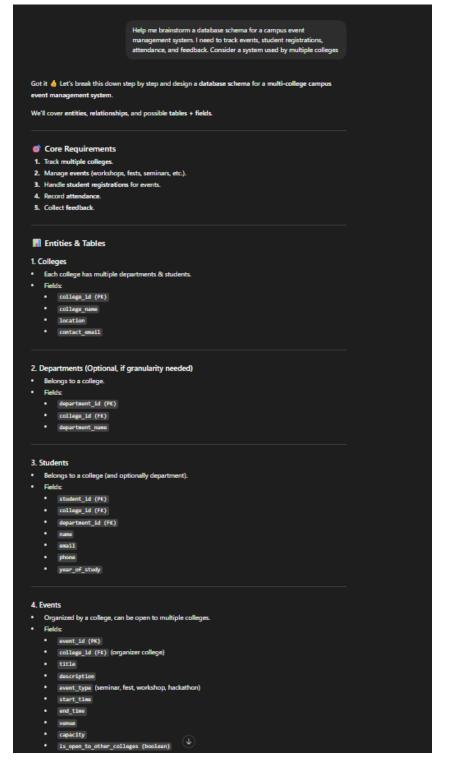
# **Prompts For Initial Design and Brainstorming**



Identify and list all potential edge cases and assumptions for a multicollege 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.

```
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
UNIQUE(reg_id)
```

Prevents duplicate attendance entries for the same student.

# For Prototype Implementation

```
REATE TABLE event_registrations (
 id INTEGER PRIMARY KEY AUTOINCREMENT.
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
REATE 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,
created_at DATETIME NOT NULL DEFAULT CORRENT_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
 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,
 FOREIGN KEY (event_id) REFERENCES events(id) ON DELETE CASCADE,
FOREIGN KEY (student_id) REFERENCES students(id) ON DELETE CASCADE,
```

