Parcial 2

Julián Andrés Guisao Fernández

Escuela de ingenierías, Universidad Pontificia Bolivariana

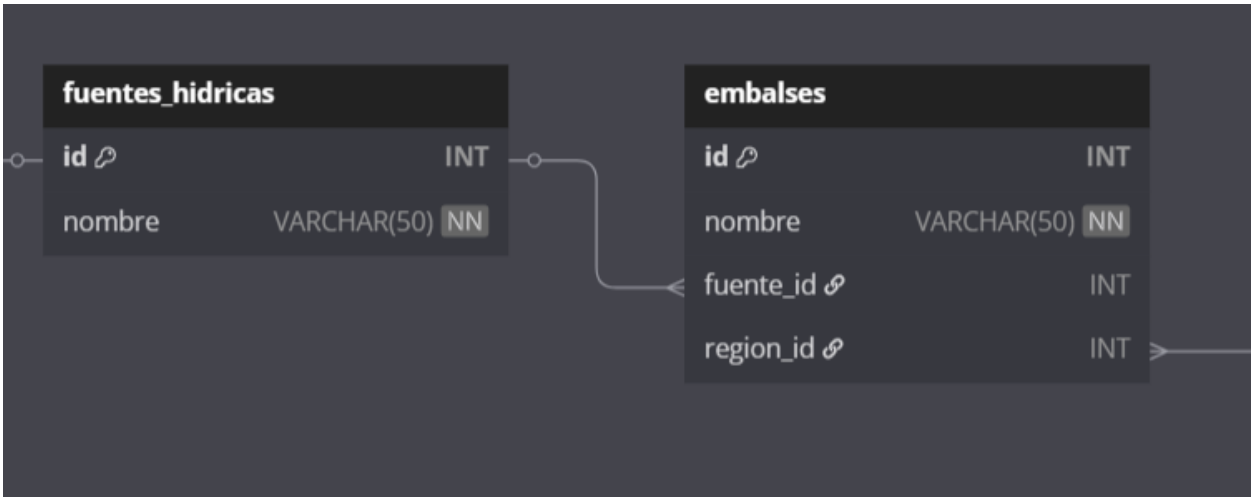
Tópicos Avanzados en Bases de datos

Juan Rodas

3 de abril de 2025

Etapa 1: Obtención colaborativa de datos faltantes. 20%

Para la normalización de los datos, se identificó que la columna "serie_hidrologica" representa embalses y fuentes hídricas. Se construyeron dos nuevas tablas: embalses y fuentes_hidricas.



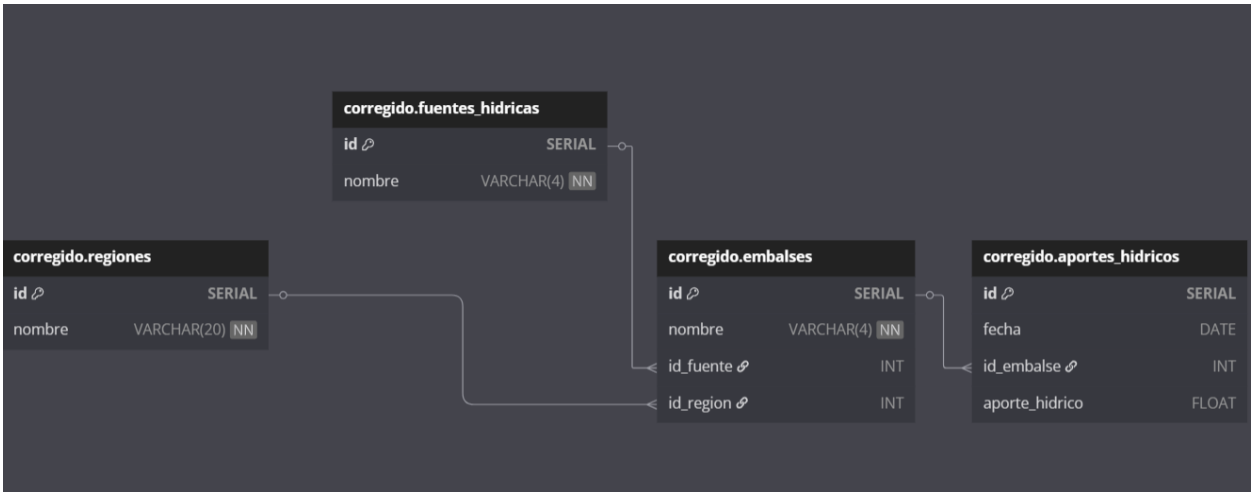
Creación de tabla de embalses y fuentes hídricas

```sql
CREATE TABLE IF NOT EXISTS corregido.fuentes_hidricas (
  "id" SERIAL PRIMARY KEY,
  "nombre" VARCHAR(4) NOT NULL
);

CREATE TABLE IF NOT EXISTS corregido.embalses (
  "id" SERIAL PRIMARY KEY,
  "nombre" VARCHAR(4) NOT NULL,
  "id_fuente" INT,
  "id_region" INT,
  FOREIGN KEY (id_fuente) REFERENCES corregido.fuentes_hidricas ("id"),
  FOREIGN KEY (id_region) REFERENCES corregido.regiones ("id")
);
```

Etapa 2: Diseño e implementación del modelo de datos – 20%

Para responder consultas de manera eficiente, se diseñó un modelo relacional en tercera forma normal (3FN). A continuación, se muestra la estructura de las tablas y su código de implementación.



Creación de tablas adicionales para completar el esquema base.

```sql
CREATE SCHEMA IF NOT EXISTS corregido;

CREATE TABLE IF NOT EXISTS corregido.regiones (
  "id" SERIAL PRIMARY KEY,
  "nombre" VARCHAR(20) NOT NULL UNIQUE
);

CREATE TABLE IF NOT EXISTS corregido.fuentes_hidricas (
  "id" SERIAL PRIMARY KEY,
  "nombre" VARCHAR(4) NOT NULL
);

CREATE TABLE IF NOT EXISTS corregido.embalses (
  "id" SERIAL PRIMARY KEY,
  "nombre" VARCHAR(4) NOT NULL,
  "id_fuente" INT,
  "id_region" INT,
  FOREIGN KEY (id_fuente) REFERENCES corregido.fuentes_hidricas ("id"),
  FOREIGN KEY (id_region) REFERENCES corregido.regiones ("id")
);

CREATE TABLE IF NOT EXISTS corregido.aportes_hidricos (
  "id" SERIAL PRIMARY KEY,
  "fecha" DATE,
  "id_embalse" INT,
  "aporte_hidrico" FLOAT,
  FOREIGN KEY (id_embalse) REFERENCES corregido.embalses ("id")
);
```

Creación de índices

```sql
CREATE INDEX IF NOT EXISTS idx_regiones_nombre ON corregido.regiones(nombre);
CREATE INDEX IF NOT EXISTS idx_regiones_id ON corregido.regiones(id);
CREATE INDEX IF NOT EXISTS idx_embalses_nombre ON corregido.embalses(nombre);
CREATE INDEX IF NOT EXISTS idx_embalses_id ON corregido.embalses(id);
CREATE INDEX IF NOT EXISTS idx_fuentes_nombre ON corregido.fuentes_hidricas(nombre);
CREATE INDEX IF NOT EXISTS idx_fuentes_id ON corregido.fuentes_hidricas(id);
CREATE INDEX IF NOT EXISTS idx_aportes_id ON corregido.aportes_hidricos(id);
CREATE INDEX IF NOT EXISTS idx_aportes_fecha ON corregido.aportes_hidricos(fecha);
```

Etapa 3: Diagnóstico de completitud de datos – 30%

Antes de índices

```sql
EXPLAIN ANALYZE
WITH dias_por_anio AS (
    SELECT 2023 AS anio, 365 AS total_dias UNION ALL
    SELECT 2024 AS anio, 366 AS total_dias
),
conteo_aportes AS (
    SELECT
        e.id AS id_embalse,
        e.id_region,
        EXTRACT(YEAR FROM ah.fecha) AS anio,
        COUNT(ah.id) AS dias_con_aporte
    FROM corregido.aportes_hidricos ah
    JOIN corregido.embalses e ON ah.id_embalse = e.id
    GROUP BY e.id, e.id_region, anio
)
SELECT
    r.nombre AS region,
    e.nombre AS embalse,
    c.anio,
    c.dias_con_aporte,
    d.total_dias,
    ROUND((c.dias_con_aporte::NUMERIC / d.total_dias) * 100, 2) AS porcentaje_completitud
FROM conteo_aportes c
JOIN dias_por_anio d ON c.anio = d.anio
JOIN corregido.embalses e ON c.id_embalse = e.id
JOIN corregido.regiones r ON e.id_region = r.id
ORDER BY r.nombre, e.nombre, c.anio;
```

	QUERY PLAN
1	Sort (cost=3459.39..3460.87 rows=590 width=154) (actual time=29.630..29.638 rows=88 loops=1)
2	Sort Key: r.nombre, e.nombre, (EXTRACT(year FROM ah.fecha))
3	Sort Method: quicksort Memory: 30kB
4	-> Hash Join (cost=1793.63..3432.24 rows=590 width=154) (actual time=29.305..29.537 rows=88 loops=1)
5	Hash Cond: (e.id_region = r.id)
6	-> Hash Join (cost=1763.38..3393.06 rows=590 width=68) (actual time=29.264..29.463 rows=88 loops=1)
7	Hash Cond: (e_1.id = e.id)
8	-> Hash Join (cost=1722.78..3350.91 rows=590 width=48) (actual time=29.173..29.356 rows=88 loops=1)

Después de índices

```
EXPLAIN ANALYZE
WITH dias_por_anio AS (
    SELECT 2023 AS anio, 365 AS total_dias UNION ALL
    SELECT 2024 AS anio, 366 AS total_dias
),
conteo_aportes AS (
    SELECT
        e.id AS id_embalse,
        e.id_region,
        EXTRACT(YEAR FROM ah.fecha) AS anio,
        COUNT(ah.id) AS dias_con_aporte
    FROM corregido.aportes_hidricos ah
    JOIN corregido.embalses e ON ah.id_embalse = e.id
    GROUP BY e.id, e.id_region, anio
)
SELECT
    r.nombre AS region,
    e.nombre AS embalse,
    c.anio,
    c.dias_con_aporte,
    d.total_dias,
    ROUND((c.dias_con_aporte::NUMERIC / d.total_dias) * 100, 2) AS porcentaje_completitud
FROM conteo_aportes c
JOIN dias_por_anio d ON c.anio = d.anio
JOIN corregido.embalses e ON c.id_embalse = e.id
JOIN corregido.regiones r ON e.id_region = r.id
ORDER BY r.nombre, e.nombre, c.anio;
```

```
QUERY PLAN
Incremental Sort  (cost=1740.28..2463.63 rows=71 width=154) (actual time=27.537..27.674 rows=88 loops=1)
  Sort Key: r.nombre, e.nombre, (EXTRACT(year FROM ah.fecha))
  Presorted Key: r.nombre
  Full-sort Groups: 3  Sort Method: quicksort  Average Memory: 27kB  Peak Memory: 27kB
  -> Nested Loop  (cost=1595.81..2461.36 rows=71 width=154) (actual time=27.009..27.516 rows=88 loops=1)
      Join Filter: (r.id = e.id_region)
      Rows Removed by Join Filter: 440
      -> Index Scan using idx_regiones_nombre on regiones r  (cost=0.13..12.22 rows=6 width=62) (actual time=0.058..0.06...
```

Etapa 4: Diagnóstico de niveles mínimos de aporte hídrico– 30%

Antes de índices

```
EXPLAIN ANALYZE
WITH aportes_2024 AS (
    SELECT
        ah.id_embalse,
        ah.aporte_hidrico
    FROM corregido.aportes_hidricos ah
    WHERE EXTRACT(YEAR FROM ah.fecha) = 2024
),
valores_extremos AS (
    SELECT
        id_embalse,
        MIN(aporte_hidrico) AS valor_minimo,
        MAX(aporte_hidrico) AS valor_maximo
    FROM aportes_2024
    GROUP BY id_embalse
)
SELECT
    e.nombre AS embalse,
    v.valor_maximo,
    v.valor_minimo,
    CASE
        WHEN v.valor_maximo = 0 THEN NULL -- Para evitar división por cero
        ELSE ((v.valor_maximo - v.valor_minimo) / v.valor_maximo) * 100
    END AS porcentaje_reduccion
FROM valores_extremos v
JOIN corregido.embalses e ON v.id_embalse = e.id
ORDER BY porcentaje_reduccion DESC;
```

```
QUERY PLAN
Sort  (cost=1195.35..1195.46 rows=44 width=44) (actual time=13.162..13.166 rows=44 loops=1)
  Sort Key: (CASE WHEN (v.valor_maximo = '0'::double precision) THEN NULL::double precision ELSE (((v.valor_maximo - v....
  Sort Method: quicksort  Memory: 27kB
  -> Hash Join  (cost=1166.53..1194.15 rows=44 width=44) (actual time=13.123..13.138 rows=44 loops=1)
        Hash Cond: (e.id = v.id_embalse)
```

Después de índices

```
EXPLAIN ANALYZE
WITH aportes_2024 AS (
    SELECT
        ah.id_embalse,
        ah.aporte_hidrico
    FROM corregido.aportes_hidricos ah
    WHERE EXTRACT(YEAR FROM ah.fecha) = 2024
),
valores_extremos AS (
    SELECT
        id_embalse,
        MIN(aporte_hidrico) AS valor_minimo,
        MAX(aporte_hidrico) AS valor_maximo
    FROM aportes_2024
```

QUERY PLAN

```
Sort  (cost=1169.74..1169.85 rows=44 width=44) (actual time=11.255..11.259 rows=44 loops=1)
  Sort Key: (CASE WHEN (v.valor_maximo = '0'::double precision) THEN NULL::double precision ELSE (((v.valor_maximo - v....
  Sort Method: quicksort  Memory: 27kB
  -> Hash Join  (cost=1166.53..1168.54 rows=44 width=44) (actual time=11.210..11.222 rows=44 loops=1)
     Hash Cond: (e.id = v.id_embalse)
     -> Seq Scan on embalses e  (cost=0.00..1.44 rows=44 width=24) (actual time=0.004..0.007 rows=44 loops=1)
     -> Hash  (cost=1165.98..1165.98 rows=44 width=20) (actual time=11.193..11.194 rows=44 loops=1)
```