TEC | Tecnológico de Costa Rica

Proyecto 2 — Base de Datos II

Plataforma OLAP y Visualización para el Sistema de Restaurantes

Autores

Sebastián Chacón
Pablo Agüero

Profesor

Kenneth Obando Rodríguez

Escuela de Ingeniería en Computación
Instituto Tecnológico de Costa Rica — Cartago

Junio, 2025

Introducción

Este trabajo documenta la construcción de un proyecto OLAP para el dominio de reservas y pedidos en restaurantes. Los datos entregados como archivos *CSV* se depuran y transforman con Apache Spark, para luego cargarse en un *data warehouse* Hive modelado con un esquema estrella. Sobre ese almacén se generan vista materializadas y cubos que hacen que la exploración de métricas clave, las cuales se pueden visualizar a través de Apache Superset. Toda la infraestructura se encapsula en contenedores Docker coordinados por `docker-compose`, garantizando portabilidad del proyecto.

1 Arquitectura general y flujo de datos

1. **Ingesta & staging**
   Archivos `CSV` con histórico de clientes, productos, fechas, pedidos y reservas se copian al contenedor *Spark Master*. También se cargan archivos ya limpios directamente al hive.
2. **Procesamiento con Spark**
   Spark realiza limpieza, tipificación y agregados ligeros, la salida se deposita en `/tmp` del contenedor *hive-server* o se puede manipular para otros motivos.
3. **Carga en Hive (ELT)**
   Vía `LOAD DATA LOCAL` los ficheros se importan en tablas `TEXTFILE` (dimensiones) y `ORC` (hechos).
4. **Consumo analítico**
   Superset se conecta a Hive (`hive://hive-hive-server-1:10000/restaurante_olap`) hive permite crear cubos, superset *datasets* y dashboards interactivos sin mover los datos necesariamente.

2 Modelado dimensional: esquema estrella

Tablas de Dimensión

Seis dimensiones ofrecen los ejes de análisis:

- **dim_cliente** Perfil nominal del consumidor.
- **dim_restaurante** Metadatos del local y administrador.
- **dim_producto** Catálogo y precios históricos.
- **dim_fecha** Jerarquía día–mes–año–semana.
- **dim_estado_pedido**, **dim_estado_reserva**: dominios de estado.

Tablas de Hechos

- **fact_pedido**: granularidad "línea de pedido". Métricas: `cantidad`, `monto_total`.
- **fact_reserva**: granularidad "reserva efectuada". Métrica: `numero_personas`.

Las claves sustitutas (`BIGINT/INT`) y las claves de negocio se alinean con el modelo estrella mencionado previamente. Las dependencias quedan reflejadas en la Figura 1.

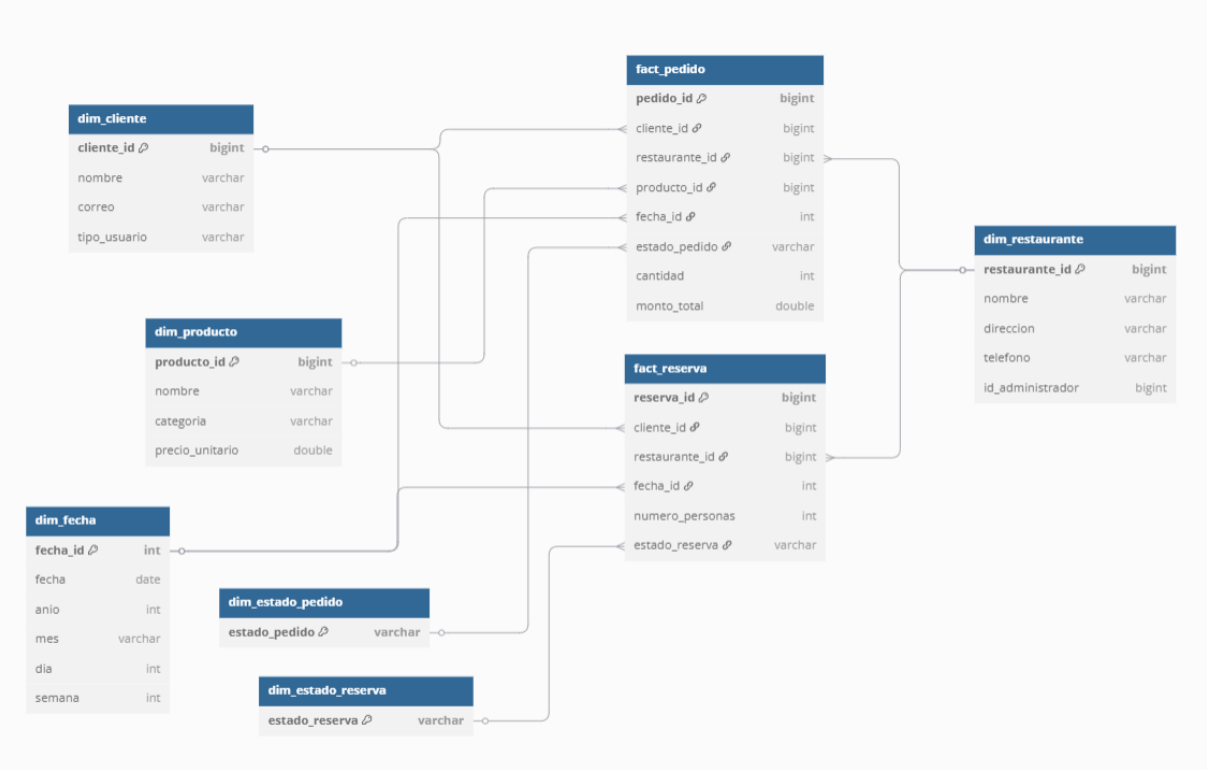

Figura 1: Esquema estrella del *data warehouse*.

Vistas OLAP (cubos)

Ejemplos creados para Superset:

- `vw_ingresos_mes_categoria` (ingresos mensual × categoría).
- `vw_top_clientes` (TOP-10 clientes por gasto).
- `vw_tasa_cancelacion` (ratio cancelaciones).
- `vw_reservas_por_dia`
- `vw_ventas_restaurante`

3 Procesamiento con Apache Spark

- Imagen `bitnami/spark:3.3.0`, modo *master*.

- Se monta `hive-site.xml` para reutilizar el metastore `thrift://hive-metastore:9083`.
- **E**jecución típica:

```
docker exec -it hive-spark-1 spark-shell --master local[*] \
  --conf spark.sql.catalogImplementation=hive
```

Listing 1: Fragmento Scala de limpieza.

```
val raw = spark.read.option("header",true)
                    .csv("/data/pedidos_raw.csv")

val pedidos = raw
  .withColumn("monto_total", col("cantidad") * col("precio_
     unitario"))
  .filter(col("estado_pedido").isNotNull)

pedidos.write.mode("overwrite")
       .option("header", "false")
       .csv("/tmp/fact_pedido.csv")
```

Los jobs se pueden programar con Cron o Airflow, se supone que para este trabajo era con Airflow pero no puso ser implementado.

4 Data Warehouse en Apache Hive

- Versión `2.3.2-postgresql-metastore`.
- `hive-server-1` expone HiveServer2 en el puerto `10000`.
- Tablas de hechos en formato `ORC` para compresión y predicate push-down.

Script de inicialización

El fichero `restaurante_olap_init.sql` automatiza la creación de BD, tablas y carga de datos.

5 Visualización con Apache Superset

Conexión

URI: `hive://hive-hive-server-1:10000/restaurante_olap`

La conexión se registró manualmente desde la consola `superset shell`:

```
from superset import db
from superset.models.core import Database

new_db = Database(
    database_name="HiveTest",
```

```
    sqlalchemy_uri="hive://hive-hive-server-1:10000/restaurante_olap",
    expose_in_sqllab=True,
    allow_run_async=True,
)
db.session.add(new_db)
db.session.commit()
```

Dashboards planteados para revisión

1. **Ingresos mensuales × categoría**
   Gráfico de área apilada con `vw_ingresos_mes_categoria`.
2. **Top-10 clientes por gasto**
   Tabla dinámica + % acumulado.
3. **Pedidos completados vs cancelados**
   KPI y donut en un único panel.

6 Decisiones de diseño

- Se prefirió **Hive + Superset** en lugar de Spark Thrift Server para simplificar la pila y minimizar puertos abiertos. Se hizo más complicado usar una instancia tan diferente de Spark como Thrift.
- **Formato ORC** en hechos para lecturas analíticas rápidas.
- Dimensiones en `TEXTFILE` por su baja cardinalidad y ingesta directa desde fichero plano.
- **Docker-compose monolítico** para no complicar el despliegue académico; se habilita volumen nombrado en `superset_home` para persistir dashboards.

7 Ejemplos de uso

1. **Planeación de horarios en el curso de Gestión de Operaciones**
   Para la práctica "Balanceo de carga laboral", el docente descarga desde Superset la tabla "*horas_pico_pedidos*" filtrada al mes de abril. Con los picos de 12:00 – 14:00 h y 19:00 – 21:00 h ya identificados, los estudiantes re-asignan turnos de meseros y comparan el costo de personal antes y después de la optimización. El cálculo de la métrica "se enlaza directamente a la vista materializada `mv_horarios_pico`, de modo que las simulaciones se actualizan en segundos.
2. **Seguimiento semanal en Práctica Profesional**
   El estudiante en práctica, asignado al área de *Business Intelligence* de un restaurante aliado, abre el tablero "*Indicadores de servicio*" cada lunes a las 09:00. Allí revisa:
   - la *tasa de cancelación de pedidos* calculada sobre `mv_estado_pedido`;
   - el *mapa de calor de reservas* derivado de `mv_actividad_geográfica`;

- y la tendencia de *clientes recurrentes* obtenida de `mv_frecuencia_uso`.

Con estos tres gráficos genera un `PDF` con el botón "Schedule Report" y lo envía automáticamente al tutor académico como evidencia de avance.

8 Neo4J y enrutamiento

Para complementar, se integró Neo4j como motor de grafos a partir de un conjunto de datos generado utilizando herramientas como Google Colab. Esta además fue enriquecida con información geoespacial para la optimización de rutas de reparto

8.1 Análisis de grafos en Neo4J

Modelo del grafo

- **Cliente** Nodo que representa a cada usuario del sistema.
  - Propiedades: `cliente_id`, `nombre`, `email`
  - Relaciones:
    - `(:Cliente) -[:ORDENA]->(:Pedido)`
- **Pedido** Nodo intermedio que agrupa un pedido realizado por un cliente.
  - Propiedades: `pedido_id`, `fecha`
  - Relaciones:
    - `(:Pedido) -[:INCLUYE]->(:Producto)`
    - `(:Pedido) -[:EN_RESTAURANTE]->(:Restaurante)`
- **Producto** Nodo que representa cada plato o ítem del menú.
  - Propiedades: `producto_id`, `nombre_plato`, `categoria`, `precio`
  - Relaciones:
    - `(:Restaurante) -[:OFRECE]->(:Producto)`
- **Restaurante** Nodo que representa el local o establecimiento.
  - Propiedades: `restaurante_id`, `nombre_restaurante`, `ciudad`
  - Relaciones:
    - `(:Restaurante) -[:OFRECE]->(:Producto)`
    - `(:Pedido) -[:EN_RESTAURANTE]->(:Restaurante)`

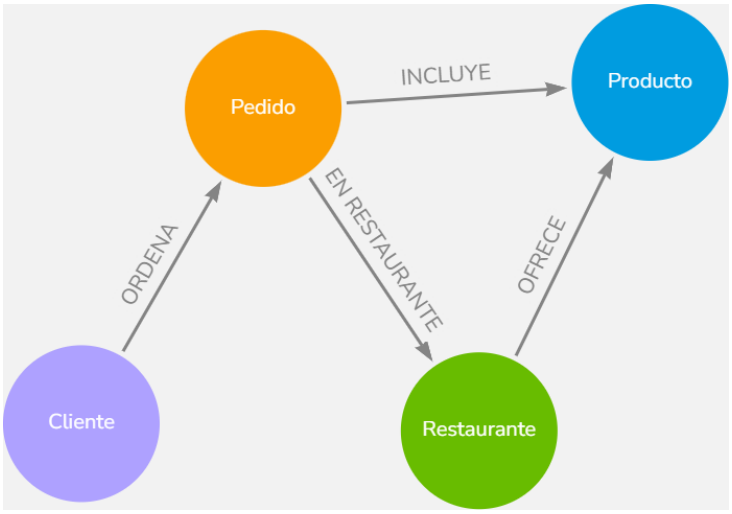

Figura 2: Esquema del Grafo

8.2 Optimización de rutas de entrega

9 Conclusiones

Logramos generar un sistema OLAP reproducible con tecnologías open-source. Hive actúa como columna vertebral, Spark como motor de pre-procesamiento y Superset como interfaz para analistas. Además el sistema trabajado con Neo4J nos permite ver otras posibilidades en las bases de datos y en usos tan interesantes como los mapas.