BASE DE DONNÉES

CITYMAPPER

Joys Nkayem, Lina Chenni, Ninah Solofoniaina

1. Présentation du Sujet:

Mise en place d'une application de gestion du réseau de transport de la ville de Kuopio en Finlande. Le but du projet est de faire un citymapper qui va permettre aux utilisateurs de rentrer une adresse de départ et d'arrivée et ensuite d'obtenir une liste de routes qu'ils peuvent emprunter. Nous n' avons eu que des données de bus comme moyen de transport pour cette ville. Pour l'exécuter : python3 mapskuopio.py avec le compte PostgreSQL : l3info_53.

2. Langage utilisé:

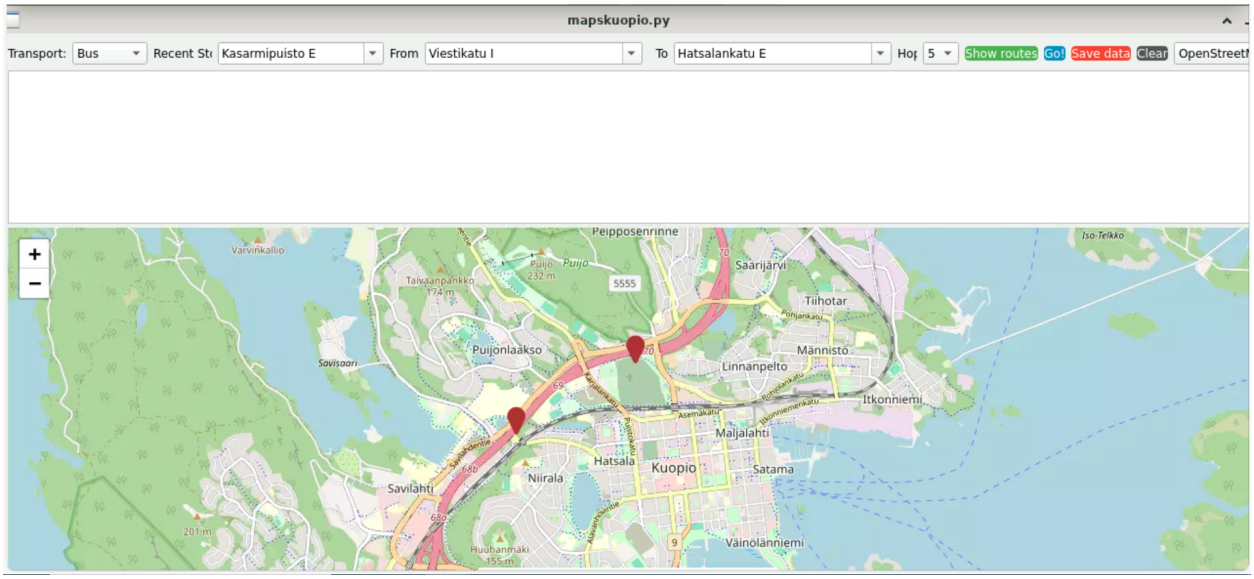
Pour la mise en place de ce projet, nous avons utilisé le langage de programmation orientée objet Python et le langage SQL pour les requêtes sur la base de données.

Sommaire

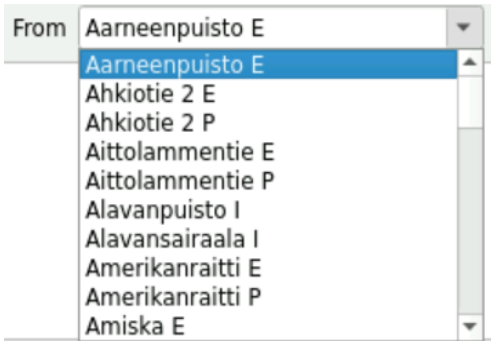
1) Les fonctionnalités de l’application

Pour ce projet nous avons choisi la ville de Kuopio en Finlande.

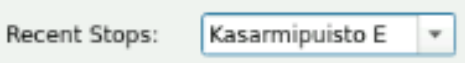
Au lancement de l’application, on trouve sur l’interface un plan de la ville de Kuopio, un panneau de contrôle ainsi qu’un tableau vide.



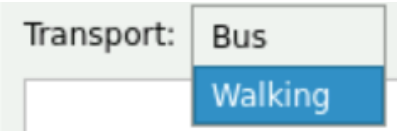
L’utilisateur peut choisir les lieux de départ et d’arrivée directement sur le plan en cliquant dessus ou bien en les sélectionnant via les listes déroulantes « From » et « To » présentes dans le panneau de contrôle.



On peut avoir une liste des arrêts récemment sélectionnés de from et to.



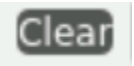
Il peut choisir sa recherche via les modes de transports suivants qui sont présents dans la ville : bus ou à pied.



La case « Hops » lui permet de choisir le nombre de correspondances pour son trajet. Pour notre cas, vu que Kuopio est une ville moyenne voire petite, c'est mieux de mettre hops>=5;

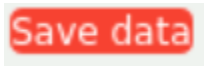
Lorsqu'on clique sur la case « Go ! » l'itinéraire est affiché dans le tableau suivant de la durée du trajet.

La case « Clear » permet de supprimer les marqueurs qui ont été faits sur la carte.

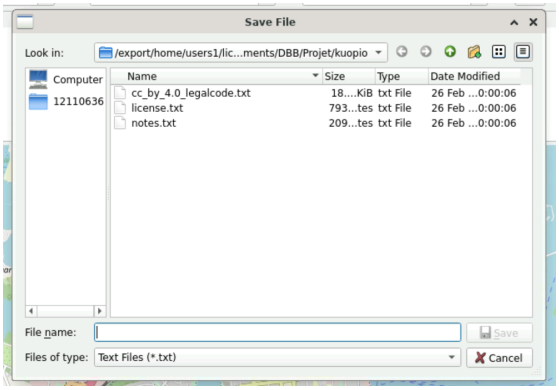


En plus des transports possibles, on peut avoir les routes qui passent par les arrêts sélectionnés car chaque arrêt a surement une histoire (monument historique).

Le bouton « Save data » permet d'enregistrer les données affichées sur le tableau ; comme cela, les personnes qui se connectent peuvent enregistrer tout pour avoir une trace



et cela s'affiche :



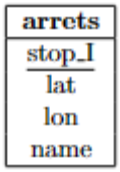
On peut choisir le type de carte**.**

2) Les tables et leurs descriptions

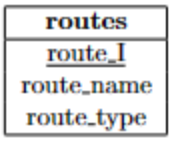
newcheminscombines
from_stop_I
to_stop_I
d
duration_avg
n_vehicles
route_I_counts
route_I
route_type

- **cheminscombines** qu'on a transformé en **newcheminscombines** car on a séparé les route_I_count en route_I : contenant les informations de déplacements entre deux arrêts (from et to), avec la distance (d), le temps moyen du trajet (duration_avg), le nombre de bus qui passe par cet intervalle (n_vehicles), chemin

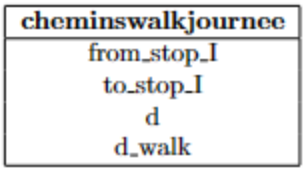
emprunté (route_I) et le type de moyen de transport utilisé (route_type) (pour nous ici ce sera toujours 3 pour bus).



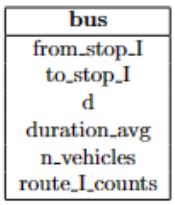
- **arrets :** contenant les informations sur les arrêts de bus dont un identifiant unique (stop_I), sa latitude (lat), sa longitude (lon) et le nom de l'arrêt (name).



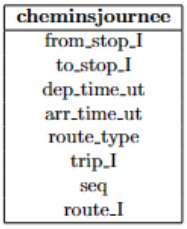
- **routes** : contenant les informations sur les routes empruntées par les bus dont le nom de la route (route_name), un identifiant unique(route_I) et route_type qui est toujours 3 dans notre cas.

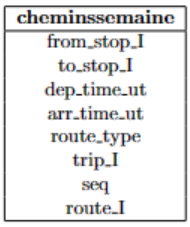


- **cheminswalkjournee** : contenant les informations entre deux arrêts (from et to) pour un trajet à pied. On y retrouve la distance brute (d) ainsi que la distance à pied (d_walk).



- **bus :** même table que newcheminscombines sans route_type.





- **cheminssemaine** / **cheminsjournee** : contenant les informations entre deux arrêts (from et to) pour la semaine et pour la journée et ont des identifiants uniques d'un trajet (trip_I). On y retrouve l'heure de départ (dep_time_ut), l'heure d'arrivée (arr_time_ut), le type de transport qui l'emprunte (route_type), son id de passage (seq) et l'identifiant des routes empruntées (route_I).

- **recherches** : cette table a été rajoutée juste pour stocker les arrêts récemment sélectionnés ; elle contient un id qui s'auto incrémente et le nom de l'arrêt.

En résumé, les principales tables qu'on a utilisées sont arrets, routes, newcheminscombines qui vient de cheminscombines et cheminswalkjournee.

3) Les dépendances entre attributs

arrets : avec **PRIMARY KEY (stop_I)**

stop_I → lat, lon, name

routes : avec **PRIMARY KEY (route_I)**

route_l → route_name, route_type

newchemincombines : avec **PRIMARY KEY (from_stop_l, to_stop_l, route_l)**

from_stop_l, to_stop_l, route_l → d, duration_avg, n_vehicles, route_type, route_l_counts

route_l → route_type

cheminswalkjournee : avec **FOREIGN KEY (from_stop_l) REFERENCES arrets(stop_l), FOREIGN KEY (to_stop_l) REFERENCES arrets(stop_l)**

from_stop_l, to_stop_l → d, d_walk

4) BCNF, 3NF?

- **arrets**(stop_l,name,lat, lon) :

BCNF? oui car stop_l est une super clé de arrets et les attributs name, lat, lon dépendent tous de stop_l.

Si on fait la clausure de {lat, lon}+, on verra que c'est aussi une super clé.

Explication:

lat, lon → stop_l, name est pas triviale

{lat, lon}+ ={ stop_l, name}

stop_l appartient à {lat, lon}+ alors {lat, lon}+= {stop_l,lat, lon, name, stop_l}

C’est une BCNF et une 3NF car {lat, lon et stop_I} sont des super clé et stop_I est une clé candidate qui se trouve à droite.

- **newchemincombines** (from_stop_I, to_stop_I, route_I, d, duration_avg, n_vehicles, route_type, route_I_counts) :

BCNF? oui car avec from_stop_I, to_stop_I, route_I → d, duration_avg, n_vehicles, route_type, route_I_counts, la partie gauche de la dépendance fonctionnelle détermine de manière unique tous les attributs de la partie droite de la dépendance.

3NF? oui car c’est BCNF.

- **routes** (route_I, route_name, route_type) :

BCNF? oui avec route_I → route_name, route_type, car c’est une dépendance à 2 attributs. Comme dans notre cas route_type est toujours égal à 3, on peut avoir route_I → route_name c’est-à-dire toujours BCNF.

3NF? oui car BCNF.

- **cheminswalkjournee** (from_stop_I, to_stop_I,d, d_walk,route_I, route_type) :

BCNF? oui avec from_stop_I, to_stop_I → d, d_walk, la partie gauche de la dépendance fonctionnelle détermine de manière unique tous les attributs de la partie droite de la dépendance.

3NF? oui car BCNF.

5) Les requêtes SQL

Nous n'avons pas mis toutes les requêtes de hops car elles sont similaires.

```
self.cursor.execute("""SELECT name FROM arrets ORDER BY name""")
```

Cette requête permet d'afficher les noms des arrêts par ordre alphabétique.

```
self.cursor.execute("SELECT DISTINCT stop_name FROM recherches")
```

Cette requête permet d'afficher les noms des arrêts récemment sélectionnés.

```
self.cursor.execute(f"""
    SELECT DISTINCT route.route_name
    FROM newcheminscombines AS A
    JOIN arrets AS depart ON depart.stop_I = A.from_stop_I
    JOIN arrets AS arrivee ON arrivee.stop_I = A.to_stop_I
    JOIN routes AS route ON route.route_i = A.route_i
    WHERE (depart.name = %s OR arrivee.name = %s)
""", (_fromstation, _tostation))
```

Cette requête permet d'afficher toutes les routes passant par les arrêts sélectionnés.

```
        self.cursor.execute(f"""
         SELECT DISTINCT A.lat, A.lon, A.name, routes.route_name, B.lat, B.lon, B.name,
ncc.duration_avg
         FROM arrets AS A, arrets AS B, routes, newcheminscombines AS ncc
         WHERE A.name = %s AND B.name = %s AND ncc.from_stop_I = A.stop_I AND ncc.to_stop_I =
B.stop_I AND ncc.route_I = routes.route_I;
    """, (_fromstation, _tostation))
        self.conn.commit()
        self.rows += self.cursor.fetchall()


      if _hops >= 2:
        self.cursor.execute(f"""
         SELECT DISTINCT A.lat, A.lon, A.name, r1.route_name, B.lat, B.lon, B.name, r2.route_name,
C.lat, C.lon, C.name, (ncc1.duration_avg + ncc2.duration_avg) AS total_duration_avg
         FROM arrets AS A
         JOIN newcheminscombines AS ncc1 ON ncc1.from_stop_I = A.stop_I
         JOIN arrets AS B ON ncc1.to_stop_I = B.stop_I
         JOIN newcheminscombines AS ncc2 ON ncc2.from_stop_I = ncc1.to_stop_I
         JOIN arrets AS C ON ncc2.to_stop_I = C.stop_I
         JOIN routes AS r1 ON ncc1.route_I = r1.route_I
         JOIN routes AS r2 ON ncc2.route_I = r2.route_I
         WHERE A.name = %s AND C.name = %s;
         """, (_fromstation, _tostation))
```

Ces requêtes permettent de trouver une ou plusieurs routes reliant 2 arrêts suivies de leur durée de trajet en se basant sur newcheminscombines. On a utilisé les jointures pour faire plus simple.

PS : Nous avons affiché la durée du trajet jusqu'à hops >= 3 car au-delà de 3, nous avons eu des erreurs que l'on arrivait pas à comprendre alors que c'est la même logique.

```
        self.cursor.execute(f"""
         SELECT A.lat AS lat_A, A.lon AS lon_A, A.name AS name_depart,
                C.d_walk AS d_walk,
                B.lat AS lat_B, B.lon AS lon_B, B.name AS name_arrivee
         FROM cheminswalkjournee AS C
         JOIN arrets AS A ON A.stop_I = C.from_stop_I
         JOIN arrets AS B ON B.stop_I = C.to_stop_I
         WHERE A.name = %s AND B.name = %s;
""", (_fromstation, _tostation))
        self.conn.commit()
        self.rows += self.cursor.fetchall()

       if _hops >= 2 :
        self.cursor.execute("""
         SELECT A.lat AS lat_A, A.lon AS lon_A, A.name AS name_depart,
                d1.d_walk AS d1_walk,
                C.lat AS lat_C, C.lon AS lon_C, C.name,
                d2.d_walk AS d2_walk,
                B.lat AS lat_B, B.lon AS lon_B, B.name AS name_arrivee
         FROM cheminswalkjournee AS d1
         JOIN cheminswalkjournee AS d2 ON d1.to_stop_I = d2.from_stop_I
         JOIN arrets AS A ON d1.from_stop_I = A.stop_I
         JOIN arrets AS B ON d2.to_stop_I = B.stop_I
         JOIN cheminswalkjournee AS d3 ON d2.to_stop_I = d3.from_stop_I
         JOIN arrets AS C ON d3.to_stop_I = C.stop_I
         WHERE A.name = %s AND B.name = %s
""", (_fromstation, _tostation))
```

: même principe que celle du dessus mais en regardant la distance à pied.

```
      self.cursor.execute(f"WITH mytable (distance , name) AS (SELECT (ABS(lat - {lat}) + ABS(lon - {lng})), name FROM arrets)
SELECT A.name FROM mytable as A WHERE A.distance <= (SELECT min(B.distance) FROM mytable as B)")
```

Cette requête permet en cliquant sur la carte, de récupérer l'arrêt le plus proche de là où l'on clique.

```
    self.cursor.execute("INSERT INTO recherches (stop_name) VALUES (%s)", (selected_stop_name,))
```

Cette requête permet d'insérer les arrêts sélectionnés de from et/ou to dans la table recherches.

6) Les difficultés rencontrées

1) **Compilation**

Deux membres du groupe ont rencontré des difficultés à afficher la map sur leurs PC :

Malgré plusieurs tentatives, les deux membres n'ont pas réussi à résoudre les erreurs affichées. Étant sur Guacamole et non nos propres PC, il n'a pas pu télécharger les paquets nécessaires. Même en recommençant le projet à zéro, la partie SQL fonctionnait bien mais une erreur persistait lors de la compilation du fichier mapskuopio.py et en recommençant aussi les TP, cela affichait des erreurs lors de la compilation. Nous avons même tenté d'utiliser une machine virtuelle mais cela nécessiterait beaucoup d'installation. Voilà l'erreur :

Mais finalement, le problème a été résolu car il fallait tout simplement changer de terminal sur guacamole.

Nous étions un peu perdues avec les data qu'on avait reçues, car même avec hops >= 5 certains arrêts ne sont pas reliés mais on avait dû vérifier avec les data à côté pour voir que nos requêtes marchent.

Vu que nous n'avons pas approfondi Python, ce projet a été un peu dur pour nous ; heureusement qu'il y a eu le TP5. Les requêtes pour les hops aussi étaient longues et compliquées à faire.

Nous avons aussi eu du mal avec le parsing du newcheminscombines car il fallait séparer les route_l_count.

2) **Organisation**

En ce qui concerne l'organisation, nous avons rencontré des problèmes au moment des vacances, car l'université était fermée. On s'est donné rendez-vous à l'extérieur de l'université et on n'avait pas toutes les mêmes disponibilités.

3) **GeoJson**

Pour récupérer les coordonnées de routes.geojson il fallait installer un paquet appelé geojson mais on ne savait pas comment s'y prendre, un camarade de classe qui a aussi rencontré ce problème et que M. Nabil Mustapha a aidé à résoudre, nous a aussi aidé à résoudre le nôtre.

7) La contribution de chacune

Nous nous sommes aidées du TP5 pour la base de notre interface graphique. Nous avons ajusté le TP5 à nos données et ce que l'on voulait.

Pour les tables, le choix des clés primaires et foreign key, les dépendances, nous avons toutes les 3 fait cela ensemble car c'était délicat et il fallait se mettre d'accord.

Lorsqu'on a fini la partie conception, on s'est partagé les parsing des fichiers .csv et si un membre rencontrait des problèmes, on l'aidait.

Chacune proposait une fonctionnalité et on faisait les requêtes ensemble et l'intégrait à notre code python jusqu'à avoir des résultats.

Ce rapport a été écrit par tous les membres du groupe au fur et à mesure de l'avancement du projet.

En résumé, chacune a donné son maximum pour ce projet et aussi ce projet nous a beaucoup appris à mieux comprendre les requêtes, nous familiariser avec Python et à améliorer notre organisation de groupe.