The QuoteServices App Case Study: Zippy Microservice Structure & Functionality (Part 2)

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

- Understand the structure & functionality of the JPAQuoteRepository & how it creates a custom SQL query with the Jakarta Persistence API (JPA)

We also give an overview of the Spring Data API
Overview of the Spring Data API
Overview of the Spring Data API

- The Spring JPA Data API is a framework that simplifies development of data access layers in Spring-based apps.

Overview of the Spring Data API

- The Spring JPA Data API is a framework that simplifies development of data access layers in Spring-based apps.
- Developers write repository interfaces that define data access methods they need.

Overview of the Spring Data API

- The Spring JPA Data API is a framework that simplifies development of data access layers in Spring-based apps
- Developers write repository interfaces that define data access methods they need
- Spring automatically generates the SQL queries to implement those data access methods
Overview of the Spring Data API

- The Spring JPA Data API is a framework that simplifies development of data access layers in Spring-based apps.
  - Developers write repository interfaces that define data access methods they need.
  - Spring automatically generates the SQL queries to implement those data access methods.
  - It reduces boilerplate code needed to implement persistence & data access layers in a Java application.

See en.wikipedia.org/wiki/Low-code_development_platform
Motivating the Need for a Custom SQL Query
Motivating the Need for a Custom SQL Query

- There are limitations to combos of Spring Data API keywords that are supported via the JPA

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Sample</th>
<th>JPQL</th>
</tr>
</thead>
<tbody>
<tr>
<td>NotLike</td>
<td>findByFirstnameNotLike</td>
<td>... where x.firstname not like ?1</td>
</tr>
<tr>
<td>StartingWith</td>
<td>findByFirstnameStartingWith</td>
<td>... where x.firstname like ?1 (parameter bound with appended %)</td>
</tr>
<tr>
<td>EndingWith</td>
<td>findByFirstnameEndingWith</td>
<td>... where x.firstname like ?1 (parameter bound with prepended %)</td>
</tr>
<tr>
<td>Containing</td>
<td>findByFirstnameContaining</td>
<td>... where x.firstname like ?1 (parameter bound wrapped in %)</td>
</tr>
<tr>
<td>OrderBy</td>
<td>findByAgeOrderByLastnameDesc</td>
<td>... where x.age = ?1 order by x.lastname desc</td>
</tr>
<tr>
<td>Not</td>
<td>findByLastnameNot</td>
<td>... where x.lastname &lt;&gt; ?1</td>
</tr>
<tr>
<td>In</td>
<td>findByAgeIn(Collection ages)</td>
<td>... where x.age in ?1</td>
</tr>
<tr>
<td>NotIn</td>
<td>findByAgeNotIn(Collection age)</td>
<td>... where x.age not in ?1</td>
</tr>
<tr>
<td>TRUE</td>
<td>findByActiveTrue</td>
<td>... where x.active = true</td>
</tr>
<tr>
<td>FALSE</td>
<td>findByActiveFalse</td>
<td>... where x.active = false</td>
</tr>
<tr>
<td>IgnoreCase</td>
<td>findByFirstnameIgnoreCase</td>
<td>... where UPPER(x.firstname) = UPPER(?1)</td>
</tr>
</tbody>
</table>

Motivating the Need for a Custom SQL Query

• There are limitations to combos of Spring Data API keywords that are supported via the JPA
  
  e.g., combining “Containing” & “In” is not possible
Motivating the Need for a Custom SQL Query

- There are limitations to combos of Spring Data API keywords that are supported via the JPA
  - e.g., combining “Containing” & “In” is not possible

Here’s ChatGPT’s explanation of why these methods can’t be combined:

**Why can't Spring Data API methods combine "Containing" and "In"?**

Spring Data API methods cannot combine "Containing" and "In" because these two operators are not compatible with each other.

"Containing" is used to search for records that contain a specific substring in a field, while "In" is used to search for records where a field matches one of several specified values.

The reason that these operators cannot be combined is that "Containing" requires a single value to be searched for, while "In" requires multiple values to be searched for. Combining these operators would require searching for multiple substrings in a field, which is not possible with the current implementation of Spring Data.

Good luck finding even this explanation with a Google search..
Structure & Functionality of the MultiQueryRepository
Structure & Functionality of the MultiQueryRepository

• The MultiQueryRepository defines an interface for creating a custom query

```java
public interface MultiQueryRepository {
    ...
}
```

This interface defines the means to perform multiple queries at once on the Quote database

See [microservices/src/main/java/edu/vandy/quoteservices/common/MultiQueryRepository.java](microservices/src/main/java/edu/vandy/quoteservices/common/MultiQueryRepository.java)
Structure & Functionality of the MultiQueryRepository

- The MultiQueryRepository defines an interface for creating a custom query

```java
public interface MultiQueryRepository {
    ...
    List<Quote> findAllByQuoteContainingIgnoreCaseAllIn(List<String> queries);
}
```

This method can’t be generated automatically by the Spring Data API
Structure & Functionality of the MultiQueryRepository

- The MultiQueryRepository defines an interface for creating a custom query

```java
public class MultiQueryRepositoryImpl implements MultiQueryRepository {
    ...
}
```

Applies the “Repository Implementation” pattern

See docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.custom-implementations
Structure & Functionality of the MultiQueryRepository

- The MultiQueryRepository defines an interface for creating a custom query

```java
public class MultiQueryRepositoryImpl
    implements MultiQueryRepository {
    @PersistenceContext
    private EntityManager entityManager;
    ...
}
```

This field defines a database session that provides the main API for performing CRUD operations & querying the database.

See docs.oracle.com/javaee/7/api/javax/persistence/EntityManager.html
Structure & Functionality of the MultiQueryRepository

- The MultiQueryRepository defines an interface for creating a custom query

```java
public class MultiQueryRepositoryImpl
    implements MultiQueryRepository {
...

List<Quote> findAllByQuoteContainingIgnoreCaseAllIn(
    List<String> queries) {
    var criteriaBuilder = entityManager.getCriteriaBuilder();
    var criteriaQuery = criteriaBuilder.createQuery(Quote.class);
    var quote = criteriaQuery.from(Quote.class);
    var idExpression = criteriaBuilder.lower(quote.get("quote"));
    var andPredicate = ...
    return getQueryResults(criteriaQuery, andPredicate);
...
```

Find Quote objects containing all queries (ignoring case)

See upcoming lesson on “Implementing the Zippy Microservice”
End of the QuoteServices App Case Study: Zippy MicroService Structure & Functionality (Part 2)