Asie – 2025 – sujet2 - Correction

Exercice 3 (8 points)

PARTIE A — Base de données (SQL)

1) Patients de plus de 60 ans

```
SELECT nom_patient, prenom
FROM Patient
WHERE age > 60;
```

👉*Commentaire : On utilise* `WHERE age > 60` *car l'énoncé précise* strictement plus de 60 ans.

---

2) Mise à jour : Alice Heartman ne tousse plus

```
UPDATE Symptome
SET toux = 'Non'
WHERE nom_patient = 'Heartman';
```

👉*Commentaire :*

- *On modifie la table* `Symptome`
- `SET` *modifie la valeur*
- `WHERE` *évite de modifier tous les patients*

---

3) Nombre de patients Covid-19 qui toussent

```
SELECT COUNT(*)
FROM Diagnostic
JOIN Symptome
ON Diagnostic.nom_patient = Symptome.nom_patient
WHERE nom_maladie = 'Covid-19'
AND toux = 'Oui';
```

👉 Commentaire :

- `JOIN` relie Diagnostic et Symptome

- `COUNT(*)` compte les lignes
- Condition double avec `AND`

---

4) Pourquoi la requête INSERT produit une erreur?

```
INSERT INTO Patients VALUES ('Douglas', 'Patrick', 168077230253829, 55)
```

Erreur car :

- La table s'appelle **Patient** (et non Patients)
- `nom_patient` est clé primaire
- Douglas existe déjà

👉 *Il y a violation de la contrainte d'unicité.*

---

5) Modification du schéma relationnel

Problème : plusieurs patients peuvent avoir le même nom.

Solution : ajouter un identifiant unique : `id_patient INT PRIMARY KEY`

Et faire de `nom_patient` un simple attribut.

👉*Bonne pratique : ne jamais utiliser un nom comme clé primaire.*

PARTIE B — Arbre de décision

6) Diagnostic du patient

Patient :

- toux = True
- fievre = True
- nausee = False
- anosmie = False

Selon l'arbre décrit : → Diagnostic : **Covid-19 positif**

(raisonnement : toux → droite, fièvre → droite)

7) Assertion de la méthode symptome

L'assertion vérifie que le nœud n'est pas une feuille.

👉*Elle garantit qu'on appelle la méthode uniquement sur un nœud interne.*

---

8) Un attribut et une méthode

Attribut : `etiquette`

Méthode : `est_feuille()`

---

9) Fonction applique (récursive)

```
def applique(arbre, patient):
    # Si feuille → renvoyer diagnostic
    if arbre.est_feuille():
        return arbre.etiquette

    # Sinon on teste le symptôme
    if patient[arbre.etiquette]:
        return applique(arbre.droit, patient)
    else:
        return applique(arbre.gauche, patient)
```

👉*Commentaire :*

- *Cas de base : feuille*
- *Appel récursif sur sous-arbre*
- *On suit la règle donnée dans l'énoncé*

---

10) Taille de l'arbre

On compte tous les nœuds (internes + feuilles).

👉 Taille = **7**

---

11) Réduction de l'arbre

On applique la règle :

Si un nœud a deux feuilles identiques → on le remplace par une feuille.

👉 L'arbre est simplifié en supprimant le test inutile.

(Nouvel arbre : un nœud supprimé au niveau le plus bas)

---

12) Méthode reduire

def reduire(arbre) :

```
    if arbre.est_feuille():
        return arbre

    arbre.gauche = reduire(arbre.gauche)
    arbre.droit = reduire(arbre.droit)

    # Si les deux sous-arbres sont feuilles
    if (arbre.gauche.est_feuille() and
        arbre.droit.est_feuille() and
        arbre.gauche.etiquette == arbre.droit.etiquette):

        return Noeud(arbre.gauche.etiquette)

    return arbre
```

👉*Commentaire :*

- *Traitement récursif postfixe*
- *Simplification locale*
- *Cas typique d'algorithme sur arbre*

PARTIE C — Intégrité des données

13) Fonction verifie

```
def verifie(num_secu):
    n = num_secu // 100
    k = num_secu % 100
    return (n + k) % 97 == 0
```

👉 Commentaire :

On teste si `(n + k)` est multiple de 97.

---

14) Fonction cle

On veut : `n + k ≡ 0 mod 97`

Donc :

```
k = 97 - (n % 97)
def cle(n):
    reste = n % 97
    if reste == 0:
        return 0
    return 97 - reste
```

👉*Commentaire : Si `n % 97 == 0`, la clé vaut 0.*
