FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico
GIS Colectivos

[75.15 - 95.05] Base de Datos
Primer cuatrimestre de 2019
Grupo L1
Fecha de entrega: 05/06/2019

Alumno:	**Número de padrón**
SPASIUK, Kevin	99849
PINTO, Santiago Augusto	96850
JUSTO NARCIZO, Edson	97775

Índice

1. Introducción

El presente informe reúne la documentación de la solución del trabajo práctico de la materia Base de Datos que consiste en diseñar una base de datos para un GIS(Sistema de Información Geográfica) que permita trabajar con los datos reales de los recorridos de colectivo y pueda efectuar distintas consultas

El desarrollo fue hecho en PostgreSQL usando el módulo `PostGIS`, para convertirlo una base de datos espacial.

Los scripts usandos se encuentran en:

- https://github.com/kevinspasiuk/base_de_datos

Los archivos fuentes procesados y utilizados en la creación de las tablas se encuentra en: https://github.com/kevinspasiuk/base_de_datos/tree/master/fuentes

En: https://github.com/kevinspasiuk/base_de_datos/tree/master/sql se ubican las queries de carga de tablas:

- 01-sql_tp_colectivos_creacion_tablas.sql
- 02-recorrido_colectivos_lucas_loader.sql
- 03-creacion_tabla_colectivos_por_parada.sql

En la misma ruta se ecuentran las soluciones a los requerimientos:

- SQL punto 1.sql
- SQL punto 2.sql
- SQL punto 3.sql
- SQL punto 4.sql

2. Distribución de tareas y trabajo realizado

2.1. Análisis previo

Antes de empezar a crear las consultas sql, se realizó un fase de análisis de los datos obtenidos de la página del gobierno de la ciudad, los mismo se encontraban en distintos formatos (csv, shp, kml) los dos últimos corresponden a formatos para exportar una base de datos geográfica.

En primera instancia se decidió usar los archivos .csv, ya que son los más conocidos, analizando los mismos vimos que presentaba la representación en tipo de texto de datos geométricos (LineString) por lo que se tuvo que investigar los tipos y funciones de las base de datos geográficas, en particular el módulo de PostGIS para PostgreSQL. Luego realizamos entre los tres el correspondiente diagrama entidad - interrelación, que cumpliera con la organización de los datos y permitiera resolver los requerimientos del trabajo de manera más sencilla.

2.2. Esquema relacional

Una vez armado el diagrama entidad - interrelacion realizamos el pasaje del mismo al esquema relacional, para visualizar que tablas deberiamos manejar en nuestra base de datos. El resultado puede apreciarse en la figura 1.

Luego del pasaje de diagrama a relaciones, hubo varios inconvenientes: En primer lugar existían filas repetidas en los dataset por lo que realizo una limpieza de los datos, por otro lado aparecian columnas que no aportaban mucha información para el desarrollo del trabajo, por lo que se decidió eliminarlas. Luego de que Kevin redujera el set de datos y quedarnos solo con la información de interés (ver sección carga de datos), se procedió al levantamiento de los datos en la SGDB. Ésta fue sin duda la parte de mayor esfuerzo, debido a los problemas que surgieron propios del formato de los tipo de datos geográficos.

2.3. Consultas SQL

Finalmente realizamos las consultas pedidas por el enunciado, en este caso la divsion de tareas fue la siguiente:

- Paradas Cercanas: Santiago
- Colectivo a utilizar: Santiago
- Ramales de parada: Edson
- Combinacion de colectivos: Kevin

3. Esquema de la base de datos

3.1. Modelo Entidad-interrelación

Comenzamos el armado del esquema modelando el problema planteado. En una primera instancia realizamos el primer modelado:

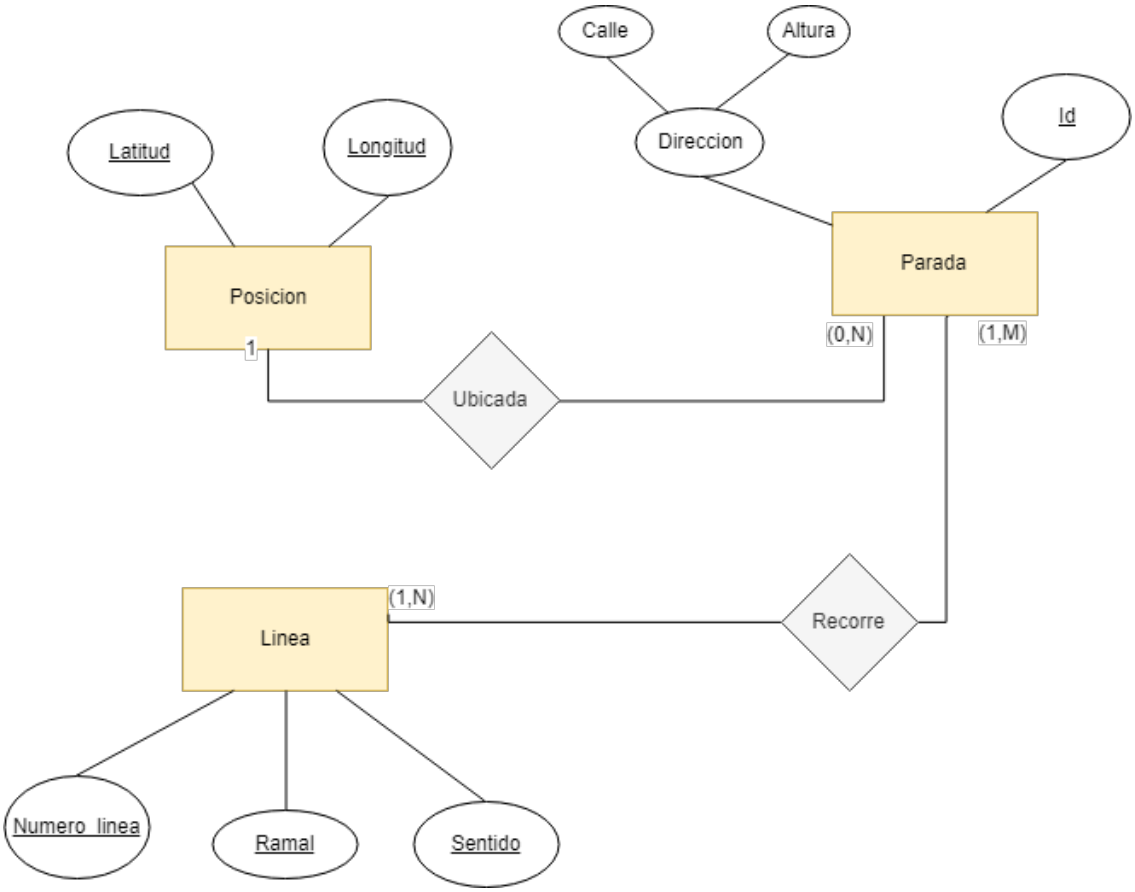

Figura 1: Modelo entidad-interrelación

Se usó este modelado para tener un mejor conocimiento del problema, de los requerimientos y de las relaciones involucradas en la resolución del trabajo práctico.

3.2. Esquema en SGBD

El esquema de la base de datos que utilizamos es el siguiente:

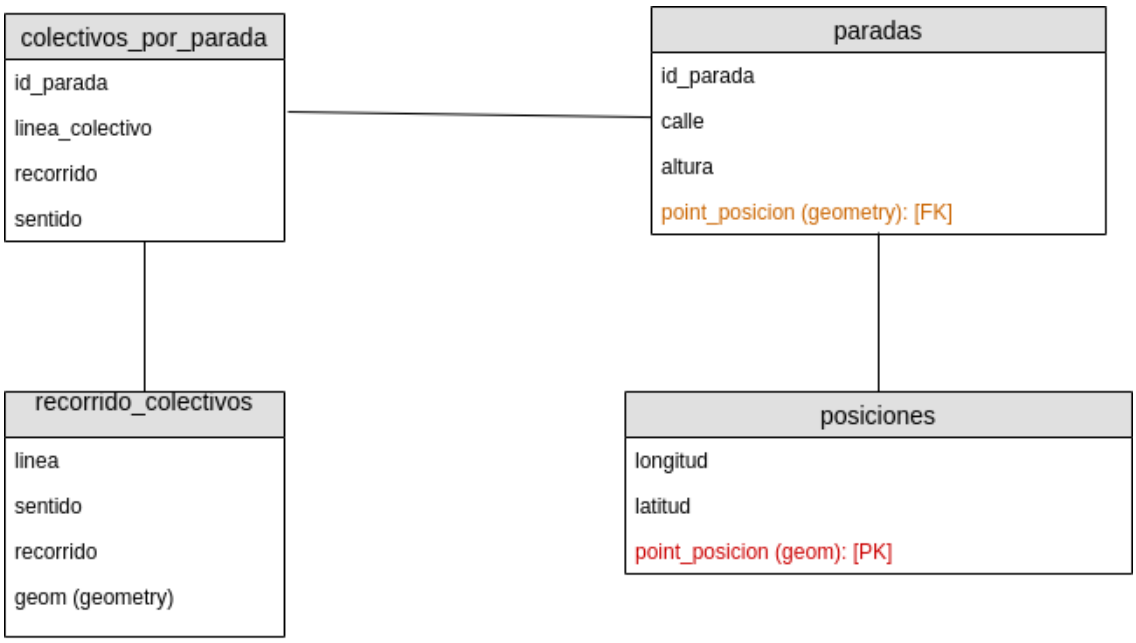

Figura 2: Esquema de la base de datos

En el pasaje del modelo Entidad-interrelación, realizamos modificaciones tanto en el nombre de las tablas como las entidades de la misma. Un ejemplo puntual fue en el caso de la entidad Línea, que en el esquema de la base de datos se denomina recorrid0_colectivos, ya que adquiere mayor importancia el recorrido en el momento de la construcción de las consultas.

3.3. Descripción de tablas:

1. **Posiciones:**
   Tabla que surge del pasaje del modelo entidad-interrelación al esquema relacional, la misma representa cualquier punto con 2 coordenadas (longitud y latitud), representable como una geometría de tipo point en la columna **point_posición**.

2. **Paradas:**
   Tabla que representa las posibles paradas de todos los recorridos existentes en el set de datos que estamos utilizando. Si bien en la creación de tabla no lo especificamos (por algunos problemas de compatibilidad con distintas versiones del pgadmin que manejamos), utilizamos a **id_parada** como la clave primaria. La **calle** y la **altura** son los campos que utilizamos para mostrar dónde se encuentra la parada en las consultas que así lo requieren (ver sección de consultas de sql).

   Finalmente **point_posicion** hace referencia a la posición de la parada , convertida a un formato geoespacial de tipo **POINT** para poder utilizar las funciones que provee *postGIS* para el manejo de consultas que involucran datos geoespaciales, como por ejemplo la distancia entre dos puntos.

3. **Colectivos por parada:**
   Tabla que se usa para no tener en la tabla de paradas los colectivos que pasan por cada parada, eliminando de esta forma redundancias. El atributo **id_parada** lo utilizamos para vincular la parada con los colectivos de la misma, por otro lado **linea_colectivo** es el

atributo que representa el número de un determinado colectivo que pasa por una parada. Por recorrido entendemos el ramal (A,B,C, etc), dado que una misma línea de colectivo puede tener más de un recorrido posible más allá de que en algunas paradas coinciden. Finalmente la columna **sentido** contiene la información de si el el colectivo está haciendo un recorrido en el sentido de ida o vuelta, es importante discriminar esto porque no da lo mismo, dados dos puntos en un cierto orden, para que sentido se debe tomar el colectivo.

4. **Recorrido colectivos:**
   Tabla creada para eliminar redundancias en *colectivos_por_parada*, la misma contiene los datos referentes a todos los recorridos (ramales) de todos los colectivos en el set de datos utilizado. Los atributos linea, sentido y ramal tienen el mismo significado que para la tabla *colectivos_por_parada*, y son usados en las juntas que involucran estas dos tablas. Por último esta tabla tiene una columna **geom** que es una geometría, concretamente del tipo Linestring y contiene el conjunto de puntos que conforman un recorrido determinado, de esta manera se tiene una representación geométrica de los recorridos para usarse en las consultas.

4. Carga de datos

4.1. Preprocesado de datos

Para la carga de los datos, en primera instancia se redujo los campos del archivo paradas-de-colectivo.csv. Luego parseamos dicho archivo para dejarlo en formato (id_parada,linea):

```
import csv
import re

OPEN_PATH = r"C:\Users\kspasiuk\Desktop\base_de_datos_tp_colectivos\
paradas-de-colectivo-reducido.csv"
WRITE_PATH = r"C:\Users\kspasiuk\Desktop\base_de_datos_tp_colectivos\
paradas-de-colectivo-parseado.csv"

with open(OPEN_PATH, mode='r') as csv_file:
    with open (WRITE_PATH,mode="w") as csv_parseado:

        csv_reader = csv.reader(csv_file,delimiter=';')
        line_count = 0

        csv_writer = csv.writer( csv_parseado, delimiter=';',
            quotechar='"', quoting=csv.QUOTE_MINIMAL)

        for row in csv_reader:
            if line_count == 0:
                csv_writer.writerow(['id_parada', 'linea'])

            if line_count != 0:
                # Obtengo los campos que me interesan
                id_parada = row[2]
                lineas = row[5]

                # Convierto las lineas en lista
                lineas_limpias = re.sub('[{}]', '', lineas)
                lista_lineas = lineas_limpias.split(",")

                # Escribo los registros en el csv
                for linea in lista_lineas:
                    csv_writer.writerow([id_parada,linea])

            line_count = line_count+1
```

4.2. Procesamiento de tablas

Para el procesamiento de las tablas se utilizaron cuatro archivos fuentes, siguiendo el siguiente esquema:

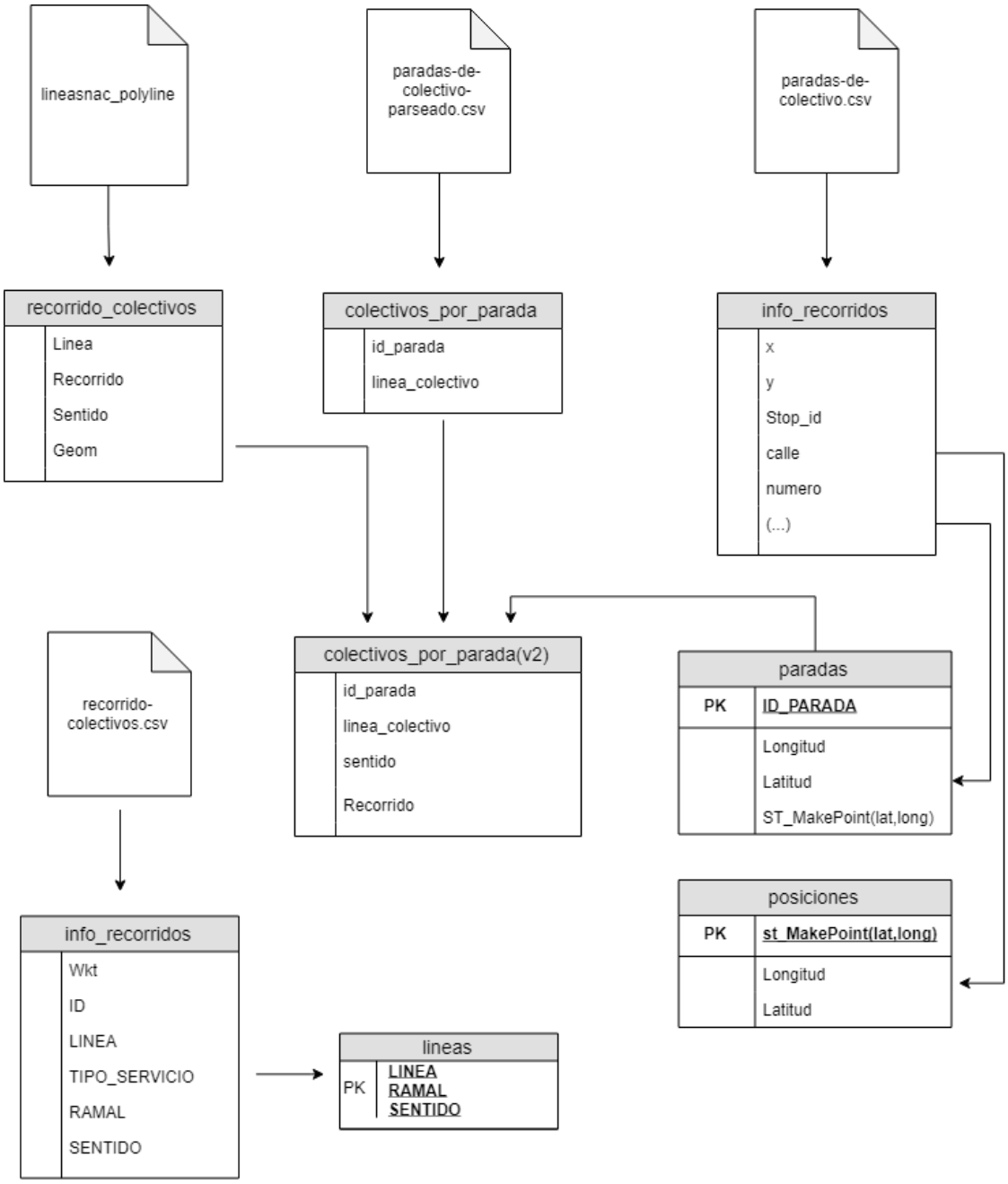

Figura 3: Esquema de carga de datos

Para alimentar la tabla inicial tabla colectivos_por_parada con el sentido y recorrido se usó la geometría del recorrido y el st_point de la parada. Exigimimos para la pertenecia de un sentido y recorrido, dada una parada y línea, que la distancia entre ambos sea la menor:

```
create table colectivos_por_parada as(
select t1.id_parada,
     t1.linea_colectivo,
     t1.sentido,
     t1.recorrido
from tmp_t1 t1
inner join (
    select
        id_parada,
        linea_colectivo,
        min(distance) as min_distance
    from tmp_t1 t1
    group by
    id_parada,
    linea_colectivo) t2 on t2.id_parada = t1.id_parada and t1.linea_colectivo = t2.linea_colectivo and
    t2.min_distance = t1.distance);
```

Figura 4: Sql creación colectivos_por_parada

La tabla tmp_1 contiene la distancia entre la parada y la geometría de la línea

4.3. Inconvenientes

En la carga de los recorridos de colectivos nos encontramos con inconvenientes en la conversión del linestring a geometría postgis. Se solucionó usando un back up provisto por el tutor del grupo (¡gracias Lucas!).

5. Requerimientos

A continuación detallamos las consultas SQL que resuelven cada uno de los requerimientos junto con un ejemplo donde se aplicó cada una.

5.1. Paradas cercanas

- **Requerimiento:** *En base a una posición (dada como una geometría de tipo POINT), indicar las paradas cercanas de colectivo que se encuentran, devolviendo calle, altura y número de línea que tiene parada. La cercanía es un parámetro a definir de la consulta (por ejemplo, 100 metros).*
- **Consulta:**

```
SELECT DISTINCT p.calle, p.altura, cp.linea_colectivo
FROM paradas p inner join colectivos_por_parada cp on p.id_parada = cp.id_parada
WHERE ST_DistanceSphere(ST_MakePoint('[long_punto_deseado]', '[lat_punto_deseado]'), p.point_posicion) < [Distancia_Deseada];
```

Figura 5: Consulta paradas cercanas parametrizada.

Los parámetros de esta consulta son todos aquellos que están entre corchetes. En este caso dichos parámetros son la latitud y longitud del punto cuyas paradas cercanas se desean visualizar y el rango en el cual se buscarán las paradas (Distancia_Deseada).

- **Ejemplo:** Para mostrar la consulta en ejecución, se decide obtener aquellos colectivos que están a menos de 100 metros de la intersección entre Rivadavia y Avenida la plata.

```
SELECT DISTINCT p.calle, p.altura, cp.linea_colectivo
FROM paradas p inner join colectivos_por_parada cp on p.id_parada = cp.id_parada
WHERE ST_DistanceSphere(ST_MakePoint('-58.429268','-34.615175'), p.point_posicion) < 100
order by cp.linea_colectivo asc;
```

Figura 6: Ejemplo paradas cercanas.

Las coordanas usadas se obtuvieron de la aplicación ComoLLego[1] del gobierno de la ciudad.

[1] https://mapa.buenosaires.gob.ar/comollego

	calle text	altura text	linea_colectivo integer
1	RIVADAVIA AV.	4485	5
2	RIVADAVIA AV.	4485	8
3	LA PLATA AV.	46	15
4	RIO DE JANEIRO	20	15
5	RIVADAVIA AV.	4451	26
6	LA PLATA AV.	50	65
7	RIO DE JANEIRO	20	65
8	LA PLATA AV.	46	85
9	RIVADAVIA AV.	4485	86
10	RIVADAVIA AV.	4467	88
11	RIVADAVIA AV.	4451	103
12	RIVADAVIA AV.	4461	104
13	LA PLATA AV.	50	112

Figura 7: Resultado del ejemplo.

Algunas líneas de colectivo aparecen más de una vez porque para la dirección que se usó se encuentran dentro del rango de consulta tanto el sentido de ida como el de vuelta de dichos colectivos.

5.2. Colectivo a utilizar

- **Requerimiento:** *En base a dos posiciones (dadas como geometrías de tipo POINT), una de origen y otra de destino, indicar qué colectivos llevan de una ubicación a otra. Se deberá devolver el número de línea de colectivo, el ramal y el sentido, los datos de la parada de origen y destino (calle y número para ambas). Un parámetro de la consulta será cuánto está dispuesto a caminar como máximo la persona, incluyendo una primer caminata entre el punto de origen y la parada de origen y una segunda caminata entre la parada de destino y el punto de destino. Es decisión del grupo si este límite será total (la suma de ambas caminatas no puede superar el límite) o particular (ninguna caminata puede superar el límite).*

- **Consulta:**

```
SELECT DISTINCT cp.linea_colectivo, rc.recorrido as ramal, rc.sentido, p.calle as calle_parada_ida, p.altura,
                p2.calle as calle_parada_destino, p2.altura
FROM paradas p INNER JOIN colectivos_por_parada cp ON p.id_parada = cp.id_parada
     INNER JOIN recorrido_colectivos rc ON cp.linea_colectivo = rc.linea
     INNER JOIN (paradas p2 INNER JOIN colectivos_por_parada cp2 ON p2.id_parada = cp2.id_parada
     INNER JOIN recorrido_colectivos rc2 ON cp2.linea_colectivo = rc2.linea) ON
     cp.linea_colectivo = cp2.linea_colectivo AND rc.recorrido = rc2.recorrido AND rc.sentido = rc2.sentido
WHERE ST_DistanceSphere(ST_MakePoint('[longitud origen]','[latitud origen]'), p.point_posicion) < [Max caminata inicial]
AND ST_DistanceSphere(ST_MakePoint('[longitud destino]','[latitud destino]'), p2.point_posicion) < [Max caminata final]
AND st_line_Locate_Point(ST_lineMerge(rc.geom), p.point_posicion) <= st_line_Locate_Point(rc2.geom,p2.point_posicion)
```

Figura 8: Consulta colectivo a utilizar parametrizada.

Los parámetros de la consulta son los que figuran entre corchetes. Se deben especificar tanto la latitud y la longitud de las posiciones de partida y de destino y además la longitud máxima

deseada de la caminata tanto en la ida, como en la vuelta.

- **Ejemplo:** Tomamos como punto de partida la calle Santander a la altura del 1900 y queremos ver qué colectivos nos llevan a Entre Ríos y Caseros realizando, en la ida, una caminata menor a 250 metros y al llegar de, a lo sumo, 100 metros.

```
SELECT DISTINCT cp.linea_colectivo, rc.recorrido as ramal, rc.sentido, p.calle as calle_parada_ida, p.altura,
                p2.calle as calle_parada_destino, p2.altura
FROM paradas p INNER JOIN colectivos_por_parada cp ON p.id_parada = cp.id_parada
     INNER JOIN recorrido_colectivos rc ON cp.linea_colectivo = rc.linea
     INNER JOIN (paradas p2 INNER JOIN colectivos_por_parada cp2 ON p2.id_parada = cp2.id_parada
     INNER JOIN recorrido_colectivos rc2 ON cp2.linea_colectivo = rc2.linea) ON
     cp.linea_colectivo = cp2.linea_colectivo AND rc.recorrido = rc2.recorrido AND rc.sentido = rc2.sentido
WHERE ST_DistanceSphere(ST_MakePoint('-58.447040','-34.638835'), p.point_posicion) < 250
AND ST_DistanceSphere(ST_MakePoint('-58.390408','-34.634161'), p2.point_posicion) < 100
AND st_line_Locate_Point(ST_lineMerge(rc.geom), p.point_posicion) <= st_line_Locate_Point(rc2.geom,p2.point_posicion)
```

Figura 9: Ejemplo colectivo a utilizar.

	linea_colectivo integer	ramal character varying (5)	sentido character varying (10)	calle_parada_ida text	altura text	calle_parada_destino text	altura text
1	50	A	VUELTA	CARABOBO AV.	1438	CASEROS AV.	2061
2	50	A	VUELTA	CARABOBO AV.	1438	ENTRE RIOS AV.	2168
3	50	A	VUELTA	DIAZ. AVELINO	2118	CASEROS AV.	2061
4	50	A	VUELTA	DIAZ. AVELINO	2118	ENTRE RIOS AV.	2168
5	50	B	VUELTA	CARABOBO AV.	1438	CASEROS AV.	2061
6	50	B	VUELTA	CARABOBO AV.	1438	ENTRE RIOS AV.	2168
7	50	B	VUELTA	DIAZ. AVELINO	2118	CASEROS AV.	2061
8	50	B	VUELTA	DIAZ. AVELINO	2118	ENTRE RIOS AV.	2168
9	133	A	VUELTA	CARABOBO AV.	1212	ENTRE RIOS AV.	2168
10	133	A	VUELTA	CARABOBO AV.	1399	ENTRE RIOS AV.	2168
11	133	A	VUELTA	CARABOBO AV.	1438	ENTRE RIOS AV.	2168
12	133	B	VUELTA	CARABOBO AV.	1212	ENTRE RIOS AV.	2168
13	133	B	VUELTA	CARABOBO AV.	1399	ENTRE RIOS AV.	2168

Figura 10: Resultado del ejemplo.

- Hay 3 resultados más que no pudimos mostrar, pero que al hacer la consulta se ven.-

Se puede apreciar que hay colectivos que aparecen más de una vez. Esto se debe a que, para el recorrido dado, 2 o más ramales se superponen o bien porque para un mismo colectivo, hay más de una parada válida que entra en el rango de la caminata seleccionado.

5.3. Ramales de parada

- **Requerimiento:** *En base a una línea de colectivo (dada como número de colectivo), analizar qué paradas del mismo no son recorridas por todos los ramales del colectivo. Devolver calle, número de la parada y nombre del ramal que pasa por la parada, únicamente para aquellas paradas que no sean recorridas por todos los ramales de la línea.*

- **Consulta:** En la solución de esta consulta solo se tuvieron en cuenta las paradas y sus recorridos, por lo que para determinar las paradas por las que no pasaban todos los recorridos solo fue necesario usar la tabla *colectivos_por_parada*. Además para mostrar la información se usa la tabla paradas. Se considera el número de parada a mostrar como el identificador único de parada (id_parada).

```
WITH paradas_colectivo AS (
    SELECT DISTINCT id_parada, recorrido
    FROM  colectivos_por_parada
    WHERE linea_colectivo = [Numero_de_colectivo]
)
SELECT p.calle, pc.id_parada, pc.recorrido
FROM paradas p
INNER JOIN paradas_colectivo pc ON pc.id_parada = p.id_parada
INNER JOIN (SELECT DISTINCT pc.id_parada
            FROM paradas_colectivo pc
            WHERE EXISTS((SELECT 1 FROM paradas_colectivo pc2
                        WHERE NOT EXISTS (SELECT 1 FROM paradas_colectivo pc3
                                          WHERE pc.id_parada = pc3.id_parada
                                          and pc2.recorrido = pc3.recorrido))
)) AS R on pc.id_parada = R.id_parada
```

Figura 11: Consulta ramales de parada parametrizada.

El parámetro de la consulta es la que figura entre corchetes, se deben especificar el número de la línea de colectivo.

- **Ejemplo:** Para el ejemplo usamos la línea de colectivo 2.

```
WITH paradas_colectivo AS (
    SELECT DISTINCT id_parada, recorrido
    FROM  colectivos_por_parada
    WHERE linea_colectivo = 2
)
SELECT p.calle, pc.id_parada, pc.recorrido
FROM paradas p
INNER JOIN paradas_colectivo pc ON pc.id_parada = p.id_parada
INNER JOIN (SELECT DISTINCT pc.id_parada
            FROM paradas_colectivo pc
            WHERE EXISTS((SELECT 1 FROM paradas_colectivo pc2
                        WHERE NOT EXISTS (SELECT 1 FROM paradas_colectivo pc3
                                          WHERE pc.id_parada = pc3.id_parada
                                          and pc2.recorrido = pc3.recorrido))
)) AS R on pc.id_parada = R.id_parada
```

Figura 12: Ejemplo ramales de parada utilizar.

	calle text	id_parada integer	recorrido character varying (5)
1	ESPAÑA AV.	1006951	B
2	ESPAÑA AV.	1006956	B
3	ESPAÑA AV.	1006953	B
4	ESPAÑA AV.	1006957	B
5	ESPAÑA AV.	1006955	B
6	ESPAÑA AV.	1006958	B
7	ESPAÑA AV.	1006954	B
8	ESPAÑA AV.	1006950	B
9	ESPAÑA AV.	1006952	B

Figura 13: Resultado del ejemplo.

De la salida se observa aquellas paradas por las que no pasan todos los ramales de la línea 2, para verificar este resultado analizando un poco los datos y recurriendo a otras fuentes (omnilineas[2]), la línea 2 tiene dos ramales A y B, comparten todas las paradas hasta Av. Ingeniero Huergo y Moreno donde termina el primero y el recorrido B continua por Av. España en el barrio de Puerto Madero.

Se muestra la salida en puntos geográficos de las paradas:

Figura 14: Ubicación geográfica de las paradas.

5.4. Combinación de colectivos

- **Requerimiento:** *En base a dos posiciones (dadas como geometrías de tipo POINT), una de origen y otra de destino, indicar qué par de colectivos llevan de una ubicación a otra. Se deberán devolver los números de línea de colectivo, ramales y sentido, los datos de la parada de origen y destino (calle y número para ambas). Nuevamente existe el parámetro de máximo a caminar, pero esta vez también se debe considerar la distancia entre el bajar del primer colectivo y tomar el segundo.*
- **Consulta:** La consulta devuelve la parada inicial y la final con la información correspondiente, ordenado de forma ascendiente la distancia total del recorrido.

[2] https://www.omnilineas.com.ar/buenos-aires/colectivo/linea-2/

```
select distinct
      -- primer tramo
      pi.id_parada as id_parada_ini,
      pi.calle as calle_parada_ini,
      pi.altura as altura_parada_ini,
      cpi.linea_colectivo as linea_ini,
      cpi.sentido as sentido_ini,
      cpi.recorrido as recorrido_ini,
      -- segundo tramo
      pf.id_parada as id_parada_fin,
      pf.calle as calle_parada_fin,
      pf.altura as altura_parada_fin,
      cpf.linea_colectivo as linea_fin,
      cpf.sentido as sentido_fin,
      cpf.recorrido as recorrido_fin,
      -- long recorrido total recorrido
        st_length(ST_LineSubstring(rci.geom,st_lineLocatePoint(rci.geom,pi.point_posicion),
                            st_lineLocatePoint(rci.geom, ST_ClosestPoint(rci.geom,rcf.geom)))::Geography)
            +
        st_length(ST_LineSubstring(rcf.geom,st_lineLocatePoint(rcf.geom, ST_ClosestPoint(rci.geom,rcf.geom)),
                                   st_lineLocatePoint(rcf.geom,pf.point_posicion))::Geography)
            +
        st_distance((rci.geom)::Geography,(rcf.geom)::Geography)
        as long_total_recorrido
from paradas pi, paradas pf,
    colectivos_por_parada cpi, recorrido_colectivos rci,
    colectivos_por_parada cpf, recorrido_colectivos rcf
where
       --paradas iniciales cercanas a x metros de la posición inicial
       ST_DWITHIN(ST_MakePoint(['Long_Ini'],['Lat_Ini'])::Geography, pi.point_posicion,'Max_Distancia') and
       --paradas finales cercanas a x metros de la posición final
       ST_DWITHIN(ST_MakePoint(['Long_Fin'],['Lat_Fin'])::Geography, pf.point_posicion,'Max_Distancia')
       --info primer tramo
       and pi.id_parada = cpi.id_parada
       and cpi.linea_colectivo = rci.linea and cpi.sentido = rci.sentido and cpi.recorrido = rci.recorrido
       --info segundo tramo
       and pf.id_parada = cpf.id_parada
       and cpf.linea_colectivo = rcf.linea and cpf.sentido = rcf.sentido and cpf.recorrido = rcf.recorrido
       -- condición de sentido
       and st_lineLocatePoint(rci.geom,pi.point_posicion) <=
       st_lineLocatePoint(rci.geom, ST_ClosestPoint(rci.geom,rcf.geom))
       and st_lineLocatePoint(rcf.geom, ST_ClosestPoint(rci.geom,rcf.geom)) <=
       st_lineLocatePoint(rcf.geom,pf.point_posicion)
       -- elimino las lineas que viajan directo
       and cpi.linea_colectivo != cpf.linea_colectivo
       -- condición que la distancia entre los recorridos no supere x metros
       and st_distance((rci.geom)::Geography,(rcf.geom)::Geography) < 'Max_Distancia'
order by long_total_recorrido;
```

Figura 15: Consulta combinación de colectivos parametrizada.

Para realizar la consulta se usaron la tabla de paradas, colectivos_por_parada y la de recorrido_colectivos, tanto para la parada y recorrido inicial como para el final. Se tuvo en cuenta que la parada esté cercana a la posicion de origen/destino, que los recorridos respeten el sentido y que la distancia entre ambos recorridos no supere la caminata máxima.

- **Ejemplo:** Se muestra un ejemplo con dos paradas específicas y se pide que la caminata entre los recorridos no supere los 300 metros.

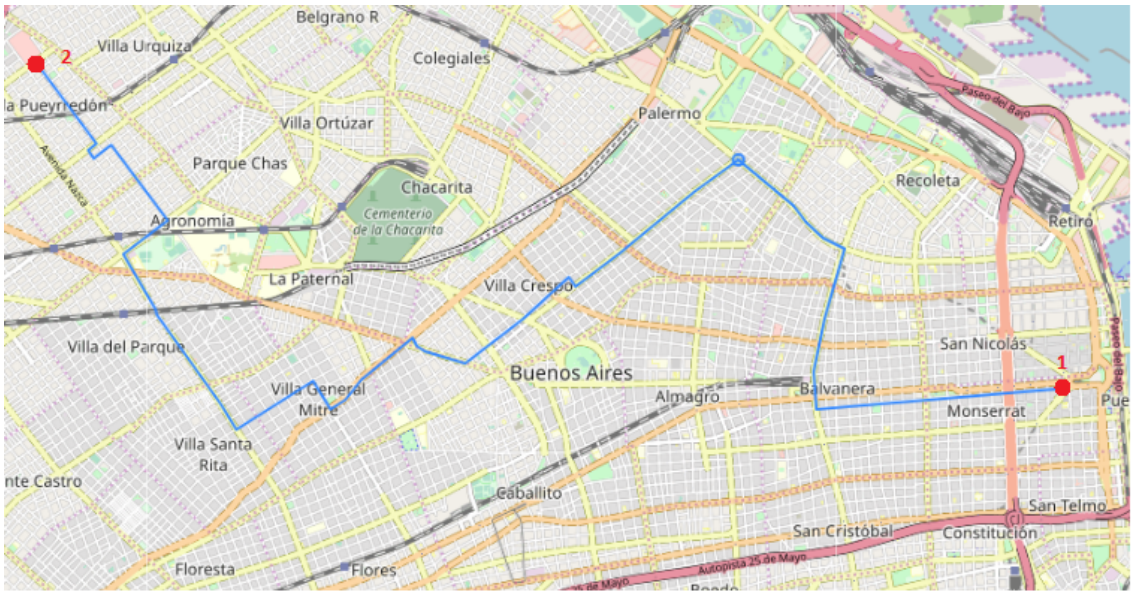

Figura 16: Ejemplo combinación de colectivos.

Data Output | Notifications | Query History | Explain | Messages | Geometry Viewer

	id_parada_ini integer	calle_parada_ini text	altura_parada_ini text	linea_ini integer	sentido_ini character varying (10)	recorrido_ini character varying (5)
1	1000046	YRIGOYEN. HIPOLI...	636	64	VUELTA	C
2	1000046	YRIGOYEN. HIPOLI...	636	64	VUELTA	D

id_parada_fin integer	calle_parada_fin text	altura_parada_fin text	linea_fin integer	sentido_fin character varying (10)	recorrido_fin character varying (5)
1004730	ARTIGAS. JOSE GE...	5781	110	IDA	B
1004730	ARTIGAS. JOSE GE...	5781	110	IDA	B

Figura 17: Resultado del ejemplo.

6. Análisis de performance

Se tuvieron inconvenientes en la ejecución de la query del requirimiento 4: *çombinación de colectivos"*. En la construcción de la misma, se inició probando con paradas específicas para conceptualizar el esquema de la solución de manera sencilla. Al pasar a la consulta coordenas tipo POINT, la consulta demoraraba en dar resultados.

Para tener como referencia, en una computadora de especificaciones normales se tuvo que detener la consulta porque demoraba más de 20 minutos en ejecutarse. En esta primera instancia se usaba la función ST_DistanceSphere() para calcular la distancia entre la coordenada y la parada.

Luego de investigar por la world wide web [3], se cambió la función por ST_DWithin() y la consulta se ejecutó en promedio de 2 minutos. También se intentó generar índices en la tabla de paradas por el 'point_posicion' y en la de recorrido_colectivos por "geom"pero no se logró optimizar los tiempos de ejecución.

[3] https://gis.stackexchange.com/questions/247113/how-to-properly-set-up-indexes-for-postgis-distance-queries/247131

7. Conclusiones

Con el desarrollo de este trabajo práctico entendimos el valor del pre-procesamiento de la información fuente y de la importancia que tiene a la hora de modelar soluciones. Si tuviésemos que indicar que tarea nos conllevó la mayor parte del tiempo, eligiriámos sin dudar la carga de los archivos. Esto se refleja en los ámbitos reales de trabajo ya que al depender de información externa, uno tiene que modelarla para ajustarse a las necesidades que tiene.

Destacamos, también, el valor del modelo entidad-interrelación y de las formas normales. Dichas herramientas proveen una forma de presentar los problemas de manera üniversal", haciendo más facil la comunicación de la solución y la construcción de la misma.

Si bien el uso de datos espaciales en una base de datos es un dominio específico, consideramos el conocimiento adquirido de alta utilidad, tanto para el ámbito académico como el profesional.

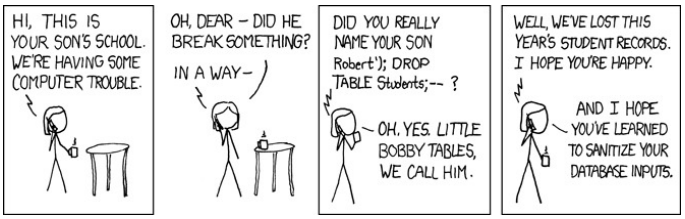

Figura 18: El día a día en el manejo de datos