After narrowing down, we updated the objective (see GitHub) and focused on:

Hospital Clinical Orders & Billing Database Design Document

1. Business Problems Being Addressed

Modern hospitals face increasing complexity in linking **clinical operations** (prescriptions, lab tests, procedures) with **financial workflows** (billing, insurance claims, and payments).
Many existing systems either:

- Lack integration between clinical and billing modules
- Cannot easily track the financial impact of each medical order
- Require manual reconciliation between clinical orders and insurance claims

This database aims to solve these issues by integrating **Clinical Orders** and **Billing/Insurance** processes within a unified relational structure.

It supports:

- Tracking all orders (medication, lab tests, procedures) per encounter
- Automatically generating billing charges based on orders
- Managing insurance policies and claims linked to patients
- Recording payments and monitoring outstanding balances
- Enabling reporting on order utilization, revenue, and claim performance

2. Entities, Key Attributes, and Descriptions

Entity	Key Attributes	Description / Role
Patient	`patient_id (PK), first_name, last_name, dob, gender, emergency_contact`	Stores demographic and contact information for each patient.
Encounter	`encounter_id (PK), patient_id (FK), encounter_date, encounter_type, location_id, chief_complaint`	Represents a single visit, admission, or treatment event for a patient.
Provider	`provider_id (PK), name, specialty, provider_type`	Holds details for clinicians who issue or fulfill clinical orders.
Encounter_Provider	`encounter_id (FK), provider_id (FK), role, is_billing_provider`	Resolves the many-to-many relationship between encounters and providers

Entity	Key Attributes	Description / Role
		(e.g., primary vs. consulting doctor).
Location	`location_id (PK), name, type, department`	Represents the physical or organizational unit where an encounter or order occurs.
Clinical_Order	`order_id (PK), encounter_id (FK), order_date, status, order_type`	Captures each set of medical orders made during an encounter (e.g., lab test, medication, imaging).
Order_Item	`order_item_id (PK), order_id (FK), dictionary_id (FK), quantity, scheduled_date`	Stores specific order details; links to standard procedure/test codes.
Dictionary	`dictionary_id (PK), code, description, category, unit_price`	Serves as a standardized catalog for all billable items—lab tests, drugs, or procedures.
Billing_Charge	`billing_id (PK), order_item_id (FK), amount, status, charge_date`	Represents the financial record created from each order item for billing purposes.
Insurance_Policy	`policy_id (PK), patient_id (FK), insurer_company, policy_number, coverage_percent, expiry_date, effective_date, policy_type`	Defines a patient's insurance coverage.
Claim	`claim_id (PK), policy_id (FK), submission_date, status, claim_amount`	Represents an insurance claim submitted for one or more charges.
Claim_Charge	`claim_charge_id (PK), claim_id (FK), billing_id (FK), approved_amount, payment_status`	Breaks down each claim into individual billed items.
Payment	`payment_id (PK), claim_id (FK), amount_paid, payment_date, payer_type`	Records actual payments received from insurer or patient.

3. Relationships Overview

Relationship	Type	Description
Patient → Encounter	1:M	A patient can have multiple encounters (visits).

Relationship	Type	Description
Encounter ↔ Provider (via Encounter_Provider)	M:N	Multiple providers can participate in a single encounter.
Encounter → Clinical_Order	1:M	Each encounter can generate multiple clinical orders.
Clinical_Order → Order_Item	1:M	A clinical order contains multiple order items.
Order_Item → Dictionary	M:1	Each order item corresponds to a standardized entry in the hospital catalog.
Order_Item → Billing_Charge	1:1	Each order item results in one billing charge.
Billing_Charge → Claim_Charge	1:M	Each billing charge can be part of one or more claim charges (in rare cases of reprocessing).
Claim_Charge → Claim	M:1	Each claim groups multiple billing charges.
Claim → Payment	1:M	A claim can receive multiple payments (partial, adjustment, etc.).
Patient → Insurance_Policy	1:M	A patient may hold multiple active or expired insurance policies.
Insurance_Policy → Claim	1:M	Each claim is submitted under a specific policy.

4. Key Design Decisions

1. **Use of "Dictionary" as a master catalog:**
   Instead of separate tables for lab tests, drugs, and procedures, a single standardized catalog simplifies reporting, pricing, and maintenance.
2. **Order_Item granularity:**
   Clinical orders (like "Blood Panel") may include multiple items ("CBC," "Glucose Test"). Storing them separately supports accurate billing and result tracking.
3. **Separate Billing and Claim layers:**
   Not all charges are insured. By distinguishing Billing_Charge (hospital-side record) from Claim (insurance submission), the system supports both **self-pay** and **insurance-covered** workflows.
4. **Insurance abstraction:**
   The `Insurance_Policy` and `Claim` entities allow tracking of coverage levels and rejection/reprocessing cycles, supporting real-world hospital billing.
5. **Payment flexibility:**
   Payments may come from different sources (insurer, patient, or government subsidy), which is reflected in the `payer_type` attribute.
6. **Normalization and scalability:**
   The design follows **3rd Normal Form (3NF)** to eliminate redundancy while allowing flexible extensions (e.g., adding "Pharmacy Fulfillment" or "Lab Result" tables later).