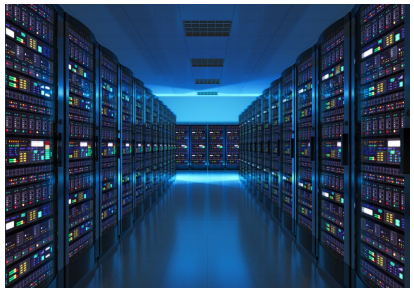
Algèbre relationnelle en PYTHON

Dans ce TP nous proposons une première approche de l'étude des bases de données relationnelles, que nous implémentons (fort naïvement) en PYTHON. 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 et l'implémentation de ces opérateurs en PYTHON. Ce TP est inspiré du sujet d'informatique commune X-ENS 2018.



1 Bases de données, relations, attributs, tuples

Nous détaillons ici la représentation des données dans le modèle relationnel. Une *base de données* est un ensemble de *relations*. Chaque relation porte un nom et est associée à un vecteur de longueur au moins 1 d'*attributs* deux à deux distincts . Le nombre d'attributs d'une relation est appelé l'*arité* de la relation. Le vecteur des attributs $[\![a_0, a_1, \ldots, a_{k-1}]\!]$ d'une relation $\mathcal{R}$ d'arité $k \in \mathbb{N}^*$ est noté $\texttt{attributs}(T)$ et la relation est notée $\mathcal{R}[\![a_0, a_1, \ldots, a_{k-1}]\!]$.

Par exemple, considérons une agence de voyages qui vend des trajets et des chambres d'hôtel. La relation $\texttt{vehicule}[\![\texttt{id\_vehicule}, \texttt{type}, \texttt{compagnie}]\!]$, d'arité 3, permet de mettre en relation les informations concernant les différents moyens de transports disponibles

Une relation $\mathcal{R}[\![a_0, a_1, \ldots, a_{k-1}]\!]$, d'arité $k \in \mathbb{N}^*$, est un *ensemble de tuples* de taille k. La *taille* d'une relation est le nombre de tuples qu'elle contient. Si t est un tuple de la relation avec $t = (t_0, t_1, \ldots, t_{k-1})$, on dit que la valeur t_i à l'indice i du tuple est la valeur associée à l'attribut a_i à l'indice i du vecteur d'attributs de la relation. On pourra donc identifier un attribut et son indice et parler de *la valeur d'un tuple associée à un indice*. La valeur d'un tuple t associée à un indice i est notée $t[i]$.

Par exemple, la relation `vehicule` ci-dessus est constituée de cinq tuples qui décrivent des véhicules : un bus de la compagnie `IBUS` et un de la compagnie `SNCF`, deux trains de la compagnie `SNCF` et un avion de la compagnie `Hop !`.

$$\texttt{vehicule}[\![\texttt{id\_vehicule}, \texttt{type}, \texttt{compagnie}]\!] = \left\{ \begin{array}{l} (\texttt{98300, Bus, IBUS}), \\ (\texttt{1832, Bus, SNCF}), \\ (\texttt{1562, TGV, SNCF}), \\ (\texttt{1781, TGV, SNCF}), \\ (\texttt{30990, A320, Hop !}) \end{array} \right\}$$

Pour chaque tuple t représentant un véhicule dans la relation `vehicule`, la valeur de t associée à l'attribut `id_vehicule` est l'identifiant du véhicule ; la valeur de t 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.

Considérons d'autre part la relation :

$$\texttt{trajet}[\![\texttt{id\_trajet}, \texttt{ville\_d}, \texttt{ville\_a}, \texttt{date\_d}, \texttt{heure\_d}, \texttt{id\_vehicule}]\!] = \left\{ \begin{array}{l} (\texttt{1, Lille, Rennes, 19-12-05, 09h00, 30990}), \\ (\texttt{2, Lille, Rennes, 19-12-05, 10h00, 98300}), \\ (\texttt{3, Lille, Rennes, 19-12-05, 14h00, 1562}), \\ (\texttt{4, Rennes, Lille, 19-12-05, 14h00, 1781}), \\ (\texttt{5, Lille, Rennes, 19-12-06, 14h00, 1562}) \end{array} \right\}$$

Cette relation 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 (au format AA-MM-JJ), l'heure de départ du trajet, l'identifiant du véhicule utilisé pour le trajet. Cette relation contient entre autres trois trajets possibles le *5 décembre 2019* pour aller de Lille à Rennes. Ils partent respectivement à *9h00*, à *10h00* et à *14h00*.

2 Représentation des relation en PYTHON

On représente un tuple d'une relation d'arité $k \in \mathbb{N}^*$ par un tuple en PYTHON de longueur k. Une relation, qui est un ensemble, est représentée en

PYTHON par une liste sans doublons des tuples qui la constitue. Comme les tuples d'une relation ne sont pas ordonnés a priori, plusieurs listes contenant les mêmes éléments dans un ordre différent peuvent représenter la même relation. Une relation vide est représentée par une liste vide.

Par exemple, la relation `vehicule` peut être représentée en PYTHON par la liste :

PYTHON
```
vehicule = [
    (30990, "A320", "Hop !")
    (98300, "Bus", "IBUS"),
    (1832, "Bus", "SCNF"),
    (1562, "TGV", "SNCF"),
    (1781, "TGV", "SNCF"),
]
```

Dans toute la suite on suppose que l'arité des tables est bornée par une constante et on exprimera la complexité en fonction de la taille des relations. La comparaison ou la concaténation de deux tuples d'une relation est considérée comme une opération élémentaire et prend un temps constant.

1. Représenter en PYTHON la relation `trajet`.
2. Écrire une fonction `sans_doublons` qui, étant donnée une liste en argument, renvoie une liste contenant les mêmes éléments que cette liste sans doublons. L'ordre des éléments dans la liste créée n'a pas d'importance. Par exemple `sans_doublons([2, 5, 4, 2, 4])` peut renvoyer la liste `[5, 4, 2]` ou la liste `[2, 5, 4]`.
3. Évaluer la complexité de cette fonction.

3 Opérateurs de l'algèbre relationnelle

Dans toute la suite, on supposera que les arguments des fonctions à rédiger sont bien formés : toutes les tuples d'une liste représentant une relation ont la même longueur qui est l'arité de cette relation, les entiers représentant des indices d'attributs appartiennent bien à l'intervalle attendu, etc. On pourra tirer profit de la syntaxe des listes en compréhension.

3.1 Sélection avec test d'égalité à une constante

L'opérateur $\sigma_{\texttt{Constante}}$ prend en argument une relation $\mathcal{R}$, un attribut de cette relation identifié par son indice i dans le vecteur `attributs(`$\mathcal{R}$`)` et une valeur c. Il renvoie une relation $\mathcal{R}'$ associée aux mêmes attributs que $\mathcal{R}$. Elle est constituée des tuples de $\mathcal{R}$ dont que la valeur de l'attribut d'indice i est égale à la valeur c. Cette relation peut être vide.

Par exemple, $\sigma_{\texttt{Constante}}$`(trajet, 1, Lille)` est la relation $\mathcal{R}'$ des mêmes attributs que la relation `trajet` qui contient les voyages dont la ville de départ est `Lille`. Dans notre exemple, c'est le cas de tous les voyages sauf de celui d'identifiant 5. La relation $\mathcal{R}'$ est donc de taille 4.

4. ✍ Que vaut $\sigma_{\texttt{Constante}}$`(vehicule, 2, SNCF)` ?
5. Écrire en PYTHON une fonction `selection_constante` qui prend en argument une relation `relation`, un indice `indice` associé à un attribut de cette relation et une valeur `constante` qui renvoie une liste correspondant à $\sigma_{\texttt{Constante}}$`(relation, indice, constante)`.
6. Donner la complexité de cette fonction.

3.2 Sélection avec test d'égalité entre deux attributs

L'opérateur $\sigma_{\text{Égalité}}$ prend en arguments une relation $\mathcal{R}$ et deux attributs de $\mathcal{R}$ identifiés à leurs indices respectifs i et j dans `attributs(`T`)` (il est possible d'avoir $i = j$). Il renvoie une relation $\mathcal{R}'$ associée aux mêmes attributs que $\mathcal{R}$ constituée des tuples de $\mathcal{R}$ tels que la valeur pour l'attribut d'indice i est égale à la valeur pour l'attribut d'indice j. Cette relation peut être vide.

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

7. ✍ Que vaut $\sigma_{\text{Égalité}}$`(vehicule, 1, 1)` ?
8. Écrire en PYTHON une fonction `selection_egalite` qui prend en arguments une relation `relation` et deux attributs identifiés à leurs indices `indice1` et `indice2` et qui renvoie une nouvelle liste représentant la relation $\sigma_{\text{Égalité}}$`(relation,indice1,indice2)`.
9. Donner la complexité de cette fonction.

3.3 Projection sur des indices

L'opérateur Π prend en argument une relation $\mathcal{R}[\![a_0, a_1, \ldots, a_{k-1}]\!]$ d'arité $k \in \mathbb{N}^*$ et un k'-uplet $(j_0, \ldots, j_{k'-1})$ d'indices deux à deux distincts dans $[\![0, k-1]\!]$, avec $0 < k' \leq k$, identifiant des attributs de la relation $\mathcal{R}$. L'opérateur Π renvoie une relation $\mathcal{R}'$ d'arité k' associée au vecteur d'attributs $[\![a_{j_0}, a_{j_1}, \ldots, a_{j_{k'-1}}]\!]$ dont les tuples sont obtenus à partir de ceux de $\mathcal{R}$ en ne prenant en compte que les composantes d'indices $(j_0, \ldots, j_{k'-1})$. Attention, deux tuples différents de $\mathcal{R}$ peuvent avoir pour image un seul et même tuple de $\mathcal{R}'$.

Par exemple, la projection $\Pi_{(1,2)}$(`trajet`) renvoie une relation associée aux attributs [\![`ville_d`, `ville_a`]\!] et comporte deux tuples : (`Lille`, `Rennes`) et (`Rennes`, `Lille`).

10. ✍ Que vaut $\Pi_{(1,2)}$(`vehicule`) ?
11. Implémenter une fonction `projection` qui prend en arguments une relation `relation` et un tuple d'indices `indices` identifiant des attributs de cette relation avec les hypothèses ci-dessus et qui renvoie une nouvelle liste représentant $\Pi_{\texttt{indices}}$(`relation`).
12. Donner la complexité de cette fonction.

3.4 Produit cartésien

L'opérateur de produit cartésien $\times$ prend en arguments deux relations $\mathcal{R}_1$ et $\mathcal{R}_2$ d'arités et de tailles respectives $k_1, k_2 \in \mathbb{N}^*$ et $n_1, n_2 \in \mathbb{N}$. La relation $\mathcal{R}' = \mathcal{R}_1 \times \mathcal{R}_2$ est d'arité $k_1 + k_2$ et son vecteur d'attributs est la concaténation des vecteurs d'attributs de $\mathcal{R}_1$ et $\mathcal{R}_2$. Elle est constituée des $n_1 \times n_2$ tuples obtenus par concaténation de chaque tuple de $\mathcal{R}1$ avec chaque tuple de $\mathcal{R}_2$. Les n_1 premiers attributs sont ceux de $\mathcal{R}_1$, dans l'ordre de $\mathcal{R}_1$, les n_2 suivants sont ceux de $\mathcal{R}_2$, dans l'ordre de $\mathcal{R}_2$.

Par exemple, la figure 1 présente quelques tuples parmi les 25 que comporte la relation `vehicule` × `trajet`.

13. ✍ Que vaut `vehicule` × `vehicule` ?
14. Implémenter une fonction `produit_cartesien` qui prend en argument deux relations `relation1` et `relation2` et qui renvoie une nouvelle liste représentant la relation `relation1` × `relation2`.
15. Donner la complexité de cette fonction.

```
(98300, "Bus", "IBUS", 1, Lille, Rennes, 19-12-05, 09h00, 30990)
(98300, "Bus", "IBUS", 2, Lille, Rennes, 19-12-05, 10h00, 98300)
(98300, "Bus", "IBUS", 3, Lille, Rennes, 19-12-05, 14h00, 1562)
(98300, "Bus", "IBUS", 4, Rennes, Lille, 19-12-05, 14h00, 1781)
(98300, "Bus", "IBUS", 5, Lille, Rennes, 19-12-06, 14h00, 1562)
(1832, "Bus", "SCNF", 1, Lille, Rennes, 19-12-05, 09h00, 30990)
(1832, "Bus", "SCNF", 2, Lille, Rennes, 19-12-05, 10h00, 98300)
```

FIGURE 1 – Quelques tuples de la relation `vehicule` × `trajet`.

3.5 Jointure

L'opérateur de jointure $\bowtie$ prend en arguments deux relations $\mathcal{R}_1$ et $\mathcal{R}_2$ d'arités et de tailles respectives $k_1, k_2 \in \mathbb{N}^*$ et $n_1, n_2 \in \mathbb{N}$, un attribut de $\mathcal{R}_1$ identifié par son indice i_1 dans le vecteur `attributs`(T_1), avec $0 \leq i_1 < k_1$, et un attribut de $\mathcal{R}_2$ identifié par son indice i_2 dans le vecteur `attributs`(T_2), avec $0 \leq i_2 < k_2$. La relation $\mathcal{R}' = \mathcal{R}_1 \underset{(i_1,i_2)}{\bowtie} \mathcal{R}_2$ est d'arité $k_1 + k_2$, son vecteur d'attributs est la concaténation des vecteurs d'attributs de $\mathcal{R}_1$ et $\mathcal{R}_2$ et elle est constituée d'au plus $n_1 \times n_2$ tuples. Ces tuples sont créés par concaténation d'un tuple t_1 de $\mathcal{R}_1$ et t_2 de $\mathcal{R}_2$ tels que $t_1[i_1] = t_2[i_2]$.

Par exemple, `vehicule` $\underset{(0,5)}{\bowtie}$ `trajet` renvoie les tuples qui décrivent les voyages de chaque véhicule suivi des informations le concernant :

```
(98300, "Bus", "IBUS", 2, Lille, Rennes, 19-12-05, 10h00, 98300)
(1562, "TGV", "SNCF", 3, Lille, Rennes, 19-12-05, 14h00, 1562)
(1562, "TGV", "SNCF", 5, Lille, Rennes, 19-12-06, 14h00, 1562)
(1781, "TGV", "SNCF", 4, Rennes, Lille, 19-12-05, 14h00, 1781)
(30990, "A320", "Hop !", 1, Lille, Rennes, 19-12-05, 09h00, 30990)
```

Remarquons que le bus 1832 de la `SNCF` n'effectue aucun trajet et que le train 1562 de la `SNCF` en effectue deux.

16. ✍ Que vaut `trajet` $\underset{(2,1)}{\bowtie}$ `trajet` ? À quoi cela correspond-il ?
17. Montrer qu'une jointure n'est en fait rien d'autre qu'un produit cartésien suivi d'une sélection.

18. Implémenter la fonction `jointure` qui prend en argument deux relations `relation1` et `relation2` et deux entiers `indice1` et `indice2` représentant respectivement la position d'un attribut de `relation1` et celle d'un attribut de `relation2` et qui renvoie une nouvelle liste représentant la relation $\texttt{relation1} \underset{(\texttt{indice1},\texttt{indice2})}{\bowtie} \texttt{relation2}$.
19. Donner la complexité de cette fonction. Est-ce plus efficace que de l'implémenter directement par appel aux opérations `produit_cartesien` et `selection_egalite` ?

3.6 Opérations ensemblistes

Les relations étant des ensembles, on peut leur appliquer directement les opérations ensemblistes, pour peu que leurs vecteurs d'attributs soient les mêmes.

20. Écrire une fonction `union` qui prend en argument deux listes sans doublons et qui renvoie une liste sans doublons contenant l'union des ensembles des éléments contenus dans les listes en argument. Quelle est sa complexité ?
21. Même question pour l'intersection ensembliste.
22. Même question pour la différence ensembliste.

3.7 Sélection

De manière plus générale, l'opérateur σ_p prend en argument une relation $\mathcal{R}$ d'arité $k \in \mathbb{N}^*$ et un prédicat p sur les tuples de taille k, c'est-à-dire une fonction qui à un tuple de taille k renvoie un booléen. Il renvoie une relation $\mathcal{R}'$ associée aux mêmes attributs que $\mathcal{R}$, constituée des tuples de $\mathcal{R}$ pour lesquels le prédicat renvoie le booléen « vrai ». Cette relation peut être vide.

Par exemple, la relation $\Pi_{(0,6,1,2)}\left(\sigma_{t\mapsto t[3]<t[9]}(\texttt{trajet} \underset{(2,1)}{\bowtie} \texttt{trajet})\right)$ permet de déterminer quels sont les trajets aller-retour avec au moins une nuit sur place, et est constituée du seul tuple `(4, 5, Rennes, Lille)`.

23. Implémenter cette fonction `selection` qui prend en argument une relation `relation` et un prédicat `predicat`.

4 Amélioration des performances

Il est possible dans certains cas d'améliorer l'implémentation 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 améliorer les performances en triant les données. Dans toute la suite *table* est un synonyme de relation et *enregistrement* est un synonyme de tuple.

4.1 Tables triées par rapport à un indice

Une table d'arité $k \in \mathbb{N}^*$ est donc représentée par une liste d'enregistrements à k éléments. Supposons tout d'abord avoir à disposition une fonction telle qu'un appel `trie_table(table, indice)` trie par ordre croissant suivant l'ordre naturel les enregistrements de la relation `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 `vehicule` définie précédemment est triée par rapport à l'attribut `type` d'indice 1 pour l'ordre lexicographique $\preceq$ sur les chaînes de caractères de Python, car `"A320"` $\preceq$ `"Bus"` $\preceq$ `"TGV"`. En revanche elle n'est pas triée par rapport à l'attribut `id_vehicule` pour l'ordre usuel sur les entiers car $98300 > 1832$.

24. Implémenter une fonction `verifie_trie(table, indice)` qui vérifie si une relation est triée pour un indice donné.
25. Écrire la fonction `trie_table` en implémentant le tri de votre choix.
26. Proposer une implémentation plus efficace de l'opération de sélection constante de la partie 3.1, en supposant que la table est triée selon l'indice donné en argument, ce que l'on ne demande pas de vérifier.
27. Quelle est sa complexité ? Est-ce plus efficace dans tous les cas ? Dans certains cas ? Y a-t-il des cas pour lesquels cela n'est pas plus efficace ?
28. De même, proposer une implémentation plus efficace de la jointure, en supposant les tables triées par rapport aux indices sur lesquels porte la jointure. Même question sur la complexité.