
	<p>ANNÉE UNIVERSITAIRE 2010/2011 SESSION 2 DE PRINTEMPS</p> <p>PARCOURS : CSB4 & CSB6 UE : INF 159, Bases de données Épreuve : INF 159 EX Date : Lundi 20 juin 2011 Heure : 11 heures Durée : 1 heure 30 Documents : non autorisés Épreuve de M. Alain GRIFFAULT</p>	
DISVE Licence		

SUJET + CORRIGE

Avertissement

- La plupart des questions sont indépendantes.
- Le barème total est de 24 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).

Exercice 1 (Conception et SQL (14 points))

L'exercice porte sur une gestion simplifiée de livres d'une bibliothèque.

Le concepteur de la base a conçu un schéma relationnel composé des cinq relations Tarifs, Adherents, Oeuvres, Livres et Emprunts. La première partie de l'exercice consiste à comprendre et à expliquer ce schéma. La seconde partie sera composée de requêtes, et la troisième de variantes.

Les sources sont au format PostGreSQL, et toutes les relations sont préfixées par le nom du schéma Bibliotheque, que vous pouvez omettre dans vos réponses.

CREATE SCHEMA Bibliotheque;

La bibliothèque est constituée de livres. Chaque livre est en fait un des exemplaires d'une oeuvre. Les oeuvres sont décrites dans une relation BCNF.

CREATE TABLE Bibliotheque.Oeuvres (

— *Typage des attributs*

ISBN text **NOT NULL**,

Titre text **NOT NULL**,

Editeur text **NOT NULL**,

Auteur text **NOT NULL**, — *Séparer les auteurs par une virgule pour la lisibilité*

— *Clefs candidates*

PRIMARY KEY (ISBN)

);

Pour pouvoir emprunter simultanément plusieurs livres, chaque adhérent doit déposer une caution dont le tarif est décrit dans une relation BCNF.

CREATE TABLE Bibliotheque.Tarifs (

— *Typage des attributs*

NbEmpruntsAutorises **integer CHECK (NbEmpruntsAutorises >= 0)**,

Caution **integer CHECK (NbEmpruntsAutorises >= 0)**,

— *Clefs candidates*

PRIMARY KEY (NbEmpruntsAutorises)

);

Les adhérents à la bibliothèque sont enregistrés dans une relation.

```

CREATE TABLE Bibliotheque.Adherents (
  — Typage des attributs
  Id serial NOT NULL,          — serial = sequence d'entier 1, 2, 3, ...
  Nom text NOT NULL,
  Prenom text NOT NULL,
  NbEmpruntsAutorises integer NOT NULL,
  — Clefs candidates
  PRIMARY KEY (id),
  — Clefs etrangeres
  FOREIGN KEY (NbEmpruntsAutorises) REFERENCES Bibliotheque.Tarifs(NbEmpruntsAutorises)
);

```

Question 1.1 (1 point) Pour la relation *Adherents*, donnez la seule dépendance fonctionnelle déclarée, puis expliquez à quoi correspond et à quoi sert la clef étrangère.

Réponse :

$$Id \rightarrow (Nom, Prenom, NbEmpruntsAutorises)$$

La clef étrangère *NbEmpruntsAutorises* fait référence à l'attribut de même nom de la relation *Tarifs*. Cela garantit que lors de l'ajout d'un adhérent, la valeur de cet attribut existe dans la relation *Tarifs*.
Les livres sont décrits dans une relation.

```

CREATE TABLE Bibliotheque.Livres (
  — Typage des attributs
  Id serial NOT NULL,          — serial = sequence d'entier 1, 2, 3, ...
  ISBN text NOT NULL,
  DateAchat date NOT NULL,
  — Clefs candidates
  PRIMARY KEY (Id),
  — Clefs etrangeres
  FOREIGN KEY (ISBN) REFERENCES Bibliotheque.Oeuvres(ISBN)
);

```

Question 1.2 (1 point) Pour la relation *Livres*, donnez la seule dépendance fonctionnelle déclarée, puis expliquez à quoi correspond et à quoi sert la clef étrangère.

Réponse :

$$Id \rightarrow (ISBN, DateAchat)$$

La clef étrangère *ISBN* fait référence à l'attribut de même nom de la relation *Oeuvre*. Cela garantit que lors de l'ajout d'un livre, l'oeuvre existe bien existe dans la relation *Oeuvres*.
Les emprunts sont enregistrés dans une relation.

```

CREATE TABLE Bibliotheque.Emprunts (
  — Typage des attributs
  Livre integer NOT NULL,
  DateEmprunt date NOT NULL,
  DateRetour date NOT NULL DEFAULT 'infinity',
  Adherent integer NOT NULL,
  — Clefs candidates
  PRIMARY KEY (Livre, DateEmprunt),
  UNIQUE (Livre, DateRetour),
  — Clefs etrangeres
  FOREIGN KEY (Adherent) REFERENCES Bibliotheque.Adherents(Id),
  FOREIGN KEY (Livre) REFERENCES Bibliotheque.Livres(Id),
  — Contraintes d'integrite elementaire
  CHECK (DateEmprunt < DateRetour)
);

```

— PostgreSQL exige une fonction pour les contraintes d'intégrité avec **SELECT**

```
CREATE FUNCTION Bibliotheque.DatesEmpruntRetourCorrectes(integer, date, date)
  RETURNS boolean AS $$
SELECT NOT EXISTS (
  SELECT *
  FROM Bibliotheque.Emprunts
  WHERE ($1 = Livre AND $2 <= DateRetour AND $3 >= DateEmprunt))
$$ LANGUAGE SQL;
```

— Contrainte d'intégrité statique

```
ALTER TABLE Bibliotheque.Emprunts ADD CONSTRAINT DatesEmpruntRetourPossibles
  CHECK(Bibliotheque.DatesEmpruntRetourCorrectes(Livre, DateEmprunt, DateRetour)=TRUE);
```

Question 1.3 (1 point) Pour la relation **Emprunts**, donnez les seules dépendances fonctionnelles déclarées, expliquez le rôle des clefs étrangères, puis décrivez l'objectif des deux contraintes d'intégrité.

Réponse :

$$(Livre, DateEmprunt) \rightarrow (DateRetour, Adherent)$$

$$(Livre, DateRetour) \rightarrow (DateEmprunt, Adherent)$$

- Adherent doit exister dans la table Adherents
- Livre doit exister dans la table Livres
- La date d'emprunt doit être antérieure à la date de retour.
- Un livre ne peut pas être emprunté par deux adhérents en même temps.

Deux contraintes d'intégrité inter relations sont ajoutées au schéma. PostgreSQL ne permet pas d'attacher des contraintes aux schémas. Ces contraintes sont donc ajoutées à la relation **Emprunts** car elles doivent être vérifiées lors de chaque nouvel emprunt.

— PostgreSQL exige une fonction pour les contraintes d'intégrité avec **SELECT**

— Contrainte d'intégrité inter relations

```
CREATE FUNCTION Bibliotheque.DatesEmpruntAchatCorrectes(integer, date)
  RETURNS boolean AS $$
SELECT EXISTS (
  SELECT *
  FROM Bibliotheque.Livres
  WHERE ($1 = Id AND $2 > DateAchat))
$$ LANGUAGE SQL;
```

— Contrainte d'intégrité inter relations statique

```
ALTER TABLE Bibliotheque.Emprunts ADD CONSTRAINT DatesEmpruntAchatPossibles
  CHECK(Bibliotheque.DatesEmpruntAchatCorrectes(Livre, DateEmprunt) = TRUE);
```

— PostgreSQL exige une fonction pour les contraintes d'intégrité avec **SELECT**

— Contrainte d'intégrité inter relations

```
CREATE FUNCTION Bibliotheque.NombreEmpruntsCorrect(integer, date)
  RETURNS boolean AS $$
SELECT NOT EXISTS (
  SELECT Adherent
  FROM Bibliotheque.Emprunts, Bibliotheque.Adherents
  WHERE ($1 = Adherent AND $1 = Id)
  AND ($2 >= DateEmprunt AND $2 <= DateRetour)
  GROUP BY Adherent, NbEmpruntsAutorises
  HAVING (count(*) >= NbEmpruntsAutorises))
$$ LANGUAGE SQL;
```

— Ajout de la contrainte par appel de la fonction

```
ALTER TABLE Bibliotheque.Emprunts ADD CONSTRAINT NombreEmpruntsPossible
  CHECK(Bibliotheque.NombreEmpruntsCorrect(Adherent, DateEmprunt) = TRUE);
```

Question 1.4 (1 point) Décrivez les objectifs de ces deux contraintes d'intégrité.

Réponse :

1. DatesEmpruntAchat interdit le prêt d'un livre avant qu'il ne soit acheté.

2. NombreEmprunts interdit le prêt de trop de livres simultanément à un adhérent.

Question 1.5 (1 point) En tenant compte uniquement des dépendances fonctionnelles que vous avez listées dans les réponses précédentes, dites si les relations Adherents, Livres et Emprunts sont 3NF et/ou BCNF.

Réponse :

	3NF	BCNF	Justification
Adherents	OUI	OUI	$Id \rightarrow (Nom, Prenom, NbEmpruntsAutorises)$ est la seule DF irréductible à gauche, donc BCNF $BCNF \Rightarrow 3NF$
Livres	OUI	OUI	$Id \rightarrow (ISBN, DateAchat)$ est la seule DF irréductible à gauche, donc BCNF $BCNF \Rightarrow 3NF$
Emprunts	OUI	OUI	Les 2 clefs candidates sont les membres gauches des 2 DFs, donc BCNF $BCNF \Rightarrow 3NF$

Question 1.6 (1 point) Après en avoir donné une écriture algébrique, écrire une requête SQL qui caractérise les identifiants et les noms des Adherents pouvant empruntés plus de 3 livres simultanément.

Réponse : $R = \pi[Identifiant, Nom](\sigma[NbEmpruntsAutorises > 3](Adherents))$

— Les noms des adherents pouvant empruntés plus de 3 livres simultanément

```
SELECT Id, Nom
FROM    Bibliotheque.Adherents
WHERE   Bibliotheque.Adherents.NbEmpruntsAutorises > 3;
```

Question 1.7 (1 point) Après en avoir donné une écriture algébrique, écrire une requête SQL qui caractérise les identifiants, noms et prénoms des lecteurs de l'ouvrage dont l'ISBN est '2-7117-4838-3'.

Réponse : Réponse non unique.

$R = \pi[Adherent, Nom, Prenom](\sigma[Adherent.Id = Emprunts.Adherent \wedge Livres.Id = Emprunts.Livre \wedge ISBN = '2-7117-4838-3'](Adherents \times Livres \times Emprunts))$

— Les nom et prenom des lecteurs de '2-7117-4838-3'

```
SELECT    Adherent, Nom, Prenom
FROM      Bibliotheque.Adherents, Bibliotheque.Livres, Bibliotheque.Emprunts
WHERE     Adherents.Id = Emprunts.Adherent
        AND    Livres.Id = Emprunts.Livre
        AND    ISBN = '2-7117-4838-3';
```

Question 1.8 (1 point) Après en avoir donné une écriture algébrique, écrire une requête SQL qui caractérise les identifiants, noms et prénoms des adherents en retard (à la date du jour) pour retourner un emprunt. La durée d'un prêt est limité à 30 jours. Vous pourrez utiliser `current_date` qui retourne la date du jour et l'opération `('une_date' + integer '30')` qui retourne une date.

Réponse : Réponse non unique.

$R = \pi[Adherent, Nom, Prenom](\sigma[Adherent.Id = Emprunts.Adherent \wedge current_date < DateRetour \wedge current_date > (DateRetour + 30jours)](Adherents \times Emprunts))$

— Les adherents en retard pour retourner un emprunt.

```
SELECT    Id, Nom, Prenom
FROM      Bibliotheque.Adherents, Bibliotheque.Emprunts
WHERE     Adherents.Id = Emprunts.Adherent
        AND    current_date < DateRetour
        AND    current_date > DateEmprunt + integer '30';
```

Question 1.9 (1,5 point) Écrire une requête SQL qui liste identifiants, noms et prénoms des adhérents qui, à la date du jour, ont la possibilité d'emprunter. Ce sont donc les adhérents qui n'ont pas actuellement autant de livres que le nombre qui leur est autorisé.

Réponse :

— *Les adherents ayant le droit d'emprunter a la date du jour*

```
SELECT Id, Nom, Prenom
FROM Bibliotheque.Adherents
WHERE NbEmpruntsAutorises > 0
EXCEPT
SELECT Id, Nom, Prenom
FROM Bibliotheque.Adherents, Bibliotheque.Emprunts
WHERE Adherents.Id = Emprunts.Adherent
AND current_date < DateRetour
GROUP BY Id, Nom, Prenom, NbEmpruntsAutorises
HAVING (NbEmpruntsAutorises <= count(*));
```

Question 1.10 (1 point) Définir une vue OeuvresLues comme le résultat de la requête qui liste les couples (ISBN, Adherent) des livres empruntés.

Réponse :

— *Toutes les couples (oeuvres lues, adherent)*

```
CREATE VIEW OeuvresLues AS
SELECT ISBN, Adherent
FROM Bibliotheque.Livres, Bibliotheque.Emprunts
WHERE Livres.Id = Emprunts.Livre;
```

Question 1.11 (2 points) Écrire une requête SQL qui liste les oeuvres lues par tous les adhérents. Vous pourrez utiliser la vue de la question précédente.

Réponse :

— *Les oeuvres lues par tous les adherents*

— *Division predicative*

```
SELECT DISTINCT ISBN
FROM Bibliotheque.OeuvresLues
WHERE NOT EXISTS
  (SELECT *
   FROM Bibliotheque.Adherents
   WHERE NOT EXISTS
     (SELECT Adherent
      FROM Bibliotheque.OeuvresLues AS R11
      WHERE R11.ISBN = OeuvresLues.ISBN
      AND R11.Adherent = Adherents.Id));
```

— *Division algebrique*

```
SELECT DISTINCT ISBN
FROM Bibliotheque.OeuvresLues
EXCEPT
SELECT ISBN
FROM (SELECT *
      FROM (SELECT DISTINCT ISBN
            FROM Bibliotheque.OeuvresLues) AS PiR1,
          (SELECT Id
            AS Adherent FROM Bibliotheque.Adherents) AS TMP
      EXCEPT
      SELECT *
      FROM Bibliotheque.OeuvresLues) AS NonEntierR1;
```

L'attribut Auteur de la relationOeuvres oblige à décrire les auteurs (lorsqu'ils sont multiples) par une liste unique.

```
INSERT INTO Bibliotheque.Oeuvres
VALUES('0-201-52983_1', 'LaTeX', 'Addison-Wesley', 'Leslie Lamport');
INSERT INTO Bibliotheque.Oeuvres
VALUES('0399-4198', 'Fiabilite des systemes', 'Eyrolles', 'A. Pages, M. Gondran');
INSERT INTO Bibliotheque.Oeuvres
VALUES('2-7117-4838-3', 'Bases de donnees', 'Vuibert', 'Chris Date');
```

```

INSERT INTO Bibliotheque.Oeuvres
VALUES( '0-201-44124-1', 'Automata Theory', 'Addison-Wesley', 'Hopcroft', 'Motwani', 'Ullman' );
INSERT INTO Bibliotheque.Oeuvres
VALUES( '0-262-03270-8', 'Model Checking', 'MIT Press', 'Clarke', 'Grumberg', 'Peled' );

```

Cette contrainte rend très difficile toute requête liée à un auteur donné.

Question 1.12 (1,5 point) Proposez une modification du schéma conceptuel relationnel afin que des requêtes liées à un auteur particulier, identifié par son nom et son prénom, soient possibles. Pour un livre, vous devrez pouvoir afficher sa liste ordonnée des auteurs telle qu'elle apparaît sur la couverture. Vous préciserez les nouvelles dépendances fonctionnelles et contraintes d'intégrité.

Réponse :

C'est le problème de la liste des prénoms d'un individu. Il faut donc créer une relation Auteurs et modifier Oeuvres.

```

CREATE TABLE Bibliotheque.Oeuvres (
  — Type des attributs
  ISBN text NOT NULL,
  Titre text NOT NULL,
  Editeur text NOT NULL,
  Nom text NOT NULL,      — Nom du premier auteur
  Prenom text NOT NULL,   — Prenom du premier auteur
  — Clefs candidates
  PRIMARY KEY (ISBN)
);

CREATE TABLE Bibliotheque.Auteurs (
  — Type des attributs
  Nom text NOT NULL,
  Prenom text NOT NULL,
  ISBN text NOT NULL,
  Ordre integer NOT NULL CHECK (Ordre > 1),
  — Clefs candidates
  PRIMARY KEY (Nom, Prenom),
  UNIQUE (ISBN, Ordre),
  — Clefs etrangeres
  FOREIGN KEY (ISBN) REFERENCES Bibliotheque.Oeuvres(ISBN)
);

```

Il faut ajouter une contrainte pour que les valeurs pour l'attribut **Ordre** pour une oeuvre donnée forme la suite (éventuellement vide) 2, 3, 4,

Exercice 2 (Normalisation (7 points))

Soit la relation *ColoniesVacances* (*Lieu*, *Transport*, *NbPlaces*, *Jour*, *Groupe*, *Activite*, *Animateur*), vision simplifiée d'une gestion d'un centre de vacances pour enfants, et un ensemble irréductible de dépendances fonctionnelles :

- {*Lieu*} → {*Transport*} : les activités se déroulent dans des lieux qui nécessitent un moyen de transport (marche, vélo, bus ...).
- {*Animateur*} → {*Activite*} : les animateurs sont spécialisés dans une seule activité.
- {*Lieu*, *Jour*} → {*Groupe*} : chaque jour, un lieu n'est utilisé que par un seul groupe.
- {*Transport*} → {*NbPlaces*} : chaque moyen de transport limite le nombre d'enfants.
- {*Groupe*, *Activite*} → {*Animateur*} : pour faciliter le planning, un groupe fait toujours une même activité avec le même animateur.
- {*Lieu*, *Jour*} → {*Animateur*} : chaque jour, un lieu n'est utilisé que par un seul animateur.
- {*Groupe*, *Jour*} → {*Lieu*} : chaque jour, un groupe ne se déplace que sur un seul lieu.
- {*Animateur*, *Jour*} → {*Lieu*} : chaque jour, un animateur ne se déplace que sur un seul lieu.

Lieu	Transport	NbPlaces	Jour	Groupe	Activite	Animateur
plage	marche	20	27-07-2010	Jaune	surf	Brice
stade	velo	10	26-07-2010	Vert	tir a l'arc	Guillaume

Question 2.1 (1 point) Donnez toutes les clefs candidates de la relation *ColoniesVacances*.

Réponse : Les dépendances fonctionnelles donnent :

- $C_1 = \{\text{Lieu}, \text{Jour}\}$
- $C_2 = \{\text{Animateur}, \text{Jour}\}$
- $C_3 = \{\text{Groupe}, \text{Jour}\}$

Question 2.2 (1 point) Même si l'on suppose qu'il n'y a aucun doublon dans *ColoniesVacances*, justifiez pourquoi la relation *ColoniesVacances* n'est pas en troisième forme normale.

Réponse : Une seule des explications suivantes est suffisante (liste non exhaustive).

Non 2NF : La clef $\{\text{Lieu}, \text{Jour}\}$ contient $\{\text{Lieu}\}$ qui détermine $\{\text{Transport}\}$.

Non 3NF : La clef $\{\text{Lieu}, \text{Jour}\}$ et $\{\text{Transport}\} \rightarrow \{\text{NbPlaces}\}$.

Non 3NF : La clef $\{\text{Lieu}, \text{Jour}\}$ et $\{\text{Animateur}\} \rightarrow \{\text{Activite}\}$.

Question 2.3 (2 points) Appliquez un algorithme (ou une technique) de normalisation pour obtenir une décomposition, sans perte d'information, de la relation *ColoniesVacances* en un ensemble de relations au moins en troisième forme normale. Vous n'écrirez sur la copie que les nouvelles relations et les dépendances fonctionnelles qui sont à la base des projections effectuées.

Réponse : Décomposition en BCNF :

1. *Transports* (*Transport*, *NbPlaces*) qui provient de $\{\text{Transport}\} \rightarrow \{\text{NbPlaces}\}$ (BCNF)
2. *Lieux* (*Lieu*, *Transport*) qui provient de $\{\text{Lieu}\} \rightarrow \{\text{Transport}\}$ (BCNF)
3. *Animateurs* (*Animateur*, *Activite*) qui provient de $\{\text{Animateur}\} \rightarrow \{\text{Activite}\}$ (BCNF) mais la dépendance $\{\text{Groupe}, \text{Activite}\} \rightarrow \{\text{Animateur}\}$ devient une contrainte inter relations.
4. *Planning* (*Lieu*, *Jour*, *Groupe*, *Animateur*) (BCNF)

Décomposition en 3NF :

1. *Transports* (*Transport*, *NbPlaces*) qui provient de $\{\text{Transport}\} \rightarrow \{\text{NbPlaces}\}$ (BCNF)
2. *Lieux* (*Lieu*, *Transport*) qui provient de $\{\text{Lieu}\} \rightarrow \{\text{Transport}\}$ (BCNF)
3. *Planning* (*Lieu*, *Jour*, *Groupe*, *Animateur*) qui provient de $\{\text{Lieu}, \text{Jour}\} \rightarrow \{\text{Groupe}, \text{Animateur}\}$ (BCNF)
4. *TypesColoniesVacances* (*Groupe*, *Activite*, *Animateur*) qui provient de $\{\text{Groupe}, \text{Activite}\} \rightarrow \{\text{Animateur}\}$ et qui contient $\{\text{Animateur}\} \rightarrow \{\text{Activite}\}$. (3NF et non BCNF)

Question 2.4 (3 points) Après avoir précisé si votre décomposition est en BCNF ou bien seulement en 3NF, répondez aux questions qui vous concernent.

Votre décomposition est en BCNF :

1. Indiquez la dépendance fonctionnelle que vous avez perdue.
2. En supposant que cette dépendance ne soit pas écrite sous forme d'une contrainte, donnez un ensemble de requêtes d'insertion pour vos relations, qui viole cette dépendance fonctionnelle.

Votre décomposition est seulement en 3NF :

1. Indiquez le problème de redondance qui subsiste.
2. Donnez un ensemble de requêtes d'insertion pour vos relations, suivi d'une requête de mise à jour qui nécessite que le SGBD modifie éventuellement plusieurs tuples, du fait de la redondance.

Réponse :

Décomposition en BCNF :

1. $\{\text{Groupe}, \text{Activite}\} \rightarrow \{\text{Animateur}\}$.
2. Les insertions suivantes sont possibles :

```
INSERT INTO Animateurs VALUES (go1, act1);
INSERT INTO Animateurs VALUES (go2, act1);
INSERT INTO Planning VALUES (lieu1, '01-01-2011', rouge, go1);
INSERT INTO Planning VALUES (lieu1, '02-01-2011', rouge, go2);
```

Votre décomposition est seulement en 3NF :

1. L'information (*Animateur*, *Activite*) est dupliquée.
2. La mise à jour suivante modifie plusieurs tuples.

```

INSERT INTO TypesColoniesVacances VALUES (rouge, act1, go1);
INSERT INTO TypesColoniesVacances VALUES (vert, act1, go1);
UPDATE TypesColoniesVacances SET (Activite = act2) WHERE animateur = go1;

```

Exercice 3 (Le risque "NULL" (3 points))

Dans la documentation anglaise PostgreSQL 8.4, on trouve :

To check whether a value is or is not null, use the constructs :

- expression IS NULL
- expression IS NOT NULL

Do not write expression = NULL because NULL is not "equal to" NULL. (The null value represents an unknown value, and it is not known whether two unknown values are equal.) This behavior conforms to the SQL standard.

Tip : Some applications might expect that expression = NULL returns true if expression evaluates to the null value. It is highly recommended that these applications be modified to comply with the SQL standard. However, if that cannot be done the transform_null_equals configuration variable is available. If it is enabled, PostgreSQL will convert x = NULL clauses to x IS NULL.

Ordinary comparison operators yield null (signifying "unknown"), not true or false, when either input is null. For example, 7 = NULL yields null.

Dans la documentation française PostgreSQL 8.4, on trouve :

Pour vérifier si une valeur est NULL ou non, on utilise les constructions

- expression IS NULL
- expression IS NOT NULL

On ne peut pas écrire expression = NULL parce que NULL n'est pas « égal à » NULL. (La valeur NULL représente une valeur inconnue et il est impossible de dire si deux valeurs inconnues sont égales.) Ce comportement est conforme au standard SQL.

Astuce : Il se peut que des applications s'attendent à voir expression = NULL évaluée à vrai (true) si expression s'évalue comme la valeur NULL. Il est chaudement recommandé que ces applications soient modifiées pour se conformer au standard SQL. Néanmoins, si cela n'est pas possible, le paramètre de configuration transform_null_equals peut être utilisé. S'il est activé, PostgreSQL convertit les clauses x = NULL en x IS NULL.

L'opérateur standard de comparaison renvoie NULL (ce qui signifie « inconnu ») si l'une des entrées est NULL, ni true ni false, c'est-à-dire 7 = NULL renvoie NULL.

Soit la relation TableNull résultat des commandes suivantes :

```

CREATE SCHEMA ProblemeNull;
SET search_path TO ProblemeNull, public;
CREATE TABLE ProblemeNull.TableNull (
    -- Typage des attributs
    Id serial NOT NULL,          -- serial = sequence d'entier 1, 2, 3, ...
    Att text DEFAULT NULL,
    -- Clefs candidates
    PRIMARY KEY (id)
);
INSERT INTO ProblemeNull.TableNull VALUES(DEFAULT, 'A');
INSERT INTO ProblemeNull.TableNull VALUES(DEFAULT, DEFAULT);

```

Pour toutes les questions, on considère que le paramètre de configuration transform_null_equals n'est pas activé.

Question 3.1 (3 points) Donnez le résultat des requêtes suivantes, ou Predicat est une expression qui retourne vrai ou faux, en écrivant "OUI" ou "NON" suivant si le tuple est dans la réponse.

```

SELECT *
FROM   ProblemeNull.TableNull
WHERE  Predicat;

```

Réponse :

<i>Id</i>	<i>Att</i>	<i>Att IS NULL</i>	<i>Att IS NOT NULL</i>	<i>NOT (Att IS NULL)</i>
<i>1</i>	<i>A</i>	<i>Non</i>	<i>Oui</i>	<i>Oui</i>
<i>2</i>	<i>“NULL”</i>	<i>Oui</i>	<i>Non</i>	<i>Non</i>

<i>Id</i>	<i>Att</i>	<i>Att = NULL</i>	<i>Att != NULL</i>	<i>NOT (Att = NULL)</i>
<i>1</i>	<i>A</i>	<i>Non</i>	<i>Non</i>	<i>Non</i>
<i>2</i>	<i>“NULL”</i>	<i>Non</i>	<i>Non</i>	<i>Non</i>