LIF10 – FONDEMENTS DES BASES DE DONNÉES
TP4 – Transactions et vues

Licence informatique – Automne 2016–2017

`http://liris.cnrs.fr/romuald.thion/dokuwiki/enseignement:lif10/`

1 Déclencheurs

Exercice 1 : automatisation des identifiants de spécialité

On reprend ici l'exercice 3 du TP précédent pour incrémenter automatiquement les identifiants de spécialités de la nouvelle table *Specialite* .

1. Créer un déclencheur avant insertion sur la table *Specialite* qui remplace le *id_specialite* inséré par une nouvelle valeur générée par la séquence [1]. Penser à utiliser la commande `SHOW ERRORS;` pour débugger le déclencheur.
2. Tester le déclencheur précédant en comparant l'état de la *Specialite* avant et après une insertion.
3. Créer un déclencheur avant mise-à-jour sur la table *Specialite* qui va interdire la modification du champs *id_specialite* en levant une exception dans le déclencheur avec `raise_application_error`.
4. Tester le déclencheur précédant en essayant de modifier *nom_specialite* puis *id_specialite*

2 Transactions

Exercice 2 : atomicité d'une transaction courante

1. Créer une table `Etudiants` et y insérer quelques lignes, consulter le contenu de la table, modifier une ligne, en supprimer une autre et enfin annuler les mises à jour venant d'être effectuées avec la commande `ROLLBACK`. Voir à nouveau le contenu de la table et sa structure. Pourquoi la table est-elle vide ?
2. Insérer à nouveau quelques lignes dans `Etudiants`, les modifier et les détruire partiellement, puis valider ces mises à jour avec la commande `COMMIT`, puis déclencher un `ROLLBACK`. Que s'est-il passé ? Maintenant détruire les données de la table et valider.
3. Insérer à nouveau dans `Etudiants` quelques lignes et clore la transaction par un `EXIT` ou un `QUIT`. Que s'est-il passé ?
4. Insérer à nouveau deux ou trois lignes dans la table `Etudiants` dans et fermer brutalement `SqlDeveloper`, puis rentrer à nouveau dans votre compte. Les données saisies ont-elles été préservées ?
5. Insérer à nouveau quelques lignes dans la table `Etudiants`, puis adjoindre une nouvelle colonne à sa table (ou plus généralement émettre n'importe quelle commande de description des données) et essayer d'annuler les dernières insertions. Pourquoi est-ce impossible ? Supprimer la colonne créée.
6. Insérer à nouveau quelques lignes dans la table `Etudiants`, puis fermer normalement `SqlDeveloper`. Une fenêtre apparaît, quels sont les choix proposés ?
7. En conclusion, qu'est-ce qu'une transaction courante et comment valider ou annuler une transaction ?

1. En cas de popup intempestif demandant la valeur de `:New`, créer le trigger dans un nouveau fichier SQL que vous exécuterez en intégralité.

Exercice 3 : transactions concurrentes

1. Vider la table `Etudiants`, faire un `COMMIT` puis ajouter quelques lignes. Ensuite, se connecter à son compte à partir *d'une autre instance* de `SqlDeveloper` (ou sur un autre poste de travail) et consulter à travers cette nouvelle fenêtre le contenu du compte. Que voyez-vous ?
2. Insérer dans chacune des deux fenêtres deux ou trois lignes distinctes. Que voit-on de l'autre fenêtre ?
3. Créer dans l'une des deux fenêtres ouvertes une nouvelle table `UE` et y insérer par chacune des fenêtres quelques lignes. Que voit-on de la table `Etudiants` ?
4. Détruire la table `UE` dans une des fenêtres. Que se passe-t-il ? Consulter le contenu de la table `UE` depuis l'autre fenêtre. Que contient-elle désormais ? Comment détruire `UE` ?
5. Adjoindre une clé primaire à `Etudiants`. Insérer *une même ligne* dans la première fenêtre puis dans la deuxième. Que se passe-t-il ? Émettre un `ROLLBACK` dans la première fenêtre. Que devient le blocage ?
6. Même question que précédemment : on insère depuis deux session différentes un même tuple qui violerait la contrainte de clé primaire, par contre, on émet un `COMMIT` dans la première. Que devient la seconde insertion ?
7. Ouvrir un nouveau fichier dans une instance de `SqlDeveloper` en utilisant la connexion déjà existante. Effectuer une insertion dans `Etudiants` dans un des onglets et consulter la table dans l'autre. Que voyez-vous ?
8. En conclusion, comment sont liés comptes, sessions et transactions ?

3 Vues

Exercice 4 : création de vues

On considère le script de création de table suivant :

```
  CREATE TABLE TA
( ID_A NUMBER PRIMARY KEY,
  NAME_A VARCHAR2(32)
);

INSERT INTO TA VALUES(0,'foo');
INSERT INTO TA VALUES(1,'bar');

CREATE TABLE TB
( ID_B NUMBER PRIMARY KEY,
  NAME_B VARCHAR2(32),
  REF_A NUMBER REFERENCES TA(ID_A)
);

INSERT INTO TB VALUES(0,'foofoo',0);
INSERT INTO TB VALUES(1,'foobar',1);

COMMIT;
```

1. Créer une vue `TB _REF0` qui sélectionne tous les tuples de `TB` qui font référence au tuple $(0, \texttt{foo})$ de `TA`. Ajouter les tuples $(2, \texttt{foofoo2}, 0)$ $(3, \texttt{foobar2}, 1)$ à `TB _REF0` avec un `INSERT`. Quels sont les tuples ajoutés à `TB _REF0` et ceux ajoutés à `TB` ? Que constatez-vous ? Annulez les modifications pour revenir à l'état initial.
2. Créer maintenant une vue `TB_TA` qui fait la jointure de `TA` et `TB` sur les attributs `TB.REF_A` et `TA.ID_A`. Essayer d'insérer le tuple $(2, \text{foofoo2}, 0, 0, \text{foo})$. Quel est le problème ?
3. Essayer d'insérer dans `TB_TA` le tuple $(2, \text{foofoo2}, 0)$ qui n'est définit que pour les attributs des `TB`. Consulter la vue `TB _TA` et la table `TB`, que constatez-vous ? Essayer ensuite d'insérer ensuite le tuple $(3, \text{foofoo2}, 3)$, quel est le problème ? Annulez les modifications pour revenir à l'état initial.

4. Essayer d'insérer dans `TB_TA` le tuple $(2, gaz)$ qui n'est définit que pour les attributs des `TA`. Quel est le problème ?
5. Reprendre la question précédente, mais cette fois sur une vue `TB_TA_ID` qui fait la jointure de `TB` et `TA` sur les attributs `TB.ID_B` et `TA.ID_A`. Pouvez-vous critiquer de la restriction imposée par Oracle sur ce type de mise-à-jour des vues ?
6. Créer une vue `TB_TA_2` similaire à celle de la question 1 avec les spécifications supplémentaires suivantes :
   — un attribut `NB_B` va compter pour chaque tuple de `TA` le nombre de tuples de `TB` qui lui font référence ;
   — les tuples de `TA` qui n'ont aucun tuple de `TB` correspondant doivent apparaître avec la valeur 0 pour `NB_B` et les attributs de `TB` à `NULL`

4 Privilèges

Exercice 5 : privilèges entre deux comptes

Les groupes de TP travaillent dans cette partie deux par deux.

1. Chaque groupe donne le droit à l'autre groupe de consulter l'une de ses tables en émettant un `GRANT SELECT ON Table TO autreGroupe ;`. Vérifier que ce privilège a été donné en effectuant une requête sur `ALL_TABLES` et `USER_TAB_PRIVS`. Consulter la table à laquelle l'autre groupe vous a donné accès.
2. Quand l'autre groupe fait une mise à jour sur sa table, que voyez-vous ?
3. Essayer d'insérer une ligne dans la table de l'autre groupe. Quel est le problème ?
4. Réaliser une requête utilisant d'une de vos tables avec une table de vos camarades.

Corrections

Solution de l'exercice 1

1.
```
CREATE OR REPLACE TRIGGER specialite_auto_increment
  BEFORE INSERT ON SPECIALITE
  FOR EACH ROW
  BEGIN
    SELECT specialite_seq.nextval INTO :NEW.id_specialite
    FROM dual;
  END;
/
```

2.
```
SELECT * FROM SPECIALITE;
SELECT specialite_seq.currval from dual;

INSERT INTO SPECIALITE VALUES (42,'Ca');

SELECT * FROM SPECIALITE;
SELECT specialite_seq.currval from dual;

ROLLBACK;
```

3.
```
CREATE OR REPLACE TRIGGER specialite_update
  BEFORE UPDATE OF ID_specialite ON SPECIALITE
  FOR EACH ROW
  BEGIN
    raise_application_error(-20000,'Cannot update id (auto increment)');
  END;
/
```

4.
```
select * from SPECIALITE;
insert into SPECIALITE values(42, 'Bases de données');
select * from SPECIALITE;

update SPECIALITE set NOM_SPECIALITE = 'BD avançées'
  where ID_SPECIALITE = 42;
-- ne fait rien : 0 rows updated.
update SPECIALITE set NOM_SPECIALITE = 'BD avançées'
  where NOM_SPECIALITE = 'Bases de données';
-- modifie : 1 row updated.
select * from SPECIALITE;

update SPECIALITE set  ID_SPECIALITE= 42
  where NOM_SPECIALITE = 'BD avançées';
-- erreur :
-- SQL Error: ORA-20000: Cannot update id (auto increment)
-- ORA-06512: at "RTHION.SPECIALITE_UPDATE", line 2

ROLLBACK;
```

Solution de l'exercice 2

```
-------------------------------------------------------------
-- Exercice -- atomicité d'une transaction courante
-------------------------------------------------------------
```

```
-- Question 1 ----------------------------------------

DROP TABLE Etudiants;
CREATE TABLE Etudiants(NOM VARCHAR2(16));

INSERT INTO Etudiants VALUES('Louis');
INSERT INTO Etudiants VALUES('Luc');
INSERT INTO Etudiants VALUES('Marc');

SELECT * FROM ETUDIANTS;
-- Etudiants = {Louis, Luc, Marc}

UPDATE Etudiants SET Nom = 'Lucie' WHERE Nom = 'Luc' ;
DELETE FROM Etudiants WHERE Nom = 'Marc' ;
SELECT * FROM ETUDIANTS;
-- Etudiants = {Louis, Lucie}

ROLLBACK;
SELECT * FROM ETUDIANTS;
-- Etudiants = {}

-- Le ROLLBACK a annulé toutes les modifications, qui ont toutes été effectuées
-- dans la même transaction. La structure n'a pas été modifiée.
-- Notez qu'il y a un commit automatique après un CREATE TABLE

-- Question 2 ----------------------------------------

INSERT INTO Etudiants VALUES('Louis');
INSERT INTO Etudiants VALUES('Luc');
INSERT INTO Etudiants VALUES('Marc');
UPDATE Etudiants SET Nom = 'Lucie' WHERE Nom = 'Luc' ;
DELETE FROM Etudiants WHERE Nom = 'Marc' ;

COMMIT ;
SELECT * FROM ETUDIANTS;
-- Etudiants = {Louis, Lucie}

ROLLBACK;
SELECT * FROM ETUDIANTS;
-- Etudiants = {Louis, Lucie}

-- Le ROLLBACK est sans effet puisque la transaction courante est vide :
-- on a rien fait depuis le dernier COMMIT

DELETE FROM ETUDIANTS;
COMMIT;

-- Question 3 ----------------------------------------

INSERT INTO Etudiants VALUES('Jules');
INSERT INTO Etudiants VALUES('Julie');
Exit;
SELECT * FROM ETUDIANTS;
-- Etudiants = {Louis, Lucie}

-- QUIT et EXIT valident les transactions en cours

-- Question 4 ----------------------------------------
INSERT INTO Etudiants VALUES('EXIT');
SELECT * FROM ETUDIANTS;
```

```
-- Après coupure brutale de la connction, un ROLLBACK est implicitement effectué

-- Question 5 ------------------------------------------

INSERT INTO Etudiants VALUES('Louis');
INSERT INTO Etudiants VALUES('Luc');
INSERT INTO Etudiants VALUES('Marc');
SELECT * FROM ETUDIANTS;
-- Etudiants = {Louis, Luc, Marc}

ALTER TABLE Etudiants ADD  Ville VARCHAR2(20);
ROLLBACK;
SELECT * FROM ETUDIANTS;
-- Etudiants = {(Louis,null), (Luc,null), (Marc,null)}

-- Les commades de modifications de schéma dont ALTER TABLE font un COMMIT
-- implicite, impossible donc d'annuler les deux insertions

ALTER TABLE Etudiants DROP COLUMN Ville;


-- Question 6 ------------------------------------------
INSERT INTO Etudiants VALUES('Jean-Marc');
SELECT * FROM ETUDIANTS;
-- Etudiants = {Louis, Luc, Marc, Jean-Marc}



-- On a une invite qui nous propose :
--   * un COMMIT de la transaction courante
--   * un ROLLBACK de la transaction courante
--   * d'annuler la fermeture

-- Question 7 ------------------------------------------
-- La suite des mises à jour effectuées dans une session (dans sqldeveloper
-- ou l'intepréteur interactif SQL) depuis la dernière validation ou depuis
-- la dernière annulation, constitue la transaction *courante*.
-- Ces mises à jour sont :
--    * validées toutes ensemble d'un seul bloc par un COMMIT
--      explicite ou implicite (QUIT/EXIT, ALTER)
--    * annulées toutes ensemble d'un seul bloc par un ROLLBACK
--      explicite ou implicite (fin de session brutale)
```

Solution de l'exercice 3

```
-------------------------------------------------------------------
-- Exercice -- transactions concurrentes
-------------------------------------------------------------------

-- Question 1 ------------------------------------------

-- A) sur la première connection
DELETE FROM ETUDIANTS;
COMMIT;
SELECT * FROM ETUDIANTS;
INSERT INTO Etudiants VALUES('Louis');
INSERT INTO Etudiants VALUES('Luc');
INSERT INTO Etudiants VALUES('Marc');
SELECT * FROM ETUDIANTS;
```

```
-- Etudiants = {Louis, Luc, Marc}

-- B) sur la deuxième connection
 SELECT * FROM ETUDIANTS;
-- Etudiants = {}

-- Si on execute SELECT * FROM ETUDIANTS; depuis une autre connection, on voit
-- le dernier état enregistré, validé par un COMMIT , à savoir Etudiants = {}

-- Question 2 ------------------------------------------

-- A) sur la première connection
SELECT * FROM ETUDIANTS;
-- Etudiants = {Louis, Luc, Marc}

-- B) sur la deuxième connection
INSERT INTO Etudiants VALUES('Jean-Marc');
SELECT * FROM ETUDIANTS;
-- Etudiants = {Jean-Marc}

-- Les modifications des transactions courantes ne sont visibles QUE dans
-- la session où elles ont été effectuées, tant qu'elles n'ont pas été validées.
-- On voit donc des résultats différents selon la connection où l'on se trouve

-- Question 3 ------------------------------------------

-- A) sur la première connection
CREATE TABLE UE
( NomUE VARCHAR2(30),
  NbEtudiants INTEGER
);

INSERT INTO UE VALUES ('SI', NULL);
INSERT INTO UE VALUES ('BD', 50);
SELECT * FROM UE;
-- UE = {(SI,null),(BD,50)}

SELECT * FROM ETUDIANTS;
-- Etudiants = {Louis, Luc, Marc}

-- B) sur la deuxième connection
SELECT * FROM ETUDIANTS;
-- Etudiants = {Louis, Luc, Marc, Jean-Marc}

INSERT INTO UE VALUES ('SE', 50);
SELECT * FROM UE;
-- UE = {(SE,50)}

-- Comme dans la question précédente, une commande de description de
-- données (ici un CREATE TABLE) valide la transaction courante.
-- Les insertions dans ETUDIANTS sont visibles dans la deuxième session.
-- Pour la table UEs, chacun voit uniquement ses modifications locales


-- Question 4 ------------------------------------------

-- A) sur la première connection

DROP TABLE UE;

-- On obtient le message d'erreur suivant :
```

```
-- SQL ERROR: ORA-00054: RESOURCE busy AND acquire WITH NOWAIT specified
--              OR TIMEOUT expired
-- 00054. 00000 -  "resource busy and acquire with NOWAIT specified"

-- B) sur la deuxième connection
SELECT * FROM UE;
-- UE = {(SI,null),(BD,50),(SE,50)}

-- Même si le DROP TABLE echoue, la transaction dans la première session
-- a été validée et les modifications sont visibles.

-- Pour détruire la table UE, il faut le faire depuis la deuxième fenëtre
-- ou alors COMMIT et supprimer d'où l'on veut : un suppression réussit
-- que si il n'y a pas d'autres sessions qui ont modifié la table à détruire.



-- Question 5 ------------------------------------------

-- A) sur la première connection
ALTER TABLE Etudiants ADD CONSTRAINT pkEtudiants PRIMARY KEY(Nom);
SELECT * FROM ETUDIANTS;
INSERT INTO Etudiants VALUES('Martine');

-- B) sur la deuxième connection
INSERT INTO Etudiants VALUES('Martine');

-- A) sur la première connection
ROLLBACK;

COMMIT;

-- La deuxième session est bloquée tant que la première session n'a pas validé
-- sa transaction. En effet, bien que les modifications ne soient pas visibles
-- dans la deuxième fenëtre. Si on accepte la deuxième insertion, on viole la
-- contrainte. Après ROLLBACK, la situation est débloquée et le tuple est
-- inséré dans la deuxième session.

-- Question 6 ------------------------------------------

-- A) sur la première connection
INSERT INTO Etudiants VALUES('Martine');

-- B) sur la deuxième connection
INSERT INTO Etudiants VALUES('Martine');

-- A) sur la première connection
COMMIT;

-- Similaire à précédement, à la différence que le tuple n'est PAS
-- inséré dans la deuxième session.


-- Question 7 ------------------------------------------
-- A) sur le premier onglet
INSERT INTO Etudiants VALUES('Martine');

-- B) sur la deuxième onglet
SELECT * FROM ETUDIANTS;

-- on utilise en fait la mëme session et toutes les modifications
```

```
-- sont visibles d'un onglet ou de l'autre

-- Question 8 ------------------------------------------
-- Un compte peut disposer de plusieurs sessions en parallèle.
-- Chaque session voit localement ses modifications effectuées
-- dans sa transaction courante. Quand la transaction est validée,
-- les modifications sont visibles depuis toutes les sessions.
```

Solution de l'exercice 4

```
-------------------------------------------------------------
-- Exercice -- création et mise à jour de vues
-------------------------------------------------------------
CREATE TABLE TA
( ID_A NUMBER PRIMARY KEY,
  NAME_A VARCHAR2(32)
);

INSERT INTO TA VALUES(0,'foo');
INSERT INTO TA VALUES(1,'bar');

CREATE TABLE TB
( ID_B NUMBER PRIMARY KEY,
  NAME_B VARCHAR2(32),
  REF_A NUMBER REFERENCES TA(ID_A)
);

INSERT INTO TB VALUES(0,'foofoo',0);
INSERT INTO TB VALUES(1,'foobar',1);

COMMIT;

-- Question 1 ------------------------------------------
CREATE OR REPLACE VIEW TB_REF0 AS
  SELECT *
  FROM TB
  WHERE TB.REF_A = 0;

SELECT * FROM TB;
SELECT * FROM TB_REF0;

INSERT INTO TB_REF0 VALUES(2,'foofoo2',0);
INSERT INTO TB_REF0 VALUES(3,'foobar2',1);

-- Les deux insertions ont réussies et ont été ajoutée à TB.
-- Le premier tuple est visible dans TB_REF0 mais pas le second !


-- Question 2 ------------------------------------------
CREATE OR REPLACE VIEW TB_TA AS
  SELECT *
  FROM TB JOIN TA ON TB.REF_A = TA.ID_A;

SELECT * FROM TB_TA;

INSERT INTO TB_TA VALUES(2,'foofoo2',0, 0, 'foo');
-- SQL Error: ORA-01776: cannot modify more than one base table
--            through a join view
-- On ne peut pas modifier deux tables avec un seul insert
```

```
-- Question 3 ------------------------------------------

INSERT INTO TB_TA(ID_B, NAME_B,REF_A) VALUES(2,'foofoo2',0);
SELECT * FROM TB_TA;
SELECT * FROM TB;

-- on voit que le tuple a été ajouté à TB et est visible dans TB_TA

INSERT INTO TB_TA(ID_B, NAME_B,REF_A) VALUES(3,'foofoo3',3);
-- violation de clef étrangère : le tuple est refusé

-- Question 4 ------------------------------------------

INSERT INTO TB_TA(ID_A, NAME_A) VALUES(2,'gaz');
-- SQL ERROR: ORA-01779: cannot MODIFY A COLUMN which maps TO A
--            non KEY-preserved TABLE
-- 01779. 00000 - "cannot modify a column which maps to
--                 a non key-preserved table"

-- il y a une restriction : on ne peut pas ajouter ce tuple
-- car l'attribut REF_A de TB n'est pas clef de la relation.

-- Question 5 ------------------------------------------

CREATE OR REPLACE VIEW TB_TA_ID AS
  SELECT *
  FROM TB JOIN TA ON TB.ID_B = TA.ID_A;

INSERT INTO TB_TA_ID(ID_A, NAME_A) VALUES(2,'gaz');
-- cette fois si c'est possible.

SELECT * FROM TA;

-- en fait, dans le cas précédent il n'y a pas d'ambiguité et
-- la modification pourrait ëtre effectuée sans risque, c'est
-- donc trop restrictif, mais on peut toujours effectuer
-- la modification sur la table source.

-- Question 6 ------------------------------------------

CREATE OR REPLACE VIEW TB_TA_2 AS
  SELECT *
  FROM TB RIGHT JOIN (
    SELECT ID_A, NAME_A, COUNT(ID_B) AS NB_B
    FROM TA LEFT JOIN TB ON TB.REF_A = TA.ID_A
    GROUP BY ID_A, NAME_A
  ) TA2 ON TB.REF_A = TA2.ID_A;


SELECT * from TB_TA_2;

-- NB : a priori, on ne peut pas le faire sans requete imbriquée dans le FROM
-- à cause du GROUP BY nécessaire pour le comptage
```

Solution de l'exercice 5

```
-------------------------------------------------------------
-- Exercice -- privilèges entre deux comptes
-------------------------------------------------------------
-- on considère que les comptes sont nommés L3IF70 et RTHION
```

```
-- Question 1 ----------------------------------------

-- A) sur le premier compte L3IF70
CREATE TABLE Etudiants(NOM VARCHAR2(16));

INSERT INTO Etudiants VALUES('Louis');
INSERT INTO Etudiants VALUES('Luc');
INSERT INTO Etudiants VALUES('Marc');
COMMIT;

GRANT SELECT ON Etudiants TO RTHION ;

SELECT * FROM ALL_TABLES WHERE OWNER = 'L3IF70';
SELECT * FROM USER_TAB_PRIVS;


-- B) sur le deuxième compte RTHION
SELECT * FROM L3IF70.ETUDIANTS;

-- NB : il faut préfixer les noms de tables par l'ID du propriétaire
--      lors des requëtes d'un compte dur l'autre

-- Question 2 ----------------------------------------

-- A) sur le premier compte L3IF70
INSERT INTO Etudiants VALUES('Jean-Marc');

-- B) sur le deuxième compte RTHION
SELECT * FROM L3IF70.ETUDIANTS;
-- Etudiants = {Louis, Luc, Marc}

-- On a bien deux sessions différentes avec leurs propres transactions courantes :
-- RTHION ne voit pas les modifications non validées effectuées par L3IF70

-- Question 3 ----------------------------------------


-- B) sur le deuxième compte RTHION
INSERT INTO L3IF70.Etudiants VALUES('Jean-Marc');
-- SQL Error: ORA-01031: insufficient privileges
-- 01031. 00000 - "insufficient privileges"

-- On a une erreur, car seul le droit SELECT a été donné, pas celui d'insertion


-- Question 4 ----------------------------------------
SELECT * FROM L3IF70.ETUDIANTS
UNION
SELECT * FROM RTHION.ETUDIANTS;


-- un exemple parmi d'autres
```