[1] MongoDB 기본 및 Spring Boot 연동

- MongoDB CRUD 실습 (insert, find, update, delete)

- 필터링, 정렬, distinct, aggregate 를 활용한 다양한 쿼리

- 조건문, 루프문, 정규표현식을 이용한 JavaScript 문법 MongoDB 쉘 활용

- Spring Boot + MongoDB 연동 (MongoRepository, @Document 등)

- Postman 을 활용한 API 테스트 및 실습

[2] 날짜 처리 및 고급 Aggregation

- 날짜 데이터 ISODate 및 new Date() 활용법

- 날짜 범위 필터링 ($gte, $lte), 정렬, 업데이트

- 날짜 기반 Aggregation ($year, $month, $dayOfMonth)

- $dateAdd, $dateDiff, $dateFromParts, $dateToParts 등 날짜 연산자 실습

- React 와 연동될 JSON 기반 API 설계 및 MongoDB 날짜 관련 필드 처리 실습

[3] 배열 연산자 및 조인 연산자 실습

- 배열 연산자: $push, $addToSet, $map, $filter, $slice, $unwind 등

- 배열 필드의 다양한 조작과 Aggregation 연습

- $lookup, $merge, $unionWith 을 활용한 조인과 셀프 조인

- 조건부 조인, Cross Join, Inner Join, Outer Join, Equi Join 실습

- React 에서 배열 데이터를 처리하는 데 필요한 쿼리 포맷 정리

[4] GridFS 를 활용한 파일 업로드 및 다운로드

- mongofiles CLI 사용법 실습: put, get, delete, search, list

- GridFS 내부 구조 이해 (fs.files, fs.chunks)

- 파일 ID 기반 업로드 및 다운로드, 쉘 스크립트 활용 자동화

- mongoimport, mongoexport 를 활용한 JSON/CSV 백업 및 복원 실습

[5] 인덱스, 성능 최적화, explain(), 샤딩 개념

- 단일 및 복합 인덱스 생성/삭제 및 성능 확인

- db.collection.aggregate([{ $indexStats: {} }]) 활용

- explain("executionStats")로 쿼리 성능 분석

- 샤딩 및 복제 개념 소개 및 구성 원리 이해

날짜 처리 및 고급 Aggregation

ggregate() 파이프라인 내에서 다양한 날짜 연산자 및 변형 연산자를 조합하여 사용

날짜 연산자 (Aggregation Stage 에서 사용)

연산자	설명
$year	날짜에서 연도 추출
$month	날짜에서 월 추출
$dayOfMonth	날짜에서 일(1~31) 추출
$dayOfWeek	요일(1:일 ~ 7:토) 추출
$dayOfYear	1 년 중 몇 번째 날인지
$hour	시간 추출
$minute	분 추출
$second	초 추출
$dateToString	날짜 → 문자열 변환 (형식 지정 가능)
$dateSubtract	날짜에서 일정 시간 차감
$dateAdd	날짜에 일정 시간 추가
$dateDiff	두 날짜 간 차이 계산
$isoWeek	ISO 기준 주차 번호
$isoDayOfWeek	ISO 기준 요일

1)날짜 파싱 및 포맷

날짜를 문자열로 포맷

```
{
  $dateToString: { format: "%Y-%m-%d", date: "$createdAt" }
}
```

특정 날짜 필드에서 연, 월, 일 추출

```
{
  year: { $year: "$createdAt" },
  month: { $month: "$createdAt" },
  day: { $dayOfMonth: "$createdAt" }
}
```

날짜 차이 계산

```
{
  $dateDiff: {
    startDate: "$startDate",
    endDate: "$endDate",
    unit: "day"
  }
}
```

2) 고급 Aggregation 기능

주요 Stage 정리

연산자	설명
$group	그룹별 집계 ($sum, $avg, $push 등과 함께 사용)
$project	필드 선택 및 가공
$match	조건 필터링 (WHERE 역할)
$sort	정렬
$limit, $skip	페이징
$unwind	배열 분해
$lookup	조인 (다른 컬렉션과)
$facet	복수의 파이프라인을 동시에 실행
$bucket, $bucketAuto	구간별 그룹핑 (histogram 생성용)
$merge	결과를 다른 컬렉션에 저장
$replaceRoot	특정 하위 문서를 루트로 변경
$addFields	계산된 새 필드 추가
$setWindowFields	윈도우 함수 사용 가능 (MongoDB 5.0+)

예제

월별 문서 수 카운트

```
db.sales.aggregate([
  {
    $group: {
      _id: { year: { $year: "$createdAt" }, month: { $month:
"$createdAt" } },
      count: { $sum: 1 }
    }
  }
])
```

$facet 을 활용한 다중 분석

```
db.orders.aggregate([  {
    $facet: {
      recentOrders: [
        { $sort: { orderDate: -1 } },
        { $limit: 5 }
      ],
      summaryByStatus: [
        { $group: { _id: "$status", total: { $sum: 1 } } }
      ]
    }
  }
])
```

$bucket 으로 범주화

```
db.students.aggregate([
  {
    $bucket: {
      groupBy: "$score",
      boundaries: [0, 60, 70, 80, 90, 100],
      default: "기타",
      output: { count: { $sum: 1 } }
    }
  }
])
```

정리 요약

분류	목적	대표 연산자
날짜 추출	연도, 월, 일, 시, 분 등	$year, $month, $dayOfMonth
날짜 가공	날짜 포맷, 더하기, 빼기	$dateToString, $dateAdd
고급 집계	그룹핑, 조건 분기, 분해 등	$group, $project, $unwind
집계 고도화	다중 결과, 구간, 윈도우 분석	$facet, $bucket, $setWindowFields

[3] 배열연산자

연산자	설명
$push	그룹 내 배열에 요소 추가 (중복 허용)
$addToSet	배열에 중복 없이 요소 추가
$size	배열의 크기 반환
$filter	배열의 조건 기반 필터링
$concatArrays	여러 배열을 병합
$slice	배열에서 특정 범위 추출
$unwind	배열의 각 요소를 문서로 분해
$arrayElemAt	배열에서 특정 위치의 요소 가져오기
$map	배열의 각 요소를 변환
$reduce	배열을 축약해 하나의 값으로 반환
$reverseArray	배열을 역순으로 정렬

1) 배열 생성 및 수집

연산자	설명
$push	값을 배열에 추가 (중복 허용)
$addToSet	중복 없이 배열에 추가
$concatArrays	여러 배열을 결합
$slice	배열에서 일부 요소만 잘라냄 ($slice: [<배열>, <개수>])

$push: { $push: "$name" }

$addToSet: { $addToSet: "$name" }

2) 배열의 정보 활용

연산자	설명
$size	배열의 길이 반환
$arrayElemAt	지정한 인덱스의 요소 추출
$reverseArray	배열의 순서를 뒤집음

$size: "$names"

$arrayElemAt: ["$names", 2]

3) 배열 필터링과 조건 처리

연산자	설명
$filter	배열 요소 중 조건에 맞는 값만 필터링 ($project, $addFields 에서 사용)
$map	배열의 각 요소를 변환 ($project 내에서 활용)
$reduce	배열의 모든 값을 누적 계산하여 하나의 값으로 축약

```
$filter: {
  input: "$names",
  as: "name",
  cond: { $regexMatch: { input: "$$name", regex: "^김" } }
}
$map: {
  input: "$names",
  as: "name",
  in: { $toUpper: "$$name" }
}
```

4). 배열 분리

연산자	설명
$unwind	배열 필드를 개별 요소로 분리 → 각 요소가 별도 문서처럼 취급

{ $unwind: "$names" }

5). 날짜 필드에서 배열로 조합

연산자	설명
$year, $month, $dayOfMonth	날짜 필드에서 연/월/일 추출 가능

```
{
  year: { $year: "$createdAt" },
  month: { $month: "$createdAt" },
  day: { $dayOfMonth: "$createdAt" }
}
```

6) 인덱스 설정

- **인덱스 설정**이 필요한 경우 배열 안의 요소에 대한 **다중키 인덱스**를 설정할 수 있음

- 배열 내 문서(array of objects)의 필터링은 $elemMatch 사용

```
db.products.find({
  specs: { $elemMatch: { key: "weight", value: "10kg" } }
})
```

조인

MongoDB 에서 구현 가능한 조인 종류

조인 종류	MongoDB 구현 방식	설명
LEFT OUTER JOIN	$lookup 기본형	매칭되는 값이 없어도 NULL 포함하여 출력됨
INNER JOIN	$lookup + $unwind + $match	조인된 결과가 있는 문서만 출력
FULL OUTER JOIN	$unionWith + 수작업 조인	MongoDB 에서는 직접 구현 필요
SELF JOIN	$lookup (자기 컬렉션 참조)	자기 자신을 참조하는 경우

$lookup: 외부 컬렉션과 조인

1. 기본 형식

```
db.collection.aggregate([
  {
    $lookup: {
      from: "foreignCollection",            // 조인할 외부 컬렉션 이름
      localField: "localFieldName",         // 현재 컬렉션의 필드
      foreignField: "foreignFieldName",     // 외부 컬렉션의 조인 필드
      as: "joinedData"                      // 조인 결과가 담길 배열 필드
    }
  }
])
```

pipeline 방식 + 변수 사용

```
db.collection.aggregate([
  {
    $lookup: {
      from: "foreignCollection",
      let: { localVar: "$localField" },   // 사용할 변수 선언
      pipeline: [
        {
          $match: {
            $expr: { $eq: ["$foreignField", "$$localVar"] }  // 변수로 조인
          }
        }
      ],
      as: "joinedData"
    }
  }
])
```

2. $sum: 합계 구하기

```
{ $group: { _id: "$groupField", total: { $sum: "$amount" } } }
```

---

3. $sortByCount: 값의 개수 세고 정렬

```
db.collection.aggregate([
  { $sortByCount: "$fieldName" }  // 빈도 높은 값부터 내림차순 정렬
])
```

---

4. $out: 결과를 새로운 컬렉션에 저장

```
db.collection.aggregate([
  { $match: { field: "value" } },
  { $out: "newCollection" }  // 기존 컬렉션이 있다면 덮어씀
])
```

---

5. $merge: 기존 컬렉션에 병합 (업서트 개념)

기본 구조

```
db.collection.aggregate([
  {
    $merge: {
      into: "targetCollection",
      on: "_id",                          // 병합 기준 키
      whenMatched: "merge",               // 병합 방식: merge | replace | fail
      whenNotMatched: "insert"            // 삽입 여부: insert | discard
    }
  }
])
```

옵션 요약

옵션	설명
whenMatched	"merge": 병합 "replace": 기존 문서 대체 "fail": 중복 오류
whenNotMatched	"insert": 새로 삽입 "discard": 무시

6. $unionWith: 컬렉션 병합

기본 구조

```
db.collection.aggregate([
  {
    $unionWith: {
      coll: "otherCollection",
      pipeline: [                          // 결합할 컬렉션의 추가 필터링
        { $match: { field: "value" } }
      ]
    }
  }
])
```

$lookup vs $unionWith 비교

항목	$lookup	$unionWith
대상	다른 컬렉션의 **관련 데이터**	다른 컬렉션의 **전체/유사 데이터**
사용 목적	관계형 조인 (외래키 연결)	컬렉션 합치기 (행 결합)
조인 구조	각 문서에 **배열 필드로 삽입**	결과 문서를 **행 단위로 추가**
내부 필터	let, pipeline, $expr 가능	pipeline 으로 조건 처리

[조인 종류에 대한 코드]

1. LEFT OUTER JOIN (기본)

```
db.students.aggregate([
  {
    $lookup: {
      from: "scores",              // 조인할 컬렉션
      localField: "name",          // 현재 컬렉션 기준 필드
      foreignField: "name",        // 외부 컬렉션 기준 필드
      as: "scoreInfo"              // 결과 저장 필드 (배열)
    }
  }
])
```

특징: 매칭되는 scores 가 없어도 scoreInfo 는 빈 배열로 반환됨

---

2. INNER JOIN

```
db.students.aggregate([
  {
    $lookup: {
      from: "scores",
      localField: "name",
      foreignField: "name",
      as: "scoreInfo"
    }
  },
  { $unwind: "$scoreInfo" },    // null 제거 (배열 해제)
  { $match: { "scoreInfo": { $ne: null } } }  // 없어도 되지만 명시 가능
])
```

특징: 매칭된 문서만 출력됨

---

3. FULL OUTER JOIN (MongoDB 에서는 직접 구현 필요)

```
const studentsJoin = db.students.aggregate([
  {
    $lookup: {
      from: "scores",
      localField: "name",
      foreignField: "name",
      as: "scoreInfo"
    }
  }
]);

// scores 기준 (반대)
const scoresJoin = db.scores.aggregate([
  {
    $lookup: {
      from: "students",
      localField: "name",
      foreignField: "name",
      as: "studentInfo"
    }
  }
]);

// 위 둘을 $unionWith 또는 JS 처리로 결합해야 함
```

4. SELF JOIN (자기 자신과 조인)

```
db.emp.aggregate([
  {
    $lookup: {
      from: "emp",
      localField: "mgr",
      foreignField: "empno",
      as: "managerInfo"
    }
  }
])
```

예시: EMP 테이블에서 mgr 과 empno 를 비교해서 매니저 정보 추가

SQL 조인 유형	MongoDB 방식	비고
INNER JOIN	$lookup + $unwind + $match	null 제거
LEFT OUTER JOIN	$lookup 기본	null 허용
RIGHT OUTER JOIN	직접 구현 필요 ($unionWith + 커스텀)	없음
FULL OUTER JOIN	$unionWith + 두 방향 $lookup 조합	수작업
SELF JOIN	$lookup from 동일 컬렉션 사용	가능