

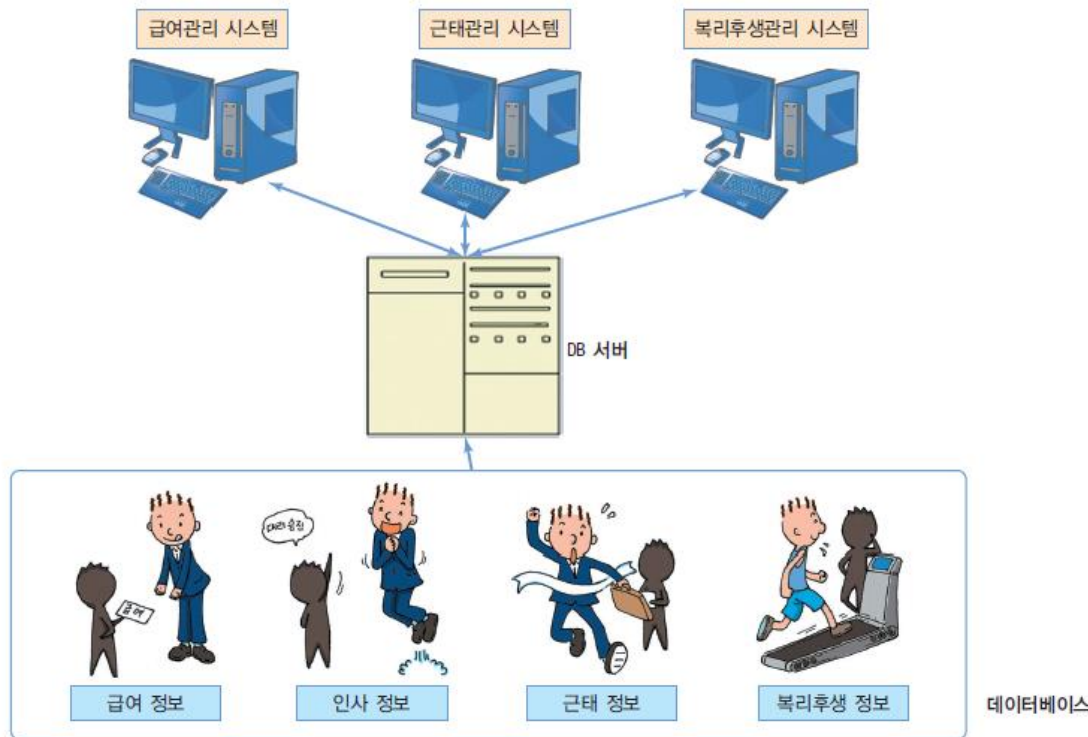
MongoDB

Quick Guide

데이터베이스와 데이터베이스 관리 시스템

▶ Database

- ▶ 데이터의 집합 (a Set of Data)
- ▶ 여러 응용 시스템(프로그램)들의 통합된 정보들을 저장하여 운영할 수 있는 공용(Shared) 데이터의 집합
- ▶ 효율적으로 저장, 검색, 갱신할 수 있도록 데이터 집합들끼리 연관시키고 조직화되어야 한다



데이터베이스의 특성

- ▶ 실시간 접근성 (Real-time Accessibility)
사용자의 요구를 즉시 처리할 수 있다
- ▶ 계속적인 변화 (Continuous Evolution)
정확한 값을 유지하기 위해 삽입, 삭제, 수정 작업 등을 이용하여 데이터를 지속적으로 갱신할 수 있다
- ▶ 동시 공유성 (Concurrent Sharing)
사용자마다 서로 다른 목적으로 사용하므로 동시에 여러 사람이 동일한 데이터에 접근하고 이용할 수 있다
- ▶ 내용 참조 (Contents Reference)
저장한 데이터 레코드의 위치나 주소가 아닌 사용자가 요구하는 데이터의 내용, 즉 데이터 값에 따라 참조할 수 있어야 한다

데이터베이스와 데이터베이스 관리 시스템

- ▶ 데이터베이스 관리 시스템 (Database Management System = DBMS)
 - ▶ 데이터베이스를 관리하는 소프트웨어
 - ▶ 여러 응용 소프트웨어(프로그램) 또는 시스템이 동시에 데이터베이스에 접근하여 사용할 수 있게 한다
 - ▶ Oracle, Microsoft SQL Server, MySQL, DB2 등의 상용 또는 공개 DBMS가 있다
- ▶ NoSQL (Not Only SQL)
 - ▶ 기존의 DBMS는 데이터베이스 스키마를 작성하고 그 구조에 적합한 SQL이라는 구문을 이용하여 데이터를 관리
 - ▶ 빅데이터가 주요 흐름이 되면서 기존의 구조화된 DBMS로는 처리가 어려운 데이터들이 발생하게 됨
 - ▶ Volume : 데이터의 양 자체가 많아졌다
 - ▶ Velocity : 데이터가 생성되는 속도 자체가 빨라졌다
 - ▶ Variety : 형태 및 종류가 다양하여 구조화된 기존 DBMS의 스키마 틀 안에서 관리하기 어렵다
 - ▶ 이러한 새로운 데이터들을 다루기 위해 DBMS에도 변화가 필요 -> NoSQL의 등장

빅데이터 시대와 NoSQL

- ▶ 관계형 데이터베이스 모델은 1970년대에 처음 소개되었기 때문에 인터넷과 클라우드 환경이 발달한 현대 애플리케이션들의 일부 수요를 만족시키지 못함
 - ▶ NoSQL은 기존 관계형 데이터베이스의 규칙 일부를 포기하는 대신, 뛰어난 **확장성**이나 **성능**을 발전시키는 방향으로 개발
- ▶ NoSQL DBMS의 네 가지 분류

종류	예시
키-값 스토어	Redis, AWS Dynamo
컬럼 지향 스토어	HBase, Cassandra
도큐먼트 지향 스토어	MongoDB
그래프 데이터베이스	Neo4J

MongoDB

- ▶ 새로이 등장한 NoSQL DBMS 중에서 가장 많은 사용 기반을 확보한 대표적 NoSQL
- ▶ MongoDB의 장점
 - ▶ Flexibility : Schema-less 여서 다양한 구조의 데이터를 데이터베이스 변경 없이 저장할 수 있다
 - ▶ Performance : Read & Write 성능이 뛰어남. Caching, 대량의 네트워크 트래픽 등에도 RDBMS에 비해 유리
 - ▶ Scalability : 설계시 Scale-Out 구조를 채택, 운용과 확장이 손쉬움
 - ▶ Document-Oriented : 문서 중심의 설계로 강력한 Query 성능을 제공
 - ▶ Conversion / Mapping : JSON(BSON) 형태로 저장, 직관적이고 개발이 편리
- ▶ MongoDB의 단점
 - ▶ 트랜잭션, 정합성이 필요한 데이터(예: 금융정보 등)에는 부적합
 - ▶ 메모리 크기에 성능이 많이 좌우됨
 - ▶ 추가/삭제/변경이 많은 데이터 관리에는 부적합

MongoDB

: Installation

- ▶ <https://www.mongodb.com/>
 - ▶ > Products > Community Server


Command Line에서 사용하기 위해서는


- Products > Shell 다운로드
- 설치 디렉터리 **PATH**에 추가


(in preview), full-text search, and data distribution across regions and clouds. Deploy in minutes on AWS, Google Cloud, and/or Azure, with no downloads necessary.

Give it a try with a free, highly-available 512 MB cluster.

Version	6.0.4 (current)	▼
Platform	Windows	▼
Package	msi	▼

Download 

 Copy link

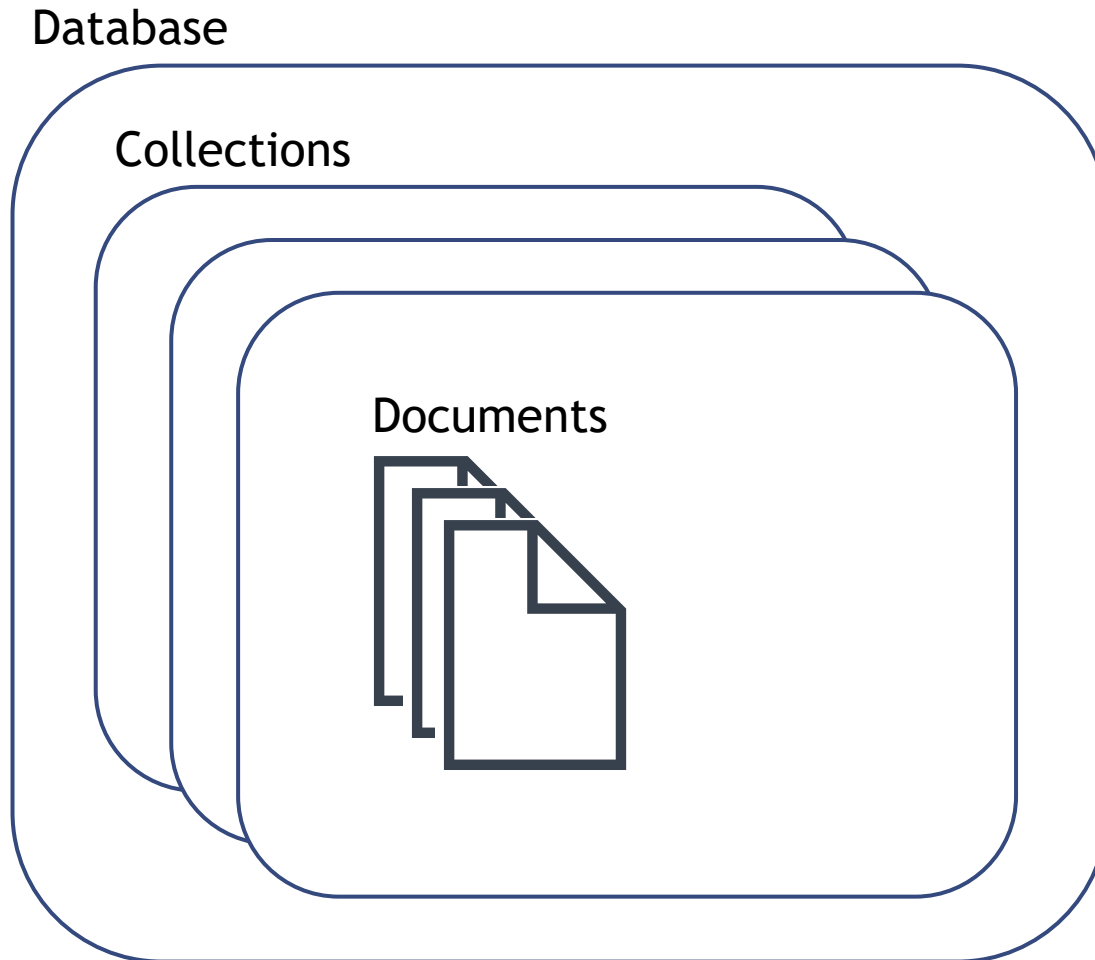
More Options 

Using MongoDB Shell

Basic

MongoDB

: Database, Collection, Document



RDBMS vs MongoDB

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple / Row	Document
Column	Field
Table Join	Embedded Document

MongoDB

: Database, Collection, Document

- ▶ 기본적인 데이터베이스 관리 명령
 - ▶ `show dbs`
현재 MongoDB 내에 있는 데이터베이스의 목록 확인
 - ▶ `use {DATABASE_NAME}`
{DATABASE_NAME} 이름의 데이터베이스를 사용함
 - ▶ `show collections`
현재 선택된 데이터베이스 내의 컬렉션을 확인함
- ▶ MongoDB는 데이터베이스 생성을 위한 별도 절차는 없다
 - ▶ 삭제는 `db.dropDatabase()`로 수행

```
C:\>mongo
MongoDB shell version v4.0.5
...

> show dbs // show databases
admin    0.000GB
config   0.000GB
local    0.000GB
> use local // use database
switched to db local
> db // print current database
local
> show collections
startup_log

> quit()
```

MongoDB Shell

: Database, Collection, Document

- ▶ MongoDB는 field:value 쌍의 집합으로 구성된 BSON document 형식으로 데이터를 저장한다
 - ▶ BSON: JSON의 바이너리 구현

MongoDB Document

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```

← field: value
← field: value
← field: value
← field: value

- ▶ 연습: mydb 데이터베이스의 posts 컬렉션에 title이 "First Post" 인 문서를 삽입해 봅시다

```
> use mydb  
switched to db mydb  
> db.posts.insertOne({title: "First Post"})  
{  
  acknowledged: true,  
  insertedId: ObjectId("63d8771a3c0708cefa44147c")  
}
```

MongoDB

: Database, Collection, Document

- ▶ 데이터베이스에 대한 좀더 자세한 정보 확인
 - ▶ `db.stats()`
- ▶ 개별 컬렉션에 대한 좀더 자세한 정보 확인
 - ▶ `db.{collection}.stats();`
- ▶ MongoDB 클라이언트(**mongo**)는 전통적인 RDBMS 처럼 **SQL**을 사용하지 않으며 그 자체가 **JavaScript** 엔진이다
 - ▶ JavaScript에서 객체를 작업하는 방식을 연상

```
> db.stats() // show info
{
  "db" : "mydb",
  "collections" : 2,
  "views" : 0,
  "objects" : 5,
  "avgObjSize" : 78.6,
  "dataSize" : 393,
  "storageSize" : 73728,
  "numExtents" : 0,
  "indexes" : 2,
  "indexSize" : 73728,
  "scaleFactor" : 1,
  "fsUsedSize" : 197324382208,
  "fsTotalSize" : 238855385088,
  "ok" : 1
}
```

MongoDB Shell

: Document Insert

- ▶ MongoDB는 field:value 쌍의 집합으로 구성된 BSON document 형식으로 데이터를 저장한다
 - ▶ BSON: JSON의 바이너리 구현

MongoDB Document

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```

← field: value
← field: value
← field: value
← field: value

- ▶ 연습: mydb 데이터베이스의 posts 컬렉션에 title이 "First Post" 인 문서를 삽입해 봅시다

```
> use mydb  
switched to db mydb  
> db.posts.insertOne({title: "First Post"})  
{  
  acknowledged: true,  
  insertedId: ObjectId("63d8771a3c0708cefa44147c")  
}
```



MongoDB Shell

: Document Search

- ▶ MongoDB 내의 모든 Document에는
 - ▶ 고유한 구분자(UUID: Unique Identifier)가 들어 있으며
 - ▶ UUID는 `_id` 라는 필드명으로, `ObjectId` 라는 특수한 객체 형태로 MongoDB에 의해 부여되고 관리된다
 - ▶ `ObjectId(12bytes)` = `Timestamp(4byte)` +
 `Client Machine ID (3byte)` +
 `Client Process ID (2byte)` +
 증가 카운트(3byte)
- ▶ 연습: mydb 데이터베이스의 `posts` 컬렉션에서 Document를 하나 조회하여 입력 당시의 Document와 비교해 봅니다

```
> db.posts.findOne()  
{  
  "_id" : ObjectId("5c28cf8e2d7b10dc3b05e265"),  
  "title" : "First Post"  
}
```



MongoDB Shell

: Document Update

- ▶ `.update()` 메서드를 이용한 기존 Document 수정
 - ▶ `.update()` 메서드를 이용하면 기존 컬렉션에 담겨 있는 Document를 변경할 수 있다.
 - ▶ 매개 객체로 두 개의 객체를 전달
 - ▶ 첫 번째 객체 : 변경할 Document 지정을 위한 객체
 - ▶ 두 번째 객체 : 변경할 내용을 담고 있는 객체. `$set` 연산자를 이용해야 `update`, `$set` 연산자를 사용하지 않으면 객체 내용을 통째로 변경하게 되므로 주의
- ▶ 연습: `mydb` 데이터베이스의 `posts` 컬렉션에서 `.updateOne()`, `updateMany()` 메서드로 조건에 맞는 문서를 변경해 보시다

```
> db.posts.updateOne(  
  {"title": "First Post"},  
  { $set:  
    { createdAt: new Date(), updatedAt: new Date() }  
  });  
{ acknowledged: true, insertedId: null, matchedCount: 1,  
  modifiedCount: 1, upsertedCount: 0 }
```

변경할 문서의 조건

변경할 내용

MongoDB Shell

: Document Delete

- ▶ `.remove()`
 - ▶ 컬렉션으로부터 Document를 삭제
 - ▶ 삭제하고자 하는 객체의 검색 조건을 부여하거나 삭제하고자 하는 객체를 직접 삭제
- ▶ 연습: mydb 데이터베이스의 posts 컬렉션에서 Document를 하나 조회하여 객체에 할당한 후 `.remove()` 메서드로 해당 Document를 삭제해 봅시다

```
> let post = db.posts.findOne()  
> db.posts.remove(post)  
WriteResult({ "nRemoved" : 1 })
```

- ▶ `db.posts.findOne()` 메서드를 이용하여 Document가 변경되었는지 확인해 봅니다.

```
> db.posts.remove({title:/Learn/})  
WriteResult({ "nRemoved" : 1 })
```

MongoDB Shell

: 연습문제

▶ 지금까지 학습한 mongodb CRUD 명령으로 컬렉션을 만들어 봅시다.

▶ Database : mydb, Collection : posts

title	by	views	likes	createdAt
First Post	bit	100	10	현재시간
Second Post	hit	90	20	현재시간
Third Post	bit	80	30	현재시간
Fourth Post	hit	70	40	현재시간
Sisth Post	bit	60	50	현재시간

MongoDB Shell

: Document Search

▶ .find()와 .findOne()

- ▶ 컬렉션으로부터 Document를 검색
- ▶ .find() 메서드는 조건을 만족하는 문서의 커서(Cursor)을, findOne() 메서드는 조건을 만족하는 문서 중 하나를 반환

▶ 검색을 위한 기본 Operation

Operation	Syntax	Example
같다	{ <key>: <value> }	db.posts.find({"by": "bit"})
작다	{ <key>: { \$lt: <value> } }	db.posts.find({"likes": { \$lt: 50 } })
작거나 같다	{ <key>: { \$lte: <value> } }	db.posts.find({"likes": { \$lte: 50 } })
크다	{ <key>: { \$gt: <value> } }	db.posts.find({"likes": { \$gt: 50 } })
크거나 같다	{ <key>: { \$gte: <value> } }	db.posts.find({"likes": { \$gte: 50 } })
같지 않다	{ <key>: { \$ne: <value> } }	db.posts.find({"likes": { \$ne: 50 } })

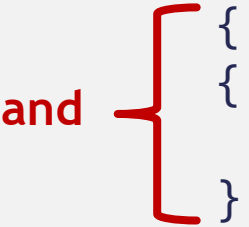
MongoDB Shell

: Document Search

▶ \$and와 \$or

- ▶ 여러 조건을 조합하여 검색을 하고자 할 경우에는 \$and, \$or 오퍼레이터를 이용하여 조건을 배열로 묶을 수 있다.

```
db.posts.find(  
  { $and:  
    [  
      { "by": "bit" },  
      { "likes":  
        { $gt: 30 }  
      }  
    ]  
  }  
)
```



```
db.posts.find(  
  { $or:  
    [  
      { "by": "bit" },  
      { "likes":  
        { $gt: 30 }  
      }  
    ]  
  }  
)
```



MongoDB Shell

: Document Search

▶ 특정 필드의 출력(Projection)

- ▶ 기본적으로 `.find()` 메서드를 수행하면 해당 Document가 가진 모든 필드를 출력
- ▶ Projection 정보를 `.find()` 메서드의 두 번째 인자 값으로 넘겨주면 필드의 출력을 제어할 수 있다
 - ▶ 출력할 필드는 1, 출력하지 않을 필드는 0으로 설정

```
> use mydb
switched to db mydb
> db.posts.find( {}, { "title": 1, _id: 0 } )
{ "title" : "First Post" }
{ "title" : "Second Post" }
```

Projection Options

- ▶ `find()` 메서드 다음에 `.pretty()` 메서드를 호출하면 조회 결과를 띄어쓰기나 줄 바꿈을 포함, 보기 좋게 출력

MongoDB Shell

: Document Search

▶ 출력의 제한

- ▶ `.limit()` 메서드 : 컬렉션으로부터 받아와야 할 **Document**의 개수를 제한
- ▶ `.skip()` 메서드 : `.find()` 메서드를 통해 받아온 **Document** 중에서 지정된 개수만큼을 건너뛴
- ▶ 예) `posts` 컬렉션으로부터 받아온 **Document**들 중, 1개를 건너뛰고 2개의 **Document**를 가져옴

```
> db.posts.find( {}, { "title": 1, _id: 0 } ).limit(2).skip(1)
{ "title" : "Second Post" }
```

▶ 데이터의 정렬

- ▶ `.sort()` 메서드 : 컬렉션으로부터 **Document**를 검색할 때, 지정한 순서대로 검색 결과를 반환
 - ▶ 정렬에 참여할 필드의 정렬 순서를 1(오름차순), -1(내림차순)으로 지정

```
> db.posts.find( {}, { "title": 1, _id: 0 } ).sort( { "title": -1 } )
{ "title" : "Second Post" }
{ "title" : "First Post" }
```

title 필드의 역순으로 정렬

Programming MongoDB

In ECMAScript

MongoDB

: JavaScript Native Driver

▶ JavaScript에서 MongoDB를 사용하기 위한 연결 드라이버

▶ 설치

```
C:\> npm install mongodb
```

▶ MongoDB 연결을 위한 드라이버 설정과 접속

```
const MongoClient = require('mongodb').MongoClient;
const assert = require('assert');

const url = 'mongodb://localhost:27017'; // Connection URL
const dbName = "mydb"; // Database Name

// Database 주소 형태 mongodb://{server-ip}:{port}

const client = new MongoClient(url, {useNewUrlParser: true});

client.connect(function(err, client) {
  assert.equal(null, err); // 에러가 없는지 체크
  console.log("MongoDB Connected");
  client.close();
});
```

MongoDB

: JavaScript Native Driver


▶ Document Query의 수행 : insertOne과 insertMany


- ▶ Node.js는 Asynchronous(비동기) 방식으로 작성하므로 Callback 함수를 잘 이해하고 활용하여야 한다

```
// ...
client.connect(function(err, client) {
  assert.equal(null, err);
  console.log("MongoDB Connected");
  const db = client.db(dbName);

  // 1개의 Document Insert
  db.collection('friends').insertOne({name: "둘리"}, function(err, result) {
    assert.equal(null, err);
    assert.equal(1, result.insertedCount);

    // 여러 개의 Document Insert
    db.collection('friends').insertMany([{name: "도우너"}, {name: "마이콜"}], function(err, result) {
      assert.equal(null, err);
      assert.equal(2, result.insertedCount);
      client.close();
    });
  });
});
```

 Callback Function

 Callback Function

MongoDB

: JavaScript Native Driver

- ▶ Document Query의 수행 : updateOne과 updateMany의 예
 - ▶ Node.js는 Asynchronous(비동기) 방식으로 작성하므로 Callback 함수를 잘 이해하고 활용하여야 한다

```
// ...
client.connect(function(err, client) {
  assert.equal(null, err);
  console.log("MongoDB Connected");
  const db = client.db(dbName);

  db.collection('friends').insertMany([{name: "둘리"}, {name: "도우너"}, {name: "마이콜"}],
    function(err, result) {
      assert.equal(null, err);
      assert.equal(3, result.insertedCount);
      db.collection('friends').updateOne({name: "둘리"}, {$set: {species: "공룡"}}, function(err, result) {
        assert.equal(null, err);
        console.log(result.upsertedCount);
        client.close();
      });
    });
});
```

Callback Function

Callback Function

MongoDB

: JavaScript Native Driver

- ▶ Document Query의 수행 : deleteOne과 deleteMany의 예
 - ▶ Node.js는 Asynchronous(비동기) 방식으로 작성하므로 Callback 함수를 잘 이해하고 활용하여야 한다

```
db.collection('friends').deleteOne({species: "공룡"}, function(err, result) {  
    assert.equal(null, err);  
    console.log(result.deletedCount);  
    client.close();  
});
```

Promise를 이용한 코드 작성

: Callback Hell에서 탈출하기

- ▶ 자바스크립트는 비동기적으로 작동하기 때문에 여러 작업을 수행할 때 콜백 방식을 많이 사용
 - ▶ 콜백 방식은 코드 가독성을 해치기 때문(**Callback Hell**)에 최근에는 **Promise**를 이용한 코드 작성을 더 선호하는 경향이 있음
- ▶ 콜백 방식과 프라미스 방식 모두를 지원하는 명령
 - ▶ save
 - ▶ insertMany
 - ▶ exec
 - ▶ updateOne / updateMany
 - ▶ deleteOne / deleteMany
- ▶ 프라미스를 지원하지 않는 명령어는 **exec** 명령을 이용하면 별도 작업 없이 **Promise**를 이용할 수 있다