Django 백엔드 스터디 3 회차 문제

2025-08-15

작성자 : 김호중

백엔드 – 논리적사고문제: 헬스케어 어플리케이션 데이터베이스 ORM

처방 제출·보험청구 이중 처리 방지 - 트랜잭션

상황

사용자가 앱에서 처방을 제출하면 POST /prescriptions/{id}/submit 이 호출됩니다. 서버는 내부 트랜잭션을 완료한 뒤 비동기로 보험심사 API 를 호출해 청구를 접수합니다. 모바일 네트워크 재전송, 사용자 재시도, 외부 보험사 웹훅의 중복/역순 도착으로 같은 처방이 두 번 이상 접수되는 사례가 가끔 발생합니다.

문제

1. "처방 1 건당 보험청구 1 회"를 **DB 불변식(제약)**과 Django 트랜잭션 경계만으로 보장하는 설계를 설명하시오. (고유 제약, 상태 전이 테이블, 멱등키 저장 위치 등을 서술할 것)

2. 동시성 경쟁(동일 처방 동시 제출, 워커 중복 실행)을 DB 락/ORM 패턴으로 어떻게 제어할지 서술하시오. (select_for_update 적용 지점, 낙관/비관 선택 근거 등을 서술하시오)

3. 외부 API 호출은 커밋 이후에만 수행되도록 트랜잭션-외부호출 분리 원칙과 실패 시 보상/재시도 정책(최대 시도, 지수 백오프, 중복 방지 키)을 제시하시오.

4. 사후 감사/대사(청구–처방 매칭)를 위해 무엇을 로그/저장해야 하는지 필드 단위로 적고, PHI/민감정보는 어떻게 취급할지 쓰시오.

헬스케어 앱에서 "처방 1 건당 보험청구 1 회"를 보장하려면, 네트워크 잡음이나 사용자 행동을 통제하려는 집착을 버리고 저장 계층의 불변식으로 문제를 정의해야 한다. 재전송·재시도·역순 웹훅은 언제나 발생한다. 중요한 것은 요청이 몇 번 왔느냐가 아니라 DB 에 무엇이 최종적으로 남느냐다. 이때 최종 심판은 애플리케이션의 if-문이 아니라 데이터베이스 제약과 트랜잭션 경계다.

불변식은 한 줄이다. "같은 처방에 대해 유효한 청구는 정확히 한 건." 이 문장을 스키마로 내리꽂으면 claims(prescription_id)에 UNIQUE 제약이 서고, 상태는 과거로 되돌아갈 수 없는 단조 전이만 허용된다. 실전에서는 Django 에서 transaction.atomic() 블록 안에서 삽입을 시도하고, 경합 시 발생하는 IntegrityError 를 '정상 흐름'으로 취급한다. 중복 삽입은 실패가 아니라 기대된 경로다. 애플리케이션은 해당 예외를 잡아 동일 자원에 대한 멱등 응답을 돌려주면 된다. 트랜잭션 격리수준은 대개 READ COMMITTED 면 충분하다. 팬텀이나 쓰기 스큐 우려가 있는 구간은 제약 + UPSERT 가 해결의 중심축이고, 락은 보조다. SERIALIZABLE 로 시스템 전체를 올리는 선택은 비용 대비 이득이 작다.

동시에 같은 처방을 처리하려고 달려드는 순간이 문제를 만든다. 존재하지 않는 행은 잠글 수 없으니, "잠그고 처리한다"만으로는 빈틈이 생긴다. 그래서 먼저 비즈니스 키 기반 UPSERT 가 필요하다. (prescription_id, op) 같은 키로 후보 레코드를 원자적으로 만들고, 그 다음 단계의 긴 처리 구간에서만 비관적 잠금(select_for_update) 으로 직렬화한다. 잠금은 서비스 전역에서 항상 같은 순서로 취득해야 데드락을 피한다. 여전히 경쟁이 과열되거나 레코드가 생기기 전 단계의 경합을 더 간단히 제어하고 싶다면 Advisory Lock 이 유용하다. 예컨대 Postgres 에서는 pg_advisory_xact_lock(hash(prescription_id))로 트랜잭션 범위에서만 유효한 경량 잠금을 걸어 존재 전 단계부터 경쟁을 줄일 수 있다. 정리하면, 삽입은 제약으로, 긴 처리만 잠금으로, 존재 전 경쟁은 UPSERT/Advisory Lock 으로 푼다.

외부 보험사 API 는 우리 트랜잭션과 무관하게 움직인다. 트랜잭션 안에서 외부를 호출했다가 내부는 롤백되고 외부는 커밋되는 순간, 부작용이 이중화된다. 따라서 트랜잭션-외부호출 분리가 원칙이다. 실무 해법은 아웃박스(Outbox) 패턴이다. 비즈니스 트랜잭션이 커밋될 때 "호출해야 할 사실"을 아웃박스 테이블에 기록하고, 별도의 디스패처가 이를 읽어 외부를 호출한다. 전달 채널은 규모에 따라 다르다. 간단히는 폴링 워커로 충분하고, 고트래픽·다중 서비스 환경에서는 CDC(Debezium 등)로 커밋 로그를 구독해 지연과 순서를 개선한다. 디스패처는 현실적으로 적어도 한 번(at-least-once) 호출한다. 이를 효과상 한 번(effectively-once) 으로 수렴시키려면 지수 백오프(예: 0.5s, 1s, 2s, …) + 지터, 최대 시도 횟수(예: 8~10 회), 그리고 멱등키(Idempotency-Key 또는 Correlation-ID) 가 필요하다. 멱등키는 외부가 지원하면 헤더로 넘기고, 미지원이면 내부에서 외부 응답의 고유 식별자와 우리 claim_id 를 매핑해 재수신 중복을 차단한다. 멱등키 저장에는 TTL 을 반드시 둔다(예: 24~72 시간). 무제한 보관은 저장소를 오염시키고 운영 리스크를 키운다. 재시도로도 실패가

누적되면 상태는 "실패-대기"로 전이시키고, 운영 경보와 함께 인간 개입 또는 보상 플로우(예: 청구 취소/정정)를 분리한다.

웹훅은 더 까다롭다. 중복·역순·유실을 가정하라. 해결은 멱등성과 버전으로 한다. 각 이벤트에는 이벤트 ID 와 버전/시퀀스를 붙이고, 처리 시 현재 상태의 버전과 비교한다. 낮은 버전은 드롭, 같은 버전은 멱등 처리, 높은 버전만 적용한다. 또한 보험사 서명 검증 결과와 원천 시각(ObservedAt), 처리 시각(ProcessedAt)을 함께 기록해 "늦게 도착했지만 더 최신인" 이벤트를 올바르게 반영할 근거를 마련한다. 이렇게 하면 역순 도착과 재전송에도 허용된 방향으로만 상태가 움직여 최종 일관성이 유지된다.

감사와 대사(청구–처방 매칭)는 사건의 재구성이 핵심이다. 무엇을 남겨야 하나. 내부 기준으로는 prescription_id, 내부 claim_id, 상태 전이 이력(상태·시각·주체), 멱등키, 트랜잭션/워크플로우 ID, 아웃박스 메시지 ID 와 시도 횟수·간격, 외부 호출의 요청/응답 요약 해시가 필요하다. 외부 기준으로는 보험사 external_claim_id(또는 상관키), 응답 코드·사유, 웹훅 딜리버리 ID, 서명 검증 결과를 남긴다. 이 필드들 위에 양방향 인덱스를 두고, "처방은 있는데 외부 없음", "외부는 있는데 내부 매칭 없음" 같은 불일치 뷰를 미리 정의하면 대사 작업이 단순해진다. 다만 로그는 개인정보보호법과 의료법의 민감정보 규정을 따른다. 진단명, 주민등록번호, 주소 등 민감정보는 기본적으로 로깅 금지이며, 비즈니스 필수 최소 범위만 가명처리/암호화된 별도 저장소에 두고 역할 기반 접근통제를 적용한다. 보존기간을 명시하고 만료 시 파기해야 한다. 일반 애플리케이션 텍스트 로그에는 PHI/민감정보를 남기지 말고, 항상 내부 참조 ID 나 해시된 토큰으로 대체하라.
운영은 설계만큼 중요하다. 모니터링 지표는 중복 차단률, IntegrityError 발생률, 아웃박스 대기열 길이·체류시간, 재시도 성공률, 웹훅 역순/중복 비율, 불일치 뷰의 항목 수가 기본 세트다. 경보 기준을 수치로 고정하면(예: 아웃박스 체류 p95>60 초, 재시도 실패율>1%) 사태를 조기에 잡는다. 배포와 마이그레이션도 안전해야 한다. 기존 데이터에 중복이 남아 있으면 UNIQUE 제약 추가가 실패하므로 정화·백필 → 제약 추가 → 코드 스위치의 순서를 밟는다. 스키마와 코드는 중간 상태에서 상호 호환되어야 하고, 피처 플래그로 신규 경로를 점진적으로 열며 지표를 관찰한다. 롤링 배포 중 두 버전이 공존해도 앞서의 제약·락·멱등 규칙이 경계를 지켜준다.

결국 "정확히 한 번 처리(exactly-once)"는 분산 시스템의 신기루다. 우리가 실현하는 것은 효과상 한 번(effectively-once) 이다. 입력에서는 멱등키로 같은 의도를 같은 결과에 귀속시키고, 가운데에서는 UNIQUE 제약과 단조 전이가 진실을 고정하며, 출력에서는 아웃박스와 재시도·멱등키가 외부 세계의 요동을 무해화한다. 이 조합이 있으면 재전송·연타·역순 이벤트 속에서도 결과는 한 번만 남고, 장애 이후에도 사건의 전말을 재구성할 수 있으며, 법적·윤리적 의무까지 충족한다. 이것이 헬스케어 백엔드에서 "이중 처리 방지"를 실무 수준으로 완성하는 배경지식의 전부다.

검사결과 수신 파이프라인 신뢰성 (중복·역순·버전 관리)

상황

외부 검사기관이 웹훅으로 검사결과를 전송합니다. 같은 결과가 중복으로 오거나, 업데이트가 역순으로 도착하기도 합니다. 서버는 웹훅을 받아 큐 작업으로 DB 에 반영하고, 완료되면 앱에 푸시 알림을 보냅니다. 간혹 이전 버전이 최신 결과를 덮어쓰는 사고가 발생합니다.

문제

1. 중복/역순에 안전하도록 DB 키/버전 전략(예: 외부 이벤트 ID·결과 버전·업데이트 시점)과 업서트 규칙을 설계하시오. 최신성 판단 기준과 덮어쓰기 방지 원칙을 명확히 쓰시오.

2. Django 에서 동기 핸들러→비동기 작업으로 넘어갈 때 멱등 처리를 보장하는 흐름을 설명하시오. (멱등키 저장 위치, 재시도 시 중복 반영 방지)

3. "DB 커밋 이전에 푸시 알림이 발송"되는 레이스를 막기 위한 커밋 후 부수효과 실행 원칙과, 실패 시 재시도/사후 검증(대사) 전략을 제시하시오.

4. 운영 관점에서 이 파이프라인의 관측 항목(메트릭/로그)을 최소 3 가지 고르고, 장애 분석을 쉽게 하기 위한 로그 스키마 핵심 필드를 적으시오. (예: trace/request/외부 이벤트 식별자 등)

검사결과 수신 파이프라인을 믿을 수 있게 만들려면, 네트워크나 외부 기관의 성격을 "예외"로 취급하지 말고 중복·역순·유실이 항상 발생한다는 전제로 모델링해야 한다. 따라서 핵심은 "요청이 몇 번 왔느냐"가 아니라 저장 계층에 무엇이 최종적으로 남느냐를 결정하는 규칙을 명확히 세우는 일이다. 이 규칙은 애플리케이션의 if-문이 아니라 DB 불변식(제약)과 상태 전이의 단조성으로 구현되어야 한다.

먼저 정체성(identity)과 버전(version) 이 필요하다. 외부 검사기관이 보낸 각 이벤트는 재전송될 수 있으므로, 이벤트 자체를 식별하는 외부 이벤트 ID 가 있어야 중복을 제거할 수 있다. 동시에, 같은 검사에 대한 여러 차례의 갱신이 시간 역전으로 도착하므로, "무엇이 최신인가"를 기계적으로 판정할 버전 스칼라가 필요하다. 이상적인 버전은 검사 단위로 단조 증가하는 정수(예: result_version)다. 외부가 시퀀스를 제공하지 못한다면, 최소한 원천 시각(ObservedAt) 을 받되 벤더 시계의 오차를 고려해 우리 시스템의 처리 시각(ProcessedAt) 을 별도로 기록하고, 최신성 판정은 (version 우선, 없으면 ObservedAt, 동률이면 ProcessedAt)처럼 결정적 타이브레이커를 둔다. "마지막으로 도착한 것이 최신"이라는 LWW(last-write-wins)는 역순 도착에서 쉽게 망가지므로 피한다.

이 정체성과 버전을 DB 로 끌어내리면 설계는 자연스럽다. 하나의 검사(예: lab_order_id 또는 sample_id)에 대해 결과를 반영하는 주 테이블과, 들어온 이벤트의 원본을 적재하는 인박스(inbox) 테이블이 분리된다. 인박스에는 (external_event_id UNIQUE)로 중복 수신을 차단하고, 주 테이블은 (subject_key UNIQUE)로 검사별 최신 스냅샷 1 행만 유지한다. 반영 규칙은 업서트(UPSERT) 이되, 단순 덮어쓰기가 아니라 WHERE incoming.version > current.version 같은 조건부 갱신이어야 한다. 버전이 없고 시각만 있을 때도 동일하게 엄격한 비교식으로 최신성 판정을 한다. 이렇게 하면 역순 도착은 무시, 중복은 NO-OP, 최신만 반영이라는 단조 전이가 DB 차원에서 보장된다.

웹훅 수신은 보통 동기 핸들러에서 비동기 작업 큐로 넘어간다. 이때 멱등성을 보장하려면 흐름에 키를 심고 저장해야 한다. 동기 핸들러는 요청을 검증하고, 인박스에 external_event_id 를 트랜잭션 내에서 기록한다(UNIQUE 제약으로 중복 차단). 그 다음에야 큐 작업을 커밋 이후에 발행한다. 비동기 작업은 인박스의 상태를 읽고 이미 처리되었는지 재확인한 뒤, 주 테이블에 조건부 업서트를 수행하고, 성공 시 인박스를 "적용됨"으로 갱신한다. 재시도나 워커 중복 실행은 인박스 UNIQUE 와 조건부 업서트에 부딪혀 무해한 중복 처리로 소거된다. 요점은 멱등키를 **메모리나 캐시가 아니라 영속 저장소(DB)**에 두는 것이다. 그래야 프로세스 재시작이나 다중 인스턴스 환경에서도 일관성이 유지된다.

부수효과(푸시 알림, 추가 파생 작업)는 DB 커밋 이전에 절대 실행되지 않도록 해야 한다. 커밋 전에 푸시를 쏘면 롤백 시 "없던 검사가 도착했다고 알림" 같은 레이스가 발생한다. 이를 막는 패턴이 트랜잭션-외부호출 분리다. 애플리케이션 트랜잭션 안에서는 오직 인박스 기록과 주 테이블 갱신만 수행하고, 커밋에 종속된 아웃박스(outbox) 에 "보낼 알림" 사실만 남긴다. 커밋이 끝난 뒤 별도

디스패처가 아웃박스를 소비해 푸시를 발송한다. 알림도 멱등해야 하므로, 알림 키(예: subject_key + version)를 남겨 같은 내용 중복 발송을 금지한다. 외부 채널 오류는 지수 백오프와 최대 시도 횟수로 재시도하고, 장시간 실패 시 대사 대상 큐로 옮겨 사후 검증에서 회수한다.

이 모든 규칙은 "정확히 한 번(exactly-once)"이 아니라 효과상 한 번(effectively-once) 을 목표로 한다. 입력단에서는 인박스 UNIQUE 로 같은 이벤트를 한 번만 인정하고, 핵심 반영은 버전 조건으로 항상 최신만 유지하며, 출력단에서는 아웃박스와 알림 멱등키로 부수효과의 중복을 무해화한다. 이렇게 양 끝을 멱등화하면, 네트워크 레벨에서 몇 번 재전송·역순이 벌어져도 결과는 한 번만 남는다.

관측 가능성은 운영의 생명줄이다. 파이프라인의 건강을 보려면 최소한 세 가지를 본다. 첫째, 중복 제거율과 역순 비율은 외부 품질과 우리 멱등 설계의 효과를 동시에 보여준다. 둘째, 수신→반영→알림까지의 종단 간 지연(p95/p99) 은 병목과 레이텐시 목표 준수를 드러낸다. 셋째, 인박스 체류 시간과 아웃박스 백로그 길이는 처리 적체를 조기 경보한다. 로그 스키마는 사건 재구성을 위해 trace_id/request_id, external_event_id, subject_key, incoming_version/observed_at/processed_at, apply_result(적용/중복/역순-드롭), previous_version→new_version 을 반드시 포함한다. 이렇게 남겨야 장애 시 "무엇이 언제 왜 반영/무시되었는지"를 한 번에 추적할 수 있다.

마지막으로 민감정보를 다루는 태도다. 검사결과에는 PHI/민감정보가 포함될 수 있으므로, 인박스와 로그에는 필요 최소의 메타데이터만 저장하고 결과 본문은 별도 보호영역에 암호화하여 둔다. 일반 텍스트 로그에는 결과 내용을 남기지 말고, 요약 해시나 내부 참조 ID 로 대체한다. 접근은 역할 기반으로 제한하고, 보존 기간을 명시해 만료 시 파기한다. 이는 기술적 선택 이전에 준수해야 할 기본 위생이다.

정리하면, 이 파이프라인의 신뢰성은 이벤트의 정체성(외부 이벤트 ID), 최신성 판정의 단조 버전 규칙, DB 차원의 조건부 업서트와 제약, 커밋 이후에만 실행되는 아웃박스 기반 부수효과, 그리고 이를 비추는 관측 지표와 사건 로그의 조합에서 나온다. 이 조합이 갖춰지면, 중복과 역순이 일상인 환경에서도 데이터는 항상 최신으로 수렴하고, 알림은 사실에만 반응하며, 장애 후에도 근거를 가지고 복구할 수 있다.