

Exceptions et Base de données SQL

TRIBOULET Corentin – D1b – DELLANDREA

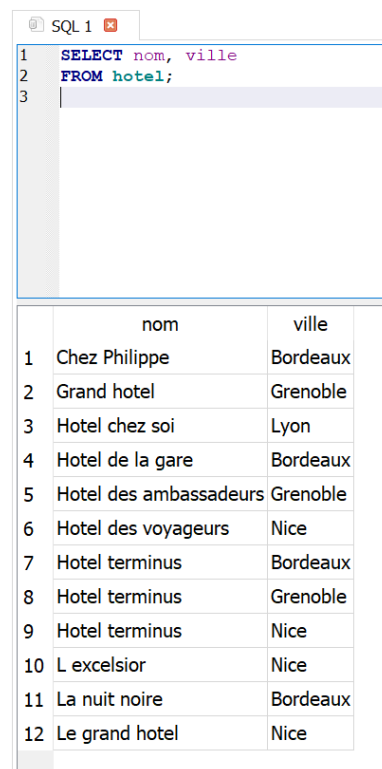
1. Mini-tutoriel sur la base Hotellerie.db
 - 1.1 DB browser for SQLite
 - 1.2 Quelques requêtes en Python
 - 1.3 La gestion des exceptions
2. Classe HotelDB
 - 2.1 Requête en lecture
 - 2.2 Requête en écriture
3. Requêtes libres
 - 3.1 Prix d'une nuit moyen par hotel
 - 3.2 Répartition des durées de réservation

1. Mini-tutoriel sur la base Hotellerie.db

1.1 DB browser for SQLite

Requête Sql:

On obtient bien la liste de tous les hôtels avec leur nom et ville.



The screenshot shows a SQL query window titled 'SQL 1' with the following code:

```
1 SELECT nom, ville
2 FROM hotel;
3
```

Below the query, a table displays the results of the query. The table has two columns: 'nom' and 'ville'. There are 12 rows of data, numbered 1 to 12 in the first column.

	nom	ville
1	Chez Philippe	Bordeaux
2	Grand hotel	Grenoble
3	Hotel chez soi	Lyon
4	Hotel de la gare	Bordeaux
5	Hotel des ambassadeurs	Grenoble
6	Hotel des voyageurs	Nice
7	Hotel terminus	Bordeaux
8	Hotel terminus	Grenoble
9	Hotel terminus	Nice
10	L excelsior	Nice
11	La nuit noire	Bordeaux
12	Le grand hotel	Nice

1.2 Quelques requêtes en Python

```
import sqlite3
if __name__ == '__main__':
    conn = sqlite3.connect('hotellerie.db')

    curseur = conn.cursor() # objet permettant de faire des requêtes
    curseur.execute("SELECT nom, ville FROM hotel;") # requête SQL
    ligne1 = curseur.fetchone() # récupère la 1ère ligne du résultat de la requête
    print('ligne1 =', ligne1)
    ligneAll = curseur.fetchall() # récupère toutes les lignes suivantes
    print('ligneAll =', ligneAll)

    conn.close()
```

```
#Saisie
>>> print(ligneAll[0][0])
Grand hotel
```

On obtient le nom du premier hotel (cf : photo liste des hotels)

```
import sqlite3
if __name__ == '__main__':
    conn = sqlite3.connect('hotellerie.db')

    curseur = conn.cursor()
    for ligne in curseur.execute("SELECT * FROM hotel WHERE etoiles=3"):
        print('ligne=', ligne)

    conn.close()
```

```
#Saisie
>>> (executing file )
ligne= (1, 'L excelsior', 'Nice', 3)
ligne= (10, 'Hotel des ambassadeurs', 'Grenoble', 3)
ligne= (11, 'Hotel des voyageurs', 'Nice', 3)
```

On obtient les hotels 3 étoiles

1.3 La gestion des exceptions

- On change le nom de la base de données

```
import sqlite3
if __name__ == '__main__':
    try:
        conn = sqlite3.connect('impossible.db')           #changement de nom
        curseur = conn.cursor()
        curseur.execute("SELECT nom, ville FROM hotel;")
        print(curseur.fetchall())
    except Exception as err:                               # interception d'une exception quelconque
        print('err:', str(err))
        print('type exception:', type(err).__name__)
    finally:                                                # fermeture de la base dans tous les cas
        conn.close()
```

```
#Saisie
>>> (executing file "<tmp 1>")
err: no such table: hotel
type exception: OperationalError
```

Il y a donc une erreur de type 'OperationalError', car dans la base de donnée impossible.db (qui n'existe pas d'ailleurs) la table 'hotel' n'existe pas

- On change le nom des tables

```
import sqlite3
if __name__ == '__main__':
    try:
        conn = sqlite3.connect('hotellerie.db')
        curseur = conn.cursor()
        curseur.execute("SELECT nom, vill FROM hotel;")    # la table ville est erronée
        print(curseur.fetchall())
    except Exception as err:                               # interception d'une exception quelconque
        print('err:', str(err))
        print('type exception:', type(err).__name__)
    finally:                                                # fermeture de la base dans tous les cas
        conn.close()
```

```
#Saisie
>>> (executing file "<tmp 1>")
err: no such column: vill
type exception: OperationalError
```

De la même manière qu'à la question précédente, la table 'vill' n'existe pas

- On modifie les champs

```
import sqlite3
if __name__ == '__main__':
    try:
        conn = sqlite3.connect('hotellerie.db')
        curseur = conn.cursor()
        curseur.execute("SELECT nom, ville FROM hotel;") # Les champs sont modifiés
        print(curseur.fetchall())
    except Exception as err: # interception d'une exception quelconque
        print('err:', str(err))
        print('type exception:', type(err).__name__)
    finally: # fermeture de la base dans tous les cas
        conn.close()
```

```
#Saisie
>>> (executing file "<tmp 1>")
err: near "hotel": syntax error
type exception: OperationalError
```

L'erreur vient maintenant de la syntaxe d'écriture. Le programme permet donc de trouver les erreurs et en donner les raisons, il est donc robuste.

- On remplace alors le code `except Exception as err:` par le code `except sqlite3.OperationalError as err:`

On obtient exactement les mêmes réponses à l'exécution du code

2. Classe HotelDB

2.1 Requête en lecture

```
def get_name_hotel_etoile(self,nbEtoiles):
    #recupere le nom des hotels avec le nombre d'étoile demandé
    try:
        if nbEtoiles<=0:
            print("Nombre d'étoiles négatifs")
            raise ValueError
        self.__curseur.execute("SELECT nom FROM hotel WHERE etoiles =="+ str(nbEtoiles))
    except Exception as err:
        print('Type exception:', type(err).__name__)
    else:
        return self.__curseur.fetchall()
```

```
#Saisie
>>> aHotelDB.get_name_hotel_etoile(-1)
Nombre d'étoiles négatifs
Type exception: ValueError

>>> aHotelDB.get_name_hotel_etoile("Hello")
Type exception: TypeError
```

Le programme est donc robuste par sa compréhension des exceptions.

2.2 Requête en écriture

```
def ajouter_client(self,prenom,nom): # ajoute un client s'il n'est pas présent dans la base
    curseur=self.__curseur
    try:
        curseur.execute(f"SELECT numclient FROM client WHERE nom='{nom}' and prenom='{prenom}'")
    except Exception:
        print('Une erreur a été relevée')
        liste=curseur.fetchall()
        if len(liste)!=0:
            print( "Le client existe déjà. Son numéro est le " + str(liste[0][0]) + ".")
        else:
            curseur.execute(f"INSERT INTO client(nom,prenom) VALUES ('{nom}','{prenom}')
```

```

        print("Son numéro est le "+ str(curseur.lastrowid) + ".")
    self.__conn.commit()
    self.__conn.close()

```

```

#Tests
>>> aHotelDB.ajouter_client(1,2)          #ce n'est pas le type attendu
Une erreur a été relevée

```

```

#Saisie

>>> aHotelDB.ajouter_client('Francois','Dupont')
Son numéro est le 180.

>>> aHotelDB.ajouter_client('Francois','Dupont') #On a donc bien ajouter un nouveau client
Le client existe déjà. Son numéro est le 180.

```

3. Requêtes libres

3.1 Prix d'une nuit moyen par hotel

Je décide de calculer, par hotel, le prix moyen d'une nuit. Je calculerais aussi la moyenne totale d'une nuit d'un hotel dans la base de données.

```

##Programme principal
def histogramme_prix_moyen_chambre():
    Prix_Moyen=[]                                #initialisation des listes
    Prix=[]
    Nom_Hotel=[]
    l=len(aHotelDB.liste_nom_hotel())

    for i in range(1,l):
        Prix_Moyen.append(aHotelDB.prix_moyen_hotel(i))    #liste des prix moyens des nuits pour chaque hotel
        Nom_Hotel.append(aHotelDB.liste_nom_hotel()[i][0]) #liste des noms des hotels
        Prix+=aHotelDB.prix_hotel(i)                       #construction de la liste de tous les prix des nuits dans tous les hotels

    AVG=sum(Prix)/len(Prix)

    plt.figure(figsize = (10, 10))                    #histograme
    plt.barh(range(1,l),Prix_Moyen, edgecolor = ['blue' for i in range(1,l)],color = ['yellow' for i in range(1,l)],linestyle = 'solid')
    plt.yticks(range(1,l), Nom_Hotel)
    plt.title("Prix moyen des chambres dans les hotels")

    plt.plot([AVG,AVG],[0,l],linestyle = 'dashed', linewidth = 2)    #ligne verticale correspondant à la moyenne totale des prix
    plt.text(AVG, -0.3, "Moyenne",horizontalalignment = 'center', verticalalignment = 'center')

    plt.show()

```

```

##Programme dans la classe 'hotel'
def prix_moyen_hotel(self,num):                    #renvoie le prix moyen d'une nuit pour un hotel donné (par son numéro)
    curseur=self.__curseur
    try:
        if type(num)!=int:
            raise TypeError
        curseur.execute(f"SELECT AVG(prixnuitht) FROM chambre WHERE numhotel='{num}' ")
    except Exception:
        print('Une erreur a été relevée')
    return curseur.fetchall()[0][0]

def prix_hotel(self,num):                          #renvoie la liste de tous les prix d'un hotel donné (par son numéro)
    curseur=self.__curseur                        #Ce programme permettra de calculer la moyenne totale
    try:
        if type(num)!=int:
            raise TypeError
        curseur.execute(f"SELECT prixnuitht FROM chambre WHERE numhotel='{num}' ")
    except Exception:
        print('Une erreur a été relevée')
    return [l[0] for l in curseur.fetchall()]

```

```

#Tests
#1
>>> aHotelDB.prix_hotel("bonjour")          #ce n'est pas le type attendu

```

Une erreur a été relevée

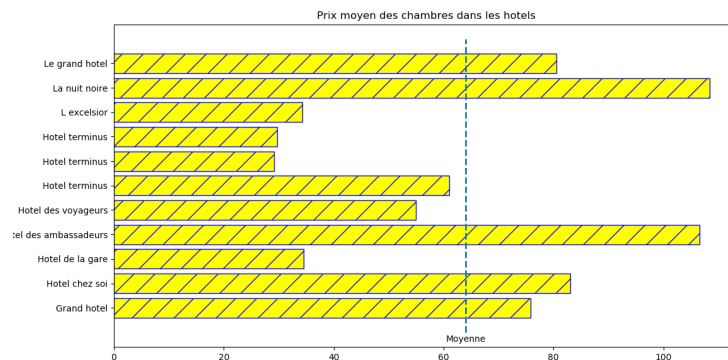
```
#2
>>> aHotelDB.prix_moyen_hotel("bonjour") #ce n'est pas le type attendu
Une erreur a été relevée

#3
def prix_moyen_hotel(self,num):#renvoie le prix moyen d'une nuit pour un hotel donné (par son numéro) #Ce programme permettra de calculer le prix moyen d'une nuit pour un hotel donné
    curseur=self.__curseur
    try:
        if type(n)!=int: #au lieu de type(num)
            raise TypeError
        curseur.execute(f"SELECT AVG(prixnuit) FROM chambre WHERE numhotel='{num}' ")
    except Exception:
        print('Une erreur a été relevée')
    return curseur.fetchall()[0][0]

>>> aHotelDB.prix_moyen_hotel(1) #ce n'est pas le type attendu
Une erreur a été relevée

#4
>>> aHotelDB.dates('bonjour') #ce n'est pas le type attendu
Une erreur a été relevée
```

```
#Saisie
>>> histogramme_prix_moyen_chambre()
```



3.2 Répartition des durées de réservation

Je me propose de réaliser un diagramme camembert donnant les proportions de la durée des réservations pour tous les hotels en général.

```
#Programme principal
def camembert_duree_occupation():
    l=len(aHotelDB.liste_resa())
    L_nbjour=[]
    Labels=['1 jour', '2 jours', '3 jours', '4 jours', '5 à 10 jours', '10 à 20 jours', 'Plus de 20 jours'] #liste de classes de proportion
    L_compteur_labels=[0,0,0,0,0,0,0] #initialisation des compteurs
    for i in range(1,l):
        nbjour=date_diff(aHotelDB.dates(i))
        if nbjour==1:
            L_compteur_labels[0]+=1
        elif nbjour==2:
            L_compteur_labels[1]+=1
        elif nbjour==3:
            L_compteur_labels[2]+=1
        elif nbjour==4:
            L_compteur_labels[3]+=1
        elif nbjour>4 and nbjour<=10:
            L_compteur_labels[4]+=1
        elif nbjour>10 and nbjour<=20:
            L_compteur_labels[5]+=1
        elif nbjour>20:
            L_compteur_labels[6]+=1

    plt.title("Durée d'occupation des clients")

    plt.pie(L_compteur_labels, labels = Labels,
            colors = ['red', 'green', 'yellow'],
            explode = [0, 0, 0, 0, 0, 0, 0.1 ],
            autopct = lambda x: str(round(x, 2)) + '%',
            pctdistance = 0.8, labeldistance = 1.1,

            #permet de faire sortir une part du camembert
            #écriture des pourcentages
            #distance du texte par rapport au centre du camembert
```

```

        shadow = True)                                #affichage de l'ombre
    plt.show()

```

```

##Sous programme
def date_diff(l):#retourne la difference de jours entre deux dates avec une liste de type [('AAAA-MM-JJ', 'AAAA-MM-JJ')]
    if type(l)!=list:
        print("Ce n'est pas une liste")
    else:
        arrivee=list(l[0][0]);depart=list(l[0][1])

        L_arrivee=[int(i) for i in [arrivee[0]+arrivee[1]+arrivee[2]+arrivee[3]]+[int(i) for i in [arrivee[5]+arrivee[6]]]+[int(i) for i in [arrivee[8]+arrivee[9]]]]
        L_depart=[int(i) for i in [depart[0]+depart[1]+depart[2]+depart[3]]+[int(i) for i in [depart[5]+depart[6]]]+[int(i) for i in [depart[8]+depart[9]]]]

        date_arrivee=date(L_arrivee[0],L_arrivee[1],L_arrivee[2])#type date de python
        date_depart=date(L_depart[0],L_depart[1],L_depart[2])

        return (date_depart-date_arrivee).days

```

```

##Programme dans la classe 'hotel'
def dates(self,num_resa): #renvoie les dates d'arrivée et de départ d'une réservation donnée par son numéro
    curseur=self.__curseur
    try:
        if type(num_resa)!=int:
            raise TypeError
        curseur.execute(f"SELECT datearrivee,datedepart FROM reservation WHERE numresa='{num_resa}' ")
    except Exception:
        print('Une erreur a été relevée')
    else:
        return curseur.fetchall()

def liste_resa(self):#renvoie la liste des réservations (permet ensuite de connaître le nombre total de réservation)
    curseur=self.__curseur
    try:
        curseur.execute("SELECT numresa FROM reservation")
    except Exception:
        print('Une erreur a été relevée')
    return curseur.fetchall()

```

```

#Tests
#1
def dates(self,num_resa):
    curseur=self.__curseur
    try:
        if type(num)!=int: #au lieu de type(num_resa)
            raise TypeError
        curseur.execute(f"SELECT datearrivee,datedepart FROM reservation WHERE numresa='{num_resa}' ")
    except Exception:
        print('Une erreur a été relevée')
    else:
        return curseur.fetchall()

>>> aHotelDB.dates(1)      #ce n'est pas le type attendu
Une erreur a été relevée

>>> camembert_duree_occupation()
Une erreur a été relevée
Ce n'est pas une liste

#2
>>> aHotelDB.dates('Bonjour') #ce n'est pas le type attendu
Une erreur a été relevée

```

```

#Saisie
>>> camembert_duree_occupation()

```

