하이러닝 성능 최적화 및 운영 안정화, 2024.01 ~ 2025.10 트러블슈팅

피크 시간대 수업 목록 쿼리를 여러 차례 고쳐 5초에서 팀 기준 1.3초 이내로 단축

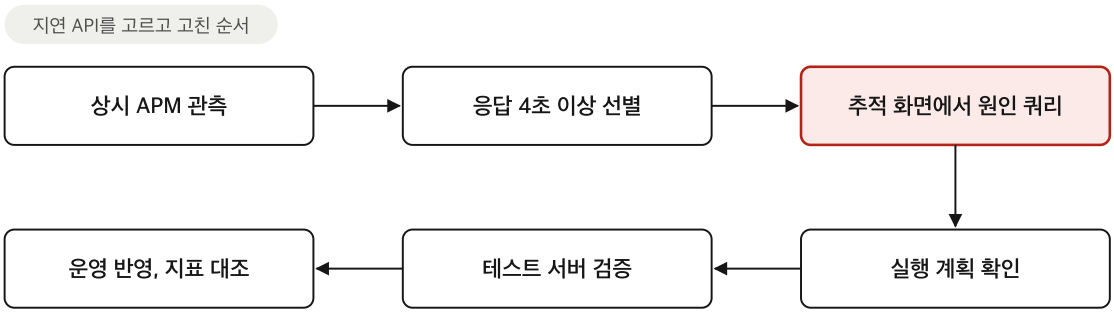
상시 APM과 실행 계획으로 수업 목록 쿼리의 원인을 하나씩 찾아, 쓰지 않는 조인 제거, 인덱스를 막던 형변환 제거, 건수 집계 분리, 인덱스 재설계를 여러 차례 나눠 반영

APM 실행 계획 복합 인덱스 점진 개선

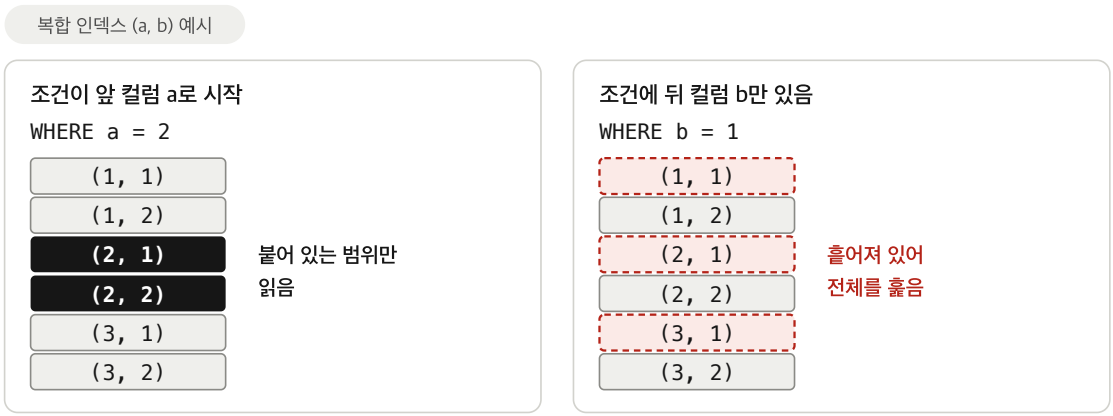
이슈 상황

- 2023년 9월 시범 운영 이후 1년 만에 사용 학교가 162개교에서 2,470개교로 늘었습니다.
- 오전 수업 시작 시간대(서비스 피크 TPS 3,000+)에 '나의 수업 목록' 조회 지연과 Primary DB 과부하가 반복돼, 수업 직전에 목록을 불러오지 못하는 일이 생겼습니다.

원인 분석



- 상시 켜 둔 APM에서 피크 시간대에 호출량이 많고 응답이 4초 이상인 API를 골랐습니다. 요청별 추적 화면에서 SQL과 호출 횟수를 보고 원인 쿼리를 좁힌 뒤, EXPLAIN ANALYZE로 실행 계획을 확인했습니다.
- 수업 목록 쿼리는 인덱스가 있었는데도 풀 스캔을 했습니다. 조회 조건이 복합 인덱스의 선행 컬럼과 맞지 않았기 때문입니다.



- 쿼리 하나는 CTE 하나와 서브쿼리 17개(화면 분기 포함)로 이루어져 있었습니다. 교사가 나의 수업 목록을 한 번 열면 테이블 11개를 JOIN 12개로 연결했습니다.

- 인덱스 말고도 쿼리 구조에 원인이 겹쳐 있었습니다.
  - 과목마다 차시(수업 회차) 정보를 LEFT JOIN해서 과목 한 행이 차시 수만큼 늘어났습니다. 그런데 바깥 쿼리는 차시 컬럼을 쓰지 않고, 늘어난 행을 `DISTINCT`로 다시 합쳤습니다.
  - 학년 이름 조인은 번호 컬럼을 `::TEXT`로 바꿔 비교해서, 그 컬럼의 인덱스를 쓸 수 없었습니다.
  - 마스킹 함수와 학년 이름 서브쿼리가 결과 행마다 실행됐습니다.
- 그래서 인덱스만 다시 만들어서는 해결되지 않았습니다.

최적화 전 쿼리 구조 보기

이름은 일반 이름으로 바꾸고, 문제가 된 부분만 남겼습니다.

```
WITH subject_cte AS (
  -- ② 늘어난 행을 다시 합침
  SELECT DISTINCT ...
  FROM subject s
    -- ① 과목 1행이 차시 수만큼 늘어남, 결과에는 쓰지 않음
    LEFT JOIN (차시 정보) c ON c.subject_seq = s.seq
    /* 학기, 담당 교사, 알림 집계 등 JOIN 생략 */
)
SELECT ...,
  -- ③ 결과 행마다 DB 함수 호출
  mask_function(a.teacher_name),
  -- ④ 행마다 서브쿼리, 컬럼 형변환으로 인덱스를 쓰지 못함
  (SELECT ... FROM school_year y WHERE y.seq::TEXT = ...)
FROM subject_cte a
```

개선

한 번에 고치지 않고 원인을 하나씩 나눠 여러 차례 반영했습니다. 저는 원인을 분석해 고칠 지점을 정하고, 변경을 검토해 반영했습니다. 바꾼 내용은 원인 종류에 따라 네 가지입니다.

원인	바꾼 것	왜 느렸는지
인덱스를 쓸 수 없는 조건	학년 번호 비교에서 컬럼의 형변환(`::TEXT`)을 없애고, 비교할 값 쪽을 정수 배열로 바꿈	컬럼에 형변환이나 함수가 걸리면 그 컬럼의 인덱스로 값을 찾을 수 없음
불필요한 조인	수업 목록 API가 함께 실행하는 탭 건수 쿼리(오늘, 신규, 알림)의 담당 교사 확인에서 회원 테이블 조인을 빼고, 담당 교사 테이블의 회원 번호로 바로 거름	회원 테이블을 거쳐 회원 ID로 거르느라, 담당 교사 테이블을 회원 번호로 바로 찾지 못함
인덱스를 쓸 수 없는 조건	바뀐 조건에 맞춰 복합 인덱스를 다시 설계하고 DB에 직접 반영	조회 조건이 복합 인덱스의 선행 컬럼과 맞지 않아 풀 스캔
결과에 쓰지 않는 계산	목록 쿼리와 건수 쿼리의 LEFT JOIN 서브쿼리(차시 정보, 영상 시작일 집계), 교사 목록 서브쿼리, 건수 쿼리의 SUM 제거	화면에 쓰지 않는 조인과 집계를 조회할 때마다 계산
반복되던 계산	과목마다 붙이던 학생 수 같은 건수 집계를 수업 복사 창을 열 때만 실행	목록을 열 때마다 과목별 건수 집계를 함께 실행
반복되던 계산	분류체계 값과 수업명 조건을 WITH 절에서 한 번 만들고 `IN (SELECT …)`로 비교	같은 조건 값을 비교할 때마다 다시 만듦
반복되던 계산	오늘 수업 목록에서 SQL이 행마다 부르던 마스킹 함수를 Java 처리로 옮김	조회한 행 수만큼 DB 함수 호출

조건을 바꾼 두 곳의 변경 전후 SQL 보기

테이블과 컬럼 이름은 일반 이름으로 바꿨습니다.

```
-- 변경 전
-- 1) 탭 건수 쿼리의 담당 교사 확인: 회원 테이블을 조인해 회원 ID로 거름
SELECT 1 FROM subject_teacher st
JOIN member m ON st.member_seq = m.member_seq
WHERE st.subject_seq = s.subject_seq AND m.member_id = :memberId AND st.del_yn = 'N'
-- 2) 학년 이름: 학년 번호 컬럼을 문자로 바꿔 비교
FROM UNNEST(STRING_TO_ARRAY(s.grade_seqs, '|')) AS g
JOIN school_year y ON y.grade_seq::TEXT = g

-- 변경 후
-- 1) 담당 교사 테이블에 있는 회원 번호로 바로 거름
SELECT 1 FROM subject_teacher st
WHERE st.subject_seq = s.subject_seq AND st.member_seq = :memberSeq AND st.del_yn = 'N'
-- 2) 쪼갠 값을 정수 배열로 바꾸고 컬럼은 그대로 비교
FROM UNNEST(STRING_TO_ARRAY(s.grade_seqs, '|')::int[]) AS g
JOIN school_year y ON y.grade_seq = g
```

- 변경마다 실행 계획으로 1.3초 기준(원래 있던 팀 개발 기준)을 확인했고, 새 기능을 붙일 때도 같은 기준을 먼저 확인했습니다. 성능이 걸린 변경이 있으면 운영과 최대한 같게 맞춘 테스트 서버에서 해당 API를 JMeter로 부하 테스트한 뒤 반영했습니다.
- 같은 방식으로 논술 평가 조회의 `과목 IN (서브쿼리)` 조건을 존재 여부 확인(`CASE WHEN EXISTS`)으로 바꿨습니다.

결과

- 수업 목록 쿼리 실행 시간을 5초에서 팀 기준 1.3초 이내로 줄였습니다(74% 이상 단축).
- 오전 수업 시작 직전에 수업 목록을 불러오지 못해 수업이 늦어지던 문제를 완화했습니다.

하이러닝 성능 최적화 및 운영 안정화, 2024.01 ~ 2025.10 트러블슈팅

반복 조회와 요청마다 하던 집계를 줄여 API 응답 지연 해소

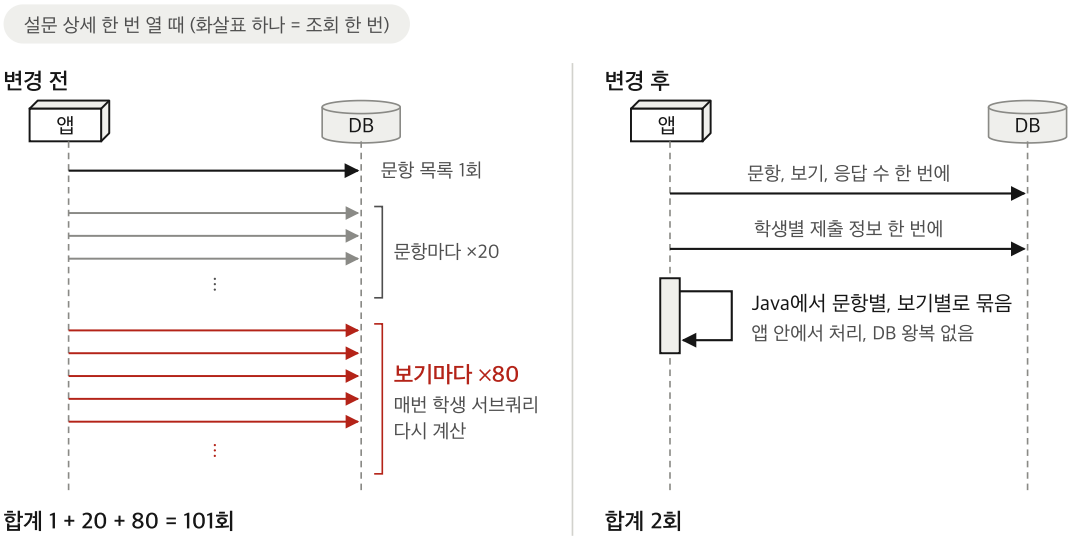
반복문 안에서 매퍼를 부르던 구조와 요청마다 원본을 집계하던 쿼리를 한 번에 읽는 구조로 바꿔, 교사용 설문 상세 응답을 4초 이상에서 1초대로 줄임

N+1 일괄 조회 월 단위 집계 배치 집계

이슈 상황

- 수업 목록 쿼리와 같은 방식으로 APM에서 응답 4초 이상인 API를 추적했습니다.
- 요청별 추적 화면의 SQL 호출 횟수를 보면, 한 번의 요청에서 같은 쿼리를 반복하거나 요청마다 원본을 집계하는 API가 있었습니다. 교사용 설문 상세의 반복 조회는 APM 호출 기록과 VoC로 찾았습니다.

원인 분석



- 교사용 설문 상세는 문항 목록을 읽은 뒤 문항마다 보기와 응답 수를, 보기마다 응답 학생을 다시 조회했습니다. N+1 조회가 문항과 보기 두 단계로 겹쳐, 20문항에 보기가 4개씩이면 1 + 20 + 80 = 101회입니다.
- 그중 가장 많은 응답 학생 조회는 실행할 때마다 학생 목록 서브쿼리와 프로필 이미지 서브쿼리를 다시 계산해, 한 번의 비용도 학생 수에 비례했습니다.

개선

- 문항과 보기를 조인해 보기별 응답 수를 `GROUP BY`로 한 번에 집계했습니다.
- 학생 목록은 CTE로 한 번만 만들고 제출 정보와 조인해 한 번에 읽은 뒤, Java에서 `groupingBy`로 문항별, 보기별로 묶었습니다. 조회는 문항 수와 관계없이 2회가 됐습니다.
- 문항 20개 이상, 학생 15명 이상인 데이터로 검증했고, 사용자와 QA가 정확성을 확인한 뒤 배포했습니다.

설문 상세의 변경 전후 코드 보기

메서드와 테이블 이름은 일반 이름으로 바꿨고, 조인 조건은 구조가 보이도록 단순화했습니다.

```
// 변경 전: 문항마다, 보기마다 매퍼 호출
List<Quest> quests = mapper.selectQuestList(activityId);                // 1회
for (Quest q : quests) {
    q.setAnswers(mapper.selectAnswerList(q.getId()));                   // 문항 수만큼
    for (Answer a : q.getAnswers()) {
        a.setStudents(mapper.selectAnsweredStudents(a.getId()));        // 보기 수만큼
    }
}

// 변경 후: 두 번 읽고 Java에서 묶음
List<QuestAnswerRow> answers = mapper.selectQuestAnswerList(activityId); // 1회
List<SubmitRow> submits = mapper.selectSubmitList(activityId);          // 1회
Map<Long, List<SubmitRow>> studentsByAnswer = submits.stream()
        .collect(Collectors.groupingBy(SubmitRow::getAnswerId));
```

```
-- 첫 번째 조회: 문항, 보기, 보기별 응답 수를 한 번에
SELECT q.quest_id, a.answer_id, COUNT(d.answer_id) AS submit_cnt
FROM quest q
JOIN answer a ON a.quest_id = q.quest_id
LEFT JOIN submit_detail d ON d.answer_id = a.answer_id
WHERE q.activity_id = :activityId
GROUP BY q.quest_id, a.answer_id

-- 두 번째 조회: 학생 목록은 CTE로 한 번만 만들고 제출 정보와 조인
WITH students AS ( /* 수업의 학생 목록 */ )
SELECT st.student_id, q.quest_id, d.answer_id, f.file_path
FROM students st
JOIN quest q ON q.activity_id = :activityId
LEFT JOIN submit s ON s.student_id = st.student_id AND s.activity_id = q.activity_id
LEFT JOIN submit_detail d ON d.submit_id = s.submit_id AND d.quest_id = q.quest_id
LEFT JOIN file f ON f.file_id = d.file_id
```

같은 방식으로 고친 다른 조회

대상	변경 전	변경 후	결과
학생 종합 리포트	학기 시작부터 한 달씩 집계 쿼리 세 개 반복	전체 기간을 월 단위로 묶어 한 번에 집계하고 달별로 나눔	조회 3M회 → 3회 (M: 개월 수)
과목별 학생 조회	과목마다 조회, 매번 프로필 이미지 CTE 재계산	과목 ID 목록을 `IN` 으로 한 번에 조회하고 과목별로 나눔	N회 → 1회 (N: 과목 수)
콘텐츠 목록	요청마다 미반영 조회 원본을 집계해 조인	배치가 반영한 조회수 컬럼만 읽음	요청마다 하던 집계 제거

- 과목 ID 목록이 비어 있으면 조회하지 않게 했습니다. 빈 목록이 `IN` 조건에 들어가면 SQL 문법 오류가 납니다.
- 콘텐츠 목록은 조회수가 배치 주기만큼 늦게 반영되는 대신, 요청마다 하던 집계가 없어졌습니다.

다른 조회의 변경 전후 SQL 보기

```sql
-- 콘텐츠 목록, 변경 전: 요청마다 미반영 조회 원본을 집계해 더함
SELECT c.content_id, c.title, c.hits + COALESCE(h.hits, 0) AS hits
FROM content c
LEFT JOIN (
    SELECT content_id, COUNT(*) AS hits
    FROM content_hit_raw
    WHERE apply_yn = 'N'
    GROUP BY content_id
) h ON c.content_id::text = h.content_id

-- 콘텐츠 목록, 변경 후: 집계 배치가 반영한 조회수만 읽음
SELECT c.content_id, c.title, c.hits
FROM content c

-- 학생 종합 리포트, 변경 후(세 쿼리 중 하나를 단순화): 기간 전체를 월 단위로 묶어 한 번에 집계
SELECT TO_CHAR(reg_dt, 'YYYYMM') AS ym, student_id, /* 집계 값 */
FROM study_log
WHERE reg_dt BETWEEN :semesterStart AND :today
  AND student_id IN (:studentIds)
GROUP BY TO_CHAR(reg_dt, 'YYYYMM'), student_id
```

결과

- 교사용 설문 상세 API 응답 시간을 4초 이상에서 운영 기준 1초대로 줄였습니다.
- 리포트와 과목별 학생 조회는 기간이나 과목 수가 늘어도 조회 횟수가 늘지 않게 됐습니다.

하이러닝 성능 최적화 및 운영 안정화, 2024.01 ~ 2025.10 트러블슈팅

끝나지 않던 수동 트랜잭션 경로를 닫고 readOnly 적용 단위를 API 호출 경로로 정함

선언적 트랜잭션 대상에서 빠진 나의 수업 메뉴 수동 트랜잭션의 종료 경로에 commit과 rollback을 넣어 커넥션 풀 고갈 위험을 줄이고, Replica 라우팅을 위한 readOnly 적용 기준을 API 호출 경로 전체로 정함

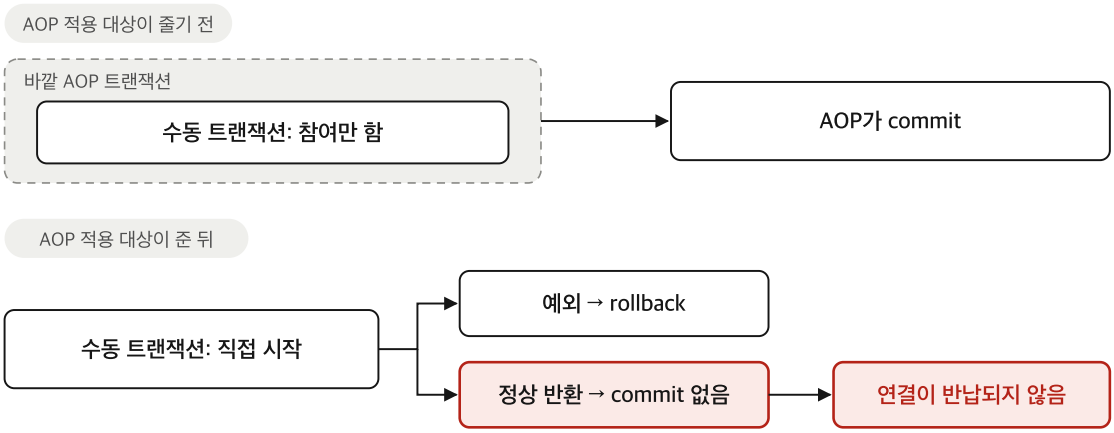
트랜잭션 전파 커넥션 풀 readOnly Read/Write 분리

이슈 상황

- 포털은 공통 AOP 설정으로 API와 서비스 클래스의 메서드에 트랜잭션을 걸었습니다. 일부 API는 이와 별도로 코드에서 트랜잭션을 직접 열었습니다(수동 트랜잭션).
- AOP 트랜잭션의 적용 대상이 줄면서, 일부 수동 트랜잭션은 참여할 바깥 트랜잭션 없이 실행돼 트랜잭션을 직접 시작하고 끝내야 하는 쪽이 됐습니다.
- 읽기와 쓰기 DB 연결을 나눈 구조에서 조회 부하를 Replica로 넘기려면, 어떤 트랜잭션을 readOnly로 둘지 정해야 했습니다.

원인 분석

끝나지 않던 수동 트랜잭션



- 수동 트랜잭션은 `PlatformTransactionManager.getTransaction()`에 전파 속성 `REQUIRED`만 지정해 열었습니다.
- 바깥 AOP 트랜잭션이 있을 때는 참여만 했고 commit은 AOP가 해서, 코드에 commit이 없어도 드러나지 않았습니다. AOP 적용 대상이 준 뒤에는 정상 반환 경로와 처리하지 않는 요청 분기에서 트랜잭션이 끝나지 않았습니다.
- Spring은 트랜잭션이 끝날 때까지 DB 연결을 요청 스레드에 묶어 둡니다. 트랜잭션이 끝나지 않으면 연결이 커넥션 풀로 돌아가지 않습니다.
- 당시 커넥션 풀 설정으로는 반납되지 않은 연결이 쌓이면 이후 요청이 끝없이 기다릴 수 있었습니다.

커넥션 풀 설정 (DBCP 1.x)	값	뜻
`maxActive`	50	동시에 빌려줄 수 있는 연결 수
`maxWait`	-1	연결이 없을 때 기다리는 시간. -1은 제한 없음
`removeAbandoned`	설정 없음	오래 반납되지 않은 연결을 풀이 회수하지 않음

메서드 단위 readOnly의 문제

- 프로젝트는 `AbstractRoutingDataSource`로 현재 트랜잭션이 readOnly이면 Replica, 아니면 Primary 연결을 골랐습니다. 실제 연결은 `LazyConnectionDataSourceProxy`가 첫 쿼리 때까지 미뤄, 트랜잭션 속성이 정해진 뒤에 고르게 했습니다.
- readOnly를 메서드 단위로 붙이면 문제가 생길 수 있었습니다. readOnly 트랜잭션 안에서 기본 전파(`REQUIRED`)로 쓰기 메서드를 부르면, 쓰기 메서드는 바깥 트랜잭션에 참여해 Replica 연결을 그대로 씁니다. 그러면 쓰기가 실패하거나, 실패하지 않더라도 복제 지연 때문에 방금 쓴 데이터가 보이지 않을 수 있습니다.

readOnly 설정 방식 보기

```xml
<!-- 메서드 이름으로 readOnly를 지정하는 방식 (예시) -->
<tx:advice id="txAdvice">
  <tx:attributes>
    <tx:method name="selectXxxList" read-only="true"/>
    <tx:method name="*" read-only="false" rollback-for="Exception"/>
  </tx:attributes>
</tx:advice>
```

- 수동 트랜잭션은 readOnly를 지정하지 않아, 조회만 하는 상세 API도 Primary 연결을 썼습니다.

개선

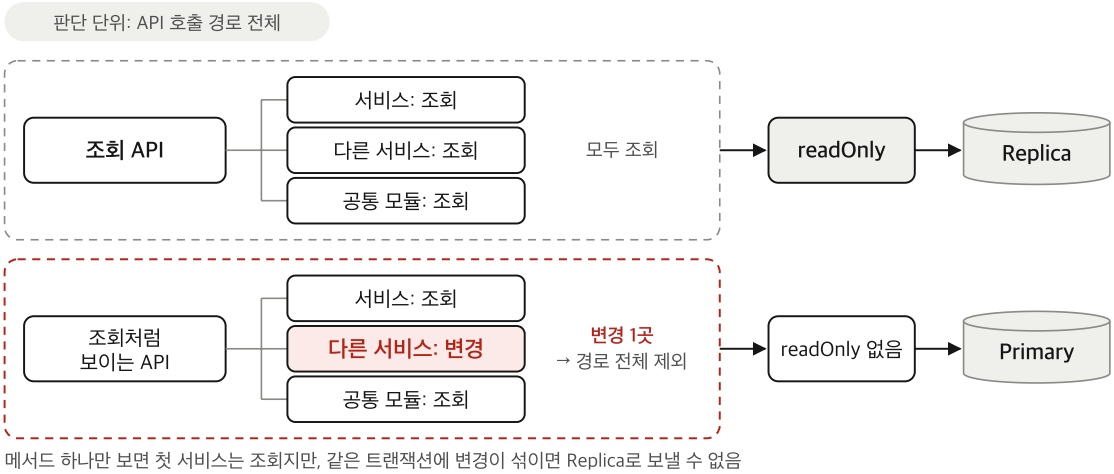
수동 트랜잭션의 종료 경로 닫기

수동 트랜잭션 코드 모양 보기 (commit과 rollback 추가 위치)

```java
// 수동 트랜잭션 (이름은 일반화)
DefaultTransactionDefinition def = new DefaultTransactionDefinition();
def.setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED); // readOnly, timeout 지정 없음
TransactionStatus status = transactionManager.getTransaction(def);
try {
    if (!isSupported(request)) {
        transactionManager.rollback(status);    // 추가
        return badRequest();
    }
    Result result = process(request);
    transactionManager.commit(status);          // 추가
    return ok(result);
} catch (Exception e) {
    // 공통 예외 처리기가 아직 끝나지 않은 트랜잭션을 rollback (원래 있던 경로)
    throw wrap(e, status);
}
```

- 정상 반환 직전에는 commit을, 처리하지 않는 요청 분기에는 rollback을 넣었습니다. 대상은 수업 활동(등록, 상세 조회, 수정), 알림장 댓글 등록, 과목 학생 삭제입니다.
- 수정 뒤 해당 API를 호출하고, DB 로그에서 트랜잭션이 commit 또는 rollback으로 끝나는지 확인했습니다.

readOnly 적용 단위를 API 호출 경로로



- 쓰기가 섞인 트랜잭션이 Replica로 가지 않도록, 판단 단위를 메서드가 아니라 API와 그 아래 호출 경로 전체로 정했습니다. 호출 경로 전체가 조회인 API에만 readOnly를 적용하는 기준을 세웠습니다.

결과

- 나의 수업 메뉴(수업 활동, 알림장 댓글, 과목 학생 삭제)의 수동 트랜잭션에서 commit이 빠진 경로를 닫아 커넥션 풀 고갈 위험을 줄였습니다.
- 데이터 변경이 섞인 트랜잭션이 Replica로 가지 않는 readOnly 적용 기준을 세워, 조회 전용 API의 부하를 Replica로 넘길 수 있게 했습니다.
- 이 기준으로 조회 API 28개의 부하를 Replica로 분산했습니다.

EBS 외부 API 연동 및 대용량 데이터 동기화 시스템 구축, 2023.10 ~ 2023.12 설계와 구현

단계를 나누고 원본을 보존해 외부 장애에도 재처리 가능한 동기화 구조 설계

외부 호출, 검증, 내부 반영을 단계별 트랜잭션으로 나누고(3단계는 1만 건 묶음마다 커밋) 외부 응답 원본을 남겨, 실패한 단계만 외부를 다시 부르지 않고 다시 실행하도록 설계

단계별 트랜잭션 원본 보존 정합성 검증 Upsert

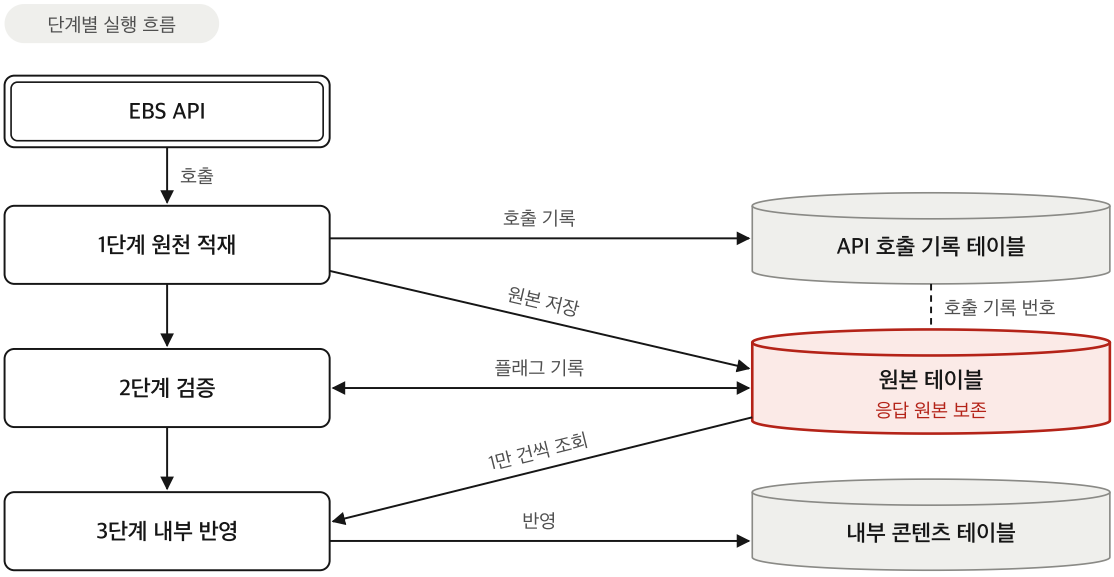
문제와 배경

- 교육과정이 바뀌면 사용자가 이전 커리큘럼 기반 콘텐츠를 보게 되는 문제가 있었습니다.
- 별도 배치 서버와 메시지 브로커 없이, 외부 콘텐츠 10만+ 건을 다시 처리할 수 있게 반영하는 파이프라인이 필요했습니다.
- EBS는 콘텐츠마다 생성, 수정, 삭제 지시를 API로 내려 줬습니다. 연동은 EBS가 제공한 API 명세서를 따랐습니다.

결과

- 외부 호출, 검증, 내부 반영을 단계별 트랜잭션으로 나눠(3단계는 1만 건 묶음마다 커밋), 실패한 단계만 다시 실행하고 2~3단계는 외부 재호출 없이 보존한 원본으로 재처리할 수 있게 했습니다.
- 교육과정 개편 2회와 EBS API 버전 변경 1회에도 같은 동기화 구조로 외부 콘텐츠 10만+ 건을 반영했습니다. 그 결과 현행 콘텐츠를 제공하고 이전 커리큘럼 노출 위험을 줄였습니다.
- 정합성 검증으로 원천 데이터 오류를 검출해 EBS 측 원본 보정 3~4회로 연결했습니다.

설계와 구현



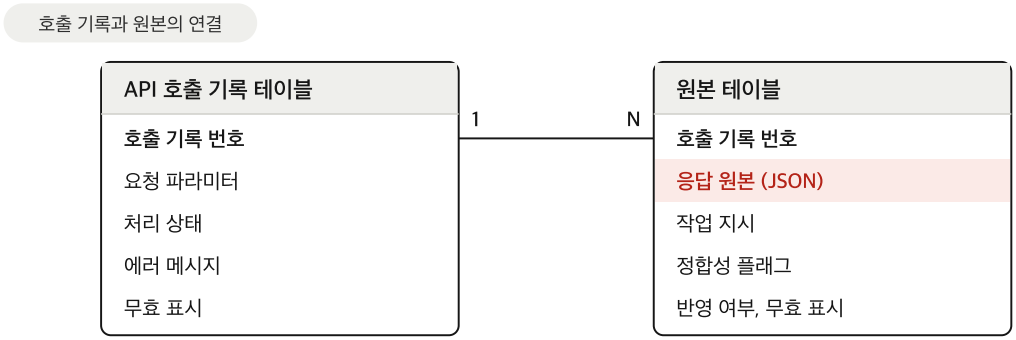
단계별 트랜잭션 분리

- 공통 AOP 설정이 서비스 메서드마다 `REQUIRED` 트랜잭션을 걸었습니다. 컨트롤러에는 트랜잭션이 없고, 단계별 실행 API가 서비스 메서드를 각각 호출하도록 나눴습니다. 그래서 단계마다 트랜잭션이 따로 커밋되고, 뒤 단계가 실패해도 앞 단계의 원본 적재와 검증 결과는 남습니다.

단계	커밋 단위	실패한 뒤 같은 파라미터로 다시 실행하면
1단계 원천 적재	실행 API 호출마다	정상 호출만 있으면 다시 부르지 않고, 오류가 섞이면 그 범위를 모두 다시 받음
2단계 검증	실행 API 호출마다	보존한 원본으로 다시 검증. EBS를 다시 부르지 않음
3단계 내부 반영	1만 건 묶음마다	반영 여부가 N인 행만 처리. EBS를 다시 부르지 않음

- 묶음 안의 오류는 종류에 따라 결과가 달랐습니다. Java 쪽 오류(JSON 변환 등)는 오류 전 행까지 커밋되고 오류 행만 표시됩니다(코드 기준). DB 오류는 PostgreSQL이 이후 SQL을 모두 거부해 묶음 전체가 오류 표시와 함께 롤백됩니다(재현 환경에서 확인).
- 청크별 진행 상태 추적 대신 실패한 실행 API를 같은 파라미터로 다시 실행해, 추적 구조를 단순하게 두었습니다. 다시 실행하는 범위가 넓어지는 비용은 감수했습니다.

원본 보존과 1단계 재호출 판단



- EBS API를 한 번 호출할 때마다 호출 기록 한 행을 남기고, 그 호출로 받은 원본 행을 호출 기록 번호로 연결했습니다.
- 호출 기록에는 요청 파라미터(대상 테이블, 작업 지시, 날짜), 처리 상태, 오류 메시지를 남겼습니다. 응답 원본은 원본 테이블에 JSON 그대로 보존했습니다. 그래서 2~3단계가 실패해도 EBS를 다시 부르지 않고 다시 처리합니다.
- 1단계를 다시 실행하면 EBS를 부르기 전에 같은 파라미터(대상 테이블, 작업 지시, 날짜)의 이전 호출 기록을 조회합니다.
  - 기록이 없으면 호출해 적재합니다.
  - 오류가 난 호출이 섞여 있으면 같은 파라미터의 호출 기록과 원본 행을 모두 무효로 표시한 뒤 범위 전체를 다시 호출합니다. 데이터 자체의 오류는 EBS가 해당 API에서 보정해 두었기 때문에, 같은 파라미터로 그대로 다시 호출했습니다.
  - 정상 호출만 있으면 이미 등록된 파라미터로 보고 다시 부르지 않습니다.

2단계 검증 규칙

작업 지시마다 내부 테이블을 기본 키로 조회해 정합성 플래그를 정했습니다.

내부 데이터 상태	생성 지시	수정 또는 삭제 지시
있음	작업 지시를 수정으로 바꾸고 통과	통과
없음	통과	보류
있지만 삭제 표시됨	보류	보류

- 이미 있는 키로 온 생성 지시는 실패로 처리하지 않고 수정 지시로 바꿔, 같은 생성 지시가 다시 와도 결과가 같게 했습니다. 원본 JSON은 그대로 두고 작업 행의 지시 컬럼만 바꿨습니다.
- 검증에 실패한 건도 원본과 함께 보관해 보정이나 내부 수정으로 이어지게 했습니다.

3단계 반영 방식

보존한 원본 JSON을 Jackson으로 VO로 바꾼 뒤, 작업 지시와 검증 결과에 따라 반영 구문을 나눴습니다.

작업 지시	통과	보류
생성	`INSERT … ON CONFLICT … DO UPDATE` (Upsert)	통과와 같이 삭제 표시를 푼 상태로 Upsert
수정	`UPDATE`	대상이 없거나 삭제 표시된 행이므로 삭제 표시를 푼 상태로 Upsert
삭제	행을 지우지 않고 삭제 표시만 바꿈 (논리 삭제)	삭제 표시된 행으로 Upsert

- Upsert와 UPDATE에는 값이 달라졌는지 비교하는 조건이 없습니다. 그래서 같은 범위를 다시 반영하면 값이 같은 행도 다시 쓰이고 동기화 시각이 갱신됩니다. 결과 데이터는 같지만, 다시 반영한 흔적은 남는 구조입니다.

오픈 후 판별 쿼리로 검증

- 오픈 후 판별 쿼리로 EBS 제공 건수, 기존 내부 건수, 실제 연동 건수를 대조했고, 교육과정별 문항 수와 단원 수 일치를 통과 기준으로 삼았습니다.
- 불일치는 바로 EBS에 요청하지 않고 내부 원인부터 확인했습니다. 원본 오류면 EBS에 보정을 요청했고, 아니면 연동 뒤 만든 뷰의 쿼리나 내부 로직을 고쳤습니다.

EBS 외부 API 연동 및 대용량 데이터 동기화 시스템 구축, 2023.10 ~ 2023.12 트러블슈팅

대량 반영 중 메모리 부족과 반영 누락을 원인부터 찾아 해소

10만+ 건 반영 중 나던 Heap OOM과 일부 원본의 반영 누락을 AOP 로깅 범위 조정, 청크 처리, 조회 방식 수정으로 해소

청크 처리 AOP 로깅 범위 페이지 조회 방식

이슈 상황

- 3단계(내부 반영)에서 외부 콘텐츠 10만+ 건을 처리하다가 Heap OOM으로 배치가 멈췄습니다.
- 메모리 문제를 고친 뒤에도, 배치는 정상 종료됐는데 일부 원본이 반영되지 않고 남는 증상이 있었습니다.

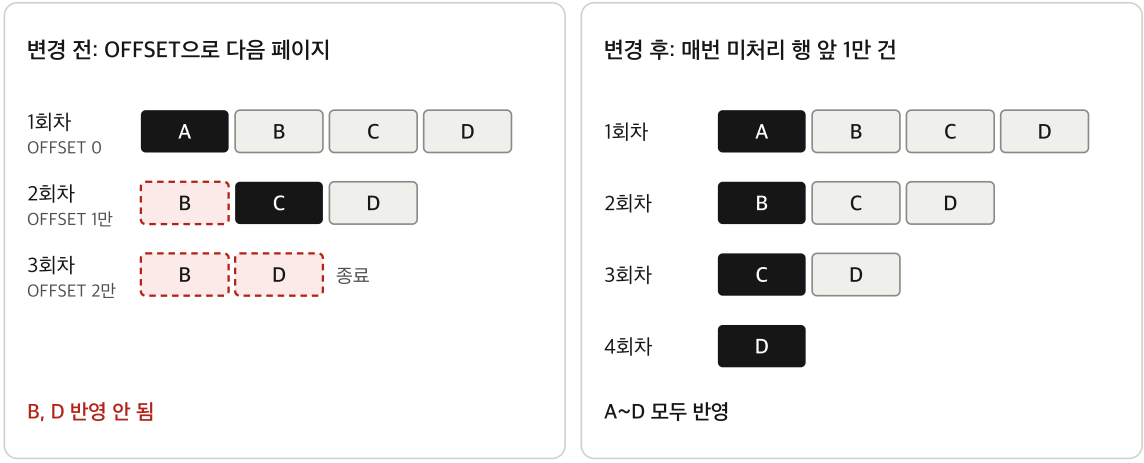
원인 분석

메모리 부족

- 서버 로그와 코드로 원인을 확인했습니다.
- 공통 AOP 로깅이 메서드 인자를 문자열 하나로 이어 붙였고, 출력하지 않는 로그 레벨에서도 문자열은 먼저 만들어졌습니다.
- 10만+ 건 목록을 받는 반영 메서드에서 이 문자열이 힙을 채웠습니다.

반영 누락

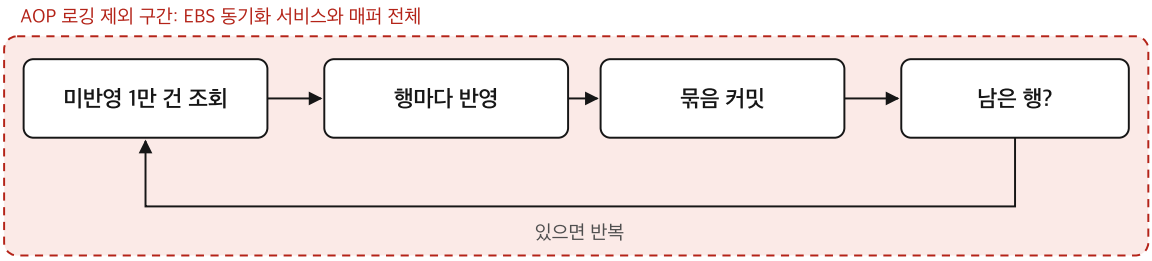
- 반영이 끝난 원본은 처리 상태가 바뀌어 다음 조회 조건에서 빠집니다.
- 그런데 다음 페이지를 OFFSET(앞에서 건너뛸 행 수)으로 찾으면, 남은 행이 앞으로 당겨진 만큼 건너뛰는 행이 생겼습니다.



원본 4만 건을 1만 건씩 A~D로 나눈 예. 검은 상자는 읽은 범위, 빨간 점선 상자는 건너뛴 범위입니다.

개선

문제	변경 전	변경 후
메모리 부족	반영 메서드도 공통 AOP 로깅 대상	EBS 동기화 서비스와 매퍼 전체를 공통 AOP 로깅 대상에서 제외
메모리 부족	원본 전체를 한 번에 읽어 반영	원본을 1만 건씩 나눠 읽고 묶음마다 커밋
반영 누락	처리 상태 조건과 OFFSET으로 다음 페이지 조회	OFFSET 없이 반복마다 “아직 처리하지 않은 행” 조건으로 앞 1만 건을 다시 읽음



결과

- 10만+ 건 반영 중 나던 Heap OOM을 해소하고, 초당 2,000건 이상 처리를 확인했습니다.
- 1만 건마다 커밋하고 반영하지 않은 행만 처리하므로, 중간에 실패해도 다시 실행하면 남은 행만 처리합니다.
- 같은 문제를 운영과 최대한 비슷한 환경과 설정에서 재현하고 저장 방식을 비교 측정한 내용은 재현과 비교 측정에 정리했습니다.

EBS 외부 API 연동 및 대용량 데이터 동기화 시스템 구축, 2023.10 ~ 2023.12 재현과 측정

메모리 부족과 반영 누락을 운영과 비슷한 환경에서 재현하고 저장 방식 비교 측정

운영과 비슷한 환경에서 수정 전후를 재현하고, 병목인 행 단위 저장을 묶으면서 실패 행만 골라내는 재설계안을 구현해 측정(합성 데이터)

재현 비교 측정 JDBC batch 합성 데이터

메모리 부족과 반영 누락 사례에서 다룬 문제를 운영과 최대한 비슷한 환경과 설정에서 다시 일으켜 측정했습니다. 이 페이지의 수치는 모두 재현 환경에서 합성 데이터로 잰 값이며, 실무에서 측정한 값이 아닙니다.

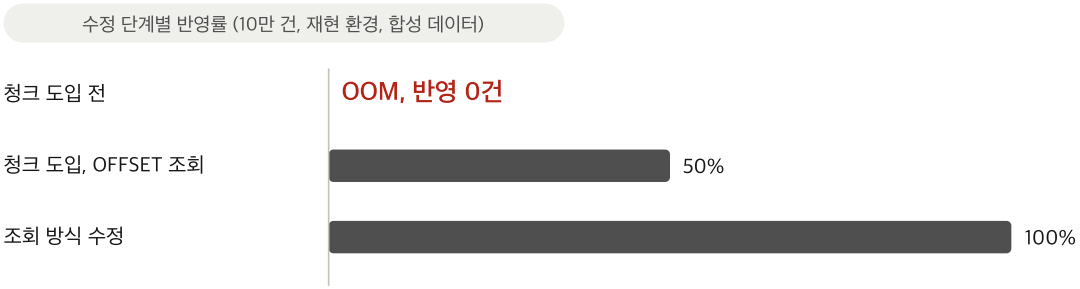
재현 환경

Java 8과 PostgreSQL로 운영과 같게 맞추고, 수정 단계별 커밋 코드를 고치지 않고 그대로 빌드해 같은 조건에서 실행했습니다. 메모리, 처리 시간, 오류 행 측정은 조건마다 3~5회 반복했고, 로그 크기 측정, 실패 때 운영 코드의 동작 비교, 재처리는 1회씩 실행했습니다.

운영과 재현 환경 비교 보기

항목	운영	재현 환경
Java	Java 8	Java 8
DB	PostgreSQL	PostgreSQL 15 (로컬)
데이터	실제 외부 콘텐츠	합성 데이터 (새로 만드는 행만 있고 수정과 삭제는 없음, 영문 문자열)
힙 크기	확인하지 못함	256MB, 512MB, 1GB로 나눠 측정

재현 결과

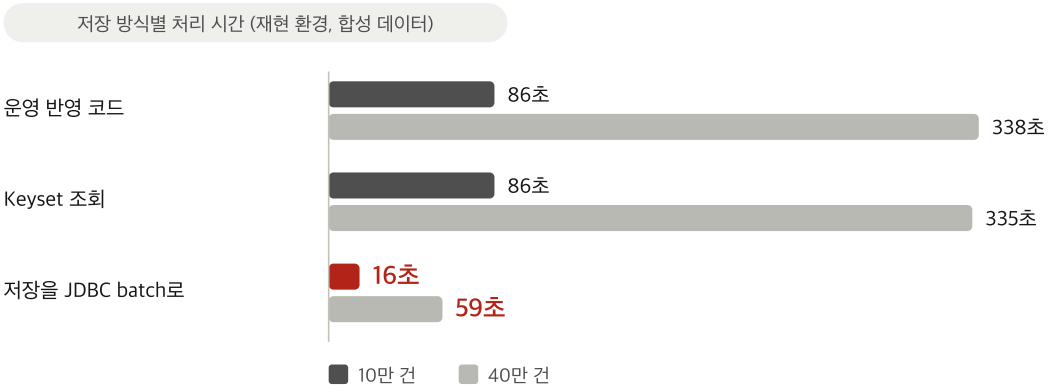


- 청크 도입 전에는 힙 256MB는 1만 건부터, 512MB는 2만 건부터, 1GB는 5만 건부터 OOM이 났습니다(조건마다 3회 같은 결과).
- OOM은 AOP 로깅이 목록 인자를 문자열로 만드는 위치에서 났고, 출력하지 않는 로그 레벨에서도 같았습니다. 청크만 도입한 코드에서 로그 레벨을 info로 켜고 10만 건을 반영하면 로그가 572MB 쌓였습니다.

- 운영 반영 코드는 힙 256MB로 10만, 20만, 40만 건을 모두 반영했고, Full GC 직후 힙 사용량은 건수와 관계없이 40~45MB였습니다. 청크 분할을 없애면 20만 건, 힙 512MB에서 다시 OOM이 났습니다.

대안 비교 측정

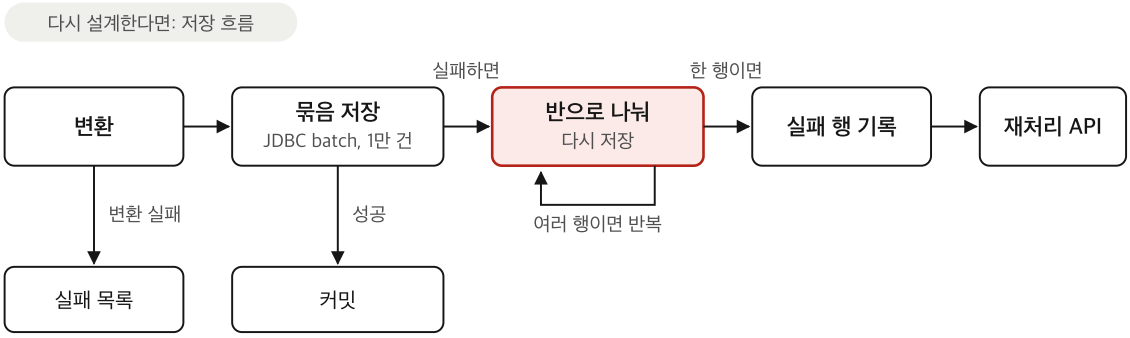
운영 반영 코드를 기준으로 조회 방식과 저장 방식을 하나씩 바꿔 측정했습니다.



- 처리 시간 대부분은 행마다 SQL 세 개를 DB와 한 번씩 주고받는 저장 단계였습니다. 그래서 Keyset은 차이가 없었고, JDBC batch는 10만 건에서 5.4배, 40만 건에서 5.7배 빨랐습니다. 세 방식 모두 반영 내용은 같았습니다.
- 다만 묶어서 보내면 어느 행에서 실패했는지가 흐려집니다. 당시 배치는 사용자 트래픽이 적은 시간에 돌아 처리 시간보다 실패 지점을 아는 것이 더 중요했고, 그래서 아래처럼 다시 설계했습니다.

다시 설계한다면: 구현과 측정

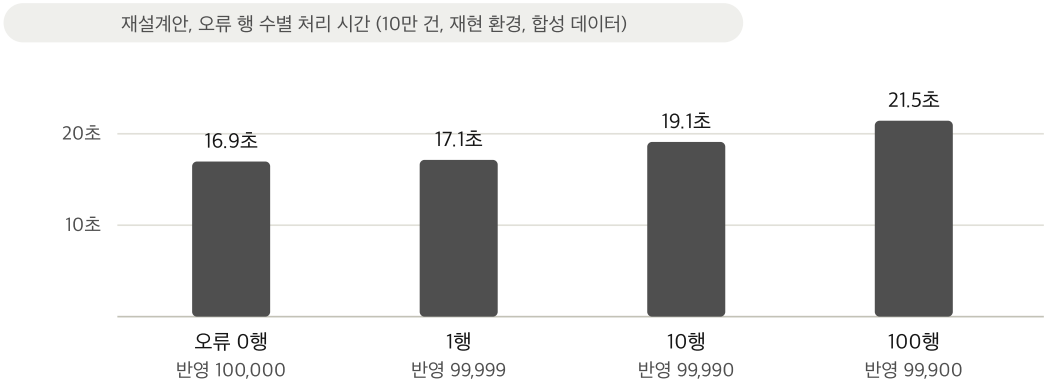
실패가 없는 경우는 묶어서 빠르게 처리하고, 실패가 난 묶음만 잘게 나눠 다시 실행해 실패 행을 정확히 남기는 구조를 같은 재현 환경에 구현해 측정했습니다.



- 묶음마다 `PROPAGATION_NESTED`로 SAVEPOINT(트랜잭션 안에서 되돌아갈 지점)를 두고, 저장이 실패하면 그 지점까지만 되돌린 뒤 나눕니다. 연결 끊김처럼 행과 관계없는 오류는 나누지 않고 그대로 실패시킵니다.
- 실패 기록에는 원본 ID, 실패 단계, 오류 코드와 메시지를 남기고 원본을 오류 상태로 바꿉니다. 묶음이 롤백돼도 기록이 남도록, 묶음마다 실패를 모아 `PROPAGATION_REQUIRES_NEW`로 별도 트랜잭션에서 한 번에 커밋합니다.

오류 행을 넣고 측정

10만 건 중 k개 행에 컬럼 길이를 넘는 값을 넣었습니다. 변환은 성공하고 DB 저장에서 오류(SQLSTATE 22001)가 나는 행입니다. 조건마다 3회 쟀습니다.



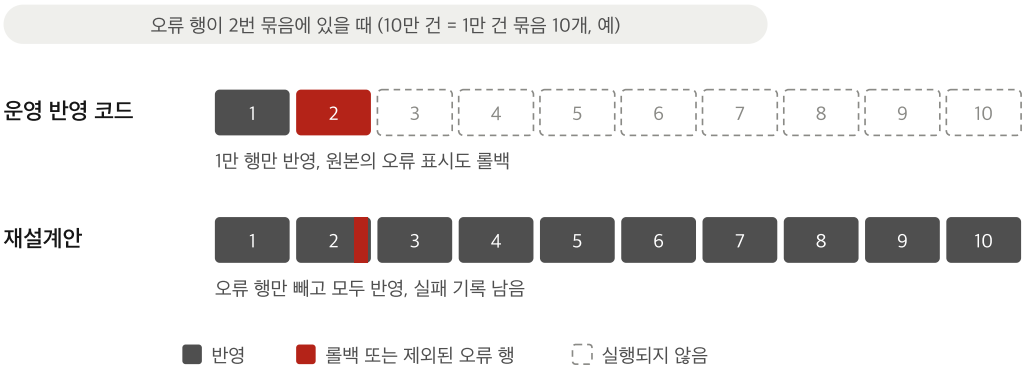
- 실패 기록은 모든 조건에서 오류를 넣은 행과 정확히 같았고, 중복 반영은 없었습니다.
- 저장이 실패하면 재설계안 코드가 그 묶음을 반으로 나눠 다시 저장하고, 실패한 쪽만 다시 나누기를 반복합니다(이분 탐색). 1만 행은 14번 나누면 한 행까지 좁혀지므로, 오류 행이 하나인 묶음은 14번 나눠 그 행을 찾았습니다.

측정값 표 보기

오류 행 수	처리 시간 (중앙값)	반영 행 수	실패 기록	나누기 횟수
0	16.9초	100,000	0	0
1	17.1초	99,999	1	14
10	19.1초	99,990	10	75~79
100	21.5초	99,900	100	462~535

운영 반영 코드와 비교

같은 오류 행(1개, 10개)을 운영 반영 코드에 넣어 비교했습니다.



- 운영 반영 코드는 응답에도 원래 오류가 아니라 “current transaction is aborted”를 돌려줘, 어떤 행이 왜 실패했는지 알 수 없었습니다. 재설계안은 실패 기록에 행별 오류 코드와 메시지를 남겼습니다.
- 운영에서는 보존한 원본으로 재처리했습니다. 오류 행의 상태값을 DB에서 직접 되돌린 뒤 3단계를 다시 실행했고, 재설계안은 이 작업을 실패 기록과 재처리 API로 대신합니다.

비교 표 보기

항목	운영 반영 코드	재설계안
반영 행 수	10,000 (첫 묶음만)	오류 행을 뺀 전부
오류가 든 묶음	1만 행 전체 롤백, 이후 묶음은 실행되지 않음	오류 행만 빼고 저장
원본의 오류 표시	0건 (롤백에 함께 사라짐)	오류 행 수와 같음
응답의 오류 내용	원래 오류가 아닌 “current transaction is aborted”	실패 기록에 행별 오류 코드와 메시지
실패한 행 다시 반영	DB에서 상태값을 직접 되돌린 뒤 3단계 다시 실행	재처리 API로 실패 행만 다시 실행

재처리

- 재처리 API는 해결되지 않은 실패 기록의 행을 오류 전 검증 상태로 되돌려 다시 저장합니다. 성공하면 기록을 해결로 바꾸고, 또 실패하면 새 기록을 남깁니다.
- 오류 10개 조건에서 오류 행의 값을 고친 뒤 재처리 API를 한 번 호출했습니다. 대상 10행이 0.78초에 모두 반영됐고, 반영 내용은 오류가 없던 조건과 같았습니다.

한계

- 이 페이지의 수치는 합성 데이터로 잰 값이며, 실무 수치와 직접 비교하지 않습니다.
- 운영 힙 설정은 확인하지 못했습니다.
- 재설계안의 실패 처리는 생성 지시에만 넣었습니다. 변환 실패와 재처리가 다시 실패하는 경로는 1만 건으로 한 번만 확인했습니다.

LMS 플랫폼 고도화(하이러닝·뉴쎔), 2022.07 ~ 2024.07 설계와 구현

EBS 문항과 자체 제작 문항을 같은 분류체계로 묶어 학년과 단원별 문제집 출제 구조 구현

EBS에서 동기화한 문항 테이블은 그대로 두고 문제집 쪽 테이블을 따로 두어, 교사가 학년과 단원을 고르면 직접, 선택, 랜덤, AI 추천 방식으로 문제집을 만들 수 있게 함

분류체계 스키마 분리 출제 유형 MyBatis

문제와 배경

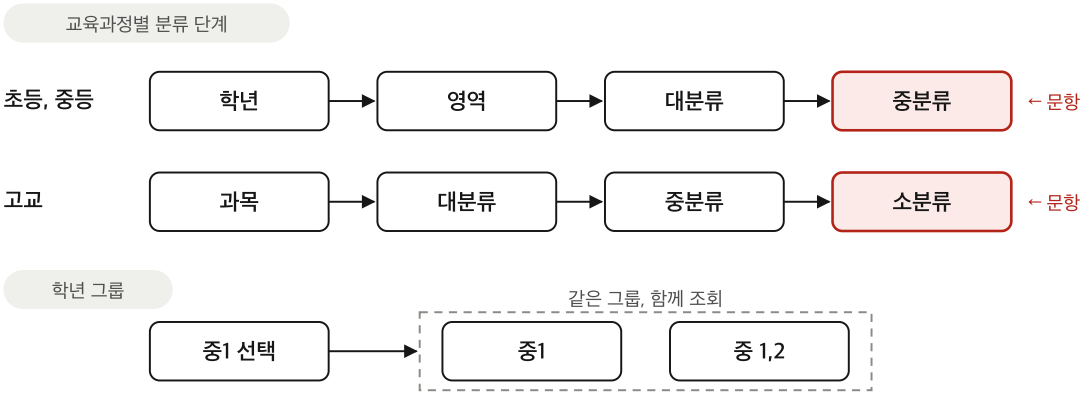
- 뉴쎔 LMS를 바탕으로 하이러닝을 구축하면서, 뉴쎔에 없던 콘텐츠 기능을 외부 AI 업체, EBS와 연계해 새로 만들어야 했습니다.
- EBS 문항은 동기화로 계속 갱신됐고, 교사는 그 문항과 직접 만든 문항으로 학년과 단원에 맞는 문제집을 만들어야 했습니다.

결과

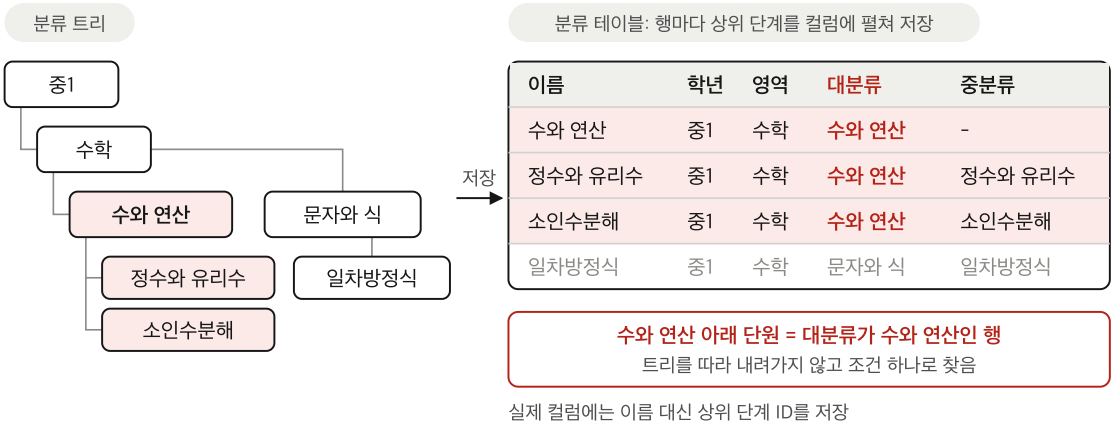
- EBS 동기화 테이블과 문제집 테이블을 나눠, EBS 데이터를 다시 반영해도 교사가 만든 문제집과 문항의 연결이 유지되게 했습니다.
- 직접, 선택, 랜덤, AI 추천, AI 출제를 하나의 등록 흐름에서 유형별로 처리했습니다.
- 문제집이 누적 약 189만 건 사용돼 전체 콘텐츠 사용(클릭 이력 기준 약 219만 건)의 86%를 차지했습니다.
- 새 영어 분류체계를 같은 구조에 반영했습니다.

설계와 구현

학년과 단원으로 문항을 찾는 분류체계

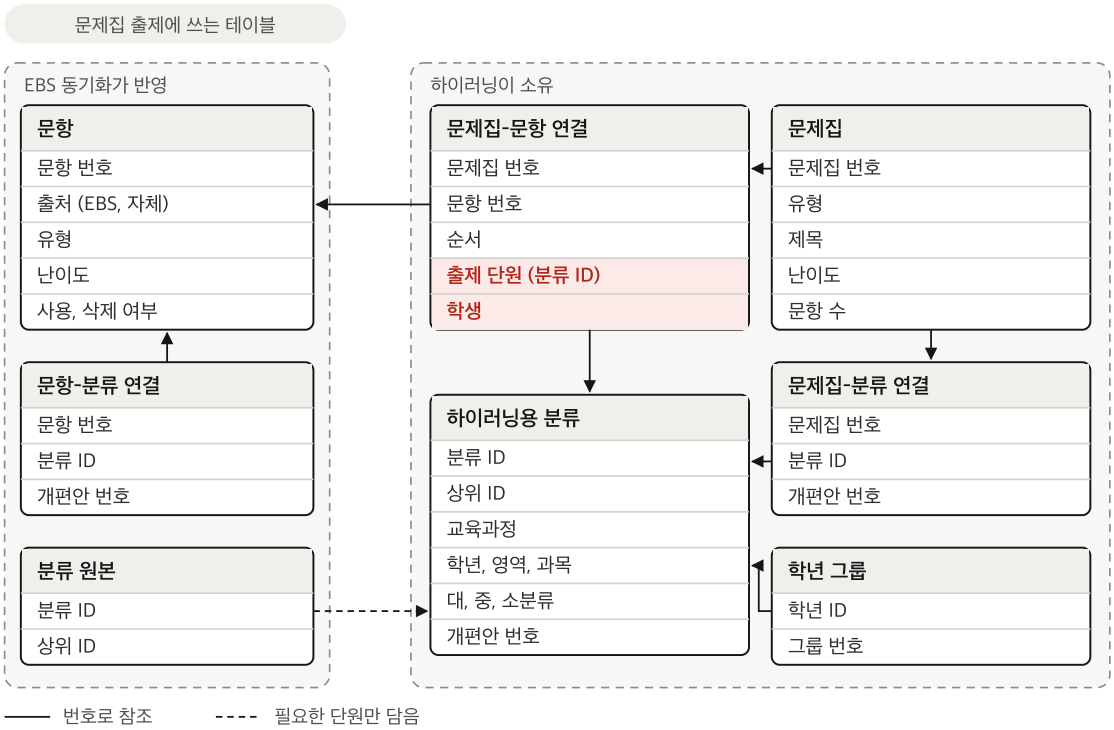


- 교육과정마다 분류 단계가 다르고, 문항이 연결되는 단계도 초등과 중등은 중분류, 고교는 소분류로 다릅니다. 그래서 쿼리에서 교육과정에 따라 비교하는 컬럼을 바꿨습니다.
- “중 1,2”처럼 두 학년에 걸친 분류가 빠지지 않게, 같은 학년 그룹의 학년 ID를 먼저 모아 함께 조회했습니다.



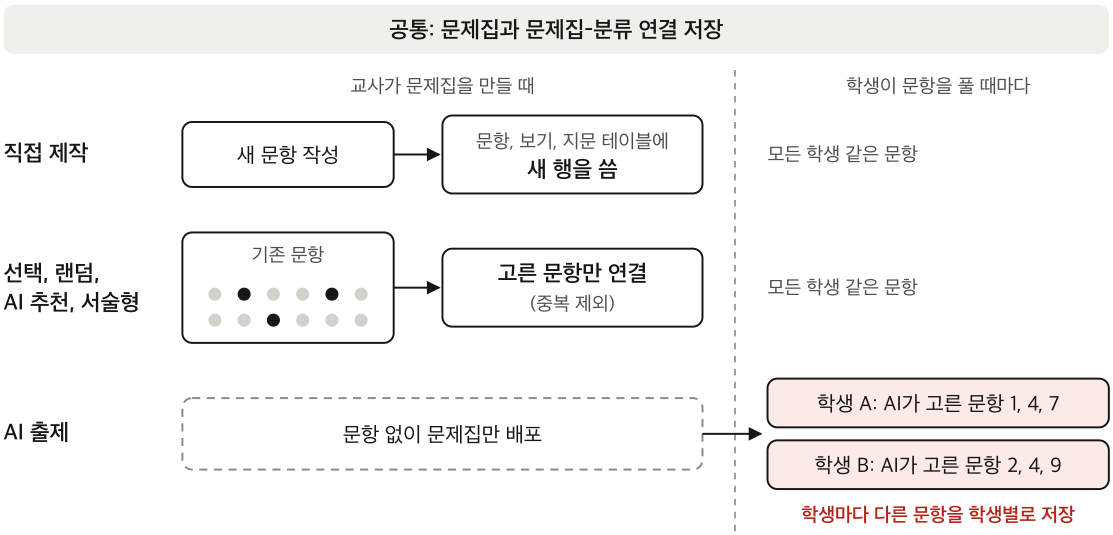
- EBS 원본 분류는 그대로 두고, 하이러닝에서만 쓰는 교육과정과 분류체계를 담은 분류 테이블을 따로 두었습니다. 원본을 바꾸지 않고 하이러닝 전용 분류체계를 쓰기로 되어 있었기 때문입니다.
- 분류 데이터는 행마다 상위 단계 ID를 컬럼에 펼쳐 둔 구조라, 상위 ID를 재귀로 따라가지 않고 조건 하나로 아래 단원을 찾습니다.

테이블 구성



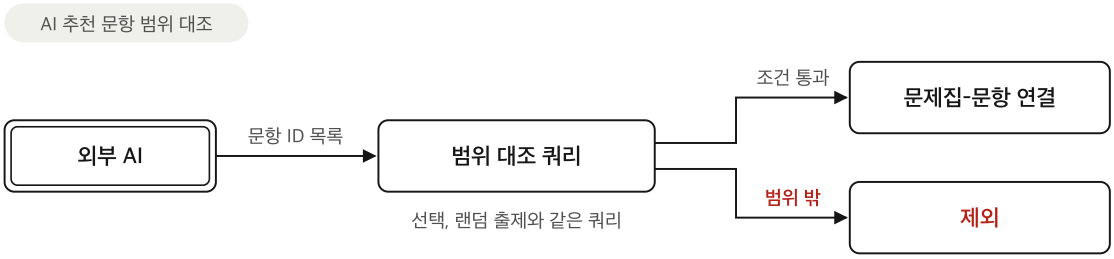
- 직접 만든 문항도 EBS와 같은 문항 테이블에 저장하고 출처 컬럼으로 구분해, 선택, 랜덤, AI 추천 출제가 같은 조회 쿼리를 씁니다. 자체 문항 번호는 EBS에서 오는 문항 번호와 형식을 맞추려고 DB 함수에서 만들었습니다.
- 문제집 쪽 테이블은 EBS 동기화가 건드리지 않고, 동기화는 문항을 논리 삭제만 합니다. 그래서 EBS 데이터를 다시 반영해도 문제집과 문항의 연결이 끊기지 않습니다.
- 문제집-분류 연결에는 만들 때의 개편안 번호를 함께 저장했습니다.
- 문항 하나가 여러 단원에 연결될 수 있어서(문항-분류 연결은 다대다), 문제집-문항 연결에 "어느 단원으로 출제했는지"를 따로 저장했습니다. 문제집 화면은 이 값으로 단원 이름을 붙여, 교사가 고른 단원 그대로 보여 줍니다.
- AI 출제는 학생마다 다른 문항을 받으므로, 문제집-문항 연결에 학생을 함께 저장했습니다.

출제 유형별 등록 흐름



- 직접 제작만 문항 테이블에 새 문항을 쓰고, 나머지 유형은 기존 문항을 골라 문제집-문항 연결에만 저장합니다.
- AI 출제는 문제집을 만들 때 문항을 저장하지 않고, 학생이 문항을 풀고 다음으로 넘어갈 때마다 외부 AI가 고른 다음 문항을 학생별로 저장합니다.
- 출제 화면에는 단원마다 실제로 고를 수 있는 문항 수만 보여 줘, 빈 문제집이 만들어지거나 형성평가 문항이 일반 문제집에 섞이지 않게 했습니다.

AI 추천 문항을 출제 범위와 대조



- 외부 AI가 준 문항 ID 목록에는 교사가 고른 범위 밖의 문항이 섞일 수 있었습니다. 예를 들어 중1 단원을 골랐는데 중2 문항이 오는 경우입니다.
- 그래서 받은 문항 ID를 선택, 랜덤 출제와 같은 문항 조회 쿼리로 내부 문항 테이블과 대조했습니다. 교사가 고른 단원(고교는 소분류, 그 밖에는 중분류)과 난이도에 맞고, 사용 중이며, 형성평가 문항이 아닌 것만 남깁니다.
- 조건 밖의 문항은 문제집에 저장하기 전에 목록에서 빠집니다.

같은 문항을 받은 학생의 문항이 저장되지 않던 문제

변경 전: 유니크 (문제집, 문항)

문제집	문항	학생	결과
W1	7	A	저장
W1	7	B	저장 안 됨

A 행만 갱신되고 B 행은 생기지 않아 B 화면이 멈춤

변경 후: 유니크 (문제집, 문항, 학생)

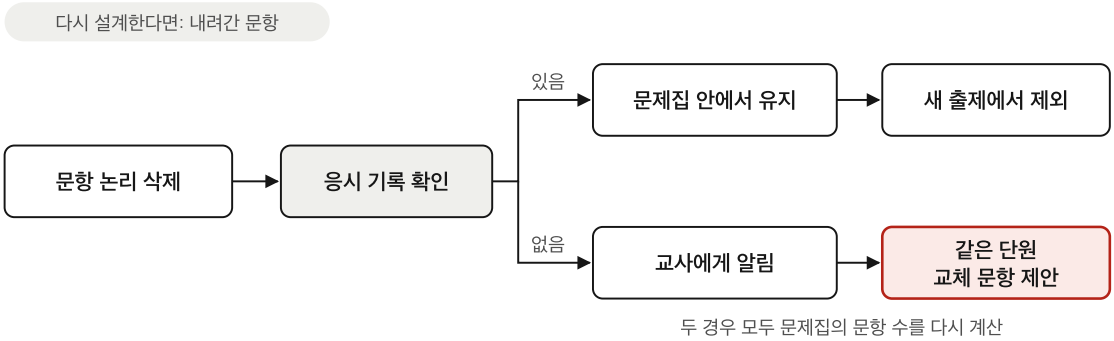
문제집	문항	학생	결과
W1	7	A	저장
W1	7	B	저장

학생마다 같은 문항을 따로 저장

- 시연 중 AI 출제 문제집에서 학생이 다음 문항으로 넘어갈 때 응시 화면이 멈추는 문제가 처음 발견됐습니다. 범위 대조를 거치는 AI 추천과 달리, AI 출제는 학생마다 문항을 따로 저장합니다.
- AI 출제는 학생별로 문항을 저장하는데, 저장의 유니크 범위와 충돌 조건이 (문제집, 문항)이었고 충돌하면 기존 행을 갱신(`DO UPDATE`)했습니다. 외부 AI가 고른 문항이 다른 학생에게 이미 배정된 문항이면 앞 학생의 행만 갱신되고 뒤 학생의 행은 생기지 않아, 화면은 문항을 받지 못한 채 멈췄습니다.
- 재현 경로를 찾아 유니크 범위와 충돌 조건에 학생 식별자를 넣었습니다(`ON CONFLICT (문제집, 문항, 학생)`). 수정 뒤 QA를 통과했고, 같은 증상은 다시 나타나지 않았습니다.

남은 한계와 해결 방향

- EBS가 문항을 내리면 배포한 문제집에서도 그 문항이 빠지고, 문제집에 저장한 문항 수와 실제 문항 수가 달라질 수 있습니다.



- 다시 설계한다면 응시 기록으로 나눠 처리하겠습니다. 응시 기록이 있는 문제집은 학생이 응시한 문항과 결과 화면이 서로 맞아야 하므로 문항을 유지하고, 응시 전 문제집만 교체를 제안합니다. 교체 문항은 문제집-문항 연결에 저장해 둔 출제 단원으로 찾습니다.

LMS 플랫폼 고도화(하이러닝·뉴쎔), 2022.07 ~ 2024.07 설계와 구현

jQuery 화면을 전면 재작성 없이 Vue 컴포넌트 구조로 전환하고 응답 규격 통일

빌드 체계가 없는 환경에서 JSP 화면에 Vue 컴포넌트를 마운트하는 방식으로 전환하고, JSP로 필드마다 매핑하던 응답을 공통 응답 객체의 JSON으로 바꿈

Vue.js JSP 공통 응답 규격 공통 컴포넌트

문제와 배경

- 기존 화면은 jQuery와 콜백 방식 ajax로 만들어져 있었습니다.
- 2023년 5월에 리뉴얼을 시작해, 9월에 구버전을 함께 운영하지 않고 새 화면으로 오픈해야 했습니다.
- npm 빌드와 배포 체계가 없었습니다.
- 기존 API는 api JSP가 응답 필드를 하나씩 매핑해 JSON을 만들었고, API마다 응답 형식이 달랐습니다.

결과

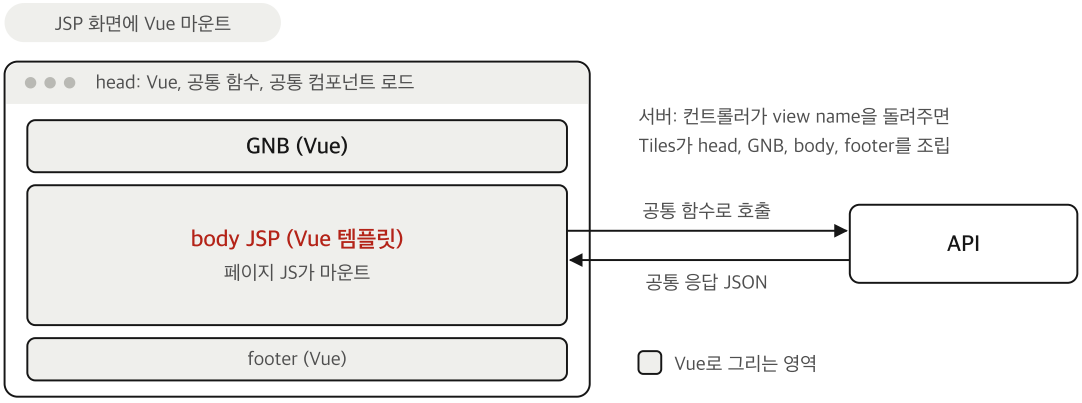
- 전면 SPA 재작성 없이 Vue 컴포넌트 구조로 전환해 2023년 9월에 오픈했습니다.
- 같은 레이아웃과 공통 컴포넌트 위에서 팀이 같은 방식으로 화면을 만들었습니다. 저장소 기준으로 새 화면 구조에서 Vue를 마운트하는 페이지 JS가 133개입니다.
- 공통 응답 객체의 JSON 응답을 팀 API에 적용했습니다. 저장소 기준으로 컨트롤러 45개 파일이 이 형식을 씁니다.

설계와 구현

전환 방식 선택

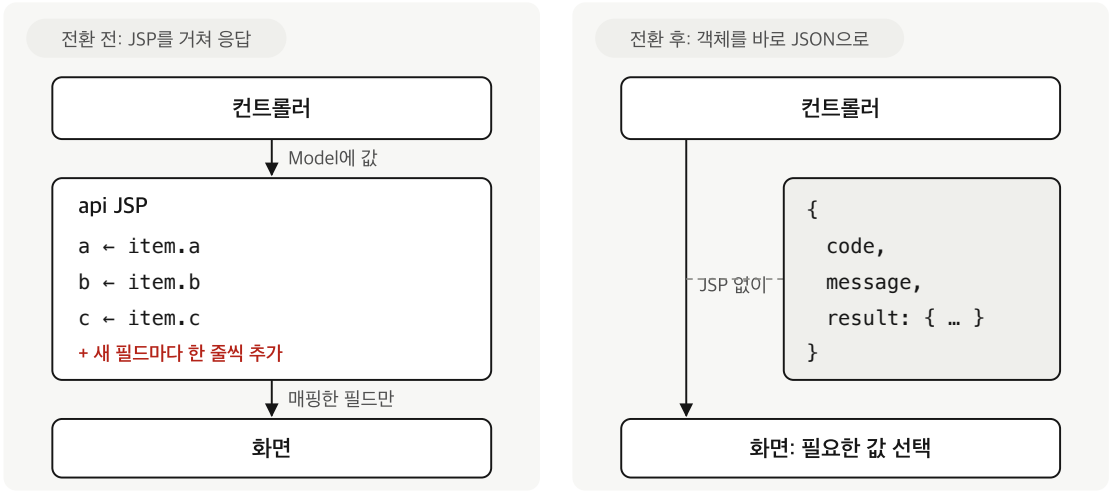
- 전환 방침은 팀 회의로 정했고, 전환 리드로서 최종 선택을 했습니다.

기준	검토한 방식: SPA 전면 재작성	선택한 방식: JSP 화면에 Vue 마운트
빌드와 배포	npm 빌드와 배포 체계를 새로 만들어야 함	**`<script>` 태그로 라이브러리를 불러와 기존 배포 그대로**
라우팅과 세션	클라이언트 라우팅과 세션 처리를 다시 구성	**서버가 view name으로 페이지를 돌려주는 흐름 유지**
다시 만들 범위	전체 화면	**페이지별 body JSP와 페이지 JS**



- Tiles를 Vue와 함께 새로 도입했습니다. Tiles는 head, GNB, footer 같은 공통 JSP 조각과 페이지별 body JSP를 하나의 레이아웃으로 조립하는 JSP 템플릿 프레임워크입니다.
- 레이아웃의 head에서 Vue, 공통 통신 함수, 공통 컴포넌트를 한 번 불러오고, 페이지마다 body JSP만 둡니다. 웹과 모바일은 레이아웃을 따로 두었습니다.
- 화면 데이터는 서버가 채우지 않습니다. GNB, body, footer 모두 JSP에 Vue 템플릿을 두고 브라우저에서 마운트합니다. body는 페이지 JS가 Vue를 마운트한 뒤 API로 받은 데이터로 그립니다.

공통 응답 규격



- 기존 API는 컨트롤러가 Model에 값을 담으면, api JSP가 JSON 태그 라이브러리로 응답 필드를 하나씩 매핑해 내려보냈습니다. 화면에 필요한 값이 늘면 JSP의 매핑도 함께 고쳐야 했습니다.
- 백엔드와 협의해, 컨트롤러가 공통 응답 객체(code, message, result)를 그대로 JSON으로 돌려주게 바꿨습니다. 화면은 받은 값 가운데 필요한 것을 골라 씁니다.

공통 통신 함수와 공통 컴포넌트

- axios 기반 공통 통신 함수를 만들었습니다. GET, POST, DELETE, multipart 요청마다 공통 함수를 두고, 함수 안에서 입력값을 검사합니다. 통신이 실패하면 공통 오류 객체로 바꿔 error 콜백에 넘겼습니다.
- modal, select, tab을 웹과 모바일 공통 컴포넌트로 만들어 전역 등록하고, 새 화면에서 재사용하게 했습니다.