# Enterprise Data Architecture for Insurance Chronic Disease Forecasting

## Executive Summary

This report details the end-to-end solution for integrating structured insurance business data with unstructured health data to support chronic disease forecasting. The solution includes an optimized logical database schema, a on-premises data lake deployed on Ubuntu 22.04 cloud server, a comprehensive reference architecture, and robust data governance. The hybrid data model enables insurance companies to leverage both structured business data and unstructured health data for product innovation, claims forecasting, and personalized customer health management.

## 1. Project Background & Objectives

### 1.1 Project Context

Traditional insurance data management relies on relational databases for structured business data (accounts, customers, contracts, etc.). However, to address the healthcare "burning platform" and achieve the IHI Triple Aim objectives, insurance companies need to integrate unstructured health data (medical records, imaging, public health statistics) to forecast chronic diseases and drive data-informed decisions.

### 1.2 Core Objectives

1. Design a hybrid data model (structured + unstructured) that extends the existing EDA.
2. Implement a data lake on Ubuntu 22.04 cloud server to store and manage multi-source data.
3. Develop an end-to-end reference architecture spanning business, application, DIKW, and infrastructure domains.
4. Ensure data governance to prevent "data swamps" and maintain data quality/security.

## 2. EDA Logical Schema Optimization

### 2.1 Existing Schema Assessment

The original schema covers core insurance entities (Account, Customer, Contract, Associate) with valid primary/foreign key constraints and business rule checks. Key limitations for the current project:

- No structured storage for chronic disease influencing factors.
- Lack of linkage between business data and unstructured health data.
- No support for public health statistics integration (e.g., CDC data).

### 2.2 Optimized Logical Schema (PostgreSQL Compatible)

**2.2.1 Schema Extension: Chronic Disease Factor Tables**

```
-- Table 1: Social & Environmental Impact Factors
CREATE TABLE social_environmental_impact (
   impact_id SERIAL PRIMARY KEY,
   population_aging_ratio NUMERIC(5,2),  -- % of population over 65
   urbanization_rate NUMERIC(5,2),       -- % of urban population
   globalization_index NUMERIC(5,2),     -- Economic globalization score
   healthcare_access_index NUMERIC(5,2), -- Accessibility to healthcare facilities
   data_source VARCHAR(100) NOT NULL,    -- e.g., "CDC 2024 Report"
   load_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

-- Table 2: Chronic Disease Core Factors
CREATE TABLE chronic_disease_factor (
   factor_id SERIAL PRIMARY KEY,
   customer_id INTEGER REFERENCES customer(customer_id) ON DELETE NO ACTION
ON UPDATE NO ACTION,
   impact_id INTEGER REFERENCES social_environmental_impact(impact_id) ON DELETE
NO ACTION ON UPDATE NO ACTION,
   unchangeable_factors JSONB NOT NULL,  -- e.g., {"age": 55, "genetics": "diabetes_history",
"gender": "male"}
   changeable_factors JSONB NOT NULL,    -- e.g., {"diet": "high_sugar", "exercise":
"sedentary", "smoking": true}
   disease_type VARCHAR(50) NOT NULL,    -- e.g., "Type 2 Diabetes", "Cardiovascular
Disease"
   risk_score NUMERIC(4,2),              -- 0-100 risk rating
   factor_updated_at TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);

-- Table 3: Unstructured Health Data Metadata (links to data lake)
CREATE TABLE unstructured_health_data (
   health_data_id SERIAL PRIMARY KEY,
   account_id INTEGER REFERENCES account(account_id) ON DELETE NO ACTION ON
UPDATE NO ACTION,
   customer_id INTEGER REFERENCES customer(customer_id) ON DELETE NO ACTION
ON UPDATE NO ACTION,
   data_type VARCHAR(50) NOT NULL,       -- e.g., "medical_report", "xray_image",
"lab_results"
   file_format VARCHAR(20) NOT NULL,     -- e.g., "pdf", "dicom", "csv"
   storage_path VARCHAR(255) NOT NULL,   -- Path on Ubuntu server: /data-
lake/unstructured/...
    file_size_mb NUMERIC(8,2) NOT NULL,
    upload_timestamp TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    uploaded_by VARCHAR(50) NOT NULL,      -- e.g., "admin", "automated_pipeline"
    data_status VARCHAR(20) NOT NULL DEFAULT 'active' CHECK (data_status IN ('active',
'archived', 'invalid'))
);

-- Index Optimization for Query Performance
CREATE INDEX idx_cdf_customer ON chronic_disease_factor(customer_id);
CREATE INDEX idx_cdf_impact ON chronic_disease_factor(impact_id);
CREATE INDEX idx_uhd_customer ON unstructured_health_data(customer_id);
CREATE INDEX idx_uhd_account ON unstructured_health_data(account_id);
```

#### 2.2.2 Existing Schema Adjustments

- Updated account table: Added health_risk_tier VARCHAR(20) CHECK (health_risk_tier IN ('low', 'medium', 'high', 'unknown')) to link business accounts with health risk.
- Updated customer table: Added date_of_birth DATE and health_record_id VARCHAR(50) to enable demographic-based chronic disease analysis.

### 2.3 Optimization Rationale

1. Normalization Compliance: New tables follow 3NF to avoid data redundancy (e.g., separating social/environmental factors from individual risk factors).
2. Unstructured Data Linkage: unstructured_health_data table uses account_id/customer_id to connect unstructured files (stored in data lake) with core business entities.
3. Flexibility with JSONB: Used PostgreSQL JSONB type for unchangeable_factors and changeable_factors to handle dynamic, semi-structured attribute sets (e.g., varying lifestyle factors).
4. Performance Tuning: Indexes on foreign keys and frequently queried fields reduce join operation latency for analytics workloads.
5. Business Alignment: Added health_risk_tier to bridge health data with insurance product pricing and claims forecasting.

## 3. Data Lake Design & Implementation (Ubuntu 22.04 Cloud Server)

### 3.1 Data Lake Architecture

Deployed on Ubuntu 22.04 cloud server using **local file system + MinIO** (open-source object storage for unstructured data). The lake is organized by data type and access frequency:

```
/data-lake/
├── structured/              # Structured data (synced from PostgreSQL)
│   ├── insurance/           # Core insurance data (account, customer, contract)
│   │   ├── daily_sync/      # Daily incremental backups (CSV format)
│   │   └── full_backup/     # Weekly full backups (Parquet format)
│   └── health/              # Chronic disease structured data
│       ├── cdc_stats/       # CDC public chronic disease statistics
│       └── patient_factors/ # Exported from chronic_disease_factor table
└── unstructured/            # Unstructured data (managed by MinIO)
    ├── medical_reports/     # PDF/Word medical records
    ├── medical_images/      # DICOM/X-Ray images
    ├── lab_results/         # Unstructured lab data (CSV/Excel)
    └── public_health/       # CDC reports, research papers (PDF/XML)
```

### 3.2 Data Ingestion Pipeline

Implemented on Ubuntu 22.04 using open-source tools:

1. Structured Data:
   - Cron jobs (scheduled via crontab) to export PostgreSQL tables to Parquet/CSV (using pg_dump and pandas).
   - Incremental sync via rsync to avoid redundant data transfer.
2. Unstructured Data:
   - MinIO S3-compatible API for secure file uploads (via mc client or custom Python scripts).
   - Automated file validation using file command to check format integrity (e.g., valid PDF/DICOM).
3. Public Data:
   - Shell scripts to scrape/download CDC chronic disease datasets (scheduled weekly via Cron).

### 3.3 Metadata Management

- Created data_lake_metadata.csv (stored in /data-lake/metadata/) with columns: file_path, data_type, source, load_date, business_entity, access_permissions, expiry_date.

- Used mlflow to track data lineage for analytics pipelines (e.g., which files feed into chronic disease risk models).

### 3.4 Data Quality & Validation

- Shell scripts (deployed via Cron) to check:
    - File existence and completeness (e.g., daily CDC data uploads).
    - Data format consistency (e.g., CSV column counts match schema).
    - Integrity of structured-unstructured links (e.g., all health_data_id in PostgreSQL map to valid MinIO objects).
- Alerting: Failed validations trigger email notifications via sendmail (configured on Ubuntu server).

## 4. Domain Details

### 4.1 Business Domain

- Use Cases:
    a. Chronic disease risk-based insurance product design.
    b. Claims forecasting for chronic disease-related treatments.
    c. Personalized health interventions to reduce claim costs.
- Stakeholders: Product teams, claims departments, customer service, data science.

### 4.2 Application Domain

| Component | Tool (Ubuntu 22.04 Compatible) | Purpose |
|---|---|---|
| Data Integration | Apache NiFi + Cron | Ingest structured/unstructured/public data |
| Structured Storage | PostgreSQL 14 | Core business + health factor data |
| Unstructured Storage | MinIO | Medical images/reports (S3-compatible) |
| Data Processing | Python (Pandas, Dask) | ETL, data transformation for analytics |

| | | |
|---|---|---|
| Analytics & AI | Scikit-learn, XGBoost | Train chronic disease prediction models |
| Visualization | Flask + Matplotlib | Internal dashboards for risk analysis |

**4.3 DIKW Domain**

- Data: Raw structured/unstructured data in data lake + PostgreSQL (governed by metadata).
- Information: Aggregated insights (e.g., "35% of medium-risk customers have diabetes risk factors").
- Knowledge: Predictive models (e.g., "Age + sedentary lifestyle → 72% Type 2 Diabetes risk").
- Wisdom: Actionable decisions (e.g., "Offer wellness program to high-risk customers; adjust premium for medium-risk tiers").

**4.4 Infrastructure Domain**

- Server Configuration: Ubuntu 22.04 LTS (4 vCPUs, 16GB RAM, 500GB SSD).
- Network Security: UFW (Uncomplicated Firewall) to restrict access to data lake/PostgreSQL (only allow internal IPs).
- Storage Scalability: Mounted additional block storage (1TB) for data lake expansion.
- Backup Strategy: Daily incremental backups to external storage (via rsync over SSH).

**5. Data Governance Framework**

**5.1 Governance Principles**

1. Data Quality: Maintain ≥95% completeness for critical fields (e.g., customer_id, disease_type).
2. Security: Comply with HIPAA (for health data) and data privacy regulations.
3. Access Control: Role-based permissions for data lake and database access.
4. Lifecycle Management: Retain raw data for 5 years; archive inactive data after 2 years.

**5.2 Implementation on Ubuntu 22.04**

- Access Control:
  - Linux file permissions (chmod/chown) for data lake directories.
  - PostgreSQL roles (read-only for analysts, write-only for ingestion pipelines).

    - MinIO IAM policies to restrict unstructured data access.
- Data Security:
    - Encrypt sensitive files with gpg (medical records, patient data).
    - Enable PostgreSQL Transparent Data Encryption (TDE) for database storage.
    - SSH key-based authentication only (disabled password login for server).
- Lifecycle Management:
    - Bash scripts (scheduled via Cron) to move old data to /data-lake/archive/.
    - Automated deletion of expired data (per compliance requirements) using find command.
- Bias Mitigation:
    - Regular audits of chronic disease prediction models to ensure fairness across demographics.
    - Documented data sources to avoid biased training data (e.g., over-reliance on urban population data).

**6. Deployment & Operational Procedures**

**6.1 Ubuntu 22.04 Server Setup Steps**

1. Base Configuration:
    - Update system packages: sudo apt update && sudo apt upgrade -y.
    - Install required tools: sudo apt install postgresql-14 minio python3-pip rsync sendmail.
    - Configure UFW: sudo ufw allow ssh; sudo ufw allow 5432/tcp; sudo ufw allow 9000/tcp; sudo ufw enable.
2. PostgreSQL Setup:
    - Create database: sudo -u postgres createdb insurance_eda.
    - Execute optimized schema SQL script (Section 2.2.1).
    - Create user roles with appropriate permissions.
3. MinIO Setup:
    - Start MinIO server: minio server /data-lake/unstructured --console-address ":9001".

    - Create buckets (medical_reports, medical_images, etc.) via MinIO web console.

4. Data Pipeline Deployment:
    - Install Python dependencies: pip3 install pandas pyarrow psycopg2-binary minio mlflow.
    - Deploy ingestion scripts to /opt/data-pipelines/ and schedule via Cron.

**6.2 Operational Maintenance**

- Daily: Check pipeline logs (/var/log/data-pipelines/) for failures.
- Weekly: Run full data backup and validate data quality reports.
- Monthly: Review access logs and update data governance policies as needed.