8. INDEX

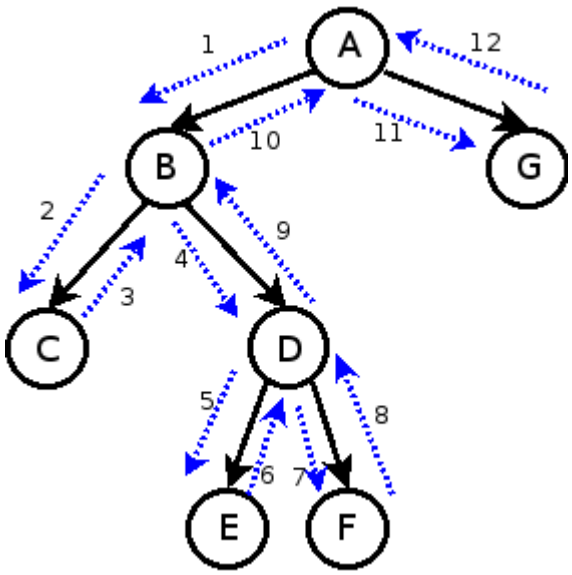
INDEX

. Index는 MongoDB에서 데이터 쿼리를 더욱 효율적 조회

. 인덱스가 없이는, MongoDB는 collection scan – 컬렉션의 데이터를 하나하나 조회 .

. document의 갯수가 매우 많다면, 많은 만큼 속도 저하 - 이 부분을 향상시키기 위하여 인덱스를 사용하면 더 적은 횟수의 조회로 원하는 데이터를 조회

. Document의 필드(들) 에 index 를 걸면, 데이터의 설정한 키 값을 가지고 document들을 가르키는 포인터값으로 이뤄진 B-Tree를 생성



인덱스 생성과 관리

. DB 문자열에 컬렉션 명 그리고 createIndex 메소드로 생성할 수 있음

. 인덱스 종류와 개수를 확인할 때는 getIndexes 메소드로 실행

. 오름차순으로 설정하고 싶다면 1을, 내림차순으로 설정하고 싶다면 -1

```
>db.<collection>.getIndexes()
>db.monsters.createIndex({ name: ±1 });
```

인덱스 재 구성과 삭제

. MongoDB에서 Index의 대소문자는 엄격히 구분

. Document를 Update할 때 해당 Index Key말 변경되지만 변경되는 Document 크기가 기존 EXTENT 공간 크기보다 큰 경우에는 더 큰 EXTENT 공간으로 마이그레이션이 될 수 있기 때문에 성능 저하 현상이 발생 할 수 있음

. sort()절과 limit()절은 함께 사용하는 것이 성능에 도움됨

. B트리 인덱스를 이용하면 빠른 검색이 가능하지만 데이터 생성, 변경, 삭제 시 빠른 성능이 보장 되지 않음

인덱스 크기

- MongoDB에서 구현된 B-Tree는 새 노드에 8KB(8192Byte)를 할당함
- (노드 할당크기 - 노드 당 오버헤드) / (키의 크기 - 키 당 오버헤드) = 노드 당 가질 수 있는 키의 개수

  e.g. 키의 평균적인 크기가 30Byte 안팎이라면, (8192 - 40) / (30 - 18) = 169.8(노드 당 대략 170개의 키를 가질 수 있음)
- B-Tree 노드는 일반적으로 60%만을 사용하도록 의도적으로 기본 설정되어 있다

인덱스 특성

. 조회

- 인덱스는 처음 db를 만들 때 자동으로 생성되는 system.indexes컬렉션에 저장
- 여러 인덱스를 조합하여 조회가 가능

```
> db.monsters.find({ name: 'Slime', hp: { $gt: 10 } })
```

. 인덱스의 단점으로는 인덱스 자체가 용량을 차지하여 너무 많은 인덱스를 사용해 용량을 낭비하지 않도록 함

. 인덱스 & 작업 중인 데이터를 RAM에서 다 처리하지 못하는 경우, 모든 인덱스가 적합하더라도 쿼리 처리 성능이 저하될 수 있음

1. WiredTiger 스토리지 엔진에서는 모든 도큐먼트, 컬렉션, 인덱스 포함 데이터 파일이 페이지page라는 4KB의 청크로 OS에 의해 RAM으로 스왑된다
2. 해당 페이지의 데이터가 요청될 때마다 OS는 그 페이지가 RAM에 있는지 확인한다—CUD 연산은 RAM에서 발생하고 수정사항은 OS가 비동지적으로 Disk에 반영한다
3. 없다면 페이지 폴트 Page Fault 예외를 발생시키고, 메모리 관리자가 Disk로부터 페이지를 RAM으로 불러들인다
4. RAM이 충분하면 필요한 모든 데이터파일이 메모리에 로드된다
5. 데이터를 다 수용하지 못하면 점점 페이지 폴트가 많이 발생하게 되고OS가 Disk를 액세스하는 횟수가 많아지면서 CRUD연산이 매우 느려지게 된다. **최악의 경우 Thrashing이 발생한다:**데이터 크기가 가용한 RAM 크기보다 훨씬 커서 모든 CRUD에 대해 Disk Access가 필요해지는 상황이 발생할 수도 있음

. 자주 발생하는 쿼리에 대한 커버링 인덱스 Covering Index를 생성하여 유용하게 사용 가능

. 인덱스는 인덱스에 포함된 데이터와 함께 RAM에 별도로 저장되며 Clustered되지 않는다 MongoDB에서는 클러스터 인덱스를 사용하지 않는다(인덱스 순서대로 데이터 순서를 정렬하지 않는다)\ 인덱스 키의 순서는 데이터의 실제 순서와 관계가 없다:모든 도큐먼트가 데이터 세트의 어느 위치에나 있을 수 있다, 지역성Locality을 보장하지 않는다

인덱스 구축

. 데이터가 먼저 삽입되고 난 후에 인덱스를 구축하게 되는 경우

1. DB를 MongoDB로 Migration할 때: 전송해 온 이후에 인덱스를 생성하면 균형&압축된 인덱스 생성 가능
2. 새로운 쿼리에 인덱스를 최적화해야 할 때: 기능 추가 혹은 변경 시

. 대량의 데이터가 있는 경우, 인덱스를 생성하는데 몇시간 ~ 며칠까지도 걸릴 수 있다진행 상황은 MongoDB 서버 로그를 확인해 보거나, `db.currentOp()` 로도 확인 가능함

인덱스 구축 단계

1. 인덱스할 필드 값을 정렬함 → B-Tree에 추가정렬의 진턱도는 도큐먼트의 총 수 대비 정렬된 도큐먼트의 개수로 나타난다
2. 정렬된 값들을 인덱스로 삽입한다인덱스 구축 완료까지 소요된 시간을 system.indexes 컬렉션에 삽입한다

. **인덱스 구축 시 문제점**

인덱스 생성 시, 해당 데이터 베이스가 쓰기 Lock이 걸려 읽기/쓰기 모두가 불가능하다

해결책

1. 백그라운드 인덱싱백그라운드에서 인덱스 구축하면, 다른 읽기/쓰기 요청 시 잠시 구축을 멈춘다.

   트래픽이 최소화되는 시간에 인덱스를 구축한다면 좋은 방식

   ```
   db.collection.createIndex({'idx_field': 1}, {background: true})
   ```

2. 오프라인 인덱싱1) 하나의 복제 노드를 오프라인으로 바꾸고, 인덱스 구축 후, 마스터에서 그 동안의 업데이트를 받아 완료함2) 새로운 인덱스로 업데이트 된 노드를 프라이머리로 변경하고, 다른 복제 노드들도 오프라인 인덱스 구축복제 oplog가 충분히 크다는 가정을 전제로 사용하는 방법
3. 백업인덱스 구축은 어려우므로 백업이 필요함, but 모든 백업이 인덱스 데이터까지 다 포함하지는 않음→ MongoDB 데이터 파일 자체를 백업해야 함
4. Defragmentation 비단편화다량의 데이터 CRUD로 인덱스의 단편화가 심해졌을 때: B-Tree가 스스로 재구성할 수준을 넘게 될 때주로 데이터 크기보다 인덱스의 크기가 훨씬 더 클 때 → 인덱스가 RAM을 필요 이상으로 차지함

   ```
   //해당 컬렉션의 모든 인덱스를 재구축함
    db.collection.reIndex()
   ```

   - 마찬가지로 재구축 되는동안 DB는 Lock이 걸림 → 오프라인 방식이 최적의 활용법

인덱스의 종류

. 싱글 키 인덱스

하나의 필드 키를 사용하는 인덱스

하나의 쿼리에서 단일 키 인덱스를 두 개 사용하면, 쿼리 옵티마이저가 그 중 제일 효율적인 인덱스를 선택함

→ 그러나 그 결과가 항상 최선인 것은 아니다.

. 복합 키 인덱스

하나 이상의 필드를 키로 사용하는 인덱스

복합 인덱스에서는 순서가 중요함:

. 첫 번째로 오는 인덱싱 조건은 '일치Equality' 이어야 하고

. 범위를 지정한 키는 그 다음에 와야 한다

. Non-Unique 인덱스 vs Unique인덱스

. 해당 도큐먼트에서 인덱스 필드 값의 고유한지 확인함실제로 _id 필드가 고유 기본 키라는 것을 보장하기 위해 MongoDB가 사용하고 있음

. 인덱스의 모든 엔트리가 고유해야 함이미 존재하는 키 값을 삽입하려고 하면 예외 발생, 삽입이 실패함

. 컬렉션에 데이터가 존재하지 않을 때 생성하는 것이 좋음: 처음부터 고유 제약 조건을 보장함

. 기존에 있는 컬렉션에 생성해야 할 때는1) 반복적으로 수행하면서 중복되는 키 삭제 2) `dropDups: true` 옵션으로 중복 키 자동 임의 삭제3) 새 컬렉션을 만들고, 고유 인덱스를 만든 다음, 도큐먼트를 복사하면서 중복 데이터 삭제.

. 해시 인덱스

- 해시 값이 도큐먼트 엔트리 순서를 결정함: 이름 순으로 서로 가까이 있지 않을 가능성이 있음
- Indexing 값은 원본 필드 값의 해시값이 됨
- 해시 인덱스의 엔트리가 균등하게 분배되기 때문
- 키 데이터가 균일하지 않게 분포할 경우, 해시 인덱스 값은 일관성을 유지한다

- _id의 가장 중요한 비트는 생성 시간을 기반으로 하는데,_id를 사용하는 인덱스로 샤드를 결정하게 되면 동 시간대에 대량의 쓰기 load가 생기면 문제 발생 가능성
- 샤딩과 연관된 개념이므로 일단 간단하게 말하면 해시된 인덱스 키 값은 인덱스 엔트리의 지역성을 변경한다는 뜻, Sharded Collection에서 유용하게 사용될 수 있다

. Sparse 인덱스

- 검색 대상 필드의 값이 전체 컬렉션에서 차지하는 밀도 가 낮은 경우 생성 시 유리
- 해당 Field가 Null인 경우 Index에 포함시키지 않음
- Full Collection 검색하는 것 보다 해당 조건을 만족하는 Document로만 검색하기 때문에 성능적으로 유리
- Sparse Index의 경우에 조건 없이 전체 Collection에 대한 검색문장만으로 인덱스 스캔 가능

. Partial 인덱스

- 데이터 검색 방법이 특정 칼럼에서 조건을 만족하는 데이터만 검색하는 것이 아니라 그 조건에서 다시 특정 값만 가진 데이터들을 추가로 조건 검색하는 경우라면 Partial 인덱스 생성을 권장
- B Tree인덱스에 비해 현저히 적은 저장 공간으로 인덱스를 생성할 수 있을 뿐만이 아니라 검색 시에도 좁은 범위의 인덱스만을 검색할 수 있기 때문에 빠른 성능이 보장

. Background 인덱스

. foreground로 만드는 것보다 속도가 느림

. 하지만 global lock을 잡지 않기 때문에 서비스에 문제가 생기지 않음

. 렘과 임시파일을 사용해서 인덱스를 만드는데 렘 용량이 모자랄 정도의 데이터가 클 경우 매우 느려지고 심각한 문제가 생길 수 있음

. 인덱스가 만들어지고 있는 진행상황은 mongod.log에 다 찍힘

. Geospatial Index

. 지도의 좌표 같은 데이터를 효율적으로 조회하기 위해 사용되는 인덱스.

. 좌표로 구성되는 2차원 구조로 하나의 Collection에 하나의 2D Index를 생성

. 저장된 위도값, 경도값에 따라 도큐먼트를 특정 위치에 '가까이' 배치하는 것

. wildCard Index

. 4.2에서 새롭게 추가된 index

. MongoDB에서 동적인 스키마 구조를 제공하기 위해 애플리케이션 단에서 어떤 필드 어떤 조건으로 검색하게 될지 어려움 → 유연한 데이터 검색을 위해 사용

. compound, unique, 2d, 2dsphere, TTL, Text Index 유형에는 생성할 수 없음

. featureCompatibilityVersion(기능호환버전)이 설정 되어야 wildcard index를 생성할 수 있음

. _id를 제외한 모든 필드에 생성 가능

. 여러 특정 필드에 wildcard index를 생성할 때는 wildcardProjection 옵션절을 사용

. 하나의 컬렉션에 여러개의 wildcard index 생성 가능

. sparse index타입으로 생성 가능

. 컬렉션의 모든 필드, 하위 문서 또는 배열을 자동으로 일치시킨 후 검색

,TTL 인덱스

. MongoDB가 특정 시간 이후 컬렉션에서 도큐먼트를 자동으로 제거하는데 사용

. 특수 단일 필드 인덱스이며, 한정된 시간 동안만 데이터베이스에 유지해야하는 이벤트데이터, 로그 및 세션 정보와 같은 특정 정보 관리에 유용

. 인덱스 prefix

. 정의 된 인덱스의 앞 쪽부터 포함하는 부분집합의 인덱스들을 '인덱스 Prefix'

. 인덱스 정렬

. 오름차순인 1과 내림차순인 -1의 정렬 순서

```
db.collection.createIndex({ a: 1, b: 1 })

// 성능이 좋습니다
db.collection.find().sort({ a: 1, b: 1 })
// 성능이 좋지 않습니다
db.collection.find().sort({ b: 1, a: 1 })
```

. 인덱스 교차(intersection)

. MongoDB 2.6부터 제공

. '단일 인덱스'와 '컴파운드 인덱스'를 하나의 컬렉션 내에서 별개로 지정하더라도 쿼리가 구동 될 때에는 내부에서 교집합처럼 동작

. 외래 키

. 기본적으로 MongoDB는 기본키-외래키 관계를 지원하지 않으나 Mongoose라는 플러그인으로 지원 가능

. 인덱스 사용법

. 모든 인덱스를 갱신해야 하기 때문에 모든 쓰기 작업은 인덱스 때문에 더 오래 걸림

. Collection 당 최대 64개까지 인덱스를 지닐수있지만, 2~3개만 지니는게 좋다.

. 몽고디비의 인덱스는 RDBMS와 유사하게 작동함

explain()

이번에는 특정 쿼리가 어떻게 수행되는지를 알아봅니다. `explain()` 메소드는 단어의 뜻 그대로 쿼리의 수행 내역을 설명해줍니다. 이 때 어떤 모드로 수행 할 것인지를 인자로 입력 할 수 있습니다.

executionStats 모드

- 인덱스 사용 여부
- 스캔한 문서들의 수
- 쿼리의 수행 시간

allPlansExecution 모드

- 쿼리 계획을 선택하는데 필요한 부분적인 실행 통계
- 쿼리 계획을 선택하게 된 이유

hint()

작성한 쿼리의 인덱스를 테스트 하니 크게 효과가 없어 보인다면 어떻게 해야 할까요? 다른 필드에 인덱스를 걸어 보거나 혹은 아예 인덱스를 제거해서 테스트 해보고 싶을지도 모릅니다. 이럴 때 사용할 수 있는 메소드가 `hint()` 입니다.

다음은 `find` 쿼리에 `zipcode` 필드를 키로 하여 인덱스를 걸고, `executionStats` 모드의 결과를 받아봅니다.

만약 인덱스가 없는 상태에서의 테스트가 필요하다면 다음과 같이 $natural 옵션을 넣어서 테스트 할 수 있습니다.