supabase -

목차

> 1 edge-function를 사용한 CRUD humans

humans테이블에 대한 crud작업.

user-id FK null 값처리하는 문제.

어떻게 해결했는지.

>2 트랜잭션

Supabase는 기본적으로 **PostgreSQL**을 백엔드로 사용합니다. 그래서 트랜잭션(여러 쿼리를 하나로 묶어 "모두 성공하거나, 하나라도 실패하면 모두 롤백"하는 것)은 **PostgreSQL** 트랜잭션 기능을 그대로 활용합니다.

Supabase에서 트랜잭션을 처리하는 방법은 크게 2가지입니다:

---

1. RPC(Stored Procedure)를 이용한 트랜잭션

Supabase는 서버리스이기 때문에 클라이언트 단에서 트랜잭션을 직접 다루는 것은 안 됩니다. 대신 **PostgreSQL Stored Procedure (함수)**를 만들어서 트랜잭션을 관리할 수 있어요.

예시:

sql
복사편집

```
-- SQL에서 트랜잭션을 제어하는 Stored Procedure 만들기
create or replace function transfer_money(from_account_id int,
to_account_id int, amount numeric)
returns void
language plpgsql
as $$
begin
  -- 트랜잭션 시작
  update accounts set balance = balance - amount where id =
from_account_id;

  update accounts set balance = balance + amount where id =
to_account_id;

  -- 여기서 오류가 발생하면 자동 롤백
end;
$$;
```

이걸 만든 다음에, Supabase 클라이언트 코드에서는 이렇게 호출합니다:

javascript
복사편집

```
const { data, error } = await supabase
  .rpc('transfer_money', {
    from_account_id: 1,
    to_account_id: 2,
    amount: 100
  });
```

ai를 이용해서 코드를 완성해서 제출해 보자.

>3 조인

SSupabase 테이블 조인 이해하기 (2개 테이블로 완전 정리)

아래에서는 `users`와 `posts` 테이블만 사용하여 모든 조인 유형을 명확히 설명드리겠습니다.

테이블 구조 예시

1. users 테이블

```
javascript

Copy

Download
```

```
[
  { "id": "user1", "name": "김철수", "email": "kim@example.com" },
  { "id": "user2", "name": "박영희", "email": "park@example.com" }
]
```

2. posts 테이블

```
javascript

Copy

Download
```

```
[
  { "id": "post1", "title": "첫 글", "content": "안녕하세요", "user_id": "user1" },
  { "id": "post2", "title": "두번째", "content": "반갑습니다", "user_id": "user1" },
  { "id": "post3", "title": "공지", "content": "중요합니다", "user_id": "user2" }
]
```

1. 기본 조인 (1:1 관계처럼 보이는 경우)

쿼리

```
javascript

Copy

Download
```

```
const { data } = await supabase
  .from('posts')
  .select(`
    id,
    title,
    users(id, name)
  `);
```

결과 (게시글별 작성자 정보)

```
javascript
Copy
Download
[
  {
    "id": "post1",
    "title": "첫 글",
    "users": { "id": "user1", "name": "김철수" } // 단일 객체
```

```
  },
  {
    "id": "post2",
    "title": "두번째",
    "users": { "id": "user1", "name": "김철수" }
  },
  {
    "id": "post3",
    "title": "공지",
    "users": { "id": "user2", "name": "박영희" }
  }
]
```

2. 반대 방향 조인 (1:N 관계)

쿼리

```
javascript

Copy

Download
```

```
const { data } = await supabase
```

```
  .from('users')
  .select(`
    id,
    name,
    posts(id, title)
  `);
```

결과 (사용자별 게시글 목록)

```
javascript
Copy
Download
```

```
[
  {
    "id": "user1",
    "name": "김철수",
    "posts": [ // 배열로 반환 (1:N 관계)
      { "id": "post1", "title": "첫 글" },
      { "id": "post2", "title": "두번째" }
```

```
    ]
  },
  {
    "id": "user2",
    "name": "박영희",
    "posts": [
      { "id": "post3", "title": "공지" }
    ]
  }
]
```

3. 필터링이 추가된 조인

쿼리 (김철수 사용자의 게시글만)

```
javascript

Copy

Download
```

```
const { data } = await supabase
  .from('users')
  .select(`
```

```
    id,
    name,
    posts!inner(id, title)
  `)
  .eq('name', '김철수');
```

결과

```
javascript
Copy
Download
```

```
[
  {
    "id": "user1",
    "name": "김철수",
    "posts": [
      { "id": "post1", "title": "첫 글" },
      { "id": "post2", "title": "두번째" }
    ]
```

```
  }
]
```

4. 별칭(Alias) 사용 예제

쿼리

```
javascript

Copy

Download
```

```
const { data } = await supabase

  .from('posts')

  .select(`

    id,

    title,

    author:users(name, email)

  `);
```

결과

```
javascript

Copy

Download
```

```
[
```

```
  {
    "id": "post1",
    "title": "첫 글",
    "author": { "name": "김철수", "email": "kim@example.com" }
  },
  // ... (생략)
]
```

5. 모든 컬럼 선택하기

쿼리

```
javascript

Copy

Download
```

```
const { data } = await supabase
  .from('posts')
  .select(`
    *,
    users(*)
  `);
```

결과 (일부 생략)

```
javascript

Copy

Download
```


```
[
  {
    "id": "post1",
    "title": "첫 글",
    "content": "안녕하세요",
    "user_id": "user1",
    "users": {
      "id": "user1",
      "name": "김철수",
      "email": "kim@example.com"
    }
  },
  // ... (생략)
```

```
]
```

핵심 정리 표

조인 유형	쿼리 예시	결과 형태	설명
1:1 조인	`.select('id, users(name)')`	`users: {객체}`	게시글 → 작성자
1:N 조인	`.select('id, posts(title)')`	`posts: [배열]`	사용자 → 게시글들
필터 조인	`.select('..., posts!inner(...)')`	조건 만족하는 것만	`!inner` 사용
별칭 사용	`.select('..., author:users(...)')`	필드명 변경	가독성 향상
전체 선택	`.select('*, users(*)')`	모든 컬럼 포함	성능 주의

이렇게 동일한 2개 테이블로 모든 경우를 정리해보았습니다. 어떤 부분이 추가로 이해가 필요하신가요?

Supabase에서의 조인 타입과 NULL 처리

Supabase는 PostgreSQL의 조인 기능을 활용하며, 기본적으로 **LEFT JOIN** 방식으로 동작합니다. 하지만 **INNER JOIN**도 사용할 수 있습니다.

1. 기본 조인 방식 (LEFT JOIN)

javascript

Copy

Download

```javascript
const { data } = await supabase
  .from('posts')
  .select(`
```

```
    id,
    title,
    users(name)
  `);
```

- 특징:
  - 왼쪽 테이블(posts)의 모든 레코드를 가져옴
  - 오른쪽 테이블(users)에 매칭되는 레코드가 없으면 NULL로 표시됨
  - 매칭되지 않는 경우에도 posts는 결과에 포함됨

2. INNER JOIN 사용 방법

javascript

Copy

Download

```
const { data } = await supabase
  .from('posts')
  .select(`
    id,
    title,
    users!inner(name)
  `);
```

- 특징:
  - `!inner` 키워드 사용
  - 양쪽 테이블에 모두 존재하는 레코드만 반환
  - 매칭되지 않는 레코드는 결과에서 제외됨

3. NULL 처리 예시

LEFT JOIN 결과 (기본):

json

Copy

Download

```
[
  {
```

```
    "id": "post1",
    "title": "존재하는 글",
    "users": { "name": "김철수" }
  },
  {
    "id": "post99",
    "title": "작성자 없는 글",
    "users": null  // 매칭되는 사용자 없음
  }
]
```

INNER JOIN 결과:

json

Copy

Download

```
[
  {
    "id": "post1",
    "title": "존재하는 글",
    "users": { "name": "김철수" }
  }
  // 작성자 없는 글(post99)은 결과에 포함되지 않음
]
```

4. 실제 사용 시나리오

INNER JOIN이 유용한 경우:

javascript

Copy

Download

```
// 작성자가 있는 게시글만 조회
const { data } = await supabase
  .from('posts')
  .select(`
    id,
```

```
    title,
    users!inner(id, name)
  `)
  .eq('users.is_active', true);  // 활성 사용자만
```

LEFT JOIN이 유용한 경우:

```
javascript
Copy
Download
// 모든 게시글 조회 (작성자 없는 글도 포함)
const { data } = await supabase
  .from('posts')
  .select(`
    id,
    title,
    users(name)
  `);
```

5. 요약 비교표

특징	**LEFT JOIN** (기본)	**INNER JOIN** (!inner)
매칭되지 않을 때	왼쪽 테이블 데이터는 포함, 오른쪽은 NULL	양쪽 모두 제외
Supabase 문법	`.select('..., users(name)')`	`.select('..., users!inner(name)')`
사용 사례	모든 게시글 보기 (작성자 유무 관계없이)	작성자가 확실히 있는 게시글만 필터링
NULL 처리	NULL 허용	NULL 발생하지 않음

Supabase에서는 명시적으로 `!inner`를 사용해야 INNER JOIN이 적용됩니다. 기본값이 LEFT JOIN이므로 NULL 처리가 필요할 때는 별도의 조건 체크가 필요합니다.

>4 슈퍼베이스에서 컬럼 자동증가

Supabase는 내부적으로 **PostgreSQL**을 사용하기 때문에, 자동 증가 컬럼(일명 "Auto Increment Column")을 만드는 방법도 **PostgreSQL** 방식을 그대로 따릅니다.

간단히 말하면, 컬럼 타입을 SERIAL이나 BIGSERIAL로 지정하면 됩니다.
또는 새 방식인 GENERATED BY DEFAULT AS IDENTITY 문법도 사용할 수 있습니다.

---

1. SERIAL 사용 (전통적인 방법)

sql
복사편집

```
create table profiles (
  id serial primary key,
  name text
);
```

- id 컬럼이 자동으로 1씩 증가합니다.

- primary key 제약 조건까지 같이 설정하는 것이 일반적입니다.

---

2. GENERATED BY DEFAULT AS IDENTITY 사용 (PostgreSQL 최신 방식)

sql
복사편집

```
create table profiles (
  id integer generated by default as identity primary key,
  name text
);
```

- PostgreSQL 10 이상에서는 이 방식이 권장됩니다.
- 나중에 시퀀스를 더 유연하게 조정할 수 있습니다.

---

3. Supabase GUI (콘솔)로 생성할 때

Supabase Studio(웹 대시보드)에서도 테이블 만들기 → 칼럼 추가 → 타입을 **int8**(또는 **int4**)로 선택하고, **"Default Value"**에 **gen_random_uuid()** 같은 게 아닌 경우:

- 숫자 타입(int4, int8)으로 만들고
- **Identity** 옵션을 체크하거나,
- Default 값을 nextval('시퀀스명') 식으로 직접 설정할 수도 있습니다. (대부분 자동 처리됩니다)

요약

방법	특징
`SERIAL`	간단하지만 나중에 커스터마이징이 힘듦
`GENERATED AS IDENTITY`	더 명시적이고, 권장되는 최신 방법
Supabase GUI	int형 선택 + Identity 활성화

추가로:

Supabase에서는 보통 UUID(`uuid_generate_v4()`)를 기본 키로 많이 쓰기도 합니다. (`id uuid default uuid_generate_v4() primary key`)

만약 숫자 기반 ID를 쓰고 싶다면 위 방법대로 하면 됩니다.

필요하면 Supabase 콘솔에서 클릭하는 방법도 캡처해서 보여드릴게요.
필요하신가요? 👀

(추가로 "ALTER TABLE"로 기존 테이블에 추가하는 방법도 설명해드릴 수 있습니다.)

추가로 어떤 상황(신규 생성, 기존 테이블 수정, PK 설정 등)이신가요?

>5 배열처리

Supabase 배열(Array) 컬럼 완전 초보자 가이드

아래는 `arraytest` 테이블을 사용한 배열 컬럼의 기초적인 사용법을 단계별로 설명한 것입니다.

1. 테이블 생성 (이미 생성하셨다면 넘어가세요)

sql

Copy

Download

```sql
-- SQL로 테이블 생성
CREATE TABLE arraytest (
  id SERIAL PRIMARY KEY,
  textarray TEXT[]  -- 문자열 배열 컬럼
);
```

2. 데이터 삽입 방법

SQL로 삽입

```sql
-- 단일 행 삽입
INSERT INTO arraytest (textarray)
VALUES (ARRAY['Harry', 'Larry', 'Moe']);

-- 여러 행 한 번에 삽입
INSERT INTO arraytest (textarray)
VALUES
  (ARRAY['Alice', 'Bob']),
  (ARRAY['Charlie']),
  (ARRAY['David', 'Eve', 'Frank']);
```

JavaScript로 삽입

```javascript
// 단일 행 삽입
const { data, error } = await supabase
  .from('arraytest')
  .insert({
    textarray: ['Harry', 'Larry', 'Moe']
  });

// 여러 행 한 번에 삽입
const { data, error } = await supabase
  .from('arraytest')
  .insert([
    { textarray: ['Alice', 'Bob'] },
    { textarray: ['Charlie'] },
    { textarray: ['David', 'Eve', 'Frank'] }
  ]);
```

3. 데이터 조회 방법

모든 데이터 가져오기

javascript

Copy

Download

```javascript
const { data, error } = await supabase
  .from('arraytest')
  .select('*');

// 결과 예시:
// [
//   { id: 1, textarray: ['Harry', 'Larry', 'Moe'] },
//   { id: 2, textarray: ['Alice', 'Bob'] },
//   ...
// ]
```

특정 배열 요소만 선택

javascript

Copy

Download

```javascript
// 첫 번째 요소만 선택 (인덱스는 1부터 시작)
const { data, error } = await supabase
  .from('arraytest')
  .select('id, textarray[1] as first_name');
```

4. 배열 필터링 (검색)

특정 값이 포함된 행 찾기

javascript

Copy

Download

```javascript
// 'Larry'가 포함된 행 찾기
const { data, error } = await supabase
```

```
  .from('arraytest')
  .select('*')
  .contains('textarray', ['Larry']);
```

배열 길이로 필터링

javascript

Copy

Download

```
// 배열 요소가 3개인 행 찾기
const { data, error } = await supabase
  .from('arraytest')
  .select('*')
  .filter('textarray', 'length', 'eq', 3);
```

5. 데이터 업데이트

전체 배열 변경

javascript

Copy

Download

```
await supabase
  .from('arraytest')
  .update({ textarray: ['New', 'Names'] })
  .eq('id', 1);
```

특정 요소만 변경

javascript

Copy

Download

```
// 두 번째 요소 변경 (인덱스 2)
await supabase
  .from('arraytest')
  .update({ 'textarray[2]': 'Lauren' })
  .eq('id', 1);
```

6. 배열에 요소 추가/제거

요소 추가

javascript

Copy

Download

```javascript
// 기존 배열 가져오기
const { data: row } = await supabase
  .from('arraytest')
  .select('textarray')
  .eq('id', 1)
  .single();

// 새 요소 추가
const updatedArray = [...row.textarray, 'New Person'];

// 업데이트
await supabase
  .from('arraytest')
  .update({ textarray: updatedArray })
  .eq('id', 1);
```

요소 제거

javascript

Copy

Download

```javascript
// 'Larry' 제외한 새 배열 생성
const filteredArray = row.textarray.filter(name => name !== 'Larry');

// 업데이트
await supabase
  .from('arraytest')
  .update({ textarray: filteredArray })
  .eq('id', 1);
```

7. 실용적인 예제: 간단한 주소록

javascript

Copy

Download

```javascript
// 주소록 테이블 생성
await supabase.rpc(`
  CREATE TABLE IF NOT EXISTS address_book (
    id SERIAL PRIMARY KEY,
    name TEXT,
    phone_numbers TEXT[],
    emails TEXT[]
  );
`);

// 데이터 삽입
await supabase
  .from('address_book')
  .insert({
    name: '김철수',
    phone_numbers: ['010-1234-5678', '02-987-6543'],
    emails: ['chulsoo@example.com', 'chulsoo.work@example.com']
  });

// 특정 이메일로 검색
const { data } = await supabase
  .from('address_book')
  .select('name, phone_numbers')
  .contains('emails', ['chulsoo@example.com']);
```

이제 배열 컬럼의 기본적인 사용법을 이해하셨을 것입니다! 😊 더 궁금한 점이 있으면 언제든 질문해주세요.

>6 객체 배열처

Supabase에서 객체 배열(JSONB) 처리 완벽 가이드

Supabase에서 객체 배열을 처리할 때는 일반적으로 `JSONB` 데이터 타입을 사용합니다. 이는 PostgreSQL의 강력한 JSON 지원 기능을 기반으로 합니다.

1. 테이블 생성 (객체 배열 컬럼 정의)

sql

Copy

Download

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  name TEXT,
  contacts JSONB,  -- 객체 배열을 저장할 컬럼
  preferences JSONB
);
```

2. 데이터 삽입 방법

SQL로 삽입

sql

Copy

Download

```
INSERT INTO users (name, contacts)
VALUES (
  '김철수',
  '[{"type": "phone", "value": "010-1234-5678"}, {"type": "email", "value": "chulsoo@example.com"}]'
);
```

JavaScript 클라이언트로 삽입

javascript

Copy

Download

```
const { data, error } = await supabase
  .from('users')
  .insert({
    name: '이영희',
    contacts: [
      { type: 'phone', value: '010-8765-4321', primary: true },
      { type: 'email', value: 'younghee@example.com' }
    ],
    preferences: {
      theme: 'dark',
      notifications: { email: true, sms: false }
    }
  });
```

3. 객체 배열 조회하기

전체 데이터 가져오기

javascript

Copy

Download

```
const { data, error } = await supabase
  .from('users')
  .select('*');
```

특정 필드만 선택

```javascript
Copy

Download
const { data } = await supabase
  .from('users')
  .select('name, contacts->type, contacts->value');
```

4. 객체 배열 검색하기

기본 검색

```javascript
Copy

Download
// 'phone' 타입의 연락처가 있는 사용자 찾기
const { data } = await supabase
  .from('users')
  .select('*')
  .contains('contacts', [{ type: 'phone' }]);
```

심화 검색 (JSON Path 사용)

```javascript
Copy

Download
// 기본 전화번호가 있는 사용자 찾기
const { data } = await supabase
  .from('users')
  .select('*')
  .contains('contacts', [{ primary: true }]);
```

5. 객체 배열 수정하기

전체 배열 교체

```javascript
```

Copy

Download

```
await supabase
  .from('users')
  .update({
    contacts: [
      { type: 'phone', value: '010-9999-8888' },
      { type: 'kakao', value: 'younghee_kakao' }
    ]
  })
  .eq('id', 2);
```

특정 요소 수정 (JSON Path)

javascript

Copy

Download

```
// 첫 번째 연락처의 값 변경
await supabase
  .from('users')
  .update({
    'contacts[0].value': '010-1111-2222'
  })
  .eq('id', 2);
```

6. 배열에 객체 추가하기

javascript

Copy

Download

```
// 기존 배열 가져오기
const { data: user } = await supabase
  .from('users')
  .select('contacts')
  .eq('id', 2)
  .single();
```

```javascript
// 새 객체 추가
const updatedContacts = [
  ...user.contacts,
  { type: 'telegram', value: 'younghee_tg' }
];

// 업데이트
await supabase
  .from('users')
  .update({ contacts: updatedContacts })
  .eq('id', 2);
```

7. 객체 배열에서 요소 제거하기

javascript

Copy

Download

```javascript
// 특정 타입의 연락처 제거 (필터링)
const filteredContacts = user.contacts.filter(
  contact => contact.type !== 'phone'
);

await supabase
  .from('users')
  .update({ contacts: filteredContacts })
  .eq('id', 2);
```

8. 실전 예제: 주문 시스템

javascript

Copy

Download

```javascript
// 주문 테이블 생성
await supabase.rpc(`
  CREATE TABLE IF NOT EXISTS orders (
    id SERIAL PRIMARY KEY,
```

```
    user_id INTEGER REFERENCES users(id),
    items JSONB,  -- 상품 객체 배열
    status TEXT
  );
`);

// 주문 데이터 삽입
await supabase
  .from('orders')
  .insert({
    user_id: 1,
    items: [
      { product_id: 101, name: 'Supabase T-Shirt', quantity: 2, price: 25000 },
      { product_id: 205, name: 'Supabase Stickers', quantity: 5, price: 3000 }
    ],
    status: 'pending'
  });

// 특정 상품이 포함된 주문 검색
const { data } = await supabase
  .from('orders')
  .select('*')
  .contains('items', [{ product_id: 101 }]);
```

성능 최적화 팁

1. **GIN** 인덱스 생성: 자주 검색하는 JSONB 컬럼에 인덱스 추가

```
sql
Copy
Download
CREATE INDEX idx_users_contacts ON users USING GIN(contacts);
```

6. 부분 업데이트: 큰 JSON 객체는 전체가 아닌 필요한 부분만 업데이트

```
javascript
Copy
Download
await supabase
  .from('users')
```

```
  .update({ 'preferences.theme': 'light' })
```

10. `  .eq('id', 2);`
11. 중첩 제한: 너무 깊은 중첩 구조는 별도 테이블로 분리 고려
12. **RLS** 설정: 보안을 위해 Row Level Security 필수 적용

```
sql
Copy
Download
-- RLS 예시
ALTER TABLE users ENABLE ROW LEVEL SECURITY;
CREATE POLICY user_access_policy ON users
  USING (auth.uid() = id::text);
```

이제 Supabase에서 객체 배열을 자유자재로 다룰 수 있을 것입니다! 😊

]

>7 supabase 로컬 설치

최재현님

>8 이메일 전송

최재현님

>8 기타 주제
