Self-querying with MyScale

> MyScale is an integrated vector database. You can access your database in SQL and also from here, LangChain. MyScale can make a use of various data types and functions for filters. It will boost up your LLM app no matter if you are scaling up your data or expand your system to broader application.

In the notebook we'll demo the `SelfQueryRetriever` wrapped around a MyScale vector store with some extra piece we contributed to LangChain. In short, it can be concluded into 4 points:

1. Add `contain` comparator to match list of any if there is more than one element matched
2. Add `timestamp` data type for datetime match (ISO-format, or YYYY-MM-DD)
3. Add `like` comparator for string pattern search
4. Add arbitrary function capability

Creating a MyScale vectorstore

MyScale has already been integrated to LangChain for a while. So you can follow this notebook to create your own vectorstore for a self-query retriever.

NOTE: All self-query retrievers requires you to have `lark` installed (`pip install lark`). We use `lark` for grammar definition. Before you proceed to the next step, we also want to remind you that `clickhouse-connect` is also needed to interact with your MyScale backend.

```
pip install lark clickhouse-connect
```

In this tutorial we follow other example's setting and use `OpenAIEmbeddings`. Remember to get a OpenAI API Key for valid accesss to LLMs.

```
import os
import getpass
```

```
os.environ["OPENAI_API_KEY"] = getpass.getpass("OpenAI API Key:")
os.environ["MYSCALE_HOST"] = getpass.getpass("MyScale URL:")
os.environ["MYSCALE_PORT"] = getpass.getpass("MyScale Port:")
os.environ["MYSCALE_USERNAME"] = getpass.getpass("MyScale Username:")
os.environ["MYSCALE_PASSWORD"] = getpass.getpass("MyScale Password:")
```

```
from langchain.schema import Document
from langchain.embeddings.openai import OpenAIEmbeddings
from langchain.vectorstores import MyScale

embeddings = OpenAIEmbeddings()
```

API Reference:

- Document from `langchain.schema`
- OpenAIEmbeddings from `langchain.embeddings.openai`
- MyScale from `langchain.vectorstores`

Create some sample data

As you can see, the data we created has some difference to other self-query retrievers. We replaced keyword `year` to `date` which gives you a finer control on timestamps. We also altered the type of keyword `gerne` to list of strings, where LLM can use a new `contain` comparator to construct filters. We also provides comparator `like` and arbitrary function support to filters, which will be introduced in next few cells.

Now let's look at the data first.

```
docs = [
    Document(
        page_content="A bunch of scientists bring back dinosaurs and
mayhem breaks loose",
        metadata={"date": "1993-07-02", "rating": 7.7, "genre": ["science
fiction"]},
    ),
    Document(
        page_content="Leo DiCaprio gets lost in a dream within a dream
within a dream within a ...",
```

```
        metadata={"date": "2010-12-30", "director": "Christopher Nolan",
"rating": 8.2},
    ),
    Document(
        page_content="A psychologist / detective gets lost in a series of
dreams within dreams within dreams and Inception reused the idea",
        metadata={"date": "2006-04-23", "director": "Satoshi Kon",
"rating": 8.6},
    ),
    Document(
        page_content="A bunch of normal-sized women are supremely
wholesome and some men pine after them",
        metadata={"date": "2019-08-22", "director": "Greta Gerwig",
"rating": 8.3},
    ),
    Document(
        page_content="Toys come alive and have a blast doing so",
        metadata={"date": "1995-02-11", "genre": ["animated"]},
    ),
    Document(
        page_content="Three men walk into the Zone, three men walk out of
the Zone",
        metadata={
            "date": "1979-09-10",
            "rating": 9.9,
            "director": "Andrei Tarkovsky",
            "genre": ["science fiction", "adventure"],
            "rating": 9.9,
        },
    ),
]
vectorstore = MyScale.from_documents(
    docs,
    embeddings,
)
```

Creating our self-querying retriever

Just like other retrievers... Simple and nice.

```
from langchain.llms import OpenAI
from langchain.retrievers.self_query.base import SelfQueryRetriever
from langchain.chains.query_constructor.base import AttributeInfo

metadata_field_info = [
    AttributeInfo(
        name="genre",
        description="The genres of the movie",
        type="list[string]",
    ),
    # If you want to include length of a list, just define it as a new
column
    # This will teach the LLM to use it as a column when constructing
filter.
    AttributeInfo(
        name="length(genre)",
        description="The length of genres of the movie",
        type="integer",
    ),
    # Now you can define a column as timestamp. By simply set the type to
timestamp.
    AttributeInfo(
        name="date",
        description="The date the movie was released",
        type="timestamp",
    ),
    AttributeInfo(
        name="director",
        description="The name of the movie director",
        type="string",
    ),
    AttributeInfo(
        name="rating", description="A 1-10 rating for the movie",
type="float"
    ),
]
document_content_description = "Brief summary of a movie"
llm = OpenAI(temperature=0)
retriever = SelfQueryRetriever.from_llm(
    llm, vectorstore, document_content_description, metadata_field_info,
verbose=True
)
```

API Reference:

- OpenAI from `langchain.llms`
- SelfQueryRetriever from `langchain.retrievers.self_query.base`
- AttributeInfo from `langchain.chains.query_constructor.base`

Testing it out with self-query retriever's existing functionalities

And now we can try actually using our retriever!

```
# This example only specifies a relevant query
retriever.get_relevant_documents("What are some movies about dinosaurs")
```

```
# This example only specifies a filter
retriever.get_relevant_documents("I want to watch a movie rated higher
than 8.5")
```

```
# This example specifies a query and a filter
retriever.get_relevant_documents("Has Greta Gerwig directed any movies
about women")
```

```
# This example specifies a composite filter
retriever.get_relevant_documents(
    "What's a highly rated (above 8.5) science fiction film?"
)
```

```
# This example specifies a query and composite filter
retriever.get_relevant_documents(
    "What's a movie after 1990 but before 2005 that's all about toys, and
preferably is animated"
)
```

Wait a second... What else?

Self-query retriever with MyScale can do more! Let's find out.

```
# You can use length(genres) to do anything you want
retriever.get_relevant_documents("What's a movie that have more than 1
genres?")
```

```
# Fine-grained datetime? You got it already.
retriever.get_relevant_documents("What's a movie that release after feb
1995?")
```

```
# Don't know what your exact filter should be? Use string pattern match!
retriever.get_relevant_documents("What's a movie whose name is like
Andrei?")
```

```
# Contain works for lists: so you can match a list with contain
comparator!
retriever.get_relevant_documents(
    "What's a movie who has genres science fiction and adventure?"
)
```

Filter k

We can also use the self query retriever to specify `k`: the number of documents to fetch.

We can do this by passing `enable_limit=True` to the constructor.

```
retriever = SelfQueryRetriever.from_llm(
    llm,
    vectorstore,
    document_content_description,
    metadata_field_info,
    enable_limit=True,
```

```
    verbose=True,
)
```

```
# This example only specifies a relevant query
retriever.get_relevant_documents("what are two movies about dinosaurs")
```