Clase 215 — Modelado dimensional: star/snowflake schemas

Parte: 5 — Ingeniería de Datos · Fuente: Kimball & Ross The Data Warehouse Toolkit (Wiley, 3ª ed.) + Reis & Housley cap. 8. Duración estimada: 75 min.

Clase 215 — Modelado dimensional: star/snowflake schemas

> *Parte: 5 — Ingeniería de Datos · Fuente: Kimball & Ross The Data Warehouse Toolkit (Wiley, 3ª ed.) + Reis & Housley cap. 8. Duración estimada: 75 min.*

Objetivo

Diseñar el esquema de un data warehouse usando modelado dimensional (Kimball): una fact table central + dimension tables alrededor (star schema), o dimensiones normalizadas (snowflake schema). Es el modelo que han usado los data warehouses serios desde los 90s y sigue vigente en BigQuery/Snowflake/dbt en 2026.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Identificar fact tables (eventos medibles: sale, click, payment) vs dimension tables (entidades descriptivas: customer, product, date).
- Diseñar un star schema con fact + dimensions desnormalizadas.
- Manejar slowly changing dimensions (SCD): Tipo 1 (overwrite), Tipo 2 (history con valid_from/valid_to), Tipo 3 (current + previous).
- Crear una date dimension completa (con feriados, fiscal year, etc.) — la dimensión más reutilizada.
- Diferenciar OLAP (modelado dimensional, query analíticas) de 3NF (modelado normalizado, OLTP).

Temas

#	Tema	Por qué importa
1	Star schema: fact + dims	El patrón fundamental.
2	Snowflake schema: dims normalizadas	Cuándo agregar el normalizado.
3	Surrogate keys vs natural keys	Por qué usar IDs autoincrement en DW.
4	Slowly Changing Dimensions (SCD 1/2/3)	Cómo manejar history.
5	Date dimension	La que casi siempre olvidás generar.
6	Granularidad de la fact	"Una fila por click" vs "una fila por sesi

Definiciones y características

- Fact table: tabla con métricas numéricas + foreign keys a dimensions. Ej: fact_sales(date_key, product_key, customer_key, store_key, qty, revenue). Filas chicas, MUCHAS (millones-billones).
- Dimension table: descripción de las entidades. Ej: dim_product(product_key, sku, name, category, brand). Filas grandes (muchas columnas), POCAS (miles-millones).
- Star schema: 1 fact + N dims, todas conectadas directo a la fact. Cada dim es plana (denormalizada). El default — mejor performance, joins simples.
- Snowflake schema: las dims están normalizadas (dim_product → dim_category → dim_segment). Ahorra storage en dims gigantes; complica queries.
- Surrogate key: ID autoincrement único para la dimensión (product_key=42), distinto del natural key del

sistema source (sku="ABC-123"). Permite SCD Tipo 2 sin colisión.
- SCD Tipo 1 (overwrite): cambia el valor in-place. No mantiene historia. Útil para correcciones tipo "fix typo en nombre".
- SCD Tipo 2 (history): inserta nueva fila con valid_from, valid_to, is_current. La fact apunta al product_key vigente al momento del evento.
- SCD Tipo 3 (current + previous): columnas current_value + previous_value. Maneja solo 1 cambio histórico. Raro en práctica.
- Date dimension: tabla precalculada con 1 row por día (date_key, year, quarter, month, day_of_week, is_weekend, is_holiday, fiscal_year, ...). Evita derivar en cada query.
- Granularidad (grain): el nivel de detalle de la fact. "1 row per transaction" vs "1 row per day per customer". Definir el grain ES la decisión de diseño más importante.

Dataset / recursos

- Caso ejemplo: e-commerce con tablas operacionales orders, order_items, products, customers. Transformar a star schema.
- Librerías: DuckDB/SQL puro (las tablas son SQL, no Python).

Ejercicios

1. Identificar grain: dado un dataset de pedidos de e-commerce, decidí el grain de tu fact. ¿"1 row per order"? ¿"1 row per order line"? Elegí, justificá.
2. Date dimension: SQL que genera dim_date con 5 años de días, columnas year, quarter, month, day_of_week_iso, is_weekend, is_holiday_us, fiscal_year. (generate_series de Postgres/DuckDB).
3. Star schema: del e-commerce, diseñá fact_sales(date_key, product_key, customer_key, store_key, qty, revenue, discount), dim_product, dim_customer, dim_store, dim_date. SQL completo.
4. SCD Tipo 2 en dim_customer: cliente cambia de ciudad. Tu pipeline detecta el cambio → UPDATE la fila vigente con valid_to=NOW(), is_current=FALSE → INSERT nueva fila con valid_from=NOW(), is_current=TRUE. Toda venta del cliente queda asociada a su ciudad al momento de la compra.
5. Query típica: "Revenue por brand × month × is_weekend" — escribíla con JOINs entre fact + dims + dim_date. Comparala con la equivalente en tablas no-modeladas (más JOINs, más subqueries).

Homework verificable

Repo con:

1. SQL completo para crear: dim_date, dim_customer (SCD 2), dim_product, dim_store, fact_sales.
2. Script de carga: lee tablas operacionales (Postgres/CSV) → transforma → carga al DW (DuckDB local).
3. Demonstración de SCD 2: un cliente que cambia de ciudad; venta antes y después del cambio quedan correctamente asociadas a la ciudad de ese momento.
4. 5 queries analíticas representativas: revenue por segment × quarter, top 10 products, cohort retention, etc.
5. README explicando: grain elegido, decisiones de denormalización, qué dim usaría snowflake (si alguna).

Criterio de aceptación: las 5 queries corren limpio; SCD 2 está correctamente implementada (verificable con un test que valida revenue agregado por ciudad-en-momento).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Star schema con grain mixto (algunas filas	Confusión sobre el grain. Fix: definir UNA
Customer cambia de ciudad y todas las vent	Usaste SCD Tipo 1. Fix: SCD Tipo 2 con sur
Date queries lentas: WHERE EXTRACT(month F	Index no se usa con función. Fix: usar dim
Fact table tiene 200 columnas	Estás metiendo dimensiones como columnas.
Snowflake schema con 10 niveles de jerarqu	Over-engineering. Fix: empezar con star (p
dbt jobs explotan: full refresh diario de	Falta incremental loading. Fix: dbt increm

Preguntas frecuentes

¿Modelado dimensional sigue vigente con BigQuery/Snowflake?

Sí. Aunque el compute moderno es elástico, queries siguen siendo más rápidas y baratas con star schema. dbt promueve el patrón. La diferencia con los 90s: no necesitás crear índices manualmente (los DW modernos lo manejan).

¿Star o snowflake?

Star por default. Snowflake solo si tu dim tiene jerarquía profunda + millones de rows + repetición de strings que duele en storage. Para dim_product con category repetido 10K veces: snowflake (sacar dim_category). Para customer.country repetido 1M veces: dejar en star (no vale la pena el JOIN extra).

¿Qué grain elijo?

El más fino posible que tenga sentido. "1 row per transaction line" mejor que "1 row per order" — siempre podés agregar al sumar, no podés desagregar después. Excepción: si la fact crece en TBs por día, agregar a daily_sales_by_product puede ser necesario.

¿dim_date o calcular en query?

dim_date siempre. Razones: (1) feriados, fiscal year, business days requieren lookup, no aritmética. (2) Performance: JOIN a dim_date es más rápido que EXTRACT/CASE. (3) Consistencia: un único lugar define "qué es weekend".

¿Data Vault, OBT (One Big Table), Activity Schema, Anchor model — y modelos alternativos?

- Data Vault: para enterprise con many sources, schema cambiando seguido. Más complejo.
- OBT: una tabla con todo desnormalizado (extremo de star). Bueno para una sola dashboard; explota con más casos.
- Activity Schema: 1 fact universal de "actividades" + 1 dim por entidad. Patrón moderno (Narrator).
- Anchor model: 6NF extremo, raro fuera de academia.

Para 90% de los casos: star schema sigue ganando.

¿dbt es necesario?

No estrictamente, pero es el estándar. dbt convierte SQL transforms en código (versionado, testeado, lineage). Si vas a hacer >5 modelos: usá dbt.

Referencias

- Kimball, R. & Ross, M. The Data Warehouse Toolkit (Wiley, 3ª ed., 2013) — la biblia del modelado dimensional.
- Reis & Housley Fundamentals of Data Engineering (O'Reilly, 2022) cap. 8.
- dbt docs — el ecosistema moderno.
- The Open Source Data Stack Conference — talks de Kimball moderno con DuckDB/dbt.
- dim_date SQL generator — dbt package listo para usar.

Siguiente clase

Clase 216 — Recommender systems: visión general

> *Fin de la Parte 5 — Ingeniería de Datos. Tenés el stack completo para mover/procesar/almacenar datos a escala: orquestación (208-209), procesamiento (210-211), warehouses (212), streaming (213), formatos (214), modelado (215). Parte 6 cierra con recommender systems.*

Apéndice: notebook (primer bloque)

Construimos un star schema de e-commerce: dim_date, dim_customer (SCD 2), dim_product, dim_store, fact_sales. Demostramos SCD 2 con cliente que cambia de ciudad.

```
import duckdb, tempfile
from pathlib import Path
DB = str(Path(tempfile.gettempdir()) / 'dw_star.duckdb')
Path(DB).unlink(missing_ok=True)
con = duckdb.connect(DB)
```

Archivos complementarios

- notebook.ipynb