Slow Query 탐색

1. MongoDB 에서 slow query 탐색

MongoDB 에서는 자세한 정보(모든 CRUD 작업 및 구성 변경 포함)를 수집하는 기본 제공 프로파일러 MongoDB Database Profiler 를 제공하고 있다.

이 정보는 admin DB 의 system collection 에 저장된다.

Profiler 의 상태를 확인하기 위해서는

```
> db.getProfilingStatus()
```

명령어를 입력하면 확인할 수 있다.

result :

Level 1	The profiler is off and does not collect any data. This is the default profiler level.
Level 2	The profiler collects data for operations that take longer than the value of slowms.
Level 3	The profiler collects data for all operations.

MongoDB 에서 conf 파일 수정을 통해 slow Query 로 선정할 기본 파라미터를 제공한다.

Profile Level 을 설정하기 위해서는

```
> db.setProfilingLevel( 2 )
```

명령어를 입력하면 가능하다

* 여기서 주의할 점은 특히 Level 2 로 설정 되어있는 Profiling Level 은 데이터베이스 성능에 영향을 미치고 이것을 주의 깊게 이해하고 평가하여야 한다

실제 데이터 베이스에서는 이런식으로 사용할 수 있다.

```
> db.system.profile .find({ op: { $eq: "command" }}) .sort({ millis: -1 }) .limit(10) .pretty();
```

admin.system 컬렉션에 저장되어있는 log 를 뒤져 제일 오랜 시간이 걸린 command 를 찾는다. 혹은

```
> db.system.profile .find({ millis: { $gt: 30 }}) .pretty();
```

와 같은 명령어로 30ms 보다 큰 query 를 찾아낼 수 있다.

```
> db.system.profile .find({ "nreturned": { $gt: 1 }}) .pretty();
```

이 명령어로는 모든 index 를 조회한 query 를 확인 할 수 있다. (full index range scan or full index scan)

```
> db.system.profile .find({ "nscanned" : { $gt: 100000 }}) .pretty();
```

이 명령어는 document 단위로 몇 개 이상 조회했는지를 파악하고 해당 query 를 조회할 수 있다.

Query Perfomance 를 실시간으로 확인해야한다면

```
> db.currentOp( true )
```

를 사용할 수 있다. 응용해서 ,

```
> db.currentOp({ "active" : true, "secs_running" : { "$gt" : 3 }})
```

처럼 사용 할 수 있는데 3 초 이상 실행된 operation 을 조회할 수 있다.

2. slow query 개선 전략 조회

MongoDB 는 EXPLAIN helper 를 3 가지 방법으로 제공한다,

- The db.collection.explain() Method
- The cursor.explain() Method
- The explain Command

각 Method 에는

- queryPlanner → MongoDB will run its query optimizer to choose the winning plan and return the details on the execution plan without executing it.
- executionStats → MongoDB will choose the winning plan, execute the winning plan, and return statistics describing the execution of the winning plan.
- allPlansExecution → MongoDB will choose the winning plan, execute the winning plan, and return statistics describing the execution of the winning plan. In addition, MongoDB will also return statistics on all other candidate plans evaluated during plan selection.

이 세가지의 상세 모드가 주어진다.

db.collection .explain("executionStats") .aggregate([{ $match: { col1: "col1_val" }}, { $group: { _id: "$id", total: { $sum: "$amount" } } }, { $sort: { total: -1 } }])

이런식으로 collection.explain() 메소드와 executionStats 를 혼합해서 사용하는 식이다.

결과로는

Query Planner (queryPlanner) section : query optimizer 에서 선택한 plan 의 세부사항을 보여준다.

Execution Statistics (executionStats) section : winning plan 의 실행 내용을 보여주는데 , 실제 실행되어 반환된 winning plan 을 보여준다.

Server Information (serverInfo) section : MongoDB instance 에 대한 일반적인 정보를 보여준다.

우리는 결과에 대한 값을 원할 뿐 실제 monoDB 내부에서 어떤식으로 최선의 쿼리 실행 계획을 세우거나 실행하는지는 잘 모른다.

아래의 다섯가지 과정이 실제 delay 가 생길 수 있는 부분으로 MongoDB 에서 기술하고 있다.

- COLLSCAN for a collection scan
- IXSCAN for scanning index keys
- FETCH for retrieving documents
- SHARD_MERGE for merging results from shards
- SHARDING_FILTER for filtering out orphan documents from shards

---

일반적으로 MongoDB 에서 Slow Query 라고 하면 100ms 이상 시간이 사용된 query 이다

이 default 로 지정된 100ms 를 수정하기 위해서는 mongod.conf 파일 을 수정하면 된다.

```
# mongod.conf

operationProfiling:
   mode: <string>
   slowOpThresholdMs: <int>
    slowOpSampleRate: <double>
   filter: <string>
```

기본 default 는 100ms 로 지정되어있고 conf 파일을 수정하는 것이 아닌

> db.setProfilingLevel()

명령어를 이용하여 수정할 수도 있다.