DISVE
Licence

Année Universitaire 2007/2008
Session 1 de Printemps

PARCOURS : CSB4 & CSB6
UE : INF 159, Bases de données
Épreuve : INF 159 EX
Date : Mercredi 23 avril 2008
Heure : 8 heures 30 **Durée :** 1 heure 30
Documents : non autorisés
Épreuve de M. Alain Griffault

SUJET + CORRIGE

Avertissement

- La plupart des questions sont indépendantes.
- Le barème total est de 22 points car le sujet est assez long.
- Le barème de chaque question est (approximativement) proportionnel à sa difficulté.
- L'espace pour répondre est suffisant (sauf si vous l'utilisez comme brouillon, ce qui est fortement déconseillé).

Le sujet

Le sujet porte sur une gestion simplifiée d'un tournoi de tennis de table. Un tournoi comprend plusieurs tableaux. Chaque tableau est un ensemble de parties. Chaque partie d'un tableau oppose deux joueurs inscrits. Un joueur inscrit est un joueur licencié dans un pays.
Soit `Tournoi(Jo1_Licence, Jo1_Nom, Jo1_Points, Jo1_Ne_le, Jo1_Iso_pays, Jo1_Pays, Jo2_Licence, Jo2_Nom, Jo2_Points, Jo2_Ne_le, Jo2_Iso_pays, Jo2_Pays, Tableau, Tab_Intitule, Tab_PointsMax, Tab_Ne_apres, Jo1_Gagne)` une relation. Un exemple est donné dans la table suivante, où vu le nombre important d'attributs, les colonnes et lignes sont inversées.

Attributs	Commentaires	partie 1	partie 2	partie 3
Jo1_Licence integer	Numéro du joueur 1	41	48	51
Jo1_Nom text	Son nom	Eloi	Karakasevic	Gardos
Jo1_Points integer	Sa valeur	11833	11776	11755
Jo1_Ne_le date	Sa date de naissance	1975-01-01	1975-01-01	1975-01-01
Jo1_Iso_pays char(3)	Le code de son pays	FRA	SRB	AUT
Jo1_Pays text	Son pays	France	Serbie	Autriche
Jo2_Licence integer	Numéro du joueur 2	58	58	62
Jo2_Nom text	Son nom	Chila	Chila	Legout
Jo2_Points integer	Sa valeur	11701	11701	11693
Jo2_Ne_le date	Sa date de naissance	1975-01-01	1975-01-01	1975-01-01
Jo2_Iso_pays char(3)	Le code de son pays	FRA	FRA	FRA
Jo2_Pays text	Son pays	France	France	France
Tableau char(1)	Un identificateur	C	C	A
Tab_Intitule text	Son intitulé	Qualif JO	Qualif JO	Qualif JO
Tab_PointsMax integer	Valeur limite	12000	12000	12000
Tab_Ne_apres date	Naissance limite	1900-01-01	1900-01-01	1900-01-01
Jo1_Gagne boolean	Vrai=(joueur1 gagne)	False	False	True

Interprétation : La première partie indique que le français `Eloi` a perdu contre le français `Chila` dans le tableau `C`, la seconde indique que le serbe `Karakasevic` a perdu contre le français `Chila` dans le tableau `C` et la troisième indique que l'autrichien `Gardos` a gagné contre le français `Legout` dans le tableau `A`.

Remarque : Les données sont réelles sauf pour les dates de naissance des joueurs.

Pour une gestion correcte, la relation est munie des contraintes d'intégrité suivantes :

C1 : Tous les attributs sont déclarés `NON NULL`.

C2 : `PRIMARY KEY (Jo1_Licence, Jo2_Licence, Tableau)` qui indique la clef primaire.

C3 : `CONSTRAINT Jo1_age_points CHECK(Jo1_Ne_le > Tab_Ne_apres AND Jo1_Points < Tab_PointsMax)` qui indique que le joueur1 n'est pas trop âgé pour pouvoir participer à ce tableau et qu'il n'est pas trop fort pour pouvoir participer à ce tableau.

C4 : `CONSTRAINT Jo2_age_points CHECK(Jo2_Ne_le > Tab_Ne_apres AND Jo2_Points < Tab_PointsMax)` qui indique que le joueur2 n'est pas trop âgé pour pouvoir participer à ce tableau et qu'il n'est pas trop fort pour pouvoir participer à ce tableau.

C5 : `CONSTRAINT Jo1_Jo2 CHECK(Jo1_Licence < Jo2_Licence)` qui indique que (conjugué avec la clef primaire) deux mêmes joueurs ne peuvent pas se rencontrer deux fois dans le même tableau.

Exercice 1 (Requêtes SQL sur `Tournoi` (9 points))

Question 1.1 (1 point) *Écrire une requête SQL qui calcule pour chaque* ***`tableau`*** *le nombre de parties disputées dans celui ci.*

Réponse :

```
-- Le nombre de parties dans chaque tableau
CREATE VIEW TournoiTT.reponse_1_1 AS
SELECT DISTINCT Tableau, COUNT(*)
FROM            TournoiTT.Tournoi
GROUP BY        Tableau;
```

Question 1.2 (1 point) *Soit l'expression algébrique*

$$\alpha[\mathtt{Jo1\_Licence :Licence,\ Jo1\_Nom :Nom}](\pi[\mathtt{Jo1\_Licence,\ Jo1\_Nom}](\sigma[\mathtt{Jo1\_Gagne}](\mathtt{Tournoi})))$$
$$\cup$$
$$\alpha[\mathtt{Jo2\_Licence :Licence,\ Jo2\_Nom :Nom}](\pi[\mathtt{Jo2\_Licence,\ Jo2\_Nom}](\sigma[\neg\ \mathtt{Jo1\_Gagne}](\mathtt{Tournoi})))$$

Écrivez la requête SQL correspondante, puis expliquez ce qu'elle calcule.

Réponse :

```
-- Les joueurs ayant gagné au moins une partie
CREATE VIEW TournoiTT.reponse_1_2 AS
SELECT DISTINCT Jo1_Licence AS Licence, Jo1_Nom AS Nom
FROM            TournoiTT.Tournoi
WHERE           Jo1_Gagne
  UNION
SELECT DISTINCT Jo2_Licence AS Licence, Jo2_Nom AS Nom
FROM            TournoiTT.Tournoi
WHERE           NOT Jo1_Gagne;
```

La requête SQL retourne les licences et les noms des joueurs ayant gagné au moins une partie.

Question 1.3 (1,5 point) *Écrire une expression algébrique qui calcule les licences et les noms des joueurs ayant gagné toutes leurs parties.*

Réponse :

$$(\alpha[\mathtt{Jo1\_Licence :Licence,\ Jo1\_Nom :Nom}](\pi[\mathtt{Jo1\_Licence,\ Jo1\_Nom}](\sigma[\mathtt{Jo1\_Gagne}](\mathtt{Tournoi})))$$
$$\cup$$
$$\alpha[\mathtt{Jo2\_Licence :Licence,\ Jo2\_Nom :Nom}](\pi[\mathtt{Jo2\_Licence,\ Jo2\_Nom}](\sigma[\neg\ \mathtt{Jo1\_Gagne}](\mathtt{Tournoi}))))$$
$$-$$
$$(\alpha[\mathtt{Jo1\_Licence :Licence,\ Jo1\_Nom :Nom}](\pi[\mathtt{Jo1\_Licence,\ Jo1\_Nom}](\sigma[\neg\ \mathtt{Jo1\_Gagne}](\mathtt{Tournoi})))$$
$$\cup$$
$$\alpha[\mathtt{Jo2\_Licence :Licence,\ Jo2\_Nom :Nom}](\pi[\mathtt{Jo2\_Licence,\ Jo2\_Nom}](\sigma[\mathtt{Jo1\_Gagne}](\mathtt{Tournoi}))))$$

Question 1.4 (1,5 point) *Écrire une requête SQL qui permet de vérifier que les contraintes C3, C4 et C5 sont satisfaites.*

Réponse :

```
-- La vérification des contraintes d'integrite
-- Cette vue doit toujours etre vide
CREATE VIEW TournoiTT.reponse_1_4 AS
SELECT      *
FROM        TournoiTT.Tournoi
WHERE       Jo1_Points > Tab_PointsMax
   OR       Jo2_Points > Tab_PointsMax
   OR       Jo1_Ne_le < Tab_Ne_apres
   OR       Jo2_Ne_le < Tab_Ne_apres
   OR       Jo1_Licence > Jo2_Licence;
```

Question 1.5 (2 points) *Écrire une requête SQL qui retourne les licences et les noms des joueurs ayant participé à tous les tableaux. Vous pouvez créer des vues intermédiaires.*

Réponse :

```
-- Les joueurs ayant partcipé à tous les tableaux
-- Algèbre relationnelle
CREATE VIEW TournoiTT.reponse_1_5_V1 AS
(SELECT DISTINCT Jo1_Licence AS Licence, Jo1_Nom AS Nom
 FROM            TournoiTT.Tournoi
   UNION
 SELECT DISTINCT Jo2_Licence AS Licence, Jo2_Nom AS Nom
 FROM            TournoiTT.Tournoi)
EXCEPT
SELECT Licence, Nom
FROM   (SELECT *
        FROM (SELECT DISTINCT Jo1_Licence AS Licence, Jo1_Nom AS Nom
              FROM            TournoiTT.Tournoi
                UNION
              SELECT DISTINCT Jo2_Licence AS Licence, Jo2_Nom AS Nom
              FROM            TournoiTT.Tournoi) As PiR1,
             (SELECT DISTINCT Tableau FROM TournoiTT.Tournoi) AS R2
          EXCEPT
        (SELECT DISTINCT Jo1_Licence AS Licence, Jo1_Nom AS Nom, Tableau
         FROM            TournoiTT.Tournoi
           UNION
         SELECT DISTINCT Jo2_Licence AS Licence, Jo2_Nom AS Nom, Tableau
         FROM            TournoiTT.Tournoi)) AS NonEntierR1;

-- Quelques vues intermediaires
CREATE VIEW TournoiTT.Joueurs AS
SELECT DISTINCT Jo1_Licence AS Licence, Jo1_Nom AS Nom
FROM            TournoiTT.Tournoi
  UNION
SELECT DISTINCT Jo2_Licence AS Licence, Jo2_Nom AS Nom
FROM            TournoiTT.Tournoi;

CREATE VIEW TournoiTT.Tableaux AS
SELECT DISTINCT Tableau FROM TournoiTT.Tournoi;

CREATE VIEW TournoiTT.JoueursTableaux AS
SELECT DISTINCT Jo1_Licence AS Licence, Jo1_Nom AS Nom, Tableau
FROM            TournoiTT.Tournoi
  UNION
SELECT DISTINCT Jo2_Licence AS Licence, Jo2_Nom AS Nom, Tableau
FROM            TournoiTT.Tournoi;

-- Algebre relationnelle avec vues intermediaires
CREATE VIEW TournoiTT.reponse_1_5_V2 AS
```

```
SELECT *
FROM   TournoiTT.Joueurs
  EXCEPT
SELECT Licence, Nom
FROM   (SELECT *
        FROM (SELECT * FROM TournoiTT.Joueurs) As PiR1,
             (SELECT * FROM TournoiTT.Tableaux) AS R2
          EXCEPT
        SELECT * FROM TournoiTT.JoueursTableaux) AS NonEntierR1;

-- Calcul relationnel avec vues intermediaires
CREATE VIEW TournoiTT.reponse_1_5_V3 AS
SELECT *
FROM   TournoiTT.Joueurs AS JO
WHERE NOT EXISTS (
  SELECT *
  FROM   TournoiTT.Tableaux AS TAB
  WHERE NOT EXISTS (
    SELECT DISTINCT Tableau
    FROM   TournoiTT.Tournoi AS COMP
    WHERE  COMP.Tableau = TAB.Tableau
      AND  (COMP.Jo1_Licence = JO.Licence OR COMP.Jo2_Licence = JO.Licence)));
```

Question 1.6 (2 points) *Écrire une requête SQL qui classe pour chaque tableau les joueurs en fonction de leurs résultats (nombre de parties gagnées moins nombre de parties perdues). Vous pouvez créer des vues intermédiaires.*

Réponse :

```
-- Quelques vues intermediaires
-- le nombre de victoires par joueur et par tableau
CREATE VIEW TournoiTT.JoueursVictoires AS
SELECT      Licence, Nom, Tableau, COUNT(*) AS Bilan
FROM        (
  SELECT DISTINCT Jo1_Licence AS Licence, Jo1_Nom AS Nom, Jo2_Licence AS Licence2, Tableau
  FROM            TournoiTT.Tournoi
  WHERE           Jo1_Gagne
    UNION
  SELECT DISTINCT Jo2_Licence AS Licence, Jo2_Nom AS Nom, Jo1_Licence AS Licence2, Tableau
  FROM            TournoiTT.Tournoi
  WHERE           NOT Jo1_Gagne) AS TMP
GROUP BY Licence, Nom, Tableau;

-- le nombre de defaites par joueur et par tableau
CREATE VIEW TournoiTT.JoueursDefaites AS
SELECT      Licence, Nom, Tableau, -COUNT(*) AS Bilan
FROM        (
  SELECT DISTINCT Jo1_Licence AS Licence, Jo1_Nom AS Nom, Jo2_Licence AS Licence2, Tableau
  FROM            TournoiTT.Tournoi
  WHERE           NOT Jo1_Gagne
    UNION
  SELECT DISTINCT Jo2_Licence AS Licence, Jo2_Nom AS Nom, Jo1_Licence AS Licence2, Tableau
  FROM            TournoiTT.Tournoi
  WHERE           Jo1_Gagne) AS TMP
GROUP BY Licence, Nom, Tableau;

-- Pour chaque tableau, un classement des joueurs (victoires - defaites)
CREATE VIEW TournoiTT.reponse_1_6_V1 AS
SELECT Licence, Nom, Tableau, SUM(Bilan) AS Bilan
FROM   (SELECT *
        FROM TournoiTT.JoueursVictoires
          UNION
```

```
        SELECT * FROM TournoiTT.JoueursDefaites) AS TMP
GROUP  BY Licence, Nom, Tableau
ORDER  BY Tableau, SUM(Bilan) DESC, Nom;
```

Exercice 2 (Normalisation (9 points)) *Pour améliorer la gestion des tournois, un informaticien (ayant entendu parler de formes normales) propose de découper la relation `Tournoi` en deux relations `Joueurs(Licence, Nom, Points, Ne_le, Iso_pays, Pays, Tableau, Tab_Intitule, Tab_PointsMax, Tab_Ne_apres)` et `Parties(Licence1, Licence2, Tableau, Jo1_Gagne)`. L'exemple est alors représenté par les deux tables.*

Attributs	*Joueur 1*	*Joueur 2*	*Joueur 3*	*Joueur 4*	*Joueur 5*
Licence integer	*41*	*48*	*51*	*58*	*62*
Nom text	*Eloi*	*Karakasevic*	*Gardos*	*Chila*	*Legout*
Points integer	*11833*	*11776*	*11755*	*11701*	*11693*
Ne_le date	*1975-01-01*	*1975-01-01*	*1975-01-01*	*1975-01-01*	*1975-01-01*
Iso_pays char(3)	*FRA*	*SRB*	*AUT*	*FRA*	*FRA*
Pays text	*France*	*Serbie*	*Autriche*	*France*	*France*
Tableau char(1)	*C*	*C*	*A*	*C*	*A*
Tab_Intitule text	*Qualif JO*	*Qualif JO*	*Qualif JO*	*Qualif JO*	*Qualif JO*
Tab_PointsMax integer	*12000*	*12000*	*12000*	*12000*	*12000*
Tab_Ne_apres date	*1900-01-01*	*1900-01-01*	*1900-01-01*	*1900-01-01*	*1900-01-01*

Attributs	*Commentaires*	*partie 1*	*partie 2*	*partie 3*
Licence1 integer	*Numéro du joueur 1*	*41*	*48*	*51*
Licence2 integer	*Numéro du joueur 2*	*58*	*58*	*62*
Tableau char(1)	*Un identificateur*	*C*	*C*	*A*
Jo1_Gagne boolean	*Vrai=(joueur1 gagne)*	*False*	*False*	*True*

Question 2.1 (1 point) *Donnez pour chacune des contraintes d'intégrité (cf page 2) la nouvelle contrainte équivalente. Précisez la (ou les) relations concernées.*

Réponse :

C'1 : C1 est inchangée.

C'2 : C2 devient `PRIMARY KEY (Licence1, Licence2, Tableau)` dans la relation `Parties`.

C'3 : C3 et C4 deviennent une seule contrainte dans la relation `Joueurs`. `CONSTRAINT Jo_age_points CHECK(Ne_le > Tab_Ne_apres AND Points < Tab_PointsMax)` indique que le joueur n'est pas trop âgé pour pouvoir participer à ce tableau et qu'il n'est pas trop fort pour pouvoir participer à ce tableau.

C'4 : C5 devient `CONSTRAINT Jo1_Jo2 CHECK(Licence1 < Licence2)` dans la relation `Parties`, et indique que (conjugué avec la clef primaire) deux mêmes joueurs ne peuvent pas se rencontrer deux fois dans le même tableau.

Un second informaticien trouve pertinent le découpage, mais explique qu'il faut maintenant lister les dépendances fonctionnelles et les contraintes d'intégrité afin de finir la normalisation. La liste des dépendances fonctionnelles trouvées est la suivante.

DF1 : {`Licence`} $\rightarrow$ {`Nom, Points, Ne_Le, Iso_pays, Pays`}

DF2 : {`Iso_pays`} $\rightarrow$ {`Pays`}

DF3 : {`Tableau`} $\rightarrow$ {`Tab_Intitule, Tab_PointsMax, Tab_Ne_apres`}

DF4 : {`Licence1, Licence2, Tableau`} $\rightarrow$ {`Jo1_Gagne`}

Question 2.2 (1 point) *Donnez toutes les clefs candidates de la relation `Joueurs`.*

Réponse :
– *Les dépendances fonctionnelles donnent :* $C =$ {`Licence, Tableau`}

Question 2.3 (1 point) *Donnez toutes les clefs candidates de la relation `Parties`.*

Réponse :
– *Les dépendances fonctionnelles donnent :* $C =$ {`Licence1, Licence2, Tableau`}

Question 2.4 (1 point) *Même si l'on suppose qu'il n'y a aucun doublon dans `Joueurs`, justifiez pourquoi la relation `Joueurs` n'est pas en troisième forme normale.*

Réponse : *Une seule des explications suivantes est suffisante (liste non exhaustive).*
Non 2NF : *La clef* {`Licence, Tableau`} *contient* {`Licence`} *qui détermine* {`Nom`}.
Non 3NF : *La clef* {`Licence, Tableau`} *et* {`Iso_pays`} ⟶ {`Pays`}.

Question 2.5 (1 point) *Justifiez que la relation* **`Parties`** *est BCNF.*

Réponse : *Le seul déterminant est l'unique clef candidate.*

Question 2.6 (2 points) *Appliquez un algorithme (ou une technique) de normalisation pour obtenir une décomposition, sans perte d'information, de la relation* **`Joueurs`** *en un ensemble de relations BCNF. Vous n'écrirez sur la copie que les nouvelles relations et les dépendances fonctionnelles utilisées lors des projections effectuées.*

Réponse : *Décomposition en BCNF :*
- *`ListePays (Iso_pays, Pays)` qui provient de* {`Iso_Pays`} ⟶ {`Pays`}
- *`ListeTableaux (Tableau, Tab_Intitule, Tab_PointsMax, Tab_Ne_apres)` qui provient de* {`Tableau`} ⟶ {`Tab_Intitule, Tab_PointsMax, Tab_Ne_apres`}
- *`Licencies (Licence, Nom, Points, Ne_le, Iso_pays)` qui provient de* {`Licence`} ⟶ {`Nom, Points, Ne_le, Iso_pays`}
- *`Inscrits (Licence, Tableau)` qui reste après les décompositions précédentes.*

Question 2.7 (1 point) *Donnez pour chaque relation de votre normalisation, les contraintes d'intégrité et les clefs étrangères présentes.*

Réponse :

```
CREATE TABLE TournoiTT.ListePays (
  Iso_pays Char(3) PRIMARY KEY,
  Pays text NOT NULL
);

CREATE TABLE TournoiTT.ListeTableaux (
  Tableau char(1) PRIMARY KEY,
  Intitule text NOT NULL,
  Points_max integer NOT NULL,
  Ne_apres date NOT NULL
);

CREATE TABLE TournoiTT.Licencies (
  Licence integer PRIMARY KEY,
  Nom text NOT NULL,
  Points integer NOT NULL,
  Ne_le date NOT NULL,
  Iso_pays char(3) REFERENCES TournoiTT.ListePays ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE TournoiTT.Inscrits (
  Licence integer REFERENCES TournoiTT.Licencies ON DELETE CASCADE ON UPDATE CASCADE,
  Tableau char(1) REFERENCES TournoiTT.ListeTableaux ON DELETE CASCADE ON UPDATE CASCADE,
  PRIMARY KEY (Licence, Tableau)
);

-- PostgreSQL n'accepte pas les contraintes d'integrite avec un SELECT
-- Il faut definir une fonction
CREATE FUNCTION TournoiTT.Inscrits_Erreur(integer, char(1)) RETURNS boolean AS $$
SELECT NOT EXISTS (
  SELECT *
  FROM TournoiTT.ListeTableaux, TournoiTT.Licencies
  WHERE TournoiTT.ListeTableaux.Tableau=$2
    AND TournoiTT.Licencies.Licence=$1
    AND (Licencies.Points > ListeTableaux.Points_max OR
         Licencies.Ne_le < ListeTableaux.Ne_apres))
$$ LANGUAGE SQL;
-- puis appeler la fonction dans une contrainte d'integrite
ALTER TABLE TournoiTT.Inscrits ADD CONSTRAINT Jo_Age_Points
  CHECK(TournoiTT.Inscrits_Erreur(Licence, Tableau) = TRUE);
```

```
CREATE TABLE TournoiTT.Parties (
  Licence1 integer NOT NULL,
  Licence2 integer NOT NULL,
  Tableau char(1) NOT NULL,
  Jo1_gagne boolean NOT NULL,
  PRIMARY KEY (Licence1, Licence2, Tableau),
  FOREIGN KEY (Licence1, Tableau) REFERENCES TournoiTT.Inscrits(Licence, Tableau)
    ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (Licence2, Tableau) REFERENCES TournoiTT.Inscrits(Licence, Tableau)
    ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT Jo1_Jo2 CHECK(Licence1 < Licence2)
);
```

Question 2.8 (1 point) *Créez une vue* ***TournoiEquiv*** *à partir de votre décomposition qui contiennent exactement les tuples de la relation* ***Tournoi*** *initiale.*

Réponse :

```
CREATE VIEW TournoiTT.TournoiEquiv AS
SELECT Jo1.Licence AS Jo1_Licence,
       Jo1.Nom AS Jo1_Nom,
       Jo1.Points AS Jo1_Points,
       Jo1.Ne_le AS Jo1_Ne_le,
       Jo1.Iso_pays AS Jo1_Iso_pays,
       Jo1.Pays AS Jo1_Pays,
       Jo2.Licence AS Jo2_Licence,
       Jo2.Nom AS Jo2_Nom,
       Jo2.Points AS Jo2_Points,
       Jo2.Ne_le AS Jo2_Ne_le,
       Jo2.Iso_pays AS Jo2_Iso_pays,
       Jo2.Pays AS Jo2_Pays,
       P.Tableau AS Tableau,
       P.Intitule AS Tab_Intitule,
       P.Points_max AS Tab_Points_max,
       P.Ne_apres AS Tab_Ne_apres,
       P.Jo1_gagne AS Jo1_Gagne
FROM   (TournoiTT.Parties NATURAL JOIN TournoiTT.ListeTableaux) AS P,
       (TournoiTT.Licencies NATURAL JOIN TournoiTT.ListePays) AS Jo1,
       (TournoiTT.Licencies NATURAL JOIN TournoiTT.ListePays) AS Jo2
WHERE  P.Licence1=Jo1.Licence AND P.Licence2=Jo2.licence;
```

Exercice 3 (Gestion des accès concurrents (4 points))

Question 3.1 (2 points) *Expliquez pourquoi une solution basée sur :*

1. *Une base accessible en lecture uniquement.*
2. *Une base pour les modifications.*
3. *Une recopie périodique de la base modifiée vers la base accessible en lecture.*

est sans inter-blocage et sans attente infinie.

Réponse : *La base accessible en lecture uniquement n'a pas besoin de verrou. Il n'y a donc ni inter-blocage, ni attente infinie. La notion de transaction est même inutile.*

La base servant aux modifications n'a besoin que du seul verrou ***X*** *exclusif. Avec un seul type de verrou, le protocole de verrouillage strict à deux phases garanti la correction de la base, l'absence d'inter-blocage et l'absence d'attente infinie.*

La recopie périodique permet aux lecteurs d'avoir des données pas trop anciennes, mais ne provoque ni inter-blocage, ni attente infinie.

Question 3.2 (2 points) *Soit une relation* ***Reservation(jour date, train text, wagon integer, place integer, reservee boolean)*** *pour la réservation de place dans les trains. Les applications pour la réservation sont disponibles dans des agences (un seul terminal par agence). Expliquez pourquoi la solution précédente n'est pas adaptée, puis proposez une mise en oeuvre sans verrou, sans inter-blocage, sans attente infinie, mais pas forcément optimale.*

Réponse : *Les réservations de places font partie des applications qui doivent avoir en permanence des données très récentes. Avec l'approche précédente, la recopie de la base devra être très fréquente, et cela pénalisera trop l'exploitation.*

Une solution (utilisée pendant longtemps par la SNCF) consiste à partitionner les places d'un train (souvent en fonction des wagons), et d'attribuer à chaque agence une ou plusieurs classes de la partition. Les requêtes pour une agence sont toujours restreintes à ces seules classes.

Ainsi, il ne peut pas y avoir de conflit, ni en écriture et ni enlecture, pour une place. Les verrous deviennent inutiles, et la solution est sans inter-blocageet sans attente infinie.

La solution n'est pas optimale, car un client peut ne pas obtenir une place car il s'adresse à une agence qui a épuisé son quota, alors que des places libres existent encore dans d'autres agences.