**Case Study: Enforcing Data Security & Governance in a Healthcare Analytics Platform**

**Industry Context**

A healthcare analytics company, **MediInsights**, processes sensitive patient records and hospital operational data. With the implementation of **Databricks Unity Catalog**, the company is tasked with enforcing data governance policies that:

* Protect sensitive PII and PHI (Protected Health Information)
* Enable controlled data access based on user role and department
* Maintain visibility into data movement (lineage)
* Ensure all access and permission changes are logged and auditable

**Use Case Objectives**

1. **Apply Row-Level Security** to limit data based on user\_region
2. **Apply Column-Masking** for sensitive fields like ssn and diagnosis
3. **Tag PII & PHI columns** to enable discovery and compliance checks
4. **Visualize Lineage** to understand how data flows into reports and dashboards
5. **Verify Audit Logs** to ensure actions are traceable for regulatory compliance

**Data Model**

Table: healthcare\_catalog.records.patient\_visits

| **patient\_id** | **name** | **ssn** | **diagnosis** | **region** | **doctor\_id** |
| --- | --- | --- | --- | --- | --- |
| 001 | John Doe | 999-55-1234 | Diabetes | East | DR\_001 |
| 002 | Jane Roe | 888-33-4567 | Hypertension | West | DR\_002 |

**Step 1: Apply Row-Level Security**

**Scenario:**

Only regional analysts should access data from their assigned region.

**Implementation:**

sql

CREATE OR REPLACE FUNCTION current\_user\_region() RETURNS STRING

RETURN get\_user\_attribute('region'); -- Assume attribute mapping is managed

CREATE OR REPLACE ROW FILTER region\_filter AS (

region = current\_user\_region()

);

ALTER TABLE healthcare\_catalog.records.patient\_visits

SET ROW FILTER region\_filter;

**Step 2: Apply Column-Level Masking**

**Scenario:**

Mask ssn and diagnosis for all users except doctors and admins.

**Implementation:**

sql

CREATE FUNCTION mask\_sensitive(col STRING) RETURNS STRING

RETURN IF(current\_user() IN ('admin@mediinsights.com', 'doctor@mediinsights.com'), col, '\*\*\*MASKED\*\*\*');

CREATE OR REPLACE VIEW patient\_visits\_masked AS

SELECT

patient\_id,

name,

mask\_sensitive(ssn) AS ssn,

mask\_sensitive(diagnosis) AS diagnosis,

region,

doctor\_id

FROM healthcare\_catalog.records.patient\_visits;

**Step 3: Tag PII & PHI Fields**

**Scenario:**

Tag ssn and diagnosis for GDPR/PHI compliance audits.

**Implementation:**

sql

ALTER TABLE healthcare\_catalog.records.patient\_visits

ALTER COLUMN ssn SET TAGS ('pii': 'true', 'gdpr': 'yes', 'phi': 'yes');

ALTER COLUMN diagnosis SET TAGS ('phi': 'yes');

Tags now appear in the schema explorer and can be searched via catalog APIs.

**Step 4: Visualize Lineage**

**Scenario:**

Data Engineers and Compliance Officers need full data traceability.

**Steps:**

1. Open **Unity Catalog > Data Lineage Explorer**
2. Search patient\_visits
3. View:
   * Upstream: ETL jobs or ingestion pipelines
   * Downstream: Masked views, dashboards, external consumers
4. Enable **column-level lineage** to confirm propagation of ssn masking.

**Step 5: Verify Audit Events**

**Scenario:**

Verify who accessed what data and whether masking was bypassed.

**Setup:**

Enable Audit Logs → Route to Azure Log Analytics or AWS S3.

**Query in Log Analytics:**

kql

DatabricksAuditLogs

| where ObjectName contains "patient\_visits"

| summarize AccessCount = count() by UserEmail, ActionName

You can create alerts for:

* Unauthorized SELECT attempts on raw table
* Privilege escalations (e.g., GRANT SELECT)
* Masking policy modifications

**Results**

| **Goal** | **Outcome** |
| --- | --- |
| Row-level security | Users restricted to their own region |
| Column-level masking | SSN and Diagnosis visible only to doctors |
| Data tagging | PII/PHI columns discoverable and classified |
| Lineage tracking | Complete upstream and downstream traceability |
| Audit logging | Every access and policy change is traceable |

**Key Takeaways**

* Unity Catalog enforces *multi-layered* data security via declarative policies.
* Tags help drive *automated discovery*, classification, and compliance.
* Lineage and audit logs offer deep *observability* for regulatory needs.