

## **Projet CEBD : Mission Climat**

### **1. Partie I : Conception**

#### **a. Normalisation**

##### **i. Question 1**

On écrit d'abord les dépendances fonctionnelles pour chaque relation.

#### **Zone climatique :**

- code\_département -> nom\_département, zone\_climatique
- nom\_département -> code\_département, zone\_climatique

Les clés sont {code\_département} ou {nom\_département}

Cette relation est déjà en BCNF.

#### **Mesures :**

- date\_obs, code\_insee\_département -> département, tmin, tmax, tmoy
- code\_insee\_département -> département
- département -> code\_insee\_département
- date\_obs, département -> code\_insee\_département, tmin, tmax, tmoy

Nous avons deux clés {date\_obs, code\_insee\_département} et {date\_obs, département}

Cette relation est déjà en BCNF car la seule transitivité que nous avons est entre des attributs clés.

#### **Commune :**

- département, code\_commune, code\_canton -> tout
- code\_département, code\_commune, code\_canton -> tout
- département -> région, code\_région, code\_département
- code\_département -> département, région, code\_région
- région -> code\_région
- code\_région -> région

On a deux clés, on prendra {code\_commune, code\_département, code\_canton}. Cette relation n'est pas BCNF car *région -> code\_région* et *code\_région -> région* et ce ne sont pas des attributs clés donc on normalise.

***R1(code\_région, région)*** est BCNF

***R2(code\_commune, code\_département, code\_canton, altitude moyenne, code\_région, superficie, population, commune, département, code\_arrondissement)***

R2 n'est pas en BCNF car *departement* -> *region*, *code\_region*, *code\_departement* et *code\_departement* est attribut clé à droite. Elle est une relation en 1NF.

*code\_commune*, *code\_departement* -> *altitude moyenne*, *superficie*, *population*, *commune*, *departement*, *code arrondissement*. *code\_region* dépend d'une partie de la clé (*code\_departement*) ou d'un attribut non clé, donc.

***R21(code\_commune, code\_departement, code\_canton, altitude moyenne, superficie, population, code\_arrondissement, commune, departement)*** en 1NF.

***R22(code\_departement, code\_region, departement)*** en BCNF.

On normalise encore R21.

***R211(code\_commune, code\_dep, commune, departement, altitude moyenne, population, code\_arrondissement, superficie)*** en BCNF

***R212(code\_commune, code\_departement, code\_canton)*** en BCNF

## **b. Conception UML**

Pour cette partie on utilisera le modèle UML fournit pour la partie II car notre modèle UML proposé est très similaire mais il est pas autant simplifié.

### **i. Question 2**

Le renommage suivra la forme:

Commune : UML nom attribut (nom) Relationnel (nom\_commune)

Departement : UML nom attribut (nom) Relationnel (nom\_departement)

Region : UML nom attribut (nom) Relationnel (nom\_region)

Altitude Moyenne : UML nom attribut (altitude moy) Relationnel (altitude\_moy)

Superficie : UML nom attribut (superficie) Relationnel (superficie\_commune)

Population : UML nom attribut (population) Relationnel (population\_commune)

Code Commune : UML nom attribut (code) Relationnel (code\_commune)

Code Departement : UML nom attribut (code) Relationnel (code\_departement)

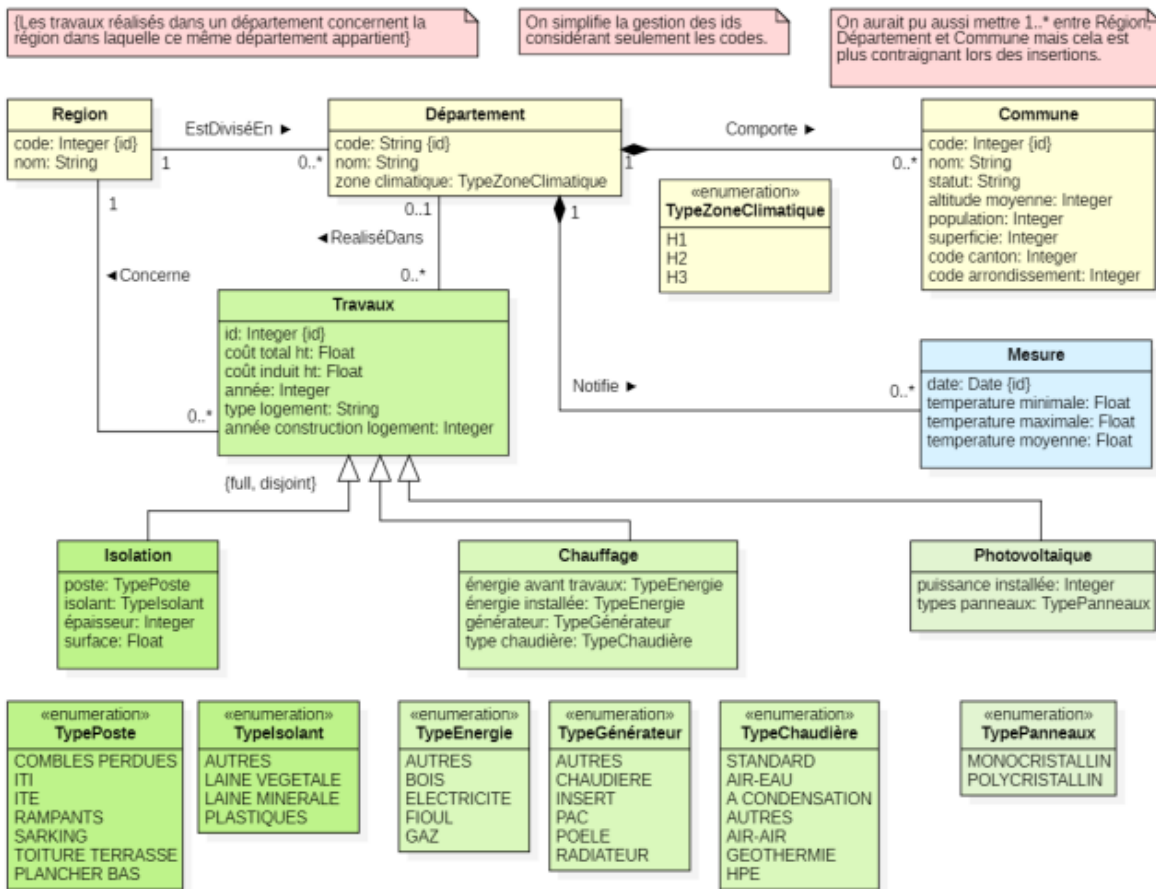
Code Commune : UML nom attribut (code) Relationnel (code\_commune)

Code Arrondissement : UML nom attribut (code arrondissement) Relationnel (code\_arrondissement)

Code Canton : UML nom attribut (code) Relationnel (code\_canton)

Code\_insee\_departement : UML nom attribut (code) Relationnel (code\_departement)

## ii. Question 3



## c. Schema remationnel

### Schéma relationnel :

- Regions(code\_region, nom\_region)
- Departements(code\_departement, nom\_departement, zone\_climatique, #code\_region)
- Mesures(#code\_departement, date\_mesure, temperature\_min\_mesure, temperature\_max\_mesure, temperature\_moy\_mesure)
- Commune(code\_commune, #code\_departement, nom, status, altitude, population, superficie, code\_canton, code\_arrondissement)
- Travaux(id\_travaux, cout\_total\_ht, cout\_induit\_ht, date\_travaux, type\_logement, annee\_construction\_logement, #code\_region)
- Photovoltaïque(#id\_travaux, puissance\_installee, types\_panneaux)
- Chauffage(#id\_travaux, energie\_avant\_travaux, energie\_installe, generateur, type\_chaudiere)
- Isolations(#id\_travaux, poste, isolant, epaisseur, surface)

### Constraints intégralité :

- $\text{Departements}[\text{code\_region}] \subseteq \text{Regions}[\text{code\_region}]$
- $\text{Mesure}[\text{code\_departement}] \subseteq \text{Departement}[\text{code\_departement}]$
- $\text{Travaux}[\text{code\_region}] \subseteq \text{Regions}[\text{code\_region}]$
- $\text{Photovoltaique}[\text{id\_travaux}] \subseteq \text{Chauffage}[\text{id\_travaux}] \subseteq \text{Isolations}[\text{id\_travaux}] \subseteq \text{Travaux}[\text{id\_travaux}]$
- $\text{Photovoltaique}[\text{types\_panneaux}] \subseteq \{\text{'MONOCRISTALLIN'}, \text{'POLYCRISTALLIN'}\}$   
 $\text{Chauffage}[\text{energie\_avant\_travaux}] \subseteq \text{Chauffage}[\text{energie\_installe}] \subseteq \{\text{'AUTRES'}, \text{'BOIS'}, \text{'ELECTRICITE'}, \text{'FIOUL'}, \text{'GAZ'}\}$
- $\text{Chauffage}[\text{generateur}] \subseteq \{\text{'AUTRES'}, \text{'CHAUDIERE'}, \text{'INSERT'}, \text{'PAC'}, \text{'POELE'}, \text{'RADIATEUR'}\}$
- $\text{Chauffage}[\text{type\_chaudiere}] \subseteq \{\text{'STANDARD'}, \text{'AIR-EAU'}, \text{'A CONDENSATION'}, \text{'AUTRES'}, \text{'AIR-AIR'}, \text{'GEO THERMIE'}, \text{'HPE'}\}$
- $\text{Isolations}[\text{poste}] \subseteq \{\text{'COMBLES PERDUES'}, \text{'ITI'}, \text{'ITE'}, \text{'RAMPANTS'}, \text{'SARKING'}, \text{'TOITURE TERRASSE'}, \text{'PLANCHER BAS'}\}$
- $\text{Isolations}[\text{isolant}] \subseteq \{\text{'AUTRES'}, \text{'LAINE VEGETALE'}, \text{'LAINE MINERALE'}, \text{'PLASTIQUES'}\}$

### Hypotheses :

- $\text{Logement}(\text{année\_construction}) \leq \text{Travaux}(\text{année\_travaux});$
- $\text{Generateur}(\text{id\_travaux}), \text{Photovoltaique}(\text{année\_travaux}), \text{Chauffage}(\text{année\_travaux}) \subseteq \text{Travaux}(\text{année\_travaux});$

## 2. Partie II : Implémentation

### a. Question 1

Dans cette question, nous avons simplement eu à rédiger une requête sql qui récupère les départements de la région d'auvergne-rhône-alpes et envoie cette requête utilisant le module *sqlite3* à la base de données. Une fois le résultat reçu, nous affichons les résultats dans la fenêtre de la question 1.

Requete :

```
# On définit les colonnes que l'on souhaite afficher dans la fenêtre et la requête
columns = ('code_departement', 'nom_departement', 'code_region')
query = """SELECT code_departement, nom_departement, code_region
           FROM Departements
           WHERE code_region = 84
           """

# On utilise la fonction createTreeViewDisplayQuery pour afficher les résultats de la requête
tree = display.createTreeViewDisplayQuery(self, columns, query, size=200)
tree.grid(row=0, sticky="nswe")
```

Affichage :

Q1 : départements de la région Auvergne-Rhône-Alpes		
code_departement	nom_departement	code_region
63	PUY-DE-DOME	84
26	DROME	84
42	LOIRE	84
69	RHONE	84
15	CANTAL	84
03	ALLIER	84
38	ISERE	84
43	HAUTE-LOIRE	84
01	AIN	84
74	HAUTE-SAVOIE	84
07	ARDECHE	84
73	SAVOIE	84

## b. Question 2

Pour cette partie, nous n'avons pas eu à modifier l'affichage de la réponse à la requête que nous soumettons. La difficulté ici était la requête en elle-même et comprendre ce que signifiait le "département le plus froid" pour ce faire nous avons fait la moyenne de température moyenne par département et nous avons ensuite calculé le minimum de cette température par régions.

Requete :

```
# On definit les colonnes que l'on souhaite afficher dans la fenetre et la requete
columns = ('code_departement', 'nom_departement', 'temperature_moy_min_mesure', 'code_region')
query = """
    WITH MoyParDep AS (
        SELECT ROUND(AVG(temperature_moy_mesure),2) as temp, code_departement
        FROM Mesures
        GROUP BY code_departement
    )
    SELECT D.code_departement, D.nom_departement, MIN(M.temp), D.code_region
    FROM Departements D JOIN MoyParDep M ON (D.code_departement = M.code_departement)
    GROUP BY D.code_region
    """
```

Affichage :

Q2 : département le plus froid par région			
code_departement	nom_departement	temperature_moy_min_mesure	code_region
78	YVELINES	12.62	11
28	EURE-ET-LOIR	12.48	24
70	HAUTE-SAONE	11.92	27
14	CALVADOS	11.99	28
02	AISNE	11.82	32
08	ARDENNES	11.04	44
53	MAYENNE	12.92	52
22	COTES-D'ARMOR	12.49	53
23	CREUSE	12.02	75
48	LOZERE	10.95	76
43	HAUTE-LOIRE	10.36	84
05	HAUTES-ALPES	12.28	93
2A	CORSE-DU-SUD	17.15	94

### c. Question 3

Nous devons faire un menu déroulant qui prend comme valeur les différentes régions de la BDD et faire en sorte qu'en cliquant sur ces régions nous affichons leurs départements. Pour ce faire nous avons eu à modifier l'affichage de la fenêtre en y ajoutant une combobox. Pour récupérer les options pour la combobox, nous avons fait une requête qui récupère toutes les régions de la table région et nous les avons mises dans la combobox. Puis nous avons fait une requête simple qui récupère les départements d'une région sélectionnée dans la combobox.

Mais pour la sélection de la région nous avons mis un "?" qui est remplacé par le retour de l'action "cliquer" sur un nom dans la combobox.

Recuperation des regions :

```
# Fonction pour extraire les régions de la base de données
! usage 1 gerarddv
def get_regions_from_database(self):
    try:
        cursor = db.data.cursor()
        result = cursor.execute("SELECT DISTINCT nom_region FROM Regions")
        regions = [row[0] for row in result.fetchall()]
        return regions
    except Exception as e:
        # Gérer l'erreur, par exemple, afficher un message d'erreur et renvoyer une liste vide
        print("Erreur lors de l'extraction des régions :", repr(e))
        return []
```

Utilisation de la combobox :

```
# Affichage du label, de la case de saisie et du bouton valider
ttk.Label(self, text='Veuillez indiquer une région :', anchor="center", font=('Helvetica', '10', 'bold')).grid(row=1, column=0)

# Création d'un Combobox pour afficher les régions extraites de la base
regions = self.get_regions_from_database() # Fonction à implémenter pour extraire les régions de la base
self.region_combobox = ttk.Combobox(self, values=regions)
self.region_combobox.grid(row=1, column=1)
self.region_combobox.bind('<Return>', self.searchRegion) # On bind l'appui de la touche entrée sur le Combobox
```

Requete SQL :

```
def searchRegion(self, event=None):
    # On vide le treeview (pour rafraichir les données si quelque chose était déjà présent)
    self.treeview.delete(*self.treeview.get_children())

    # On récupère la valeur sélectionnée dans le Combobox
    region = self.region_combobox.get()

    # Si la sélection est vide, on affiche une erreur
    if len(region) == 0:
        self.errorLabel.config(foreground='red', text="Veuillez sélectionner une région !")
    else:
        try:
            cursor = db.data.cursor()
            result = cursor.execute("""SELECT code_departement, nom_departement
                                     FROM Departements JOIN Regions USING (code_region)
                                     WHERE nom_region = ?
                                     ORDER BY code_departement""", [region])
```

Resultat :

code_departement	nom_departement
21	COTE-D'OR
25	DOUBS
39	JURA
58	NIEVRE
70	HAUTE-SAONE
71	SAONE-ET-LOIRE
89	YONNE
90	TERRITOIRE DE BELFORT

#### d. Question 4

Cette question vise à compléter la base de données en se basant sur le modèle UML conçu. Nous devons alors rajouter les tables Commune, Travaux, Isolation, Photovoltaïque et Chauffage. Ces trois dernières héritent leur clé de la table travaux. Ceci signifie que ces trois tables auront une clé étrangère faisant une référence aux clés de travaux. D'après le cours, nous pouvons gérer ce cas de deux manières, soit on traite ces tables par référence ou par unification, nous on se basera sur le traitement par référence.

Dans un premier temps ceci nous oblige à créer une nouvelle fonction qui nous permettra de faire des insertions dans la table Travaux et une de ces tables Filles (Photovoltaïque, Isolation et Chauffage).

Afin de respecter les clés créées, on récupère la clé créée et on la réinsère dans la table Travaux. Cette fonction nous permet d'avoir des clés différentes dans les trois tables filles, ce qui rend les clés dans travaux uniques aussi.

Donc en résumé dans cette partie nous avons ajouté le code SQLite permettant la créations des Tables (et la destruction) :

```
CREATE TABLE IF NOT EXISTS Commune(  
    code_commune TEXT,  
    nom TEXT,  
    status TEXT,  
    altitude FLOAT,  
    population FLOAT,  
    superficie FLOAT,  
    code_canton INTEGER,  
    code_arrondissement INTEGER,  
    code_departement TEXT,  
    CONSTRAINT pk_commune PRIMARY KEY (code_commune, code_departement)  
);
```



```
CREATE TABLE IF NOT EXISTS Travaux (
    id_travaux INTEGER PRIMARY KEY,
    cout_total_ht FLOAT,
    cout_induit_ht FLOAT,
    date_travaux DATE,
    type_logement TEXT,
    annee_construction_logement TEXT,
    code_region INTEGER,
    CONSTRAINT fk_travaux FOREIGN KEY (code_region) REFERENCES Regions(code_region) ON DELETE CASCADE
);
```

```
CREATE TABLE IF NOT EXISTS Photovoltaïque(
    id_travaux INTEGER PRIMARY KEY AUTOINCREMENT,
    puissance_installee FLOAT,
    types_panneaux TEXT CHECK (types_panneaux IN ('MONOCRISTALLIN', 'POLYCRISTALLIN')),
    CONSTRAINT fk_photo FOREIGN KEY (id_travaux) REFERENCES Travaux(id_travaux) ON DELETE CASCADE
);
```

```
CREATE TABLE IF NOT EXISTS Chauffage(
    id_travaux INTEGER PRIMARY KEY AUTOINCREMENT,
    energie_avant_travaux TEXT CHECK (energie_avant_travaux IN ('AUTRES', 'BOIS', 'ELECTRICITE', 'FIOUL', 'GAZ')),
    energie_installee TEXT CHECK (energie_installee IN ('AUTRES', 'BOIS', 'ELECTRICITE', 'FIOUL', 'GAZ')),
    generateur TEXT CHECK (generateur IN ('AUTRES', 'CHAUDIERE', 'INSERT', 'PAC', 'POELE', 'RADIATEUR')),
    type_chaudiere TEXT CHECK (type_chaudiere IN ('STANDARD', 'AIR-EAU', 'A CONDENSATION', 'AUTRES', 'AIR-AIR', 'GEOTHERMIE', 'HPE')),
    CONSTRAINT fk_chauff FOREIGN KEY (id_travaux) REFERENCES Travaux(id_travaux) ON DELETE CASCADE
);
```

```
CREATE TABLE IF NOT EXISTS Isolations(
    id_travaux INTEGER PRIMARY KEY AUTOINCREMENT,
    poste TEXT CHECK (poste IN ('COMBLES PERDUES', 'ITI', 'ITE', 'RAMPANTS', 'SARKING', 'TOITURE TERASSE', 'PLANCHER BAS')),
    isolant TEXT CHECK (isolant IN ('AUTRES', 'LAINE VEGETALE', 'LAINE MINERALE', 'PLASTIQUES')),
    epaisseur FLOAT,
    surface FLOAT,
    CONSTRAINT fk_isolant FOREIGN KEY (id_travaux) REFERENCES Travaux(id_travaux) ON DELETE CASCADE
);
```

Ensuite pour l'insertion nous avons créé la fonction **def read\_csv\_multi\_table(csvFile, separator, query1, query2, columns1, columns2):** qui prends un fichier csv, deux requêtes et deux schémas et insérés les données correspondantes. On s'est basé sur la fonction de base pour ce faire (pour voir la fonction, fichier **db.py ligne 171**).

Finalement, nous avons rajouté l'affichage de ces nouvelles tables, afin de les afficher lorsque nous appuyons sur consultez base de données.

Comme le code proposé comptait avec beaucoup de répétition de code nous avons simplifié la gestion de l'affichage avec la création d'un tableau comptant avec toutes les informations à afficher par table, ceci nous permet d'afficher la base des données en bouclant sur ce tableau.

Le fichier est **tablesData.py**. Le tableau :

```
tables = [
    {'name': 'Regions', 'columns': ('code_region', 'nom_region'),
     'query': 'SELECT code_region, nom_region FROM Regions ORDER BY code_region'},
    {'name': 'Departements',
     'columns': ('code_departement', 'nom_departement', 'code_region', 'zone_climatique'),
     'query': 'SELECT code_departement, nom_departement, code_region, zone_climatique FROM Departements ORDER BY code_departement'},
    {'name': 'Mesures', 'columns': (
     'code_departement', 'date_mesure', 'temperature_min_mesure', 'temperature_max_mesure',
     'temperature_moy_mesure'),
     'query': 'SELECT code_departement, date_mesure, temperature_min_mesure, temperature_max_mesure, temperature_moy_mesure '
     'FROM Mesures ORDER BY date_mesure LIMIT 1,1000'},
    {'name': 'Commune', 'columns': (
     'code_commune', 'nom', 'status', 'altitude', 'population', 'superficie', 'code_canton',
     'code_arrondissement', 'code_departement'),
     'query': 'SELECT code_commune, nom, status, altitude, population, superficie, code_canton, code_arrondissement, code_departement '
     'FROM Commune ORDER BY code_commune, code_departement'},
    {'name': 'Travaux', 'columns': (
     'id_travaux', 'cout_total_ht', 'cout_induit_ht', 'date_travaux', 'type_logement', 'annee_construction_logement',
     'code_region'),
     'query': 'SELECT id_travaux, cout_total_ht, cout_induit_ht, date_travaux, type_logement, annee_construction_logement, code_region '
     'FROM Travaux ORDER BY id_travaux'},
    {'name': 'Photovoltaique', 'columns': ('id_travaux', 'puissance_installee', 'types_panneaux'),
     'query': 'SELECT id_travaux, puissance_installee, types_panneaux FROM Photovoltaique ORDER BY id_travaux'},
    {'name': 'Chauffage',
     'columns': ('id_travaux', 'energie_avant_travaux', 'energie_installee', 'generateur', 'type_chaudiere'),
     'query': 'SELECT id_travaux, energie_avant_travaux, energie_installee, generateur, type_chaudiere FROM Chauffage ORDER BY id_travaux'},
    {'name': 'Isolations', 'columns': ('id_travaux', 'poste', 'isolant', 'epaisseur', 'surface'),
     'query': 'SELECT id_travaux, poste, isolant, epaisseur, surface FROM Isolations ORDER BY id_travaux'}
]
```

Finalement dans l’affichage :

Consultation des données de la base

Regions	Departements	Mesures	Commune	Travaux	Photovoltaique	Chauffage	Isolations				
		code_commune		nom		status		altitude		population	
		1		L'ABERGEMENT-CLEMENC		Commune simple		242.0		0.8	
		1		ABBECCOURT		Commune simple		47.0		0.5	
		1		ABREST		Commune simple		304.0		2.7	
		1		AIGLUN		Commune simple		631.0		1.2	
		1		ABRIES		Commune simple		2272.0		0.4	
		...		...		...		...		...	

Consultation des données de la base

Regions	Departements	Mesures	Commune	Travaux	Photovoltaique	Chauffage	Isolations							
		id_travaux		cout_total_ht		cout_induit_ht		date_travaux		type_logement		annee_construction_logem		code_region
		1		12069.19431		None		2012-01-20		null		null		53
		2		22748.81517		None		2015-11-16		null		null		53
		3		23696.68246		None				null		null		84
		4		8289.099526		None				null		null		84
		5		11374.40758		None				null		null		84
		6		11253.08057		None				null		null		84
		7		2400.0		None				null		null		84
		8		20000.0		None				null		null		84
		9		35000.0		None				null		null		84
		10		13744.07583		None		2011-11-04		null		null		28
		11		22559.24171		None		2011-07-29		null		null		24
		12		18009.47867		None		2011-12-20		null		null		24
		13		15165.87678		None		2011-12-20		null		null		24
		14		16113.74408		None		2011-07-29		null		null		24
		15		39241.70616		None		2012-02-07		null		null		24
		16		49763.03318		None		2011-06-22		null		null		24
		17		17061.61137		None		2013-04-19		null		null		24

Consultation des données de la base							
Regions	Departements	Mesures	Commune	Travaux	Photovoltaïque	Chauffage	Isolations
id_travaux		puissance_installee		types_panneaux			
1		3000.0		MONOCRISTALLIN			
2		3000.0		POLYCRISTALLIN			
23		3000.0		POLYCRISTALLIN			
24		6160.0		MONOCRISTALLIN			
25		3000.0		POLYCRISTALLIN			
26		3000.0		POLYCRISTALLIN			
27		170.0		MONOCRISTALLIN			
30		8400.0		POLYCRISTALLIN			
31		2943.0		MONOCRISTALLIN			
33		6720.0		MONOCRISTALLIN			

### e. Question 5

Cette question est centrée sur l'optimisation et la propreté des requêtes SQL. Dans un premier temps le code proposé exécute trois requêtes individuelles qui sont affichées ensemble à la fin du code. Pour optimiser le code, nous avons réduit cette requête en une seule et nous pouvons apprécier un changement sur le temps d'exécution. Sur mon ordinateur, le temps d'exécution de la requête non optimisée prend entre 0.9 et 1.2 secondes alors que la version modifiée prend seulement 0.2 secondes, ce qui signifie une réduction du temps d'exécution du 80%.

Requête optimisée :

```
query = """
SELECT D.code_departement, D.nom_departement, strftime('%Y', M.date_mesure) as annee,
       ROUND(avg(M.temperature_moy_mesure), 2) AS moyenne,
       min(M.temperature_min_mesure) AS minimum,
       max(M.temperature_max_mesure) AS maximum
FROM Departements D
INNER JOIN Mesures M ON D.code_departement = M.code_departement
GROUP BY D.code_departement, annee
ORDER BY D.code_departement, annee
"""
```

Le reste du code fonctionne comme le code non optimisé proposé.

### f. Question 6

Dans cette partie nous voulons utiliser les données dans les tables pour avoir une comparaison visuelle entre la variation moyenne des températures minimales dans l'iseres au cours d'une année désigné (2018 car pour 2022 nous avons des données incomplètes) et le coût des travaux dans un graphe (qu'on tracera avec le module matplotlib).

Suite a un problème de données manquantes avec la colonne date tavaux, nous récupérons plutôt date\_x du fichier csv mais ces dates sont sous un format différent, "YYYY/MM/DD" au lieu de "YYYY-MM-DD". Pour régler ceci, lors de l'insertion nous utiliserons la datetime afin de transformer les dates dans le bon format.

Pour l’affichage nous avons deux courbes ce qui se traduit en deux requêtes :

```
temperature_query = """
    SELECT strftime('%m', date_mesure) as mois, AVG(temperature_min_mesure) as moy_min
    FROM Mesures
    WHERE code_departement = 38 AND strftime('%Y', date_mesure) = '2018'
    GROUP BY mois
    ORDER BY mois;
    """

temperature_result = self.execute_query(temperature_query)
tabx = [row[0] for row in temperature_result]
tabminmoy = [row[1] for row in temperature_result]
```

```
# Fetch travaux data
travaux_query = """
    SELECT strftime('%m', date_travaux) as mois, SUM(cout_total_ht) as total_cout_travaux
    FROM Travaux JOIN Departements USING(code_region)
    WHERE strftime('%Y', date_travaux) = '2018'
    GROUP BY mois
    ORDER BY mois;
    """

travaux_result = self.execute_query(travaux_query)
tabx1 = [row[0] for row in travaux_result]
tabcout = [row[1] for row in travaux_result]
```

Affichage du graphe :

```
datetime_dates = [datetime.strptime(date, _format='%m').strftime("%B") for date in tabx]
datetime_dates1 = [datetime.strptime(date, _format='%m').strftime("%B") for date in tabx1]

fig, ax1 = plt.subplots(figsize=(15, 8))

# axe y gauche temperature
color = 'tab:blue'
ax1.set_xlabel('Mois 2018')
ax1.set_ylabel('Température Min Moyenne', color=color)
ax1.plot(datetime_dates, tabminmoy, color=color)
ax1.tick_params(axis='y', labelcolor=color)

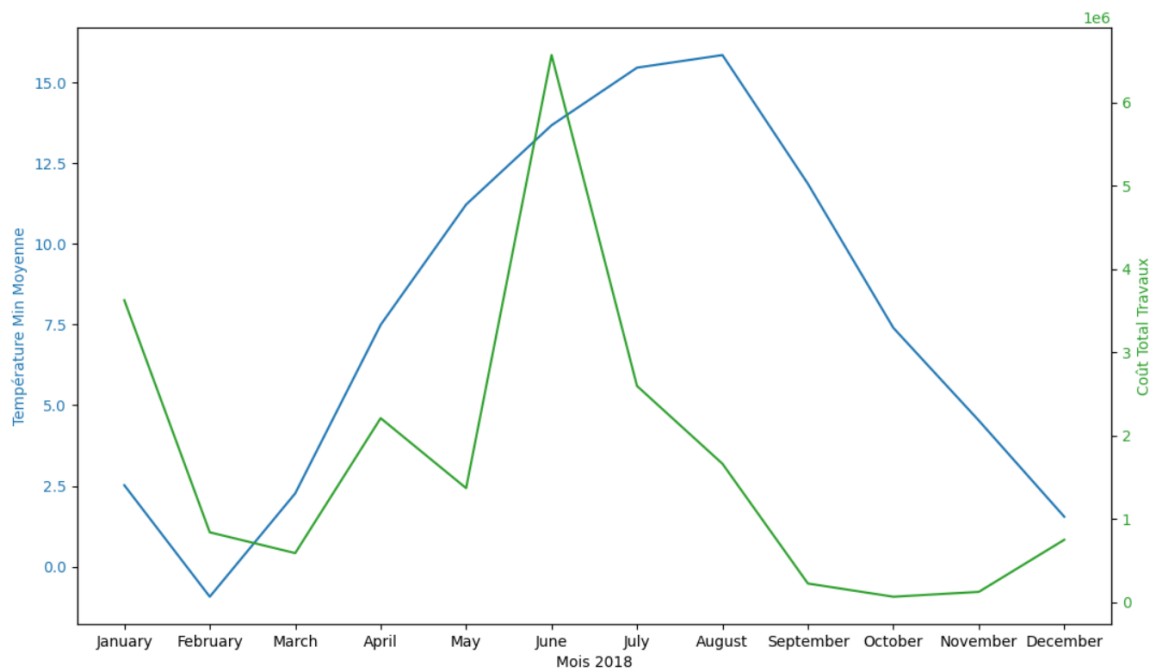
#axe y cout
ax2 = ax1.twinx()
color = 'tab:green'
ax2.set_ylabel('Coût Total Travaux', color=color)
ax2.plot(datetime_dates1, tabcout, color=color)
ax2.tick_params(axis='y', labelcolor=color)

canvas = FigureCanvasTkAgg(fig, master=self)
canvas.draw()
canvas.get_tk_widget().pack()

plt.xticks(rotation='vertical')

plt.show()
```

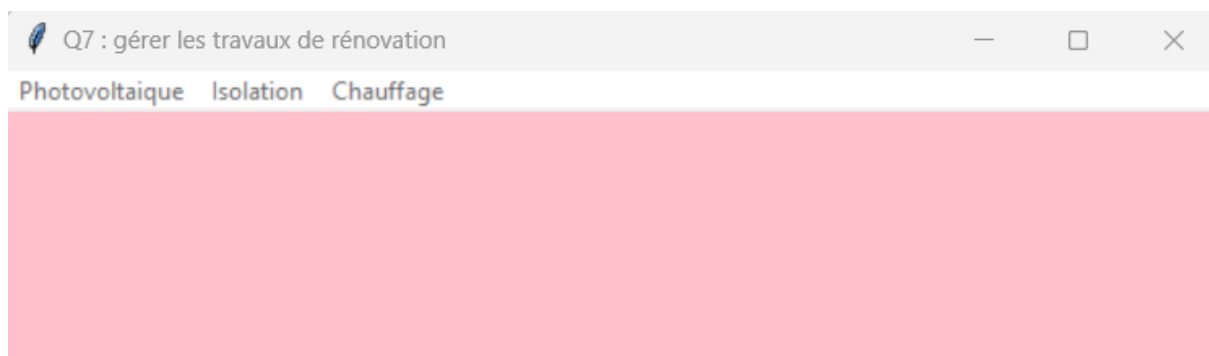
Resultat :



Remarque: Pour le coût des travaux nous prenons les coût de tous les départements car pour l'Isère nous n'avons pas des mesures pour tous les mois.

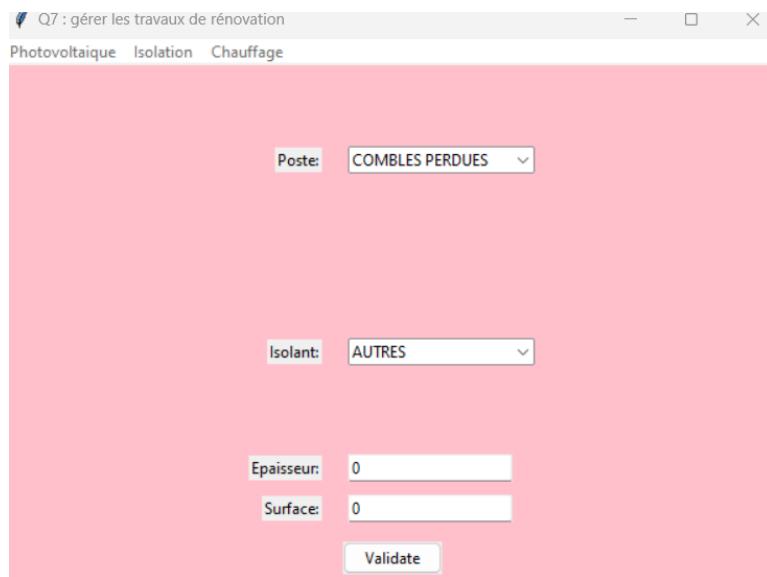
### g. Question 7

Pour cette partie, nous nous sommes basées sur l'utilisation de tkinter dans le code donnée. Dans la fenêtre Q7, nous avons un menu avec les trois types des travaux à gérer, en passant le curseur sur un des travaux nous avons les options *add*, *delete* ou *modify* affichés.

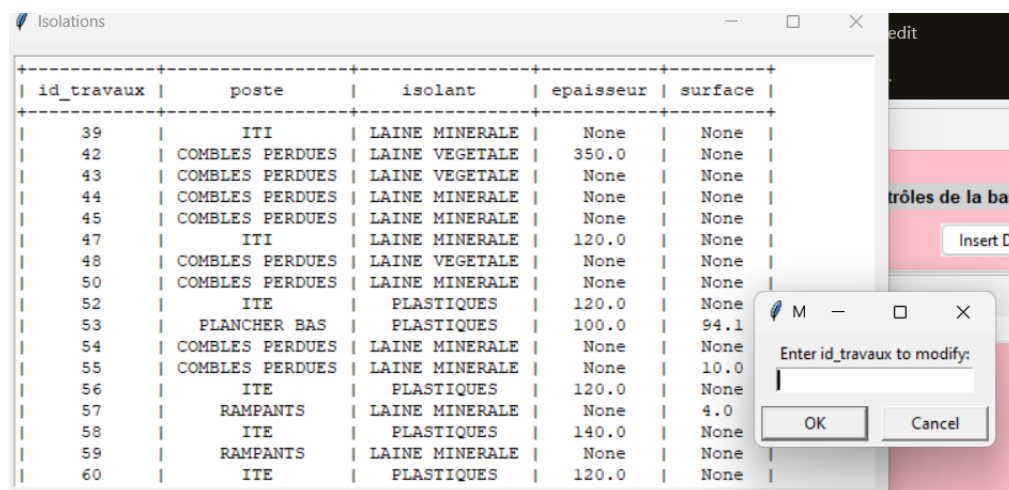


Ces trois fonctions sont les mêmes pour les trois type de travaux, on change seulement le schéma à traiter.

add photovoltaïque :



Pour les fonctions modify et delete on affiche d'abord la table (requête + prettyTable module) à traiter afin que l'utilisateur puisse repérer le `id_travaux` à fournir pour modifier ou supprimer la ligne correspondante :



id_travaux	poste	isolant	epaisseur	surface
39	ITI	LAINE MINERALE	None	None
42	COMBLES PERDUES	LAINE VEGETALE	350.0	None
43	COMBLES PERDUES	LAINE VEGETALE	None	None
44	COMBLES PERDUES	LAINE MINERALE	None	None
45	COMBLES PERDUES	LAINE MINERALE	None	None
47	ITI	LAINE MINERALE	120.0	None
48	COMBLES PERDUES	LAINE VEGETALE	None	None
50	COMBLES PERDUES	LAINE MINERALE	None	None
52	ITE	PLASTIQUES	120.0	None
53	PLANCHER BAS	PLASTIQUES	100.0	94.1
54	COMBLES PERDUES	LAINE MINERALE	None	None
55	COMBLES PERDUES	LAINE MINERALE	None	10.0
56	ITE	PLASTIQUES	120.0	None
57	RAMPANTS	LAINE MINERALE	None	4.0
58	ITE	PLASTIQUES	140.0	None
59	RAMPANTS	LAINE MINERALE	None	None
60	ITE	PLASTIQUES	120.0	None

Ensuite on passe dans une fenêtre comme celle de add, ou on pourra fournir les nouvelles données. Delete supprime la ligne sélectionnée si elle existe.

Selon l'option sélectionnée *add*, *delete* ou *modify* la requête sera *INSERT*, *DELETE* ou *UPDATE*.

Nous avons bien pris en compte que lors de l'insertion d'un nouveau type de travaux' on doit l'ajouter dans travaux avec le même id : `id_travaux` pour garantir les contraintes d'intégralité des tables.

On arrive à bien insérer les ids dans les deux tables et les données dans la table du type de travaux mais on n'a pas réussi à ajouter les valeurs dans la table Travaux, nous avons écrit

les fonctions qui ouvrent une nouvelle fenêtre pour insérer les valeurs mais il y a un problème avec la requête de mise à jour des données dans la table Travaux alors que nous avons réussi à insérer le bon *id\_travaux*.

Les fonctions de modification et suppression fonctionnent correctement, néanmoins il y a des corrections à faire pour rendre le code plus fonctionnel, les éléments à corriger sont :

- Update données dans le nouveau travaux inséré dans la table Travaux
- Détruire correctement les fenêtres (problèmes d'affichage lors du passage entre les options du menu, il faut ouvrir la fenêtre mère).
- Optimisation du code avec l'utilisation de classes (réduire le fichier Q7.py contenant +600 lignes de code redondantes).

### **3. Utilisation du Git**

L'utilisation du Git nous a aidé à suivre une trace des avances, ainsi que les problèmes retrouvés et non résolus. Vous pouvez récupérer le code :

HTTPS : <https://github.com/gerarddv/MissionClimat.git>

SSH : <git@github.com:gerarddv/MissionClimat.git>

Repertoire : <https://github.com/gerarddv/MissionClimat>

User : gerarddv