

## #5 세 번째 요구사항 추가하기 - 책 통계

1. SQL의 다양한 기능들 (sum, avg, count, group by, order by)을 이해한다.
2. 간결한 함수형 프로그래밍 기법을 사용해보고 익숙해진다.
3. 동일한 기능을 애플리케이션과 DB로 구현해보고, 차이점을 이해한다.

# 32강. 책 통계 보여주기 - 프로덕션 코드 개발

# 요구사항 3 확인

## 책 통계 화면

- 현재 대여 중인 책이 몇 권이 보여준다.
- 분야별로 도서관에 등록되어 있는 책이 각각 몇 권인지 보여준다.

등록하기   목록   히스토리   통계

등록된 책 분류	총 권 수
과학	3권
사회	5권
경제	12권
컴퓨터	4권
언어	1권

대출 중인 권 수 : 10권

# 마찬가지로 API가 나와 있다!

GET /book/loan (현재 대여 중인 책의 권수 보여주기)

요청 : 파라미터 없음

응답 (바로 숫자가 반환)

number

# 마찬가지로 API가 나와 있다!

GET /book/stat (분야별로 등록되어 있는 책의 권수 보여주기)

요청 : 파라미터 없음

응답

```
[{  
  "type": "COMPUTER",  
  "count": 10  
}, ...]
```

- count가 0이면, 반환 리스트에 존재하지 않아도 된다.

# 34강. 다양한 SQL을 알아보자!

# 살펴볼 SQL 종류

sum / avg / count  
group by / order by

# sum 쿼리

유저 테이블		
id	이름	나이
1	A	10
2	B	20
3	C	30

주어진 column의 합계를 계산한다.



# sum 쿼리

유저 테이블		
id	이름	나이
1	A	10
2	B	20
3	C	30

```
select sum(age) from user;
```

# sum 쿼리

유저 테이블		
id	이름	나이
1	A	10
2	B	20
3	C	30

```
select sum(age) from user;  
60
```

# avg 쿼리

유저 테이블		
id	이름	나이
1	A	10
2	B	20
3	C	30

주어진 column의 평균을 계산한다.

# avg 쿼리

유저 테이블		
id	이름	나이
1	A	10
2	B	20
3	C	30

select avg(age) from user;

# avg 쿼리

유저 테이블		
id	이름	나이
1	A	10
2	B	20
3	C	30

select avg(age) from user;  
 $(10 + 20 + 30) / 3 = 20$

# count 쿼리

유저 테이블		
id	이름	나이
1	A	10
2	B	20
3	C	30

개수를 센다.

# count 쿼리

유저 테이블		
id	이름	나이
1	A	10
2	B	20
3	C	30

select count(\*) from user;

# count 쿼리

유저 테이블		
id	이름	나이
1	A	10
2	B	20
3	C	30

```
select count(*) from user;  
3
```



# group by 쿼리

책 테이블		
id	이름	유형
1	책 1	COMPUTER
2	책 2	COMPUTER
3	책 3	SCIENCE
4	책 4	COMPUTER
5	책 5	COMPUTER
6	책 6	SOCIETY
7	책 7	COMPUTER
8	책 8	LANGUAGE
9	책 9	SCIENCE
10	책 10	ECONOMY
11	책 11	ECONOMY
12	책 12	ECONOMY

COMPUTER 5권,  
SCIENCE : 2권,  
SOCIETY : 1권,  
LANGUAGE : 1권,  
ECONOMY : 3권

# group by 쿼리

책 테이블		
id	이름	유형
1	책 1	COMPUTER
2	책 2	COMPUTER
3	책 3	SCIENCE
4	책 4	COMPUTER
5	책 5	COMPUTER
6	책 6	SOCIETY
7	책 7	COMPUTER
8	책 8	LANGUAGE
9	책 9	SCIENCE
10	책 10	ECONOMY
11	책 11	ECONOMY
12	책 12	ECONOMY

주어진 column을 기준으로  
데이터를 그룹핑 한다  
(groupBy와 유사)

# group by 쿼리

책 테이블		
id	이름	유형
1	책 1	COMPUTER
2	책 2	COMPUTER
3	책 3	SCIENCE
4	책 4	COMPUTER
5	책 5	COMPUTER
6	책 6	SOCIETY
7	책 7	COMPUTER
8	책 8	LANGUAGE
9	책 9	SCIENCE
10	책 10	ECONOMY
11	책 11	ECONOMY
12	책 12	ECONOMY

```
select type, count(1)
from book
group by type;
```

# group by 쿼리

책 테이블		
id	이름	유형
1	책 1	COMPUTER
2	책 2	COMPUTER
3	책 3	SCIENCE
4	책 4	COMPUTER
5	책 5	COMPUTER
6	책 6	SOCIETY
7	책 7	COMPUTER
8	책 8	LANGUAGE
9	책 9	SCIENCE
10	책 10	ECONOMY
11	책 11	ECONOMY
12	책 12	ECONOMY

COMPUTER, 5  
SCIENCE, 2  
SOCIETY, 1  
LANGUAGE, 1  
ECONOMY, 3

# order by 쿼리

책 테이블		
id	이름	유형
1	책 1	COMPUTER
2	책 2	COMPUTER
3	책 3	SCIENCE
4	책 4	COMPUTER
5	책 5	COMPUTER
6	책 6	SOCIETY
7	책 7	COMPUTER
8	책 8	LANGUAGE
9	책 9	SCIENCE
10	책 10	ECONOMY
11	책 11	ECONOMY
12	책 12	ECONOMY

주어진 column을 정렬을 한다.  
내림차순, 오름차순을 지정할 수 있다.


# order by 쿼리

책 테이블		
id	이름	유형
1	책 1	COMPUTER
2	책 2	COMPUTER
3	책 3	SCIENCE
4	책 4	COMPUTER
5	책 5	COMPUTER
6	책 6	SOCIETY
7	책 7	COMPUTER
8	책 8	LANGUAGE
9	책 9	SCIENCE
10	책 10	ECONOMY
11	책 11	ECONOMY
12	책 12	ECONOMY

```
select * from book  
order by type desc;
```

# order by 쿼리

책 테이블		
id	이름	유형
6	책 6	SOCIETY
3	책 3	SCIENCE
9	책 9	SCIENCE
8	책 8	LANGUAGE
10	책 10	ECONOMY
11	책 11	ECONOMY
12	책 12	ECONOMY
1	책 1	COMPUTER
2	책 2	COMPUTER
4	책 4	COMPUTER
5	책 5	COMPUTER
7	책 7	COMPUTER



```
select * from book  
order by type desc;
```

# order by 쿼리

책 테이블		
id	이름	유형
1	책 1	COMPUTER
2	책 2	COMPUTER
3	책 3	SCIENCE
4	책 4	COMPUTER
5	책 5	COMPUTER
6	책 6	SOCIETY
7	책 7	COMPUTER
8	책 8	LANGUAGE
9	책 9	SCIENCE
10	책 10	ECONOMY
11	책 11	ECONOMY
12	책 12	ECONOMY

```
select * from book  
order by type asc;
```



# order by 쿼리

책 테이블		
id	이름	유형
1	책 1	COMPUTER
2	책 2	COMPUTER
4	책 4	COMPUTER
5	책 5	COMPUTER
7	책 7	COMPUTER
10	책 10	ECONOMY
11	책 11	ECONOMY
12	책 12	ECONOMY
8	책 8	LANGUAGE
3	책 3	SCIENCE
9	책 9	SCIENCE
6	책 6	SOCIETY



select \* from book  
order by type asc;

# 35강. 애플리케이션 대신 DB로 기능 구현하기

# 대출 권수 - 기존과 어떤 차이가 있을까?!

```
@Transactional(readOnly = true)
fun countLoanedBook(): Int {
    return userLoanHistoryRepository.findAllByStatus(UserLoanStatus.LOANED).size
}
```

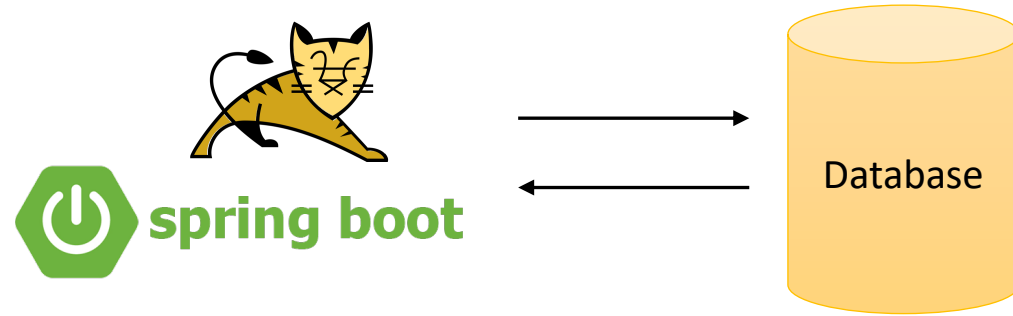
# 대출 권수 - 기존과 어떤 차이가 있을까?!

```
@Transactional(readOnly = true)
fun countLoanedBook(): Int {
    return userLoanHistoryRepository.countByStatus(UserLoanStatus.LOANED).toInt()
}
```

# 서버 코드를 보고 Query를 생각할 수 있어야 한다!



# 서버 코드를 보고 Query를 생각할 수 있어야 한다!

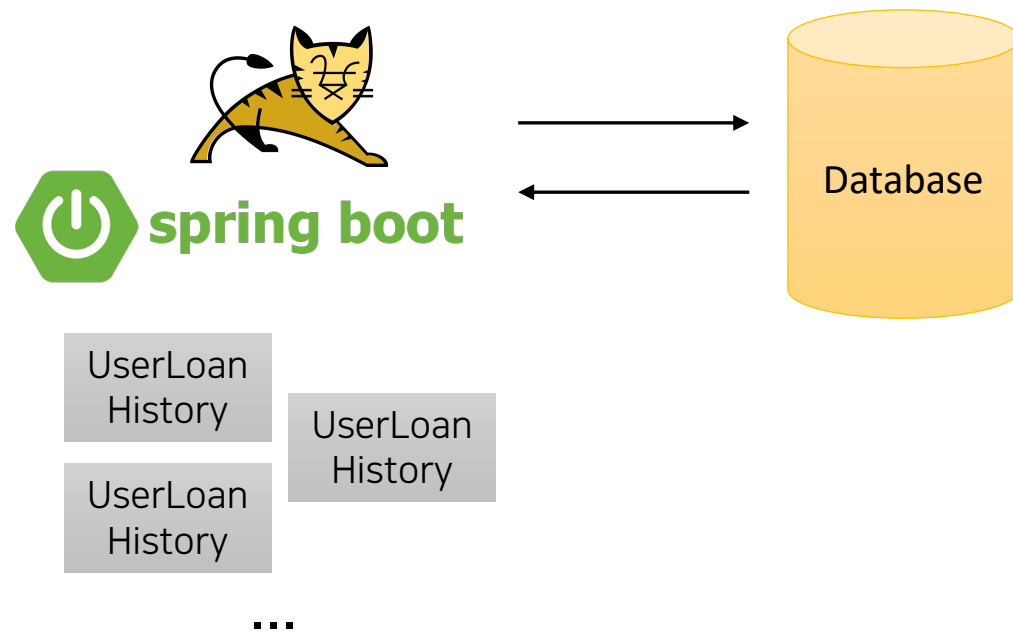


select \* from user\_loan\_history where status = ?;

```
@Transactional(readOnly = true)
fun countLoanedBook(): Int {
    return userLoanHistoryRepository.findAllByStatus(UserLoanStatus.LOANED).size
}
```

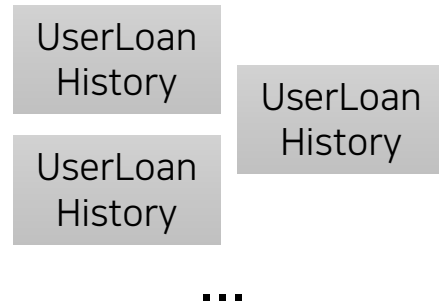
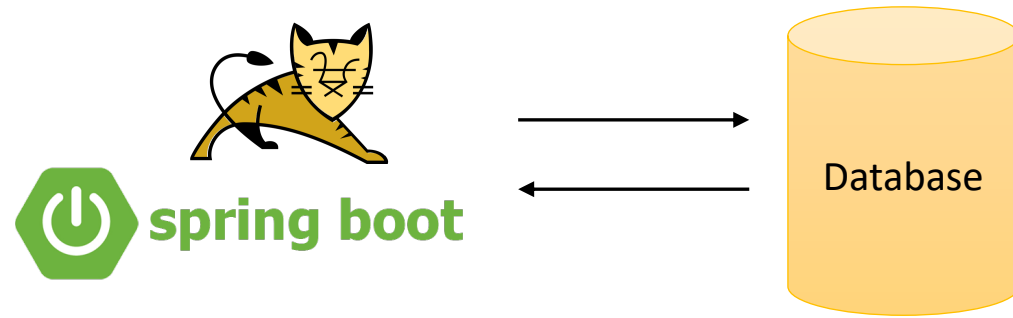
# 서버 코드를 보고 서버의 메모리를 생각할 수 있어야 한다

```
select * from user_loan_history where status = ?;
```



# 서버 코드를 보고 서버의 메모리를 생각할 수 있어야 한다

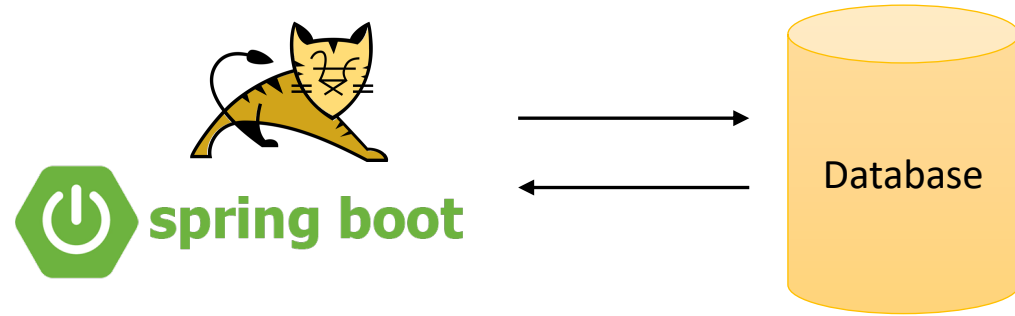
```
select * from user_loan_history where status = ?;
```



DB에 있는 모든  
UserLoanHistory가 List에 있다



# 서버 코드를 보고 서버의 동작을 생각할 수 있어야 한다



메모리에 존재하는 List의 size를 계산한다.

```
@Transactional(readOnly = true)
fun countLoanedBook(): Int {
    return userLoanHistoryRepository.findAllByStatus(UserLoanStatus.LOANED).size
}
```

# 코드 변경 전 방법은!

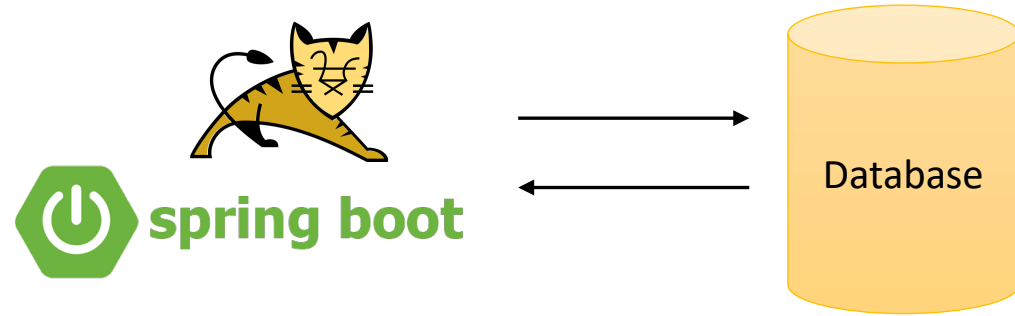
```
@Transactional(readOnly = true)
fun countLoanedBook(): Int {
    return userLoanHistoryRepository.findAllByStatus(UserLoanStatus.LOANED).size
}
```

- 1) DB에 존재하는 데이터를 모두 가져와서
- 2) 애플리케이션이 그 size를 계산한다.

# 변경된 방법도 확인해보자!

```
@Transactional(readOnly = true)
fun countLoanedBook(): Int {
    return userLoanHistoryRepository.countByStatus(UserLoanStatus.LOANED).toInt()
}
```

# 서버 코드를 보고 Query를 생각할 수 있어야 한다!

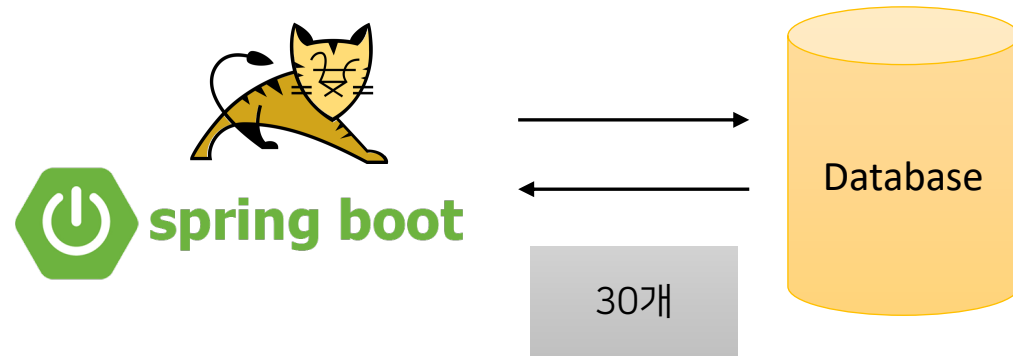


select count(\*) from user\_loan\_history where status = ?;

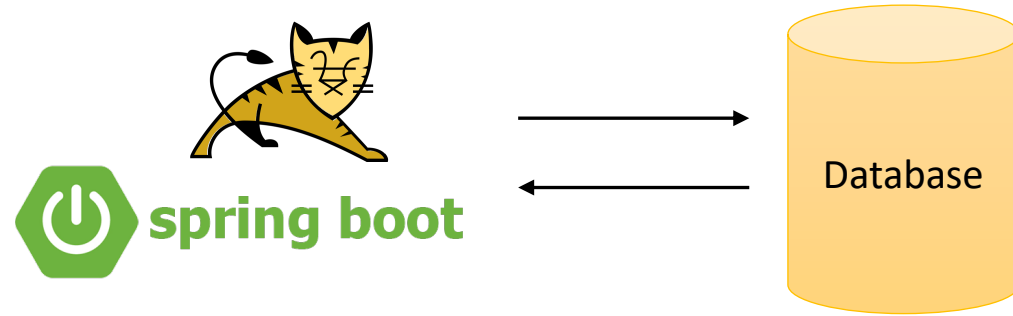
```
@Transactional(readOnly = true)
fun countLoanedBook(): Int {
    return userLoanHistoryRepository.countByStatus(UserLoanStatus.LOANED).toInt()
}
```

# 서버 코드를 보고 서버의 동작을 생각할 수 있어야 한다

```
select count(*) from user_loan_history where status = ?;
```



# 서버 코드를 보고 Query를 생각할 수 있어야 한다!



30L을 30으로 변환한다.

```
@Transactional(readOnly = true)
fun countLoanedBook(): Int {
    return userLoanHistoryRepository.countByStatus(UserLoanStatus.LOANED).toInt()
}
```

# 코드 변경 후 방법은!

```
@Transactional(readOnly = true)
fun countLoanedBook(): Int {
    return userLoanHistoryRepository.countByStatus(UserLoanStatus.LOANED).toInt()
}
```

- 1) DB로부터 숫자를 가져와
- 2) 적절히 타입을 변환해준다.

# 어떤 방법이 더 좋을까?!

겉보기에 두 기능은 완전히 동일하다.  
하지만 내부 동작은 전혀 다르다.



# 어떤 방법이 더 좋을까?!

전체 데이터 쿼리  
메모리 로딩 + size

count 쿼리  
타입 변환

# 어떤 방법이 더 좋을까?!

전체 데이터 쿼리  
메모리 로딩 + size

count 쿼리  
타입 변환

# 어떤 방법이 더 좋을까?!

전체 데이터 쿼리  
메모리 로딩 + size

count 쿼리  
타입 변환

DB 및 Network 부하, 애플리케이션 부하가 덜 든다.  
10만건, 100만건, 1000만건, 1억건 → 부하를 고려해야 한다.

# 어떤 방법이 더 좋을까?!

전체 데이터 쿼리  
메모리 로딩 + size

count 쿼리  
타입 변환

DB 및 Network 부하, 애플리케이션 부하가 덜 든다.

# 분야별 통계 - 기존과 어떤 차이가 있을까?!

```
@Transactional(readOnly = true)
fun getBookStatistics(): List<BookStatResponse> {
    return bookRepository.findAll() (Mutable)List<Book!>
        .groupBy { book -> book.type } Map<BookType, List<Book!>>
        .map { (type, books) -> BookStatResponse(type, books.size) }
}
```

# 분야별 통계 - 기존과 어떤 차이가 있을까?!

```
@Transactional(readOnly = true)
fun getBookStatistics(): List<BookStatResponse> {
    return bookRepository.getStats()
}
```

```
@Query(
    "SELECT NEW com.group.libraryapp.dto.book.response.BookStatResponse(b.type, COUNT(b.id)) " +
    "FROM Book b GROUP BY b.type"
)
fun getStats(): List<BookStatResponse>
```

# 어떤 방법이 더 좋을까?!

전체 데이터 쿼리  
메모리 로딩 + grouping

group by 쿼리

# 어떤 방법이 더 좋을까?!

전체 데이터 쿼리  
메모리 로딩 + grouping

group by 쿼리



# 어떤 방법이 더 좋을까?!

전체 데이터 쿼리  
메모리 로딩 + grouping

group by 쿼리

(상황에 따라 다르지만)  
Network 부하, 애플리케이션 부하가 덜 든다.  
인덱스를 이용해 튜닝할 여지가 있다.

# 사실은 끝이 아니다!

데이터의 양 / 트래픽 / 다른 비즈니스 요구사항  
등에 따라 **또 다른 방법**을 사용해야 할 수 있다.

# 사실은 끝이 아니다!

대용량 통계 처리 배치를 이용한 구조

이벤트 발행과 메시징 큐를 이용한 구조

# 36강. 세 번째 요구사항 클리어!

# 요구사항 3 추가하기

## 책 통계 화면

- 현재 대여 중인 책이 몇 권이 보여준다.
- 분야별로 도서관에 등록되어 있는 책이 각각 몇 권인지 보여준다.

등록하기   목록   히스토리   통계

등록된 책 분류	총 권 수
과학	3권
사회	5권
경제	12권
컴퓨터	4권
언어	1권

대출 중인 권 수 : 10권

## #5 세 번째 요구사항 추가하기 - 책 통계

1. SQL의 다양한 기능들(sum, avg, count, group by, order by)을 이해한다.
2. 간결한 함수형 프로그래밍 기법을 사용해보고 익숙해진다.
3. 동일한 기능을 애플리케이션과 DB로 구현해보고 특징과 장단점을 이해한다.

**감사합니다**