ORM в Java

ORM (Object-Relational Mapping)

- И реляционная и объектная модели относятся к логическому уровню проектирования программной системы
- SQL — высокоуровневый декларативный специализированный язык четвёртого поколения
- Java или C#, по-прежнему относятся к третьему поколению языков императивных
- в ORM необходимо создавать собственный SQL-подобный язык для манипуляции объектами и уже его транслировать в SQL

- *ORM* — это отображение объектов какого-либо объектно-ориентированного языка в структуры реляционных баз данных
- Проблемы отображения
  - ООП (инкапсуляция, наследование, полиморфизм)
  - РСУБД (нет составных атрибутов, идентификация объектов, объект – кортеж или таблица)

JPA

- Спецификация JPA (Java Persistence API)
- Сложность разработки
- Поддержка в фреймворках

- Пакет javax.persistance

Hibernate

- использует POJO – классы с открытым конструктором без параметров
- используются аннотации

```
CREATE TABLE Student
(id             NUMERIC(10) NOT NULL,
name            varchar2(100) NOT NULL,
 age            NUMERIC(3) NOT NULL,

CONSTRAINT pk_Student PRIMARY KEY(id));
```

```
@Entity
@Table(name="Student")
   public class   implements   Serializable    Student {
@Id
@GeneratedValue  (strategy  =   GenerationType.IDENTITY)
       private int  id;
       private String name;
       private Long age;
       public Student(){
          name = null;
       }
       public Student(String name, Long age) {
           this.name = name;
           this.age= age;
       }
```

```
 @Column(name="id")
public Integer getIdStudent() {
    return idStudent;
}
@Column(name="name", nullable = false, length = 100)
public String getName() { return name; }
public void setName(String name) {
    this.name = name;
}
@Column(name="age")
public String getAge() {
    return age;
}
public void setAgr(Integer age) {
   this.age = age;
}
```

Основные аннотации

@Entity — указывает на то, что данный класс является сущностью.

@Table — задает имя таблицы, в которой будут храниться объекты класса

@Id — обозначает поле id

@GeneratedValue и *@GenericGenerator* — указывает на то, как будет генерироваться id

@Column — обозначает имя колонки, соответствующей данному полю.

Все классы-сущности должны обязательно иметь *геттеры*, *сеттеры* и *конструктор по умолчанию*.

Дополнительные аннотации

- @Temporal для полей типа

  java.util.Date и java.util.Calendar
- @Transient для полей, которые не хранятся в БД

Для ассоциации между классамии

- @ManyToOne
- @JoinColumn (name=“ … “,

  referencedColumnName = “ …”)

Фабрика сеансов Hibernate

Инициализация Hibernate заключается в создании фабрики сеансов.

Фабрика сеансов—это объект класса SessionFactory, существующий в программе (как правило) в единственном экземпляре

Сеансы представляют собой элементарные единицы взаимодействия приложения с базой данных и могут включать в себя транзакции

Фабрика сеансов Hibernate

-создать фабрику

```
SessionFactory
sessionFactory = new
AnnotationConfiguration().configure().buildSessionFactory();
```

-получить сеанс

```
Session session = sessionFactory.getCurrentSession();
```

Фабрика сеансов Hibernate

-начать транзакцию

```
session.beginTransaction();
```

-завершить транзакцию

```
session.getTransaction().commit();
session.getTransaction().rollback();
```

-закрыть фабрику

```
sessionFactory.close();
```

Конфигурационный файл

```
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
    <hibernate-configuration>
     <session-factory>
       <property name="connection.driver_class">
            org.firebirdsql.jdbc.FBDriver</property>
       <property name="connection.url">
jdbc:firebirdsql:class.mmcs.sfedu.ru/3050:/fbdata/stud.fdb</property>
       <property name="connection.username">IT38</property>
       <property name="connection.password">it8</property>
        <property name="connection.pool_size">10</property>
        <property name="dialect">org.hibernate.dialect.FirebirdDialect</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
        <property name="hibernate.connection.autocommit">false</property>
        <property name="current_session_context_class">thread</property>

        <mapping class="logic.Student" />

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

Состояния объектов

- Transient (временное)
- Persistent (постоянное)
- Detached (отсоединённое)

Сохранение объекта

- Transient  Persistent

```
Student s = new Student("Иванов А. А.", 20);
session.save(s);
session.getTransaction().commit();
```

Получение объекта

```
id=1;
Student s =
        (Student) session.get(Student.class, id);

Список студентов
List <Sudent> ls =
    session.createQuery(“from Student order by name”).list();
```

Удаление через proxy-объект

```
Student s =
        (Student) session.get(Student.class, id);
 session.delete(s);
session.getTransaction().commit();
```

Изменение

```
Student  s = (Student) session.get(Student.class, std. getId());
s.setXXX(….);
….
session.save(s);
session.getTransaction().commit();
```

Синхронизация с БД

session.commit();

или

session.flush();

«обратная» синхронизация

session.refresh(s);

Отсоединенные объекты

- Persistent → Detached

  session.commit();

- Detached → Persistent

  session.beginTransaction();

  session.update(s);

  session.getTransaction().commit();

```
session.beginTransaction();
session.merge(s);
session.getTransaction().commit();
```