Hospital Management System
Database Design Specification

Your Team Name

July 21, 2025

Contents

1 Introduction

This document outlines the core database structure and requirements for the Hospital Management System (HMS). It provides design guidelines so developers can implement a robust, normalized relational database that supports all major workflows, user/staff management, role-based access, patient care, appointments, bed and ambulance management, billing, and notifications.

Special emphasis is given to flexible and unified storage of all user and staff details.

2 Database Schema Overview

2.1 General Principles

1. **Relational model:** All major entities in their own tables, using foreign keys for relationships.
2. **Staff & Roles:** All staff (doctors, nurses, receptionists, managers, ambulance, admins) are stored in a single `users` table. Additional user profile fields are in a unified `user_details` table, linked by `user_id`.
3. **Patients:** All patient info in one table.
4. **References:** Use foreign keys (e.g., `patient_id`, `doctor_id`, `ward_id`) for links.
5. **Status Fields:** Use ENUM or status columns for workflow and availability.

users

Field	Type	Description
id	INT (PK)	Unique User ID
name	VARCHAR	Staff full name
email	VARCHAR (unique)	Login email
password_hash	VARCHAR	Encrypted password
role	ENUM	User's main role in system. Options: `doctor, nurse, receptionist, admin, ward_manager, ambulance, user_admin`
active	BOOL	Whether user is active (for disabling users without deleting)

Table 1: Core user table for all staff and system users. Every staff member, regardless of role, gets a row here.

Field	Type	Description
user_id	INT (FK)	Links to users.id; one-to-one with each user.
job_title	VARCHAR	e.g. Consultant, Nurse, Receptionist; shown on dashboards, profile, etc.
department	VARCHAR	e.g. Pediatrics, ICU, Admin; used for filtering and access.
qualification	VARCHAR	e.g. MBBS, RN; degrees or certificates.
phone_number	VARCHAR	Staff contact info (internal use)
address	VARCHAR	Staff address (optional)
joining_date	DATE	When the user started working
emergency_contact	VARCHAR	For contacting in emergencies
license_number	VARCHAR	Professional license (doctors, optional for others)

Table 2: Unified details table for all staff types; fields are nullable if not applicable. Only fill the relevant fields for each user type.

Field	Type	Description
id	INT (PK)	Unique patient ID (auto-increment)
name	VARCHAR	Full name
dob	DATE	Date of birth
gender	VARCHAR	Gender
phone	VARCHAR	Primary contact number
address	VARCHAR	Patient address
reg_date	DATE	Date patient first registered

Table 3: Table to store patient personal and contact information.

Field	Type	Description
id	INT (PK)	Appointment ID
patient_id	INT (FK)	Linked patient (patients.id)
doctor_id	INT (FK)	Assigned doctor (users.id)
appointment_time	DATETIME	Scheduled time for appointment
status	ENUM	booked, completed, cancelled
notes	TEXT	Doctor's notes or summary

Table 4: Outpatient appointments and doctor consultations.

user_details

patients

appointments

admissions

Other Key Tables

1. **wards**: id, name, type (e.g., ICU, General), manager_id

Field	Type	Description
id	INT (PK)	Admission ID
patient_id	INT (FK)	Admitted patient
ward_id	INT (FK)	Ward assigned
bed_id	INT (FK)	Bed assigned
admit_time	DATETIME	When patient was admitted
discharge_time	DATETIME	When patient was discharged (nullable)
status	ENUM	admitted, discharged, transferred, etc.

Table 5: Tracks inpatient admissions and bed assignments.

2. **beds**: id, ward_id, bed_number, status (available, occupied, cleaning), patient_id
3. **ambulances**: id, number, status (available, busy, maintenance), driver, last_known_location
4. **ambulance_requests**: id, patient_id, request_time, status, pickup, dropoff, ambulance_id
5. **invoices**: id, patient_id, created_by (user_id), date, total, paid, status, details (JSON/text)
6. **inventory**: id, ward_id, item, quantity, updated_at
7. **notifications**: id, user_id, type, message, read, created_at

3 Role-Based Workflows

Receptionist

1. **Login:** Authenticates with system.
2. **Patient Registration:** Adds new patients or finds existing ones.
3. **Appointment Booking:** Schedules appointments for patients with doctors.
4. **Admission:** Requests bed/ward assignment for inpatients.
5. **Ambulance Requests:** Logs and tracks ambulance requirements.
6. **Billing:** Generates initial and discharge invoices.
7. **Notifications:** Receives alerts about admissions, requests, or completed actions.

Doctor

1. **Login:** Authenticates and views personalized dashboard.
2. **Consultation:** Views daily appointments and patient histories.
3. **Patient Notes:** Records diagnoses, prescriptions, and follow-up instructions.
4. **Admission Requests:** Requests admissions or transfers for patients when needed.

5. **Billing:** Reviews or adds billable services to patient invoices.
6. **Notifications:** Receives alerts for new appointments or urgent cases.

Nurse

1. **Login:** Authenticates with system.
2. **Care Task Management:** Sees list of admitted patients in assigned wards.
3. **Task Updates:** Marks daily care activities (medication, vital signs, etc.) as completed.
4. **Bed Management:** Helps update bed/room status (e.g., occupied, needs cleaning).
5. **Notifications:** Receives updates about admissions, discharges, and patient transfers.

Ward Manager

1. **Login:** Authenticates and views assigned ward status.
2. **Bed Assignment:** Assigns available beds/rooms to admitted patients after requests from reception.
3. **Bed Status:** Marks beds as 'needs cleaning', 'available', or 'occupied'.
4. **Ward Inventory:** Tracks essential supplies for the ward.
5. **Reporting:** Generates daily/weekly occupancy and bed utilization reports.
6. **Notifications:** Receives admission/discharge requests and supply alerts.

Ambulance Staff

1. **Login:** Authenticates with system.
2. **Dispatch View:** Sees today's ambulance assignments and their status.
3. **Update Status:** Marks when an ambulance is en route, arrived, completed, or under maintenance.
4. **Reporting:** Logs issues or maintenance as needed.
5. **Notifications:** Receives new dispatches and urgent requests.

User Admin

1. **Login:** Authenticates with system.
2. **User Management:** Creates, edits, deactivates, or deletes user accounts for all roles.

3. **Role Assignment:** Assigns or changes roles as required.
4. **Profile Management:** Updates user details (e.g., department, job title).
5. **Audit:** Reviews user logs and system access for compliance.

4 Unified User/Staff Information Storage

1. The `users` table contains authentication and role for every staff member.
2. The `user_details` table stores all "profile" info (job title, department, qualification, license, etc.) for any user, not just doctors.
3. All profile fields in `user_details` are nullable.
4. This avoids creating "doctor_details", "nurse_details", etc.—all staff details are in one place.
5. On login, fetch user's info by joining `users` and `user_details` on `user_id`.
6. Any new roles or profile fields can be added easily by extending `user_details`.
7. Developers should provide profile pages so any logged-in user can see their job title, department, and other attributes from `user_details`.

Example:
A doctor's profile will have job title, department, license, qualification, phone, and joining date.
A receptionist will only have job title, phone, and joining date filled.

5 Relationships and Key Workflows

1. **Appointments:** Each links a patient and a doctor. Use `doctor_id` (users.id where role=doctor).
2. **Admissions:** Links patient, ward, bed, and admitting/discharging staff (via user_id).
3. **Bed Assignment:** Each bed row knows its ward and (if occupied) patient.
4. **Ambulance Request:** Links to patient, requesting staff, ambulance, and status.
5. **Invoices:** Linked to patients and created_by (user_id).
6. **Notifications:** Each has user_id (who should be alerted).

6 Implementation Guidelines for Developers

1. Do not make role-specific tables for staff; use unified `users` and `user_details`.
2. Make all fields in `user_details` nullable and fill only relevant ones for each role.
3. Use foreign keys (`user_id`, `patient_id`, etc.) for all relationships.
4. Always use JOIN queries for staff profile info.
5. All notifications, logs, and references use `user_id`.
6. Use indexes on columns frequently filtered/searched (e.g., role, department, ward_id).
7. Use ON DELETE CASCADE or SET NULL for foreign key constraints where it makes sense.

7 Summary Table: Staff User Storage

Table	**What it stores**	**Why**
users	Core login info, role, status	Every staff member
user_details	Job title, department, phone, etc.	Flexible details for any staff role
patients	Patient info	Patient registration
appointments	Consultations	Doctor-patient meetings
admissions	Inpatient stays	Bed assignment/tracking
wards, beds	Hospital structure	Resource assignment
ambulances, ambulance_requests	Transport data	Emergency dispatch
invoices	Bills/payments	Financial tracking
inventory	Supplies	Resource management
notifications	Alerts	System events

Table 6: Quick summary: what each main table is for.

8 Key Queries (in Plain Language)

1. Show logged-in user's profile (doctor, nurse, etc.): select from `users`, join with `user_details`.
2. Assign ward manager: set `wards.manager_id` = `users.id` (role=ward_manager).
3. List all nurses in Pediatrics: select from `users` join `user_details` where role=nurse and department='Pediatrics'.
4. List available beds in a ward: select from `beds` where status='available' and ward_id matches.

5. All appointments for a doctor: select from `appointments` where doctor_id matches and status='booked'.

6. All ambulance requests for today: select from `ambulance_requests` where date matches.

9 Entity-Relationship Diagram (ERD)

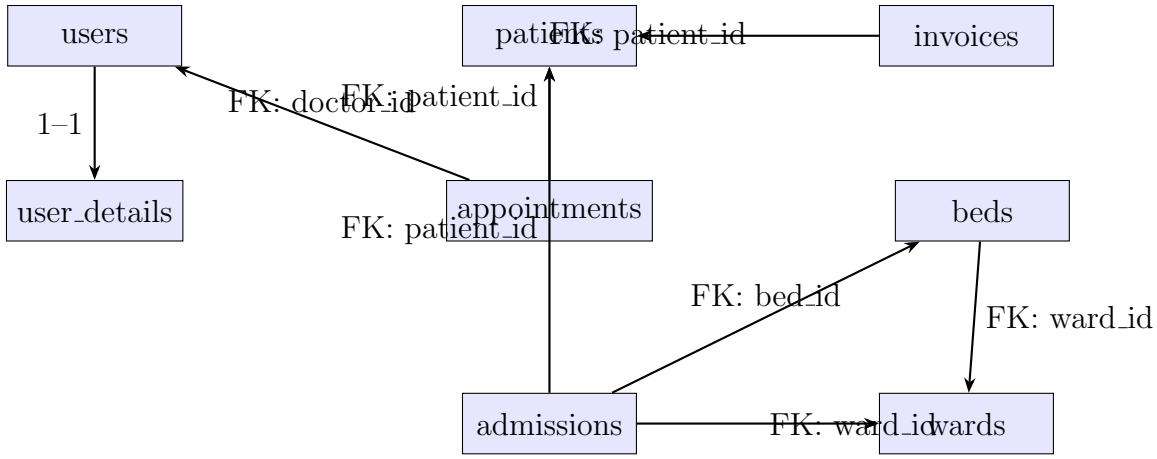

Figure 1: Simplified ER diagram showing the relationships between main tables.

10 API Endpoints (Suggested)

The system should expose REST API endpoints for all main entities. Example endpoints:

1. **Users:** GET/POST/PUT/DELETE /api/users
2. **User Details:** GET/PUT /api/user-details/{user_id}
3. **Patients:** GET/POST/PUT/DELETE /api/patients
4. **Appointments:** GET/POST/PUT/DELETE /api/appointments
5. **Admissions:** GET/POST/PUT/DELETE /api/admissions
6. **Beds, Wards:** GET/POST/PUT/DELETE /api/beds, /api/wards
7. **Ambulances/Requests:** /api/ambulances, /api/ambulance-requests
8. **Invoices:** /api/invoices
9. **Inventory:** /api/inventory
10. **Notifications:** /api/notifications

Each endpoint should support listing, creation, update, and deletion (where appropriate). Authentication is required for all but the login endpoint.

11 User Stories & Example Scenarios

1. **Receptionist:** As a receptionist, I want to quickly register a new patient and book an appointment with a doctor, so the patient can get timely care.
2. **Doctor:** As a doctor, I want to see my appointments for the day and review each patient's history before their visit.
3. **Nurse:** As a nurse, I want to view the daily care tasks for my assigned patients and update their status after each task.
4. **Ward Manager:** As a ward manager, I want to assign available beds to admitted patients and monitor bed cleaning status.
5. **Ambulance Staff:** As ambulance staff, I want to see my dispatches for the day and update status as each request is completed.
6. **User Admin:** As a user admin, I want to add new staff, assign roles, and reset passwords when needed.

12 Field Validations and Constraints

1. **Emails:** Must be unique, valid email format.
2. **Phone numbers:** Must be 11 digits (Bangladesh) or country-specific rule.
3. **All foreign keys:** Enforce referential integrity; use `ON DELETE SET NULL` or `CASCADE` as appropriate.
4. **No nulls in required fields:** Use NOT NULL where possible.
5. **Password:** Always store hashed; never plain text.
6. **Status fields:** Use ENUM with allowed values only.
7. **Date fields:** Use DATE or DATETIME, never plain strings.

13 Security Considerations

1. Passwords must be securely hashed using a strong algorithm (e.g., bcrypt).
2. All user actions (logins, edits, critical updates) should be logged.
3. Use HTTPS for all API endpoints.
4. Role-based access: Enforce strict permissions for each user type; validate access server-side.
5. Prefer soft-delete (marking records as inactive/deleted) to preserve history.
6. Limit failed login attempts to prevent brute-force attacks.
7. Patient data is sensitive; never expose to unauthorized users.

14 Future-Proofing & Scalability Notes

1. The schema is extensible for future modules: labs, pharmacy, insurance, reporting, etc.
2. Add new roles by extending the `role` ENUM and, if needed, add fields to `user_details`.
3. All logs, relationships, and user data refer only to user IDs, making integration with other systems easy.
4. Use versioning for APIs so changes don't break current clients.

15 Glossary

Admission
The process of entering a patient into hospital care, assigning a bed/ward.

Ambulance Request
A service request for patient transport.

Bed Assignment
Allocating a hospital bed to a specific patient.

Discharge
The process of releasing a patient from inpatient care.

ENUM
A datatype representing a set of allowed values.

FK (Foreign Key)
A column that refers to the primary key of another table.

Invoice
The document/bill for medical services.

Role
The assigned job type for a user (doctor, nurse, etc.), which determines access and UI.

Soft-delete
Marking a record as deleted (inactive), not actually removing it from the database.

User Admin
A system user who can create, edit, or remove staff/user accounts.