Bases de Données Relationnelles

TP4 : Concurrence

MAM4 - SI3

Créez une base de données vide TP4.

1 Un petit test

1. Créez une table test avec un seul attribut de type entier sans clé primaire.

```
drop table if exists test;
create table test(a integer);
```

2. Commencez une transaction, insérez les deux valeurs 1 et 2 dans la table test, validez la transaction et vérifiez que les valeurs sont bien dans la table.

```
begin;
insert into test (a) values (1);
insert into test (a) values (2);
commit;
select * from test;
```

3. Commencez une transaction, insérez les deux valeurs 3 et 4 dans la table, et regardez le contenu de la table: vous devez voir les valeurs 1,2,3 et 4. Annulez la transaction, réaffichez la table: vous ne devez plus voir que les valeurs 1 et 2.

```
begin;
insert into test (a) values (3);
insert into test (a) values (4);
select * from test;
rollback;
select * from test;
```

2 Un test de crash

1. Commencez une transaction, insérez les valeurs 9 et 10 dans la table test, affichez le contenu de la table test. Voyez-vous les valeurs 9 et 10 ?

```
begin;
insert into test (a) values (9);
insert into test (a) values (10);

select * from test;
```

Les valeurs 9 et 10 sont bien affichées.

2. Simulez un crash du serveur en fermant le terminal ou la fenêtre de l'explorateur. Reconnectez vous et affichez le contenu de la table test. Résultat de l'experience?

La transaction ayant été interrompue, le système est à l'état au début de la transaction, les valeurs 9 et 10 ne sont plus affichées.

---

3 Simulation de concurrence

1. Ouvrez deux connections avec deux utilisateurs différents sur la base de données TP4. Créez deux rôles user1 et user2 à qui vous donnerez des droits suffisants pour pouvoir faire des INSERT et des SELECT sur la table test.

```
CREATE ROLE user1;
CREATE ROLE user2;
GRANT INSERT,SELECT,UPDATE ON TABLE public.test TO user1, user2;
```

Tester aussi:

```
CREATE ROLE user3;
GRANT user1 TO user3;
```

Tester la commande `SELECT` avant et après la commande `GRANT` suivie du changement de rôle:

```
CREATE ROLE user4;
SET ROLE user4;
SELECT * FROM test; -- ne sera pas possible
RESET ROLE;
GRANT SELECT ON TABLE test TO user4;
SET ROLE user4;
SELECT * FROM test; -- devient possible
```

---

2. En tant que user1, insérez la valeur 100 dans la table test puis affichez le contenu de la table.

   Dans un autre terminal ou dans un autre éditeur de requêtes prenez le rôle de user2 et affichez le contenu de la même table test.

   user2 voit-il la même chose que user1? Pourquoi?

```
SET ROLE user1;
INSERT INTO test (a) VALUES (100);
SELECT * FROM test;
SET ROLE user2;
SELECT * FROM test;
```

user2 voit la même chose que user1 car par défaut on est en mode autocommit, et donc l'insertion est considérée comme une transaction à part entière.

---

3. Desactivez le mode autocommit.

   En tant que user1 insérez la valeur 200 dans la table test et affichez le contenu de la table.

   En tant que user2 affichez le contenu de la même table test.

   user2 voit-il la même chose que user1? Pourquoi?

Dans une fenêtre:

```
SET ROLE user1; -- puis d sactiver le mode autocommit
INSERT INTO test (a) VALUES (200);
SELECT * FROM test;
```

Dans une autre fenêtre:

```
SET ROLE user2; -- puis d sactiver le mode autocommit
SELECT * FROM test;
```

Non, user2 ne voit pas la dernière insertion de user1. On n'est plus en mode autocommit, l'insertion de user1 n'est pas considérée comme une transaction à part entière, elle est invisible pour user2 tant que la transaction de user1 n'est pas terminée par un `COMMIT`.

---

4. Revenez en mode autocommit.

   user1 commence une transaction, puis insère la valeur 1000 dans la table test. Il affiche le contenu de la table.

   user2 affiche le contenu de la même table. Voit-il la même chose que user1?

Dans la fenêtre de user1:

```
BEGIN;
INSERT INTO test (a) VALUES (1000);
SELECT * FROM test;
```

Dans la fenêtre de user2:

```
SELECT * FROM test;
```

user2 ne voit pas la dernière insertion de user1. Tant que la transaction de user1 n'est pas terminée par un `COMMIT;`, du point de vue de user2 elle n'a pas commencé.

---

5. user1 termine la transaction, puis il affiche le contenu de la table.

   user2 affiche le contenu de la même table. Voit-il la même chose que user1?

Oui.

---

6. Executez en respectant l'ordre:

   (a) user2 commence une transaction.
   (b) user1 commence une transaction.
   (c) user1 demande une mise à jour de la table test en multipliant les valeurs par 2.
   (d) user2 demande une mise à jour de la table test en ajoutant 1 aux valeurs.

   A l'affichage de la table quelles valeurs voient les deux utilisateurs ?

Dans deux fenêtre différentes :

```
user2 : BEGIN;
user1 : BEGIN;
user1 : UPDATE test SET a = 2*a;
user2 : UPDATE test SET a = a+1
user1 : SELECT * FROM test;
user2 : SELECT * FROM test;
```

user1 voit la table telle qu'il l'a mise à jour: des valeurs paires uniquement.

user2 ne voit rien du tout; il est en attente de l'exécution de son update car user1, avec son update, a posé un verrou sur la table test.

---

7. user1 termine sa transaction (`COMMIT;`). A l'affichage de la table quelles valeurs voient les deux utilisateurs?

---

user1 voit les mêmes valeurs que précédemment, résultat de son update (des valeurs paires).

La fin de la transaction de user1 lève le verrou de celui-ci sur la table test. L'update de user2 peut s'exécuter. user2 voit la table mise à jour avec son update, à partir de l'état de la table après update par user1: des valeurs impaires uniquement.

---

8. user2 termine sa transaction. A l'affichage de la table quelles valeurs voient les deux utilisateurs?

---

user2 voit la même chose que précédemment et user1 voit la même chose que user2 car à la fin de la transaction de ce dernier, son verrou sur la table test a été levé et le résultat de son update est visible pour tous les utilisateurs de la base.
