M1 MIAGE FC/FA – 2015/2016

Système de Gestion de Bases de Données

octobre 2015

TP - intégrité des données

Exercice 1 :

Créez une table T, et ajoutez quelques lignes de la façon suivante :

```
create table T(
  a number(3) constraint t_pkey primary key,
  b number(3) constraint valeur_b check (b between 1 and 5)
);

alter table t disable constraint t_pkey disable constraint valeur_b ;

insert into t values (4,2);
insert into t values (1,6);
insert into t values (1,3);
insert into t values (4,9);
insert into t values (3,3);
commit ;
```

Question 1.1 : Exécutez les instructions suivantes :

```
create table exceptions (row_id rowid,
                owner varchar2(30),
                table_name varchar2(30),
        constraint varchar2(30));

alter table t ENABLE VALIDATE constraint t_pkey EXCEPTIONS INTO exceptions;
```

Que provoque la dernière instruction ? pourquoi ? Regardez le contenu de la table `Exceptions`, et déduisez-en les lignes de `T` qui posent problème.

Question 1.2 : Même question avec l'instruction suivante :

```
ALTER TABLE t ENABLE VALIDATE CONSTRAINT valeur_b EXCEPTIONS INTO exceptions;
```

On a vu en cours que `ENABLE NOVALIDATE` permet d'activer une contrainte sans vérifier le contenu de la table.

Question 1.3 : Exécutez l'instruction suivante :

```
ALTER TABLE t ENABLE NOVALIDATE constraint valeur_b ;
```

Ajoutez 1 ligne qui vérifie la contrainte, et 1 ligne qui invalide la contrainte.

Question 1.4 : Exécutez l'instruction suivante et expliquez ce qui se passe :

```
ALTER TABLE t ENABLE NOVALIDATE PRIMARY KEY;
```

Exercice 2 : Créez une table `X` et un trigger sur cette table :

```
create table x(
a1 varchar2(5) not null);

create or replace trigger tt1
before insert on x
for each row
begin
  if :new.a1 is null then
    raise_application_error(-20000,'erreur tt1');
  end if ;
end ;
```

Question 2.1 : Exécutez `insert into x values(null);`. Que se passe-t-il ? Que peut-on en déduire ?

Question 2.2 : Transformez le trigger `before` en trigger `after`. Reprenez la question précédente.

Exercice 3 : On créé deux tables avec une contrainte d'intégrité référentielle :

```
create table primaire(x number(2) primary key, y number(2));
create table etrange(x number(2) references primaire, z number(2));

insert into primaire values(1,1);
commit ;
```

On définit un trigger sur la table `etrange` :

```
create or replace trigger tt2
after insert on etrange
for each row
begin
  delete from primaire where x=:new.z;
end ;
```

Question 3.1 : Exécutez `insert into etrange values(1,1);`. Que se passe-t-il ? Que peut-on en déduire ?

Question 3.2 : Transformez le trigger `after` en trigger `before`. Reprenez la question précédente.

Supprimez le trigger `tt2`, insérez deux nouvelles lignes et définissez le trigger `tt3` :

```
drop trigger tt2 ;

insert into primaire values(2,2);
insert into etrange values(1,2);
commit ;
```

```
create or replace trigger tt3
after update on etrange
for each row
begin
  delete from primaire where primaire.x = :new.x ;
end ;
```

Question 3.3 : Exécutez `update etrange set x=2 where x=1 ;`. Que se passe-t-il ? Quel est le contenu des 2 tables ?

Supprimez et recréez la table `etrange` avec une contrainte `on delete cascade` :

```
drop table etrange ;

create table etrange(x number(2) references primaire on delete cascade, z number(2));
```

Redéfinissez le trigger `tt2` :

```
create or replace trigger tt2
after insert on etrange
for each row
begin
  delete from primaire where primaire.x = :new.x ;
end ;
```

Question 3.4 : Exécutez `insert into etrange values(2,8);`. Que se passe-t-il ? Expliquez.

Exercice 4 : On considère des employés qui occupent dans une ESN un certain nombre d'emplois liés à des fonctions (par exemple développeur, chef de projets, ...). On créé 2 tables `Emp` et `Fonction`, et on mémorise le plus bas et le plus haut salaire versés pour chaque fonction au sein de l'entreprise :

```
create table Fonction(
  fonc_id number(2) constraint fonc_pkey primary key,
  fonc_lib varchar2(20) not null,
  fonc_sal_min number(6), -- plus bas salaire pour cette fonction
  fonc_sal_max number(6), -- plus haut salaire pour cette fonction
  constraint coherence_min_max check (fonc_sal_min <= fonc_sal_max)
);

create table Emp(
  emp_id number(3) constraint emp_pkey primary key,
  emp_nom varchar2(20) not null,
  emp_prenom varchar2(20) not null,
  emp_salaire number(6), -- salaire brut annuel
  emp_fonction number(2) not null constraint emp_fonction_fkey references FONCTION
);
```

Question 4.1 : Comment à l'aide de triggers s'assurer que le calcul de la fourchette de salaires soit correcte quand le contenu de la table `Emp` évolue ? Quelle sont les difficultés ?

Exercice 5 : Avec un curseur lié à une requête `select ... for update`, on peut modifier (ou supprimer) la ligne courante en y faisant référence grâce à la clause `where current of ...` de l'instruction `update` (ou `delete`).

- `Current of` fait référence à la dernière ligne acquise par un `fetch` ou dans une boucle `for`.
- Attention : on va retrouver les mêmes problèmes que lorsqu'on veut modifier une vue. Il faut que la requête liée au curseur permette la modification.
- Quand la requête porte sur plusieurs tables, on précise la colonne qui va être modifiée : clause `for update of`. En effet, le "current of" fait référence à 1 rowid dans 1 bloc, il y a donc un problème d'ambiguité si la requête porte sur plusieurs tables.
- Quand on ne met pas de `of`, toutes les lignes de toutes les tables sont verrouillées. Préciser 1 colonne dans la clause `for update of` permet de verrouiller seulement les lignes de la table qui contient cette colonne.
- Les lignes sont verrouillées à l'ouverture du curseur.

Exemple de la documentation Oracle :

```
DECLARE
   my_emp_id NUMBER(6);
   my_job_id VARCHAR2(10);
   my_sal    NUMBER(8,2);
   CURSOR c1 IS SELECT employee_id, job_id, salary FROM employees FOR UPDATE;
BEGIN
   OPEN c1;
   LOOP
      FETCH c1 INTO my_emp_id, my_job_id, my_sal;
      IF my_job_id = 'SA_REP' THEN
        UPDATE employees SET salary = salary * 1.02 WHERE CURRENT OF c1;
      END IF;
      EXIT WHEN c1%NOTFOUND;
   END LOOP;
END;
```

Exemple qui utilise `For Update Of` :

```
CURSOR c1 IS SELECT last_name, department_name FROM employees, departments
    WHERE employees.department_id = departments.department_id
          AND job_id = 'SA_MAN'
      FOR UPDATE OF salary; -- row locking on Employees
```

La procédure suivante permet d'aligner au plus haut salaire tous les salaires d'une même fonction.

```
create or replace
procedure augmentation is
  cursor c is
  select * from emp for update ;
begin
  open c;
  --- on fait un traitement qcq ---
  update emp set emp_salaire = (select fonc_sal_max from fonction where fonc_id = emp_fonction);
  close c;
end augmentation;
```

Question 5.1 : Transformez la procédure pour que le update se fasse durant le parcours du curseur (la requête select portera sur les 2 tables)