Chapter 6. GemFire OQL Query

6.1. Introduction

In this lab, we will be leveraging Object Query Language (OQL) to access data within GemFire. When you are accessing data, you don't always have the key available, nor do you know what the exact structure of the data is. In this lesson, we will be implementing methods that query GemFire data.

Concepts you will gain experience with:

- Using a query connection in GemFire
- Writing an OQL style query
- Performing a query for individual attributes of an object
- Writing queries involving joins between regions

Estimated completion time: 30 minutes

6.2. Quick Instructions

Quick instructions for this exercise have been embedded within the lab materials in the form of TODO comments. To display them, open the `Tasks` view (Window -> Show view -> Tasks (*not Task List*)).

6.3. Detailed Instructions

Instructions for this lab are divided into specific sections. Each section describes the steps to perform specific tasks. Before beginning this lab, make sure you have started the server side processes using the `startServer.sh` script (`startServer.bat` for Windows) in the `server-bootstrap` lab folder.

6.3.1. Implementing the general query functionality

To start, you will implement a general purpose query method, `doQuery()` that simply takes a `String` as an argument and returns a `SelectResults` result set. While not robust enough for broad usage, it will suit the purpose for this lab to have this functionality encapsulated in a method such that the remaining sections can focus primarily on writing the query string.

(`TODO-01`) Open the `OQLInquirer.java` class file (in the `com.gopivotal.bookshop.buslogic` package) and locate the `doQuery()` method, which has been stubbed out for you. Implement the functionality of this method to perform the following tasks:

- Get a `QueryService` from the cache
- Create a new query using the supplied query string
- Execute the query, which will have to be done inside a `try/catch` block
- Either return the results of executing the query or throw a `QueryException` wrapping an exceptions caught in the catch block

6.3.2. Basic OQL Query with Objects

This section will do a query against the `Customer` region to access data as `Customer` Objects.

- (`TODO-02`) In the same file, locate the `doCustomerQuery()` method. In this method, you will only need to do two things.
  - Write a proper OQL query in the query string to fetch all entries from the `Customer` region
  - Return the results of calling the `doQuery()` method with the query string you just wrote
- (`TODO-03`) Open the `OqlInquirerTests.java` class file in the `com.gopviotal.bookshop.tests` package. Execute the tests. If you correctly implemented the `doCustomerQuery()` method in the prior step, the first test, `testBasicQuery()` should pass.

> **Note**
>
> Don't worry yet that the other two tests are failing. You will fix this in the remaining sections.

6.3.3. OQL Query with Struct objects

Once you have the basic customer query working, you can move on to write a similar query to only return certain fields of the Customer object.

- (`TODO-04`) Locate the doStructQuery() method in the `OQLInquirer` class. Implement the method using the following steps.
  - First, write the correct query string to return a projection list. That is, perform a query where you will return

only the `customerNumber`, `firstName` and `lastName` fields of the `Customer` entries.

- Return the results of calling `doQuery()` with the query string you just wrote

- (`TODO-05`) Return to the `OqlInquirerTests` class and re-run the tests to verify you correctly implemented this functionality. If so, then both the `testBasicQuery()` and `testStructQuery()` tests should now pass. The `testJoinQuery()` test will still fail until you complete the last section.

6.3.4. Performing Joins

In this final section, you will be creating a more complex query by performing a join operation between the Customer region and BookOrder region. The goal is to return a list of all customers who have placed an order with total price greater than $45.00. To better understand the requirements for this query, take a moment to examine the following diagram showing the class definitions found within these two regions.

You will be performing an equ-join query in which you link `Customer` entries in the `/Customer` region to the `BookOrder` entries in the `/BookOrder` region by the `customerNumber` property of each object. With this in mind, perform the following tasks.

- (`TODO-06`) Locate the `doJoinQuery()` method in the `OQLInquirer` class. Implement the method using the following steps.
  - First write the correct query string to perform the join query. You are selecting a unique list of customers who's total order is greater than $45.00.
  - Return the results of calling `doQuery()` with the query string - you should be returning a `SelectResults` for `Customer` objects.
- (`TODO-07`) Return to the `OqlInquirerTests` class and re-run the tests to verify you correctly implemented the join query functionality. If so, then all 3 tests should now pass.

Congratulations!! You have completed this lab.
