BASES DE DATOS II

DATA ENVIRONMENT

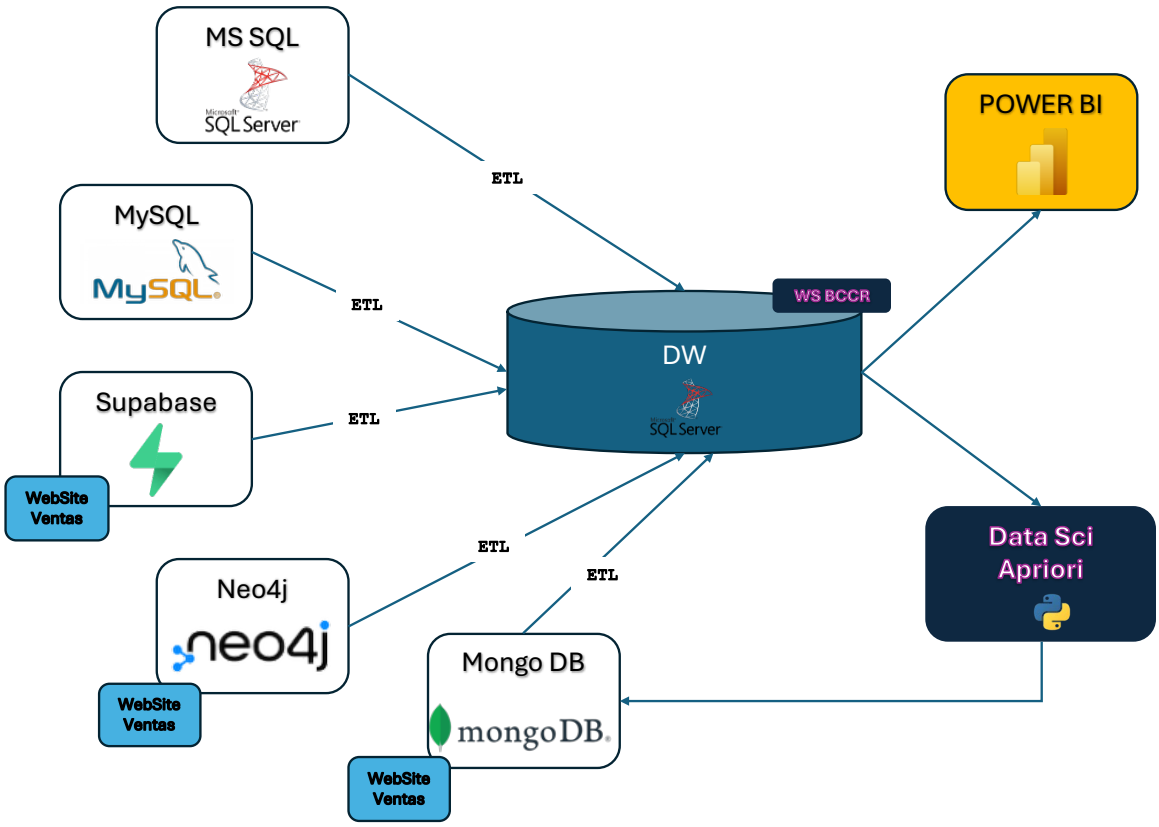
Contexto

La empresa dispone de múltiples fuentes transaccionales de ventas en diferentes motores de bases de datos. Cada una almacena información de clientes, productos, órdenes y detalles de órdenes, pero con diferencias de formatos, claves, monedas y catálogos.

El objetivo del proyecto es que los estudiantes diseñen un proceso ETL que integre todos los datos en un Data Warehouse central en SQL Server, aplicando un esquema estrella o copo de nieve, y que posteriormente desarrollen un dashboard de ventas en Power BI y un análisis de reglas de asociación con Apriori en Python.

Objetivos

- Diseñar procesos ETL para integrar información heterogénea.
- Resolver problemas de calidad de datos: monedas, formatos de fechas, duplicados, codificación de productos y género.
- Implementar un Data Warehouse en SQL Server.
- Crear un dashboard de análisis de ventas en Power BI.
- Aplicar el algoritmo Apriori en Python para descubrir asociaciones entre productos.



Modelos Transaccionales

1. SQL Server

Ventas en USD; género como 'Masculino/Femenino'.

Tablas: Cliente, Producto, Orden, OrdenDetalle.

Diagrama: Cliente (1)——(N) Orden (1)——(N) OrdenDetalle (N)——(1) Producto

2. MySQL

Códigos alternos, montos como string, fechas en texto.

Diagrama: Cliente (1)——(N) Orden (1)——(N) OrdenDetalle (N)——(1) Producto

3. Supabase/PostgreSQL

UUIDs, algunos productos sin SKU que serían servicios, solo con descripción en la línea de ventas.

Diagrama: Cliente (1)——(N) Orden (1)——(N) OrdenDetalle (N)——(1) Producto si no es servicio

4. MongoDB

Documentos, totales en CRC enteros.

Diagrama lógico: Cliente — Orden — items[] — Producto

5. Neo4j

Grafo de compras con nodos Cliente, Producto, Categoría y Orden.

Relaciones: (Cliente)-[:REALIZO]->(Orden), (Orden)-[:CONTIENTE]->(Producto), (Producto)-[:PERTENECE_A]->(Categoria)

Diagrama simplificado: Cliente → Orden → Producto → Categoria

Reglas de Integración (ETL)

- Homologación de productos: construir tabla puente para SKU ↔ codigo_alt ↔ codigo_mongo.
- Normalización de moneda: convertir CRC a USD con tabla de tipo de cambio. Esta table se debe llenar con un WebService del tipo de cambio al BCCR. Debe buscar los TC de 3 años para atrás y además, cada día a las 5 a.m. y programable, un Job actualizará el TC.
- Estandarización de género: M/F, Masculino/Femenino, Otro/X → valores únicos.
- Conversión de fechas: castear VARCHAR a DATE/DATETIME.
- Transformación de totales: montos string → decimal; montos enteros (CRC) → decimal.

Data Warehouse (MS SQL Server)

Tablas de Hechos y Dimensiones: DimCliente, DimProducto, DimTiempo, FactVentas.

Modelo Estrella: DimCliente – FactVentas – DimTiempo – DimProducto

Associations

Debe implementar el algoritmo Apriori en Python o R y almacenar los resultados, automáticamente, en una base de datos que pueda ser consumida en las Web de creación de ventas para poder ver recomendaciones de productos que se compran juntos y agregarlos a la venta.

Power BI

El Dashboard debe permitir analizar el desempeño de ventas de la empresa en distintos niveles de detalle: por cliente, producto, categoría, canal y período de tiempo. Además, debe integrar una tabla de **metas de ventas mensuales** que se comparará contra los valores reales del **FactVentas**, para medir el cumplimiento y detectar desviaciones.

El dashboard debe incluir filtros dinámicos por:

- Rango de fechas
- Cliente
- Producto
- Categoría
- Canal de ventas

Y debe poder compararse **con años anteriores** para identificar tendencias.

Indicadores Clave (KPIs)

1. **Ventas Totales (USD)**
    - Ventas acumuladas en el período seleccionado.
2. **Crecimiento vs Año Anterior (%)**
    - Comparación entre ventas del período actual vs mismo período del año anterior.

3. **Top 10 Clientes por Ventas**
    - Ranking con los clientes de mayor facturación.
4. **Top 10 Productos por Ventas**
    - Ranking con los productos más vendidos.
5. **Ventas por Categoría**
    - Distribución de ventas según la categoría del producto.
6. **Cumplimiento de Metas (%)**
    - (Ventas Reales / Meta) * 100, por cliente, producto y mes.
7. **Ticket Promedio por Orden**
    - Ventas Totales / Número de Órdenes.
8. **Número de Clientes Activos**
    - Cantidad de clientes con al menos una compra en el período.
9. **Ventas por Canal de Venta**
    - Comparación de ventas en ONLINE, RETAIL, PARTNER.
10. **Margen de Crecimiento Acumulado**
    - Variación porcentual mes a mes, acumulada durante el año.

Gráficos

1. **Serie Temporal de Ventas (línea)**
    - Ventas mensuales por año (comparando actual vs año anterior).
2. **Ventas vs Metas (barras agrupadas)**
    - Ventas reales vs metas, por cliente-producto-mes.
3. **Cumplimiento de Metas (heatmap o matriz)**
    - Cliente vs producto, con colores de cumplimiento:
        - Verde (>100%),
        - Amarillo (80-100%),
        - Rojo (<80%).
4. **Top Clientes y Productos (barras horizontales)**
    - Rankings de ventas acumuladas.

5. **Distribución de Ventas por Categoría (gráfico circular o donut)**
    - Proporción de ventas por cada categoría de producto.
6. **Ventas por Canal (columnas apiladas)**
    - Comparar canales de venta en el período seleccionado.

Nueva tabla: MetasVentas

En el **Data Warehouse (MS SQL Server)** se debe crear una tabla para almacenar las metas mensuales de ventas, a nivel de cliente y producto:

```
CREATE TABLE MetasVentas (

    MetaID INT IDENTITY PRIMARY KEY,

    ClienteID INT NOT NULL FOREIGN KEY REFERENCES DimCliente(ClienteID),

    ProductoID INT NOT NULL FOREIGN KEY REFERENCES DimProducto(ProductoID),

    Anio INT NOT NULL,

    Mes INT NOT NULL,

    MetaUSD DECIMAL(18,2) NOT NULL

);
```

ClienteID, ProductoID: Llaves que conectan con las dimensiones.

Anio, Mes: Permiten filtrar por período.

MetaUSD: Monto de la meta en dólares.

Entregables

- Script de creación del Data Warehouse en SQL Server.
- Procesos ETL (SSIS, Python/pandas, o similar).
- Dashboard en Power BI con métricas de ventas.
- Análisis de reglas de asociación (Apriori en Python) que se utilicen en las Web pages de Mongo y Supabse para realizar recomendaciones de compras.
- Web que permita CRUD de crear venta en Mongo
- Web que permita CRUD de crear venta en Supabase
- Web que pemita CRUD de crear venta en NEO4J
- Web que permita hacer carga masiva de ventas en MS SQL y MySQL (LOADER)

Entrega: Jueves de semana 16 con citas de revisión.

ANEXOS

Modelos de datos propuestos, los puede variar siempre que cumple con las reglas:

1) MS SQL Server (ventas en USD; género como 'Masculino/Femenino')

```
-- Esquema: sales_ms
CREATE SCHEMA sales_ms;
GO

CREATE TABLE sales_ms.Cliente (
    ClienteId INT IDENTITY PRIMARY KEY,
    Nombre NVARCHAR(120) NOT NULL,
    Email NVARCHAR(150) UNIQUE,
    Genero NVARCHAR(12) CHECK (Genero IN ('Masculino','Femenino')),
    Pais NVARCHAR(60) NOT NULL,
    FechaRegistro DATE NOT NULL DEFAULT (GETDATE())
);

CREATE TABLE sales_ms.Producto (
```

```
    ProductoId INT IDENTITY PRIMARY KEY,
    SKU NVARCHAR(40) UNIQUE NOT NULL,  -- SKU “oficial”
    Nombre NVARCHAR(150) NOT NULL,
    Categoria NVARCHAR(80) NOT NULL
);

CREATE TABLE sales_ms.Orden (
    OrdenId INT IDENTITY PRIMARY KEY,
    ClienteId INT NOT NULL FOREIGN KEY REFERENCES sales_ms.Cliente(ClienteId),
    Fecha DATETIME2 NOT NULL DEFAULT (SYSDATETIME()),
    Canal NVARCHAR(20) NOT NULL CHECK (Canal IN ('WEB','TIENDA','APP')),
    Moneda CHAR(3) NOT NULL DEFAULT 'USD',                -- homogénea en esta
fuente
    Total DECIMAL(18,2) NOT NULL
);

CREATE TABLE sales_ms.OrdenDetalle (
    OrdenDetalleId INT IDENTITY PRIMARY KEY,
    OrdenId INT NOT NULL FOREIGN KEY REFERENCES sales_ms.Orden(OrdenId),
    ProductoId INT NOT NULL FOREIGN KEY REFERENCES
sales_ms.Producto(ProductoId),
    Cantidad INT NOT NULL CHECK (Cantidad > 0),
    PrecioUnit DECIMAL(18,2) NOT NULL,                    -- en USD
    DescuentoPct DECIMAL(5,2) NULL
);

-- Índices sugeridos
CREATE INDEX IX_Orden_Fecha ON sales_ms.Orden(Fecha);
CREATE INDEX IX_Detalle_Prod ON sales_ms.OrdenDetalle(ProductoId);
```

Heterogeneidades planeadas

- Moneda siempre “USD”.
- Género con texto completo.
- SKU “oficial” que deberán mapear con códigos alternos en otras fuentes.

2) MySQL (códigos de producto alternos; algunos precios en string; fechas como VARCHAR)

```
-- database: sales_mysql

CREATE TABLE Cliente (
  id INT AUTO_INCREMENT PRIMARY KEY,
  nombre VARCHAR(120) NOT NULL,
  correo VARCHAR(150),
  genero ENUM('M','F','X') DEFAULT 'M',      -- distinto a SQL Server
  pais VARCHAR(60) NOT NULL,
  created_at VARCHAR(10) NOT NULL            -- 'YYYY-MM-DD' (forzará
parsing en ETL)
);

CREATE TABLE Producto (
  id INT AUTO_INCREMENT PRIMARY KEY,
  codigo_alt VARCHAR(64) UNIQUE NOT NULL,    -- código alterno (no coincide
con SKU)
  nombre VARCHAR(150) NOT NULL,
  categoria VARCHAR(80) NOT NULL
);

CREATE TABLE Orden (
  id INT AUTO_INCREMENT PRIMARY KEY,
  cliente_id INT NOT NULL,
  fecha VARCHAR(19) NOT NULL,                -- 'YYYY-MM-DD HH:MM:SS'
  canal VARCHAR(20) NOT NULL,                -- libre (no controlado)
  moneda CHAR(3) NOT NULL,                   -- puede ser 'USD' o 'CRC'
  total VARCHAR(20) NOT NULL,                -- a veces '1200.50' o
'1,200.50'
  FOREIGN KEY (cliente_id) REFERENCES Cliente(id)
);

CREATE TABLE OrdenDetalle (
  id INT AUTO_INCREMENT PRIMARY KEY,
  orden_id INT NOT NULL,
  producto_id INT NOT NULL,
  cantidad INT NOT NULL,
  precio_unit VARCHAR(20) NOT NULL,          -- string con comas/puntos
```

```
    FOREIGN KEY (orden_id) REFERENCES Orden(id),
    FOREIGN KEY (producto_id) REFERENCES Producto(id)
  );

  CREATE INDEX IX_Orden_cliente ON Orden(cliente_id);
  CREATE INDEX IX_Detalle_producto ON OrdenDetalle(producto_id);
```

Heterogeneidades planeadas

- codigo_alt (no es el SKU oficial).
- Fechas y montos como **texto** (requiere limpieza, reemplazo de comas/puntos).
- Género M/F/X.
- Moneda mezclada USD/CRC.

3) Supabase / PostgreSQL (género 'M'/'F'; claves UUID; ventas en USD y CRC)

```
-- schema: public (o ventas_pg)

CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

CREATE TABLE cliente (
  cliente_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  nombre TEXT NOT NULL,
  email TEXT UNIQUE,
  genero CHAR(1) NOT NULL CHECK (genero IN ('M','F')),
  pais TEXT NOT NULL,
  fecha_registro DATE NOT NULL DEFAULT CURRENT_DATE
);

CREATE TABLE producto (
  producto_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  sku TEXT UNIQUE,                              -- puede venir vacío en algunos
  nombre TEXT NOT NULL,
  categoria TEXT NOT NULL
);
```

```
CREATE TABLE orden (
  orden_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  cliente_id UUID NOT NULL REFERENCES cliente(cliente_id),
  fecha TIMESTAMPTZ NOT NULL DEFAULT NOW(),
  canal TEXT NOT NULL CHECK (canal IN ('WEB','APP','PARTNER')),  -- 'PARTNER'
distinto
  moneda CHAR(3) NOT NULL,                        -- USD/CRC
  total NUMERIC(18,2) NOT NULL
);

CREATE TABLE orden_detalle (
  orden_detalle_id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  orden_id UUID NOT NULL REFERENCES orden(orden_id),
  producto_id UUID NOT NULL REFERENCES producto(producto_id),
  cantidad INT NOT NULL CHECK (cantidad > 0),
  precio_unit NUMERIC(18,2) NOT NULL
);

CREATE INDEX ix_orden_fecha ON orden(fecha);
CREATE INDEX ix_detalle_producto ON orden_detalle(producto_id);
```

Heterogeneidades planeadas

- IDs como **UUID**.
- Canal incluye valor **'PARTNER'** no presente en otras fuentes.
- Algunos productos sin sku (obliga a mapeo por nombre/categoría).

4) MongoDB (documentos anidados; montos en CRC; género como 'Masculino/Femenino/Otro')

Colecciones: clientes, productos, ordenes

```
// clientes
{
  "_id": ObjectId(),
  "nombre": "Ana Rojas",
```

```
  "email": "ana@ejemplo.com",
  "genero": "Otro",                // valores: Masculino/Femenino/Otro
  "pais": "CR",
  "preferencias": { "canal": ["WEB","TIENDA"] },
  "creado": { "$date": "2025-03-12T00:00:00Z" }
}

// productos
{
  "_id": ObjectId(),
  "codigo_mongo": "MN-9981",       // ni SKU ni codigo_alt
  "nombre": "Tomate grande",
  "categoria": "Alimentos",
  "equivalencias": {
     "sku": "SKU-1002",            // a veces presente, a veces no
     "codigo_alt": "ALT-AB12"
  }
}

// ordenes (CRC)
{
  "_id": ObjectId(),
  "cliente_id": ObjectId("..."),
  "fecha": { "$date": "2025-04-10T13:20:00Z" },
  "canal": "WEB",
  "moneda": "CRC",
  "total": 84500,                  // entero en colones
  "items": [
    { "producto_id": ObjectId("..."), "cantidad": 2, "precio_unit": 12000 },
    { "producto_id": ObjectId("..."), "cantidad": 1, "precio_unit": 60500 }
  ],
  "metadatos": { "cupon": "ABRIL10" }
}
```

Heterogeneidades planeadas

- Montos en **enteros CRC** (sin decimales).
- Campo equivalencias para linkear con otras fuentes (a veces incompleto).
- Género con tercer valor **‘Otro’**.

- Estructura anidada (arrays items).

5) Neo4j (grafo de compras; nodos y relaciones)

Labels y propiedades

- (Cliente {id: string, nombre, genero: 'M'|'F'|'Otro'|'Masculino'|'Femenino', pais})
- (Producto {id: string, nombre, categoria, sku, codigo_alt, codigo_mongo})
- (Categoria {id: string, nombre})
- (Orden {id: string, fecha: datetime, canal, moneda, total})

Relaciones

- (Cliente)-[:REALIZO]->(Orden)
- (Orden)-[:CONTIENTE {cantidad:int, precio_unit:float}]->(Producto)
- (Producto)-[:PERTENECE_A]->(Categoria)
- **Homologación entre catálogos** (para ETL):
    - (Producto)-[:EQUIVALE_A]->(Producto) // para unir SKU vs codigo_alt vs codigo_mongo

Índices/constraints

```
CREATE CONSTRAINT cliente_id IF NOT EXISTS
FOR (c:Cliente) REQUIRE c.id IS UNIQUE;

CREATE CONSTRAINT producto_id IF NOT EXISTS
FOR (p:Producto) REQUIRE p.id IS UNIQUE;

CREATE INDEX orden_fecha IF NOT EXISTS FOR (o:Orden) ON (o.fecha);
```

Heterogeneidades planeadas

- Múltiples códigos por producto (propiedades sku, codigo_alt, codigo_mongo).
- Monedas variadas por Orden.
- Género en formatos mixtos.

Consideraciones para el ETL (clave para los equipos)

1. **DimProducto puente de equivalencias**
   - Construyan una tabla auxiliar en el staging: map_producto(source, source_code, sku_oficial) para resolver SKU ↔ codigo_alt ↔ codigo_mongo.
2. **Normalización de moneda**
   - Tabla tipo_cambio(fecha, de, a, tasa); convertir **CRC→USD** según fecha de la orden.
3. **Estandarización de género**
   - Regla: M|Masculino → Masculino, F|Femenino → Femenino, X|Otro|NULL → No especificado.
4. **Fechas y números**
   - MySQL trae fechas/números como **texto**; limpiar comas/puntos y castear.
5. **Neo4j y Mongo**
   - Flatten de items (una fila por ítem) para la **tabla de hechos**.
6. **Claves**
   - Mantener **clave natural de la fuente** y **clave surrogate** en staging para trazabilidad: source_system, source_key.

Dataset mínimo sugerido (por equipo)

- **Clientes:** ≥ 3 000 (distribuidos entre fuentes).

- **Productos:** ≥ 500 (con solapamiento parcial entre catálogos).
- **Órdenes:** ≥ 25 000 en total (mezcla de USD/CRC y canales). (Considerar ventas al menos en 2024 y 2025)
- **Reglas Apriori:** dataset transaccional (orden–producto) tras ETL.