주니어 개발자의 성능 개선 도전기

발표자 : 이상민 발표 일자 : 2024/01/27

자기소개

이상민

마이다스인 채용 솔루션 개발팀 Backend 개발자

목차

주니어 개발자의 성능 개선 도전기

개요

발표 소개

2년차 주니어 웹 백엔드 개발자

성능 개선 도전기

주니어 개발자의 성능 개선도전기

발표 소개

2년차 주니어 웹 백엔드 개발자

성능 개선 도전기

엄청난 고급 기술 X

주니어 개발자의 성능 개선도전기

발표 소개

2년차 주니어 웹 백엔드 개발자

성능 개선 도전기

엄청난 고급 기술 X

성능을 고려한 개발의 중요성

주니어 개발자의 성능 개선도전기

발표 소개

2년차 주니어 웹 백엔드 개발자

성능 개선 도전기

엄청난 고급 기술 X

성능을 고려한 개발의 중요성

성능 개선을 앞둔 분들께 조금이나마 도움

주니어 개발자의 성능 개선도전기

성능을 고려한 개발의 중요성

태어난 지 8년이 가까이 되어가는 레거시 채용 솔루션 프로젝트 개발팀에 투입

주니어 개발자의 성능 개선도전기

성능을 고려한 개발의 중요성

태어난 지 8년이 가까이 되어가는 레거시 채용 솔루션 프로젝트 개발팀에 투입

8년 간의 변화

성능을 고려한 개발의 중요성

태어난 지 8년이 가까이 되어가는 레거시 채용 솔루션 프로젝트 개발팀에 투입

8년 간의 변화

솔루션의 성장과 함께 늘어난 고객사의 수

대량의 데이터를 다루는 대형 고객사의 등장

축적된 수 백만개의 데이터

대량 데이터 처리 기능에 대한 요청 증가

주니어 개발자의 성능 개선도전기

성능을 고려한 개발의 중요성

대량의 데이터 처리를 고려하지 않은 레거시 코드의 축적

성능 이슈 발생

고객사 A : 10000명이 넘는 지원자를 전형에 등록하는데 시간이 너무 오래 걸려요

고객사 B : 대량의 데이터를 엑셀로 추출하는데 너무 오래 걸리고 중간에 실패하는 경우도 있어요

고객사 C : PDF 추출하는데 언제까지 기다려야 해요??

고객사 D : 지원자가 만명이 넘는데 어느 세월에 100건 씩 나눠서 다운로드 받나요?

성능을 고려한 개발의 중요성

과거

- 100건, 1000건이 솔루션이 기능 상 지원하는 대량 데이터 처리의 기준
- 성능을 고려하지 않은 코드와 로직만으로도 충분히 소화 가능

현재

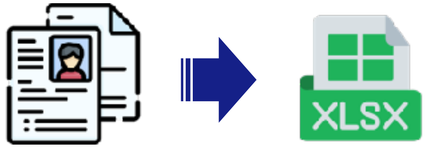
- 100건 정도의 처리는 몇 만 명의 지원자가 지원하는 대형 고객사에게는 턱없이 부족한 스펙
- 수 천, 수 만 건의 데이터 처리 필요
- 기존 로직으로는 너무나도 느린 성능

주니어 개발자의 성능 개선도전기

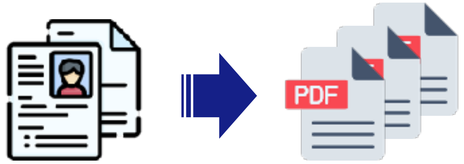
솔루션 전반적인 성능 개선 임무

고객사들의 성능 문제로 인한 페인 포인트 분석

대량의 지원자 CRUD 기능



대량 지원서 엑셀 다운로드 기능



대량의 지원서 PDF 추출

주니어 개발자의 성능 개선도전기

레거시 프로젝트의 성능 저하 요인들

너무 많은 데이터베이스 쿼리 요청

나도 모르게 성능을 느리게 하는 비즈니스 로직 외의 로직

컴퓨팅 리소스 부족으로 성능 개선의 한계

레거시 프로젝트에서의 성능 개선기

대량 데이터 CRUD 성능 개선 – 기존의 문제점

Insert를 반복문으로 처리

```java
for (MemberDto memberDto : memberDtoList) {
    Member member = memberDto.toEntity();
    dao.insert(member);
}
```

만 명을 저장하기 위해 만 번의 쿼리 수행

```
insert into member values(값, 값, 값);
insert into member values(값, 값, 값);
insert into member values(값, 값, 값);
insert into member values(값, 값, 값);
insert into member values(값, 값, 값);
.
.
.
```

너무 많은 insert 쿼리 요청

주니어 개발자의 성능 개선도전기

대량 데이터 CRUD 성능 개선 – 단건 쿼리를 벌크 쿼리로

리스트로 모아서 한번에 bulk insert 쿼리

```
List<Member> memberList = new ArrayList<>();

for (MemberDto memberDto : memberDtoList) {
    Member member = memberDto.toEntity();
    memberList.add(member);
}

dao.bulkInsert(memberList);
```

```
<insert id="bulkInsert" useGenerateKey="true" keyProperty="memberSn">
  INSERT INTO member (name, age, id, password, ...)
  VALUES
  <foreach collection="memberList" separator="," item="member">
    #{member.name},
    #{member.age},
    #{member.id},
    #{member.password}
    ...
  </foreach>
</insert>
```

단 한번의 bulkInsert 쿼리로 만 명의 데이터 저장

```
insert into member values (값, 값, 값, 값), (값, 값, 값, 값)...
```

대량 데이터 CRUD 성능 개선 – 단건 쿼리를 벌크 쿼리로(시행착오)

그래서 누가 이걸 몰라?

대량 데이터 CRUD 성능 개선 – 단건 쿼리를 벌크 쿼리로(시행착오)

그래서 누가 이걸 몰라?

사실 처음엔 저는 몰랐습니다…

대량 데이터 CRUD 성능 개선 – 단건 쿼리를 벌크 쿼리로(시행착오)

그래서 누가 이걸 몰라?

사실 처음엔 저는 몰랐습니다…

하지만 이런 간단한 성능 개선 방법도 실제 프로젝트에서는
바로 적용하기 힘든 경우가 많았습니다

대량 데이터 CRUD 성능 개선 – 단건 쿼리를 벌크 쿼리로(시행착오)

그래서 누가 이걸 몰라?

사실 처음엔 저는 몰랐습니다…

하지만 이런 간단한 성능 개선 방법도 실제 프로젝트에서는
바로 적용하기 힘든 경우가 많습니다

저는 힘들었습니다…

주니어 개발자의 성능 개선도전기

대량 데이터 CRUD 성능 개선 – 단건 반복 쿼리를 벌크 쿼리로(시행착오)

처음엔 이 메서드도 하나의 책임만 가졌을 겁니다

```
public void createScreeningResumeList(List<Resume> ResumeList,
                                       Screening screening) {
    for (Resume resume : resume) {
        ScreeningResume screeningResume = new ScreeningResume();
        screeningResume.setResumeSn(resume.getResumeSn());
        screeningResume.setScreeningSn(screening.getScreeningSn());
        ...
        dao.insert(screeningResume);
    }
}
```

```
public void createScreeningResumeListAndHistory(List<Resume> ResumeList,
                                                 Screening screening) {
    for (Resume resume : resume) {
        // 전형 지원서 저장
        ScreeningResume sr = new ScreeningResume();
        sr.setResumeSn(resume.getResumeSn());
        sr.setScreeningSn(screening.getScreeningSn());
        ...
        // insert 되면 PK값이 auto_generate되어 screeningResume에 세팅
        screeningResumeDao.insert(sr);

        // 히스토리 테이블 저장
        ScreeningResumeHistory srh = new screeningResumeHistory();
        srh.setScreeningResumeSn(sr.getScreeningResumeSn());
        ...
        screeningResumeHistoryDao.insert(srh);
    }
}
```

```
public void createScreeningResumeListAndHistoryAnd...(
  List<Resume> ResumeList, Screening screening, ....) {
    for (Resume resume : resume) {
        // 전형 지원서 저장
        ScreeningResume sr = new ScreeningResume();
        sr.setResumeSn(resume.getResumeSn());
        sr.setScreeningSn(screening.getScreeningSn());
        ...
        // insert 되면 PK값이 auto_generate되어 screeningResume에 세팅
        screeningResumeDao.insert(sr);

        // 히스토리 테이블 저장
        ScreeningResumeHistory srh = new screeningResumeHistory();
        srh.setScreeningResumeSn(sr.getScreeningResumeSn());
        ...
        screeningResumeHistoryDao.insert(srh);
        ...
        // 연관 테이블 추가 저장
        // 연관 테이블 추가 저장
        // 연관 테이블 추가 저장
        // 연관 테이블 추가 저장
        // 연관 테이블 추가 저장
        // 연관 테이블 추가 저장
        // 메서드 호출 안에서 연관 테이블 저장

    }
}
```

전형에 지원서를 배정하는 기능을 만들어 주세요

이번 주 내로 지원서가 전형에 배정된 기록을 남겨주세요

추가된 전형 지원서의 채용분야의 평가자가 이미 배정되어 있다면 바로 평가자가 배정되게 해주세요 그리고 면접전형인 경우에는 면접일자도 추가해주시고…
@#$@## 도 추가되면 좋겠네요
ASAP!!!!!!!!!

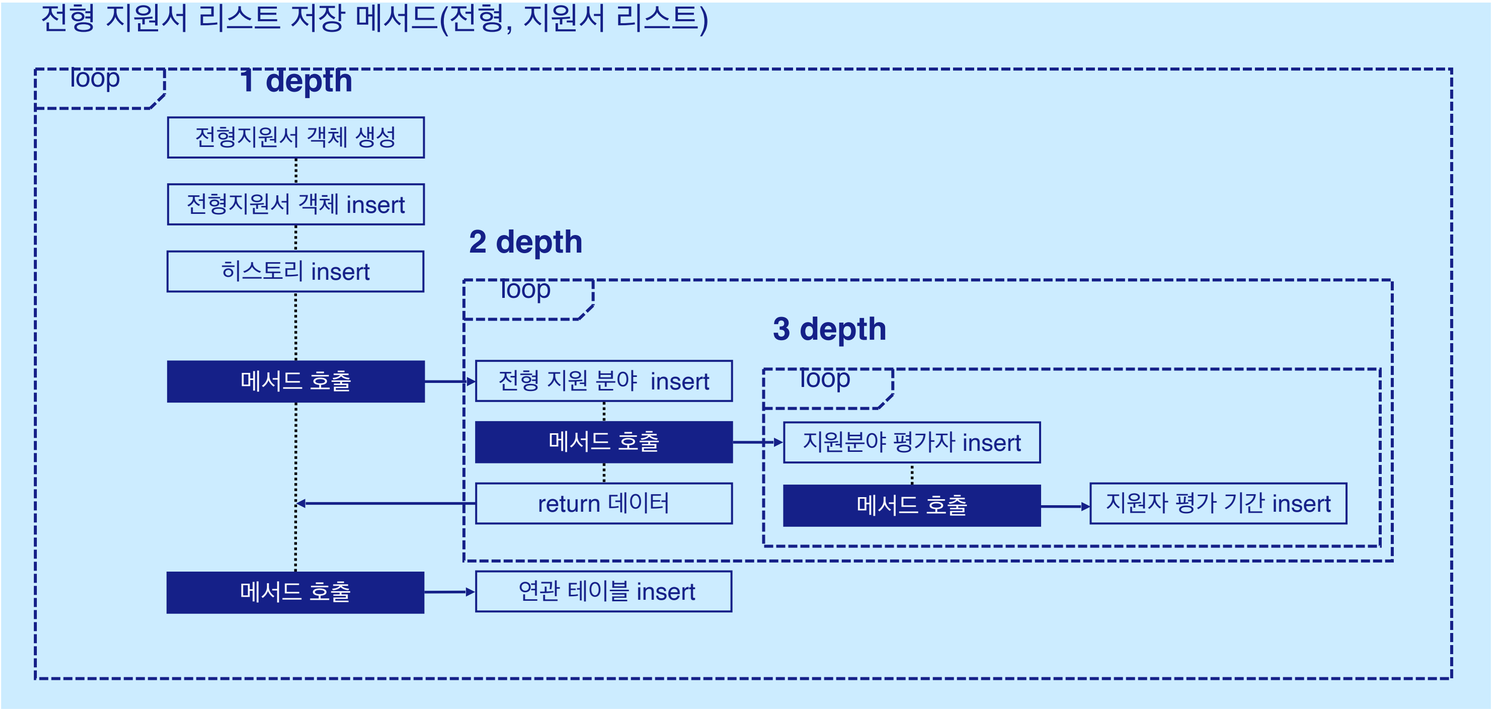
대량 데이터 CRUD 성능 개선 – 단건 쿼리를 벌크 쿼리로(시행착오)

실제 프로젝트에서, 특히 레거시 코드는 그렇게 간단하게 작성되어 있지 않다

대량 데이터 CRUD 성능 개선 – 단건 쿼리를 벌크 쿼리로(시행착오)

실제 프로젝트에서, 특히 레거시 코드는 그렇게 간단하게 작성되어 있지 않다

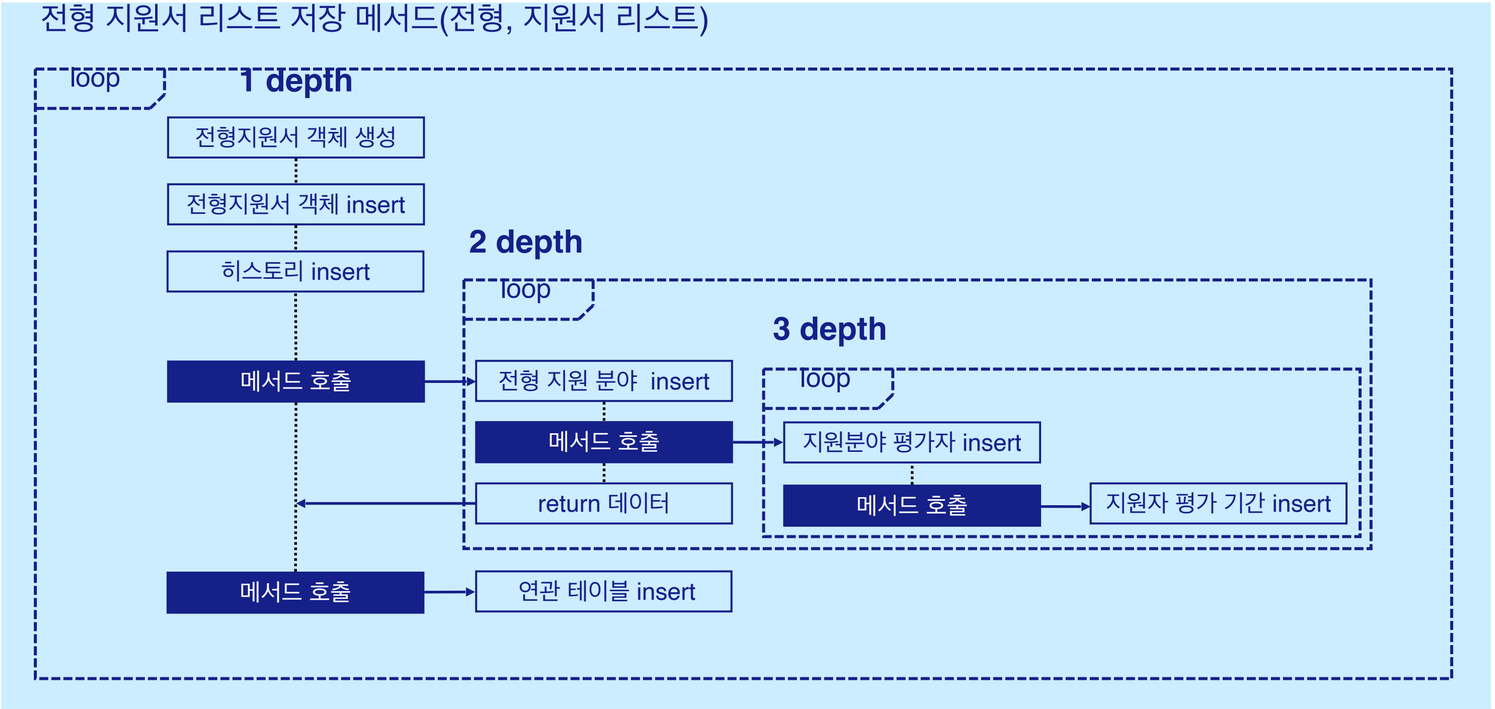
반복문 안에 곳곳에 숨겨진 복잡하게 얽힌 연관 테이블들을 저장하는 로직



대량 데이터 CRUD 성능 개선 – 단건 쿼리를 벌크 쿼리로(시행착오)

실제 프로젝트에서, 특히 레거시 코드는 그렇게 간단하게 작성되어 있지 않다

반복문 안에 곳곳에 숨겨진 복잡하게 얽힌 연관 테이블들을 저장하는 로직



벌크 쿼리를 모두 적용하려면 연관된 모든 테이블의 insert 로직을 반복문 바깥으로 빼내야 한다

대량 데이터 CRUD 성능 개선 – 단건 반복 쿼리를 벌크 쿼리로(시행착오)

대량의 벌크 인서트 로직이 있다면 연관된 테이블을 저장하는 로직을

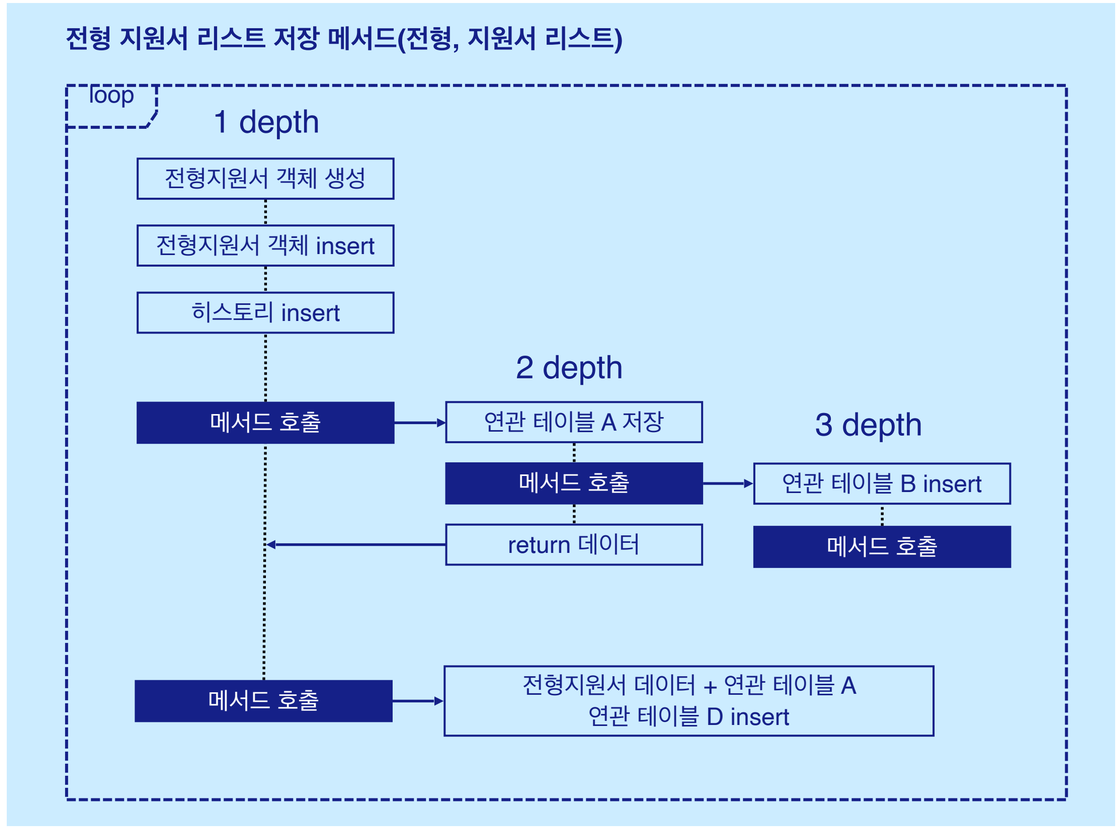
원자성있고 플랫하게 만들자

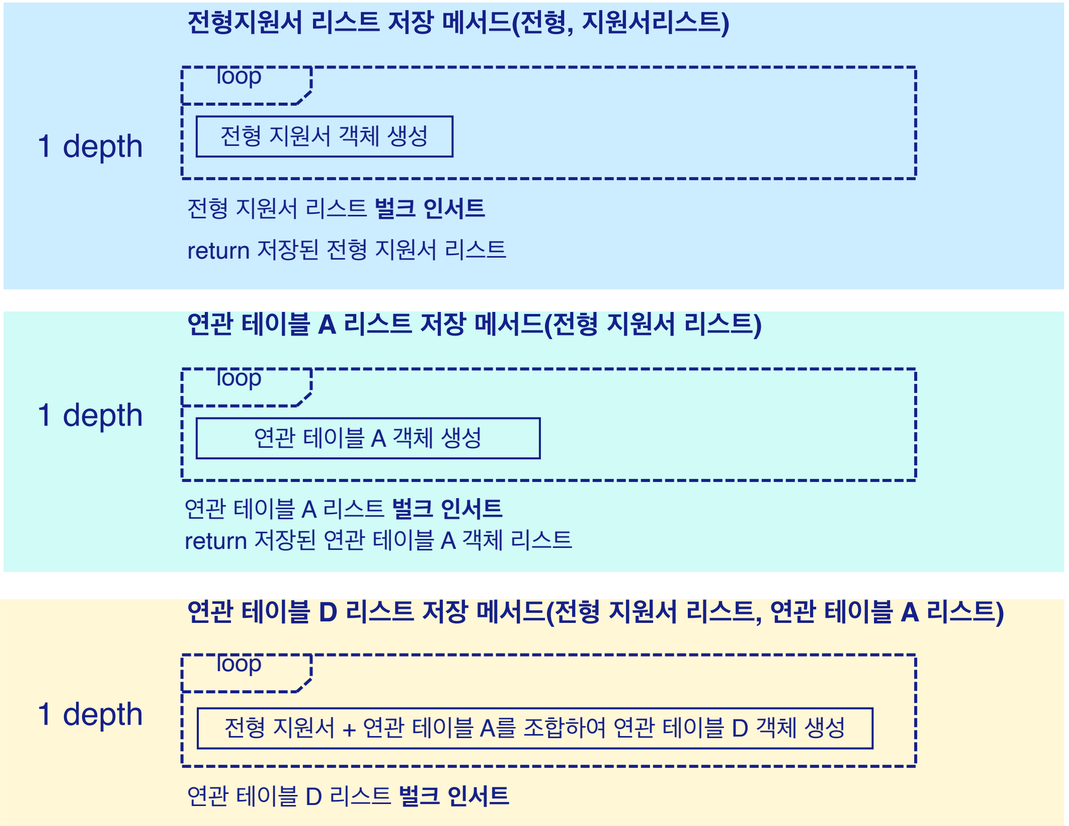
주니어 개발자의 성능 개선도전기

대량 데이터 CRUD 성능 개선 – 단건 반복 쿼리를 벌크 쿼리로(시행착오)

대량의 벌크 인서트 로직이 있다면 연관된 테이블을 저장하는 로직을

원자성 있고 플랫하게 만들자





대량 데이터 CRUD 성능 개선 – 단건 반복 쿼리를 벌크 쿼리로(시행착오)

두 번 째 시련

오히려 서버 환경에서는 오히려 벌크 쿼리가 더 느린 현상

```
|3432|==>|StopWatch 'bulkInsert vs LoopInsert Test ! dataSize : 1000':
running time (millis) = 20712
-----------------------------------------
ms     %     Task name
-----------------------------------------
18947  024%  bulkInsert(벌크 인서드)
01765  002%  LoopInsert(단건 인시트 반복문)
!
```

지옥의 디버깅 시작

대량 데이터 CRUD 성능 개선 – 단건 반복 쿼리를 벌크 쿼리로(시행착오)

처음엔 Spring의 StopWatch로 로그를 남겨 각 로직 별 수행시간을 측정

```
StopWatch stopWatch = new StopWatch("성능 측정");

stopWatch.start("methodA 소요 시간");
methodA();
stopWatch.stop();

stopWatch.start("methodB 수유 시간");
methodB();
stopWatch.stop();

log.info(stopWatch.prettyPrint());
```

```
StopWatch '성능 측정': 4.012633699 seconds
-----------------------------------------
Seconds        %       Task name
-----------------------------------------
3.007875599    75%     methodA 소요 시간
1.0047581      25%     methodB 소요 시간
```

대량 데이터 CRUD 성능 개선 – 단건 반복 쿼리를 벌크 쿼리로(시행착오)

처음엔 Spring의 StopWatch로 로그를 남겨 각 로직 별 수행시간을 측정


```
sqlSession.insert("bulkInsertScreeningResume", list);
```

대량 데이터 CRUD 성능 개선 – 단건 반복 쿼리를 벌크 쿼리로(시행착오)

처음엔 Spring의 StopWatch로 로그를 남겨 각 로직 별 수행시간을 측정


```
sqlSession.insert("bulkInsertScreeningResume", list);
```

실제 DB 쿼리 수행 로그에서는 bulk Insert가 매우 빠르게 수행된 것을 확인

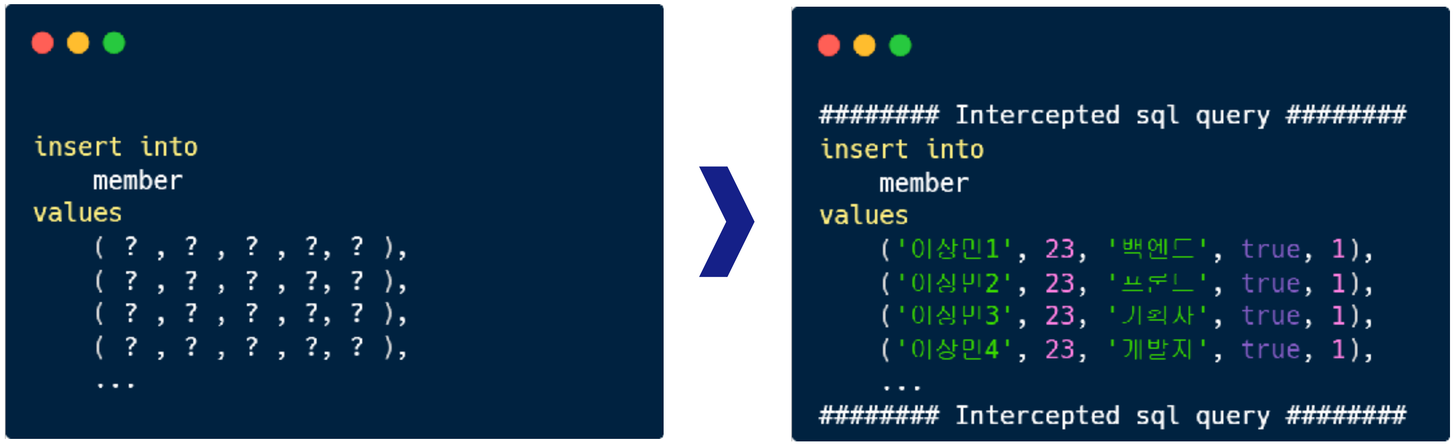
대량 데이터 CRUD 성능 개선 – 단건 반복 쿼리를 벌크 쿼리로(시행착오)

이는 곧 어플리케이션 단의 성능 저하가 발생하고 있음을 의미

PinPoint, VisualVM 등 모니터링 툴을 사용해 얻은 단서
벌크 쿼리가 수행될 경우 많은 메모리와 CPU 사용량이 든다는 것

대량 데이터 CRUD 성능 개선 – 단건 반복 쿼리를 벌크 쿼리로(시행착오)

범인은 쿼리 로깅을 위한 커스텀 마이바티스 인터셉터



```
insert into
    member
values
    ( ? , ? , ? , ?, ? ),
    ( ? , ? , ? , ?, ? ),
    ( ? , ? , ? , ?, ? ),
    ( ? , ? , ? , ?, ? ),
    ...
```

```
######## Intercepted sql query ########
insert into
    member
values
    ('이상민1', 23, '백엔드', true, 1),
    ('이상민2', 23, '프론트', true, 1),
    ('이상민3', 23, '기획사', true, 1),
    ('이상민4', 23, '게발지', true, 1),
    ...
######## Intercepted sql query ########
```

쿼리를 실행 시키기 전 인터셉트 하여 런타임의 실제 객체의 값으로 SQL 로그를 이쁘게 찍어주는 로깅 인터셉터

코어 DB의 모듈에서 등록한 인터셉터로 이런 성능저하가 발생하는 지도 모른 채 개발

전혀 비즈니스 로직과 상관없는 로깅으로 인해 성능 저하가 일어나고 있었던 것

대량 데이터 CRUD 성능 개선 – 단건 반복 쿼리를 벌크 쿼리로(시행착오)

Reflection으로 인한 성능 저하 발생

```
public Field getFieldByFieldName(Object obj, String fieldName) {
    if (obj == null) {
        return null;
    }
    Field field = null;
    for (Class<?> clazz = obj.getClass(); clazz != Object.class; clazz = clazz.getSuperclass()) {
        try {
            field = clazz.getDeclaredField(fieldName);
            break;
        } catch (NoSuchFieldException e) {
            logger.debug("error on getFieldByFieldName()");
        }
    }
    return field;
}
```

컴파일 타임에는 알 수 없는 런타임 객체의 값을 매핑하기 위해 리플렉션을 사용

리플렉션은 런타임 시점에 Class를 동적으로 Heap 공간에 Load하고, JVM이 코드를 최적화 할 수 없기에 속도가 느리고 더 많은 CPU 사이클을 필요로 함

벌크 쿼리 수행 시 한번에 너무 많은 객체를 리플렉션 하다보니 서버 환경의 CPU와 메모리로는 오히려 성능저하 발생

대량 데이터 CRUD 성능 개선 – 기존의 문제점

지원자 정보와 지원자의 경력 사항 같이 포함된 객체

```
/* 경력과 자격증 정보를 포함한 지원자 정보 */
public Class ApplicantInfo {
  // 지원자 정보
  private Applicant applicant;
  // 지원자의 직장 경력 목록
  private List<Carrer> carrerList;
  // 지원자의 수상 경력 목록
  private List<Award> awardList;
  // 지원자의 자격증 목록
  private List<License> liecenseList;
}
```

1:N 관계의 테이블을 조회하는 예제

주니어 개발자의 성능 개선도전기

대량 데이터 CRUD 성능 개선 – 기존의 문제점

너무 많은 조회 쿼리(N+1 문제)

```
List<Applicant> applicantList = applicantDao.findList(applicantSnList);

for (Applicant applicant : applicantList) {
    Integer applicantSn = applicant.getApplicantSn();
    // 커리어 리스트 조회
    List<Carrer> carrerList = careerDao.findByApplicantSn(applicantSn);
    // 수상경력 리스트 조회
    List<Award> awardList = awardDao.findByApplicantSn(applicantSn);
    // 자격증 리스트 조회
    List<License> LicenseList = licenseDao.findByApplicantSn(applicantSn);

    ApplicantInfo applicantInfo = new ApplicantInfo(applicant, carrerList, awardList, LicneseList)
}
```

주니어 개발자의 성능 개선도전기

대량 데이터 CRUD 성능 개선 – 기존의 문제점

너무 많은 조회 쿼리(N+1 문제)

```
List<Applicant> applicantList = applicantDao.findList(applicantSnList);

for (Applicant applicant : applicantList) {
    Integer applicantSn = applicant.getApplicantSn();
    // 커리어 리스트 조회
    List<Carrer> carrerList = careerDao.findByApplicantSn(applicantSn);
    // 수상경력 리스트 조회
    List<Award> awardList = awardDao.findByApplicantSn(applicantSn);
    // 자격증 리스트 조회
    List<License> LicenseList = licenseDao.findByApplicantSn(applicantSn);

    ApplicantInfo applicantInfo = new ApplicantInfo(applicant, carrerList, awardList, LicneseList)
}
```

① 리스트 조회(1)

대량 데이터 CRUD 성능 개선 – 기존의 문제점

너무 많은 조회 쿼리(N+1 문제)

```
List<Applicant> applicantList = applicantDao.findList(applicantSnList);

for (Applicant applicant : applicantList) {
    Integer applicantSn = applicant.getApplicantSn();
    // 커리어 리스트 조회
    List<Carrer> carrerList = careerDao.findByApplicantSn(applicantSn);
    // 수상경력 리스트 조회
    List<Award> awardList = awardDao.findByApplicantSn(applicantSn);
    // 자격증 리스트 조회
    List<License> LicenseList = licenseDao.findByApplicantSn(applicantSn);

    ApplicantInfo applicantInfo = new ApplicantInfo(applicant, carrerList, awardList, LicneseList)
}
```

① 리스트 조회(1)

② 반복문 내에서 1:N 관계 테이블 조회(10000*3)

주니어 개발자의 성능 개선도전기

대량 데이터 CRUD 성능 개선 – 기존의 문제점

너무 많은 조회 쿼리(N+1 문제)

```
List<Applicant> applicantList = applicantDao.findList(applicantSnList);

for (Applicant applicant : applicantList) {
    Integer applicantSn = applicant.getApplicantSn();
    // 커리어 리스트 조회
    List<Carrer> carrerList = careerDao.findByApplicantSn(applicantSn);
    // 수상경력 리스트 조회
    List<Award> awardList = awardDao.findByApplicantSn(applicantSn);
    // 자격증 리스트 조회
    List<License> LicenseList = licenseDao.findByApplicantSn(applicantSn);

    ApplicantInfo applicantInfo = new ApplicantInfo(applicant, carrerList, awardList, LicneseList)
}
```

① 리스트 조회(1)

② 반복문 내에서 1:N 관계 테이블 조회(10000*3)

페이징이 있는 경우는 괜찮으나 페이징이 없는 경우에는?

10000명 기준 1(지원자 리스트 조회) + 30000(각각의 1:N 관계 테이블 조회) 총 30001 번의 조회 쿼리

주니어 개발자의 성능 개선도전기

대량 데이터 CRUD 성능 개선 – IN절과 Map을 사용한 조회 쿼리 최소화

```
private Map<Integer, List<Carrer>> getCarrerMap(List<Integer> applicantSnList)
{
    // 커리어 리스트 In절로 모든 지원자의 커리어 일괄 조회
    List<Carrer> carrerList = careerDao.findListIn(applicantSnList);

    // 지원자 식별 번호로 그룹핑
    Map<Integer, List<Carrer>> carrearMap =
      carrerList.stream().collect(groupingBy(Carrer::getApplicantSn));

    return carrearMap;
}
```

주니어 개발자의 성능 개선도전기

대량 데이터 CRUD 성능 개선 – IN절과 Map을 사용한 조회 쿼리 최소화

```
private Map<Integer, List<Carrer>> getCarrerMap(List<Integer> applicantSnList)
{
    // 커리어 리스트 In절로 모든 지원자의 커리어 일괄 조회
    List<Carrer> carrerList = careerDao.findListIn(applicantSnList);

    // 지원자 식별 번호로 그룹핑
    Map<Integer, List<Carrer>> carrearMap =
      carrerList.stream().collect(groupingBy(Carrer::getApplicantSn));

    return carrearMap;
}
```

주니어 개발자의 성능 개선도전기

대량 데이터 CRUD 성능 개선 – IN절과 Map을 사용한 조회 쿼리 최소화

```
private Map<Integer, List<Carrer>> getCarrerMap(List<Integer> applicantSnList)
{
    // 커리어 리스트 In절로 넘는 지원자의 커리어 전부 조회
    List<Carrer> carrerList = careerDao.findListIn(applicantSnList);

    // 지원자 식별 넘버로 그룹핑
    Map<Integer, List<Carrer>> carrearMap =
      carrerList.stream().collect(groupingBy(Carrer::getApplicantSn));

    return carrearMap;
}
```

```
// 지원자 리스트 조회
List<Applicant> applicantList = applicantDao.findList(applicantSnList);
// 모든 지원자의 커리어 Map
Map<Integer, List<Carrer>> carrearMap = getCarrerMap(applicantSnList);
// 모든 지원자의 수상 경력 Map
Map<Integer, List<Award>> awardMap = getAwardMap(applicantSnList);
// 모든 지원자의 자격증 Map
Map<Integer, List<License>> licenseMap = getlicenseMap(applicantSnList);

for (Applicant applicant : applicantList) {
    Integer applicantSn = applicant.getApplicantSn();
    // Map에서 조회
    List<Carrer> carrerList = carrearMap.get(applicantSn);
    List<Award> awardList = awardMap.get(applicantSn);
    List<License> licenseList = licenseMap.get(applicantSn);

    ApplicantInfo applicantInfo =
      new ApplicantInfo(applicant, carrerList, awardList, LicneseList)
}
```

주니어 개발자의 성능 개선도전기

대량 데이터 CRUD 성능 개선 – IN절과 Map을 사용한 조회 쿼리 최소화

```
private Map<Integer, List<Carrer>> getCarrerMap(List<Integer> applicantSnList)
{
    // 커리어 리스트 In절로 모든 지원자의 커리어 모두 조회
    List<Carrer> carrerList = careerDao.findListIn(applicantSnList);

    // 지원자 식별 번호로 그룹핑
    Map<Integer, List<Carrer>> carrearMap =
      carrerList.stream().collect(groupingBy(Carrer::getApplicantSn));

    return carrearMap;
}
```

```
// 지원자 리스트 조회
List<Applicant> applicantList = applicantDao.findList(applicantSnList);
// 모든 지원자의 커리어 Map
Map<Integer, List<Carrer>> carrearMap = getCarrerMap(applicantSnList);
// 모든 지원자의 수상 경력 Map
Map<Integer, List<Award>> awardMap = getAwardMap(applicantSnList);
// 모든 지원자의 자격증 Map
Map<Integer, List<License>> licenseMap = getlicenseMap(applicantSnList);

for (Applicant applicant : applicantList) {
    Integer applicantSn = applicant.getApplicantSn();
    // Map에서 조회
    List<Carrer> carrerList = carrearMap.get(applicantSn);
    List<Award> awardList = awardMap.get(applicantSn);
    List<License> licenseList = licenseMap.get(applicantSn);

    ApplicantInfo applicantInfo =
      new ApplicantInfo(applicant, carrerList, awardList, LicneseList)
}
```

10000건 기준 1회(지원자 리스트 조회) + 3회(1:N 관계 테이블 리스트 조회) 총 4번의 조회 쿼리

대량 데이터 CRUD 성능 개선 – 정리

대부분 성능 저하는 시스템의 DB와 밀접한 연관이 있다

벌크 쿼리나 IN 절 조회 같은 방법을 통해 쿼리 실행 횟수를 최소화하자

너무 많은 책임을 가진 메서드는 벌크 쿼리 적용에 큰 걸림돌이 되므로 원자성 있고 플랫하게 만들자

인터셉터와 같이 비즈니스 로직 외의 로직에서도 성능 저하가 발생할 수도 있다

로깅과 모니터링 툴을 사용해 성능 저하가 발생하는 지점을 찾아 개선하자

대량 데이터 CRUD 성능 개선 – 슬로우 쿼리 제거

슬로우 쿼리란

DBMS가 요청 받은 쿼리를 수행할 때 일정 시간 이상 수행되지 못한 쿼리
성능 저하의 주범

슬로우 쿼리의 원인

잘못된 인덱스 설계, 데이터베이스 리소스 부족, 비효율적인 쿼리 등 다양한 이유가 있다.
대부분의 슬로우 쿼리는 풀 테이블 스캔을 유발하며 DB에 큰 부하를 일으켜 성능 저하를 일으킨다.

예방 방법 & 해결 방안

- DBMS의 슬로우 쿼리 로그 설정 활성화를 통한 슬로우 쿼리 추출
- EXPLANE을 통한 쿼리 실행 계획 분석 및 쿼리 최적화
- 올바른 테이블 설계와 올바른 인덱스 사용
- 좋은 쿼리(커버링 인덱스를 사용하는 쿼리) 작성하기

EXPLANE 사용법

Explain 명령어를 통해 옵티마이저가 산출한 쿼리의 실행 계획을 사용자가 확인할 수 있다.

```
explain select * from user where user.id = 1;
```

	id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	1	SIMPLE	S	const	PRIMARY		4	const	1	Using temporary; Using filesort
2	1	SIMPLE	SRN	ref	PRIMARY,s		4	const	5	Using where
3	1	SIMPLE	RN	eq_ref	PRIMARY		4		1	
4	1	SIMPLE	SR	ref	PRIMARY,s		4		35	Using where
5	1	SIMPLE	SAS	ref	PRIMARY,s		4		1	Using index condition; Using where
6	1	SIMPLE	SRS	eq_ref	PRIMARY		4		1	
7	1	SIMPLE	RAS	eq_ref	PRIMARY		4		1	
8	1	SIMPLE	U	ref	idx_user_		767		1	Using where
9	1	SIMPLE	R	eq_ref	PRIMARY		4		1	Using where

EXPLANE 사용법

type 칼럼

- 테이블에 접근하는 방식

- **system** < **const** < **eq_ref** < **ref** < fulltext < ref_or_null < index_merge < unique_subquery < index_subquery < range < index < **ALL**

rows 칼럼

- 쿼리를 수행하기 위해 읽어야 할 행 수를 예측한 값으로 정확하진 않지만 행 수가 적을 수록 좋음

possible_keys, key 칼럼

- possible_keys는 쿼리에서 사용 가능한 인덱스를, key는 실제로 사용한 인덱스
- possible_keys가 있는데 key가 없다면 인덱스를 사용하지 않았음을 의미

extra 칼럼

- 쿼리 실행 계획의 추가 정보
- 옵티마이저가 어떻게 동작하는 지에 대해 알려주는 힌트 값
- **GOOD : Using Index, Using Where** / **BAD : Using Filesort, Using Join Buffer, Using Temporary**

대량 데이터 CRUD 성능 개선 – 슬로우 쿼리 제거

EXPLAIN으로 슬로우 쿼리를 없애 보자

슬로우 쿼리의 실행 계획

조회 로우 수 : 5000개

수행속도 : 50s

	id	select_type	table	type	rows	Extra
1	1	SIMPLE	S	const	1	Using temporary; Using filesort
2	1	SIMPLE	SRN	ref	5	Using where
3	1	SIMPLE	RN	eq_ref	1	
4	1	SIMPLE	SR	ref	35	Using where
5	1	SIMPLE	SAS	ref	1	Using where
6	1	SIMPLE	SRS	eq_ref	1	
7	1	SIMPLE	RAS	eq_ref	1	
8	1	SIMPLE	R	eq_ref	1	Using where
9	1	SIMPLE	U	ALL	21135	Using where; Using join buffer (flat, BNL join)

조인한 U 테이블에서 type이 ALL로 20000개의 로우를 풀스캔하여 슬로우 쿼리 발생

대량 데이터 CRUD 성능 개선 – 슬로우 쿼리 제거

EXPLAIN으로 슬로우 쿼리를 없애 보자

U테이블의 조인하는 칼럼에 인덱스 추가 후 실행 계획

조회 로우 수 : 5000개

수행속도 : 107ms

	id	select_type	table	type	rows	Extra
1	1	SIMPLE	S	const	1	Using temporary; Using filesort
2	1	SIMPLE	SRN	ref	5	Using where
3	1	SIMPLE	RN	eq_ref	1	
4	1	SIMPLE	SR	ref	35	Using where
5	1	SIMPLE	SAS	ref	1	Using where
6	1	SIMPLE	SRS	eq_ref	1	
7	1	SIMPLE	RAS	eq_ref	1	
8	1	SIMPLE	U	ref	1	Using where
9	1	SIMPLE	R	eq_ref	1	Using where

Explain으로 분석 후 인덱스 추가라는 간단한 방법으로 큰 성능 개선 성공

대량 데이터 CRUD 성능 개선 – 정리

슬로우 쿼리는 성능 저하의 주범

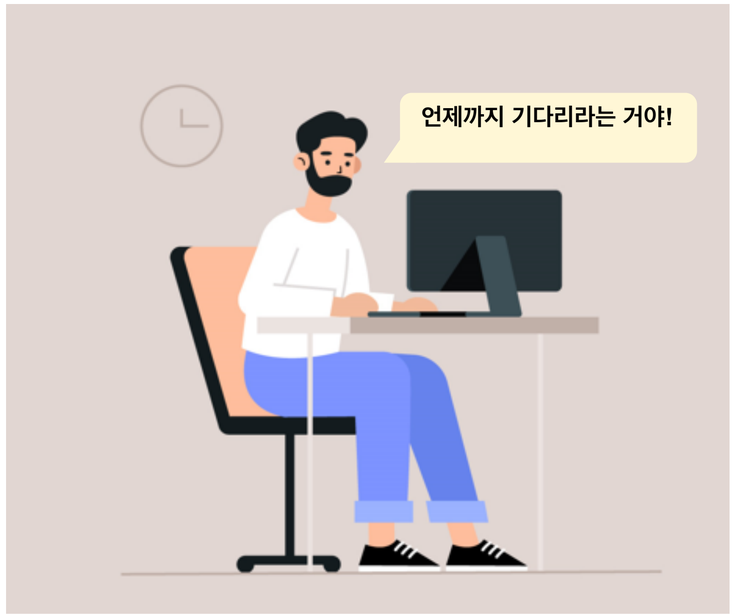
쿼리 작성 시 실행 계획 분석을 통해 좋은 쿼리인지 확인하는 습관을 들이자

기존 슬로우 쿼리 발견 시 묵인하지 말고 바로 개선하자 폭탄은 늘 조용한 법

대용량 엑셀 다운로드 성능 개선 – 기존의 문제점

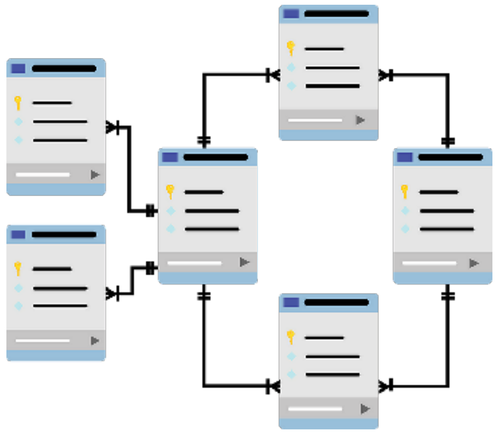
동기식 방식으로 불편한 사용성

엑셀을 생성 중입니다 잠시만 기다려주세요



대용량 엑셀 다운로드 성능 개선 – 기존의 문제점

느릴 수 밖에 없는 데이터 조회 로직



지원서와 연관된 백 개가 넘어가는 정규화 된 지원서 항목 테이블

지원서 한 명 씩 반복문을 도는 행 기반 조회 로직

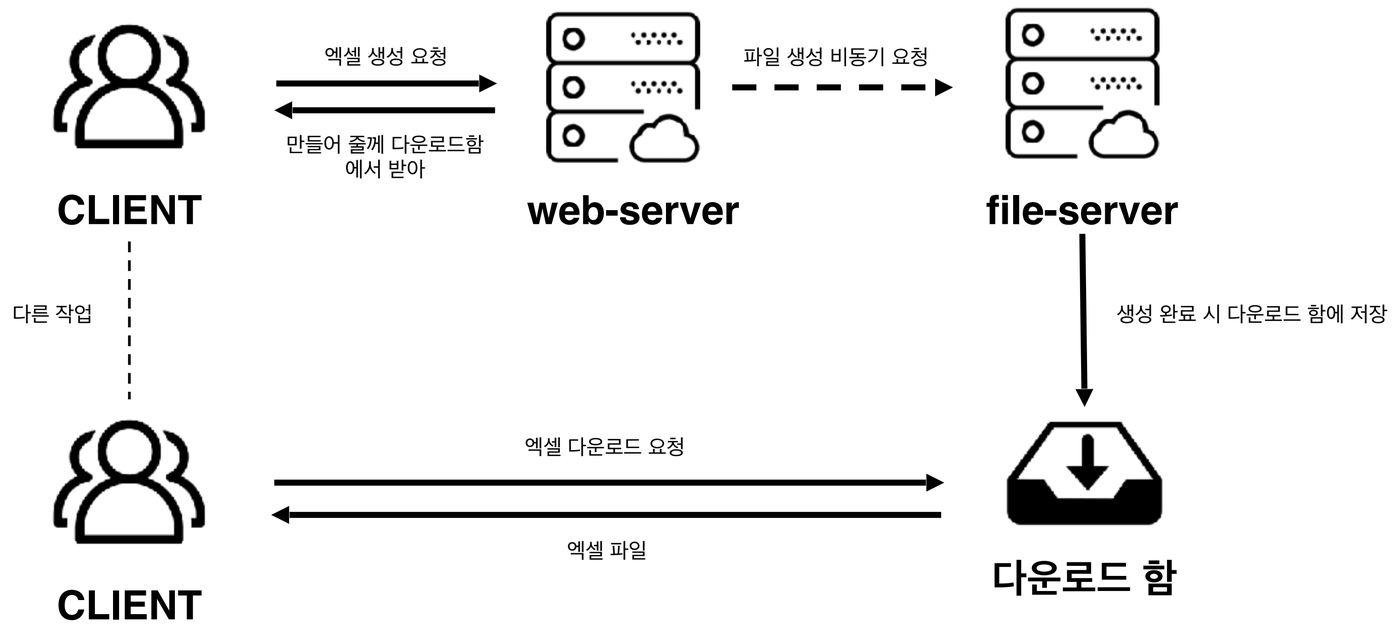
몇 백, 몇 천 명의 지원서를 엑셀 다운로드 시 기하급수적으로 늘어나는 쿼리 수행

몇 백 명 엑셀 다운로드에도 수십분이 소요

주니어 개발자의 성능 개선도전기

대용량 엑셀 다운로드 성능 개선 – 동기를 비동기로

요청을 받는 웹서버와 파일을 생성하는 파일서버로 분리
별도의 파일 다운로드함 기능 개발



오래 걸리는 작업은 비동기 방식으로 변경하여 사용자의 불편함을 덜어주자

주니어 개발자의 성능 개선도전기

대용량 엑셀 다운로드 성능 개선 – 행 기반 조회에서 열 기반 조회로

수험번호	이름	지원분야 : 1지망	지원분야 첫수	지원경로	이메일	핸드폰번호	가족사항 - 관계	병역	병역구분	최종학력	최종출신 대학교명	재직중인 대학교 전공	대학교1 - 학교명	연구실적(학술논문 발표) - 발표구분	총 경력	총 프로젝트	프로젝트 - 제목	총 자격증 개수	자격증 - 명칭
0049-000008	이씨고슬유학	1지망 : 소업-회계-프론트엔드	1	-	test01@na.com	010-0000-0000	2남 1녀	무고	파이동 허야지	ㅂ대상	고슬(유학)	-	-	-	3년	0	A 프로젝트	1개	정보처리기사
0049-000009	이씨물가조회	1지망 : 경력-IT-백엔드	1	-	test02@na.com	010-0000-0000	2남 1녀	무고	파이동 허야지	ㅂ대상	고슬(유학)	-	-	-	1년	0	A 프로젝트	1개	정보처리기사
0049-000010	이씨고슬슬의	1지망 : 소업-회계-백엔드	1	-	test03@na.com	010-0000-0000	2남 1녀	무고	파이동 허야지	ㅂ대상	고슬(유학)	-	-	-	2년	0	A 프로젝트	3개	정보처리기사
0049-000011	이씨전학중덕	1지망 : 소업-회계-백엔드	1	-	test04@na.com	010-0000-0000	2남 1녀	무고	파이동 허야지	ㅂ대상	전문학사(유학)	조선이공대학교(분자)	주전공 : 경상계열(사회과학)_(산업회학))	조선이공대학교	4년	0	A 프로젝트	4개	정보처리기사
0049-000012	이씨전학중덕	1지망 : 소업-회계-백엔드	1	-	test05@na.com	010-0000-0000	2남 1녀	무고	파이동 허야지	ㅂ대상	전문학제(유학)	경복진문대학교(전후)	주전공 : 법,경찰학부(공학계열_(전기,전자))	경복진문대학교	5년	0	A 프로젝트	5개	정보처리기사
0049-000013	이씨전학재학	1지망 : 경력-IT-프론트엔드	3		test06@na.com	010-0000-0000	2남 1녀	무고	파이동 허야지	면제	전문학사	대전이공대학교(大德캠퍼스)(일본)	주전공 : 경상계열(공학계열_(교통운송))	대전이공대학교(大德캠퍼스)	5년	0	A 프로젝트	6개	정보처리기사
0049-000014	이씨전학중덕	1지망 : 경력-IT-프론트엔드	1	사인수개	test07@na.com	010-0000-0000	2남 1녀	무고	파이동 허야지	복무중	전문학사	서울과학기술대학교 NID융합기술대학원(서울)	주전공 : 항공,우주공학(공학계열_(건축))	서울과학기술대학교 NID융합기술대학원	6년	0	A 프로젝트	7개	정보처리기사
0049-000015	이씨주학	1지망 : 경력-IT-프론트엔드	1		test08@na.com	010-0000-0000	2남 1녀	무고	파이동 허야지	ㅂ대상	전문학사	전주대학교(전남)	주전공 : 수산기공학(공학계열_(교통운송))	전주대학교	7년	0	A 프로젝트	8개	정보처리기사
0049-000016	이씨학사	1지망 : 경력-IT-프론트엔드	1	-	test09@na.com	010-0000-0000	2남 1녀	무고	파이동 허야지	ㅂ대상	학사(경후)	서울대학교(서울)	주전공 : 법,경찰학부(공학계열_(건축))	서울대학교	8년	0	A 프로젝트	9개	정보처리기사
0049-000017	이씨학사유학	1지망 : 경력-IT-프론트엔드	1		test10@na.com	010-0000-0000	2남 1녀	무고	파이동 허야지	ㅂ대상	학사(유학)	서울대학교	주전공 : 행정사회복지학부(공학계열_(교통운송))	서울대학교	9년	0	A 프로젝트	10개	정보처리기사
0049-000018	이씨학사세학	1지망 : 경력-IT-백엔드	1	서칭고력	test11@na.com	010-0000-0000	2남 1녀	무고	파이동 허야지	비필	학사	-	주전공 : 건건시학석사(공학계열_(기계,금속))	-	10년	0	A 프로젝트	11개	정보처리기사
0049-000019	이씨석사송회	1지망 : 소업-회계-프론트엔드	1	거기이	test12@na.com	010-0000-0000	2남 1녀	무고	파이동 허야지	ㅂ대상	석사(송회)	-	-	-	11년	0	A 프로젝트	12개	정보처리기사
0049-000020	이씨석사유학	1지망 : 경력-IT-백엔드	1		test13@na.com	010-0000-0000	2남 1녀	무고	파이동 허야지	ㅂ대상	석사수료(유학)	서울대학교 산업대학교 NID융합기술대학원(산업니)(서울)	수전공 : 신호(공학계열_(교통운송))	서울대학교 산업대학교 NID융합기술대학원(산업니)	12년	0	A 프로젝트	13개	정보처리기사
0049-000021	이씨식수재학	1지망 : 경력-IT-프론트엔드	1	-	test14@na.com	010-0000-0000	2남 1녀	무고	파이동 허야지	ㅂ대상	식사수료	-	-	-	13년	0	A 프로젝트	14개	정보처리기사
0049-000022	박씨고등문회	1지망 : 신입-IT-프론트엔드	1	-	test15@na.com	010-0000-0000	2남 1녀	무고	파이동 허야지	ㅂ대상	고졸(문호)	-	-	-	14년	0	A 프로젝트	15개	정보처리기사
0049-000023	박씨고등졸이	1지망 : 경력-IT-프론트엔드	1	-	test16@na.com	010-0000-0000	2남 1녀	무고	파이동 허야지	ㅂ대상	고졸	-	-	-	15년	0	A 프로젝트	16개	정보처리기사

테이블 조회 테이블 조회 테이블 조회 테이블 조회 테이블 조회 테이블 조회 테이블 조회 테이블 조회 테이블 조회 테이블 조회

지원서 기준으로 행 기반 조회

주니어 개발자의 성능 개선도전기

대용량 엑셀 다운로드 성능 개선 – 행 기반 조회에서 열 기반 조회로

테이블 조회 테이블 조회 테이블 조회 테이블 조회 테이블 조회 테이블 조회

수험번호	이름	지원분야 : 1지망	지원분야 첫수	지원성로	이메일	핸드폰번호	가족사항 - 관계	승교	좌우명	병역구분	최종학력	최종출신 대학교명	최종졸업 대학교 전공	대학교1 - 학교명	연구실적(학술논문 발표) 발표구분	총 경력	총 프로젝트	프로젝트 - 제목	총 자격증 개수	자격증 - 명칭
0049-000008	이씨고슬유학	1지망 : 소업-회계-프론드엔드	1	-	test01@na.com	010-0000-0000	2남 1녀	무고	파이팅 허야지	비대상	고슬(유학)	-	-	-	논문 발표	3년	0	A 프로젝트	1개	정보처리기사
0049-000009	이씨고슬개학	1지망 : 경력-IT-백엔드	1	-	test02@na.com	010-0000-0000	2남 1녀	무고	파이팅 허야지	비대상	고슬	-	-	-	논문 발표	1년	0	A 프로젝트	2개	정보처리기사
0049-000010	이씨고슬슬어	1지망 : 소업-회계-백어드	1	-	test03@na.com	010-0000-0000	2남 1녀	무고	파이팅 허야지	비대상	고슬(유토)	-	-	-	논문 발표	2년	0	A 프로젝트	3개	정보처리기사
0049-000011	이씨저작권되	1지망 : 소업-회계-백인드	1		test04@na.com	010-0000-0000	2남 1녀	무고	파이팅 허야지	비대상	주문학사(유되)	조선이공대학교(불명)	주전공 : 경상계열(사회계열_(사회과학))	조선이공대학교	논문 발표	3년	0	A 프로젝트	4개	정보처리기사
0049-000012	이씨전학유학	1지망 : 소업-회계-백인드	1	-	test05@na.com	010-0000-0000	2남 1녀	무고	파이팅 허야지	비대상	전문학사(유학)	경복전문대학교(졸업)	주전공 : 법,경찰행정학부(공학계열_(전기·전자))	경복전문대학교	논문 발표	4년	0	A 프로젝트	5개	정보처리기사
0049-000013	이씨저작재딱	1지망 : 경력-IT-프론드엔드	1		test06@na.com	010-0000-0000	2남 1녀	무고	파이팅 허야지	면제	전문학사	대전이공대학교(大專課程)(일본)	주전공 : 경상계열(공학계열_(고등운송))	대전이공대학교(大專課程)	논문 발표	5년	0	A 프로젝트	6개	정보처리기사
0049-000014	이씨전학준비	1지망 : 경력-IT-프론드엔드	1	시인수계	test07@na.com	010-0000-0000	2남 1녀	무고	파이팅 허야지	복무중	전문학사	서울과학기술대학교 NID융합기술대학원(서울)	주전공 : 항공,우주공학(공학계열_(건축))	서울과학기술대학교 NID융합기술대학원	논문 발표	6년	0	A 프로젝트	7개	정보처리기사
0049-000015	이씨주파	1지망 : 경력-IT-프론드엔드	1		test08@na.com	010-0000-0000	2남 1녀	무고	파이팅 허야지	비대상	전문학사	전주대학교(전남)	주전공 : 수산기공학(공학계열_(교통·운송))	전주대학교	논문 발표	7년	0	A 프로젝트	8개	정보처리기사
0049-000016	이씨학시	1지망 : 경력-IT-프론드엔드	1	-	test09@na.com	010-0000-0000	2남 1녀	무고	파이팅 허야지	비대상	학사(공학)	서울대학교(서울)	주전공 : 법,경찰행정학부(공학계열_(건축))	서울대학교	논문 발표	8년	0	A 프로젝트	9개	정보처리기사
0049-000017	이씨학사유학	1지망 : 경력-IT-프론드엔드	1		test10@na.com	010-0000-0000	2남 1녀	무고	파이팅 허야지	비대상	학사(유학)	서울대학교	주전공 : 행정사회복지학부(공학계열_(교통·운송))	서울대학교	논문 발표	9년	0	A 프로젝트	10개	정보처리기사
0049-000018	이씨학시세학	1지망 : 경력-IT-백엔드	1	서징근력	test11@na.com	010-0000-0000	2남 1녀	무고	파이팅 허야지	미필	석사	-	주전공 : 건축학과(공학계열_(기계·금속))	-	논문 발표	10년	0	A 프로젝트	11개	정보처리기사
0049-000019	이씨석사송회	1지망 : 소업-회계-프론드엔드	1	거기이	test12@na.com	010-0000-0000	2남 1녀	무고	파이팅 허야지	비대상	석사(송토)	-	-	-	논문 발표	11년	0	A 프로젝트	12개	정보처리기사
0049-000020	이씨석사유학	1지망 : 경력-IT-백엔드	1		test13@na.com	010-0000-0000	2남 1녀	무고	파이팅 허야지	비대상	석사수료(유학)	서울대학교 NID융합기술대학원(산업디)(서울)	수선공 : 신호(공학계열_(교통·운송))	서울과학기술대학교 NID융합기술대학원(산업디)	논문 발표	12년	0	A 프로젝트	13개	정보처리기사
0049-000021	이씨식수재딱	1지망 : 경력-IT-프론드엔드	1	-	test14@na.com	010-0000-0000	2남 1녀	무고	파이팅 허야지	비대상	석사수료	-	-	-	논문 발표	13년	0	A 프로젝트	14개	정보처리기사
0049-000022	박씨고등문회	1지망 : 신입-IT-프론드엔드	1	-	test15@na.com	010-0000-0000	2남 1녀	무고	파이팅 허야지	비대상	고등(중퇴)	-	-	-	논문 발표	14년	0	A 프로젝트	15개	정보처리기사
0049-000023	박씨고등폴이	1지망 : 경력-IT-프론드엔드	1	-	test16@na.com	010-0000-0000	2남 1녀	무고	파이팅 허야지	비대상	고폴	-	-	-	논문 발표	15년	0	A 프로젝트	16개	정보처리기사

지원서 항목 기준으로 열 기반 조회

주니어 개발자의 성능 개선도전기

대용량 엑셀 다운로드 성능 개선 – 행 기반 조회에서 열 기반 조회로

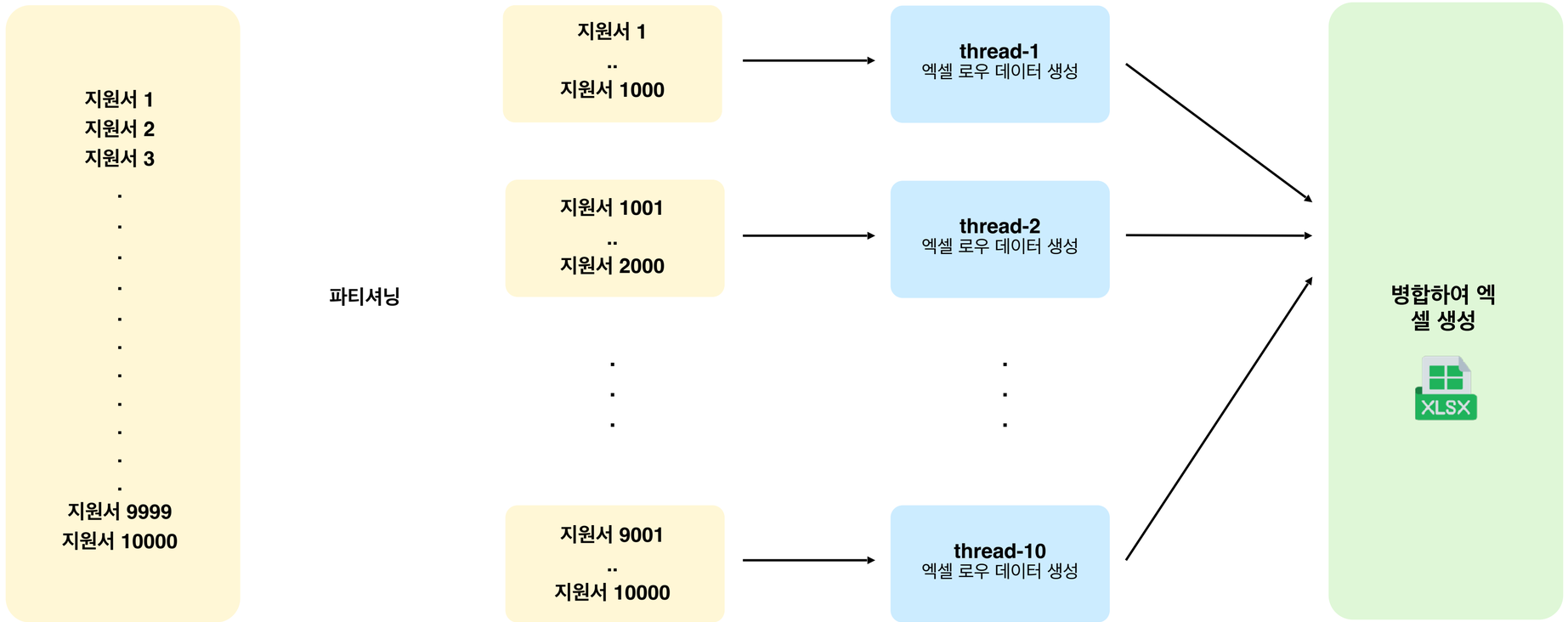
```
// 지원서 엑셀 로우 데이터 조회 로직
for (Integer resumeSn : resumeSnList) {
  Resume resume = findResume(resumeSn);
  // 지원서 항목 A 테이블 단건 조회
  ResumeItemA itemA = findResumeItemA(resumeSn);
  // 지원서 항목 B 테이블 리스트 조회
  List<ResumeItemB> itemBList = findResumeItemB(resumeSn);
  // 지원서 항목 C 테이블 리스트 조회
  List<ResumeItemC> itemCList = findResumeItemC(resumeSn);
  ...
  // 엑셀 로우 데이터 생성
}
```

```
// 지원서 리스트 조회
List<Resume> resumeList = findResumeListIn(resumeSnList);

// 지원서 항목 A 테이블 In 절로 조회 후 Map<지원서 번호, ItemA>로 맵 생성
Map<Integer, ItemA> aItemMap = findResumeItemAIn(resumeSnList)
                                    .stream()
                                    .collect(toMap(ItemA::resumeSn, Function.identity()));

// 지원서 항목 B 테이블 In 절로 조회 후 Map<지원서 번호, List<ItemB>>로 맵 생성
Map<Integer, List<ItemB>> bItemMap = findResumeItemBIn(resumeSnList)
                                        .stream()
                                        .collect(Collectors.groupingBy(ItemB::getResumeSn))

for (Resume resume : resumeList) {
    Integer resumeSn = resume.getResumeSn();
    ItemA itemA = aItemMap.get(resumeSn);
    List<ItemB> itemBList = bItemMap.get(resumeSn);

    // 엑셀 로우 데이터 생성
}
```

조회 쿼리 수 대폭 감소

주니어 개발자의 성능 개선도전기

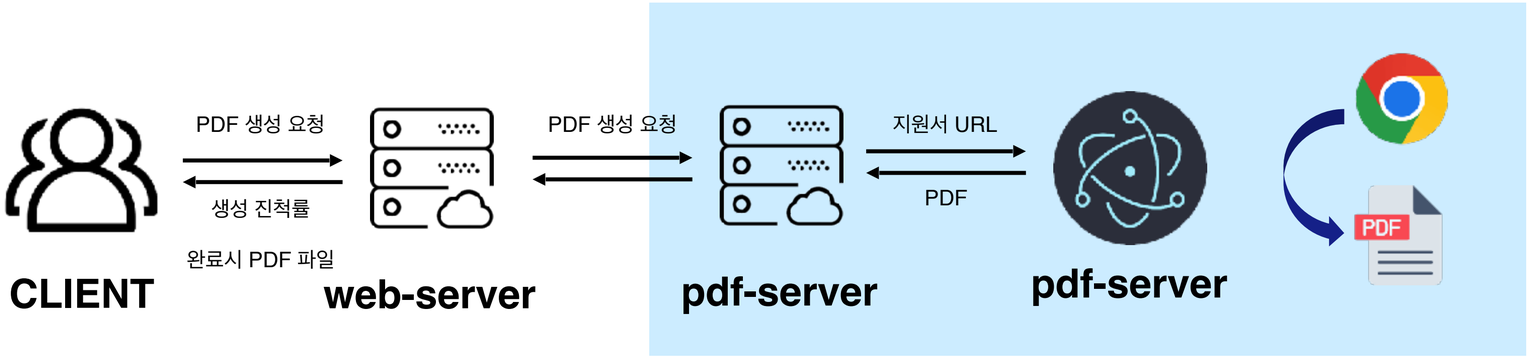
대용량 엑셀 다운로드 성능 개선 – 직렬에서 병렬로



CompletableFuture를 통한 병렬 처리로 성능 개선

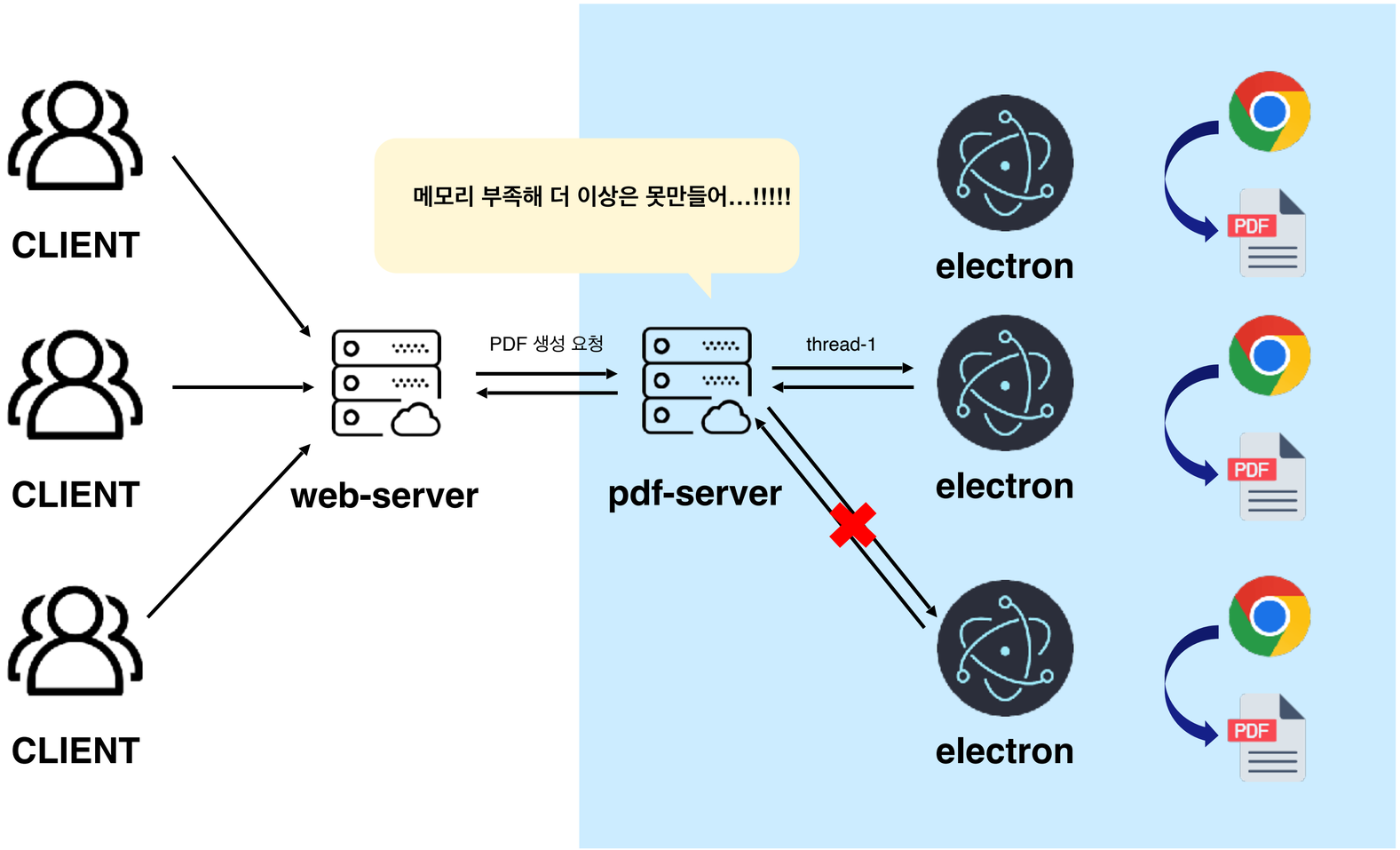
대용량 PDF 다운로드 기능 성능 개선 – 기존의 문제점

PDF 서버 내에서 많은 컴퓨팅 리소스를 사용하는 웹 스크래핑 방식으로 PDF를 생성



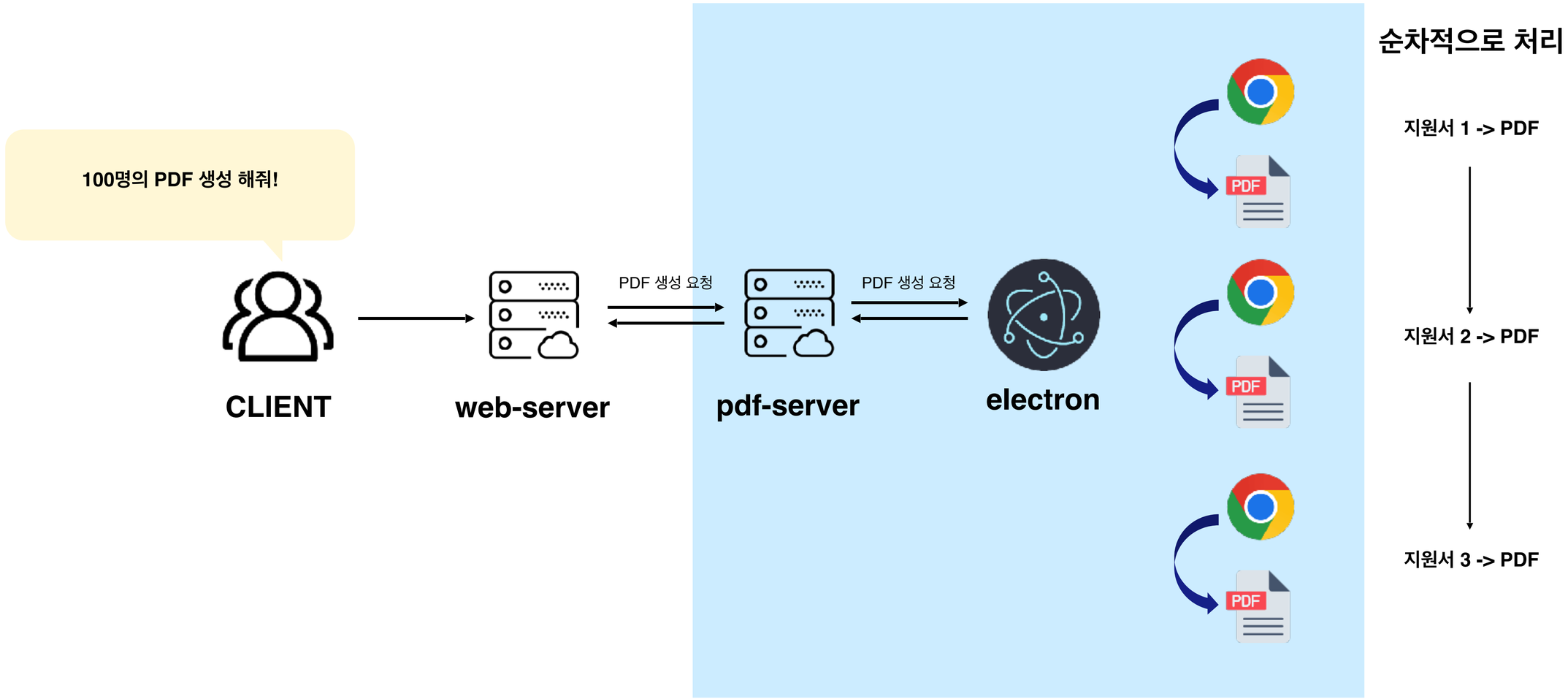
주니어 개발자의 성능 개선도전기

대용량 PDF 생성 기능 성능 개선 – 기존의 문제점



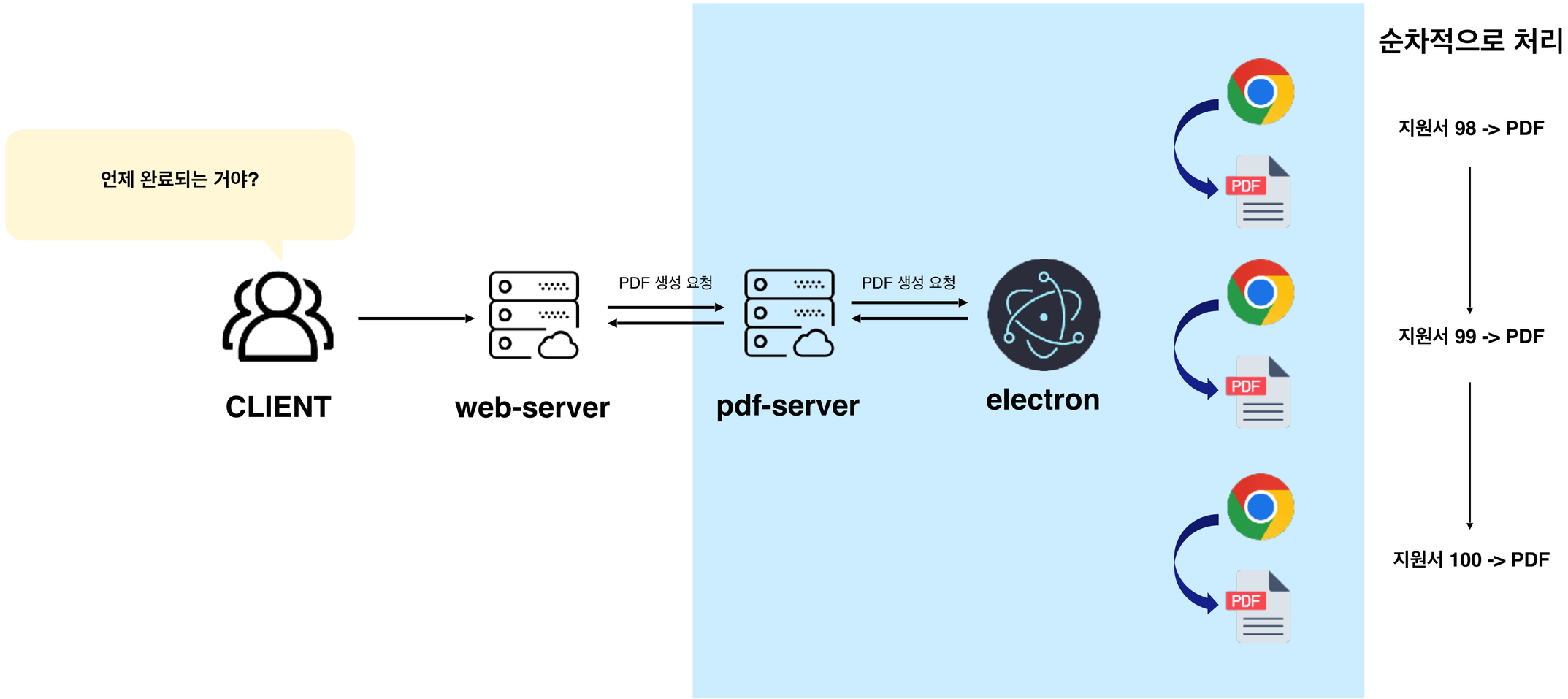
주니어 개발자의 성능 개선도전기

대용량 PDF 다운로드 기능 성능 개선 – 기존의 문제점



주니어 개발자의 성능 개선도전기

대용량 PDF 다운로드 기능 성능 개선 – 기존의 문제점



대용량 PDF 다운로드 기능 성능 개선 – 기존의 문제점

PDF 서버의 한정된 메모리와 CPU로 띄울 수 있는 크롬의 수는 한정적

동시에 여러 요청 시 PDF 생성 기능 안정성 ↓

대량의 PDF 생성 시 병렬 처리가 힘든 구조

항상 높은 스펙의 상시 서버를 유지함으로 서버 비용 낭비

대용량 PDF 다운로드 기능 성능 개선 – 기존의 문제점

PDF 서버의 한정된 메모리와 CPU로 띄울 수 있는 크롬의 수는 한정적

동시에 여러 요청 시 PDF 생성 기능 안정성 ↓

대량의 PDF 생성 시 병렬 처리가 힘든 구조

항상 높은 스펙의 상시 서버를 유지함으로 서버 비용 낭비

컴퓨팅 리소스와의 싸움

대용량 PDF 다운로드 성능 개선 – 아키텍처 변경을 통한 성능 개선

결국 대량의 PDF를 빠르게 병렬로 생성하려면…

빠르고 유연한 컴퓨팅 리소스 획득이 관건

대용량 PDF 다운로드 성능 개선 – 아키텍처 변경을 통한 성능 개선

결국 대량의 PDF를 빠르게 병렬로 생성하려면...

빠르고 유연한 컴퓨팅 리소스 획득이 관건

높은 확장성을 가진 서버리스 아키텍처 채택

대용량 PDF 다운로드 성능 개선 – 아키텍처 변경을 통한 성능 개선

AWS Lambda란?

AWS에서 제공하는 서버리스 컴퓨팅 서비스

높은 확장성과 컴퓨팅 자원(메모리, CPU)을 유연하게 획득 가능한 고가용성 컴퓨팅 인프라

특정 기간 또는 특정 주기로 코드를 실행시켜야 하는 경우 또는 트리거가 실행 될 때만 코드를 실행시키고 싶은 경우 유용

AWS Lambda를 선택한 이유

요청 수와 사용한 시간에 따라 비용을 청구하므로 서버 비용 절감

PDF 생성 요청이 있을 때만 잠깐 뜨고 사라짐

확장에 유연하여 람다를 병렬로 호출하여 여러 개의 컴퓨팅 리소스를 할당 받아 병렬로 PDF 생성 가능

대용량 PDF 다운로드 성능 개선 – 아키텍처 변경을 통한 성능 개선

Amazon Elastic File System(EFS)

AWS에서 제공하는 탄력적인 서버리스 파일 스토리지

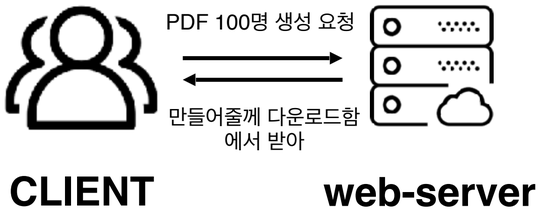
빠른 확장과 병렬 접근을 지원하여 ECS, EC2, 람다와 같은 여러 개의 컴퓨팅 인스턴스에서 공유된 스토리지 공간이 필요할 때 채택

AWS EFS를 선택한 이유

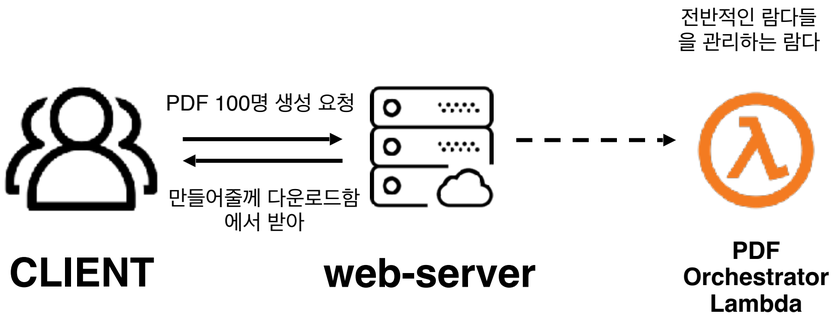
한 요청에 대해 여러 개의 Lambda 인스턴스에서 병렬로 생성된 PDF를 하나의 EFS 공유 저장소에 저장하기 하기 위함

주니어 개발자의 성능 개선도전기

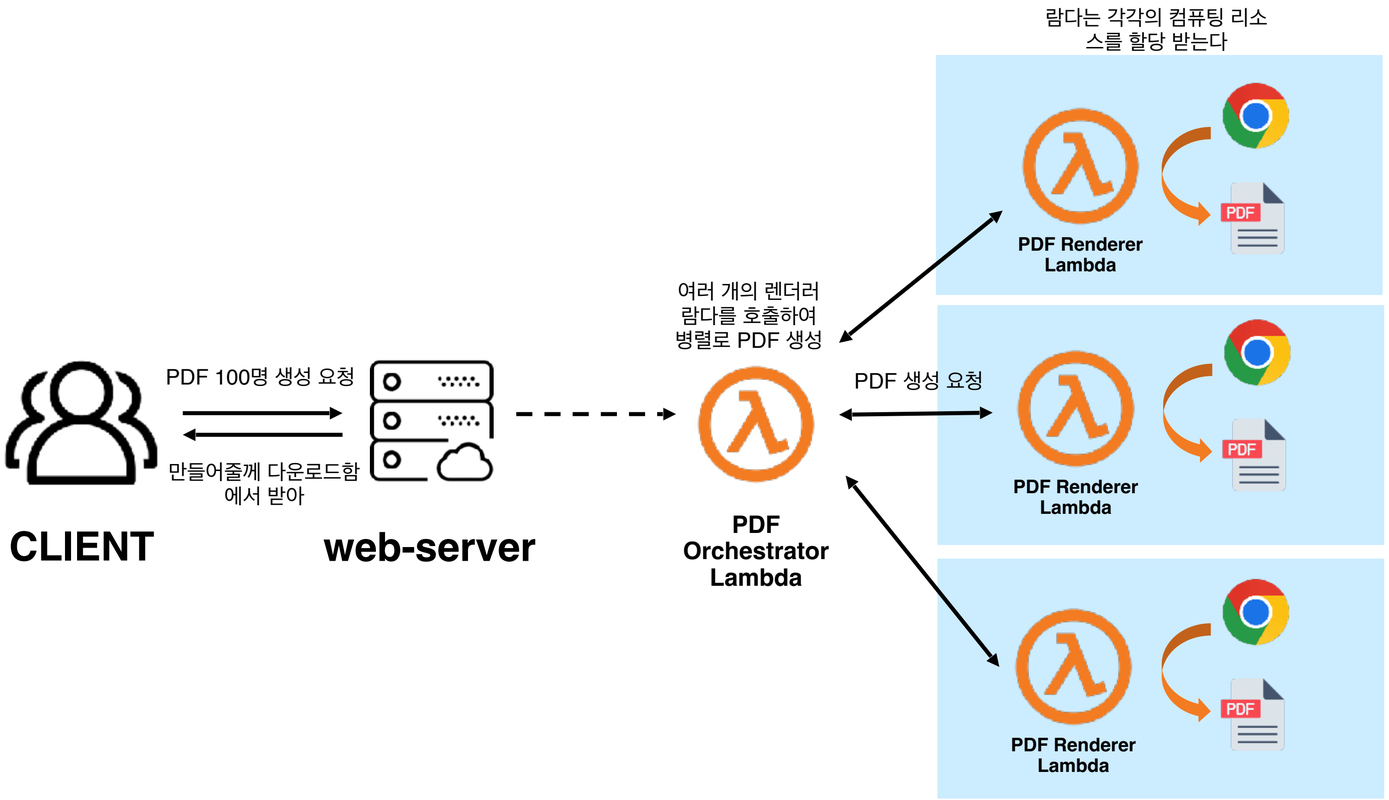
대용량 PDF 다운로드 성능 개선 – 아키텍처 변경을 통한 성능 개선



대용량 PDF 다운로드 성능 개선 – 아키텍처 변경을 통한 성능 개선

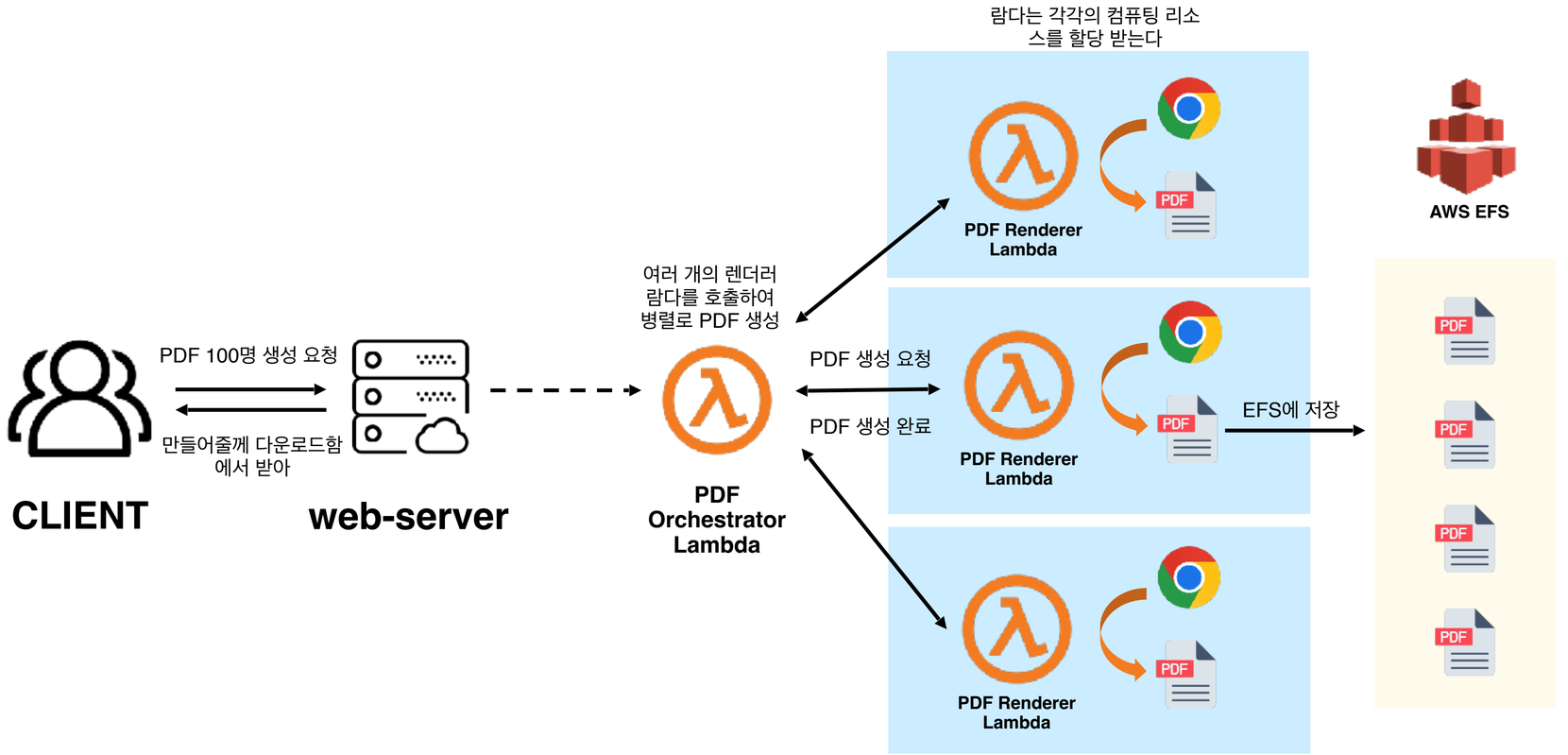


대용량 PDF 다운로드 성능 개선 – 아키텍처 변경을 통한 성능 개선



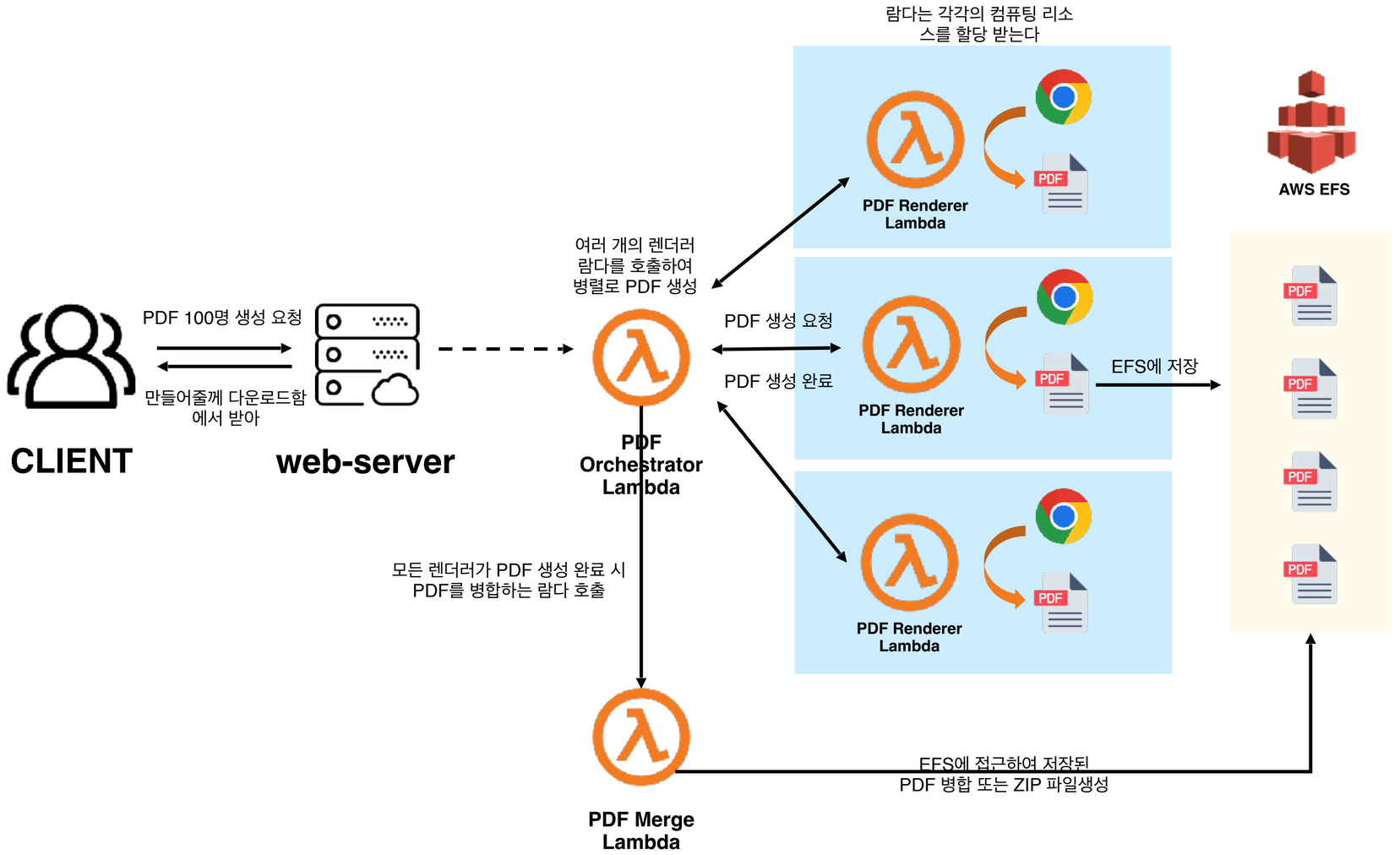
주니어 개발자의 성능 개선도전기

대용량 PDF 다운로드 성능 개선 – 아키텍처 변경을 통한 성능 개선



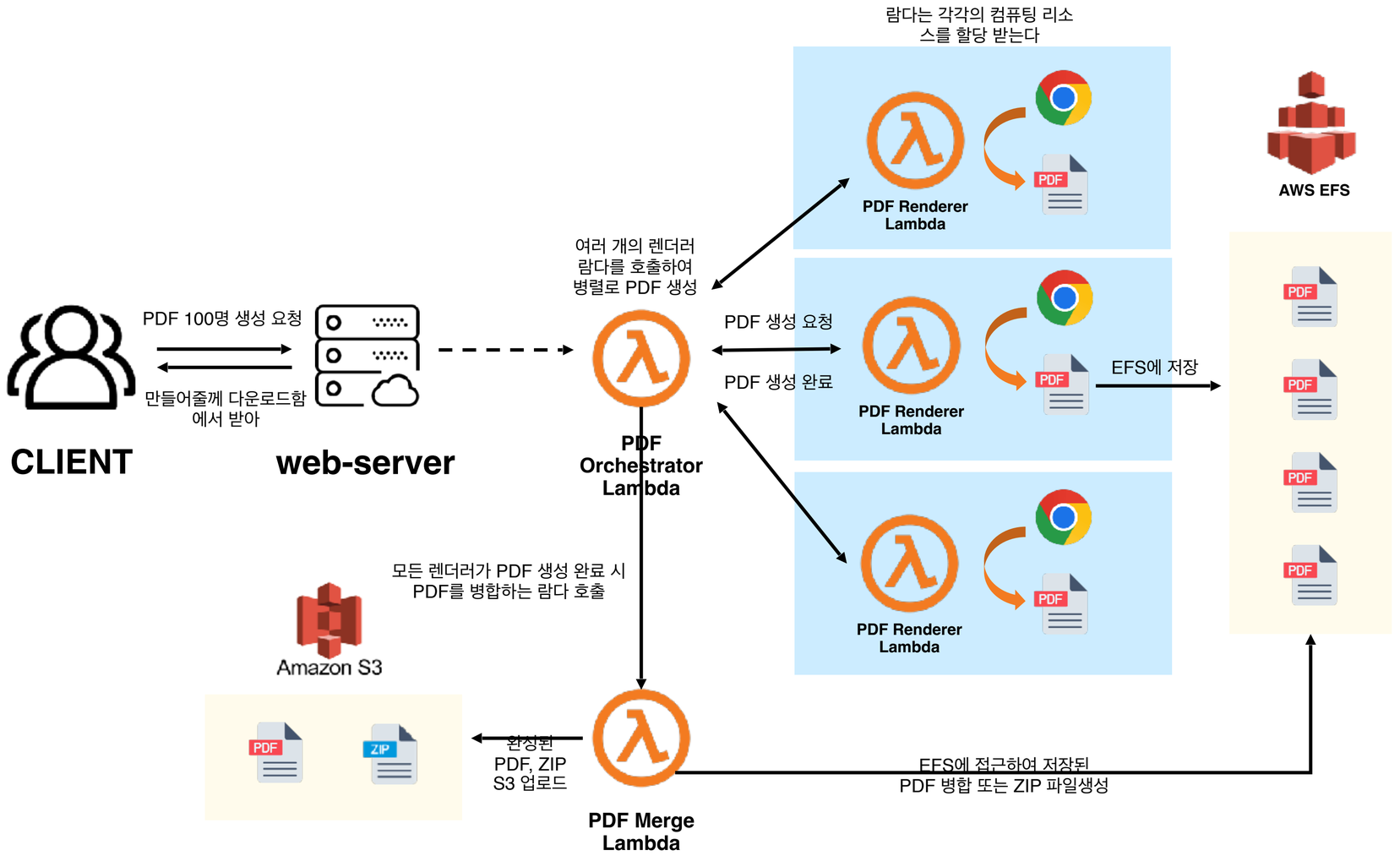
주니어 개발자의 성능 개선도전기

대용량 PDF 다운로드 성능 개선 – 아키텍처 변경을 통한 성능 개선



주니어 개발자의 성능 개선도전기

대용량 PDF 다운로드 성능 개선 – 아키텍처 변경을 통한 성능 개선



대용량 엑셀, PDF 다운로드 성능 개선 – 정리

시간이 오래 걸리거나 서버에 부하를 줄 수 있는 기능은 서버를 분리하고 비동기 방식을 택하는 것도 방법

멀티 스레드와 람다로 병렬 처리를 통해 성능 개선

컴퓨팅 리소스 부족으로 인한 성능 개선의 한계가 있다면 유연한 컴퓨팅 리소스를 획득할 수 있는 아키텍처 도입

마무리

마무리

성능 개선을 통해 사용자들의 소중한 시간을 절약시켜 주자

성능을 생각하는 개발자가 되자

E.O.D