# Access to Chemical Database - Mcul

In this example, we show how the Mcul chemical database can be accessed using a Python script in a MatDeck document. The queries are prepared in json format according to the Mcul API format, and sent using the http post in Python. The details about the Mcul API can be found at https://doc.mcule.com/doku.php?id=ultimate-api#query_types. In order to test a selected query type, the appropriate Python code should be uncommented. The results obtained from the Mcul database are in json format, which is printed in the system console.

## Exact Searching

Exact searches allow you to specify multiple queries. Here is a Python code which is used to specify three different queries using the SMILES query type. It is recommended to use the SMILES queries and to send the queries in SMILES format.An example of a SMILES string is:

C(N1C(C(OC2CCCN(C3=CC=CC=C3)C2)=O)CN(C)C1=O)(=O)C(=C)CN1CCOCC1

The idx value in the response corresponds to the index of the query in the queries list. As you can see from the above response, the first and the third query resulted in a hit, while the second query did not give any hits.

```
#py
import requests

url = 'https://ultimateapp.mcule.com/api/v1/searches/'
myjson = {
    "query": {
        "type": "exact",
        "queries": [
            "C(N1C(C(OC2CCCN(C3=CC=CC=C3)C2)=O)CN(C)C1=O)(=O)C(=C)CN1CCOCC1",
            "O(C1=CN(N=C1)C)CC1C=CC2C=CC=CC=2C=1",
            "C(N1C(C(OC2CCCN(C3=CC=CC=C3)C2)=O)CN(C)C1=O)(=O)C1CC2=CC=CC(OC)=C2SC1"
        ]
    }
}

x = requests.post(url, json = myjson)

#print the response text (the content of the requested file):

print(x.text)
###
```

## Similarity Searches

Similarity and substructure searches allow you to specify only one query at a time for now. If you have more queries you have to send them separately. Be aware of the rate limits in this case. Also note that similarity and substructure searches can take more time and the response time can depend on many things.

```
#py
import requests

url = 'https://ultimateapp.mcule.com/api/v1/searches/'
myjson = {
    "query": {
        "type": "sim",
        "query": "C(N1C(C(OC2CCCN(C3=CC=CC=C3)C2)=O)CN(C)C1=O)
(=O)C(=C)CN1CCOCC1",
        "limit": 5
    }
}
x = requests.post(url, json = myjson)

#print the response text (the content of the requested file):

print(x.text)
###
```

In the case of similarity searches the tan value and the response contain the similarity value. With similarity searches you can control the minimum similarity with the sim_threshold param. Its default value is 0.7. Its minimum value is also 0.7 unless you are allowed to use a lower value. For example:

```
#py
import requests

url = 'https://ultimateapp.mcule.com/api/v1/searches/'
myjson = {
    "query": {
        "type": "sim",
        "query": "C(N1C(C(OC2CCCN(C3=CC=CC=C3)C2)=O)CN(C)C1=O)
(=O)C(=C)CN1CCOCC1",
        "limit": 5,
        "sim_threshold": 0.8
    }
}
x = requests.post(url, json = myjson)

#print the response text (the content of the requested file):

print(x.text)
###
```

## Substructure Search

A substructure search is similar to a similarity search.  In order to use substructure search the value sss is used in the type parameter.  The number of hits can be  can limited with the limit parameter. In this example we fetched a maximum of 5 hits. The maximum allowed number limit is 1000.

```
#py
import requests

url = 'https://ultimateapp.mcule.com/api/v1/searches/'
myjson = {
    "query": {
        "type": "sss",
        "query": "C1=CC2=C(C=C1)C=NC=C2",
        "limit": 5
    }
}
x = requests.post(url, json = myjson)

#print the response text (the content of the requested file):

print(x.text)
###
```