IT-Academy

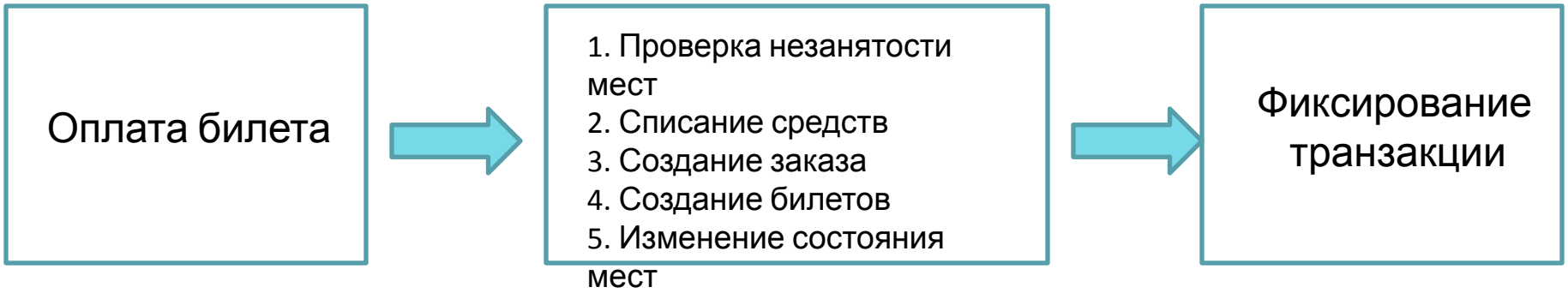
Hibernate
Транзакции и параллелизм

Вопросы

IT-Academy

Понимание транзакций БД

- БД реализует понятие единицы работы, как **транзакцию БД** (иногда называемую системная транзакция). Транзакции БД группируют операции доступа к данным. Транзакция гарантирует один из результатов работы: она или будет зафиксирована или откачена назад. Следовательно, транзакции БД действительно всегда атомарны.



Понимание транзакций БД

IT-Academy

```
EntityManager em = HibernateUtil.getEntityManager();
Department economist = new Department("Economist");
try {
    em.getTransaction().begin();
    em.persist(economist);
    // Other business logic stuff
    em.getTransaction().commit();
} catch (RollbackException e) {
    em.getTransaction().rollback();
}
em.close();
```

Проблемы изоляции транзакций

IT-Academy

- *Потерянное обновление*
- *Грязное чтение*
- *Неповторяемое чтение*
- *Фантомное чтение*

Установка уровня изоляции

```
<property name="hibernate.dialect">
   org.hibernate.dialect.MySQLDialect
</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">yuli</property>
<property name="hibernate.connection.pool_size">10</property>
<property name="hibernate.connection.isolation">2</property>
<property name="show_sql">true</property>
<property name="hibernate.hbm2ddl.auto">update</property>
```

Значения для этой опции выглядят следующим образом (вы также можете найти их, как константы в java.sql.Connection):

- **0 – транзакции не поддерживаются;**
- **1 – изоляция уровня чтения неподтвержденного;**
- **2 – изоляция уровня чтения подтвержденного;**
- **4 – изоляция уровня повторяемого чтения;**
- **8 – упорядоченная изоляция.**

Пессимистичная блокировка

IT-Academy

LockModeType	LockMode	Description
NONE	NONE	Блокировки отсутствуют. Все объекты переключаются в этот режим блокировки в конце транзакции. Объекты, связанные с объектом Session с помощью вызова методов update () или saveOrUpdate (), также запускаются в этом режиме блокировки.
READ and OPTIMISTIC	READ	Версия сущности проверяется в конце текущей транзакции.
WRITE and OPTIMISTIC_FORCE_INCREMENT	WRITE	Версия сущности автоматически увеличивается, даже если объект не изменился.
PESSIMISTIC_FORCE_INCREMENT	PESSIMISTIC_FORCE_INCREMENT	Объект заблокирован пессимистично, и его версия автоматически увеличивается, даже если объект не изменился.
PESSIMISTIC_READ	PESSIMISTIC_READ	Объект заблокирован пессимистично с использованием разделяемой блокировки, если база данных поддерживает такую функцию. В противном случае используется явная блокировка.
PESSIMISTIC_WRITE	PESSIMISTIC_WRITE, UPGRADE	Объект заблокирован с использованием явной блокировки.
PESSIMISTIC_WRITE with a javax.persistence.lock. timeoutsetting of 0	UPGRADE_NOWAIT	Запрос на захват блокировки мгновенно терпит неудачу, если строка s уже заблокирована.
PESSIMISTIC_WRITE with a javax.persistence.lock .timeoutsetting of -2	UPGRADE_SKIPLOCKED	Запрос на захват блокировки пропускает уже заблокированные записи. Он использует SELECT ... FOR UPDATE SKIP LOCKED в Oracle и PostgreSQL 9.5, или SELECT ... с (rowlock, updlock, readpast) в SQL Server.

Нет блокировки

```
EntityManager em = HibernateUtil.getEntityManager();
em.getTransaction().begin();
Cat cat = em.find(Cat.class, 1L);
cat.setName("New");
em.getTransaction().commit();
em.close();
```

select catlockall0_.id as id1_0_0_, catlockall0_.name as name2_0_0_
from CatLockAll catlockall0_ where catlockall0_.id=?
update CatLockAll set name=? where id=? and name=?

Оптимистическая блокировка. Version

IT-Academy

```
@Data @AllArgsConstructor @NoArgsConstructor
@Entity
@OptimisticLocking(type = OptimisticLockType.VERSION)
public class CatLockVersion {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String owner;
    @Version
    private Integer version;
}
```

```
EntityManager em = HibernateUtil.getEntityManager();
em.getTransaction().begin();
CatLockVersion cat = em.find(CatLockVersion.class, 1L);
cat.setName("New");
em.getTransaction().commit();
em.close();
```

Select id, name, owner, version
from CatLockVersion where id=1
update CatLockVersion
set name= New, owner= Tim, version=1 **where** id=1 and version=0

Оптимистическая блокировка. All

```
@Data @AllArgsConstructor @NoArgsConstructor
@Entity
@DynamicUpdate
@OptimisticLocking(type = OptimisticLockType.ALL)
public class CatLockAll {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String owner;
}
```

```
EntityManager em = HibernateUtil.getEntityManager();
em.getTransaction().begin();
CatLockAll cat = em.find(CatLockAll.class, 1L);
cat.setName("New");
em.getTransaction().commit();
em.close();
```

Select id, name, owner
from CatLockAll where id=1
update CatLockAll
set name= New **where** id=1 and name=AllCat and owner=Tim

Оптимистическая блокировка. Dirty

IT-Academy

```
@Data @AllArgsConstructor @NoArgsConstructor
@Entity
@DynamicUpdate
@SelectBeforeUpdate
@OptimisticLocking(type = OptimisticLockType.DIRTY)
public class CatLockDirty {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String owner;
}
```

```
EntityManager em = HibernateUtil.getEntityManager();
em.getTransaction().begin();
CatLockDirty cat = em.find(CatLockDirty.class, 1L);
cat.setName("New");
em.getTransaction().commit();
em.close();
```

Select id, name, owner
from CatLockDirty **where** id=1
update CatLockDirty
set name= New where **where** id=1 and name=Dirty

Оптимистическая блокировка

IT-Academy

```
EntityManager em = HibernateUtil.getEntityManager();
em.getTransaction().begin();
CatLockVersion cat = em.find(CatLockVersion.class, 1L,
    LockModeType.OPTIMISTIC);
cat.setName("New");
em.getTransaction().commit();
em.close();
```

select id, name, owner, version
from CatLockVersion where id=1
update CatLockVersion set name=New, owner=Tim, version=1 where id=1 and version=0
select version from CatLockVersion where id =1

Оптимистическая блокировка с ошибкой

IT-Academy

```
EntityManager em = HibernateUtil.getEntityManager();
em.getTransaction().begin();
CatLockVersion cat = em.find(CatLockVersion.class, 1L, LockModeType.OPTIMISTIC);
cat.setName("New");
new Thread(()-> {
    EntityManager entityManager = HibernateUtil.getEntityManager();
    entityManager.getTransaction().begin();
    CatLockVersion updatedCat = entityManager.find(CatLockVersion.class, 1L);
    updatedCat.setName("Updated Cat");
    entityManager.getTransaction().commit();
}).start();
Thread.sleep(500);
em.getTransaction().commit();
```

select id , name, owner, version from CatLockVersion where id=1
select id , name, owner, version from CatLockVersion where id=1
update CatLockVersion set name= Updated Cat, owner= Tim, version=1 where id=1 and version=0
update CatLockVersion set name= New, owner= Tim, version=1 where id=1 and version=0
HHH000346: Error during managed flush [Batch update returned unexpected row count from update [0]; actual row count: 0; expected: 1]

Оптимистическая блокировка с инкрементом версии

```
EntityManager em = HibernateUtil.getEntityManager();
em.getTransaction().begin();
CatLockVersion cat = em.find(CatLockVersion.class, 1L,
    LockModeType.OPTIMISTIC_FORCE_INCREMENT);
cat.setName("New");
em.getTransaction().commit();
em.close();
```

select id, name, owner, version
from CatLockVersion where id=1
update CatLockVersion set version=1 where id=1 and version=0

Пессимистическая блокировка

IT-Academy

```
EntityManager em = HibernateUtil.getEntityManager();
em.getTransaction().begin();
Cat cat = em.find(Cat.class, 1L, LockModeType.PESSIMISTIC_WRITE);
cat.setName("New");
new Thread(()-> {
    EntityManager entityManager = HibernateUtil.getEntityManager();
    entityManager.getTransaction().begin();
    Cat updatedCat = entityManager.find(Cat.class, 1L);
    updatedCat.setName("Updated Cat");
    entityManager.getTransaction().commit();
}).start();
Thread.sleep(500);
em.getTransaction().commit();
em.clear();
```

select id , name, owner, version from Cat where id=1 **for update**
select id , name, owner, version from Cat where id=1
update CatLockVersion set name= Updated Cat, owner= Tim, version=1 where id=1 and version=0
update CatLockVersion set name= New, owner= Tim, version=1 where id=1 and version=0
select id , name, owner, version from Cat where id=1

Cat(id=1, name=Updated Cat, owner=Tim)

Вопросы

IT-Academy

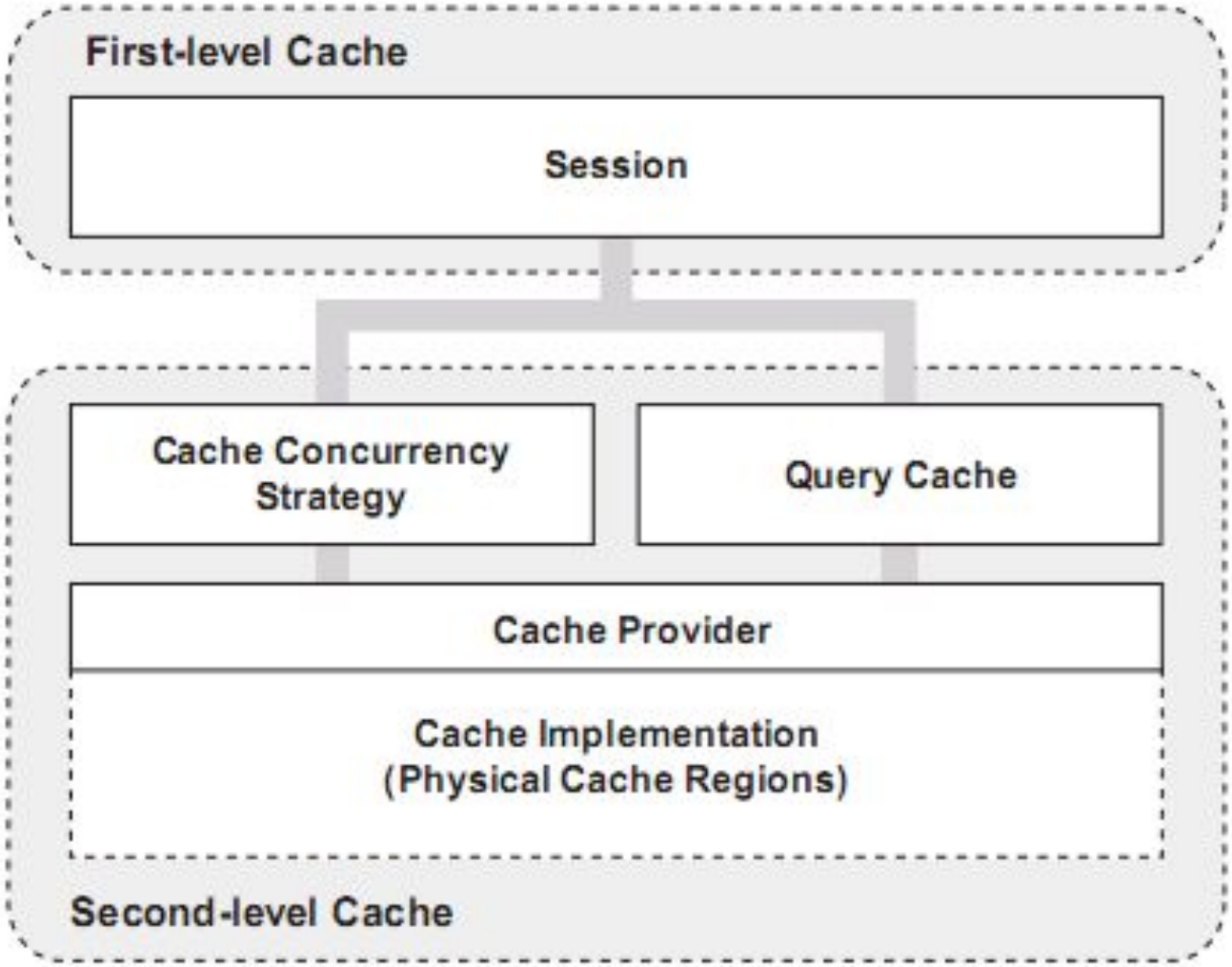
Кэш второго уровня Hibernate

Стратеги кэширования и области

- *Транзакционный* – связанный с текущей единицей работы, которая может быть фактическая транзакция БД или транзакция приложения. Она корректна и используется во время работы единицы работы. Каждая единица работы имеет свой кэш.
- *Процессный* – распределяется между многими (возможно одновременными) единицами работы или транзакции. Это означает, что данные в процессном КЭШе доступны одновременно выполняемым операциям, очевидно с последствиями для изоляции транзакций. Процессный кэш может хранить хранимые объекты целиком в КЭШе, или может хранить их состояние в разобранном формате.
- *Кластерный* – распределяется между несколькими процессами на одной машине или между несколькими машинами в кластере

Стратеги кэширования и области

Архитектура КЭШа в Hibernate



Кэш второго уровня Hibernate

IT-Academy

Политика КЭШа включает в себя настройку следующих параметров:

- Включен ли кэш второго уровня
- Стратегию параллелизма Hibernate
- Политика истекания срока кэширования (такую, как тайм-аут, LRU, зависимую от ОП)
- Физическое устройство КЭШа (в памяти, индексируемые файлы, кластерная репликация)

Встроенные стратегии параллелизма

IT-Academy

Есть четыре встроенных стратегии параллелизма, представляющие снижение уровня строгости, в терминах изолированности транзакции:

- *Транзакционная* – доступна только в управляемой среде. Она гарантирует полную изоляцию транзакций до повторяемого чтения, если это требуется. Используйте эту стратегию для данных которых в большинстве считываются, в которых очень важно предотвратить появление устаревших данных в параллельных транзакциях, в редких случаях обновления.
- *Чтение-запись* – поддерживает изоляцию чтения подтвержденного, используя механизм временных меток. Она доступна только в некластерных средах.
- *Нестрогое-чтение-запись* - не дает никакой гарантии согласованности между КЭШем и БД. Если есть возможность одновременного доступа к одной сущности, то вам необходимо настроить достаточно короткий срок истечения тайм-аута.
- *Только-для-чтения* – данная стратегия подходит для данных, которые никогда не меняются. Используйте её только для справочных данных.

Выбор поставщика КЭШа

IT-Academy

Следующие поставщики встроены в Hibernate:

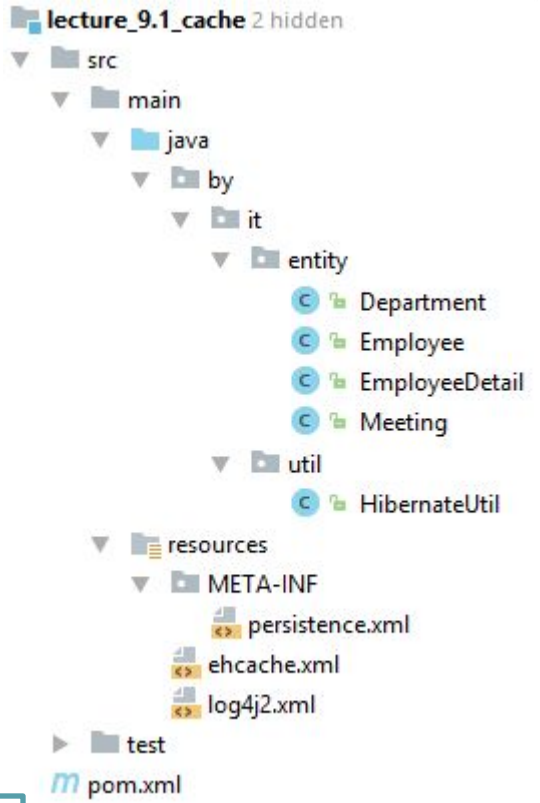
- *EHCache* предназначен для процессного и кластерного кэширования в одной JVM. Он может кэшировать в памяти или на диске и поддерживает опциональный Hibernate кэш результатов запроса.
- *Infinispan* – это полностью транзакционно-репликационный кластеризованный кэш с поддержкой КЭШа запросов, предполагая, что часы в кластере синхронизированы.
- *SwarmCache* – это кластерный кэш, основанный на JGroups. Он использует кластерное аннулирование, но не поддерживает кэш Hibernate запросов.
- *JCache* – это полностью транзакционно-репликационный кластеризованный кэш, также основанных на *JGroups*. Кэш запросов Hibernate поддерживается, предполагая, что часы в кластере синхронизированы.

Hibernate кэширование на практике

IT-Academy

```
<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_2_0.xsd"
             version="2.0">

  <persistence-unit name="by.it.test" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <class>by.it.entity.Department</class>
    <class>by.it.entity.Employee</class>
    <class>by.it.entity.EmployeeDetail</class>
    <class>by.it.entity.Meeting</class>
    <exclude-unlisted-classes>true</exclude-unlisted-classes>
    <properties>
      <property name="javax.persistence.jdbc.driver" value="org.h2.Driver"/>
      <property name="javax.persistence.jdbc.url" value="jdbc:h2:mem:jpadb"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.password" value=""/>
      <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>
      <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
      <property name="hibernate.use_sql_comments" value="false"/>
      <property name="hibernate.show_sql" value="false"/>
      <property name="hibernate.cache.region.factory_class"
                value="org.hibernate.cache.ehcache.EhCacheRegionFactory"/>
      <property name="hibernate.cache.use_second_level_cache" value="true"/>
      <property name="hibernate.cache.use_query_cache" value="true"/>
      <property name="net.sf.ehcache.configurationResourceName" value="ehcache.xml"/>
    </properties>
  </persistence-unit>
</persistence>
```



Hibernate кэширование на практике

IT-Academy

```
<!--EhCache-->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-ehcache</artifactId>
    <version>${hibernate.version}</version>
</dependency>
<dependency>
    <groupId>net.sf.ehcache</groupId>
    <artifactId>ehcache-core</artifactId>
    <version>2.6.11</version>
</dependency>
```

Hibernate кэширование на практике

IT-Academy

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache>
    <defaultCache maxEntriesLocalHeap="20"
                  eternal="false"
                  timeToIdleSeconds="120"
                  timeToLiveSeconds="200"
                  memoryStoreEvictionPolicy="LRU"/>
    <cache name="by.it.pojos.Employee"
           maxEntriesLocalHeap="6"
            eternal="true">
    </cache>
    <cache name="by.it.pojos.Department"
           maxEntriesLocalHeap="2"
           eternal="false"
           timeToIdleSeconds="300"
           timeToLiveSeconds="600">
        <persistence strategy="localTempSwap"/>
    </cache>
    <cache name="org.hibernate.cache.StandardQueryCache"
           maxEntriesLocalHeap="5"
            eternal="false"
            timeToLiveSeconds="120">
        <persistence strategy="localTempSwap"/>
    </cache>
</ehcache>
```

Hibernate кэширование на практике

IT-Academy

```
@Data @NoArgsConstructor @AllArgsConstructor
@Entity
@Table(name="EMPLOYEE")
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class Employee implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id @GeneratedValue(strategy = IDENTITY)
    @Column(name = "EMPLOYEE_ID", unique = true)
    private Long employeeId;
    @Column(name = "FIRST_NAME")
    private String firstname;
    @Column(name = "LAST_NAME")
    private String lastname;
    @Temporal(TemporalType.TIMESTAMP)
    private Date date;

    @OneToOne(mappedBy = "employee", cascade = CascadeType.PERSIST)
    private EmployeeDetail employeeDetail;

    @ManyToOne
    @JoinColumn(name = "DEPARTMENT_ID")
    private Department department;

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(name = "EMPLOYEE_MEETING", joinColumns = {@JoinColumn(name = "EMPLOYEE_ID")},
            inverseJoinColumns = {@JoinColumn(name = "MEETING_ID")}
    )
    private Set<Meeting> meetings = new HashSet<>(0);
```

Hibernate кэширование на практике

IT-Academy

```
public class MainLoader {
    private static HibernateUtil util;

    public static void main(String... args) throws Exception {
        util = HibernateUtil.getInstance();
        loadEmployee(1L);
        loadEmployee(1L);
        loadEmployee(1L);
//...
    }
    private static void loadEmployee(Long id) {
        Session session = util.getSession();
        session.beginTransaction();
        Employee e = (Employee) session.get(Employee.class, id);
        session.getTransaction().commit();
    }
```

```
Hibernate: select employee0_.EMPLOYEE_ID as EMPLOYEE1_1_0_, employee0_.date as date2_1_0_, employee0_.DEPARTMENT_ID a
 DEPARTME1_0_1_, department1_.NAME as NAME2_0_1_, employeede2_.EMPLOYEE_ID as EMPLOYEE1_2_2_, employeede2_.CITY as C
 employee0_ left outer join DEPARTMENT department1_ on employee0_.DEPARTMENT_ID=department1_.DEPARTMENT_ID left oute
```

Hibernate кэширование на практике

IT-Academy

```java
package by.academy.it.dao;

import by.academy.it.pojos.Employee;
import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;

import java.util.List;

public class EmployeeDao extends BaseDao <Employee> {

    public EmployeeDao(SessionFactory sessionFactory) {
        super(sessionFactory);
    }

    public List<Employee> getEmployeeList() {
        Session session = getSession();
        Query query = session.createQuery("FROM by.academy.it.pojos.Employee");
        query.setCacheable(true);
        List employee = query.list();
        session.close();
        return employee;
    }
}
```

Вопросы

IT-Academy

IT-Academy

Спасибо за внимание