С помощью какого метода Statement можно выполнить запрос UPDATE?

Один верный ответ

executeUpdate

executeQuery

executeSQL

execute

Для чего нужны ORM системы?

Один верный ответ

преобразовывать записи из базы данных в объекты Java

быстро соединяться с базой данных

создавать нереляционные базы данных

На курсы записано множество различных студентов, студенты ходят на различные курсы. Какой аннотацией Hibernate описывается данная связь?

Один верный ответ

@OneToOne

@OneToMany

@ManyToOne

@ManyToMany

Учитель может преподавать на нескольких курсах, а у курса может быть один преподаватель. Какой аннотацией Hibernate описывается связь к классу Учителя в классе Курс?

Один верный ответ

@OneToOne

@OneToMany

@ManyToOne

@ManyToMany

Какой механизм загрузки данных загрузит связанные данные только при их запросе?

Один верный ответ

FetchType.EGER

FetchType.LAZY

ДОМАШНЕЕ ЗАДАНИЕ:

В этом задании вам необходимо выполнить несколько SQL-запросов к базе данных с помощью Hibernate:

1. **Агрегатная функция:** Получение количества студентов на каждом курсе.
2. **Объединение:** Получение списка курсов для конкретного студента.

Проекты с занятия:

```
import jakarta.persistence.criteria.*;
import models.Course;
import models.Teacher;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

import java.util.List;

public class Main {
    public static void main(String[] args) {
        StandardServiceRegistry registry = new
StandardServiceRegistryBuilder()
                .configure("hibernate.cfg.xml").build();
        Metadata metadata = new
MetadataSources(registry).getMetadataBuilder().build();
        SessionFactory sessionFactory =
metadata.getSessionFactoryBuilder().build();
        Session session = sessionFactory.openSession();

        Transaction transaction = session.beginTransaction();
        CriteriaBuilder builder = session.getCriteriaBuilder();

        // todo Использование агрегатных функций
//        CriteriaQuery<Double> query = builder.createQuery(Double.class);
//        Root<Teacher> teacherRoot = query.from(Teacher.class);
//        query.select(builder.avg(teacherRoot.get("age")));
//        Double resultList5 = session.createQuery(query).getSingleResult();
//        System.out.println(resultList5);

        // todo Использование функционального интерфейса Predicate
//        query.where(builder.greaterThan(courseRoot.get("duration"), 50));
//        CriteriaQuery<Course> query = builder.createQuery(Course.class);
//        Root<Course> courseRoot = query.from(Course.class);
//        query.select(courseRoot);
//        Predicate predicate = builder.or(
//                builder.greaterThan(courseRoot.get("duration"), 30),
//                builder.lessThan(courseRoot.get("teacher").get("age"), 40)
//        );
//        query.where(predicate);
//        List<Course> resultList =
session.createQuery(query).getResultList();
//        resultList.forEach(course ->
System.out.println(course.getDuration() + " - " +
course.getTeacher().getAge()));

        // todo ParameterExpression для реализации BETWEEN
//        CriteriaQuery<Course> query = builder.createQuery(Course.class);
```

```
//        Root<Course> courseRoot = query.from(Course.class);
//        query.select(courseRoot);
//        ParameterExpression<Integer> minPriceParameter =
builder.parameter(Integer.class);
//        ParameterExpression<Integer> maxPriceParameter =
builder.parameter(Integer.class);
//        query.where(builder.between(courseRoot.get("price"),
minPriceParameter, maxPriceParameter));
//        List<Course> resultList1 = session.createQuery(query)
//                .setParameter(minPriceParameter, 50_000)
//                .setParameter(maxPriceParameter, 150_000)
//                .getResultList();
//        resultList1.forEach(course ->
System.out.println(course.getPrice()));

        // todo Пагинация данных с использованием setFirstResult и
setMaxResults
//        CriteriaQuery<Course> query = builder.createQuery(Course.class);
//        Root<Course> courseRoot = query.from(Course.class);
//        query.select(courseRoot);
//        query.orderBy(builder.asc(courseRoot.get("name")));
//        List<Course> resultList3 = session.createQuery(query)
//                .setFirstResult(1)
//                .setMaxResults(10)
//                .getResultList();
//        resultList3.forEach(course ->
System.out.println(course.getName()));

//         todo Использование оператора like
//        CriteriaQuery<Course> query = builder.createQuery(Course.class);
//        Root<Course> courseRoot = query.from(Course.class);
//        query.select(courseRoot);
//        String keyword = "Java";
//        query.select(courseRoot).where(builder.like(courseRoot.get("name"),
"%" + keyword + "%"));
//        List<Course> resultList4 =
session.createQuery(query).getResultList();
//        resultList4.forEach(course ->
System.out.println(course.getName()));

        transaction.commit();
        session.close();
        sessionFactory.close();
    }
}
```

```
import enumeration.CourseType;
import jakarta.persistence.criteria.CriteriaBuilder;
import jakarta.persistence.criteria.CriteriaQuery;
import jakarta.persistence.criteria.Root;
import models.Course;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
```

```
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

import java.util.List;

public class HibernateMain {
    public static void main(String[] args) {
        StandardServiceRegistry registry = new
StandardServiceRegistryBuilder()
                .configure("hibernate.cfg.xml").build();
        Metadata metadata = new
MetadataSources(registry).getMetadataBuilder().build();
        SessionFactory sessionFactory =
metadata.getSessionFactoryBuilder().build();
        Session session = sessionFactory.openSession();

        Transaction transaction = session.beginTransaction();
        CriteriaBuilder builder = session.getCriteriaBuilder();

        // todo Подстановка параметра maxPrice
//        CriteriaQuery<Course> query = builder.createQuery(Course.class);
//        Root<Course> root = query.from(Course.class);
//        int maxPrice = 30_000;
//        query.select(root).where(builder.greaterThan(root.get("price"),
maxPrice));
//        List<Course> resultList =
session.createQuery(query).getResultList();
//        resultList.forEach(course ->
System.out.println(course.getPrice()));

        // todo Получение количества курсов для каждого типа курса
(группировка по типу)
        CriteriaQuery<Object[]> query = builder.createQuery(Object[].class);
        Root<Course> courseRoot = query.from(Course.class);
        query.multiselect(
                courseRoot.get("type"),
                builder.count(courseRoot)
        );
        query.groupBy(courseRoot.get("type"));
        List<Object[]> resultList1 =
session.createQuery(query).getResultList();
        for (Object[] result : resultList1) {
            CourseType type = (CourseType) result[0];
            long count = (long) result[1];
            System.out.println("Тип курса: " + type + ", " + "Кол-во: " +
count);
        }

        transaction.commit();
        session.close();
        sessionFactory.close();
    }
}
```

```
import jakarta.persistence.criteria.*;
import models.Course;
import models.Student;
import models.Teacher;
```

```java
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

import java.util.List;
import java.util.Objects;

public class JoinExp {
    /*
    ManyToOne и OneToOne - Одиночная ассоциация

    OneToMany и ManyToMany - Коллекция ассоциаций
     */

    public static void main(String[] args) {
        StandardServiceRegistry registry = new
StandardServiceRegistryBuilder()
                .configure("hibernate.cfg.xml").build();
        Metadata metadata = new
MetadataSources(registry).getMetadataBuilder().build();
        SessionFactory sessionFactory =
metadata.getSessionFactoryBuilder().build();
        Session session = sessionFactory.openSession();

        Transaction transaction = session.beginTransaction();
        CriteriaBuilder builder = session.getCriteriaBuilder();

        // todo Коллекция ассоциаций - ListJoin
        CriteriaQuery<Object[]> query = builder.createQuery(Object[].class);
        Root<Student> studentRoot = query.from(Student.class);
        ListJoin<Student, Course> courseListJoin =
studentRoot.joinList("courses");

        query.multiselect(studentRoot, courseListJoin);

        List<Object[]> resultList1 =
session.createQuery(query).getResultList();
        for (Object[] result : resultList1) {
            Student student = (Student) result[0];
            Course course = (Course) result[1];
            System.out.println("Студент: " + student.getName() + ", Курс: " +
course.getName());
        }

        // todo Одиночная ассоциация - Join
//        CriteriaQuery<Object[]> query =
builder.createQuery(Object[].class);
//        Root<Course> courseRoot = query.from(Course.class);
//        Join<Course, Teacher> teacherJoin = courseRoot.join("teacher");
//
//        query.multiselect(courseRoot, teacherJoin.get("name"),
teacherJoin.get("age"));
//
//        List<Object[]> resultList =
session.createQuery(query).getResultList();
//
//        for (Object[] result : resultList) {
//            Course course = (Course) result[0];
//            String teacherName = (String) result[1];
//            int teacherAge = (int) result[2];
```

```
//            System.out.println("Kypc: " + course.getName() + " - " +
teacherName + ": " + teacherAge);
//        }

        transaction.commit();
        session.close();
        sessionFactory.close();
    }
}
```