Github link: ***https://github.com/azmat-s/hospital-management-System***
You can also find the above doc by Github repo -> **logical_DB_design** & **logical_ERD.png**

Hospital Database Management System

– Logical ERD Documentation

Project Overview

This document presents the finalized Conceptual and Logical ERD for a comprehensive Hospital Database Management System. The system manages patient care, insurance billing, clinical operations, and financial transactions while maintaining data integrity and supporting complex healthcare workflows.

Mission Statement

To create a comprehensive Hospital Database Management System that streamlines patient care, optimizes resource allocation, ensures efficient healthcare delivery, and manages the complete billing cycle from service provision through insurance claims to final patient payment collection, while maintaining data security and regulatory compliance.

Mission Objectives

1. **Patient Management**: Track patient demographics, medical history, and contact information
2. **Insurance Management**: Manage insurance policies, companies, and coverage details
3. **Encounter Tracking**: Record patient visits, admissions, and healthcare encounters
4. **Clinical Order Management**: Track medical orders, procedures, and treatments
5. **Provider Management**: Manage healthcare professionals and their roles in patient care
6. **Billing Operations**: Generate and track billing charges for services rendered
7. **Claims Processing**: Submit and track insurance claims for reimbursement
8. **Payment Tracking**: Record both insurance and patient payments with complete audit trail
9. **Inventory Control**: Maintain dictionary of billable services and procedures
10. **Financial Reporting**: Support comprehensive financial analysis and reconciliation

Changes from Previous ERD Design

1. Insurance Entity Normalization

Change: Separated Insurance Company from Insurance Policy

Initial Design:

- Insurance Policy entity contained insurance_company as an attribute

Final Design:

- Created separate Insurance_Company entity
- Established relationship: Insurance_Company (1) → Insurance_Policy (N)

Rationale:

- **Reduces redundancy**: Company information (name, address, phone) no longer duplicated across multiple policies
- **Achieves 3NF**: Removes transitive dependency (policy → company_name → company_address)
- **Improves maintainability**: Company information updates in one location
- **Supports business logic**: Multiple policies can belong to same insurance company
- **Enables queries**: "Show all policies from Aetna" becomes straightforward

Normalization Form: Achieves Third Normal Form (3NF) by eliminating transitive dependencies

2. Enhanced Claim Entity with Comprehensive Attributes

Change: Expanded Claim entity with detailed financial tracking attributes

Initial Design:

- Minimal claim attributes (claim_id, submission_date, claim_amount)

Final Design:

- Added: `approved_amount`, `adjudication_date`, `status` (Submitted/Approved/Denied/Paid)

Purpose of Claim Entity:

- **Primary Function**: Represents a formal request submitted to an insurance company for reimbursement of medical services provided during a patient encounter
- **Financial Tracking**: Records the complete lifecycle from submission through adjudication to payment
- **Audit Trail**: Maintains history of claim status changes and insurance company decisions

Connection to Billing Charge:

- **Many-to-Many Relationship**: Claim (N) ↔ Claim_Charge (junction) ↔ Billing_Charge (N)
- **Why Junction Table**: One claim can include multiple billing charges (surgery + lab + medications), and one billing charge could theoretically appear on multiple claims (resubmissions, appeals)
- **Data Tracked**: Claim_Charge junction table tracks which specific charges are included in each claim submission

What Billing Charge Tracks:

- **Service-Level Charges**: Individual billable services (CT scan, blood test, consultation)
- **Financial Details**: Original charge amount, adjustments, patient responsibility

- **Status Tracking**: Pending/Billed/Paid/Denied status for each charge
- **Audit Information**: Charge date, CPT codes, ICD codes for billing compliance

Relationship Flow:

```
Encounter → Clinical_Order → Order_Item → Billing_Charge → Claim_Charge → Claim
```

3. Payment Subtype Implementation (Disjoint Specialization)

Change: Implemented Payment supertype with two disjoint subtypes as advised by professor

Design Structure:

```
Payment (Supertype)
    ↓ (disjoint, total)
    ├── Patient_Payment
    └── Claim_Payment
```

Discriminator: `payment_type` field in Payment table ('PATIENT' or 'CLAIM')

Rationale for Disjoint Constraint:

- **Mutually Exclusive**: Each payment transaction is EITHER from patient OR from insurance company, never both
- **Different Attributes**: Patient payments need receipt numbers and deposit tracking; Claim payments need EOB numbers and check numbers
- **Different Workflows**: Insurance payments follow claims adjudication; Patient payments are direct transactions
- **Clear Semantics**: No ambiguity about payment source

Why NOT Overlapping:

- A single payment record cannot represent both patient money AND insurance money simultaneously
- Payment methods differ (patient credit card vs. insurance EFT)
- Payment timing differs (patient immediate vs. insurance delayed)
- Documentation differs (receipt vs. EOB)

Business Rule: `payment_type` acts as discriminator ensuring completeness (every payment must be one type) and disjointness (cannot be both types)

4. Billing Charge Relationship Adjustments

Change: Established clear relationships between Billing_Charge, Patient, Encounter, and Patient_Payment

4.1 Encounter → Billing_Charge (1:N)

Added: Direct foreign key `encounter_id` in Billing_Charge table

Rationale:

- **Business Requirement**: Charges must be grouped by patient visit for billing statements
- **Query Performance**: Enables efficient "show all charges for this encounter" queries without complex joins
- **Patient View**: Patients need to see charges organized by hospital visit
- **Audit Trail**: Clear association between visit and financial charges

Path Without Direct Link:

```
Billing_Charge → Order_Item → Clinical_Order → Encounter (3 joins)
```

Path With Direct Link:

```
Billing_Charge → Encounter (1 join - much faster)
```

Denormalization Justification: While `encounter_id` can be derived through Order_Item, storing it directly in Billing_Charge improves query performance for the most common use case (viewing charges by encounter). This controlled denormalization is justified by business requirements and performance needs.

4.2 Billing_Charge → Patient_Payment (N:1)

Relationship: One payment can pay multiple charges

Design Decision: Not allow multiple payments from patient

Rationale:

- **Flexibility**: One patient payment CAN cover multiple charges
- **No Interest Complexity**: System tracks actual payments without payment plan contracts, interest rates, or scheduled installments
- **Real-World Scenario**: Patient pays $500 to cover 3 different charges

Business Rule: Each patient payment can apply to many billing charges. No installment payment, interest rates, or late fees are tracked.

5. Normalization and 3NF Compliance

5.1 Normalization Achievements

First Normal Form (1NF):

- All attributes contain atomic values (no multi-valued attributes)
- Each attribute contains single value per row
- All entries in a column are of same data type

Second Normal Form (2NF):

- In 1NF
- No partial dependencies (all non-key attributes fully dependent on entire primary key)
- Example: In Claim_Charge junction table, `billed_amount` depends on both `claim_id` AND `charge_id`, not just one

Third Normal Form (3NF):

- In 2NF
- No transitive dependencies (non-key attributes don't depend on other non-key attributes)
- Example: Separating Insurance_Company from Insurance_Policy eliminates transitive dependency (policy → company_name → company_address)

Complete Table With Data Types

1. Patient

Purpose: Store patient demographic and contact information

Attribute	Data Type	Constraints	Description
patient_id	INT	PK, AUTO_INCREMENT	Unique patient identifier
first_name	VARCHAR(50)	NOT NULL	Patient's first name
last_name	VARCHAR(50)	NOT NULL	Patient's last name
dob	DATE	NOT NULL	Date of birth
gender	VARCHAR(10)	NOT NULL	Male/Female/Other/Prefer not to say
emergency_contact_phone	VARCHAR(20)	NULL	Emergency contact phone number
phone	VARCHAR(20)	NULL	Patient phone number

Relationships:

- Patient (1) → Insurance_Policy (N)
- Patient (1) → Encounter (N)

2. Insurance_Company

Purpose: Store insurance company information (separated for normalization)

Attribute	Data Type	Constraints	Description
insurance_company_id	INT	PK, AUTO_INCREMENT	Unique insurance company identifier
name	VARCHAR(100)	NOT NULL, UNIQUE	Insurance company name (e.g., Aetna, Blue Cross)
company_country	VARCHAR(50)	NULL	Company headquarters address
company_state	VARCHAR(50)	NULL	Company headquarters address
company_city	VARCHAR(50)	NULL	Company headquarters address
company_zipcode	VARCHAR(15)	NULL	Company headquarters address
email	VARCHAR(100)	NULL	Company contact email

Relationships:

- Insurance_Company (1) → Insurance_Policy (N)

Normalization Note: Extracted from Insurance_Policy to eliminate redundancy and achieve 3NF

3. Insurance_Policy

Purpose: Store patient insurance policy details and coverage

Attribute	Data Type	Constraints	Description
policy_id	INT	PK, AUTO_INCREMENT	Unique policy identifier
patient_id	INT	FK, NOT NULL	References Patient(patient_id)
insurance_company_id	INT	FK, NOT NULL	References Insurance_Company(insurance_company_id)
policy_number	VARCHAR(50)	NOT NULL	Insurance policy number
effective_date	DATE	NOT NULL	Policy start date
expiration_date	DATE	NULL	Policy end date (NULL if ongoing)
policy_order	VARCHAR(20)	NOT NULL	Primary/Secondary/Tertiary insurance
coverage_rate	DECIMAL(5,2)	NULL	Coverage percentage (0.00-100.00)
deductible	DECIMAL(10,2)	NULL	Annual deductible amount
out_of_pocket_max	DECIMAL(10,2)	NULL	Maximum out-of-pocket amount per year

Relationships:

- Patient (1) → Insurance_Policy (N): One patient can have multiple policies
- Insurance_Company (1) → Insurance_Policy (N): One company issues many policies
- Insurance_Policy (1) → Claim (N): One policy can have multiple claims

4. Encounter

Purpose: Store patient visit and encounter information

Attribute	Data Type	Constraints	Description
encounter_id	INT	PK, AUTO_INCREMENT	Unique encounter identifier
patient_id	INT	FK, NOT NULL	References Patient(patient_id)
location_id	INT	FK, NOT NULL	References Location(location_id)
encounter_type	VARCHAR(50)	NOT NULL	Inpatient/Outpatient/Emergency/Observation
chief_complaint	TEXT	NULL	Patient's main complaint or reason for visit
date	DATETIME	NOT NULL	Encounter date and time

Relationships:

- Patient (1) → Encounter (N)

- Location (1) → Encounter (N)
- Encounter (1) → Clinical_Order (N)
- Encounter (1) → Billing_Charge (N): Direct link for efficient charge retrieval
- Encounter (N) → Provider (N): Many-to-many through Encounter_Provider junction

5. Provider

Purpose: Store healthcare provider information

Attribute	Data Type	Constraints	Description
provider_id	INT	PK, AUTO_INCREMENT	Unique provider identifier
first_name	VARCHAR(50)	NOT NULL	Provider's first name
last_name	VARCHAR(50)	NOT NULL	Provider's last name
specialty	VARCHAR(100)	NULL	Medical specialty (Cardiology, Pediatrics, etc.)
provider_type	VARCHAR(50)	NOT NULL	Physician/Nurse Practitioner/PA/Specialist
license_number	VARCHAR(50)	NULL	Medical license number

Relationships:

- Provider (N) → Encounter (N): Many-to-many through Encounter_Provider

6. Encounter_Provider (Junction Table)

Purpose: Link providers to encounters (many-to-many relationship)

Attribute	Data Type	Constraints	Description
encounter_provider_id	INT	PK, AUTO_INCREMENT	Unique identifier
encounter_id	INT	FK, NOT NULL	References Encounter(encounter_id)
provider_id	INT	FK, NOT NULL	References Provider(provider_id)
role	VARCHAR(50)	NULL	Attending/Consulting/Assisting/Referring physician
is_billing_provider	BOOLEAN	DEFAULT FALSE	Whether this provider bills for the encounter

Composite Unique Key: (encounter_id, provider_id, role)

Relationships:

- Encounter (1) → Encounter_Provider (N)
- Provider (1) → Encounter_Provider (N)

7. Location

Purpose: Store hospital location information (clinics, wards, emergency rooms)

Attribute	Data Type	Constraints	Description
location_id	INT	PK, AUTO_INCREMENT	Unique location identifier
location_type	VARCHAR(50)	NOT NULL	Clinic/Ward/Emergency/ICU/Operating Room
department	VARCHAR(100)	NULL	Department name (Surgery, Radiology, etc.)
Building_number	INT	NOT NULL	Building number

Relationships:

- Location (1) → Encounter (N)

8. Clinical_Order

Purpose: Store medical orders (lab tests, medications, procedures)

Attribute	Data Type	Constraints	Description
order_id	INT	PK, AUTO_INCREMENT	Unique order identifier
encounter_id	INT	FK, NOT NULL	References Encounter(encounter_id)
order_date	DATETIME	NOT NULL	When order was placed
order_type	VARCHAR(50)	NOT NULL	Lab/Radiology/Medication/Procedure/Consultation
status	VARCHAR(20)	NOT NULL	Ordered/In Progress/Completed/Cancelled

Relationships:

- Encounter (1) → Clinical_Order (N)
- Clinical_Order (1) → Order_Item (N)

9. Dictionary

Purpose: Reference table for standard medical items, procedures, and services

Attribute	Data Type	Constraints	Description
item_id	INT	PK, AUTO_INCREMENT	Unique item identifier
item_name	VARCHAR(200)	NOT NULL	Name of procedure/test/medication/service
category	VARCHAR(50)	NOT NULL	Lab/Radiology/Medication/Procedure/Supply
code	VARCHAR(50)	UNIQUE, NULL	CPT/ICD/NDC standardized code
unit_price	DECIMAL(10,2)	NULL	Standard price for billing

Relationships:

- Dictionary (1) → Order_Item (N)

Purpose: Centralized reference for all billable items, ensuring consistency and enabling price updates

10. Order_Item

Purpose: Individual items within a clinical order

Attribute	Data Type	Constraints	Description
order_item_id	INT	PK, AUTO_INCREMENT	Unique order item identifier
order_id	INT	FK, NOT NULL	References Clinical_Order(order_id)
item_id	INT	FK, NOT NULL	References Dictionary(item_id)
quantity	INT	NOT NULL	Quantity ordered (default 1)
scheduled_date	DATETIME	NULL	When item is scheduled to be performed

Relationships:

- Clinical_Order (1) → Order_Item (N)
- Dictionary (1) → Order_Item (N)
- Order_Item (1) → Billing_Charge (1): Each order item generates exactly one billing charge

11. Billing_Charge

Purpose: Store billing charges for services rendered

Attribute	Data Type	Constraints	Description
billing_charge_id	INT	PK, AUTO_INCREMENT	Unique charge identifier
order_item_id	INT	FK, NOT NULL	References Order_Item(order_item_id)
encounter_id	INT	FK, NOT NULL	References Encounter(encounter_id) - Direct link for performance
patient_id	INT	FK, NOT NULL	References Patient(patient_id) - Direct link for performance
patient_payment_id	INT	FK, NOT NULL	References Patient_Payment(patient_payment_id)
status	VARCHAR(20)	NOT NULL	Pending/Billed/Paid/Partial/Denied/Overdue

Relationships:

- Order_Item (1) → Billing_Charge (1)
- Encounter (1) → Billing_Charge (N)
- Billing_Charge (N) → Claim (N): Many-to-many through Claim_Charge junction
- Billing_Charge (N) → Patient_Payment (1): One payment can pay multiple charges

Denormalization Note: `encounter_id, patient_id` stored here for query performance despite being derivable from Order_Item

12. Claim

Purpose: Store insurance claim information and adjudication results

Attribute	Data Type	Constraints	Description
claim_id	INT	PK, AUTO_INCREMENT	Unique claim identifier
policy_id	INT	FK, NOT NULL	References Insurance_Policy(policy_id)
submission_date	DATE	NOT NULL	Date claim was submitted to insurance
adjudication_date	DATE	NULL	Date insurance processed the claim
claim_amount	DECIMAL(10,2)	NOT NULL	Total amount claimed from insurance
approved_amount	DECIMAL(10,2)	NULL	Amount insurance approved to pay
status	VARCHAR(50)	NOT NULL	Submitted/Under Review/Approved/Partially Approved/Denied/Appealed/Paid

Business Logic:

- Total claimed should equal sum of all associated billing charges
- `approved_amount` ≤ `claim_amount`
- Insurance pays: `approved_amount`

Relationships:

- Insurance_Policy (1) → Claim (N)
- Claim (N) → Billing_Charge (N): Many-to-many through Claim_Charge junction
- Claim (1) → Claim_Payment (N): One claim can receive multiple insurance payments

Note: Encounter relationship is indirect through Billing_Charge (business rule: each claim contains charges from single encounter)

13. Claim_Charge (Junction Table)

Purpose: Link multiple billing charges to a claim (many-to-many relationship)

Attribute	Data Type	Constraints	Description
claim_charge_id	INT	PK, AUTO_INCREMENT	Unique identifier
claim_id	INT	FK, NOT NULL	References Claim(claim_id)
billing_charge_id	INT	FK, NOT NULL	References Billing_Charge(charge_id)
billed_amount	DECIMAL(10,2)	NOT NULL	Amount billed for this specific charge on this claim

Composite Unique Key: (claim_id, billing_charge_id) - prevents duplicate entries

Relationships:

- Claim (1) → Claim_Charge (N)
- Billing_Charge (1) → Claim_Charge (N)

Purpose: Resolves many-to-many relationship (one claim includes multiple charges; one charge could appear on multiple claims if resubmitted)

14. Payment (Supertype)

Purpose: Store all payment information (parent table for payment hierarchy)

Attribute	Data Type	Constraints	Description
payment_id	INT	PK, AUTO_INCREMENT	Unique payment identifier
payment_date	DATE	NOT NULL	Date payment was received
amount_paid	DECIMAL(10,2)	NOT NULL	Total amount of this payment
payment_method	VARCHAR(50)	NOT NULL	Cash/Check/Credit Card/Debit/Wire Transfer/EFT
payment_status	VARCHAR(20)	NOT NULL	Pending/Completed/Failed/Refunded/Cancelled
payment_type	VARCHAR(20)	NOT NULL	**'PATIENT' or 'CLAIM'** (Discriminator for subtypes)

CHECK Constraint: `payment_type IN ('PATIENT', 'CLAIM')`

Subtype Completeness: Total (every payment must be either Patient or Claim payment) **Subtype Disjointness**: Disjoint (payment cannot be both Patient and Claim simultaneously)

Relationships:

- Payment (1) → Patient_Payment (1): Supertype to subtype (when payment_type = 'PATIENT')
- Payment (1) → Claim_Payment (1): Supertype to subtype (when payment_type = 'CLAIM')

15. Patient_Payment (Subtype)

Purpose: Store patient out-of-pocket payments (subtype of Payment)

Attribute	Data Type	Constraints	Description
patient_payment_id	INT	PK, AUTO_INCREMENT	Unique identifier
payment_id	INT	FK, UNIQUE, NOT NULL	References Payment(payment_id)
patient_id	INT	FK, NOT NULL	References Patient(patient_id)
encounter_id	INT	FK, NULL	References Encounter(encounter_id)

Business Rule: `payment_id` must reference a Payment record where `payment_type = 'PATIENT'`

Relationships:

- Payment (1) → Patient_Payment (1): Inherits from Payment supertype
- Patient (1) → Patient_Payment (N): Optional - tracks who are payment relates to
- Billing_Charge (1) → Patient_Payment (N): One payment can pay multiple charges

16. Claim_Payment (Subtype)

Purpose: Store insurance company payments (subtype of Payment)

Attribute	Data Type	Constraints	Description
claim_payment_id	INT	PK, AUTO_INCREMENT	Unique identifier
payment_id	INT	FK, UNIQUE, NOT NULL	References Payment(payment_id)
claim_id	INT	FK, NOT NULL	References Claim(claim_id)

Business Rule: `payment_id` must reference a Payment record where `payment_type = 'CLAIM'`

Relationships:

- Payment (1) → Claim_Payment (1): Inherits from Payment supertype
- Claim (1) → Claim_Payment (N): One claim can receive multiple insurance payments

Note: Multiple Claim_Payment records for one Claim handles scenarios like partial payments, payment adjustments, or secondary insurance payments

Relationship Summary

Complete Relationship Matrix

From Entity	To Entity	Cardinality	Relationship Type	Notes
Patient	Insurance_Policy	1:N	One-to-Many	One patient can have multiple insurance policies
Patient	Encounter	1:N	One-to-Many	One patient has many encounters over time
Insurance_Company	Insurance_Policy	1:N	One-to-Many	One company issues many policies
Insurance_Policy	Claim	1:N	One-to-Many	One policy can have multiple claims
Encounter	Clinical_Order	1:N	One-to-Many	One encounter generates multiple orders
Encounter	Billing_Charge	1:N	One-to-Many	One encounter has multiple charges (direct link)
Encounter	Encounter_Provider	1:N	One-to-Many	Junction table for many-to-many with Provider

From Entity	To Entity	Cardinality	Relationship Type	Notes
Provider	Encounter_Provider	1:N	One-to-Many	Junction table for many-to-many with Encounter
Location	Encounter	1:N	One-to-Many	One location hosts many encounters
Clinical_Order	Order_Item	1:N	One-to-Many	One order contains multiple items
Dictionary	Order_Item	1:N	One-to-Many	One dictionary item used in many orders
Order_Item	Billing_Charge	1:1	One-to-One	Each order item generates exactly one charge
Billing_Charge	Claim_Charge	1:N	One-to-Many	Junction table for many-to-many with Claim
Billing_Charge	Patient_Payment	1:N	One-to-Many	One charge can receive multiple patient payments
Claim	Claim_Charge	1:N	One-to-Many	Junction table for many-to-many with Billing_Charge
Claim	Claim_Payment	1:N	One-to-Many	One claim can receive multiple insurance payments
Payment	Patient_Payment	1:1	Supertype-Subtype	Disjoint specialization
Payment	Claim_Payment	1:1	Supertype-Subtype	Disjoint specialization
Patient	Patient_Payment	1:N	One-to-Many	One patient makes many payments
Insurance_Company	Claim_Payment	1:N	One-to-Many	One company makes many payments

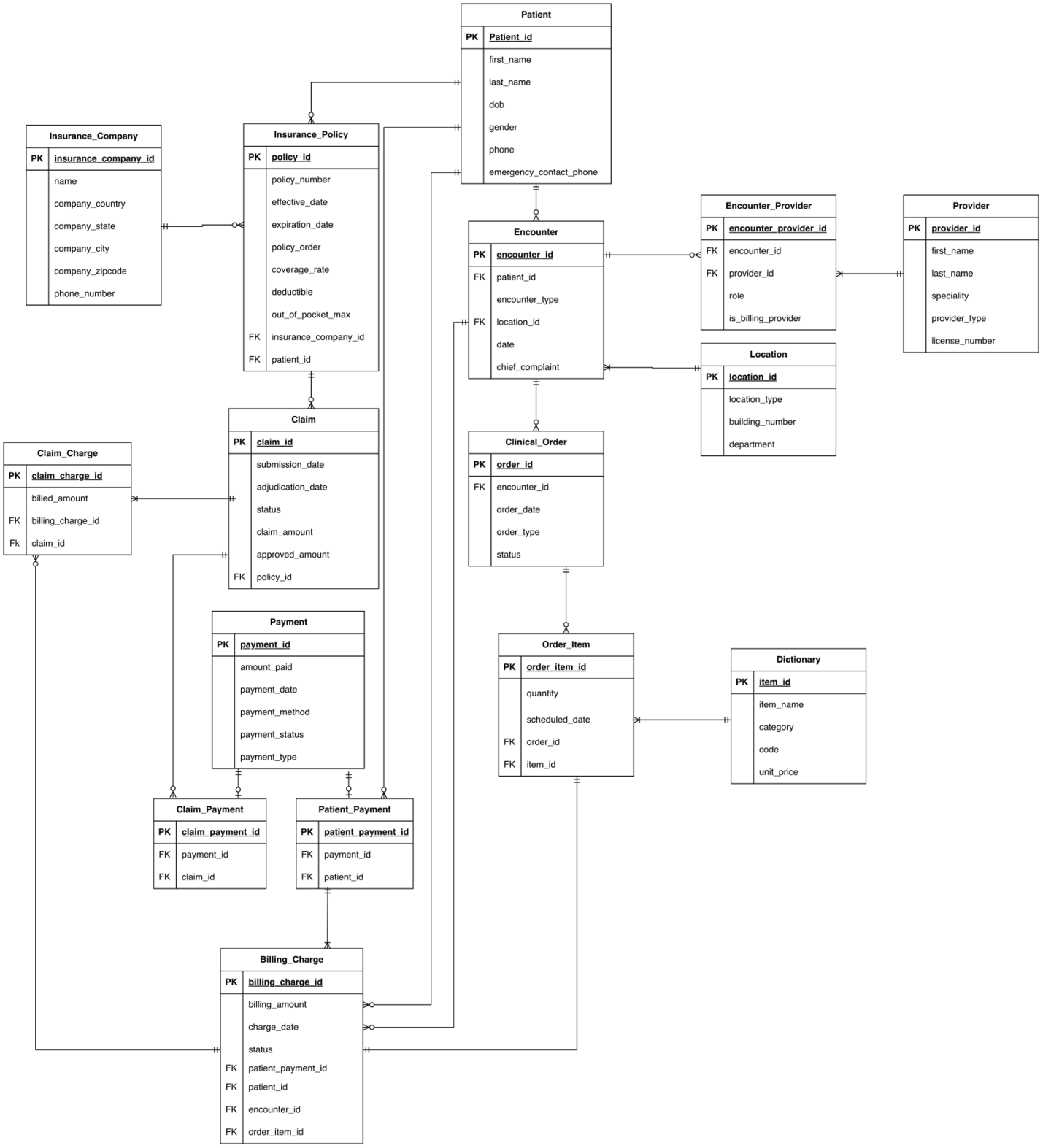
Business Rules and Constraints

Data Integrity Constraints

1. **Patient Information**
   - `patient_id` must be unique
   - `dob` must be in the past
   - `gender` must be valid value
2. **Insurance Policy**
   - `policy_number` must be unique per insurance company
   - `effective_date` must be before `expiration_date`
   - `coverage_rate` must be between 0 and 100
   - Only one policy can be marked as "Primary" per patient at a time
3. **Encounter**
   - `date` must not be in the future
   - `encounter_status` must follow valid state transitions
4. **Billing Charge**
   - `patient_paid` ≤ `patient_responsibility`
   - `charge_date` must be on or after encounter date
5. **Claim**
   - `approved_amount` ≤ `claim_amount`
   - `adjudication_date` must be after `submission_date`

    - Sum of Claim_Charge `billed_amount` should equal `claim_amount`
    - All charges in one claim must belong to same encounter (business rule)
6. **Payment Subtype Constraints**
    - If `payment_type` = 'PATIENT', must have corresponding Patient_Payment record
    - If `payment_type` = 'CLAIM', must have corresponding Claim_Payment record
    - Each Payment record must have exactly ONE subtype record
7. **Payment Application**
    - Patient_Payment `amount_paid` must not exceed remaining charge balance
    - Sum of all Patient_Payment amounts for a charge ≤ `patient_responsibility`
    - Claim_Payment `paid_amount` must not exceed claim `approved_amount`

Logical ERD Diagram



Conclusion

This finalized ERD represents a comprehensive, normalized hospital database management system that efficiently tracks the complete patient care and billing workflow from initial encounter through clinical services, insurance claims processing, and final payment collection. The design achieves Third Normal Form while incorporating strategic denormalization for performance where justified, implements proper subtype hierarchies for payment tracking, and maintains clear audit trails for financial transactions.

The system supports complex healthcare workflows including multiple insurance coverage, partial payments, claim resubmissions, and multi-provider encounters while maintaining data integrity and enabling efficient querying for both operational and financial reporting needs.