CONSIGNES GÉNÉRALES

DOCUMENTS NON AUTORISES
CE CAHIER D'EXAMEN COMPORTE 17 PAGES + 2 PAGES SUPPLÉMENTAIRES
LES RÉPONSES DOIVENT ÊTRE ÉCRITES DANS
LES ESPACES RÉPONSES DÉDIÉES
EN CAS DE BESOIN UTILISER LES PAGES VIDES EN FIN DU CAHIER, DANS
CE CAS, IL FAUT LE SIGNALER DANS L'ESPACE RÉPONSE CORRESPONDANT
L'USAGE DES CALCULATRICES EST INTERDIT
Il FAUT RESPECTER IMPÉRATIVEMENT LES NOTATIONS DE L'ÉNONCÉ
VOUS POUVEZ ÉVENTUELLEMENT UTILISER L'ANNEXE

Le sujet comporte deux parties qui traitent les aspects suivants :

Partie I : Programmation procédurale (Q1..Q7).

Partie II : Base de données relationnelle (Q8..Q21).

Partie I : Programmation Procédurale

Dans cette partie il s'agit de mettre en place un système de recommandation élémentaire. Les systèmes de recommandation sont des algorithmes d'intelligence artificielle utilisés pour suggérer des éléments pertinents aux internautes par rapport à leurs préférences. Ils sont largement utilisés dans de nombreux domaines tels que le commerce électronique, les services de streaming de musique et de vidéo, les applications de voyage, l'éducation et bien d'autres.

Dans le présent sujet on s'intéresse à recommander des cours en ligne pour des étudiants. La méthode de recommandation adoptée est connue sous le nom de IBCF "Item-Based Collaborative Filtering". La méthode exploite des données qui représentent les préférences des étudiants pour certains cours sous forme de scores. Les données peuvent être organisées dans une matrice où les lignes représentent les étudiants, les colonnes représentent les cours et les entrées de la matrice représentent les évaluations.

Exemple 1:

		Math	History	Programming	English	Chemistry	Art
	User1	4	3	5	–	2	1
	User2	–	2	4	3	–	5
	User3	5	–	4	2	3	–
data =	User4	3	4	–	5	1	2
	User5	2	–	3	1	–	4
	User6	1	5	2	4	5	3
	User7	–	3	–	2	4	1
	User8	4	–	1	5	2	3

FIGURE 1 – Exemple de matrice data.

Dans cet exemple :

— Chaque ligne i représente un étudiant user_i

— Chaque colonne j représente un cours item_j.

— Un score data_{ij} est donné par un étudiant i pour un cours j sur une échelle de 1 à 5., où "-" indique aucune évaluation

Cette matrice est dite creuse car elle présente un nombre important d'entrées vides qui représentent le fait qu'un étudiant n'interagit qu'avec un nombre très restreint de cours. L'objectif de la méthode de recommandation basée sur IBCF est de prédire les scores manquants en fonction de la similitude entre les cours ce qui aide à recommander à un étudiant donné les cours qui pourraitent l'intéresser le plus.

On suppose que les données d'interaction étudiant-cours sont stockées dans un fichier texte ayant le format suivant :

```
4,3,5,-,2,1
-,2,4,3,-,5
5,-,4,2,3,-
3,4,-,5,1,2
2,-,3,1,-,4
1,5,2,4,5,3
-,3,-,2,4,1
4,-,1,5,2,3
```

FIGURE 2 – Exemple d'un fichier d'interaction (étudiant-cours)

Chaque ligne représente un étudiant, et les valeurs sont séparées par des virgules, où "-" indique aucune interaction.
N.B pour ce problème une matrice est représentée sous forme d'une liste de listes où chaque liste fille (sous-liste) représente une ligne de la matrice.

Travail demandé

On propose d'implémenter les fonctions suivantes :

Q1. Écrire une fonction nommée `loadData` qui prend en entrée une chaîne de caractères `nomF` contenant le nom du fichier et qui retourne une liste de listes `data` contenant les données chargées du fichier. Dans cette liste, les valeurs doivent être converties en des entiers, et l'absence de valeur (marqué par "-") doit être représentée par `None`. Pour l'exemple de fichier de la figure 2 `data` sera comme suit :

```
[[4, 3, 5, None, 2, 1],
[None, 2, 4, 3, None, 5],
[5, None, 4, 2, 3, None],
[3, 4, None, 5, 1, 2],
[2, None, 3, 1, None, 4],
[1, 5, 2, 4, 5, 3],
[None, 3, None, 2, 4, 1],
[4, None, 1, 5, 2, 3]]
```

FIGURE 3 – résultat de la fonction `loadData` pour les données de la figure 2.

Espace de réponse supplémentaire

Espace de réponse supplémentaire

Espace de réponse pour Q1

```
def loadData(nomF):
```

Pour la recommandation des cours, on calcule généralement leurs scores moyens. L'idée est de trouver le score moyen pour chaque cours en considérant les évaluations de tous les étudiants. Ce score moyen servira ensuite de facteur de standardisation qui reflète la popularité du cours ou du niveau de satisfaction moyen auprès de tous les étudiants présents dans l'échantillon.

Voici les étapes générales qu'on va suivre :

1. Transposer la matrice data.
2. Calculer le score moyen pour chaque cours.
3. Standardiser les scores en soustrayant le score moyen de chaque cours des scores des étudiants correspondants.
4. Calculer la similarité cosinus entre chaque paire de cours basées sur les scores standardisés et les stocker dans une matrice.

Ce processus aide à identifier les cours qui ont des schémas de notation similaires par les étudiants. Une fois qu'on obtient la matrice de similarité cosinus, on pourra l'utiliser pour faire des recommandations.

Q2. Écrire une fonction `tranpose` qui prend en entrée la liste `data` et retourne une liste `tdata` représentant la matrice transposée (voir figure 4).

```
[[4, None, 5, 3, 2, 1, None, 4],
[3, 2, None, 4, None, 5, 3, None],
[5, 4, 4, None, 3, 2, None, 1],
[None, 3, 2, 5, 1, 4, 2, 5],
[2, None, 3, 1, None, 5, 4, 2],
[1, 5, None, 2, 4, 3, 1, 3]]
```

FIGURE 4 – liste résultante de la fonction `tranpose`.

Espace de réponse pour Q2

```
def transpose(data):
```

L'objectif de cette étape est de calculer le score moyen pour chaque cours. Le score moyen pour un cours est obtenu en additionnant tous les scores attribués à ce cours et en divisant la somme obtenue par le nombre total des scores non vides. Mathématiquement, le score moyen μ pour un cours i est calculé en utilisant la formule suivante :

$$\mu_i = \frac{\sum_{j \in \mathcal{U}_i} tdata_{ij}}{|\mathcal{U}_i|}$$

Équation 1 – Score moyen par cours.

où :

- μ_i est le score moyen pour le cours i,
- $\mathcal{U}_i$ est l'ensemble des étudiants qui ont évalué le cours i,
- tdata_{ij} est le score attribué par l'étudiant j au cours i,
- $|\mathcal{U}_i|$ est le nombre total de scores non vides pour le cours i.

Ce score moyen est ensuite utilisé pour standardiser le score de chaque cours pendant le processus de filtrage collaboratif.

Q3. Écrire une fonction `meanPerCourse` qui prend en entrée `tdata` calcule et retourne la liste `means` contenant le score moyen pour chaque cours (voir figure 5).

```
resultat = [3.1666666666666665, 3.4,
3.1666666666666665, 3.142857142857143,
2.8333333333333335, 2.7142857142857144]
```

FIGURE 5 – résultat de la fonction `meanPerCourse`.

Espace de réponse pour Q21

```
def DCavailability(cur, dataCenterID):
```

Espace de réponse pour Q20

```
def DCStatutTimeSpan(cur, dataCenterID):
```

Q21. Écrire la fonction `DCavailability` qui prend deux paramètres `cur`et `dataCenterID`, l'identifiant d'un centre de calcul, et retourne `disponibility`, le ratio de disponibilité de ce centre de calcul. Sachant que le ratio de disponibilité d'un centre de calcul est le rapport entre toutes les durées de temps cumulées de ses serveurs pour le statut `"En ligne"`, sur la somme des durées de temps cumulées pour tous les états possibles.

$$\text{Disponibility} = \frac{D_{\text{en ligne}}}{D_{\text{total}}}$$

— $D_{\text{en ligne}}$ représente la durée cumulée où le datacenter était en ligne,
— D_{total} représente la somme de toutes les durées cumulées pour tous les statuts possibles.

Pour exprimer D_{total} sous forme d'une équation, on peut écrire :

$$D_{\text{total}} = \sum_{s \in \text{Statuts}} D_s$$

Espace de réponse pour Q3

```
def meanPerCourse(tdata):
```

Dans la troisième étape, on va standardiser les notes en soustrayant le score moyen de chaque cours à partir des scores qui lui ont été attribués par les étudiants.

Q4. Écrire une fonction `standardizeRatings` qui prend en entrée deux listes :
— `data` qui représente les scores des cours attribués par les étudiants (Q1)
— `means` qui représente le score moyen de chaque cours (Q3)
et qui retourne une liste `sdata` contenant les scores standardisés.

Espace de réponse pour Q4

```
def standardizeRatings(data, means):
```

La quatrième étape consiste à calculer la similarité cosinus entre les cours en se basant sur les scores standardisés obtenus dans l'étape précédente. La similarité cosinus mesure la similitude entre deux vecteurs, et dans ce cas, ces vecteurs représentent les scores standardisés de deux cours. La similarité cosinus entre deux cours j_1 et j_2 est calculée à l'aide de la formule suivante :

$$\text{CosineSimilarity}(j_1, j_2) = \frac{\sum_{i \in \mathcal{U}_{j_1} \cap \mathcal{U}_{j_2}} sdata_{ij_1} sdata_{ij_2}}{\sqrt{\sum_{i \in \mathcal{U}_{j_1} \cap \mathcal{U}_{j_2}} sdata_{ij_1}^2} \sqrt{\sum_{i \in \mathcal{U}_{j_1} \cap \mathcal{U}_{j_2}} sdata_{ij_2}^2}}$$

Équation 2 – Similarité cosinus

où :

- sdata_{ij_1} est le score normalisé attribué par l'utilisateur i au cours j_1.
- sdata_{ij_2} est le score normalisé attribué par l'utilisateur i au cours j_2.

Cette formule calcule le cosinus de l'angle entre les vecteurs représentant les scores des deux cours j_1 et j_2. Plus la similarité cosinus est élevée plus la similitude entre deux cours est dite forte.

Q5. Écrire une fonction `cosineSimilarity` qui prend en entrée deux listes représentant les scores standardisés de deux cours (deux colonnes `course1` et `course2` de la liste `sdata`) et retourne un réel quantifiant la similarité cosinus entre ces deux cours. Les valeurs `None` doivent être ignorés dans les calculs.

Espace de réponse pour Q5

```
def cosineSimilarity(course1, course2):
```


Espace de réponse pour Q18

SQLITE

Dans la suite, les fonctions demandées doivent être écrites en Python en désignant par `cur` le curseur d'exécution des requêtes.

Q19. Écrire la fonction `ServerMonitoring` qui prend deux paramètres `cur` et `serverID` l'identifiant d'un serveur et retourne un dictionnaire `dStat` où les clés sont les statuts qui ont été observés durant le monitoring de ce serveur et les valeurs sont des ensembles contenant les dates et heures des observations de ces statuts.

Espace de réponse pour Q19

```
def ServerMonitoring(cur, serverID):
```

Q20. Écrire la fonction `DCStatutTimeSpan` qui prend deux paramètres `cur` et `dataCenterID` l'identifiant d'un centre de calcul et retourne un dictionnaire `ddcStat` où les clés sont les statuts qui ont été observés durant le monitoring des serveurs installés dans ce centre de calcul (`"En ligne"`, `"Hors ligne"`, `"En maintenance"`, `"En panne"`, `"En surcharge"`) et les valeurs sont les durées de temps cumulés de tous les serveurs pour chaque statut.

Espace de réponse pour Q15

Q16. Déterminer toutes les informations relatives aux serveurs qui hébergent des logiciels avec des licences de type `"logiciel libre"`.

Espace de réponse pour Q16

Q17. Déterminer toutes les informations relatives aux logiciels qui ont une licence expirée. N.B la fonction prédéfinie `CURRENT_DATE` retourne la date courante.

Espace de réponse pour Q17

Q18. Déterminer les identifiants, les capacités serveurs et les capacités disponibles (égale à la capacité serveur du DataCenter – nombre de serveurs effectivement affectés à ce DataCenter) le résultat sera trié par ordre décroissant des capacités disponibles.

Q6. Écrire une fonction `cosineSimilarityMatrix` qui prend en entrée la liste `sdata` et retourne une nouvelle liste `sMatrix` qui contient les similarités cosinus de tous les couples des cours.

Espace de réponse pour Q6

```
def cosineSimilarityMatrix(sdata):
```

On arrive à l'étape de prédiction qui consiste à :

- Pour chaque étudiant i trouver la liste `selected` contenant les indices k des cours avec lesquels il a interagit.
- Pour chaque cours j tel que l'étudiant i n'a pas interagit avec le cours j, calculer la prédiction en appliquant l'équation suivante :

$$\text{prediction}_{ij} = \frac{\sum_{k \in \text{selected}} \text{sMatrix}_{jk}\ \text{data}_{ik}}{\sum_{k \in \text{selected}} |\text{sMatrix}_{jk}|}$$

Équation 3 – prédiction du score d'un étudiant i pour un cours j

où :

- prediction_{ij} est la prédiction du score de l'étudiant i pour le cours j.
- data_{ik} est le score effectif de l'étudiant i pour le cours k.
- sMatrix_{jk} est la similarité entre le cours j et le cours k.

Q7. Écrire une fonction `predict` qui prend en entrée la liste `data` et la liste `sMatrix`. Cette fonction devrait retourner une liste `predictions` où les valeurs `None` de la liste `data` sont remplacées par les valeurs prédites selon le processus IBCF. Pour l'exemple de la figure 3 la fonction doit retourner :

```
predictions = [ [4, 3, 5, 3.5392879769922487, 2, 1],
                [2.1775466724990715, 2, 4, 3, 4.819885217424523, 5],
                [5, 8.182699199086588, 4, 2, 3, 4.245806057606589],
                [3, 4, 3.832527965142944, 5, 1, 2],
                [2, 3.4673236699753405, 3, 1, 1.634746221774161, 4],
                [1, 5, 2, 4, 5, 3],
                [2.9687604890559744, 3, 2.513364579350161, 2, 4, 1],
                [4, 1.4500871129392305, 1, 5, 2, 3] ]
```

FIGURE 6 – résultat de la prédiction selon IBCF.

Espace de réponse pour Q7

```
def predict(data, sMatrix):
```


Espace de réponse pour Q12

Q13. Déterminer tous les détails relatifs aux logiciels qui pour le moment n'ont aucune licence.

Espace de réponse pour Q13

Q14. Pour chaque statut déterminer le nombre de serveurs qui ont actuellement ce statut.

Espace de réponse pour Q14

Q15. Pour chaque nom de centre de calcul et pour chaque statut donner le nombre de serveurs qui ont actuellement ce statut.

SQL

Q10. Écrire la requête qui permet de créer la table `Affectations` en supposant que toutes les tables référencées sont déjà créées.

Espace de réponse pour Q10

Dans la suite on suppose que les tables de la base de données sont remplies en respectant les contraintes et les règles de gestion su citées.

Q11. Calculer la capacité totale de stockage pour chaque centre de calcul.

Espace de réponse pour Q11

Q12. Déterminer les id des centres de données qui sont saturés par rapport aux nombres de serveurs.

Partie II : Base de Données Relationnelle

Le cloud computing fait référence à l'utilisation de la mémoire et des capacités de calcul des ordinateurs et des serveurs répartis dans le monde entier et liés par un réseau. Les applications et les données ne se trouvent plus sur un ordinateur déterminé d'une entreprise donnée mais dans un cloud composé de nombreux serveurs distants interconnectés formant un centre de calcul réparti. Les entreprises louent donc du matériel, des applications et des services chez des fournisseurs cloud computing pour réduire leurs charges de développement IT.

L'objectif de cette partie est de gérer la consommation des entreprises clientes en terme de services et de ressources matérielles.

Le schéma relationnel de la base de données décrit ci-dessous a été élaboré pour les besoins de l'énoncé.

- `DataCenters(DataCenterID, NomDC, Emplacement, CapacitéServ)`
  La table `DataCenters` enregistre les informations relatives aux centres de calcul qui offrent leurs services et ressources.
  - `DataCenterID` : identifiant d'un centre de calcul, de type entier clé primaire.
  - `NomDC` : nom du centre de calcul, de type chaîne de caractères.
  - `Emplacement` : désigne l'emplacement physique du centre de calcul, de type chaîne de caractères.
  - `CapacitéServ` : le nombre maximum de serveurs qu'un centre de calcul peut contenir, de type entier.
- `Serveurs(ServeurID, NomSrv, AdresseIP, CapacitéStk, Statut, #DataCenterID)`
  La table `Serveurs` enregistre les informations relatives aux serveurs installés dans les différents centres de données.
  - `ServeurID` : utilisé pour identifier de manière unique chaque serveur, de type entier clé primaire.
  - `NomSrv` : nom du serveur, de type chaîne de caractères.
  - `AdresseIP` : c'est l'adresse IP du serveur, de type chaîne de caractères.
  - `CapacitéStk` : indique la capacité de stockage du serveur, exprimée en téraoctets, de type entier.
  - `Statut` : représente l'état actuel du serveur, par exemple, s'il est en ligne, hors ligne, en maintenance, etc., de type chaîne de caractères.
  - `DataCenterID` : identifiant d'un centre de calcul de type entier, clé étrangère qui fait référence à la table DataCenter.
- `Logiciels(LogicielID, NomLog, Version, Editeur, Type)`
  La table `Logiciels` enregistre les informations relatives aux applications offertes par le centre de calcul.
  - `LogicielID` : utilisé pour identifier de manière unique chaque logiciel, de type entier clé primaire.
  - `NomLog` : nom du logiciel, de type chaîne de caractères.
  - `Version` : version spécifique actuelle du logiciel, de type chaîne de caractères.
  - `Editeur` : nom de l'entreprise ou de l'individu qui a développé ou édité le logiciel, de type chaîne de caractères.

— `Type` : type du logiciel, par exemple, système d'exploitation, application de commerce électronique, application de gestion de stock, utilitaire, etc., de type chaîne de caractères.

- `Licences(LicenceID, TypeLicence, DateExp, #LogicielID)`
  La table `Licences` enregistre les informations relatives aux licences permises aux différents logiciels. Certains logiciels libres sont livrés sans licences.
  — `LicenceID` : utilisé pour identifier de manière unique chaque licence, de type entier clé primaire.
  — `TypeLicence` : représente le type de distribution d'une licence, par exemple, propriétaire, open source, libre, commercial, etc., de type chaîne de caractères.
  — `DateExp` : date à laquelle la licence expire, indiquant jusqu'à quand le logiciel peut être légalement utilisé avec cette licence, de type date selon le format `"AAAA-MM-JJ"`.
  — `LogicielID` : identifiant du logiciel associé à une licence donnée, clé étrangère qui fait référence à la table Logiciels.
- `Utilisateurs(UserID, NomUser, Mail, Rôle)`
  La table `Utilisateurs` enregistre les informations relatives aux entreprises qui louent des services de Cloud Computing.
  — `UserID` : identifiant de l'entreprise locataire d'un ou plusieurs services, de type entier clé primaire.
  — `NomUser` : nom de l'entreprise locataire, de type chaîne de caractères.
  — `Mail` : adresse mail de l'entreprise locataire, de type chaîne de caractères.
  — `Rôle` : activité économique de l'entreprise locataire, par exemple, une banque, une assurance, une boîte de développement, etc., de type chaîne de caractères.
- `Affectations(AffectationID, DateAffect, DuréeEstimée, #UserID, #ServeurID, #LicenceID)`
  La table `Affectations` enregistre les informations relatives à un acte de location qui se traduit par une affectation d'une licence logiciel installée sur un serveur pour un utilisateur donné.
  — `AffectationID` : utilisé pour identifier de manière unique chaque affectation de type entier, clé primaire.
  — `DateAffect` : Indique la date à laquelle l'affectation a été effectuée, de type date selon le format `"AAAA-MM-JJ"`.
  — `DuréeEstimée` : Représente la durée estimée de l'affectation, exprimée en heures, de type entier.
  — `UserID` : identifiant de l'entreprise locataire, de type entier clé étrangère qui fait référence à la table Utilisateurs.
  — `ServeurID` : identifiant d'un serveur sur lequel les données de l'utilisateur sont enregistrées, de type entier clé étrangère qui fait référence à la table Serveurs.
  — `LicenceID` : identifiant de la licence louée par un utilisateur, de type entier, clé étrangère qui fait référence à la table Licences.
- `Monitoring(MonitoringID, DateSurv, Statut, Commentaires, DuréeStatut, #UserID, #ServeurID)`
  La table `Monitoring` enregistre les informations relatives à des prélèvements réguliers pour assurer la surveillance de performances et de disponibilité des serveurs pour les utilisateurs. C'est le basculement de statut d'un serveur qui provoque l'insertion d'un nouveau tuple dans cette base.

— `MonitoringID` : utilisé pour identifier de manière unique chaque enregistrement de surveillance, de type entier clé primaire.
— `DateSurv` : indique la date et l'heure auxquelles la surveillance a été effectuée, de type date et heure selon le format `"AAAA-MM-JJ HH:MM:SS"`.
— `Statut` : Il indique l'état opérationnel du serveur au moment de la surveillance, ce qui peut être utile pour analyser les performances et la disponibilité du serveur au fil du temps. Ce champ pourrait inclure des valeurs telles que `"En ligne"`, `"Hors ligne"`, `"En maintenance"`, `"En panne"`, `"En surcharge"`, etc., de type chaîne de caractères.
— `Commentaires` : champ permettant d'enregistrer des commentaires ou des notes sur le déroulement de la surveillance, de type chaîne de caractères.
— `DuréeStatut` : La durée exprimée en millisecondes pendant laquelle le serveur reste dans le statut actuel avant de basculer vers un autre statut, de type entier.
— `UserID` : identifiant de l'entreprise locataire de type entier clé étrangère qui fait référence à la table Utilisateurs.
— `ServeurID` : identifiant d'un serveur sur lequel les données de l'utilisateur sont enregistrées de type entier, clé étrangère qui fait référence à la table Serveurs.

Travail demandé

Algèbre relationnelle

Exprimer en algèbre relationnelle les requêtes permettant de :

Q8. Lister les identifiants des serveurs avec leurs capacités de stockage.

Espace de réponse pour Q8

Q9. Lister les identifiants et les adresses IP des serveurs qui hébergent des logiciels utilisés par des entreprises qui œuvrent dans le domaine de l'assurance.

Espace de réponse pour Q9