Devoir n°3 Processus & Bases de données 1h45

Exercice 1 – Base de données

On s'intéresse à la gestion de la base de données d'un hôpital. On pourra utiliser les mots-clés SQL suivants : AND, FROM, INSERT, INTO, JOIN, ON, SELECT, SET, UPDATE, VALUES, WHERE.
On utilisera également la fonction d'agrégation COUNT qui renvoie le nombre d'enregistrements correspondant à une requête.

La table `Patient` possède les attributs suivants :

- nom_patient de type TEXT (clé primaire) ;
- `prenom` de type TEXT ;
- `numero_secu` de type INT ;
- age de type INT.

Patient			
nom_patient	prenom	numero_secu	age
Heartman	Alice	207053523800187	17
Douglas	Bob	100017500155572	24
Woods	Caroll	258125930610747	65

La table `Symptome` possède les attributs suivants :

- nom_patient de type TEXT (clé primaire et clé étrangère) ;
- toux de type TEXT ;
- fievre de type TEXT ;
- nausee de type TEXT ;
- anosmie de type TEXT.

Symptome				
nom_patient	toux	fievre	nausee	anosmie
Heartman	Oui	Non	Non	Oui
Douglas	Non	Oui	Oui	Non
Woods	Oui	Oui	Non	Non

La table `Maladie` possède, entre autres, l'attribut `nom_maladie` de type TEXT, qui est la clé primaire. Les autres attributs de cette table ne sont pas représentés car ils ne sont pas utiles pour l'exercice.

Maladie
nom_maladie
Covid-19
Gastroentérite

La table Diagnostic possède les attributs suivants :

- `nom_patient` de type TEXT (clé primaire et clé étrangère) ;
- `nom_maladie` de type TEXT (clé étrangère).

Diagnostic	
nom_patient	nom_maladie
…	…
…	…

1. Écrire une requête SQL permettant d'obtenir les noms et prénoms des patients ayant strictement plus de 60 ans.
2. Alice Heartman ne tousse plus. Ecrire une requête SQL permettant de mettre à jour la base de données avec cette information.
3. On souhaite effectuer des statistiques sur les symptômes des patients atteints de Covid-19. Écrire une requête SQL permettant de connaître le nombre de patients avec un diagnostic de Covid-19 qui toussent.

Un employé de l'hôpital saisit la requête suivante :

```
INSERT INTO Patient VALUES ('Douglas', 'Patrick', 168077230253829, 55)
```

4. Expliquer pourquoi cette requête produit une erreur.
5. Proposer une modification du schéma relationnel qui permettrait de résoudre ce problème.

Exercice 2 – Processus

On souhaite élaborer un programme système permettant de gérer l'ordre d'exécution des processus sur le processeur.

1. Comment s'appelle cette tâche du système d'exploitation ?
2. Donner les différents états possibles d'un processus.

Chaque processus dispose d'une valeur de priorité. Un processus est prioritaire sur un autre processus si sa valeur de priorité est plus petite. Ainsi pour rendre un processus moins prioritaire, il faut augmenter sa valeur de priorité, par exemple en la faisant passer de 2 à 3.

Fonctionnement du programme gérant l'ordre d'exécution des processus :

On dispose d'une liste d'attente dont les éléments sont des sous-listes de processus en attente. La première sous-liste contient les processus ayant la valeur de priorité la plus élevée 0, la seconde ceux ayant la valeur de priorité 1, etc.

À l'arrivée d'un nouveau processus :

- Attribuer au nouveau processus la valeur de priorité 0 ;
- Placer le nouveau processus en dernier dans la sous-liste des processus de priorité 0.

À chaque cycle d'horloge :

S'il n'y a pas de processus en cours d'exécution et s'il reste des processus en attente :

- sélectionner le premier processus de la sous-liste non vide de priorité la plus élevée ;
- élire ce processus comme nouveau processus en cours d'exécution (le processus est retiré de la liste d'attente) ;

Si un processus est en cours d'exécution et qu'il termine son exécution pendant le cycle, le retirer du processeur (ce processus est terminé, il ne revient pas dans la liste d'attente) ;

Si un processus est en cours d'exécution et qu'il ne termine pas son exécution à la fin du cycle d'horloge :

- si des processus de priorité supérieure ou égale attendent :
  - retirer le processus en cours d'exécution du processeur ;
  - réduire sa priorité de 1 et le mettre dans la sous-liste correspondant à sa priorité ;
  - élire le premier processus de la sous-liste non vide de priorité la plus élevée ;
- sinon, réduire sa priorité de 1 et continuer à exécuter ce processus.

3. Parmi les propositions suivantes, donner la structure la plus adaptée pour stocker les processus d'une même priorité : liste, file ou pile.

Pour représenter le processus, on utilise une classe `Processus` qui possède les variables d'instances `PID` (l'identifiant du processus) et `priorite` (la priorité du processus).

4. Compléter le constructeur de la classe `Processus` :

```
class Processus:
    ...(self, ..., priorite):
        ... priorite = priorite
        ... PID = ...
```

5. On considère les trois processus suivants :

```
P1 = Processus(PID=1,priorite=0)
P2 = Processus(PID=2,priorite=0)
P3 = Processus(PID=3,priorite=0)
```

On a donc `liste_files=[[P1, P2, P3], [], []]`.
Compléter la simulation suivante, dans laquelle la variable `CPU` contient le processus en cours d'exécution. On supposera qu'au cours de ces 5 cycles, aucun processus ne se termine.

```
Cycle 1:  CPU = P1    liste_files=[[P2, P3], [], []]
Cycle 2:  CPU = P2    liste_files=[[P3],[P1],[]]
Cycle 3:  CPU = P3    liste_files=[[], [...],[]]
Cycle 4:  CPU = P3    liste_files=[[], [...], []]
Cycle 5:  CPU = ...   liste_files=[[], [...], [...]]
```

Dans les questions 6 et 7, on dispose :

- d'un processus P0 qui nécessite un temps d'utilisation de 10 cycles pour se terminer ;
- de deux processus (P1, P2) nécessitant 2 cycles pour se terminer. On suppose de plus que les processus P1 et P2, une fois terminés, sont remplacés par un nouveau processus similaire (P1 est remplacé par P3, P2 est remplacé par P4, etc).

6. Expliquer pourquoi P0 ne se terminera jamais avec le programme de gestion présenté au début de l'exercice. Indiquer notamment la priorité de P0 au bout de quelques temps.

Pour régler ce problème, on décide d'ajouter la variable d'instance `temps_attente` au processus, et on définit une constante appelée `Max_Temps` qui correspond au temps maximum qu'un processus attend avant de remonter sa priorité. L'idée est qu'à chaque cycle, le `temps_attente` augmente de 1. Lorsque sa valeur dépasse `Max_Temps`, sa priorité augmente.

7. Expliquer pourquoi P0 pourra se terminer avec cet ajout.

8. Écrire la fonction `meilleur_priorite` qui renvoie la priorité de sous-liste la plus prioritaire. S'il n'y a aucun processus dans la file d'attente, elle doit renvoyer `None`.

```
def meilleure_priorite(liste_files):
    ...
```

Exemple :

```
# p1, p2 et p3 sont des instances de la classe 'Processus'
>>> liste_files = [[], [p2], [p3, p1]]
>>> meilleur_priorite(liste_files)
1
```

9. Écrire la fonction `prioritaire` qui renvoie le premier processus de la sous-liste la plus prioritaire (la fonction prioritaire supprimera le processus choisi de la file d'attente dans laquelle il se trouvait). S'il n'y a aucun processus dans la file d'attente, elle doit renvoyer `None`.

```
def prioritaire(liste_files):
    ...
```

On pourra utiliser `liste.pop(i)` pour renvoyer l'élément de la liste à l'indice `i`, tout en le supprimant de la liste.

10. Écrire une fonction `gerer` qui reçoit le processus en cours d'exécution `p` ainsi que la liste d'attente `liste_files` et qui, si `p` doit être interrompu pour être remplacé par un processus prioritaire, place `p` dans la bonne sous-liste de `liste_files` et renvoie le nouveau processus à élire. Si `p` reste élu, la fonction ne fait rien d'autre que diminuer la priorité de `p`.

On fera les hypothèses simplificatrices suivantes :

- il y a toujours un processus en cours d'exécution (`p` ne peut pas être `None`)
- Dans `liste_files`, il y a toujours une sous-liste existante pour une priorité donnée (il est inutile de prévoir le cas où il faut créer cette sous-liste si elle n'existe pas).

```
def gerer(p, liste_files):
    ...
```

Correction

Ex.1

1. SELECT nom_patient, prenom FROM Patient WHERE age > 60 **[1]**

2. UPDATE Symptome SET toux = "Non" WHERE nom_patient = "Heartman" **[1]**

D si seulement UPDATE

3. SELECT COUNT(*) FROM Diagnostic
JOIN Symptome ON Diagnostic.nom_patient = Symptome.nom_patient
WHERE Symptome.toux = "Oui" AND Diagnostic.nom_maladie = "Covid-19". **[2]**

0 si pas de JOIN

4. Nom_patient est la clé primaire de la table Patient. Or, la valeur "Douglas" existe déjà, donc elle ne peut pas être introduite une deuxième fois, car la valeur d'une clé primaire *doit* être unique. **[1]**

-1 si le terme « clé primaire » n'apparaît pas dans la réponse

5. Le plus simple est de créer un attribut id unique pour chaque patient. **[1]**

Ex.2

1. Ordonnancement **[0,5]**

2. Prêt, bloqué, élu **[0,5]**

3. file (car on prend les premiers éléments des sous-listes à chaque fois) **[0,5]**

Pas de justification attendue

4. classe `Processus` à compléter **[1]**

```
class Processus:
    def _init_(self, PID, priorite):
      self.priorite = priorite
      self.PID = PID
```

5. Simulation à compléter **[1,5]**

```
Cycle 1:  CPU = P1    liste_files=[[P2, P3], [], []]
Cycle 2:  CPU = P2    liste_files=[[P3],[P1],[]]
Cycle 3:  CPU = P3    liste_files=[[], [P1, P2],[]]
Cycle 4:  CPU = P3    liste_files=[[], [P1, P2], []]
Cycle 5:  CPU = P1    liste_files=[[], [P2], [P3]]
```

6. P0 va être traité pendant un cycle, puis ça sera le tour de P1, puis de P2. Au 4e cycle, P0 sera de nouveau traité, avec une priorité de 1. P1 et P2 seront en attente dans la sous-liste de priorité 1.

Au cycle 5, P0 sera placé en attente dans la sous-liste de priorité 2. Mais P1 et P2 se termineront et seront remplacés par d'autres processus qui auront une priorité de 0.

Leur priorité n'aura jamais le temps d'arriver à 2, car ils terminent leur exécution avant et sont remplacés par de nouveau processus de priorité 0. Ainsi, une fois que P0 est dans la sous-liste de priorité 2, il ne sera plus jamais élu dans cette situation. **[1,5]**

7. Au bout d'un certain temps, la priorité de P0 remonte à 1 et il pourra de nouveau avoir un cycle où il sera élu. Ce fonctionnement durera jusqu'à ce que P0 soit terminé. **[0,5]**

8. Fonction meilleure priorité **[2]**

```
def meilleure_priorite(liste_files):
  for i, sl in enumerate(liste_files):
    if len(sl): return i
  return None
```

9. Fonction prioritaire **[2]**

```
1  def prioritaire(liste_files):
2    mp = meilleure_priorite(liste_files)
3    if mp is None: return None
4    return liste_files[mp].pop(0)
```

B si manque le « pop »

10. Fonction gerer **[3]**

```
1  def gerer(p, liste_files):
2    p.priorite += 1
3    mp = meilleure_priorite(liste_files)
4    if not mp is None and p.priorite >= mp:
5      liste_files[mp].append(p)
6      return prioritaire(liste_files)
```