HOW SOFTWARE WORKS · BUILDER · 15 MIN READ

Design your database: tables, keys and relationships

Most app bugs that lose data start with the tables. Turn your idea into tables, link them with keys, add the rules that stop duplicates and nonsense, and check the design your AI helper writes.

Pass the check (4 of 5) online to earn the “Schema Builder” badge:
https://digitalbuilders757.com/university/study/design-your-database

Start from the nouns

Before any code, list the things your app has to remember. For a workshop app, that's members, workshops and signups. Each kind of thing becomes a table, each detail you keep about it becomes a column, and each single item is a row.

Then list what people do: 'a member signs up for a workshop', 'an instructor sees who's coming'. Every sentence that links two nouns is a relationship you'll need to store.

Keys: how rows find each other

- A primary key is a column whose value is unique for every row and never empty, usually called id. It's how you find, change and point at exactly one row.
- Let the database make ids: bigint generated always as identity counts up 1, 2, 3, while uuid gives random ids that are hard to guess.
- A foreign key is a column that holds another table's primary key, such as signups.workshop_id. Declared with references workshops (id), it makes the database refuse a signup for a workshop that doesn't exist.
- Names and emails make poor keys: they change, and two people can share a name.

Three kinds of relationship

- One-to-many: one workshop has many signups. Put the key on the 'many' side, so signups gets a workshop_id column.
- Many-to-many: members take many workshops, and workshops have many members. Neither table can hold a list, so add a join table, signups, with a member_id and a workshop_id.
- One-to-one: each user has one profile. In Supabase, profiles.id is both its primary key and a foreign key to auth.users (id).

- A join table can also hold facts about the link itself, such as when someone signed up or whether they came.

Habits that save you later

- Keep each fact in one place. The email lives on members and you reach it with a join; copies on every signup drift apart the day someone changes it.
- Use the right type: timestamptz for times, integer cents or numeric for money, boolean for yes or no. As text, '100' sorts before '25' and prices can't be added up.
- Mark required columns not null, and give them a default where one makes sense, such as created_at timestamptz not null default now().
- Add a unique rule wherever a duplicate would be a bug: unique (workshop_id, member_id) means nobody takes two seats in one workshop.
- Decide what happens on delete. With on delete cascade, a member's signups go with them; without it, the database refuses to delete a workshop that still has signups.
- Postgres doesn't index foreign key columns for you. Add an index to the ones you filter or join by often, such as signups.workshop_id.
- Name things in lowercase snake_case, like workshop_id, so you never need quotes in SQL.

Ownership and safety

Rows that belong to someone need an owner column, such as member_id uuid references auth.users (id). Then row level security policies can say 'members see only their own signups'.

Turn RLS on for every table people can reach, and change the design with migrations you can review and replay, not by clicking around in your live database.

Ask AI to design it, then check

Describe your nouns and what people do, then ask: 'Design Postgres tables for this as a Supabase migration, with primary keys, foreign keys, not null and unique rules, and RLS policies. Explain each relationship.'

Then read the answer like a reviewer. Draw the tables and the lines between them on paper, and ask about any column you can't explain.

Design bugs to spot in AI-written schemas

- Lists squeezed into one column, such as tags = 'art,music,games'.
- The same email or name copied into several tables.

- Columns ending in _id with no foreign key behind them.
- Dates, prices or yes-or-no answers stored as text.
- No unique rule where a duplicate would double-book a seat.
- New tables with RLS switched off, or a policy that lets anyone read everything.

Key terms

Table
One kind of thing in a database, like users or orders.

Column
One detail stored for every row, like email.

Primary key
The column whose value is unique for every row and never empty, usually id. Other tables use it to point at that row.

Foreign key
A column that holds another table's primary key, like signups.workshop_id. The database refuses values that don't exist there.

Join table
A table that links two others in a many-to-many relationship, like signups linking members and workshops.

Constraint (database)
A rule the database checks on every row it saves, such as not null, unique or a foreign key.

JOIN (SQL)
Combines rows from two tables that share a value, like orders and the customers who placed them.

Row level security (RLS)
Database rules that decide which rows each user may read or change.

Policy (RLS)
One row level security rule, like 'users can read their own notes'.

Migration
A saved, ordered script that changes the database structure.

Try it: Sketch a database before you build

1. Pick an app idea, such as a book club, and write down its nouns: members, books, meetings.
2. Under each noun, list the details you'd keep, and circle the one that identifies each row.
3. Draw a line for each relationship and label it one-to-many or many-to-many. Add a join table for each many-to-many.
4. Ask your AI helper for the tables as a Supabase migration, with keys, not null and unique rules, and RLS policies.
5. Check its answer against this guide's list of design bugs, and ask it to fix anything you find.

KNOWLEDGE CHECK

Design your database: tables, keys and relationships

5 questions. Circle one answer for each; 4 or more correct passes. Answers are at the back.

1. Each signup row keeps a copy of the member's email. What goes wrong when a member changes their email?

A. Nothing, because Postgres updates every copy for you as soon as the first one changes
B. They can't change it until every one of their signups is deleted
C. Email columns can never be changed once a row is saved
D. Their old signups keep the old address, so the copies disagree

2. Members can join many workshops, and each workshop has many members. How should you store that?

A. A join table, signups, with a member_id and a workshop_id
B. A workshops column on members, with the names separated by commas
C. One table with a yes-or-no column for every workshop
D. A copy of the members table for every workshop

3. signups.workshop_id is a foreign key to workshops (id). What does that do?

A. It hides the column from people who aren't signed in
B. It refuses a signup whose workshop isn't in workshops
C. It makes each workshop_id unique, so a workshop can have one signup
D. It encrypts the id so nobody can read it

4. One member ended up with two seats in the same workshop. Which rule stops that in the database?

A. not null on the signups email column
B. A note on the page asking people to click once
C. unique (workshop_id, member_id) on signups
D. An index on the members table's name column

5. An AI-written table stores prices as text, like '25.00'. Why change it?

A. Text is slower to type into the Supabase table editor
B. Postgres only lets you store prices as yes-or-no values
C. Text columns can't hold a full stop
D. Text sorts '100' before '25' and can't add prices up, so use integer cents or numeric

Official sites

PostgreSQL docs: Constraints: https://www.postgresql.org/docs/current/ddl-constraints.html
Supabase docs: Tables and data: https://supabase.com/docs/guides/database/tables

Supabase docs: Querying joins and nested tables:
https://supabase.com/docs/guides/database/joins-and-nesting

Last reviewed 2026-11-25. AI tools change fast; check each tool's own site for current features, plans and prices.

ANSWER KEY

Design your database: tables, keys and relationships

1. D: Their old signups keep the old address, so the copies disagree
Copies don't update themselves. Keep the email on members only, and reach it from signups with a join.

2. A: A join table, signups, with a member_id and a workshop_id
Many-to-many needs a join table. Each signup row links one member to one workshop, and can hold facts about the link, such as when it was made.

3. B: It refuses a signup whose workshop isn't in workshops
A foreign key checks that the value exists in the other table, so you can't save a signup for a workshop that was never made or has been deleted.

4. C: unique (workshop_id, member_id) on signups
A unique rule on the pair means the database refuses a second row for the same member and workshop, however the request arrives.

5. D: Text sorts '100' before '25' and can't add prices up, so use integer cents or numeric
Text compares letter by letter, so '1' beats '2'. Integer cents or numeric sort and add up correctly.