

Module 4 — 데이터 관리: 가장 비싼 실패는 '성공했다고 보고된 실패'다

이 프로젝트의 데이터 사고는 하나도 예외 없이 **성공 로그를 남기고** 일어났습니다. DB 초기화는 성공 표시가 떴는데 테이블이 비어 있었고, 배포는 매번 정상 종료했는데 실제로는 전부 건너뛰어졌으며, 위키 페이지 22개는 발행에 성공했는데 본문이 없었습니다.

모듈	난이도	학습 시간	선행 학습
Module 4	실무	90분	Module 2

학습 목표 (Learning Objectives)

이 모듈을 마치면 수강자는 다음을 할 수 있습니다.

- ✓ 데이터 파이프라인의 **조용한 실패**(0건 수집·스키마 드리프트)를 실패로 분류할 수 있다
- ✓ 재실행해도 행이 늘지 않는 **먹등 시딩**을 유니크 키로 설계할 수 있다
- ✓ "먹등하다"는 주장을 **겹(layer)별로 쪼개** 어디까지 검증됐는지 정확히 말할 수 있다
- ✓ 이식성을 위해 **스키마가 쓰지 않을 기능**을 의도적으로 정할 수 있다
- ✓ 배포·빌드 관정도 데이터로 보고, **조용한 누락**을 감시 목록으로 막을 수 있다

실무 배경 — 어떤 문제에서 출발했나

- **상황:** 강의 웹(`lecture-web`)은 코스·섹션·레슨·실습 데이터셋·시뮬레이터 설정·회원을 DB 에 들고 있고, 교안 본문은 저장소 마크다운에서 읽으며, 배포는 세 Vercel 프로젝트가 한 저장소를 공유합니다. **데이터가 사는 곳이 세 군데**입니다.
- **기존 방식의 비용:** 세 군데가 어긋나도 아무도 에러를 내지 않습니다. 실제로 다음 세 건이 전부 "성공"으로 보고됐습니다.

#	성공이라고 보고된 것	실제 상태	근거
1	웹 SQL 콘솔의 덤프 실행	테이블이 빈 채로 남음	커밋 <code>d8abad2</code>
2	Vercel 배포 판정	모든 빌드가 건너뛰어짐	커밋 <code>191eab7</code>
3	위키 교안 페이지 발행	본문이 <code>{{...}}</code> 자리표시자	커밋 <code>ef31b5c</code>

실무 배경 — 어떤 문제에서 출발했나 (계속)

- 참고 실습 데이터: 수집 파이프라인 30일 장애 로그에서도 같은 패턴이 보입니다 — 37건 중 **30건(81%)**이 스키마 드리프트이고, 평균 발견까지 20시간 이상 걸렸습니다 (`module-4-schema-drift.json`). 인증 만료(4건)·레이트 리밋(3건)은 즉시 발견됐습니다. **시끄러운 실패는 안 위험합니다.**

💡 이 모듈의 핵심 교훈: "성공했다"는 로그를 근거로 쓰지 마세요.

개수를 세세요. 개수는 거짓말하지 않습니다.

3.1 프롬프팅 프레임워크 적용 — R-C-T-C-F

요소	실제 입력값
Role	데이터 파이프라인 검 릴리스 담당. 로그가 아니라 관측치로 판정 하는 사람
Context	데이터가 DB·저장소 마크다운·Vercel 세 곳에 있고 한 저장소를 세 프로젝트가 공유한다
Task	스키마 정의 · 먹등 시딩 · 배포 판정 스크립트 구현
Constraints	0건·필드 누락·타입 변경을 서로 다른 오류로 구분 · 재실행 안전 · 성공을 주장하려면 관측치를 함께 낼 것
Format	Prisma 스키마 + 시드 스크립트 + 종료 코드가 있는 판정 스크립트

3.2 아키텍처 — 검증을 앞으로 당기고, 판정을 관측치로

```

[외부 소스] → [수집] → [스키마 검증] ── 통과 → [적재] → [실습 데이터셋 발행]
                        └─ 위반 → [즉시 알림 + 적재 중단]
                              (0건 수집도 '위반')
  
```

```

[저장소 마크다운] → [교안 로더] → 출처 배지(REPO / LOCAL / NONE)
  
```

```

[커밋] → [무시 빌드 판정] ── 감시 경로 변경 있음 → 빌드
                        └─ 없음 → 건너뛰기           ← 종료 코드가 반대라 함정
  
```

§4

구현 과정 (Step by Step)

단계	작업	소요	산출물
1	소스별 필수 필드·타입 정의, 0건을 실패로 분류	30분	<code>module-4-schema-drift.json</code>
2	먹등 시딩 — 유니크 키 설계	30분	<code>prisma/schema.prisma</code>
3	SQLite ↔ Postgres 이식성 확보	30분	스키마 제약 규칙
4	DB 이관 (덤프 → 웹 콘솔)	40분	실패 · §4.4
5	배포 판정 스크립트와 감시 경로	30분	<code>scripts/vercel-ignore-build.sh</code>

4.1 0건은 성공이 아니다

수집 결과가 0건이면 "성공적으로 아무것도 못 가져온 것"이 아니라 실패입니다. 세 가지 오류를 서로 다르게 취급하는 것이 규칙입니다.

오류	증상	취급
필드 누락	필수 키가 사라짐	적재 중단 + 알림
타입 변경	<code>version</code> 이 숫자 → 문자열	적재 중단 + 알림
0건 수집	응답은 200, 행이 없음	실패로 분류 (가장 놓치기 쉬움)

```
// 0건은 '성공적으로 아무것도 못 가져온 것'이 아니라 실패입니다.  
if (rows.length === 0) {  
    throw new Error('0건 수집 - 소스 스키마가 바뀌었는지 먼저 확인하세요.');}
```

4.1 0건은 성공이 아니다 (계속)

검증 위치도 규칙입니다. 적재 뒤가 아니라 수집 앞단입니다. 뒤에 두면 이미 오염된 데이터를 되돌려야 하고, 되돌리는 비용이 검증 비용보다 항상 큼니다.

4.2 멍등 시딩 — 유니크 키가 곧 계약

시드 스크립트는 `upsert` 만 씁니다. 몇 번을 돌려도 행이 늘지 않아야 "일단 돌려보자"가 가능해집니다. 멍등성을 보장하는 것은 코드가 아니라 스키마의 유니크 키입니다.

```
model Lesson {  
    /// (moduleId, title) 로 멍등 시딩·업서트합니다. 재실행해도 레슨이 늘지 않습니다.  
    @@unique([moduleId, title])  
    @@index([moduleId, order])  
}  
  
model PracticeDataset {  
    @@unique([lessonId, title])  
    @@index([lessonId])  
}
```

키를 안 잡고 `create` 를 부르면 재실행할 때마다 레슨이 두 배가 됩니다. 그 사고는 운영 데이터에서 나면 되돌리기 어렵습니다 — 어느 쪽이 원본인지 알 수 없기 때문입니다.

4.2 맥등 시딩 — 유니크 키가 곧 계약 (계속)

2026-08-10 배포본 실측: 코스 5 · 섹션 23 · 레슨 51 · 회원 6. 숫자를 사람이 세지 않고 조회로 확인합니다(Module 2 Ch1 §5 의 "18개 vs 22개" 사고와 같은 이유입니다).

4.3 이식성은 '쓰지 않을 기능'을 정하는 일

운영은 PostgreSQL 이고, 로컬에서는 SQLite 로 돌리고 싶습니다. Prisma 는 `datasource provider` 에 `env()` 를 못 쓰므로 provider 가 파일에 박힙니다. 그래서 스키마가 **SQLite** 가 지원하지 않는 기능을 아예 쓰지 않습니다.

명세대로라면	실제 스키마	이유
<code>enum Role</code>	<code>String</code> + <code>ROLES</code> 리터럴 유니온	Prisma + SQLite 는 enum 미지원
<code>String[]</code>	<code>String</code> (JSON 직렬화)	배열 타입 미지원
<code>Json</code>	<code>String</code> (JSON 직렬화)	동일

허용값의 단일 원본은 `src/lib/constants.ts` 이고, 저장되는 문자열이 그대로 **enum** 라벨이라 Postgres 로 넘어가 enum 으로 승격해도 데이터 마이그레이션이 필요 없습니다.

한 가지 함정이 있습니다. 로컬 개발용으로 provider 를 `sqlite` 로 바꾼 채 커밋하면, 배포 빌드가 **SQLite** 로 나가 서버리스에서 쓰기가 안 되고 파일도 휘발됩니다. 그 사실을 스키마 첫 줄 주석에 적었습니다(커밋 [6207d80](#)).

4.4 DB 이관에서 실제로 뱉은 함정 — 성공 표시가 뜬 빈 테이블

Claude 웹 세션의 네트워크 정책이 Postgres(5432)를 막고 있어 `db:push` · `db:seed` 를 쓸 수 없었습니다. 대신 SQL 덤프를 뽑아 웹 SQL 에디터에 붙여 넣었습니다.

```
pg_dump ...           → COPY "Lesson" (...) FROM stdin;   ← 본문이 뒤따르는 형식
웹 SQL 에디터         → 이 구문의 본문을 무시하면서 "성공" 으로 표시
결과                 → 스키마만 생기고 테이블은 빈 채로 남음
```

`--inserts` 없이 뽑으면 데이터가 `COPY ... FROM stdin` 으로 적히는데, 웹 콘솔은 그 본문을 처리하지 않으면서도 실패를 알리지 않습니다(커밋 `d8abad2`).

교훈은 도구가 아니라 판정 방법에 있습니다. "성공"이라는 표시를 근거로 삼지 말고 `select count(*)` 를 세면 5초 만에 드러납니다.

4.5 배포 판정도 데이터다

한 저장소에 Vercel 프로젝트가 셋(`lecture-web` · `action` · `hajeehyun-portfolio`)이라 아무 커밋에나 셋이 전부 빌드됩니다. Ignored Build Step 스크립트가 걸러 내는데, 여기 함정이 둘 있습니다.

```
# ⚠️ 종료 코드 규약이 직관과 반대입니다.  
# 0 → 빌드 건너뛰  
# 0 아님 → 빌드 진행  
WATCH=('/:lecture-web' ':/docs/4-product-system/lecture')  
  
if git diff --quiet 'HEAD^' HEAD -- "${WATCH[@]}"; then  
    echo "감시 경로에 변경 없음 - 빌드를 건너뛰니다: ${WATCH[*]}"  
    exit 0  
fi  
exit 1
```

4.5 배포 판정도 데이터다 (계속)

- 종료 코드 반전: `0` 이 "건너뛸"입니다. 반대로 알고 쓰면 정확히 반대로 동작하는데, **양쪽 다 에러를 내지 않습니다.**
- 경로 앞의 `:/`: "저장소 루트 기준"이라는 git 표기입니다. Vercel 은 이 스크립트를 Root Directory(`lecture-web/`) 안에서 실행하므로, `lecture-web` 이라고만 쓰면 `lecture-web/lecture-web` 을 찾다가 "변경 없음"이 되어 **모든 빌드가 조용히 건너뛰어집니다.**

감시 목록도 데이터입니다. 목록에서 빠진 경로는 고쳐도 반영되지 않습니다 — 뒤이어 `api/` 와 함수 번들에 포함되는 원문 경로를 추가해야 했습니다(커밋 `f078fce`). 교안 디렉토리(`docs/4-product-system/lecture/`)가 감시 목록에 들어 있는 것도 같은 이유입니다. 교안은 앱 폴더 밖에 있지만 배포본에 함께 실립니다.

판단 근거가 없을 때의 기본값도 정해 뒀습니다 — 얇은 클론이라 `HEAD^` 를 못 찾으면 **건너뛰지 않고 빌드합니다.** 잘못 건너뛰면 사람이 한참 뒤에야 압니다.

이 교안을 배포하다가 같은 함정에 다시 빠졌습니다 (2026-08-11)

4.5 배포 판정도 데이터다 (계속)

이 교안이 담긴 PR(#48)을 배포하려는데 빌드가 `CANCELED` 로 끝났습니다. 사유는 `ignored-build-step` 이었습니다.

커밋 d6acb90 교안 3개 추가 → 배포 시도, Vercel 일일 한도로 rate limit. 빌드 안 됨
커밋 7ea869f origin/main 머지 → 무시 빌드 판정: "감시 경로에 변경 없음" → 건너뛴

판정 자체는 규칙대로입니다. `7ea869f` 는 머지 커밋이라 `HEAD^` 가 첫 부모(`d6acb90`)이고, 그 사이에 들어온 `main` 의 변경은 전부 감시 경로 **밖**이었습니다(`pm-po-automation-tools*`, `docs/4-product-system/README.md`). 그러니 "변경 없음"이 맞습니다.

틀린 것은 **가정**입니다. 스크립트는 이렇게 전제하고 있습니다.

직전 커밋 대비 변경이 없다 ⇒ 직전 배포가 아직 유효하다

4.5 배포 판정도 데이터다 (계속)

이 브랜치에서는 앞줄이 참인데 뒷줄이 거짓입니다. `d6acb90` 의 배포가 한도에 걸려 애초에 존재하지 않기 때문입니다. 물려받을 배포가 없는데 "그대로 두면 된다"고 판정한 것입니다.

사실	값
배포 ID	<code>dpl_EVX3rqH6vHh4AYWqNd3yzHeBaavv</code>
상태	<code>CANCELED</code> · <code>errorLink: .../projects#ignored-build-step</code>
직전 배포	없음 (<code>d6acb90</code> 은 <code>Deployment rate limited – retry in 24 hours</code>)

⚠ 일반화: 증분 판정은 직전 상태가 성공했다는 것에 기대고 있습니다.

그 전제가 깨지는 순간(rate limit·취소·롤백) 판정은 조용히 틀립니다.

"변경 없음"으로 건너뛰기 전에 물려받을 배포가 실제로 있는지를 함께 봐야 합니다.

§4

4.5 배포 판정도 데이터다 (계속)

교안 §2 의 표현을 그대로 쓰면, 이것도 **성공했다고 보고된 실패**입니다 — Vercel 화면에는 빨간 실패가 아니라 회색 **Ignored** 로 표시됩니다.

결과 & 검증 (Result & Validation)

지표	Before	After	측정 근거
시딩 재실행	행이 늘어남	유니크 키로 먹등	<code>@@unique</code> 2종 · 코스 5/섹션 23/레슨 51
DB 이관 판정	"성공" 표시	<code>count(*)</code> 관측치	커밋 <code>d8abad2</code>
배포 누락	조용히 전부 건너뛴	커밋 3종으로 판정 확인	커밋 <code>191eab7</code> · <code>f078fce</code>
증분 판정의 전제	"변경 없음 = 직전 배포 유효"	반례 확보 (\$4.5)	<code>dpl_EVX3rqH6vHh4AYWqNd3yzHeBaavv</code>
교안 본문 출처	빈 템플릿에 밀림	저장소 우선 + 출처 배지	<code>tests/moduleGuide.test.ts</code>
스키마 이식성	provider 고정	한 줄 교체로 SQLite ↔ Postgres	<code>prisma/schema.prisma</code> 주석

배포 판정은 실제 커밋 3종으로 확인했습니다 — 빈 커밋은 건너뛴, `lecture-web` 변경은 빌드, Product 1 커밋은 건너뛴.

5.1 "멱등하다"고 말할 수 있는 범위 — 두 겹 중 한 겹만 확인됨

Q&A → Jira 티켓 생성의 멱등성은 두 겹입니다. 2026-08-10 실측에서 바깥쪽만 확인됐습니다(04 §9).

겹	동작	상태
① DB 가드	<code>jiraTicketKey</code> 가 있으면 Jira 를 호출조차 하지 않음	✅ 확인 (2회차에 새 티켓 없음)
② Jira 가드	<code>ensureIssue</code> 가 제목 <code>[Q&A] ...</code> 으로 기존 티켓을 먼저 조회	❌ 미확인

②는 **DB** 의 티켓 키가 비었는데 **Jira** 에는 티켓이 남아 있을 때만 발동합니다. DB 를 초기화하고 재시드하면 실제로 생기는 상황이라 언젠가 반드시 확인해야 합니다.

확인 방법을 스크립트로 만들어 뒀습니다 — `npm run verify:idempotency`.

5.1 "먹등하다"고 말할 수 있는 범위 — 두 겹 중 한 겹만 확인됨 (계속)

처음 적어 둔 방법은 "`jiraTicketKey` 를 null 로 만들고 백필을 재실행"이었는데, 운영 데이터를 건드리는 절차라 아무도 실행하지 않게 됩니다. 다시 보니 그럴 필요가 없었습니다. ②의 판정은 전부 `ensureIssue` 안에 있고 그 함수는 DB 를 보지 않으므로, 같은 제목으로 한 번 더 호출하는 것이 "DB 키가 비어 있는 상황"과 같은 입력입니다.

더 중요한 것은 실패해도 안전하다는 점입니다. `INTEGRATION_DRY_RUN=true` 에서 읽기는 나가고 쓰기만 막히므로 두 결과가 명확히 갈립니다.

결과	반환 키	뜻
② 통과	<code>LECTWEB-12 (created: false)</code>	조회가 기존 티켓을 찾아 재사용
② 실패	<code>LECTWEB-DRY (created: true)</code>	생성으로 갔고 쓰기가 막힘 — 실제 티켓은 안 생김

검증 때문에 쓰레기 티켓이 남지 않으므로 스크립트는 DRY-RUN 을 강제합니다.

5.1 "먹등하다"고 말할 수 있는 범위 — 두 겹 중 한 겹만 확인됨 (계속)

이 표가 이 모듈에서 가장 중요합니다. "먹등합니다"는 한 문장으로 쓰면 검증할 수 없습니다. 겹으로 쪼개면 어디까지 확인됐고 무엇이 남았는지가 드러납니다. 그리고 **확인 절차가 위험하면 영영 확인되지 않습니다** — 절차를 안전하게 만드는 것까지가 검증 설계입니다.

 이 세션은 awesomest.atlassian.net 과 Postgres(5432)가 네트워크 정책으로 차단돼 있어 ②를 여기서 확인하지 못했습니다. 확인은 배포본에서 해야 합니다.

배운 것 / 다시 한다면 바꿀 것

-  **잘한 것:** 맥등성을 겹으로 쪼개고 **미확인 겹을 미확인이라고 적은 것.** 다 됐다고 적었으면 아무도 다시 안 봤을 겁니다.
-  **잘한 것:** 판정 스크립트를 설정칸이 아니라 저장소에 둔 것. 종료 코드 반전 같은 함정을 주석으로 남길 수 있고, 판정 근거가 git 이력에 남습니다.
-  **다시 한다면:** 이관 직후에 `count(*)` 를 먼저 세겠습니다. 콘솔의 "성공" 표시를 한 번 믿은 대가가 재작업 한 라운드였습니다.
-  **다시 한다면:** 감시 목록을 처음부터 "배포본에 실리는 파일 전부"로 정의하겠습니다. `src/` 만 감시한 탓에 함수 번들에 들어가는 원문이 옛 사본으로 남았습니다.

산출물 링크 (Deliverables)

구분	URL
교안 PDF/슬라이드	/downloads/module-4-data-ops.pdf
슬라이드 (HTML)	/downloads/module-4-data-ops.html
실습용 데이터 세트	/downloads/module-4-schema-drift.json
시뮬레이터 실행	(해당 없음)
GitHub 저장소	https://github.com/ursulax/action
DB 스키마	lecture-web/prisma/schema.prisma
역등 시드 스크립트	lecture-web/prisma/seed.ts
배포 판정 스크립트	lecture-web/scripts/vercel-ignore-build.sh
환경 점검 (읽기 전용)	lecture-web/scripts/doctor.ts
연동 실측 기록	docs/4-product-system/04-product3-course-web.md
관련 PR	https://github.com/ursulax/action/pull/45

§6

산출물 링크 (Deliverables) (계속)

실습 과제 (Action Item)

1. 자기 파이프라인의 소스 하나를 골라 필수 필드 목록을 적고, 필드를 하나 지웠을 때 무슨 일이 나는지 실제로 확인하세요. 에러가 안 나면 그게 이 모듈의 문제입니다.
1. **0건 처리를 확인하고 실패로 바꾸세요.** 지금 0건이 어떻게 기록되는지 로그를 직접 찾아보세요 — 대부분 "정상 종료"로 남아 있습니다.
1. 본인 시스템의 "역등합니다" 주장을 겹으로 쪼개세요. `` 겹 동작 확인됨? 확인 방법 ① 로컬 상태로 중복 호출 차단 ☒ 재실행 후 개수 비교 ② 외부 시스템에서 기존 항목 재사용 ☐ 로컬 상태를 지우고 재실행 `` 확인 안 된 겹은 ☐ 로 남겨 두는 것이 정답입니다.
1. 성공 로그가 아니라 개수로 판정하는 확인 절차를 하나 만드세요. 배포 후 `count(*)`, 수집 후 행 수, 발행 후 페이지 수 — 무엇이든 셀 수 있는 것.

제출물: 스키마 정의 + 0건 처리 변경 diff + 역등성 겹 표 + 개수 기반 확인 절차 자가 점검 기준:

실습 과제 (Action Item) (계속)

- ✓ 필드 누락·타입 변경·0건이 서로 다른 오류로 구분되는가
- ✓ 스크립트를 두 번 돌려도 행 수가 같은가
- ✓ "확인됨"과 "미확인"이 문서에서 구분되는가
- ✓ 성공 판정의 근거가 로그가 아니라 개수인가

Q&A / 자주 막히는 지점

증상	원인	해결
파이프라인이 성공했는데 데이터가 없다	0건을 성공으로 처리한다	0건을 위반으로 분류하고 알림을 낸다
어제 되던 게 오늘 안 된다	소스 스키마가 바뀌었다	수집 앞단 검증 + 필수 필드 목록
시드를 돌릴 때마다 행이 늘어난다	유니크 키 없이 <code>create</code> 를 쓴다	<code>@@unique</code> 를 잡고 <code>upsert</code> 로
SQL 콘솔이 성공했는데 테이블이 비었다	<code>COPY ... FROM stdin</code> 본문이 무시됨	<code>pg_dump --inserts</code> 로 뽑고 <code>count(*)</code> 로 확인
배포했는데 반영이 안 된다	무시 빌드 판정이 건너뛰었다	종료 코드 규약(0=건너뛸)과 <code>:/</code> 경로 표기 확인
머지 커밋에서만 배포가 건너뛰어진다	<code>HEAD^</code> 가 첫 부모라 머지로 들어온 변경이 안 잡힌다	물려받을 직전 배포가 실제로 있는지 함께 확인 (§4.5)
파일을 고쳤는데 함수는 옛날 것이다	감시 목록에 그 경로가 없다	배포본에 실리는 파일 전부를 감시 목록에
로컬은 되는데 배포에서 쓰기가 안 된다	provider 가 <code>sqlite</code> 로 커밋됐다	provider 를 <code>postgresql</code> 로. 스키마 주석 참고
"먹등합니다"를 검증할 수 없다	주장이 한 문장이다	겹으로 쪼개고 겹마다 확인 절차를 적는다