Using LiQL

Learn about our LiQL query language used with GET requests to Community API v2. This guide describes the LiQL syntax, using LiQL for combinatorial and keyword searches, pagination with LiQL queries, and sorting query results

data using a familiar, SQL-like syntax. LiQL queries against the search index and is used with Read actions over HTTP and with the rest and restadmin FreeMarker context objects.

> 👍 **Looking for LiQL reference?**
>
> See LiQL collection and object reference for object schemas, field, constraint, sort, and function support for our Khoros Communities objects.

This guide describes the building blocks of a LiQL query. Once you understand how to build a query, see The Community API v2 call for examples of using LiQL in requests to the Community API v2.

With the LiQL syntax, you select fields from a collection of resources where you optionally define constraints to limit the data returned and order by specific parameters. See each resource definition to understand which fields can be used in with query clauses.

SELECT <fields> FROM <collection>	WHERE <constraints>	ORDER BY <fields> <DIRECTION>	LIMIT <number> OFFSET <number>

> 📘 LiQL variables are case sensitive. In the image above , , and must be in lower case. The SELECT, FROM, WHERE, ORDER BY, and IN clauses are case insensitive.

Collections, resources, and fields

In the Community API, a collection is a group of same-type resources (objects). The Messages collection, for example, is a group of Message resources. Collections are always plural, a resource

Using LiQL

Learn about our LiQL query language used with GET requests to Community API v2. This guide describes the LiQL syntax, using LiQL for combinatorial and keyword searches, pagination with LiQL queries, and sorting query results

Let's walk through a few LiQL queries and look at the responses. To do this, we'll use a tool in Studio called the API Browser. This tool lets you test LiQL queries without making a formal HTTPS request or writing any FreeMarker code.

The API Browser uses your Studio login for authentication and policy checks, so the roles and permissions assigned to your user account will affect the query results. Consider creating test accounts for different user test cases, such as a test administrator, moderator, and basic user with your base Community permissions.

To use the API Browser:

- You need to be granted the Use REST API browser permission
- You need a community that includes boards, messages, and users

To open the API Browser sign in to the community and navigate to **Studio > API Browser**.

If you do not see the API Browser tab, file a Support ticket.

First, we'll run a very basic query:

```
SELECT * FROM messages
```

Let's take a look at the results for that query. If you have boards and messages in your environment, you should see a response that looks similar to this:

SELECT * FROM messages


```
{
  "status" : "success",
  "message" : "",
  "http_code" : 200,
  "data" : {
```

Using LiQL

Learn about our LiQL query language used with GET requests to Community API v2. This guide describes the LiQL syntax, using LiQL for combinatorial and keyword searches, pagination with LiQL queries, and sorting query results


```
  "teaser" : "",
  "board" : {
    "type" : "board",
    "id" : "sauconyPeregrine7",
    "href" : "/boards/sauconyPeregrine7",
    "view_href" : "/t5/Saucony-Peregrine-7-Q-A/qa-p/sauconyPeregrine7"
  },
  "conversation" : {
    "type" : "conversation",
    "id" : "987",
    "view_href" : "/t5/Saucony-Peregrine-7-Q-A/Do-these-shoes-run-narrow-normal-or-
    "style" : "qanda",
    "thread_style" : "qanda",
    "messages_count" : 4,
    "solved" : false,
    "last_post_time" : "2019-02-20T21:52:02.250-08:00",
    "last_post_time_friendly" : "a month ago"
  },
  "topic" : {
    "type" : "message",
    "id" : "987",
    "href" : "/messages/987",
    "view_href" : "/t5/Saucony-Peregrine-7-Q-A/Do-these-shoes-run-narrow-normal-or-
```


Next, let's query the number of unanswered posts (topic messages with no replies):

```
SELECT count(*) FROM messages WHERE replies.count(*)=0 and depth=0
```

> 👍 In this example, we used a WHERE clause to limit the message object returned in the result set to topic messages (signified by 'depth=0' in the query). The WHERE clause can

Using LiQL

Learn about our LiQL query language used with GET requests to Community API v2. This guide describes the LiQL syntax, using LiQL for combinatorial and keyword searches, pagination with LiQL queries, and sorting query results

```
    "metadata" : { }
}
```

Finally, we'll query the `id`, `subject`, and `view_href` fields of unsolved forum posts ordered by the last post time

```
SELECT id, subject, view_href FROM messages WHERE conversation.style = 'forum' AND dept
```

> 👍 Notice the ORDER BY statement in that last query. We recommend using an ORDER BY clause when querying collections where fields in the ORDER BY clause are supported. This is because results from LiQL queries without ORDER BY are ordered in an inconsistent, arbitrary order. This means that every REST call will be ordered differently. To learn more, see the Best practices section of Using LiQL.

If you have posts with the conversation style of forum, you will see a response similar to this:

```
SELECT id, subject, view_href FROM messages WHERE conversation.style = 'forum' AND depth = 0 ORDER BY conversation.l
```


```
{
    "status":"success",
    "message":"",
    "http_code":200,
    "data": {
         "type" : "messages",
         "list_item_type" : "message",
         "size" : 22,
           "items": [  {
              "type":"message",
              "id":"62",
```

Using LiQL

Learn about our LiQL query language used with GET requests to Community API v2. This guide describes the LiQL syntax, using LiQL for combinatorial and keyword searches, pagination with LiQL queries, and sorting query results

Best practices

Avoid using SELECT * in SELECT statements within code

For performance reasons, we recommend against using an asterisk (*) *in a SELECT statement in your code. Instead, we recommend making your queries as specific as possible, especially for collections, like the Messages collection, that contain many fields. Using SELECT* is safe in the Studio API Browser, however, and can be quite useful to understand the data structure.

Use an ORDER BY clause in your queries

Use an ORDER BY clause when querying collections where fields in the ORDER BY clause are supported

Results from LiQL queries without ORDER BY are ordered in an inconsistent, arbitrary order. This means that every REST call will be ordered differently. This has an impact when using OFFSET and CURSOR for paginated results. Without an ORDER BY clause, you will see duplication in paginated lists because each set will be ordered differently and your offset or cursor could hit the same item multiple times.

Sorting with ORDER BY is supported by the following collections.

- boards
- bookmarks
- categories
- floated_messages
- images
- inbox_notes

Using LiQL

Learn about our LiQL query language used with GET requests to Community API v2. This guide describes the LiQL syntax, using LiQL for combinatorial and keyword searches, pagination with LiQL queries, and sorting query results

- boards
- categories
- grouphubs (currently in Early Access)

The SELECT statement

Syntax: SELECT field(s) FROM collection

Example

Return the id and subject fields for all Message resources in the messages collection.

```
SELECT id, subject FROM messages
```

A SELECT statement defines which fields to retrieve with your query. A SELECT statement is required for all LiQL queries. Use a comma-separated list to select multiple fields. Selecting a subset of fields in a SELECT statement returns a partial response. Use an asterisk (*) to return all fields on a resource (e.g., a full response).

> 👍 **Best Practice**
>
> We strongly recommend using SELECT * only in the API Browser in Studio. See the Best practices section of this guide for more details.

The WHERE clause

Syntax: WHERE

Using LiQL

Learn about our LiQL query language used with GET requests to Community API v2. This guide describes the LiQL syntax, using LiQL for combinatorial and keyword searches, pagination with LiQL queries, and sorting query results

with that.

- Supported logical operators: AND, OR, IN( ), MATCHES, LIKE. As above, implementation is limited and specified in each field description, usually, but it can't always be counted on to be accurate.
- Here is a table of fields and which operators are available on them
  [TABLE]
  Field Name Operators Available Logical Operators
  message = AND
  user = AND, LIKE*
  ETC...

> 📘 *The LIKE operator is supported only with the `users` collection `login` constraint.

> 📘 OR operator support is limited to keyword searches using MATCHES. See Keyword searches in a LiQL WHERE clause for more details.

Constraints are evaluated and combined together in order from left to right. You cannot group constraints within parenthesis.

The MATCHES operator enables you to create keyword and combinatorial searches to mimic the search in the Community UI. We describe these options in Keyword searches in a LiQL WHERE clause and Combinatorial searches with LiQL.

Working with the LIKE operator

Using LiQL

Learn about our LiQL query language used with GET requests to Community API v2. This guide describes the LiQL syntax, using LiQL for combinatorial and keyword searches, pagination with LiQL queries, and sorting query results

- You may include a % sign as a non-wildcard by escaping it. Escape a % by including two percent signs next to each other (%%) in your search string.

> 👍 If you use LIKE in a query that also uses the `following.id` or `followers.id` constraints in the WHERE clause, then your wildcard characters must be before and after the rest of the LIKE string.

These queries are valid when also using `followers.id` or `following.id` in the WHERE clause:

```
SELECT id, login FROM users WHERE following.id = '2' AND login LIKE 'doug%'
SELECT id, login FROM users WHERE followers.id = '4' AND login LIKE '%suz%'
SELECT id, login FROM users WHERE following.id = '5' AND login LIKE 'john%'
```

These are *invalid* uses of LIKE:

- % is used in the middle of the LIKE string
  `SELECT id, login FROM users WHERE following.id = '2' AND login LIKE 'do%g%'`
- LIKE string includes only two non-wildcard characters. Three minimum are required.
  `SELECT id, login FROM users WHERE followers.id = '4' AND login LIKE '%su%'`

The ORDER BY clause

Syntax: ORDER BY <field(s)>
Example
Order results by the date/time posted in descending order
ORDER BY post_time DESC

Using LiQL

Learn about our LiQL query language used with GET requests to Community API v2. This guide describes the LiQL syntax, using LiQL for combinatorial and keyword searches, pagination with LiQL queries, and sorting query results

Default sort orders in ORDER BY

As of Community release 19.10, if an ORDER BY block doesn't exist, we apply default sort orders to queries to the following collections:

- messages
- users
- nodes
- boards
- categories
- grouphubs (currently in Early Access)

Collection	Default Sort Order
The default sort order for queries to the `messages` collection	ORDER BY post_time DESC, id DESC
Default sort order for a query to the `users` collection	ORDER BY registration_data.registration_time DESC, id DESC

Default sort order for queries to:
`nodes`
`boards`
`categories`
`group_hubs`
	ORDER BY depth ASC, position ASC

LIMIT

Using LiQL

Learn about our LiQL query language used with GET requests to Community API v2. This guide describes the LiQL syntax, using LiQL for combinatorial and keyword searches, pagination with LiQL queries, and sorting query results

recommended pagination method. Support for cursors was added in Community release 19.9.

OFFSET

OFFSET is optional and is used in conjunction with LIMIT for pagination. See the Pagination with LIMIT and OFFSET section of *Pagination with Community API v2* for details.

Updated 7 months ago

← Pagination with Community API v2

Date ranges in the WHERE clause →

Using LiQL

Learn about our LiQL query language used with GET requests to Community API v2. This guide describes the LiQL syntax, using LiQL for combinatorial and keyword searches, pagination with LiQL queries, and sorting query results