INF3710 –Fichiers et Bases de données

Hiver 2019

TP No. [4]

Groupe [1]

[1991079] – [Brus Mathieu]

[1923715] – [Lyes Heythem BETTACHE]

[1991103] – [Taliercio Antoine]

Soumis à : Moodle

[03/26/2019]

Partie 2 : Algèbre relationnelle et SQL

Question 1.1

1) Retrouver les détails de tous les spectacles en 2010

$$\sigma_{annee=2010}(Spectacle)$$

2) Retrouver le détail de tous les danseurs qui ne sont pas dans la vingtaine

$$\sigma_{age<20 \vee age>30}(Danseur)$$

3) Retrouver le nom de tous les directeurs artistiques Canadiens

$$\sigma_{nationalite="Canadien"}(DirecteurArtistique)$$

4) Retrouver le nom de chaque danseur ainsi que les titres des Spectacles dans lesquels il/elle s'est produit

$$\pi_{nom,titre}(Danseur \bowtie \text{Performance} \bowtie \text{Spectacle})$$

5) Trouver les noms de tous les danseurs qui ont dansé le rôle du 'cygne' ainsi que l'année du spectacle

$$\pi_{nom,annee}(Danseur \bowtie \sigma_{role="cygne"}(Performance) \bowtie \text{Spectacle})$$

6) Retrouver toutes les informations des danseurs du Spectacle 'Opus Cactus' sans opération non nécessaire (indice : vous ne pouvez pas utiliser uniquement un join)

$$\pi_{DanseurId,nom,nationalité,annee}(Danseur \bowtie \text{Performance} \bowtie \sigma_{titre="Opus\ Cactus"}(Spectacle))$$

7) Retrouver les titres de tous les spectacles dans lesquels les danseurs Philippe et Kate ont dansé ensemble

R1 ← $\pi_{titre}(\sigma_{nom="Philippe"}(Danseur) \bowtie \text{Performance} \bowtie Spectacle)$

R2 ← $\pi_{titre}(\sigma_{nom="Kate"}(Danseur) \bowtie \text{Performance} \bowtie Spectacle)$

R ← R1 ∩ R2

Question 1.2

8) Quel est l'âge moyen des danseurs ? Stockez-le dans une colonne nommée AgeMoyen.

$$\rho_R\ (\text{AgeMoyen})\ \Im_{AVG\ age}(\text{Danseur})$$

SQL : SELECT AVG(age) AS AgeMoyen FROM Danseur

9) Quels danseurs (Nom) ont dansé dans au moins un spectacle où la danseuse Lucie Tremblay n'a pas dansé ?

SpectacleTremblay ← $\pi_{SpectacleId}\left(\sigma_{nom="Lucie\ Tremblay"}(\text{Danseur}\bowtie\text{Performance})\right)$

SpectacleSansTremblay ← $\pi_{SpectacleId}(\text{Performance}) - SpectacleTremblay$

PerformanceSansTremblay ← $Performance\bowtie SpectacleSansTremblay$

R ← $\pi_{nom}(\text{Danseur}\bowtie\text{PerformanceSansTremblay})$

SQL : SELECT nom FROM Danseur NATURAL JOIN

(SELECT SpectacleId FROM Performance EXCEPT

(SELECT SpectacleId FROM Danseur, Performance WHERE nom = 'Lucie

Tremblay'))

10) Quel est le nombre de spectacles du danseur dont l'id = 1 ? Stockez le résultat dans une colonne nommée nbSpectacle.

$$\rho_R\ (\text{nbSpectacle})\ \Im_{COUNT*}(\sigma_{DanseurId=1}(\text{Performance}))$$

SQL : SELECT COUNT(*) AS nbSpectacle FROM Performance WHERE danseurId=1

11) Affichez une liste des danseurs ainsi que les spectacles (ID) qui leur sont associés s'ils existent, sinon affichez null. L'attribut en commun ne doit pas être répété.

$$\pi_{DanseurId,nom,nationalite,age,SpectacleId}(Danseur\bowtie\text{Performance}\bowtie Spectacle)$$

SQL : SELECT DanseurId, nom, nationalite, age, SpectacleId FROM Danseur

NATURAL LEFT OUTER JOIN Performance NATURAL LEFT OUTER JOIN Spectacle

12) Combien de spectacles existent par catégorie ? Stockez le résultat en donnant un nom à la ou les colonnes correspondantes de la relation résultat.

$$\rho_R\ (\text{Categorie}, \text{nbSpectacleParCat})\ {}_{categorie}\Im_{COUNT*}\ (\text{Spectacle})$$

SQL : SELECT Categorie, COUNT(*) AS nbSpectaclesParCat FROM Spectacle GROUP BY Categorie

13) Quels danseurs (affichez leurs détails) n'ont participé à aucun spectacle ?

DanseurSpectacle ← $\pi_{DanseurId,nom,nationalite,age}(Danseur\bowtie\text{Performance})$

R ← Danseur - DanseurSpectacle

SQL : SELECT * FROM Danseur EXCEPT

(SELECT DISTINCT danseurId, nom, nationalite, age FROM Danseur NATURAL JOIN Performance)

Partie 3 : Transactions

Question 1-a

Transaction A :

```
postgres=# \set AUTCOMMIT 'off';
postgres=# BEGIN;
BEGIN
postgres=# SET TRANSACTION
postgres-# ISOLATION LEVEL READ
postgres-# COMMITTED;
SET
postgres=# SELECT balance -200 as bal
postgres-# into balancea
postgres-# FROM Accounts WHERE acctID
postgres-# = 101;
SELECT 1
postgres=# SELECT bal FROM balancea;
 bal
-----
 800
(1 ligne)


postgres=#
```

Transaction B :

```
postgres=# \set AUTCOMMIT 'off';
postgres=# BEGIN;
BEGIN
postgres=# SET TRANSACTION
postgres-# ISOLATION LEVEL READ
postgres-# COMMITTED;
SET
postgres=# SELECT balance - 500 as bal
postgres-# into balanceb
postgres-# FROM Accounts WHERE acctID =
postgres-# 101;
SELECT 1
postgres=# SELECT bal from balanceb;
 bal
-----
 500
(1 ligne)


postgres=#
```

Transaction A :

```
postgres=# UPDATE Accounts
postgres-# SET balance = (select bal
postgres(# from balancea)
postgres-# WHERE acctID = 101;
UPDATE 1
postgres=#
```

Transaction B :

```
postgres=# UPDATE Accounts
postgres-# SET balance = (select bal
postgres(# from balanceb)
postgres-# WHERE acctID = 101;
```

Transaction A :

```
postgres=# SELECT acctID, balance
postgres-# FROM Accounts
postgres-# WHERE acctID = 101;
 acctid | balance
--------+---------
    101 |     800
(1 ligne)


postgres=# COMMIT;
COMMIT
postgres=#
```

Transaction B :

```
postgres=# UPDATE Accounts
postgres-# SET balance = (select bal
postgres(# from balanceb)
postgres-# WHERE acctID = 101;
UPDATE 1
postgres=# SELECT acctID, balance
postgres-# FROM Accounts
postgres-# WHERE acctID = 101;
 acctid | balance
--------+---------
    101 |     500
(1 ligne)


postgres=# COMMIT;
COMMIT
postgres=#
```

Lorsque la transaction B veut faire l'UPDATE sur la balance de acctId=101, elle est bloquée/en attente car la transaction A n'a pas fait de commit et B veut agir sur le même tuple que A. La transaction A affiche ensuite la balance mise à jour par A (800). Après le commit, la transaction B est débloquée. Puis, la transaction B affiche la balance mise à jour par B (500). Le problème est qu'à la fin le tuple n'a pris en compte que la modification effectuée par B et pas du tout celle de A.

Question 1-b

Transaction A :

```
postgres=# \set AUTCOMMIT 'off';
postgres=# BEGIN;
BEGIN
postgres=# SET TRANSACTION
postgres-# ISOLATION LEVEL REPEATABLE READ;
SET
postgres=# SELECT balance -200 as bal
postgres-# into balancea
postgres-# FROM Accounts WHERE acctID
postgres-# = 101;
SELECT 1
postgres=# SELECT bal FROM balancea;
 bal
-----
 800
(1 ligne)


postgres=#
```

Transaction B :

```
postgres=# \set AUTCOMMIT 'off';
postgres=# BEGIN;
BEGIN
postgres=# SET TRANSACTION
postgres-# ISOLATION LEVEL REPEATABLE READ;
SET
postgres=# SELECT balance - 500 as bal
postgres-# into balanceb
postgres-# FROM Accounts WHERE acctID =
postgres-# 101;
SELECT 1
postgres=# SELECT bal from balanceb;
 bal
-----
 500
(1 ligne)


postgres=#
```

Transaction A :

```
postgres=# UPDATE Accounts
postgres-# SET balance = (select bal
postgres(# from balancea)
postgres-# WHERE acctID = 101;
UPDATE 1
postgres=#
```

Transaction B :

```
postgres=# UPDATE Accounts
postgres-# SET balance = (select bal
postgres(# from balanceb)
postgres-# WHERE acctID = 101;
```

Transaction A :

```
postgres=# SELECT acctID, balance
postgres-# FROM Accounts
postgres-# WHERE acctID = 101;
 acctid | balance
--------+---------
    101 |     800
(1 ligne)


postgres=# COMMIT;
COMMIT
postgres=#
```

Transaction B :

```
postgres=# UPDATE Accounts
postgres-# SET balance = (select bal
postgres(# from balanceb)
postgres-# WHERE acctID = 101;
ERREUR:  n'a pas pu sérialiser un accès à cause d'une mise à jour en parallèle
```

Pour s'assurer que les résultats soient cohérents, on change le niveau d'isolation des transactions a REPEATABLE READ. Ainsi, on ne peut pas modifier le même objet au même moment, la deuxième transaction est annulée et une erreur apparait.

Question 2-a

Transaction A :

```
postgres=# \set AUTCOMMIT 'off';
postgres=# BEGIN;
BEGIN
postgres=# SET TRANSACTION
postgres-# ISOLATION LEVEL READ
postgres-# COMMITTED;
SET
postgres=# SELECT * FROM Accounts
postgres-# WHERE balance > 500;
 acctid | balance
--------+---------
    101 |    1000
    202 |    2000
(2 lignes)


postgres=#
```

Transaction B :

```
postgres=# \set AUTCOMMIT 'off';
postgres=# BEGIN;
BEGIN
postgres=# UPDATE Accounts
postgres-# SET balance = balance - 500
postgres-# WHERE acctID = 101;
UPDATE 1
postgres=# UPDATE Accounts
postgres-# SET balance = balance + 500
postgres-# WHERE acctID = 202;
UPDATE 1
postgres=# SELECT * FROM Accounts;
 acctid | balance
--------+---------
    101 |     500
    202 |    2500
(2 lignes)


postgres=# COMMIT;
COMMIT
postgres=#
```

Transaction A :

```
postgres=# SELECT * FROM Accounts
postgres-# WHERE balance > 500;
 acctid | balance
--------+---------
    202 |    2500
(1 ligne)


postgres=#
```

Le problème est que dans la transaction A, la même requête SELECT renvoie deux résultats différents. On a une anomalie Nonrepeatable read (lecture non reproductible).

Question 2-b

Transaction A :

```
postgres=# \set AUTCOMMIT 'off';
postgres=# BEGIN;
BEGIN
postgres=# SET TRANSACTION
postgres-# ISOLATION LEVEL
postgres-# REPEATABLE READ;
SET
postgres=# SELECT * FROM Accounts
postgres-# WHERE balance > 500;
 acctid | balance
--------+---------
    101 |    1000
    202 |    2000
(2 lignes)


postgres=#
```

Transaction B :

```
postgres=# \set AUTCOMMIT 'off';
postgres=# BEGIN;
BEGIN
postgres=# UPDATE Accounts
postgres-# SET balance = balance - 500
postgres-# WHERE acctID = 101;
UPDATE 1
postgres=# UPDATE Accounts
postgres-# SET balance = balance + 500
postgres-# WHERE acctID = 202;
UPDATE 1
postgres=# SELECT * FROM Accounts;
 acctid | balance
--------+---------
    101 |     500
    202 |    2500
(2 lignes)


postgres=# COMMIT;
COMMIT
postgres=#
```

Transaction A :

```
postgres=# SELECT * FROM Accounts
postgres-# WHERE balance > 500;
 acctid | balance
--------+---------
    101 |    1000
    202 |    2000
(2 lignes)


postgres=#
```

Si on change le niveau d'isolation de la transaction A à REPEATABLE READ, la requête A renvoie le même résultat pour les deux SELECT car quand la transaction commence, elle ne voit pas les changement faits par une autre transaction. Il n'y a plus l'anomalie Nonrepeatable read (lecture non reproductible).

Question 2-c

Transaction A :

```
postgres=# \set AUTCOMMIT 'off';
postgres=# BEGIN;
BEGIN
postgres=# SET TRANSACTION
postgres-# ISOLATION LEVEL REPEATABLE
postgres-# READ READ ONLY;
SET
postgres=#
```

Transaction B :

```
postgres=# \set AUTCOMMIT 'off';
postgres=# BEGIN;
BEGIN
postgres=# INSERT INTO Accounts
postgres-# (acctID,
postgres(# balance) VALUES (301,3000);
INSERT 0 1
postgres=#
```

Transaction A :

```
postgres=# SELECT * FROM Accounts
postgres-# WHERE balance > 1000;
 acctid | balance
--------+---------
    202 |    2000
(1 ligne)


postgres=#
```

Transaction B :

```
postgres=# INSERT INTO Accounts
postgres-# (acctID,
postgres(# balance) VALUES (302,3000);
INSERT 0 1
postgres=#
```

Transaction A :

```
postgres=# SELECT * FROM Accounts
postgres-# WHERE balance > 1000;
 acctid | balance
--------+---------
    202 |    2000
(1 ligne)


postgres=# COMMIT;
COMMIT
postgres=#
```

Le problème est que la transaction A ne voit pas les insertions effectuées par B.

Question 3

Transaction A :

```
postgres=# \set AUTCOMMIT 'off';
postgres=# BEGIN;
BEGIN
postgres=# UPDATE Accounts
postgres-# SET balance = balance - 500
postgres-# WHERE acctID = 101;
UPDATE 1
postgres=#
```

Transaction B :

```
postgres=# \set AUTCOMMIT 'off';
postgres=# BEGIN;
BEGIN
postgres=# UPDATE Accounts
postgres-# SET balance = balance + 500
postgres-# WHERE acctID = 202;
UPDATE 1
postgres=# UPDATE Accounts
postgres-# SET balance = balance + 500
postgres-# WHERE acctID = 101;
```

Transaction A :

```
postgres=# UPDATE Accounts
postgres-# SET balance = balance - 500
postgres-# WHERE acctID = 202;
ERREUR:  Bloquage mortel détecté
DÉTAIL : Le processus 2560 attend ShareLock sur transaction 907 ; bloqué par le processus 15020.
Le processus 15020 attend ShareLock sur transaction 906 ; bloqué par le processus 2560.
ASTUCE : Voir les journaux applicatifs du serveur pour les détails sur la requête.
CONTEXTE : lors de la mise à jour de la ligne (0,70) dans la relation « accounts »
postgres=#
```