1. 관계형DB와 NOSQL DB

❖ RDB vs NOSQL

구분	RDBMS	NOSQL
장단점	•데이터 무결성 보장(CA) •정규화된(정형) 데이터 처리 •확장성 문제. 분산환경에 적합(X)	•데이터 무결성, 정합성을 보장하지 않을 수 있음. •비정형, 반정형 데이터 처리
특징	•JOIN •ACID	•강한 Consistency(X) •Schema가 없거나 변경이 유연함.
Use Cases	•중요한 트랜잭션 처리(ex:금융)가 요구되는 경우	•대량의 데이터 처리가 필요한 경우 •빠른 성능을 요구하는 경우

- RDB는 데이터의 품질에 집중
  - 무결성, 정합성 중심 --> 정규화
  - 수직적 확장에 적합함. (CPU, Memory, Disk 용량을 증가시킴)
- NOSQL은 대량의 데이터 처리에 집중
  - 무결성, 정합성 보다는 빠르게 대량의 데이터를 처리하는 것에 집중 -> 비정규화
  - 대량 데이터의 쓰기, 대용량 데이터에 대한 쿼리
  - 수평적 확장을 활용함. (용량이 부족하다면 서버 인스턴스를 추가해 확장함. 샤딩(Sharding))

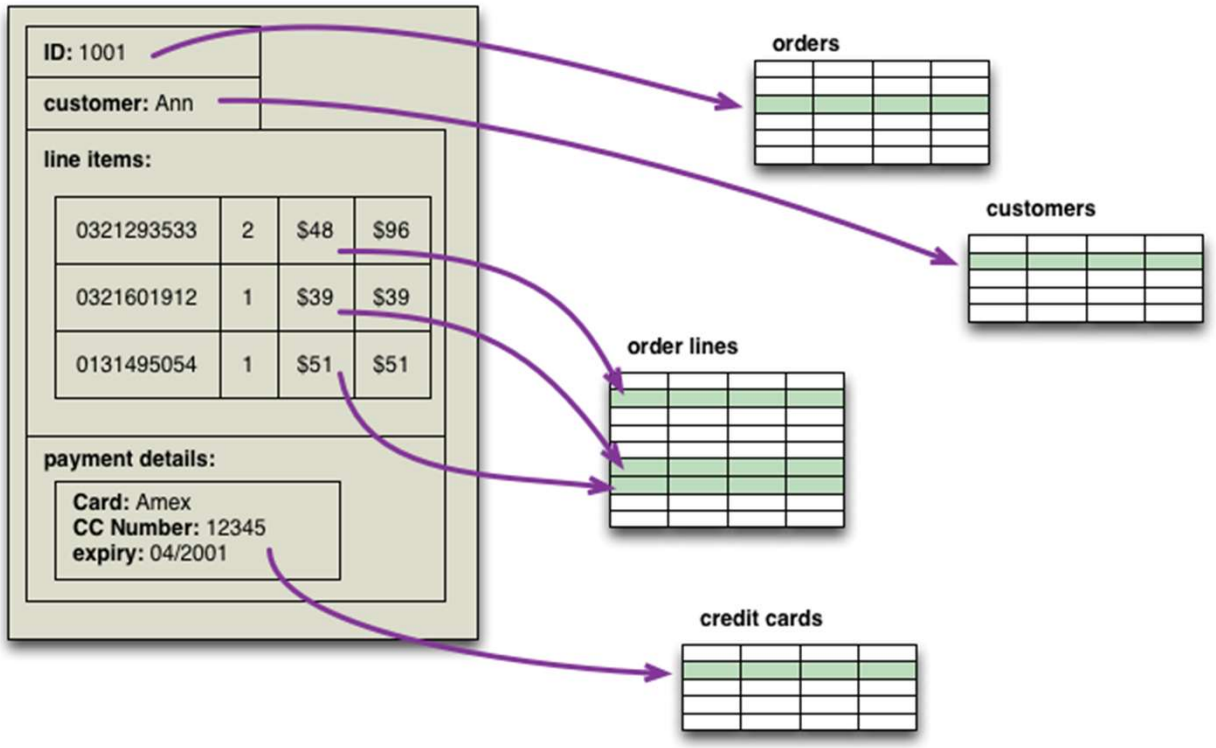
2. MongoDB 소개

- Humongous DB
- Schema Free
- 수평적 데이터 확장 기능 뛰어남
  - Auto Sharding
- 데이터 저장 구조
  - BSON : Binary Json
  - Document 예

```
{
  "_id" : "mspark11",
  "name" : "박명수",
  "phones" : [ "010-2452-8864", "02-2214-3521" ],
  "title" : "수석 컨설턴트",
  "team" : { "name" : "기술컨설팅팀", "code" : "Z03212" },
  "schedules" : [
      { "time" : "20150311130000", "loc" : "과천", "work" : "업무협의" },
      { "time" : "20150402150000", "loc" : "강남역", "work" : "제휴상담" },
      { "time" : "20150211100000", "loc" : "종로", "work" : "전략회의", "done": true },
      { "time" : "20150211170000", "loc" : "삼성동", "work" : "세미나 참석", "done" :true }
  ]
}
```

2. MongoDB 소개

❖ Document 지향 DB



http://martinfowler.com/bliki/AggregateOrientedDatabase.html

2. MongoDB

❖ 일관성과 트랜잭션

- 일관성(Consistency)
  - Replica Set을 기본 고려한 데이터베이스
    - 애플리케이션에서 일관성 수준을 결정할 수 있음.(WriteConcern)
    - 쓰기 일관성과 성능은 Trade Off 관계
  - 읽기 일관성
    - 'Secondary 읽기 우선' 설정은 성능 ▲, 읽기 일관성 ▼
- 트랜잭션
  - 대부분 원자적 트랜잭션만을 지원
  - 엄격한 트랜잭션 처리가 필요하다면...
    - 애플리케이션 측에서 처리를 하거나
    - 관계형 데이터베이스를 고려한다.
  - ** 4.0 버전부터 Replica set을 사용할 경우 트랜잭션을 지원함.

2. MongoDB

❖ Auto Sharding
- 여러 장비에 데이터를 나누어 분산확장
  - Scale Out
- MongoDB는 처음부터 분산확장을 염두에 두고 설계되었음
- 용량이 더 필요하다면, 새 장비를 클러스터에 추가하고 약간의 설정을 하는 것만으로 완료

❖ 복제
- Replica Set
  - 자동 장애조치 제공
- Primary Election

❖ 쿼리
- SQL 지원(X)
- B+-Tree 기반 인덱스 잘 지원됨
- Full Text Search 기능 지원
- 집계 기능 : aggregate

3. MongoDB 도구

❖ mongod
- 핵심 데이터베이스 서버
- 데이터파일 저장 위치
  - /data/db
  - --dbpath 옵션으로 변경 가능
- 여러가지 모드로 실행 가능
  - stand alone
  - replica set
  - shard cluster
- 설정은 최소화
  - 메모리관리 등은 운영체제에게 맡김

❖ mongosh : mongo shell
- javascript 기반의 Shell 지원 클라이언트 도구
- 가장 기본적인 클라이언트
- 개발자라면 mongo shell

❖ mongo compass
- GUI 형태의 도구

4. MongoDB 설치 및 실행

❖ Community Edition

- 무료로 사용가능한 DB 엔진
- 다운로드
  - https://www.mongodb.com/try/download/community
  - 윈도우 버전 : mongodb-windows-x86_64-6.0.x-signed.msi
  - 맥 버전(ARM 기준) : mongodb-macos-arm64-6.0.x.tgz
- 설치
  - 윈도우 : msi 파일 실행
  - 맥 : 터미널에서 tar ball 파일 압축 해제
- 실행 : 윈도우 기준
  - dbpath 디렉터리 생성하고.... dbpath가 c:₩data₩db, 실행할 Port가 27017이라면?
    --> .₩mongod.exe --dbpath c:₩data₩db --bind_ip_all --port 27017
  - 서버 기동 후 mongosh.exe localhost:27017 로 접속하여 기능 테스트
  - 이렇게 개인 피씨에서 실행하는 것은 개발, 테스트할 때만!!!
    - 보안 설정 등 할 것이 많음
- MongoDB Atlas
  - DB as a Service(DaaS)
  - 500MB 용량 ReplicaSet 무료!!

4. MongoDB 설치 및 실행

❖ mongod 서버 실행 화면(윈도우 파워셸)

```
PS C:\mongodb60\bin> .\mongod.exe --dbpath ..\db --bind_ip_all --port 27017
{"t":{"$date":"2023-04-15T18:18:57.151+09:00"},"s":"I",  "c":"CONTROL",  "id":23285,   "ctx":"-","msg":"Automatically di
sabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2023-04-15T18:18:57.153+09:00"},"s":"I",  "c":"NETWORK",  "id":4915701, "ctx":"-","msg":"Initialized wire
 specification","attr":{"spec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":17},"incomingInternalClien
t":{"minWireVersion":0,"maxWireVersion":17},"outgoing":{"minWireVersion":6,"maxWireVersion":17},"isInternalClient":true}
}}
{"t":{"$date":"2023-04-15T18:18:58.703+09:00"},"s":"I",  "c":"NETWORK",  "id":4648602, "ctx":"thread1","msg":"Implicit T
CP FastOpen in use."}
{"t":{"$date":"2023-04-15T18:18:58.706+09:00"},"s":"I",  "c":"REPL",     "id":5123008, "ctx":"thread1","msg":"Successful
ly registered PrimaryOnlyService","attr":{"service":"TenantMigrationDonorService","namespace":"config.tenantMigrationDon
ors"}}
{"t":{"$date":"2023-04-15T18:18:58.706+09:00"},"s":"I",  "c":"REPL",     "id":5123008, "ctx":"thread1","msg":"Successful
ly registered PrimaryOnlyService","attr":{"service":"TenantMigrationRecipientService","namespace":"config.tenantMigratio
nRecipients"}}
```

4. MongoDB 설치 및 실행

❖ mongosh 실행 : .₩mongosh.exe localhost:27017

- .₩mongosh.exe mongodb://localhost:27017/?directConnection=true&serverSelectionTimeout=2000

```
PS C:\Users\stepa> C:\mongodb60\shell\mongosh.exe localhost:27017
Current Mongosh Log ID: 643a7721a0962b60bb42eae3
Connecting to:          mongodb://localhost:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+1
.8.0
Using MongoDB:          6.0.5
Using Mongosh:          1.8.0

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2023-04-15T18:18:59.230+09:00: Access control is not enabled for the database. Read and write access to data and conf
iguration is unrestricted
------


------
   Enable MongoDB's free cloud-based monitoring service, which will then receive and display
   metrics about your deployment (disk utilization, CPU, operation statistics, etc).

   The monitoring data will be available on a MongoDB website with a unique URL accessible to you
   and anyone you share the URL with. MongoDB may use this information to make product
   improvements and to suggest MongoDB products and deployment options to you.

   To enable free monitoring, run the following command: db.enableFreeMonitoring()
   To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
------

test>
```

5. mongoshell 기본 명령어

❖ Mongo Shell 사용법

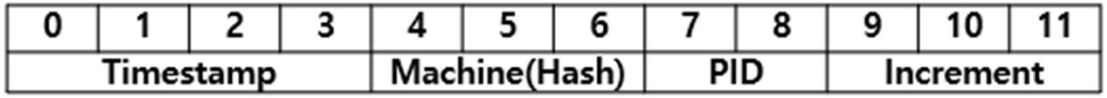
- show dbs : 전체 데이터베이스 목록 조회
- show collections : 현재 DB의 컬렉션 목록 조회
- use <dbname> : 해당 DB로 이동, 생성하는 경우라면 생성할 DB이름으로 이동
- db.<collection>.find(cond, projection) : 여러건 조회 기능
- db.<collection>.findOne(cond, projection) : 한건 조회
- db.<collection>.insert(doc) : document 추가
- db.<collection>.update(cond, doc, option) : 업데이트
- db.createCollection(colName, option) : 컬렉션을 명시적으로 생성

❖ _id필드와 ObjectID

- _id 필드 : 고유키 필드, 중복을 허용하지 않음
- 명시적으로 입력하지 않으면 ObjectID 객체 값이 자동 부여
- 명시적으로 입력하면 입력한 값으로 부여됨. 유일성이 확보되지 않은 경우 입력 불가

5. mongoshell 기본 명령어

❖ ObjectID



❖ 명령 예시

```
db.peoples.insert({ _id : "gdhong", name : "hong gil dong", gender : "m", tel : "02-3429-5789" })
db.peoples.insert({ _id : "mrlee", name : "lee mong ryong", gender : "m", tel : "02-3429-5790" })
db.peoples.insert({ _id : "sana", name : "sana", gender : "f", tel : "02-3429-5791" })

db.peoples.find({ _id: "sana" })
db.peoples.find({ gender: "m" })

db.peoples.update({ _id:"sana" }, { $set : { tel:"010-1111-2222"} })
db.peoples.find({ _id: "sana" })
```

5. mongoshell 기본 명령어

❖ 내장 문서 쿼리

```
db.users.insert({
  _id : "A001", name : "홍길동",
  phones : { home : "02-2211-5678", office : "02-3429-1234" },
  email : [ "gdhong@hotmail.com", "gdhong@gmail.com" ]
})

//집전화 변경
db.users.update({ _id:"A001" }, { $set : { "phones.home" : "02-7788-9900" } })
db.users.findOne({ _id:"A001" })

//배열에 email 추가
db.users.update({ _id:"A001" }, { $addToSet : { email : "gdhong7788@daum.net" } })
db.users.findOne({ _id:"A001" })
```

❖ 삭제

- db.<collection>. remove(cond)
- db.<collection>.drop()
- db.dropDatabase()

6. 인덱스 설정

❖ 설정 방법 : createIndex() 함수

```
//name 순 오름 차순 정렬된 인덱스 생성
db.scores.createIndex({ name : 1 });
//복합 인덱스
db.scores. createIndex({ name : 1, scores : 1});
// 내장 문서 인덱스
db.persons. createIndex({ "name.last" : 1 });
//배열 필드 인덱스(Multikeys 인덱스)
db.food. createIndex({ fruit:1 })
```

7. MongoDB Node Driver & Mongoose

❖ MongoDB Node Driver

- Mongo Shell 에서의 쿼리와 유사함
- 비동기 처리 : Promise 또는 async/await

```
const findResult = await orders.find({
    name: "Lemony Snicket",
    date: {
      $gte: new Date(new Date().setHours(00, 00, 00)),
      $lt: new Date(new Date().setHours(23, 59, 59)),
    },
});
```

- Schemeless

❖ Mongoose

- ODM : Object Document Mapper
- Application Side Schema

7. MongoDB Node Driver & Mongoose

❖ Mongoose를 이용한 연결

```
let uri = "mongodb://127.0.0.1:27017/todolistdb";
mongoose.connect(uri, { useNewUrlParser: true, useUnifiedTopology:true  })
```

❖ Schema 정의

```
const todoSchema = new mongoose.Schema({
  _id : { type:String, default: ()=> new ObjectId().toHexString() },
  owner : String,
  todo : String,
  desc : String,
  completed: { type:Boolean, default: false }
})
```

❖ Model 정의

```
const  Todo = mongoose.model("todos", todoSchema);
```

7. MongoDB Node Driver & Mongoose

❖ 조건에 부합하는 Document 카운트 구하기
- let count = await Todo.countDocuments({ owner : owner });

❖ 조건에 부합하는 Document 조회
- let todolist = await Todo.find({ owner }).sort({ _id: -1 });

❖ 한건의 Document만 조회하기
- let todoOne = await Todo.findOne({ owner, _id:id });

❖ Document 추가하기
- Model 객체의 save() 사용
  - let todoitem = new Todo({ owner, todo, desc });
  - await todoitem.save();
- Model의 insertMany() 사용
  - Todo.insertMany([d1, d2, d3])

❖ 업데이트와 삭제
- let result = await Todo.updateOne({ _id:id, owner }, { todo, desc, completed })
  - result.matchedCount, result.modifiedCount, result.upsertedCount 값 확인

8. todolist-svc 변경(1)

- ❖ mongoose, mongodb를 사용하도록...
- ❖ 의존 패키지 변경
  - npm uninstall bson-objectid
  - npm install mongoose
- ❖ ./dao/todolistDB.js 작성

```
const mongoose = require('mongoose');
const mongodb = require('mongodb');
const ObjectId = mongodb.ObjectId;


let uri = process.env.MONGODB_URI
        ? process.env.MONGODB_URI
        : "mongodb://127.0.0.1:27017/todolistdb";

mongoose.connect(uri, {
  useNewUrlParser: true,
  useUnifiedTopology:true
})
```

```
const todoSchema = new mongoose.Schema({
  _id : { type:String, default: ()=> new ObjectId().toHexString() },
  owner : String,
  todo : String,
  desc : String,
  completed: { type:Boolean, default: false }
})

const  Todo = mongoose.model("todos", todoSchema);

module.exports = { Todo };
```

8. todolist-svc 변경(2)

❖ ./dao/todolistDao.js 변경

```
const {  Todo } = require('./todolistDB');

exports.createNewOwner = async ({ owner }) => {
    try {
        let count = await Todo.countDocuments({ owner : owner });
        if (count === 0) {
            let todo1 = new Todo({ owner, todo: "ES6 공부", desc: "ES6공부를 해야 합니다" })
            let todo2 = new Todo({ owner, todo: "Vue 학습", desc: "React 학습을 해야 합니다" })
            let todo3 = new Todo({ owner, todo: "야구장", desc: "프로야구 경기도 봐야합니다." })
            await todo1.save();
            await todo2.save();
            await todo3.save();
            return { status: "success", message: "샘플 데이터 생성 성공!" };
        } else {
            return { status: "fail", message: "생성 실패 : 이미 존재하는 owner입니다." };
        }
    } catch (ex) {
        return { status: "fail", message: "생성 실패 : " + ex };
    }
};
```

8. todolist-svc 변경(3)

```
exports.getTodoList = async ({ owner }) => {
   try {
      let todolist = await Todo.find({ owner }).sort({ _id: -1 });
      todolist = todolist.map((t)=> {
         let { _id, users_id, todo, desc, completed} = t;
         return { id:_id, users_id, todo, desc, completed };
      })
      return { status:"success", todolist };
   } catch (ex) {
     return { status: "fail", message: "조회 실패 : " + ex };
   }
};

exports.getTodoItem = async ({ owner, id }) => {
   try {
      let todoOne = await Todo.findOne({ owner, _id:id });
      if (todoOne) {
         let { _id, owner, todo, desc, completed } = todoOne;
         return { status : "success", todoitem: { id:_id, owner, todo, desc, completed } };
      } else {
         return { status : "fail", message: "할일(Todo)이 존재하지 않습니다." };
      }
   } catch (ex) {
      return { status: "fail", message: "조회 실패 : " + ex };
   }
};
```

8. todolist-svc 변경(4)

```
exports.addTodo = async ({ owner, todo, desc }) => {
   try {
     if (todo === null || todo.trim() === "") {
       throw new Error("할일을 입력하셔야 합니다.");
     }
     let todoitem = new Todo({ owner, todo, desc });
     await todoitem.save();
     return { status: "success", message: "추가 성공", item: { id: todoitem._id, owner, todo, desc } };
   } catch (ex) {
     return { status: "fail", message: "추가 실패 : " + ex };
   }
};

exports.updateTodo = async ({ owner, id, todo, desc, completed }) => {
   try {
       let result = await Todo.updateOne({ _id:id, owner }, { todo, desc, completed })
       if (result.matchedCount === 1) {
           return { status:"success", message:"할일 업데이트 성공", todoitem : { id, todo, desc, completed } };
       } else {
           return { status:"fail", message:"할일 업데이트 실패", result };
       }
   } catch (ex) {
     return { status: "fail", message: "할일 변경 실패 : " + ex };
   }
};
```

8. todolist-svc 변경(5)

```
exports.deleteTodo = async ({ owner, id }) => {
   try {
      await Todo.deleteOne({ owner, _id:id });
      return { status: "success", message: "삭제 성공", item: { id } };
   } catch (ex) {
     return { status: "fail", message: "삭제 실패 : " + ex };
   }
};

exports.toggleCompleted = async ({ owner, id }) => {
   try {
      let todoOne = await Todo.findOne({ owner, _id:id });
      let completed = !todoOne.completed;
      let result = await Todo.updateOne({ _id:id, owner }, { completed })
      if (result.matchedCount === 1) {
         return { status:"success", message:"할일 완료 처리 성공", todoitem: { id, completed }};
      } else {
         return { status:"fail", message:"할일 완료 처리 실패", result };
      }
   } catch (ex) {
      return { status: "fail", message: "완료 변경 실패 : " + ex };
   }
};
```

8. todolist-svc 변경(6)

❖ ./router.js 변경

- 비동기 MongoDB 액세스 하는 부분을 찾아 모두 async/await 적용

```
router.get("/todolist/:owner/create", async (req, res) => {
    console.log("### GET /todolist/:owner/create");
    const { owner } = req.params;
    const result = await todolistDao.createNewOwner({ owner });
    res.json(result);
});
```

❖ Postman 도구로 테스트!!