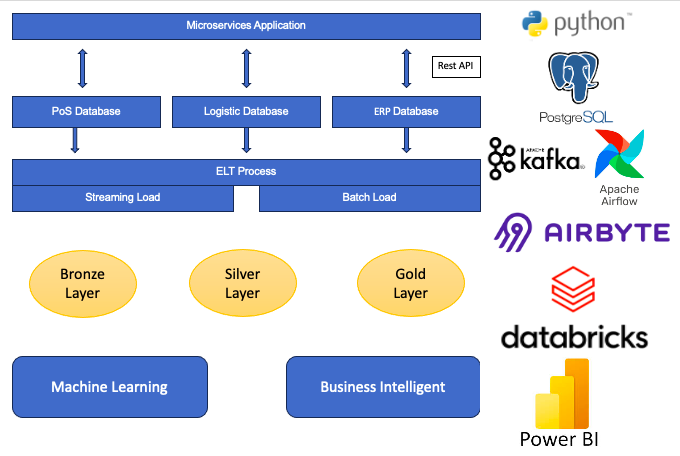
**Mô hình dự án:**



**Tổng quan dự án:**

Công ty giả định hoạt động trong lĩnh vực **thương mại điện tử** với ba mảng hệ thống chính:

1. **PoS (Point of Sale – Bán hàng):**
   * Quản lý giao dịch tại cửa hàng và kênh online.
   * Dữ liệu phát sinh liên tục, khối lượng lớn (hàng triệu order/tháng).
2. **Logistics (Kho vận & vận chuyển):**
   * Theo dõi tình trạng tồn kho, điều phối vận chuyển, giao hàng.
   * Yêu cầu tích hợp dữ liệu từ nhiều chi nhánh/kho khác nhau.
3. **ERP (Enterprise Resource Planning – Quản lý vận hành):**
   * Bao gồm quản lý nhân sự, tài chính, mua hàng, CMS nội bộ.
   * Nhiều dữ liệu nhạy cảm, yêu cầu phân quyền và kiểm soát chặt chẽ.

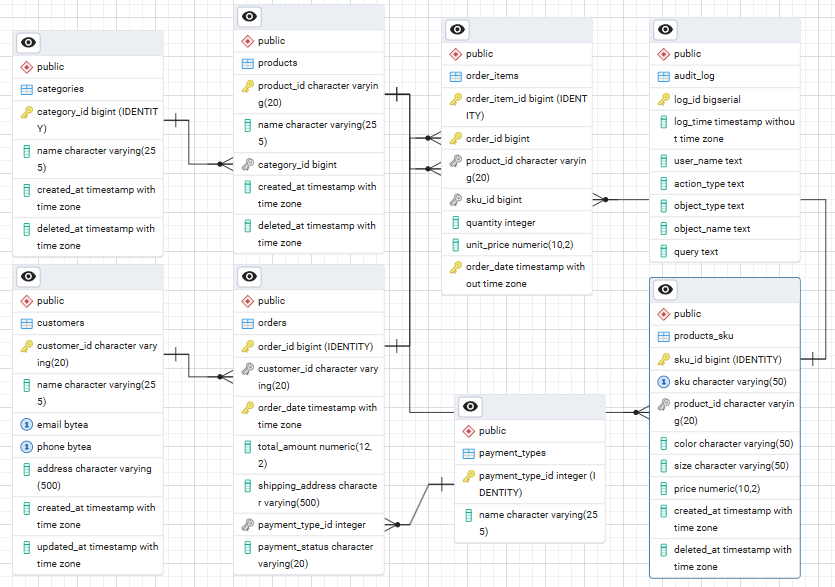
## Giải pháp triển khai

* **Cơ sở dữ liệu giao dịch (OLTP):** **PostgreSQL** cho từng microservice (PoS, Logistics, ERP).
* **Streaming ingestion:** **Kafka / Debezium** để đồng bộ dữ liệu real-time về Data Lake.
* **Batch ingestion: Sử dụng Airflow/ Airbyte để tạo các scheduled job nhằm load dữ liệu lên warehouse theo script.**
* **Data Lakehouse:** **Databricks** theo mô hình Medallion (Bronze → Silver → Gold).

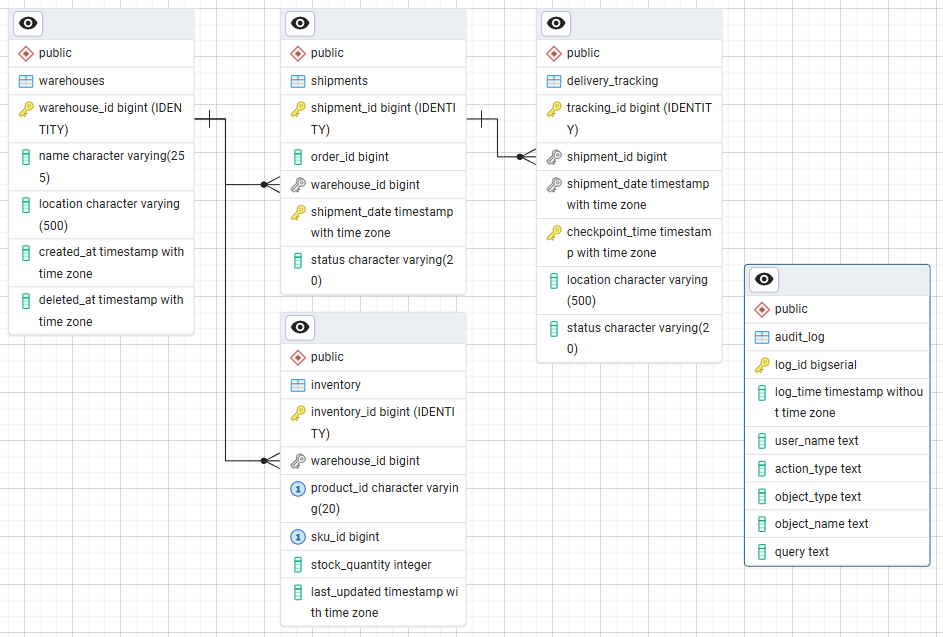
**Cấu trúc lớp OLTP Database**

1. **Yêu cầu kĩ thuật:**

* **Audit & Logging:** Các thao tác trên database phải được ghi lại log.
* **PII Protection:** Các dữ liệu PII phải được mã hoá và chỉ có thể truy cập trên một role nhất định được cấp.
* **RBAC:** nguyên tắc **Least Privilege**- mỗi role chỉ được cấp quyền đúng theo nhu cầu sử dụng.
* **Data Integrity & Constraints:** Bắt buộc **PK, FK, Unique, Check** để đảm bảo toàn vẹn. Duy trì **referential integrity** kể cả khi soft delete.
* **Backup & Recovery:** Phải có phương án backup (full backup và incremental backup) và recover dữ liệu định kỳ
* **Performance & Scalability:** Các dữ liệu được truy vấn thường xuyên được indexing và partition để đảm bảo hiệu suất truy vấn.
* **Soft Delete & Data Lifecycle:** Các dữ liệu được thiết kế để soft-deleted và có các trigger để xử lý dữ liệu liên quan sau khi delete.
* Dữ liệu đảm bảo tiêu chuẩn **ACID.**
* **View Layer for Data Access:** Tạo lớp View để truy cập dữ liệu nhằm hạn chế các request trên database.
* **Retention Limit:** Để đảm bảo hiệu suất, Database chỉ lưu dữ liệu active và đã unactive trong 3 tháng. Các dữ liệu cũ hơn sẽ được đồng bộ và lưu tại warehouse

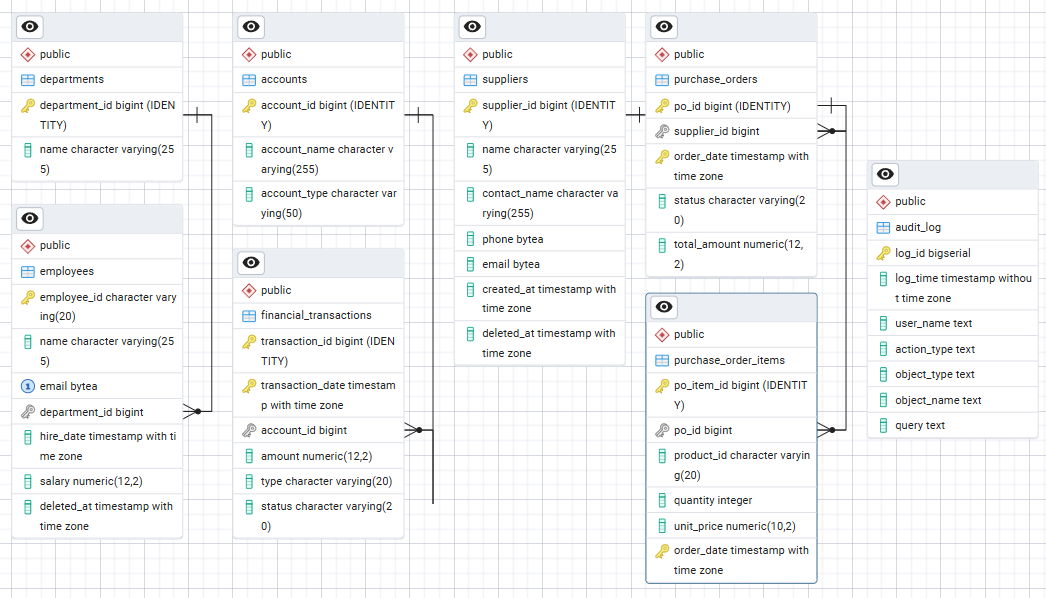
1. Cấu trúc Database:   
     
   

Hình 1. PoS\_database ERD



Hình 3. Logistic\_database ERD

Hình 2. ERP\_database ERD



1. **Cơ chế phân quyền: (môi trường production)**

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Role** | **Đối tượng** | **Quyền** |
| 1 | DB Admin | Pos.\*, logistic.\*,ERP.\* | ALL PRIVILEGES (DDL + DML + Security + Resource) |
| 2 | DB Engineer | Pos.\*, logistic.\*,ERP.\* | DML (SELECT,INSERT,UPDATE )  Không có quyền delete row |
| 3 | DB Analyst | Pos.view.\*, logistic.view.\*,ERP.view.\* | Only SELECT |
| 4 | Sale Application | Pos. [các bảng nghiệp vụ] | SELECT,INSERT,UPDATE |
| 5 | Logistic Application | Logistic. [các bảng nghiệp vụ] | SELECT,INSERT,UPDATE |
| 6 | ERP Application | ERP. [các bảng nghiệp vụ] | SELECT,INSERT,UPDATE |
| 7 | Các role nghiệp vụ liênquan PII. | Chỉ trên bảng chứa các PII cần thiết | SELRCT |
| 8 | Các role khác tuỳ nghiệp vụ | Chỉ các View tuỳ nghiệp vụ | SELECT |

* Các role application chỉ được cấp cho lớp application backend. Các yêu cầu truy cập từ end user chỉ cấp quyền trên view.
* Các role application không được xem dữ liệu PII.
* Các role nghiệp vụ liên quan PII chỉ được cấp và sử dụng trên một số function nghiệp vụ nhất định.

1. **Mã hoá PII:**

* Các dữ liệu được đánh dấu là PII gồm sđt va mail của khách hàng, nhân viên, nhà cung cấp.
* PostgreSQL hỗ trợ mã hóa thông qua extension **pgcrypto**.
* Symmetric key nên được thay đổi định kỳ (6 tháng).

1. **Trigger & Constraint**

* **Constraint về tính toàn vẹn thực thể.** 
  + Mỗi bảng phải có **Primary Key** để định danh duy nhất bản ghi.
  + Các thuộc tính quan trọng phải có **NOT NULL** để tránh rỗng.
  + Các thuộc tính cần đảm bảo **không trùng lặp** dùng **UNIQUE**
  + **Trigger** có thể được dùng để sinh tự động các mã định danh.
* **Constraint về tính toàn vẹn tham chiếu.**
  + Các **Foreign Key (FK)** liên kết giữa bảng phải được định nghĩa, đảm bảo giá trị tồn tại trong bảng cha.
  + Với các thuộc tính tham chiếu liên bảng qua hệ thống khác thì không thể dùng FK trực tiếp → sẽ được **check ở tầng ứng dụng qua API** trước khi insert/update.
  + **Trigger** có thể kiểm tra tính hợp lệ của dữ liệu tham chiếu trong các trường hợp đặc biệt (nếu không check được bằng constraint SQL)
* **Constraint về tính toàn vẹn ngữ nghĩa.**

Đảm bảo dữ liệu lưu trữ phản ánh đúng logic nghiệp vụ. Ví dụ:

* total\_amount của đơn hàng phải bằng tổng (quantity \* unit\_price) của order\_items.
* shipping\_address mặc định bằng địa chỉ khách hàng nhưng có thể override khi order.
* payment\_status chỉ được thay đổi sang completed nếu total\_amount > 0 và hệ thống thanh toán trả về thành công.
* **Constraint ràng buộc miền.**
  + Các thuộc tính tượng trưng cho số lượng thực phải đảm bảo dương.
  + Các thuộc tính kiểu ngày tháng phải đảm bảo nhỏ hơn thời gian hiện tại.
  + Các giá trị kiểu chuỗi phải có độ dài lớn hơn 0.
  + Các giá trị soft\_delete time hoặc update\_time nếu có phải nằm sau thời gian khởi tạo create\_at.
* **Constraint ràng buộc hợp lệ giá trị**
  + Một số thuộc tính có tập giá trị cố định phải được check trước khi insert hoặc update

1. **Function trong Database:**

* Tất cả insert/update/delete sẽ đi qua function/procedure, không thao tác trực tiếp bảng.
* Function cần:
  + Validate input (check null, check FK hợp lệ, check constraint logic).
  + Ghi log/audit nếu có thay đổi.
  + Tự động sinh ID nếu cần (nếu không dùng trigger).
  + Update timestamp (created\_at, updated\_at).
* **Function nghiệp vụ**: function đảm bảo đồng bộ khi có sự thay đổi thông tin giữa các đối tượng nghiệp vụ.

1. **Scheduled Job:**Trong dự án này ta sẽ sử dụng công cụ Airflow để tạo các scheduled job cho PostgreSQL. Airflow ngoài giúp quản lý các scheduled job và xây dựng dependency graph còn có các cơ chế quản lý retry khi có job failure và thông báo đến email mỗi khi có vấn đề.

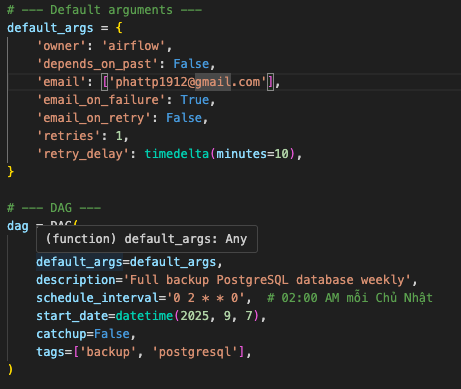


Figure 1. airflow config

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Job | Mô tả | Tần suất | Ghi chú | Ưu tiên |
| Change Encryption Key | Thay key mã hóa dữ liệu nhạy cảm | Hàng tháng | Cần backup trước khi thực hiện | Cao |
| Delete Old Data | Xóa dữ liệu theo retention plan | Hàng tuần | Có thể chạy sau backup | Cao |
| Load Data to Warehouse | Đồng bộ dữ liệu OLTP → OLAP | Hàng ngày | Chạy sau khi backup và khi dữ liệu ít thay đổi nhất. | Cao |
| Load Audit Log | Đồng bộ audit log sang warehouse | Hàng ngày | Chạy sau dữ liệu nguồn sẵn sàng | Trung bình |
| Backup Database | Full / incremental backup | Full: hàng tuần; Incremental: hàng ngày | Nên chạy trước Delete Old Data / sau Change Key | Cao |
| Create Monthly Partition | Tạo partition mới cho table time-based | Đầu tháng | Chạy trước Load Audit Log | Trung bình |

1. **Audit & Logging**

Hệ thống sử dụng bảng audit\_log và trigger trong mỗi database để ghi lại mọi thao tác **DML (INSERT, UPDATE, DELETE)** và **DDL (CREATE, ALTER, DROP)**. Cơ chế này giúp theo dõi **ai, khi nào, làm gì** trên cơ sở dữ liệu, hỗ trợ kiểm toán và đảm bảo an toàn dữ liệu.

1. **Sao lưu và backup**

**Hình thức backup:**

Full backup: Sử dụng công cụ pg\_dump để tạo bản sao lưu logic, bao gồm cả dữ liệu và cấu trúc cơ sở dữ liệu trong tệp.

Incremental backup: Tận dụng cơ chế Write-Ahead Logging (WAL) của PostgreSQL để ghi lại các thay đổi kể từ bản sao lưu toàn phần gần nhất.

**Lịch backup:** 1 tuần 1 lần (Chủ nhật, 02:00 AM) với Full backup. Và 1 ngày/ 1lần với incremental backup.

**Nơi lưu trữ:** Cloud Storage (AWS S3, Google Cloud Storage, hoặc Google Drive).

**Quản lý vòng đời:** Giữ lại các bản backup trong vòng **30 ngày**, tự động xóa các bản cũ hơn để tiết kiệm dung lượng.

1. **Retention limit:**Vì database đóng vai trò là OLTP chỉ lưu các dữ liệu “hot” để đảm bảo khả năng lưu trữ, truy vấn nhanh nên các dữ liệu được đồng bộ liên tục lên warehouse đóng vai trò OLAP. Do vậy, có thể định kỳ kiểm tra và xoá các dữ liệu không còn quan trọng với vai trò của OLTP mà không ảnh hưởng đến các tiêu chí về toàn vẹn dữ liệu. Các dữ liệu đấy bao gồm:

* Các dữ liệu bị soft-delete quá 3 tháng.
* Thông tin các giao dịch đã hoàn thành hoặc bị huỷ quá 3 tháng.
* Dữ liệu log về hoạt động trên database quá 3 tháng.

Các thông tin đã bị xoá vẫn có thể truy vấn trực tiếp từ warehouse nên không ảnh hưởng đến tính toàn vẹn dữ liệu của hệ thống

**Đánh giá một số rủi ro trên thiết kế lớp Database:  
 1. Quá tải giao dịch (Transaction Overhead):**

Toàn bộ thao tác DML bắt buộc đi qua **function/procedure** → tốt cho kiểm soát, nhưng sẽ làm tăng độ trễ khi hệ thống có hàng triệu giao dịch/tháng.

**2. Đồng nhất dữ liệu FK giữa các Database:**

Không dùng FK giữa các hệ thống khác nhau (PoS ↔ Logistics ↔ ERP) mà rely on API check → dễ xảy ra **data inconsistency** khi service API fail hoặc có latency.

**3. PII encryption (pgcrypto):**

Symmetric key đổi mỗi 6 tháng → cần **key rotation plan.**

**Một số phương hướng phát triển tiếp để cải thiện hiệu suất và khắc phục các rủi ro:**

**1. Xây dựng replica server:**Sử dựng cơ chế streaming replication sao chép toàn bộ dữ liệu ở mức binary (WAL) từ primary server sang replica server gần như real-time để:

- Tăng khả năng chịu tải (scalability): Các replica server có thể xử lý các truy vấn Select (Read-only), giảm tải cho primary server trong giai đoạn cao điểm.

- Đảm bảo high availability: nếu primary server gặp lỗi (crash, network failure), replica server có thể tiếp quản.

- Hỗ trợ disaster recovery: Dữ liệu được sao lưu tại nhiều Node, giúp giảm nguy cơ mất dữ liệu.

Các công cụ như Patroni có thể hỗ trợ cơ chế Failover mechanism nhằm theo dõi primary server, tự động phát hiện lỗi và chuyển sang replica server.

**2. Xây dựng Monitoring Tools:**

Monitoring Tools là các công cụ được sử dụng để:  
 - Theo dõi hiệu suất: query performance, CPU/disk usage, replication log, …

- Phát hiện vấn đề: failure như job Airflow fail, constraint violation hoặc database overload.  
- Alert: Thông báo ngay lập tức các vấn đề phát sinh qua Slack, Email,…khi có sự cố và daily Report.

- Phân tích và tối ưu: Cung cấp dữ liệu để tối ưu và debug hệ thống.

Các công cụ như Prometheus nhằm giám sát và thu thập metrics theo thời gian thực và Grafana để trực quan hoá dữ liệu.  
3. **Phát triển thêm các cơ chế bảo mật dữ liệu khi cấu trúc bảng phức tạp và phục vụ multi-tenant.**Ngoài encryption PII thì cần phát triển thêm cơ chế Row-Level-Security cho dữ liệu nhằm tránh truy cập thông tin ngoài thẩm quyền.

Áp dụng các cơ chế key rotation và key management có thể với AWS KMS nhằm nâng cao khả năng encrytion.

4. **Áp dụng các phương án cải thiện Performance và Scalability:**Nếu sử dụng với quy mô hàng triệu query / tháng thì Db có rủi ro quá tải vào các giai đoạn cao điểm như flash sale thì phát triền các phương án cân bằng tải, connection pooling, query optimization tool sẽ giúp cải thiện Performance.

**Cấu trúc lớp ELT Process**

Hệ thống OLTP thường dùng để xử lý các giao dịch hàng ngày, có cấu trúc dữ liệu quan hệ chuẩn hóa, tối ưu cho **insert/update/delete** nhanh. Trong khi đó, hệ thống **Data Warehouse (OLAP)** được tối ưu cho **phân tích, truy vấn và báo cáo**.   
Việc **chuyển dữ liệu từ OLTP sang OLAP** giúp doanh nghiệp phân tích dữ liệu lịch sử và ra quyết định dựa trên các thông tin tổng hợp. Quá trình này được gọi chung là **ELT (Extract - Load - Transform).**ELT có 2 phương án:

**Streaming Processing**Với **streaming ELT**, dữ liệu từ OLTP được **đẩy liên tục** sang OLAP nhờ **Kafka** và **Debezium**:

* **Debezium:** Là công cụ **Change Data Capture (CDC)**, giám sát các thay đổi trên database OLTP (insert/update/delete) và phát hiện sự kiện.
* **Kafka:** Là hệ thống **messaging/streaming**, nhận các sự kiện từ Debezium và phân phối chúng đến các hệ thống downstream (bao gồm cả Data Warehouse).

**Quá trình:**

* Bước 1: launch EC2 instance.
* Bước 2: Cài postgreSQL và cấu hình postgresql.conf / pg\_hba.conf .
* Bước 3: Tạo user replication trong PostgreSQL và publication cho logical replication.
* Bước 4: Triển khai kafka + kafka connect + debezium.
* Bước 5: Kết nối Databricks đến kafka

**Thực tế dự án:**  
Vì phiên bản Databricks Community Edition không hỗ trợ full feature cho việc streaming dữ liệu từ kafka đồng thời hạn chế về mặt môi trường cloud để chạy database và kafka nên tôi đã triển khai mô hình như sau:

* Sử dụng docker để cài đặt môi trường cho kafka + debezium + zookeeper.



Figure 2. Cấu hình kafka + debezium trong docker

* Kết nối debezium và cấu hình WAL cho postgresSQL để đọc thay đổi.

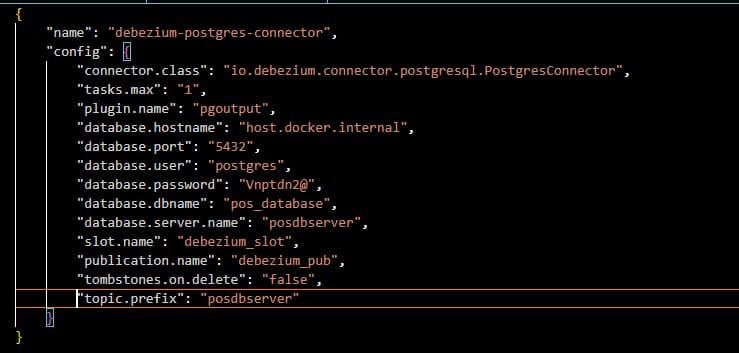
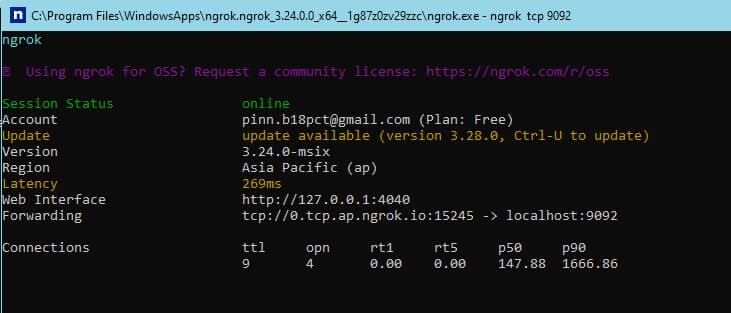


Figure 3. Cấu hình connection cho debezium

* Sử dụng ngrok để mở tunel từ máy local cho kết nối databricks



* Databricks sẽ kết nối và test connection bằng lệnh batch thay cho Streaming.

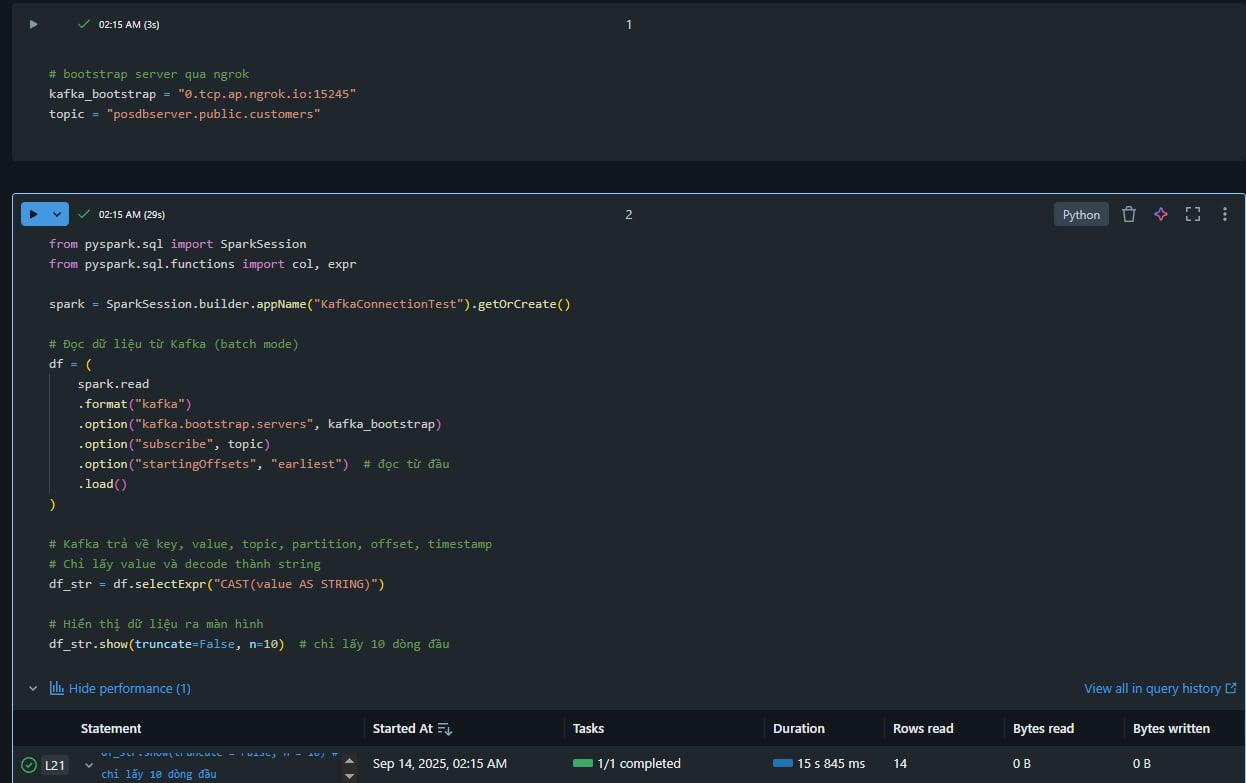


Figure 4. Kết nối dữ liệu từ Databricks

**Batch Processing.**Để giảm chi phí vận hành so với việc duy trì các pipeline xử lý dữ liệu theo thời gian thực (streaming), đối với các bảng có tần suất thay đổi thấp, ta có thể sử dụng phương pháp incremental load dựa trên watermark thông qua các scheduled job. Công cụ phổ biến để triển khai batch processing là Airflow với cách tiếp cận như sau:

* **Batch Load bằng công cụ Airflow:**
  + Airflow là một **nền tảng mã nguồn mở** dùng để **lập lịch, quản lý và giám sát luồng công việc (workflow orchestration).**
  + Airflow cho phép bạn định nghĩa các **workflow** dưới dạng **DAG** – tức là một tập hợp các tác vụ (tasks) có quan hệ phụ thuộc lẫn nhau, nhưng không có vòng lặp.
  + Mỗi tác vụ (task) thường là một đoạn code (Python, Bash, SQL…) hoặc một job cụ thể.
  + Airflow cung cấp một giao diện web để giám sát, khởi chạy và kiểm soát luồng công việc.

**Quy trình:**

* Scheduled job được cấu hình định kỳ hằng ngày sẽ query dữ liệu mới từ các bảng dựa trên giá trị watermark updated\_at.
* Dữ liệu sau đó được ghi thành file parquet với tên file theo ngày tháng được tạo.
* Dữ liệu được up lên cloud storage.
* Luồng AutoLoader được cấu hình chạy hằng ngày trên Databricks sẽ theo dõi cloud storage và load dữ liệu vào bảng bronze.

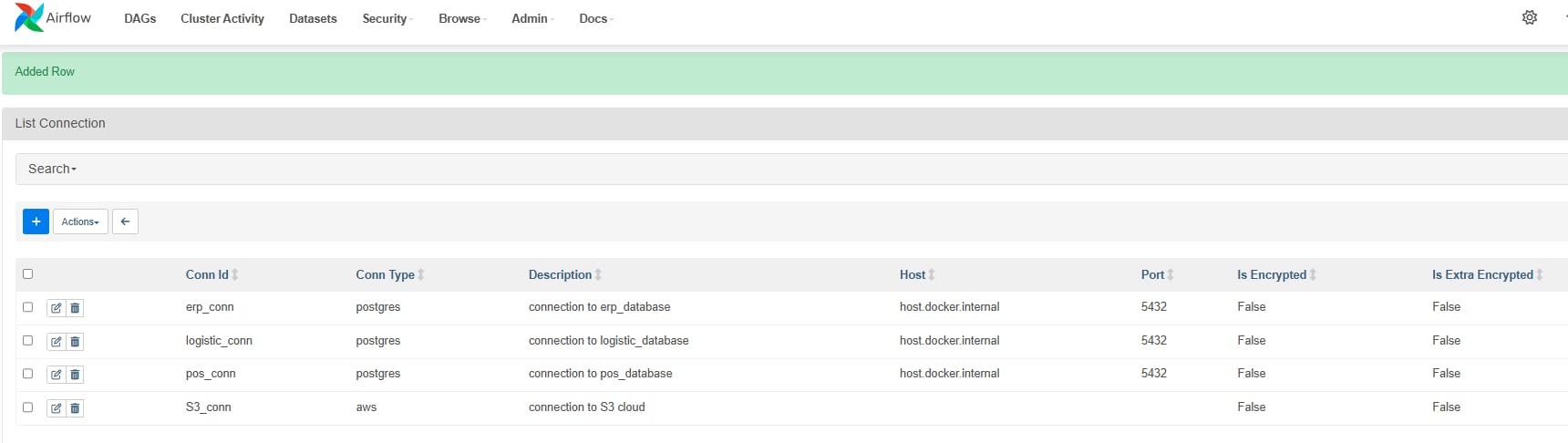
Các bước cấu hình Airflow: ****

Figure 5. Cấu hình kết nối Airflow đến PostgreSQL và S3

****

Figure 6. Function query dữ liệu từ PostgreSQL

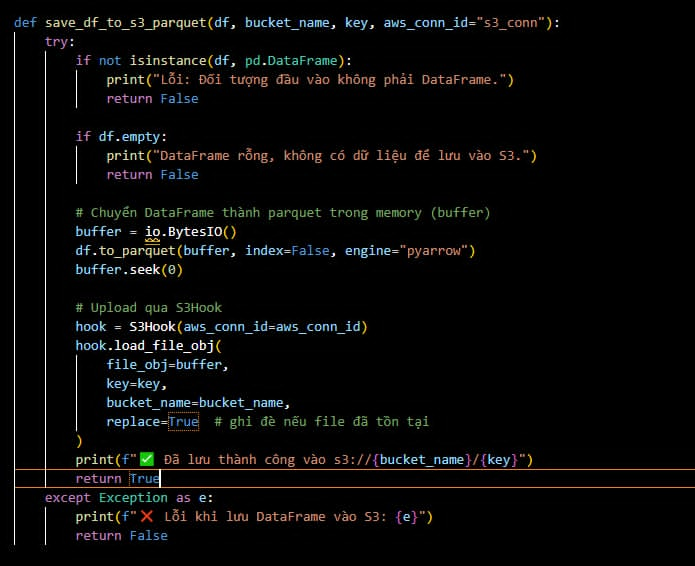
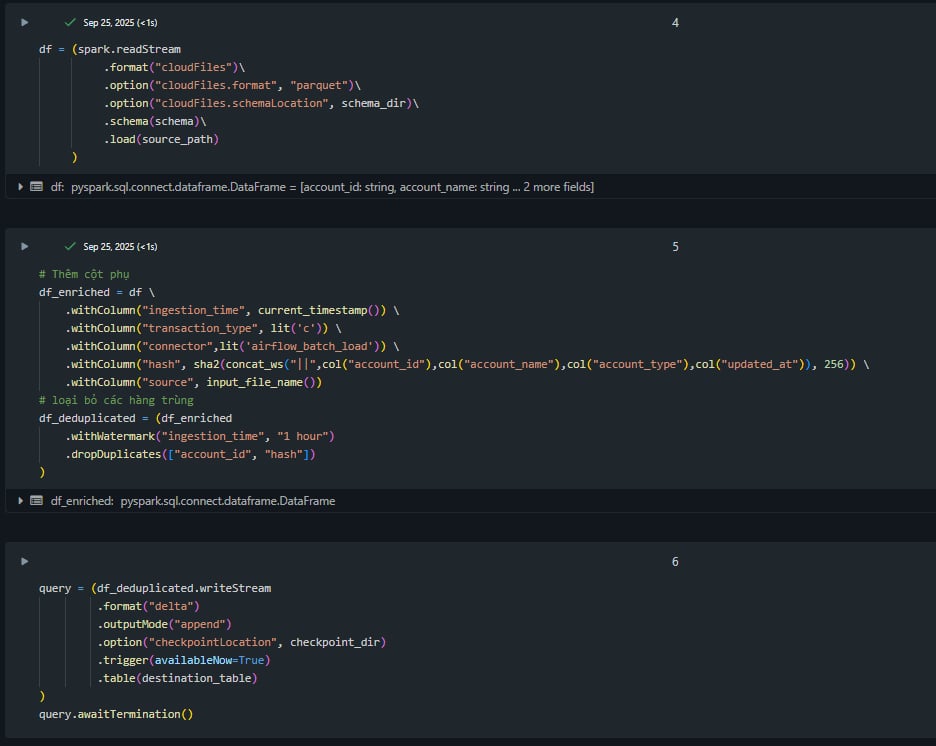
****

Figure 7. Function lưu file vào parquet

Thiết lập luồng AutoLoader của Databricks:

****

****

Bảng so sánh 2 luồng dữ liệu và các tiêu chí để chọn phương pháp ELT:

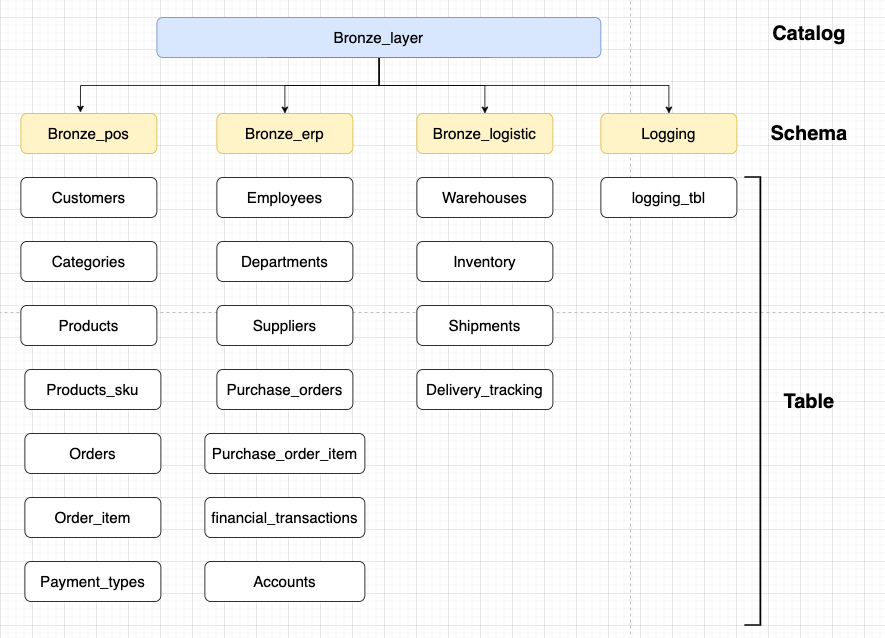
|  |  |  |
| --- | --- | --- |
|  | **Streaming** | **Batch** |
| Ưu điểm | - Độ trễ thấp (real time).  - Xử lý dữ liệu liên tục, không cần chờ batch.  - Hỗ trợ analytics thời gian thực (e.g : realtime dashboard cho orders, inventory) | - Đơn giản triển khai và debug (dễ kiểm soát job qua DAGs)  - Chi phí thấp hơn (chạy định kỳ, không cần tài nguyên liên tục).  - Phù hợp cho dữ liệu ổn định, dễ đảm bảo tính nhất quán |
| Nhược điểm | - Phức tạp cao (cần xử lý fault-tolerance, exactly-once semantics)  - Chi phí cao hơn (tài nguyên Databricks chạy liên tục)  - Khó debug và monitor nếu lỗi xảy ra (e.g: kafka lag) | - Độ trễ cao (e.g: hàng giờ/ hàng ngày)  - Không xử lý tốt dữ liệu thay đổi nhanh.  - Có thể overload nếu batch quá lớn. |
|  | Áp dụng cho dữ liệu có high velocity như orders, transactions,… | Áp dụng cho dữ liệu có low velocity như product, categories,.. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Tiêu chí** | **Mô tả** | **Streaming** | **Batch** |
| Tần suất cập nhật dữ liệu | Số lần thay đổi/giờ hoặc % dữ liệu mới so với tổng. | **Customers**: Khách hàng đăng ký liên tục và tần suất cao đặc biệt là thời điểm khuyến mãi.  **Orders** & **Order\_item**: dữ liệu liên quan đến đơn hàng cần nhu cầu phân tích realtime và nghiệp vụ kho.  **Inventory**: liên quan đến phân tích quản lý tồn kho nên cần nhu cầu realtime.  **Shipment** & **Delivery Tracking:** nhu cầu cập nhật realtime vì liên quan đến nghiệp vụ quản lý vận đơn.  **Purchase\_orders** & **Purchase\_order\_items**: liên quan đến nghiệm vụ mua hàng và quản lý kho, cần real-time.  **Financial\_Transactions**: liên quan đến báo cáo dòng tiền của công ty. | **Produc**t & **Product SKU** & **Categories** : quá trình nghiên cứu và phát triển sản phẩm cần thời gian nghiên cứu dài, k diễn ra thường xuyên.  **Payment\_type**: dimension table chứa dữ liệu hầu như ít thay đổi và thêm dữ liệu mới.  **Warehouse:** liên quan đến cơ sở vật lý nên rất ít thay đổi.  **Employees & Departments**: liên quan đến nhân sự công ty nên tần suất thay đổi thấp.  **Supplier**: liên quan đến nhà cung cấp. tần suất thay đổi thấp.  **Accounts**: liên quan đến tài khoản thanh toán.tần suất thay đổi thấp.  **Audit Logging:** chỉ nhằm mục đích lưu trữ cho việc truy xuất. không có nhu cầu phải real-time. |
| Độ nhạy cảm thời gian thực | Cần dữ liệu mới nhất cho business decision (e.g., fraud detection). |
| Khối lượng dữ liệu | Volume và velocity (e.g., records/phút). |
| Độ phức tạp xử lý | Cần join, aggregation phức tạp hoặc stateful processing. |
| Tính nhất quán và lỗi | Yêu cầu exactly-once hoặc at-least-once semantics. |
| Chi phí và Tài nguyên | Ước tính cost Databricks clusters và Kafka throughput. |

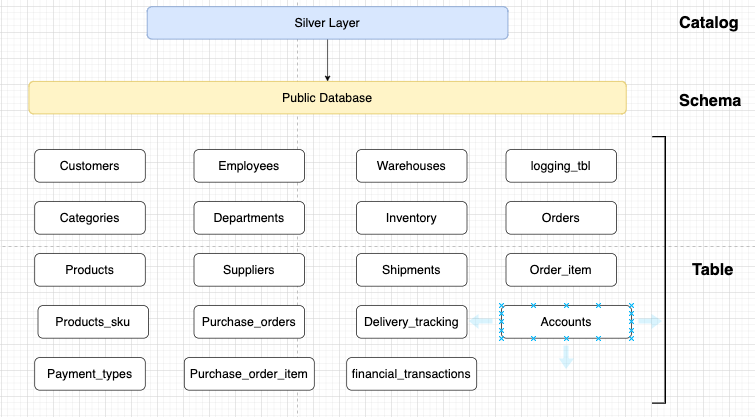
**Cấu trúc lớp OLAP Databricks**

**Kiến trúc tổng thể**

* **Mô tả mô hình Medallion**:
  + **Bronze**: Lưu trữ dữ liệu thô từ các databse với định dạng gốc nhằm đảm bảo “ true of source” (trung thực với dữ liệu gốc.)



* + **Silver**: dữ liệu đã làm sạch, chuẩn hoá và enrich. Các nguồn dữ liệu từ các microservice khác nhau được gom vào một database duy nhất để dễ dàng xử lý.



* + **Gold**: dữ liệu đã được tổng hợp, xử lý và tối ưu hoá để làm nguồn dữ liệu cho BI và Machine Learning.

**Diagram kiến trúc:**

1. **Bronze Layer:**

* Dữ liệu từ database giữ nguyên cấu trúc, mỗi bảng nguồn tương ứng một bảng Bronze.
* Thêm các cột liên quan ingestion metadata như: ingestion\_time (thời gian ingestion vào ware house), connector (cổng dữ liệu đầu vào qua kafka, API,..), source (nguồn dữ liệu như tên table ở Database,..)
* Thêm các cột liên quan đến Change Tracking Data như : record\_update\_at, record\_delete\_at nếu chưa có.
* Dữ liệu hash hoặc checksum tổng hợp từ data chính để theo dõi và so sánh dữ liệu mà không cần so sánh từng dòng.
* Ở đây ta giữ lại các thông tin trong mục payload ta giữ lại các data chính trong mục after để mapping vào các cột tương ứng cho bảng trong lớp bronze.
* Mục “ts\_ms” chứa thông tin timestamp dữ liệu được thu thập từ database mapping vào cột ingestion\_time.
* Mục “connector” và “name” cho ta biết cổng đầu vào của dữ liệu được mapping vào cột connector.
* Mục “schema” , “db”,”table” cho ta biết nguồn gốc của dữ liệu được mapping vào cột source.
* Mục “op” cho ta biết phân loại của dòng dữ liệu (“r”: snapshot, “c”: create, “u”: update “d”: delete).

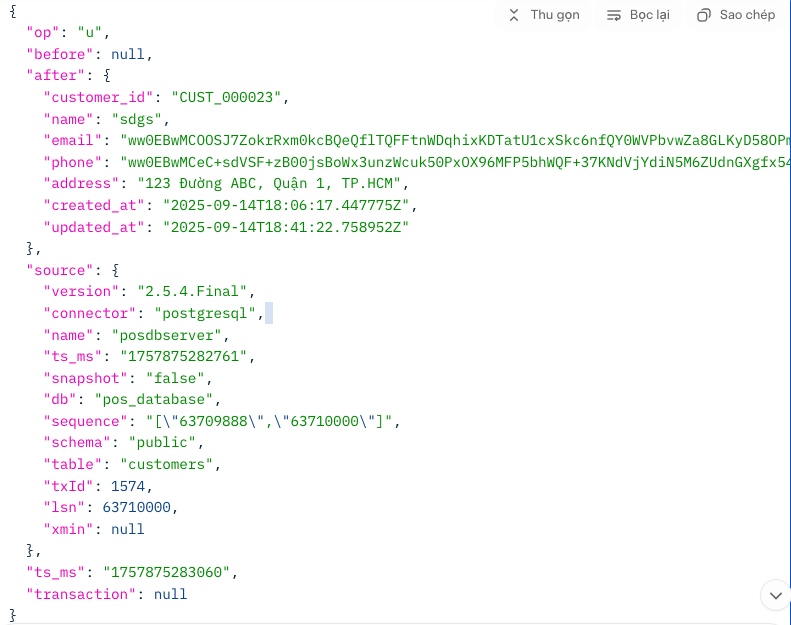
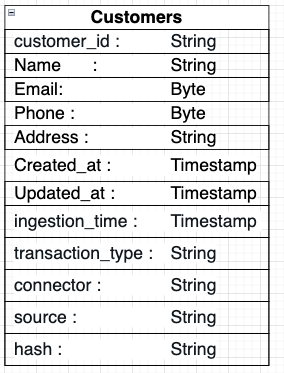
**** 

Figure 8. Mẫu cấu trúc payload message từ kafka.

1. **Silver Layer:**

* Làm sạch và chuẩn hoá các thông tin.
* Dữ liệu từ bảng bronze phải được xử lý Null và chuẩn hoá theo các tiêu chuẩn kinh doanh.
* Dữ liệu từ bảng bronze sẽ được so sánh với dựa trên ID và hash (mảng hash được tạo ra từ tất cả các dữ liệu chính của bảng và được sử dụng để so sánh xem 2 record có sai khác không).

**Quy trình:**

* Dữ liệu được lấy từ bảng bronze trong thời gian 2 ngày (vì data pipeline được kích hoạt mỗi ngày).
* Dữ liệu trong batch được lấy từ bronze sẽ được deduplicated dựa trên dữ liệu hash trước. Sau đó được group lại theo ID của mỗi bảng và sắp xếp chỉ lấy dữ liệu mới nhất và merge vào bảng Silver.
* Điều kiện merge phải dựa trên cả ID và cột hash tránh overwrite lại dữ liệu giống nhau.



Figure 9. xử lý dữ liệu từ bảng bronze trước khi merge

****

Figure 10. Merge dữ liệu vào bảng Silver

3. Gold Layer: join các bảng liên quan lại và sắp xếp lại các thông tin cần thiết cho việc truy vấn của BI cũng như nguồn dữ liệu cần thiết cho ML. **Yêu cầu kỹ thuật của lớp OLTP:**

1. **RBAC (Role-Based Access Control):**

Áp dụng nguyên tắc Least Privilege trong Unity Catalog. Quyền truy cập được phân chia theo layer:

|  |  |  |  |
| --- | --- | --- | --- |
| Role | Bronze | Silver | Gold |
| Data Engineer | Read/Write | | Write |
| Data Analyst | None | Read | |
| BI Tools | None | None | Read |
| ML Engineer | None | Read (Table/View), Read/Write (Feature Store) | |

1. **Partition Strategy:**

* **Bronze Layer:** 
  + Với lớp bronze thì các truy vấn duy nhất là luồng truy vấn dữ liệu từ tầng bronze lên Silver. Vì vậy các bảng này phải được partition theo cột ingestion\_time. Theo tiêu chuẩn thì mỗi partition tối thiểu nên là 1GB dữ liệu, nên dựa theo lượng dữ liệu của hệ thống mỗi ngày thì có thể chọn partition theo ngày hoặc theo tháng.
  + Để tối ưu hệ suất thì dữ liệu thường chỉ query trong 1 tháng đầu tiên có thể áp dụng chế độ Hybrid-partition. Định kỳ mỗi tháng chạy job re-partition lại các dữ liệu cũ hơn 1 tháng từ ngày thành partition theo tháng.
* **Silver Layer:**
  + Cần thiết kế riêng partition strategy tuỳ thuộc theo nghiệp vụ của mỗi bảng. Mục đích chính là nhằm tối ưu query performance.
  + Ngoài áp dụng các cột theo partition theo tiêu chuẩn. thì có thể áp dụng thêm Z-order để tối ưu performance.
* **Gold Layer:**
  + Tương tự như tầng silver thì partition trên layer này nhằm tối ưu query performance. Tuy nhiên nhiều bảng gold layer đã ở mức aggregated/denormalized nên có thể không cần partition phức tạp.

1. **Audit & Logging:**

* Kích hoạt chức năng Unity Catalog audit logs cho mọi hành động và có thể lưu vào cloud storage.
* Kích hoạt Delta Table Change Data Feed (CDF) để theo dõi sự thay đổi dữ liệu ở các bảng.
* Thêm các cột metadata cần thiết ở các bảng như (batch\_id/run\_id, source system + ingestion timestamp, checksum/hash) để quản lý và tracing data.
* Thiết kế một bảng chứa log cho các data pipeline được scheduled để theo dõi hoạt động của hệ thống.
* Thông tin cần log:
  + job\_id, task\_id, user
  + start\_time, end\_time, duration
  + input\_rows, output\_rows
  + error stacktrace (nếu có)

1. **PII Protection:**
   * Sử dụng Centralized Access Control của Unity Catalog để giới hạn truy cập dữ liệu PII.
   * Sử dụng Column Masking Policies cho các cột chứa PII.
   * Sử dụng Row-Level Security khi thiết kế query của end user trong trường hợp user thuộc role được quyền xem thì chỉ xem được dữ liệu thuộc scope quản lý của mình.
   * Sử dụng Customer Managed Keys (CMK) thay cho mã hoá mặc định của Databricks đối với dữ liệu At rest.
   * Key- Rotation định kỳ.
2. **Data Integrity & Constraints:**
   * **Entity integrity:** Mỗi bản ghi phải có khoá định danh duy nhất.
     1. **Bronze:** Mỗi dòng có một giá trị hash tổng hợp từ tất cả các column cần so sánh của record đấy. Chỉ cần so sánh giá trị hash sẽ đảm bảo các row không bị trùng. Và so sánh hase giữa giá trị tại OLTP và bronze cũng như giữa bronze và silver giúp phát hiện sai khác dữ liệu.
     2. **Silver-Gold:** Mỗi dòng của bản silver có giá trị ID làm khoá định danh vì chỉ cần giữ giá trị mới nhất.
   * **Referential integrity:** đảm bảo quan hệ giữa các bảng phải tồn tại và luôn update dữ liệu mới từ cả 2 chiều.
   * **Domain integrity:** Dữ liệu phải nằm trong phạm vi hợp lệ.
   * **Temporal integrity:** Dữ liệu phản ánh đúng thời điểm.
   * **Pipeline integrity:** Dữ liệu không bị duplicate / mất trong quá trình ELT.
   * Job định kỳ kiểm tra dữ liệu.
3. **Backup & Recovery:**
   * Sử dụng chức năng Time Travel để recovery dữ liệu.
   * Lưu data tại cloud storage và định kỳ back up dữ liệu tại storage.
   * Sync dữ liệu với các nền tảng version control như github để lưu config và setup hệ thống.
4. **Soft Delete & Data Lifecycle & Retention Limit**

* Chỉ sử dụng soft-delete, chứ không hard- delete dữ liệu thật.
* Có kế hoạch lưu trữ dữ liệu dựa trên nghiệp vụ kinh doanh để tối ưu chi phí.
* Các dữ liệu cũ ngoài hạn mức và ít có nhu cầu truy cập thì đóng gói chuyển sang vùng lưu trữ cho chi phí thấp hơn (archive zone).

1. **Performance & Scalability**
2. **ACID Compliance**
3. **View Layer for Data Access**