Article
Benjamin De Boe · Jun 28, 2016 7m read

Open Exchange

iKnow demo apps (part 5) - iFind search portal

Earlier in this series, we've presented four different demo applications for iKnow, illustrating how its unique bottom-up approach allows users to explore the concepts and context of their unstructured data and then leverage these insights to implement real-world use cases. We started small and simple with core exploration

through the Knowledge Portal, then organized our records according to content with the Set Analysis Demo,

organized our domain knowledge using the Dictionary Builder Demo and finally build complex rules to extract nontrivial patterns from text with the Rules Builder Demo.

This time, we'll dive into a different area of the iKnow feature set: **iFind**. Where iKnow's core APIs are all about exploration and leveraging those results programmatically in applications and analytics, iFind is focused specifically on search scenarios in a pure SQL context. We'll be presenting a simple search portal implemented in Zen that showcases iFind's main features.

A brief introduction to iFind

In the iKnow capabilities we've discussed in previous articles, a *domain* was the primary subject of exploration. With iFind, we stick to regular SQL tables, on which an *iFind index* is defined. An iFind index is an implementation of the Functional Index infrastructure, which allows for the development of custom index types, implementing a particular capability that is triggered when records are added, updated or removed from the table on which the index is defined. In the case of iFind, that particular capability is rich text indexing, leveraging the power of the iKnow engine. In case you want to learn more about functional indices, this article discusses another custom index type for coordinates. In addition to the indexing part, which ensures words as well as iKnow concepts and context are written to a dedicated index structure, iFind also includes a custom SQL search implementation, which leverages these data structures at query time.

In practice, all you have to do to work with iFind is define the index using the index type syntax in the class definition:

```
Class ThePackage.MyClass Extends %Persistent
{
 Property MyStringProperty As %String;
 Index MyBasicIndex On (MyStringProperty) As %iFind.Index.Basic;
}
```

And then use the %FIND syntax to query the index:

```
SELECT * FROM ThePackage.MyClass WHERE %ID %FIND search_index(
MyBasicIndex, 'interesting')
```

In the background , all plumbing is taken care of automatically.

There are currently three "levels" of iFind functionality, which you can choose from through selecting the index type class in the class definition:

- **%iFind.Index.Basic**: offers basic word-level search (no iKnow entities)
- **%iFind.Index.Semantic**: offers everything in the Basic index, plus iKnow entities and corresponding search features
- **%iFind.Index.Analytic**: offers everything in the Semantic index, plus iKnow paths, dominance and proximity information

To configure additional properties of the index, such as the language (defaults to English), case sensitivity or whether SQL projections of the additional data should be generated, you can use index parameters in the index definition:

```
Class ThePackage.MyClass Extends %Persistent {
  Property MyStringProperty As %String;
  Index MyBasicIndex
On (MyStringProperty) As %iFind.Index.Basic(LANGUAGE="fr", IFINDMAPPINGS=1);
}
```

More details on the available parameters and their values can be found in the class reference of %iFind.Index.Basic and its subclasses.

Now, after this quick techie intro, let's turn our attention to the demo application, which will explore the actual query features iFind has to offer.

Installation & Setup

This demo is available for download from a GitHub repository, where you can either check it out using your preferred Git client, or download it as a zip outright. You should be able to import and compile all classes in any release of Caché starting with 2016.1.

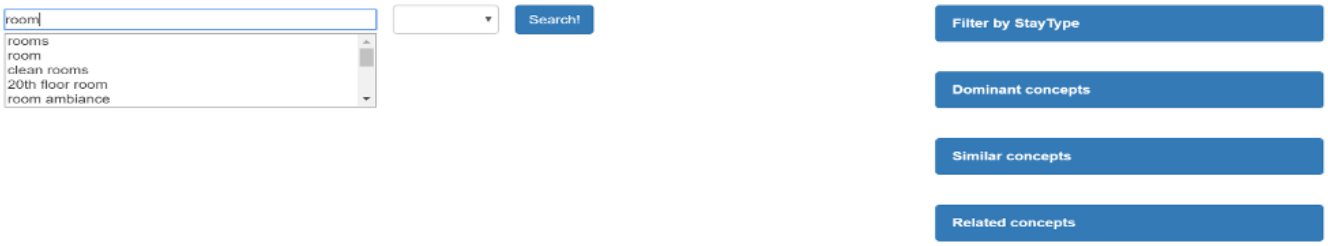
Similar to the earlier Knowledge Portal demo, the app consists of a Zen page spiced up with JQuery and Bootstrap for a slick look. Those third-party libraries should already be available as part of your default 2016.1 installation. You can then access the demo from the following URL: http://localhost:57772/csp/user/Demo.SearchPortal.Home.zen

The search portal obviously needs to be pointed to a table in order to query it. By default, it looks for a table named DemoHotels.Review that is also part of the demo package and has an iFind index defined on it. For copyright reasons, we couldn't include the demo data we used, but either you can make some up of your own or use a tool like import.io to scrape some from a site like TripAdvisor. If you'd like to point to a different table, either override the value of the IFPTABLE class parameter or use the "t" request parameter, appending it to your URL "?t=MyOwn.TableName". The page will automatically look for an iFind Analytic index in the table, but if you have more than one, you can select which one to use using the IFPINDEX class parameter or "i" request parameter.

If your table has, next to the text column we'll be searching against, other columns that are relevant to filter search results, make sure you define a bitmap index on them. The search portal will recognize those and use them for faceting, allowing you to easily drill down into a particular subset of the results based on that column's value. In the sample table, "stay type" is used as such. You can again override this automatic recognition by setting the IFPFILTERFIELDS class parameter or "mf" request parameter. Other reserved columns for an order field and title field can be designated using the IFPORDERFIELD and IFPTITLEFIELD class parameters or "of" and "tf" request parameters respectively.

Using the Search Portal

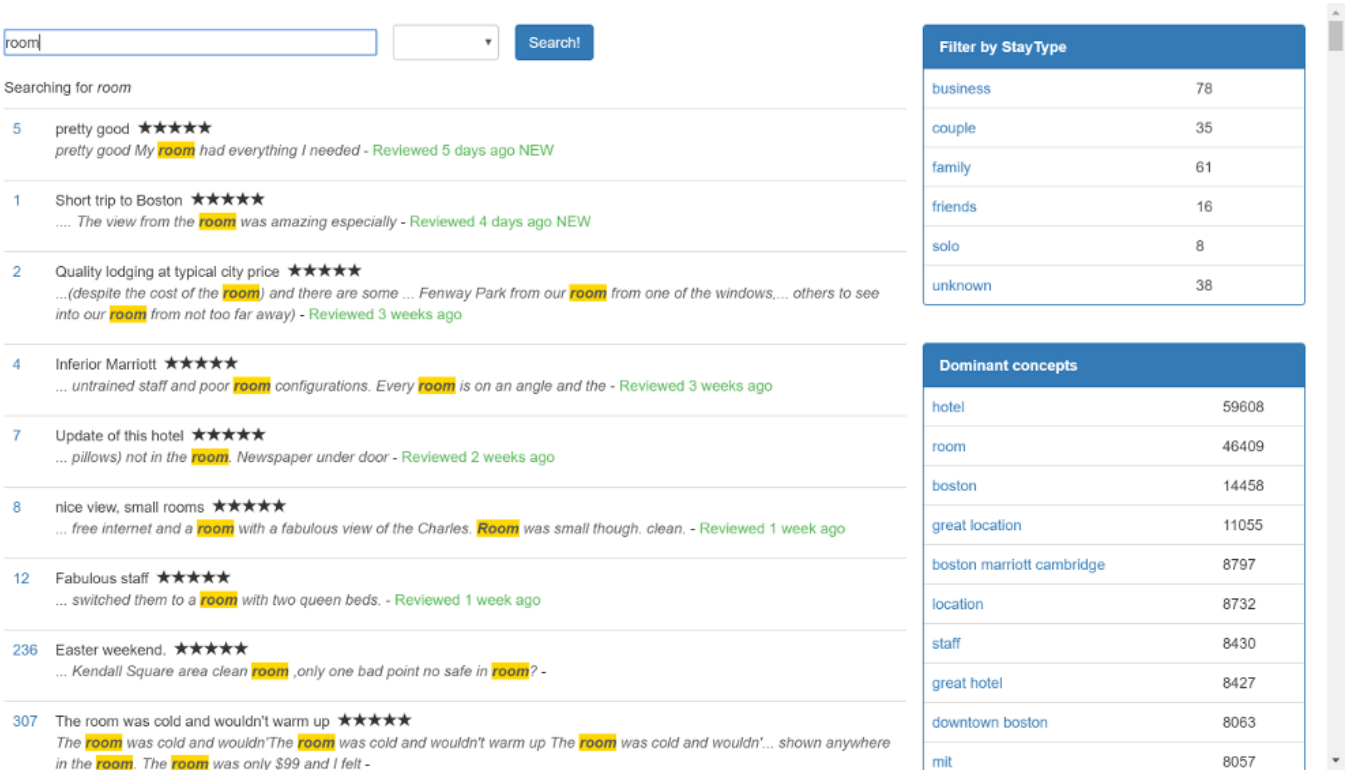
Now, time for some action. If your table contains data and the app has been set up through the right parameters as described above, you'll see search suggestions appear as soon as you start typing anything in the search box at the top:



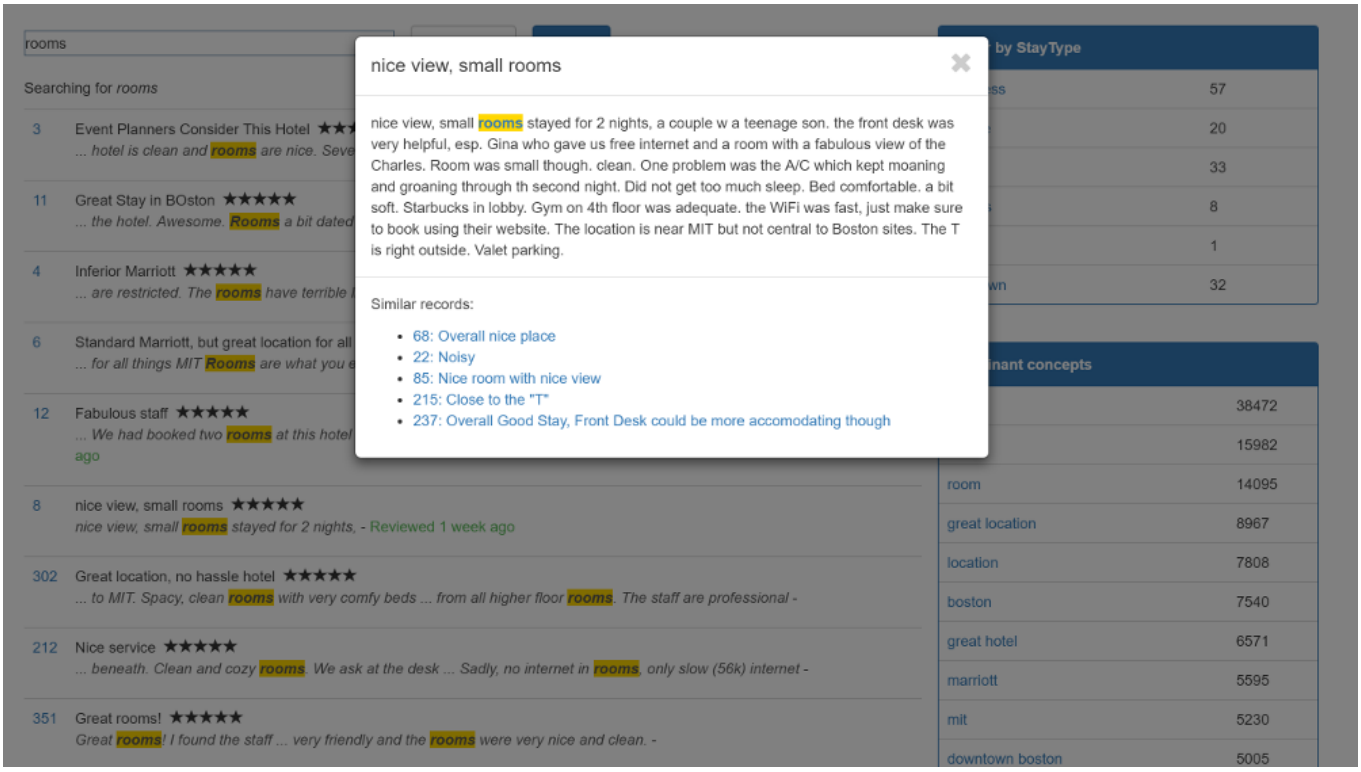
These are the results of the %iFind.FindEntities() stored procedure that will look in all the iKnow entities stored by iFind in that namespace.

Searching

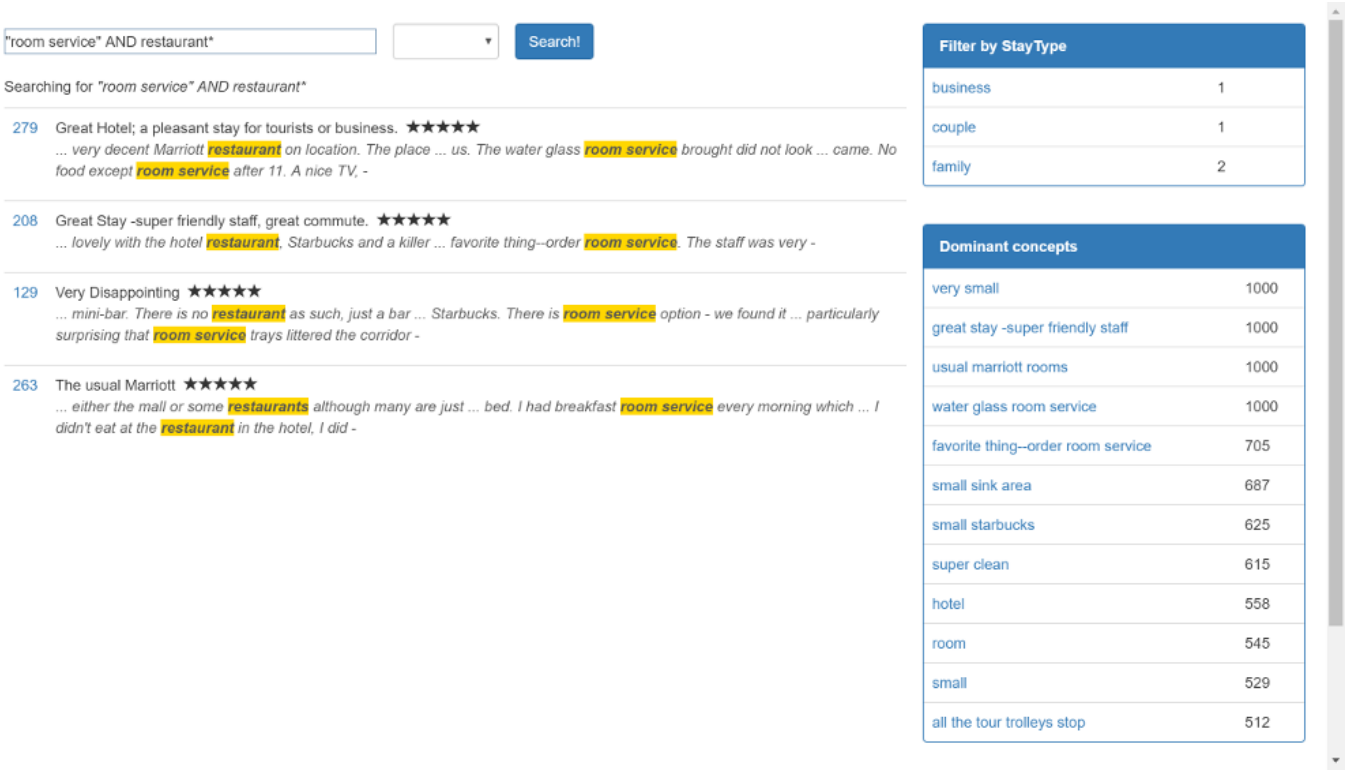
When you click enter or search, the page will display your search results:



In the main overview, you'll see the search results, with snippets highlighting the search term matches based on DemoHotels.ReviewiFindHighlight() and the star rating is based DemoHotels.ReviewiFindRank(), which would also have been a good measure to sort the results by. If you click on any record, you'll see a popup with the entire document, and a number of similar documents based on the custom query in Demo.SearchPortal.Utils.



Apart from simple word searches, you can use all the syntax options available in iFind, including entity search and positional search as well as AND/OR/NOT compositions, to make your search request more specific.



Facets & Filtering

On the right side of the screen, you'll see four boxes with additional information, based on the search results of your current query. The top one displays facets of your results, based on the first filter column described earlier (with a bitmap index on it). These are the different values of that column and the number of query result records that have that particular value. By clicking on them, you'll be filtering your current query results according to that field's value and the next available bitmap-indexed column will appear as a faceting option.

The second widget displays **dominant concepts**, which are the iKnow entities deemed most relevant in the text searched. The third one will display all **similar entities** for the search term you supplied (not supported for composite search strings) and the last one all **related concepts** (as based on iKnow's proximity metric), aggregated over all concepts matching your search term. All of these results are aggregated over all the current search results and clicking on an entity will reset your search to that search term. These results are based on queries against the SQL projections iFind creates to expose the additional information generated by the iKnow engine underneath.

Under the Hood

While the code on the Demo.SearchPortal.Home page may look a bit confusing at first, you'll see that in fact all it does is build the appropriate SQL query for every widget on the screen. This illustrates how we designed iFind, as an all-SQL technology you can take advantage of in any SQL-based application.

If you take a closer look at the generated SQL statements, you'll see it uses the following predicate each time:

```
... WHERE RecId %FIND Demo_SearchPortal.Find(?) ...
```

This is in fact a little performance trick we implemented to ensure we could reuse the "set of search results" when populating the different widgets on the screen. When the user issues a search request, but before any of the widgets are refreshed, we'll do the "main search query", based on the search string and eventual facet filters, storing the results in a ^CacheTemp global. This is what happens in the DoSearch() method of the main Zen page. Then, the "identifier" for this set of search results, as registered by DoSearch(), is used as the argument to DemoSearchPortal.Find() , which is another implementation of the %SQL.AbstractFind interface we briefly discussed earlier. This allows us to do the work of satisfying the main search task just once and reuse the results as a bitmap throughout refreshing the rest of the page. No rocket science and iFind is pretty fast by itself, but another nice example of how Caché offers you the hooks to implement your own convenience functions right into the core of the system.

So, after four demo apps focusing on the iKnow APIs, this one demonstrated the capabilities of our iFind technology. It's still using iKnow under the hood, but exposing its functionality through an all-SQL interface that works well with any SQL-based application you may be developing. And while the basic word search functions may seem trivial at first, the options for refining your search to entities and, even more so, the ability to tap into the rich context information provided by the iKnow engine (dominant concepts, related concepts, relevance, ...) makes for a much richer interaction with your data.

#iFind #Indexing #SQL #iKnow
Check the related application on InterSystems Open Exchange