МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Кафедра інформаційних систем та мереж

Звіт

до лабораторної роботи №3

з дисципліни «Екстримальне програмування»

на тему «Створення об'єктів доступу до даних»

Виконала

студентка групи КН-311

Мельничук М.П.

Прийняв

Щербак С.С.

Львів-2020

Мета: створення об'єктів доступу до даних за допомогою JPA Criteria API.

Теоретичні відомості

Об'єкт доступу до даних (англ. data access object) (DAO) - об'єкт що надає абстрактний інтерфейс до деяких видів баз даних чи механізмів персистентності реалізуючи певні операції без розкриття деталей бази даних. Він надає відображення від програмних викликів до рівня персистентності. Така ізоляція розділює запити до даних в термінах предметної області та їх реалізацію засобами СКБД.

Цей паттерн проєктування можна застосовувати до більшості мов програмування, видів програмного забезпечення з потребою персистентності та більшості типів баз даних, але він традиційно асоціюється з застосунками Java EE та реляційними БД доступ до яких здійснюють через JDBC API що пов'язано з походженням паттерна із збірки найкращих практик Sun Microsystems. ("Core J2EE Patterns") для цієї платформи.

Перевагою використання об'єкту доступу до даних є досить просте розділення двох частин програми, які мають бути розділені: бізнес логіки, та логіки персистентності. В такому разі зміна бізнес логіки зовсім не буде впливати на роботу механізмів персистентності, а заміна схеми даних чи способу їх зберігання - не впливати на роботу бізнес логіки, якщо інтерфейс реалізований правильно.

Одним із способів маніпулювання об'єктів і перекладу його в таблицю базових даних (BD) є Criteria API, яка створює запити з критеріями програмного методу. Criteria API:

- значення значення агрегатних функцій типу sum (), min (), max () і т.д .; випущені дані тільки з вибраних колонок із використанням ProjectionList;
- з'єднання декількох таблиць для приєднання запросів із використанням методів createAlias(), setFetchMode() та setProjection();
- Вибір результатів відповідно до умови з використанням методу add(), які мають обмеження (обмеження);
- визначення порядку вибору сортування додає к критеричній методі addOrder().

Для створення об'єкта org.hibernate.Criteria використовує метод createCriteria, інтерфейс сесії Session, яка використовується в якості необхідної передачі сутності. Наступний код формує об'єкт Criteria для виконання транзакцій з суттєвістю розробника.

Хід роботи

Об’єкти доступу до даних мають бути створенні для кожного класу моделі даних для того щоб система могла маніпулювати з даними.

Клас моделі: User.java -> Клас об’єкт доступу до даних: UserRepository.java

Було реалізовано такі методи:

- getUsers() – повертає всіх користувачів;
- getUser(Integer userId) – повертаєкористувача за ідентифікатором;
- getLoginedUser(String email, String password) – повернути залогіненого користувача;
- saveUser(User user) – зберегти/створити користувача;
- deleteUser(String userId) – видалити користувача за ідентифікатором;
- updateUser(Integer id, User newUser) – оновити інформацію про користувача;

```
public class UserRepository {

  private static Session session = HibernateUtil.getSession();
  private static CriteriaBuilder cb = session.getCriteriaBuilder();

  public static List<User> getUsers() {
    CriteriaQuery<User> cr = cb.createQuery(User.class);
    Root<User> root = cr.from(User.class);
    cr.select(root);
    Query<User> query = session.createQuery(cr);
    return query.getResultList();
  }

  public static User getUser(Integer userId) {
    CriteriaQuery<User> cr = cb.createQuery(User.class);
    Root<User> root = cr.from(User.class);
    cr.where(cb.equal(root.get("id"), userId));
    cr.select(root);
    Query<User> query = session.createQuery(cr);
    return query.getResultList().get(0);
  }

  public static List<User> getLoginedUser(String email, String password) {
    CriteriaQuery<User> cr = cb.createQuery(User.class);
    Root<User> root = cr.from(User.class);
    cr.where(cb.equal(root.get("email"), email));
    cr.where(cb.equal(root.get("password"), password));
    cr.select(root);
    Query<User> query = session.createQuery(cr);
    return query.getResultList();
  }

  public static void saveUser(User user) {
    Transaction transaction = session.beginTransaction();
    session.save(user);
    transaction.commit();
  }
```

```
    public static void deleteUser(String userId) {
        CriteriaDelete<User> criteriaDelete = cb.createCriteriaDelete(User.class);
        Root<User> root = criteriaDelete.from(User.class);
        criteriaDelete.where(cb.equal(root.get("id"), userId));
        Transaction transaction = session.beginTransaction();
        session.createQuery(criteriaDelete).executeUpdate();
        transaction.commit();
    }

    public static void updateUser(Integer id, User newUser) {
        CriteriaUpdate<User> criteriaUpdate = cb.createCriteriaUpdate(User.class);
        Root<User> root = criteriaUpdate.from(User.class);
        criteriaUpdate.set("name", newUser.getName());
        criteriaUpdate.set("surname", newUser.getSurname());
        criteriaUpdate.set("email", newUser.getEmail());
        criteriaUpdate.set("password", newUser.getPassword());
        criteriaUpdate.where(cb.equal(root.get("id"), id));
        Transaction transaction = session.beginTransaction();
        session.createQuery(criteriaUpdate).executeUpdate();
        transaction.commit();
    }
}
```

Клас моделі: Good.java -> Клас об’єкт доступу до даних: GoodRepository.java

Було реалізовано такі методи:

- getGoods() – повертає всі існуючі товари;
- getGoodsByPurchaseId(Integer purchaseId) – повертає всі товари які складають покупку;
- getGoodsShop(Integer shopId) – повернути товари які існують в заданому магазині;
- saveGood(Good good) – зберегти/створити товар;
- deleteGood(String goodId) – видалити товар;
- updateGood(Integer id, Good newGood) – оновити інформацію за товар;

```
public class GoodRepository {

    private static Session session = HibernateUtil.getSession();
    private static CriteriaBuilder cb = session.getCriteriaBuilder();

    public static List<Good> getGoods() {
        CriteriaQuery<Good> cr = cb.createQuery(Good.class);
        Root<Good> root = cr.from(Good.class);
        cr.select(root);
        Query<Good> query = session.createQuery(cr);
        return query.getResultList();
    }

    public static List<Good> getGoodsByPurchaseId(Integer purchaseId) {
        CriteriaQuery<Purchase> cr = cb.createQuery(Purchase.class);
        Root<Purchase> root = cr.from(Purchase.class);
        cr.where(cb.equal(root.get("purchaseId"), purchaseId));
        cr.select(root);
        Query<Purchase> query = session.createQuery(cr);
        List<Purchase> purchases = query.getResultList();
```

```
        List<Good> goods = new ArrayList<Good>();
        for(Purchase purchase : purchases) {
            goods.add(purchase.getGood());
        }
        return goods;
    }

    public static List<Good> getGoodsShop(Integer shopId){
        Shop shop = ShopRepository.getShop(shopId);
        List<Category> categories = shop.getCategories();
        List<Good> goodsToReturn = new ArrayList<>();
        for(Category category : categories) {
            goodsToReturn.addAll(category.getGoods());
        }
        return goodsToReturn;
    }

    public static Good getGood(Integer goodId) {
        CriteriaQuery<Good> cr = cb.createQuery(Good.class);
        Root<Good> root = cr.from(Good.class);
        cr.where(cb.equal(root.get("id"), goodId));
        cr.select(root);
        Query<Good> query = session.createQuery(cr);
        return query.getResultList().get(0);
    }

    public static void saveGood(Good good) {
        Transaction transaction = session.beginTransaction();
        session.save(good);
        transaction.commit();
    }

    public static void deleteGood(String goodId) {
        CriteriaDelete<Good> criteriaDelete = cb.createCriteriaDelete(Good.class);
        Root<Good> root = criteriaDelete.from(Good.class);
        criteriaDelete.where(cb.equal(root.get("id"), goodId));
        Transaction transaction = session.beginTransaction();
        session.createQuery(criteriaDelete).executeUpdate();
        transaction.commit();
    }

    public static void updateGood(Integer id, Good newGood) {
        CriteriaUpdate<Good> criteriaUpdate = cb.createCriteriaUpdate(Good.class);
        Root<Good> root = criteriaUpdate.from(Good.class);
        criteriaUpdate.set("name", newGood.getName());
        criteriaUpdate.set("category", newGood.getCategory());
        criteriaUpdate.where(cb.equal(root.get("id"), id));
        Transaction transaction = session.beginTransaction();
        session.createQuery(criteriaUpdate).executeUpdate();
        transaction.commit();
    }
}
```

Клас моделі: Purchase.java -> Клас об’єкт доступу до даних: PurchaseRepository.java

Було реалізовано такі методи:

- getPurchases() – повертає всі покупки;

- getPurchaseShop(Integer shopId) – повернути список покупок за ідентифікатором магазину;
- getPurchase(String purchaseId) – повернути покупку за ідентифікатором покупки;
- savePurchase(Purchase purchase) – зберегти/створити покупку;
- deletePurchase(String purchaseId) – видалити покупку;
- updatePurchase(Integer id, Purchase newPurchase) – оновити дані покупки;

```
public class PurchaseRepository {

   private static Session session = HibernateUtil.getSession();
   private static CriteriaBuilder cb = session.getCriteriaBuilder();

   public static List<Purchase> getPurchases() {
      CriteriaQuery<Purchase> cr = cb.createQuery(Purchase.class);
      Root<Purchase> root = cr.from(Purchase.class);
      cr.select(root);
      Query<Purchase> query = session.createQuery(cr);
      return query.getResultList();
   }

   public static List<Purchase> getPurchaseShop(Integer shopId) {
      CriteriaQuery<Purchase> cr = cb.createQuery(Purchase.class);
      Root<Purchase> root = cr.from(Purchase.class);
      cr.where(cb.equal(root.join("shop").get("id"), shopId));
      cr.groupBy(root.get("purchaseId"));
      cr.select(root);
      Query<Purchase> query = session.createQuery(cr);
      return query.getResultList();
   }
   public static Purchase getPurchase(String purchaseId) {
      CriteriaQuery<Purchase> cr = cb.createQuery(Purchase.class);
      Root<Purchase> root = cr.from(Purchase.class);
      cr.where(cb.equal(root.get("id"), purchaseId));
      cr.select(root);
      Query<Purchase> query = session.createQuery(cr);
      return query.getResultList().get(0);
   }

   public static void savePurchase(Purchase purchase) {
      Transaction transaction = session.beginTransaction();
      session.persist(purchase);
      transaction.commit();
   }

   public static void deletePurchase(String purchaseId) {
      CriteriaDelete<Purchase> criteriaDelete = cb.createCriteriaDelete(Purchase.class);
      Root<Purchase> root = criteriaDelete.from(Purchase.class);
      criteriaDelete.where(cb.equal(root.get("id"), purchaseId));
      Transaction transaction = session.beginTransaction();
      session.createQuery(criteriaDelete).executeUpdate();
      transaction.commit();
   }
   public static void updatePurchase(Integer id, Purchase newPurchase) {
      CriteriaUpdate<Purchase> criteriaUpdate = cb.createCriteriaUpdate(Purchase.class);
```

```
        Root<Purchase> root = criteriaUpdate.from(Purchase.class);
        criteriaUpdate.set("customer", newPurchase.getCustomer());
        criteriaUpdate.set("shop", newPurchase.getShop());
        criteriaUpdate.set("location", newPurchase.getLocation());
        criteriaUpdate.where(cb.equal(root.get("id"), id));
        Transaction transaction = session.beginTransaction();
        session.createQuery(criteriaUpdate).executeUpdate();
        transaction.commit();
    }
}
```

Також було розроблено такі об'єкти доступу до даних, ресурсний код яких можна подивитися у репозиторії:

- Клас моделі: CategoryLocation.java -> Клас об'єкт доступу до даних: CategoryLocationRepository.java(https://github.com/marynamelnichuk/EP--PROJECT/blob/master/src/main/java/com/kursova/repository/CategoryLocationRepository.java);
- Клас моделі: Customer.java -> Клас об'єкт доступу до даних: CustomerRepository.java(https://github.com/marynamelnichuk/EP--PROJECT/blob/master/src/main/java/com/kursova/repository/CustomerRepository.java);
- Клас моделі: Category.java -> Клас об'єкт доступу до даних: CategoryRepository.java(https://github.com/marynamelnichuk/EP--PROJECT/blob/master/src/main/java/com/kursova/repository/CategoryRepository.java);
- Клас моделі: Location.java -> Клас об'єкт доступу до даних: LocationRepository.java(https://github.com/marynamelnichuk/EP--PROJECT/blob/master/src/main/java/com/kursova/repository/LocationRepository.java);
- Клас моделі: Shop.java -> Клас об'єкт доступу до даних: ShopRepository.java(https://github.com/marynamelnichuk/EP--PROJECT/blob/master/src/main/java/com/kursova/repository/ShopRepository.java);

Можемо побачити існуючі класи і методи в в пакеті repository:

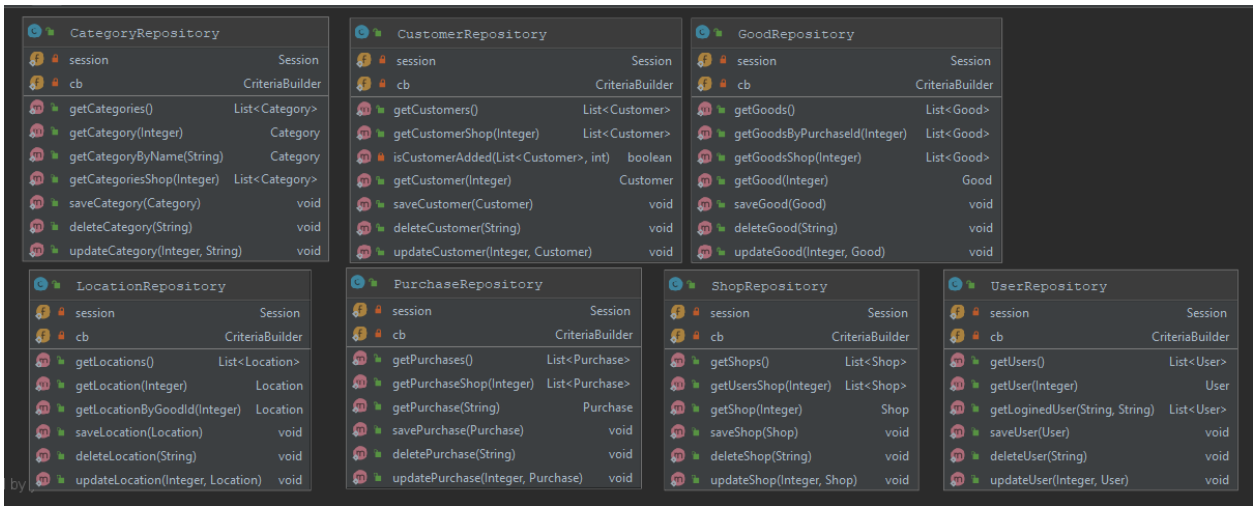

Рис.3.1 Класи-об'єкти доступу до даних в пакеті repository

Висновок: у результаті виконання лабораторної роботи було розроблено об'єкти доступу до даних за допомогою використання JPA Criteria API.