Proyecto: Sistema de Analytics para E-commerce con Arquitectura Medallion

Plataforma de Ventas Online "TechStore"

🎯 Objetivo General

Crear un sistema completo de analytics para una tienda online que venda productos tecnológicos, implementando arquitectura medallion en Databricks y sirviendo datos a través de FastAPI.

📋 Descripción del Proyecto

Desarrollarás un sistema de analytics end-to-end que incluye:

- **Ingesta de datos**: Poblado de tablas transaccionales en Databricks
- **Transformación**: Implementación de arquitectura medallion con DBT
- **Consumo**: API REST con FastAPI para servir insights de negocio

🏗️ Arquitectura del Sistema

```
None
[Tablas Transaccionales] → [Bronze Layer] → [Silver Layer] → [Gold Layer] → [FastAPI]
```

📚 Competencias y Tecnologías

Tecnologías Principales

- **Base de Datos**: Databricks (Delta Lake)
- **ETL/ELT**: DBT Core
- **API**: FastAPI + Pydantic
- **Orquestación**: DBT (transformaciones)
- **Gestión de Dependencias**: UV (Python)

Competencias Desarrolladas

- Diseño y manejo de bases de datos relacionales
- Implementación de arquitectura medallion
- Desarrollo de APIs RESTful
- Modelado de datos con DBT
- Métricas de negocio y KPIs
- Streaming de datos (opcional)

🗓️ Fases del Proyecto

📅 Fase 1: Setup y Población de Datos

Objetivo: Preparar el entorno y datos base

Configuración del Entorno

- **Databricks Setup:**
  - Crear workspace en Databricks Community Edition
  - Configurar cluster (Runtime 13.3 LTS o superior)
  - Configurar acceso y permisos
- **Estructura de Carpetas:**

```
None
/techstore/
├── raw_data/
├── bronze/
├── silver/
├── gold/
└── scripts/
```

Creación de Tablas

- **Ejecutar Scripts DDL:** Crear todas las tablas usando los scripts proporcionados
- **Validación de Estructura:** Verificar que todas las tablas se crearon correctamente

Aquí tienes como referencia el modelo de datos a través de las queries SQL.

```sql
-- Tabla: customers

CREATE TABLE IF NOT EXISTS techstore.customers (

    id BIGINT GENERATED ALWAYS AS IDENTITY,

    name STRING NOT NULL,

    email STRING NOT NULL,

    registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP(),

    birth_date DATE,

    phone STRING,

    city STRING,

    country STRING,

    is_active BOOLEAN DEFAULT TRUE,

    CONSTRAINT customers_email_unique UNIQUE (email)

) USING DELTA

LOCATION '/delta/techstore/customers'

TBLPROPERTIES (

    'delta.autoOptimize.optimizeWrite' = 'true',

    'delta.autoOptimize.autoCompact' = 'true'

);


-- Tabla: product_categories

CREATE TABLE IF NOT EXISTS techstore.product_categories (

    id BIGINT GENERATED ALWAYS AS IDENTITY,

    name STRING NOT NULL,
```

```
    parent_category_id BIGINT,
    description STRING,
    is_active BOOLEAN DEFAULT TRUE,
    CONSTRAINT categories_name_unique UNIQUE (name),
    CONSTRAINT categories_parent_fk FOREIGN KEY (parent_category_id)
REFERENCES techstore.product_categories(id)
) USING DELTA
LOCATION '/delta/techstore/product_categories'
TBLPROPERTIES (
    'delta.autoOptimize.optimizeWrite' = 'true'
);


-- Tabla: products
CREATE TABLE IF NOT EXISTS techstore.products (
    id BIGINT GENERATED ALWAYS AS IDENTITY,
    name STRING NOT NULL,
    description STRING,
    category STRING NOT NULL,
    subcategory STRING,
    brand STRING NOT NULL,
    model STRING,
    price DECIMAL(10,2) NOT NULL,
    cost DECIMAL(10,2) NOT NULL,
    stock INTEGER DEFAULT 0,
    weight_kg DECIMAL(5,2),
```

```
    dimensions STRING,
    launch_date DATE,
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
) USING DELTA
LOCATION '/delta/techstore/products'
TBLPROPERTIES (
    'delta.autoOptimize.optimizeWrite' = 'true',
    'delta.autoOptimize.autoCompact' = 'true'
);

-- Tabla: orders
CREATE TABLE IF NOT EXISTS techstore.orders (
    id BIGINT GENERATED ALWAYS AS IDENTITY,
    customer_id BIGINT NOT NULL,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP(),
    status STRING NOT NULL,
    payment_method STRING,
    payment_status STRING,
    subtotal DECIMAL(10,2),
    tax_amount DECIMAL(10,2),
    shipping_cost DECIMAL(10,2),
    discount_amount DECIMAL(10,2) DEFAULT 0,
    total_amount DECIMAL(10,2) NOT NULL,
```

```
    shipping_address STRING,

    estimated_delivery DATE,

    actual_delivery DATE,

    notes STRING,

    CONSTRAINT orders_status_check CHECK (status IN ('pending',
'confirmed', 'shipped', 'delivered', 'cancelled')),

    CONSTRAINT orders_payment_status_check CHECK (payment_status IN
('pending', 'completed', 'failed', 'refunded')),

    CONSTRAINT orders_customer_fk FOREIGN KEY (customer_id)
REFERENCES techstore.customers(id)

) USING DELTA

PARTITIONED BY (DATE(order_date))

LOCATION '/delta/techstore/orders'

TBLPROPERTIES (

    'delta.autoOptimize.optimizeWrite' = 'true',

    'delta.autoOptimize.autoCompact' = 'true'

);


-- Tabla: order_items

CREATE TABLE IF NOT EXISTS techstore.order_items (

    id BIGINT GENERATED ALWAYS AS IDENTITY,

    order_id BIGINT NOT NULL,

    product_id BIGINT NOT NULL,

    quantity INTEGER NOT NULL,

    unit_price DECIMAL(10,2) NOT NULL,
```

```
    unit_cost DECIMAL(10,2) NOT NULL,

    line_total DECIMAL(10,2) NOT NULL,

    discount_percent DECIMAL(5,2) DEFAULT 0,

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP(),

    CONSTRAINT order_items_order_fk FOREIGN KEY (order_id) REFERENCES
techstore.orders(id),

    CONSTRAINT order_items_product_fk FOREIGN KEY (product_id)
REFERENCES techstore.products(id)

) USING DELTA

LOCATION '/delta/techstore/order_items'

TBLPROPERTIES (

    'delta.autoOptimize.optimizeWrite' = 'true'

);


-- Tabla: reviews

CREATE TABLE IF NOT EXISTS techstore.reviews (

    id BIGINT GENERATED ALWAYS AS IDENTITY,

    product_id BIGINT NOT NULL,

    customer_id BIGINT NOT NULL,

    order_id BIGINT,

    rating INTEGER,

    title STRING,

    comment STRING,

    review_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP(),

    is_verified_purchase BOOLEAN DEFAULT FALSE,
```

```
    helpful_votes INTEGER DEFAULT 0,

    total_votes INTEGER DEFAULT 0,

    CONSTRAINT reviews_rating_check CHECK (rating BETWEEN 1 AND 5),

    CONSTRAINT reviews_product_fk FOREIGN KEY (product_id) REFERENCES
techstore.products(id),

    CONSTRAINT reviews_customer_fk FOREIGN KEY (customer_id)
REFERENCES techstore.customers(id),

    CONSTRAINT reviews_order_fk FOREIGN KEY (order_id) REFERENCES
techstore.orders(id)

) USING DELTA

PARTITIONED BY (DATE(review_date))

LOCATION '/delta/techstore/reviews'

TBLPROPERTIES (

    'delta.autoOptimize.optimizeWrite' = 'true'

);
```

Población de Datos

Generar y cargar datos realistas que incluyan:

- **Customers:** 1,000+ clientes de diferentes países
- **Product_categories:** 15-20 categorías
- **Products:** 200+ productos tecnológicos variados
- **Orders:** 5,000+ órdenes distribuidas en 24 meses
- **Order_items:** Items correspondientes (promedio 2.5 items/orden)
- **Reviews:** 2,000+ reseñas distribuidas

📅 Fase 2: Arquitectura Medallion - Capas Bronze y Silver

Objetivo: Implementar las primeras dos capas de la arquitectura medallion

Setup DBT

- **Instalación y Configuración:**

```shell
pip install dbt-databricks

dbt init techstore_analytics
```

- **Configuración `dbt_project.yml`:**

yaml

```none
name: 'techstore_analytics'

version: '1.0.0'

config-version: 2


profile: 'databricks'


model-paths: ["models"]

test-paths: ["tests"]


models:

  techstore_analytics:

    bronze:

      +materialized: table

      +file_format: delta

    silver:
```

```
      +materialized: table
      +file_format: delta
    gold:
      +materialized: table
      +file_format: delta
```

Capa Bronze (Raw Data)

- **Objetivo:** Ingesta sin transformaciones, solo metadatos
- **Modelos a Crear:**
  - `bronze_customers.sql`
  - `bronze_products.sql`
  - `bronze_orders.sql`
  - `bronze_order_items.sql`
  - `bronze_reviews.sql`
  - `bronze_product_categories.sql`

Capa Silver (Clean & Standardized)

- **Objetivo:** Limpieza, estandarización y enriquecimiento
- **Transformaciones a Implementar:**
  - Limpieza de datos nulos y duplicados
  - Estandarización de formatos (emails, teléfonos, nombres)
  - Enriquecimiento con datos calculados
  - Validaciones de calidad de datos

📅 Fase 3: Capa Gold - Métricas de Negocio

Objetivo: Crear insights y métricas de valor para el negocio

Customer Analytics

- **Modelo**: `gold_customer_analytics.sql`
- **Métricas a Calcular**: **Customer Lifetime Value (CLV)**
  - **Definición**: Valor monetario total proyectado de un cliente
  - **Fórmula**: `(Ticket Promedio × Frecuencia × Tiempo de Vida) - Costo Adquisición`
- **Segmentación RFM**
  - **Recency**: Días desde última compra (1-5 scale)
  - **Frequency**: Número de órdenes (1-5 scale)

    - **Monetary**: Valor total gastado (1-5 scale)
- **Clasificación de Clientes**:
    - VIP: RFM ≥ 444
    - Loyal: RFM ≥ 333
    - New: R ≥ 4, F = 1
    - At Risk: R ≤ 2, F ≥ 2
    - Lost: R ≤ 2, F ≤ 2

Product Performance

- **Modelo**: `gold_product_performance.sql`
- **Métricas a Calcular**: **Revenue y Margen**
    - **Revenue por Producto**: `SUM(quantity × unit_price)`
    - **Margen de Ganancia**: `((precio - costo) / precio) × 100`
    - **Profit Total**: `revenue - (quantity × unit_cost)`
- **Análisis de Inventario**
    - **Rotación**: `Ventas Anuales / Stock Promedio`
    - **Meses de Inventario**: `Stock Actual / (Ventas Mensuales Promedio)`
    - **Clasificación de Stock**: Critical (<10), Low (<50), Medium (<100), High (≥100)
- **Performance de Ventas**
    - **Categorización**: Best Seller (≥100), Good (≥50), Average (≥10), Slow (>0), No Sales (0)
    - **Rating Promedio**: Media de reseñas por producto

Sales Metrics

- **Modelo**: `gold_sales_metrics.sql`
- **Métricas Temporales**: Ventas por Período
    - Daily, Weekly, Monthly revenue
    - Year-over-year growth
    - Estacionalidad por trimestre
- **Análisis de Cohortes**
    - Retención por mes de registro
    - Valor por cohorte a lo largo del tiempo
- **Conversión y Performance**
    - AOV (Average Order Value) por período
    - Número de clientes únicos por período
    - Orders per customer

Operational Dashboard

- **Modelo**: `gold_operational_metrics.sql`
- **Métricas Operacionales**: Gestión de Inventario
    - Productos con stock crítico
    - Valor total de inventario
    - Productos sin movimiento (>90 días)
- **Análisis de Órdenes**
    - Tiempo promedio de procesamiento
    - Tasa de cancelación por categoría

    - Distribución de métodos de pago
- **Satisfaction Metrics**
    - Rating promedio por categoría
    - Productos con rating < 3.0
    - Tendencia de ratings por mes

📅 Fase 4: API Development con FastAPI

Objetivo: Crear API RESTful para servir los datos de la capa Gold

Setup FastAPI

- **Estructura del Proyecto**:

```
None
techstore_api/
├── app/
│   ├── __init__.py
│   ├── main.py
│   ├── models/
│   │   ├── __init__.py
│   │   └── schemas.py
│   ├── routers/
│   │   ├── __init__.py
│   │   ├── customers.py
│   │   ├── products.py
│   │   └── sales.py
│   └── database/
│       ├── __init__.py
│       └── connection.py
├── requirements.txt
└── pyproject.toml
```

Endpoints Implementation

Customer Analytics (`/api/customers/`):

- `GET /analytics` - Lista de customer analytics
- `GET /analytics/{customer_id}` - Analytics de cliente específico
- `GET /segments/{segment}` - Clientes por segmento
- `GET /top-customers` - Top clientes por CLV

Product Performance (`/api/products/`):

- `GET /performance` - Performance de todos los productos
- `GET /performance/{product_id}` - Performance de producto específico
- `GET /top-performers` - Top productos por revenue
- `GET /low-stock` - Productos con stock crítico

Sales Metrics (`/api/sales/`):

- `GET /metrics` - Métricas de ventas por período
- `GET /trends` - Tendencias de ventas
- `GET /summary` - Resumen ejecutivo

📅 Fase 5: Documentación

Objetivo: Finalizar el proyecto con documentación completa.

Documentación

- README.md Completo:
  - Descripción del proyecto
  - Arquitectura del sistema
  - Instrucciones de setup
  - Ejemplos de uso de API
- API Documentation: Swagger/OpenAPI automática con FastAPI
- DBT Documentation: `dbt docs generate && dbt docs serve`
- Data Lineage: Documentación del flujo de datos entre capas