Université Sétif 0I
Faculté des sciences, Département Informatique
1ère année Master

Examen Base de données Avancée

(Corrigé Type + Brème détaillé)

Exercice 01 : *(6.5 Pts)*

Soit la base de données suivante qui représente des services d'un hôpital ainsi que des patients (Malades) qui occupent des lits dans ces services :

Service (**Code-S**, Nom_S, Nombre_Lits).

Patient (**Matricule**, Nom_P, Prenom, Date_Nais, Sexe, **# code-S**).

Questions :

Instructions :

(Moins) -0.25 points Pour chaque instruction manquante.
00 Pour chaque instruction erronée.
00 En cas d'utilisation d'une variable non déclarée.

1. Ecrire un **trigger** qui permet de contrôler que le nombre de patients dans un service ne doit pas dépasser le nombre de lits dans ce service.

```
create or replace trigger T BEFORE insert or UPDATE on patient        } 0.5
for each row
DECLARE                                                               }
N_Max NUMBER;                                                         } 0.25
Lits NUMBER;                                                          }
BEGIN
select count(*) into N_Max from patient where code_S = :new.code_S;   0.5
select Nombre_Lits into Lits from service where code_S= :new.code_S;  0.5
IF n_max >= Lits THEN     0.5
   RAISE_APPLICATION_ERROR(-20000, 'nombre de lits dépassés');        0.5
END IF;       0.25
END;
```

2. Ecrire une **fonction** qui permet de trouver le nombre de lits vides (Non occupés par des patients) dans un service donné.

```
CREATE OR REPLACE FUNCTION Lvide (num_serv IN service.code_S%type)     0.5
RETURN number                                                          }
IS                                                                     } 0.5
occupe number;                                                         }
Total number;                                                          } 0.5
Lits_vides number;                                                     }
BEGIN
SELECT count(*) INTO occupe from patient where code_S = num_serv;      0.75
SELECT Nombre_Lits INTO Total from service where code_s = num_serv;    0.75
Lits_vides:=Total-occupe;   0.25
RETURN Lits_vides;    0.25
END;
```

Exercice 02: *(07Pts)*

On considère le schéma relationnel suivant qui représente une table (non vide) qui contient le stock des médicaments et une autre table MED_PERIME (Non vide) qui contient le stock des médicaments périmés.

MED (**Code_M**, DCI, Prix, date_Peremption, Qte_Stock).

MED_PERIME (**Code_M**, Qte_Perime).

Questions :

À l'aide de SQL*Plus, créer un curseur qui permet de parcourir la table MED afin de retirer (**supprimer**) les médicaments périmés et de les **ajouter** à la table MED_PERIME.

Instructions :

(Moins) - 0.5 Pour chaque "end " manquant (End If ou end loop).

00 Si instruction erronée ou en dehors d'une boucle ou d'un bloc IF.

00 En cas d'utilisation d'une variable non déclarée.

On accepte SYSDATE ou '15/01/2023' comme date d'oujourd'hui.

```
DECLARE
CURSOR retrait is (select Code_M, date_Peremption, Qte_Stock FROM MED);   } 0.5
existe number;    0.25
BEGIN
FOR c IN retrait      0.5
LOOP                  0.5
    IF C.date_Peremption <=SYSDATE THEN    0.5
      SELECT count(*) into existe from med_perime where Code_M=C.Code_M;    0.75
      IF existe >0 THEN
        UPDATE med_perime SET qte_perime=qte_perime+C.Qte_Stock;
        DELETE FROM MED WHERE Code_M = C.Code_M;                    } 02
        Commit;                                                     (0.5 pour chaque instruction)
      ELSE
        INSERT INTO med_perime VALUES (C.Code_M,C.Qte_Stock);
        DELETE FROM MED WHERE Code_M = C.Code_M;                    } 02
        Commit;                                                     (0.5 pour chaque instruction)
      END IF;
    END IF;
END LOOP;
END;
```

Pour les étudiants qui ont utilisés la boucle WHILE

```
DECLARE
CURSOR retrait is (select Code_M, date_Peremption, Qte_Stock FROM MED);        } 0.5
existe number;    0.25
BEGIN
open retrait;          0.25
LOOP
    fetch retrait into c;
    EXIT WHEN retrait%notfound;        } 0.5
    IF C.date_Peremption <=SYSDATE THEN      0.5
      SELECT count(*) into existe from med_perime where Code_M=C.Code_M;    0.75
      IF existe >0 THEN
        UPDATE med_perime SET qte_perime=qte_perime+C.Qte_Stock;
        DELETE FROM MED WHERE Code_M = C.Code_M;                    } 02
        Commit;                                                     (0.5 pour chaque instruction)
      ELSE
        INSERT INTO med_perime VALUES (C.Code_M,C.Qte_Stock);
        DELETE FROM MED WHERE Code_M = C.Code_M;                    } 02
        Commit;                                                     (0.5 pour chaque instruction)
      END IF;
    END IF;
END LOOP;
close retrait;    0.25
END;
```

3ème solution correcte (On fait la condition dans le curseur, donc le premier « IF » sera supprimé)

```
DECLARE
CURSOR retrait is (select Code_M, date_Peremption, Qte_Stock FROM MED
                   WHERE  date_peremption  <= SYSDATE);
existe number;
BEGIN
FOR c IN retrait
LOOP
   IF C.date_Peremption <=SYSDATE THEN
      SELECT count(*) into existe from med_perime where Code_M=C.Code_M;
      IF existe >0 THEN
        UPDATE med_perime SET qte_perime=qte_perime+C.Qte_Stock;
        DELETE FROM MED WHERE Code_M = C.Code_M;
        Commit;
      ELSE
        INSERT INTO med_perime VALUES (C.Code_M,C.Qte_Stock);
        DELETE FROM MED WHERE Code_M = C.Code_M;
        Commit;
      END IF;
   END IF;
END LOOP;
END;
```

0.5

0.75

0.25

0.5

0.5

0.5

02 (0.5 pour chaque instruction)

02 (0.5 pour chaque instruction)

Exercice QCM: *(6.5Pts)*

Questions 1..6 : 0.25 point (00 Si aucune réponse ou plusieurs réponses)

Questions 7..15 : 0.5 point (00 Si aucune réponse ou plusieurs réponses)

Questions 16 : 0.5 point ((+0.25) pour chaque réponse correcte), (-0.25) pour chaque fausse réponse))

Cochez la ou les bonnes réponses.

1) Qu'est-ce qu'une transaction ?

A. Une opération d'écriture dans la base

B. Une opération suivie d'une opération de lecture

C. Une séquence d'opérations, lecture ou écriture, terminée par commit ou rollback

D. La séquence des opérations effectuées par un programme

2) Dire que deux programmes sont concurrents, c'est dire que

A. Ils s'exécutent sur la même machine

B. Ils communiquent avec le même serveur de données

C. Ils peuvent échanger des messages

3) J'exécute plusieurs fois de suite le même programme

A. J'obtiens toujours la même transaction

B. J'obtiens toujours la même séquence de transactions

C. À chaque exécution les transactions peuvent changer

D. Je n'obtiens jamais la même séquence de transactions

4) On représente une transaction par une séquence de lecture et d'écriture parce que

A. Ce serait trop compliqué de prendre en compte les opérations effectuées par le programme client

B. **Les opérations effectuées par le programme sont inconnues du serveur de données**

C. Connaître les opérations effectuées par le programme ne sert à rien

5) Les propriétés ACID des transactions sont

A. Programmées par le développeur d'application

B. Garanties par le SGBD

C. Une situation idéale qui est souvent mise en échec en pratique

6) À quel moment doit-on effectuer un commit

A. Dès que la base est arrivée à un état cohérent

B. Après chaque mise à jour

C. À intervalles périodiques (par exemple toutes les 5 mn)

7) Que signifie « Atomicité »

A. Le serveur effectue toutes les opérations en même temps

B. Le serveur sait annuler ou valider solidairement toutes les opérations d'une même transaction.

C. Le serveur effectue et valide les opérations de lecture ou d'écriture une par une

8) Que se passe-t-il en cas de panne au milieu d'une transaction ?

A. La transaction se finit automatiquement quand le SGBD redémarre

B. Toutes les mises à jour déjà effectuées sont annulées quand le SGBD redémarre

C. Le système a un dispositif de sécurité pour toujours effectuer un `commit` juste avant qu'une panne survienne

9) Pourquoi des anomalies peuvent-elles apparaître dans une exécution concurrente ?

A. Parce qu'il y a une erreur dans le programme

B. Parce que le niveau d'isolation n'est que partiel

C. Parce que les deux transactions qui s'imbriquent sont incompatibles

10) Une « lecture sale », c'est la lecture

A. D'une valeur corrompue

B. D'une valeur modifiée et validée par une autre transaction

C. D'une valeur modifiée et non validée par une autre transaction

11) Qu'est-ce qu'une lecture non répétable ?

A. C'est le fait d'obtenir des résultats différents pour une même requête effectuée à intervalles réguliers

B. C'est le fait d'obtenir des résultats différents pour une même requête effectuée dans la même transaction

C. C'est la lecture dans une table temporaire

12) Lectures sales et lectures non répétables sont des exemples de défauts

A. De durabilité

B. **D'isolation**

C. D'atomicité

D. De cohérence

13) Qu'est-ce qui caractérise le déroulement correct d'une exécution concurrente ?

A. Elle termine sans panne si blocage ni rejet

B. L'état de la base à la fin pourrait être obtenu par une exécution en série des transactions

C. Les opérations de chaque transaction s'exécutent dans l'ordre où elles sont soumises.

14) L'anomalie des « mises à jour perdues » se caractérise par

A. Deux transactions lisent une même donnée chacun de leur côté pour la modifier ensuite

B. Une transaction met à jour une donnée qui vient juste d'être modifiée par une autre transaction

C. Une mise à jour est mise en attente et finit par ne jamais être effectuée

D. Une mise à jour est effectuée mais annulée par un rollback.

15) Une « écriture sale », c'est une écriture

A. Qui remplace une valeur modifiée et validée par une autre transaction

B. Qui remplace une valeur modifiée et non validée par une autre transaction

C. Qui modifie une valeur qui entretemps a été détruite par une autre transaction

16) Une transaction T1 a lu un nuplet x et posé un verrou partagé. Quelles affirmations sont vraies ?

A. si aucun autre verrou n'est posé, T1 peut poser un verrou exclusif sur x

B. T2 peut poser un verrou partagé sur x

C. T2 peut poser un verrou exclusif sur x