Karen Giselle Valdez Muñoz

JPA

Overview

This document provides an explanation of a Java Persistence API (JPA) implementation that connects to a MySQL database to manage user data. The JPA code includes various CRUD operations for a Users table with fields such as id, lastName, firstName, email, and cellphone.

Table Definition

The Users table in the MySQL database is structured as follows:

- id (Primary Key, Integer)
- lastName (String)
- firstName (String)
- email (String)
- cellphone (String)

Configuration Settings

These settings configure the connection to a MySQL database with specified credentials, optionally control the display of the Spring Boot startup banner, and adjust the logging level to show only warnings and errors.

```
spring.datasource.url=jdbc:mysql://localhost:3306/pwdatabase
spring.datasource.username=WebPage
spring.datasource.password=WebPage

# Turn off the Spring Boot banner
# spring.main.banner-mode=off

# Reduce logging level. Set logging level to warn
logging.level.root=warn
```

MySQL table Users

id	last_name	first_name	email	cellphone
1	Doe	John	john.doe@foo.com	8181759370
2	Public	Mary	mary.public@foo.com	8181759375
3	Queue	Susan	susan.queue@foo.com	8181759374
4	Williams	David	david.williams@foo.com	8183759370
5	Johnson	Lisa	lisa.johnson@foo.com	8181759270
6	Smith	Paul	paul.smith@foo.com	8181759371

Karen Giselle Valdez Muñoz

Users

The Users class represents a user entity in the application. It maps to a database table named users and contains fields for storing user information. This class is used to model user data and interact with the database through JPA.

```java
package com.luv2code.cruddemo.entity;

import jakarta.persistence.*;

@Entity
@Table(name="users")
public class Users {

    // define fields
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private int id;

    @Column(name="first_name")
    private String firstName;

    @Column(name="last_name")
    private String lastName;

    @Column(name="email")
    private String email;

    @Column(name="cellphone")
    private String cellphone;

    // define constructors
    public Users() {

    }

    public Users(String firstName, String lastName, String email, String cellphone) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.email = email;
        this.cellphone = cellphone;
    }

    // define getters/setters

    public String getCellphone() {
        return cellphone;
    }

    public void setCellphone(String cellphone) {
        this.cellphone = cellphone;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
```

```
    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }


    // define toString() method

    @Override
    public String toString() {
        return "Student{" +
                "id=" + id +
                ", firstName='" + firstName + '\'' +
                ", lastName='" + lastName + '\'' +
                ", email='" + email + '\'' + ", cellphone='" + cellphone +
                '}';
    }
}
```

UsersDAO

The UsersDAO interface defines a set of methods for managing Users entities in a database. It acts as a contract for any class that implements it, outlining the operations that can be performed on user data.

```
package com.luv2code.cruddemo.dao;

import com.luv2code.cruddemo.entity.Users;

public interface UsersDAO {

    void save(Users theUser);

    Users findById(Integer id);

    List<Users> findAll();

    List<Users> findByLastName(String theLastName);

    List<Users> findByCellphone(String cellphone);

    void update(Users theUser);

    void delete(Integer id);

    int deleteAll();

}
```

Karen Giselle Valdez Muñoz

UsersDAOImpl

The UsersDAOImpl class is responsible for managing user data in a database. It implements the UsersDAO interface and uses JPA (Java Persistence API) to perform various operations on user records.

```java
@Repository
public class UsersDAOImpl implements UsersDAO {

    // define field for entity manager
    private EntityManager entityManager;

    // inject entity manager using constructor injection
    @Autowired
    public UsersDAOImpl(EntityManager entityManager) {
        this.entityManager = entityManager;
    }

    // implement save method
    @Override
    @Transactional
    public void save(Users theUser) {
        System.out.println("Save User");
        entityManager.persist(theUser); //<==JPA
    }

    @Override
    public Users findById(Integer id) {
        return entityManager.find(Users.class, id);//<==JPA
    }


    @Override
    public List<Users> findAll() {
        // create query
        TypedQuery<Users> theQuery = entityManager.createQuery("FROM Users", Users.class);

        // return query results
        return theQuery.getResultList();//<==JPA
    }

    @Override
    public List<Users> findByLastName(String theLastName) {
        // create query
        TypedQuery<Users> theQuery = entityManager.createQuery(
                                        "FROM Users WHERE lastName=:theData", Users.class);

        // set query parameters
        theQuery.setParameter("theData", theLastName);

        // return query results
        return theQuery.getResultList();
    }

    @Override
    public List<Users> findByCellphone(String cellphone) {
        // create query
        TypedQuery<Users> theQuery = entityManager.createQuery(
                                        "FROM Users WHERE cellphone=:thePhone", Users.class);

        // set query parameters
        theQuery.setParameter("thePhone", cellphone);

        // return query results
        return theQuery.getResultList();
    }
```

```java
    @Override
    @Transactional
    public void update(Users theStudent) {
        entityManager.merge(theStudent); //<==JPA
    }

    @Override
    @Transactional
    public void delete(Integer id) {

        // retrieve the student
        Users theStudent = entityManager.find(Users.class, id);

        // delete the student
        entityManager.remove(theStudent);//<==JPA
    }

    @Override
    @Transactional
    public int deleteAll() {

        int numRowsDeleted = entityManager.createQuery("DELETE FROM Users").executeUpdate(); //<==JPA

        return numRowsDeleted;
    }
}
```

CruddemmoApplication

The CruddemoApplication class is a Spring Boot application that demonstrates various CRUD (Create, Read, Update, Delete) operations on Users entities using a UsersDAO data access object.

- save(Users theUser): Adds a new user to the database.
- findById(Integer id): Retrieves a user by their ID.
- findAll(): Retrieves all users from the database.
- findByLastName(String theLastName): Retrieves users with a specific last name.
- findByCellphone(String cellphone): Retrieves users with a specific cellphone number.
- update(Users theUser): Updates an existing user's information.
- delete(Integer id): Deletes a user by their ID.
- deleteAll(): Deletes all users from the database.

```java

    private void queryForUserCellphone(UsersDAO usersDAO) {
        // get a list of students
        List<Users> theUsers = usersDAO.findByCellphone("8181759375");

        // display list of students
        for (Users tempStudent : theUsers) {
            System.out.println(tempStudent);
        }

    }

    private void deleteAllUser(UsersDAO usersDAO) {

        System.out.println("Deleting all students");
        int numRowsDeleted = usersDAO.deleteAll();
        System.out.println("Deleted row count: " + numRowsDeleted);
    }

    private void deleteUser(UsersDAO usersDAO) {

        int studentId = 2;
        System.out.println("Deleting student id: " + studentId);
        usersDAO.delete(studentId);
    }

    private void updateUser(UsersDAO usersDAO) {

        // retrieve student based on the id: primary key
        int studentId = 9;
        System.out.println("Getting student with id: " + studentId);
        Users myStudent = usersDAO.findById(studentId);

```

```
        // change first name to "John"
        System.out.println("Updating student ...");
        myStudent.setFirstName("John");

        // update the student
        usersDAO.update(myStudent);

        // display the updated student
        System.out.println("Updated student: " + myStudent);
    }

    private void queryForUserByLastName(UsersDAO usersDAO) {

        // get a list of students
        List<Users> theUsers = usersDAO.findByLastName("Doe");

        // display list of students
        for (Users tempStudent : theUsers) {
            System.out.println(tempStudent);
        }
    }

    private void queryForUser(UsersDAO usersDAO) {

        // get a list of students
        List<Users> theUsers = usersDAO.findAll();

        // display list of students
        for (Users newUser : theUsers) {
            System.out.println(newUser);
        }
    }
     private void readUser(UsersDAO usersDAO) {

         // create  a user object
         System.out.println("Creating new user object ...");
         Users newUser = new Users("Karen", "Valdez", "daffy@luv2code.com", "8181764890");

         // save the user
         System.out.println("Saving the student ...");
         usersDAO.save(newUser);

         // display id of the saved user
         int theId = newUser.getId();
         System.out.println("Saved student. Generated id: " + theId);

         // retrieve user based on the id: primary key
         System.out.println("Retrieving student with id: " + theId);
         Users newUser1 = usersDAO.findById(theId);

         // display user
         System.out.println("Found the user: " + newUser1);
     }
```

```
    private void createMultipleUser(UsersDAO usersDAO) {

        // create multiple users
        System.out.println("Creating 3 student objects ...");
        Users newUser1 = new Users("Pablo", "Muñoz", "correo@luv2code.com", "8181744890");
        Users newUser2 = new Users("Luis", "Escobedo", "email@luv2code.com", "8483764890");
        Users newUser3 = new Users("Ana", "Peña", "contra@luv2code.com", "8281764890");

        // save the users objects
        System.out.println("Saving the users ...");
        usersDAO.save(newUser1);
        usersDAO.save(newUser2);
        usersDAO.save(newUser3);
    }

    private void createUser(UsersDAO usersDAO) {

        // create the user object
        System.out.println("Creating new student object ...");
        Users newUser = new Users("Juan", "Perez", "pedro@luv2code.com", "123456789");

        // save the user object
        System.out.println("Saving the user ...");
        usersDAO.save(newUser);

        // display id of the saved user
        System.out.println("Saved user. Generated id: " + newUser.getId());
    }
}
```

Karen Giselle Valdez Muñoz

Output:

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.3.2)

Creating new student object ...
Saving the user ...
Save User
Saved user. Generated id: 7
Creating 3 student objects ...
Saving the users ...
Save User
Save User
Save User
Creating new user object ...
Saving the student ...
Save User
Saved student. Generated id: 11
Retrieving student with id: 11
Found the user: Student{id=11, firstName='Karen', lastName='Valdez', email='daffy@luv2code.com', cellphone='8181764890}
Student{id=1, firstName='John', lastName='Doe', email='john.doe@foo.com', cellphone='8181759370}
Student{id=2, firstName='Mary', lastName='Public', email='mary.public@foo.com', cellphone='8181759375}
Student{id=3, firstName='Susan', lastName='Queue', email='susan.queue@foo.com', cellphone='8181759374}
Student{id=4, firstName='David', lastName='Williams', email='david.williams@foo.com', cellphone='8183759370}
Student{id=5, firstName='Lisa', lastName='Johnson', email='lisa.johnson@foo.com', cellphone='8181759270}
Student{id=6, firstName='Paul', lastName='Smith', email='paul.smith@foo.com', cellphone='8181759371}
Student{id=7, firstName='Juan', lastName='Perez', email='pedro@luv2code.com', cellphone='123456789}
Student{id=8, firstName='Pablo', lastName='Muñoz', email='correo@luv2code.com', cellphone='8181744890}
Student{id=9, firstName='Luis', lastName='Escobedo', email='email@luv2code.com', cellphone='8483764890}
Student{id=10, firstName='Ana', lastName='Peña', email='contra@luv2code.com', cellphone='8281764890}

Student{id=11, firstName='Karen', lastName='Valdez', email='daffy@luv2code.com', cellphone='8181764890}
Student{id=1, firstName='John', lastName='Doe', email='john.doe@foo.com', cellphone='8181759370}
Student{id=2, firstName='Mary', lastName='Public', email='mary.public@foo.com', cellphone='8181759375}
Getting student with id: 9
Updating student ...
Updated student: Student{id=9, firstName='John', lastName='Escobedo', email='email@luv2code.com', cellphone='8483764890}
Deleting student id: 2
Deleting all students
Deleted row count: 10
```

The output from the CruddemoApplication class demonstrates the successful execution of various CRUD operations on the Users entity. Each operation is performed in sequence, illustrating how the application interacts with the database.

- Creation Operations:
    - Single User Creation: A new Users object is created and saved, and its generated ID is displayed.
    - Multiple Users Creation: Several Users objects are created and saved, showcasing the ability to handle batch operations.

- Read Operations:
    - Single User Read: After saving a user, the application retrieves and displays the same user based on its ID, confirming that the data was correctly saved.
    - Find All Users: All Users records are retrieved and displayed, providing a complete view of the current dataset.
    - Query by Last Name: Users are filtered and displayed based on their last name, demonstrating the application's capability to perform targeted searches.

    - Query by Cellphone: The application retrieves users based on their cellphone number, highlighting its ability to handle queries with specific criteria.

- Update Operation:
    - User Update: An existing user's first name is updated, and the changes are saved and displayed, showing how the application can modify existing records.

- Delete Operations:
    - Single User Deletion: A specific user is deleted by ID, and the operation's effect is confirmed.
    - Delete All Users: All user records are removed from the database, demonstrating the application’s ability to clear the entire dataset.

This sequence of operations provides a comprehensive demonstration of how the CruddemoApplication manages user data using Spring Boot and JPA. It validates the functionality of creating, reading, updating, and deleting records, and effectively showcases the integration between the application and the MySQL database.

Karen Giselle Valdez Muñoz