10.Quiz part 2

QuestionRepository Explanation:

- **Interface**: Extends JpaRepository<Question, Integer>, making it a repository for performing database operations on Question entities.
- **Custom Methods**:
  - findByCategory(String category): Retrieves a list of questions filtered by category.
  - findByQuestionTitle(String questionTitle): Fetches questions based on the question title.
  - findRandomQuestionsByCategory(String category, int numQ): Uses a **native query** to retrieve a specified number (numQ) of random questions for a given category. The query orders questions by RAND() and limits the result set.

Example:

```
@Repository
public interface QuestionRepository extends JpaRepository<Question, Integer> {

   public List<Question> findByCategory(String category);

   public List<Question> findByQuestionTitle(String questionTitle);

   @Query(value = "SELECT * FROM question WHERE category = :category ORDER BY RAND() LIMIT :numQ",
nativeQuery = true)
   public List<Question> findRandomQuestionsByCategory(@Param("category") String category,
@Param("numQ") int numQ);
}
```

QuizRepository Explanation:

- **Interface**: Extends JpaRepository<Quiz, Integer>. This repository is responsible for basic CRUD operations related to the Quiz entity.;

Example:

```
@Repository
public interface QuizRepository extends JpaRepository<Quiz, Integer> {

}
```

QuizService Explanation:

- **Service Layer**: Handles the business logic for creating quizzes.
- **Dependency Injection**: Uses @Autowired to inject QuizRepository and QuestionRepository instances.
- **createQuiz Method**:
  - Fetches random questions from the specified category using findRandomQuestionsByCategory().
  - Checks if enough questions were retrieved to create a quiz.
  - Creates a new Quiz object, sets the title, and associates the fetched questions with the quiz.
  - Saves the quiz to the database via quizRepository.
  - Returns appropriate responses for success, insufficient questions, or any errors.

Example:

```
@Service
public class QuizService {

    @Autowired
    QuizRepository quizRepository;

    @Autowired
    QuestionRepository questionRepository;

    public ResponseEntity<String> createQuiz(String category, int numQ, String title) {
        try {
            List<Question> questions =
                    questionRepository.findRandomQuestionsByCategory(category, numQ);

            if (questions.isEmpty()) {
                return new ResponseEntity<>("Not enough questions available for the given
category", HttpStatus.BAD_REQUEST);
            }

            Quiz quiz = new Quiz();
            quiz.setTitle(title);
            quiz.setQuestions(questions);

            quizRepository.save(quiz);

            return new ResponseEntity<>("Quiz created successfully with title: " + title,
HttpStatus.CREATED);
        } catch (Exception e) {
            e.printStackTrace();
            return new ResponseEntity<>("Error occurred while creating the quiz",
HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
}
```

QuizController Explanation:

- **Controller Layer**: Exposes the REST API for managing quizzes.
- **Endpoint**:
  - POST /quiz/create: Takes query parameters (category, numQ, title) and calls the createQuiz() method from QuizService to create a new quiz.
  - Returns an appropriate HTTP response based on the success or failure of the quiz creation.

Example:

```
@RestController
@RequestMapping("quiz")
public class QuizController {

    @Autowired
    QuizService quizService;

    @PostMapping("create")
    public ResponseEntity<String> createQuiz(
            @RequestParam String category,
            @RequestParam int numQ,
            @RequestParam String title) {
        return quizService.createQuiz(category, numQ, title);
    }
}
```

Quiz Entity Explanation:

- **Entity Class**: Represents the Quiz entity in the database.
- **Annotations**:
  - @Entity: Marks this class as a JPA entity.
  - @Id, @GeneratedValue, @SequenceGenerator: Configures the primary key and uses a sequence generator for auto-generating IDs starting from 1.
  - @ManyToMany: Establishes a many-to-many relationship between Quiz and Question entities, where a quiz can have multiple questions, and a question can belong to multiple quizzes.

Example:

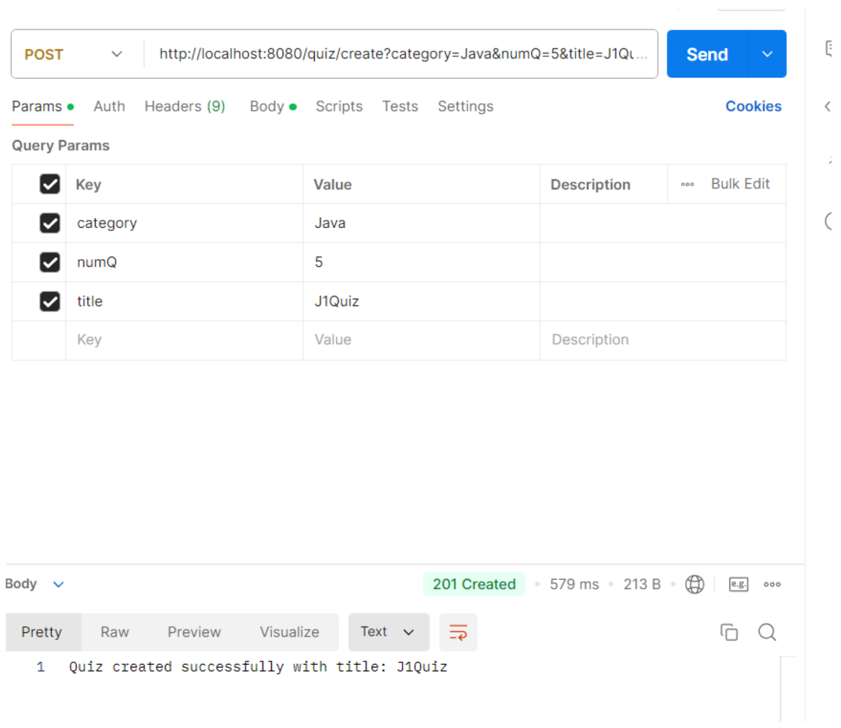
```
@Entity
@Data
@NoArgsConstructor
public class Quiz {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "quiz_seq")
    @SequenceGenerator(name = "quiz_seq", sequenceName = "quiz_sequence", allocationSize = 1,
initialValue = 1)
    private Integer id;

    private String title;

    @ManyToMany
    private List<Question> questions;
}
```

Postman:



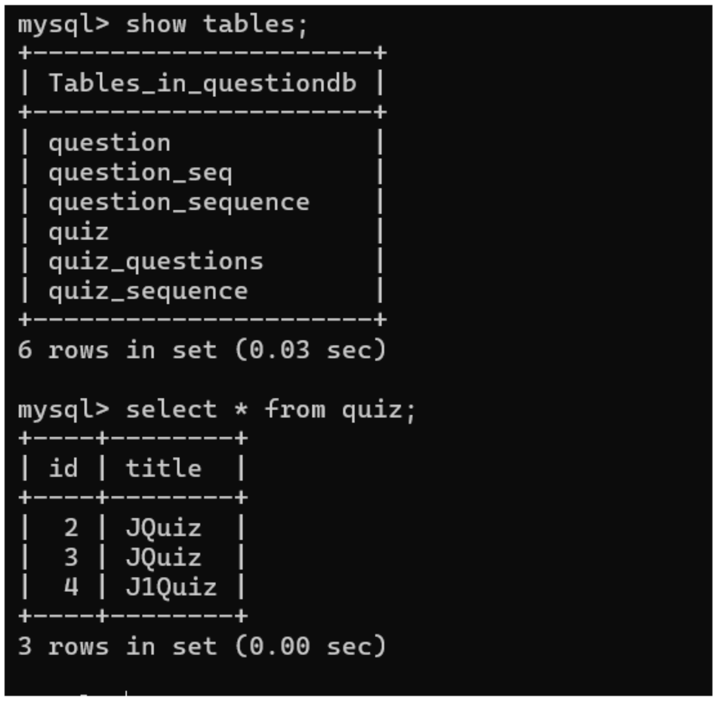
```
http://localhost:8080/quiz/create?category=Java&numQ=5&title=J1Qu...
```

Query Params

	Key	Value	Description
✓	category	Java	
✓	numQ	5	
✓	title	J1Quiz	
	Key	Value	Description

201 Created • 579 ms • 213 B

```
Quiz created successfully with title: J1Quiz
```



```
mysql> show tables;
+----------------------+
| Tables_in_questiondb |
+----------------------+
| question             |
| question_seq         |
| question_sequence    |
| quiz                 |
| quiz_questions       |
| quiz_sequence        |
+----------------------+
6 rows in set (0.03 sec)

mysql> select * from quiz;
+----+--------+
| id | title  |
+----+--------+
|  2 | JQuiz  |
|  3 | JQuiz  |
|  4 | J1Quiz |
+----+--------+
3 rows in set (0.00 sec)
```