BD2 - Laboratorio 4.2: GiST
Grupo 8
Integrantes:
Jesús Valentin Niño Castañeda
Daniel Ignacio Casquino Paz

1. Consulta por rango

```
SELECT cities.name
FROM cities
WHERE ST_DWithin( geog1 ubicacion,  geog2 ST_MakeEnvelope(
        -79.4742, -8.6159, -- Esquina inferior izquierda
        -78.4742, -7.6159, -- Esquina superior derecha
        4326),  tolerance 10000);
```

	name
1	Gran Chimu
2	Julcan
3	Otuzco
4	Provincia de Otuzco
5	Viru
6	Ascope
7	Chicama
8	Chocope
9	Laredo
10	Moche
11	Paiján
12	Provincia de Trujillo
13	Salaverry
14	Santiago de Cao
15	Trujillo

2. 10 ciudades más cercanas a Lima

```sql
SELECT c.name,
       c.country_name,
       ST_Distance( geog1 c.ubicacion::geography,  geog2 qp.ubicacion::geography) AS distance
FROM cities c,
     (SELECT ST_SetSRID( geog ST_MakePoint(-12.04318, -77.02824),  srid 4326) AS ubicacion) qp
ORDER BY c.ubicacion <-> qp.ubicacion
LIMIT 10;
```

	name character varying (100)	country_name character varying (100)	distance double precision
1	Antártica	Chile	1549986.14680743
2	Cabo de Hornos	Chile	3288808.35026241
3	Ushuaia	Argentina	3318347.63251109
4	Tolhuin	Argentina	3322280.82648414
5	Río Grande	Argentina	3409481.49581949
6	Timaukel	Chile	3474992.21457804
7	Porvenir	Chile	3524313.64513647
8	Punta Arenas	Chile	3551730.91972221
9	Primavera	Chile	3557766.3689789
10	San Gregorio	Chile	3609834.32160389

3. Stored procedure KNN

```sql
CREATE OR REPLACE FUNCTION _knn(in_city_name VARCHAR(100), in_country_name VARCHAR(100))
    RETURNS TABLE(id INTEGER, name VARCHAR, state_name VARCHAR, country_name VARCHAR,
    latitude DOUBLE PRECISION, longitude DOUBLE PRECISION, distance DOUBLE PRECISION)
LANGUAGE SQL
AS $$
WITH sourceInfo AS (
    SELECT c.latitude, c.longitude, c.id AS source_id
    FROM cities c
    WHERE c.name = in_city_name AND c.country_name = in_country_name
    LIMIT 1
)
SELECT c.id, c.name, c.state_name, c.country_name, c.latitude, c.longitude,
    ST_Distance(c.ubicacion::geography, ST_SetSRID(ST_MakePoint(si.longitude, si.latitude), 4326)::geography)
    AS distance
FROM cities c, sourceInfo si
WHERE NOT c.id = si.source_id
ORDER BY c.ubicacion <-> ST_SetSRID(ST_MakePoint(si.longitude, si.latitude), 4326)
LIMIT 5;
$$;

SELECT * FROM _knn('Lima', 'Peru');
```

En la función creada, decidimos recibir tanto el nombre de la ciudad como el nombre del país al que pertenece, ya que la base de datos contiene ciudades con el mismo nombre en distintos países. Para evitar redundancias, limitamos la búsqueda de la ciudad source a 1 elemento.

Primero buscamos las coordenadas de la ciudad especificada con el input, y luego se realiza el knn search que se usó en el ítem anterior. Adicionalmente, excluimos a la misma ciudad input, ya que al tener distancia 0 a sí misma, termina siendo uno de los knns. El tiempo promedio, luego de 5 consultas, es de 284ms.

Resultados con búsqueda ‘Lima’, ‘Peru’:

	id integer	name character varying	state_name character varying	country_name character varying	latitude double precision	longitude double precision	distance double precision
1	80950	San Luis	Lima	Peru	-12.07674	-76.99435	5234.30672401
2	80672	Independencia	Lima	Peru	-11.99	-77.04583	6186.88786968
3	80945	San Isidro	Lima	Peru	-12.11667	-77.05	8467.81759334
4	80587	Callao	Callao	Peru	-12.05659	-77.11814	9900.31437933
5	80967	Santiago de Surco	Lima	Peru	-12.13588	-77.00742	10502.25799597

4. Análisis de rendimiento

```sql
CREATE MATERIALIZED VIEW tokyo AS
SELECT ubicacion
FROM cities
WHERE name = 'Tokyo'
LIMIT 1;

--Con GiST
EXPLAIN ANALYSE
SELECT name, c.ubicacion <-> (SELECT ubicacion FROM tokyo) AS distance
FROM cities20k AS c
ORDER BY distance
LIMIT 100;

--Sin GiST
EXPLAIN ANALYSE
SELECT name, c.ubicacion2 <-> (SELECT ubicacion FROM tokyo) AS distance
FROM cities20k AS c
ORDER BY distance
LIMIT 100;
```

Plan de ejecución con GiST

```
QUERY PLAN
Limit  (cost=1.29..40.25 rows=100 width=19) (actual time=0.338..1.135 rows=100 loops=1)
  InitPlan 1
    ->  Seq Scan on tokyo  (cost=0.00..1.01 rows=1 width=32) (actual time=0.032..0.032 rows=1 loops=1)
  ->  Index Scan using idx_cities20k_geom_gist on cities20k c  (cost=0.28..7794.27 rows=20000 width=19) (actual time=0.338..1.126 rows…
        Order By: (ubicacion <-> (InitPlan 1).col1)
Planning Time: 2.488 ms
Execution Time: 1.194 ms
```

Plan de ejecución sin GiST

```
QUERY PLAN
Limit  (cost=6394.40..6394.65 rows=100 width=19) (actual time=34.838..34.854 rows=100 loops=1)
  InitPlan 1
    ->  Seq Scan on tokyo  (cost=0.00..1.01 rows=1 width=32) (actual time=0.007..0.008 rows=1 loops=1)
  ->  Sort  (cost=6393.39..6443.39 rows=20000 width=19) (actual time=34.837..34.843 rows=100 loops=1)
        Sort Key: ((c.ubicacion2 <-> (InitPlan 1).col1))
        Sort Method: top-N heapsort  Memory: 34kB
        ->  Seq Scan on cities20k c  (cost=0.00..5629.00 rows=20000 width=19) (actual time=0.035..31.818 rows=20000 loops=1)
Planning Time: 1.674 ms
Execution Time: 34.888 ms
```

Cuadro comparativo de rendimiento en ms

	20k	40k	60k	80k
Con GiST	1.940	1.189	1.143	1.704
Sin GiST	34.676	68.327	123.015	143.532

5. Detección de ciudades aisladas

Se probaron las siguientes consultas, de las cuales se eligió la última.

```sql
--Decente(demora ~30 segundos)
SELECT c1.name, distance
FROM cities c1
         JOIN LATERAL (
    SELECT c1.ubicacion <-> c2.ubicacion AS distance
    FROM cities c2
    WHERE c1.id != c2.id
    ORDER BY distance
    LIMIT 1
    ) c2 ON TRUE
WHERE distance > 50000;

--Muy Lento(no termina)
SELECT c1.name
FROM cities c1
WHERE NOT EXISTS(SELECT 1
                 FROM cities c2
                 WHERE c1.id != c2.id
                   AND c1.ubicacion <-> c2.ubicacion < 50000);

--Óptimo
SELECT c1.name
FROM cities c1
WHERE NOT EXISTS(SELECT 1
                 FROM cities c2
                 WHERE c1.id != c2.id
                   AND ST_DWITHIN( geog1 c1.ubicacion,  geog2 c2.ubicacion,  tolerance 50000));
```

A continuación la tabla comparativa de rendimiento, no se muestran pruebas sin el índice GiST porque la consultas se vuelven extremadamente lentas, siendo que con la tabla de 20k pasan 5 minutos y no termina de ejecutarse.

	20k	40k	60k	80k	Total
Con GiST	757.101	1937.622	1484.928	2025.481	3536.961

Plan de ejecución de la query:

```
QUERY PLAN
Gather  (cost=1000.40..11621089.62 rows=147723 width=11) (actual time=409.646..3661.735 rows=2590 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Nested Loop Anti Join  (cost=0.41..11605317.32 rows=61551 width=11) (actual time=391.958..3586.357 rows=863 loops=3)
        ->  Parallel Seq Scan on cities c1  (cost=0.00..3965.92 rows=61692 width=47) (actual time=0.012..25.158 rows=49354 loops=3)
        ->  Index Scan using idx_cities_geom_gist on cities c2  (cost=0.41..188.33 rows=15 width=36) (actual time=0.071..0.071 rows=1 loops=148061)
              Index Cond: (ubicacion && _st_expand(c1.ubicacion, '50000'::double precision))
              Filter: ((c1.id <> id) AND st_dwithin(c1.ubicacion, ubicacion, '50000'::double precision, true))
              Rows Removed by Filter: 2
Planning Time: 6.176 ms
Execution Time: 3662.052 ms
```

6. Experimento con dataset distinto

Pendiente.