# **Demo Sử dụng Native Query Hibernate để thao tác với dữ liệu.**



- Phần bài Lab này thực hiện Demo như bài lab 4.2 nhưng sẽ viết những lệnh Native Sql thay cho Criteria: FROM, SELECT, WHERE, ORDER BY, UPDATE, DELETE, PAGGING.

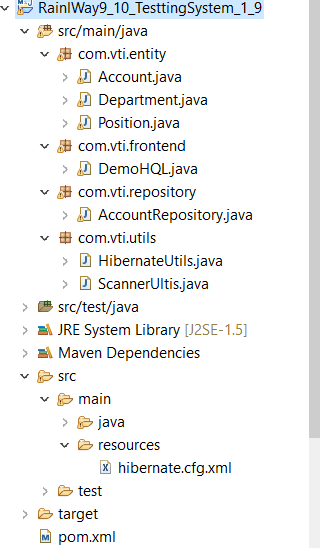
- Thực hiện Demo trên bảng Account, chú ý các nội dung thay đổi được viết lại dưới dạng Native Sql trong **AccountRepository.**

- Thêm hàm tạo theo Name trong các class Department và Possition

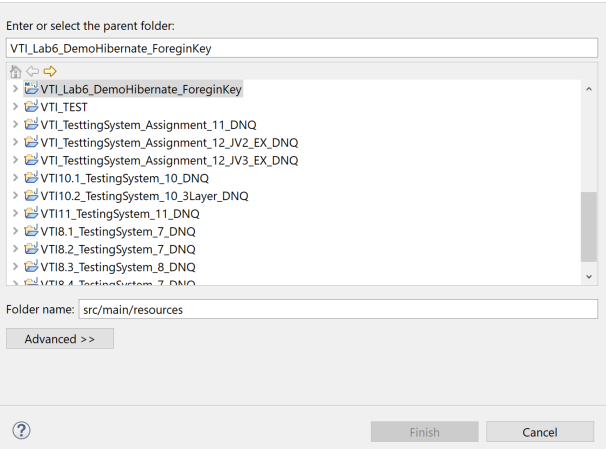
# **Tạo** khung **chương trình:**

Tạo mới 1 project maven: RainlWay9\_10\_TesttingSystem\_1\_9

Tạo các Package trong src: com.vti.repository, com.vti.entiy, com.vti.frontend, com.vti.utils



Tạo thư mục src/main/resources:



Tạo file: hibernate.cfg.xml

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <!DOCTYPE hibernate-configuration PUBLIC  "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">  <hibernate-configuration>  <session-factory>  <!-- Database connection settings -->  <property name=*"connection.driver\_class"*>com.mysql.cj.jdbc.Driver</property>  <property name=*"connection.url"*>jdbc:mysql://localhost:3306/TestingSystem</property>  <property name=*"connection.username"*>root</property>  <property name=*"connection.password"*>root</property>  <!-- format code SQL -->  <property name=*"show\_sql"*>true</property>  <property name=*"hibernate.format\_sql"*>true</property>  <property name=*"connection.pool\_size"*>10</property>  <!-- other -->  <property name=*"hibernate.connection.characterEncoding"*>utf8</property>  </session-factory>  </hibernate-configuration> |



Sửa file: pom.xml

|  |
| --- |
| <project xmlns=*"http://maven.apache.org/POM/4.0.0"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>  <modelVersion>4.0.0</modelVersion>  <groupId>com.vti</groupId>  <artifactId>RainlWay9\_10\_TesttingSystem\_1\_9</artifactId>  <version>0.0.1-SNAPSHOT</version>  <packaging>jar</packaging>  <name>RainlWay9\_10\_TesttingSystem\_1\_9</name>  <url>http://maven.apache.org</url>  <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  </properties>  <dependencies>  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>3.8.1</version>  <scope>test</scope>  </dependency>  <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>8.0.20</version>  </dependency>  <dependency>  <groupId>org.projectlombok</groupId>  <artifactId>lombok</artifactId>  <version>1.18.20</version>  </dependency>  <dependency>  <groupId>org.hibernate</groupId>  <artifactId>hibernate-core</artifactId>  <version>5.4.17.Final</version>  </dependency>    </dependencies>  </project> |



Tạo file: TestingSystem.sql



# **Tạo Class HibernateUtils trong package ultis:**

|  |
| --- |
| **package** com.vti.ultis;  **import** org.hibernate.Session;  **import** org.hibernate.SessionFactory;  **import** org.hibernate.boot.registry.StandardServiceRegistryBuilder;  **import** org.hibernate.cfg.Configuration;  **import** org.hibernate.service.ServiceRegistry;  **import** com.vti.entity.Account;  **import** com.vti.entity.Department;  **import** com.vti.entity.Position;  **public** **class** HibernateUtils {  **private** **static** HibernateUtils instance;  **private** Configuration configuration;  **private** SessionFactory sessionFactory;  **public** **static** HibernateUtils getInstance() {  **if** (**null** == instance) {  instance = **new** HibernateUtils();  }  **return** instance;  }  **private** HibernateUtils() {  configure();  }  **private** **void** configure() {  // load configuration  configuration = **new** Configuration();  configuration.configure("hibernate.cfg.xml");  // add entity  configuration.addAnnotatedClass(Account.**class**);  configuration.addAnnotatedClass(Department.**class**);  configuration.addAnnotatedClass(Position.**class**);  }  **private** SessionFactory buildSessionFactory() {  **if** (**null** == sessionFactory || sessionFactory.isClosed()) {  ServiceRegistry serviceRegistry = **new** StandardServiceRegistryBuilder()  .applySettings(configuration.getProperties()).build();  sessionFactory = configuration.buildSessionFactory(serviceRegistry);  }  **return** sessionFactory;  }  **public** **void** closeFactory() {  **if** (**null** != sessionFactory && sessionFactory.isOpen()) {  sessionFactory.close();  }  }  **public** Session openSession() {  buildSessionFactory();  **return** sessionFactory.openSession();  }  } |

****

# **Tạo Class ScannerUltis trong package ultis:**

|  |
| --- |
| **package** com.vti.utils;  **import** java.text.SimpleDateFormat;  **import** java.time.LocalDate;  **import** java.util.Scanner;  **public** **class** ScannerUltis {  **private** **static** Scanner *sc* = **new** Scanner(System.***in***);  **public** **static** **int** inputInt() {  **while** (**true**) {  **try** {  **return** Integer.*parseInt*(*sc*.next().trim());  } **catch** (Exception e) {  System.***err***.println("Nhập lại:");  }  }  }  **public** **static** **int** inputIntPositive() {  **while** (**true**) {  **try** {  **int** intPositive = Integer.*parseInt*(*sc*.next());  **if** (intPositive >= 0) {  **return** intPositive;  } **else** {  System.***err***.println("Nhập lại:");  }  } **catch** (Exception e) {  System.***err***.println("Nhập lại:");  }  }  }  **public** **static** Float inputFloat() {  **while** (**true**) {  **try** {  **return** Float.*parseFloat*(*sc*.next());  } **catch** (Exception e) {  System.***err***.println("Nhập lại:");  }  }  }  **public** **static** Double inputDouble() {  **while** (**true**) {  **try** {  **return** Double.*parseDouble*(*sc*.next());  } **catch** (Exception e) {  System.***err***.println("Nhập lại:");  }  }  }  **public** **static** String inputString() {  **while** (**true**) {  String string = *sc*.next().trim();  **if** (!string.isEmpty()) {  **return** string;  } **else** {  System.***err***.println("Nhập lại:");  }  }  }  **public** **static** LocalDate inputLocalDate() {  System.***out***.println("Nhập theo định dạng yyyy-MM-dd");  SimpleDateFormat format = **new** SimpleDateFormat("yyyy-MM-dd");  **while** (**true**) {  String localdate = *sc*.next().trim();  **try** {  **if** (format.parse(localdate) != **null**) {  LocalDate lc = LocalDate.*parse*(localdate);  **return** lc;  }  } **catch** (Exception e) {  System.***err***.println("Nhập lại:");  }  }  }  **public** **static** String inputEmail() {  **while** (**true**) {  String email = ScannerUltis.*inputString*();  **if** (email == **null** || !email.contains("@")) {  System.***out***.print("Nhập lại: ");  } **else** {  **return** email;  }  }  }  **public** **static** **int** inputFunction(**int** a, **int** b, String errorMessage) {  **while** (**true**) {  **int** number = ScannerUltis.*inputInt*();  **if** (number >= a && number <= b) {  **return** number;  } **else** {  System.***err***.println(errorMessage);  }  }  }  **public** **static** String inputPassword() {  **while** (**true**) {  String password = ScannerUltis.*inputString*();  **if** (password.length() < 6 || password.length() > 12) {  System.***out***.print("Nhập lại: ");  **continue**;  }  **boolean** hasAtLeast1Character = **false**;  **for** (**int** i = 0; i < password.length(); i++) {  **if** (Character.*isUpperCase*(password.charAt(i)) == **true**) {  hasAtLeast1Character = **true**;  **break**;  }  }  **if** (hasAtLeast1Character == **true**) {  **return** password;  } **else** {  System.***out***.print("Mời bạn nhập lại password: ");  }  }  }  **public** **static** String inputPhoneNumber() {  **while** (**true**) {  String number = ScannerUltis.*inputString*();  **if** (number.length() > 12 || number.length() < 9) {  **continue**;  }  **if** (number.charAt(0) != '0') {  **continue**;  }  **boolean** isNumber = **true**;  **for** (**int** i = 0; i < number.length(); i++) {  **if** (Character.*isDigit*(number.charAt(i)) == **false**) {  isNumber = **false**;  **break**;  }  }  **if** (isNumber == **true**) {  **return** number;  } **else** {  System.***out***.print("Nhập lại: ");  }  }  }  } |

****

# **Tạo Class Account trong Entity:**

|  |
| --- |
| **package** com.vti.entity;  **import** java.io.Serializable;  **import** java.util.Date;  **import** javax.persistence.Column;  **import** javax.persistence.Entity;  **import** javax.persistence.GeneratedValue;  **import** javax.persistence.GenerationType;  **import** javax.persistence.Id;  **import** javax.persistence.JoinColumn;  **import** javax.persistence.ManyToOne;  **import** javax.persistence.Table;  **import** javax.persistence.Temporal;  **import** javax.persistence.TemporalType;  **import** org.hibernate.annotations.Cascade;  **import** org.hibernate.annotations.CascadeType;  **import** org.hibernate.annotations.CreationTimestamp;  **import** lombok.Getter;  **import** lombok.NoArgsConstructor;  **import** lombok.Setter;  **import** lombok.ToString;  @Entity  @Table(name = "`Account`", catalog = "TestingSystem")  @Setter  @Getter  @NoArgsConstructor  @ToString  **public** **class** Account **implements** Serializable {  @Column(name = "AccountID")  @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  **private** **short** id;  @Column(name = "Email", length = 50, nullable = **false**, unique = **true**, updatable = **false**)  **private** String email;  @Column(name = "Username", length = 50, nullable = **false**, unique = **true**, updatable = **false**)  **private** String username;  @Column(name = "FullName", length = 50, nullable = **false**)  **private** String fullname;  @ManyToOne  @JoinColumn(name = "DepartmentID", nullable = **false**)  @Cascade(value = { CascadeType.REMOVE, CascadeType.SAVE\_UPDATE })  **private** Department department;  @ManyToOne  @JoinColumn(name = "PositionID", nullable = **false**)  **private** Position position;  @Column(name = "CreateDate")  @Temporal(TemporalType.TIMESTAMP)  @CreationTimestamp  **private** Date createDate;  @Override  **public** String toString() {  **return** "Account [id=" + id + ", email=" + email + ", username=" + username + ", fullname=" + fullname  + ", department=" + department + ", position=" + position + ", createDate=" + createDate + "]";  }  } |

****

# **Tạo Class Department trong Entity: trong Class này chú ý thêm hàm khởi tạo theo Name để sử dụng.**

|  |
| --- |
| **package** com.vti.entity;  **import** java.io.Serializable;  **import** java.util.Date;  **import** java.util.List;  **import** javax.persistence.Column;  **import** javax.persistence.Entity;  **import** javax.persistence.FetchType;  **import** javax.persistence.GeneratedValue;  **import** javax.persistence.GenerationType;  **import** javax.persistence.Id;  **import** javax.persistence.OneToMany;  **import** javax.persistence.Table;  **import** org.hibernate.annotations.Cascade;  **import** org.hibernate.annotations.CascadeType;  **import** lombok.Getter;  **import** lombok.NoArgsConstructor;  **import** lombok.Setter;  **import** lombok.ToString;  @Entity  @Table(name = "Department", catalog = "TestingSystem")  @Setter  @Getter  @NoArgsConstructor  @ToString  **public** **class** Department **implements** Serializable {  @Column(name = "DepartmentID")  @Id  @GeneratedValue(strategy = GenerationType.IDENTITY)  **private** **short** id;  @Column(name = "DepartmentName", length = 30, nullable = **false**, unique = **true**)  **private** String name;  @OneToMany(mappedBy = "department", fetch = FetchType.EAGER)  @Cascade(value = { CascadeType.REMOVE, CascadeType.SAVE\_UPDATE })  **private** List<Account> account;  **public** Department(String name) {  **super**();  **this**.name = name;  }  } |

****

# **Tạo Class Position trong Entity: trong Class này chú ý thêm hàm khởi tạo theo Name để sử dụng.**

|  |
| --- |
| **package** com.vti.entity;  **import** java.util.Date;  **import** java.util.List;  **import** javax.persistence.Column;  **import** javax.persistence.Entity;  **import** javax.persistence.EnumType;  **import** javax.persistence.Enumerated;  **import** javax.persistence.GeneratedValue;  **import** javax.persistence.GenerationType;  **import** javax.persistence.Id;  **import** javax.persistence.OneToMany;  **import** javax.persistence.Table;  **import** lombok.Getter;  **import** lombok.NoArgsConstructor;  **import** lombok.Setter;  **import** lombok.ToString;  @Entity  @Table(name = "Position", catalog = "TestingSystem")  @Setter  @Getter  @NoArgsConstructor  @ToString  **public** **class** Position {  @Column(name = "PositionID")  @Id  @GeneratedValue(strategy = GenerationType.***IDENTITY***)  **private** **short** id;  @Column(name = "PositionName", nullable = **false**, unique = **true**)  @Enumerated(EnumType.***STRING***)  **private** PositionName name;  @OneToMany(mappedBy = "position")  List<Account> accounts;  **public** **enum** PositionName {  ***Dev***, ***Test***, ***Scrum\_Master***, ***PM***  }  **public** Position(PositionName name) {  **super**();  **this**.name = name;  }      } |

****

# **Tạo Class AccountRepository trong Repository:**

|  |
| --- |
| **package** com.vti.repository;  **import** java.text.ParseException;  **import** java.text.SimpleDateFormat;  **import** java.time.LocalDate;  **import** java.util.ArrayList;  **import** java.util.Date;  **import** java.util.List;  **import** javax.persistence.criteria.CriteriaBuilder;  **import** javax.persistence.criteria.CriteriaQuery;  **import** javax.persistence.criteria.CriteriaUpdate;  **import** javax.persistence.criteria.Expression;  **import** javax.persistence.criteria.Root;  **import** org.hibernate.Session;  **import** org.hibernate.query.NativeQuery;  **import** org.hibernate.query.Query;  **import** com.vti.entity.Account;  **import** com.vti.entity.Department;  **import** com.vti.entity.Position;  **import** com.vti.entity.Position.PositionName;  **import** com.vti.utils.HibernateUtils;  **import** javassist.expr.NewArray;  **public** **class** AccountRepository {  **private** HibernateUtils hibernateUtils;  **public** AccountRepository() {  hibernateUtils = HibernateUtils.getInstance();  }  @SuppressWarnings("unchecked")  // Lấy danh sách tất cả các Account trên hệ thống.  **public** List<Account> get\_FROM() {  Session session = **null**;  **try** {  // get session  session = hibernateUtils.openSession();  // create Native query  // Câu lênh sql như câu lệnh viết trong Workbench  NativeQuery query = session.createNativeQuery(  "SELECT a.AccountID, a.Email, a.Username, a.FullName, d.DepartmentName, p.PositionName FROM account a \r\n"  + "INNER JOIN department d ON a.DepartmentID = d.DepartmentID\r\n"  + "INNER JOIN position p ON a.PositionID = p.PositionID;");  // Phần query này sẽ tạo ra 1 list các array object, sử dụng vòng lặp foreach để truy vấn tới các phần tử trong array, lấy các giá trị trong DB theo Index của array bắt đầu từ 0.  List<Account> accountList = **new** ArrayList<Account>();  List<Object[]> accounts = query.getResultList();  **for** (Object[] objects : accounts) {  Account account = **new** Account();  account.setId(Short.parseShort(objects[0].toString()));  account.setEmail(objects[1].toString());  account.setUsername(objects[2].toString());  account.setFullname(objects[3].toString());  // Chý ý cần tạo mới hàm tạo 1 tham số cho Department  account.setDepartment(**new** Department(objects[4].toString()));  account.setPosition(**new** Position(PositionName.valueOf(objects[5].toString())));  accountList.add(account);  }  // Cách khác.  // NativeQuery query = session.createNativeQuery("SELECT \* FROM account", Account.class );  // List<Account> accountList = query.getResultList();  **return** accountList;  } **finally** {  **if** (session != **null**) {  session.close();  }  }  }  //Giới hạn điều kiện trả về với WHERE theo id  **public** Account get\_ByID(**short** id) {  Session session = **null**;  **try** {  // get session  session = hibernateUtils.openSession();  Account account = **new** Account();  // create Native query  // Câu lênh sql như câu lệnh viết trong Workbench  NativeQuery query = session.createNativeQuery(  "SELECT a.AccountID, a.Email, a.Username, a.FullName, d.DepartmentName, p.PositionName FROM account a \r\n"  + "INNER JOIN department d ON a.DepartmentID = d.DepartmentID\r\n"  + "INNER JOIN position p ON a.PositionID = p.PositionID\r\n" + "WHERE a.AccountID = ?;");  query.setParameter(1, id);  Object[] objects = (Object[]) query.getSingleResult();  account.setId(Short.parseShort(objects[0].toString()));  account.setEmail(objects[1].toString());  account.setUsername(objects[2].toString());  account.setFullname(objects[3].toString());  account.setDepartment(**new** Department(objects[4].toString()));  account.setPosition(**new** Position(PositionName.valueOf(objects[5].toString())));  **return** account;  } **finally** {  **if** (session != **null**) {  session.close();  }  }  }  // Giới hạn số lượng cột cần lấy sử dụng mệnh đề SELECT.  **public** String get\_FullName(**short** id) {  Session session = **null**;  **try** {  // get session  session = hibernateUtils.openSession();  // create Native query  // Câu lênh sql như câu lệnh viết trong Workbench  NativeQuery query = session  .createNativeQuery("SELECT a.FullName FROM account a \r\n" + "WHERE a.AccountID = ?;");  query.setParameter(1, id);  Object object = query.getSingleResult();  **return** object.toString();  } **finally** {  **if** (session != **null**) {  session.close();  }  }  }  // Giới hạn điều kiện trả về với WHERE: Lấy ra tất cả các Account được tạo trong  // tháng hiện tại.  **public** List<Account> get\_ByCurrentMonth() {  Session session = **null**;  **try** {  // get session  session = hibernateUtils.openSession();  // create Native query  NativeQuery query = session.createNativeQuery(  "SELECT a.AccountID, a.Email, a.Username, a.FullName, d.DepartmentName, p.PositionName FROM account a \r\n"  + " INNER JOIN department d ON a.DepartmentID = d.DepartmentID\r\n"  + " INNER JOIN position p ON a.PositionID = p.PositionID WHERE month(a.CreateDate) = ? AND year(a.CreateDate) = ?;");  query.setParameter(1, LocalDate.now().getMonthValue());  query.setParameter(2, LocalDate.now().getYear());  List<Account> accountList = **new** ArrayList<Account>();  List<Object[]> objectList = query.getResultList();  **for** (Object[] objects : objectList) {  Account account = **new** Account();  account.setId(Short.parseShort(objects[0].toString()));  account.setEmail(objects[1].toString());  account.setUsername(objects[2].toString());  account.setFullname(objects[3].toString());  account.setDepartment(**new** Department(objects[4].toString()));  account.setPosition(**new** Position(PositionName.valueOf(objects[5].toString())));  accountList.add(account);  }  **return** accountList;  } **finally** {  **if** (session != **null**) {  session.close();  }  }  }  // Sắp xếp các kết quả theo bất kỳ thuộc tính nào trên các đối tượng trong tập kết quả tăng dần (ASC) hoặc giảm dần (DESC).  // Lấy danh sách Account được tạo trong tháng hiện tại và sắp xếp theo ngảy tạo giảm dần hoặc tăng dần.  **public** List<Account> get\_ByCurrentMonthOderBy() **throws** ParseException {  Session session = **null**;  **try** {  // get session  session = hibernateUtils.openSession();  // create Native query  NativeQuery query = session.createNativeQuery(  "SELECT a.AccountID, a.Email, a.Username, a.FullName, d.DepartmentName, p.PositionName, a.CreateDate FROM account a \r\n"  + " INNER JOIN department d ON a.DepartmentID = d.DepartmentID\r\n"  + " INNER JOIN position p ON a.PositionID = p.PositionID WHERE month(a.CreateDate) = ? AND year(a.CreateDate) = ?\r\n"  + " ORDER BY a.CreateDate DESC;");  query.setParameter(1, LocalDate.now().getMonthValue());  query.setParameter(2, LocalDate.now().getYear());  List<Account> accountList = **new** ArrayList<Account>();  List<Object[]> objectList = query.getResultList();  **for** (Object[] objects : objectList) {  Account account = **new** Account();  account.setId(Short.parseShort(objects[0].toString()));  account.setEmail(objects[1].toString());  account.setUsername(objects[2].toString());  account.setFullname(objects[3].toString());  account.setDepartment(**new** Department(objects[4].toString()));  account.setPosition(**new** Position(PositionName.valueOf(objects[5].toString())));  String dateString = objects[6].toString();  Date date1 = **new** SimpleDateFormat("yyyy-MM-dd").parse(dateString);  account.setCreateDate(date1);  accountList.add(account);  }  **return** accountList;  } **finally** {  **if** (session != **null**) {  session.close();  }  }  }  // Update theo trường bất kỳ.  **public** **void** updateEmailUsernameAccountByID(String newEmail, String newUsername, **short** id) {  Session session = **null**;  **try** {  // get session  session = hibernateUtils.openSession();  session.beginTransaction();  // create Native query  NativeQuery query = session.createNativeQuery(  " UPDATE `account` SET `Email` = ?, `Username` = ? WHERE (`AccountID` =?);\r\n" + "");  query.setParameter(1, newEmail);  query.setParameter(2, newUsername);  query.setParameter(3, id);  query.executeUpdate();  session.getTransaction().commit();  } **finally** {  **if** (session != **null**) {  session.close();  }  }  }  // Delete theo trường bất kỳ  **public** **void** deleteAccount(String email) {  Session session = **null**;  **try** {  session = hibernateUtils.openSession();  session.beginTransaction();  // create Native query  NativeQuery query = session.createNativeQuery("DELETE FROM `account` WHERE (`Email` = ?);");  query.setParameter(1, email);  query.executeUpdate();  session.getTransaction().commit();  } **finally** {  **if** (session != **null**) {  session.close();  }  }  }  // Phân trang bằng hibernate  **public** List<Account> get\_AccountByPaging() {  Session session = **null**;  **try** {  // get session  session = hibernateUtils.openSession();  NativeQuery query = session.createNativeQuery(  "SELECT a.AccountID, a.Email, a.Username, a.FullName, d.DepartmentName, p.PositionName FROM account a \r\n"  + "INNER JOIN department d ON a.DepartmentID = d.DepartmentID\r\n"  + "INNER JOIN position p ON a.PositionID = p.PositionID\r\n" + "ORDER BY a.AccountID\r\n"  + "LIMIT 3\r\n" + "OFFSET 5;");  // Phần query này sẽ tạo ra 1 list các array object, sử dụng vòng lặp foreach để truy vấn tới các phần tử trong array, lấy các giá trị trong DB theo Index của array bắt đầu từ 0.  List<Account> accountList = **new** ArrayList<Account>();  List<Object[]> accounts = query.getResultList();  **for** (Object[] objects : accounts) {  Account account = **new** Account();  account.setId(Short.parseShort(objects[0].toString()));  account.setEmail(objects[1].toString());  account.setUsername(objects[2].toString());  account.setFullname(objects[3].toString());  // Chý ý cần tạo mới hàm tạo 1 tham số cho Department  account.setDepartment(**new** Department(objects[4].toString()));  account.setPosition(**new** Position(PositionName.valueOf(objects[5].toString())));  accountList.add(account);  }  **return** accountList;  } **finally** {  **if** (session != **null**) {  session.close();  }  }  }  } |

****

# **Tạo Class DemoHQL trong Frontend:**

|  |
| --- |
| **package** com.vti.frontend;  **import** java.text.ParseException;  **import** java.time.LocalDate;  **import** java.util.List;  **import** org.hibernate.Session;  **import** org.hibernate.query.Query;  **import** com.vti.entity.Account;  **import** com.vti.repository.AccountRepository;  **import** com.vti.utils.ScannerUltis;  **public** **class** DemoHQL {  **public** **static** **void** main(String[] args) **throws** ParseException {  AccountRepository accountRepository = **new** AccountRepository();  // List<Account> list = accountRepository.get\_FROM();  // for (Account account : list) {  // System.out.println("ID: " + account.getId() + " Email:" + account.getEmail() + " Username: "  // + account.getUsername() + " FullName: " + account.getFullname() + " Department: "  // + account.getDepartment().getName() + " Possition: " + account.getPosition().getName());  // }  // System.out.println("Nhập vào ID của Account cần tìm kiếm: ");  // int id = ScannerUltis.inputIntPositive();  // Account account = accountRepository.get\_ByID((short) id);  // System.out.println("ID: " + account.getId() + " Email:" + account.getEmail() + " Username: "  // + account.getUsername() + " FullName: " + account.getFullname() + " Department: "  // + account.getDepartment().getName() + " Possition: " + account.getPosition().getName());  // System.out.println("Nhập vào ID của Account cần tìm kiếm: ");  // int id = ScannerUltis.inputIntPositive();  // String fullname = accountRepository.get\_FullName((short) id);  // System.out.println("Fullname là: "+ fullname);  // System.out.println("Thông tin User được tạo trong tháng hiện tại.");  // List<Account> list = accountRepository.get\_ByCurrentMonth();  // for (Account account : list) {  // System.out.println("ID: " + account.getId() + " Email:" + account.getEmail() + " Username: "  // + account.getUsername() + " FullName: " + account.getFullname() + " Department: "  // + account.getDepartment().getName() + " Possition: " + account.getPosition().getName());  // }  // System.out.println("Thông tin User được tạo trong tháng hiện tại.");  // List<Account> list = accountRepository.get\_ByCurrentMonthOderBy();  // for (Account account : list) {  // System.out.println("ID: " + account.getId() + " Email:" + account.getEmail() + " Username: "  // + account.getUsername() + " FullName: " + account.getFullname() + " Department: "  // + account.getDepartment().getName() + " Possition: " + account.getPosition().getName()  // + " CreateDate: " + account.getCreateDate());  // }  // System.out.println("Nhập vào ID cần Update: ");  // int id = ScannerUltis.inputIntPositive();  // System.out.println("Nhập vào NewEmail: ");  // String email = ScannerUltis.inputEmail();  // System.out.println("Nhập vào NewUsername: ");  // String username = ScannerUltis.inputString();  //  // accountRepository.updateEmailUsernameAccountByID(email, username, (short) id);  // System.out.println("Nhập vào email cần xóa: ");  // String email = ScannerUltis.inputEmail();  // accountRepository.deleteAccount(email);  // System.out.println("Demo Phân trang với Hibernate: ");  // List<Account> list = accountRepository.get\_AccountByPaging();  // for (Account account : list) {  // System.out.println("ID: " + account.getId() + " Email:" + account.getEmail() + " Username: "  // + account.getUsername() + " FullName: " + account.getFullname() + " Department: "  // + account.getDepartment().getName() + " Possition: " + account.getPosition().getName());  // }  }  } |

****