

Bases de données relationnelles

avril 2024

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

```
create table T(a int, b int);
insert into t values (0,0), (2,3), (0,1);
```

Par la suite, nous utiliserons les abréviations :

- `select T` pour `select * from T` ;
- `select T(A = 1)` pour `select * from T where A=1` ;
- `select A(B = 1)` pour `select A from T where B=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'écrit pas les ; après `begin`, `commit` et `rollback`, n'oubliez pas de les ajouter.

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 ?

Situation A

Question 1.2 :

session 1	session 2
begin	
select A (B=3)	begin
select A (B=3)	update(A <- A+1)
	rollback

Quelle valeur a lu la session 1 ? Quelle propriété est mise en évidence ?

Situation B

Question 1.3 :

session 1	session 2
begin	
select A (B=3)	begin
select A (B=3)	update(A <- A+1)
select A (B=3)	commit

Quelle valeur a lu la session 1 ? Quel problème est mis en évidence ?

Situation C

Question 1.4 :

session 1	session 2
begin	
select T (A = 3)	begin
select T (A = 3)	insert(3,4)
select T (A = 3)	commit

Quelle valeur a lu la session 1 ? Quel problème est mis en évidence ?

Situation D

Question 1.5 :

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 sérialisable, c'est à dire équivalent à une exécution en série ?

Phénomènes indésirables lors des accès concurrents

La documentation de Postgres détaille 4 phénomènes qui peuvent se produire lors d'accès concurrents à une base de données. En voici un extrait : The phenomena which are prohibited at various levels are :

- **dirty read** A transaction reads data written by a concurrent uncommitted transaction.
- **nonrepeatable read** A transaction re-reads data it has previously read and finds that data has been modified by another transaction (that committed since the initial read).
- **phantom read** A transaction re-executes a query returning a set of rows that satisfy a search condition and finds that the set of rows satisfying the condition has changed due to another recently-committed transaction.
- **serialization anomaly** The result of successfully committing a group of transactions is inconsistent with all possible orderings of running those transactions one at a time.

Question 1.6 : Détaillez pour les 4 situations A, B, C et D précédentes, quel phénomène elles mettent en évidence.

Dans la suite, nous allons élaborer un tableau récapitulatif des phénomènes qui peuvent se produire, et des méthodes de protection proposées par Postgres. Pour chacune des combinaisons de phénomène avec une méthode de protection (par exemple **Dirty Read + Read Committed**), vous indiquerez dans la case correspondante du tableau si la situation est possible, ou impossible. Par exemple, **Dirty Read** est impossible en **Read Committed**.

Question 1.7 : Complétez le tableau pour les 4 situations présentées précédemment, avec le mode **Read Committed**. A la fin du TP, vous pourrez vérifier votre tableau sur Moodle.

TABLE 1 – Phénomènes indésirables et efficacité des méthodes de protection

Méthode de protection	Phénomène indésirable			
	Dirty Read	Non Repeatable Read	Phantom Read	Serialization Anomaly
Read committed	Impossible	?	?	?
...				
...				

Méthodes de protection

Verrou implicite

Pour cette exercice, la table T contient les valeurs (2,5) ; (5,3) ; (3,7) et (2,0) :

```
begin;
truncate t;
insert into t values (2,5), (5,3), (3,7), (2,0);
commit;
```

Question 1.8 :

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 ?

Verrou Select for update

Question 1.9 : 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 ?

Question 1.10 : Proposer 4 expériences pour vérifier si **select for update** autorise ou non les 4 situations illustrées au début (**Dirty Read**, **Non Repeatable Read**, **Phantom Read**, **Serialization Anomaly**). Complétez le tableau en fonction de vos résultats.

TABLE 2 – Phénomènes indésirables et efficacité des méthodes de protection

Méthode de protection	Phénomène indésirable			
	Dirty Read	Non Repeatable Read	Phantom Read	Serialization Anomaly
Read committed	Impossible	?	?	?
Select for update	?	?	?	?
...				

Isolation level serializable

Avant cet exercice il est nécessaire de vider la table T et d'y insérer les valeurs (3,7); (5,2); (3,5) et (3,0).

```
begin;
truncate t;
insert into t values (3,7), (5,2), (3,5), (3,0);
commit;
```

Question 1.11 :

session 1	session 2

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

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

Question 1.12 :

session 1	session 2
begin	begin
	set transaction isolation level serializable ;
select T	
update(A<-A-1)	
	select T
	insert (2,2)
	delete(B=7)
select T	
rollback	
	select T
	commit

Comparez avec l'ordonnancement de la question précédente.

Question 1.13 : Proposer 4 expériences pour vérifier si **serializable** autorise ou non les 4 situations illustrées au début (Dirty Read, Non Repeatable Read, Phantom Read, Serialization Anomaly). Complétez le tableau en fonction de vos résultats. Vérifiez vos résultats sur Moodle.

TABLE 3 – Phénomènes indésirables et efficacité des méthodes de protection

Méthode de protection	Phénomène indésirable			
	Dirty Read	Non Repeatable Read	Phantom Read	Serialization Anomaly
Read committed	Impossible	?	?	?
Select for update	?	?	?	?
Serializable	?	?	?	?

Report du contrôle des contraintes

Exercice 2 : Une ESN (Entreprise de Services du Numérique) prête des ordinateurs à ses salariés : chaque salarié a un ordinateur, et chaque ordinateur est associé à un et un seul salarié. Elle souhaite tracer tout cela dans une base de données et propose les tables suivantes :

```
CREATE TABLE programmeur (
  pro_id INTEGER CONSTRAINT programmeur_pk PRIMARY KEY,
  pro_nom VARCHAR(150) NOT NULL,
  pro_prenom VARCHAR(150),
  ord_id INTEGER NOT NULL
);
CREATE TABLE ordinateur (
  ord_id INTEGER CONSTRAINT ordinateur_pk PRIMARY KEY,
  ord_numero_serie VARCHAR(150) NOT NULL,
  pro_id INTEGER NOT NULL
);
```

```
ALTER TABLE programmeur ADD CONSTRAINT programmeur_fk
FOREIGN KEY (ord_id) REFERENCES ordinateur;
ALTER TABLE ordinateur ADD CONSTRAINT ordinateur_fk
FOREIGN KEY (pro_id) REFERENCES programmeur;
```

Question 2.1 : Insérez **Gilfoyle Bertram** et son ordinateur **ABX0123459** dans la base de données. Que se passe-t-il ? Expliquez pourquoi il n'est pas possible de le faire.

Question 2.2 : 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 Programmeur
alter constraint programmeur_fk deferrable initially deferred;
```

2. Modifier le programme de la question précédente pour pouvoir effectuer l'insertion. Supprimer et ajouter une contrainte se fait avec un ALTER TABLE.