5일차. 데이터 적재 서비스 실습 - MongoDB

> *몽고디비를 통해 다양한 데이터 적재 예제를 실습합니다*

- 목차
  - 1. 데이터 모델링 기본
  - 2. 데이터 모델링 고급
  - 3. 몽고디비 서버 기동 및 접속
  - 4. 기본 명령어 예제 실습
  - 5. 모니터링 및 트러블슈팅
  - References
    - https://medium.com/@igorkhomenko/troubleshooting-mongodb-100-cpu-load-and-slow-queries-da622c6e1339
    - https://docs.mongodb.com/manual/tutorial/manage-the-database-profiler/

실습 이전에 docker-compose 최신 버전으로 업데이트 합니다

> *이전에 설치된 버전에서는 볼륨을 강제로 재시작 하는 기능이 없어 mongo-express 가 제대로 기동되지 않는 문제가 있습니다.*

- How to install Docker Compose on Ubuntu
  - 현재 버전이 1.24 이상이면 괜찮습니다 1.17 버전이라면 업데이트가 필요합니다

```
bash>
docker-compose --version
docker-compose version 1.24.0, build 0aa59064
```

- Uninstall docker-compose

```
sudo apt-get update
sudo apt-get upgrade
sudo apt install curl
```

sudo rm /usr/local/bin/docker-compose sudo apt-get remove docker-compose sudo apt-get autoremove

```
* Install docker-compose
```bash
bash>
sudo curl -L "https://github.com/docker/compose/releases/download/1.24.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
docker-compose --version
```

몽고디비 실습

1 데이터 모델링 기본

1-1 one-to-few

- 대부분의 경우 내장 문서로 저장하고 가져오는 방식이 간결하고 편하다 (단, 16mb 제약)
```

```
// testdb database 를 생성합니다
// person collection 을 생성합니다
{
  _id: ObjectId('5f391be125b953000783db79'),
  name: 'Edward Kim',
  hometown: 'Jeju',
  addresses: [
      {
          street: 'Samdoil-Dong',
          city: 'Jeju',
          cc: 'KOR'
      },
      {
          street: 'Albert Rd',
          city: 'South Melbourne',
          cc: 'AUS'
      }
  ]
}
```

- 생성된 데이터를 통해 조회합니다

```
db.person.findOne();
```

1-2 one-to-many

- 실제 객체의 ObjectID 값을 리스트로 가지고 다시 조회하는 방식 2번 이상 호출이 필요함 (_id 도 은근히 큰 데이터 16mb)

```
// parts collection 을 생성합니다
{
  _id: ObjectId('5f391af525b953000783db73'),
  partno: '123-aff-456',
  name: 'Awesometel 100Ghz CPU',
  qty: 102,
  cost: 1.21,
  price: 3.99
}
{
  _id: ObjectId('5f391b6425b953000783db75'),
  partno: '789-aff-012',
  name: 'Awesometel 1000Ghz CPU',
  qty: 1000,
  cost: 10.21,
  price: 30.99
}
{
  _id: ObjectId('5f391b9925b953000783db77'),
  partno: '012-afx-777',
  name: 'Awesometel 128GB Memory',
  qty: 100,
  cost: 100.21,
```

```
  price: 330.99
}
```

- 생성된 데이터를 통해 조회합니다

```
product = db.products.findOne({catalog_number: 1234});
product_parts = db.parts.find({_id: { $in : product.parts } } ).toArray();
```

1-3 one-to-many

- 실제 객체의 ObjectID 값을 리스트로 가지고 다시 조회하는 방식 2번 이상 호출이 필요함 (_id 도 은근히 큰 데이터 16mb)
  - ObjectId (12bytes)

```
// hosts collection 을 생성합니다
{
_id: ObjectId('5f3926f925b953000783db7d'),
name: 'goofy.example.com',
ipaddr: '127.66.66.66'
}
// logmsg collection 을 생성합니다
{
_id: ObjectId('5f39273f25b953000783db7f'),
time: ISODate('2020-08-02T11:10:09.032Z'),
message: 'cpu is on fire!',
host: ObjectId('5f3926f925b953000783db7d')
}
```

- 결과를 조회합니다
  - 부모 host 문서를 검색 합니다 // 유일한 index로 가정
  - 최근 5000개의 로그를 부모 host의 ObjectID를 이용해 검색

```
host = db.hosts.findOne({ipaddr : '127.66.66.66'});
last_5k_msg = db.logmsg.find({host: host._id}).sort({time :
-1}).limit(5000).toArray()
```

2 데이터 모델링 고급

2-1 양방향 참조 방식

- 삭제 시에 2개의 문서를 다시 삭제해야 하는 문제점이 있으므로 Atomic Delete 가 필요한 경우는 사용할 수 없습니다

```
// person
{
_id: ObjectId("AAAA"),
name: "Koala",
tasks [
  ObjectID("BBBB")
]
}
```

// tasks { _id: ObjectId("BBBB"), description: "Practice Jiu-jitsu", due_date: ISODate("2015-10-01"), owner: ObjectID("AAAA") }

2-2 비정규화를 통한 Many → One 참조 방식
* 자주 조회되지만, 자주 업데이트 되지 않는 라벨 혹은 이름의 경우 비정규화를 통해 조회 횟수를 줄일 수 있다
```javascript
// products - before
{
  name: 'Weird Computer WC-3020',
  manufacturer: 'Haruair Eng.',
  catalog_number: 1234,
  parts: [
    ObjectID('AAAA'),
    ObjectID('DEFO'),
    ObjectID('EJFW')
  ]
}

// products - after
{
  name: 'Weird Computer WC-3020',
  manufacturer: 'Haruair Eng.',
  catalog_number: 1234,
  parts: [
    { id: ObjectID('AAAA'), name: 'Awesometel 100Ghz CPU' }, // 부품 이름 비정규화
    { id: ObjectID('DEFO'), name: 'AwesomeSize 100TB SSD' },
    { id: ObjectID('EJFW'), name: 'Magical Mouse' }
  ]
}
```

2-3 비정규화를 통한 One → Many

- 대부분의 값들을 비정규화 하여 아주 유용하지만 해당 콜렉션을 유지하는 비용도 고려해야만 한다

```
// parts - before
{
_id: ObjectID('AAAA'),
partno: '123-aff-456',
name: 'Awesometel 100Ghz CPU',
qty: 102,
cost: 1.21,
price: 3.99
}
```

// parts - after { _id: ObjectID('AAAA'), partno: '123-aff-456', name: 'Awesometel 100Ghz CPU', product_name: 'Weird Computer WC-3020', // 상품 문서 비정규화 product_catalog_number: 1234, // 얘도 비정규화 qty: 102, cost: 1.21, price: 3.99 }

```
### 3 몽고디비 서버 기동 및 접속
* 최신 소스를 내려 받습니다
```bash
bash>
cd /home/ubuntu/work/data-engineer-intermediate-training
git pull
```
```

- 모든 컨테이너를 종료하고, 더 이상 사용하지 않는 도커 이미지 및 볼륨을 제거합니다

```
bash>
docker rm -f `docker ps -aq`
docker image prune
docker volume prune
```

- 몽고디비 및 몽고 익스프레스 인스턴스 기동 (http://localhost:8081)
  - 도커 컴포즈 옵션
  - --build : Build images before starting containers.
  - --force-recreate : Recreate containers even if their configuration and image haven't changed.
  - --renew-anon-volumes : Recreate anonymous volumes instead of retrieving data from the previous containers.
- 아래의 명령어가 실행됩니다
  - `bash docker-compose up --build --force-recreate --renew-anon-volumes -d`

```
bash>
./docker-compose-up.sh
```

- Mongo Express 에 접속하여 테스트 데이터를 직접 입력합니다

4 기본 명령어 예제 실습

- 기본 명령어 실습

```
// Log on MongoDB
bash>
docker-compose exec mongo mongo -u root -p
password: pass
```

mongodb> // Authenticate - root 로그인 시에는 필요 없습니다 db.auth("user", "pass");

// Show All Databases show dbs

// Change Database use testdb;

// Show All Collections, Users, Roles show tables; db.getCollectionNames();

show users; db.getUsers();

show roles;

// Create Collection - 별도로 생성할 필요 없이 db 명과 같이 문서를 생성하면 자동으로 생성됩니다 db.createCollection("foo");

// Insert Document(s) db.foo.insert({ field1: "string_value", field2: 1984 }); db.foo.insertMany([{ field1: "string_value" }, { field1: "string_value" }]);

// Update Whole Docuemnt - 문서 전체를 변경합니다 db.foo.save({"_id": new ObjectId(), field1: "value", field2: "value"}); db.foo.update({field2: 1984}, {field1: "modified", field2: "also modified"});

// Update Column of Document(s) - https://docs.mongodb.com/manual/reference/method/db.collection.update/ // db.foo.update(, ,); db.foo.update({}, {$set: {tag:"foo"}}, {multi:true}); // 모든 문서에 tag:"foo" 를 추가

// Display Document(s) // db.foo.find(,); db.foo.find().limit(10); db.foo.find({field1:"string_value"}, {tag:1});

// Remove Document(s) // db.foo.remove(,
) db.foo.remove({}, {multi:true})

```
### 5 모니터링 및 트러블슈팅
#### 5-1 CLI 를 통해 모니터링하는 방법
* mongotop → mongod 데몬이 어느 콜렉션에 얼마나 read / write 에 시간을 많이 사용하는 지 한 눈에 볼 수 있습니다
```bash
bash>
docker-compose exec mongo bash
mongotop --host localhost --port 27017 -u root -p pass --authenticationDatabase admin

2019-04-29T15:35:27.785-0400 connected to: 127.0.0.1
                    ns    total    read    write    2019-04-29T15:35:57-04:00
    admin.system.roles      0ms     0ms      0ms
    admin.system.users      0ms     0ms      0ms
  admin.system.version      0ms     0ms      0ms
config.system.sessions      0ms     0ms      0ms
     local.startup_log      0ms     0ms      0ms
  local.system.replset      0ms     0ms      0ms
```

- mongostat → 현재 기동 중인 mongod, mongos 등의 프로세스의 상태를 한 눈에 볼 수 있습니다

  ```
  bash>
  $> mongostat --host localhost --port 27017 -u root -p pass --
  authenticationDatabase admin
  ```

insert query update delete getmore command dirty used flushes vsize res qrw arw net_in net_out conn time 991 *0 *0 *0 0 2|0 3.4% 4.5% 0 2.90G 297M 0|0 0|0 12.9m 84.2k 2 Oct 6 09:45:37.478 989 *0 *0 *0 0 2|0 3.6% 4.7% 0 2.91G 310M 0|0 0|0 12.9m 84.1k 2 Oct 6 09:45:38.476 988 *0 *0 *0 0 1|0 3.7% 4.8% 0 2.92G 323M 0|0 0|0 12.8m 83.8k 2 Oct 6 09:45:39.481 976 *0 *0 *0 0 2|0 3.9% 5.0% 0 2.94G 335M 0|0 0|0 12.7m 83.7k 2 Oct 6 09:45:40.476

```
  - inserts / query / update / delete / getmore / commands : 초당 유입 / 조회 / 변경 / 삭제 / 커서 조회 되는 문서 / 명령어 수
  - dirty : cache 의 내용과 disk 의 내용이 다른 경우의 비율
  - used : 몽고디비의 cache 가 사용되는 비율
  - flushes : wiredTiger 엔진이 체크포인트를 통해 트리거링 되는 횟수
  - vsize : 가상 메모리 사용 mb 크기
  - res : 사용되고 있는 레지던트 mb 크기
  - qrw : 클라이언트가 몽고디비 데이터를 읽기 (r) 혹은 쓰기 (w) 를 위해 대기하는 큐 (q) 의 길이
  - arw : 액티브 (a) 클라이언트의 읽기 (r) 혹은 쓰기 (w) 오퍼레이션들의 수
  - net_in : 인 바운드 네트워크 트래픽의 바이트 수
  - net_out : 아웃 바운드 네트워크 트래픽의 바이트 수
  - conn : 현재 열려 있는 연결 수
  - time : 측정된 시간

#### 5-2 환경설정
* 적절한 메모리 크기를 확인 - storage.wiredTiger.engineConfig.cacheSizeGB : 몽고디비 캐시 크기 결
```
```

정
  * 몽고서버 실행 시에 아래와 같이 conf 파일을 지정합니다 (mongod --config /etc/mongod.conf)
  * [몽고디비 Configuration](https://docs.mongodb.com/manual/reference/configuration-options/)
  * [docker-library.Dockerfile](https://github.com/docker-library/mongo/blob/master/3.6/Dockerfile)

```bash
bash>
cat /etc/mongod.conf

storage:
   dbPath: <string>
   journal:
      enabled: <boolean>
      commitIntervalMs: <num>
   directoryPerDB: <boolean>
   syncPeriodSecs: <int>
   engine: <string>
   wiredTiger:
      engineConfig:
         cacheSizeGB: <number> // MAX( 0.5 * (RAM - 1GB), 256 MB)
         journalCompressor: <string>
         directoryForIndexes: <boolean>
         maxCacheOverflowFileSizeGB: <number> // deprecated in MongoDB 4.4
      collectionConfig:
         blockCompressor: <string>
      indexConfig:
         prefixCompression: <boolean>
   inMemory:
      engineConfig:
         inMemorySizeGB: <number>
   oplogMinRetentionHours: <double>
```

- 도커 설정 변경 - docker-compose.yml

```
volumes:
     - ./mongodb/conf/mongod.conf:/etc/mongod.conf
     - ./mongodb/data/db:/data/db
```

5.3. 자주 발생하는 문제점들

- 트러블 슈팅의 첫 걸음 로그 조회

```
bash>
sudo grep mongod /var/log/mongodb/mongod.log
sudo grep score /var/log/mongodb/mongod.log
```

- 자주 발생하는 오류
  - Lack of memories - cache 크기가 너무 작게 잡혀있거나, 너무 많은 데이터를 조회 하는 경우
  - 100% CPU load and slow queries - index 가 잡혀있지 않아 쿼리가 계속 늘어나는 상황
  - Mongo opens too many connections - 진짜 유저가 많거나, slow query 가 몰리는 상황
  - MongoDB running out of disk space - ssd 와 같은 제한된 storage 의 경우 주의가 필요함
- 디버깅 순서

- test 데이터베이스에 system.profile 콜렉션이 생성되고 profile 수준에 해당하는 쿼리문들이 생성됩니다. 이를 통해 디버깅이 가능합니다

```
db.setProfilingLevel(1, { slowms: 20 })
db.system.profile.find().limit(10).sort( { ts : -1 } ).pretty()
```

- 예를 들어 3초 이상 시간이 소요되고 있는 오퍼레이션 찾기 // full collection scan

```
db.currentOp({"secs_running": {$gte: 3}})
```