DB 스키마 설계서 작성법

최종 프로젝트 STEP 2 · PostgreSQL(Supabase) · pgvector · Redis

AI 애플리케이션 개발 과정 · 최종 프로젝트 산출물 가이드

`users 1:N incidents` `PRIMARY KEY` `VECTOR(1536)` `ROW LEVEL SECURITY`

DB 스키마 설계서는 데이터의 설계도다

API 설계 문서가 '주고받는 것'의 약속이라면, 이 문서는 '저장하는 것'의 약속이다

무엇을

저장할 데이터를 정한다

테이블과 컬럼, 타입, 제약을 정한다. API가 주고받는 필드가 어디에 저장되는지 보인다.

어떻게

테이블을 잇는다

기본키와 외래키로 관계를 정한다. 잘못된 데이터가 들어오지 못하게 DB가 막아 준다.

누가

볼 수 있는 사람을 정한다

RLS 정책으로 역할별 조회·수정 권한을 정한다. 보안 가드레일 문서의 근거가 된다.

API 설계 문서에서 이어지는 자리

두 문서는 STEP 2의 짝 산출물이다. 나란히 놓고 함께 쓴다

기획서

기능 요구사항
FR-01 …

→

API 설계 문서

지난 시간에
다룬 산출물

→

DB 스키마 설계서

오늘 다루는
산출물

→

에이전트 설계

메모리와
체크포인트 저장

→

통합 테스트

권한과 데이터
정합성 확인

API 요청과 응답의 필드가 테이블의 컬럼이 된다. 필드명과 컬럼명, ID 형식을 두 문서에서 똑같이 쓴다.

문서에 들어가는 여섯 가지

01

ERD와 관계 명세

테이블 사이의 관계를 그림과 표로

02

엔터티 정의

컬럼마다 타입·제약·의미

03

pgvector 스키마

문서·청크 테이블과 벡터 인덱스

04

RLS 정책

역할별 조회·등록·수정·삭제 권한

05

세션·캐시 설계

JWT와 Redis 세션의 수명주기

06

인덱스와 쿼리 최적화

자주 쓰는 조회의 인덱스와 검증 방법

테이블은 행과 열로 된 표다

엑셀 시트와 비슷하지만, 열마다 타입과 규칙이 정해져 있다는 점이 다르다

테이블 (엔터티)

대상 하나에 표 하나

사용자, 장애, 복구 작업처럼 서비스가 다루는 대상의 종류마다 테이블을 하나씩 만든다.

```
users
incidents
recovery_jobs
```

컬럼 (열)

속성과 타입, 제약

대상이 갖는 속성이다. 어떤 타입의 값이 들어가는지, 비워도 되는지를 함께 정한다.

```
status TEXT NOT NULL
```

행 (레코드)

실제 데이터 한 건

장애 한 건, 사용자 한 명이 행 하나다. API 응답의 JSON 객체 하나에 해당한다.

```
inc-001 | detected
```

키와 제약이 데이터를 지킨다

용어	뜻	예시
기본키 (PK)	행 하나를 유일하게 가리키는 컬럼	`incidents.id`
외래키 (FK)	다른 테이블의 기본키를 가리키는 컬럼	`recovery_jobs.incident_id → incidents.id`
UNIQUE	같은 값이 두 번 들어갈 수 없다	`users.email`
NOT NULL	값을 비워 둘 수 없다	`incidents.status`

외래키를 걸어 두면, 존재하지 않는 장애에 대한 복구 작업은 저장 자체가 거부된다. 검증을 코드가 아니라 DB가 해 준다.

테이블 사이의 관계는 세 종류다

ERD를 그리기 전에, 두 테이블이 몇 대 몇으로 짝지어지는지 먼저 말로 적어 본다

1 : N

가장 흔한 관계

사용자 한 명이 장애 여러 건을 등록한다. 외래키는 N 쪽 테이블에 둔다.

```
incidents.created_by
→ users.id
```

N : M

중간 테이블로 푼다

장애 하나에 태그 여러 개, 태그 하나에 장애 여러 건. 둘을 잇는 테이블을 따로 만든다.

```
incident_tags
(incident_id, tag_id)
```

1 : 1

드물게 쓴다

사용자와 프로필처럼 하나씩 짝지어진다. 한 테이블로 합칠 수 있는지 먼저 본다.

```
profiles.user_id
→ users.id (UNIQUE)
```

자주 쓰는 타입 여섯 가지

타입	쓰는 곳	예시
`UUID`	사용자, 문서 등의 식별자	`users.id`
`TEXT`	문자열. 이름, 상태, 본문	`incidents.status`
`INT`	개수, 순번	`document_chunks.chunk_index`
`TIMESTAMPTZ`	시각 (시간대 포함)	`created_at`
`JSONB`	구조가 자주 바뀌는 부가 정보	`documents.metadata`
`VECTOR(n)`	임베딩 벡터 (pgvector)	`document_chunks.embedding`

시각은 TIMESTAMPTZ로 통일한다. 시간대가 없는 타입을 섞으면 한국 시간과 9시간 어긋나는 오류가 생긴다.

API의 필드에서 테이블을 뽑아낸다

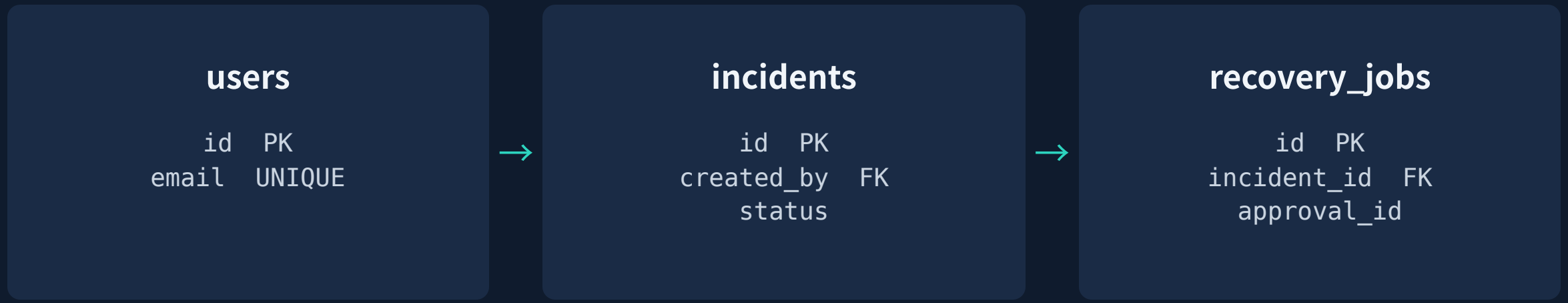
API 설계 문서의 요청·응답 JSON을 펼쳐 놓고, 필드마다 저장할 곳을 정한다

API 필드	테이블.컬럼 (예시)	비고
`incident_id`	`incidents.id`	경로 파라미터가 곧 기본키
`action, target_id`	`recovery_jobs.action, target_id`	요청 본문을 그대로 기록
`approval_id`	`recovery_jobs.approval_id`	NOT NULL. 승인 없이는 저장 불가
`job_id, status`	`recovery_jobs.id, status`	응답은 저장된 행을 돌려준 것
`JWT의  사용자`	`incidents.created_by → users.id`	누가 등록했는지 남긴다

예시는 산출물 샘플(InfraCare AI, 교육용 가상 서비스) 기준이다. 팀 주제의 대상으로 바꿔 쓴다.

ERD는 테이블을 상자로, 관계를 선으로 그린다

화살표 하나가 1:N 관계 하나다. 상자 안에는 키 컬럼을 반드시 적는다



그림은 Mermaid erDiagram이나 ERD 도구로 그려서 붙인다. 손으로 그린 그림보다 수정과 버전 관리가 쉽다.

관계 명세 표로 그림을 글로 확인한다

부모 테이블	관계	자식 테이블	연결 키
users	1:N	incidents	incidents.created_by → users.id
incidents	1:N	recovery_jobs	recovery_jobs.incident_id → incidents.id
documents	1:N	document_chunks	document_chunks.document_id → documents.id
users	1:N	sessions	sessions.user_id → users.id

ERD의 선 개수와 이 표의 행 수가 같아야 한다. 다르면 둘 중 하나가 빠진 것이다.

엔터티 정의는 컬럼마다 네 칸을 채운다

테이블	컬럼	타입 / 제약	의미
users	id	UUID / PK	사용자 ID
users	email	TEXT / UNIQUE	로그인 이메일
incidents	id	TEXT / PK	장애 ID
incidents	created_by	UUID / FK users.id	등록한 사용자
incidents	status	TEXT / NOT NULL	장애 상태
recovery_jobs	incident_id	TEXT / FK incidents.id	대상 장애

status처럼 값이 정해진 컬럼은 허용 값도 적는다: detected → diagnosing → ⋯ → resolved
모든 테이블에 created_at TIMESTAMPTZ를 둔다.

설계서의 표는 그대로 SQL이 된다

- 엔터티 정의 표의 한 행이 SQL의 한 줄이다.
- REFERENCES가 외래키다.
- DEFAULT now()로 생성 시각을 자동 기록한다.
- 문서에 싣는 SQL은 Supabase SQL Editor에서 실제로 실행해 본 것만 넣는다.

schema.sql

```
CREATE TABLE incidents (
  id          TEXT PRIMARY KEY,
  created_by  UUID NOT NULL REFERENCES users(id),
  status      TEXT NOT NULL DEFAULT 'detected',
  created_at  TIMESTAMPTZ NOT NULL DEFAULT now()
);

CREATE TABLE recovery_jobs (
  id           TEXT PRIMARY KEY,
  incident_id  TEXT NOT NULL REFERENCES incidents(id),
  action       TEXT NOT NULL,
  approval_id  TEXT NOT NULL,
  status       TEXT NOT NULL DEFAULT 'queued',
  created_at   TIMESTAMPTZ NOT NULL DEFAULT now()
);
```

pgvector는 문서와 청크를 나눠 저장한다

RAG에 쓰는 지식 문서는 원문 정보와, 잘게 자른 조각(청크)의 벡터로 나뉜다

documents

원문의 정보

제목, 출처, 부가 정보를 저장한다. 청크가 어느 문서에서 나왔는지 추적하는 기준이다.

```
documents
(title, source_uri)
```

document_chunks

조각과 벡터

잘라 낸 본문과 임베딩 벡터, 문서 안에서의 순번을 저장한다. 검색은 이 테이블에서 일어난다.

```
document_chunks
(content, embedding)
```

모델 정보

어떤 모델로 만들었나

모델을 바꾸면 벡터의 차원과 의미가 달라진다. 기존 벡터와 섞어 쓸 수 없으므로 기록해 둔다.

```
embedding_model
embedding_version
```

벡터 컬럼과 인덱스를 명세한다

- VECTOR(n)의 n은 임베딩 모델의 출력 차원과 같아야 한다.
- 인덱스의 연산자(vector_cosine_ops)와 검색식의 연산자(<=>)를 맞춘다.
- UNIQUE (document_id, chunk_index)로 같은 조각이 두 번 적재되지 않게 한다.

pgvector 스키마

```
CREATE TABLE document_chunks (
  id           UUID PRIMARY KEY,
  document_id  UUID NOT NULL REFERENCES documents(id),
  chunk_index  INT  NOT NULL,
  content      TEXT NOT NULL,
  embedding    VECTOR(1536),
  UNIQUE (document_id, chunk_index)
);

CREATE INDEX idx_chunks_embedding
  ON document_chunks
  USING hnsw (embedding vector_cosine_ops);

-- 검색: 코사인 거리가 가까운 순서로 5건
SELECT content FROM document_chunks
ORDER BY embedding <=> :query_vec LIMIT 5;
```

RLS 정책은 역할과 명령의 표로 적는다

RLS(Row Level Security)는 행 단위 권한이다. 같은 테이블이라도 사용자마다 보이는 행이 다르다

테이블	SELECT	INSERT	UPDATE	DELETE
`incidents`	viewer는 본인 것, 운영자는 전체	operator / admin	operator / admin	admin
`recovery_jobs`	operator / admin	복구 서비스 전용	복구 서비스 전용	금지
`documents`	허용 그룹의 로그인 사용자	admin	admin	admin
`document_chunks`	문서의 접근 권한을 따름	관리 서비스	관리 서비스	관리 서비스

이 표는 API 설계 문서의 403 조건, 보안 가드레일 문서의 권한표와 같은 내용이어야 한다.

RLS 표는 정책 SQL로 옮긴다

- RLS를 켜면 정책에 없는 접근은 모두 거부된다.
- auth.uid()는 요청에 실린 JWT의 사용자 ID다.
- service_role 키는 RLS를 건너뛴다. 서버에만 두고 브라우저에 노출하지 않는다.
- 다른 사용자의 토큰으로 조회해 0건이 나오는지 테스트한다.

Supabase RLS (형식 예시)

```
ALTER TABLE incidents ENABLE ROW LEVEL SECURITY;

-- 본인이 등록한 장애만 조회
CREATE POLICY incidents_select_own
  ON incidents FOR SELECT
  TO authenticated
  USING (created_by = auth.uid());

-- 자기 이름으로만 등록
CREATE POLICY incidents_insert_own
  ON incidents FOR INSERT
  TO authenticated
  WITH CHECK (created_by = auth.uid());
```

JWT와 Redis 세션의 수명주기를 적는다

PostgreSQL 밖에 저장하는 데이터도 스키마다. 키 이름 규칙과 만료 시간을 문서에 남긴다

① 발급
Supabase Auth가 JWT 발급

→

② 검증
서명·만료·역할 확인

→

③ 저장
`session:{id}`
TTL 30분

→

④ 갱신
요청마다 TTL 갱신 정책 적용

→

⑤ 폐기
로그아웃·만료 시 세션 삭제

JWT의 만료 시간과 Redis의 TTL은 서로 다른 값이다. 둘 다 적고, 어느 쪽이 먼저 끝나면 어떻게 되는지도 적는다.

인덱스는 자주 쓰는 조회에서 출발한다

API 목록의 GET부터 본다

어떤 조건으로 찾고 정렬하는가.

```
GET /incidents?status=open
```

조건과 정렬을 묶어 만든다

필터 컬럼 먼저, 정렬 컬럼 다음.

```
INDEX incidents (status, created_at DESC)
```

실행 계획으로 확인한다

인덱스 전후 결과를 문서에 싣는다.

```
EXPLAIN ANALYZE SELECT …
```

벡터 인덱스는 짝을 맞춘다

차원, 연산자, 검색식이 같아야 한다.

```
vector_cosine_ops  ↔  <=>
```

다른 산출물과 같은 이름을 쓴다

DB 스키마는 거의 모든 산출물이 참조한다. 이름이 어긋나면 전체가 어긋난다

API 설계 문서

필드명과 컬럼명을 맞춘다

incident_id, approval_id와 ID 형식을 똑같이 쓴다.

에이전트 설계서

메모리가 저장되는 곳

체크포인트와 장기 메모리(pgvector)의 테이블을 밝힌다.

AI 보안 가드레일 문서

RLS 표가 권한표다

역할별 권한이 두 문서에서 같아야 한다.

통합 테스트 결과서

차단되는지 확인한다

권한 없는 조회가 막히는 것이 테스트 케이스다.

자주 나오는 실수와 고치는 법

실수	이렇게 고친다
같은 컬럼이 문서 안에서 이름이 다르다 (created_by 와 owner_id)	엔터티 정의 표를 기준으로 삼아 ERD와 관계 표를 맞춘다
ID 형식이 API 문서와 다르다 (UUID라고 적고 예시는 inc-001)	형식을 하나로 정하고 두 문서를 함께 고친다
ERD 그림만 있고 제약이 없다	PK·FK·NOT NULL·UNIQUE를 컬럼마다 적는다
임베딩 모델의 차원과 VECTOR(n)이 다르다	모델 이름과 차원을 컬럼 옆에 함께 기록한다
RLS를 켜지 않고 service_role로만 접근한다	테이블마다 RLS 표를 만들고 차단 테스트를 남긴다

제출 전 체크리스트

- [ ] ERD 그림과 관계 명세 표가 서로 일치한다
- [ ] 벡터 차원, 인덱스 연산자, 임베딩 모델이 적혀 있다
- [ ] 모든 컬럼에 타입·제약·의미가 적혀 있다
- [ ] 테이블마다 역할별 RLS 표가 있다
- [ ] 모든 테이블에 기본키와 created_at이 있다
- [ ] Redis 키 이름 규칙과 TTL이 적혀 있다
- [ ] API 설계 문서와 필드명·ID 형식이 같다
- [ ] 문서의 SQL을 실제로 실행해 확인했다

오늘 팀에서 할 일

API 설계 문서를 옆에 펼쳐 놓고 시작한다

① 테이블 목록

API 필드에서
저장할 대상 뽑기

→

② ERD와 관계 표

키 컬럼과
1:N 관계 그리기

→

③ 엔터티 정의

핵심 테이블의
SQL 실행까지

→

④ RLS 표 초안

역할별 권한을
팀이 함께 정하기

문서 머리말에 문서 버전과 작성 상태를 적는다. 예: 문서 버전 v0.1 | 작성 상태: 초안