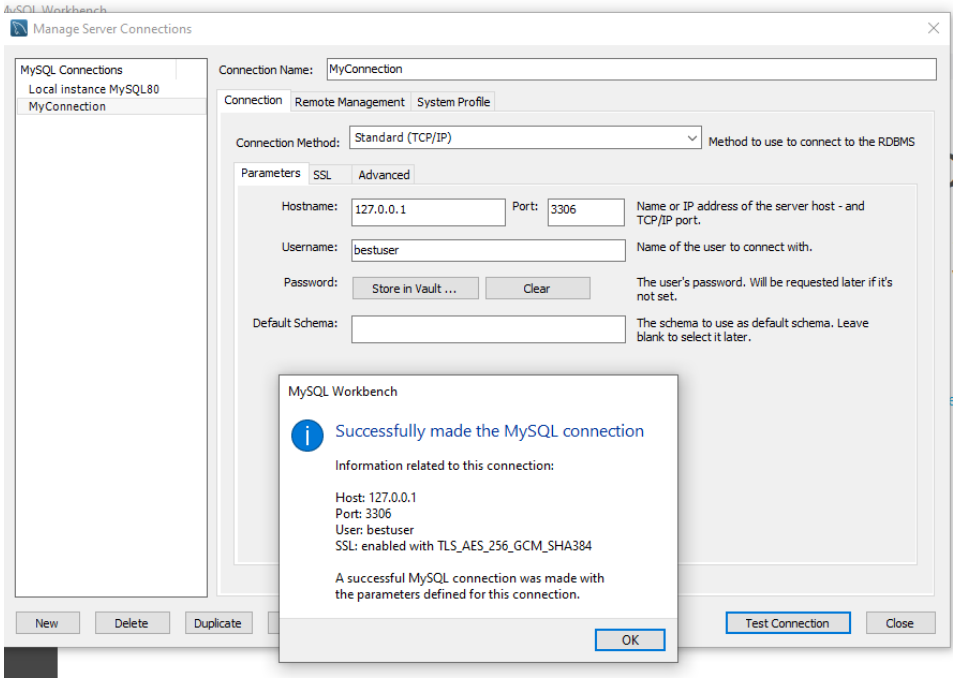
19. Spring Framework (Hibernate)

- Конфигурация MySQL Server и создание структуры БД

- Подключение БД к приложению

- Создание связей между таблицами и модификация данных средствами Hibernate

Создание соединения и проверка подключения в среде MySQL Workbench:



Создание пользователя и задание привилегий (SQL):

```
CREATE USER 'bestuser'@'localhost' IDENTIFIED BY 'bestuser';

GRANT ALL PRIVILEGES ON * . * TO 'bestuser'@'localhost';
```

Создание базы данных (SQL):

```
CREATE DATABASE  my_db;
USE my_db;

CREATE TABLE employees (
  id int NOT NULL AUTO_INCREMENT,
  name varchar(15),
  surname varchar(25),
  department varchar(20),
  salary int,
  PRIMARY KEY (id)
);
```

Добавление коннектора mysql-connector-java.jar и конфигурация Hibernate (файл hibernate.cfg.xml):

```
<hibernate-configuration>
    <session-factory>
        <property
name="connection.url">jdbc:mysql://localhost:3306/my_db?useSSL=false&amp;serv
erTimezone=UTC</property>
        <property
name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property name="connection.username">bestuser</property>
        <property name="connection.password">bestuser</property>

        <property name="current_session_context_class">thread</property>
        <property
name="dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="show_sql">true</property>

    </session-factory>
</hibernate-configuration>
```

Создание таблицы Employees:

```
CREATE TABLE my_db.employees (
  id int NOT NULL AUTO_INCREMENT,
  name varchar(15),
  surname varchar(25),
  department varchar(20), salary int, details_id int
,  PRIMARY KEY (id)
, FOREIGN KEY (details_id) REFERENCES my_db.details(id));
```

Связывание таблицы Employees с классом Employee в приложении: (@Entity, @Table, @Id, @GeneratedValue, @Column)

```
@Entity
@Table(name = "employees") // можно не писать если одинаковые названия
public class Employee {

    @Id //столбец с данным полем является primary key
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id") // можно не писать если одинаковые названия
    private int id;

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

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

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

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

    public Employee(){
    }
```

Создание нового Employee и добавление в БД с последующим получением работника и выводом на экран: (SessionFactory, Session, beginTransaction(), getTransaction(), commit(), save(), get())

```
    public static void main(String[] args) {
        SessionFactory factory = new Configuration()
                .configure("hibernate.cfg.xml")
                .addAnnotatedClass(Employee.class)
                .buildSessionFactory();
        try {
            Session session = factory.getCurrentSession();
            Employee emp = new Employee("Oleg", "Sidorov", "HR", 700);
            session.beginTransaction();
            session.save(emp);
            //session.getTransaction().commit();
            int myId = emp.getId();
            //session = factory.getCurrentSession();
            //session.beginTransaction();
            Employee employee = session.get(Employee.class, myId);
            session.getTransaction().commit();
            System.out.println(employee);
            System.out.println("Done!");
        }
        finally {
            factory.close();
        }
    }
}
```

Выборка данных методом createQuery():

```
    public static void main(String[] args) {
        SessionFactory factory = new Configuration()
                .configure("hibernate.cfg.xml")
                .addAnnotatedClass(Employee.class)
                .buildSessionFactory();
        try {

            Session session = factory.getCurrentSession();
            session.beginTransaction();
            List<Employee> emps = session.createQuery("from Employee WHERE "
+
                    "name = 'Forza' AND salary > 500")
                    .getResultList();
            for (Employee e : emps)
                System.out.println(e);
            session.getTransaction().commit();
            System.out.println("Done!");
        }
        finally {
            factory.close();
        }
    }
}
```

Обновление данных:

```
public static void main(String[] args) {
    SessionFactory factory = new Configuration()
```

```
            .configure("hibernate.cfg.xml")
            .addAnnotatedClass(Employee.class)
            .buildSessionFactory();
    try {
        Session session = factory.getCurrentSession();
        session.beginTransaction();
        //Employee emp = session.get(Employee.class, 1);
        //emp.setSalary(1500);
        session.createQuery("update Employee set salary=1000 where
name='Forza'").executeUpdate();
        session.getTransaction().commit();
        System.out.println("Done!");
    }
    finally {
        factory.close();
    }
}
```

Удаление работника (метод createQuery().executeUpdate(), возможно также с помощью session.delete()):

```
public static void main(String[] args) {
    SessionFactory factory = new Configuration()
            .configure("hibernate.cfg.xml")
            .addAnnotatedClass(Employee.class)
            .buildSessionFactory();
    try {
        Session session = factory.getCurrentSession();
        session.beginTransaction();
        //Employee emp = session.get(Employee.class, 8);
        //session.delete(emp);
        session.createQuery("delete Employee where name
='Oleg'").executeUpdate();
        session.getTransaction().commit();
        System.out.println("Done!");
    }
    finally {
        factory.close();
    }
}
```

Консоль-лог выполнения запроса:

```
INFO: HCANN000001: Hibernate Commons Annotations {5.1.0.Final}
апр. 20, 2021 6:09:13 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl configure
WARN: HHH10001002: Using Hibernate built-in connection pool (not for production use!)
апр. 20, 2021 6:09:13 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001005: using driver [com.mysql.cj.jdbc.Driver] at URL [jdbc:mysql://localhost:3306/my_db?useSSL=false&serverTimezone=UTC]
апр. 20, 2021 6:09:13 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001001: Connection properties: {password=****, user=bestuser}
апр. 20, 2021 6:09:13 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH10001003: Autocommit mode: false
апр. 20, 2021 6:09:13 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
апр. 20, 2021 6:09:13 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect
Hibernate: insert into employees (department, name, salary, surname) values (?, ?, ?, ?)
Employee{id=12, name='Oleg', surname='Sidorov', department='HR', salary=700}
Done!
апр. 20, 2021 6:09:15 PM org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PoolState stop
INFO: HHH10001008: Cleaning up connection pool [jdbc:mysql://localhost:3306/my_db?useSSL=false&serverTimezone=UTC]
```

Создание связи One-to-One:

Таблицы Employees и Details: (связывание с помощью ForeignKey)

```
CREATE TABLE my_db.details (
  id int NOT NULL AUTO_INCREMENT,
  city varchar(15),
  phone_number varchar(25),
  email varchar(30), PRIMARY KEY (id)
);
CREATE TABLE my_db.employees (
  id int NOT NULL AUTO_INCREMENT,
  name varchar(15),
  surname varchar(25),
  department varchar(20), salary int, details_id int
,  PRIMARY KEY (id)
, FOREIGN KEY (details_id) REFERENCES my_db.details(id));
```

Java класс таблицы Employees (@One-to-One, @JoinColumn – связать с полем из другой таблицы):

```
@Entity
@Table(name = "employees") // можно не писать если одинаковые названия
public class Employee {

    @Id //столбец с данным полем является primary key
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id") // можно не писать если одинаковые названия
    private int id;

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

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

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

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

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn (name = "details_id")
    private Detail empDetail;
```

Java класс таблицы Details: (для bi-directional связи необходимо также указать @One-to-One, и параметр mappedBy – поле, где искать связь, также параметр cascade)

```
@Entity
@Table(name = "details")
public class Detail {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
```

```
    private int id;

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

    @Column(name = "phone_number")
    private String phoneNumber;

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

    @OneToOne(mappedBy = "empDetail" , cascade = {CascadeType.PERSIST,
CascadeType.REFRESH})
    private Employee employee;
```

Создание связи One-To-Many:

Создание таблиц департаментов и сотрудников:

```
CREATE TABLE my_db.departments (
  id int NOT NULL AUTO_INCREMENT,
  name varchar(15),
  max_salary int,
  min_salary int,
  PRIMARY KEY (id)
);

CREATE TABLE my_db.employees (
  id int NOT NULL AUTO_INCREMENT,
  name varchar(15),
  surname varchar(25),
  salary int,
  department_id int,
  PRIMARY KEY (id),
  FOREIGN KEY (department_id) REFERENCES my_db.departments(id));
```

Java класс для сотрудников:

```
@Entity
@Table(name = "employees") // можно не писать если одинаковые названия
public class Employee {

    @Id //столбец с данным полем является primary key
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id") // можно не писать если одинаковые названия
    private int id;

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

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

    @Column(name = "salary")
    private int salary;
```

Java класс для департаментов: (аннотация @One-To-Many)

```
@Entity
@Table(name = "departments")
public class Department {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

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

    @Column(name = "max_salary")
    private int maxSalary;

    @Column(name = "min_salary")
    private int minSalary;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name= "department_id")
    private List<Employee> emps;
```

Для bi-deirectional связи необходимо также указать, поле где искать связь, с параметром mappedBy, и типом загрузки данных EAGER или LAZY:

```
@Entity
@Table(name = "departments")
public class Department {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

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

    @Column(name = "max_salary")
    private int maxSalary;

    @Column(name = "min_salary")
    private int minSalary;

    @OneToMany(cascade = CascadeType.ALL,
            mappedBy = "department",
            fetch = FetchType.LAZY)
    private List<Employee> emps;
```

Создание связи Many-To-Many

Создание таблицы для детей, секций, и общей таблицы с 2-мя параметрами ForeignKey:

```
CREATE TABLE my_db.children (
  id int NOT NULL AUTO_INCREMENT,
  name varchar(15),
  age int,
  PRIMARY KEY (id)
);
```

```
CREATE TABLE my_db.section (
  id int NOT NULL AUTO_INCREMENT,
  name varchar(15),
  PRIMARY KEY (id)
);

CREATE TABLE my_db.child_section (
  child_id int NOT NULL,
  section_id int NOT NULL,
  PRIMARY KEY (child_id, section_id),
  FOREIGN KEY (child_id) REFERENCES my_db.children(id),
  FOREIGN KEY (section_id) REFERENCES my_db.section(id));
```

Таблица Children (@Many-to-Many, @JoinTable – указать общую таблицу, и параметры joinColumns и inverseJoinColumns):

```
@Entity
@Table(name="children")
public class Child {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

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

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

    @ManyToMany(cascade = {CascadeType.MERGE, CascadeType.PERSIST,
CascadeType.REFRESH, CascadeType.DETACH})
    @JoinTable(name = "child_section",
    joinColumns = @JoinColumn(name = "child_id"),
    inverseJoinColumns = @JoinColumn(name = "section_id"))
    private List<Section> sections;
```

Таблица Sections (аналогично таблице Children, с противоположными полями в параметрах joinColumns и inverseJoinColumns):

```
@Entity
@Table(name = "section")
public class Section {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private int id;

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

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

```
    @ManyToMany(cascade = {CascadeType.MERGE, CascadeType.PERSIST,
CascadeType.REFRESH, CascadeType.DETACH})
    @JoinTable(name = "child_section",
            joinColumns = @JoinColumn(name = "section_id"),
            inverseJoinColumns = @JoinColumn(name = "child_id"))
    private List<Child> children;

    public void addChildToSection(Child child) {
        if (children == null) {
            children = new ArrayList<>();
        }
        children.add(child);
    }
```