

# supabase -

## 목차

---

> 1. supabase 소개	2
> 2. 회원가입	4
> 3. 프로젝트 생성	5
> 4. 관계형 데이터베이스 소개	10
> 5. supabase 전체 구조 개요	14
> 6. 백업과 삭제 방법	17
> 7. supabase 소개	1
> 8. supabase 소개	1
> 8. 테이블 만들어 조작해 보기 supabase로 변경	1
> 9. 권한	1
> 10. 권한	1
>4/10일	1
> 11. supabase 소개	1
> 1. supabase 소개	1
> 1. supabase 소개	1
> 1. supabase 소개	1

---

## > 1. supabase 소개

---

슈퍼베이스(Supabase)는 실시간 데이터베이스, 인증, 스토리지 등을 제공하는 오픈소스 **BaaS(Backend-as-a-Service)** 플랫폼으로, **Firebase**의 대안으로 주목받고 있습니다

**BaaS(Backend-as-a-Service)**는 개발자가 서버 사이드 로직, 데이터베이스 운영, 인증·권한 관리, 파일 저장, 실시간 통신 등 백엔드(서버) 측 기능을 직접 구현·운영하지 않고도, 클라우드로 제공되는 **API**와 관리형 서비스를 통해 손쉽게 이용할 수 있도록 해주는 서비스입니다.

한마디로, 별도의 백엔드 코딩 없이도 디자이너나 비개발자도 **GUI**나 **API** 호출만으로 백엔드 기능이 포함된 웹 애플리케이션을 빠르게 만들 수 있다는 이야기이다.

**Firebase**는 **Google**에서 제공하는 **BaaS(Backend-as-a-Service)** 플랫폼이고 슈퍼베이스는 오픈소스다.

**Firebase**는 **Google**이 제공하는 **BaaS** 플랫폼이며, **Supabase**는 이를 오픈소스 방식으로 구현한 대안입니다.

**Firebase**는 구글이 제공하는 클라우드 플랫폼에서만 사용할 수 있지만, **Supabase**는 웹 대시보드 외에도 도커(**Docker**)를 이용해 로컬 환경에 설치하여 직접 운영할 수 있습니다.

**Supabase**는 **PostgreSQL** 기반의 관계형 데이터베이스를 사용합니다.

**PostgreSQL**은 무료로 제공되는 오픈소스 관계형 데이터베이스 관리 시스템(DMS)으로, 강력한 **SQL** 기능과 유연한 확장성을 제공합니다.

**Supabase**는 **PostgreSQL**을 기반으로 하기 때문에, **PostgreSQL**의 모든 **SQL** 문법을 그대로 사용할 수 있습니다.

슈퍼베이스 오픈소스 링트다. <https://github.com/supabase/supabase>

항목	Firebase	Supabase
제공 방식	클라우드 전용 (Google에서 호스팅)	오픈소스 + 클라우드 (웹 사용 가능, 로컬 설치 가능)
설치 가능 여부	❌ 불가능 (Google 플랫폼에서만 사용)	✅ 가능 (Docker 등으로 로컬 설치 가능)
데이터베이스	NoSQL (Firestore, Realtime Database)	SQL (PostgreSQL 기반)
실시간 기능	✅ Realtime Database	✅ PostgreSQL 기반 실시간 구독
인증(Authentication)	✅ 제공	✅ 제공
스토리지	✅ Cloud Storage	✅ Supabase Storage (S3 호환)
서버리스 함수	✅ Cloud Functions	✅ Edge Functions (Deno 기반)
오픈소스 여부	❌ 비공개 소스	✅ 완전 오픈소스
커스터마이징	❌ 제한적	✅ 코드 수정 및 자유로운 확장 가능
가격 정책	사용량 기반 과금	무료 요금제 + 유료 플랜, 직접 호스팅 시 비용 없음

---

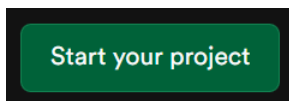
## > 2. 회원가입

---

### Supabase 회원가입

Supabase를 사용하려면 먼저 회원가입을 하고 프로젝트를 생성해야 합니다. GitHub 계정이 이미 있으면 더욱 쉽게 가입할 수 있습니다.

1. Supabase 웹사이트 방문 <https://supabase.com/>



2. 사이트 가운데에 상위 이미지 처럼 "Start your project" 버튼을 클릭합니다.

3. Supabase는 GitHub 계정을 통해 아주 간편하게 가입할 수 있습니다. 별도의 이메일 인증이나 복잡한 절차 없이, GitHub 계정만 있다면 몇 번의 클릭으로 가입이 완료 됩니다.

왼쪽 체크 부분 버튼을 누르면 된다.

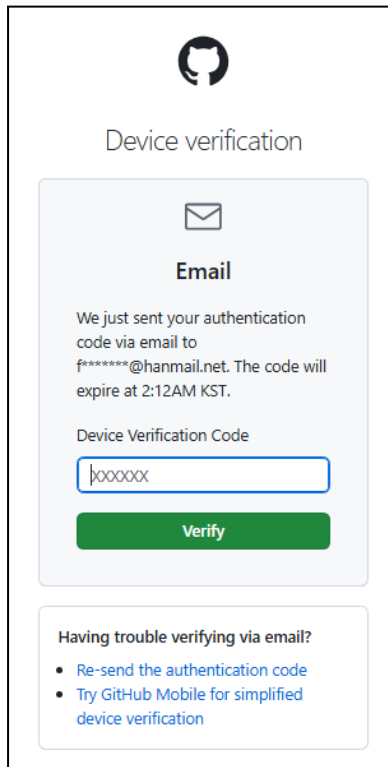
GitHub 인증 절차를 완료하면, Supabase 계정이 자동으로 생성됩니다.

GitHub 계정이 없다면?

먼저 [GitHub](#)에 접속해 계정을 생성합니다. 이후 Supabase 회원가입 화면에서 **GitHub** 버튼을 클릭하면 됩니다.

이후 로그인은? **GitHub** 계정만으로 간편하게 로그인할 수 있다.

깃허브(GitHub)에 로그인에 성공하더라도, 추가적인 본인 인증 절차가 필요합니다. 아래 이미지처럼, 깃허브는 등록된 이메일 주소로 인증 코드(verification code)를 전송하고, 사용자는 해당 코드를 아래 입력창에 직접 입력해야 로그인이 완료됩니다.



이 과정을 정리하면 다음과 같아요:

1. GitHub에 아이디와 비밀번호로 로그인
2. 로그인 성공 시, GitHub는 등록된 이메일로 8자리 인증코드를 전송
3. 사용자 메일함에서 GitHub가 보낸 이메일 열기
4. 메일에 적힌 인증코드를 복사해서 로그인 창의 입력란에 붙여넣기
5. 인증코드를 제출하면 로그인이 최종적으로 완료됨

다음과 같은 화면이 나오면 다음과 같이 기술 또는 선택한 다음에 **Create organization** 버튼을 누른다.

---

## > 3. 프로젝트 생성

---

**Create a new organization**

This is your organization within Supabase.  
For example, you can use the name of your company or department.

**Name**  What's the name of your company or team?

**Type**  What would best describe your organization?

**Plan**  The Plan applies to your new organization.  
[Pricing](#)

[Cancel](#) [You can rename your organization later](#) [Create organization](#)

아래 이미지는 PostgreSQL 기반 프로젝트를 생성할 수 있는 화면이며, 각 항목에 맞게 값을 입력하여 새 프로젝트를 만드는 절차입니다. 다음은 이미지 기준으로 작성한 사용 설명서입니다:

#### [새 프로젝트 생성 절차 안내]

##### 1. Organization (조직 선택)

- `startOrg`와 같은 이름의 조직을 선택합니다.
- 조직이 없는 경우, 사전에 조직을 생성해야 합니다.

##### 2. Project name (프로젝트 이름)

- 생성할 프로젝트의 이름을 입력합니다. 예: `startProject`

- 이름은 영어, 숫자 등을 조합해 자유롭게 정할 수 있습니다.

### 3. Database Password (데이터베이스 비밀번호)

- 사용할 데이터베이스 비밀번호를 입력합니다.
- 비밀번호는 보안 강도가 높은 조합으로 설정해야 하며, 강력한 비밀번호는 **"This password is strong"** 문구가 표시됩니다.
- 원할 경우 **Generate a password**를 클릭해 자동 생성할 수도 있습니다.
- 이책안에서는 사용할 일이 없지만 혹시 모르니 잘 보관해 두시고 누출되면 큰일 납니다.

### 4. Region (지역 선택)

- 서비스 성능 향상을 위해 사용자와 가까운 지역을 선택합니다.
- 예: **Northeast Asia (Seoul)** (대한민국 사용자에게 적합)

### 5. [Create new project] 버튼 클릭

- 모든 설정을 완료한 뒤, 녹색 버튼 **"Create new project"** 를 클릭하여 프로젝트를 생성합니다.

**Create a new project**  
Your project will have its own dedicated instance and full Postgres database.  
An API will be set up so you can easily interact with your new database.

Organization  Free ▼

Project name

Database Password  Copy

This password is strong. [Generate a password](#)

Region  ▼

Select the region closest to your users for the best performance.

SECURITY OPTIONS >

ADVANCED CONFIGURATION >

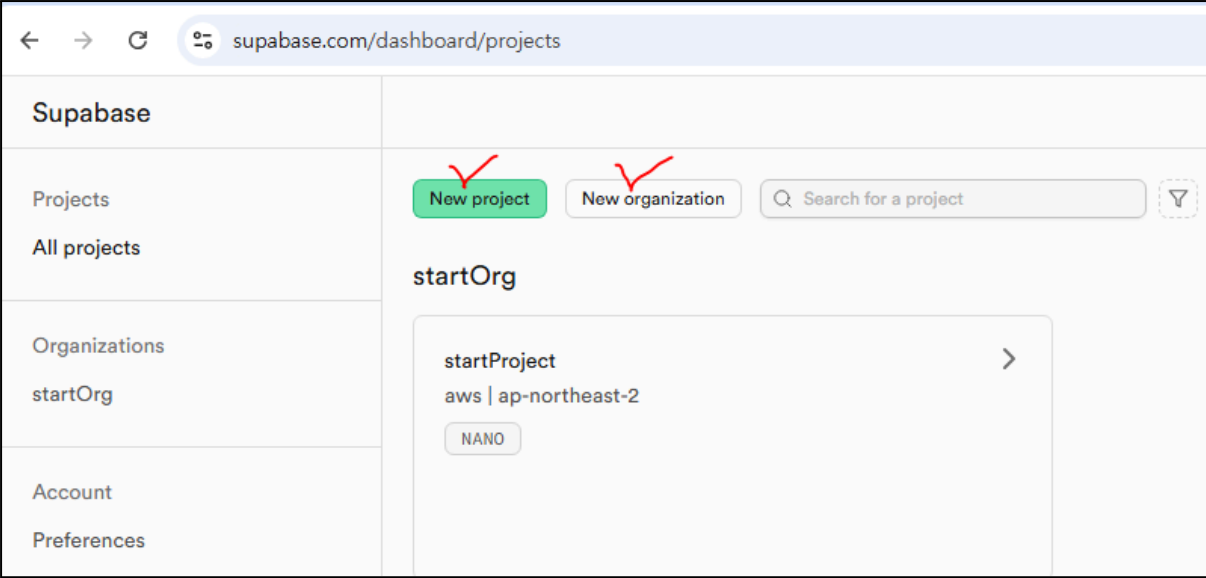
Cancel Create new project

Start your project

이후 새로 조직과 프로젝트를 만든다면 다음과 같이 하면 된다.

[www.supabase.com](https://supabase.com) 에 들어가서 **start your project** 들을 클릭하면 <https://supabase.com/dashboard/projects> 으로 이동한 다음 조직을 만들려면 **New organization**, 프로젝트를 만들려면 **New project**를 클릭한다. 다음 이미지를 참조하자.





---

## > 4. 관계형 데이터베이스 소개

---

슈파베이스는 슈퍼베이스(Supabase)는 PostgreSQL 데이터베이스를 기반으로 구축되었습니다. PostgreSQL 데이터베이스를 사용해서 슈퍼베이스를 사용할 수 있다.

요약 :

1. 데이터베이스는 현실 세계의 데이터를 컴퓨터에 저장하는 가장 대표적인 방법이다.
2. 데이터 베이스를 설치하면 설치된 컴퓨터를 데이터 베이스로 사용할 수 있다.
3. 학생들의 정보를 종이에 기록 하려면 표를 사용한다. 이런 표 형태로 데이터를 관리하는 데이터베이스를 관계형 데이터베이스라고 한다.(RDBMS) **Relational Database Management System** 표형태로 데이터를 저장하는 데이터베이스를 관계형 데이터베이스 라고 한다.

column(열)				table (표)
row(행)	name	Age	height	
	홍길동	30	152.1	
	홍길남	31	156.4	
	홍길영	30	173.5	
	홍길아	21	143.2	

사람 정보를 만들 계획 이라고 생각해 보자. 모양의 표(table)를 그리고 상단에 저장할 속성 (column, 컬럼, 열)를 작성 할 것이다. 속성, 컬럼, 열 을 예로 들자면 이름, 나이, 키, 생일 등등이 있다.

표에 컬럼을 추가후 학생 데이터를 하나씩 입력 하는 작업을 할 것 이고 이때 입력하는 데이터를 로우(row), 행, 데이터라 한다. 이런 표 형태 저장 방법을 사용하는 데이터베이스를 관계형 데이터베이스라고 한다.

데이터베이스에서 사용하는 용어

테이블(표)을 생성 하는 것을 create table, 테이블을 삭제하는 것을 drop table, 변경하는것을 alter table이라고 한다.

데이터를 조작하는 방법으로 insert 입력, select 검색, update 수정, delete 삭제가 있다.

CRUD란? create read update delete의 의미로 insert select update delete 작업을 줄여서 표시한다.

상의 키워드들은 sql에 사용 된다.

sql이란? 질의어라 하는데 여러 사람이 데이터베이스를 사용 할 때 사용하는 약속된 언어이다.

네트워크는 많은 컴퓨터로 연결되어 있고, 데이터를 제공해주는 컴퓨터를 서버, 데이터를 사용하는 컴퓨터를 클라이언트라 한다.

오라클 mssql SQLite mysql MariaDB 등과 같은 많은 데이터 베이스 중에 슈퍼베이스(Supabase)에서 사용하는 PostgreSQL 데이터베이스를 공부할 예정입니다.

내용:

데이터베이스는 현실 세계의 데이터를 컴퓨터에 저장하는 가장 대표적인 방법이다.

컴퓨터에 데이터를 저장하는 방법으로 파일 시스템이 있는데 컴퓨터에 파일로 저장하는 방법 이어서 혼자 사용하기는 쉬우나 여럿이 사용하기는 어렵다. 차후 이를 개선하여 여러 사람이 함께 데이터를 쉽게 사용할 수 있도록 데이터 베이스를 사용 하였다.

파일 시스템은 컴퓨터에 데이터를 파일로 저장하는 시스템 이다. 한글, 워드에 글을 작성 하고 파일로 저장하면 원할때 불러와 저장한 데이터를 확인할 수 있고 이런 방식이 파일 시스템이다.

파일은 혼자 사용하기 편하나 여럿이 사용하는데 불편함이 있어 여러 사람이 24시간 함께 데이터를 공유하고 사용할 수 있는 데이터베이스를 만들어 사용하게 되었다.

데이터베이스란? 데이터를 컴퓨터에 저장해 놓고 매일 24시간 여러 사람과 함께 데이터를 사용하는 시스템이다. 현실 세계의 데이터를 컴퓨터에 저장하는 방법을 고민 하다 파일 시스템을 사용하게 되었고 여러 사람이 같은 파일을 사용하면서 생기는 문제점을 개선하여 데이터베이스 시스템을 만들었다. 데이터 베이스에 저장된 데이터는 컴퓨터를 꺼도 사라지지 않고 나중에 다시 컴퓨터를 켜서 데이터를 사용할 수 있다.

누군가가 데이터 베이스를 설치한 컴퓨터를 날마다 쉬지 않고 운영한다면 여러 사람들이 데이터를 공유하면서 원하는 시점에 데이터를 사용할 수 있다. 이렇게 24시간 데이터를 제공해 주는 컴퓨터를 데이터 베이스 서버라 한다.

네이버, 다음 같은 웹사이트 들이 24시간 운영되면서 지금 우리가 배울 데이터 베이스에서 데이터를 가져다가 여러 사용자에게 브라우저를 통해서 원하는 데이터 정보를 제공해 주는 일을 하고 있다. 이런 웹사이트를 만드는 작업을 웹 프로그램이라 한다.

이때 데이터를 제공하는 컴퓨터를 서버 사용하는 컴퓨터를 클라이언트라 한다.

한글, 워드와 같이 문서 제작 소프트웨어가 여러개 있듯이 데이터 베이스 소프트웨어에는 오라클 mssql SQLite mysql MariaDB 등이 있으며, 이 중 책에서 다룰 데이터 베이스는 PostgreSQL이다. 다른 소프트웨어 처럼 컴퓨터에 PostgreSQL을 설치 하면 본인 컴퓨터를 데이터 베이스로 사용 할 수 있게 된다. 데이터 베이스를 사용하게 되면 간단하게 메모장을 열어 데이터를 컴퓨터에 저장하는 방법을 뛰어 넘어 여러 사람이 함께 저장한 데이터를 쉽게 가져다 사용 할 수 있게 된다. 결국에 데이터 베이스는 여러 사람이 언제 어디서 누구나 데이터를 저장하고 가져다 쓸 수 있도록 관리하는 프로그램이다. 대부분의 데이터베이스는 관계형 데이터베이스 이론을 사용하여 만들어 졌다. 우리가 배울 PostgreSQL도 관계형 데이터베이스이다. 관계형 데이터베이스는 아래와 같은 표 형태로 데이터를 읽어 오고 저장하는 데이터베이스를 의미한다. 우리는 이미 현실 세계에서 표를 사용해

데이터를 기록하는 것 처럼 관계형 데이터 베이스와 같은 방식으로 데이터를 사용하고 있다. 관계형 데이터 베이스에서 사용하는 용어를 확인해 보자. 교실에서 학생들의 정보를 종이에 기록 한다면 다음과 같은 표를 그리고 학생들의 정보를 입력할 것입니다. 이런 방식으로 데이터를 저장하는 데이터베이스를 관계형 데이터베이스라고 한다. (RDBMS) Relational Database Management System 표형태로 데이터를 저장하는 데이터베이스를 관계형 데이터베이스 라고 한다.

name	Age	height	birthday
홍길동	30	152.1	2000.02.03
홍길남	31	156.4	2001.02.03
홍길영	30	173.5	2000.12.21
홍길아	21	143.2	2011.04.17

새로운 모임에 모여 있고 사람들이 사람정보를 만들 계획이라고 생각해 보자. 대표로 한 사람이 종이를 꺼내 직사각형 모양의 표(table)를 그리고 상단에 기록 할 데이터 속성 중 저장할 속성의 column(컬럼, 열)를 작성 할 것이다. 속성, 컬럼, 열 을 예로 들자면 이름, 나이, 키, 생일 등등이 있다. 컬럼 작성 후 학생 데이터를 하나씩 입력 하는 작업을 할 것 이고 이를 로우(row), 행, 데이터 입력이라 한다. 이렇게 사람들이 많이 사용 하는 표 형태 저장 방법을 응용해서 만든 데이터베이스를 관계형 데이터베이스라고 한다.

데이터베이스에서 현실세계의 데이터를 저장하는 과정은 객체를 테이블 이름으로 만들고 객체에서 저장할 속성을 뽑아서 컬럼으로 만들고 테이블에 저장할 데이터를 테이블안에 넣으면 된다.

객체는 현실 세계에서 식별 가능한 존재하는 모든 것을 의미한다

속성은 사물의 상태나 특징을 기술할 것을 의미 한다.

row는 테이블에 입력된 데이터를 의미 한다 .

컬럼은 데이터의 여러 속성중 저장하기 위해 선택된 속성이고 로우는 실제 저장할 대상의 속성값에 해당한다. 상위 예를 가지고 설명해 보자.

현실 세계의 데이터를 데이터베이스에 넣기 위해서 표(테이블)을 사용한다. 현실 세계에서 종이에 표를 그리고 데이터 기록 할 목록 (컬럼)을 기술하는 작업을 데이터 베이스에서는 테이블 생성이라 이야기 하고 명령으로 **create**를 사용 한다.

테이블(표)을 생성 하는 것을 create table, 테이블을 삭제하는 것을 drop table,

변경하는것을 **alter table**이라고 한다. 상위 예를 가지고 설명해 보자.

그 다음에 해야 할 작업은 테이블에 데이터를 채워넣는 작업이다. 테이블에 데이터를 넣는 작업을 **insert**라고 한다. 테이블에 들어 있는 원하는 데이터를 찾는 명령어를 **select**라 한다. 표에 들어 있는 데이터를 수정 할 때 사용하는 명령어를 **update**라 한다. 데이터를 삭제할 때 사용하는 명령어를 **delete**라 한다. 정리하면 **insert** 입력, **select** 검색, **update** 수정, **delete** 삭제에 해당한다. **CRUD**란? **create read update delete**의 의미로 **insert select update delete** 작업을 줄여서 표시한다.

상의 키워드들은 **sql**에 사용 된다. **sql**이란? 질의어라 하는데 여러 사람이 데이터베이스를 사용 할 때 사용하는 약속된 언어이다. **Sql**를 이용하여 원하는 데이터를 데이터베이스에 요청 하면 요청한 데이터를 제공해 준다.

연습문제들을 날마다 확인해서 익숙해 질수 있도록 답해 보자.

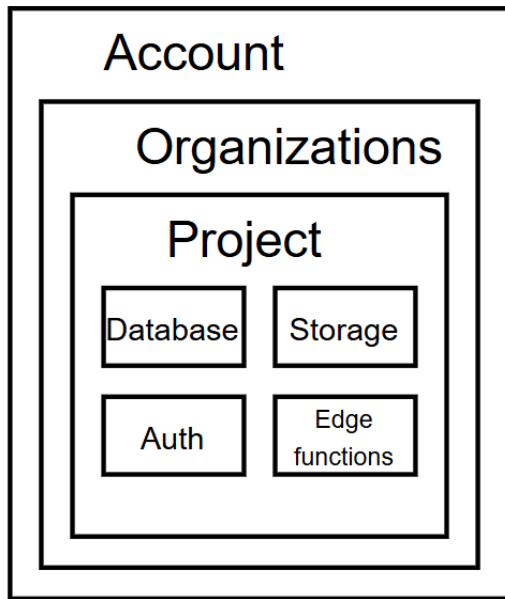
연습문제

- 데이터베이스란?
- 파일시스템이란?
- 서버 클라이언트 란?
- 데이터베이스는 서버와 클라이언트 중 보통 어디에 설치되는가?
- 관계형 데이터베이스에 대해서 설명하시오.
- 객체 속성 로우 테이블 컬럼 **row** 각각에 대해 설명하시오.
- 테이블 조작할 때 사용되는 명령어 세계에 대해서 설명하시오.
- 테이블 안에 들어갈 입력데이터 조작 방법 네 가지 와 각각의 용도를 설명하시오.
- **sql**에 대해서 설명하시오.
- **crud**란?

---

## > 5. supabase 전체 구조 개요

---



Account

└─ Organizations

└─ Project

└─ Database

└─ Storage

└─ Auth

└─ Edge Functions

Supabase는 하나의 계정(Account) 아래 조직(Organization)을 만들고, 각 조직 안의 프로젝트(Project)에서 데이터베이스(Database), 스토리지(Storage), 인증(Auth), 엣지 함수(Edge Functions)를 통합적으로 구성해 앱 개발에 필요한 모든 기능을 제공합니다.

## Supabase 구성 용어 정리

영어 용어	한국어 해석	설명
Account	계정	Supabase에 가입한 사용자. 개인 단위
Organization	조직	팀/회사 단위의 그룹. 여러 사용자가 함께 관리
Project	프로젝트	실제 앱이나 서비스 단위. 하나의 데이터베이스 포함
Database	데이터베이스	PostgreSQL 기반 DB. 테이블, 뷰, 함수 등을 가짐
Storage	스토리지 (파일 저장소)	이미지, 문서, 동영상 등의 파일을 업로드하고 관리
Auth	인증 (Authentication)	로그인, 회원가입, 사용자 관리 기능
Edge Functions	엣지 함수	서버리스 함수. 백엔드 로직 처리용 (TypeScript 기반)

다음 메모장 프로젝트를 만들면서 각각이 어떤 용도로 사용되는지 이해해 보자.  
Supabase로 만드는 메모장 앱 - 기능별 간단 정리

메모장 앱을 만들면서 Supabase의 주요 기능들을 하나씩 사용해볼 수 있습니다.  
Supabase는 백엔드를 빠르게 구성할 수 있게 도와주는 서비스입니다.

### 1. Account (계정)

Supabase에 가입해서 개인 계정을 만들고 로그인합니다.  
→ Supabase에 접근할 수 있는 기본 권한을 제공합니다.

### 2. Organization (조직)

팀 단위로 프로젝트를 관리할 수 있는 공간을 만듭니다.  
예: **memoTeam**  
→ 협업을 위한 팀 공간을 제공합니다.

### 3. Project (프로젝트)

실제 앱을 구성할 작업 공간을 만듭니다.  
예: **memoApp**  
→ 앱 기능을 구성할 독립적인 공간을 제공합니다.

### 4. Database (데이터베이스)

메모 프로그램과 관련된 내용을 저장할 테이블을 생성합니다.  
→ 메모 데이터를 저장하고 관리할 공간을 제공합니다.

### 5. Auth (인증)

메모장 프로젝트 프로그램에 사용하는 로그인/회원가입 기능을 설정합니다.  
이메일, 비밀번호, Google 등 소셜 로그인도 가능해요.  
→ 사용자 인증과 접근 제한 기능을 제공합니다.

### 6. Storage (파일 저장소)

메모장 프로젝트 프로그램에 이미지나 파일을 첨부할 수 있게 설정합니다.  
→ 파일 업로드와 저장 기능을 제공합니다.



## 7. Edge Functions (엣지 함수)

메모장 프로젝트 프로그램과 관련된 자동화된 서버 로직을 실행합니다.

예: 태그 자동 생성, 오래된 메모 삭제, AI 요약 등

→ 백엔드 자동 처리 기능을 제공합니다.

요약

기능	설명
Account	서비스 로그인 및 관리 권한 제공
Organization	협업을 위한 팀 공간 제공
Project	앱 구성 및 리소스 설정 공간 제공
Database	메모 저장 테이블 제공
Auth	로그인 및 인증 기능 제공
Storage	이미지 및 파일 저장 기능 제공
Edge Functions	자동화 로직 처리 기능 제공

---

## > 6. 백업과 삭제 방법

---

백업

현책에서는 백업방법을 제공하지 않는다. 무료버전은 **gui**로 백업방법을 제공하지 않으며 필요하면 **AI**를 활용해서 만들어 달라고 하자. 무료 버전에서도 **Supabase CLI**를 설치해서 백업방법을 제공해 주기는 한다. 다음 링크를 참조 하자.

<https://supabase.com/docs/guides/cli>

프로젝트 삭제 절차

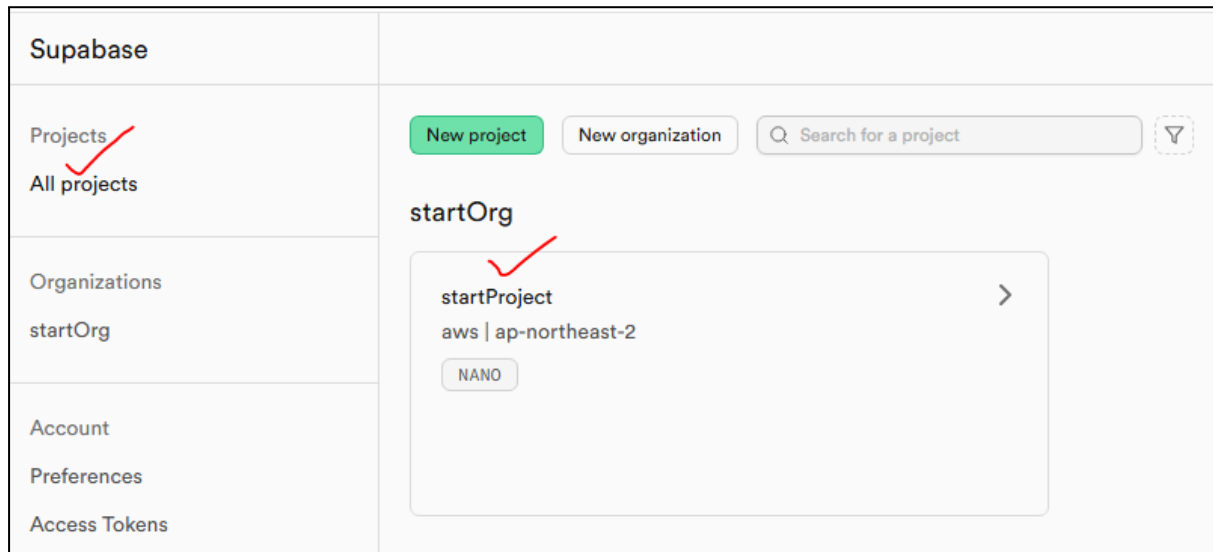
아래 이미지를 확인하면서 다음을 확인하자.

### 1. Supabase 대시보드에 로그인

- a. <https://supabase.com> 접속 후, GitHub 계정 또는 이메일로 로그인합니다.

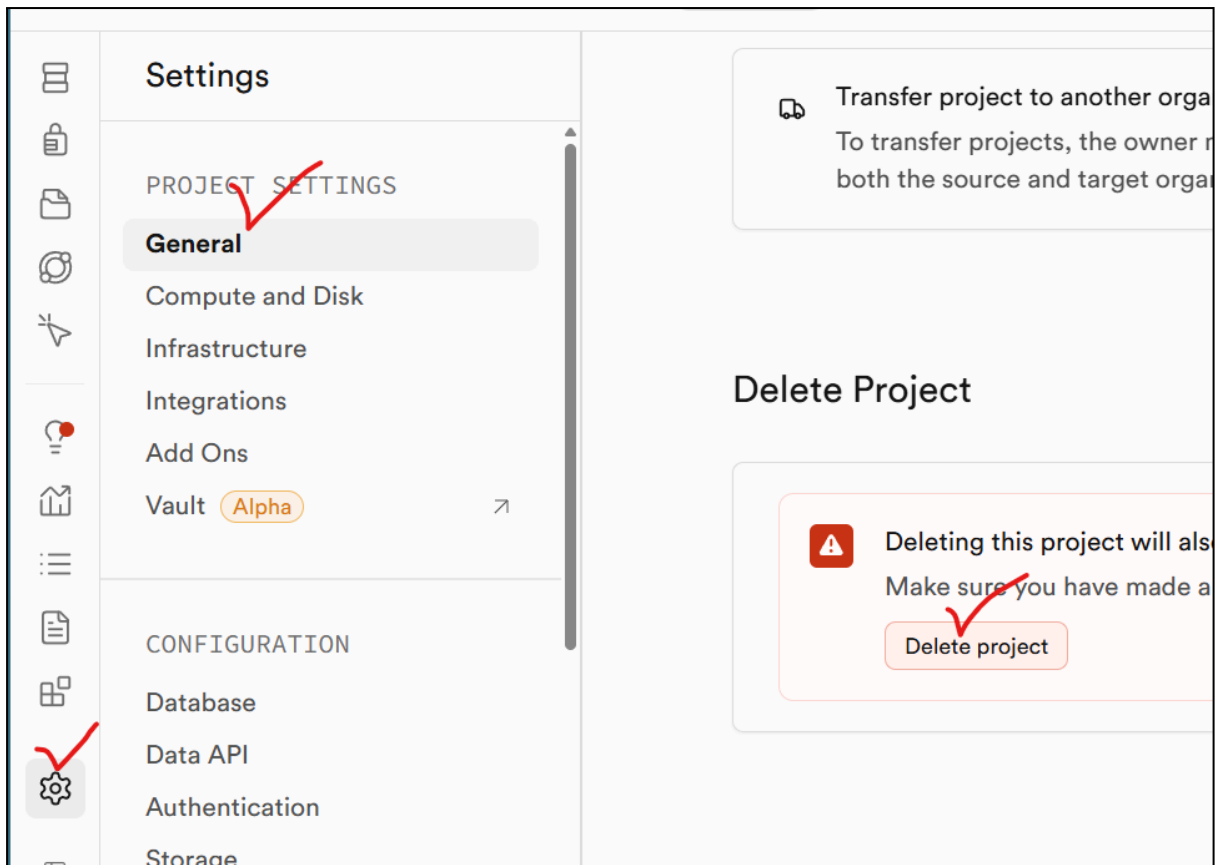
### 2. 삭제할 프로젝트 선택

- a. 로그인 후 대시보드에서 삭제하고자 하는 프로젝트를 클릭합니다.



### 3. 프로젝트 설정(⚙️ Settings) 메뉴 진입

- a. 좌측 사이드바에서 "Settings" → "General" 메뉴를 클릭합니다.

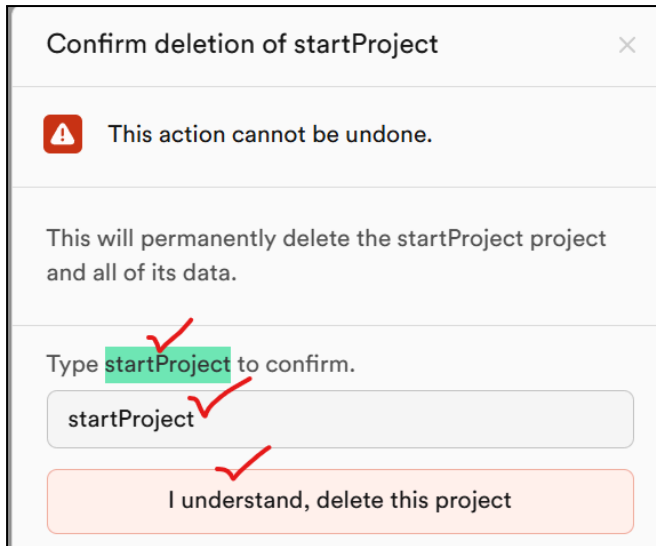


#### 4. 오른쪽 하단의 "Delete Project" 클릭

- a. General 설정 화면 하단으로 스크롤하면 "Delete Project" 버튼이 있습니다.
- b. 버튼을 클릭하면 삭제 확인 창이 나타납니다.

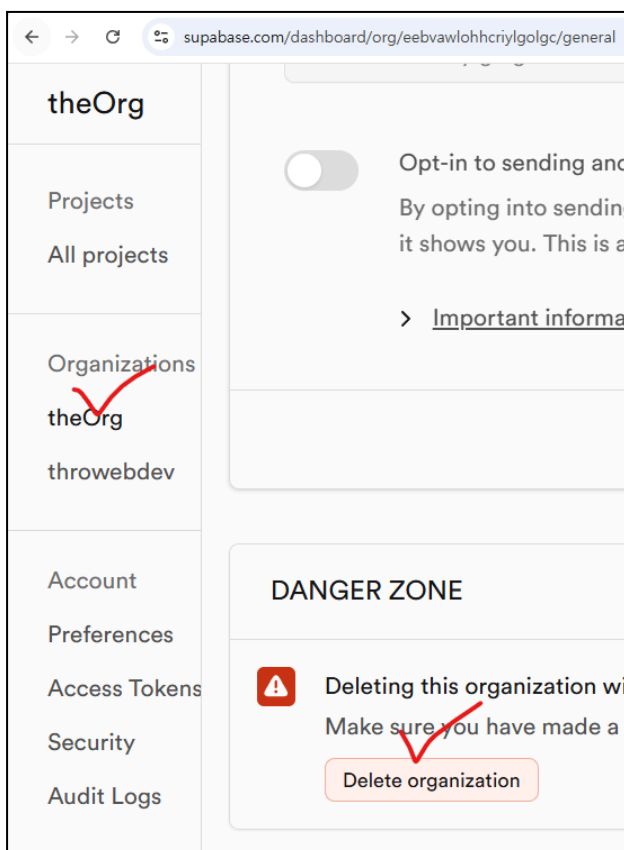
#### 5. 프로젝트 이름 입력 → 최종 삭제

- a. 실수 방지를 위해 해당 프로젝트 이름을 직접 입력해야 삭제가 가능합니다.
- b. 이름을 정확히 입력한 후 "Delete" 버튼을 클릭하면 프로젝트가 완전히 삭제됩니다.



## 조직(Organization) 삭제하기

- Supabase 좌측 메뉴에서 **Organizations** 항목 아래의 삭제하려는 조직(예: **theOrg**)을 클릭합니다.



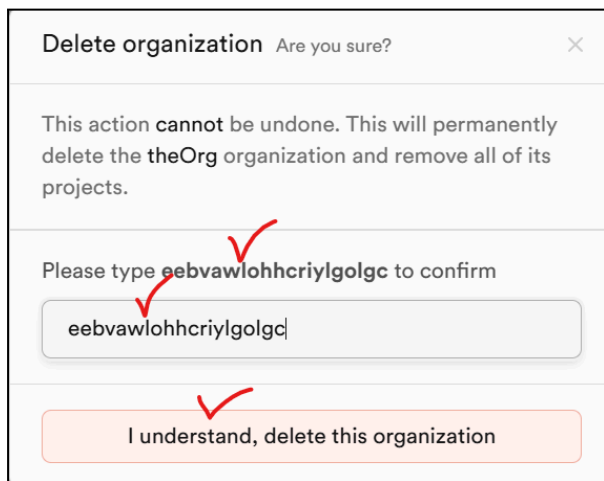
1. 오른쪽 조직 설정 화면 하단으로 스크롤하면 **DANGER ZONE**이 나타납니다.

2. 그 안의 빨간 버튼 "**Delete organization**" 을 클릭합니다.

3. 이후의 확인 절차를 따라 조직 삭제를 완료하면 됩니다.

4. ⚠ 조직 삭제 시, 해당 조직 안의 모든 프로젝트 및 데이터가 함께

삭제됩니다.



#### 주의사항

- 프로젝트나 조직 삭제는 되돌릴 수 없습니다.
- 데이터를 반드시 백업하고 진행하세요.
- 조직 삭제는 해당 조직의 소유자만 수행할 수 있습니다.

---

## > 7. supabase 소개

---

슈퍼 베이스에서 만들어 보자.

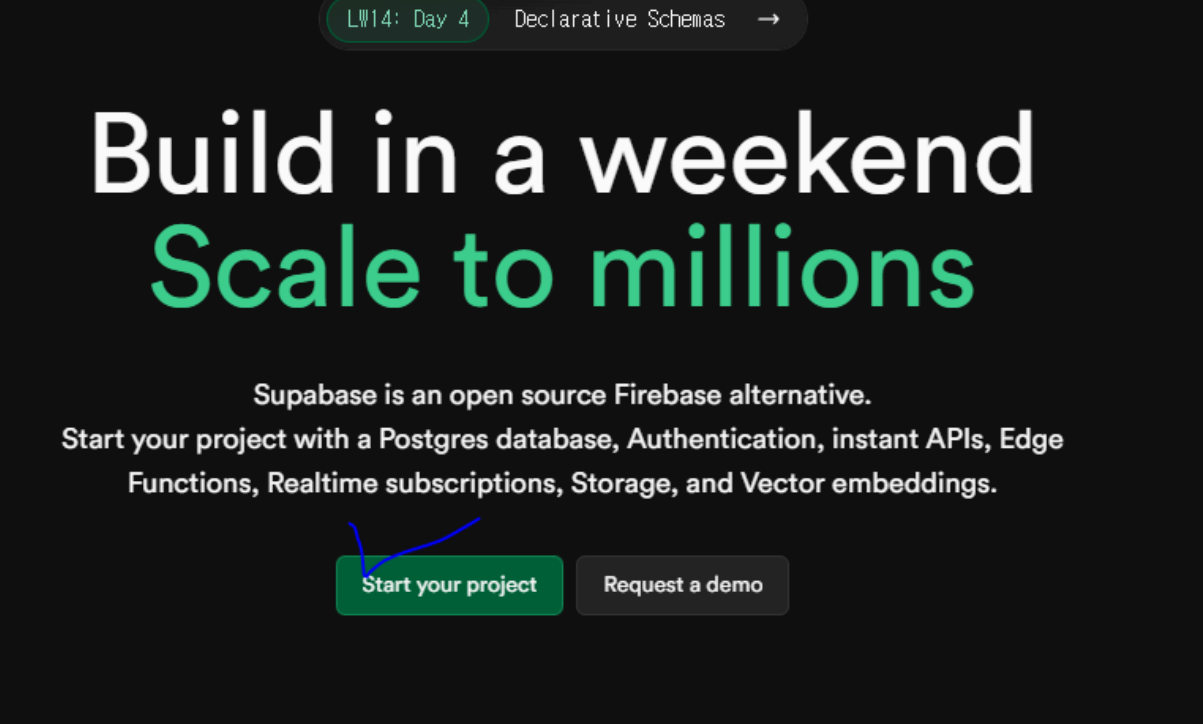
```
create table human(
```

```
  id
```

```
name nvarchar2(30),  
age number(3),  
height number(4,1),  
birthday date  
);
```

html를 이용해서 연결해보자.

참고 문서

A promotional banner for Supabase. At the top, a dark bar contains the text 'LW14: Day 4' in a green pill and 'Declarative Schemas' with a right arrow. The main headline reads 'Build in a weekend' in white and 'Scale to millions' in green. Below this, it states 'Supabase is an open source Firebase alternative.' and lists features: 'Start your project with a Postgres database, Authentication, instant APIs, Edge Functions, Realtime subscriptions, Storage, and Vector embeddings.' At the bottom are two buttons: 'Start your project' (green) and 'Request a demo' (grey). A blue checkmark is drawn over the 'Start your project' button.

LW14: Day 4 Declarative Schemas →

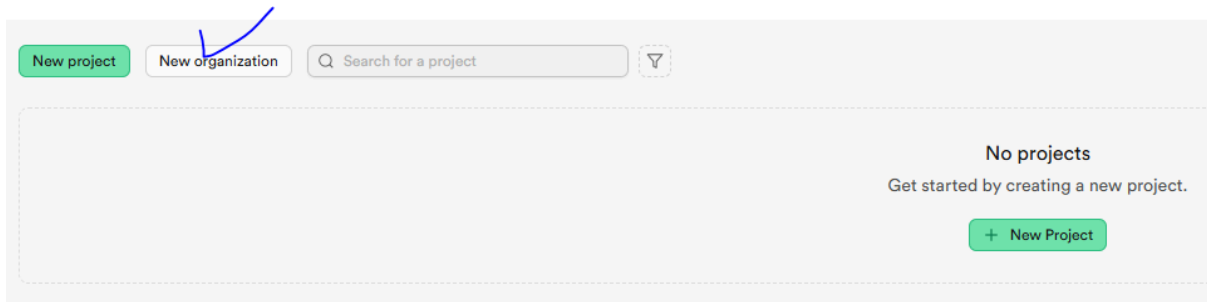
# Build in a weekend Scale to millions

Supabase is an open source Firebase alternative.

Start your project with a Postgres database, Authentication, instant APIs, Edge Functions, Realtime subscriptions, Storage, and Vector embeddings.

Start your project Request a demo

프로젝트를 생성 버튼을 클릭해서 프로젝트를 생성해 보자.



### Create a new organization

This is your organization within Supabase.  
For example, you can use the name of your company or department.

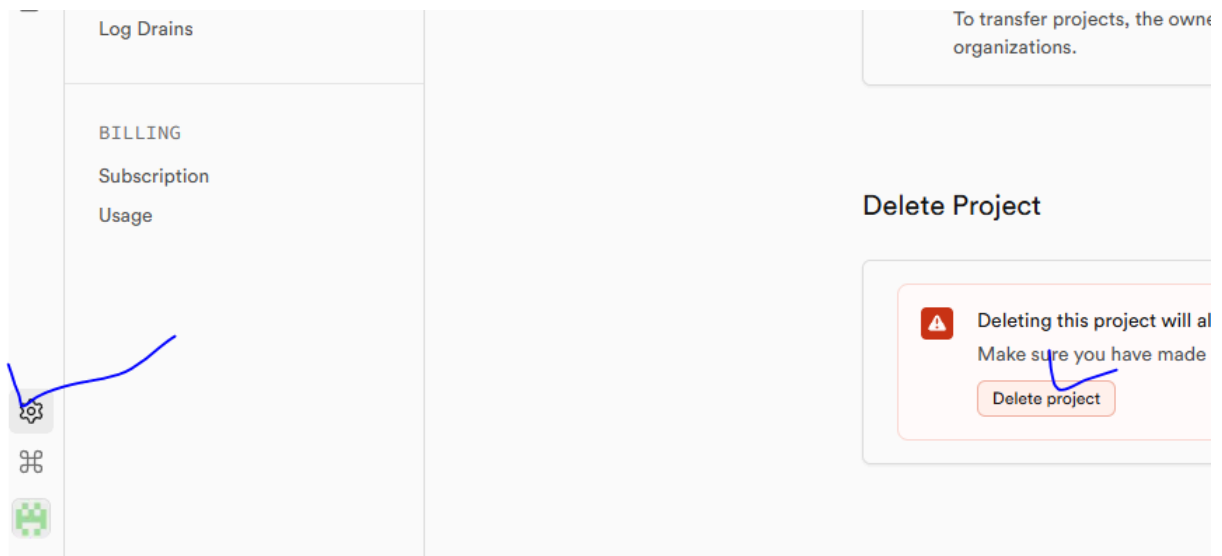
Name   
What's the name of your company or team?


Type of organization   
What would best describe your organization?


Plan   
Pricing [↗](#)


Cancel You can rename your organization later


## 삭제 방법





 grapefoxman's Org Free / grapefoxman's Project Enable branching Connect


 Project overview


 Table Editor


 SQL Editor


 Database


 Authentication


 Storage


 Edge Functions


 Realtime

 Advisors

 Reports

 Logs

 API Docs

 Integrations


# grapefoxman's Project NANO


## Welcome to your new project

Your project has been deployed on its own instance, with

### Get started by building out your database

Start building your app by creating tables and inserting data. It's as easy to use as a spreadsheet, but there's also our SQL Editor

 Table Editor

 SQL Editor


 About Database



Table Editor

There are no tables available in this schema.

Create a new table

Create a new table under public

Name

page

Description

Optional

☒ Enable Row Level Security (RLS) Recommended

Restrict access to your table by enabling RLS and writing Postgres policies.

Policies are required to query data

You need to create an access policy before you can query data from this table. Without a policy, querying this table will return an **empty array** of results. You can create policies after saving this table.

[Documentation](#)

☐ Enable Realtime

Broadcast changes on this table to authorized subscribers

Columns

[About data types](#) [Import data from CSV](#)

Name ?	Type	Default Value ?	Primary
<div><div></div>id</div>	<div># int8</div>	<div>NULL</div>	<div><input checked="" type="checkbox"/></div>
<div><div></div>created_at</div>	<div>timestamp</div>	<div>now()</div>	<div><input type="checkbox"/></div>

We use first-party cookies to improve our services. [Learn more](#)

Columns

About data types
Import data from CSV

Name ?	Type	Default Value ?	Primary
id	# int8	NULL	<input checked="" type="checkbox"/>
created_at	timestamp	now()	<input type="checkbox"/>
title	Choose a ...	NULL	<input type="checkbox"/>
body	Choose a ...	NULL	<input type="checkbox"/>

Add column

Cancel
Save

타입을 text로 설정하자.

grapefoxman's Project
Enable branching
Connect

Filter
Sort
Insert


id int8
1














body text

Insert row  
Insert a new row into page

Insert column  
Insert a new column into page

Import data from CSV  
Insert new rows from a CSV

 grapecfoxman's Org Free / grapecfoxman's Project Enable branching



## SQL Editor

+

Templates

Quickstarts

> SHARED

> FAVORITES

PRIVATE

No private queries created yet

Queries will be automatically saved once you start writing in the editor

dummy\_query\_1

dummy\_query\_2

dummy\_query\_3

dummy\_query\_4

1 |

Hit CTRL+K to edit with the Assistant

rd/project/gysrkorooijxdwddam/sql/60f5b074-2a3e-4aed-a5cf-961d780af050

cfoman's Project Enable branching Connect Feedback Share Close

```

1 CREATE TABLE public.students (
2   id bigint primary key generated always as identity,
3   name text NOT NULL,
4   email text NOT NULL UNIQUE,
5   created_at timestamp with time zone DEFAULT now()
6 );
7

```

Results Chart Source Primary Database Role postgres Run CTRL + R

Click Run to execute your query.

**Assistant** Untitled Feedback Share Close

**Project metadata is not shared**  
The Assistant can improve the quality of the answers if you send project metadata along with your prompts. Opt into sending anonymous data to share your schema and table definitions.

[Update AI settings](#)

page2데이플을 학생으로 만들거야

It seems that there are currently no tables in the public schema. I will create a new table named students for you.

This table will include a primary key and have Row Level Security (RLS) enabled. You will need to create RLS policies before being able to read or write to the table over Supabase APIs.

Here is the SQL to create the students table:

```

-- Create Students Table
CREATE TABLE public.students (
  id bigint primary key generated always as identity,
  name text NOT NULL,
  email text NOT NULL UNIQUE,
  created_at timestamp with time zone DEFAULT now()
) WITH (oids=FALSE);

ALTER TABLE public.students ENABLE ROW LEVEL SECURITY;

```

-- You will need to create RLS policies for this table to control access.

Success. No rows returned

This SQL statement creates a students table with the following columns:

- id: A unique identifier for each student (primary key).
- name: The name of the student.
- email: The email address of the student, which must be unique.
- created\_at: A timestamp indicating when the record was created.

[CREATE TABLE public.students \(](#)

Reply to the assistant...

<https://supabase.com/docs/reference/javascript/installing>

## Installing

```
1 npm install @supabase/supabase-js
```

## Install via CDN

You can install @supabase/supabase-js via CDN links.

```

1 <script src="https://cdn.jsdelivr.net/npm/@supabase/supabase-js@2"></script>
2 //or
3 <script src="https://unpkg.com/@supabase/supabase-js@2"></script>

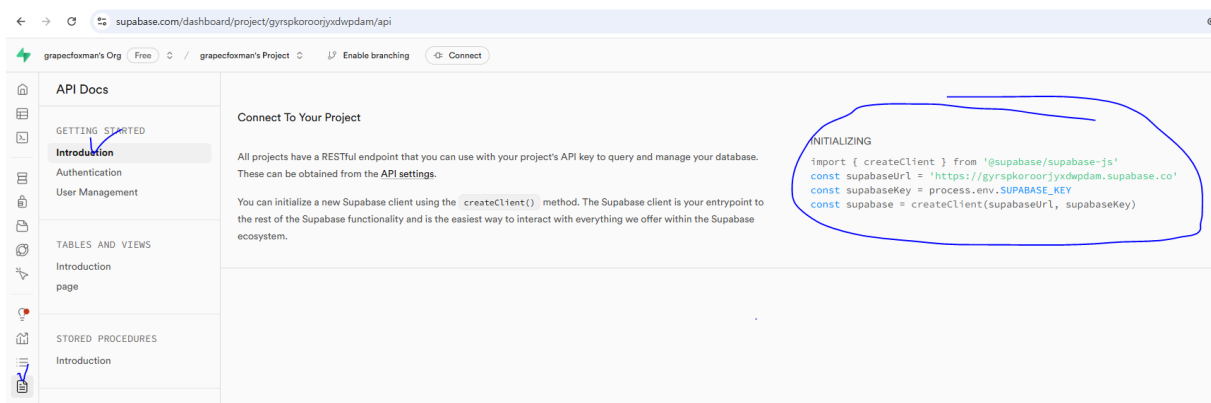
```

## Use at runtime in Deno

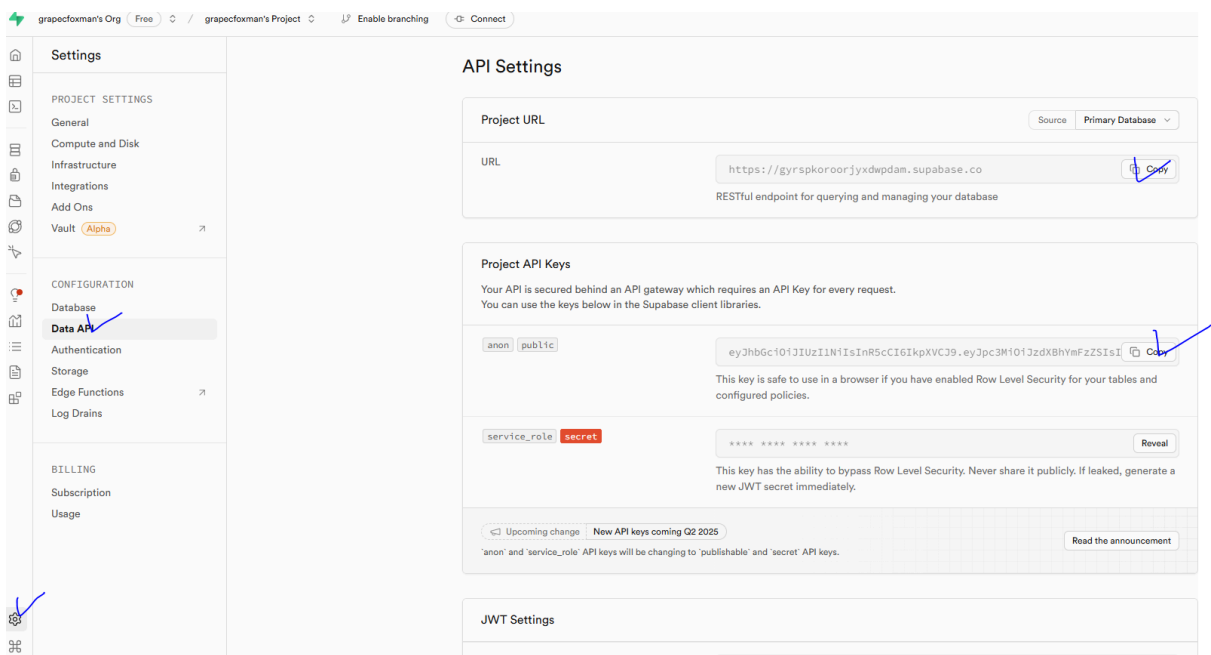
You can use supabase-js in the Deno runtime via JSR:

```
1 import { createClient } from 'jsr:@supabase/supabase-js@2'
```

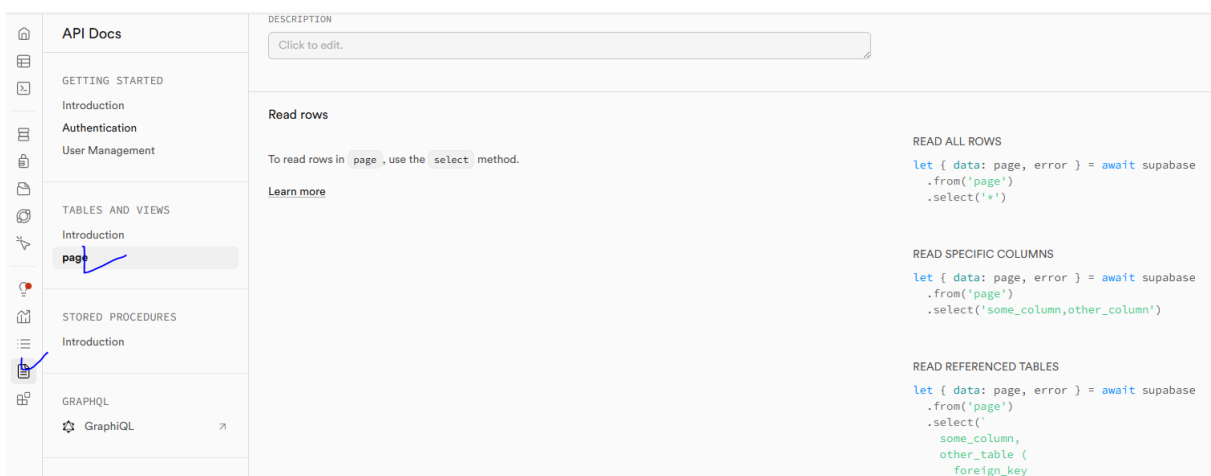
## 연결방법 알아보기



## 연결정보 알아보기



## 테이블 sql알아보기



## 웹서버 만들기

폴더 만들고 서버 설치시 주소로 `index.html`에 접근할수 있다.

```
PS C:\Users\TJ\Desktop\reactwork\ch05> npx local-web-server
Need to install the following packages:
local-web-server@5.4.0
Ok to proceed? (y)

Listening on http://DESKTOP-8FSFFJK:8000, http://192.168.0.102:8000, http://127.0.0.1:8000
```

```
//index.html
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <script
src="https://cdn.jsdelivr.net/npm/@supabase/supabase-js@2"></script>
  <title>Supabase CRUD</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      margin: 20px;
    }
    #container {
      max-width: 500px;
      margin: auto;
      padding: 20px;
      border: 1px solid #ddd;
      border-radius: 8px;
      box-shadow: 2px 2px 12px rgba(0,0,0,0.1);
    }
    input, textarea {
      width: 100%;
      margin: 10px 0;
      padding: 8px;
      border: 1px solid #ccc;
      border-radius: 4px;
    }
    button {
      width: 48%;
```

```

        padding: 10px;
        margin: 5px;
        border: none;
        cursor: pointer;
        color: white;
        border-radius: 4px;
    }
    .btn-create { background: #4CAF50; }
    .btn-read { background: #008CBA; }
    .btn-update { background: #f39c12; }
    .btn-delete { background: #e74c3c; }
    .btn-create:hover { background: #45a049; }
    .btn-read:hover { background: #007bb5; }
    .btn-update:hover { background: #e67e22; }
    .btn-delete:hover { background: #c0392b; }
    .record {
        text-align: left;
        background: #f9f9f9;
        padding: 10px;
        margin: 10px 0;
        border-left: 5px solid #4CAF50;
    }
    .record h3 {
        margin: 0;
        color: #333;
    }
    .record p {
        margin: 5px 0;
        color: #555;
    }
}
</style>
</head>
<body>

<div id="container">
    <h1>Supabase CRUD</h1>
    <input type="text" id="title" placeholder="Title 입력">
    <textarea id="body" rows="4" placeholder="Body 입력"></textarea>
    <button class="btn-create" onclick="createRecord()">Create</button>
    <button class="btn-read" onclick="readRecords()">Read</button>
    <button class="btn-update" onclick="updateRecord()">Update</button>
    <button class="btn-delete" onclick="deleteRecord()">Delete</button>

```

```

    <div id="history"></div>
</div>

<script>
const supabaseUrl = "https://roszuiciqchzsrpxrpsc.supabase.co";
const supabaseKey =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSI6InJlZiI6InJvc3p1aWNpcWNoenNyeHByc3NjIiwicm9sZSI6ImFub24iLCJpYXQiOiJlE3NDQyODAxOTUsImV4cCI6MjA1OTg1NjE5NX0.02RzTXsyG5AtOPJg4HHaWnc8NbRI-AWi8B4QPBm1tNQ"; // 🔥 여기에
Supabase에서 생성한 새로운 키를 넣으세요.
const client = supabase.createClient(supabaseUrl, supabaseKey);

async function createRecord() {
  const title = document.getElementById("title").value;
  const body = document.getElementById("body").value;
  if (!title || !body) return alert("Title과 Body를 입력하세요.");

  const { data, error } = await client.from("page").insert([{ title, body
}]).select();
  if (error) return alert("Create Error: " + error.message);
  console.log("Created:", data);
  readRecords();
}

async function readRecords() {
  let { data: records, error } = await client.from("page").select("*");
  if (error) return alert("Read Error: " + error.message);
  console.log("Fetched:", records);

  const historyDiv = document.getElementById("history");
  historyDiv.innerHTML = "";
  records.forEach(record => {
    const div = document.createElement("div");
    div.className = "record";
    div.innerHTML =
`<h3>${record.title}</h3><p>${record.body}</p><p><small>ID:
${record.id}</small></p>`;
    historyDiv.appendChild(div);
  });
}

async function updateRecord() {

```



```

const id = prompt("수정할 ID를 입력하세요:");
if (!id) return;
const title = document.getElementById("title").value;
const body = document.getElementById("body").value;
if (!title || !body) return alert("Title과 Body를 입력하세요.");

const { error } = await client.from("page").update({ title, body
}).eq("id", id);
if (error) return alert("Update Error: " + error.message);
console.log("Updated ID:", id);
readRecords();
}

async function deleteRecord() {
const id = prompt("삭제할 ID를 입력하세요:");
if (!id) return;

const { error } = await client.from("page").delete().eq("id", id);
if (error) return alert("Delete Error: " + error.message);
console.log("Deleted ID:", id);
readRecords();
}

// 초기 데이터 로딩
readRecords();
</script>

</body>
</html>

```

---

## > 8. supabase 소개

---

---

## > 8. 테이블 만들어 조작해 보기 supabase로 변경

---

### 요약:

1. 기본적으로 sql은 대소문자를 구분하지 않지만 테이블에 들어갈 데이터는 대소문자를 구분한다.

2. sqlplus에서 조작하기 어려울 수 있으니 메모장에서 작업한 다음 ctrl+c로 복사한 다음 콘솔창에서 ctrl+v 를 이용해서 붙여 넣기 해서 사용해 보자.

3.create 테이블 만들기

column(열)

name	Age	height	birthday
홍길동	30	152.1	2000.02.03
홍길남	31	156.4	2001.02.03
홍길영	30	173.5	2000.12.21
홍길아	21	143.2	2011.04.17

row(행)

table  
(표)

```
create table human(  
    name nvarchar2(30),  
    age number(3),  
    height number(4,1),  
    birthday date  
);
```

3. 데이터 입력하기(insert)

```
insert into human(name,age,height,birthday) values ('홍길동', 30, 152.1,  
to_date('2000:02:03 00:00:00', 'YYYY:MM:DD HH24:MI:SS'));
```

- human(name,age,height,birthday) 테이블에 전체데이터를 넣는다면 다음처럼 생략이 가능하다.

```
insert into human values ('홍길남', 31, 156.4, to_date('2001:02:03  
00:00:00', 'YYYY:MM:DD HH24:MI:SS'));
```

```
insert into human values ('홍길영', 30, 173.5, to_date('2000:12:21  
00:00:00', 'YYYY:MM:DD HH24:MI:SS'));
```

```
insert into human values ('홍길아', 21, 143.2, to_date('2011:04:17
00:00:00', 'YYYY:MM:DD HH24:MI:SS'));
```

**commit;** --모든 insert 작업이 끝나면 반드시 **commit**를 실행해 주어야 한다.  
일단은 모든 작업을 적용시키고 마무리 한다는 의미이고, 일단 그냥 써주어야 한다고  
생각하면 된다.

#### 4. 데이터 검색하기(select)

**set linesize 180** --과 같이 화면 크기를 적절히 조절해서 보기 편하게 만들 수  
있다.

column(열)

name	Age	height	birthday
홍길동	30	152.1	2000.02.03
홍길남	31	156.4	2001.02.03
홍길영	30	173.5	2000.12.21
홍길아	21	143.2	2011.04.17

row(행)

table  
(표)

- 모두 출력하고 싶다면 **select** 문앞에 \*를 기술했다. **select \* from human**

column(열)

name	Age	height	birthday
홍길동	30	152.1	2000.02.03
홍길남	31	156.4	2001.02.03
홍길영	30	173.5	2000.12.21
홍길아	21	143.2	2011.04.17

row(행)

table  
(표)

- 특정 컬럼의 데이터만 출력하고 싶다면 **select** 옆에 출력하고자 하는 컬럼을  
,로 연결하여 출력한다. **select age,height from human**

column(열)

name	Age	height	birthday
홍길동	30	152.1	2000.02.03
홍길남	31	156.4	2001.02.03
홍길영	30	173.5	2000.12.21
홍길아	21	143.2	2011.04.17

row(행)

table (표)

- 특정 row만 출력하고 싶다면 where 절을 사용한다.

```
select * from human where height > 155;
```

column(열)

name	Age	height	birthday
홍길동	30	152.1	2000.02.03
홍길남	31	156.4	2001.02.03
홍길영	30	173.5	2000.12.21
홍길아	21	143.2	2011.04.17

row(행)

table (표)

- 특정 컬럼의 특정 row를 출력하고 싶다면 select 다음 특정 컬럼을 ,로 연결하여 기술한 다음 where 절을 사용하여 원하는 row를 선택하여야 한다.

```
select height,birthday from human where height > 155;
```

## 5. 데이터 변경하기(update)

```
update human set age=100 where name= '홍길남';
```

```
update human set age=60,height=170 where name = '홍길아';
```

rollback; --commit;은 작업한 내용 적용 rollback; 작업한 내용을 취소

## 6. 삭제하기(delete)

```
DELETE FROM human; 또는 DELETE human;
```

```
DELETE FROM human WHERE 이름 = 'John';
```

## 7. 테이블 삭제

```
drop table human;
```

```
select * from tab; -- human테이블이 삭제되어 존재하지 않는다.
```

rollback은 insert,update,delete에서만 쓸수 있다 테이블을 drop하면 롤백할 수 없다.

## 8. 주석

```
-- 이것은 한 줄 주석입니다.
```

```
/*
```

```
이것은 다중 줄
```

```
주석입니다.
```

```
*/
```

## 9. null

데이터가 없다는 의미로 null를 사용한다.

```
insert into human values ('', null,0,NULL);
```

null, NULL, '',은 null를 의미 한다.

,, 오라클에서는 null로 처리하지 않는다.

‘null’, ‘없음’, 0 은 데이터가 없다는 의미가 아니다.

## 10. 명명법

영문을 제외한 다른 것은 사용하지 않는 것이 좋다.

2개이상의 단어로 테이블이나 컬럼을 만들때 \_를 사용한다.

## 내용:

기본적으로 sql은 대소문자를 구분하지 않지만 테이블에 들어갈 데이터는 대소문자를 구분한다.

SQL은 "Structured Query Language"의 약자로, 데이터베이스에서 데이터를 관리하기 위한 표준화된 언어입니다. SQL은 관계형 데이터베이스 관리 시스템(RDBMS)에서 데이터를 정의, 조작, 제어하는 데 사용됩니다. RDBMS는 데이터를 테이블 형태로 저장하고 관리하는 시스템을 의미합니다.

SQL은 다음과 같은 주요 작업을 수행할 수 있는 명령어로 구성되어 있습니다:

데이터 정의 언어 (DDL - Data Definition Language):

데이터베이스의 구조를 정의하는 명령어로, 테이블, 인덱스, 뷰 등을 생성, 수정 또는 삭제합니다.

주요 명령어: CREATE, ALTER, DROP.

데이터 조작 언어 (DML - Data Manipulation Language):

데이터를 쿼리하고 조작하는 명령어로, 테이블에 데이터를 삽입, 수정, 삭제하고 조회합니다.

주요 명령어: SELECT, INSERT, UPDATE, DELETE.

데이터 제어 언어 (DCL - Data Control Language):

데이터베이스의 접근 권한을 관리하는 명령어로, 사용자에게 권한을 부여하거나 회수합니다.

주요 명령어: GRANT, REVOKE.

SQL은 다양한 데이터베이스 시스템에서 사용되며, 대표적으로 Oracle, MySQL, PostgreSQL, Microsoft SQL Server 등이 있습니다. SQL을 사용하면 데이터베이스와 상호 작용하여 데이터를 효과적으로 관리하고 조회할 수 있습니다.

sql 절의어를 이용하여 다음 페이지에 있는 테이블을 조작해 보면서 이해해 보자.

sqlplus에서 조작하기 어려울 수 있으니 메모장에서 작업한 다음 ctrl+c로 복사한 다음 콘솔창에서 ctrl+v 를 이용해서 붙여 넣기 해서 사용해 보자.

The diagram shows a table with four columns and four rows. A bracket above the columns is labeled 'column(열)'. A bracket to the left of the rows is labeled 'row(행)'. A bracket to the right of the entire table is labeled 'table (표)'.

name	Age	height	birthday
홍길동	30	152.1	2000.02.03
홍길남	31	156.4	2001.02.03
홍길영	30	173.5	2000.12.21
홍길아	21	143.2	2011.04.17

### 테이블 만들기(create)

테이블명은 식별자여서 같은 이름이 있거나 **sql**에서 사용하는 키워드를 사용하면 안된다.

```
CREATE TABLE 테이블명 (
    컬럼1명 자료형1,
    컬럼2명 자료형2,
    컬럼3명 자료형3,
    -- 추가 컬럼들
);
```

테이블명: 생성할 테이블의 이름을 나타냅니다. 테이블명은 중복되지 않는 고유한 이름이어야 합니다.

컬럼1명, 컬럼2명, 컬럼3명: 테이블에 포함될 각 열의 이름을 나타냅니다. 열의 이름은 해당 테이블 내에서 고유해야 합니다.

자료형1, 자료형2, 자료형3: 각 컬럼의 데이터 형식을 나타냅니다. 숫자, 문자열, 날짜 등 다양한 자료형을 사용할 수 있습니다.

예를 들어, 실제로 사용할 수 있는 예제는 다음과 같습니다:

```
CREATE TABLE employees (
    employee_id NUMBER(6),
    first_name VARCHAR2(50),
    last_name VARCHAR2(50),
    hire_date DATE,
    salary NUMBER(10, 2)
);
```

위의 예제에서는 **"employees"**라는 테이블을 생성하며, 다섯 개의 열을 가지고 있습니다. 이 열들은 각각 사원 ID, 이름, 성, 입사일, 급여를 나타내며 각각의 데이터 형식에 맞게 정의되어 있습니다.

상위 이미지의 테이블명은 사람 정보를 저장할 예정이므로 **human**으로 하자. 다음 **sql**를 **sqlplus**에 출력해서 결과를 확인해 보자.

```
create table human(  
    name nvarchar2(30),  
    age number(3),  
    height number(4,1),  
    birthday date  
);
```

--현재 데이터베이스에 있는 테이블정보 출력하여 방금 만든 테이블을 확인해 보자.  
내가 만들지 않은 테이블이 있을 수 있는데 시스템이 관리하는 테이블이라고  
생각하고 무시하자.

```
select * from tab;
```

```
desc human;          --human 테이블 상세 내용 출력
```

데이터 입력하기(**insert**)

**INSERT INTO** 문은 데이터베이스 테이블에 새로운 레코드(행,row)를 추가하는 데  
사용됩니다. 다음은 **INSERT INTO** 문의 일반적인 구조입니다

**INSERT INTO 테이블명 (컬럼1, 컬럼2, 컬럼3, ...)**

**VALUES (값1, 값2, 값3, ...);**

테이블명: 데이터를 삽입할 대상 테이블의 이름

(컬럼1, 컬럼2, 컬럼3, ...): 데이터를 삽입할 테이블의 컬럼 목록. 중괄호 안에  
여러 컬럼을 콤마로 하여 기술 한다.

**VALUES (값1, 값2, 값3, ...):** 각 컬럼에 삽입될 실제 값들의 목록입니다. 이  
목록도 괄호 안에 값들을 콤마로 구분한다.

예제:

```
INSERT INTO employees (employee_id, first_name, last_name, hire_date,  
salary)
```

```
VALUES (1, 'John', 'Doe', '2022-01-06', 50000.00);
```



위의 예제에서는 "employees" 테이블에 새로운 레코드를 추가하고 있습니다. 각 컬럼에는 새로운 직원의 정보가 들어가게 됩니다. 이 때, 데이터 유형에 주의하여 값을 삽입해야 합니다. 날짜 형식이나 숫자 형식 등을 잘 맞춰주어야 합니다.

다음은 human 테이블에 데이터를 넣는 방법이다.

```
insert into human(name,age,height,birthday) values ('홍길동', 30, 152.1,
to_date('2000:02:03 00:00:00', 'YYYY:MM:DD HH24:MI:SS'));
```

- human(name,age,height,birthday) 테이블에 전체데이터를 넣는다면 다음처럼 생략이 가능하다.

```
insert into human values ('홍길남', 31, 156.4, to_date('2001:02:03
00:00:00', 'YYYY:MM:DD HH24:MI:SS'));
```

```
insert into human values ('홍길영', 30, 173.5, to_date('2000:12:21
00:00:00', 'YYYY:MM:DD HH24:MI:SS'));
```

```
insert into human values ('홍길아', 21, 143.2, to_date('2011:04:17
00:00:00', 'YYYY:MM:DD HH24:MI:SS'));
```

commit; --모든 insert 작업이 끝나면 반드시 commit를 실행해 주어야 한다.

일단은 모든 작업을 적용시키고 마무리 한다는 의미이고, 일단 그냥 써주어야 한다고 생각하면 된다.

## 데이터 검색하기(select)

SELECT 문을 사용하여 테이블에서 원하는 정보를 검색하는 작업을 한다.

사용 방법은 select 다음에 검색 하고자 하는 컬럼명을 기술하고 from 다음에 검색할 테이블 이름을 기술하고 where절에 검색 조건을 기술하면 해당 테이블의 원하는 데이터가 출력 된다.

모든 컬럼을 출력하고 싶다면 \*을 사용하면 된다. 특정 컬럼명만 출력하고 싶다면 컬럼명 위치에다 출력 하고자 하는 컬럼명을 ,로 연결하여 기술하면 된다. 전체 row(행) 데이터가 아닌 특정 row(행) 데이터만 출력하고 싶다면 where절을 사용한다.

-- 테이블의 모든 컬럼을 조회

```
SELECT * FROM human;
```

-- 특정 컬럼들만 선택하여 조회

SELECT 컬럼1, 컬럼2, 컬럼3

FROM 테이블명;

-- WHERE 절을 사용하여 특정 데이터만 조회

SELECT \* FROM 테이블명 WHERE 조건식;

-- and나 or로 연결하여 여러 조건을 조합하여 조회할 수 있다.

SELECT \* FROM 테이블명 WHERE 조건1 AND/OR 조건2;

컬럼1, 컬럼2, 컬럼3: 조회하고자 하는 특정 컬럼들을 ,로 연결해서 나열한다.

\* : 모든 컬럼을 출력하고자 할때 사용한다.

WHERE: 특정 조건을 지정하여 특정 데이터만을 조회할 수 있습니다. 예를 들어,  
WHERE salary > 50000은 급여가 50000 이상인 데이터만을 조회합니다.

AND, OR: 여러 조건을 결합하여 사용할 수 있습니다.

WHERE salary>50000 and first\_name='John'; --두 조건 모두 만족하면 데이터가  
출력

WHERE salary>50000 or first\_name='John'; --둘 중 하나의 조건을 만족하면  
데이터출력

다양한 예제를 확인해 보자.

-- "employees" 테이블에서 "John"이라는 이름을 가진 직원의 모든 정보를 조회

SELECT \* FROM employees WHERE first\_name = 'John';

위의 예제에서는 "employees" 테이블에서 "John"이라는 이름을 가진 직원의 모든  
정보를 조회하고 있습니다.

-- "employees" 테이블에서 이름과 급여 정보만 조회

SELECT first\_name, last\_name, salary

FROM employees;

하나의 데이터가 한줄에 출력 할 수 없다면 다음과 같이 한줄에 출력하는 글자수를 늘려 보기 편하게 만들 수 있다.

`set linesize 180` --과 같이 화면 크기를 적절히 조절해서 보기 편하게 만들 수 있다.

column(열)

name	Age	height	birthday
홍길동	30	152.1	2000.02.03
홍길남	31	156.4	2001.02.03
홍길영	30	173.5	2000.12.21
홍길아	21	143.2	2011.04.17

row(행)

table (표)

- 모두 출력하고 싶다면 `select` 문옆에 `*`를 기술한다. `select * from human`

column(열)

name	Age	height	birthday
홍길동	30	152.1	2000.02.03
홍길남	31	156.4	2001.02.03
홍길영	30	173.5	2000.12.21
홍길아	21	143.2	2011.04.17

row(행)

table (표)

- 특정 컬럼의 데이터만 출력하고 싶다면 `select` 옆에 출력하고자 하는 컬럼을 ,로 연결하여 출력한다. `select age,height from human`

column(열)

name	Age	height	birthday
홍길동	30	152.1	2000.02.03
홍길남	31	156.4	2001.02.03
홍길영	30	173.5	2000.12.21
홍길아	21	143.2	2011.04.17

row(행)

table  
(표)

- 특정 row만 출력하고 싶다면 where 절을 사용한다.

```
select * from human where height > 155;
```

column(열)

name	Age	height	birthday
홍길동	30	152.1	2000.02.03
홍길남	31	156.4	2001.02.03
홍길영	30	173.5	2000.12.21
홍길아	21	143.2	2011.04.17

row(행)

table  
(표)

-특정 컬럼의 특정 row를 출력하고 싶다면 select 다음 특정 컬럼을 ,로 연결하여 기술한 다음 where절을 사용하여 원하는 row를 선택하여야 한다.

```
select height,birthday from human where height > 155;
```

다음을 확인해 보자.

이름과 나이만 출력하고 싶다면 다음과 같이 기술하면 된다.

```
select name,age from human;
```

생일만 출력하고 싶다면 `select birth from human;` 과 같이 기술하면 된다. 원하는 데이터만(row)을 출력하고 싶다면 where절을 사용한다.

홍길남만 출력하고 싶을때 `select * from human where name='홍길남';` 나이가 25보다 큰 사람들의 정보를 출력하고 싶다면 다음과 같이 하면 된다.

```
select * from human where age>25;
```

키가 150보다 작은 사람의 이름과

생일을 출력해 보자.

```
select name,birthday
```

```
from human where height<150;
```

## 데이터 변경하기(update)

**UPDATE** 문은 데이터베이스 테이블에서 기존의 레코드를 수정할 때 사용됩니다.  
기본 구조는 다음과 같다

**UPDATE** 테이블명

**SET** 컬럼1 = 변경값1, 컬럼2 = 변경값2, ...

**WHERE** 조건;

테이블명: 수정하고자 하는 테이블의 이름을 나타냅니다.

**SET**: 수정할 컬럼과 해당하는 값을 설정하는 키워드입니다.

컬럼1 = 변경값1, 컬럼2 = 변경값2, ...: 변경하고자 하는 각 컬럼과 그에 해당하는 새로운 값을 설정합니다.

**WHERE**: 어떤 레코드를 수정할 것인지를 결정하는 조건을 지정하는 키워드입니다. 이 부분이 없으면 테이블의 모든 레코드가 변경됩니다.

예를 들어, "employees" 테이블에서 사원의 급여를 60000으로 변경하고자 할 때:

```
UPDATE employees SET salary = 60000 WHERE employee_id = 1;
```

이렇게 하면 "employees" 테이블에서 employee\_id가 1인 사원의 급여가 60000으로 변경됩니다.

또한 ,로 여러 개의 컬럼을 동시에 변경할 수 있으며, **WHERE** 절을 사용하여 특정 조건을 만족하는 레코드만을 수정할 수 있습니다. 이를 통해 정확한 대상 레코드를 지정하여 원하는 부분만 수정할 수 있습니다. **where**절을 사용하지 않으면 선택된 테이블의 모든 데이터가 변경된다.

human 테이블의 age를 100으로 변경해보자.

update human set age=100; -- human테이블의 모든 데이터가 100으로  
변경됨 select \* from human;를 실행 시키면 변경된 모든 데이터를 확인 할  
수 있다. 특정 데이터만 변경하고 싶다면 where절을 사용하면 된다.

rollback; --commit;은 작업한 내용 적용 rollback; 작업한 내용을 취소한다.  
rollback를 입력하여 작업을 취소하자.

모든 age컬럼의 값이 100으로 변경된 상태에서 commit;를 하면 적용되지만

`rollback;`를 통해서 `age` 컬럼을 `100`으로 변경하기 전으로 복구할 수 있다.

`rollback;`후 `select * from human;`를 실행시켜 원래 데이터로 복구 되었는지 확인해 보자.

만약 `rollback;`전에 `commit;`를 하고 나면 데이터가 모두 적용되어 `rollback;`를 하더라도 `age`가 `100`일때 이전 데이터로 돌아 갈 수 없다. `rollback;`은 이전 `commit;`를 기준으로 이전 데이터로 복귀 한다.

특정 `row`만 변경하고 싶으면 `where`절을 사용해야 한다.

홍길남의 나이를 `100`으로 변경해 보자.

`update human set age=100 where name= '홍길남';`

홍길아의 나이와 키를 `60,170`으로 변경해 보자.

`update human set age=60,height=170 where`

`name = '홍길아';`

작업을 취소 하려면 `rollback;` 적용하려면

`commit;`를 입력하자.

`sqlplus`에서 제대로 변경되고 있는지

`select`문을 이용하여 그때 그때 확인해 보자.

삭제하기(`delete`)

원하는 행(`row`)을 삭제할때 사용한다. `delete from` 다음에 삭제할 데이터가 들어 있는 테이블명을 기입하면 테이블의 모든 데이터가 삭제된다. 특정 데이터만 삭제하고 싶다면 `where`절을 사용한다.

`DELETE` 문의 기본 구조는 다음 2개의 예제와 같다

`DELETE FROM 테이블명 WHERE 삭제조건;`

`DELETE 테이블명 WHERE 삭제조건; -- from`를 생략 할 수 있다.

테이블명: 삭제 하고자 하는 테이블의 이름입니다.

**WHERE:** 어떤 레코드를 삭제할 것인지를 결정하는 조건을 지정하는 키워드입니다. 이 부분이 없으면 테이블의 모든 레코드가 삭제됩니다.

예를 들어, "`human`" 테이블에서 모든 데이터를 삭제하고자 할 때:

`DELETE FROM human; 또는 DELETE human;`

예를 들어, "`human`" 테이블에서 이름이 "`John`"인 행을 삭제하고자 한다면:

```
DELETE FROM human WHERE 이름 = 'John';
```

DELETE 문은 사용할 때 신중하게 사용해야 합니다. WHERE 절을 사용하지 않으면 테이블의 모든 레코드가 삭제되므로 주의가 필요합니다.

human테이블에 모든 데이터를 삭제해 보자.

```
delete human;      --혹은  delete from human;
```

select \* from human;를 통해서 모든 데이터가 삭제되었는지 확인해 보자. 확인이 완료되었으면 rollback;를 입력하여 이전 데이터로 돌아가 보자. rollback;은 작업 내용을 취소하여 이전 데이터로 되돌릴때 사용한다. 제대로 rollback 되었는지 select문을 이용해서 확인해 보자. where절에 삭제할 데이터를 선택하면 선택된 데이터가 삭제된다.

나이가 30보다 큰 사람의 데이터를 삭제해 보자. delete human where age>30;

데이터 관련 작업이 마무리되면 반드시 commit;이나 rollback;중하나를 선택해서 실행해야 한다. 적용하고 싶으면 commit; 취소하고 싶으면 rollback;를 실행하면 된다.

## 테이블 삭제

delete는 테이블에 들어 있는 데이터를 삭제하는 것이고 테이블 삭제는 테이블 자체를 지우는 것이다.

테이블을 삭제하고 싶다면 drop table 다음에 삭제하고자 하는 테이블 이름을 기술하면 된다. 테이블이 삭제되면 당연히 해당 테이블의 데이터는 접근 할 수 없다.

drop table 테이블명; --데이터 베이스에서 해당 테이블을 삭제한다.

```
drop table human;
```

```
select * from tab; -- human테이블이 삭제되어 존재하지 않는다.
```

rollback은 insert,update,delete에서만 쓸수 있다 테이블을 drop하면 롤백할 수 없다.

SQL 주석은 SQL 문장에서 설명이나 메모를 추가하는 데 사용됩니다. 주석은 SQL 엔진에 의해 무시되며 쿼리 실행에 영향을 미치지 않습니다. SQL 주석은 코드를 이해하기 쉽게 만들고 코드의 일부를 임시로 비활성화하는 데 유용합니다.

-- 이것은 한 줄 주석입니다.

/\*

이것은 다중 줄

주석입니다.

\*/

null이란? 데이터가 없을때 데이터가 없다는 의미로 null를 사용한다.

```
insert into human values ('', null,0,NULL);
```

null, NULL, '',은 null를 의미 한다. , , 처럼 , ,사이에 값을 기술해야 할때 아무것도 기술하지 않으면 null로 처리되는 데이터베이스도 있지만 오라클에서는 null로 처리하지 않는다.

null은 데이터가 없음을 의미하고 'null', '없음', 0 은 데이터가 없다는 의미가 아니고 문자열 null, 없음, 숫자 0 데이터가 있다는 의미이다 헛갈리지 않도록 조심하자.

테이블명이나 컬럼명에 영문을 제외한 다른 것은 사용하지 않는 것이 좋다.

2개이상의 단어로 테이블이나 컬럼을 만들때에는 \_를 사용한다. mybag인경우 my\_bag으로 이름을 짓는다. 이런 방식을 뱀이 지나가는 것 같다고 해서 스네이크 케이스라고 한다.

---

## > 9. 권한

---

로그인 구현

myauth프로젝트를 만든다.



### Create a new project

Your project will have its own dedicated instance and full Postgres database.  
An API will be set up so you can easily interact with your new database.

Organization

grapefoxman's Org

Project name

myauth

Database Password

.....

Copy

This password is strong. [Generate a password](#)

Region

Southeast Asia (Singapore)

Select the region closest to your users for the best performance.

SECURITY OPTIONS >

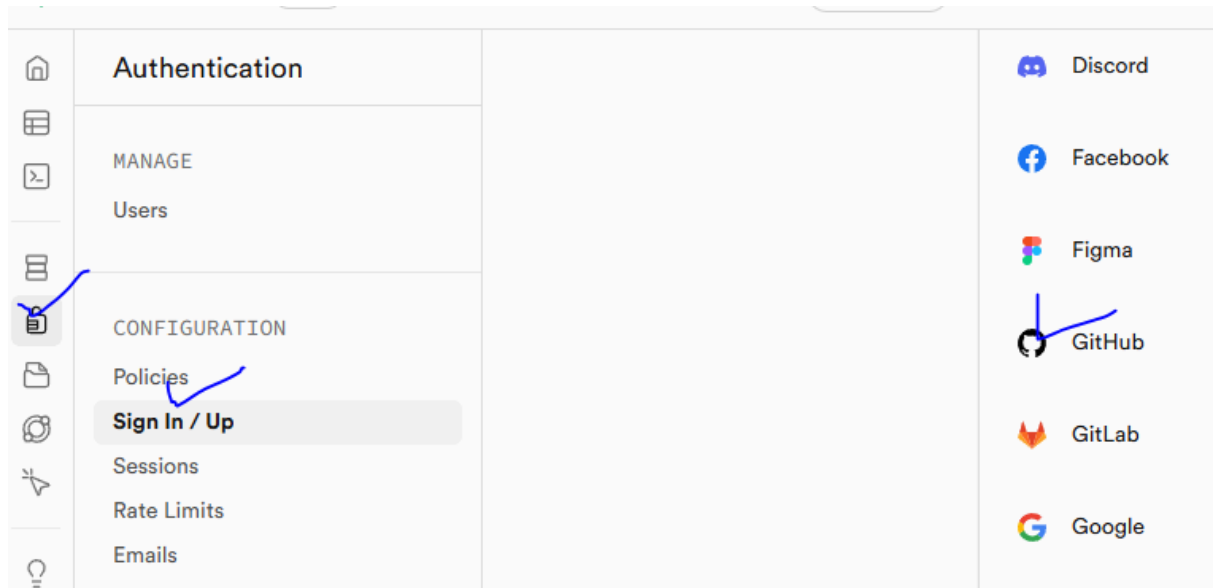
ADVANCED CONFIGURATION >

Cancel

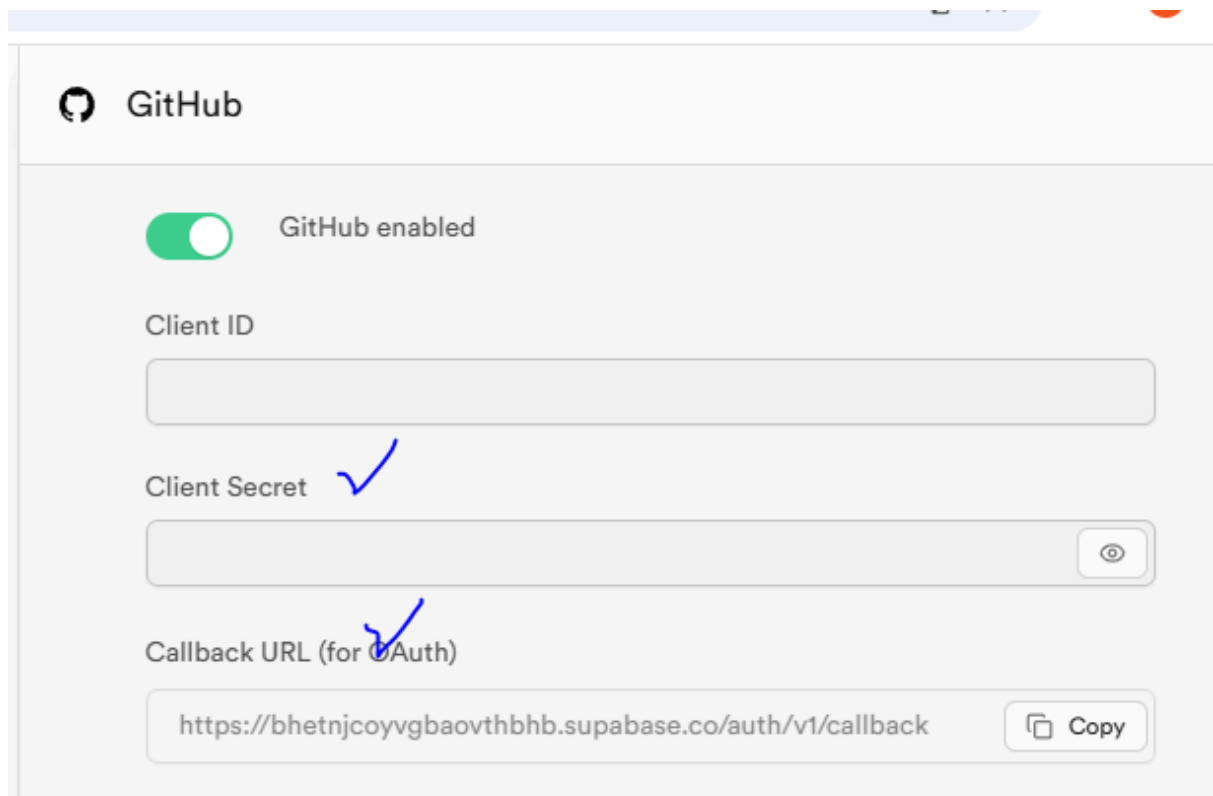
You can rename your project later

Create new project

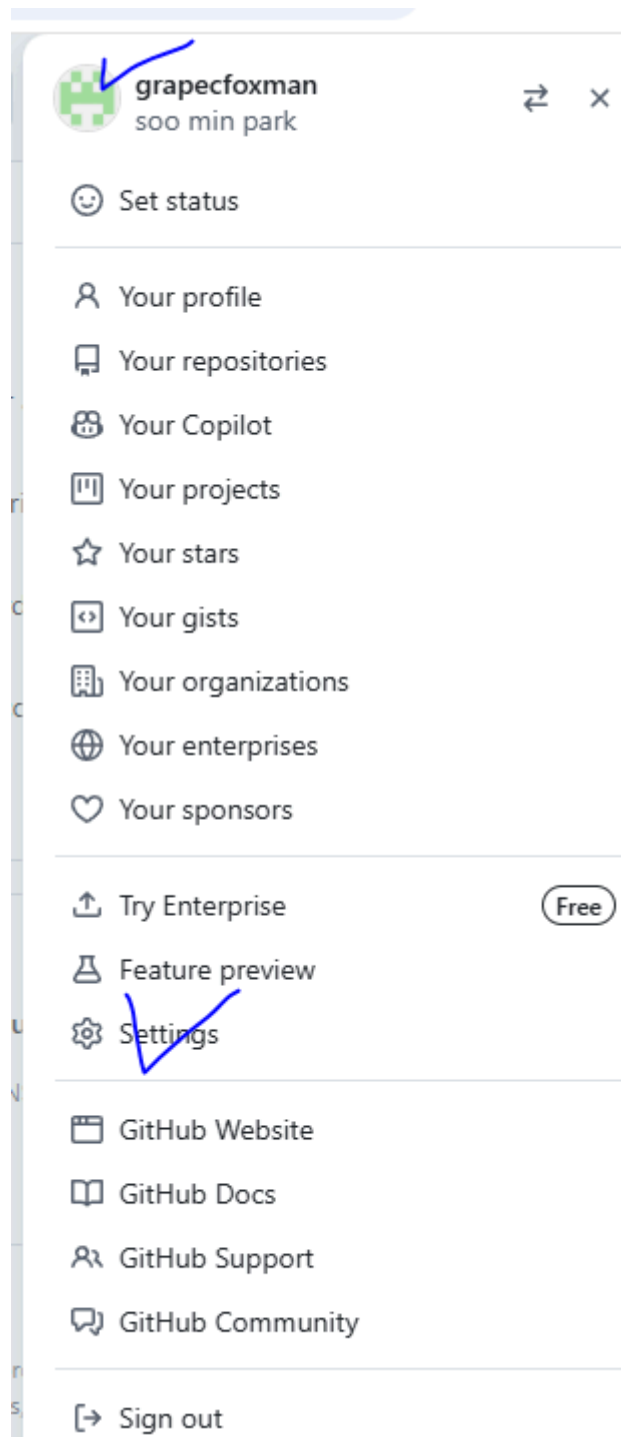
내가 원하는 인증 사이트 등록 방법



다음에 내용을 세팅하기 위해서 **git**를 로그인 한다.



<https://github.com/> 에서 오른쪽 프로필을 클릭한 다음 **setting**를 선택한다.



🔧 Applications

🕒 Scheduled reminders

Archives


📖 Security log

📖 Sponsorship log

<>  Developer settings

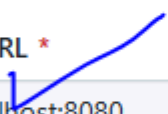
## Register a new OAuth app

**Application name \***

 myauth

Something users will recognize and trust.

**Homepage URL \***

 https://localhost:8080

The full URL to your application homepage.

**Application description**

Application description is optional

This is displayed to all users of your application.

**Authorization callback URL \***



Your application's callback URL. Read our [OAuth documentation](#) for more information.

☐ **Enable Device Flow**

Allow this OAuth App to authorize users via the Device Flow.

Read the [Device Flow documentation](#) for more information.

**Register application**

[Cancel](#)

이전 슈퍼베이스 화면에 콜백 URL 링크가 있다. 로그인후 돌아갈 위치이다.  
**Register application**를 클릭하면 등록이 완료 된다.



GitHub enabled

Client ID

Ov23liP7huUhauC01WYU

Client Secret

.....



Callback URL (for OAuth)

https://bhetnjcoyvvgbaovthbhb.supabase.co/auth/v1/callback

Copy

## 관련 문서

grapefoxman's Org Free / myauth Enable branching Connect

### API Docs

- GETTING STARTED
  - Introduction
  - Authentication
  - User Management**
- TABLES AND VIEWS
  - Introduction
- STORED PROCEDURES
  - Introduction
- GRAPHQL
  - GraphQL
- MORE RESOURCES
  - Guides

### User Management

Supabase makes it easy to manage your users.

Supabase assigns each user a unique ID. You can reference this ID anywhere you create a `profiles` table references the user using a `user_id` field.

Supabase already has built in the routes to sign up, login, and log out

### Sign Up

Allow your users to sign up and create a new account.

After they have signed up, all interactions using the Supabase JS client

### Log In With Email/Password

If an account is created, users can login to your app.

After they have logged in, all interactions using the Supabase JS client

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width" />
    <script
src="https://cdn.jsdelivr.net/npm/@supabase/supabase-js@2"></script>
  </head>
  <body>
    <h1>Supabase Auth</h1>

    <input type="button" value="login" id="login" />

    <script>
const supabaseUrl = "https://bhetnjcoyvgbaovthbhb.supabase.co";
const supabaseKey =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSI6InJlZiI6ImJoZXRuamNveXZnYmFvdnRoYmhiIiwicm9sZSI6ImFub24iLCJpYXQiOiJlE3NDM3MzcwMDksImV4cCI6ImJA1OTMxMzEwOX0.0LOgM2BSXm5mf-TyXxfYEsd0KUUBTRExZpss5PhwbnQ"; // 🔥 여기에
Supabase에서 생성한 새로운 키를 넣으세요.

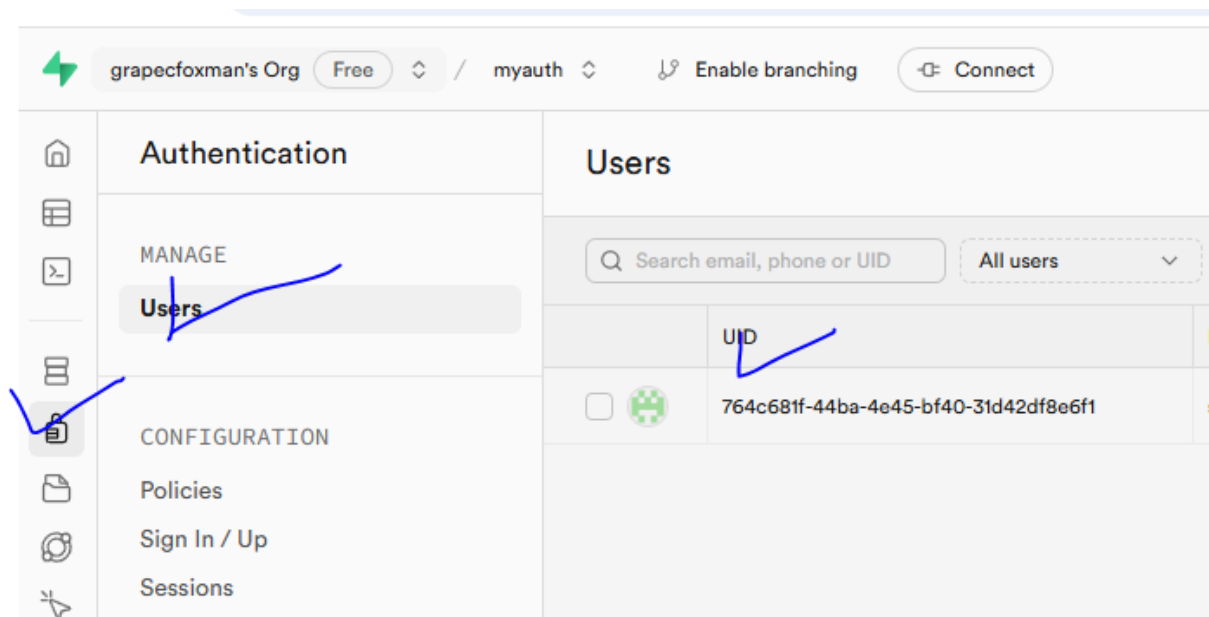
const client = supabase.createClient(supabaseUrl, supabaseKey);

async function signInWithGithub() {
  const { data, error } = await client.auth.signInWithOAuth({
    provider: "github",
    options: {
      redirectTo: "http://localhost:8000",
    },
  });
  console.log(error);
  console.log(data);
}
document
  .querySelector("#login")
  .addEventListener("click", signInWithGithub);
</script>
</body>
</html>

```



슈퍼베이스 에서 인증한 사용자를 확인할 수 있다.



```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Supabase Auth</title>
  <script src="https://cdn.jsdelivr.net/npm/@supabase/supabase-js@2"></script>
</head>
<body>
  <h1>Supabase GitHub 로그인</h1>

  <!-- 로그인 및 로그아웃 버튼 -->
  <button id="login">GitHub 로그인</button>
  <button id="logout" style="display: none;">로그아웃</button>
```

```
<!-- 로그인 상태 메시지 -->
<p id="status">로그아웃</p>
```

```
<script>
  document.addEventListener("DOMContentLoaded", () => {
    // Supabase 설정
    const SUPABASE_URL = "https://roszuiciqchzsrxrssc.supabase.co";
    const SUPABASE_KEY =
      "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSI6InJlZiI6InJvc3p1aWNpcWNoenNyeHByc3NjIiwicm9sZSI6ImFub24iLCJpYXQiOiJlE3NDQyODAxOTUsImV4cCI6ImJA1OTg1NjE5NX0.O2RzTXsyG5AtOPJg4HHaWnc8NbRI-AWi8B4QPBm1tNQ";

    const supabase = window.supabase.createClient(SUPABASE_URL,
SUPABASE_KEY);

    // GitHub OAuth 로그인
    async function signInWithGithub() {
      const { error } = await supabase.auth.signInWithOAuth({
        provider: "github",
        options: { redirectTo: "http://localhost:8000" }
      });
      if (error) console.error("로그인 실패:", error.message);
    }

    // 로그아웃
    async function signOut() {
      const { error } = await supabase.auth.signOut();
      if (error) console.error("로그아웃 실패:", error.message);
      updateUI();
    }

    // 현재 로그인 상태 확인 및 UI 업데이트
    async function updateUI() {
      const { data: { session } } = await supabase.auth.getSession();
      const loginButton = document.getElementById("login");
      const logoutButton = document.getElementById("logout");
      const statusText = document.getElementById("status");

      console.log(session)

      if (session && session.user) {
        loginButton.style.display = "none";
        logoutButton.style.display = "inline";
      }
    }
  });
}
```

```

        statusText.textContent = session.user.email; // 이메일 표시
    } else {
        loginButton.style.display = "inline";
        logoutButton.style.display = "none";
        statusText.textContent = "로그아웃"; // 로그아웃 상태 표시
    }
}

// 이벤트 리스너 추가
document.getElementById("login").addEventListener("click",
signInWithGithub);
document.getElementById("logout").addEventListener("click", signOut);

// 페이지 로드 시 UI 업데이트
updateUI();
});
</script>
</body>
</html>

```

로그인 구성하고 로그아웃 확인할 수 있다.

---

## > 10. 권한

---

! 아래의 코드는 Supabase에서 현재 인증된 사용자의 세션 정보를 가져오는 코드입니다:

js

복사편집

```
const { data: { session } } = await supabase.auth.getSession();
```

이 코드를 이해하려면 다음 요소들을 순서대로 살펴보는 게 좋습니다:

---

◆ 1. `supabase.auth.getSession()` 이란?

- Supabase에서 현재 로그인된 사용자의 세션 정보를 비동기적으로 가져오는 메서드입니다.
- 반환값은 **Promise** 객체이며, 다음과 같은 형태로 객체가 옵니다:

js

복사편집

```
{
  data: {
    session: { /* 세션 객체 */ } // 로그인된 사용자의 세션 정보
  },
  error: null | { /* 에러 객체 */ }
}
```

---

◆ 2. 구조 분해 할당 (`const { data: { session } } = ...`)

이 코드는 위 응답에서 필요한 데이터만 꺼내서 변수로 저장하는 방식입니다.

예를 들어:

js

복사편집

```
const response = await supabase.auth.getSession();
const session = response.data.session;
```

이걸 한 줄로 줄이면:

js

복사편집

```
const { data: { session } } = await supabase.auth.getSession();
```

---

### ◆ 3. session 이란?

**session**은 로그인한 사용자의 세션 정보입니다. 다음과 같은 정보를 담고 있습니다:

js

복사편집

```
{
  access_token: "사용자 액세스 토큰",
  token_type: "bearer",
  expires_in: 3600,
  refresh_token: "리프레시 토큰",
  user: {
    id: "사용자 UUID",
    email: "user@example.com",
    ...
  }
}
```

즉, **session**이 **null**이면 사용자가 로그인하지 않은 상태이고, **session**에 값이 있으면 로그인된 사용자라는 뜻입니다.

---

### ◆ 요약

코드	설명
<code>await</code>	현재 로그인된 세션 정보를 비동기적으로
<code>supabase.auth.getSession()</code>	요청

`data: { session }`

응답 객체 중 `data.session`만 구조 분해  
할당

`session`

로그인한 사용자 정보 및 토큰이 담긴 세션  
객체

---

◆ 예시 사용

js

복사편집

```
const { data: { session } } = await supabase.auth.getSession();
```

```
if (session) {  
  console.log("로그인된 사용자:", session.user.email);  
} else {  
  console.log("로그인 안됨");  
}
```

```
const session = {  
  data: {  
    session: {  
      access_token: "사용자 액세스 토큰",  
      token_type: "bearer",  
      expires_in: 3600,  
      refresh_token: "리프레시 토큰",
```

```

user: {
  id: "사용자 UUID",
  email: "user@example.com",
  created_at: "2021-01-01T00:00:00Z",
  updated_at: "2021-01-01T00:00:00Z",
  user_metadata: {
    nickname: "홍길동",
    profile_image: "https://example.com/image.png",
    bio: "안녕하세요! 저는 홍길동입니다."
  },
  app_metadata: {
    roles: ["admin"],
    provider: "github"
  }
}
};

```

### 1. `user_metadata`는 변경 가능

사용자는 `user_metadata` 내의 값을 자유롭게 업데이트할 수 있습니다. 예를 들어, 사용자가 닉네임을 변경하거나 프로필 사진을 업데이트하려면 `user_metadata`를 수정하면 됩니다:

js

복사편집

```

await supabase.auth.updateUser({
  data: {
    nickname: "새로운닉네임",
    profile_image: "https://example.com/new_image.png"
  }
});

```

이렇게 업데이트된 `nickname`이나 `profile_image`는 `session` 객체 내에서 즉시 반영됩니다.

---

## 2. `user_metadata`의 제한 사항

- 사용자 정의: Supabase는 `user_metadata`에 대해 별도의 제한을 두지 않습니다. 즉, 자유롭게 추가하고 변경할 수 있지만, 앱의 요구 사항에 따라 관리해야 할 책임은 개발자에게 있습니다.
  - 저장 가능 데이터 유형: `user_metadata`는 기본적으로 간단한 데이터를 저장하는 용도로 사용됩니다 (예: 텍스트, URL 등). 복잡한 객체나 대용량 데이터는 다른 테이블에 저장하는 것이 좋습니다.
- 

## 3. 중요한 점: `user_metadata`의 신뢰성

`user_metadata`는 클라이언트 측에서 직접 수정할 수 있기 때문에, 민감한 데이터를 저장하는 데는 적합하지 않습니다. 예를 들어, 사용자의 권한 정보나 비밀번호와 같은 중요한 정보는 `user_metadata`에 담지 않고, 서버 측에서 관리하는 별도의 테이블에서 처리하는 것이 좋습니다.

테이블 만들기



## Create a new table under public

Name

page

Description

Optional

☒ Enable Row Level Security (RLS) Recommended

Restrict access to your table by enabling RLS and writing Postgres policies.



### Policies are required to query data

You need to create an access policy before you can query data from this table. Without a policy, querying this table will return an empty array of results. You can create policies after saving this table.

[Documentation](#)

☐ Enable Realtime

Broadcast changes on this table to authorized subscribers

## Columns

[About data types](#)

[Import data from CSV](#)

Name ?	Type	Default Value ?	Primary
id	# int8	NULL	<input checked="" type="checkbox"/>
created_at	timestarr	now()	<input type="checkbox"/>
title	T text	NULL	<input type="checkbox"/>
body	T text	NULL	<input type="checkbox"/>
user_id	T uuid	auth.uid()	<input type="checkbox"/>

Add column

Cancel

Save

## Add foreign key relationship to page

? What are foreign keys?

Select a schema

auth

Select a table to reference to

users

Select columns from `auth.users` to reference to

public.page

user\_id



auth.users

id



Add another column

Column types will be updated

The following columns will have their types updated to match their referenced column

- `user_id` → `uuid`

? Which action is most appropriate?

Action if referenced row is updated

Cascade

Cascade: Updating a record from `auth.users` will **also update** any records that reference it in this table

Action if referenced row is removed

Docs

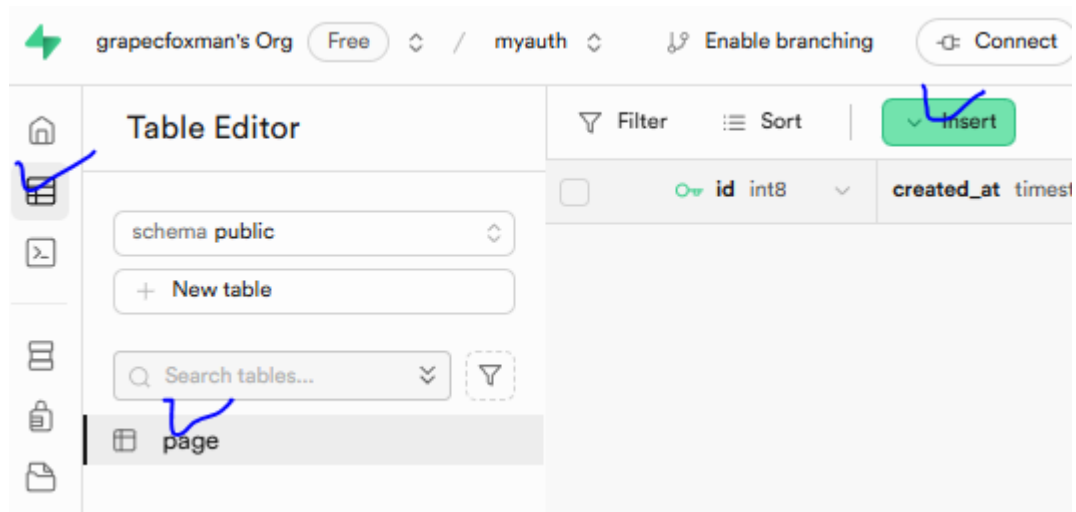
No action

No action: Deleting a record from `auth.users` will **raise an error** if there are records existing in this table that reference it

Cancel

Save

user\_id 사슬 모양 클릭해서 폴린키를 입력 하세요.



테이블을 저장

Add new row to page

id

int8

Automatically generated as identity

created\_at

timestampz

2025-04-03 오후 03:58:57



Default: now()

Your local timezone will be automatically applied (+0900)

### Optional Fields

These are columns that do not need any value

title

text

ㅅㅇㄴㅇ



body

text

ㅅㅇㄴㅇ  
~~~~~



user\_id

uuid

764c681f-44ba-4e45-bf40-31d42df8e6f1



Has a foreign key relation to auth.users.id

Cancel

Save

다음과 같은 권한 설정을 해보자.

|        | owner | authencated | everybody |
|--------|-------|-------------|-----------|
| SELECT | O     | O           | O         |
| INSERT |       | O           |           |
| UPDATE | O     |             |           |
| DELETE | O     |             |           |

myauth / Enable branching / Connect / Feedback / API Docs

Filter | Sort | **Insert** | **+ Add RLS policy** | Role postgres | Realtime off | API Docs

|                          | id int8 | created_at timestamp   | title text | body text |
|--------------------------|---------|------------------------|------------|-----------|
| <input type="checkbox"/> | 1       | 2025-04-03 06:58:57+00 | 인사         | 인사        |

다음을 클릭해서 정책을 추가하자.

schema public / 29321 / Docs

**page** | Disable RLS | **Create policy** |

No policies created yet

×

Update policy: policy\_select\_all

View policy details ▾

🔍

Policy Name

policy\_select\_all

Table on clause

public.page ▾

Policy Behavior as clause

Permissive ▾

Policy Command for clause

☒ SELECT

☐ INSERT

☐ UPDATE

☐ DELETE

☐ ALL

Target Roles to clause

Defaults to all (public) roles if none selected ▾

🔒 USE OPTIONS ABOVE TO EDIT

1 alter policy "policy\_select\_all"

2 on "public"."page"

5 to public

6 using (

7 true

8 );

alter policy "policy\_select\_all"

on "public"."page"

to public

using (

true

);

누구나 볼수 있다.

Add new row to page

id

int8

Automatically generated as identity

created\_at

timestampz

2025-04-03 오후 03:58:57



Default: now()

Your local timezone will be automatically applied (+0900)

### Optional Fields

These are columns that do not need any value

title

text

ㅅㅇㄴㅇ



body

text

ㅅㅇㄴㅇ  
~~~~~



user\_id

uuid

764c681f-44ba-4e45-bf40-31d42df8e6f1



Has a foreign key relation to auth.users.id

Cancel

Save

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Supabase Auth</title>
  <script src="https://cdn.jsdelivr.net/npm/@supabase/supabase-js@2"></script>
</head>
<body>
  <h1>Supabase GitHub 로그인</h1>

  <!-- 로그인 및 로그아웃 버튼 -->
  <button id="login">GitHub 로그인</button>
  <button id="logout" style="display: none;">로그아웃</button>

  <!-- 로그인 상태 메시지 -->
  <p id="status">로그아웃</p>

  <h1>히스토리</h1>
  <div id="history"></div>

  <script>
    document.addEventListener("DOMContentLoaded", () => {
      // Supabase 설정
      const SUPABASE_URL = "https://bhetnjcoyvgbaovthbhb.supabase.co";
      const SUPABASE_KEY =
        "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSI6InJlZiI6ImJoZXRuamNveXZnYmFvdnRoYmhiIiwicm9sZSI6ImFub24iLCJpYXQiOiJlZ3NDM3MzcwMDksImV4cCI6ImJA1OTMxMzEwOX0.0LOgM2BSXm5mf-TyXxfYEds0KUUBTRExZpss5PhwbnQ";
      const supabase = window.supabase.createClient(SUPABASE_URL,
        SUPABASE_KEY);

      // GitHub OAuth 로그인
      async function signInWithGithub() {
        const { error } = await supabase.auth.signInWithOAuth({
          provider: "github",
          options: { redirectTo: "http://localhost:8000" }
        });
        if (error) console.error("로그인 실패:", error.message);
      }
    });
  </script>

```



```

}

// 로그아웃
async function signOut() {
  const { error } = await supabase.auth.signOut();
  if (error) console.error("로그아웃 실패:", error.message);
  updateUI();
}

// 현재 로그인 상태 확인 및 UI 업데이트
async function updateUI() {
  const { data: { session } } = await supabase.auth.getSession();
  const loginButton = document.getElementById("login");
  const logoutButton = document.getElementById("logout");
  const statusText = document.getElementById("status");

  if (session && session.user) {
    loginButton.style.display = "none";
    logoutButton.style.display = "inline";
    statusText.textContent = session.user.email; // 이메일 표시
  } else {
    loginButton.style.display = "inline";
    logoutButton.style.display = "none";
    statusText.textContent = "로그아웃"; // 로그아웃 상태 표시
  }
}

// 히스토리 데이터 가져오기
async function refreshHistory() {
  let { data: record, error } = await supabase.from("page").select("*");
  // 🔥 client → supabase 변경
  if (error) {
    console.error("데이터 불러오기 실패:", error.message);
    return;
  }

  let tag = "";
  for (let i = 0; i < record.length; i++) {
    tag += `<div style="margin:20px
0"><h2>${record[i].title}</h2>${record[i].body}</div>`;
  }
  document.querySelector("#history").innerHTML = tag;
}

```

```
}

// 이벤트 리스너 추가
document.getElementById("login").addEventListener("click",
signInWithGithub);
document.getElementById("logout").addEventListener("click", signOut);

// 페이지 로드 시 실행
updateUI();
refreshHistory();
});
</script>
</body>
</html>
```



## Update policy: policy\_page\_insert



[View policy details](#) ▾

Policy Name

policy\_page\_insert

Table on clause

public.page ▾

Policy Behavior as clause

Permissive ▾

Policy Command for clause

☐ SELECT

☒ INSERT

☐ UPDATE

☐ DELETE

☐ ALL

Target Roles to clause

authenticated × ▾



USE OPTIONS ABOVE TO EDIT

```
1 alter policy "policy_page_insert"
2 on "public"."page"
5 to authenticated
6 with check (
7   true
8 );
```

```

<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Supabase Auth</title>
  <script src="https://cdn.jsdelivr.net/npm/@supabase/supabase-js@2"></script>
</head>
<body>
  <h1>Supabase GitHub 로그인</h1>

  <!-- 로그인 및 로그아웃 버튼 -->
  <button id="login">GitHub 로그인</button>
  <button id="logout" style="display: none;">로그아웃</button>

  <!-- 로그인 상태 메시지 -->
  <p id="status">로그인이 필요합니다.</p>

  <h1>히스토리</h1>
  <div id="history"></div>

  <!-- 데이터 입력 폼 -->
  <h2>새 기록 추가</h2>
  <input type="text" id="title" placeholder="제목 입력" />
  <input type="text" id="body" placeholder="내용 입력" />
  <button id="create_btn">저장</button>

  <script>
    document.addEventListener("DOMContentLoaded", () => {
      // Supabase 설정
      const SUPABASE_URL = "https://bhetnjcoyvgbaovthbhb.supabase.co";
      const SUPABASE_KEY =
        "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSI6InJlZiI6ImJoZXRUamNveXZnYmFvdnRoYmhiIiwicm9sZSI6ImFub24iLCJpYXQiOiJlZ3NDM3MzcwMDksImV4cCI6ImMjA1OTMxMzEwOX00L0LogM2BSXm5mf-TyXxfYEds0KUUBTRExZpss5PhwbnQ";
      const supabase = window.supabase.createClient(SUPABASE_URL,
SUPABASE_KEY);

```

```

let currentUser = null;

// GitHub OAuth 로그인
async function signInWithGithub() {
  const { error } = await supabase.auth.signInWithOAuth({
    provider: "github",
    options: { redirectTo: "http://localhost:8000" }
  });
  if (error) console.error("로그인 실패:", error.message);
}

// 로그아웃
async function signOut() {
  const { error } = await supabase.auth.signOut();
  if (error) console.error("로그아웃 실패:", error.message);
  updateUI();
}

// 현재 로그인 상태 확인 및 UI 업데이트
async function updateUI() {
  const { data: { session } } = await supabase.auth.getSession();
  const loginButton = document.getElementById("login");
  const logoutButton = document.getElementById("logout");
  const statusText = document.getElementById("status");

  if (session && session.user) {
    currentUser = session.user;
    loginButton.style.display = "none";
    logoutButton.style.display = "inline";
    statusText.textContent = `로그인됨: ${session.user.email}`;
  } else {
    currentUser = null;
    loginButton.style.display = "inline";
    logoutButton.style.display = "none";
    statusText.textContent = "로그인이 필요합니다.";
  }
}

// 히스토리 데이터 가져오기
async function refreshHistory() {
  let { data: record, error } = await supabase.from("page").select("*");
  if (error) {

```

```

        console.error("데이터 불러오기 실패:", error.message);
        return;
    }

    let tag = "";
    for (let i = 0; i < record.length; i++) {
        tag += `<div style="margin:20px
0"><h2>${record[i].title}</h2>${record[i].body}</div>`;
    }
    document.querySelector("#history").innerHTML = tag;
}

// 기록 추가 (로그인 확인 추가)
async function recordHandler() {
    if (!currentUser) {
        alert("로그인 후 데이터를 저장할 수 있습니다.");
        return;
    }

    const title = document.getElementById("title").value.trim();
    const body = document.getElementById("body").value.trim();

    if (!title || !body) {
        alert("제목과 내용을 입력하세요.");
        return;
    }

    const { error } = await supabase.from("page").insert([{ title, body
}]);

    if (error) {
        console.error("데이터 저장 실패:", error.message);
        return;
    }

    document.getElementById("title").value = "";
    document.getElementById("body").value = "";
    refreshHistory();
}

// 이벤트 리스너 추가
document.getElementById("login").addEventListener("click",
signInWithGithub);

```

```

document.getElementById("logout").addEventListener("click", signOut);
document.getElementById("create_btn").addEventListener("click",
recordHandler);

// 페이지 로드 시 실행
updateUI();
refreshHistory();
});
</script>
</body>
</html>

```

로그인 되면 자동으로 세팅됨

auth Enable branching Connect

<input type="checkbox"/>	Filter	Sort	Insert						
<input type="checkbox"/>	id int8	created_at timestampz	title text	body text	user_id uuid				
<input type="checkbox"/>	1	2025-04-03 06:58:57+00	ㄱ ㄷ ㄴ ㄱ	ㄱ ㄷ ㄴ ㄱ	764c681f-44ba-4e45-bf40-31d42...				
<input type="checkbox"/>	3	2025-04-04 07:49:46.440827+( tset	tset	test	764c681f-44ba-4e45-bf40-31d42...				
<input type="checkbox"/>	4	2025-04-04 07:50:52.552926+( ㄴ	ㄴ	ㄴ	764c681f-44ba-4e45-bf40-31d42...				

다음은 delete기능을 추가한 상태입니다.



## Update policy: policy\_page\_delete



View policy details ▾

Policy Name

policy\_page\_delete

Table on clause

public.page

Policy Behavior as clause

Permissive ▾

Policy Command for clause

☐ SELECT

☐ INSERT

☐ UPDATE

☒ DELETE

☐ ALL

Target Roles to clause

authenticated × ▾



USE OPTIONS ABOVE TO EDIT

```
1 alter policy "policy_page_delete"
2 on "public"."page"
5 to authenticated
6 using (
7   (auth.uid() = user_id)
8 );
```

Cancel

Save policy



```

alter policy "pages_delete"
2
on "public"."pages"
5
to authenticated
6
using (
7
    (auth.uid() = user_id)
8
);

```

```

<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Supabase CRUD</title>
  <script src="https://cdn.jsdelivr.net/npm/@supabase/supabase-js@2"></script>
</head>
<body>
  <h1>Supabase GitHub 로그인</h1>

  <!-- 로그인 및 로그아웃 버튼 -->
  <button id="login">GitHub 로그인</button>
  <button id="logout" style="display: none;">로그아웃</button>

  <!-- 로그인 상태 메시지 -->
  <p id="status">로그인이 필요합니다.</p>

  <h1>히스토리</h1>
  <div id="history"></div>

  <!-- 데이터 입력 폼 -->
  <h2>새 기록 추가</h2>

```

```
<input type="text" id="title" placeholder="제목 입력" />
<input type="text" id="body" placeholder="내용 입력" />
<button id="create_btn">저장</button>
```

```
<script>
  document.addEventListener("DOMContentLoaded", () => {
    // Supabase 설정
    const SUPABASE_URL = "https://roszuiciqchzsrpxrssc.supabase.co";
    const SUPABASE_KEY =
      "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIsInJlZiI6InJvc3p1aWNpcWNoenNyeHByc3NjIiwicm9sZSI6ImFub24iLCJpYXQiOiJlE3NDQyODAxOTUsImV4cCI6MjA1OTg1NjE5NX0.O2RzTXsyG5AtOPJg4HHaWnc8NbRI-AWi8B4QPBm1tNQ";
    const supabase = window.supabase.createClient(SUPABASE_URL,
      SUPABASE_KEY);

    let currentUser = null;

    // GitHub OAuth 로그인
    async function signInWithGithub() {
      const { error } = await supabase.auth.signInWithOAuth({
        provider: "github",
        options: { redirectTo: "http://192.168.0.50:8000" }
      });
      if (error) console.error("로그인 실패:", error.message);
    }

    // 로그아웃
    async function signOut() {
      const { error } = await supabase.auth.signOut();
      if (error) console.error("로그아웃 실패:", error.message);
      updateUI();
    }

    // 현재 로그인 상태 확인 및 UI 업데이트
    async function updateUI() {
      const { data: { session } } = await supabase.auth.getSession();
      const loginButton = document.getElementById("login");
```

```

const logoutButton = document.getElementById("logout");
const statusText = document.getElementById("status");

if (session && session.user) {
  currentUser = session.user;
  loginButton.style.display = "none";
  logoutButton.style.display = "inline";
  statusText.textContent = `로그인됨: ${session.user.email}`;
} else {
  currentUser = null;
  loginButton.style.display = "inline";
  logoutButton.style.display = "none";
  statusText.textContent = "로그인이 필요합니다.";
}
}

// 히스토리 데이터 가져오기
async function refreshHistory() {
  let { data: records, error } = await
supabase.from("pages").select("*");
  if (error) {
    console.error("데이터 불러오기 실패:", error.message);
    return;
  }

  let content = "";
  records.forEach(record => {
    content += `
    <div style="margin:20px 0">
      <h2>${record.title}</h2>
      <p>${record.body}</p>
      <button onclick="deleteRecord(${record.id})">삭제</button>
    </div>
    `;
  });

  document.querySelector("#history").innerHTML = content;
}

```

```

// 기록 추가 (로그인 확인 추가)
async function recordHandler() {
  if (!currentUser) {
    alert("로그인 후 데이터를 저장할 수 있습니다.");
    return;
  }

  const title = document.getElementById("title").value.trim();
  const body = document.getElementById("body").value.trim();

  if (!title || !body) {
    alert("제목과 내용을 입력하세요.");
    return;
  }

  const { error } = await supabase.from("pages").insert([{ title, body
}]);

  if (error) {
    console.error("데이터 저장 실패:", error.message);
    return;
  }

  document.getElementById("title").value = "";
  document.getElementById("body").value = "";
  refreshHistory();
}

// 기록 삭제
async function deleteRecord(id) {
  if (!confirm("정말 삭제하시겠습니까?")) {
    return;
  }

  try {

```

```

        const { error } = await supabase.from("pages").delete().eq("id",
id);

        if (error) throw error;
        refreshHistory();
    } catch (error) {
        console.error("삭제 실패:", error.message);
    }
}

// 이벤트 리스너 추가
document.getElementById("login").addEventListener("click",
signInWithGithub);
document.getElementById("logout").addEventListener("click", signOut);
document.getElementById("create_btn").addEventListener("click",
recordHandler);

// 초기 실행
updateUI();
refreshHistory();

// 삭제 함수를 전역으로 사용 가능하게 설정
window.deleteRecord = deleteRecord;
});
</script>
</body>
</html>

```

삭제 작업

### 3. USING vs WITH CHECK 차이점

항목	USING	WITH CHECK
주요 용도	JOIN 연산	CHECK 제약 조건 또는 VIEW 조건 검증

사용 위치	<code>JOIN</code> 에서 공통 칼럼을 지정	<code>CHECK</code> 제약 조건, <code>WITH CHECK OPTION</code> (뷰)에서 사용
예제	<code>JOIN table1 USING (id)</code>	<code>CHECK (salary &gt;= 3000), WITH CHECK OPTION</code>

즉, `USING`은 `JOIN`에서 컬럼을 매칭할 때 사용되고, `WITH CHECK`는 데이터 무결성을 유지하기 위한 제약 조건을 설정할 때 사용됩니다.



## Update policy: policy\_page\_update



View policy details ▾

Policy Name

policy\_page\_update

Table on clause

public.page

Policy Behavior as clause

Permissive ▾

Policy Command for clause

☐ SELECT

☐ INSERT

☒ UPDATE

☐ DELETE

☐ ALL

Target Roles to clause

authenticated × ▾



USE OPTIONS ABOVE TO EDIT

```
1 alter policy "policy_page_update"
2 on "public"."page"
5 to authenticated
6 using (
7   (auth.uid() = user_id)
8 with check (
9   (auth.uid() = user_id)
10 );
```

☒ Use check expression


Cancel

Save policy

```
alter policy "policy_page_update"
on "public"."page"
to authenticated
using (
  (auth.uid() = user_id)
with check (
  (auth.uid() = user_id)
);
```

이 코드는 Supabase의 RLS(Row-Level Security) 정책 설정 화면이며, 특정 사용자가 자신에게 속한 데이터만 수정할 수 있도록 제어하는 정책을 보여줍니다. 아래에 각 줄을 한 줄씩 해석해드릴게요.


---

 전체 코드

sql

복사편집

```
alter policy "policy_page_update"
on "public"."page"
to authenticated
using (
  (auth.uid() = user_id)
)
with check (
  (auth.uid() = user_id)
);
```

 한 줄씩 해석

---



### 1. alter policy "policy\_page\_update"

👉 "policy\_page\_update"라는 이름의 정책을 수정합니다.

### 2. on "public"."page"

👉 이 정책이 적용될 테이블은 **public** 스키마의 **"page"** 테이블입니다.

### 3. to authenticated

👉 인증된 사용자(**authenticated** 역할을 가진 유저)에게 이 정책을 적용합니다.

---

## 🔒 핵심 로직

### 5. using (auth.uid() = user\_id)

👉 이 조건은 행을 선택(읽기/업데이트) 할 때 사용됩니다.

즉, 현재 로그인한 사용자의 **UID**가 해당 행의 **user\_id**와 같을 때만 접근(조회, 수정 등)이 가능합니다.

### 7. with check (auth.uid() = user\_id)

👉 이 조건은 새로운 행을 삽입하거나 기존 행을 업데이트할 때 사용됩니다.

즉, 변경된 데이터의 **user\_id**가 로그인한 사용자의 **UID**와 같아야만 **INSERT**나 **UPDATE**가 허용됩니다.

---

## ✅ "Use check expression" 체크박스는 뭘 의미하나요?

체크되어 있으므로 **with check** 절이 활성화된 상태입니다.

이 말은 즉, 사용자가 자신의 **UID**로만 행을 수정하거나 삽입할 수 있도록 제한하는 것입니다.

(만약 체크 해제하면 **with check**가 비활성화되어 업데이트/삽입 시 조건 검사를 하지 않게 됩니다.)

---

## 📌 요약

이 정책은 로그인한 사용자가 오직 자신이 소유한(**user\_id**) 페이지 데이터만 수정할 수 있도록 합니다.

- **using**: 어떤 행을 읽거나 수정할 수 있는지 결정
  - **with check**: 어떤 데이터를 삽입하거나 업데이트할 수 있는지 결정
- 

```
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>Supabase CRUD</title>
  <script src="https://cdn.jsdelivr.net/npm/@supabase/supabase-js@2"></script>
</head>
<body>
  <h1>Supabase GitHub 로그인</h1>
  <button id="login">GitHub 로그인</button>
  <button id="logout" style="display: none;">로그아웃</button>
  <p id="status">로그인이 필요합니다.</p>

  <h1>히스토리</h1>
  <div id="history"></div>

  <h2 id="form-title">새 기록 추가</h2>
  <input type="hidden" id="record_id" />
  <input type="text" id="title" placeholder="제목 입력" />
  <input type="text" id="body" placeholder="내용 입력" />
  <button id="create_btn">저장</button>

  <script>
    document.addEventListener("DOMContentLoaded", () => {
      const SUPABASE_URL = "https://bhetnjcoyvgbaovthbhb.supabase.co";
      const SUPABASE_KEY =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSI6InJlZiI6ImJoZXRUamNveXZnYmFvdnRoYmhiIiwicm9sZSI6ImFub24iLCJpYXQiOiJlE3NDM3MzcwMDksImV4cCI6IjA1OTMxMzEwOX0.0LOgM2BSXm5mf-TyXxfYEds0KUUBTRExZpss5PhwbnQ";
    });
  </script>
```

```

const supabase = window.supabase.createClient(SUPABASE_URL,
SUPABASE_KEY);
let currentUser = null;

async function signInWithGithub() {
  const { error } = await supabase.auth.signInWithOAuth({
    provider: "github",
    options: { redirectTo: "http://localhost:8000" }
  });
  if (error) console.error("로그인 실패:", error.message);
}

async function signOut() {
  await supabase.auth.signOut();
  updateUI();
}

async function updateUI() {
  const { data: { session } } = await supabase.auth.getSession();
  document.getElementById("login").style.display = session ? "none" :
"inline";
  document.getElementById("logout").style.display = session ? "inline" :
"none";
  document.getElementById("status").textContent = session ? `로그인됨:
${session.user.email}` : "로그인이 필요합니다.";
  currentUser = session ? session.user : null;
}

async function refreshHistory() {
  let { data: records, error } = await
supabase.from("page").select("*");
  if (error) return console.error("데이터 불러오기 실패:",
error.message);
  document.querySelector("#history").innerHTML = records.map(record => `
    <div style="margin:20px 0">
      <h2>${record.title}</h2>
      <p>${record.body}</p>
      <button onclick="editRecord(${record.id}, '${record.title}',
'${record.body}')">수정</button>
      <button onclick="deleteRecord(${record.id})">삭제</button>
    </div>
  `).join('');
}

```

```
}
```

```
async function recordHandler() {  
  if (!currentUser) return alert("로그인 후 데이터를 저장할 수 있습니다.");  
  const id = document.getElementById("record_id").value;  
  const title = document.getElementById("title").value.trim();  
  const body = document.getElementById("body").value.trim();  
  if (!title || !body) return alert("제목과 내용을 입력하세요.");  
  
  if (id) {  
    await supabase.from("page").update({ title, body }).eq("id", id);  
  } else {  
    await supabase.from("page").insert([{ title, body }]);  
  }  
  resetForm();  
  refreshHistory();  
}
```

```
async function deleteRecord(id) {  
  if (!confirm("정말 삭제하시겠습니까?")) return;  
  await supabase.from("page").delete().eq("id", id);  
  refreshHistory();  
}
```

```
function editRecord(id, title, body) {  
  document.getElementById("record_id").value = id;  
  document.getElementById("title").value = title;  
  document.getElementById("body").value = body;  
  document.getElementById("create_btn").textContent = "수정 완료";  
  document.getElementById("form-title").textContent = "기록 수정";  
}
```

```
function resetForm() {  
  document.getElementById("record_id").value = "";  
  document.getElementById("title").value = "";  
  document.getElementById("body").value = "";  
  document.getElementById("create_btn").textContent = "저장";  
  document.getElementById("form-title").textContent = "새 기록 추가";  
}
```

```
document.getElementById("login").addEventListener("click",  
signInWithGithub);
```

```
        document.getElementById("logout").addEventListener("click", signOut);
        document.getElementById("create_btn").addEventListener("click",
recordHandler);
        updateUI();
        refreshHistory();
        window.deleteRecord = deleteRecord;
        window.editRecord = editRecord;
    });
</script>
</body>
</html>
```

## CREATE POLICY 문법 분석

```
CREATE POLICY policy_name
ON table_name
AS { PERMISSIVE | RESTRICTIVE } //에즈 퍼미시브 또는 리스트릭티브
FOR { ALL | SELECT | INSERT | UPDATE | DELETE }
TO { role_name [, ...] | PUBLIC }
USING ( using_expression )
[ WITH CHECK ( check_expression ) ]
```

## 예제 분석

```
CREATE POLICY admin_can_delete
ON public.page
AS PERMISSIVE
FOR DELETE
TO public
USING (
    EXISTS (
        SELECT 1 FROM users WHERE id = auth.uid() AND role = 'admin'
    )
)
WITH CHECK (
    EXISTS (
        SELECT 1 FROM users WHERE id = auth.uid() AND role = 'admin'
    )
)
```

);

✓ 1. CREATE POLICY admin\_can\_delete

- admin\_can\_delete라는 이름의 정책을 만든다는 뜻.

✓ 2. ON public.page

- 이 정책은 public 스키마의 page 테이블에 적용됨.

✓ 3. AS PERMISSIVE

- 허용적인(**PERMISSIVE**) 정책이란 뜻. 여러 개의 정책이 있을 때 하나라도 조건을 만족하면 허용.
- 반대는 **RESTRICTIVE**로, 여러 정책 중 모두를 만족해야 허용이야.

✓ 4. FOR DELETE

- 이 정책은 DELETE 작업에만 적용됨 (삭제).

✓ 5. TO public

- public은 모든 사용자 역할을 의미해. 즉, 로그인한 모든 유저가 이 정책의 대상이 됨.

✓ 6. USING (...)

- 여기 조건이 true여야 삭제가 가능해.

## 7. with check

해당 값만 입력

조건 설명:

```
EXISTS (  
  SELECT 1 FROM users  
  WHERE id = auth.uid() AND role = 'admin'  
)
```

- - 현재 로그인한 유저의 ID (`auth.uid()`)가 `users` 테이블에 있고
  - 그 유저의 `role`이 `'admin'`이어야 한다.
  - 즉, 관리자인 경우에만 삭제 가능하게 함.

---

### WITH CHECK 절이란?

- `WITH CHECK`은 `INSERT`나 `UPDATE` 작업에서 사용됨.
- 새로 추가되거나 수정되는 값이 이 조건을 만족해야 한다는 의미.
- 

---

### ✨ 요약

키워드

의미

`CREATE`      행 보안 정책 생성  
`POLICY`



AS	조건을 하나라도 만족하면 허용 (기본값)
PERMISSIV E	
FOR	삭제 시 적용됨
DELETE	
TO public	모든 사용자에게 적용
USING	이 조건을 만족해야 해당 작업 허용
WITH CHECK	INSERT/UPDATE에서 변경될 값이 이 조건을 만족해야 함

## PERMISSIVE의 진짜 의미

Row-Level Security(RLS)에서는 \*\*한 테이블에 여러 개의 정책(CREATE POLICY)\*\*을 정의할 수 있어. 이때 그 정책들 간에 \*\*"어떻게 조합해서 최종 허용/거부를 판단할 것인가"\*\*가 중요한데, 그걸 제어하는 게 바로 PERMISSIVE와 RESTRICTIVE야.

---

### ✅ AS PERMISSIVE (기본값)

- 정책 여러 개 중 하나라도 조건을 만족하면 허용
- OR 조건처럼 작동해

예시:

```
CREATE POLICY admin_can_delete
ON page
AS PERMISSIVE
FOR DELETE
TO public
USING (auth.role() = 'admin');
```

```
CREATE POLICY owner_can_delete
ON page
AS PERMISSIVE
FOR DELETE
TO public
USING (auth.uid() = user_id);
```

- 위 두 개의 DELETE 정책이 있으면:

- **admin**이거나
  - 자기 글이면  
👉 삭제 허용됨
- 

! 그럼 AS RESTRICTIVE는?

- 모든 정책 조건을 만족해야 허용됨
- AND 조건처럼 작동해

예시:

sql

복사편집

```
CREATE POLICY policy1
ON page
AS RESTRICTIVE
FOR DELETE
TO public
USING (auth.role() = 'admin');
```

```
CREATE POLICY policy2
ON page
AS RESTRICTIVE
FOR DELETE
TO public
USING (auth.uid() = user_id);
```

- 이 경우엔:

- admin이면서
- 자기 클리어야  
👉 삭제 가능

---

#### 🔧 정리하면

정책 종류	작동 방식	허용 조건 예
-------	-------	---------

PERMISSIVE	OR 조건	A 또는 B 만족
------------	-------	-----------

RESTRICTIVE	AND 조건	A 그리고 B 모두 만족
-------------	--------	---------------

- DELETE든 SELECT든 INSERT든 상관없이 이 방식이 똑같이 적용돼.
- 삭제 관련 조건이 여러 개 있을 때, 허용을 좀 더 유연하게 하고 싶으면 **PERMISSIVE**, \*\*보안을 좀 더 엄격하게 하고 싶으면 **RESTRICTIVE**\*\*를 써.

---

필요하면 **RESTRICTIVE**가 섞였을 때 어떤 식으로 우선순위가 적용되는지도 설명해줄게. 더 궁금한 거 있어?

4o

나의 말:

or와 and를 조합하면 복잡해지겠네

ChatGPT의 말:

완전 맞아! **PERMISSIVE**와 **RESTRICTIVE**를 섞으면 정책이 많아질수록 로직이 머리아파질 수 있어.

그럼에도 그걸 조합하는 이유는, 보안이 중요한 시스템에서 정밀한 제어가 필요하기 때문이다.

---

## PERMISSIVE + RESTRICTIVE 조합의 작동 방식

PostgreSQL의 RLS는 다음 순서대로 정책을 평가해:

1. **RESTRICTIVE** 정책부터 평가
  - 하나라도 조건을 만족 못 하면 → 거부
2. **RESTRICTIVE**를 모두 통과하면,
3. **PERMISSIVE** 정책 중 하나라도 만족하면 → 허용
4. 아니면 → 거부

즉: **RESTRICTIVE**는 필터 역할, **PERMISSIVE**는 허용 역할이야

---

## 예제: 복합 조합

sql

복사편집

-- 정책 1: 관리자만 삭제 가능 (RESTRICTIVE)

CREATE POLICY only\_admin\_can\_delete

ON page

AS RESTRICTIVE

FOR DELETE

TO public

USING (auth.role() = 'admin');

-- 정책 2: admin이거나 글 작성자면 삭제 가능 (PERMISSIVE)

CREATE POLICY admin\_or\_owner\_can\_delete

ON page

```
AS PERMISSIVE
FOR DELETE
TO public
USING (auth.role() = 'admin' OR auth.uid() = user_id);
```

😎 결과:

- 유저가 **admin**이 아니면 → **RESTRICTIVE**에서 걸려서 무조건 거부
- 유저가 **admin**이면 → **PERMISSIVE**까지 가고 → 조건 만족하니까 허용됨
- 자기 글이라도 **admin**이 아니면 삭제 안 됨

---

## > 11. 이메일 권한

---

아까전에는 가입하는 로직 없이 로그인 요청을 했내요.  
가입후 메일 인증 한다음 로그인 하면 됩니다.

```
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <title>Supabase JWT 확인</title>
</head>
<body>
  <h1>🔒 Supabase Access Token 확인기</h1>

  <form id="auth-form">
    <label>
```

이메일:

```
<input type="email" id="email" required>
```

```
</label>
```

```
<br><br>
```

```
<label>
```

비밀번호:

```
<input type="password" id="password" required>
```

```
</label>
```

```
<br><br>
```

```
<button type="submit" id="login-btn">로그인</button>
```

```
<button type="button" id="signup-btn">회원가입</button>
```

```
</form>
```

```
<pre id="output"></pre>
```

```
<script type="module">
```

```
import { createClient } from 'https://esm.sh/@supabase/supabase-js';
```

```
//  본인의 Supabase 프로젝트 정보 입력
```

```
const supabaseUrl = 'https://roszuiciqchzsrpxrssc.supabase.co';
```

```
const supabaseAnonKey =
```

```
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSI6InJlZiI6InJvc3p1aWNPcWNoenNyeHByc3NjIiwicm9sZSI6ImFub24iLCJpYXQiOi0jE3NDQyODAxOTUsImV4cCI6I6MjA1OTg1NjE5NX0.02RzTXsyG5AtOPJg4HHaWnc8NbRI-AWi8B4QPbm1tNQ';
```

```
const supabase = createClient(supabaseUrl, supabaseAnonKey);
```

```
const form = document.getElementById('auth-form');
```

```
const output = document.getElementById('output');
```

```
const loginBtn = document.getElementById('login-btn');
```

```
const signupBtn = document.getElementById('signup-btn');
```

```
//  로그인 기능
```

```
form.addEventListener('submit', async (e) => {
```

```
  e.preventDefault();
```

```

const email = document.getElementById('email').value;
const password = document.getElementById('password').value;

output.textContent = '🔄 로그인 중...';

const { data, error } = await supabase.auth.signInWithPassword({
  email, password });

if (error) {
  output.textContent = `❌ 로그인 실패: ${error.message}`;
  return;
}

const token = data.session.access_token;
output.textContent = `✅ 로그인 성공!\n\n🔑
access_token:\n${token}`;
console.log('🔑 access_token:', token);
});

// ✅ 회원가입 기능
signupBtn.addEventListener('click', async () => {
  const email = document.getElementById('email').value;
  const password = document.getElementById('password').value;

  output.textContent = '🔄 회원가입 중...';

  const { data, error } = await supabase.auth.signUp({ email, password
});

if (error) {
  output.textContent = `❌ 회원가입 실패: ${error.message}`;
  return;
}

output.textContent = `✅ 회원가입 성공! 이메일 인증을 완료해주세요.`;
console.log('📄 회원가입 데이터:', data);

```



```
    });  
</script>  
</body>  
</html>
```

---

## > 1. supabase 소개

---

Supabase에서 데이터를 추가할 때 사용하는 `insert()` 문법은 아래와 같은 형태입니다:

---

### ✅ 기본 문법

js

복사편집

```
const { data, error } = await client  
  .from("테이블명")  
  .insert([  
    { 컬럼1: 값1, 컬럼2: 값2, ... }  
  ]);
```

예시:

js

복사편집

```
const { data, error } = await client  
  .from("human")  
  .insert([  
    { name: "홍길동", age: 28, height: 172.5, birthday:  
"1996-03-21T00:00:00" }  
  ]);
```

---

## 상세 설명

요소	설명
<code>from("human")</code>	조작할 테이블 이름 (" <code>human</code> ")
<code>insert([...])</code>	객체 배열 형태로 놓으며, 각 객체가 한 레코드 ( <code>row</code> )를 의미
<code>{ name: ..., age: ..., ... }</code>	컬럼 이름과 값 매핑
<code>await</code>	비동기 호출을 위해 <code>await</code> 사용 ( <code>async</code> 함수 안에서만 사용 가능)
<code>data</code>	삽입된 결과 데이터 (선택적으로 <code>.select()</code> 를 붙이면 포함됨)
<code>error</code>	오류가 생기면 여기에 저장됨

---

## ✨ 옵션: `.select()`로 삽입 결과 확인

js

복사편집

```
const { data, error } = await client
  .from("human")
  .insert([
    { name: "김철수", age: 30, height: 180.0, birthday: "1993-01-01T00:00:00" }
  ])
  .select(); // 삽입된 행의 정보를 다시 가져올 수 있음
```

---

## 여러 개 한 번에 넣기

js

복사편집

```
await client.from("human").insert([
  { name: "이영희", age: 25, height: 165.2, birthday: "1999-07-15T00:00:00" },
  { name: "박민수", age: 27, height: 178.9, birthday: "1997-02-02T00:00:00" }
]);
```

---

필드 타입은 Supabase에서 사용하는 PostgreSQL 타입과 일치해야 하고, 날짜/시간 필드는 ISO 8601 형식으로 넣는 게 좋아요 (YYYY-MM-DDTHH:mm:ss 형식).

필요하다면 **default**, **null**, **auto-increment** 컬럼도 활용 가능합니다!

더 깊게 들어가고 싶으면 예외 처리나 유효성 검사 쪽도 도와줄게요!

**update()** 문법은 특정 조건에 맞는 행(row)을 업데이트(수정)할 때 사용합니다.

---

## ✅ 기본 문법

js

복사편집

```
const { data, error } = await client
  .from("테이블명")
  .update({ 컬럼1: 새로운값1, 컬럼2: 새로운값2, ... })
  .eq("조건컬럼", 조건값);
```

---

## ✨ 예시

js

복사편집

```
const { data, error } = await client
  .from("human")
  .update({ name: "김수정", age: 29 })
  .eq("id", 3); // id가 3번인 행만 수정
```

---

## 📌 주요 메서드 설명

메서드

설명

`.from("테이블명")` 어떤 테이블에서 작업할지 지정

`.update({ ... })` 수정할 컬럼과 새로운 값 지정

`.eq("컬럼", 값)` WHERE 조건 (지정한 컬럼이 해당 값일 때만 적용됨)

---

## 여러 조건 쓰기 (AND 조건)

js

복사편집

```
.update({ height: 180 })
.eq("name", "홍길동")
.eq("age", 28)
```

---

## 조건 없이 쓰면 모든 행이 바뀔 수 있어요

js

복사편집

```
// 📌 이런 식으로 조건 없이 쓰면 전체 테이블이 수정될 수 있으니 조심!
.update({ name: "전체이름수정됨" });
```

---

## 전체 흐름 예시 (HTML input에서 값 받아와 수정)

js

복사편집

```
async function updateRecord() {
  const id = prompt("수정할 ID:");
  const name = document.getElementById("name").value;
  const age = parseInt(document.getElementById("age").value);
```

```

const { data, error } = await client
  .from("human")
  .update({ name, age })
  .eq("id", id);

if (error) {
  alert("수정 오류: " + error.message);
} else {
  console.log("수정 성공:", data);
}
}

```

---

## 추가 팁

- `.eq()` 외에도 `.gt()`, `.lt()`, `.like()` 같은 다양한 조건 메서드를 쓸 수 있어요.
- `.select()`를 뒤에 붙이면 수정된 결과를 다시 받아볼 수 있어요.

js

복사편집

```

.update({ name: "변경된이름" })
.eq("id", 1)
.select(); // 업데이트된 행을 리턴

```

여기서 말씀하신 "조건 여러 개", "조건 없이 전체 수정", "트랜잭션처럼 처리하기"에 대한 예제를 하나씩 설명드릴게요.

---

### 1. 조건 여러 개 (AND 조건)

여러 조건을 이용해 한 번에 여러 행을 수정하려면 `eq()` 외에도 다양한 조건 메서드를 사용할 수 있습니다. 예를 들어, `name`이 "홍길동"이고, `age`가 30인 사람의 `height`를 175로 수정하는 경우는 다음과 같습니다.

예시: 여러 조건으로 업데이트

js

복사편집

```
const { data, error } = await client
  .from("human")
  .update({ height: 175 })
  .eq("name", "홍길동")      // name이 "홍길동"인 사람
  .eq("age", 30);            // age가 30인 사람
```

조건 메서드 종류:

- `.eq("컬럼", 값)` : 컬럼이 해당 값일 때
- `.gt("컬럼", 값)` : 컬럼이 해당 값보다 클 때
- `.lt("컬럼", 값)` : 컬럼이 해당 값보다 작을 때
- `.gte("컬럼", 값)` : 컬럼이 해당 값보다 크거나 같을 때
- `.lte("컬럼", 값)` : 컬럼이 해당 값보다 작거나 같을 때
- `.like("컬럼", 패턴)` : 컬럼이 패턴과 일치할 때

예를 들어, 나이가 20살 이상이고, 이름이 "홍길동"인 사람들만 키를 180으로 바꾸는 방법은 아래와 같습니다.

js

복사편집

```
const { data, error } = await client
  .from("human")
  .update({ height: 180 })
  .gte("age", 20)            // age가 20 이상인 사람
  .eq("name", "홍길동");    // name이 "홍길동"인 사람
```

---

## 2. 조건 없이 전체 수정

조건을 생략하고 전체 테이블의 데이터를 수정하려면 단순히 `.eq()` 조건을 빼고, `update()`만 사용하면 됩니다.

예시: 조건 없이 전체 수정

js

복사편집

```
const { data, error } = await client
  .from("human")
  .update({ height: 180 }); // 모든 사람의 height를 180으로 수정
```

이 경우, `human` 테이블의 모든 행이 수정됩니다. 주의해야 할 점은 전체 업데이트가 되므로, `WHERE` 조건을 꼭 사용하여 수정하려는 행을 특정해야 합니다.

---

### 3. 트랜잭션처럼 처리하기 (여러 작업 묶기)

Supabase는 기본적으로 SQL 트랜잭션을 명시적으로 지원하지 않지만, 여러 작업을 순차적으로 처리하면서 그 중 하나라도 실패하면 전부 롤백되도록 구현할 수 있습니다. 이를 위해 `async/await`을 사용한 순차적 처리가 필요합니다.

예시: 트랜잭션처럼 처리하기

js

복사편집

```
async function updateMultipleRecords() {
  try {
    // 첫 번째 업데이트
    const { data: data1, error: error1 } = await client
      .from("human")
      .update({ height: 170 })
      .eq("id", 1);
    if (error1) throw new Error(error1.message);

    // 두 번째 업데이트
    const { data: data2, error: error2 } = await client
      .from("human")
      .update({ height: 180 })
      .eq("id", 2);
    if (error2) throw new Error(error2.message);
  }
}
```

```

// 세 번째 업데이트
const { data: data3, error: error3 } = await client
  .from("human")
  .update({ height: 190 })
  .eq("id", 3);
if (error3) throw new Error(error3.message);

console.log("모든 업데이트 성공:", data1, data2, data3);
} catch (err) {
  console.error("업데이트 중 오류 발생:", err.message);
  // 하나라도 실패하면 전부 롤백하고, 에러 메시지를 콘솔에 출력합니다.
}
}

```

트랜잭션처럼 동작하는 이유:

- 각 **update** 작업에 대해 오류가 발생하면 **throw** 문으로 예외를 발생시키고, 전체 작업이 취소되도록 처리할 수 있습니다.
- 만약 모든 업데이트가 성공적으로 끝나면 로그를 출력하여 완료된 상태를 알 수 있습니다.



## 추가 팁

- Supabase는 자동 커밋 방식을 사용하므로 명시적인 트랜잭션을 지원하지 않지만, **try/catch** 문을 사용하여 예외 처리를 잘 해주면 유사하게 구현할 수 있습니다.
- 예를 들어, **insert**나 **delete**도 마찬가지로 여러 작업을 순차적으로 처리하고 그 중 하나라도 실패하면 롤백할 수 있게 처리할 수 있습니다.

Supabase에서 **delete**를 사용하여 데이터를 삭제하는 문법은 매우 간단합니다. 아래에 기본적인 **delete** 문법과 조건을 추가하여 데이터를 삭제하는 방법을 설명드리겠습니다.



---

## 1. 기본 **delete** 문법

조건 없이 특정 테이블에서 데이터를 삭제하는 기본적인 방법은 다음과 같습니다.

예시: 전체 삭제

js

복사편집

```
const { data, error } = await client
  .from("human")
  .delete(); // 'human' 테이블의 모든 데이터를 삭제
```

이렇게 사용하면 테이블에 있는 모든 데이터가 삭제됩니다. 하지만 대부분의 경우, 조건을 걸어 특정 데이터를 삭제하는 방식이 필요합니다.

---

## 2. 조건을 넣어 삭제하기

특정 조건을 넣어서 데이터를 삭제할 수 있습니다. 예를 들어, **id**가 1인 사람의 데이터를 삭제하려면 아래와 같이 작성합니다.

예시: 조건에 맞는 데이터 삭제

js

복사편집

```
const { data, error } = await client
  .from("human")
  .delete()
  .eq("id", 1); // 'id'가 1인 데이터를 삭제
```

여기서 **.eq("id", 1)**은 **id** 값이 1인 데이터를 삭제하는 조건입니다. 다양한 조건을 추가하여 원하는 데이터만 삭제할 수 있습니다.

---

## 3. 여러 조건으로 삭제하기

여러 조건을 결합하여 데이터를 삭제할 수 있습니다. 예를 들어, **name**이 "홍길동"이고, **age**가 30인 데이터를 삭제하려면 아래와 같이 작성할 수 있습니다.

예시: 여러 조건으로 삭제

js

복사편집

```
const { data, error } = await client
  .from("human")
  .delete()
  .eq("name", "홍길동") // 'name'이 "홍길동"인 사람
  .eq("age", 30);       // 'age'가 30인 사람
```

이렇게 여러 조건을 결합하여 삭제할 수 있습니다.

---

#### 4. 조건 없이 전체 데이터 삭제하기 (주의)

테이블 내의 모든 데이터를 삭제할 때 조건을 생략할 수 있습니다. 주의해야 할 점은 이 작업이 되돌릴 수 없기 때문에 실제 운영 환경에서 사용할 때는 꼭 주의해야 합니다.

예시: 조건 없이 모든 데이터 삭제

js

복사편집

```
const { data, error } = await client
  .from("human")
  .delete(); // 모든 데이터를 삭제
```

---

#### 5. 삭제 후 오류 처리

삭제 작업이 잘 되지 않았을 경우 **error** 객체를 통해 오류를 처리할 수 있습니다. 예를 들어 삭제 작업이 실패하면 오류 메시지를 출력하도록 할 수 있습니다.

예시: 삭제 후 오류 처리

js

복사편집

```
const { data, error } = await client
  .from("human")
  .delete()
  .eq("id", 1); // 'id'가 1인 데이터를 삭제
```

```
if (error) {
  console.error("Delete Error: " + error.message);
} else {
  console.log("Deleted Data:", data);
}
```

---

## 6. 트랜잭션처럼 여러 개의 **delete** 작업 묶기

Supabase에서는 명시적인 트랜잭션을 지원하지 않지만, 여러 개의 **delete** 작업을 순차적으로 실행하면서 실패할 경우 롤백하는 방식으로 처리할 수 있습니다.

예시: 여러 **delete** 작업을 트랜잭션처럼 처리하기

js

복사편집

```
async function deleteMultipleRecords() {
  try {
    // 첫 번째 삭제
    const { data: data1, error: error1 } = await client
      .from("human")
      .delete()
      .eq("id", 1);
    if (error1) throw new Error(error1.message);

    // 두 번째 삭제
    const { data: data2, error: error2 } = await client
      .from("human")
      .delete()
      .eq("id", 2);
    if (error2) throw new Error(error2.message);

    // 세 번째 삭제
    const { data: data3, error: error3 } = await client
      .from("human")
      .delete()
      .eq("id", 3);
    if (error3) throw new Error(error3.message);
  }
}
```

```

    console.log("모든 삭제 작업이 성공했습니다:", data1, data2, data3);
  } catch (err) {
    console.error("삭제 작업 중 오류 발생:", err.message);
    // 하나라도 실패하면 모든 작업을 롤백
  }
}

```

위 코드는 하나라도 오류가 발생하면 나머지 작업을 취소하는 방식입니다. 이를 통해 트랜잭션처럼 여러 삭제 작업을 묶을 수 있습니다.

---

## 추가 팁

- `delete` 작업 후에 `data`는 삭제된 데이터에 대한 정보를 반환하지 않을 수 있습니다. 보통은 삭제가 성공하면 `data`가 `[]` 형태로 반환되며, `error` 객체로 삭제가 실패했을 때의 원인을 알 수 있습니다.
- `delete`는 실제로 데이터를 테이블에서 제거하므로, 이 작업을 신중히 사용해야 하며, 실제 운영 환경에서는 백업을 고려하는 것이 좋습니다.

이 문법들을 이용하면 Supabase에서 원하는 데이터를 효율적으로 삭제할 수 있습니다!

## 1. 기본 `select` 문법

`select`는 기본적으로 테이블에서 모든 데이터를 가져올 때 사용됩니다. 필드를 지정하지 않으면 모든 컬럼이 조회됩니다.

예시: 모든 데이터 조회

js

복사편집

```

const { data, error } = await client
  .from("human")
  .select("*"); // 모든 필드 조회

```

위의 예시에서는 **human** 테이블에서 모든 레코드를 조회합니다. **\***는 모든 필드를 조회하는데 사용됩니다.

---

## 2. 특정 필드만 조회

특정 필드만 조회하고 싶을 때는 조회할 필드를 지정할 수 있습니다.

예시: 특정 필드 조회

js

복사편집

```
const { data, error } = await client
  .from("human")
  .select("name, age, height"); // name, age, height 필드만 조회
```

위의 예시에서는 **name**, **age**, **height** 필드만 조회하여 반환합니다.

---

## 3. 조건에 맞는 데이터 조회

**eq**, **gt**, **lt**, **in** 등 다양한 조건을 사용하여 데이터를 필터링할 수 있습니다. 여러 조건을 결합하여 원하는 데이터를 조회할 수 있습니다.

예시: 특정 조건을 만족하는 데이터 조회

js

복사편집

```
const { data, error } = await client
  .from("human")
  .select("*")
  .eq("age", 25); // 나이가 25인 사람들만 조회
```

위의 예시는 **human** 테이블에서 **age**가 25인 레코드만 조회합니다.

예시: 여러 조건 결합하여 조회

js

복사편집

```
const { data, error } = await client
  .from("human")
```

```
.select("*")
.eq("level", 3) // level이 3인 사람들
.gt("age", 30); // 나이가 30보다 큰 사람들
```

위의 예시는 **level**이 3이고 **age**가 30보다 큰 사람들만 조회합니다. **eq**, **gt**, **lt** 등 다양한 조건을 결합할 수 있습니다.

---

## 4. 정렬 (Order By)

**order** 메서드를 사용하여 데이터를 정렬할 수 있습니다. **asc**(오름차순) 또는 **desc**(내림차순)으로 정렬할 수 있습니다.

예시: 나이 순으로 정렬

js

복사편집

```
const { data, error } = await client
  .from("human")
  .select("*")
  .order("age", { ascending: true }); // 나이를 오름차순으로 정렬
```

위의 예시는 **age** 필드를 기준으로 오름차순으로 데이터를 정렬합니다.

예시: 나이 내림차순 정렬

js

복사편집

```
const { data, error } = await client
  .from("human")
  .select("*")
  .order("age", { ascending: false }); // 나이를 내림차순으로 정렬
```

이 예시는 **age** 필드를 기준으로 내림차순으로 정렬합니다.

---

## 5. 제한 (Limit)

**limit** 메서드를 사용하여 조회할 데이터의 개수를 제한할 수 있습니다.

예시: 상위 5개 레코드 조회

js

복사편집

```
const { data, error } = await client
  .from("human")
  .select("*")
  .limit(5); // 첫 5개 레코드만 조회
```

위의 예시는 **human** 테이블에서 첫 5개의 레코드만 조회합니다.

---

## 6. 특정 레코드 조회 (ID로 조회)

특정 ID 값을 사용하여 한 레코드를 조회할 수 있습니다.

예시: 특정 ID로 레코드 조회

js

복사편집

```
const { data, error } = await client
  .from("human")
  .select("*")
  .eq("id", 1); // id가 1인 레코드 조회
```

위의 예시는 **id**가 1인 레코드를 조회합니다.

---

## 7. 조인 (Join)

Supabase는 기본적으로 SQL에서 제공하는 **JOIN** 기능을 지원하지 않습니다. 그러나, 관련된 데이터를 여러 테이블에서 가져오는 방법은 **foreign** 필드를 사용하는 방식으로 해결할 수 있습니다.

예시: 두 테이블에서 데이터 조회 (조인 방식 유사)

js

복사편집

```
const { data, error } = await client
  .from("order")
```

```
.select("*, human(*)") // 'order' 테이블과 'human' 테이블을 조인
.eq("order_id", 123); // order_id가 123인 주문과 관련된 사람 정보 조회
```

위 예시는 `order` 테이블과 `human` 테이블을 `order`의 `human_id`와 `human`의 `id`를 기준으로 조인하여 데이터를 가져옵니다.

---

## 8. 범위 지정 (Range)

`range` 메서드를 사용하여 데이터의 범위를 지정할 수 있습니다. 이 기능은 페이지네이션에 유용합니다.

예시: 10번째부터 20번째까지 데이터 조회

js

복사편집

```
const { data, error } = await client
  .from("human")
  .select("*")
  .range(10, 20); // 10번째부터 20번째까지의 레코드 조회
```

위의 예시는 `human` 테이블에서 10번째부터 20번째까지의 레코드를 조회합니다.

---

## 9. 결과 처리 및 오류 처리

`select` 메서드를 호출한 후, `data`에는 결과 데이터가, `error`에는 발생한 오류 정보가 담깁니다. 이를 통해 결과를 처리하거나 오류를 처리할 수 있습니다.

예시: 결과 처리 및 오류 처리

js

복사편집

```
const { data, error } = await client
  .from("human")
  .select("*")
  .eq("level", 2);
```

```
if (error) {
```



```
    console.error("조회 오류:", error.message);  
  } else {  
    console.log("조회된 데이터:", data);  
  }  
}
```

위 예시는 `level`이 2인 사람들을 조회하고, 오류가 발생하면 오류 메시지를 출력하며, 결과가 성공적으로 조회되면 데이터를 콘솔에 출력합니다.

---

## 결론

Supabase에서 `select`를 사용하면 데이터를 효율적으로 조회하고 조건을 추가하여 원하는 데이터만 필터링할 수 있습니다. `order`, `limit`, `range` 등의 기능을 통해 결과를 정렬하거나 페이지네이션을 구현할 수도 있습니다. 다양한 조건을 사용하여 복잡한 조회를 수행할 수 있으므로, 요구 사항에 맞는 방식으로 `select`를 활용하시면 됩니다.

---

## > 1. supabase 소개

---

## > 1. supabase 소개

---