Hospital Case Management System (SQLAlchemy ORM)

Overview:

You are required to build a small but realistic Hospital Case Management System using Python and SQLAlchemy ORM. The system will support multiple health centers, each having their own clinicians who manage patient cases. The focus is on correct schema design, ORM implementation, clean code structure, and properly written database service functions.

Before starting any coding, draw the complete database schema on paper (tables, fields, and relationships) and get it reviewed.

Requirements

1. Database Schema & ORM Models

Design and implement the following entities using SQLAlchemy ORM:

HealthCenter

- id
- name
- address
- is_active (boolean)
- created_at

Clinician

- id
- health_center_id (FK → HealthCenter.id)
- name
- email
- role (e.g., doctor, nurse, admin)
- is_active

PatientCase

- id
- health_center_id (FK → HealthCenter.id)
- clinician_id (FK → Clinician.id)
- patient_name
- patient_dob
- summary
- status (open / closed)
- created_at
- updated_at

CaseNote

- id
- case_id (FK → PatientCase.id)
- clinician_id (FK → Clinician.id)

- note_text
- created_at

Relationships

- One HealthCenter has many Clinicians
- One HealthCenter has many PatientCases
- One PatientCase has many CaseNotes
- One Clinician can create many CaseNotes

ORM models must include proper relationships, foreign keys, and basic constraints.

---

2. Service Layer (Database Operations)

Create a `services/` directory containing database service modules. Each major entity should have functions for:

- Create
- Fetch by ID
- List (with optional filtering by health center or status)
- Update (partial updates allowed)
- Delete (hard delete or soft delete depending on design)

The service layer must be clean, modular, and use SQLAlchemy sessions properly.

---

3. Transaction Requirement

Implement at least one operation that must run inside a database transaction. Example scenario:

Create a new HealthCenter along with:

- Its lead clinician
- An initial sample patient case assigned to that clinician

All inserts must succeed or none should be committed.
You may choose your own transactional scenario if you prefer, as long as it demonstrates proper transactional handling.

---

4. Repository Deliverables

All project work must be delivered through a **GitHub repository** containing:

- Complete SQLAlchemy models (`models/` directory)
- Service functions (`services/` directory)
- A simple demo script (`demo.py`) that demonstrates creation of:
  - a health center
  - a clinician
  - a patient case
  - case notes
  - and an update + delete operation

- Simple **README** with:
    - project description
    - setup steps
    - example usage or code snippets showing how to call service functions
- The photo or scan of your initial pen-paper schema design included in the repository (e.g., in a `/design/` folder)

---

5. Example Project Structure (Only for reference):

```
hospital_case_mgmt/
│  README.md
│  demo.py
├── models/
│     base.py
│     health_center.py
│     clinician.py
│     patient_case.py
│     case_note.py
├── services/
│     health_center_service.py
│     clinician_service.py
│     patient_case_service.py
│     case_note_service.py
└── design/
    schema.jpg (pen-paper schema photo)
```

---

6. Completion Criteria

The project is considered complete when:

- The schema is reviewed and approved
- Models are correctly implemented with proper relationships
- CRUD operations work as expected
- The transactional workflow is implemented correctly
- Demo script runs without errors
- The GitHub repository contains all required files and documentation

---

This assignment is intended to simulate a **real-world backend** development task involving schema design, ORM modeling, and structured service-based database operations.

Terms and Conditions

1. **GitHub Usage**
    - The entire project must be maintained in a **proper GitHub repository**.
    - Commit messages must be clear, meaningful, and specific. Each commit should accurately describe what part of the project was implemented or modified.
    - Avoid committing unnecessary files. Use a proper `.gitignore` (Python, virtual environments, temporary files, editor configs, etc.).

2. **No AI-Generated Code Allowed**
   - You must write all the code on your own. Use of AI-generated code or assistance tools (ChatGPT, GitHub Copilot, etc.) is **strictly not allowed**.
   - If you have any doubts, you may:
     - Ask mentors directly
     - Use Google
     - Refer to official documentation and trusted technical sites
   - Any **AI-generated code** found in the project will result in rejection of the submission.

3. **Code Quality & Structure**
   - Code must be clean, readable, and modular.
   - Maintain consistent naming conventions across models, services, and files.

4. **Original Work**
   - The project must be implemented from **scratch**.
   - Re-using any previously built codebases, templates, or external projects is not permitted unless openly discussed and approved.

5. **Responsibility of Understanding**
   - You must be able to explain every line of your code during the review session.

These conditions ensure you learn the full process end-to-end and build industry-level habits in code structuring, documentation, and repository management.