M1–Master Informatique – 2020/2021

Bases de données relationnelles

avril 2021

Transactions

Exercice 1 : Vous allez réaliser cet exercice en ouvrant deux connexions à postgres via `psql`. L'objectif des questions qui suivent est de mettre en évidence les problèmes liés à la concurrence.
Pour commencer, créez la table T :

```
create table T(a int, b int);
```

Par la suite, nous utiliserons les abbréviations :

- `select T` pour `select * from T ;`
- `select T(A = 1)` pour `select * from T where A=1 ;`
- `insert (3,6)` pour `insert into T values (3,6);`
- `update(A <- A+1)` pour `update T set A = A+1 ;`
- `update(A <- 3)|(A = 4)` pour `update T set A=3 where A=4 ;`
- `delete(A = 4)` pour `delete from T where A=4 ;`
- on n'écrira pas les `;` après `begin`, `commit` et `rollback`, **n'oubliez pas de les ajouter.**

Insérez les lignes `(0,0)`, `(2,3)` et `(0,1)`.

Transactions standards - le mode READ COMMITTED

Question 1.1 :

```
      session 1
------------------
insert (4,4)
insert ('a','b')
select T
```

Quelles instructions sont annulées ?

```
      session 1
------------------
begin
insert (5,5)
insert (6,6)
select T
rollback
select T
```

Et maintenant ?

Question 1.2 :

```
      session 1                 session 2
-------------------------------------------------------
   begin                     begin
   select T
                             update(A <- A+1)
   select T
                             commit
   select T
   commit
-------------------------------------------------------
```

Quelle données voit la session 1 ; quel problème est mis en évidence ?

Question 1.3 :

```
      session 1                 session 2
-------------------------------------------------------
   begin                     begin
   update(A <- A+1)
   select T
                             insert(3,7)
                             select T
                             delete(A=2)
                             delete(A=1)
   commit
                             commit
-------------------------------------------------------
```

L'ordonnancement de ces deux transactions est-il sériable, c'est à dire équivalent à une exécution en série ?

Question 1.4 :

```
      session 1                 session 2
-------------------------------------------------------
   begin                    begin
   update(A<-3)|(A=2)
                            update(B<-2)|(B=3)
                            select T
   select T
                            update(A<-3)|(A=2)
   update(B<-2)|(B=3)
   commit                   commit
-------------------------------------------------------
```

Quels sont les verrous posés par `session 1` et `session 2` avant le deuxième `update` de `session 2` ? Comment se terminent ces transactions ?

Question 1.5 : En SQL, il existe une clause `for update` à l'instruction `select`. L'expérience suivante va vous permettre de comprendre son utilité.

```
      session 1                   session 2
-------------------------------------------------------
   begin                        begin
   select T(B=7) for update
                                update(B<-10)|(B=0)
                                select T ;
                                update(A<-5)|(A=3)
   update(B<-6)|(B=7) ;
   commit

                                commit
                                select T
-------------------------------------------------------
```

Quels verrous sont posés ; est-ce qu'il y a des verrous au niveau ligne ou seulement au niveau table ?

Transaction serialisable

Question 1.6 :

```
session 1                  session 2
-------------------------------------------------------
begin                       begin
select T
update(A<-5 | A=6)
                           set transaction isolation level serializable ;
                           select T
                           update(B<-9 | B=4)
select T
commit
                           select T
                           rollback
-------------------------------------------------------
```

Que se passe-t-il à chaque étape de cet ordonnancement ?

Question 1.7 :

```
session 1                  session 2
-------------------------------------------------------
begin
                           begin
select T
update(A<-A-1)
                           set transaction isolation level serializable ;
                           select T
                           insert (2,2)
```

```
                            delete(B=1)
select T
rollback
                            select T
                            commit
-----------------------------------------------------
```

Comparez avec l'ordonnancement de la question précédente.

Question 1.8 :

```
session 1                   session 2
-----------------------------------------------------
                            begin
                            set transaction isolation level serializable ;
                            select T
begin
update(A<-6 | B=10)
                            select T
select T
insert(1,1)
select T                    select T
commit
                            select T
                            commit
                            select T
-----------------------------------------------------
```

Comparez avec la question 1.2, quel problème est corrigé ?

Exercice 2 : Dans cet exercice, nous allons étudier la vérification des contraintes par le serveur, dans un contexte transactionnel. Parfois il est nécessaire de modifier les clés primaires, donc les clés étrangères, et ça peut poser des problèmes d'interaction avec la vérification des contraintes.
La table choisie en exemple contient un ensemble d'écrivains, chacun ayant influencé un autre écrivain (au plus). Le schéma que vous trouverez dans les script `ecrivains.sql` est le suivant :

```
create table Ecrivain(
pid int primary key,
nom varchar(30) not null,
influence int references Ecrivain
);
```

Question 2.1 : Essayez avec un `update` de modifier toutes les lignes de la table en incrémentant de 1 les clés primaires et les clés étrangères. Quel est le message d'erreur ? Est-ce que les contraintes sont vérifiées à la fin de la transaction ou en cours de transaction ?

Question 2.2 : Pour éviter d'avoir des conflits avec les valeurs de clé primaire, dans quel ordre doit-on modifier les lignes de la table ? Ecrivez un programme (bloc anonyme) qui réalise cette mise à jour. La syntaxe d'un bloc anonyme est la suivante :

```
DO $$
DECLARE
  déclaration des variables
BEGIN
  code
END ;
$$ LANGUAGE plpgsql;
```

Malheureusement ce programme ne peut pas encore s'exécuter correctement à cause des clés étrangères.

Question 2.3 : Postgres permet de différer la vérification des contraintes de clé étrangère à la fin de la transaction (on ne peut pas différer les autres types de contraintes). Essayez ces deux solutions pour corriger le problème de la clé étrangère :

1. Reporter la vérification de la contrainte de clé étrangère à la fin de la transaction. Cela se fait avec l'instruction SQL suivante :

   ```
   alter table Ecrivain
   alter constraint ecrivain_influence_fkey deferrable initially deferred;
   ```

2. Définir la contrainte de clé étrangère avec la clause `ON UPDATE CASCADE` et modifier le programme de la question précédente pour ne modifier que la colonne `pid`. Supprimer et ajouter une contrainte se fait avec un `ALTER TABLE`.