BACCALAURÉAT BLANC
SESSION DE DÉCEMBRE 2023

Épreuve de l'enseignement de spécialité

NUMÉRIQUE et SCIENCES INFORMATIQUES
Partie écrite

Classe terminale de la voie générale

Épreuve 2

DURÉE DE L'ÉPREUVE : 3 heures 30 minutes

Le sujet comporte 14 pages numérotées de 1 à 14, ainsi qu'une annexe, ajoutée à la fin du sujet. Dès que le sujet vous est remis, assurez-vous qu'il est complet.

Il est rappelé que la qualité de la rédaction, la clarté et la précision des raisonnements entreront pour une part importante dans l'appréciation des copies.

L'usage de la calculatrice est interdit.

1 Coloriage (6 points)

`Cet exercice traite de programmation orientée objet en Python et d'algorithmique.`

Un pays est composé de différentes régions. Deux régions sont voisines si elles ont au moins une frontière en commun. L'objectif est d'attribuer une couleur à chaque région sur la carte du pays sans que deux régions voisines aient la même couleur et en utilisant le moins de couleurs possibles.

La figure 1 ci-dessous donne un exemple de résultat de coloration des régions de la France métropolitaine.

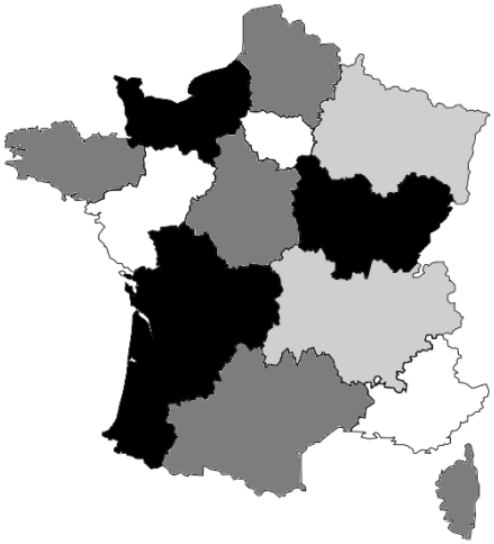

Figure 1 – Carte coloriée des régions de France métropolitaine

On rappelle quelques fonctions et méthodes des tableaux (le type `list` en Python) qui pourront être utilisées dans cet exercice :

- `len(tab)` : renvoie le nombre d'éléments du tableau `tab` ;
- `tab.append(elt)` : ajoute l'élément elt en fin de tableau `tab` ;
- `tab.remove(elt)` : enlève la première occurrence de elt de `tab` si elt est dans `tab`. Provoque une erreur sinon.

Exemple :

- `len([1, 3, 12, 24, 3])` renvoie `5` ;
- avec `tab = [1, 3, 12, 24, 3]`, l'instruction `tab.append(7)` modifie `tab` en `[1, 3, 12, 24, 3, 7]` ;
- avec `tab = [1, 3, 12, 24, 3]`, l'instruction `tab.remove(3)` modifie `tab` en `[1, 12, 24, 3]`.

Les deux parties de cet exercice forment un ensemble. Cependant, il n'est pas nécessaire d'avoir répondu à une question pour aborder la suivante. En particulier, on pourra utiliser les méthodes des questions précédentes même quand elles n'ont pas été codées.

Pour chaque question, toute trace de réflexion sera prise en compte

Partie 1

On considère la classe `Region` qui modélise une région sur une carte et dont le début de l'implémentation est :

```
1  class Region:
2      '''Modélise une région d'un pays sur une carte.'''
3      def __init__(self, nom_region):
4          '''
5          initialise une région
6          : param nom_region (str) le nom de la région
7          '''
8          self.nom = nom_region
9          # tableau des régions voisines, vide au départ
10         self.tab_voisines = []
11         # tableau des couleurs disponibles pour colorier la région
12         self.tab_couleurs_disponibles = ['rouge', 'vert', 'bleu', 'jaune
               ', 'orange', 'marron']
13         # couleur attribuée à la région et non encore choisie au départ
14         self.couleur_attribuee = None
```

1. Associer, en vous appuyant sur l'extrait de code précédent, les noms nom, `tab_voisines`, `tab_couleurs_disponibles` et `couleur_attribuee` au terme qui leur correspond parmi : objet, attribut, méthode ou classe.

2. Indiquer le type du paramètre `nom_region` de la méthode `__init__` de la classe `Region`.

3. Donner une instruction permettant de créer une instance nommée ge de la classe `Region` correspondant à la région dont le nom est "Grand Est".

4. Recopier et compléter la ligne 6 de la méthode de la classe `Region` ci-dessous :

```
1  def renvoie_premiere_couleur_disponible(self):
2      '''
3      Renvoie la première couleur du tableau des couleurs disponibles
           supposé non vide.
4      : return (str)
5      '''
6      return ...
```

5. Recopier et compléter la ligne 6 de la méthode de la classe `Region` ci-dessous :

```
1  def renvoie_nb_voisines(self) :
2      '''
3      Renvoie le nombre de régions voisines.
```

```
4       : return (int)
5       '''
6       return ...
```

6. Compléter la méthode de la classe `Region` ci-dessous à partir de la ligne 6 :

```
1  def est_coloriee(self):
2      '''
3      Renvoie True si une couleur a été attribuée à cette région et
           False sinon.
4      : return (bool)
5      '''
6      ...
```

7. Compléter la méthode de la classe `Region` ci-dessous à partir de la ligne 8 :

```
1  def retire_couleur(self, couleur):
2      '''
3      Retire couleur du tableau de couleurs disponibles de la région
           si elle est dans ce tableau. Ne fait rien sinon.
4      : param couleur (str)
5      : ne renvoie rien
6      : effet de bord sur le tableau des couleurs disponibles
7      '''
8      ...
```

8. Compléter la méthode de la classe `Region` ci-dessous, à partir de la ligne 7, en utilisant une boucle :

```
1  def est_voisine(self, region):
2      '''
3      Renvoie True si la region passée en paramètre est une voisine
           et False sinon.
4      : param region (Region)
5      : return (bool)
6      '''
7      ...
```

Partie 2

Dans cette partie :

- on considère qu'on dispose d'un ensemble d'instances de la classe `Region` pour lesquelles l'attribut `tab_voisines` a été renseigné ;
- on pourra utiliser les méthodes de la classe `Region` évoquées dans les questions de la partie 1 :
  - `renvoie_premiere_couleur_disponible`
  - `renvoie_nb_voisines`

- `est_coloriee`
- `retire_couleur`
- `est_voisine`

On a créé une classe `Pays` :

- cette classe modélise la carte d'un pays composé de régions ;
- l'unique attribut `tab_regions` de cette classe est un tableau (type `list` en Python) dont les éléments sont des instances de la classe Region.

9. Recopier et compléter la méthode de la classe `Pays` ci-dessous à partir de la ligne 7 :

```
1 def renvoie_tab_regions_non_coloriees(self):
2     '''
3     Renvoie un tableau dont les éléments sont les régions du pays
          sans couleur attribuée.
4     : return (list) tableau d'instances de la classe
5     Region
6     '''
7     ...
```

10. On considère la méthode de la classe `Pays` ci-dessous.

```
1 def renvoie_max(self):
2     nb_voisines_max = -1
3     region_max = None
4     for reg in self.renvoie_tab_regions_non_coloriees():
5         if reg.renvoie_nb_voisines() > nb_voisines_max:
6             nb_voisines_max = reg.renvoie_nb_voisines()
7             region_max = reg
8     return region_max
```

a. Expliquer dans quel cas cette méthode renvoie `None`.

b. Indiquer, dans le cas où cette méthode ne renvoie pas `None`, les deux particularités de la région renvoyée.

11. Coder la méthode `colorie(self)` de la classe Pays qui choisit une couleur pour chaque région du pays de la façon suivante :

- On récupère la région non coloriée qui possède le plus de voisines.
- Tant que cette région existe :
  - La couleur attribuée à cette région est la première couleur disponible dans son tableau de couleurs disponibles.
  - Pour chaque région voisine de la région :
    - si la couleur choisie est présente dans le tableau des couleurs disponibles de la région voisine alors on la retire de ce tableau.
  - On récupère à nouveau la région non coloriée qui possède le plus de voisines.

2 Evaluation (6 points)

`Cet exercice porte sur les listes, les dictionnaires et la récursivité`

Pour son évaluation de fin d'année, l'école de la nouvelle programmation (ENP) a opté pour le principe du QCM. Chaque évaluation se présente sous la forme d'une liste de questions numérotées de 0 à 19, et pour chaque question, cinq réponses sont proposées, numérotées de 1 à 5. Chaque candidat doit cocher exactement une réponse par question, et une seule réponse est correcte par question. La correction de chaque évaluation est représentée par une liste appelée "corr" qui contient, pour chaque question, la bonne réponse. Par exemple, la correction de l'épreuve 0, notée `corr0`, est donnée par la liste :

```
corr0 = [4, 2, 1, 4, 3, 5, 3, 3, 2, 1, 1, 3, 3, 5, 4, 4, 5, 1, 3, 3]
```

ce qui signifie que la bonne réponse à la question 0 est 4, et à la question 19 est 3.

Avant d'attribuer une note, il est nécessaire de corriger les copies question par question. Cela implique d'associer à chaque copie une liste de booléens de longueur 20, indiquant, pour chaque question, si la réponse donnée est correcte. Prenons l'exemple du candidat Tom Matt qui a rendu la copie suivante pour l'épreuve 0, notée `copTM` :

```
copTM = [4, 1, 5, 4, 3, 3, 1, 4, 5, 3, 5, 1, 5, 5, 5, 1, 3, 3, 3, 3]
```

La liste de booléens correspondante, notée `corrTM`, est alors :

```
corrTM = [True, False, False, True, True, False, False, False, False, False,
False, False, False, True, False, False, False, False, True, True]
```

1. Écrire en Python une fonction `corrige` qui prend en paramètre `cop` et `corr`, deux listes d'entiers entre 1 et 5 et qui renvoie la liste des booléens associée à la copie `cop` selon la correction `corr`.

   Par exemple, `corrige(copTM, corr0)` renvoie `corrTM`.

La note accordée à une copie correspond tout simplement au nombre de bonnes réponses. Si l'on a accès à la liste de booléens associée à une copie, basée sur la correction, il suffit de compter le nombre de `True` dans la liste. Ainsi, Tom Matt obtient un score de 6/20 à l'épreuve 0. Il est à noter que la création de cette liste de booléens n'est pas indispensable pour calculer la note d'une copie.

2. Écrire en Python une fonction `note` qui prend en paramètre `cop` et `corr`, deux listes d'entiers entre 1 et 5 et qui renvoie la note attribuée à la copie `cop` selon la correction `corr`, sans construire de liste auxiliaire.

   Par exemple, `note(copTM, corr0)` renvoie `6`.

L'ENP souhaite automatiser totalement la correction de ses copies. Pour cela, elle a besoin d'une fonction pour corriger des paquets de plusieurs copies. Un paquet de copies est donné sous la forme d'un dictionnaire dont les clés sont les noms des candidats et les valeurs sont les listes représentant les copies de ces candidats. On peut considérer un paquet `p1` de copies où l'on retrouve la copie de Tom Matt :

```
p1 = {('Tom', 'Matt'): [4, 1, 5, 4, 3, 3, 1, 4, 5, 3, 5, 1, 5, 5, 5, 1, 3, 3,
3, 3], ('Lambert', 'Ginne'): [2, 4, 2, 2, 1, 2, 4, 2, 2, 5, 1, 2, 5, 5, 3, 1, 1,
1, 4, 4], ('Carl', 'Roth'): [5, 4, 4, 2, 1, 4, 5, 1, 5, 2, 2, 3, 2, 3, 3, 5, 2,
2, 3, 4], ('Kurt', 'Jett'): [2, 5, 5, 3, 4, 1, 5, 3, 2, 3, 1, 3, 4, 1, 3, 1, 3,
2, 4, 4], ('Ayet', 'Finzerb'): [4, 3, 5, 3, 2, 1, 2, 1, 2, 4, 5, 5, 1, 4, 1, 5,
4, 2, 3, 4]}
```

3. Écrire en Python une fonction `notes_paquet` qui prend en paramètre un paquet de copies `p` et une correction `corr` et qui renvoie un dictionnaire dont les clés sont les noms des candidats du paquet `p` et les valeurs sont leurs notes selon la correction `corr`.

Par exemple, `notes_paquet(p1, corr0)` renvoie `{('Tom', 'Matt'): 6, ('Lambert', 'Ginne'): 4, ('Carl', 'Roth'): 2, ('Kurt', 'Jett'): 4, ('Ayet', 'Finzerb'): 3}`.

La fonction `notes_paquet` peut faire appel à la fonction `note` demandée en question 2, même si cette fonction n'a pas été écrite.

Pour éviter les problèmes d'identification des candidats qui porteraient les mêmes noms et prénoms, un employé de l'EPN propose de prendre en compte les prénoms secondaires des candidats dans les clés des dictionnaires manipulés.

4. Expliquer si on peut utiliser des listes de noms plutôt qu'un couple comme clés du dictionnaire.

5. Proposer une autre solution pour éviter les problèmes d'identification des candidats portant les mêmes prénoms et noms. Cette proposition devra prendre en compte la sensibilité des données et être argumentée succinctement.

Un ingénieur de l'EPN a démissionné en laissant une fonction Python énigmatique sur son poste. Le directeur est convaincu qu'elle sera très utile, mais encore faut-il comprendre à quoi elle sert.

Voici la fonction en question :

```
1  def enigme(notes) :
2      a = None
3      b = None
4      c = None
5      d = {}
6      for nom in notes :
7          tmp = c
8          if a == None or notes[nom] > a[1] :
9              c = b
10             b = a
11             a = (nom, notes[nom])
12         elif b == None or notes[nom] > b[1] :
13             c = b
14             b = (nom, notes[nom])
15         elif c == None or notes[nom] > c[1] :
```

```
16          c = (nom, notes[nom])
17        else :
18          d[nom] = notes[nom]
19        if tmp != c and tmp != None :
20          d[tmp[0]] = tmp[1]
21      return (a, b, c, d)
```

6. Calculer ce que renvoie la fonction enigme pour le dictionnaire tt {('Tom', 'Matt'): 6, ('Lambert', 'Ginne'): 4, ('Carl', 'Roth'): 2 ('Kurt', 'Jett'): 4, ('Ayet', 'Finzerb'): 3}.

7. En déduire ce que calcule la fonction enigme lorsqu'on l'applique à un dictionnaire dont les clés sont les noms des candidats et les valeurs sont leurs notes.

8. Expliquer ce que la fonction enigme renvoie s'il y a strictement moins de 3 entrées dans le dictionnaire passées en paramètre.

9. Écrire en Python une fonction classement prenant en paramètre un dictionnaire dont les clés sont les noms des candidats et les valeurs sont leurs notes et qui, en utilisant la fonction enigme, renvoie la liste des couples `((prénom, nom), note)` des candidats classés par notes décroissantes. Par exemple, `classement({('Tom', 'Matt'): 6, ('Lambert', 'Ginne'): 4, ('Carl', 'Roth'): 2, ('Kurt', 'Jett'): 4, ('Ayet', 'Finzerb'): 3})` renvoie `[(('Tom', 'Matt'), 6), (('Lambert', 'Ginne'), 4), (('Kurt', 'Jett'), 4), (('Ayet', 'Finzerb'), 3), (('Carl', 'Roth'), 2)]`.

Le professeur Paul Tager a élaboré une évaluation particulièrement innovante de son côté. Toutes les questions dépendent des précédentes. Il est donc assuré que dès qu'un candidat s'est trompé à une question, alors toutes les réponses suivantes sont également fausses. M. Tager a malheureusement égaré ses notes, mais il a gardé les listes de booléens associées. Grâce à la forme particulière de son évaluation, on sait que ces listes sont de la forme `[True, True, ..., True, False, False, ..., False]`.

Pour recalculer ses notes, il a écrit les deux fonctions Python suivantes (dont la seconde est incomplète) :

```
1  def renote_express(copcorr) :
2    c = 0
3    while copcorr[c] :
4      c = c + 1
5    return c
6  def renote_express2(copcorr) :
7    gauche = 0
8    droite = len(copcorr)
9    while droite - gauche > 1 :
10     milieu = (gauche + droite)//2
11     if copcorr[milieu] :
12       ...
13     else :
14       ...
15   if copcorr[gauche] :
16     return ...
17   else :
18     return ...
```

10. Compléter le code de la fonction Python `renote_express2` pour qu'elle calcule la même chose que `renote_express`.

11. Déterminer les coûts en temps de renote_express et `renote_express2` en fonction de la longueur n de la liste de booléens passée en paramètre.

12. Expliquer comment adapter `renote_express2` pour obtenir une fonction qui corrige très rapidement une copie pour les futures évaluations de M. Tager s'il garde la même spécificité pour ses énoncés. Cette fonction ne devra pas construire la liste de booléens correspondant à la copie corrigée, mais directement calculer la note.

3 Livres (8 points)

`Cet exercice porte sur la programmation Python (dictionnaire), la programmation orientée objet, les bases de données relationnelles et les requêtes SQL`

Cet exercice se divise en trois parties distinctes. L'objectif est de développer une application dédiée au stockage et au traitement d'informations relatives à des livres de science-fiction. Les données que l'on souhaite enregistrer comprennent les éléments suivants :

- l'identifiant du livre (id) ;
- le titre (titre) ;
- le nom de l'auteur (nom_auteur) ;
- l'année de première publication (ann_pub) ;
- une note sur 10 (note).

Un extrait des informations à consigner est présenté ci-dessous :

figure 1
`livres`

id	titre	auteur	ann pub	note
1	1984	Orwell	1949	10
2	Dune	Herbert	1965	8
14	Fondation	Asimov	1951	9
4	Ubik	K. Dick	1953	9
8	Blade Runner	K. Dick	1968	8
7	Les Robots	Asimov	1950	10
15	Ravage	Barjavel	1943	6
17	Chroniques martiennes	Bradbury	1950	7
9	Dragon déchu	Hamilton	2003	8
10	Fahrenheit 451	Bradbury	1953	8

Partie A

Dans cette première partie, on utilise un dictionnaire Python. On considère le programme suivant :

```
1 dico_livres = {
2     'id' : [1, 2, 14, 4, 5, 8, 7, 15, 9, 10],
3     'titre' : ['1984', 'Dune', 'Fondation', 'Ubik', 'Blade␣Runner', 'Les
        ␣Robots', 'Ravage', 'Chroniques␣martiennes', 'Dragon␣déchu', '
        Fahrenheit␣451'],
4     'auteur' : ['Orwell', 'Herbert', 'Asimov', 'K. Dick', 'K. Dick', '
        Asimov', 'Barjavel', 'Bradbury', 'Hamilton', 'Bradbury'],
5     'ann_pub' : [1949, 1965, 1951, 1953,1968, 1950, 1943, 1950, 2003,
        1953],
6     'note' : [10, 8, 9, 9, 8, 10, 6, 7, 8, 8]
7     }
8 a = dico_livres['ann_pub']
9 b = dico_livres['titre'][4]
```

1. Déterminer les valeurs des variables a et b après l'exécution de ce programme.

La fonction `titre_livre` prend en paramètre un dictionnaire (de même structure que `dico_livres`) et un identifiant, et renvoie le titre du livre qui correspond à cet identifiant. Dans le cas où l'identifiant passé en paramètre n'est pas présent dans le dictionnaire, la fonction renvoie `None`.

```
1 def titre_livre(dico, id_livre):
2   for i in range(len(dico['id'])):
3     if dico['id'][i] == ... :
4       return dico['titre'][...]
5   return ...
```

2. Recopier et compléter les lignes 3, 4 et 5 de la fonction `titre_livre`.

3. Écrire une fonction `note_maxi` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et qui renvoie la note maximale.

4. Écrire une fonction `livres_note` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et une note `n`, et qui renvoie la liste des titres des livres ayant obtenu la note `n` (on rappelle que `t.append(a)` permet de rajouter l'élément `a` à la fin de la liste `t`).

5. Écrire une fonction `livre_note_maxi` qui prend en paramètre un dictionnaire `dico` (de même structure que `dico_livres`) et qui renvoie la liste des titres des livres ayant obtenu la meilleure note sous la forme d'une liste Python.

Partie B

Dans cette partie, on utilise le paradigme orientée objet (POO). On propose deux classes : `Livre` et `Bibliotheque`.

```
class Livre:
  def __init__(self, id_livre, titre, auteur, ann_pub, note):
    self.id = id_livre
    self.titre = titre
    self.auteur = auteur
    self.ann_pub = ann_pub
    self.note = note
  def get_id(self):
    return self.id
  def get_titre(self):
    return self.titre
  def get_auteur(self):
    return self.auteur
  def get_ann_pub(self):
    return self.ann_pub

class Bibliotheque:
  def __init__(self):
    self.liste_livre = []
  def ajout_livre(self, livre):
    self.liste_livre.append(livre)
  def titre_livre(self, id_livre):
    for livre in self.liste_livre :
      if ... == id_livre :
        return ...
    return ...
```

6. Citer un attribut et une méthode de la classe `Bibliotheque`.

7. Écrire la méthode `get_note` de la classe `Livre`. Cette méthode devra renvoyer la note d'un livre.

8. Écrire le programme permettant d'ajouter le livre Blade Runner à la fin de la "bibliothèque" en utilisant la classe `Livre` et la classe `Bibliotheque` (voir le tableau en début d'exercice).

9. Recopier et compléter la méthode `titre_livre` de la classe `Bibliotheque`. Cette méthode prend en paramètre l'identifiant d'un livre et renvoie le titre du livre si l'identifiant existe, ou `None` si l'identifiant n'existe pas.

Partie C

On utilise maintenant une base de données relationnelle. Les commandes nécessaires ont été exécutées afin de créer une table livres qui est exactement la table de la figure 1.

L'attribut `id` est la clé primaire pour la table `livres`.

10. Expliquer pourquoi l'attribut auteur ne peut pas être choisi comme clé primaire.

11. Donner le résultat renvoyé par la requête SQL suivante :

```
1 SELECT titre
2 FROM livres
3 WHERE auteur = 'Asimov';
```

12. Écrire une requête SQL permettant d'obtenir les titres des livres écrits par Bradbury publiés avant 1951.

13. Écrire une requête SQL permettant de modifier la note du livre Dune en la passant de 8/10 à 9/10.

On souhaite proposer plus d'informations sur les auteurs des livres. Pour cela, on crée une deuxième table `auteurs` avec les attributs suivants :

- `id` de type `INT` ;
- `nom` de type `TEXT` ;
- `prenom` de type `TEXT` ;
- `annee_naissance` de type `INT` (année de naissance).

`auteurs`

ID	Nom	Prénom
1	Orwell	George
2	Herbert	Franck
3	Asimov	Isaac
4	K. Dick	Philip
5	Bradbury	Ray
6	Barjavel	René
7	Hamilton	Peter

La table `livres` est aussi modifiée comme suit :

`livres`

id	titre	id_auteur	ann_pub	note
1	1984	1	1949	10
2	Dune	2	1965	8
14	Fondation	3	1951	9
4	Ubik	4	1953	9
8	Blade Runner	4	1968	8
7	Les Robots	3	1950	10
15	Ravage	6	1943	6
17	Chroniques martiennes	5	1950	7
9	Dragon déchu	7	2003	8
10	Fahrenheit 451	5	1953	8

14. Expliquer l'intérêt d'utiliser deux tables (livres et auteurs) au lieu de regrouper toutes les informations dans une seule table.

15. Expliquer le rôle de l'attribut id_auteur de la table livres.

16. Écrire une requête SQL qui renvoie le nom et le prénom des auteurs des livres publiés avant 1950.

17. Décrire par une phrase en français le résultat de la requête SQL suivante :

```
SELECT titre
FROM livres
JOIN auteurs ON id_auteur = auteurs.id
WHERE ann_pub - annee_naissance < 40;
```

Un élève décide de créer une application d'annuaire pour sa classe. On pourra retrouver, grâce à cette application, différentes informations sur les élèves de la classe : nom, prénom, date de naissance, numéro de téléphone, adresse email, etc.

18. Expliquer en quoi la réalisation de ce projet pourrait être problématique.

4 ANNEXE

Types de données	
CHAR(t)	Texte fixe de t caractères.
VARCHAR(t)	Texte de t caractères variables.
TEXT	Texte de 65 535 caractères max.
INT	Nombre entier
FLOAT	Réel à virgule flottante
DATE	Date format AAAA-MM-JJ
DATETIME	Date et heure format AAAA-MM-JJ HH:MI

Quelques exemples de syntaxe SQL :

- Insérer des enregistrements :

```
INSERT INTO Table (attribut1, attribut2) VALUES(valeur1 , valeur2)
```

Exemple :

```
INSERT INTO client(id_client,nom,prenom) VALUES (112,'Dehnou','Danièle')
```

- Modifier des enregistrements :

```
UPDATE Table SET attribut1=valeur1, attribut2=valeur2 WHERE Selecteur
```

- Supprimer des enregistrements :

```
DELETE FROM Table WHERE Selecteur
```

- Sélectionner des enregistrements :

```
SELECT attributs FROM Table WHERE Selecteur
```

- Effectuer une jointure :

```
SELECT attributs FROM TableA JOIN TableB ON TableA.cle1=TableB.cle2 WHERE
Selecteur
```