3. Index

1. Index 기본 개념
2. MongoDB Index 생성 및 관리
3. Index 사용

1. Index의 기본 개념

❖ Index

- index ➔ 디비의 검색을 빠르게 하기 위해 미리 데이터의 순서를 정리해두는 과정
- MongoDB는 고정된 스키마는 없지만 원하는 데이터 필드를 인덱스로 지정하여 검색결과를 빠르게 하는것이 가능
- NOSQL에서도 index를 잘 설계해야 최대의 효율이 가능
- MongoDB를 효율적으로 사용하고 하드웨어의 성능을 최대로 끌어내려면 index의 종류와 사용 방법에 이해 필요
- MongoDB는 B트리 구조로 index를 구현
- 고유 index, 희소 index, 다중 키 index 지원
- 복합 index, 단일 index 지원

❖ Index의 개념

- index는 도큐먼트를 쿼리해오기 위한 작업량을 줄임
  - ✓적당한 index가 없으면 질의 조건을 만족할 때까지 모든 도큐먼트를 순차적으로 스캔
- 한 쿼리당 하나의 index만 유효
- 두 개의 index가 필요하다면 복합 index를 사용
  - ✓a와 b필드로 구성된 복합 index를 가지고 있다면 a에 대해 단일 index는 제거해도 됨
  - ✓복합 index에서 키의 순서는 매우 중요
- _id는 기본적으로 생성되는 index로 도큐먼트를 가르키는 유일한 키값으로 사용 (도큐먼트에 빠르게 접근하기 위해서 각 _id는 index로 관리됨)

❖ Index 효율

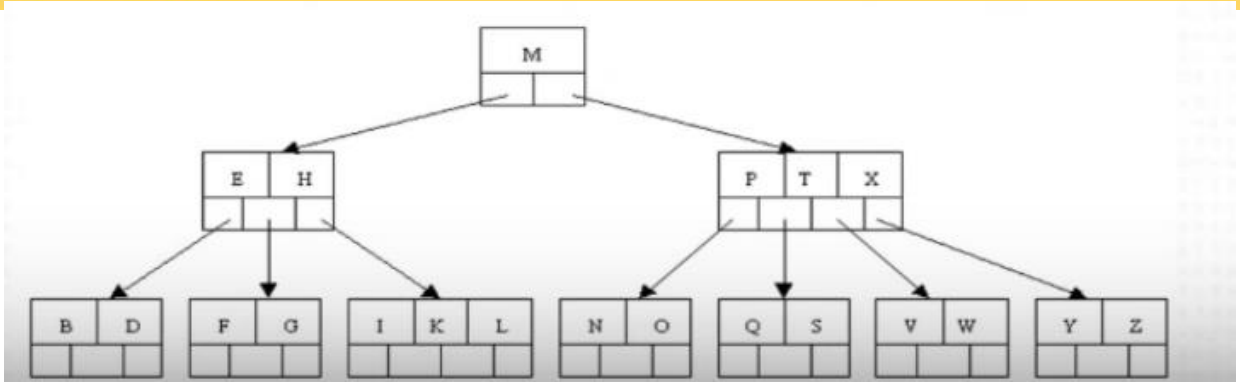
- 쿼리 성능 향상을 위해 무한히 index를 추가하는 것은 불가능
- 모든 index에는 결국 유지비가 소요됨
- 어떤 데이터가 도큐먼트에 추가되거나 수정될 때마다 그 컬렉션에 대해 생성된 index도 그 새로운 도큐먼트를 포함시키도록 수정되어야 함
- 최악의 경우에는 결국 데이터를 다시 정렬해야 하는 상황 발생
- index는 읽기 위주의 어플리케이션에서 유용
- 읽기보다 쓰기 작업이 많다면 어느 정도 index를 포기하거나 index를 위한 컬렉션을 따로 운영해야 함

❖ Index 효율

- Mongodb는 기동시 모든 데이터 파일을 메모리에 매핑함
- 모든 도큐먼트, 컬렉션, 인덱스를 포함하는 모든 데이터 파일이 페이지(page)라고 부르는 4kb정도의 청크 단위로 운영체제에 의해 램에 적재
- 램의 모든 데이터를 수용만 할 수 있다면 디스크 액세스 횟수 최소화 가능
- 모든 데이터를 수용하지 못하면 페이지 폴트가 자주 발생하게 되고 운영체제가 디스크를 빈번하게 액세스하게 됨으로 인해 읽기/쓰기 연산 지연 발생
- 스래싱(thrashing) : 모든 데이터를 디스크에서 액세스 해야하는 경우 굉장한 성능 저하를 초래
- 최소한의 인덱스가 메모리에 위치 할 수있도록 최소화 될 필요가 있음
- 복합인덱스는 더 많은 공간을 필요로 함을 고려해야 함

❖ B트리

- Mongodb는 내부적으로 B트리(B-Tree) 알고리즘을 이용하여 인덱스 구성
- B트리는 1970년대 후반부터 데이터베이스의 레코드와 인덱스로 활발하게 사용되고 있음
- B트리는 트리구조와 유사한 데이터 구조
- 트리에서 각 노드는 여러 개의 키를 갖는 것이 가능
- Mongodb에서 사용하는 B트리는 새 노드에 대해 8,192바이트를 할당함 (각 노드가 수백 개의 키를 가질수 있음을 의미)
- 인덱스키의 평균 크기에 따라 달라질 수도 있는데, 보통 키의 평균적인 크기는 30바이트 안팎임



2. Index 생성 및 관리

- 인덱스 생성 및 조회

```
> db.emp.ensureIndex ({ eno: 1 }, { unique: true });
                                  ← 1 (Asc), -1 (Desc)
> db.emp.ensureIndex ({ job : -1 });
```

```
> db.emp.getIndexes()
{           "v" : 1,
            "key" : {
                  "eno" : 1
            },
            "unique" : true,
            "ns" : "test.emp",
            "name" : "eno_1"
      }

> db.system.indexes.find()
{ "v" : 1, "key" : { "eno" : 1 },
  "unique" : true, "ns" : "test.emp", "name" : "eno_1" }
```

1. Index 생성 및 관리

❖ 인덱스 삭제와 재구성

```
db.employees.getIndexes()
db.employees.dropIndex({ename:1})

db.employees.createIndex({com:1})
db.employees.reIndex()
db.runCommand({reIndex:'employees'})
```

- MongoDB의 인덱스 특징
  - 대소문자 엄격하게 구분
  - Document를 업데이터 할때 index key만 변경, 변경된 Document크기가 기존 EXTENT 더 큰 경우 더 큰 공간으로 마이그레이션 될 수 있어 성능 저하 발생
  - sort() 절과 limit 절을 함께 사용하는 것이 성능이 도움이 됨

2. Index 생성 및 관리

❖ 인덱스의 종류

- Non-Unique/Unique Index
- Background Index
- Covered Index
- DropDups Index
- Sparse Index
- TTL Index
- GeoSpatial(2d) Index
- GeoSpatial(2dsphere) Index
- GeoHayStack Index
- Text Index
- Hashed Index

2. Index 생성 및 관리–주요 인덱스–1

INDEX 타입	설 명
Non Unique Index (Single Key Index Compound Key Index)	하나 또는 하나 이상의 중복 값을 가진 Field로 구성되는 Index 타입으로 가장 대표적인 Balance Tree Index 구조로 생성된다. (예) db.things.ensureIndex({"city": 1}) db.things.ensureIndex({ deptno:1, loc:-1})
Unique Index	Index가 생성되는 Field가 유일한 속성 값을 가진 Index 타입이다. (예) db.things.ensureIndex({fname: 1, lname: 1}, {unique: true})
Sparse Index	하나 이상의 필드에 Null 값을 가진 데이터가 대부분이고 드물게 어떤 데이터를 값을 가지고 있는경우에 생성하는 효율적이다. (예) db.people.ensureIndex({title : 1}, {sparse : true}) db.people.save({name:"Jim"}) db.people.save({name:"Sarah", title:"Princess"}) db.people.find().sort({title:1}) {name:"Sarah", title:"Princess"}

2. Index 생성 및 관리– 주요 인덱스

INDEX 타입	설 명
Background Index	일반적으로 Index의 생성은 데이터베이스 전체의 성능 지연 현상을 유발시킬 수 있다. V1.3.2부터 Background에서 Index를 생성할 수 있다. (예) db.people.ensureIndex({ idate : 1}, {background : true})
Covered Index	여러 개의 Field로 생성된 Index를 검색할 때 Index 만의 검색 만으로도 조건을 만족하는 Document를 추출할 수 있는 타입이다. (예) db.users.ensureIndex({ username : 1, password : 1, roles : 1}); db.users.save ({username: "joe", password: "pass", roles: 2}) db.users.save ({username: "liz", password: "pass2", roles: 4}) db.users.find ({username: "joe"}, {_id: 0, roles: 1}) { "roles" : 2 } db.users.find ({username: "joe"}, {_id: 0, roles: 1}).explain() { "cursor" : "BtreeCursor username_1_password_1_roles_1", ... "indexOnly" : true }
DropDups Index	동일한 값이 여러 개 저장되어 있는 필드에 DropDups Index를 생성하면 최초 입력된 Document 만 남고 나머지 Document는 제거된다. (예) db.people.ensureIndex({ idate : 1}, {dropdups: true})
GeoSpatial Index	좌표로 구성되는 2차원 구조로 하나의 Collection에 하나의 2D Index를 생성할 수 있다.(다음 페이지에서 자세히 소개됨)

3. Index 사용

- Single-key Index & Compound Index

```
use test
db.employees.getIndexes()  # Single-key 인덱스
db.employees.createIndex({empno:1})

db.employees.createIndex({empno:1, deptno:-1}) # Compound 인덱스
db.employees.getIndexes()

db.employees.createIndex({deptno:1})
db.employees.find({deptno:10}).pretty()
db.employees.find({deptno:10}).explain()

db.employees.find({deptno:10}).sort({empno:-1})
db.employees.find({deptno:10}).sort({empno:-1}).explain()
db.employees.dropIndex({empno:1})
```

1. Index 생성 및 관리

❖ Nun-Unique Index & Unique Index

```
db.employees.createIndex({empno:1},{unique:true})
db.employees.createIndex({enane:1})
db.employees.getIndexes()
db.employees.dropIndex({empno:1})

db.employees.insert({empno:7369, ename:"ADAM"})
db.db.employees.dropIndex({empno:1})
db.employees.find({empno:7369}).pretty()
db.employees.createIndex({empno:1},{unique:true})
```

1. Index 생성 및 관리

❖ Sparse 인덱스

- 검색대상이 되는 필드가 전체 컬렉션에서 차지하는 밀도가 낮은 경우 생성

```
db.employees.dropIndex({comm:1})
db.employees.createIndex({comm:1},{sparse:true

}) db.employees.find({comm:300}).pretty()
db.employees.find({comm:300}).explain()
db.employees.dropIndex({comm:1})
```

1. Index 생성 및 관리

- Partial 인덱스
  - 인덱스 필드에 추가 조건을 주어 인덱스를 생성
  - 인덱스 크기를 줄일 수 있다.

```
db.employees.createIndex({deptno :1,ename:1},
                {partialFilterExpression: {sal:{$gt:500} } })
db.employees.getIndexes()
db.employees.find( {deptno: 10, sal : { $gte:2500} } ).pretty()
db.employees.find( {deptno: 10, sal : { $gte:2500} } ).explain()
```

1. Index 생성 및 관리

- background 인덱스
  - 대량의 인덱스 생성시 성능저하 현상 유발
  - 인덱스 생성시 활용가능한 system 자원을 이용에 인덱스 작성

```
db.employees.createIndex( {hiredate: 1},{background: true})
db.employees.find({hiredate: "20-02-1981"})
db.employees.find({hiredate: "20-02-1981"}).explain()
db.employees.find({deptno: 10, ename : "CLARK"}, {_id:0 , ename:1}).explain()
```

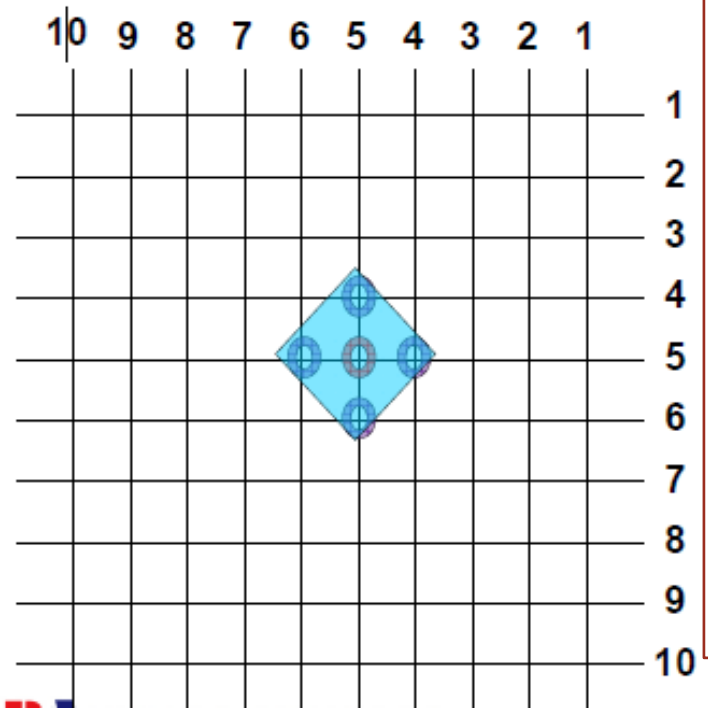
1. Index 생성 및 관리

- convered 인덱스
  - 검색에 필요한 필드를 모두 인덱스 필드로 하여 인덱스 생성
  - 검색을 할때 실제 데이터에 접근하지 않고 인덱스만으로 데이터를 검색 가능

```
db.employees.createIndex( {deptno: 1, name:1})
db.employees.find({deptno: 10, ename : "CLARK"}, {_id:0 , ename:1}).explain()
```

1. Index 생성 및 관리

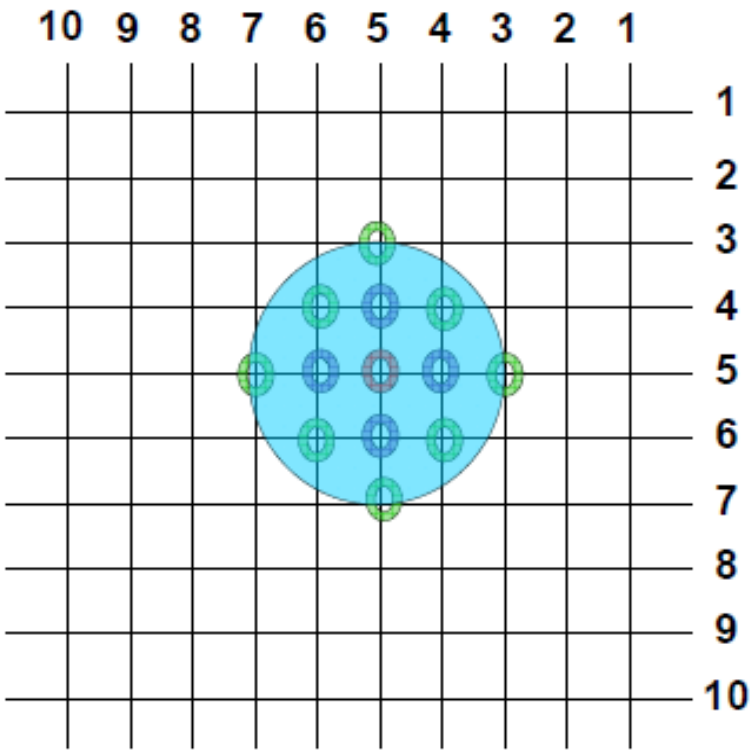
- GeoSpatial INDEX
  - 좌표에 의해 구성되는 2차원 구조로 하나의 **Collection**에 하나의 **Index**를 생성할 수 있다




```
for(var i=0;i<100; i++)
        db.spatial.insert({
                pos : [i%10, Math.floor(i/10)] } )
db.spatial.ensureIndex({ pos : "2d"})
db.spatial.find({pos : {$near : [5,5] }  },{_id:0}).limit(5)
{ "pos" : [ 5, 0 ] }
{ "pos" : [ 4, 0 ] }
{ "pos" : [ 6, 0 ] }
{ "pos" : [ 3, 0 ] }
{ "pos" : [ 7, 0 ] }
```

2. Index 생성 및 관리

❖ $CENTER

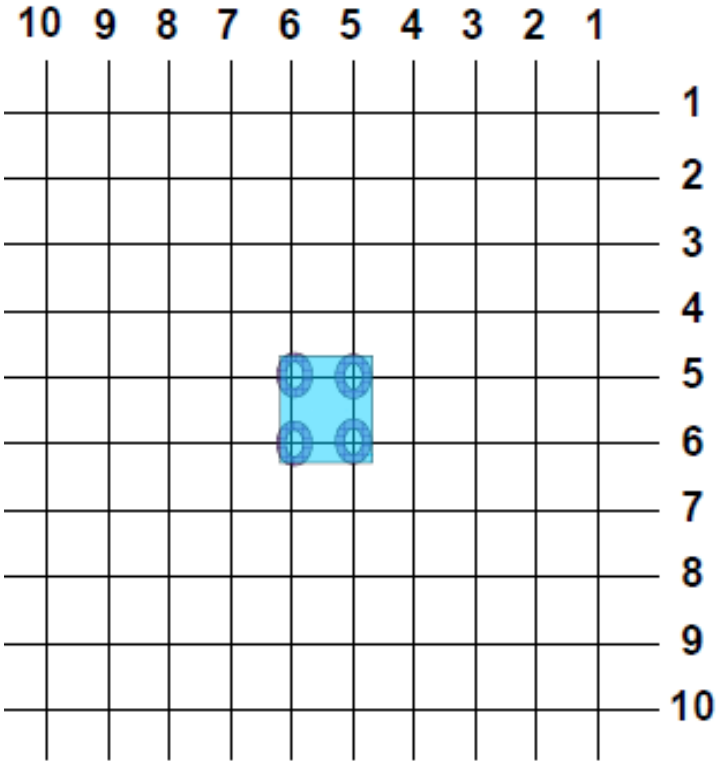



```
> db.square.find(
   { pos : { $within : { $center : [ [5, 5], 2 ] }}},
    { _id : 0 })
```

```
{ "pos" : [ 5, 5 ] }
{ "pos" : [ 5, 4 ] }
{ "pos" : [ 5, 3 ] }
{ "pos" : [ 4, 5 ] }
{ "pos" : [ 3, 5 ] }
{ "pos" : [ 4, 4 ] }
{ "pos" : [ 4, 6 ] }
{ "pos" : [ 5, 6 ] }
{ "pos" : [ 5, 7 ] }
{ "pos" : [ 6, 4 ] }
{ "pos" : [ 6, 5 ] }
{ "pos" : [ 7, 5 ] }
{ "pos" : [ 6, 6 ] }
```

1. Index 생성 및 관리

❖ $BOX




```
> db.square.find(
  { pos : { $within : { $box : [  [5, 5], [6, 6] ] }}},
   { _id : 0 })
```

{ “pos” : [5, 5] }

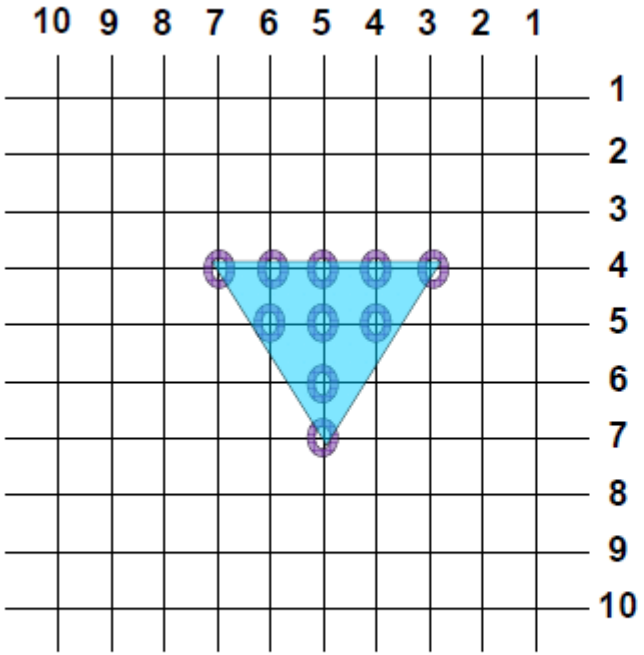
{ “pos” : [5, 6] }

{ “pos” : [6, 5] }

{ “pos” : [6, 6] }

1. Index 생성 및 관리

❖ $POLYGON




```
> db.square.find({ pos :  { $within :
  { $polygon : [[3, 4], [5, 7], [7, 4]] }}}, { _id : 0 })
{ "pos" : [ 5, 5 ] }
{ "pos" : [ 6, 5 ] }
{ "pos" : [ 7, 4 ] }
{ "pos" : [ 6, 4 ] }
{ "pos" : [ 5, 7 ] }
{ "pos" : [ 5, 6 ] }
{ "pos" : [ 3, 4 ] }
{ "pos" : [ 4, 4 ] }
{ "pos" : [ 4, 5 ] }
{ "pos" : [ 5, 4 ] }
```

1. Index 생성 및 관리

❖ Multi-Location Documents

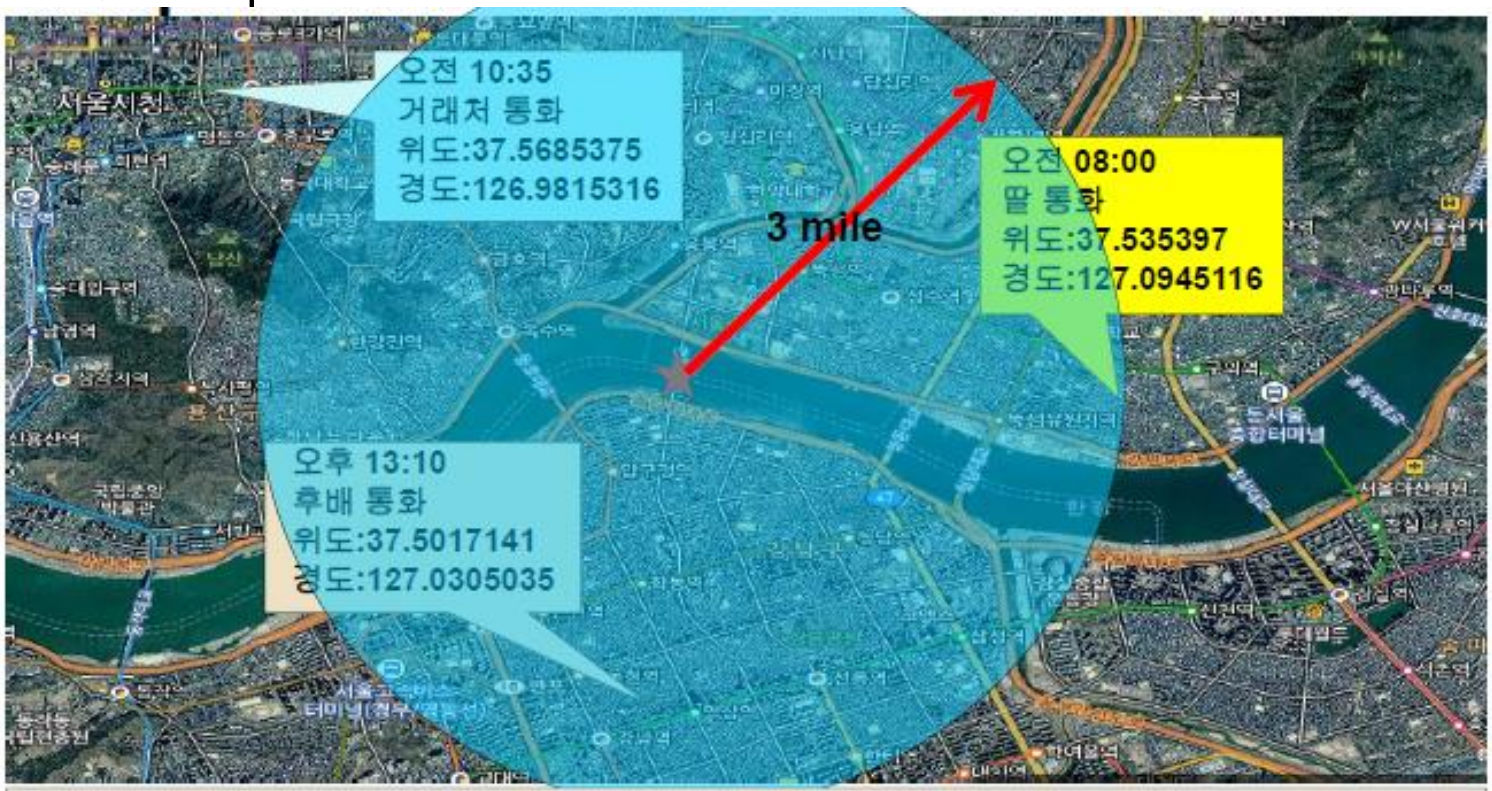

```
> db.tel_pos.save ({ mobile_no : 01038641858,
                     last_pos    : [[ 127.0945116, 37.535397],
                                    [ 126.9815316, 37.5685375],
                                    [ 127.0305035, 37.5017141]]})
```

Multi-Location Documents 검색 예:

```
db.tel_pos.save( { mobile_no : "01038631858",
            last_pos : [ [127.0945116,37.5353970],
                         [126.9815316,37.5685375],
                         [127.0305035, 37.5017141]    ]  } )

db.tel_pos.save( {mobile_no : "01075993678",
            last_pos : [ [127.1353452,37.4576521],
                         [127.1359081,37.4512311],
                          [125.7823091, 36.3339801]    ] })

db.tel_pos.save( {mobile_no : "01071229021",
            last_pos : [ [126.3411234,36.1098761],
                         [124.3410922,37.3409901],
                         [127.2223331, 37.0912090]    ] })

db.tel_pos.ensureIndex( {last_pos : "2d" } )

b.tel_pos.find( { last_pos : { $within : { $centerSphere :
[[ 127.0352915,37.5360206 ],30/3963]  }} },  {_id : 0, mobile_no : 1, last_pos :
1 }).pretty()
```

1. Index 생성 및 관리

❖ $nearSphere

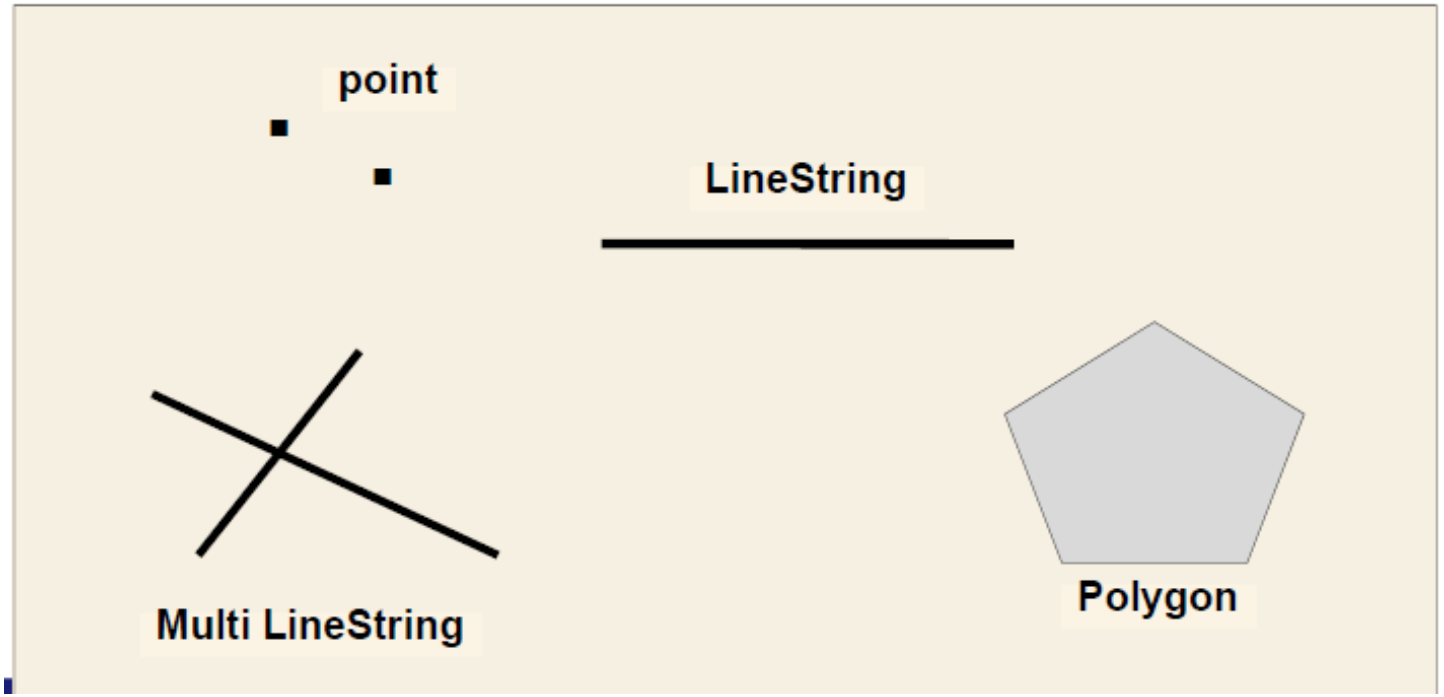



```
> db.tel_pos.find( { last_pos :  { $nearSphere :
  [[ 127.0352915, 37.5360206]] }}, {_id:0, last_pos : 0 })
```

← 성수대교를 기준으로 반경 3 Mile 이내

GeoMetry 인덱스

- geoJSON은 직선 또는 곡선의 교차에 의하여 이루어지는 추상적인 구조나 다각형(Polygon)과 같은 기하학(geoMetry)구조를 나타냄
- 기하학 구조에 만들어지는 인덱스를 GeoMetry 인덱스라고 함.
- GeoMetry 인덱스 타입



❖ point 타입

```
db.position.ensureIndex({ loc : "2dsphere"})
db.position.insert( { "_id" : “m91",
"loc" : { "type" : "Point",
          "coordinates" : [127.0980748, 37.5301218] },
          “name"  : [ "name=동서울 터미널” ] })
db.position.insert( { "_id" : “m90",
"loc" : { "type" : "Point",
          "coordinates" : [127.0952154, 37.5398467] },
          “name"  : [ "name=강변역”] })
db.position.insert( { "_id" : “m89",
"loc" : { "type" : "Point",
          "coordinates" : [127.0742172, 37.5419541] },
          “name"  : [ "name=건대입구역 ”] })
```


```
db.position.find( { loc : { $near : { $geometry : { type : "Point" ,
 coordinates: [127.1058431, 37.5164113] }}, $maxDistance : 2000 } })
```