Evaluación Práctica Unidad 1
AWOS y BDA 5°A

Desarrollar una app en Next.js (TypeScript) que visualiza reportes SQL obtenidos desde VIEWS en PostgreSQL. La solución corre completa con **Docker Compose** y aplica seguridad real (usuario app distinto a postgres, con SELECT solo sobre VIEWS).

Escenario.
Eres responsable de analitica en una cafeteria del campus. No necesitan un punto de venta completo, sino un dashboard de reportes para entender ventas, productos estrella, inventario en riesgo, clientes frecuentes y mezcla de pagos. Tu app debe permitir consultar estos insights con filtros y paginación.

Modelo de datos sugerido (al menos 5 tablas).
Puedes ajustar nombres, pero debes mantener relaciones reales (FK) y datos suficientes para reportes.

- categories(id, name)
- products(id, name, category_id, price, stock, active)
- customers(id, name, email)
- orders(id, customer_id, created_at, status, channel)
- order_items(id, order_id, product_id, qty, unit_price)
- payments(id, order_id, method, paid_amount).

Reportes (VIEWS) obligatorios para este escenario.
Debes implementar al menos estas 5 VIEWS (puedes agregar mas).

- `vw_sales_daily`: 1 fila por dia. Metricas: total_ventas, tickets, ticket_promedio. Incluye filtros por rango de fechas.
- `vw_top_products_ranked (Window)`: ranking por revenue/unidades. Debe permitir búsqueda por nombre y paginación.

- `vw_inventory_risk`: productos o categorias con stock bajo y porcentaje en riesgo.
- `vw_customer_value`: total_gastado, num_ordenes, gasto_promedio por cliente. Debe permitir paginación.
- `vw_payment_mix`: total y porcentaje por método de pago

Plan obligatorio de filtros, búsqueda y paginación.

- Filtros: vw_sales_daily (date_from, date_to) y vw_inventory_risk (category_id o categoria por whitelist).
- Búsqueda: vw_top_products_ranked (search por nombre).
- Paginacion server-side: vw_top_products_ranked y vw_customer_value (page, limit).
- Todas las consultas deben ser SELECT sobre VIEWS (sin acceso directo a tablas desde la app).

Requisitos obligatorios.

A) Base de datos (db/)

- Incluye: db/schema.sql, db/seed.sql, db/migrate.sql (y se ejecutan al levantar el contenedor).
- Tu BD debe tener mínimo 5 tablas y mínimo 2 relaciones FK reales.
- db/seed.sql inserta datos suficientes para que todas las views tengan resultados, y para demostrar filtros y paginación.

B) VIEWS (db/reports_vw.sql)

- Crea db/reports_vw.sql con mínimo 5 VIEWS.
- Cada view incluye: 1 agregado (SUM/COUNT/AVG/MIN/MAX), GROUP BY y 1 campo calculado (ratio/porcentaje/CASE/COALESCE).
- Mínimo 2 VIEWS con HAVING.
- Mínimo 2 VIEWS con CASE o COALESCE significativo.
- Mínimo 1 VIEW con CTE (WITH...).
- Mínimo 1 VIEW con Window Function.
- En mínimo 2 VIEWS está prohibido SELECT * (lista columnas con aliases legibles).
- Arriba de cada VIEW agrega comentario: que devuelve, grain, métricas e incluye 1-2 queries VERIFY.

C) Indices (db/indexes.sql)

- Crea db/indexes.sql con mínimo 3 índices relevantes.
- Incluye evidencia con EXPLAIN en README (al menos 2 consultas).

D) Seguridad (db/roles.sql)

- La app NO se conecta como postgres.
- Crea un usuario/rol app con SELECT solo sobre VIEWS (no sobre tablas).
- Documenta en README cómo verificarlo.

E) Next.js (App Router)

- Dashboard (/): tarjetas o links a reportes.
- Mínimo 5 pantallas de reportes, una por VIEW (por ejemplo */reports/1 .../reports/5*).
- Cada reporte incluye: título, descripción del insight, tabla legible, y al menos 1 KPI destacado.
- **Prohibido exponer credenciales al cliente.** Data fetching server-side (Server Components o API Routes).
- Consultas: solo SELECT sobre VIEWS.

F) Filtros y paginación

- Mínimo 2 reportes con filtros o búsqueda (Zod + whitelist + query parametrizada).
- Mínimo 2 reportes con paginación server-side (limit/offset o cursor) con validación de page/limit.

G) Docker Compose

- Debe correr con: `docker compose up --build`
- Debe levantar Postgres + Next.js (y lo que uses adicional debe estar documentado).