TD N° 5

Correction

Gestion d'une chaîne d'hôtels (suite)

CONCURRENCE ET CONTRAINTE D'INCLUSION

Soit la contrainte d'inclusion :

(C1) Il est impossible de passer une réservation pour un type de chambre dans un hôtel qui ne possède aucune chambre de ce type.

Qui est implantée par les triggers suivants :

```
create or replace trigger C1_Ajout_Reservations
after insert or update of NumTy, NumHo on Reservations
for each row
declare
   N binary_integer;
begin
   select 1 into N from Chambres
   where NumHo = :New.NumHo and NumTy = :New.NumTy;
exception
   when too_many_rows then null;
   when no_data_found then
      raise_application_error(-20000,'PB C1');
end;

create or replace trigger C1_Supp_Chambres
after delete or update of NumHo, NumTy on Chambres
declare
   N binary_integer;
begin
   select 1 into N from Reservations
   where (NumHo, NumTy) not in
      (select NumHo, NumTy from Chambres);
   raise too_many_rows;
exception
   when no_data_found then null;
   when too_many_rows then
      raise_application_error(-20000,'PB C1');
end;
```

1) Analyser l'interférence liée à plusieurs transactions qui s'exécutent simultanément.

Etat initial :

Chambres :

NumCh	NumHo	NumTy
101	1	10
102	1	20
103	1	20
104	1	30
105	1	40
106	1	40

Reservations :

NumCl	NumHo	NumTy	DateA	NbChambres	NbJours
1000	1	20	d1	1	i1
2000	1	40	d2	1	i2

T1	T2
insert into Reservations values(3000,1,10,d3,1,i3); commit;	delete Chambres where NumCh=101 and NumHo=1; commit;
T3	**T4**
update Reservations set NumTy = 30 where NumCl=1000 and NumHo=1 and NumTy=20 and DateA=d1; commit;	update Chambres set NumTy = 20 where NumCh=104 and NumHo=1 ; commit;
T5	**T6**
delete Chambres where NumCh=105 and NumHo=1; commit;	delete Chambres where NumCh=106 and NumHo=1 ; commit;

Résultat de l'exécution des transactions :

Chambres :

NumCh	NumHo	NumTy
102	1	20
103	1	20
104	1	20

Réservations :

NumCl	NumHo	NumTy	DateA	NbChambres	NbJours
1000	1	**30**	d1	1	i1
2000	1	**40**	d2	1	i2
3000	1	**10**	d3	1	i3

La contrainte est donc violée : les réservations portent sur des types de chambre qui n'existe plus dans l'hôtel

2) Proposer une gestion de la concurrence avec des verrous de table. Et faire une évaluation de degré de blocage des transactions (i.e. du degré de concurrence).

<u>Remarque</u> : Une gestion avec une isolation par snapshot ne corrige pas le problème

Etat initial :

Chambres :

NumCh	NumHo	NumTy
101	1	10
102	1	20
103	1	20
104	1	30

Réservations :

NumCl	NumHo	NumTy	DateA	NbChambres	NbJours
1000	1	20	d1	1	i1

T1	T2
set transaction isolation level serializable;	
insert into Reservations values(2000,1,10,d2,1,i2);	
	set transaction isolation level serializable;
	delete Chambres where NumCh=101 and NumHo=1;
	commit;
commit;	

Résultat de l'exécution des transactions :

Chambres :

NumCh	NumHo	NumTy
102	1	20
103	1	20
104	1	30

Réservations :

NumCl	NumHo	NumTy	DateA	NbChambres	NbJours
1000	1	20	d1	1	i1
2000	1	**10**	d2	1	i2

La contrainte est donc violée : la nouvelle réservation porte sur un type de chambre qui n'existe plus dans l'hôtel

Ajout des triggers de pose de verrous :

```
Create or replace trigger C1_Verrou_Reservations
before insert or update of NumTy, NumHo on Reservations
begin
     lock table Chambres in share mode;
end;


Create or replace trigger C1_Verrou_Chambres
before delete or update of NumHo, NumTy on Chambres
begin
     lock table Chambres in share mode;
     lock table Reservations in share mode;
end;
```

Remarque : Il est possible de supprimer le verrou posé par le trigger **C1_Verrou_Chambres** sur la table **Reservations**, car il est redonnant avec celui posé par le trigger **C1_Verrou_Reservations** sur la table **Chambres**. En effet, l'instruction déclenchante (delete ou update) du trigger **C1_Verrou_Chambres** pose systématiquement un verrou **ROW EXCLUSIVE** sur la table **Chambres**, et celui-ci est incompatible avec le verrou **SHARE** posé par le trigger **C1_Verrou_Reservations** sur la table **Chambres**.

Évaluation du degré de blocage des transactions :

1) Transaction qui insère une réservation ou qui modifie le type de chambre ou l'hôtel d'une réservation :

   Blocage des autres transactions en fonction de ce qu'elles font sur la table Chambres :

Select	Exclusive	Share	Row Share Select for update	Row Ex. Insert Delete Update	Share Row Exclusive
Non Bloqué	Bloqué	Non Bloqué	Non Bloqué	Bloqué	Bloqué

2) Transaction qui supprime une chambre ou qui modifie le type ou l'hôtel d'une chambre:

   Blocage des autres transactions en fonction de ce qu'elles font sur la table Reservations :

Select	Exclusive	Share	Row Share Select for update	Row Ex. Insert Delete Update	Share Row Exclusive
Non Bloqué	Bloqué	Non Bloqué	Non Bloqué	Bloqué	Bloqué

   Blocage des autres transactions en fonction de ce qu'elles font sur la table Chambres :

Select	Exclusive	Share	Row Share Select for update	Row Ex. Insert Delete Update	Share Row Exclusive
Non Bloqué	Bloqué	Non Bloqué	Non Bloqué	Bloqué	Bloqué

3) Proposer une gestion de la concurrence avec des verrous de ligne. Et faire une évaluation de degré de blocage des transactions

1) Pose de verrous coté Réservations
Modification du trigger **C1_Ajout_Reservations** :

```
create or replace trigger C1_Ajout_Reservations
after insert or update of NumTy, NumHo on Reservations
for each row
declare
   N binary_integer;
begin
   select 1 into N from Chambres
   where NumHo = :New.NumHo and NumTy = :New.NumTy
   and rownum = 1 for update;
exception
   when too_many_rows then null;
   when no_data_found then
      raise_application_error(-20000,'PB C1');
end;
/
```

Autre solution, qui permet de verrouiller plusieurs lignes :

```
create or replace trigger C1_Ajout_Reservations
after insert or update of NumTy, NumHo on Reservations
for each row
declare
   N binary_integer;
begin
   N := 0;
   for I in (  select 1 from Chambres
               where NumHo = :New.NumHo and NumTy = :New.NumTy
               for update)
   loop
     N := N + 1;
   end loop;
   if N = 0 then raise_application_error(-20000,'PB C1');
   end if;
end;
/
```

2) Pose de verrous coté Chambres, utilisation d'une table temporaire

```
create table Temp_CC1 (NumHo numeric(10), NumTy numeric(10));
```

```
create or replace trigger Remplir_TempCC1
after delete or update of NumHo, NumTy on Chambres
for each row
begin
   insert into Temp_CC1 values(:old.NumHo,:old.NumTy);
end;

create or replace trigger C1_Supp_Chambres
after delete or update of NumHo, NumTy on Chambres
declare
   N binary_integer;
begin
   -- Pose de verrous
   for i in (select 1 from Chambres
             where (NumHo,NumTy) in
                   (select NumHo,NumTy from Temp_CC1)
             for update)
   loop
      null;
   end loop;

   -- Vérification de la contrainte
   select 1 into N from Reservations
   where (NumHo, NumTy) not in
      (select NumHo, NumTy from Chambres);
   raise too_many_rows;
exception
   when no_data_found then delete Temp_CC1;
   when too_many_rows then
      raise_application_error(-20000,'PB C1');
end;
```

Analyser du degré de blocage des transactions (i.e. analyse de la concurrence).

Etat initial :

Chambres :

NumCh	NumHo	NumTy
101	1	10
102	1	20
103	1	20
104	1	30
105	1	40
106	1	40

Réservations :

NumCl	NumHo	NumTy	DateA	NbChambres	NbJours
1000	1	20	d1	1	i1
3000	1	20	d3	1	i3
4000	1	40	d4	1	i4

T1	T2
insert into Reservations values(2000,1,10,d2,1,i2);	delete Chambres where NumCh=101 and NumHo=1; BLOQUEE jusqu'au commit de T1 PUIS ERREUR : violation C1
Suite T1	T3
commit;	delete Chambres where NumCh=102 and NumHo=1; -- NON BLOQUEE commit;
T4	T5
update Reservations set NumTy = 30 where NumCl=1000 and NumHo=1 and NumTy=20 and DateA=d1; BLOQUEE jusqu'au commit de T5 PUIS ERREUR : violation C1	update Chambres set NumTy = 20 where NumCh=104 and NumHo=1 ;
T6	Suite T5
update Reservations set NumTy = 10 where NumCl=3000 and NumHo=1 and NumTy=20 and DateA=d3; -- NON BLOQUEE commit;	commit;
T7	T8
delete Chambres where NumCh=105 and NumHo=1;	delete Chambres where NumCh=106 and NumHo=1 ; BLOQUEE jusqu'au commit de T7 PUIS ERREUR : violation C1
Suite T7	T9
commit;	delete Chambres where NumCh=103 and NumHo=1 ; -- NON BLOQUEE commit; commit;

Résultat :

Chambres :

NumCh	NumHo	NumTy
101	1	10
104	1	20
106	1	40

Réservations :

NumCl	NumHo	NumTy	DateA	NbChambres	NbJours
1000	1	20	d1	1	i1
2000	1	10	d2	1	i2
3000	1	10	d3	1	i3
4000	1	40	d4	1	i4

La contrainte n'est pas violée. De plus certaines instructions ne sont pas bloquées.

Évaluation du degré de blocage des transactions :

1) Transaction qui insère une réservation ou qui modifie le type de chambre ou l'hôtel d'une réservation :

Blocage des autres transactions en fonction de ce qu'elles font sur la table Chambres :

Select	Exclusive	Share	Row Share Select for update	Row Ex. Insert Delete Update	Share Row Exclusive
Non Bloqué	Bloqué	Non Bloqué	Non Bloqué[1]	Non Bloqué[2]	Non Bloqué

(1) Sauf si l'instruction porte sur une ligne déjà verrouillée.
(2) Sauf si l'instruction porte sur une ligne déjà verrouillée. L'insertion n'est jamais bloquée.

2) Transaction qui supprime une chambre ou qui modifie le type ou l'hôtel d'une chambre :

Blocage des autres transactions en fonction de ce qu'elles font sur la table Reservations :

Select	Exclusive	Share	Row Share Select for update	Row Ex. Insert Delete Update	Share Row Exclusive
Non Bloqué	Non Bloqué	Non Bloqué	Non Bloqué[1]	Non Bloqué[2]	Non Bloqué

(1) Sauf si l'instruction porte sur une ligne déjà verrouillée.
(2) Sauf si le trigger déclenche le verrouillage d'une ligne déjà verrouillée dans la table Chambres. La suppression n'est jamais bloquée.

Blocage des autres transactions en fonction de ce qu'elles font sur la table Chambres :

Select	Exclusive	Share	Row Share Select for update	Row Ex. Insert Delete Update	Share Row Exclusive
Non Bloqué	Bloqué	Non Bloqué	Non Bloqué[1]	Non Bloqué[2]	Bloqué

(1) Sauf si l'instruction porte sur une ligne déjà verrouillée.
(2) Sauf si l'instruction porte sur une ligne déjà verrouillée. L'insertion n'est jamais bloquée.

CONCURRENCE ET CONTRAINTE D'UNICITÉ

Soit la contrainte d'unicité :

(C2) Il est impossible qu'une chambre soit occupée 2 fois en même temps par deux clients différents ou par le même client.

Qui est implantée par le trigger suivant :

```
create or replace trigger C2
after insert or update on Occupations
declare
   N binary_integer;
begin
   select 1 into N from Occupations O1, Occupations O2
   where O1.NumCh = O2.NumCh and O1.NumHo = O2.NumHo
   and O1.DateA <> O2.DateA
   and (O1.DateA, O1.DateD) overlaps (O2.DateA, O2.DateD);
exception
   when no_data_found then null;
   when too_many_rows then
     raise_application_error(-20000,'PB C2');
end;
```

1) Analyser l'interférence liée à plusieurs transactions qui s'exécutent simultanément.

T1	T2
insert into OCCUPATIONS values(1000,101,3,timestamp '2019-10-15 12:00:00',timestamp '2019-10-30 12:00:00');	
	insert into OCCUPATIONS values(2000,101,3,timestamp '2019-10-20 12:00:00',timestamp '2019-10-21 12:00:00');
	commit;
commit;	

Résultat :

Occupations :

NumCl	NumCh	NumHo	DateA	DateD
1000	101	3	15/10/19 12h00	30/10/19 12h00
2000	101	3	20/10/19 12h00	21/10/19 12h00

La contrainte est donc violée : la chambre 101 de l'hôtel 3 est occupée par 2 clients entre le 20/10/19 12h00 et le 21/10/19 12h00.

2) Proposer une gestion de la concurrence avec des verrous. Et faire une évaluation de degré de blocage des transactions.

<u>Remarque</u> : Une gestion avec lecture consistante ne corrige pas le problème

Ajout d'un trigger de pose de verrous :

```
Create or replace trigger C1_Verrou_Occupations
before insert or update of NumTy, NumHo on Occupations
begin
     lock table Occupations in share mode;
end;
```

<u>Évaluation du degré de blocage des autres transactions</u> :

La transaction qui déclenche le trigger ci-dessus pose sur la table Occupations les verrous suivants:

- verrou **ROW EXCLUSIVE** posé par l'instruction déclenchante
- verrou **SHARE** posé par le trigger

Lorsque les 2 verrous sont posés, il y a conversion en un verrou **SHARE ROW EXCLUSIVE**.

Cette transaction pose aussi des verrous exclusifs sur toutes les lignes concernées par l'instruction déclenchante.

Blocage des autres transactions en fonction de ce qu'elles font sur la table Occupations :

Select	Exclusive	Share	Row Share Select for update	Row Ex. Insert Delete Update	Share Row Exclusive
Non Bloqué	Bloqué	Bloqué	Non Bloqué[1]	Bloqué	Bloqué

(1) Sauf si l'instruction porte sur une ligne déjà verrouillée.

3) En utilisant une table temporaire, proposer une nouvelle implantation qui améliore la gestion de la concurrence. Et faire une évaluation de degré de blocage des transactions.

<u>Principe</u> : Bloquer toute transaction qui insère ou modifie une ligne qui interfère avec la lecture en utilisant le verrouillage sous-jacent à l'implantation de la contrainte déclarative d'unicité SQL (i.e. PRIMARY KEY ou UNIQUE).

```
create table TempCC2 (
NumCh numeric(10),
NumHo numeric(10),
primary key(NumCh,NumHo));
```

```
create or replace trigger remplir_TempCC2
after insert or update on Occupations
for each row
begin
   insert into TempCC2 values(:New.NumCh,:New.NumHo);
exception
   when dup_val_on_index then null;
end;

create or replace trigger C2
after insert or update on Occupations
declare
   N binary_integer;
begin
   select 1 into N
   from TempCC2 T, Occupations O1, Occupations O2
   where T.NumCh = O1.NumCh and T.NumHo = O1.NumHo
   and T.NumCh = O2.NumCh and T.NumHo = O2.NumHo
   and O1.DateA <> O2.DateA
   and (O1.DateA, O1.DateD) overlaps (O2.DateA, O2.DateD);
   raise too_many_rows;
exception
   when no_data_found then delete TempCC2;
   when too_many_rows then
     raise_application_error(-20000,'PB C2');
end;
```

Évaluation du degré de blocage des transactions :

La transaction qui déclenche le trigger ci-dessus pose sur la table Occupations le verrou suivant :

- verrou **ROW EXCLUSIVE** posé par l'instruction déclenchante

Elle pose aussi des verrous exclusifs sur toutes les lignes concernées par l'instruction déclenchante.

Blocage des autres transactions en fonction de ce qu'elles font sur la table Occupations :

Select	Exclusive	Share	Row Share Select for update	Row Ex. Insert Delete Update	Share Row Exclusive
Non Bloqué	Bloqué	Bloqué	Non Bloqué[1]	Non Bloqué[2]	Bloqué

(1) Sauf si l'instruction porte sur une ligne déjà verrouillée.

(2) Sauf pour les instructions Insert et Update lorsqu'elles portent sur un couple (NumCh, NumHo) qui se trouve déjà dans la table temporaire. La suppression n'est jamais bloquée.