Rendu final TP1 & TP2

BENJELLOUN Youssef

I54 - TP1 TP Transactions

Partie 1

A. Transaction unique

Comprendre commit et rollback

1. Créez la table `Departement(DID,Libelle)` où la clé primaire DID est un code à deux lettres, Libelle une chaîne de 20 caractères maximum. Créez la table `Employe(EID,Nom,Dept)` où la clé primaire EID est un nombre à deux chiffres, Nom une chaîne de 20 caractères maximum et Dept un code à deux lettres. Dept est clé étrangère référençant Departement.

```
CREATE TABLE Departement(
DID varchar(2) PRIMARY KEY,
Libelle varchar(20)
);
```

Pour la table Employe je trouve qu'il y a deux solutions possibles

```
CREATE TABLE Employe (
  EID integer primary key CHECK (eid > 9 and eid < 100),
  Nom varchar(20),
  Dept varchar(2),
  CONSTRAINT employe_departement_fkey FOREIGN KEY (Dept) REFERENCES
Departement(did));
```

ou

```
CREATE TABLE Employe (
  EID numeric(2) primary key,
  Nom varchar(20),
  Dept varchar(2),
  CONSTRAINT employe_departement_fkey FOREIGN KEY (Dept) REFERENCES
Departement(did)
);
```

La première a comme contrainte pas prendre les id du type `0X` et la deuxième a comme contrainte d'accepter les nombres a un chiffre

```
INSERT INTO departement VALUES ("mt", math);
rollback;
```

Il faut commencer un transaction par défaut on est en mode "autocommit".

3. Débutez une transaction (begin transaction). Créez un département. Consultez la table Département. Effectuez un rollback. Consultez la table à nouveau. Que se passe – il ?

```
begin transaction;
INSERT INTO departement VALUES ("mt", math);
SELECT * FROM departement;
rollback;
```

Avant de faire le rollback si on consulte la table Departement le département est bien inséré par contre lorsque on exécute le rollback et on consulte la table Departement la valeur disparaît. C'est à dire que la insertion n'a pas été enregistré.

4. Débutez une transaction. Créez un département. Consultez la table Département. Effectuez un Commit. Consultez la table à nouveau. Que se passe – il ?

```
begin transaction;
INSERT INTO departement VALUES ("mt", math);
select * from departement;
commit;
select * from departement;
```

Cette fois les changements on été effectifs et le departement a bien été inséré.

5. Débutez une transaction. Créez un département. Consultez la table Département. Effectuez un Commit. Consultez la table à nouveau. Effectuez un rollback. Consultez la table à nouveau. Que se passe – il ?

```
begin transaction;
INSERT INTO departement VALUES ("mt", math);
select * from departement;
commit;
select * from departement;
rollback;
```

Les changements on été effectifs et le departement a bien été inséré. Le rollback échoue car il n'y a pas des transaction en cours.

6. Débutez une transaction. Insérez un nouveau département et quitter votre session. Ouvrez une nouvelle session. La dernière Transaction a-t-elle été validée ?

```
begin transaction;
INSERT INTO departement VALUES ("mt", math);
quit
```

Non la transaction n'a pas été validé

7. Insérez un nouveau département et quitter votre session. Ouvrez une nouvelle session. La dernière Transaction a-t-elle été validée ?

```
session1> begin transaction;
session1> INSERT INTO departement VALUES ("mt", math);
session1> quit
session2> SELECT * FROM departement;
```

Oui la transaction a bien été validé car on est en mode "autocommit" par défaut.

8. Commencez une transaction. Vider le contenu de la table Departement et effectuez un rollback. Consultez la table.

```
begin transaction;
truncate table departement CASCADE;
rollback;
select * from departement;
```

Le `rollback` empêche que les changements soit effectifs, donc la table reste inchangé

9. Commencez une transaction. Vider le contenu de la table Departement et effectuez un Commit. Consultez la table. Comprendre les mises à jour et la transaction (vider les 2 tables)

```
begin transaction;
truncate table departement CASCADE;
commit;
select * from departement;
```

Du fait que la table Departement est référencé dans la table Employe on doit utiliser `CASCADE` alors cela vide toutes les n-uplets de Departement et tout les n-uplets de Employé en cascade

10. Créez un département <'C1','Info'>. Créez un employé <01,'Quentin,'C1'> appartenant au département C1. 11)

```
insert into departement values ('C1', 'Info');
insert into employe values (01, 'Quentin', 'C1');
```

11. Créez un département <'C1','Informatique'>. Que se passe-t-il ?

```
insert into departement values ('C1', 'Informatique')
```

Cette insertion échoue car elle ne respecte pas la contrainte de unicité de la clé primaire .

12) Créez un employé <02,'Grégoire','C2'>. Que se passe-t-il ?

```
insert into employe values (02, 'Gregoire', 'C2');
```

L'insertion échoue car la contrainte de la clé étranger `dept` n'a été respecté i.e `C2` n'existe pas.

13) Supprimez le département C1. Que se passe-t-il ?

```
delete from departement where did='C1';
```

La suppression échoue car `C1` est référencé ailleurs plus précisément dans la table `Employe`.

14) Modifier la table Employé en supprimant la clé étrangère (ALTER TABLE ....DROP CONSTRAINT ...). Supprimez le département C1. Interrogez les deux tables. Expliquez les résultats.

```
ALTER TABLE employe DROP CONSTRAINT employe_departement_fkey;
delete from departement where did='C1';
select * from departement;
select * from employe;
```

Lorsque on effectué cela on a un état de nos table est absurde plus précisément il existe un employé qui travaille pour un département qui n'existe pas

15) Vider les deux tables.

```
truncate table departement;
truncate table employe;
```

16) Modifier la table Employé : Recréez la clé étrangère en mode ON DELETE CASCADE (ALTER TABLE ....ADD CONSTRAINT ...). Recréer le département C1 et l'employé 01.

```
ALTER TABLE employe ADD CONSTRAINT employe_departement_fkey FOREIGN KEY (dept)
REFERENCES departement(did) ON DELETE CASCADE;
insert into departement values ('C1', 'Info');
insert into employe values (01, 'Quentin', 'C1');
```

17) Supprimez le département C1. Interrogez les deux tables. Expliquez les résultats.

```
delete from departement where did='C1';
select * from departement;
select * from employe;
```

En considérant que la contrainte de clef étrangère est en mode "on delete cascade" maintenant les suppressions du departement `C1` efface aussi l'employé `01`

18) Recréer le département C1 et l'employé 01.

```
insert into departement values ('C1', 'Info');
insert into employe values (01, 'Quentin', 'C1');
```

19. Supprimez à nouveau la clé étrangère. Recréez la clé étrangère sans le mode ON DELETE CASCADE mais avec le mode DEFERRABLE.

```
alter table employe drop constraint employe_departement_fkey;
ALTER TABLE employe ADD CONSTRAINT employe_departement_fkey FOREIGN KEY (dept)
REFERENCES departement(did) DEFERRABLE;
```

20. Activer le mode DEFERRED avec la commande `SET CONSTRAINTS <nom_de_votre_contrainte> DEFERRED;`

```
set constraints employe_departement_fkey deferred ;
-- WARNING:  SET CONSTRAINTS can only be used in transaction blocks
-- SET CONSTRAINTS
```

Cela produit un avertissement, la commande `SET CONSTRAINTS` ne peut être utilisé que dans les blocs transaction.

21) Supprimez le département C1. Expliquez le résultat.

```
begin transaction;
set  constraints employe_departement_fkey deferred;
delete  from departement where did='C1';
commit;
```

Lors de la suppression, aucun message d'erreur apparaît comme auparavant, c'est que lors du `commit` que le message apparaît.

B. Contrôle de la Concurrence

1. Connectez-vous à votre base à partir de deux sessions différentes, simultanément.

`Pour préparer la partie B, supprimer la clé étrangère pour la recréer sans mode DEFFERABLE ni On DELETE CASCADE. Videz la table Employé, puis la table Departement.`

```
alter table employe drop constraint employe_departement_fkey;
ALTER TABLE employe ADD CONSTRAINT employe_departement_fkey FOREIGN KEY (dept)
REFERENCES departement(did);
TRUNCATE TABLE departement CASCADE;
```

2. Dans chacune des deux fenêtres, insérez un département (code C1 pour transaction 1, code C2 pour transaction 2). Consulter la table depuis chaque fenêtre. Que constate-t-on ?

```
session1> insert into departement values ('C1', 'informatique');
session2> insert into department values ('C2', 'math');
```

Les deux valeurs s'insèrent correctement car on est en mode "autocommit" par défaut.

3. Dans chacune des deux fenêtres, commencez une transaction.

1. Insérez un département dans la session 1. Consultez la table. N'effectuez pas encore de commit.
2. Insérez un autre dpt dans la session2. Commitez dans la session2.
3. Consulter la table dans les 2 sessions. Que constate-t-on ?
4. Commitez dans la session 1.
5. Consulter la table dans les 2 sessions. Que constate-t-on ?

```
session1> BEGIN TRANSACTION;
session2> BEGIN TRANSACTION;
session1> insert into departement values ('C3', 'bio');
session1> select * from departement;
-- la valeur s'inséré correctement
```

```
session2> insert into department values ('C4', 'svt');
session2> commit;
session2> select * from departement;
-- la valeur inséré par la session2 est bien inséré mais pas celle de la session1
session1> select * from departement;
-- ici les deux insertion sont afficher car on est toujous dans le bloc de
transaction
session1> commit;
session2/session1> select * from departement;
-- les deux insertions sont effectifs et s'affichent sur les deux sessions.
```

Lors qu'on fait un commit sur une session les tables se mettent à jour même dans des transaction ouverts en parallèles

4. Dans chacune des deux fenêtres, commencez une transaction.

1. Insérez un département dans la session 1. N'effectuez pas encore de commit.
2. Insérez le même département dans la session 2. Que constatez vous ?
3. Commitez dans la session 1. Que constatez-vous dans la session 2 ?
4. Terminez la session 2.
5. Consulter la table dans les 2 sessions. Que constate-t-on ?

```
session1/session2> BEGIN TRANSACTION;
session1> insert into departement values ('C5', 'algo');
session2> insert into department values ('C5', 'algo');
-- la session 2 est en attente
session1> commit;
-- la session 2 n'est plus en attente et échoue lors de l'insertion
```

La session 2 n'est plus en attente et renvoie une erreur car la contrainte d'unicité de la cle primaire est violé. L'insertion de la session 1 réussi sans problèmes.

5. Dans chacune des deux fenêtres, commencez une transaction.

1. Insérez un département dans la session 1. N'effectuez pas encore de commit.
2. Insérez le même département dans la session 2. Que constatez vous ?
3. Annulez l'insertion dans la session 1. Que constatez-vous dans la session 2 ?
4. Terminez la session
5. Consulter la table dans les 2 sessions. Que constate-t-on ?

```
session1/session2> BEGIN TRANSACTION;
session1> insert into departement values ('C6', 'poo');
session2> insert into department values ('C6', 'poo');
-- La session 2 se met en attente
session1> rollback;
-- La session 2 n'est plus en attente et réussi a insérer la valeur
```

```
session2> quit;
session1/session2> select * from departement;
```

Du fait que on termine la session 2 et on effectue pas des commits la insertion n'est pas effectif.

6. Dans l'un des 2 sessions, vider le contenu de la table Departement (c'est un autocommit).

```
truncate table departement on cascade;
```

7. Dans la session 1, commencez une transaction et insérez le département <'C1','Info'>. Effectuer un Commit. Consultez la table Departement.

```
session1> BEGIN TRANSACTION;
session1> insert into departement values ('C1', 'info');
session1> commit;
session1> select * from departement;
-- la valeur a été inséré avec succès
```

8. Depuis la session 2, commencez une transaction et modifiez le libellé de ce département en 'Informatique' et validez

```
session2> begin transaction;
session2> update departement set libelle = 'Info' where did='C1';
session2> commit;
```

9. Depuis la session 1, consultez la table Departement à nouveau.

```
select * from departement
```

Le libellé a bien été changé même dans la session1.

10. Depuis la session 1 commencez une transaction. modifiez le libellé de ce département en 'Info' sans validez encore ; faites la même chose dans la session 2, mais pour modifier ce libelle en 'Biologie'. Que se passe-t-il ?

```
--terminal 1
session1> begin transaction;
session1> update departement set libelle='informatique' where did='C1';
session2> begin transaction;
session2> update departement set libelle='biologie' where did='C1';
```

La session 2 se met en attente.

11. Annulez la modification dans la première session. Consultez les tables dans les deux sessions. Que se produit-il?

```
session1> rollback;
session1> select * from departement;
-- Dans le session 1 rien n'a change car on a pas encore effectué un commit dans
la session 2,
session2> select * from departement;
-- Le libellé a bien été changé car on est toujours dans le bloc de transaction
```

12. Validez dans la première session. Consultez les tables dans les deux sessions. Que se produit-il?

```
--terminal 1
session1> commit;
-- la session 2 n'est plus en attente et réussi a changer aussi le libellé
```

Les update c'est font a la suite et donc si on commit la session 2 sera cette libellé qui perdurera.

13. Dans l'un des 2 sessions, vider le contenu de la table Departement (c'est un autocommit).

```
truncate table departement cascade;
```

14. Depuis la session 1, insérez le département <'C1','Info'> et depuis la session 2 Insérez un employé <01,'Henri','C1'>. Que se passe-t-il ?

```
session1/2> begin transaction;
session1> insert into departement values ('C1', 'info');
session2> insert into employe values (01, 'Henri', 'C1');
```

Si on commit pas l'insertion de la session 1 on obtient une erreur car l'insertion de la session2 viole la contrainte de clé étrangère.

15. Annulez l'insertion dans la session 1. Que se passe-t-il dans la session 2?

```
session1/2> begin transaction;
session1> insert into departement values ('C1', 'info');
session1> rollback;
session2> insert into employe values (01, 'Henri', 'C1');
```

La insertion de la session 2 échoue car elle viole la contrainte de clé étrangère.

16. Depuis la session 1, insérez le département <'C1','Info'> puis depuis la session 2 Insérez un employé <01,'Henri','C1'>. Consultez les deux tables depuis les deux sessions.

```
--terminal 1
session1/2> begin transaction;
session1> insert into departement values ('C1', 'info');
session1>commit;
session2>insert into employe values (01, 'Henri', 'C1');
session2>commit;
```

Si on commit la session 1 avant la session 2 les deux insertions réussis.

17. Depuis la session 1 supprimez le département C1. Que se passe-t-il ?

```
session1> delete from departement where did='C1';
```

On obtient une erreur car `C1` est encore referencé en employé.

18. Ajoutez depuis la session 1 l'employé <02,'Grégoire,'C1'>. Validez la transaction.

```
begin transaction;
insert into employe values (02, 'gregoire', 'C1');
commit;
```

19. Depuis la session 1, modifiez le nom de l'employé 01 en Coleen, ne validez pas encore. Depuis la fenêtre 2, modifiez le nom de l'employé 02 en Camille, ne validez pas encore. Consultez les tables. Que se passe-t-il ?

```
--terminal 1
session1/2> begin transaction;
session1> update employe set nom='Coleen' where eid=1;
session2> update employe set nom='Camille' where eid=2;
session1/2> select * from employe;
```

Dans chaque bloc de transaction les changements on été effectifs mais les changements de l'autre session n'en pas été effectifs.

20. Validez la session1, consultez la table. Validez la session 2, consultez la table. Consultez à nouveau la table dans la session 1.

```
session1>  commit;
session1> select * from employe;
-- Le nom a bien été changé
session2> commit;
session1/2> select * from employe;
-- Maintenant les deux noms on bien été change et son visibles dans les deux
sessions.
```

Transactions

Le but est de réserver des places : Quentin réserve 2 places pour Harry Potter, et Grégoire en réservé 5. On a donc deux exécutions de la procédure de réservation, l'une à dérouler dans la session 1, l'autre dans la session 2. On se place en mode sérialisable.

1. Commencez par écrire les requêtes effectuées par chaque transaction : a) La réservation de Quentin : T1 b) La réservation de Grégoire : T2 Pensez à vérifier la cohérence (1) à réaliser dans l'une et l'autre session (dans T1 et T2). T1 :

```
begin transaction;
set session transaction isolation level serializable;
insert into reservation values ('Quentin', 'Happy Potter', 2);
update spectacle set nbplaces = nbplaces -2 where noms='Happy Potter';
update client set solde = solde - 2*20 where client='Quentin';
commit
```

T2 :

```
begin transaction;
set session transaction isolation level serializable;
insert into reservation values ('Gregoire', 'Happy Potter', 5);
update spectacle set nbplaces = nbplaces -5 where noms='Happy Potter';
update client set solde = solde - 2*20 where client='Gregoire';
commit;
```

2. Effectuez les deux transactions l'une après l'autre (en série). Quel est l'état de la base à la fin?

```
--session 1
begin transaction;
set session transaction isolation level serializable;
insert into reservation values ('Quentin', 'Happy Potter', 2);
update spectacle set nbplaces = nbplaces -2 where noms='Happy Potter';
commit;
tp2db=> select * from spectacle ;
     noms     | nbplaces | nbplaceslibres | tarif
```

```
-------------+----------+----------------+-------
 Happy Potter |      248 |            250 | 20.00
(1 row)
--session2
begin transaction;
set session transaction isolation level serializable;
insert into reservation values ('Gregoire', 'Happy Potter', 5);
update spectacle set nbplaces = nbplaces -5 where noms='Happy Potter';
commit;
select * from reservation ;
   nomc    |     noms     | nbplacesreservees
----------+--------------+-------------------
 Quentin  | Happy Potter |                 2
 Gregoire | Happy Potter |                 5
(2 rows)

tp2db=> select * from spectacle;
     noms     | nbplaces | nbplaceslibres | tarif
--------------+----------+----------------+-------
 Happy Potter |      243 |            250 | 20.00
(1 row)
```

Tout se passe bien

3. Réinitialiser vos tables.

```
truncate table reservation ;
truncate table client cascade;
insert into client values ('Quentin', 50);
insert into client values ('Gregoire', 50);
select * from client;
   nomc    | solde
----------+-------
 Quentin  |    50
 Gregoire |    50
(2 rows)
update spectacle set nbplaces = 250 where noms = 'Happy Potter';
select * from spectacle;
     noms     | nbplaces | nbplaceslibres | tarif
--------------+----------+----------------+-------
 Happy Potter |      250 |            250 | 20.00
(1 row)
```

4. Testez par la suite, une histoire (une exécution, un entrelacement) ou plus d'une, sous forme de suite d'opérations grâce aux transactions que vous venez de produire. Analysez les situations.

```
session1/2> begin transaction ;
session1/2> set session transaction isolation level serializable;
session1> insert into reservation values ('Quentin', 'Happy Potter', 2);
```

```
INSERT 0 1
session2> insert into reservation values ('Gregoire', 'Happy Potter', 5);
INSERT 0 1
session1> update spectacle set nbplaces = nbplaces -2 where noms='Happy Potter';
UPDATE 1
session2> update spectacle set nbplaces=nbplaces-5 where noms='Happy Potter';
-- en attente
session1> commit;
session2>
ERROR:  could not serialize access due to concurrent update
```

La transaction échoue car cette entrelacement n'est pas serializable car lorsque on commit la session 1 le état des tables changent et en mode serializable il ne peut pas y avoir des lectures non répétables

5. Choisissez ensuite une des commandes ci-dessous. set session transaction isolation level read uncommitted; set session transaction isolation level read committed; set session transaction isolation level repeatable read;

set session transaction isolation level read uncommitted;

```
session1/2> begin transaction;
BEGIN
session1/2> set session transaction isolation level read uncommitted;
SET
session1> insert into reservation values ('Quentin', 'Happy Potter', 2);
INSERT 0 1
session1>  update spectacle set nbplaces= 250-2 where noms= 'Happy Potter';
UPDATE 1
session2> select * from spectacle ;
     noms     | nbplaces | nbplaceslibres | tarif
--------------+----------+----------------+-------
 Happy Potter |      250 |            250 | 20.00
(1 row)
session1>  commit;
COMMIT
```

Postgres a une politique anti read uncommitted donc on a les même résultats qu'avec le mode read committed

set session transaction isolation level read committed;

```
session1/2> begin transaction;
session1/2> set session transaction isolation level read committed;
session1> select * from reservation;
nomc | noms | nbplacesreservees
------+------+-------------------
(0 rows)
session2> insert into reservation values ('Quentin', 'Happy Potter', 2);
```

```
session2> commit;
session1> select * from reservation;
select * from reservation ;
nomc    |     noms     | nbplacesreservees
----------+--------------+-------------------
Gregoire | Happy Potter |                 5
(1 row)
```

Comme on peut constater avec le mode read committed on des lectures non répétables.

set session transaction isolation level repeatable read ;

```
session1/2> begin transaction ;
BEGIN
session1/2> set session transaction isolation level repeatable read;
SET

session1/2> select * from reservation ;
 nomc | noms | nbplacesreservees
------+------+-------------------
(0 rows)
session2=*> insert into reservation values ('Quentin', 'Harry Potter', 5);
INSERT 0 1
session2=*> select * from reservation ;
   nomc    |     noms     | nbplacesreservees
----------+--------------+-------------------
 Gregoire | Harry Potter |                 5
(1 row)
session2=*> commit;
COMMIT
session1=*> select * from reservation ;
  nomc    |     noms     | nbplacesreservees
---------+--------------+-------------------
(0 row)
session1=*> insert into reservation values ('Quentin', 'Harry Potter', 2);
INSERT 0 1
session1=*> select * from reservation ;
   nomc    |     noms     | nbplacesreservees
----------+--------------+-------------------
 Quentin  | Harry Potter |                 2
(1 row)
session2=*> commit;
COMMIT
session1/2=> select * from reservation ;
   nomc    |     noms     | nbplacesreservees
----------+--------------+-------------------
 Gregoire | Harry Potter |                 5
 Quentin  | Harry Potter |                 2
(2 rows)
```

Avec le mode repeatable read on ne peut pas avoir des lectures non répétables mais on peut avoir quand meme des lectures fantômes

Partie 3

Le niveau d'isolation que je choisirai pour l'application précédent serait le mode read commited. Un soucis qui pourrait survenir avec ce mode serait le cas où, il reste une seul place pour le spectacle et, que deux personnes simultanément commencent une reservation. Alors seulement le plus rapide entre eux pourra avoir sa place mais ce n'est pas un soucis car on peut annuler la reservation du plus lent a posteriori.