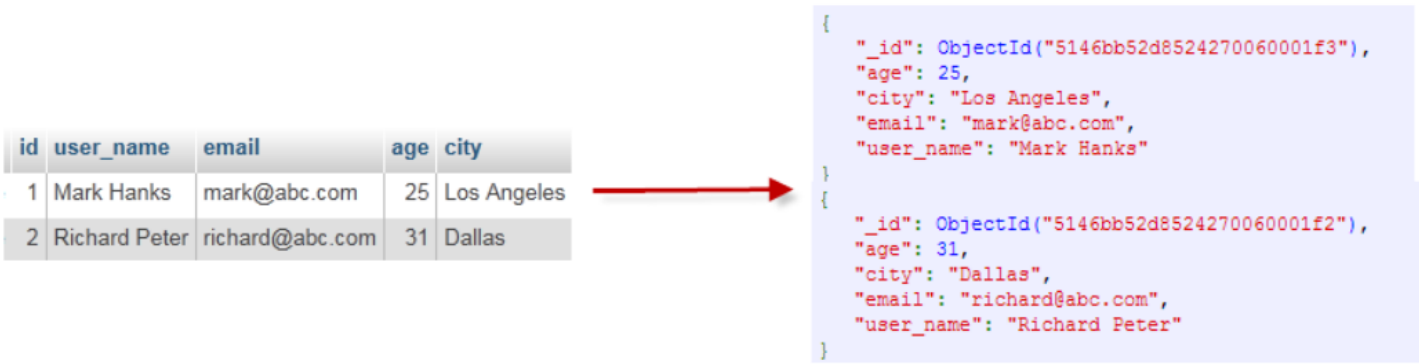
# MongoDB使用

## 1.MongoDB概念

| SQL 术语/概念 | MongoDB 术语/概念 | 解释/说明 |
| --- | --- | --- |
| database | database | 数据库 |
| table | collection | 数据库表/集合 |
| row | document | 数据记录行/文档 |
| column | field | 数据字段/域 |
| index | index | 索引 |
| table joins | | 表连接,MongoDB不支持 |
| primary key | primary key | 主键,MongoDB自动将_id字段设置为主键 |

通过下图实例，我们也可以更直观的了解Mongo中的一些概念：




## 2.MongoDB使用

### 1.安装相关库

```
pip install pymongo
# 清华镜像库
pip install pymongo -i https://pypi.tuna.tsinghua.edu.cn/simple --trusted-host pypi.tuna.tsinghua.edu.cn
```

## 2.连接MongoDB

```
import pymongo

# 本地连接
url = "mongodb://localhost:27017/"
# 连接到本地 MongoDB 实例，指定数据库
url = "mongodb://localhost/mydatabase"
# 使用用户名和密码连接到本地 MongoDB 实例
url = "mongodb://username:password@localhost/mydatabase"
# 连接到远程 MongoDB 实例
url = "mongodb://remotehost:27017"
# 使用 SSL 连接到 MongoDB
url = "mongodb://username:password@localhost:27017/mydatabase?ssl=true"
# 使用多个选项连接
url = "mongodb://username:password@localhost:27017/mydatabase?
authSource=admin&ssl=true"

myclient = pymongo.MongoClient(url)
```

## 3.数据库相关操作

### 1.创建数据库

在 MongoDB 中，数据库只有在内容插入后才会创建! 就是说，数据库创建后要创建集合(数据表)并插入一个文档(记录)，数据库才会真正创建。

```
mydb = myclient["runoobdb"] # ""内为数据库名字

# 判断数据库是存在
dblist = myclient.list_database_names() # 获取所有的数据库名字
if "runoobdb" in dblist:
    print("数据库已存在! ")
```

## 4.集合（表）相关操作

### 1.创建

```
mydb = myclient["runoobdb"]
mycol = mydb["sites"]

```

```python
# 判断集合是否存在
collist = mydb.list_collection_names()
if "sites" in collist:
    print("集合已存在! ")
```

## 2.写入数据

### 1.插入

#### insert_one()插入单条数据

```python
# 插入单条数据
mydict = { "name": "RUNOOB", "alexa": "10000", "url": "https://www.runoob.com" }
x = mycol.insert_one(mydict)
# insert_one() 方法返回 InsertOneResult 对象，该对象包含 inserted_id 属性，它是插入文档的 id 值
print(x.inserted_id)
# 如果我们在插入文档时没有指定 _id，MongoDB 会为每个文档添加一个唯一的 id
```

| _id | name | alexa | url |
|---|---|---|---|
| 668a42cde4b836d1802fc7a8 | RUNOOB | 10000 | https://www.runoob.com |
| 668a430d8d93e863aa73275a | Google | 1 | https://www.google.com |

#### insert_many()插入多条数据

```python
mylist = [
  { "name": "Taobao", "alexa": "100", "url": "https://www.taobao.com" },
  { "name": "QQ", "alexa": "101", "url": "https://www.qq.com" },
  { "name": "Facebook", "alexa": "10", "url": "https://www.facebook.com" },
  { "name": "知乎", "alexa": "103", "url": "https://www.zhihu.com" },
  { "name": "Github", "alexa": "109", "url": "https://www.github.com" }
]
# 指定id插入
mylist = [
  { "_id": 1, "name": "RUNOOB", "cn_name": "菜鸟教程"},
  { "_id": 2, "name": "Google", "address": "Google 搜索"},
  { "_id": 3, "name": "Facebook", "address": "脸书"},
  { "_id": 4, "name": "Taobao", "address": "淘宝"},
  { "_id": 5, "name": "Zhihu", "address": "知乎"}
]
# 插入多条数据
x = mycol.insert_many(mylist)
```

```
18 # 也可以看到不同的字段会自动置空
```

| _id | name | alexa | url | cn_name | address |
|---|---|---|---|---|---|
| 668a42cde4b836d1802fc7a8 | RUNOOB | 10000 | https://www.runoob.com | (N/A) | (N/A) |
| 668a430d8d93e863aa73275a | Google | 1 | https://www.google.com | (N/A) | (N/A) |
| 668aa175af6d7a96bac77997 | Taobao | 100 | https://www.taobao.com | (N/A) | (N/A) |
| 668aa175af6d7a96bac77998 | QQ | 101 | https://www.qq.com | (N/A) | (N/A) |
| 668aa175af6d7a96bac77999 | Facebook | 10 | https://www.facebook.com | (N/A) | (N/A) |
| 668aa175af6d7a96bac7799a | 知乎 | 103 | https://www.zhihu.com | (N/A) | (N/A) |
| 668aa175af6d7a96bac7799b | Github | 109 | https://www.github.com | (N/A) | (N/A) |
| 1 | RUNOOB | (N/A) | (N/A) | 菜鸟教程 | (N/A) |
| 2 | Google | (N/A) | (N/A) | (N/A) | Google 搜索 |
| 3 | Facebook | (N/A) | (N/A) | (N/A) | 脸书 |
| 4 | Taobao | (N/A) | (N/A) | (N/A) | 淘宝 |
| 5 | Zhihu | (N/A) | (N/A) | (N/A) | 知乎 |

## 2.更新

### update_one()更新指定条件下的第一条数据

其中的参数myquery为通用的查询筛选器，用于查找到需要的文档

```
1 myquery = { "alexa": "10000" }
2 newvalues = { "$set": { "alexa": "12345" } }
3 # 为了证实只更改一个数据，将其中一个数据的alexa字段修改为10000
4 mycol.update_one(myquery, newvalues)
```

| _id | name | alexa | url |
|---|---|---|---|
| 668a42cde4b836d1802fc7a8 | RUNOOB | 10000 | https://www. |
| 668a430d8d93e863aa73275a | Google | 10000 | https://www. |
| 668aa175af6d7a96bac77997 | Taobao | 100 | https://www. |
| 668aa175af6d7a96bac77998 | QQ | 101 | https://www. |
| 668aa175af6d7a96bac77999 | Facebook | 10 | https://www. |
| 668aa175af6d7a96bac7799a | 知乎 | 103 | https://www. |
| 668aa175af6d7a96bac7799b | Github | 109 | https://www. |

| _id | name | alexa | url |
|---|---|---|---|
| 668a42cde4b836d1802fc7a8 | RUNOOB | 12345 | https://\ |
| 668a430d8d93e863aa73275a | Google | 10000 | https://\ |
| 668aa175af6d7a96bac77997 | Taobao | 100 | https://\ |
| 668aa175af6d7a96bac77998 | QQ | 101 | https://\ |
| 668aa175af6d7a96bac77999 | Facebook | 10 | https://\ |
| 668aa175af6d7a96bac7799a | 知乎 | 103 | https://\ |
| 668aa175af6d7a96bac7799b | Github | 109 | https://\ |

### update_many()更新指定条件下的所有数据

```
1 myquery = { "name": { "$regex": "^G" } } # 正则表达式匹配开头为G的字段
2 newvalues = { "$set": { "alexa": "123" } }
3
4 x = mycol.update_many(myquery, newvalues)
```

| _id | name | alexa | ur |
|---|---|---|---|
| 668a42cde4b836d1802fc7a8 | RUNOOB | 12345 | ht |
| 668a430d8d93e863aa73275a | Google | 12345 | ht |
| 668aa175af6d7a96bac77997 | Taobao | 100 | ht |
| 668aa175af6d7a96bac77998 | QQ | 101 | ht |
| 668aa175af6d7a96bac77999 | Facebook | 10 | ht |
| 668aa175af6d7a96bac7799a | 知乎 | 103 | ht |
| 668aa175af6d7a96bac7799b | Github | 12345 | ht |
| 1 | RUNOOB | (N/A) | (N |
| 2 | Google | 12345 | (N |

### 3.替换

replace_one()替换指定条件下的第一条数据

```
myquery = { "name": { "$regex" : "^G"} }
newdocument = { "name" : "Xhs",
                "alexa" : "111",
                 "url": "https://www.xiaohongshu.com" }

mycol.replace_one(myquery, newdocument)
# 这里的_id值是没有变化的
```

| _id | name | alexa | url | cn_name | address |
|---|---|---|---|---|---|
| 668a42cde4b836d1802fc7a8 | RUNOOB | 12345 | https://www.runoob.com | (N/A) | (N/A) |
| 668a430d8d93e863aa73275a | Xhs | 111 | https://www.xiaohongshu.com | (N/A) | (N/A) |
| 668aa175af6d7a96bac77997 | Taobao | 100 | https://www.taobao.com | (N/A) | (N/A) |
| 668aa175af6d7a96bac77998 | QQ | 101 | https://www.qq.com | (N/A) | (N/A) |

### 4.删除

delete_one()删除查询条件下单条数据

delete_many()删除查询条件下所有数据

这里就不做演示了

```
myquery = { "name": "Xhs" }

```

```
3 mycol.delete_one(myquery)
4 mycol.delete_many(myquery)
```

### 5.批量操作

可以对某个集合使用bulk_write来合并使用上述操作

- insert_one()
- insert_many()
- update_one()
- update_many()
- replace_one()
- delete_one()
- delete_many()

```
1  operations = [
2      pymongo.InsertOne(
3          {
4              "name": "快手",
5              "alexa": "1234",
6              "url": "https://www.kuaishou.com"
7          }
8      ),
9      pymongo.UpdateMany(
10         { "name": "知乎" },
11         { "$set": { "alexa": "101" }},
12     ),
13     pymongo.DeleteOne(
14         { "name": "Xhs" }
15     )
16 ]
17 results = mycol.bulk_write(operations)
```

## 3.读取数据

### 1.查找函数

find_one()查找一条

find()查找全部

```python
print(mycol.find_one())

for x in mycol.find():
    print(x)
```

```
{'_id': ObjectId('668a42cde4b836d1802fc7a8'), 'name': 'RUNOOB', 'alexa': '12345', 'url': 'https://www.runoob.com'}
```

```
{'_id': ObjectId('668a42cde4b836d1802fc7a8'), 'name': 'RUNOOB', 'alexa': '12345', 'url': 'https://www.runoob.com'}
{'_id': ObjectId('668a430d8d93e863aa73275a'), 'name': 'Xhs', 'alexa': '111', 'url': 'https://www.xiaohongshu.com'}
{'_id': ObjectId('668aa175af6d7a96bac77997'), 'name': 'Taobao', 'alexa': '100', 'url': 'https://www.taobao.com'}
{'_id': ObjectId('668aa175af6d7a96bac77998'), 'name': 'QQ', 'alexa': '101', 'url': 'https://www.qq.com'}
{'_id': ObjectId('668aa175af6d7a96bac77999'), 'name': 'Facebook', 'alexa': '10', 'url': 'https://www.facebook.com'}
{'_id': ObjectId('668aa175af6d7a96bac7799a'), 'name': '知乎', 'alexa': '103', 'url': 'https://www.zhihu.com'}
{'_id': ObjectId('668aa175af6d7a96bac7799b'), 'name': 'Github', 'alexa': '12345', 'url': 'https://www.github.com'}
{'_id': 1, 'name': 'RUNOOB', 'cn_name': '菜鸟教程'}
{'_id': 2, 'name': 'Google', 'address': 'Google 搜索', 'alexa': '12345'}
{'_id': 3, 'name': 'Facebook', 'address': '脸书'}
{'_id': 4, 'name': 'Taobao', 'address': '淘宝'}
{'_id': 5, 'name': 'Zhihu', 'address': '知乎'}
```

## 2.指定查询-查询筛选器

### 精确匹配

指定某个字段的某个值

```python
for x in mycol.find({"name" : "Google"}):
# 多个字段在最外侧的{}内保证"key" : value,的格式即可
# for x in mycol.find({"name": "Google", "url": "111"})
    print(x)
```

### 比较操作符

- `$gt`：大于
- `$lte`：小于或等于
- `$ne`：不等于

```python
for x in mycol.find({"alexa": {"$ne" : "12345"}}):
    print(x)
```

```
{'_id': ObjectId('668a430d8d93e863aa73275a'), 'name': 'Xhs', 'alexa': '111', 'url': 'https://www.xiaohongshu.com'}
{'_id': ObjectId('668aa175af6d7a96bac77997'), 'name': 'Taobao', 'alexa': '100', 'url': 'https://www.taobao.com'}
{'_id': ObjectId('668aa175af6d7a96bac77998'), 'name': 'QQ', 'alexa': '101', 'url': 'https://www.qq.com'}
{'_id': ObjectId('668aa175af6d7a96bac77999'), 'name': 'Facebook', 'alexa': '10', 'url': 'https://www.facebook.com'}
{'_id': ObjectId('668aa175af6d7a96bac7799a'), 'name': '知乎', 'alexa': '103', 'url': 'https://www.zhihu.com'}
{'_id': 1, 'name': 'RUNOOB', 'cn_name': '菜鸟教程'}
{'_id': 3, 'name': 'Facebook', 'address': '脸书'}
{'_id': 4, 'name': 'Taobao', 'address': '淘宝'}
{'_id': 5, 'name': 'Zhihu', 'address': '知乎'}
```

逻辑操作符

- `$and` ，返回符合*所有*子句条件的所有文档
- `$or` ，返回符合一个子句条件的所有文档
- `$nor` ，返回所有*不*符合任何子句条件的文档
- `$not` ，返回与表达式*不*匹配的所有文档

```python
# 返回name字段首字母为T或alexa字段值不等于12345的文档
for x in mycol.find({
    "$or" : [
        {"name" : {"$regex" : "^T"}},
        {"alexa" : {"$ne" : "12345"}}
    ]
}):
    print(x)
```

```
{'_id': ObjectId('668a430d8d93e863aa73275a'), 'name': 'Xhs', 'alexa': '111', 'url': 'https://www.xiaohongshu.com'}
{'_id': ObjectId('668aa175af6d7a96bac77997'), 'name': 'Taobao', 'alexa': '100', 'url': 'https://www.taobao.com'}
{'_id': ObjectId('668aa175af6d7a96bac77998'), 'name': 'QQ', 'alexa': '101', 'url': 'https://www.qq.com'}
{'_id': ObjectId('668aa175af6d7a96bac77999'), 'name': 'Facebook', 'alexa': '10', 'url': 'https://www.facebook.com'}
{'_id': ObjectId('668aa175af6d7a96bac7799a'), 'name': '知乎', 'alexa': '103', 'url': 'https://www.zhihu.com'}
{'_id': 1, 'name': 'RUNOOB', 'cn_name': '菜鸟教程'}
{'_id': 3, 'name': 'Facebook', 'address': '脸书'}
{'_id': 4, 'name': 'Taobao', 'address': '淘宝'}
{'_id': 5, 'name': 'Zhihu', 'address': '知乎'}
```

### 数组操作符

- `$all` ，它会返回带有数组的文档，而该数组包含查询中的所有元素
- `$elemMatch` ，如果数组字段中的元素与查询中的所有条件匹配，则返回文档
- `$size` ，返回包含指定大小数组的所有文档

### 元素操作符

- `$exists` ，匹配具有指定字段的文档
- `$type` ，如果字段为指定类型，则选择文档

### 评估操作符

求值操作符根据对单个字段或整个文集文件的求值结果返回数据

- `$text` ，对文档执行文本Atlas Search
- `$regex` ，返回与指定正则表达式匹配的文档
- `$mod` ，它对字段的值执行模运算并返回余数为指定值的文档

## 3.指定要返回的字段

指定要包含的字段，默认情况包含_id字段

```python
# { "<Field Name>": 1 }
for x in mycol.find({}, { "name" : 1, "url" : 1}):
    print(x)
```

```
PS D:\study\grade6\shixi\social> python -u "d:\study\grade6\shixi\social\demo.py"
{'_id': ObjectId('668a42cde4b836d1802fc7a8'), 'name': 'RUNOOB', 'url': 'https://www.runoob.com'}
{'_id': ObjectId('668a430d8d93e863aa73275a'), 'name': 'Xhs', 'url': 'https://www.xiaohongshu.com'}
{'_id': ObjectId('668aa175af6d7a96bac77997'), 'name': 'Taobao', 'url': 'https://www.taobao.com'}
{'_id': ObjectId('668aa175af6d7a96bac77998'), 'name': 'QQ', 'url': 'https://www.qq.com'}
{'_id': ObjectId('668aa175af6d7a96bac77999'), 'name': 'Facebook', 'url': 'https://www.facebook.com'}
{'_id': ObjectId('668aa175af6d7a96bac7799a'), 'name': '知乎', 'url': 'https://www.zhihu.com'}
{'_id': ObjectId('668aa175af6d7a96bac7799b'), 'name': 'Github', 'url': 'https://www.github.com'}
```

指定要排除的字段

```python
# { "<Field Name>": 0 }
for x in mycol.find({}, { "url" : 0}):
    print(x)
```

```
{'_id': ObjectId('668a42cde4b836d1802fc7a8'), 'name': 'RUNOOB', 'alexa': '12345'}
{'_id': ObjectId('668a430d8d93e863aa73275a'), 'name': 'Xhs', 'alexa': '111'}
{'_id': ObjectId('668aa175af6d7a96bac77997'), 'name': 'Taobao', 'alexa': '100'}
{'_id': ObjectId('668aa175af6d7a96bac77998'), 'name': 'QQ', 'alexa': '101'}
{'_id': ObjectId('668aa175af6d7a96bac77999'), 'name': 'Facebook', 'alexa': '10'}
{'_id': ObjectId('668aa175af6d7a96bac7799a'), 'name': '知乎', 'alexa': '103'}
{'_id': ObjectId('668aa175af6d7a96bac7799b'), 'name': 'Github', 'alexa': '12345'}
```

除了 _id，你不能在一个对象中同时指定 0 和 1，如果你设置了一个字段为 0，则其他都为 1，反之亦然。

```python
for x in mycol.find({}, { "_id" : 0, "name" : 1, "url" : 1}):
    print(x)
# 下面这种写法会报错
# for x in mycol.find({}, { "name" : 0, "url" : 1}):
```


## 4.指定要返回的文档

### 读取操作返回的最大文档数limit

```python
# 第一种
for x in mycol.find().limit(3):
    print(x)
# 第二种
for x in mycol.find(limit=3):
    print(x)
```

### 文档排序sort

需要保证是同种类型数据（如汉字和英文排序会报错）

```python
# 默认升序
for x in mycol.find().sort("alexa"):
    print(x)
# 降序
for x in mycol.find().sort("alexa", pymongo.DESCENDING):
    print(x)
# 第二种方式
for x in mycol.find(sort={"alexa", pymongo.ASCENDING}):
    print(x)
```

### 先跳过某个数量文档，再查询skip

```python
# 跳过前两个文档
for x in mycol.find().skip(2):
    print(x)
# 第二种写法
for x in mycol.find({}, skip=2):
    print(x)
```

这些方法可以组合，但执行顺序为排序，跳过，限制

```python
# 第一种写法
for x in mycol.find().skip(2).sort("alexa", pymongo.DESCENDING).limit(3):
    print(x)
# 第二种写法
for x in mycol.find(skip=2, sort={"alexa", pymongo.DESCENDING}, limit=3):
    print(x)
```

| _id | name | alexa | url | cn_name | address |
|---|---|---|---|---|---|
| 668a42cde4b836d1802fc7a8 | RUNOOB | 12345 | https://www.runoob.com | (N/A) | (N/A) |
| 668a430d8d93e863aa73275a | Xhs | 111 | https://www.xiaohongshu.com | (N/A) | (N/A) |
| 668aa175af6d7a96bac77997 | Taobao | 100 | https://www.taobao.com | (N/A) | (N/A) |
| 668aa175af6d7a96bac77998 | QQ | 101 | https://www.qq.com | (N/A) | (N/A) |
| 668aa175af6d7a96bac77999 | Facebook | 10 | https://www.facebook.com | (N/A) | (N/A) |
| 668aa175af6d7a96bac7799a | 知乎 | 103 | https://www.zhihu.com | (N/A) | (N/A) |
| 668aa175af6d7a96bac7799b | Github | 12345 | https://www.github.com | (N/A) | (N/A) |
| 1 | RUNOOB | (N/A) | (N/A) | 菜鸟教程 | (N/A) |
| 2 | Google | 12345 | (N/A) | (N/A) | Google 搜索 |
| 3 | Facebook | (N/A) | (N/A) | (N/A) | 脸书 |
| 4 | Taobao | (N/A) | (N/A) | (N/A) | 淘宝 |
| 5 | Zhihu | (N/A) | (N/A) | (N/A) | 知乎 |

```
{'_id': 2, 'name': 'Google', 'address': 'Google 搜索', 'alexa': '12345'}
{'_id': ObjectId('668a430d8d93e863aa73275a'), 'name': 'Xhs', 'alexa': '111', 'url': 'https://www.xiaohongshu.com'}
{'_id': ObjectId('668aa175af6d7a96bac7799a'), 'name': '知乎', 'alexa': '103', 'url': 'https://www.zhihu.com'}
```

## 5.多表联合查询

### 一对一的表连接

即A表与B表连接的属性为B表的主键，即A中连接的值在B表中唯一

以以下两个集合为例，用A表orders的product_id和B表products的id作为连接，来获取每个order包含的具体产品信息

| _id | customer_id | orderdate | product_id | value |
|---|---|---|---|---|
| 668fb8e96e7cdd1d91a3dbc7 | elise_smith@myemail.com | 2020-05-30 08:35:52.000 | a1b2c3d4 | 431.43 |
| 668fb8e96e7cdd1d91a3dbc8 | tj@wheresmyemail.com | 2019-05-28 19:13:32.000 | z9y8x7w6 | 5.01 |
| 668fb8e96e7cdd1d91a3dbc9 | oranieri@warmmail.com | 2020-01-01 08:25:37.000 | ff11gg22hh33 | 63.13 |
| 668fb8e96e7cdd1d91a3dbca | jjones@tepidmail.com | 2020-12-26 08:55:46.000 | a1b2c3d4 | 429.65 |

| _id | id | name | category | description |
|---|---|---|---|---|
| 668fb8e96e7cdd1d91a3dbcb | a1b2c3d4 | Asus Laptop | ELECTRONICS | Good value laptop for st |
| 668fb8e96e7cdd1d91a3dbcc | z9y8x7w6 | The Day Of The Triffids | BOOKS | Classic post-apocalyptic |
| 668fb8e96e7cdd1d91a3dbcd | ff11gg22hh33 | Morphy Richardds Food | KITCHENWARE | Luxury mixer turning go |
| 668fb8e96e7cdd1d91a3dbce | pqr678st | Karcher Hose Set | GARDEN | Hose + nosels + winder |

```
orders_coll = mydb["orders"] # 集合A
products_coll = mydb["products"] # 集合B

pipeline = []

pipeline.append({
    "$match": { # match里是之前的查询筛选器,筛选的是集合A
        "orderdate": {
            "$gte": datetime(2020, 1, 1, 0, 0, 0),
            "$lt": datetime(2021, 1, 1, 0, 0, 0)
        }
    }
})

pipeline.append({
    "$lookup": {
        "from": "products", # 需要联合的集合B
        "localField": "product_id", # 集合A的字段
        "foreignField": "id", # 集合B的字段
        "as": "product_mapping"
    }
})
# 等价于以下mysql语句
# SELECT *, product_mapping
# FROM orders
# WHERE product_mapping IN (SELECT *
# FROM products
# WHERE id= orders.product_id);
pipeline.extend([
    {
        # 将product_mapping字段设置为上一个$lookup阶段中创建的product_mapping对象中的第一个元素
        # 因为在B表中值唯一，所以只取1个
        "$set": {
            "product_mapping": {"$first": "$product_mapping"}
        }
    },
    {
        "$set": { # 根据product_mapping对象字段中的值创建两个新字段product_name和product_category
            "product_name": "$product_mapping.name",
            "product_category": "$product_mapping.category"
        }
    }
])
# 从文档中删除不必要的字段
```

```
pipeline.append({"$unset": ["_id", "product_id", "product_mapping"]})

aggregation_result = orders_coll.aggregate(pipeline)
```

```
{
  'customer_id': 'elise_smith@myemail.com',
  'orderdate': datetime.datetime(2020, 5, 30, 8, 35, 52),
  'value': 431.43,
  'product_name': 'Asus Laptop',
  'product_category': 'ELECTRONICS'
}
{
  'customer_id': 'oranieri@warmmail.com',
  'orderdate': datetime.datetime(2020, 1, 1, 8, 25, 37),
  'value': 63.13,
  'product_name': 'Morphy Richardds Food Mixer',
  'product_category': 'KITCHENWARE'
}
{
  'customer_id': 'jjones@tepidmail.com',
  'orderdate': datetime.datetime(2020, 12, 26, 8, 55, 46),
  'value': 429.65,
  'product_name': 'Asus Laptop',
  'product_category': 'ELECTRONICS'
}
```

### 多字段的表连接

以以下两个集合为例，用A表products的variation和name字段与B表orders的product_name和product_variation连接，来获取某个product产生的所有订单

| _id | variation | name | category | description |
|---|---|---|---|---|
| 669092eb91b6f198e1c7ad51 | Ultra HD | Asus Laptop | ELECTRONICS | Great for watching movi |
| 669092eb91b6f198e1c7ad52 | Standard Display | Asus Laptop | ELECTRONICS | Good value laptop for st |
| 669092eb91b6f198e1c7ad53 | 1st Edition | The Day Of The Triffids | BOOKS | Classic post-apocalyptic |
| 669092eb91b6f198e1c7ad54 | 2nd Edition | The Day Of The Triffids | BOOKS | Classic post-apocalyptic |
| 669092eb91b6f198e1c7ad55 | Deluxe | Morphy Richards Food M | KITCHENWARE | Luxury mixer turning go |

| _id | customer_id | orderdate | product_name | product_variation | value |
|---|---|---|---|---|---|
| 669092eb91b6f198e1c7ad56 | elise_smith@myemail.com | 2020-05-30 08:35:52.000 | Asus Laptop | Standard Display | 431.43 |
| 669092eb91b6f198e1c7ad57 | tj@wheresmyemail.com | 2019-05-28 19:13:32.000 | The Day Of The Triffids | 2nd Edition | 5.01 |
| 669092eb91b6f198e1c7ad58 | oranieri@warmmail.com | 2020-01-01 08:25:37.000 | Morphy Richards Food M | Deluxe | 63.13 |
| 669092eb91b6f198e1c7ad59 | jjones@tepidmail.com | 2020-12-26 08:55:46.000 | Asus Laptop | Standard Display | 429.65 |

```
pipeline = []
```

```
# 这里的embedded_pl是使用在lookup中，所以都是对要连接的B表进行处理
embedded_pl = [
    {
        "$match": { # 匹配连接的B表中的连接字段并重命名
            "$expr": {
                "$and": [
                    {"$eq": ["$product_name", "$$prdname"]},
                    {"$eq": ["$product_variation", "$$prdvartn"]}
                ]
            }
        }
    }
]

embedded_pl.append({
    "$match": { # 筛选B表中数据
        "orderdate": {
            "$gte": datetime(2020, 1, 1, 0, 0, 0),
            "$lt": datetime(2021, 1, 1, 0, 0, 0)
        }
    }
})

embedded_pl.append({ # 选择B表中不需要的字段，两表的连接字段会出现两次，所以需要删除一张表中的数据
   "$unset": ["_id", "product_name", "product_variation"]
})

pipeline.append({ # 这里是多字段的匹配处理
    "$lookup": {
        "from": "orders",
        "let": {
            "prdname": "$name",
            "prdvartn": "$variation"
        },
        "pipeline": embedded_pl,
        "as": "orders"
    }
})

pipeline.append({
    "$match": { # 对连接完后的数据进行筛选
        "orders": {"$ne": []}
    }
})

pipeline.append({ # 最后选择需要的字段
```

```
    "$unset": ["_id", "description"]
})

aggregation_result = products_coll.aggregate(pipeline)

for document in aggregation_result:
    print(document)
```

```
{
  'name': 'Asus Laptop',
  'variation': 'Standard Display',
  'category': 'ELECTRONICS',
  'orders': [
    {
      'customer_id': 'elise_smith@myemail.com',
      'orderdate': datetime.datetime(2020, 5, 30, 8, 35, 52),
      'value': 431.43
    },
    {
      'customer_id': 'jjones@tepidmail.com',
      'orderdate': datetime.datetime(2020, 12, 26, 8, 55, 46),
      'value': 429.65
    }
  ]
}
{
  'name': 'Morphy Richards Food Mixer',
  'variation': 'Deluxe',
  'category': 'KITCHENWARE',
  'orders': [
    {
      'customer_id': 'oranieri@warmmail.com',
      'orderdate': datetime.datetime(2020, 1, 1, 8, 25, 37),
      'value': 63.13
    }
  ]
}
```

## 5.GridFS-大数据（图片和视频）的上传与下载

如果文件大小超过16 MB 的 BSON 文档大小限制，则应使用 GridFS

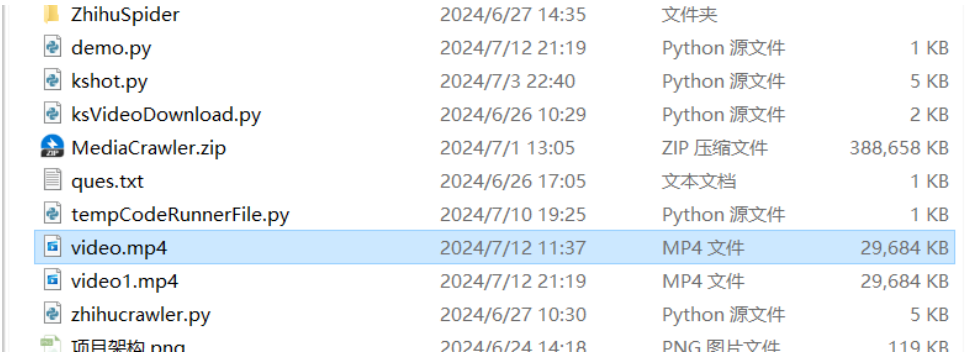
存储桶包含以下集合，使用 GridFS 规范中定义的约定命名：

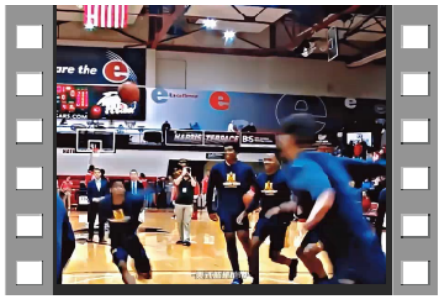
- `chunks` 集合存储二进制文件数据段。

- `files` 集合存储文件元数据。

使用 GridFS 存储文件时，驱动程序会将文件拆分成较小的数据块，每个数据块由 `chunks` 集合中的单独文档表示。它还在 `files` 集合中创建文档，其中包含文件 ID、文件名和其他文件元数据。

这里使用下方大小为28MB的视频作为示例



```python
import pymongo
from gridfs import GridFS

myclient = pymongo.MongoClient('mongodb://localhost:27017/')

mydb = myclient["runoobdb"]
fs = GridFS(mydb) # 创建该数据库下的GridFS桶

myfile = open(file='D:/study/grade6/shixi/social/video.mp4', mode='rb') # 本地上传
mydata = myfile.read()

myfile = fs.put(data=mydata, filename="video.mp4") # 这里的filename命名为GridFS.files中的filename值

print(myfile) # 这里myfile返回的是_id值
```

| _id | filename | chunkSize | length | uploadDate |
|---|---|---|---|---|
| 66912d05479a7ce228a8edc8 | video.mp4 | 261120 | 30396205 | 2024-07-12 13:17:57.68 |

开始事务 文本 筛选 排序 列 全部折叠 类型颜色 导入 导出 分析 数据生成

| _id | files_id | n | data |
|---|---|---|---|
| 66912d05479a7ce228a8edc9 | 66912d05479a7ce228a8edc8 | 0 | (BLOB) 255.00 KB |
| 66912d05479a7ce228a8edca | 66912d05479a7ce228a8edc8 | 1 | (BLOB) 255.00 KB |
| 66912d05479a7ce228a8edcb | 66912d05479a7ce228a8edc8 | 2 | (BLOB) 255.00 KB |
| 66912d05479a7ce228a8edcc | 66912d05479a7ce228a8edc8 | 3 | (BLOB) 255.00 KB |
| 66912d05479a7ce228a8edcd | 66912d05479a7ce228a8edc8 | 4 | (BLOB) 255.00 KB |

| | | | | |
|---|---|---|---|---|
| | 66912d05479a7ce228a8ee38 | 66912d05479a7ce228a8edc8 | 111 | (BLOB) 255.00 KB |
| | 66912d05479a7ce228a8ee39 | 66912d05479a7ce228a8edc8 | 112 | (BLOB) 255.00 KB |
| | 66912d05479a7ce228a8ee3a | 66912d05479a7ce228a8edc8 | 113 | (BLOB) 255.00 KB |
| | 66912d05479a7ce228a8ee3b | 66912d05479a7ce228a8edc8 | 114 | (BLOB) 255.00 KB |
| | 66912d05479a7ce228a8ee3c | 66912d05479a7ce228a8edc8 | 115 | (BLOB) 255.00 KB |
| ▶ | 66912d05479a7ce228a8ee3d | 66912d05479a7ce228a8edc8 | 116 | (BLOB) 103.79 KB |

可以看到fs.files的_id字段值是和fs.chunks的filles_id是对应的，查找的时候也是通过该值来查找的

```
1 mydata = fs.get_version("video.mp4").read()
2 myfile = open(file='D:/study/grade6/shixi/social/video1.mp4', mode='wb')
3 rs = myfile.write(mydata)
```