



FlexibleSearch

Exercise

TABLE OF CONTENTS

GOAL	3
INSTRUCTIONS	3
Preparation	3
Step 1 • Define a Spring Bean for the Customer DAO	4
Step 2 • Create Test Customers and Test Orders.....	4
Step 3 • Write a FlexibleSearch Query for a Customer's Recent Orders	5
Step 4 • Implement the <code>getRecentOrdersForCustomer()</code> Method	5
Step 5 • Define a Spring Bean for Your Orders DAO	6
VERIFY	7
Solution	7
RECAP	7
Query syntax example for use in the HAC	8

GOAL

In this exercise, you will learn how to define a DAO class to encapsulate your data access. You will also practice using FlexibleSearch to create queries that retrieve the data you need.

INSTRUCTIONS

You should already be familiar with the syntax and features of FlexibleSearch. Armed with this knowledge, you will first prepare your system before tackling this exercise. (Not quite slaying dragons, but still exciting, right?)

Preparation

P1 First, stop the server and run the **setup** Ant target in the FlexibleSearch exercise build file:

In your *cmd* or *Terminal* window, navigate to

`MYPATH/workspace/TrainingLabTools/exercise_FlexibleSearch` and execute:

```
ant -f flexible_search_tasks.xml setup
```

P2 Once the setup task completes, you will have a new extension called, **trainingflexiblesearch**. Some partially completed Java classes will have been copied to it, and an entry for this extension will have been added to `hybris/config/localextensions.xml`.

To work with this new extension in Eclipse, you need to import it into your workspace. (For instance, you can select File | Import | General | Existing Projects into Workspace, then select the **trainingflexiblesearch** folder under custom folder and import it.)

P3 Start the server.

Our task in this exercise is to provide programmers a convenient way to retrieve **Orders** placed in the last 30 days for a given customer in the electronics store.

In the steps below, we will (though not in this order):

- Write a FlexibleSearch query capable of obtaining the desired data from the persistence layer.
- Complete a partially written method in a DAO class that encapsulates this query.
- Declare Spring beans (and an alias) that configures your DAO and allows other Spring beans to use **your** DAO in their configurations.

Now you can start working on the exercise. You will be editing three files: `trainingflexiblesearch-spring.xml`, `MyOrderDao.java` and `MyOrderDaoImpl.java` located here:

`MYPATH/workspace/hybris/bin/custom/trainingflexiblesearch/src/my/commerce/trainingflexiblesearch/dao/MyOrderDao.java`

`MYPATH/workspace/hybris/bin/custom/trainingflexiblesearch/src/my/commerce/trainingflexiblesearch/dao/impl/MyOrderDaoImpl.java`

`MYPATH/workspace/hybris/bin/custom/trainingflexiblesearch/resources/trainingflexiblesearch-spring.xml`

Step 1 • Define a Spring Bean for the Customer DAO

You can proceed with this step even if the server hasn't finished starting up.
You won't need it until Step 2.



- 1.1 In **trainingflexiblesearch-spring.xml**, define a bean instance of `DefaultGenericDAO` with the id: **defaultGenericCustomerDao**. Provide an appropriate **constructor-arg** as shown in an example in this chapter's slides).

Our verification script will use a bean by that exact name to obtain the `CustomerModel` instance needed to test your **myOrderDao** bean.

Let's break with normal procedure and NOT define an alias for this bean.

Why not a `customerDao` alias?



Normally we would have created a **customerDao** alias that refers to the bean we just created to query Customers. But we won't do that, since OOTB, the **customerDao** alias is already defined, and refers to a Java interface unrelated to this class, called **CustomerDao**.

Step 2 • Create Test Customers and Test Orders

- 2.1 **Before we start:** Since we don't cover how to set up the composable storefront until a later lesson, we will use the accelerator storefront for now to perform the following steps:

Open a browser tab and navigate to your electronics store (<https://electronics.local:9002/yacceleratorstorefront>) and register a new Customer – with the **uid** (email address): **test@sap.com**, first name: **Test**, last name: **Customer**. Make up values for the rest of this customer's info. This customer will be used by the verification script to test your solution. Give it an easy-to-remember password, like 123456.

- 2.2 Create and complete three orders while logged-in to the electronics store as **test@sap.com**:

Go on a shopping spree: put one or more in-stock items into your cart and complete the checkout process. Make up shipping and billing addresses; use a VISA credit-card with the number 4111111111111111 (four, followed by fifteen one's), any 3-digit number for the CVC code, and a date in the future for the expiration date. Select the checkbox to save the addresses and credit card, so that you can use "express checkout" when placing your second and third order. Make sure you place 3 orders in total.

- 2.3 Go into Backoffice (log in as *admin|nimda*), find the customer **test@sap.com**, (under the User navigation node), and view this Customer's details. Under the ORDERS tab, scroll down to see the list of this customer's Orders.

Edit two of the three orders (double-click an order to open it for editing) to change the **date** attribute (under the Order's POSITIONS AND PRICES tab). Change one of the orders so it was placed less than **30** days in the past (e.g., 25 days ago), and the other so it was placed 31 days or more (>30 days) in the past. Leave the third order with today's date (remember, our query's requirement is to pick only orders placed in the previous 30 days). Save all your changes

- 2.4 From the electronics store, logout as **test@sap.com** and register a second Customer, **test2@sap.com**. Place one order logged-in as **test2@sap.com** (leave the Order's **date** alone). This Order (and our 31-day-old Order) should be omitted by our query, since it's for a different customer.

Step 3 • Write a FlexibleSearch Query for a Customer's Recent Orders

- 3.1 In HAC's FlexibleSearch Console, (*Console > FlexibleSearch*), write and execute (test) a FlexibleSearch query that finds all orders in the last 30 days for a specific customer (in this specific case, use **test@sap.com**) in the electronics store.

Hint 1: One way to design this query is to use a three-way join (three Commerce types with two joins, each defined by a *join* condition) – think about which attribute pairs between two joined types would have to be equal to each other in each join condition. Refer to the diagram below for help.

Hint 2: In HAC's FlexibleSearch Console, placeholders cannot be used, so you must use either literals or DB functions. Where a DATE is required, you must use either date literals (technically speaking, that means VARCHAR2 literals with the default DATE format) or HSQLDB Date Functions, such as:

...where {date} >= trunc(TODAY) - '30' DAY

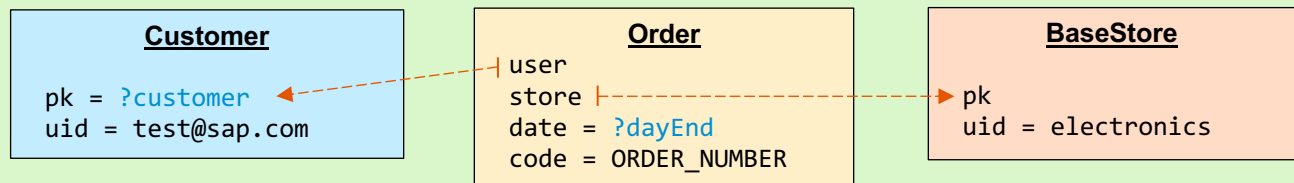
Hint 3: Once you are confident that your FlexibleSearch Query includes (and excludes) the right Order data, rewrite it in preparation for its use in Java code. Verify one last time in the HAC FlexibleSearch Console, (prior to coding your query as Java code) that its syntax is still legal.

Query Hint: Examples of how to build up the query progressively in the HAC can be found at the end of the chapter.

Solution Hint: If you have trouble writing the query, you can look at the solution in:

[MYPATH/workspace/TrainingLabTools/resources/exercise_FlexibleSearch/solution/MyOrderDaoImpl.java](#)

Have a look at this simplified UML diagram to understand how Order, Customer, and BaseStore items are typically linked:



We want to return only recent orders (≤ 30 days), so we'll compare each Order's *date* attribute to the date 30 days ago. Remember that if we want to benefit from query caching, we'll need to truncate the *current date* you obtained from your system. The *getRecentOrdersForCustomer()* method fragment we provided defines the variable *date30DaysAgo*, which you can compare to the *date* of each Order using query *?placeholders*. (Generally, it's best to avoid hard-coded literals or DB-vendor-specific functions in Java-based FlexibleSearch queries). ✓

Step 4 • Implement the `getRecentOrdersForCustomer()` Method

- 4.1 In the Eclipse IDE, look inside the **trainingflexiblesearch** project to locate the following 2 classes provided by the setup script:
- `my.commerce.trainingflexiblesearch.dao.MyOrderDao`
 - `my.commerce.trainingflexiblesearch.dao.impl.MyOrderDaoImpl`.

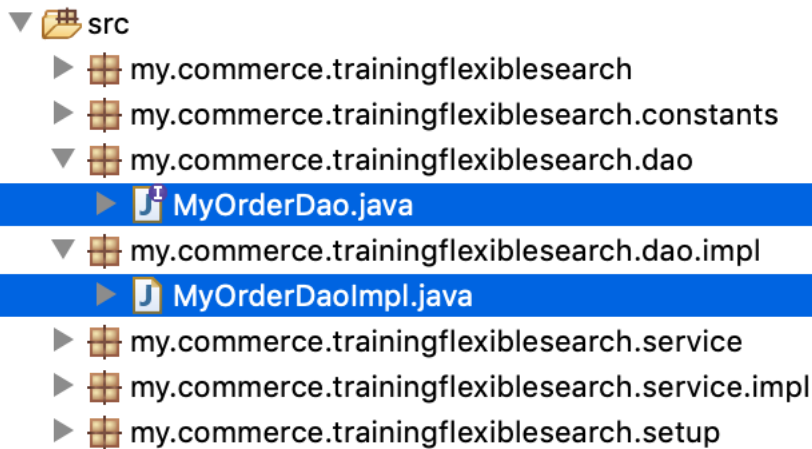


Figure: DAO interface and DAO class

As a best practice, the interface *MyOrderDao* defines a method signature for the *getRecentOrdersForCustomer(...)* method, which accepts a *CustomerModel* item as an input parameter and return a *List* of *OrderModel* objects. This interface is then implemented by *MyOrderDaoImpl*.

Make the necessary changes now to the *MyOrderDao* interface to apply the best practice.

When writing Data Access Objects for SAP Commerce, another best practice is for your DAO interface (*MyOrderDao*) to extend the OOTB *de.hybris.platform.servicelayer.internal.dao.Dao* interface, and for your implementation class (*MyOrderDaoImpl*) to extend the OOTB class *de.hybris.platform.servicelayer.internal.dao.AbstractItemDao*. This is already done in the prepared *MyOrderDaoImpl* class, please take a check of it.

4.2 Now complete the implementation of the following method in *MyOrderDaoImpl*:

```
List<OrderModel>
```

```
my.commerce.trainingflexiblesearch.dao.MyOrderDao.getRecentOrdersForCustomer(CustomerModel).
```

This method returns all the recent orders (i.e., within 30 days) for the *CustomerModel* provided. Use the query you wrote in the previous exercise as a *starting* point. It's actually simpler here, since you have a *Customer item* at hand (it's the method parameter), so you will no longer be working with just a *Customer uid* (instead, use the *pk* directly). Also, remember (from the slides) that when adapting *FlexibleSearch* queries for use with the Java API, it's easiest if the query can be adapted to select only *PKs*.

(NOTE 1: after adapting your *FlexibleSearch* query to take parameter objects and only return *PKs*, your query might not need as many *JOINS* or subqueries as it started with).

(NOTE 2: if you use the *SearchResult* class, it is under the package of *de.hybris.platform.servicelayer.search*, don't confuse it with the other *SearchResult* classes. Also, maybe you don't really have to use/import the class at all, you could check the solution implementation in *TrainingLabTools/exercise_FlexibleSearch/solution* to know why.)


(NOTE 3: if you can't get the query implemented with the Java API, it's ok to check the solution implementation and use the code there directly).

Step 5 • Define a Spring Bean for Your Orders DAO

- 5.1 In *trainingflexiblesearch/resources/trainingflexiblesearch-spring.xml*, define a bean with id: **defaultMyOrderDao**, class: **MyOrderDaoImpl** (use its fully qualified name), and parent: **abstractItemDao**. This bean does not need any *property* sub-tags (see notes below).

As usual, define a corresponding Spring *alias* for your new bean; call it **myOrderDao**.

NOTE: In real life, our DAO class would likely extend the out-of-the-box DAO with the methods we defined. In that case, our alias would likely be named `orderDao`, overriding the OOTB alias. For this example, we just created an independent bean and alias, for which we should probably have selected more informative names.

You can access the `FlexibleSearchService` Spring bean from within **MyOrderDaoImpl** by calling `getFlexibleSearchService()`. Your class inherited this getter, along with its setter counterpart, from its parent class, `AbstractItemDao`. When you define the Spring bean **defaultMyOrderDao** in **trainingflexiblesearch-spring.xml**, you could set a property to refer to the `flexibleSearchService` bean, but you don't need to after specifying the **abstractItemDao** bean as its parent, because it causes *your* bean definition to inherit the reference from the parent's definition. 

Recall that the slides showed you how to use static constants from a Model class to ensure the correctness of attribute names within a Java-based `FlexibleSearch` query. This also ensures that changes to the model will cause a compilation error, quickly alerting you to the problem. While we won't ask you to do this here, you might notice that we did so in the solution.

VERIFY

- V1 Recompilation is necessary for your code changes to be used by the SAP Commerce Cloud platform environment. To compile, stop the SAP Commerce Cloud Platform, run an `ant all`, and once again start the SAP Commerce server.
- V2 Go to the **HAC** and run the script **verifyFlexibleSearchExercise**. This script will show you the two orders created by customer **test@sap.com** within the 30-day period. Note that the third order, that you changed the date for 31 days ago should not be shown.

Solution

If you don't wish to complete this exercise manually, you can install the solution provided:

- S1 Navigate to the *Terminal* or *cmd* window where the server is running, and if it is running, stop it by entering `CTRL-C`.
- S2 Navigate to `MYPATH/workspace/TrainingLabTools/exercise_FlexibleSearch` and execute:

```
ant -f flexible_search_tasks.xml solution
```

Please remember that if you execute the solution script directly: after restarting the server, you still need to complete Step 2, since the solution script doesn't create the test customer or test orders for you. This test user is referenced by the verification script.

RECAP

This exercise taught you how to write `FlexibleSearch` queries to access data inside the SAP Commerce Cloud. You have also made use of `FlexibleSearch` inside Java code to retrieve data.

Query syntax example for use in the HAC

One way of coding your FlexibleSearch query is to build it up in several iterative steps and test each one in the HAC FlexibleSearch console. As you prove out each step, you can copy and paste the statements into your Java code FlexibleSearch statement – of course, adding in parameters as needed.

Let's look at an example of this to build up our first query in a series of steps. You can run each of these in the HAC FlexibleSearch.

First, let's just select the uid from customers where the uid is **test@sap.com**. Notice the single quotes in the SELECT statement.

```
SELECT {uid} FROM {Customer} WHERE {uid}='test@sap.com'
```

OK, you saw how simple that was. Now let's JOIN customers with orders where the user attribute in the order item is the same as the PK of a customer item.

```
SELECT {c.uid}, {o.code} FROM {Order as o JOIN Customer as c on {c.pk}={o.user}}
```

Fine, now let's keep going by only getting the orders where the customer is **test@sap.com**.

```
SELECT {c.uid}, {o.code} FROM {Order as o JOIN Customer as c on {c.pk}={o.user}}  
WHERE {c.uid}='test@sap.com'
```

Now we can practice with HSQL's Date Functions: CURRENT_DATE returns today's date (an alias for this function is TODAY). In Java code, you would need to use the Calendar object with only the date portions set (i.e., with the time fields zeroed-out) – you can refer to the slides to see how to do that. This will find all the orders that was created in the last 30 days.

```
SELECT * FROM {Order as o} WHERE {o.date} >= (CURRENT_DATE - '30' DAY)
```

Now let's find the customers associated with those orders.

```
SELECT {c.uid}, {o.code} FROM {Order as o JOIN Customer as c on {c.pk}={o.user}} WHERE  
{c.uid}='test@sap.com' AND {o.date } >= (CURRENT_DATE - '30' DAY)
```

And finally, we can JOIN-in the stores associated with those orders and then filter out the orders that were not placed in the electronics store.

```
SELECT {c.uid}, {o.code}, {s.uid} FROM {Order as o JOIN Customer as c on {c.pk}={o.user}}  
JOIN BaseStore as s on {s.pk}={o.store}} WHERE {s.uid} = 'electronics' AND  
{c.uid}='test@sap.com' AND {o. date} >= (CURRENT_DATE - '30' DAY
```


www.sap.com

© 2023 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company.

The information contained herein may be changed without prior notice. Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

In particular, SAP SE or its affiliated companies have no obligation to pursue any course of business outlined in this document or any related presentation, or to develop or release any functionality mentioned therein. This document, or any related presentation, and SAP SE's or its affiliated companies' strategy and possible future developments, products, and/or platform directions and functionality are all subject to change and may be changed by SAP SE or its affiliated companies at any time for any reason without notice. The information in this document is not a commitment, promise, or legal obligation to deliver any material, code, or functionality. All forward-looking statements are subject to various risks and uncertainties that could cause actual results to differ materially from expectations. Readers are cautioned not to place undue reliance on these forward-looking statements, and they should not be relied upon in making purchasing decisions.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies. See www.sap.com/copyright for additional trademark information and notices.

SAP Customer Experience

