Justificación de Diseño y Arquitectura de Datos

Proyecto: Retail Analytics Data Warehouse
Autor: Luis Arturo Sánchez
Rol: Ingeniero de Datos II / Líder Técnico

Introducción y Objetivos de Negocio

El objetivo de este proyecto fue diseñar e implementar una arquitectura de datos analítica capaz de responder a preguntas críticas de negocio para una cadena de retail. El sistema debe procesar transacciones históricas y estados de inventario para habilitar la toma de decisiones basada en datos.

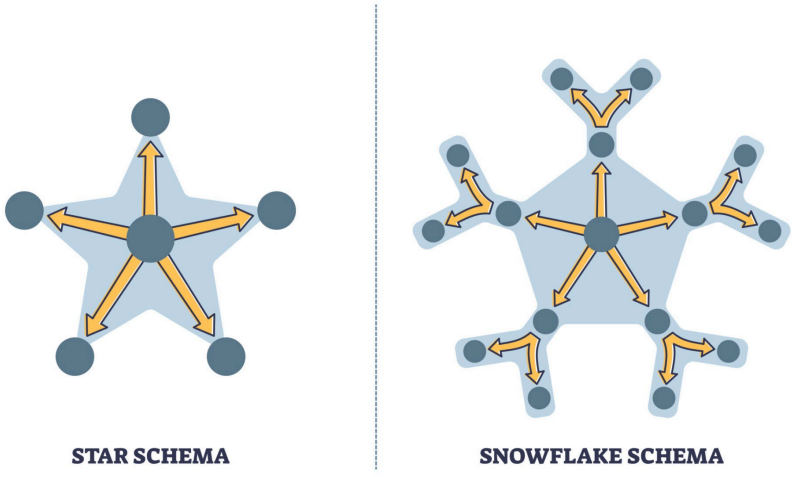
Los casos de uso principales que guiaron el diseño incluyen:

- Análisis de ventas agregadas por cadena, formato y temporalidad (Year-Over-Year).
- Identificación de productos "Best-Sellers" y comportamiento por categoría.
- Cálculo de métricas derivadas complejas como el "Ticket Promedio" mensual.

Arquitectura del Modelo de Datos

Selección del Modelo: Esquema Estrella (Star Schema)

El Esquema Estrella es una técnica de modelado de datos utilizada en Data Warehousing (almacenes de datos). Se llama así por su forma visual:



- En el centro hay una tabla grande y masiva (**Tabla de Hechos**).
- Alrededor, conectadas directamente como puntas de una estrella, están las tablas pequeñas (**Dimensiones**).

A diferencia de las bases de datos transaccionales (como la de una app de ventas) que buscan ahorrar espacio (**Normalización**), el Esquema Estrella busca velocidad de lectura (**Desnormalización**).

Tabla de Hechos (Fact Table)

Es la tabla central, registra eventos o acciones que ocurren en el negocio.

¿Qué contiene?

- **Claves Foráneas (FKs)**: Para conectarse con las dimensiones (tienda_sk, producto_sk, fecha_sk).
- **Métricas (Números)**: Datos que se pueden sumar, promediar o contar (monto_venta, cantidad_vendida, stock_actual).

Características: Es muy larga (millones de filas) pero estrecha (pocas columnas). Crece muy rápido.

Tablas de Dimensión (Dimension Tables)

Son las tablas satélite, describen el contexto de los hechos (Quién, Qué, Dónde, Cuándo).

¿Qué contiene? Texto descriptivo y atributos.

Características: Son tablas "anchas" (muchas columnas descriptivas) pero cortas (pocas filas comparadas con los hechos).

Entidades Definidas

- **Fact Tables (Hechos)**:
  - **fact_ventas**: Tabla transaccional. Granularidad a nivel de ítem de ticket. Contiene métricas aditivas (monto_total, cantidad).
  - **fact_inventario**: Tabla de snapshot periódico. Granularidad día-tienda-producto.

- **Dimension Tables (Dimensiones):**
  - **dim_tienda**: Contiene la jerarquía geográfica y operativa (Cadena > Formato > Tienda).
  - **dim_producto**: Contiene la jerarquía del catálogo (Categoría > Marca > Producto).
  - **dim_fecha**: Dimensión calendario para facilitar la navegación temporal y cálculos de estacionalidad (días festivos, fines de semana).

Diferencias: Estrella vs. Copo de Nieve (Snowflake)

- **Estrella (Star):** Las dimensiones están desnormalizadas.
  - *Ejemplo:* En `dim_producto` repites la palabra "Samsung" en cada fila de un celular Samsung.
  - *Ventaja:* Rendimiento (menos JOINs).

    - *Desventaja:* Ocupa un poco más de espacio en disco (hoy en día, el espacio es barato, así que no importa).

- **Copo de Nieve (Snowflake):** Las dimensiones se normalizan.
    - *Ejemplo:* Tienes `dim_producto` conectada a una tabla `dim_marca`.
    - *Ventaja:* Ahorra espacio, evita redundancia de datos.
    - *Desventaja:* Muchos JOINs, consultas lentas.

Justificación:

- **Rendimiento en Lectura (OLAP)**: El esquema estrella reduce la complejidad de los JOINs necesarios para consultas analíticas. Al tener una tabla de hechos central rodeada de dimensiones denormalizadas, el motor de base de datos puede optimizar los planes de ejecución eficientemente.

- **Facilidad de uso**: Este modelo es intuitivo para los analistas de negocio y herramientas de BI (como Power BI o Tableau), separando claramente las métricas (Hechos) de los atributos descriptivos (Dimensiones).

- **Extensibilidad**: Permite agregar nuevas dimensiones o métricas sin alterar la lógica de las consultas existentes.

Decisiones Técnicas y Gobernanza

Integridad y Calidad de Datos

Para garantizar la robustez del Data Warehouse, se implementaron las siguientes reglas:

- **Claves subrogadas (Surrogate Keys _sk)**: Se utilizaron claves numéricas o hashes internos como Primary Keys en lugar de las claves naturales del sistema fuente (_id). Esto aísla al Data Warehouse de cambios en los sistemas operativos y mejora el rendimiento de los índices (enteros vs strings).

- **Tipado estricto**: Se utilizó DECIMAL(10,2) para montos monetarios en lugar de FLOAT para evitar errores de redondeo financiero.

- **Auditoría**: Se agregaron columnas de metadatos (fecha_carga, proceso_origen) en las tablas de hechos para trazabilidad (Data Lineage).

Plataforma: AWS RDS PostgreSQL

Se seleccionó PostgreSQL sobre AWS RDS (Free Tier db.t4.micro) debido a:

- **Capacidad analítica**: Soporte nativo y eficiente para Window Functions y consultas complejas requeridas por el negocio.

- **Gestión**: Al ser un servicio administrado, AWS maneja los backups, parches y disponibilidad, reduciendo la carga operativa.

Estrategia de Rendimiento y Optimización

Dado el volumen de datos transaccionales, se implementó una estrategia de indexación agresiva enfocada en lectura, aceptando un ligero trade-off en la velocidad de escritura/carga.
A continuación, se detalla la optimización realizada sobre tres consultas críticas de negocio:

Caso 1: Ventas Totales por Cadena y Año

- **Desafío**: Esta consulta requiere agregaciones pesadas (SUM, COUNT) agrupando datos de la tabla de hechos (fact_ventas) cruzada con dos dimensiones (dim_tienda, dim_fecha).

- **Cuello de botella**: El motor realizaba un Sequential Scan (lectura completa) de la tabla de ventas para realizar los JOINs.

- **Optimización**: Se crearon índices B-Tree en las claves foráneas:
    - **idx_ventas_tienda** en fact_ventas(**tienda_sk**)
    - **idx_ventas_fecha** en fact_ventas(**fecha_sk**)

- **Resultado**: El plan de ejecución cambió a utilizar Bitmap Heap Scans, reduciendo drásticamente el costo computacional al acceder directamente a los bloques de datos relevantes.

Caso 2: Top 5 Productos más Vendidos

- **Desafío**: Requiere agrupar por producto, sumar cantidades y ordenar (ORDER BY DESC) para aplicar un LIMIT.

- **Cuello de botella**: El ordenamiento (Sort Method) consumía memoria y CPU excesiva al tener que procesar el dataset completo antes de filtrar los top 5.

- **Optimización**: Se creó un índice en la referencia de producto:
    - **idx_ventas_producto** en fact_ventas(**producto_sk**)

- **Resultado**: Aceleración significativa del GROUP BY, permitiendo al motor agregar las ventas por producto de manera eficiente antes de la etapa de ordenamiento.

Caso 3: Ticket Promedio (Count Distinct)

- **Desafío**: Calcular el ticket promedio requiere contar tickets únicos (COUNT(DISTINCT ticket_id)). Esta es una operación costosa ya que el motor debe verificar la unicidad fila por fila en una columna de alta cardinalidad.

- **Optimización**: Se creó un índice específico sobre la columna transaccional:

    - **idx_ventas_ticket** en fact_ventas(**ticket_id**)

- **Resultado**: Habilitó un Index Only Scan (o Index Scan eficiente), donde el motor cuenta las entradas en el árbol del índice sin necesidad de visitar la tabla principal (Heap) para cada verificación, mejorando los tiempos de respuesta.

Escalabilidad y Futuros Pasos

Aunque la arquitectura actual cumple con los requisitos del desafío, se proponen las siguientes mejoras para un escenario de producción con crecimiento exponencial de datos:

- **Particionamiento de tablas**: Implementar particionamiento nativo de PostgreSQL en fact_ventas por rango de fechas (ej. particiones mensuales). Esto permitiría "podar" particiones (Partition Pruning) en consultas que solo analizan el mes actual, ignorando datos históricos.

- **Views**: Para reportes ejecutivos recurrentes (como el "Ventas del Año a la Fecha"), se pueden crear vistas materializadas que pre-calculen las agregaciones y se refresquen periódicamente, eliminando el costo de cómputo en tiempo real.

- **Arquitectura Data Lake**: Mover la ingesta de datos crudos hacia Amazon S3 y utilizar AWS Glue para el procesamiento ETL pesado, dejando a RDS PostgreSQL únicamente como la capa de servicio (Serving Layer) para herramientas de BI.