PHALAVANDISHVILI Demetre - JAMET Felix - Groupe 601B

TP3 PL/SQL

Mise à jour de la note moyenne

Q1

a)

Pour ajouter la nouvelle colonne dans la relation *Livres*, on utilise la commande suivante :

```
Alter table Livres add note_moy(4,2);
```

```
DECLARE
    num number(4,2);
    reflLivre varchar2(10) := '&reflLivre';
BEGIN
    select avg(note) into num
        from avis
        where refl = reflLivre;
    update livres set note_moy = num where refl = reflLivre;
END;
/
```

b)

Pour récupérer toutes les références des livres que l'on possède dans notre base de données, on utilise un curseur nous permettant de parcourir et de récupérer une à une les lignes du résultat de la requête suivante:

```
cursor lesrefls is select refl from livres;
```

Puis, en utilisant une boucle *for*, on parcours les références en calculant la moyenne pour chaque livre

```
DECLARE
    moy Livres.note_moy%TYPE;
    cursor lesrefls is select refl from livres;
BEGIN
    for n in lesrefls LOOP
        select avg(note) into moy
        from avis
        where refl = n.refl;
        update livres set note_moy = moy where refl = n.refl;
    End LOOP;
END;
/
```

c)

```
CREATE OR REPLACE PROCEDURE calculMoyenne IS
    moy Livres.note_moy%TYPE;
    cursor lesrefls is select refl from livres;
BEGIN
    for n in lesrefls LOOP
        select avg(note) into moy
        from avis
        where refl = n.refl;
        update livres set note_moy = moy where refl = n.refl;
    End LOOP;
END;
/
```

Q2

```
SET SERVEROUTPUT ON
CREATE OR REPLACE TRIGGER maj_note_moy
AFTER INSERT OR UPDATE on avis
FOR EACH ROW
DECLARE
    moy Livres.note_moy%type;
BEGIN
    if inserting or updating then
        select avg(note) into moy
        from avis
        where refl = :new.refl;
        update livres set note_moy = moy where refl = :new.refl;
    end if;
END;
/
```

On écrit le trigger ci-dessus qui s'applique après l'ajout ou la modification d'un avis. Le but est de modifier dynamiquement la note moyenne de livre pour chaque avis exprimé. Mais on obtient une erreur en lançant la requête suivante :

```
INSERT INTO AVIS Values (6,'03B3',16.5,NULL);
```

Voici l'erreur obtenue :

```
ERROR at line 1:
ORA-04091: table DEMNA.AVIS is mutating, trigger/function may not see it
ORA-06512: at "DEMNA.MAJ_NOTE_MOY", line 5
ORA-04088: error during execution of trigger 'DEMNA.MAJ_NOTE_MOY'
```

Le SGBD nous empêche de modifier la table livre afin de garantir l'intégrité de la base de données.

Pour continuer le TP sans cette erreur, on ignore ce trigger avec la ligne suivante :

```
DROP TRIGGER maj_note_moy;
```

Cohérence Avis-Achat

Q1

```
SET SERVEROUTPUT ON
CREATE OR REPLACE TRIGGER coherenceAA
after INSERT on avis
FOR EACH ROW
DECLARE
    nb number;
    acheter_Liv exception;
BEGIN
    if inserting then
        select count(*) into nb
        from avis a left outer join achats b on a.idcl = b.idcl and a.refl = b.refl
        where a.idcl = :new.idcl and a.refl = :new.refl;
        if nb = 0 then
            raise acheter_Liv;
        END if;
    end if;
exception
    when acheter_Liv then
        raise_application_error(-20010,'Vous devez acheter un livre avant donner un avis');
END;
/
DROP TRIGGER coherenceAA;
```

Ici on verifie bien que le client qui veut mettre la commentaire a bien acheter le livre en utilisant jointure externe a gauche; pour obtenier juste le nombre de tuplet existant dans le relation *achats*. S'il n'existe pas on leve un erreur d'application.

Q2

```
CREATE OR REPLACE PROCEDURE modifNoteCom(idc in number,refe in varchar2) IS
    eval avis.note%type :=  &eval;
    comment avis.commentaire%type := '&comment';
    nbLigne number;
    pas_droit exception;
BEGIN
   Select count(*) into nbLigne from avis where idcl = idc and refl = refe;
   if nbLigne = 0 then
    raise pas_droit;
    else
         if eval is not null then
             UPDATE avis set note = eval where idcl = idc and refl = refe;
         end if;
         if comment is not null then
             UPDATE avis set commentaire = comment where idcl = idc and refl = refe;
         end if;
    end if;
exception
    when pas_droit then
         DBMS_OUTPUT.PUT_LINE('Il faut avoir un avis, avant de modifier l''avis');
END;
/
```

Ici le but de cette procédure est de vérifier que l'utilisateur modifie bien son avis sur un livre. Dans la procédure, quand le client modifie la note attribuée ou le commentaire, s'il laisse vide les champs de la note ou le commentaire dans la base de données, ces valeurs vont conserver leur état initial. Avant de lancer la requête *update*, on vérifie que les valeurs de *idcl* et *refl* passées en paramètre correspondent bien à un tuple existant. On lève un erreur d'application si ça n'est pas le cas.

Traitement d'une inscription à un parcours

Q1

En utilisant les valeurs passés en paramètre, on remplit automatiquement la relation *inscrip_parcours*, puis en utilisant le curseur, on récupère les *id_evt* à partir de la relation *compo_parcours* en utilisant la requête suivante :

```
select distinct id_evt from compo_parcours cmp where cmp.idp = idpa;
```

Puis, en utilisant la boucle *for*, on parcours le curseur et on remplis la relation inscrip_evt.

```
For Unevt in lesEvt LOOP
    INSERT INTO inscrip_evt values (idCa,idpa,Unevt.id_evt);
END LOOP;
```

Q2

Pour afficher les parcours correspondant au genre du livre acheté, on utilise la requête suivante :

```
select distinct intitulep,date_deb
from parcours p
where p.genre = (select genre
                 FROM Livres l
                 WHERE l.refl = :new.refl)
and p.date_deb > (select sysdate from dual)
order by date_deb asc;
```

- *select sysdate from dual* = permet d'obtenir la date d'aujourd'hui.
- Le résultat de cette requête est mis dans le curseur, puis on affiche le contenu du curseur en utilisant les instructions suivantes :

```
DBMS_OUTPUT.PUT_LINE('On vous propose les parours suivants :');
For n in lesParcs LOOP
    DBMS_OUTPUT.PUT_LINE('"'||n.intitulep||'" commencant le '||n.date_deb);
END LOOP;
```