Lab3 MongoDB 的聚合管道

何为聚合操作

聚合操作主要是通过对数据进行分组后做出一些简单的运算，例如平均，求和，最值等。MongoDB 中的聚合运算主要通过 `aggregate()` 方法实现

Python 中 `aggregate()` 方式实现了利用聚合管道对文档进行变化计算和展示。文档进入聚合管道会依次经过筛选 (filtering)，分组 (grouping) 并聚合，排序 (sorting)，投射 (projecting)，限制 (limiting) (或者跳过 (skipping)) 变化。

符号	含义	描述
$match	筛选	按照一定条件筛选出特定的文档记录
$project	选择	修改输入文档的结构。可以用来重命名、增加或删除域，也可以用于创建计算结果以及嵌套文档
$group	分组	对文档按照字段分组，以便做一些聚合运算
$sort	排序	对文档按照字段排序
$limit	限制	限制MongoDB聚合管道返回的文档数
$skip	跳过	在聚合管道中跳过指定数量的文档，并返回余下的文档

简单的聚合

首先连接数据库创建数据表并且插入数据 用户名:ecnu学号；密码:ECNU学号；数据库名: ecnu学号。

In [14]:
```
# 连接Mongodb数据库
import pymongo

client = pymongo.MongoClient("mongodb://ecnu10222140402:ECNU10222140402@172.16.2
db = client["ecnu10222140402"]
users_col = db["ecnu10222140402"]
```

In [15]:
```
# 插入数据
users_col.delete_many({})
users = [
    {
        "name": "Joe",
        "gender": "m",
        "age": 23,
        "birthdate": {"day": 15, "month": 3, "year": 1997},
        "hobby": ["football", "basketball", "reading"],
        "city": "Beijing",
        "time": [9, 18],
    },
    {
        "name": "Kate",
        "gender": "f",
        "age": 22,
        "birthdate": {"day": 25, "month": 7, "year": 1998},
        "hobby": ["reading", "piano"],
        "city": "Hangzhou",
        "time": [8, 17],
    },
    {
        "name": "Rose",
        "gender": "f",
        "age": 24,
        "birthdate": {"day": 3, "month": 3, "year": 1996},
        "hobby": ["basketball", "running", "traveling"],
        "city": "Shanghai",
        "time": [9, 19],
    },
    {
        "name": "Jason",
        "gender": "m",
        "age": 21,
        "birthdate": {"day": 17, "month": 12, "year": 1999},
        "hobby": ["cooking", "photography"],
        "city": "Chengdu",
        "time": [8, 20],
    },
    {
        "name": "Grace",
        "gender": "f",
        "age": 22,
        "birthdate": {"day": 10, "month": 6, "year": 1998},
        "hobby": ["photography", "cooking", "drama"],
        "city": "Nanjing",
        "time": [9, 18],
    },
    {
        "name": "Jessica",
        "gender": "f",
        "age": 22,
        "birthdate": {"day": 21, "month": 3, "year": 1998},
        "hobby": ["cooking", "piano"],
        "city": "Shanghai",
        "time": [10, 19],
    },
    {
        "name": "Donna",
        "gender": "f",
        "age": 22,
        "birthdate": {"day": 24, "month": 9, "year": 1998},
        "hobby": ["violin", "drama"],
        "city": "Shanghai",
        "time": [9, 20],
    },
    {
        "name": "Apple",
        "gender": "m",
        "age": 23,
        "birthdate": {"day": 20, "month": 9, "year": 1997},
        "hobby": ["violin", "running"],
        "city": "Chengdu",
        "time": [9, 19],
    },
    {
        "name": "Baba",
        "gender": "f",
        "age": 25,
        "birthdate": {"day": 20, "month": 9, "year": 1995},
        "hobby": ["violin", "basketball"],
        "city": "Chengdu",
        "time": [10, 19],
    },
]

users_col.insert_many(users)
content = users_col.find()
for each in content:
    print(each)
```

```
{'_id': ObjectId('66f503a294bc303c9eed1b97'), 'name': 'Joe', 'gender': 'm', 'ag
e': 23, 'birthdate': {'day': 15, 'month': 3, 'year': 1997}, 'hobby': ['football',
'basketball', 'reading'], 'city': 'Beijing', 'time': [9, 18]}
{'_id': ObjectId('66f503a294bc303c9eed1b98'), 'name': 'Kate', 'gender': 'f', 'ag
e': 22, 'birthdate': {'day': 25, 'month': 7, 'year': 1998}, 'hobby': ['reading',
'piano'], 'city': 'Hangzhou', 'time': [8, 17]}
{'_id': ObjectId('66f503a294bc303c9eed1b99'), 'name': 'Rose', 'gender': 'f', 'ag
e': 24, 'birthdate': {'day': 3, 'month': 3, 'year': 1996}, 'hobby': ['basketbal
l', 'running', 'traveling'], 'city': 'Shanghai', 'time': [9, 19]}
{'_id': ObjectId('66f503a294bc303c9eed1b9a'), 'name': 'Jason', 'gender': 'm', 'ag
e': 21, 'birthdate': {'day': 17, 'month': 12, 'year': 1999}, 'hobby': ['cooking',
'photography'], 'city': 'Chengdu', 'time': [8, 20]}
{'_id': ObjectId('66f503a294bc303c9eed1b9b'), 'name': 'Grace', 'gender': 'f', 'ag
e': 22, 'birthdate': {'day': 10, 'month': 6, 'year': 1998}, 'hobby': ['photograph
y', 'cooking', 'drama'], 'city': 'Nanjing', 'time': [9, 18]}
{'_id': ObjectId('66f503a294bc303c9eed1b9c'), 'name': 'Jessica', 'gender': 'f',
'age': 22, 'birthdate': {'day': 21, 'month': 3, 'year': 1998}, 'hobby': ['cookin
g', 'piano'], 'city': 'Shanghai', 'time': [10, 19]}
{'_id': ObjectId('66f503a294bc303c9eed1b9d'), 'name': 'Donna', 'gender': 'f', 'ag
e': 22, 'birthdate': {'day': 24, 'month': 9, 'year': 1998}, 'hobby': ['violin',
'drama'], 'city': 'Shanghai', 'time': [9, 20]}
{'_id': ObjectId('66f503a294bc303c9eed1b9e'), 'name': 'Apple', 'gender': 'm', 'ag
e': 23, 'birthdate': {'day': 20, 'month': 9, 'year': 1997}, 'hobby': ['violin',
'running'], 'city': 'Chengdu', 'time': [9, 19]}
{'_id': ObjectId('66f503a294bc303c9eed1b9f'), 'name': 'Baba', 'gender': 'f', 'ag
e': 25, 'birthdate': {'day': 20, 'month': 9, 'year': 1995}, 'hobby': ['violin',
'basketball'], 'city': 'Chengdu', 'time': [10, 19]}
```

按照城市分组计数

In [16]:
```
result = users_col.aggregate([{"$group": {"_id": "$city", "count": {"$sum": 1}}}
for each in result:
    print(each)
```

```
{'_id': 'Beijing', 'count': 1}
{'_id': 'Hangzhou', 'count': 1}
{'_id': 'Shanghai', 'count': 3}
{'_id': 'Chengdu', 'count': 3}
{'_id': 'Nanjing', 'count': 1}
```

从以上代码中，聚合管道中只有 group 一个操作。在 group 中，可以看到是按照 city 字段进行分组，最后通过"加一"聚合来实现分组计数的。在group中指定了两个字段，第一个是主键 '_id'，来源于 city，第二个是 count，来源于求和。当然也可以根据我们的需要，修改/增减字段。

以下列出了常用的聚集运算

符号	含义
$sum	求和
$avg	求平均
$min	最小值
$max	最大值
$push	聚合成数组
$addToSet	聚合成几何
$first	排序取第一个
$last	排序取最后一个

练习

Task 1 计算不同性别用户的平均年龄，最大年龄，最小年龄并且输出

In [17]:
```
# todo
"""
目标结果
{'_id': 'f', 'avg_age': 22.833333333333332, 'max_age': 25, 'min_age': 22}
{'_id': 'm', 'avg_age': 22.333333333333332, 'max_age': 23, 'min_age': 21}
"""
```

Out[17]:
```
"\n目标结果\n{'_id': 'f', 'avg_age': 22.833333333333332, 'max_age': 25, 'min_ag
e': 22}\n{'_id': 'm', 'avg_age': 22.333333333333332, 'max_age': 23, 'min_age':
21}\n"
```

In [18]:
```
# Task1
# 执行聚合查询
pipeline = [
    {
        "$group": {
            "_id": "$gender",  # 按性别分组
            "avg_age": {"$avg": "$age"},  # 计算平均年龄
            "max_age": {"$max": "$age"},  # 计算最大年龄
            "min_age": {"$min": "$age"}   # 计算最小年龄
        }
    }
]
```

```
# 获取聚合结果
results = users_col.aggregate(pipeline)

# 输出结果
for result in results:
    print(result)
```

```
{'_id': 'f', 'avg_age': 22.833333333333332, 'max_age': 25, 'min_age': 22}
{'_id': 'm', 'avg_age': 22.333333333333332, 'max_age': 23, 'min_age': 21}
```

Task 2 列出不同性别的同学名单 (提示：$push)

In [19]:
```
# todo
"""
目标结果
{'_id': 'm', 'list': ['Joe', 'Jason', 'Apple']}
{'_id': 'f', 'list': ['Kate', 'Rose', 'Grace', 'Jessica', 'Donna', 'Baba']}
"""
```

Out[19]:
```
"\n目标结果\n{'_id': 'm', 'list': ['Joe', 'Jason', 'Apple']}\n{'_id': 'f', 'lis
t': ['Kate', 'Rose', 'Grace', 'Jessica', 'Donna', 'Baba']}\n"
```

In [20]:
```
# Task2
# 聚合查询，按性别分组并收集同学名字
pipeline = [
    {
        "$group": {
            "_id": "$gender",  # 按性别进行分组
            "list": {"$push": "$name"}  # 收集每个性别的学生姓名
        }
    }
]

# 执行聚合查询
result = users_col.aggregate(pipeline)

# 打印结果
for group in result:
    print(group)
```

```
{'_id': 'f', 'list': ['Kate', 'Rose', 'Grace', 'Jessica', 'Donna', 'Baba']}
{'_id': 'm', 'list': ['Joe', 'Jason', 'Apple']}
```

复杂查询

一个完整的查询一般需要经过：

1. 通过条件筛选文档记录（选择文档的行记录）
2. 分组并聚合
3. 对文档按照某些字段排序
4. 调整文档的键值对形式（调整文档的列）
5. 通过limit或者skip展示特定数量的记录

例如：筛选年龄大于等于20岁的同学，并将这些学生的按照城市分组计算平均年龄后升序排列,显示城市和平均年龄

In [21]:
```
match = {"$match": {"age": {"$gte": 20}}}
group = {"$group": {"_id": "$city", "avg_age": {"$avg": "$age"}}}
sort = {"$sort": {"avg_age": 1}}  # 1代表升序，-1代表降序
project = {"$project": {"avg_age": 1}}
result = users_col.aggregate([match, group, sort, project])
for each in result:
    print(each)
```

```
{'_id': 'Hangzhou', 'avg_age': 22.0}
{'_id': 'Nanjing', 'avg_age': 22.0}
{'_id': 'Shanghai', 'avg_age': 22.666666666666668}
{'_id': 'Beijing', 'avg_age': 23.0}
{'_id': 'Chengdu', 'avg_age': 23.0}
```

格式化输出每个同学的生日日期，按照生日日期排序

In [22]:
```
sort = {"$sort": {"birthdate.year": -1, "birthdate.month": -1, "birthdate.day":
project = {
    "$project": {
        "_id": 0,
        "name": 1,
        "birthday": {
            "$concat": [
                {"$toString": "$birthdate.year"},
                "-",
                {"$toString": "$birthdate.month"},
                "-",
                {"$toString": "$birthdate.month"},
            ]
        },
    }
}  # 0代表不显示该字段，1代表显示该字段
result = users_col.aggregate([sort, project])
for each in result:
    print(each)
```

```
{'name': 'Jason', 'birthday': '1999-12-12'}
{'name': 'Donna', 'birthday': '1998-9-9'}
{'name': 'Kate', 'birthday': '1998-7-7'}
{'name': 'Grace', 'birthday': '1998-6-6'}
{'name': 'Jessica', 'birthday': '1998-3-3'}
{'name': 'Apple', 'birthday': '1997-9-9'}
{'name': 'Joe', 'birthday': '1997-3-3'}
{'name': 'Rose', 'birthday': '1996-3-3'}
{'name': 'Baba', 'birthday': '1995-9-9'}
```

以上的'toString'的作用是将数字转换成字符串，更多函数可以参见 https://blog.csdn.net/weixin_43632687/article/details/104201185

按照城市，性别分组计数

In [23]:
```
group = {
    "$group": {"_id": {"city": "$city", "gender": "$gender"}, "count": {"$sum":
}
project = {
    "$project": {
        "_id": 0,
        "city": "$_id.city",
        "gender": "$_id.gender",
        "count": "$count",
    }
}
result = users_col.aggregate([group, project])
for each in result:
    print(each)
```

```
{'city': 'Beijing', 'gender': 'm', 'count': 1}
{'city': 'Hangzhou', 'gender': 'f', 'count': 1}
{'city': 'Shanghai', 'gender': 'f', 'count': 3}
{'city': 'Chengdu', 'gender': 'm', 'count': 2}
{'city': 'Chengdu', 'gender': 'f', 'count': 1}
{'city': 'Nanjing', 'gender': 'f', 'count': 1}
```

以上用到的重命名方法和多字段分组聚合的方法需要好好体会

练习

Task 3 找出喜欢'violin'的人数（提示：$in）

In [24]:
```
# todo
"""
目标结果
{'_id': 'like_violin', 'count': 3}
"""
```

Out[24]:
```
"\n目标结果\n{'_id': 'like_violin', 'count': 3}\n"
```

In [25]:
```
#Task3
# 查询喜欢 'violin' 的人数，使用 $in 操作符
count = users_col.count_documents({"hobby": {"$in": ["violin"]}})
# 输出结果
result = {"_id": "like_violin", "count": count}
print(result)
```

```
{'_id': 'like_violin', 'count': 3}
```

$unwind 拆分数组，查询拥有各个爱好的学生人数

In [26]:
```
# 先通过数组拆分将一条记录拆分成多条记录
unwind = {"$unwind": "$hobby"}
group = {"$group": {"_id": "$hobby", "count": {"$sum": 1}}}
result = users_col.aggregate([unwind, group])
for each in result:
    print(each)
```

```
{'_id': 'piano', 'count': 2}
{'_id': 'running', 'count': 2}
{'_id': 'drama', 'count': 2}
{'_id': 'basketball', 'count': 3}
{'_id': 'cooking', 'count': 3}
{'_id': 'reading', 'count': 2}
{'_id': 'football', 'count': 1}
{'_id': 'violin', 'count': 3}
{'_id': 'photography', 'count': 2}
{'_id': 'traveling', 'count': 1}
```

练习

Task 4 找出爱好个数为3的同学，展示姓名，年龄与爱好（不使用 \$size 来求长度，要求使用 \$unwind 来拆分数组和 \$push 来合并数组）

In [27]:
```
# todo
"""
目标结果
{'name': 'Rose', 'age': 24, 'hobby': ['basketball', 'running', 'traveling']}
{'name': 'Grace', 'age': 22, 'hobby': ['photography', 'cooking', 'drama']}
{'name': 'Joe', 'age': 23, 'hobby': ['football', 'basketball', 'reading']}
"""
```

Out[27]:
```
"\n目标结果\n{'name': 'Rose', 'age': 24, 'hobby': ['basketball', 'running', 'tr
aveling']}\n{'name': 'Grace', 'age': 22, 'hobby': ['photography', 'cooking', 'd
rama']}\n{'name': 'Joe', 'age': 23, 'hobby': ['football', 'basketball', 'readin
g']}\n"
```

In [28]:
```
#Task4
unwind = {"$unwind": "$hobby"}
group = {"$group": {"_id": {"name": "$name", "age": "$age"},
                    "hobby": {"$push": "$hobby"}, "hobbyCount": {"$sum": 1}}}
match = {"$match": {"hobbyCount" : 3}}
project = {"$project": {"_id":0, "name": "$_id.name",
                        "age": "$_id.age", "hobby":"$hobby"}}
result = users_col.aggregate([unwind, group, match, project])
for each in result:
    print(each)
```

```
{'name': 'Joe', 'age': 23, 'hobby': ['football', 'basketball', 'reading']}
{'name': 'Grace', 'age': 22, 'hobby': ['photography', 'cooking', 'drama']}
{'name': 'Rose', 'age': 24, 'hobby': ['basketball', 'running', 'traveling']}
```

索引：对 Lab2 的一些补充

单条索引：按照姓名升序

In [29]:
```
users_col.create_index([("name", 1)], unique=True)
```

Out[29]:
```
'name_1'
```

复合索引：创建复合索引，按照姓名升序，按照年龄降序

In [30]:
```
users_col.create_index([("name", 1), ("age", -1)], unique=True)
```

Out[30]:
```
'name_1_age_-1'
```

删除索引

In [31]:
```
print("删除前索引信息\n", users_col.index_information())
users_col.drop_index("name_1")  # 括号里面的参数是索引名
users_col.drop_index("name_1_age_-1")
print("删除后索引信息\n", users_col.index_information())
```

```
删除前索引信息
 {'_id_': {'v': 2, 'key': [('_id', 1)]}, 'name_1': {'v': 2, 'key': [('name', 1)],
'unique': True}, 'name_1_age_-1': {'v': 2, 'key': [('name', 1), ('age', -1)], 'un
ique': True}}
删除后索引信息
 {'_id_': {'v': 2, 'key': [('_id', 1)]}}
```

性能测试

索引的价值在于提高基于索引字进行段数据查询的效率，我们可以通过构造一批数据，对比建索引前的查询时间来体会索引的价值

In [ ]:

```python
# 构造3000000条数据
import random


# 先清空一下数据库
users_col.delete_many({})

batch_users = []
sex = ["f", "m"]
for i in range(300000):
    user = {
        "name": "xxx" + str(i),
        "age": random.randint(20, 55),  # 产生20, 55之间的随机数
        "gender": sex[random.randint(0, 1)],
    }
    batch_users.append(user)
users_col.insert_many(batch_users)
```

In [33]:

```python
# 直接查询用时
import datetime

starttime = datetime.datetime.now()

result = users_col.find(
    {
        "$or": [
            {"name": "xxx10000"},
            {"name": "xxx140000"},
            {"name": "xxx9000"},
            {"name": "xxx23000"},
            {"name": "xxx24050"},
            {"name": "xxx12000"},
            {"name": "xxx14300"},
            {"name": "xxx9300"},
            {"name": "xxx23300"},
            {"name": "xxx24350"},
            {"name": "xxx11100"},
            {"name": "xxx15200"},
            {"name": "xxx8100"},
            {"name": "xxx22100"},
            {"name": "xxx26150"},
            {"name": "xxx10200"},
            {"name": "xxx14020"},
            {"name": "xxx9020"},
            {"name": "xxx23020"},
            {"name": "xxx24070"},
            {"name": "xxx10300"},
            {"name": "xxx14030"},
            {"name": "xxx9030"},
            {"name": "xxx23030"},
            {"name": "xxx24080"},
        ]
    }
)
for each in result:
    print(each)
endtime = datetime.datetime.now()

print("开始时间:", starttime)
print("结束时间:", endtime)
print("时间差（微秒）:", (endtime - starttime).microseconds)
```

```
{'_id': ObjectId('66f503c094bc303c9eed3b44'), 'name': 'xxx8100', 'age': 36, 'gend
er': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed3ec8'), 'name': 'xxx9000', 'age': 36, 'gend
er': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed3edc'), 'name': 'xxx9020', 'age': 23, 'gend
er': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed3ee6'), 'name': 'xxx9030', 'age': 30, 'gend
er': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed3ff4'), 'name': 'xxx9300', 'age': 27, 'gend
er': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed42b0'), 'name': 'xxx10000', 'age': 20, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed4378'), 'name': 'xxx10200', 'age': 26, 'gen
der': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed43dc'), 'name': 'xxx10300', 'age': 50, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed46fc'), 'name': 'xxx11100', 'age': 47, 'gen
der': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed4a80'), 'name': 'xxx12000', 'age': 55, 'gen
der': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed5264'), 'name': 'xxx14020', 'age': 35, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed526e'), 'name': 'xxx14030', 'age': 23, 'gen
der': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed537c'), 'name': 'xxx14300', 'age': 33, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed5700'), 'name': 'xxx15200', 'age': 49, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed71f4'), 'name': 'xxx22100', 'age': 40, 'gen
der': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed7578'), 'name': 'xxx23000', 'age': 31, 'gen
der': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed758c'), 'name': 'xxx23020', 'age': 24, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed7596'), 'name': 'xxx23030', 'age': 43, 'gen
der': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed76a4'), 'name': 'xxx23300', 'age': 26, 'gen
der': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed7992'), 'name': 'xxx24050', 'age': 21, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed79a6'), 'name': 'xxx24070', 'age': 21, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed79b0'), 'name': 'xxx24080', 'age': 25, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed7abe'), 'name': 'xxx24350', 'age': 27, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed81c6'), 'name': 'xxx26150', 'age': 20, 'gen
der': 'm'}
{'_id': ObjectId('66f503c194bc303c9eef3e80'), 'name': 'xxx140000', 'age': 24, 'ge
nder': 'm'}
开始时间: 2024-09-26 06:48:36.956365
结束时间: 2024-09-26 06:48:37.143803
时间差（微秒）: 187438
```

In [34]:
```python
# 创建索引查询用时间
users_col.create_index([("name", 1)], unique=True)
starttime = datetime.datetime.now()
result = users_col.find(
    {
        "$or": [
            {"name": "xxx10000"},
            {"name": "xxx140000"},
            {"name": "xxx9000"},
            {"name": "xxx23000"},
            {"name": "xxx24050"},
            {"name": "xxx12000"},
            {"name": "xxx14300"},
            {"name": "xxx9300"},
            {"name": "xxx23300"},
            {"name": "xxx24350"},
            {"name": "xxx11100"},
            {"name": "xxx15200"},
            {"name": "xxx8100"},
            {"name": "xxx22100"},
            {"name": "xxx26150"},
            {"name": "xxx10200"},
            {"name": "xxx14020"},
            {"name": "xxx9020"},
            {"name": "xxx23020"},
            {"name": "xxx24070"},
            {"name": "xxx10300"},
            {"name": "xxx14030"},
            {"name": "xxx9030"},
            {"name": "xxx23030"},
            {"name": "xxx24080"},
        ]
    }
)
for each in result:
    print(each)
endtime = datetime.datetime.now()

print("开始时间:", starttime)
print("结束时间:", endtime)
print("时间差（微秒）:", (endtime - starttime).microseconds)
users_col.drop_index("name_1")  # 结束后删除索引以防之后忘记删除
```

```
{'_id': ObjectId('66f503c094bc303c9eed42b0'), 'name': 'xxx10000', 'age': 20, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed4378'), 'name': 'xxx10200', 'age': 26, 'gen
der': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed43dc'), 'name': 'xxx10300', 'age': 50, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed46fc'), 'name': 'xxx11100', 'age': 47, 'gen
der': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed4a80'), 'name': 'xxx12000', 'age': 55, 'gen
der': 'f'}
{'_id': ObjectId('66f503c194bc303c9eef3e80'), 'name': 'xxx140000', 'age': 24, 'ge
nder': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed5264'), 'name': 'xxx14020', 'age': 35, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed526e'), 'name': 'xxx14030', 'age': 23, 'gen
der': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed537c'), 'name': 'xxx14300', 'age': 33, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed5700'), 'name': 'xxx15200', 'age': 49, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed71f4'), 'name': 'xxx22100', 'age': 40, 'gen
der': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed7578'), 'name': 'xxx23000', 'age': 31, 'gen
der': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed758c'), 'name': 'xxx23020', 'age': 24, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed7596'), 'name': 'xxx23030', 'age': 43, 'gen
der': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed76a4'), 'name': 'xxx23300', 'age': 26, 'gen
der': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed7992'), 'name': 'xxx24050', 'age': 21, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed79a6'), 'name': 'xxx24070', 'age': 21, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed79b0'), 'name': 'xxx24080', 'age': 25, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed7abe'), 'name': 'xxx24350', 'age': 27, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed81c6'), 'name': 'xxx26150', 'age': 20, 'gen
der': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed3b44'), 'name': 'xxx8100', 'age': 36, 'gend
er': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed3ec8'), 'name': 'xxx9000', 'age': 36, 'gend
er': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed3edc'), 'name': 'xxx9020', 'age': 23, 'gend
er': 'm'}
{'_id': ObjectId('66f503c094bc303c9eed3ee6'), 'name': 'xxx9030', 'age': 30, 'gend
er': 'f'}
{'_id': ObjectId('66f503c094bc303c9eed3ff4'), 'name': 'xxx9300', 'age': 27, 'gend
er': 'f'}
开始时间: 2024-09-26 06:48:37.977359
结束时间: 2024-09-26 06:48:38.050852
时间差（微秒）: 73493
```

在我的机器上运行后发现建立索引前后明显的查询时间分别为307881微秒和11248，创建索引之后的查询耗时有了明显的降低

In [35]:
```
users_col.delete_many({})
# 插入无关数据后记得删除，保持良好习惯，以免学院服务器崩坏
```

```
Out[35]: DeleteResult({'n': 300000, 'ok': 1.0}, acknowledged=True)
```

练习

Task 5 下面需要同学们探索对比，创建索引对插入数据的影响

```
In [ ]: # todo
        """
        过程&结论
        """
```

1.无索引的插入：

操作过程：创建一个集合，不添加任何索引；在集合中添加大量数据，记录插入操作所需时间。

```
In [38]: #无索引插入
         import time
         from pymongo import MongoClient

         # 创建大量数据
         data = [{"name": f"Student {i}", "age": i % 100, "hobby": f"Hobby {i % 10}"} for
```

```
In [41]: # 插入数据并记录时间
         start_time = time.time()
         users_col.insert_many(data)
         end_time = time.time()

         print(f"Time taken without index: {end_time - start_time} seconds")
```

```
Time taken without index: 0.06215023994445801 seconds
```

2.有索引插入：

操作过程：创建一个集合，在需要字段上创建索引（name/age）；在集合中添加大量数据，记录插入操作所需时间。

```
In [40]: #有索引插入
         collection_with_index = db["with_index_collection"]

         # 在 name 字段上创建索引
         collection_with_index.create_index("name")

         # 插入数据并记录时间
         start_time = time.time()
         collection_with_index.insert_many(data)
         end_time = time.time()

         print(f"Time taken with index: {end_time - start_time} seconds")

         users_col.delete_many({})
```

```
Time taken with index: 0.08653044700622559 seconds
Out[40]: DeleteResult({'n': 10000, 'ok': 1.0}, acknowledged=True)
```

结论：插入时，如果插入的列有索引，插入速度会减慢。

无索引：插入操作会更快，因为不需要更新任何索引。 有索引：插入时，数据库不仅要将数据插入文档，还需要更新索引。因此，索引越多，插入性能可能会越低，特别是在大规模数据插入的情况下。