ÉCOLE POLYTECHNIQUE — ÉCOLES NORMALES SUPÉRIEURES
ÉCOLE SUPÉRIEURE DE PHYSIQUE ET DE CHIMIE INDUSTRIELLES

CONCOURS D'ADMISSION 2018 **FILIÈRES MP HORS SPECIALITÉ INFO PC et PSI**

COMPOSITION D'INFORMATIQUE – B – (XELCR)

(Durée : 2 heures)

L'utilisation des calculatrices **n'est pas autorisée** pour cette épreuve.
Le langage de programmation sera **obligatoirement** PYTHON.

⋆ ⋆ ⋆

Introduction

Une requête sur une base de données est décrite au moyen d'un langage *déclaratif*. Le langage SQL est le plus connu. Pour évaluer une requête, un système de gestion de base de données (SGBD) établit un plan d'exécution combinant les opérateurs de l'*algèbre relationnelle*. L'objectif de ce sujet est l'étude de ces opérateurs.

Nous étudierons en partie I l'implémentation en Python de ces opérateurs. Nous appliquerons ensuite en partie II ces résultats à des requêtes SQL. Nous verrons en partie III comment il est possible de tirer parti des propriétés des données pour améliorer les performances.

Les parties peuvent être traitées indépendamment. Néanmoins, chaque partie utilise des notations et des fonctions introduites dans les parties précédentes.

Notion de complexité. La complexité, ou le coût, d'un algorithme ou d'une fonction Python est le nombre d'opérations élémentaires nécessaires à son exécution dans le pire cas. Lorsque cette complexité dépend d'un ensemble de paramètres $(n, p, \dots)$, on pourra donner cette estimation sous forme asymptotique. On rappelle qu'une application $c(n, p, \dots)$ est dans la classe $\mathcal{O}(f)$ s'il existe une constante $\alpha > 0$ telle que $|c(n, p, \dots)| < \alpha \times f(n, p, \dots)$, pour toutes les valeurs de $n, p, \dots$ assez grandes. Nous préciserons plus loin le coût de chacune des opérations utilisées dans ce sujet.

Bases de données, tables, attributs, enregistrements

Nous détaillons ici la représentation des données dans le modèle relationnel. Une *base de données* est un ensemble de *tables*. Chaque table porte un nom et est associée à un vecteur d'*attributs* de longueur au moins 1. Le nombre d'attributs d'une table est appelé l'*arité* de la table. Le vecteur des attributs $\langle a_0, a_1, \dots, a_{k-1} \rangle$ d'une table T d'arité k est noté $\text{attributs}(T)$ et la table est notée $T[\![a_0, a_1, \dots, a_{k-1}]\!]$.

Une table $T[\![a_0, a_1, \ldots, a_{k-1}]\!]$ est constituée d'*enregistrements*. La *taille* d'une table est le nombre des enregistrements qu'elle contient. Dans ce sujet, nous considérerons qu'un enregistrement est un vecteur $\langle v_0, v_1, \ldots, v_{k-1} \rangle$ de longueur l'arité k de la table. Chaque élément de ce vecteur est la valeur de cet enregistrement par rapport à l'attribut correspondant de la table. La valeur v_i à l'indice i de l'enregistrement est la valeur associée à l'attribut a_i à l'indice i du vecteur d'attributs de cette table. On pourra donc identifier un attribut et son indice et parler de *la valeur d'un enregistrement associée à un indice*. La valeur d'un enregistrement e associée à un indice i est notée $e[i]$.

Nous considérons dans ce sujet que toutes les valeurs d'attributs sont des chaînes de caractères et que la comparaison entre deux valeurs d'un attribut a un coût unitaire quelles que soient ces valeurs.

Deux enregistrements représentés par des vecteurs contenant les mêmes valeurs aux mêmes indices sont *égaux*.

Une table peut contenir des enregistrements égaux. L'élimination des enregistrements égaux est une opération complexe qui est l'objet de l'opérateur SQL appelé `DISTINCT` *que nous étudierons plus loin.*

Exemple (Tables et enregistrements)**.** Considérons une agence de voyages qui vend des trajets et des chambres d'hôtel. La table

$$\mathsf{Vehicule}[\![\mathsf{IdVehicule}, \mathsf{Type}, \mathsf{Compagnie}]\!]$$

contient les données relatives aux divers véhicules disponibles. Pour chaque enregistrement e représentant un véhicule dans la table Vehicule, la valeur de e associée à l'attribut IdVehicule est l'identifiant du véhicule ; la valeur de e associée à l'attribut Type est le type de véhicule ; la valeur associée à l'attribut Compagnie est le nom de la compagnie qui gère ce véhicule.

Cette table contient trois *enregistrements* qui décrivent des véhicules : un bus de la compagnie *IBUS*, un train de la compagnie *SNCF* et un avion de la compagnie *Hop !*.

$$\langle 98300, \mathsf{Bus}, \mathsf{IBUS} \rangle$$
$$\langle 1562, \mathsf{TGV}, \mathsf{SNCF} \rangle$$
$$\langle 30990, \mathsf{A320}, \mathsf{Hop\,!} \rangle$$

Considérons d'autre part la table

$$\mathsf{Trajet}[\![\mathsf{IdTrajet}, \mathsf{VilleD}, \mathsf{VilleA}, \mathsf{DateD}, \mathsf{HeureD}, \mathsf{IdVehicule}]\!]$$

Cette table contient les trajets élémentaires possibles avec les valeurs des attributs associés : l'identifiant du trajet, la ville de départ, la ville d'arrivée, la date du départ de ce trajet, l'heure de départ du trajet, l'identifiant du véhicule utilisé pour le trajet. On rappelle que toutes ces valeurs sont des chaînes de caractères.

Cette table contient 3 trajets possibles le *5 octobre 2016* pour aller de Lille à Rennes. Ils partent respectivement à *9h00*, à *10h00* et à *14h00*.

$$\langle \mathsf{Trajet1}, \mathsf{Lille}, \mathsf{Rennes}, \mathsf{5\ oct.\ 2016}, \mathsf{09h00}, 30990 \rangle$$
$$\langle \mathsf{Trajet2}, \mathsf{Lille}, \mathsf{Rennes}, \mathsf{5\ oct.\ 2016}, \mathsf{10h00}, 98300 \rangle$$
$$\langle \mathsf{Trajet3}, \mathsf{Lille}, \mathsf{Rennes}, \mathsf{5\ oct.\ 2016}, \mathsf{14h00}, 1562 \rangle$$

Représentation des tables et des enregistrements en Python

Dans ce sujet, nous représentons un enregistrement d'une table d'arité k par une *liste Python* de longueur k. L'élément d'indice i de cette liste représente la valeur de l'enregistrement pour l'attribut d'indice i de la table.

Nous représentons une table d'arité k par une liste d'enregistrements. Une table vide est représentée par une liste vide.

Voici par exemple une représentation en Python de la table Vehicule d'arité 3.

```
>>> Vehicule
[['98300', 'Bus', 'IBUS'], ['1562', 'TGV', 'SNCF'], ['30990', 'A320', 'Hop!']]
```

Notez qu'une table peut être représentée par plusieurs listes différentes. Voici une autre représentation possible de cette table.

```
>>> Vehicule
[['1562', 'TGV', 'SNCF'], ['98300', 'Bus', 'IBUS'], ['30990', 'A320', 'Hop!']]
```

Rappel sur les listes Python

Nous rappelons brièvement les opérateurs sur les listes en Python.

Il est attendu que les candidats rédigent leurs réponses à l'aide de ces fonctions seulement. En particulier, l'opérateur d'égalité entre listes ne doit pas être utilisé.

Longueur. L'opération `len(l)` renvoie la longueur de la liste `l`. On considérera que cette opération a un coût unitaire.

Ajout. L'opération `l.append(x)` ajoute l'élément `x` à la fin de la liste `l`. On considérera que cette opération a un coût unitaire, indépendamment de la longueur de la liste et de la valeur de l'élément.

Extraction. L'opération `l.pop()` enlève le dernier élément de la liste `l` et renvoie cet élément. Une erreur est signalée si la liste est vide. On considérera que cette opération a un coût unitaire, indépendamment de la longueur de la liste et de la valeur de l'élément.

Accès. L'opération `l[i]` renvoie l'élément d'indice i de la liste `l` de longueur n. Cette opération ne peut être utilisée dans ce cadre qu'avec un indice compris entre 0 et $n-1$. On considérera que cette opération a un coût unitaire, indépendamment de la longueur de la liste et de la valeur de l'indice.

Concaténation. L'opération `l1 + l2` renvoie la concaténation des deux listes `l1` et `l2`. Les listes ne sont pas modifiées. On considérera que cette opération a un coût unitaire.

Itération. Il est possible de parcourir une liste `l` par la commande d'itération

```
for x in l: ...
```

Ce parcours respecte l'ordre des éléments apparaissant dans la liste. On considérera que le coût d'un parcours est la somme des coûts des opérations effectuées, sans surcoût additionnel.

I Implémentation des opérateurs de l'algèbre relationnelle en Python

Dans toute la suite, on supposera que les arguments des fonctions Python à rédiger sont bien formés : toutes les listes représentant les enregistrements d'une table ont la même longueur qui est l'arité de cette table, les entiers représentant des indices d'attributs appartiennent bien à l'intervalle attendu, etc.

Sélection avec test d'égalité à une constante

L'opérateur $\sigma_{\text{Constante}}$ prend en arguments une table T d'enregistrements, un attribut de cette table identifié à son indice i dans le vecteur $\text{attributs}(T)$ et une valeur c.

Il renvoie une table T' associée aux mêmes attributs que T. Elle est constituée des enregistrements de T tels que la valeur de l'attribut d'indice i est égale à la valeur c. Cette table peut être vide.

Exemple. $\sigma_{\text{Constante}}(\text{Trajet}, 1, \text{Lille})$ renvoie une table T' avec les mêmes attributs que la table Trajet. Elle contient tous les voyages dont la ville de départ est Lille. Dans notre exemple, c'est le cas de tous les voyages. La table T' contient donc les mêmes enregistrements que la table Trajet.

Question I.1. *Implémentez la fonction*

```
SelectionConstante(table, indice, constante)
```

qui prend en arguments une table `table` *représentée par une liste d'enregistrements, un entier* `indice` *associé à un attribut de cette table* `table` *et une valeur* `constante`*. Elle renvoie une nouvelle liste représentant la table* $\sigma_{\textit{Constante}}(\textit{table}, \textit{indice}, \textit{constante})$.

Question I.2. *Donnez la complexité de votre implémentation de la fonction* `SelectionConstante` *par rapport à la taille de la table* `table`*. Justifiez votre réponse en vous appuyant sur la structure du programme.*

Sélection avec test d'égalité entre deux attributs

L'opérateur $\sigma_{\text{Égalité}}$ prend en arguments une table T d'enregistrements et deux attributs de T identifiés à leurs indices respectifs i et j dans $\text{attributs}(T)$. Notez qu'il est possible que $i = j$.

Il renvoie une table T' associée aux mêmes attributs que T. Elle est constituée des enregistrements de T tels que la valeur pour l'attribut d'indice i est égale à la valeur pour l'attribut d'indice j. Cette table peut être vide.

Exemple. $\sigma_{\text{Égalité}}(\text{Trajet}, 1, 2)$ renvoie une table avec les mêmes attributs que la table Trajet. Elle contient tous les voyages dont la ville de départ est la même que la ville d'arrivée. Le résultat est une table vide.

Question I.3. *Implémentez la fonction*

`SelectionEgalite(table, indice1, indice2)`

qui prend en arguments une table `table` *d'enregistrements et deux attributs identifiés à leurs indices* `indice1` *et* `indice2`*. Elle renvoie une nouvelle liste représentant la table* $\sigma_{\text{Égalité}}(\textit{table}, \textit{indice1}, \textit{indice2})$.

Projection sur des indices

L'opérateur Π prend en arguments une table d'enregistrements $T[\![a_0, a_1, \ldots, a_{k-1}]\!]$ d'arité k et un vecteur $L = \langle l_0, \ldots, l_{k'-1} \rangle$ tel que $0 < k' \leq k$ d'indices identifiant des attributs $\langle a_{l_0}, \ldots, a_{l_{k'-1}} \rangle$ de la table T.

On se restreint au cas où la liste L est ordonnée dans le sens croissant, sans répétition. On supposera que les valeurs du vecteur L sont bien comprises entre 0 et $k-1$.

L'opérateur Π renvoie la table T' d'arité k' associée au vecteur d'attributs $\langle a_{l_0}, \ldots, a_{l_{k'-1}} \rangle$. Les enregistrements de T' sont obtenus à partir des enregistrements de T en conservant uniquement les valeurs de ces enregistrements pour les attributs de T'. Deux enregistrements distincts et différents de T peuvent ainsi créer deux enregistrements égaux dans T'.

Exemple. $\Pi(\mathsf{Trajet}, \langle 1, 2 \rangle)$ renvoie une table associée aux attributs $\langle \mathsf{VilleD}, \mathsf{VilleA} \rangle$.

$\langle \mathsf{Lille}, \mathsf{Rennes} \rangle$
$\langle \mathsf{Lille}, \mathsf{Rennes} \rangle$
$\langle \mathsf{Lille}, \mathsf{Rennes} \rangle$

Il se trouve dans ce cas précis que tous ces enregistrements sont égaux. La table n'en est pas moins constituée de 3 enregistrements. Sa taille est 3.

Question I.4. *Considérons une table d'enregistrements $T[\![a_0, a_1, \ldots, a_{k-1}]\!]$ d'arité k. Implémentez la fonction* `ProjectionEnregistrement(enregistrement, listeIndices)` *qui prend en arguments un enregistrement* `enregistrement` *de cette table et une liste* `listeIndices` $\langle l_0, \ldots, l_{k'-1} \rangle$ *telle que $0 < k' \leq k$ d'indices identifiant des attributs de cette table. Elle renvoie une nouvelle liste représentant l'enregistrement $\langle a_{l_0}, \ldots, a_{l_{k'-1}} \rangle$.*

On se restreint au cas où la liste `listeIndices` *d'indices est ordonnée dans le sens croissant, sans répétition. On supposera également que tous les indices l_i de* `listeIndices` *sont compris entre 0 et $k-1$.*

Question I.5. *Implémentez la fonction* `Projection(table, listeIndices)` *qui prend en arguments une table* `table` *d'enregistrements d'arité k et une liste* `listeIndices` $\langle l_0, \ldots, l_{k'-1} \rangle$ *telle que $0 < k' \leq k$ d'indices identifiant des attributs de cette table dans attributs*$(\textit{table})$*. Cette fonction renvoie une nouvelle liste représentant la table* $\Pi(\textit{table}, \textit{listeIndices})$.

Produit cartésien

L'opérateur X prend en arguments deux tables T_1 et T_2 d'enregistrements. La table T_1, d'arité k_1, est constituée de n_1 enregistrements. La table T_2, d'arité k_2, est constituée de n_2 enregistrements. La table T' résultante est d'arité $k_1 + k_2$. Son vecteur d'attributs attributs(T') est la concaténation des vecteurs d'attributs attributs(T_1) et attributs(T_2).

La table T' est constituée de $n_1 \times n_2$ enregistrements. Ces enregistrements sont créés par concaténation de chaque enregistrement de T_1 avec chaque enregistrement de T_2. Les n_1 premiers attributs sont ceux de T_1 dans l'ordre de T_1, les n_2 suivants sont ceux de T_2, dans l'ordre de T_2. L'ordre des enregistrements ainsi synthétisés dans T' est arbitraire.

Exemple. X(Vehicule, Trajet) renvoie une table T'. Les enregistrements de T' sont formés par la concaténation deux à deux des enregistrements de la table Vehicule et de ceux de la table Trajet.

⟨98300, Bus, IBUS, Trajet1, Lille, Rennes, 5 oct. 2016, 09h00, 30990⟩
⟨98300, Bus, IBUS, Trajet2, Lille, Rennes, 5 oct. 2016, 10h00, 98300⟩
⟨98300, Bus, IBUS, Trajet3, Lille, Rennes, 5 oct. 2016, 14h00, 1562⟩
⟨1562, TGV, SNCF, Trajet1, Lille, Rennes, 5 oct. 2016, 09h00, 30990⟩
⟨1562, TGV, SNCF, Trajet2, Lille, Rennes, 5 oct. 2016, 10h00, 98300⟩
⟨1562, TGV, SNCF, Trajet3, Lille, Rennes, 5 oct. 2016, 14h00, 1562⟩
⟨30990, A320, Hop !, Trajet1, Lille, Rennes, 5 oct. 2016, 09h00, 30990⟩
⟨30990, A320, Hop !, Trajet2, Lille, Rennes, 5 oct. 2016, 10h00, 98300⟩
⟨30990, A320, Hop !, Trajet3, Lille, Rennes, 5 oct. 2016, 14h00, 1562⟩

Question I.6. *Implémentez la fonction* `ProduitCartesien(table1, table2)` *qui prend en arguments deux tables* `table1` *et* `table2` *d'enregistrements. Elle renvoie une nouvelle liste représentant la table X(table1, table2).*

Jointure

L'opérateur $\bowtie$ prend en arguments deux tables, T_1 d'arité k_1 et de taille n_1, et T_2 d'arité k_2 et de taille n_2. Il prend aussi en arguments un attribut de T_1 identifié par son indice i_1 tel que $0 \leq i_1 < k_1$ dans le vecteur attributs(T_1) noté A_1 et un attribut de T_2 identifié par son indice i_2 tel que $0 \leq i_2 < k_2$ dans le vecteur attributs(T_2) noté A_2. Posons $A_2 = \langle a_0, \ldots, a_{i_2}, \ldots, a_{k_2-1} \rangle$.

La table T' résultante est d'arité $k_1 + k_2 - 1$. Son vecteur d'attributs attributs(T') est la concaténation du vecteur A_1 et du vecteur A'_2 défini par $\langle a_0, \ldots, a_{i_2-1}, a_{i_2+1}, \ldots, a_{k_2-1} \rangle$, obtenu en effaçant la coordonnée i_2 de A_2.

La table T' est constituée d'au plus $n_1 \times n_2$ enregistrements. Les enregistrements de T' sont créés par concaténation des enregistrements e_1 de T_1 et e_2 de T_2 tels que $e_1[i_1] = e_2[i_2]$, en supprimant la valeur d'indice $k_1 + i_2$ pour éviter la répétition avec celle d'indice i_1. L'enregistrement résultant de cette opération est appelé *jointure* des deux enregistrements e_1 et e_2. Notez qu'il est possible que plusieurs couples (e_1, e_2) produisent des jointures égales dans T'.

Exemple. $\bowtie$(Vehicule, Trajet, 0, 5) renvoie les enregistrements qui décrivent les voyages de chaque véhicule suivi des informations le concernant.

⟨98300, Bus, IBUS, Trajet2, Lille, Rennes, 5 oct. 2016, 10h00⟩
⟨1562, TGV, SNCF, Trajet3, Lille, Rennes, 5 oct. 2016, 14h00⟩
⟨30990, A320, Hop !, Trajet1, Lille, Rennes, 5 oct. 2016, 09h00⟩

Question I.7. *Implémentez la fonction*

```
Jointure(table1, table2, indice1, indice2)
```

qui prend en arguments deux tables `table1` *et* `table2` *et deux entiers représentant respectivement la position d'un attribut de* `table1` *et celle d'un attribut de* `table2`*. Elle renvoie une nouvelle liste représentant la table* $\bowtie$(*table1, table2, indice1, indice2*)*.*

On pourra commencer par implémenter une fonction qui prend en arguments deux enregistrements e_1 *et* e_2 *et deux indices* i_1 *et* i_2 *tels que* $e_1[i_1] = e_2[i_2]$ *et qui renvoie leur jointure au sens ci-dessus.*

Question I.8. *Donnez la complexité de votre implémentation de*

```
Jointure(table1, table2, indice1, indice2)
```

par rapport aux tailles et arités respectives des tables `table1` *et* `table2`*. Justifiez votre réponse en vous appuyant sur la structure du programme.*

Distinct

Nous ajoutons aux opérateurs précédents un nouvel opérateur Distinct qui n'appartient pas à l'algèbre relationnelle classique. Cet opérateur permet de supprimer les répétitions d'enregistrements égaux dans une table T. Il renvoie une table T' qui est associée aux mêmes attributs que T. Cette table contient exactement un représentant pour chaque classe d'enregistrements égaux de T.

Exemple. $\mathsf{Distinct}(\Pi(\mathsf{Trajet}, \langle 1, 2\rangle))$ renvoie une table avec un unique représentant de chaque couple possible de villes de départ et d'arrivée. Cette table ne contient qu'un seul enregistrement : $\langle \mathsf{Lille}, \mathsf{Rennes}\rangle$.

Question I.9. *Implémentez la fonction*

```
SupprimerDoublons(table)
```

qui prend en argument une table `table`*. Elle renvoie une nouvelle liste représentant la table* *Distinct*(*table*)*.*

On rappelle que l'opérateur Python d'égalité entre listes ne doit pas être utilisé dans ce sujet. Il est seulement possible de tester l'égalité de deux valeurs associées à un même attribut.

Question I.10. *Donnez la complexité de votre implémentation de*

```
SupprimerDoublons(table)
```

par rapport à la taille et l'arité de la table `table`*. Justifiez votre réponse en vous appuyant sur la structure du programme.*

II Implémentation de requêtes SQL en Python

Les données de notre agence de voyage sont enregistrées par les tables suivantes.

Vehicule(**IdVehicule**, **Type**, **Compagnie**) **:** enregistre les véhicules disponibles — l'identifiant du véhicule, son type et sa compagnie.

Trajet(**IdTrajet**, **VilleD**, **VilleA**, **IdVehicule**) **:** enregistre les trajets élémentaires possibles — l'identifiant du trajet, la ville de départ, la ville d'arrivée ainsi que le véhicule utilisé.

Ticket(**IdTicket**, **IdTrajet**, **Place**, **Date**, **Heure**, **Prix**) **:** enregistre les tickets disponibles — l'identifiant du ticket, l'identifiant du trajet auquel ce ticket donne accès, le numéro de la place, la date, l'horaire et le prix.

Hotel(**IdHotel**, **Classe**, **Ville**) **:** enregistre les hôtels connus — l'identifiant de l'hôtel, sa classe et sa ville.

Chambre(**IdReservation**, **IdHotel**, **Date**, **Prix**) **:** enregistre les chambres d'hôtel qui sont disponibles — l'identifiant de réservation à utiliser, l'identifiant de l'hôtel où se trouve la chambre, la date et le prix.

L'objectif de cette partie est d'étudier l'implémentation de requêtes SQL en combinant les fonctions de l'algèbre relationnelle présentées dans la partie I. Tout commentaire expliquant et justifiant la traduction sera apprécié.

Par convention, la liste Python représentant une table aura le même nom que cette table. On représentera un attribut avec sa position. Par exemple, l'attribut IdTrajet de la table Trajet est représenté par l'entier 0.

Dans chaque cas, le résultat de la requête sera affecté à une variable nommée `resultat`. Par exemple, la requête SQL

```
SELECT Vehicule.Compagnie FROM Vehicule
```

pourra être implémentée par

```
resultat = Projection(Vehicule,[2])
```

en supposant que la variable Python `Vehicule` représente la table Vehicule. Dans des cas plus complexes, on pourra simplifier l'expression en utilisant des variables auxiliaires pour stocker la valeur de certaines sous-expressions, comme dans l'exemple suivant.

```
r1 = Vehicule
resultat = Projection(r1, [2])
```

Il est attendu que les candidats rédigent leurs réponses en combinant uniquement les fonctions de l'algèbre relationnelle présentées dans la partie I, à l'exclusion de toute autre fonction ou structure de contrôle Python.

Question II.1. *Proposez une implémentation pour la requête suivante.*

```
SELECT *
  FROM Trajet
  WHERE Trajet.VilleD = Rennes
```

Question II.2. *Proposez une implémentation pour la requête suivante.*

```
SELECT *
  FROM Trajet, Vehicule
```

Question II.3. *Proposez une implémentation pour la requête suivante.*

```
SELECT *
  FROM Trajet, Vehicule
  WHERE Trajet.IdVehicule = Vehicule.IdVehicule
```

Question II.4. *Proposez une implémentation pour la requête suivante.*

```
SELECT Classe, Ville, Date, Prix
  FROM Hotel JOIN Chambre
    ON Hotel.IdHotel = Chambre.IdHotel
```

Question II.5. *Proposez une implémentation pour la requête suivante.*

```
SELECT Hotel.IdHotel
  FROM Hotel, Trajet, Ticket
    WHERE Hotel.Ville = Trajet.VilleA
     AND Trajet.IdTrajet = Ticket.IdTrajet
     AND Ticket.Prix = '50'
```

Question II.6. *Proposez une implémentation pour la requête suivante.*

```
SELECT *
  FROM Chambre
  WHERE Chambre.Prix = '100'
    AND Chambre.IdHotel IN
      (SELECT Hotel.IdHotel
         FROM Hotel, Trajet, Ticket
           WHERE Hotel.Ville = Trajet.VilleA
             AND Trajet.IdTrajet = Ticket.IdTrajet
             AND Ticket.Prix = '50')
```

III Amélioration des performances

Il est possible dans certains cas d'améliorer l'implémentation d'une requête en tenant compte de propriétés particulières de la représentation des données ou en utilisant des structures de données supplémentaires. Dans cette partie, nous allons montrer que l'on peut amé-

liorer les performances en triant les données avant de les traiter ou en utilisant des tables associatives (dictionnaires) auxiliaires.

Tables triées par rapport à un indice

Une table d'arité k est représentée par une liste d'enregistrements, eux-mêmes représentés par des listes à k éléments. Supposons tout d'abord avoir à disposition une fonction

```
TrieTableIndice(table, indice)
```

qui trie par ordre croissant suivant l'ordre lexicographique les enregistrements de la liste `table` d'arité k par rapport à la valeur de l'attribut d'indice `indice` dans le vecteur des attributs de cette table. On suppose que la valeur `indice` est strictement inférieure à k.

Par exemple, la liste `Trajet` ci-dessous est triée par rapport à l'attribut d'indice 1 pour l'ordre lexicographique $<$ sur les chaînes de caractères de Python.

```
>>> Trajet
[['30990', 'A320', 'Hop!'], ['98300', 'Bus', 'IBUS'], ['1562', 'TGV', 'SNCF']]
```

Question III.1. *Implémentez la fonction* `VerifieTrie(table, indice)` *qui renvoie* `True` *si la table* `table` *est triée pour l'indice* `indice` *et* `False` *sinon.*

Question III.2. *Considérez la fonction de la question I.1.*

```
SelectionConstante(table, indice, constante)
```

Hypothèse. *On suppose que la liste représentant la table* `table` *est triée selon l'indice* `indice`*.*

Proposez une implémentation de cette fonction qui utilise cette hypothèse pour améliorer les performances. Elle sera nommée comme suit :

```
SelectionConstanteTrie(table, indice, constante)
```

Question III.3. *Considérez la fonction de la question I.7.*

```
Jointure(table1, table2, indice1, indice2)
```

Hypothèse. *On suppose dans cette question que les enregistrements de la table* `table1` *ont des valeurs deux à deux distinctes pour l'attribut d'indice* `indice1`*, et de même pour les enregistrements de la table* `table2` *avec l'indice* `indice2`*.*
On suppose de plus que la liste représentant la table `table1` *est triée selon l'indice* `indice1` *et que celle représentant la table* `table2` *est triée selon l'indice* `indice2`*.*

Proposez une implémentation de cette fonction qui utilise cette hypothèse pour améliorer les performances. Elle sera nommée comme suit :

```
JointureTrie(table1, table2, indice1, indice2)
```

Question III.4. *Donnez la complexité de votre implémentation en vous appuyant sur la structure du programme. Donnez des exemples pour lesquels cette nouvelle approche est plus performante. Y a-t-il des cas où elle n'est pas plus performante ?*

Utilisation d'un dictionnaire (index)

Dictionnaires Python

Un dictionnaire est une structure de données de Python qui permet d'associer à une clé c une valeur v. On parle aussi de *table d'association*. Dans notre cas, les clés sont des chaînes de caractères et les valeurs sont des listes d'entiers.

Nous donnons ici quelques opérations permettant de manipuler les dictionnaires en Python.


Il est attendu que les candidats rédigent leurs réponses en utilisant exclusivement ces opérations pour manipuler les dictionnaires.

Création d'un dictionnaire. L'opération `dico = {}` crée le dictionnaire `dico` et l'initialise à vide. Cette opération a un coût unitaire.

Ajout d'une association. L'opération `dico[c] = liste` ajoute au dictionnaire une association entre la clé `c` et la liste `liste`. Si une association existait déjà pour la clé dans le dictionnaire, celle-ci est perdue. Cette opération a un coût unitaire, indépendamment de l'état du dictionnaire et des valeurs de la clé et de la liste.

Extraction d'une clé. L'opération `dico[c]` renvoie la liste associée à la clé `c` dans le dictionnaire `dico`. Cette opération n'est autorisée que si la clé `c` est effectivement associée à une valeur dans le dictionnaire `dico`. Si aucune association n'existe pour la clé, une erreur se produit. Cette opération a un coût unitaire, indépendamment de l'état du dictionnaire et de la valeur de la clé.

Test de présence. L'opération `c in dico` renvoie `True` si la clé `c` est associée à une valeur dans le dictionnaire `dico` et `False` sinon. Cette opération a un coût unitaire, indépendamment de l'état du dictionnaire et de la valeur de la clé.

On peut itérer sur les clés présentes dans un dictionnaire `dico` par la commande

```
for c in dico: ...
```

Voici un exemple d'utilisation.

```
>>> dico = {}
>>> dico['aaa'] = [1]
>>> dico['bbb'] = [2]
>>> dico['ccc'] = [3]
>>> dico['aaa'].append(4)
>>> dico['ccc'] = [5]
>>> for c in dico: print c, '␣→␣', dico[c]
aaa  →  [1, 4]
bbb  →  [2]
ccc  →  [5]
>>> dico['ddd']
KeyError: 'ddd'
```

Utilisation de dictionnaires pour indexer les bases de données

Une autre idée pour explorer les tables efficacement est d'utiliser des dictionnaires.

Considérons une table T d'arité k et de taille n représentée par une liste Python d'enregistrements $[e_0, \ldots, e_{n-1}]$. Soit L cette liste. Considérons un indice i d'attribut de T tel que $0 \leq i < k$.

On peut associer à T et i un dictionnaire Python $Dico_{T,i}$ de la manière suivante. Les clés de ce dictionnaire sont les valeurs possibles pour les valeurs v qui apparaissent pour l'attribut d'indice i dans la table T. L'image associée à une clé v est la liste des positions dans L des enregistrements e tels que $e[i] = v$.

Si l'attribut d'indice i de T ne prend la valeur v pour aucun enregistrement, cette clé n'est pas enregistrée dans le dictionnaire. L'image d'une clé est donc une liste non vide.

Exemple (Dictionnaire associé à une table)**.** Considérons la table

$$\mathsf{Vehicule}[\![\mathsf{IdVehicule}, \mathsf{Type}, \mathsf{Compagnie}]\!]$$

avec les enregistrements suivants.

$$\langle 98300, \mathsf{Bus}, \mathsf{IBUS}\rangle$$
$$\langle 1562, \mathsf{TGV}, \mathsf{SNCF}\rangle$$
$$\langle 30990, \mathsf{A320}, \mathsf{Hop\,!}\rangle$$
$$\langle 1789, \mathsf{TGV}, \mathsf{SNCF}\rangle$$

Soit `dico` le dictionnaire $Dico_{\mathsf{Vehicule},1}$ associé à l'attribut Type de position 1.

Nous avons

```
>>> for c in dico: print c, '␣→␣', dico[c]
Bus  →  [0]
A320  →  [2]
TGV  →  [1, 3]
>>> dico['Ariane6']
KeyError: 'Ariane6'
```

Application à la sélection

Question III.5. *Implémentez la fonction* `CreerDictionnaire(table, indice)` *qui prend en arguments une table* `table` *et un indice* `indice` *d'attribut de* `table`*. Elle renvoie un dictionnaire de la table* `table` *pour l'attribut d'indice* `indice`*.*

Question III.6. *Considérez la fonction de la question I.1.*

```
SelectionConstante(table, indice, constante)
```

Implémentez la fonction

```
SelectionConstanteDictionnaire(table, indice, constante, dico)
```

qui a la même fonctionnalité que `SelectionConstante`*, mais qui prend en plus en argument un dictionnaire* `dico` *de la table* `table` *pour l'indice* `indice`*.*

Question III.7. *Comparez la complexité de la fonction* `SelectionConstanteDictionnaire` *avec celle de la fonction* `SelectionConstante`*. Donnez des exemples pour lesquels cette nouvelle approche est plus performante. Y a-t-il des cas où elle n'est pas plus performante ?*

Application à la jointure

Question III.8. *Considérez la fonction de la question I.7.*

```
Jointure(table1, table2, indice1, indice2)
```

Implémentez la fonction

```
JointureDictionnaire(table1, table2, indice1, indice2, dico2)
```

qui a la même fonctionnalité que `Jointure`*, mais qui prend en plus en argument un dictionnaire* `dico2` *pour la table* `table2` *par rapport à l'indice* `indice2`*.*

Question III.9. *Donnez la complexité de votre implémentation par rapport aux tailles et arités respectives des tables* `table1` *et* `table2` *ainsi que par rapport à la longueur maximale d'une liste renvoyée par le dictionnaire* `dico2`*, qui sera notée* k_2*. Justifiez votre réponse en vous appuyant sur la structure du programme.*

Question III.10. *L'opérateur de jointure prend en arguments deux tables qui jouent des rôles analogues. Il serait donc possible d'utiliser un dictionnaire pour* `table1` *au lieu d'un dictionnaire pour* `table2`*. Comment pourrait-on choisir la table à indexer pour obtenir les meilleures performances ?*

⋆ ⋆ ⋆