VERSIÓN 1.0 | CURSO PROFESIONAL

BASES DE DATOS Y SUPABASE

SQL, PostgreSQL y Backend Profesional

La guía esencial para dominar bases de datos relacionales

Aprende SQL desde comandos básicos hasta consultas avanzadas.
Configura Supabase y conecta tu base de datos a aplicaciones reales.
Domina relaciones, índices y optimización de consultas.

20+ Ejercicios | 5 Proyectos | Backend Listo

Ingeniero de Sistemas

Especializado en Backend y Arquitectura de Datos

Enero 2026 | Actualizado

Índice general

1

Introducción a Bases de Datos

1.1 ¿Qué es una Base de Datos?

Una base de datos es un conjunto organizado de datos almacenados en una computadora de forma estructurada, permitiendo guardar, buscar, actualizar y eliminar información de manera eficiente y segura.

Piensa en una base de datos como: Un archivo inteligente que entiende relaciones, organiza datos en tablas, y permite consultas rápidas sin cargar todo en memoria.

1.2 Tipos de Bases de Datos

Bases de Datos Relacionales (SQL)

Organizan datos en **tablas** conectadas mediante claves:

- Datos estructurados (filas y columnas)
- Relaciones entre tablas mediante claves foráneas
- ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad)
- Ejemplo: PostgreSQL, MySQL, SQL Server

Bases de Datos No Relacionales (NoSQL)

Almacenan datos de forma flexible sin estructura fija:

- Documentos JSON, pares clave-valor, grafos
- Mayor escalabilidad horizontal
- Menor garantía de consistencia
- Ejemplo: MongoDB, Firebase, Redis

En esta guía: Nos enfocamos en **bases de datos relacionales** usando PostgreSQL a través de Supabase.

1.3 Estructura de una Tabla Relacional

Una tabla es como una hoja de Excel con filas y columnas:

id	nombre	edad	email
1	Juan García	16	juan@mail.com
2	María López	15	maria@mail.com
3	Carlos Ruiz	17	carlos@mail.com

1.4 Conceptos Clave

Concepto	Descripción
Tabla	Estructura que almacena datos de un tipo (ej: ESTUDIANTES).
Registro (Fila)	Una línea completa de datos (un estudiante).
Campo (Columna)	Un atributo (nombre, edad, email).
Clave Primaria	Identificador único de cada registro (nunca se repite).
Clave Foránea	Campo que conecta dos tablas (relaciones).
Índice	Acelera búsquedas en campos grandes.

2

SQL: Comandos Básicos

2.1 ¿Qué es SQL?

SQL (Structured Query Language) es el lenguaje estándar para comunicarse con bases de datos relacionales. Funciona de forma similar en PostgreSQL, MySQL, SQL Server, etc.

2.2 Las 4 Operaciones Fundamentales (CRUD)

CRUD es un acrónimo que representa las operaciones básicas:

- **C**reate — Crear datos (INSERT)
- **R**ead — Leer datos (SELECT)
- **U**pdate — Actualizar datos (UPDATE)
- **D**elete — Eliminar datos (DELETE)

2.2.1 CREATE TABLE — Crear una Tabla

Define la estructura de una nueva tabla:

```
CREATE TABLE ESTUDIANTES (
  id SERIAL PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  edad INTEGER,
  email VARCHAR(100) UNIQUE,
  ciudad VARCHAR(50),
  fecha_registro TIMESTAMP DEFAULT NOW()
);
```

Explicación de tipos de datos:

Tipo	Descripción
SERIAL	Número entero que se auto-incrementa.
VARCHAR(n)	Texto de hasta n caracteres.
INTEGER	Número entero.
DECIMAL(p,s)	Número decimal con precisión.
BOOLEAN	Verdadero o falso (true/false).
DATE	Fecha (YYYY-MM-DD).
TIMESTAMP	Fecha y hora.
TEXT	Texto largo sin límite.

Restricciones comunes:

- `PRIMARY KEY` — Identifica únicamente cada registro.
- `NOT NULL` — El campo es obligatorio.
- `UNIQUE` — No pueden haber dos valores iguales.
- `DEFAULT` — Valor automático si no se proporciona.
- `FOREIGN KEY` — Conecta con otra tabla.

2.2.2 INSERT — Insertar Datos

Añade nuevos registros a una tabla:

```
-- Insertar un registro
INSERT INTO ESTUDIANTES (nombre, edad, email, ciudad)
VALUES ('Ana Mart'inez', 15, 'ana@email.com', 'Cartagena');

-- Insertar m'ultiples registros
INSERT INTO ESTUDIANTES (nombre, edad, email, ciudad)
VALUES
  ('Juan Garc'ia', 16, 'juan@email.com', 'Cartagena'),
  ('Mar'ia L'opez', 15, 'maria@email.com', 'Cartagena'),
  ('Carlos Ruiz', 17, 'carlos@email.com', 'Bogot'a');
```

Nota: Si no especificas los campos, debes dar valores en el orden exacto de la tabla. Es mejor ser explícito.

2.2.3 SELECT — Consultar Datos

Lee y filtra datos existentes:

```
-- Ver todos los datos
SELECT * FROM ESTUDIANTES;

-- Ver solo campos espec'ificos
SELECT nombre, email FROM ESTUDIANTES;

-- Con alias (renombrar columnas)
SELECT nombre AS "Nombre Completo", edad AS "Años"
FROM ESTUDIANTES;

-- Ordenar resultados
SELECT * FROM ESTUDIANTES ORDER BY nombre ASC;
SELECT * FROM ESTUDIANTES ORDER BY edad DESC;

-- Limitar resultados
SELECT * FROM ESTUDIANTES LIMIT 5;
SELECT * FROM ESTUDIANTES LIMIT 5 OFFSET 10;
```

2.2.4 UPDATE — Actualizar Datos

Modifica registros existentes:

```sql
-- Actualizar un registro espec'ifico
UPDATE ESTUDIANTES
SET email = 'juan.nuevo@email.com'
WHERE nombre = 'Juan Garc'ia';

-- Actualizar m'ultiples campos
UPDATE ESTUDIANTES
SET email = 'nuevo@email.com', edad = 17
WHERE id = 1;

-- Actualizar basado en c'alculo
UPDATE ESTUDIANTES
SET edad = edad + 1
WHERE ciudad = 'Cartagena';
```

¡Advertencia! Siempre usa una condición `WHERE` clara para evitar actualizar registros equivocados. Prueba primero con `SELECT WHERE`.

2.2.5 DELETE — Eliminar Datos

Borra registros de la tabla:

```sql
-- Eliminar un registro espec'ifico
DELETE FROM ESTUDIANTES WHERE id = 1;

-- Eliminar m'ultiples registros
DELETE FROM ESTUDIANTES WHERE ciudad = 'Bogot'a';

-- Eliminar todos ('usalo con cuidado)
DELETE FROM ESTUDIANTES;
```

¡Cuidado! `DELETE` sin `WHERE` borra toda la tabla. Siempre verifica la condición primero.

3

Consultas Condicionales y Avanzadas

3.1 Cláusula WHERE — Filtros

WHERE permite filtrar datos según condiciones:

Operadores de Comparación

```
-- Igualdad
SELECT * FROM ESTUDIANTES WHERE edad = 16;

-- Desigualdad
SELECT * FROM ESTUDIANTES WHERE edad != 16;
SELECT * FROM ESTUDIANTES WHERE edad <> 16;

-- Mayor que, menor que
SELECT * FROM ESTUDIANTES WHERE edad > 15;
SELECT * FROM ESTUDIANTES WHERE edad >= 16;
SELECT * FROM ESTUDIANTES WHERE edad < 18;
SELECT * FROM ESTUDIANTES WHERE edad <= 17;
```

Operadores Lógicos

```
-- AND: ambas condiciones deben ser verdaderas
SELECT * FROM ESTUDIANTES
WHERE edad >= 15 AND ciudad = 'Cartagena';

-- OR: al menos una condici'on es verdadera
SELECT * FROM ESTUDIANTES
WHERE ciudad = 'Cartagena' OR ciudad = 'Bogot'a';

-- NOT: negar una condici'on
SELECT * FROM ESTUDIANTES
WHERE NOT ciudad = 'Bogot'a';
```

BETWEEN — Rangos

```
-- Edades entre 15 y 17
SELECT * FROM ESTUDIANTES
WHERE edad BETWEEN 15 AND 17;

-- Fechas en rango
```

```
SELECT * FROM ESTUDIANTES
WHERE fecha_registro BETWEEN '2025-01-01' AND '2025-12-31';
```

IN — Valores en Lista

```
-- Buscar varias ciudades
SELECT * FROM ESTUDIANTES
WHERE ciudad IN ('Cartagena', 'Bogot'a', 'Medell'in');

-- Equivalente a m'ultiples OR
SELECT * FROM ESTUDIANTES
WHERE ciudad = 'Cartagena' OR ciudad = 'Bogot'a' OR ciudad = 'Medell'in';
```

LIKE — Búsqueda de Patrones

```
-- Nombres que empiezan con 'J'
SELECT * FROM ESTUDIANTES WHERE nombre LIKE 'J%';

-- Nombres que terminan con ''ia'
SELECT * FROM ESTUDIANTES WHERE nombre LIKE '%'ia';

-- Nombres que contienen 'Garc'ia'
SELECT * FROM ESTUDIANTES WHERE nombre LIKE '%Garc'ia%';

-- Emails con un patr'on espec'ifico
SELECT * FROM ESTUDIANTES
WHERE email LIKE '%.com%' OR email LIKE '%.co%';
```

Símbolos en LIKE:

- % — Cualquier número de caracteres
- _ — Un solo carácter

3.2 GROUP BY — Agrupar Datos

Agrupa registros y calcula estadísticas:

```
-- Contar estudiantes por ciudad
SELECT ciudad, COUNT(*) AS cantidad
FROM ESTUDIANTES
GROUP BY ciudad;

-- Edad promedio por ciudad
SELECT ciudad, AVG(edad) AS edad_promedio
FROM ESTUDIANTES
GROUP BY ciudad;

-- Edad m'inima y m'axima
```

```sql
SELECT
  ciudad,
  MIN(edad) AS edad_minima,
  MAX(edad) AS edad_maxima
FROM ESTUDIANTES
GROUP BY ciudad;
```

3.3 HAVING — Filtrar Grupos

Filtra grupos después de agrupar:

```sql
-- Ciudades con m'as de 2 estudiantes
SELECT ciudad, COUNT(*) AS cantidad
FROM ESTUDIANTES
GROUP BY ciudad
HAVING COUNT(*) > 2;

-- Edad promedio mayor a 16
SELECT ciudad, AVG(edad) AS promedio
FROM ESTUDIANTES
GROUP BY ciudad
HAVING AVG(edad) > 16;
```

3.4 ORDER BY — Ordenar Resultados

Ordena los resultados de manera ascendente o descendente:

```sql
-- Ordenar alfab'eticamente por nombre
SELECT * FROM ESTUDIANTES ORDER BY nombre ASC;

-- Ordenar por edad descendente
SELECT * FROM ESTUDIANTES ORDER BY edad DESC;

-- M'ultiples criterios
SELECT * FROM ESTUDIANTES
ORDER BY ciudad ASC, edad DESC;

-- Ordenar por posici'on de columna
SELECT nombre, edad FROM ESTUDIANTES
ORDER BY 2 DESC;  -- Ordena por la 2 columna (edad)
```

3.5 Funciones de Agregación

Función	Descripción
COUNT(*)	Cuenta el número de registros.

COUNT(campo)	Cuenta registros no NULL en un campo.
SUM(campo)	Suma valores numéricos.
AVG(campo)	Calcula el promedio.
MIN(campo)	Encuentra el valor mínimo.
MAX(campo)	Encuentra el valor máximo.

```
-- Ejemplos de agregaci'on
SELECT COUNT(*) AS total_estudiantes FROM ESTUDIANTES;
SELECT AVG(edad) AS edad_promedio FROM ESTUDIANTES;
SELECT SUM(edad) AS suma_edades FROM ESTUDIANTES;
SELECT MIN(edad), MAX(edad) FROM ESTUDIANTES;
```

4

Relaciones y JOINs

4.1 ¿Por Qué Relaciones?

Las relaciones evitan duplicación de datos y mantienen integridad. En lugar de guardar el nombre del profesor en cada estudiante, guardamos su ID.

4.2 Crear Tablas Relacionadas

```sql
-- Tabla de PROFESORES
CREATE TABLE PROFESORES (
  id SERIAL PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  materia VARCHAR(50),
  email VARCHAR(100)
);

-- Tabla de ESTUDIANTES con clave for'anea
CREATE TABLE ESTUDIANTES (
  id SERIAL PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  edad INTEGER,
  email VARCHAR(100),
  id_profesor INTEGER REFERENCES PROFESORES(id)
);
```

La línea clave: `FOREIGN KEY (id_profesor) REFERENCES PROFESORES(id)`

Esto significa: el campo `id_profesor` debe coincidir con un `id` válido en la tabla PROFESORES.

4.3 Inserts en Tablas Relacionadas

```sql
-- Primero inserta profesores
INSERT INTO PROFESORES (nombre, materia, email)
VALUES
  ('Dr. L'opez', 'Matem'aticas', 'lopez@escuela.com'),
  ('Dra. Garc'ia', 'F'isica', 'garcia@escuela.com');

-- Luego inserta estudiantes con referencia a profesores
INSERT INTO ESTUDIANTES (nombre, edad, email, id_profesor)
VALUES
  ('Juan Garc'ia', 16, 'juan@email.com', 1),
```

```
  ('Mar'ia L'opez', 15, 'maria@email.com', 1),
  ('Carlos Ruiz', 17, 'carlos@email.com', 2);
```

4.4 JOINs — Combinar Tablas

Los JOINs te permiten consultar datos de múltiples tablas conectadas.

INNER JOIN — Solo registros que coinciden

```
-- Ver estudiantes y sus profesores
SELECT
  e.nombre AS estudiante,
  p.nombre AS profesor,
  p.materia
FROM ESTUDIANTES e
INNER JOIN PROFESORES p ON e.id_profesor = p.id;
```

LEFT JOIN — Todos los de la izquierda

```
-- Todos los estudiantes, incluso sin profesor asignado
SELECT
  e.nombre AS estudiante,
  p.nombre AS profesor
FROM ESTUDIANTES e
LEFT JOIN PROFESORES p ON e.id_profesor = p.id;
```

RIGHT JOIN — Todos los de la derecha

```
-- Todos los profesores, incluso sin estudiantes
SELECT
  e.nombre AS estudiante,
  p.nombre AS profesor
FROM ESTUDIANTES e
RIGHT JOIN PROFESORES p ON e.id_profesor = p.id;
```

CROSS JOIN — Producto cartesiano

```
-- Todas las combinaciones posibles
SELECT
  e.nombre,
  p.nombre
FROM ESTUDIANTES e
CROSS JOIN PROFESORES p;
```

5

Supabase: Backend Moderno

5.1 ¿Qué es Supabase?

Supabase es una plataforma que proporciona:

- Base de datos PostgreSQL alojada en la nube
- APIs REST automáticas
- Autenticación de usuarios
- Almacenamiento de archivos
- Actualizaciones en tiempo real (Realtime)

Ventaja principal: No necesitas configurar un servidor, instalar PostgreSQL, ni crear APIs manualmente. Todo está listo para usar.

5.2 ¿Por Qué Supabase y No Hacer Todo Desde Cero?

Aspecto	Desde Cero	Supabase
Configurar servidor	Difícil	Automático
Base de datos	Instalar	Incluida
APIs	Código manual	Automáticas
Autenticación	Compleja	Integrada
Seguridad	Tu responsabilidad	Incluida
Escalabilidad	Complicada	Automática
Costo inicial		Gratis

6

Configuración de Supabase

6.1 Paso 1: Crear una Cuenta

1. Ve a `https://supabase.com`
2. Haz clic en “Sign Up”
3. Regístrate con email o GitHub
4. Verifica tu email
5. Listo para crear proyectos

6.2 Paso 2: Crear un Proyecto

1. En el panel, haz clic en “New Project”
2. Dale un nombre descriptivo (ej: `mi-app-escolar`)
3. Elige una **región cercana** (ej: São Paulo para Latinoamérica)
4. Crea una contraseña segura para la BD
5. Espera 2-3 minutos mientras se crea

Consejo: Anota tu contraseña en un lugar seguro. La necesitarás si quieres conectarte con herramientas externas como pgAdmin.

6.3 Paso 3: Acceder al Panel

Una vez creado el proyecto, verás el panel con varias secciones:

Sección	Descripción
Table Editor	Vista visual estilo Excel de tus tablas.
SQL Editor	Ejecuta comandos SQL directamente.
Database	Estructura de tablas y relaciones.
Auth	Autenticación y usuarios.
Storage	Guarda archivos (imágenes, documentos).
Logs	Historial de errores y eventos.
API Docs	Documentación automática de tu API.

6.4 Paso 4: Crear tu Primera Tabla

En el **SQL Editor**, copia y ejecuta:

```
CREATE TABLE ESTUDIANTES (
  id SERIAL PRIMARY KEY,
```

```
  nombre VARCHAR(100) NOT NULL,
  edad INTEGER,
  email VARCHAR(100) UNIQUE,
  ciudad VARCHAR(50),
  fecha_registro TIMESTAMP DEFAULT NOW()
);
```

Haz clic en “Run” (botón azul). Si todo es correcto, verás “Success!”.

6.5 Paso 5: Insertar Datos

Opción A: Mediante SQL Editor

```
INSERT INTO ESTUDIANTES (nombre, edad, email, ciudad)
VALUES
  ('Juan Garc'ia', 16, 'juan@email.com', 'Cartagena'),
  ('Mar'ia L'opez', 15, 'maria@email.com', 'Cartagena'),
  ('Carlos Ruiz', 17, 'carlos@email.com', 'Bogot'a');
```

Opción B: Mediante Table Editor (interfaz visual)

1. Haz clic en “Table Editor” en el menú izquierdo
2. Selecciona la tabla `ESTUDIANTES`
3. Haz clic en “Insert Row”
4. Llena los datos y haz clic en guardar

6.6 Paso 6: Consultar Datos

En SQL Editor:

```
-- Ver todos
SELECT * FROM ESTUDIANTES;

-- Filtrar
SELECT * FROM ESTUDIANTES WHERE ciudad = 'Cartagena';

-- Contar
SELECT COUNT(*) FROM ESTUDIANTES;
```

O en Table Editor: selecciona la tabla y verás los datos en una vista de tabla visual.

7

Conectar Supabase a Aplicaciones

7.1 Obtener Credenciales

1. Ve a **Project Settings** (icono de engranaje)
2. Busca **API** en el menú
3. Copia:
   - `PROJECT URL` (dirección de tu BD)
   - `anon key` (clave pública para el cliente)
   - `service_role key` (clave privada para el servidor)

¡Seguridad importante! Nunca compartas la `service_role key` en público. Úsala solo en el backend.

7.2 Conexión desde JavaScript/React

Instalación

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

Inicializar Cliente

```
import { createClient } from '@supabase/supabase-js'

const SUPABASE_URL = 'https://tu-proyecto.supabase.co'
const SUPABASE_ANON_KEY = 'tu-anon-key-aqui'

const supabase = createClient(SUPABASE_URL, SUPABASE_ANON_KEY)
```

Leer Datos (SELECT)

```
async function obtenerEstudiantes() {
  const { data, error } = await supabase
    .from('ESTUDIANTES')
    .select('*')

  if (error) console.error('Error:', error)
  else console.log('Datos:', data)
}
```

```
obtenerEstudiantes()
```

Insertar Datos (INSERT)

```
async function agregarEstudiante() {
  const { data, error } = await supabase
    .from('ESTUDIANTES')
    .insert([
      { nombre: 'Ana Mart'inez', edad: 15, email: 'ana@email.com', ciudad: '
   Cartagena' }
    ])

  if (error) console.error('Error:', error)
  else console.log('Inserci'on exitosa:', data)
}

agregarEstudiante()
```

Actualizar Datos (UPDATE)

```
async function actualizarEstudiante(id, nuevosDatos) {
  const { data, error } = await supabase
    .from('ESTUDIANTES')
    .update(nuevosDatos)
    .eq('id', id)

  if (error) console.error('Error:', error)
  else console.log('Actualizaci'on exitosa:', data)
}

actualizarEstudiante(1, { edad: 17 })
```

Eliminar Datos (DELETE)

```
async function eliminarEstudiante(id) {
  const { data, error } = await supabase
    .from('ESTUDIANTES')
    .delete()
    .eq('id', id)

  if (error) console.error('Error:', error)
  else console.log('Eliminaci'on exitosa')
}

eliminarEstudiante(1)
```

7.3 Conexión desde Python

Instalación

```
pip install supabase
```

Inicializar Cliente

```
from supabase import create_client

SUPABASE_URL = "https://tu-proyecto.supabase.co"
SUPABASE_KEY = "tu-anon-key-aqui"

supabase = create_client(SUPABASE_URL, SUPABASE_KEY)
```

Operaciones CRUD

```
# Leer
data = supabase.table("ESTUDIANTES").select("*").execute()
print(data.data)

# Insertar
new_student = {"nombre": "Ana", "edad": 15, "email": "ana@email.com"}
supabase.table("ESTUDIANTES").insert(new_student).execute()

# Actualizar
supabase.table("ESTUDIANTES").update({"edad": 17}).eq("id", 1).execute()

# Eliminar
supabase.table("ESTUDIANTES").delete().eq("id", 1).execute()
```

8

Proyectos Prácticos

8.1 Proyecto 1: Sistema de Biblioteca

Objetivo: Crear una base de datos para una biblioteca donde se registren libros, miembros y préstamos.

Estructura

```
CREATE TABLE LIBROS (
  id SERIAL PRIMARY KEY,
  titulo VARCHAR(200) NOT NULL,
  autor VARCHAR(100),
  isbn VARCHAR(20) UNIQUE,
  anio_publicacion INTEGER,
  cantidad INTEGER DEFAULT 1
);

CREATE TABLE MIEMBROS (
  id SERIAL PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  email VARCHAR(100),
  telefono VARCHAR(15),
  fecha_registro DATE DEFAULT CURRENT_DATE
);

CREATE TABLE PRESTAMOS (
  id SERIAL PRIMARY KEY,
  id_miembro INTEGER NOT NULL REFERENCES MIEMBROS(id),
  id_libro INTEGER NOT NULL REFERENCES LIBROS(id),
  fecha_prestamo DATE DEFAULT CURRENT_DATE,
  fecha_devolucion DATE,
  activo BOOLEAN DEFAULT true
);
```

Datos de Prueba

```
INSERT INTO LIBROS (titulo, autor, isbn, anio_publicacion, cantidad)
VALUES
  ('Cien Años de Soledad', 'Gabriel Garc'ia M'arquez', '978-0060-883286', 1967,
   3),
  ('El Principito', 'Antoine de Saint-Exup'ery', '978-0156-012195', 1943, 5),
  ('1984', 'George Orwell', '978-0451-524935', 1949, 2);
```

```sql
INSERT INTO MIEMBROS (nombre, email, telefono)
VALUES
  ('Juan Garc'ia', 'juan@email.com', '3001234567'),
  ('Mar'ia L'opez', 'maria@email.com', '3009876543');

INSERT INTO PRESTAMOS (id_miembro, id_libro, fecha_prestamo, activo)
VALUES
  (1, 1, '2026-01-20', true),
  (2, 2, '2026-01-15', true);
```

Consultas Útiles

```sql
-- Qu'e libros tiene cada miembro
SELECT
  m.nombre AS miembro,
  l.titulo AS libro,
  p.fecha_prestamo AS desde
FROM PRESTAMOS p
JOIN MIEMBROS m ON p.id_miembro = m.id
JOIN LIBROS l ON p.id_libro = l.id
WHERE p.activo = true;

-- Libros m'as prestados
SELECT
  l.titulo,
  COUNT(p.id) AS veces_prestado
FROM LIBROS l
LEFT JOIN PRESTAMOS p ON l.id = p.id_libro
GROUP BY l.id, l.titulo
ORDER BY veces_prestado DESC;

-- Miembros sin devoluciones pendientes
SELECT m.nombre
FROM MIEMBROS m
WHERE m.id NOT IN (
  SELECT DISTINCT id_miembro
  FROM PRESTAMOS
  WHERE activo = true
);
```

8.2 Proyecto 2: Red Social Simplificada

Objetivo: Una plataforma donde usuarios publican posts y otros comentan.

```sql
CREATE TABLE USUARIOS (
  id SERIAL PRIMARY KEY,
  username VARCHAR(50) UNIQUE NOT NULL,
  email VARCHAR(100) UNIQUE NOT NULL,
```

```
  nombre VARCHAR(100),
  fecha_registro TIMESTAMP DEFAULT NOW()
);

CREATE TABLE POSTS (
  id SERIAL PRIMARY KEY,
  id_usuario INTEGER NOT NULL REFERENCES USUARIOS(id),
  contenido TEXT NOT NULL,
  fecha_creacion TIMESTAMP DEFAULT NOW(),
  likes INTEGER DEFAULT 0
);

CREATE TABLE COMENTARIOS (
  id SERIAL PRIMARY KEY,
  id_post INTEGER NOT NULL REFERENCES POSTS(id),
  id_usuario INTEGER NOT NULL REFERENCES USUARIOS(id),
  contenido TEXT NOT NULL,
  fecha_creacion TIMESTAMP DEFAULT NOW()
);

CREATE TABLE SEGUIDORES (
  id_seguidor INTEGER NOT NULL REFERENCES USUARIOS(id),
  id_seguido INTEGER NOT NULL REFERENCES USUARIOS(id),
  fecha_seguimiento DATE DEFAULT CURRENT_DATE,
  PRIMARY KEY (id_seguidor, id_seguido)
);
```

8.3 Proyecto 3: Sistema de Tienda Online

Objetivo: Gestionar productos, clientes y pedidos.

```
CREATE TABLE CLIENTES (
  id SERIAL PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  email VARCHAR(100) UNIQUE,
  direccion TEXT,
  fecha_registro TIMESTAMP DEFAULT NOW()
);

CREATE TABLE PRODUCTOS (
  id SERIAL PRIMARY KEY,
  nombre VARCHAR(100) NOT NULL,
  descripcion TEXT,
  precio DECIMAL(10, 2) NOT NULL,
  stock INTEGER DEFAULT 0,
  categoria VARCHAR(50)
);

CREATE TABLE PEDIDOS (
  id SERIAL PRIMARY KEY,
```

```
  id_cliente INTEGER NOT NULL REFERENCES CLIENTES(id),
  fecha_pedido TIMESTAMP DEFAULT NOW(),
  estado VARCHAR(20) DEFAULT 'pendiente',
  total DECIMAL(12, 2)
);

CREATE TABLE DETALLES_PEDIDO (
  id SERIAL PRIMARY KEY,
  id_pedido INTEGER NOT NULL REFERENCES PEDIDOS(id),
  id_producto INTEGER NOT NULL REFERENCES PRODUCTOS(id),
  cantidad INTEGER NOT NULL,
  precio_unitario DECIMAL(10, 2)
);
```

9

Ejercicios para Estudiantes

9.1 Nivel 1: Fundamentos

Ejercicio 1.1 — Crear Tabla y Datos

1. En tu proyecto Supabase, crea la tabla `MASCOTAS`:
   - id (PRIMARY KEY)
   - nombre
   - tipo (perro, gato, etc.)
   - edad
   - dueño
2. Inserta 5 registros de mascotas
3. Consulta todos con `SELECT *`
4. Consulta solo mascotas tipo “perro”

Ejercicio 1.2 — UPDATE y DELETE

1. Actualiza la edad de una mascota
2. Elimina la mascota con id = 1
3. Verifica que se eliminó con `SELECT`

9.2 Nivel 2: Relaciones

Ejercicio 2.1 — Crea dos Tablas Relacionadas

1. Tabla `DUEÑOS` (id, nombre, email, teléfono)
2. Tabla `MASCOTAS` (id, nombre, tipo, id_dueño)
3. Inserta 3 dueños y 5 mascotas asignadas a ellos
4. Usa `JOIN` para ver cada mascota con su dueño

Ejercicio 2.2 — Consultas con Agregación

1. Cuenta cuántas mascotas tiene cada dueño
2. Encuentra el dueño con más mascotas
3. Agrupa mascotas por tipo y cuenta

9.3 Nivel 3: Desde tu Aplicación

Ejercicio 3.1 — Conectar con JavaScript

1. Crea un proyecto simple en Replit o CodePen

2. Conecta a Supabase con JavaScript
3. Lee datos de tu tabla `ESTUDIANTES`
4. Muestra los datos en una tabla HTML
5. Crea un formulario para insertar nuevos estudiantes

Ejercicio 3.2 — Aplicación Python

1. Crea un script Python que se conecte a Supabase
2. Lee datos y guárdalos en una lista
3. Calcula estadísticas (edad promedio, estudiante más viejo, etc.)
4. Exporta los datos a CSV

10

Resumen de Comandos Esenciales

10.1 Tabla de Comandos SQL

Comando	Ejemplo
CREAR Y ELIMINAR	
CREATE TABLE	`CREATE TABLE TABLA (id SERIAL PRIMARY KEY, nombre VARCHAR(100));`
DROP TABLE	`DROP TABLE TABLA;`
ALTER TABLE	`ALTER TABLE TABLA ADD COLUMN edad INTEGER;`
INSERTAR	
INSERT	`INSERT INTO TABLA (col1, col2) VALUES (val1, val2);`
INSERT (múltiple)	`INSERT INTO TABLA VALUES (...), (...), (...);`
CONSULTAR	
SELECT básico	`SELECT * FROM TABLA;`
SELECT con filtro	`SELECT * FROM TABLA WHERE edad >18;`
SELECT ordenado	`SELECT * FROM TABLA ORDER BY nombre ASC;`
SELECT limitado	`SELECT * FROM TABLA LIMIT 10;`
ACTUALIZAR	
UPDATE	`UPDATE TABLA SET nombre = 'Nuevo' WHERE id = 1;`
UPDATE (múltiple)	`UPDATE TABLA SET col1 = val1, col2 = val2 WHERE condición;`
ELIMINAR	
DELETE	`DELETE FROM TABLA WHERE id = 1;`
DELETE (condicional)	`DELETE FROM TABLA WHERE edad <18;`
AGREGACIÓN	
COUNT	`SELECT COUNT(*) FROM TABLA;`
SUM	`SELECT SUM(edad) FROM TABLA;`
AVG	`SELECT AVG(edad) FROM TABLA;`
MIN/MAX	`SELECT MIN(edad), MAX(edad) FROM TABLA;`
GROUP BY	`SELECT ciudad, COUNT(*) FROM TABLA GROUP BY ciudad;`
JOINS	

INNER JOIN	`SELECT * FROM T1 INNER JOIN T2 ON T1.id = T2.id_t1;`
LEFT JOIN	`SELECT * FROM T1 LEFT JOIN T2 ON T1.id = T2.id_t1;`

11

Buenas Prácticas y Seguridad

11.1 Antes de Hacer Queries

- Verifica que la condición `WHERE` sea correcta
- Prueba con `SELECT` antes de `UPDATE` o `DELETE`
- Usa transacciones para operaciones críticas
- Respeta las relaciones (FOREIGN KEY)

11.2 Seguridad en Supabase

Nunca hagas esto:

- Publiques tu `service_role key` en GitHub
- Guardes contraseñas en texto plano
- Confíes en validación solo en el cliente
- Uses credenciales directamente en el código

11.3 Seguridad: Usar Variables de Entorno

En JavaScript:

```
// .env.local
VITE_SUPABASE_URL=https://tu-proyecto.supabase.co
VITE_SUPABASE_ANON_KEY=tu-anon-key

// En tu c'odigo
const supabase = createClient(
  import.meta.env.VITE_SUPABASE_URL,
  import.meta.env.VITE_SUPABASE_ANON_KEY
)
```

En Python:

```
import os
from dotenv import load_dotenv

load_dotenv()

SUPABASE_URL = os.getenv("SUPABASE_URL")
SUPABASE_KEY = os.getenv("SUPABASE_KEY")
```

11.4 Indexación para Velocidad

Para columnas que usas frecuentemente en `WHERE`:

```
-- Crear ’indice
CREATE INDEX idx_email ON USUARIOS(email);
CREATE INDEX idx_ciudad ON ESTUDIANTES(ciudad);

-- Esto acelera b’usquedas como:
-- SELECT * FROM USUARIOS WHERE email = ’algo@email.com’;
```

12

Recursos y Referencias

12.1 Documentación Oficial

- Supabase Docs: `https://supabase.com/docs`
- PostgreSQL Documentation: `https://www.postgresql.org/docs`
- SQL Tutorial (W3Schools): `https://www.w3schools.com/sql`

12.2 Tutoriales Interactivos

- SQL Murder Mystery: `https://mystery.knightlab.com`
- Mode Analytics SQL Tutorial: `https://mode.com/sql-tutorial`
- Codecademy SQL: `https://www.codecademy.com/learn/learn-sql`

12.3 Herramientas Útiles

- **pgAdmin:** Gestor visual de PostgreSQL
- **DBeaver:** Cliente SQL profesional (multiplataforma)
- **TablePlus:** Cliente elegante para bases de datos
- **Beekeeper Studio:** SQL IDE moderna

12.4 Proyectos para Practicar

- Crea una aplicación de películas (películas, géneros, reseñas)
- Sistema de inventario para una tienda
- Red social simple (usuarios, posts, comentarios)
- Gestor de tareas con categorías y prioridades
- Base de datos de universidad (estudiantes, cursos, calificaciones)

¡Felicitaciones!

Has completado la guía de Bases de Datos y Supabase.

Ahora tienes las herramientas para crear aplicaciones profesionales con backend robusto.

Próximos pasos:

- Crea tu propio proyecto real
- Aprende sobre optimización de queries
- Explora autenticación y autorización en Supabase
- Domina el almacenamiento de archivos