1105 NoSQL

"not only SQL"

NoSQL 등장 배경

기존의 데이터베이스를 만든다는것은 결국 스토리지를 사용해야한다는 것이다.

데이터를 어떻게 저장할 것인지 프로젝트 설계 당시에 E-R 다이어그램을 작성해 두며 프로젝트를 진행하게 되었다. 때문에 스토리지 하나하나의 값이 비쌌을 시절 사람들은 데이터의 정재작업을 통해서 예산을 줄이는 방식을 사용하였다.

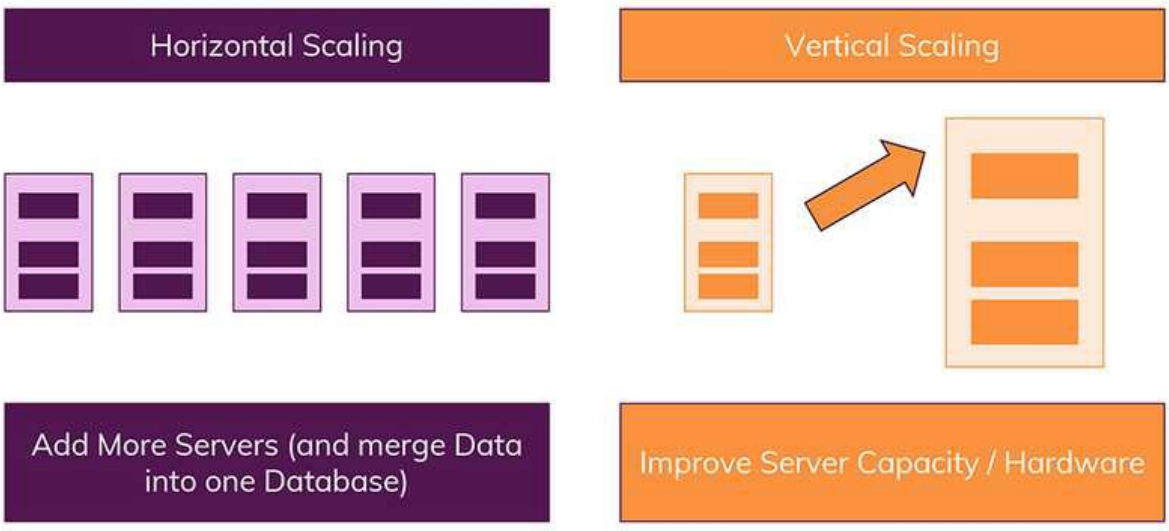
동시에 프로젝트 진행시에 데이터를 변경하는 경우가 있다면 그만큼 예산과 시간이 상당히 늘어나는 경우가 태반이였다.

DB의 확장 (수직/ 수평)

처리할 작업량이 늘어날때마다, 늘어나는 요구에 맞춰 크기를 키우거나 기능을 확장시킬 수 있는 시스템, 네트워크 혹은 과정

수직 : 서버의 부품을 업그레이드하는 등 하드웨어적인 성능을 향상시키는 것을 말함.

수평 : 데이터베이스 서버의 갯수를 늘리는 방식

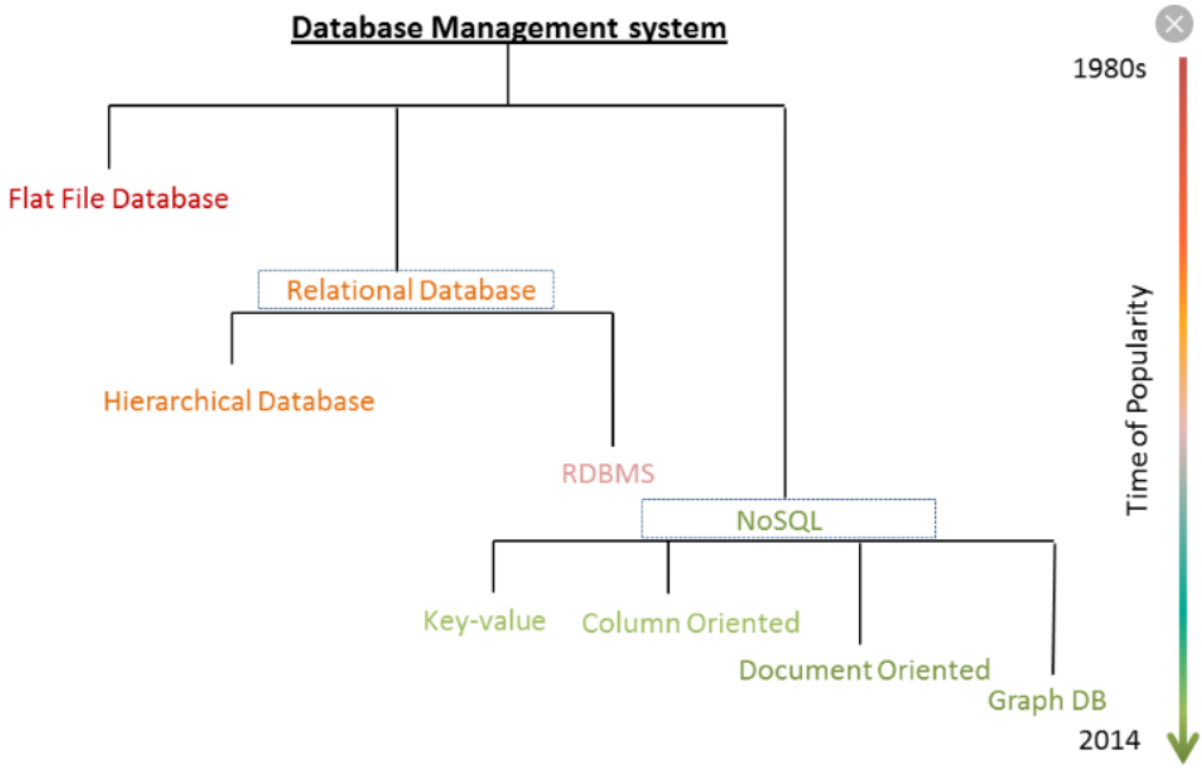


(왼)수평적 확장, (오) 수직적 확장

NoSQL이 필요한 시점

- 정확한 데이터 요구사항을 알 수 없거나 관계를 맺고 있는 데이터가 자주 변경(수정)되는 경우
- 읽기(read)처리를 자주하지만, 데이터를 자주 변경하지 않는 경우 (즉, 한번의 변경으로 수십 개의 문서를 수정 할 필요가 없는 경우) - ??
- 데이터베이스를 수평으로 확장해야 하는 경우 (즉, 막대한 양의 데이터를 다뤄야 하는 경우, 읽기/쓰기 처리량이 큰 경우)

NoSQL의 대표적인 갈래



document DB - JSON 객체와 같은 방식으로 저장됨. 각문서에서 필드와 값의 쌍으로 표현된다.

대량의 데이터를 수용할수있도록 수평 스케일아웃이 가능하게 된다.

테이블이 스키마가 유동적이다. 즉 레코드마다 각각 다른 스키마를 가질 수 있습니다. XML, JSON과 같은 Document를 이용해 레코드를 저장합니다. 트리형 구조로 레코드

를 저장하거나 검색하는데 좋은 Database입니다. (하단 보충 설명)

MongoDB, Azure Cosmos DB, CouchDB, MarkLogic, OrientDB

Key-Value Stores - 기본적인 패턴으로 Key,value가 하나의 묶음으로 저장되는 구조로 단순한 구조이기에 속도가 빠르며 분산 저장 시 용이하다. Key 안에 (Column, Value) 형태로 된 여러개의 필드를 갖습니다.. 주로 Serverconfig, Session Clustering등에 사용되고 엑세스 속도는 빠르지만, Scan에는 용이하지 않습니다.

Redis, Oracle NoSQL Database, Voldmorte, Oracle Berkeley DB, Memcached, Hazelcast

wide column - 행마다 키와 해당값을 저장할 때마다 각각 다른값의 다른 수의 스키마를 가질 수 있습니다. 사용자 이름(Key)에 해당하는 값에 스키마들이 각각 다르다는 것을 알 수 있고. 이러한 구조를 갖는 Wide Column Database는 대량의 데이터의 압축, 분산처리, 집계처리(sum, count, avg 등) 및 쿼리 동작 속도 그리고 확장성이 뛰어난 것이 그 대표적인 특징입니다.

Cassandra, HBase, Google BigTable, Vertica, Druid, Accumulo, HyperTable

Graph DB - 데이터를 노드로 표현하며, 노드 사이의 관계를 엣지로 표현, ,RDBMS 보다 Performance가 좋고 유연하며 유지보수에 용이한것이 특징. Social networks, Network diagrams 등에 사용할 수 있다고 합니다.

Neo4j, Blazegraph, OrientDB ,AgensGraph(국내솔루션)

MongoDB의 특징

- 컬럼 없음 → Schema 없음 - Document 내에 Field를 정의함 (Key : Value) - Key에 대한 값은 Document가 될 수 있음 (Embedded Document) - Key에 대한 값은 배열이 될 수 있으며, 배열의 값으로 Document를 포함할 수 있음

집합적 데이터 모델

관계형 DB에서의 여러개 테이블 데이터를 하나의 Document에 모아둘 수 있음

Secondary Index

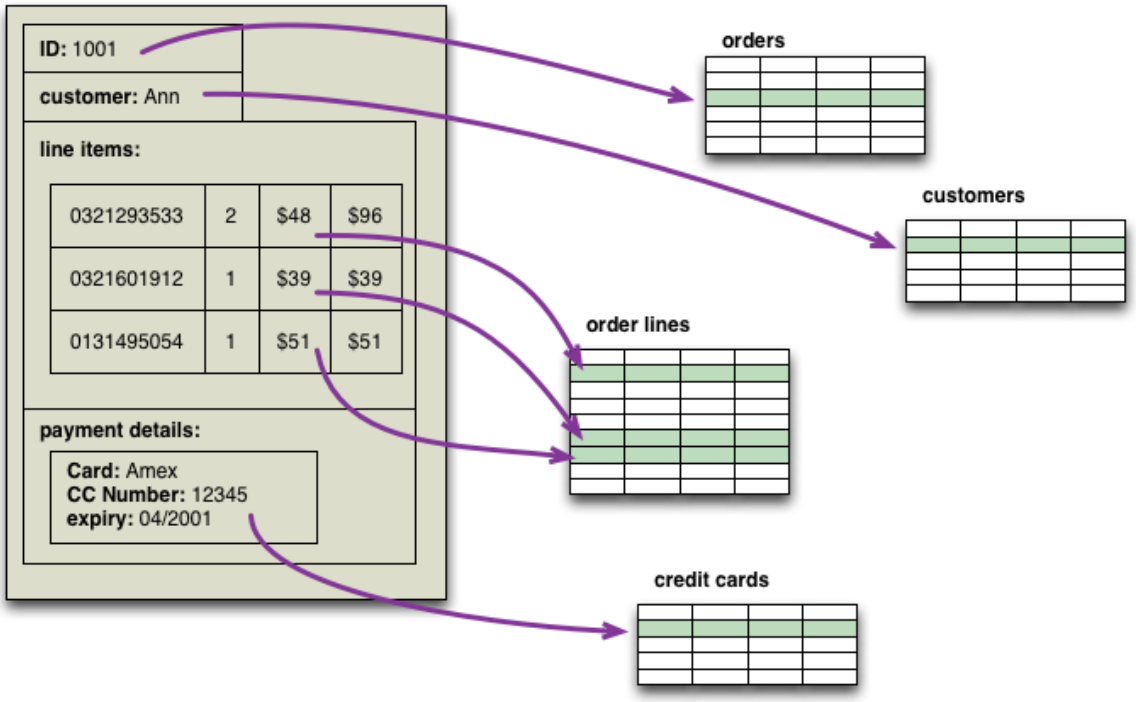
다른 NOSQL 보다 secondary index 기능이 발달되어 있음

샤드키 지정

_id : 키 필드

Shard Key <> _id

대부분의 NOSQL은 Row Key = Shard Key 임



변환 예시)

```
{"_id": 1001,
 "customer":"Ann",
"line items":
  [
    {"productId":"0321293533","count":"2","price":"$48","resell":"$96"},
    {"productId":"0321293533","count":"2","price":"$48","resell":"$96",
    {"productId":"0321293533","count":"2","price":"$48","resell":"$96"},
    {"productId":"0321293533","count":"2","price":"$48","resell":"$96"},
    {"productId":"0321293533","count":"2","price":"$48","resell":"$96"}
  ],
"payment details":
  [
    {"Card":"Amex","CC Number":"12345","expiry":"04/2001"}
  ]
}
```

관계 데이터를 저장하는 방법

NoSQL은 관련 데이터를 단일 데이터 구조 내에 중접하여 저장하게 된다.

단방향 참조

```
[Categories 컬렉션]
{ "_id" : 1, "name" : "한식", "desc" : "불고기, 식혜 등" }
{ "_id" : 2, "name" : "분식", "desc" : "라면, 김밥 등" }
{ "_id" : 3, "name" : "음료", "desc" : "식혜,콜라,사이다 등" }

[Products 컬렉션]
{
  "_id" : 1001, "productname" : "김밥", "unitprice" : 2000,
  "categoryid" : [ 1, 2 ]
}
{
  "_id" : 1002, "productname" : "식혜", "unitprice" : 3000,
  "categoryid" : [ 1, 3 ]
}
```

양방향 참조

```
[Categories 컬렉션]
{ "_id" : 1, "name" : "한식", "desc" : "불고기, 식혜 등", productid : [ 1001, 1002 ] }
......

[ Products 컬렉션 ]
{ "_id" : 1001, "productname" : "김밥", "unitprice" : 2000, "categoryid" : [ 1, 2 ] }
{ "_id" : 1002, "productname" : "식혜", "unitprice" : 3000, "categoryid" : [ 1, 3 ] }
```

Tree-패턴

```
a
|-b
   |-c
   |-d
|- e
   |- f
```

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

Tree패턴 - 임베디드 패턴

```
{
 _id : tree,
 name : "a",
 childs : [
   {
     name : "b",
     childs ": [
       { name : "c" },
       { name : "d" }
       ]
     }, {
     name : "e",
     childs : [
       { name : "f" }
       ]
   }
 ]
}
```

Tree패턴 - Linked Document

```
{ _id: "a" }
{ _id: "b", ancestors: [ "a" ], parent: "a" }
{ _id: "c", ancestors: [ "a", "b" ], parent: "b" }
{ _id: "d", ancestors: [ "a", "b" ], parent: "b" }
{ _id: "e", ancestors: [ "a" ], parent: "a" }
{ _id: "f", ancestors: [ "a", "e" ], parent: "e" }
```

출처

https://devuna.tistory.com/25

https://www.udemy.com/course/nodejs-the-complete-guide

https://www.mongodb.com/ko-kr/nosql-explained

https://ko.wikipedia.org/wiki/와이드_컬럼_스토어

https://blog.voidmainvoid.net/241