Diseño e Implementación de Base de Datos Analítica - Día 5

Change Status

in-progress · 40 min

Learning Objectives

1. Comprender los principios del diseño de bases de datos analíticas
2. Aprender arquitecturas modernas para analytics (Lambda y Kappa)
3. Conocer estrategias de optimización para diferentes cargas de trabajo

Practical exercise to apply the concepts learned.

Ejercicio: Diseño conceptual de data warehouse para e-commerce

Paso 1: Identificar dimensiones clave

```
# Dimensiones para análisis de e-commerce
dimensiones_ecommerce = {
    'dim_customer': [
        'customer_id', 'email', 'registration_date',
        'customer_segment', 'total_orders', 'lifetime_value'
    ],
    'dim_product': [
        'product_id', 'sku', 'name', 'category',
        'brand', 'unit_cost', 'current_price'
    ],
    'dim_time': [
        'date_key', 'full_date', 'year', 'quarter',
        'month', 'day_of_week', 'is_weekend', 'is_holiday'
    ],
    'dim_location': [
        'location_id', 'country', 'region', 'city',
        'postal_code', 'timezone'
    ]
}

print("DIMENSIONES IDENTIFICADAS:")
for dim, atributos in dimensiones_ecommerce.items():
    print(f"• {dim}: {', '.join(atributos[:3])}...")
```

Paso 2: Definir tabla de hechos principal

```
-- Tabla de hechos para pedidos de e-commerce
CREATE TABLE fact_orders (
    order_id BIGINT PRIMARY KEY,
    customer_id INTEGER REFERENCES dim_customer(customer_id),
    product_id INTEGER REFERENCES dim_product(product_id),
    time_id INTEGER REFERENCES dim_time(date_key),
    location_id INTEGER REFERENCES dim_location(location_id),

    -- Métricas del pedido
    quantity_ordered INTEGER,
    unit_price DECIMAL(10,2),
    discount_amount DECIMAL(10,2),
    tax_amount DECIMAL(10,2),
    shipping_cost DECIMAL(10,2),
    total_amount DECIMAL(10,2),

    -- Métricas calculadas
    profit_margin DECIMAL(10,2),  -- (total_amount - cost) / total_amount
    is_first_purchase BOOLEAN,
    order_channel TEXT,  -- 'web', 'mobile', 'api'
    payment_method TEXT
);
```

Paso 3: Crear vistas analíticas optimizadas

```
-- Vista para análisis de productos populares
CREATE VIEW product_performance AS
SELECT
    dp.product_name,
    dp.category,
    dp.brand,
```

```
    SUM(fo.quantity_ordered) as total_units_sold,
    SUM(fo.total_amount) as total_revenue,
    AVG(fo.unit_price) as avg_selling_price,
    COUNT(DISTINCT fo.customer_id) as unique_customers,
    -- Ranking por categoría
    ROW_NUMBER() OVER (PARTITION BY dp.category ORDER BY SUM(fo.total_amount) DESC) as category_rank
FROM fact_orders fo
JOIN dim_product dp ON fo.product_id = dp.product_id
JOIN dim_time dt ON fo.time_id = dt.date_key
WHERE dt.year = 2024  -- Filtro temporal
GROUP BY dp.product_id, dp.product_name, dp.category, dp.brand;

-- Vista materializada para dashboards ejecutivos
CREATE MATERIALIZED VIEW executive_dashboard AS
SELECT
    dt.year,
    dt.month,
    SUM(fo.total_amount) as monthly_revenue,
    COUNT(DISTINCT fo.customer_id) as active_customers,
    COUNT(fo.order_id) as total_orders,
    AVG(fo.total_amount) as avg_order_value,
    -- Crecimiento mensual
    (SUM(fo.total_amount) - LAG(SUM(fo.total_amount)) OVER (ORDER BY dt.year, dt.month)) /
    LAG(SUM(fo.total_amount)) OVER (ORDER BY dt.year, dt.month) as growth_rate
FROM fact_orders fo
JOIN dim_time dt ON fo.time_id = dt.date_key
GROUP BY dt.year, dt.month
ORDER BY dt.year, dt.month;
```

Verificación: ¿Qué índices crearías para optimizar estas consultas? ¿Cómo manejarías el crecimiento de datos históricos en este warehouse?

Requerimientos:

PostgreSQL con soporte para vistas materializadas
Conocimiento básico de SQL DDL y consultas analíticas
Familiaridad con conceptos de modelado dimensional