How to find entities and relationships

An Egeria Metadata Repository contains entities that represent the objects in your data lake and relationships that describe how those objects are related to one another. Collectively, these entities and relationships are known as metadata instances.

Every Metadata Repository implements the Metadata Collection interface, which supports methods to add and remove metadata instances, and read or update existing instances.

As a user of Egeria - whether through an application or a service such as an access service (OMAS) or integration service (OMIS) - there will be times when you want to search for a particular entity or relationship instance. The Metadata Collection interface supports search by providing a number of 'find' methods.

The findEntities and findRelationships methods

The most versatile way to find entities or relationships is to use the `findEntities` or `findRelationships` method. These methods were introduced relatively recently and are implemented by Egeria's built-in repositories.

They accept a `SearchProperties` parameter which provides a powerful way to describe what you are looking for. A `SearchProperties` object is a set of conditions based on a fairly comprehensive set of predicates and capable of being composed from a number logical expressions.

For example, you could specify that you would like to find all entities that were created by a particular user, between a specified pair of dates/times and have names containing a particular string of characters. The conditions can be applied to any property (core or type-specific) and logically combined using AND, OR and NOT. The operators available include existence (IS_NULL, NOT_NULL), equality (EQ, NEQ), numeric comparison (GT, GTE, LT, LTE), text comparison (LIKE) and set membership (IN).

A `SearchProperties` object allows you to specify nested conditions, so it's possible to write very specific search criteria. Just as important, the comparison operators enable us to write expressions that include date ranges.

The other 'find' methods

The Metadata Collection interface also provides `findEntitiesByProperty` and `findRelationshipsByProperty` methods. These methods accept a `MatchProperties` object which can be used to specify a match value for each property.

The Metadata Collection interface also provides `findEntitiesByPropertyValue` and `findRelationshipsByPropertyValue` methods. These methods accept a string parameter called 'searchCriteria' which is compared to all string properties.

Unlike the newer `findEntities` and `findRelationships` methods, these other methods offered a more basic search capability. A `MatchProperties` object only supports a single value per property and it is not possible to nest or logically combine multiple conditions to the same extent that is possible with a SearchProperties object. Also, `searchCriteria` string comparison is conducted on all properties that are of type 'string', and combined using a single AND, OR or NOT criterion. Whilst they are adequate for relatively simple searches, they are not as powerful or flexible as the newer methods.

Search using string expressions

You can search using string properties with any of the above 'find' methods. A string used in a `SearchProperies`, `MatchProperties` or `searchCriteria` parameter can be treated as an exact match or as a regular expression ('regex'). The author of an OMAS needs to consider their end-user's expectations to decide whether a string should be matched exactly or treated as a regular expression ('regex').

Under the covers, the Metadata Collection interface treats all search strings as regular expressions. Note that not all repositories may support full regular expression syntax, so the caller can override it as described below.

Because the interface treats any string as a regex, a string such as "department" will match a property whose value is exactly that string. The string ".*department.*" uses regex syntax to match any value that contains the string "department", the surrounding ".*" patterns matching any number of leading or trailing characters.

Dealing with special characters

So what happens if the string you need to look for contains characters that have significant meaning in the regex syntax? In this case, you can literalise (or 'escape') the search string. Suppose, for example that you needed to look for values that contain '.' or '*' characters, such as "John.Smith". If you wanted, you could individually escape each significant character ("John\.Smith") but that can become onerous. A simpler way is to escape the whole string, and there is a set of helper methods in the `OMRSRepositoryHelper`. The helper methods support escaping of relatively simple search strings in a manner that is supported by most OMRS repository connectors, including those for repositories that do not support full regular expression syntax.

For example, if an OMAS author wants an exact match of a string, they could call getExactMatchRegex() which will 'escape' the whole string, regardless of content. This helper method frames the whole string with `\Q` and `\E` escape characters. It's OK to call `getExactMatchRegex()` even if the string value only contains alphanumeric characters and has no regex special characters. However, it should only be used for escaping a single, simple string - don't use it for a string that already contains either of these escape sequences. Also, don't use it to build up complex regular expressions.

Here's an example. Metadata objects frequently have compound names composed of multiple fields with separators. For example, an OMAS may need to retrieve the entity with qualifiedName equal to "[table]EMPLOYEE.[column]FNAME". Some of these characters are special characters in a regex. If the OMAS needs an exact match, it can call `getExactMatchRegex()` to escape the whole string. Although the string contains regex special characters, the search will only return an entity with the exact value.

There are additional helper methods that provide escaping and wrap the escaped string in additional regex syntax that enables it to match values that begin, contain or end with the original string. These are `getsStartsWithRegex()`, `getContainsRegex()` and `getsEndsWithRegex()`. These methods combine exact match processing with relatively simple regex substring expressions. If an OMAS needs a more complicated regex the author should code it directly instead of using the `OMRSRepositoryHelper` methods.

In addition to the above helper methods, there are corresponding methods to test whether an expression is an example of one of the supported patterns and to remove the escaping from a pattern.

An OMAS author should use the repository helper methods when they can. For more complex searches, beyond the level supported by the helper methods, an OMAS author could implement their own regular expression, but it is important to remember that not all repositories support all regular expressions. The regular expressions provided by the repository helper are a minimal set that most repositories are able to support. More complex expressions can be used with repositories that have full regex processing, such as the in-memory repository or graph repository.

Case sensitivity

There are times when you need to match a value exactly - even including the case of search pattern. Other times, case sensitivity gets in the way and you really don't want to have to construct long-winded regular expressions like "[Aa][Nn][Yy][Cc][Aa][Ss][Ee]".

To avoid this, the repostiory helper methods provide variants that have a boolean parameter that indicates whther you want case-sensitive matching or not. If you are not using the helper methods, and just writing 'raw' regular exression syntax, you can use (?i) at the start of a regex to indicate you want case-insensitive matching.

Search for dates and times

You can search using dates and times, although it is only since the introduction of the new findEntities and findRelationships that this became really flexible. This is because it is now possible to perform a comparison rather than a simple equality check. So, by combining a pair of conditions it is possible to specify a date range, such as between 01/01/2021 12:00 and 01/01/2021 13:00'. You would simply combine a GTE and a LTE using an AND criterion.

If you are accessing the metadata collection via a REST interface, specify a date/time using its (Long) timestamp. It will be converted and compared as a Date when used against a core property (such as `createTime`, `updateTime`) and will be used directly (as a Long) if being compared to a type-specific date or time property, such as the `dueDate` property on a `ToDo` entity.

Related articles

- How to find entities and relationships