11. MongoDB 데이터 모델링

MongoDB의 모델링 특징

. Rich Document 구조

- RDMS는 정규화를 통해 데이터 중복을 제거하며 무결성을 보장하는 설계 기법을 지향, NoSQL은 데이터 중복을 허용하며 역정규화된 설계를 지향(저장장치의 비약적 발전 및 저렴한 가격)

. 중첩구조

- RDBMS는 Entity 간의 Relationship을 중심으로 데이터의 무결성을 보장하지만 불필요한 JOIN을 유발시킴으로써 코딩양을 증가, 검색 성능을 저하
- NoSQL은 중첩 데이터 구조를 설계 할 수 있기 때문에 불필요한 JOIN을 최소화

. 논스키마 구조

- MondoDB에서 사용자 계정은 단지 인증의 의미만 가지고 있기 때문에 스키마를 고려한 설계를 할 필요가 없다.

. 데이터 타입

Type	Number	Alias	Notes
Double	1	“double”	
String	2	“string”	
Object	3	“object”	
Array	4	“array”	
Binary data	5	“binData”	
Undefined	6	“undefined”	Deprecated.
ObjectId	7	“objectId”	
Boolean	8	“bool”	
Date	9	“date”	
Null	10	“null”	
Regular Expression	11	“regex”	
DBPointer	12	“dbPointer”	Deprecated.
JavaScript	13	“javascript”	
Symbol	14	“symbol”	Deprecated.
JavaScript (with scope)	15	“javascriptWithScope”	
32-bit integer	16	“int”	
Timestamp	17	“timestamp”	
64-bit integer	18	“long”	
Decimal128	19	“decimal”	New in version 3.4.
Min key	-1	“minKey”	
Max key	127	“maxKey”	

null형 { "name" : null }

undefined형 { "name" : undefined }

boolean형 { "name" : true }

64비트 부동소숫점 { "age" : 21 }

string형 { "name" : "neo" }

objectId형 { "oid" : ObjectId() }

date형 { "regdate" : new Date() }

정규식형 { "name" : /neo/i }

javascript형 { "func" : function() { /- ... *- } }

array형 { "array" : [1, 2, 3] }

문서형 { "person" : { "name" : "neo" } }

. MongoDB의 모델링 고려 사항

1. 데이터 조작은 어떻게 수행되는가?
   - 하나의 Collection은 여러 개의 필드로 구성
   - 필드마다 데이터의 크기, 참조하는 주기 등이 다양하기 때문에 이러한 필드들을 하나의 Collection으로 생성 시 불필요하게 메모리, CPU와 같은 시스템 자원을 낭비, 시스템 성능을 저하시키는 원인이 됨

2. ACCESS PATTERN은 어떤가?
   - 해당 Collecion에 대해 얼마나 많은 데이터를 자주 READ, WRITE하는지에 대한 비율과 데이터가 얼마나 오래 보관 되어야 하는지에 대한 LIFE CYCLE에 대한 이해에 따라 설계를 결정
   - 빅데이터에 대한 쓰기 작업이 빈번한 Collection을 분리 설계하게 되면 데이터를 빠르게 저장하는데 한계가 존재하며 읽기 작업이 빈번한 필드들을 여러개로 분리 설계하면 읽기 성능이 저하

3. SCHEMA 설계 시 고려사항은?
   - 기업에서 발생하는 데이터들은 업무적 성격에 따라 다양성을 가지고 있으므로 그에 맞는 데이터 구조를 설계할 수 있어야함

. MongoDB 모델링 시 6가지 원칙

1. 피할 수 없는 이유가 없다면 도큐먼트에 포함할 것.
2. 객체에 직접 접근할 필요가 있다면 도큐먼트에 포함하지 않아야 함.
3. 배열이 지나치게 커져서는 안됨. 데이터가 크다면 one-to-many로, 더 크다면 one-to-squillions(임베디드)로. 배열의 밀도가 높아진다면 문서에 포함하지 않아야 함.
4. 애플리케이션 레벨 join을 두려워 말 것. index를 잘 지정했다면 관계 데이터베이스의 join과 비교해도 큰 차이가 없음.
5. 비정규화는 읽기/쓰기 비율을 고려할 것. 읽기를 위해 join을 하는 비용이 각각의 분산된 데이터를 찾아 갱신하는 비용보다 비싸다면 비정규화를 고려해야 함.
6. MongoDB에서 어떻게 데이터를 모델링 할 것인가는 각각의 애플리케이션 데이터 접근 패턴에 달려있음. 어떻게 읽어서 보여주고, 어떻게 데이터를 갱신한 것인가.

. 임베디드, 레퍼런스 모델

Embedded 저장 방법은 2가지 종류의 Document가 있을 때, 1개의 Document 데이터를 다른 Document key의 value에 저장하는 방법입니다.

```
// Person
{
   _id: "joe",
   name: "Joe Bookreader",
   address: {
      street: "123 Fake Street",
      city: "Faketon",
      state: "MA",
      zip: "12345"
  }
}

// Address
{
   pataron_id: "joe",
   street: "123 Fake Street",
   city: "Faketon",
   state: "MA",
   zip: "12345"
}
```

Reference 저장 방법은 pointer 개념으로 이해하면 쉬울 것 같습니다. Embedded 방식과 달리 Document를 통째로 저장하는것이 아니라 참조 할 수 있도록 ID를 저장하는 것입니다.

장점

. 별도의 논리적 구조로 저장되기 때문에 도큐먼트 크기에 제한 X

. 비즈니스 룰 상 별도로 처리되는 데이터 구조에 적합

단점

. 매번 논리적 구조간 Link를 해야하기에 Embeded보다 성능이 떨어짐

. 컬렉션 개수가 증가하며, 관리 비용이 증가

. 배열

. 도큐먼트 유효성

몽고DB는 특정 컬렉션에 도큐먼트를 넣을 때 유효성 검사를 거쳐 검사를 통과했을 때만 도큐먼트가 컬렉션에 입력되도록 할 수 있다. 컬렉션 유효성 검사 추가는 아래의 예시와 같이 데이터베이스에 컬렉션을 추가 할 때 할 수 있다.

Behavior

유효성 검사는 업데이트 및 입력 중에 발생합니다. collection에 유효성 검사를 추가하면 수정 될 때까지 기존 document가 유효성 검사를 거치지 않습니다.

Existing Documents

MongoDB가 validationLevel 옵션을 사용하여 기존 document를 처리하는 방법을 제어할 수 있다.

기본적으로 validationLevel은 엄격하며 MongoDB는 모든 입력 및 업데이트에 유효성 검사 규칙을 적용한다. validationLevel을 moderate로 설정하면 유효성 검사 규칙을 삽입에 적용하고 유효성 검사 기준을 충족하는 기존 document를 업데이트한다. 중간 수준에서는 유효성 검사 기준을 충족시키지 못하는 기존 document에 대한 유효성 검사가 유효하지 않는다.

스키마 유효성 검사를 통해 validator에서 정의한대로 도큐먼트가 컬렉션에 입력되지 않으면 에러가 발생했고 해당 도큐먼트는 데이터베이스에 입력이 되지 않았다. 하지만 에러를 발생시키는 대신에 경고만 할 수도 있다. 즉, 유효성 검사 레벨을 에러에서 경고로 낮출 수 있다. 이를 위해 다음과 같이 validator의 설정을 바꿔야 한다. collMod 는 Collection

Modifier을 뜻한다. 디폴트 유효성 검사 레벨인 validationAction: 'error' 을 validationAction: 'warn'로 바꿨다.

Restrictions

admin, local 및 config 데이터베이스에는 collection에 대한 validator를 지정할 수 없다.

system. * collection에 대해 validator를 지정할 수 없다.

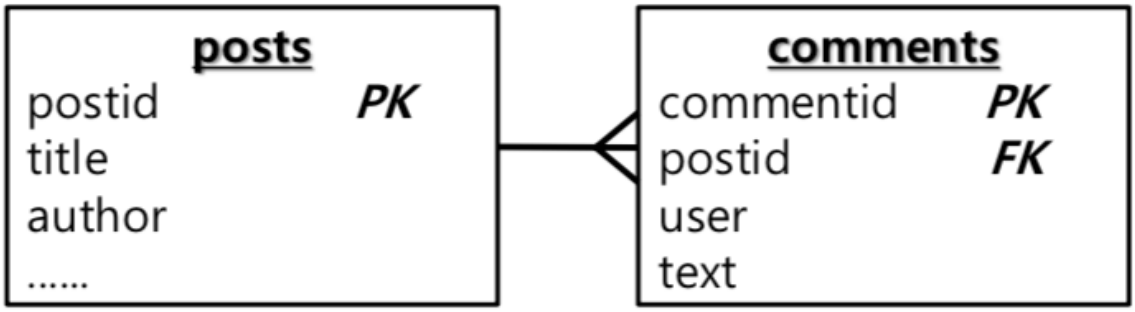
Bypass Document Validation

사용자는 bypassDocumentValidation 옵션을 사용하여 document 유효성 검사를 우회할 수 있다.

MongoDB, NoSQL의 데이터 모델링

NoSQL은 Data와 프로세스, 모두를 설계의 중심으로 둠(Big Data의 수집 및 저장이 중심)

1:N 패턴



■ Linked Document는 기존 관계형 데이터베이스 처럼 비정규화하지 않고, 두개의 테이블로 분리시키는 방법이다. 제약조건이 없다는 점을 제외하면 관계형 데이터베이스와 동일하다고 볼 수 있다.

방법 1) embedded : 자식 객체가 단독으로 사용되지 않고 부모객체 내에서만 사용될 때 사용.

- 예) 주문 정보와 주문 세부 항목 정보한번의 Read로 필요한 정보 모두를 읽어옴 → 읽기 성능 향상
- Strong Association

방법 2) linked : 자식객체가 부모객체와는 별개로 단독으로 사용될 때 적용

- 예) 상품 분류 정보와 상품 정보
- 상품분류별 상품 정보들을 조회하려면 여러번 Read를 해야 함. → 읽기 성능 저하
- 데이터 일관성이 상대적으로 중요할 때 사용
- Weak Association

N:M 패턴

▪ 관계형 데이터베이스에서는 주로 관계 테이블을 정의하여 조인하지만, MongoDB에서는 배열 키를 이용할 수 있다.

- 배열 필드의 각 값에도 인덱싱이 가능하다.

▪ 예) 한 분류 코드에 여러 상품이 포함될 수 있으며, 동시에 한 상품이 여러 분류 코드에 포함될 수 있는 경우

▪ 참조하는 측의 컬렉션에서 배열키 필드 사용하여 인덱싱