TP12 : Persistance des données

De nombreux programmes utilisent des données situées sur une base de données séparée du programme lui-même. Le programme qui effectue les requêtes est appelé le “client”, le programme qui répond aux requêtes est le “serveur”. Le serveur est en général situé sur une machine distante, accessible par son adresse IP. Ici, pour des raisons de simplicité, nous travaillerons sur une base locale gérée par le programme sqlite.

Le but de ce TP est d’écrire un programme client (en Python) qui se connecte à une base de données et affiche le résultat de quelques requêtes.

1. Lecture de la base

Nous travaillerons sur la base `foodmart.db` contenant la description d’environ 1200 employés d’un magasin de grande distribution.



Les employés sont décrits par les attributs :

Employee (employee_id, full_name, first_name, position_id, position_title, store_id, department_id, birth_date, hire_date, end_date, salary, supervisor_id, education_level, marital_status, gender, management_role)[1)]

Téléchargez foodmart.db, placez-vous dans le bon répertoire et lancez:

```
sqlite3 foodmart.db
```

La base contient 2 tables : `employee` et `department`.

tapez par exemple :

```
.schema employee
```

Vous voyez apparaître les commandes sql qui définissent la structure de la table. tapez ensuite :

```
SELECT * FROM employee;
```

la liste de employés s’affiche.

Testez les requêtes suivantes dans la base :

Par exemple :

- Donner le nombre total d’employés
- Donner le nombre d’employés et le salaire moyen par magasin

- Donner les salaires min et max et moyen par catégorie managériale
- Donner la liste des magasins employant plus de 30 employés
- Donner l'identifiant des employés supervisant plus de 5 employés
- Donner le nom du superviseur D'Ana Quick
- Donner le nom complet, la fonction et la catégorie managériale des employés supervisant plus de 5 employés
- Donner le nom complet, la fonction et la catégorie managériale des 10 employés ayant les plus hauts salaires

etc...

2. Création d'un nouveau projet Python

Le TP sera réalisé en Python. Lancez le programme Pycharm et créez un nouveau projet.

Nous utiliserons comme dans le TP précédent la librairie sqlite3 dont les fonctionnalités permettent d'envoyer des requêtes vers un serveur de bases de données.

Commencez par importer la librairie mentionnée :

```
import sqlite3
```

Déplacez le fichier `foodmart.db` dans le dossier de votre projet Python.

Créez un programme principal contenant le code suivant et exécutez-le:

```
import sqlite3
import os, sys

def connecte_base(db_name):
    try:
        assert os.path.isfile(db_name)
        db = sqlite3.connect(db_name)
        print("Connexion à ", db_name, "OK.")
        return db
    except:
        print("Erreur de connexion : la base n'existe pas!")
        sys.exit()

db = connecte_base("foodmart.db")
```

Pour pouvoir exécuter des requêtes dans la base, il faut créer un "curseur" :

```
c = db.cursor()
```

Vous pouvez maintenant exécuter des requêtes à l'aide de ce curseur:

Il est conseillé d'encadrer les requêtes avec une commande de gestion des exceptions pour éviter un arrêt brutal en cas d'erreur de syntaxe SQL :

```
try:
    c.execute("SELECT * FROM employee")
except sqlite3.OperationalError as e:
    print("Erreur SQL :" + e.args[0])
```

Une fois la requête exécutée sans erreur, les réponses sont disponibles à l'aide de la commande `fetchall`:

```
liste_tuples = c.fetchall()
for t in liste_tuples:
    print(t)
```

Exécutez le programme et vérifiez qu'il n'y a pas d'erreur (la liste des clients s'affiche sous la forme d'une liste de tuples)

3. Premières requêtes

Le but de ce TP est d'écrire un programme capable de récupérer des informations de la table "employee", afin de les stocker dans des dictionnaires, et d'afficher certaines informations concernant un ou plusieurs employés.

Recopiez maintenant le code suivant (remplace le code précédent):

```
db = connecte_base("foodmart.db")
c = db.cursor()
try:
    c.execute("SELECT employee_id, full_name FROM employee")
    liste_tuples = c.fetchall()
    for t in liste_tuples:
        num_employe = t[0]
        nom = t[1]
        print(num_employe , nom)
except sqlite3.OperationalError as e:
    print("Erreur SQL :" + e.args[0])
```



Explications :

L'objet `liste_tuples` est une liste de tuples correspondant au résultat de la requête.

- L'objet c a une structure de flux.
  - L'opération `c.fetchone()` permet de lire les tuples un par un.
  - L'opération `c.fetchall()` permet de récupérer toutes les tuples dans une liste en une opération.
- Pour afficher tout ou partie des réponses, il suffit parcourir les éléments de `liste_reponses`, du premier au dernier.

```
for t in liste_tuples:
```



```
...
```

Testez cette requête ainsi que les requêtes suivantes :

3.1 - Comptez et affichez le nom, le prénom et le salaire des employés qui gagnent plus de 10000 dollars.

3.2 - Comptez et affichez le nom complet (full_name), la fonction (position_title), la date de naissance (birth_date) et le salaire (salary) de tous les employés supervisés par Mona Jaramillo.

4. Classe Employe

Nous définissons ici une classe `Employe` contenant les informations suivantes :

- nom complet
- fonction
- date de naissance
- salaire



Les classes en Python

Les classes servent à programmer de façon modulaire via la définition d'**objets** ainsi que de **méthodes** agissant sur les objets de la classe.

- Les classes en Python
- Class objects

Les classes servent à définir des **objets**. Les objets sont des variables contenant des données obéissant à un certain format. Ici, un objet de la classe Employé contiendra quatre valeurs correspondant aux quatre attributs définis dans le constructeur `__init__` donné ci-dessous : `nom_complet`, `fonction`, `date_de_naissance` et `salaire`.

Ajoutez dans votre projet un fichier `employe.py` contenant les définitions suivantes :

```
class Employe:
    def __init__(self, nom_complet, fonction, date_de_naissance, salaire):
        self.nom_complet = nom_complet
        self.fonction = fonction
        self.date_de_naissance = date_de_naissance
        self.salaire = salaire
```



- Les noms de classes commencent par une majuscule
- Toutes les classes doivent être testées ! (voir Python, les tests)

4.1 - Testez cette méthode dans le programme principal:

- Importez tout d'abord la classe : `from employe import Employe`



L'importation permet d'utiliser certaines des définitions contenues dans le fichier `employe.py`. Ici on précise qu'il s'agit de la classe `Employe`.

- initialisez un objet e de type Employe à l'aide des attributs `full_name`, `position_title`, `birth_date` et `salary` obtenus dans la question 3.2



Voici la ligne de code permettant d'initialiser l'objet e

```
e = Employe(full_name, position_title, birth_date, salary)
```

Apres initialisation, l'objet e "contient" les quatre valeurs fournies en argument. Ces valeurs sont accessibles une par une par notation "pointée : `e.nom_complet`, `e.fonction`, `e.date_de_naissance` et `e.salaire`.

- Affichez ensuite les valeurs contenues dans l'objet en notation pointée .

4.2 - Définissez selon ce principe la liste des employés supervisés par Mona Jaramillo

4.3 - Ajoutez dans la classe `Employe` une méthode `mieux_paye_que()` qui compare le salaire de deux employés (retourne `True` ou `False`).



Les méthodes sont des fonctions définies dans une classe. Le premier argument d'une méthode (nommé `self`) désigne l'objet sur lequel on applique la méthode. Le deuxième argument désigne un employé auquel on souhaite le comparer.

```
def mieux_paye_que(self, autre_employe):
    ...
```

Pour utiliser une méthode, on utilise également la notation pointée. Ainsi, si `e1` et `e2` sont deux employés, la syntaxe `e1.mieux_paye_que(e2)` correspond à l'appel à la méthode `mieux_paye_que` appliquée à l'employé `e1`. Lors de l'appel, `e1` est le premier argument (`self`) et `e2` est le deuxième argument (`autre_employe`).

Pour comparer les salaires, on utilisera `self.salaire` et `autre_employe.salaire`.

4.3 - Affichez dans le programme principal le nom et le prénom du mieux payé des employés de Mona Jaramillo.

5. Classe Departement

Définir une classe `Departement` contenant les informations suivantes :

- `numDep` (numéro du département)
- `description` (description du département)
- `listeEmployes` (liste des employés appartenant à ce département)
- définir un constructeur (prenant en paramètres le numéro de département et la description)
- définir une méthode `ajouteEmploye()` qui ajoute un employé à la liste des employés.

5.1 - Affichez le contenu de la table Department.

5.2 - Au niveau du programme principal, créez puis remplissez une liste de départements à partir de la table Department.

5.3 - Définir une méthode `coutDep()` calculant le coût salarial d'un département (la somme des salaires de ses personnels). Afficher le coût de chacun des départements.

5.4 - Définir une méthode `plusCouteuxQue()` comparant le coût de deux départements. Afficher la description et le coût du département le moins coûteux et du département le plus coûteux.

1)

autrement dit en français : **Employé**(id_employé, nom_complet, prénom, id_fonction, titre_fonction, id_magasin, id_département, date_naiss, date_embauche, date_fin, salaire, id_supérieur, niveau_educatif, statut_marital, sexe, rôle_managérial)

From:
https://wiki.centrale-med.fr/informatique/ - **WiKi informatique**

Permanent link:
https://wiki.centrale-med.fr/informatique/restricted:tc-d:tp3:travaux_pratiques_troisieme_seance_2017

Last update: **2019/01/13 17:38**