

Coveo Enterprise Search

How to Use Query Ranking Expressions

Applies to CES 5 and later

Query Ranking Expressions (QRE) allows the users to customize the ranking algorithm at runtime, using query expressions. A QRE contains any valid query expression (one or many terms and one or many operators supported by *Coveo Enterprise Search* (CES), including date queries, Boolean queries, fielded queries, etc.), as well as a ranking modifier that either boosts or dampens the ranking of documents that match the QRE. There are no limits to the number of QREs that can be evaluated, although it has an impact on performance if there are too many.

Contextual Search

A good example of QRE is contextual search: a developer is browsing the development intranet site, looking for something in a Document Library and decides to perform a search. A good way to perform contextual search is to limit the results to the origin of the search, in that case, the Document Library. However, note that this approach is very constraining. If the developer is looking for something not related to the Document Library, he will have to perform another search. The QRE feature helps because every search can be performed without a filter expression, but gives a “boost” (temporary increase in ranking weight) to documents originating from that search location.

Practical Applications

In the index, each item can have many fields, each having a different ranking value. The ranking can be computed solely on those fields. For example, if there are 4 fields: @itemname, @itemdescription, @itemcategory, @itembrand. The following QRE can be used to perform a completely customized ranking (where userquery stands for the query entered by the user and number in parentheses stands for the boost to give):

- @itemname = userquery (+40)
- @itemdescription = userquery (+30)
- @itemcategory = userquery (+20)
- @itembrand = userquery (+10)

Every document matching the item name gets 40 points; every document matching the item description gets 30 points, and so on. It is cumulative, so the more matching expressions there are for a given document, the more points it gets. In this case, it is possible to provide an as-needed search user interface enhancement to promote specific items, based on the criteria specified in the QRE.

Technical Details

▶ What is the impact on the final ranking score?

The QRE feature is computed during the second phase of the ranking algorithm. The impact on the final score is set to equal the maximum possible score of the second phase for a modifier of 100.

However, as there are no limits, it is possible to completely override the regular ranking algorithm if the values used are too high.

▶ **Can QREs be used with an out of the box CES?**

CES 5 does not include an interface to manage QREs in the Interface Editor. QRE must be used through admin-level development. Future CES releases will include a tool for QREs management.

▶ **How is QRE implemented in the Search API?**

In a new user interface–named IRankingModifiers–manage the QRE:

```
interface IRankingModifiers : IDispatch
{
[helpstring("Returns the number of items in the collection.")]
[id(1), propget] HRESULT Count([out, retval] long* p_pValue);

[helpstring("Adds a new ranking modifier to the collection.")]
[id(2)] HRESULT Add([out, retval] IRankingModifier** p_ppRankingModifier);
};

IRankingModifiers is a collection of IRankingModifier:
interface IRankingModifier : IDispatch
{
[helpstring("The expression for the ranking modifier.")]
[propget] HRESULT Expression([out, retval] BSTR* p_pExpression);
[propput] HRESULT Expression([in] BSTR p_Expression);

[helpstring("The ranking modifier. A negative value dampens, a positive value boosts.")]
[propget] HRESULT Modifier([out, retval] long* p_pModifier);
[propput] HRESULT Modifier([in] long p_Modifier);
};

You have access to the IRankingModifiers collection via the ISearchQuery2 interface:
interface ISearchQuery2 : IDispatch
{
...
[helpstring("The ranking modifiers to apply.")]
[propget] HRESULT RankingModifiers([out, retval] IRankingModifiers** p_ppModifiers);
[propput] HRESULT RankingModifiers([in] IRankingModifiers* p_ppModifiers);
...
};
```

Example Code in CoveoSearch.ascx

```
<%@ Assembly Name="Coveo.CES.Web.Search, Version=5.0.0.0, Culture=neutral,
PublicKeyToken=44110d16825221f2" %>
<%@ Import Namespace="Coveo.CES.Web.Search.Settings" %>
```

```
<%@ Import Namespace="Coveo.CES.Web.Search.Providers" %>

<script language="c#" runat="server">
    protected override void OnInit(EventArgs p_Args)
    {
        SetupSearchBuilder += this.Search_SetupSearchBuilder;
        base.OnInit(p_Args);
    }

    private void Search_SetupSearchBuilder(object sender, SetupSearchBuilderEventArgs args)
    {
        // Retrieve current query expression from the Effective query state
        QuerySetting settings = ((QuerySetting)State.Effective[QuerySetting.NAME]);

        if (settings != null) {
            string QueryExpression = settings.Expression;

            // Use SearchQuery object to parse query and retrieve mandatory terms
            // which we'll use to build the query ranking expressions
            ICESQuery query = Provider.CreateQuery();
            query.Expression = QueryExpression;
            IParsedQuery parsedQuery = query.GetParsedQuery();

            StringBuilder rankingQueryBuilder = new StringBuilder();
            foreach (object term in parsedQuery.MandatoryTerms) {
                if (rankingQueryBuilder.Length > 0 ) {
                    rankingQueryBuilder.Append(",");
                }
                rankingQueryBuilder.Append(term.ToString());
            }

            // Apply various ranking expressions based on query expression
            RankingExpressionInfo expr = null;

            // Field1 expression
            expr = new RankingExpressionInfo();
            expr.Expression = String.Format("{0}={1}", "@title", rankingQueryBuilder.ToString());
            expr.Modifier = 95; // -100 to 100
            args.Builder.RankingExpressions.Add(expr);
        }
    }
</script>
```