RAPPORT DE PROJET : CITY MAPPER

Sous la direction de :

Nabil H. Mustafa

Réalisé par :

JOSEPH Darryll Genève Junior
FRARMA Yannis
KOUNOUHO Kpessou Jermiel

1. Rappel du sujet

L'objectif dans ce projet très intéressant est de confectionner une application pour les transports publics qui utilise une base de données PostgreSQL pour la gestion des données. Autrement dit, il s'agira de créer notre propre City Mapper. Pour développer un tel outil des données détaillées sur les opérations du réseau de transport public seront nécessaires voire indispensable.

2. Fonctionnalités utilisées

Plusieurs fonctionnalités nous ont été très utile dans le cadre de la réalisation de ce projet. Au nombre de celles-ci, nous énumérons les suivantes :

- L'utilisateur a la possibilité de choisir directement le lieu de départ et le lieu d'arrivée en cliquant directement sur la carte

- L'utilisateur peut aussi entrer directement dans le tableau le lieu de départ et le lieu d'arrivée voulu.

- L'utilisateur à la possibilité de faire jusqu'à quatre correspondances pour atteindre la destination souhaité

3. Diagramme Relations-Entités (E-R)

Pour confectionner efficacement notre application, nous sommes allés puiser dans la collection organisée des réseaux de transport public de 25 différentes villes dans plusieurs formats faciles d'utilisation. Dans cette collection et comme cible la ville de Paris, nous avons pris en compte les listes *Paris_routeI_routeName_routeType* contenant les différents types de transport ; *Network_combined* qui quantifie les distances moyennes en fonction des transports, les véhicules ; *Network_nodes* montrant la valeurs de la longitude et de la latitude de chaque différent point de la ville de Paris ; Stops quant à elle donne tous les arrêts possible et susceptibles d'être considéré comme un départ ou une destination.

Avec cette mine de données la prochaine étape dans notre projet est de parvenir à conceptualiser son contenu afin de discuter et de comprendre la relation entre transports, distances, données de géolocalisation, positions. C'est ainsi qu'entre en jeu le diagramme ER ci-dessous :

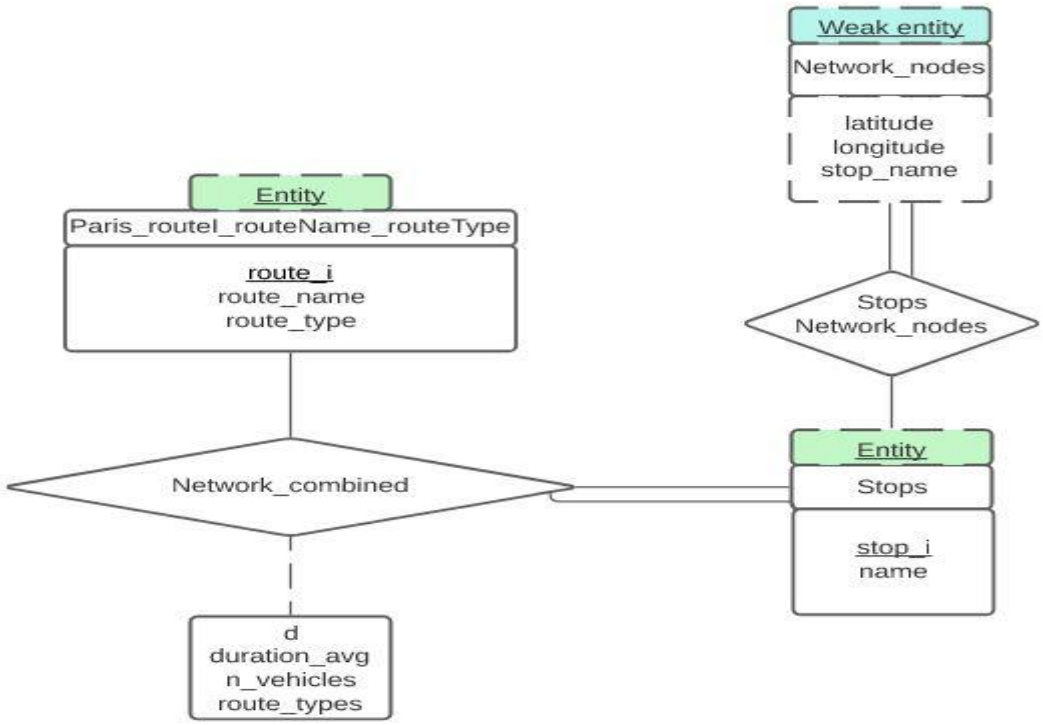


Diagramme entité relation

4. Différents étapes pour la création du diagramme ER

Comme nous pouvons le remarquer, notre diagramme se compose de trois entités dont une de type faible et deux relations. Ces dernières décrivent très bien comment nos trois entités interagissent entre elles.

Deux étapes majeures ont été nécessaire pour conceptualiser toutes les données à notre disposition :

a- Suppression des attributs doubles :

A l'exemple de la table *network_nodes*, une entité faible, nous avons supprimé l'attribut *stop_i* . En effet, il apparait déjà dans la table stops de type entité.

Le statut d'entité faible pour la table network_nodes confère aux attributs *longitude*, *latitude* et *stop_name* le titre de clés partielles.

b- Définition des relations

La table *network_combined* contient les clés primaires des tables *paris_routei_routename_routetype* et *stops*. Par conséquent, elle devient une relation à laquelle on ajoute les attributs *d*, *duration_avg*, *n_vehicles*, *route_types*

La table *network_nodes* à la clé primaire de la table stops, elle devient une entité faible.

Les tables *stops* et *paris_routei_routename_routetype* restent des entités.

5. Description des tables

Ci-après se trouve un schéma de chacune de nos tables, leurs attributs ainsi que leurs types.

Create table **network_nodes** (
stop_i integer ,
latitude numeric (21,17),
longitude numeric (21,17),
stop_name text,
PRIMARY KEY (stop_i),
FOREIGN KEY (stop_i) REFERENCES stops) ;

Create table **stops** (
Stop_i integer,
Name text ,
PRIMARY KEY (stop_i));

Create table **network_combined** (
from_stop_i integer ,
to_stop_i integer,
d integer,
duration_avg text,
n_vehicles integer,
route_i_counts integer,
routes_type integer ,
PRIMARY KEY (from_stop_i , to_stop_i , route_i_counts),
FOREIGN KEY (route_i_counts) REFERENCES
paris_routeI_routeName_routeType),
FOREIGN KEY (from_stop_i , to_stop_i) REFERENCES stops);

Create table paris_routeI_routeName_routeType (
route_i integer,
route_name text,
route_type integer,
PRIMARY KEY (route_i));

6. Liste des dépendances

a- Dans la table paris_routeI_routeName_routeType on a la dépendance suivante :

Stop_i → latitude, longitude, stop_name

b- Dans la table Network_combined on a la dépendance suivante :

From_stop_i, to_stop_i, route_i_counts → d, n_vehicles, duration_avg, routes_type

c- Dans la table Stops on a la dépendance suivante :

Stop_i → name

d- Dans la table Network_nodes on a la dépendance suivante :

Route_i → route_name, route_type

7. Forme BCNF ou 3NF

- Commençons par la table **Stops** :

Stop_i → name

Soit R (Stop_i, name),

Soit F l'ensemble des fonctions de dépendances :
F = { Stop_i → name}

Vérifions si R est BCNF :

Calculons la cloture de stop_i :
$\{Stop\_i\}^+ = \{Stop\_i\}$

Vu qu'on a: Stop_i → name, $\{Stop\_i\}^+ = \{Stop\_i, name\}$, donc R est inclus dans $\{Stop\_i\}^+$ et par conséquent, R est BCNF.

- Prenons la table **Network_combined** :

From_stop_i, to_stop_i, route_i_counts → d, n_vehicles, duration_avg, routes_type

Soit R1 (From_stop_i, to_stop_i, n, duration_avg , d, n_vehicles, route_i_counts, routes_type)

Soit F l'ensemble des fonctions de dépendances

F = { From_stop_i, to_stop_i, route_i_counts → d, n_vehicles, duration_avg, routes_type}

Vérifions si R1 est BCNF :

Calculons la cloture de From_stop_i, to_stop_i, route_i_counts :

{ From_stop_i, to_stop_i, route_i_counts }$^+$ = { From_stop_i, to_stop_i, route_i_counts}

Vu qu'on a From_stop_i, to_stop_i, route_i_counts → d, n_vehicles, duration_avg, routes_type,

{ From_stop_i, to_stop_i, route_i_counts }$^+$ = { From_stop_i, to_stop_i, route_i_counts, d, n_vehicles, duration_avg, routes_type}.

Donc R1 est inclus dans { From_stop_i, to_stop_i, route_i_counts }$^+$ et par conséquent, R1 est BCNF.

- En ce qui concerne la table **Network_nodes** :

  Route_i → route_name, route_type

Soit R2 (Route_i ,route_name, route_type),

Soit F l'ensemble des fonctions de dépendances

{ Route_i → route_name, route_type}

Vérifions si R2 est BCNF :

Calculons la cloture de Route_i :

{Route_i}$^+$ = { Route_i}

Vu qu'on a Route_i → route_name, route_type,

{Route_i}$^+$ = { Route_i ,route_name, route_type }, donc R2 est inclus dans {Route_i}$^+$ et par conséquent, R2 est BCNF.

- Quant à la table paris_routeI_routeName_routeType, on a :
  Stop_i → latitude, longitude, stop_name

Soit R3 (Stop_i, latitude, longitude, stop_name)

Soit F l'ensemble des fonctions de dépendances

{ Stop_i → latitude, longitude, stop_name}

Vérifions si R3 est BCNF :

Calculons la cloture de stop_i :

$\{Stop\_i\}^+ = \{stop\_i\}$

Vu qu'on a Stop_i → latitude, longitude, stop_name,

$\{Stop\_i\}^+ = \{stop\_i, latitude, longitude, stop\_name\}$, donc R3 est inclus dans $\{stop\_i\}^+$ et par conséquent, R3 est BCNF.

En somme, R, R1, R2, R3 sont aussi en troisième forme normale (3NF) car d'une part elles sont en BCNF et d'autre part il n'existe pas d'attributs non clés qui dépendent d'attribut non clés .

8. Requêtes SQL associées à chaque fonctionnalité

```
def connect_DB(self):
    self.conn = psycopg2.connect(database="ProjetBDD", user="DarYanJer", host="localhost", password="projet")
    self.cursor = self.conn.cursor()

    self.cursor.execute("""SELECT distinct name FROM network_combined,stops,paris_routei_routename_routetype WHERE network_combined.from_stop_i =
    stops.stop_i ORDER BY name""")
    self.conn.commit()
    rows = self.cursor.fetchall()

    for row in rows :
        self.from_box.addItem(str(row[0]))
        self.to_box.addItem(str(row[0]))
```

Grâce au module Psycopg2, l'utilisateur se connecte à la base de données. Ensuite l'interface de recherche lui est affiché avec la liste de tous les points susceptibles d'être un point de départ ou une destination.

```
if _hops >= 1 :
    self.cursor.execute(""f" SELECT distinct A.name, A.route_name, B.name
                        FROM (SELECT * FROM network_combined,stops,paris_routei_routename_routetype
                            WHERE network_combined.from_stop_i = stops.stop_i
                            AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts) AS A,
                        (SELECT * FROM network_combined,stops,paris_routei_routename_routetype
                            WHERE network_combined.to_stop_i = stops.stop_i
                            AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts) AS B
                        WHERE A.name= $${_fromstation}$$ AND B.name= $${_tostation}$$ AND A.route_i_counts = B.route_i_counts""")
    self.conn.commit()
    self.rows += self.cursor.fetchall()
```

Dans la figure ci-dessus, nous avons la requête SQL associé au cas où l'utilisateur souhaite faire une seule correspondance (si elle existe) entre son point de départ et sa destination.

```
if _hops >= 2 :
    self.cursor.execute(""f" SELECT distinct A.name, A.route_name, B.name, C.route_name,D.name
                            FROM (SELECT *
                                  FROM network_combined,stops,paris_routei_routename_routetype
                                  WHERE network_combined.from_stop_i = stops.stop_i
                                  AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts) AS A,
                                  (SELECT *
                                  FROM network_combined,stops,paris_routei_routename_routetype
                                  WHERE network_combined.from_stop_i = stops.stop_i
                                   AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts) AS B,
                                  (SELECT * FROM network_combined,stops,paris_routei_routename_routetype
                                  WHERE network_combined.from_stop_i = stops.stop_i
                                  AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts) AS C,
                                  (SELECT * FROM network_combined,stops,paris_routei_routename_routetype
                                   WHERE network_combined.to_stop_i = stops.stop_i
                                  AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts ) AS D
                            WHERE A.name = $${_fromstation}$$
                            AND D.name = $${_tostation}$$
                            AND A.route_i_counts = B.route_i_counts
                            AND B.name = C.name  AND C.route_i_counts = D.route_i_counts
                            AND A.route_i_counts <> C.route_i_counts
                            AND A.name <> B.name
                            AND B.name <> D.name""")
```

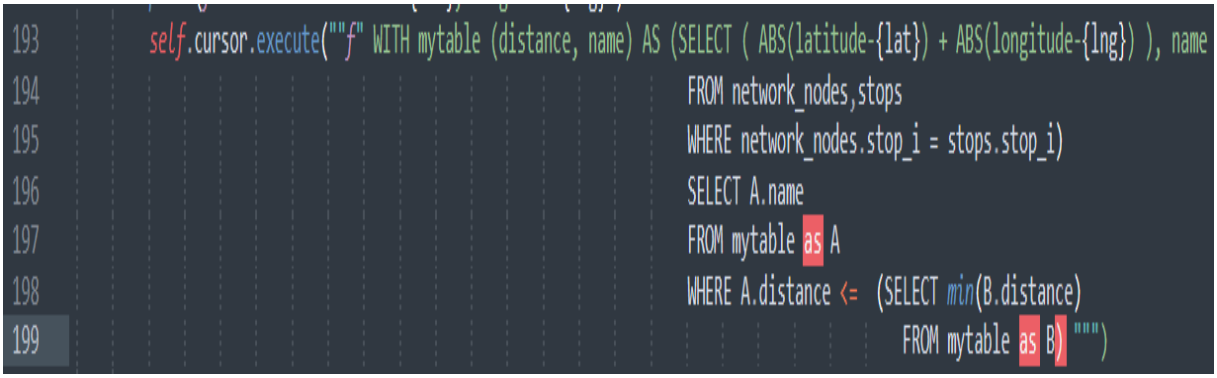
Le schéma ci-dessus quant à lui prend en compte la volonté de l'utilisateur de faire deux correspondance au maximum entre son point de départ et sa destination.

```
if _hops >= 3 :
    self.cursor.execute(""f" SELECT distinct A.name, A.route_name, B2.name, B2.route_name, C2.name, C2.route_name, D.name
                        FROM (SELECT *FROM network_combined,stops,paris_routei_routename_routetype
                            WHERE network_combined.from_stop_i = stops.stop_i
                            AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts) AS A,
                        (SELECT *FROM network_combined,stops,paris_routei_routename_routetype
                            WHERE network_combined.from_stop_i = stops.stop_i
                            AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts) AS B1,
                        (SELECT *FROM network_combined,stops,paris_routei_routename_routetype
                            WHERE network_combined.from_stop_i = stops.stop_i
                            AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts) AS B2,
                        (SELECT *FROM network_combined,stops,paris_routei_routename_routetype
                            WHERE network_combined.from_stop_i = stops.stop_i
                            AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts) AS C1,
                        (SELECT *FROM network_combined,stops,paris_routei_routename_routetype
                            WHERE network_combined.from_stop_i = stops.stop_i
                            AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts) AS C2,
                        (SELECT *FROM network_combined,stops,paris_routei_routename_routetype
                            WHERE network_combined.to_stop_i = stops.stop_i
                            AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts) AS D
                        WHERE A.name = $${_fromstation}$$ AND A.route_i_counts = B1.route_i_counts AND B1.name = B2.name
                        AND B2.route_i_counts = C1.route_i_counts AND C1.name = C2.name AND C2.route_i_counts = D.route_i_counts
                        AND D.name = $${_tostation}$$ AND A.route_i_counts <> B2.route_i_counts AND B2.route_i_counts <> C2.route_i_counts
                        AND A.route_i_counts <> C2.route_i_counts AND A.name <> B1.name AND B2.name <> C1.name AND C2.name <> D.name""")
    self.conn.commit()
    self.rows += self.cursor.fetchall()
```

La requête SQL intégrer dans le code ci-dessus, répond au cas où l'utilisateur aurait besoin de 3 correspondances maximum

```
if _hops >= 4 :
    self.cursor.execute(""f" SELECT distinct A.name, A.route_name, B2.name, B2.route_name, C2.name, C2.route_name, D2.name, D2.route_name, D.name
                        FROM (SELECT *
                            FROM network_combined,stops,paris_routei_routename_routetype
                            WHERE network_combined.from_stop_i = stops.stop_i
                              AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts
                            ) AS A,
                            (SELECT *
                             FROM network_combined,stops,paris_routei_routename_routetype
                             WHERE network_combined.from_stop_i = stops.stop_i
                             AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts
                             ) AS B1,
                             (SELECT *
                             FROM network_combined,stops,paris_routei_routename_routetype
                             WHERE network_combined.from_stop_i = stops.stop_i
                             AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts
                             ) AS B2,
                             (SELECT *
                             FROM network_combined,stops,paris_routei_routename_routetype
                              WHERE network_combined.from_stop_i = stops.stop_i
                               AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts
                             ) AS C1,
                             (SELECT *
                              FROM network_combined,stops,paris_routei_routename_routetype
                              WHERE network_combined.from_stop_i = stops.stop_i
                                AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts
                             ) AS C2,
                             (SELECT *
                              FROM network_combined,stops,paris_routei_routename_routetype
                              WHERE network_combined.from_stop_i = stops.stop_i
                               AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts
                             ) AS D1,
                             (SELECT *
                             FROM network_combined,stops,paris_routei_routename_routetype
                             WHERE network_combined.from_stop_i = stops.stop_i
                               AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts
                             ) AS D2,
                              (SELECT *
                              FROM network_combined,stops,paris_routei_routename_routetype
                              WHERE network_combined.to_stop_i = stops.stop_i
                               AND paris_routei_routename_routetype.route_i = network_combined.route_i_counts
                              ) AS D
                            WHERE A.name = $${_fromstation}$$
                             AND A.route_i_counts = B1.route_i_counts AND  B2.route_i_counts = C1.route_i_counts
                             AND C2.route_i_counts = D1.route_i_counts AND D2.route_i_counts = D.route_i_counts
                             AND  B1.name = B2.name  AND C1.name = C2.name AND D1.name = D2.name AND A.name <> B1.name
                             AND B2.name <> C1.name AND C2.name <> D1.name AND D2.name <> D.name AND A.route_i_counts <> B2.route_i_counts
                             AND B2.route_i_counts <> C2.route_i_counts AND C2.route_i_counts <> D2.route_i_counts
                             AND A.route_i_counts <> D2.route_i_counts AND A.route_i_counts <> C2.route_i_counts AND D.name = $${_tostation}$$ """)
    self.conn.commit()
```

La requête SQL intégrer dans le code précédent affiche au maximum 4 correspondances entre le point de départ et la destination souhaité.



```
193     self.cursor.execute(""f" WITH mytable (distance, name) AS (SELECT ( ABS(latitude-{lat}) + ABS(longitude-{lng}) ), name
194                                                 FROM network_nodes,stops
195                                                 WHERE network_nodes.stop_i = stops.stop_i)
196                                                 SELECT A.name
197                                                 FROM mytable as A
198                                                 WHERE A.distance <= (SELECT min(B.distance)
199                                                                     FROM mytable as B) """)
```

La requête ci-dessus permet de récupérer la longitude et la latitude d'un point à partir des stop_names quand l'utilisateur clique sur la carte.

Remarques :

Dans le but de rendre plus pratique aux utilisateurs, notre application, nous avions ajouté à notre cahier des charges, la possibilité d'afficher et de proposer de façon ordonnée les chemins avec la plus petite durée pour joindre deux points différents. Autrement dit, nous avions jugé bon de mettre en pratique les notions acquises en Algorithme des graphes afin d'afficher à l'utilisateur les plus courts chemins reliant son point de départ et sa destination.

Outre cette mesure, la mise en place d'une icône cliquable pour l'ouverture de l'interface graphique nous a paru très intéressant à mettre en place toujours pour rendre la vie plus facile aux utilisateurs.

Mais malheureusement nous n'avions eu le temps que pour mettre en place le plus important des fonctionnalités.

9. Difficultés rencontrées

- Le temps pour rajouter la table network_temporal_weeks dans la base de données était très considérable au vu du nombre important d'informations se trouvant dans cette table.

- Il nous était difficile d'ajouter la table section dans la base de donnée car certaines colonnes contenaient des caractères tels que des apostrophes. Nous avons dû remplacer ces dernières par des espaces, pour une utilisation plus facile de cette table dans la base de données ;

- Nous avions aussi du mal à extraire les identifiants des routes dans la liste route_I_counts de la table network_combined. Mais quelques recherches, la fonction split intégré dans python nous a permis de séparer la liste en fonction des délimiteurs tels que les virgules et les deux points « : ».

10. Contributions au projet

Pour mener à bien ce projet, la contribution de chaque membre du groupe était importante. En effet,

La rédaction d'un rapport clair, concret, synthétique suivant le plan donné par le professeur était aux soins de **Jermiel**. Ajouter à cela il s'est occupé de la mise en place de la base de données devant servir au projet.

Darryll s'est chargé de la mise en place des requêtes sql nécessaire pour le bon fonctionnement de notre application.

A l'instar de Darryll en back-end, **Yannis** à pris en charge la rédaction du code python nécessaire à l'analyse et au traitement des données dans la base de données.