# Amérique du Nord – 2026 – sujet1

## Exercice 3 (8 points)

*Cet exercice porte sur la récursivité, la programmation dynamique et les bases de données.*

Les deux parties de l'exercice sont indépendantes.

### PARTIE A

Dans cette partie, on s'intéresse à la gestion des immeubles par une agence immobilière.

On pourra utiliser les clauses du langage SQL pour :

- construire des requêtes d'interrogation à l'aide de `SELECT`, `FROM`, `WHERE` (avec les opérateurs logiques `AND` et `OR`), `JOIN ... ON` ;
- construire des requêtes d'insertion et de mise à jour à l'aide de `UPDATE`, `INSERT` et `DELETE` ;
- affiner les recherches à l'aide de `DISTINCT` et `ORDER BY`.

On considère une base de données composée des deux tables suivantes :

- `immeuble` :
  - `id_immeuble` est un numéro identifiant l'immeuble ;
  - `nb_etage_immeuble` est le nombre d'étages de l'immeuble ;
  - `numero_immeuble` est le numéro de l'immeuble dans la rue ;
  - `rue_immeuble` est le nom de la rue dans laquelle se trouve l'immeuble.
- `appartement` :
  - `id_appart` est un numéro identifiant l'appartement ;
  - `etage_appart` est l'étage où se trouve l'appartement ;
  - `prix_appart` est le prix de l'appartement ;
  - `id_immeuble` est le numéro de l'identifiant de l'immeuble dans lequel se trouve l'appartement.

Le schéma relationnel de la base de données est donné en figure 1, avec la convention que les attributs formant une clé primaire sont soulignés tandis que ceux d'une clé étrangère sont précédés d'un croisillon (symbole #) avec une flèche vers l'attribut référencé.

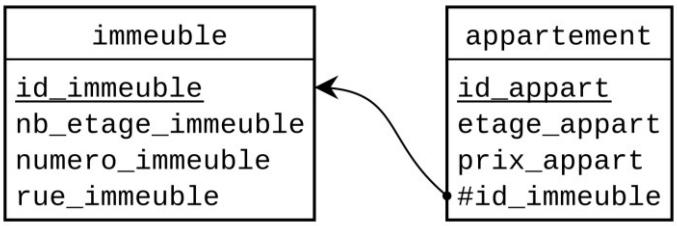


*Figure 1. Schéma de la base de données*

Dans toute la suite, pour simplifier, l'appartement d'identifiant 603 sera désigné simplement par "appartement 603" et l'immeuble d'identifiant 16 sera désigné par "immeuble 16".

1. Expliquer pourquoi le numéro d'un immeuble dans la rue n'a pas été choisi comme clé primaire de la relation `immeuble`.

2. Donner une requête qui renvoie les identifiants des immeubles de la rue `'la mer'` ordonnés dans l'ordre croissant d'identifiants.

3. Donner une requête qui renvoie les identifiants des appartements de l'immeuble 16 et qui se trouvent au moins au 5e étage.

L'immeuble 16 vient d'être détruit. On souhaite effacer les données concernant cet immeuble. On commence par la requête suivante :

```
DELETE FROM immeuble WHERE id_immeuble = 16;
```

4. Expliquer pourquoi cette requête risque de rompre l'intégrité de la base de données.

Suite à la construction d'un immeuble, on souhaite mettre à jour la table `immeuble`.

5. Donner une requête qui ajoute un immeuble de 6 étages, situé au numéro 13 de la rue `'Turing'` en lui conférant l'identifiant 140.

Suite à la démolition d'un immeuble, l'appartement 603 a maintenant la vue sur la mer et son prix a doublé.

6. Donner une requête qui modifie en conséquence le prix de cet appartement.

La fonction d'agrégation `MAX` permet d'obtenir la plus grande valeur d'un attribut.

Par exemple, la requête suivante renvoie le nombre maximal d'étages des immeubles dont l'identifiant est inférieur ou égal à 23 :

```
SELECT MAX(nb_etage_immeuble) FROM immeuble
WHERE id_immeuble <= 23;
```

7. Donner une requête qui renvoie le prix maximal d'un appartement situé dans un immeuble de la rue `'la mer'`.

## PARTIE B

On considère une route perpendiculaire à la mer et des immeubles construits le long de cette route. On dit qu'un immeuble, d'un certain nombre d'étages, bénéficie d'une vue sur la mer lorsque les immeubles situés entre lui et la mer ont moins d'étages que lui. On souhaite que les immeubles de la rue aient tous au moins un appartement avec vue sur mer en détruisant le moins d'immeubles possible.

Pour résoudre ce problème, on considère la liste des nombres d'étages de chaque immeuble de cette rue, en partant de la mer : il s'agit de trouver ce qu'on appelle une *sous-séquence strictement croissante de longueur maximale de cette liste*.

Une *sous-séquence* d'une liste est une suite d'éléments obtenue en supprimant certains éléments, éventuellement aucun, de la liste d'origine sans changer l'ordre des éléments restants.

Exemples : pour la liste `L1 = [10, 22, 9, 33, 21, 50, 41, 60]`, les listes `[22, 9, 50]`, `[10, 22]` ou `[33]` sont des sous-séquences.

La *longueur* d'une sous-séquence est son nombre d'éléments.

On appelle *sous-séquence strictement croissante* d'une liste une sous-séquence telle que chaque élément est **strictement** supérieur au précédent.

Exemples : pour la liste `L1` précédente,

- `[10]` est une sous-séquence strictement croissante de longueur 1 ;
- `[9, 33]` est une sous-séquence strictement croissante de longueur 2 ;
- `[10, 22, 33, 50, 60]` est une sous-séquence strictement croissante de longueur maximale de `L1` ; elle est de longueur 5.

Pour les questions suivantes, on définit la liste `L2 = [3, 1, 8, 2, 5]`.

8. Donner toutes les sous-séquences strictement croissantes de longueur 2 de la liste `L2`.

9. Déterminer la plus longue sous-séquence strictement croissante de la liste `L2`.

10. Écrire une fonction `est_strict_croissante` qui prend en paramètre une liste `seq` et renvoie `True` si la liste est strictement croissante, `False` sinon.

### Récursivité

On cherche à écrire une fonction récursive qui renvoie la longueur d'une plus longue sous-séquence strictement croissante (`llsc`) d'un tableau donné en paramètre. La fonction `llsc_rec` réalise ce calcul à l'aide d'une fonction auxiliaire appelée `llsc_fin`.

**Principe** :

- pour chaque élément du tableau, on considère deux possibilités : l'inclure ou non dans la sous-séquence ;

- l'appel de la fonction auxiliaire `llsc_fin(i)` donne la longueur maximale d'une sous-séquence strictement croissante se terminant à l'indice `i` du tableau.

11. Recopier et compléter les lignes 2, 3 et 6 de la fonction `llsc_fin`.

```
1 def llsc_fin(tab, i):
2     if ...:
3         return ...
4     max_len = 1
5     for j in range(i):
6         if tab[j] < ...:
7             max_len = max(max_len, llsc_fin(tab, j)+1)
8     return max_len
9
10 def llsc_rec(tab):
11     n = len(tab)
12     return max([llsc_fin(tab, i) for i in range(n)])
```

### Programmation dynamique

Afin de résoudre le même problème, cette fois-ci on utilise la programmation dynamique. On cherche à écrire la fonction `llsc_dyn`.

### Principe :

- on crée une liste `dyn` telle que `dyn[i]` contient la longueur d'une plus longue sous-séquence strictement croissante se terminant à l'indice `i` ;
- pour chaque `i`, on parcourt les indices `j < i` et si `tab[j] < tab[i]`, on met à jour `dyn[i]`.

12. Recopier et compléter les lignes 7 et 8 de la fonction `llsc_dyn`. On pourra utiliser la fonction native de Python `max` qui renvoie le maximum des valeurs données en paramètres.

```
1 def llsc_dyn(tab):
2     n = len(tab)
3     dyn = [1] * n
4     for i in range(1, n):
5         for j in range(i):
6             if tab[j] < tab[i]:
7                 dyn[i] = max(..., ...)
8     return ...
```

13. Citer un avantage de cette implémentation par rapport à l'implémentation récursive.