Evaluación Práctica Unidad 1
AWOS y BDA 5°B

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 parte de coordinación académica. Necesitan un dashboard de reportes para identificar rendimiento, reprobación, asistencia y alumnos en riesgo. Tu app debe permitir filtrar por periodo y buscar alumnos, además de paginar resultados.

Modelo de datos sugerido (al menos 5 tablas)

Puedes ajustar nombres, pero debes mantener relaciones reales (FK) y datos suficientes para reportes.

- students(id, name, email, program, enrollment_year)
- teachers(id, name, email)
- courses(id, code, name, credits)
- groups(id, course_id, teacher_id, term)
- enrollments(id, student_id, group_id, enrolled_at)
- grades(id, enrollment_id, partial1, partial2, final)
- attendance(id, enrollment_id, date, present)

Reportes (VIEWS) obligatorios para este escenario

Debes implementar al menos estas 5 VIEWS (puedes agregar más).

- `vw_course_performance`: 1 fila por course+term. Promedios y reprobados (CASE). Filtro por term y/o program.
- `vw_teacher_load (HAVING)`: carga por docente+term (grupos, alumnos_totales, promedio_general). Paginable.
- `vw_students_at_risk (CTE)`: alumnos con promedio bajo o asistencia baja. Búsqueda por name/email y paginación.

- `vw_attendance_by_group (CASE/COALESCE)`: asistencia promedio por grupo (term).
- `vw_rank_students (Window)`: ranking por program y term (RANK/ROW_NUMBER).

Plan obligatorio de filtros, búsqueda y paginación

- Filtros (Zod + parametrizado): vw_course_performance (term obligatorio) y vw_rank_students (program por whitelist).
- Búsqueda (Zod): vw_students_at_risk (search por name/email).
- Paginación server-side: vw_students_at_risk y vw_teacher_load (page, limit).
- Todas las consultas: SELECT desde VIEWS; no se permite lectura de 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).