파이썬 Mongo DB 연동
- 데이터 다루기

pymongo 모듈 이해

pymongo라이브러리 소개 및 설치

- mongodb을 python에서 사용할 수 있는 라이브러리

(pymongo 라이브러리 설치)

- !pip install pymongo
- MongoClient 라이브러리
- from pymongo import MongoClient

pymongo 모듈 이해

일반적인 pymogo 핸들링 코드 작성 순서

- pymogo모듈 import
- client.MogoClient()를 사용하여 mongo에 연결
- 호스트명, 포트, 로그인, 암호, 접속할 DB 등을 파라미터로 지정
- connection.Database 사용
  - db = Connection.test-db
  - db = Conection[“test-db”]

pymongo 모듈 이해

일반적인 pymogo 핸들링 코드 작성 순서

```
from pymongo import MongoClient

   # client = MongoClient('localhost', 27017)

client = MongoClient("mongodb://localhost:27017/")

     # db = client.test_database

db = client['test-db'] # test-db라는 이름의 데이터베이스에 접속

print(client.list_database_names()) # ['admin', 'config', 'local', 'test-database']
```

패턴으로 익히는 pymongo

데이터 삽입(INSERT)

- 데이터 생성(JSON-{Key:Values})

```
import datetime
 post = {"author": "Mike",
              "text": "My first blog post!",
             "tags": ["mongodb", "python", "pymongo"],
              "date": datetime.datetime.utcnow() }
post
```

```
{'author': 'Mike',
             'text': 'My first blog post!',
             'tags': ['mongodb', 'python', 'pymongo'],
             'date': datetime.datetime(2020, 11, 4, 12, 42, 56, 217943)}
```

패턴으로 익히는 pymongo

데이터 삽입(INSERT)

- **insert_one()**

```
# Collection 접근 - 'posts' Collection
        posts = db.posts
# Document 추가 - insert_one() 메서드 이용
post_id = posts.insert_one(post)
print(post_id)
```

- **db 인스턴스의 list_collection_names()를 호출하면 DB에 존재하는 Collection들의 목록을 출력**

패턴으로 익히는 pymongo

데이터 삽입(INSERT)

- **insert_many()**

```
new_posts = [
 { "author": "Mike", "text": "Another post!", "tags": ["bulk",
"insert"], "date": datetime.datetime(2009, 11, 12, 11, 14) },
{ "author": "Eliot", "title": "MongoDB is fun", "text": "and
pretty easy too!", "date": datetime.datetime(2009, 11, 10,
10, 45) }
]
 result = posts.insert_many(new_posts)
```

패턴으로 익히는 pymongo

데이터 조회하기(find)

- **find_one()**

```
for d in db['posts'].find():
    print(d['author'], d['text'], d['tags'])


# 'author':'hun' 인 데이터 조회
print(db.posts.find_one({'author':'hun'})['text'])


# hun mongoDB is what..? ['mongoDB', 'python', 'pymongo']
# lee Who are you? ['person', 'lee']
 # text 칼럼을 제외하고 데이터 가져오기
for d in db['posts'].find({}, {'text' : 0}):
    print(d)
```

패턴으로 익히는 pymongo

데이터 조회하기(find)

- **find_one()**

```
# Collection 내 단일 Document 조회
import pprint
pprint.pprint(posts.find_one())
```

```
# 쿼리를 통한 Documents 조회
pprint.pprint(posts.find_one({"author": "Mike"}))
```

패턴으로 익히는 pymongo

데이터 조회하기(find)

- **find()**

```
# Collection 내 단일 Document 조회
import pprint
for post in posts.find({"author": "Mike"}):
    pprint.pprint(post)
```

패턴으로 익히는 pymongo

데이터 수정(update)

- update_one() 메서드 - 1개 document 수정
  - update_one(query, newvalues)
    - [매개변수]
    - query
    - 필수. 수정할 문서를 정의하는 쿼리 객체.
    - ※ 형식: {"field_name": "current_value"}
    - 참고 : 쿼리에서 둘 이상의 문서를 찾으면 첫 번째 항목만 수정됨.
    - newvalues
    - 필수. 문서의 새 값을 정의하는 객체.
    - ※ 형식: {"$set" : {"field_name": "new_value"}}

패턴으로 익히는 pymongo

데이터 수정(update)

- update_many() 메서드 - 다수 document 수정
  - update_many(query, newvalues)
    - [매개변수]
    - query
    - 필수. 수정할 문서를 정의하는 쿼리 객체.
    - ※ 정규표현식 등 사용해 여러 문서 선택.
    - ※ 형식: {"field_name" : {"$regex": "정규표현식"}}
    - newvalues
    - 필수. 문서의 새 값을 정의하는 객체.
    - ※ 형식: {"$set" : {"field_name": "new_value"}}

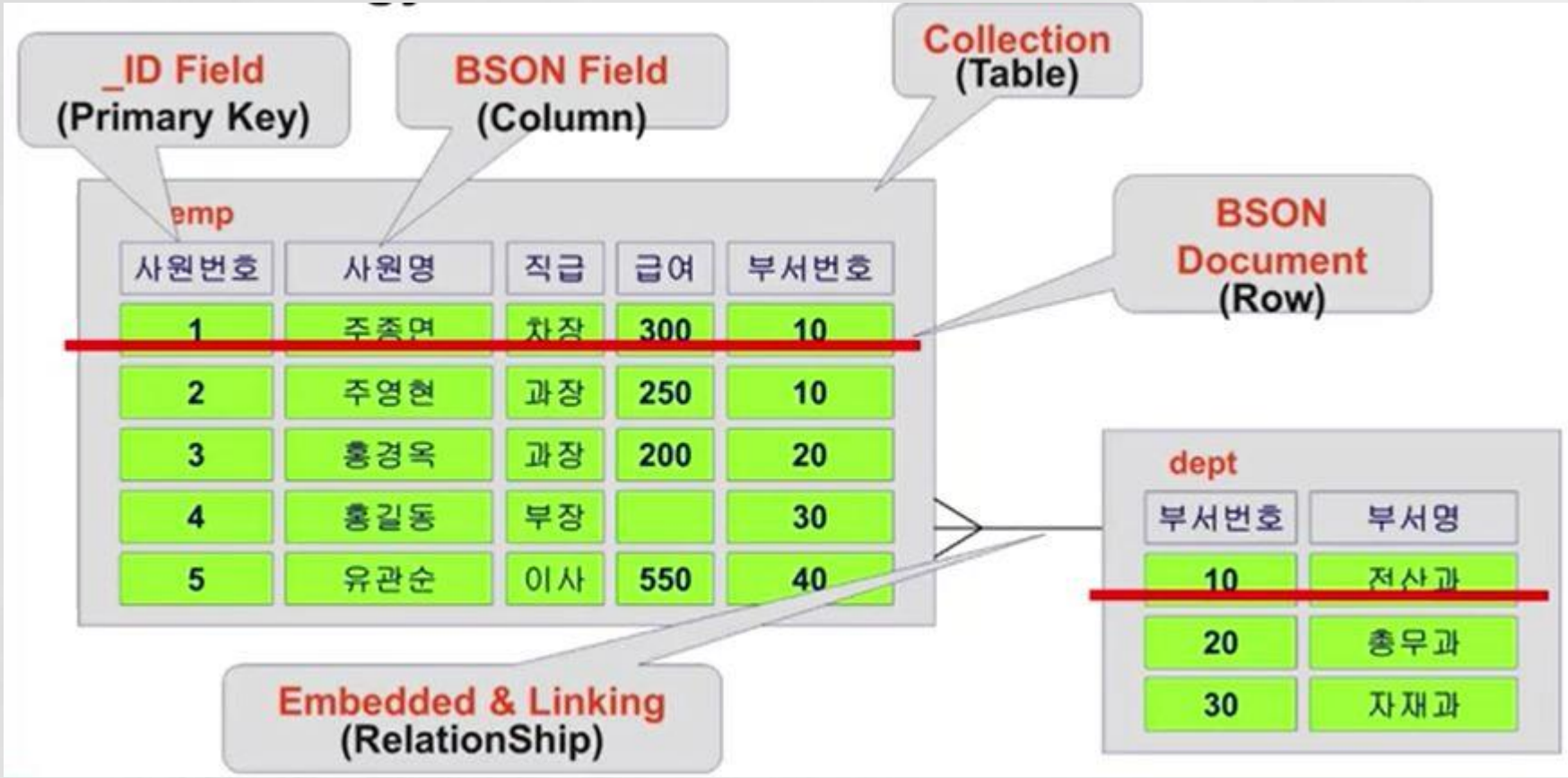
패턴으로 익히는 pymongo

데이터 삭제(delete)

- **delete_one() 메서드 – 1개 document 삭제**
  - delete_one() 메서드의 첫 번째 매개변수는 삭제할 document를 정의하는 객체
  - query에서 두 개이상의 documents를 찾으면 첫 번째 항목만 삭제
- **delete_many() 메서드 – 다수 document 삭제**
  - delete_many() 메서드의 첫 번째 매개변수는 삭제할 문서를 정의하는 쿼리 객

참고) MongoDB 데이터 처리

Terminology



참고) MongoDB 데이터 처리

Collection 생성

- 관계형 데이터베이스의 논리적 저장 구조인 테이블(Table) 에 해당되는 데이터 구조를 MongoDB 에서는 컬렉션(Collection) 이라고 표현
- 관계형 데이터베이스의 데이블 구조는 하나의 테이블을 생성하기 위해서는 반드시 구성 요소(컬럼명, 데이터 타입과 길이, 제약 조건)에 대한 정의가 먼저 되어야 하는데 이것을 정형화된 데이터 구조라고 표현
- Collection을 생성할 때 구성 요소(필드명 ,데이터 타입과 길이 등)가 결정되어 있지 않더라도 데이터 저장 구조를 생성

참고) MongoDB 데이터 처리

Collection 생성과 삭제

```
> db.createCollection ("emp", { capped : false, size:8192 });
{ "ok" : 1 }                          ← capped : 해당 공간이 모두 사용되면 다시 처음부터
                                                 재 사용할 수 있는 데이터 구조를 생성할 때
                                        size     : 해당 Collection의 최초 생성 크기 지정 가능
> show collections
 emp
>
> db.emp.validate();                   ← Collection의 현재 상태 및 정보 분석
> {
      "ns" : "test.emp",
      "firstExtent" : "0:61000 ns:test.emp",
      "lastExtent" : "0:61000 ns:test.emp",
      "extentCount" : 1,
      "datasize" : 0,
      "nrecords" : 0,
      "lastExtentSize" : 8192,

> db.emp.renameCollection( "employees" )    ← 해당 Collection 이름 변경
> db.employees.drop();                      ← 해당 Collection 삭제
```

참고) MongoDB 데이터 처리

데이터의 입력 수정 삭제

```
> db.emp.insert ({ eno : 1101, fname : “JIMMY" });
> db.emp.insert ({ eno : 1102, fname : “ADAM“, lname : “KROLL” });
> db.emp.insert ({ eno : 1103, fname : “SMITH“, job : “CLERK” });
```

```
> db.emp.update ({ eno:1101 }, { $set: { fname : “JOO” } } );
> db.emp.update ({ eno:1102 }, { $set: { job : “CHIEF” } } );
> db.emp.update ({ eno:1103 }, { $set: { lname : “STANFORD” } } );
```

```
> db.emp.find().sort ({eno:-1});

{ "_id" : ObjectId("4fe6852f5642c534a77fbdb1"),
      "eno" : 1103, "fname" : "SMITH", "job" : “CERK", "lname" : "STANFORD" }
{ "_id" : ObjectId("4fe685195642c534a77fbdb0"),
      "eno" : 1102, "fname" : "ADAM", "job" : "CHIEF", "lname" : "KROLL" }

> db.emp.remove ({ eno: 1101});
```

참고) MongoDB 데이터 처리

SQL과 Mongo Query 비교

SQL Statement	Mongo Query Statement
CREATE TABLE emp (empno Number, ename Number)	db.createCollection("emp")
INSERT INTO emp VALUES(3,5)	db.emp.insert({empno:3, ename:5})
SELECT * FROM emp	db.emp.find()
SELECT empno, ename FROM emp	db.emp.find({}, {empno:1, ename:1})
SELECT * FROM emp WHERE empno=3	db.emp.find({empno:3})
SELECT empno, ename FROM emp WHERE empno=3	db.emp.find({empno:3}, {empno:1, ename:1})
SELECT * FROM emp WHERE empno=3 ORDER BY ename	db.emp.find({empno:3}).sort({ename:1})

참고) MongoDB 데이터 처리

SQL과 Mongo Query 비교

SQL Statement	NoSQL Statement
SELECT * FROM emp WHERE empno > 3	db.emp.find({empno:{$gt:3}})
SELECT * FROM emp WHERE empno != 3	db.emp.find({empno:{$ne:3}})
SELECT * FROM emp WHERE ename LIKE "%Joe%"	db.emp.find({ename:/Joe/})
SELECT * FROM emp WHERE ename LIKE "Joe%"	db.emp.find({ename:/^Joe/})
SELECT * FROM emp WHERE empno>1 AND empno <=4	db.emp.find({empno:{$gt:1,$lte:4}})
SELECT * FROM emp ORDER BY ename DESC	db.emp.find().sort({ename:-1})
SELECT * FROM emp WHERE empno=1 and ename='Joe'	db.emp.find({empno:1,ename:'Joe'})
SELECT * FROM emp WHERE empno=1 or empno=3	db.emp.find({ $or : [{ empno : 1 } , { empno : 3 }] })
SELECT * FROM emp WHERE rownum = 1	db.emp.findOne()

참고) MongoDB 데이터 처리

SQL과 Mongo Query 비교

SQL Statement	NoSQL Statement
SELECT empno FROM emp o, dept d WHERE d.deptno=o.deptno AND d.deptno=10	o = db.emp.findOne({empno:1}); name = db.dept.findOne({deptno:o.deptno})
SELECT DISTINCT ename FROM emp	db.emp.distinct('ename')
SELECT COUNT(*) FROM emp	db.emp.count()
SELECT COUNT(*) FROM emp where deptno > 10	db.emp.find({deptno: {'$gt': 10}}).count()
SELECT COUNT(sal) from emp	db.emp.find({sal: {'$exists': true}}).count()
CREATE INDEX i_emp_ename ON emp(ename)	db.emp.ensureIndex({ename:1})
CREATE INDEX i_emp_no ON emp(deptno ASC, ename DESC)	db.emp.ensureIndex({deptno:1,ename:-1})
UPDATE emp SET ename='test' WHERE empno=1	db.emp.update({empno:1}, {$set:{ename:' test'}})
DELETE FROM emp WHERE deptno=10	db.emp.remove({deptno:10});

정리

정리

- pymogodb 모듈 이해
- 패턴으로 익히는 pymogodb
- DB연동하여 데이터 삽입(INSERT)
- DB연동하여 데이터 조회(FIND)
- DB연동하여 데이터 수정(UPDATE)
- DB연동하여 데이터 삭제(DELETE)