PROJET: Base de Données
Kuopio

L3 Info 2022-2023

Maricic Alexandre 11803731
Chen Kévin 11807217
Mokeddem Nassim 11922864

Tables des matières

Introduction

Afin de parvenir à la réalisation de ce projet nous avons commencé par prendre comme base de code la correction du TP5 déjà travaillé et l'a modifié en adéquation à ce qui était demandé.

Nous avons choisi de prendre Kuopio pour rendre la vie facile à nos ordinateurs puisque c'était celle qui prenait le moins de place 4,7MB pour 300MB Paris, le choix a été vite fait.

Ce projet consiste donc à l'implémentation d'un programme en python qui va à l'aide de fichiers CSV préalablement fourni, créer des requêtes SQL sur les chemins à prendre pour se déplacer d'un point A à un point B. C'est donc une sorte de simulation d'un Citymapper.

Pour avoir les data il faut faire ***python3 transport.py nodes.py walk.py***
Pour faire fonctionner transport.py il faut d'abord faire ***python3 route.py*** qui va faire le fichier route.csv qui est utilisée dans transport.

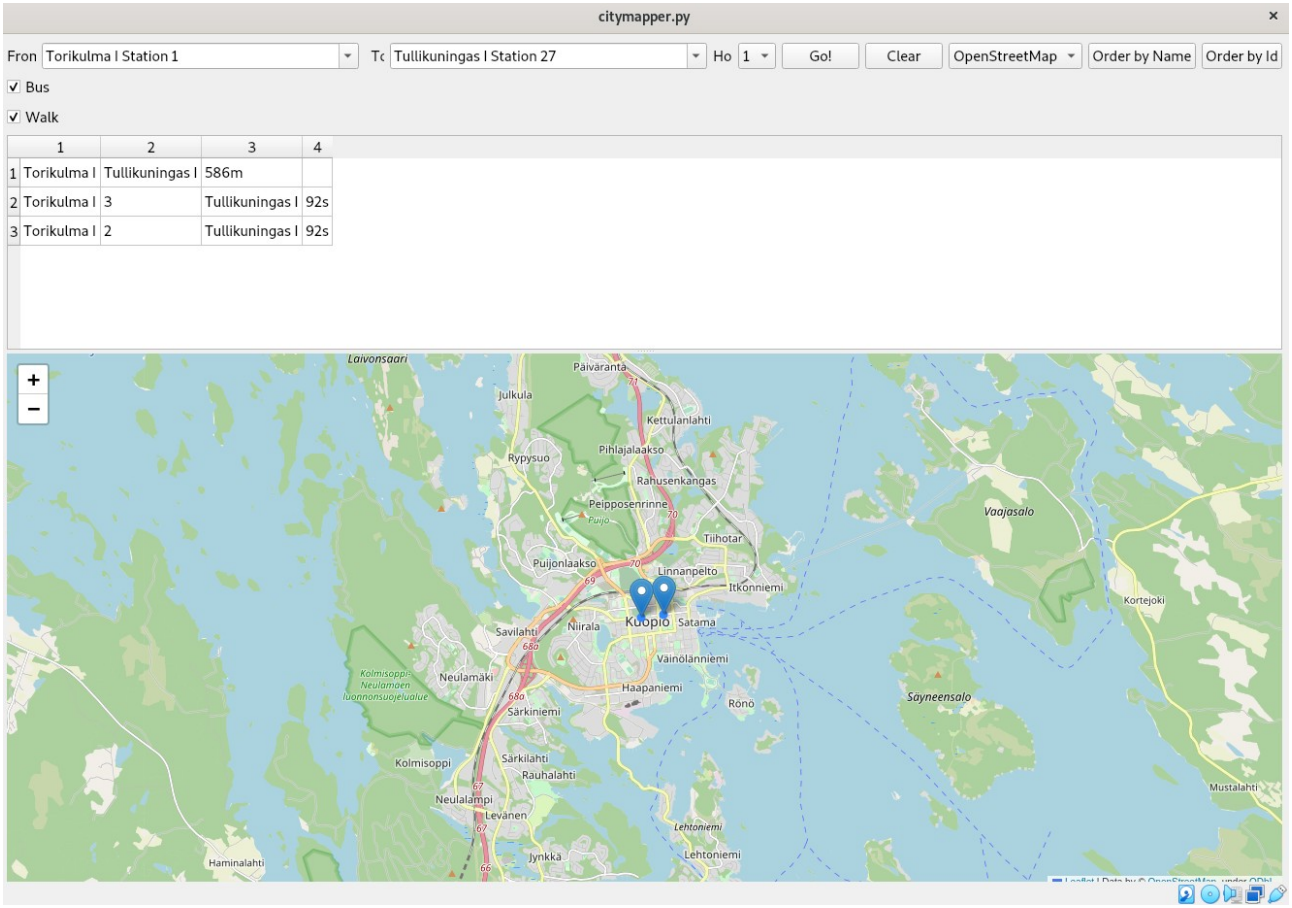
Pour que ça marche il faut bien évidemment se connecter à la base de données avant avec : ***psql -U l3info_23 -h 10.11.11.22 -d l3info_23***
avec comme mot de passe L3INFO_23 .
Une fois connecter il faut rajouter les tables dans la base de données (normalement cette étape est déjà faite dans notre cas) pour cela il faut commencer par les schema, puis les data.

\i nodes_createschema.sql
\i nodes_data.sql

Pareil pour les autres, une fois que c'est fait on peut lancer le citymapper.

En faisant ***python3 citymapper.py*** l'interface avec la map va se lancer tout en se connectant avec psycopg2 à la base de données l3info_23 .

Voici un visuel que nous obtenons en lançant notre programme:



Comme c’est une petite ville il peut arriver que parfois il n’y a rien qui s’affiche donc nous avons des point A et B précis que l’on utilise souvent pour vérifier la fonctionnalité du programme.

En voici quelques-unes trier par leur ID :

FROM :	TO :
193	393
1	27
74	24
193	15
48	29

Fonctionnalité

Nos codes se séparent en deux parties, tout d'abord nous avons la grosse partie que nous avons appelé citymapper,py qui reprend l'intégralité du TP5 qui s'occupe de faire le fonctionnement de l'interface avec les requêtes.

Dans ce long code de citymapper nous avons donc réglé la taille de l'interface à notre envie, et avons situé la map sur les coordonnées exactes de la ville grâce à sa longitude et latitude.

Nous avons rajouté des fonctionnalités en plus comme ORDER BY NAME et ORDER BY ID.

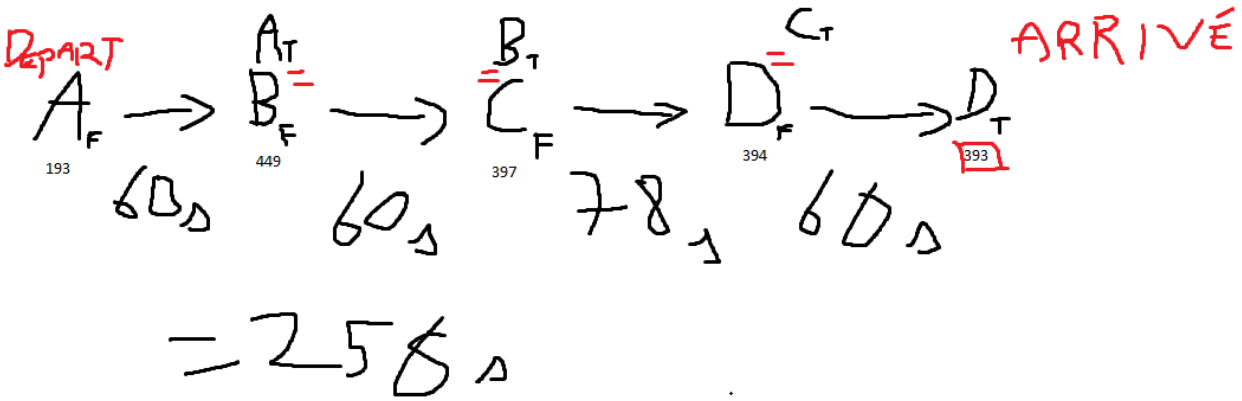
On a aussi rajouté un petit onglet bus à cocher si on veut que ce soit par bus ou autres moyen de transport pour arriver à destination.
Comme dans notre ville nous n'avons que le bus cela ne changera rien mais dans le cas d'une plus grande ville on aurait pu rajouter métro, tramway, train et autres.

Sur la toute fin on a réussi à mettre un onglet walk qui donne juste la distance à pied du point A au point B.

Si on ne coche aucun onglet rien ne s'affichera !

Le plus gros changement qu'on a effectué dans le code était surtout les requêtes dans hops, elles sont très longues et nous avons fait en sorte d'aller jusqu'à hops =>5.

Voici un petit schéma (certes moche) que nous avons fait pour mieux comprendre le fonctionnement des hops :



On a crée une fonction route.py à partir du code « create-routeI-routeName-routeTypefile.py » qui permet de créer le fichier route.csv qui va notamment nous permettre d'avoir le type de route.

Par la suite nous avons les 3 fichiers transport.py , nodes.py et walk.py qui ont été réaliser afin de pouvoir convertir les fichiers CSV en data SQL.
Comme leurs noms l'indiquent, ils se réfèrent aux mêmes fichiers du dossier Kuopio.

Pour ces 3 fonctions nous avons pris comme référence le code du prof du TP3.

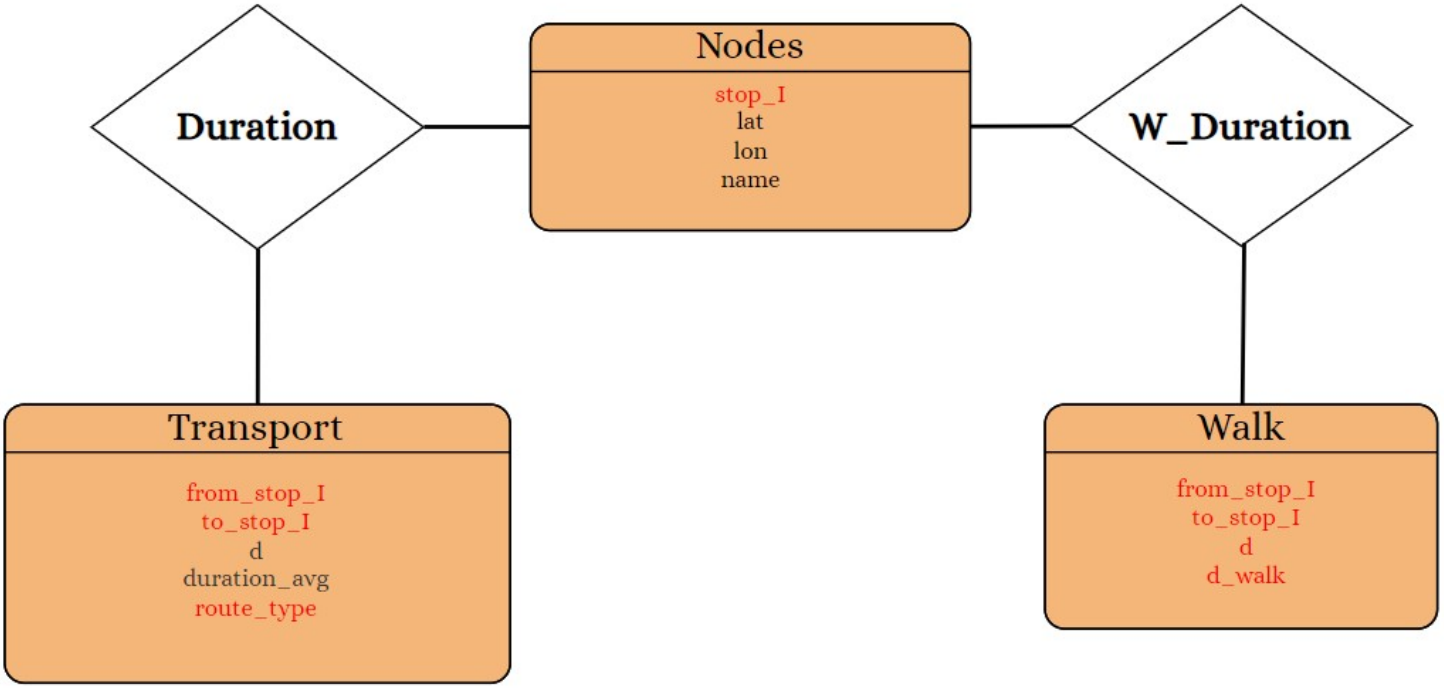
Kuopio étant une petite ville nous n'avons que les arrêts de bus.

Pour walk c'est le déplacement d'un point A à un point B avec sa distance entre les deux et la distance à pied entre les deux. Ce qui fait que la distance ≥ à la distance à pied.

Pour Nodes on donc son id, avec la latitude, longitude et le nom des stations. Ce qui nous le positionnement des arrêts.

Pour transport on utilise le fichier route.csv que l'on a créé au préalable avec notre fonction route.py et on utilise combined.csv qui est exactement le même que bus.csv. Ce qui nous intéresse est donc où on va, depuis où, la distance, la durée et le type de route.

Entity-Relationship diagram



Explication diagramme et liste de dépendance

Le diagramme est donc composé de 3 tableaux d'entité et de 2 relations.
Pour nodes nous avons stop_i qui est une superkey, en effet elle permet de caractériser d'elle seule le tableau.
Transport a pour superkey from_stop_i, to_stop_i, route_type.
Walk a tous ses attributs comme superkey, ils sont tous nécessaires afin de caractériser cette entité.

Les dépendance :

Nodes :
stop_I -> lat, long, name

Transport :
from_stop_I, to_stop_I, route_type -> d, duration_avg

Walk n'est pas soumis au dépendance.

Les tables

Voici les 3 tables dans notre base de donnée, obtenu en faisant :
SELECT * FROM name_table ;

```
projet=# \d
          List of relations
 Schema |   Name    | Type  | Owner
--------+-----------+-------+-------
 public | nodes     | table | alex
 public | transport | table | alex
 public | walk      | table | alex
(3 rows)
```

1) Nodes

stop_i	lat	lon	name
1	62.893025	27.676100	Torikulma I
2	62.893399	27.676660	Haapaniemenkatu E
3	62.892649	27.676809	Torikuja E
4	62.891996	27.677356	Kauppahalli P
5	62.893143	27.675906	Tullinkulma 1 L
6	62.893060	27.674469	Tullinkulma 2 L
7	62.894067	27.662708	Poliisiasema L
8	62.893176	27.659442	Mustinlammenkatu E
9	62.890739	27.654568	Umpikuja E
10	62.888562	27.653674	Tupatie E

2) Transport

from_stop_i	to_stop_i	d	d_walk
1	2	51	86
1	3	56	113
1	4	132	188
1	5	17	67
1	6	83	99
1	7	690	754
1	8	846	988
1	194	565	629
1	399	169	249
1	21	623	825

3) Walk

from_stop_i	to_stop_i	d	duration_avg	route_type
1	27	388	92	3
1	27	388	92	2
2	50	470	60	2
2	50	470	60	1
2	191	453	60	30
2	23	486	120	30
3	191	369	30	6X
3	191	369	30	6
3	191	369	30	5
3	191	369	30	5X

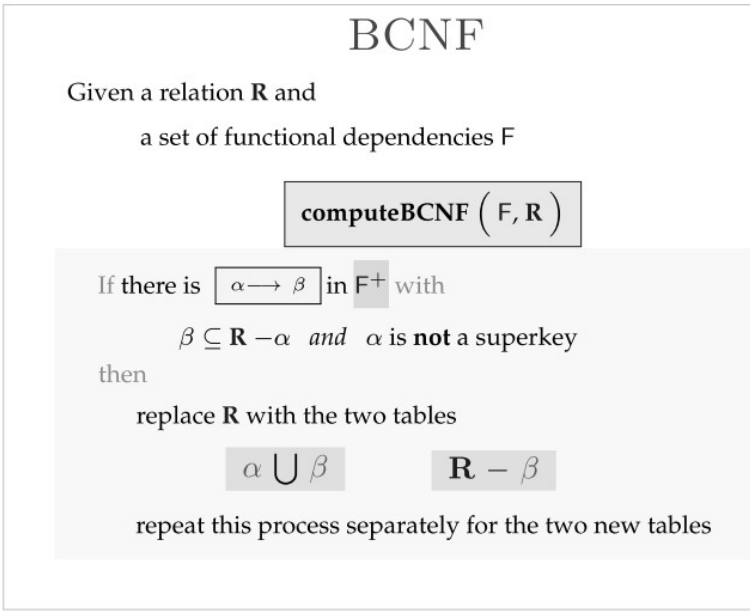
On a donc :

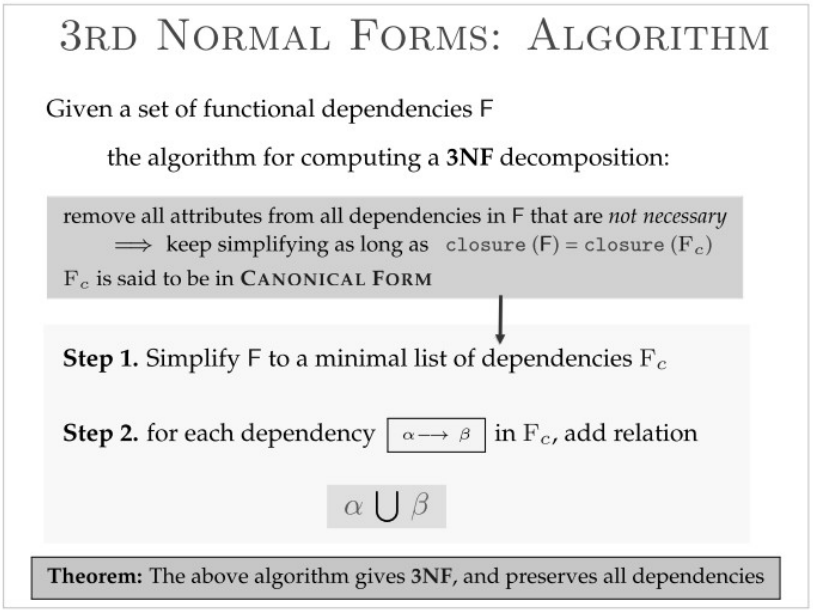
nodes(stop_i, lat, lon, name)
primary_key(stop_i)

transport(from_stop_i, to_stop_i, d, duration_avg, route_type)
primary_key(from_stop_i, to_stop_i, route_type)

walk(from_stop_i, to_stop_i, d, d_walk)
primary_key(from_stop_i, to_stop_i, d, d_walk)

Explication BCNF et 3NF forme





L’entité « Nodes » est dans la forme BCNF car l'attribut ‘‘stop_I’’ est une superclé et que du coup tous les autres attributs peuvent être retrouvés avec seulement ‘‘id’’.

L’entité « Transport » est dans la forme BCNF car les attributs ‘‘from_stop_I’’, ‘‘to_stop_I’’ ainsi que ‘‘route_type’’ forment ensemble une superclé du tableau.

Dans ce ER-Diagramme il n'y a pas de relation à la forme 3NF, en effet seule la forme BCNF est présente en l'enceinte de ce diagramme.

Requêtes SQL

Voici un exemple de ce que donne la requête de hops => 1 et de hops => 2.

```
projet=# WITH transport_1(from_stop_i, name_from, to_stop_i, name_stop, route_type, duration_avg) as(
            SELECT from_stop_i, node1.name, to_stop_i, node2.name, route_type, duration_avg
            FROM transport, nodes as node1, nodes as node2
            WHERE from_stop_i = node1.stop_I and to_stop_i = node2.stop_I)
        SELECT name_from, route_type, name_stop,(CONCAT(duration_avg, 's')) as duration_avg_tot
        FROM transport_1
        WHERE from_stop_i = 1 AND to_stop_i = 27
        ORDER BY duration_avg_tot;
  name_from   | route_type |    name_stop    | duration_avg_tot
--------------+------------+-----------------+------------------
 Torikulma I | 3          | Tullikuningas I | 92s
 Torikulma I | 2          | Tullikuningas I | 92s
(2 rows)
```

```
   name_from   | route_type |    name_stop    | route_type |  name_stop   | duration_avg_tot
---------------+------------+-----------------+------------+--------------+------------------
 Torikulma I | 3          | Tullikuningas I | 8A         | Työnkulma P | 151s
 Torikulma I | 3          | Tullikuningas I | 8          | Työnkulma P | 151s
 Torikulma I | 3          | Tullikuningas I | 2          | Työnkulma P | 151s
 Torikulma I | 3          | Tullikuningas I | 3          | Työnkulma P | 151s
 Torikulma I | 2          | Tullikuningas I | 8A         | Työnkulma P | 151s
 Torikulma I | 2          | Tullikuningas I | 8          | Työnkulma P | 151s
 Torikulma I | 2          | Tullikuningas I | 2          | Työnkulma P | 151s
 Torikulma I | 2          | Tullikuningas I | 3          | Työnkulma P | 151s
(8 rows)
```

Nous avons donc 5 grosses requêtes, jusqu'à hops => 5.

```sql
WITH transport_1(from_stop_i, name_from, to_stop_i,
name_stop,
     route_type, duration_avg) as (
    SELECT from_stop_i, node1.name, to_stop_i, node2.name,
           route_type, duration_avg
    FROM transport, nodes as node1, nodes as node2
    WHERE from_stop_i = node1.stop_I and
          to_stop_i = node2.stop_I)
SELECT name_from, route_type, name_stop,
(CONCAT(duration_avg, 's')) as duration_avg_tot
FROM transport_1
WHERE from_stop_i = 1 AND to_stop_i = 27
ORDER BY duration_avg_tot;
```

```
WITH transport_2(from_stop_i, name_from, to_stop_i,
name_stop,
     route_type, duration_avg) as(
    SELECT from_stop_i, node1.name, to_stop_i, node2.name,
           route_type,duration_avg
    FROM transport,nodes as node1,nodes as node2
    WHERE from_stop_i=node1.stop_I and
to_stop_i=node2.stop_I)
SELECT A.name_from, A.route_type, A.name_stop, B.route_type,
       B.name_stop,(CONCAT(A.duration_avg +
B.duration_avg,'s'))
       as duration_avg_tot
FROM transport_2 as A, transport_2 as B
WHERE A.from_stop_i = 1 AND B.to_stop_i = 29 AND
      A.to_stop_i = B.from_stop_i
ORDER BY duration_avg_tot;
```

```
WITH transport_3(from_stop_i, name_from, to_stop_i,
name_stop,
     route_type, duration_avg) as(
    SELECT from_stop_i, node1.name, to_stop_i, node2.name,
           route_type,duration_avg
    FROM transport,nodes as node1,nodes as node2
    WHERE from_stop_i=node1.stop_I and
to_stop_i=node2.stop_I)
SELECT A.name_from, A.route_type, A.name_stop,
       B.route_type, B.name_stop, C.route_type,
       C.name_stop, (CONCAT(A.duration_avg + B.duration_avg
+
       C.duration_avg, 's')) as duration_avg_tot
FROM transport_3 as A, transport_3 as B, transport_3 as C
WHERE A.from_stop_i = 193 AND C.to_stop_i = 198 AND
A.to_stop_i =
      B.from_stop_i AND B.to_stop_i = C.from_stop_i AND
      B.to_stop_i <> A.from_stop_i AND B.to_stop_i <>
C.to_stop_i
ORDER BY duration_avg_tot;
```

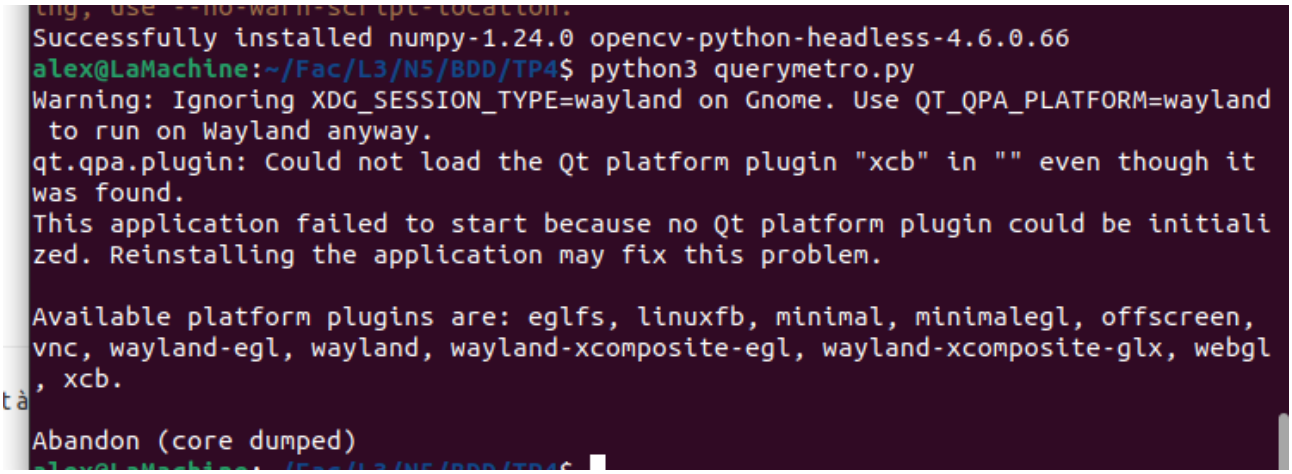
```
WITH transport_4(from_stop_i, name_from, to_stop_i,name_stop,
     route_type, duration_avg) as(
    SELECT from_stop_i, node1.name, to_stop_i, node2.name,
route_type, duration_avg
    FROM transport,nodes as node1,nodes as node2
    WHERE from_stop_i=node1.stop_I and
to_stop_i=node2.stop_I)
SELECT A.name_from, A.route_type, A.name_stop, B.route_type,
       B.name_stop, C.route_type, C.name_stop, D.route_type,
       D.name_stop, (CONCAT(A.duration_avg + B.duration_avg +
       C.duration_avg + D.duration_avg, 's')) as
duration_avg_tot
FROM transport_4 as A, transport_4 as B, transport_4 as C,
     transport_4 as D
WHERE A.from_stop_i = 193 AND D.to_stop_i = 394 AND
      A.to_stop_i = B.from_stop_i AND B.to_stop_i =
C.from_stop_i
      AND C.to_stop_i = D.from_stop_i AND B.to_stop_i <>
      A.from_stop_i AND B.to_stop_i <> C.to_stop_i AND
      A.from_stop_i <> C.to_stop_i
ORDER BY duration_avg_tot;
```

```
WITH transport_5(from_stop_i, name_from, to_stop_i,name_stop,
     route_type, duration_avg) as(
    SELECT from_stop_i, node1.name, to_stop_i, node2.name,
route_type, duration_avg
    FROM transport,nodes as node1,nodes as node2
    WHERE from_stop_i=node1.stop_I and
to_stop_i=node2.stop_I)
SELECT A.name_from, A.route_type, A.name_stop, B.route_type,
       B.name_stop, C.route_type, C.name_stop, D.route_type,
       D.name_stop, E.route_type, E.name_stop,
       (CONCAT(A.duration_avg + B.duration_avg +
C.duration_avg
       + D.duration_avg + E.duration_avg,'s')) as
duration_avg_tot
FROM transport_5 as A, transport_5 as B, transport_5 as C,
     transport_5 as D, transport_5 as E
WHERE A.from_stop_i = 193 AND E.to_stop_i = 194 AND
      A.to_stop_i = B.from_stop_i AND
      B.to_stop_i = C.from_stop_i AND
      C.to_stop_i = D.from_stop_i AND
      D.to_stop_i = E.from_stop_i AND
      B.to_stop_i <> A.from_stop_i AND
      B.to_stop_i <> C.to_stop_i AND
      A.from_stop_i <> C.to_stop_i AND
      B.to_stop_i <> D.from_stop_i AND
      A.to_stop_i <> D.from_stop_i
ORDER BY duration_avg_tot;
```

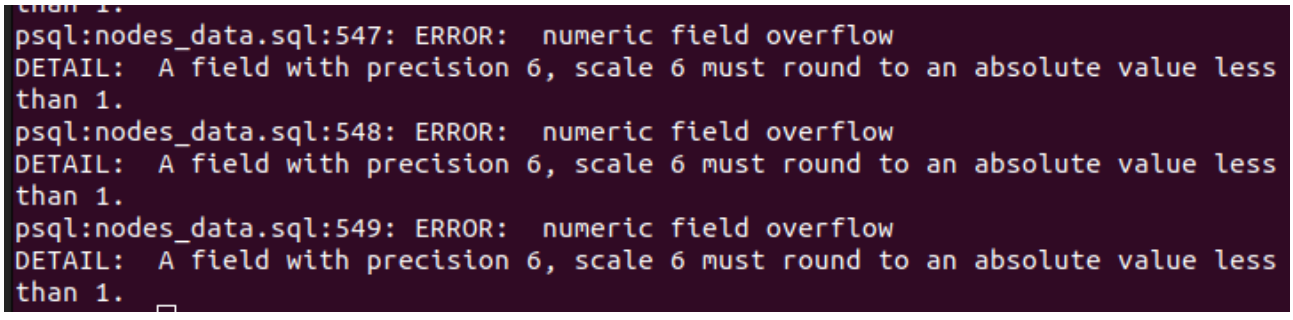
Les difficultés rencontrées

La toute première difficulté rencontrée est l'installation des logiciels python pour faire marcher la map. Même si ce n'était pas demandé nous avons voulu comprendre comment fonctionnaient ces logiciels.
Nous avons alors fait de nombreuses installations comme PyQT5, PyQtWebEngine …

Internet notre meilleur ami nous a donné des centaines de solutions que nous avons presque tout appliqué afin de réussir.
Cela nous à permis de comprendre un peu mieux le fonctionnement de python.
Malgré toutes ces installations seul l'ordinateur de Nassim arrivait à faire fonctionner la map.



Après ce problème réglé nous avons eu 2 3 petits soucis sur les schema, les valeurs que nous avons mises n'étaient pas assez grande. Ce fut un problème de quelques minutes on a rajouté +1 sur les valeurs jusqu'à son bon fonctionnement.



Une fois ces problèmes réglés nous avons eu beaucoup de mal avec la partie requête de « hops », à chaque fois que nous appuyons sur go l'interface crash et nous n'avions aucun tableau de requête qui s'affichait.

Ce fut notre plus gros problème durant ce projet nous avons relu tout notre code en long et en large afin de se rendre compte que ce fut un problème de nommage des entités.
Une fois que « hops >=1 » les autres se faisaient naturellement sans trop de soucis.

```
nassim@nassim:~/Telechargements/Projet$ python3 citymapper_pasfini.py
Traceback (most recent call last):
  File "/home/nassim/Téléchargements/Projet/citymapper_pasfini.py", line 137, i
n button_Go
    self.cursor.execute(""f"\
psycopg2.errors.InvalidTextRepresentation: ERREUR:  syntaxe en entrée invalide
pour le type numeric : « E »
LINE 1: ... as A, transport_1 AS B     WHERE A.from_stop_i = $$E$$ AND ...
                                                           ^
```

On a eu aussi quelques soucis avec la fonction transport il nous a fallu un certain temps pour comprendre comment utiliser le fichier route.csv que nous avons créee avec la fonction route.py.

Une grosse difficulté était aussi d'utiliser le python aucun de nous n'avait réellement fait du python avant nous avons donc dû revoir les bases du python. Heureusement que son fonctionnement se rapprochait grandement des autres langages que nous connaissions déjà.

Nous avons eu des problèmes à bien faire marcher les ORDER BY NAME et ID, qui permettent de filtrer les stations mais uniquement une fois, il fallait le faire dès le début avant de choisir une destination.
Maintenant il n'y a plus aucun problème ça marche parfaitement.

Problème avec l'onglet bus il se situait en dessous de la map avant, nous n'arrivions pas à le placer là où on le souhaite. Après quelques recherches nous avons trouvons la solution.

Nous n'avons pas réussi à faire une fonction time qui nous aurait permis de traduire le temps donné dans network_temporal_day et week. Afin de pouvoir prévoir le prochain temps du prochain trajet.
Nous pensions rajouter un onglet en plus (encore un oui) pour les prochains trajets mais le temps nous manquait et nous ne savions pas vraiment nous y prendre.

On a aussi un problème avec le click sur la map, pour une raison qu'on le ne comprend pas il arrivait qu'une fois qu'on clickait sur map l'interface crash et nous affichait ce message :

```
nassim@nassim:~/Téléchargements/Projet$ python3 citymapper.py
Clicked on: latitude 62.88833406169878, longitude 27.655334472656254
Traceback (most recent call last):
  File "/home/nassim/Téléchargements/Projet/citymapper.py", line 487, in javaScr
iptConsoleMessage
    self.parent.handleClick(msg)
  File "/home/nassim/Téléchargements/Projet/citymapper.py", line 386, in handleC
lick
    window.mouseClick(lat, lng)
  File "/home/nassim/Téléchargements/Projet/citymapper.py", line 315, in mouseCl
ick
    self.cursor.execute(""f" WITH mytable (distance, nom)\
AttributeError: 'builtin_function_or_method' object has no attribute 'execute'
Abandon
nassim@nassim:~/Téléchargements/Projet$
```

Ce problème ne devrait plus apparaître maintenant.

Après avoir fini l'ensemble du projet, à la toute fin nous avons voulu rajouter un onglet en plus walk qui serait faire le chemin à pied a la place du bus. Nous avions partiellement réussi, en effet quand on coche walk il nous affichait le résultat des bus avec.
Après un éclair de génie nous avons juste rajouté and self.bus.isChecked() dans la boucle if des hops et le problème fut réglé.

Une autre difficulté qui n'a rien à voir avec le code, faire un rapport de plus de 20 pages n'est pas quelque chose dont nous avons l'habitude, la réalisation de ce rapport nous a donc pris énormément de temps.

Les contributions de chacun

Pour commencer on à tous bien lu le sujet afin de bien le comprendre et on s'expliquait en quoi allait consister les démarches à suivre. Par la suite on s'est départagé un peu les taches selon les facilités et les difficultés de chacun. C'est ainsi que la fonction citymapper était une contribution commune de nous 3 en utilisant la base du TP5.

Dans cette même fonction Kévin s'est occupé de la partie requête pour « hops » c'est l'une des parties qui nous avaient posé le plus de problèmes. Il s'est aussi occupé de faire le diagramme Entity-Relationship ainsi que l'explication et la partie forme BCNF et 3NF.

La partie data c'est-à-dire la partie création des schema a été réaliser par Alexandre ainsi que la fonction walk et transport.

La majorité restante à faire du code de citymapper a été faite par Nassim ainsi que la fonction Nodes.

Transport, Nodes et Walk ont été faites ensemble, par Alexandre et Nassim. On par ailleurs voulut en faire une nouvelle « time » pour avoir accès au temps de passage mais nous n'avons pas réussi à la réaliser.

La fonction route.py qui permet de lire le fichier routes.geojson et de créer un fichier csv à été fait par Alexandre et Nassim, en prenant le code create-routel du prof comme base.

Nous avons tous les 3 travaillés ensemble et nous communiquions sur chaque fonction que nous faisions.

Conclusion

Ce projet nous a demandé une quantité de travail beaucoup plus élever que ce que nous pensions. On a appris grâce à ce projet le fonctionnement derrière les applications comme maps, cittymapper…

Un projet que nous n'aurions jamais réussi à faire de nous-mêmes si nous devions partir de 0, sans le code du prof que ce soit le TP5 ou le TP3. Nous avons grâce au projet acquis certaines connaissances et base du python mais aussi une meilleure vision de l'utilisation des requêtes SQL.
Ce projet était la parfaite manière d'illustrer le travail sur les bases de données nous montrant bien comment s'en servir en situation réel.

L'expérience retenue est excellente, nous sommes satisfaits de notre travail mais si certaines fonctions auquel nous pensions n'ont pas été abouti (notamment celle sur le temps).

Source

Pour le fonction de l’interface :

https://stackoverflow.com/questions/51154871/python-3-7-0-no-module-named-pyqt5-qtwebenginewidgets

https://www.youtube.com/watch?v=lUmS6XhuPLI

Pour le code python CSV → data.SQL :

https://docs.python.org/3/library/csv.html

Pour régler la map sur Kuopio :

https://latitudelongitude.org/fi/kuopio/

Pour une meilleur compréhension du code python :

https://courspython.com/introduction-python.html

https://courspython.com/dictionnaire.html

https://stackoverflow.com/questions/27652686/python-what-does-for-x-in-a1-mean

La référence utiliser pour mener à bien ce projet :

https://lipn.univ-paris13.fr/~mustafa/Teaching/USPN/G3IN4A-2022f/

Site où se trouve les fichiers de la ville de Kuopio :

https://zenodo.org/record/1186215