  
WHERE Author.Id = AuthorArticle.AuthorId  
AND AuthorArticle.ArticleId = Article.Id  
AND Article.Title = 'Технология программирования'
```

# INSERT, UPDATE, DELETE

INSERT:

```
INSERT INTO phone_books VALUES ('Peter Doe', '555-2323');
```

UPDATE:

```
UPDATE persons SET  
    street = 'Nissestien 67',  
    city = 'Sandnes',  
WHERE lastname = 'Tjessem' AND firstname = 'Jakob';
```

DELETE:

```
DELETE ab, b  
FROM Authors AS a, AuthorArticle AS ab, Articles AS b  
WHERE a.AuthID = ab.AuthID AND ab.ArticleID = b.ArticleID  
    AND AuthorLastName = 'Henry';
```

# Работа с метаинформацией

CREATE TABLE:

```
CREATE TABLE Students (  
  Code INTEGER NOT NULL,  
  Name NCHAR(30) NOT NULL,  
  Address NVARCHAR(50),  
  Mark DECIMAL);
```

DROP TABLE:

```
DROP TABLE Students;
```



© XKCD



# Работа с метаинформацией

ALTER TABLE:

**ALTER TABLE** Students **ADD** email **VARCHAR**(**MAX**);

**ALTER TABLE** Students **DROP COLUMN** email;

**ALTER TABLE** Students **ADD PRIMARY KEY** (Code);

# Низкоуровневый API: JDBC

- ▶ API для доступа к реляционным базам данных из программ на Java
- ▶ Позволяет выполнять SQL-запросы
- ▶ Пример:

```
try (Connection conn = DriverManager.getConnection(  
    "jdbc:mariadb://localhost:3306/mydb",  
    "root",  
    "1")) {  
    try (Statement stmt = conn.createStatement()) {  
        stmt.executeUpdate(  
            "INSERT INTO cities(Name) VALUES ('St. Petersburg')");  
    }  
}
```

- ▶ Нужна библиотека для общения с СУБД — драйвер

## JDBC: Пример запроса к базе

```
try (Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT * FROM Cities")
) {
    while (rs.next()) {
        int numColumns = rs.getMetaData().getColumnCount();
        for (int i = 1 ; i <= numColumns ; i++) {
            System.out.println("COLUMN " + i + " = " + rs.getObject(i));
        }
    }
}
```

# SQLite

- ▶ База данных, работающая в пространстве процесса
  - ▶ Не требует отдельного сервера, поставляется как .jar-ник
  - ▶ Не требует администрирования и настройки
- ▶ Альтернатива хранению данных в файлах
- ▶ Умеет полноценные SQL-запросы
  - ▶ Зато плоха в обработке больших объёмов данных или когда нагрузка слишком большая
  - ▶ Не умеет “из коробки” работать по сети
- ▶ Хранит всю базу в одном файле
- ▶ Размер библиотеки и используемая память могут быть очень маленькими
  - ▶ Хороша для встроенных устройств

# SQLite, пример

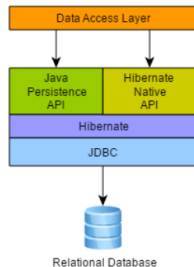
```

try (Connection conn = DriverManager.getConnection(
    "jdbc:sqlite:sample.db")) {
    try (Statement stmt = conn.createStatement()) {
        stmt.executeUpdate("drop table if exists cities");
        stmt.executeUpdate("create table cities (id integer, name varchar(50))");
        stmt.executeUpdate("insert into cities values(1, 'St. Petersburg')");
        stmt.executeUpdate("insert into cities values(2, 'Moscow')");
        try (ResultSet rs = stmt.executeQuery("select * from cities")) {
            while (rs.next()) {
                // read the result set
                System.out.println("name = " + rs.getString("name"));
                System.out.println("id = " + rs.getInt("id"));
            }
        }
    }
}

```

# ORM-системы, Hibernate

- ▶ Умеет генерировать прокси-объекты, которые можно использовать в приложении для работы с таблицами БД
- ▶ Основные классы:
  - ▶ Configuration — хранит конфигурацию, в частности, к чему и как подключаться, умеет создавать SessionFactory
  - ▶ SessionFactory — представляет отображение таблиц БД на классы из Data Access Layer, создаёт объекты Session, одна на приложение
  - ▶ Session — “Единица работы”, штука, которая реально связывается с БД и обновляет её таблицы результатами изменений в DAL
  - ▶ Transaction — транзакция (атомарная операция) в БД. Сессия создаёт транзакции, но может иметь только одну открытую транзакцию в каждый момент



# NoSQL, MongoDB

- ▶ Документо-ориентированная СУБД
  - ▶ Единица хранения — документ
  - ▶ Набор пар “имя”-“значение”, значение может быть сложного типа
  - ▶ Коллекция — список документов
- ▶ Сервер — отдельный процесс
  - ▶ Впрочем, не требует особого администрирования
- ▶ Morphia — типобезопасный API для работы с базой поверх mongodb-driver
- ▶ Не использует JDBC
- ▶ Использует разумные умолчания — если БД или коллекции при первом обращении нет, она будет создана

# Пример аннотированного объекта

@Entity

```
public class City {  
    @Id private int id;  
    private String name;  
  
    public City() { }  
    public City(String name) {  
        this.name = name;  
    }  
  
    public String getName() {  
        return name;  
    }  
}
```



## Пример добавления в базу

```
final Morphia morphia = new Morphia();  
morphia.mapPackage("com.example");  
final Datastore datastore = morphia.createDatastore(new MongoClient(), "mydb");  
  
City stPetersburg = new City("St. Petersburg");  
datastore.save(stPetersburg);
```

## Пример чтения из базы

```
final Morphia morphia = new Morphia();
morphia.mapPackage("com.example");
final Datastore datastore = morphia.createDatastore(new MongoClient(), "mydb");

for (City city : datastore.find(City.class)) {
    System.out.println(city.getName());
}
```