

Prompts For Initial Design and Brainstorming

Help me brainstorm a database schema for a campus event management system. I need to track events, student registrations, attendance, and feedback. Consider a system used by multiple colleges

Got it 🧯 Let's break this down step by step and design a database schema for a multi-college campus event management system.

We'll cover entities, relationships, and possible tables + fields.

📌 Core Requirements

1. Track multiple colleges.
2. Manage events (workshops, fests, seminars, etc.).
3. Handle student registrations for events.
4. Record attendance.
5. Collect feedback.

📊 Entities & Tables

1. Colleges

- Each college has multiple departments & students.
- Fields:
 - `college_id (PK)`
 - `college_name`
 - `location`
 - `contact_email`

2. Departments (Optional, if granularity needed)

- Belongs to a college.
- Fields:
 - `department_id (PK)`
 - `college_id (FK)`
 - `department_name`

3. Students

- Belongs to a college (and optionally department).
- Fields:
 - `student_id (PK)`
 - `college_id (FK)`
 - `department_id (FK)`
 - `name`
 - `email`
 - `phone`
 - `year_of_study`

4. Events

- Organized by a college, can be open to multiple colleges.
- Fields:
 - `event_id (PK)`
 - `college_id (FK)` (organizer college)
 - `title`
 - `description`
 - `event_type` (seminar, fest, workshop, hackathon)
 - `start_time`
 - `end_time`
 - `venue`
 - `capacity`
 - `is_open_to_other_colleges` (boolean) ⬇

Identify and list all potential edge cases and assumptions for a multi-college event management system. Consider like duplicate student entries, students from different colleges registering for the same event

Good call 👍 — edge cases & assumptions are critical for your design doc, especially since you're working on a multi-college event management system. Here's a structured list you can directly include:

🔗 Assumptions & Edge Cases for Multi-College Event Management System

✅ Assumptions

1. Unique Identifiers

- Each `student_id` is unique across the system (not just within a college).
- Each `event_id` is unique, but events are always tied to a specific college.
- If the same event is hosted by multiple colleges, it should be created as separate events.

2. Student Enrollment

- A student belongs to exactly one college.
- Students may only register for events of their own college, unless explicitly marked as inter-college events.

3. Event Lifecycle

- Events can have statuses: `scheduled`, `ongoing`, `completed`, `cancelled`.
- Attendance and feedback are only valid for `completed` events.

4. Feedback Rules

- Only students who actually attended (`present`) can give feedback.
- Ratings are restricted to an integer scale of 1–5.

5. Data Integrity

- A registration must exist before attendance or feedback is recorded.
- Registration automatically prevents duplicate entries with a (`student_id`, `event_id`) uniqueness constraint.

⚠️ Potential Edge Cases

◆ Student Data

- Duplicate student profiles: Same student registered multiple times (typo in email/name).
- Cross-college registration: Student from College A trying to register for College B's event (if not allowed).
- Alumni/external participants: If allowed, they may not belong to any college.

◆ Event Registration

- Duplicate registrations: Student clicks register multiple times (should be blocked).
- Over-capacity: Registrations exceed event capacity (e.g., hall limit 200 seats).
- Late registration: Student tries to register after event has started or closed.

How can we prevent a student from being marked as 'attended' if they were never registered?

Great catch 🕒 — that's a classic integrity problem in event systems.

You need to enforce business rules both at the database level and the API level.

Here's how you can prevent a student from being marked as `attended` if they were never registered:

✅ Database-Level Safeguards

1. Foreign Key Constraint

- In the `attendance` table, instead of directly linking `(student_id, event_id)`, make it reference the `registrations` table.

```
sql Copy code

CREATE TABLE registrations (
  reg_id INTEGER PRIMARY KEY,
  student_id INTEGER NOT NULL,
  event_id INTEGER NOT NULL,
  UNIQUE(student_id, event_id)
);

CREATE TABLE attendance (
  attendance_id INTEGER PRIMARY KEY,
  reg_id INTEGER NOT NULL,
  status TEXT CHECK(status IN ('present', 'absent')),
  FOREIGN KEY (reg_id) REFERENCES registrations(reg_id) ON DELETE CASCADE
);
```

- ✅ This way, attendance can only exist if a registration exists.

2. Unique Constraint

- Ensure each registration gets only one attendance record:

```
sql Copy code

UNIQUE(reg_id)
```

- ✅ Prevents duplicate attendance entries for the same student.

For Prototype Implementation

Generate the full SQL schema for the database design we just discussed. Provide CREATE TABLE statements for all the entities: colleges, students, events, event_registrations, attendance, and feedback. Ensure the primary keys, foreign keys, and other constraints (like NOT NULL and UNIQUE values) are correctly defined to enforce the relationships and data integrity we discussed. The schema should be self-contained and ready to use with a database like SQLite

Thought for 13s

Here's a self-contained SQLite schema with proper keys, constraints, sensible defaults, and update triggers.

```
sql
PRAGMA foreign_keys = ON;

-- =====
-- Core entities
-- =====

CREATE TABLE colleges (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL UNIQUE,
  code TEXT UNIQUE,
  city TEXT,
  state TEXT,
  created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE students (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  college_id INTEGER NOT NULL,
  student_number TEXT NOT NULL,
  email TEXT NOT NULL,
  full_name TEXT NOT NULL,
  enrollment_year INTEGER CHECK (enrollment_year BETWEEN 1900 AND 2100),
  status TEXT NOT NULL DEFAULT 'active' CHECK (status IN ('active','inactive','graduated','suspended')),
  created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  UNIQUE (college_id, student_number),
  UNIQUE (email),
  FOREIGN KEY (college_id) REFERENCES colleges(id) ON DELETE CASCADE
);

CREATE TABLE events (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  college_id INTEGER NOT NULL,
  title TEXT NOT NULL,
  description TEXT,
  location TEXT,
  start_at DATETIME NOT NULL,
  end_at DATETIME NOT NULL,
  capacity INTEGER CHECK (capacity >= 0),
  is_public INTEGER NOT NULL DEFAULT 1 CHECK (is_public IN (0,1)),
  created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  CHECK (end_at > start_at)
);

-- =====
-- Relationships
-- =====

CREATE TABLE event_registrations (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  event_id INTEGER NOT NULL,
  student_id INTEGER NOT NULL,
  status TEXT NOT NULL DEFAULT 'registered' CHECK (status IN ('registered','waitlisted','cancelled')),
  waitlist_position INTEGER,
  registered_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  UNIQUE (event_id, student_id),
  FOREIGN KEY (event_id) REFERENCES events(id) ON DELETE CASCADE,
  FOREIGN KEY (student_id) REFERENCES students(id) ON DELETE CASCADE
);

CREATE TABLE attendance (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  event_id INTEGER NOT NULL,
  student_id INTEGER NOT NULL,
  check_in_at DATETIME NOT NULL,
  check_out_at DATETIME,
  method TEXT NOT NULL DEFAULT 'manual' CHECK (method IN ('manual','qr','nfc')),
  created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  UNIQUE (event_id, student_id),
  CHECK (check_out_at IS NULL OR check_in_at IS NOT NULL),
  FOREIGN KEY (event_id) REFERENCES events(id) ON DELETE CASCADE,
  FOREIGN KEY (student_id) REFERENCES students(id) ON DELETE CASCADE
);

CREATE TABLE feedback (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  event_id INTEGER NOT NULL,
  student_id INTEGER NOT NULL,
  rating INTEGER NOT NULL CHECK (rating BETWEEN 1 AND 5),
  comments TEXT,
  submitted_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP,
  UNIQUE (event_id, student_id),
  FOREIGN KEY (event_id) REFERENCES events(id) ON DELETE CASCADE,
  FOREIGN KEY (student_id) REFERENCES students(id) ON DELETE CASCADE
);
```

Campus event homepage

View Project

ChatDesign

Create a homepage for a campus event management platform. It should have a clean, modern design with a hero section that includes a call-to-action button to 'Browse Events.' Below the hero, display a grid of featured events with cards that show the event title, date, a small image, and a 'View Details' button

2:02 PM

Thought for 5s

I'll create a modern homepage for your campus event management platform. Let me start by generating some design inspiration to ensure we create something visually compelling.

Generated design inspiration

Explored codebase

> Created homepage v1

I've created a modern, clean homepage for your campus event management platform following the design brief. The page features a compelling hero section with a gradient background and clear call-to-action, followed by a responsive grid of featured event cards. Each card displays the event title, date, location, attendee count, and category with hover effects for enhanced interactivity. The design uses semantic color tokens from the updated globals.css file to ensure consistent theming and maintainability.

Design an events listing page with a filterable and searchable list. The page should have a sidebar on the left with filters for event type (e.g., Workshop, Fest) and college. The main content area should display a responsive grid of event cards

Thought for 3s

I'll create an events listing page with filtering and search capabilities. Let me first understand the current codebase structure and then build the new page.

Read homepage

> Created events listing page v2

I've created a comprehensive events listing page with a clean sidebar filter system and responsive event grid. The page includes real-time search functionality, checkbox filters for event types and colleges, and displays filtered results with event cards showing all the requested information. The sidebar is sticky on desktop and the layout adapts beautifully to mobile devices with proper spacing and hover effects.

