Cartouche du document

Année : ING 2
Activité : Travail dirigé

Objectifs

L'objectif de cette série d'exercices est :
- de se familiariser avec la gestion d'erreur dans un bloc PL/SQL
- de maîtriser la conception de fonctions et procédures stockées
- d'apprendre à utiliser des curseurs dans un bloc PL/SQL

Sommaire des exercices

Corps des exercices

1 - Gestion de commandes

Énoncé :

Soit le MLD suivant :

```
produit: idproduit,designationprod,prixprod,stockprod
commande : idcommande,datecom,montanttotalcom,#idclient
lignecommande : idligcom,quantiteprod,#idproduit,#idcommande
ligneerreur : idcommande,idligcom,idproduit,quantiteprod,stockprod
```

On désire, à chaque insertion d'une ligne de commande mettre à jour automatiquement le montant total de la commande correspondante ainsi que le stock du produit concerné par la ligne de commande.

Question 1)

Énoncé de la question

Ecrire le code correspondant en gérant le fait qu'une ligne de commande peut ne pas être satisfaite (stock inférieur à la quantité commandée).

Une ligne de commande non satisfaite devra faire l'objet d'une insertion dans la table LIGNEERREUR par l'intermédiaire d'une gestion d'exception.

1) Pour exécuter dans une fonction une transaction indépendante de la transaction courante, on utilise la clause PRAGMA AUTONOMOUS_TRANSACTION;. Cette clause apparaît dans la zone de déclaration des variables du bloc PL/SQL

2) Pour provoquer un rollback de la transaction courante en raison d'une erreur, on utilise la fonction raise_application_error(noErreur,msgErreur) où noErreur doit être un nombre négatif strictement inférieur à -20000 (les autres codes d'erreur sont réservés par Oracle).

Ci-dessous, vous trouverez le script de création des tables en vue tester le trigger.

```
drop table produit cascade constraints;
drop table commande cascade constraints;
drop table lignecommande cascade constraints;
drop table ligneerreur cascade constraints;
create table produit
(idproduit varchar2(5) primary key,
desprod varchar(50),
prixprod number(8,2),
stockprod number(4)
);
create table commande
(idcommande varchar2(5) primary key,
datecde date,
montanttotalcom number(10,2),
idclient varchar2(5)
);
create table lignecommande
(idlig varchar2(5),
idcommande varchar2(5) references commande(idcommande),
quantiteprod number(3),
idproduit varchar2(5) references produit(idproduit),
primary key (idlig,idcommande)
);
create table ligneerreur (
idcommande varchar2(5),
idlig varchar2(5),
idproduit varchar2(5),
quantiteprod number(3),
stockprod number(4)
);
insert into produit values ('P1','',50,50);
insert into produit values ('P2','',500,0);
insert into commande values ('cde1','',0,'c1');
```

Solution de la question

Le code qui vous est présenté permet de gérer l'insertion dans la table ligneerreur indépendemment du trigger.

C'est pourquoi, l'insertion est gérée dans une fonction en utilisant la directive pragma autonomous_transaction qui indique à oracle que la transaction définie dans la fonction est indépendante de la transaction courante.

Vous pouvez tester sans cette directive et vous constaterez que l'insertion n'est pas commité car la clause raise_application_error effectue un rollback et considère que le code comme atomique.

```
create or replace trigger maj_lignecde
before insert on lignecommande
for each row
declare
ok number ;
wdelta_stock produit.stockprod%type ;
wprixprod produit.prixprod%type ;
wstockprod produit.stockprod%type ;
erreur_stock exception ;
begin
select stockprod,stockprod-:new.quantiteprod ,prixprod into
wstockprod,wdelta_stock, wprixprod from produit
where idproduit = :new.idproduit ;
if wdelta_stock < 0 then raise erreur_stock ;
end if ;
update commande set
montanttotalcom=montanttotalcom+(wprixprod*:new.quantiteprod)
where idcommande=:new.idcommande ;
update produit set stockprod = stockprod-:new.quantiteprod where
idproduit=:new.idproduit ;
exception
when no_data_found then raise_application_error(-20001,'erreur prod') ;
when erreur_stock
then
begin
ok:=ecriture(:new.idcommande,:new.idlig,:new.idproduit,:new.quantiteprod,wstockprod);
raise_application_error(-20001,'erreur stock') ;
end ;
when others then null ;
end;
/
create or replace function ecriture(pidcommande varchar2,plig
varchar2,pidproduit varchar2,pquantiteprod
number,pstockprod number)
return number
is
PRAGMA AUTONOMOUS_TRANSACTION; -- transaction indépendante
begin
insert into ligneerreur values
(pidcommande,plig,pidproduit,pquantiteprod,pstockprod) ;
commit;
return 1;
end;
/
```

Question 2)

Énoncé de la question

On désire maintenant afficher la facture correpondant à une commande. Sur cette facture doit

apparaître : le client,le numéro de la commande et la date de la commande, le montant total ainsi que pour chaque produit commandé, la quantité, le prix et le montant à payer correspondant.

Ecrire la procédure qui acceptera en paramètre le numéro de la commande et qui permettra d'afficher la facture.

Solution de la question

```
create or replace procedure facture(pidcommande varchar2)
is
-- déclaration des variables utiles
vidcommande commande.idcommande%type ;
vidclient commande.idclient%type ;
vdatecde commande.datecde%type ;
vmontanttot commande.montanttotalcom%type ;
vmontant commande.montanttotalcom%type ;
-- déclaration du curseur qui contiendra les lignes de commande
-- pour une commande (vidcommande) concernée
cursor cur_lignecde(vidcommande varchar2) is
select idlig,quantiteprod,idproduit
from lignecommande
where idcommande = vidcommande ;
begin
-- on initialise la variable vidcommande avec la valeur passée en paramètre
vidcommande :=pidcommande ;
-- on récupère les valeurs de la table commande pour la commande --
--concernée(pidcommande)
select idclient,datecde,montanttotalcom into vidclient,vdatecde,vmontanttot
from commande
where idcommande = vidcommande ;
-- on affiche l entête de la facture
dbms_output.put_line('FACTURE client numéro ' || vidclient) ;
dbms_output.put_line('Commande numéro : '|| vidcommande);
dbms_output.put_line('Date de la commande : '|| vdatecde);
dbms_output.put_line('-----------------------------------------') ;
dbms_output.put_line('') ;
-- on gère les lines de commande à l aide du curseur
for vcur in cur_lignecde(pidcommande)
loop
-- pour chaque ligne on récupère le prix et on calcul le montan pour le produit
-- commandé
select prixprod*vcur.quantiteprod into vmontant from produit where
idproduit=vcur.idproduit ;
-- on affiche les info demandés pour chaque produit commandé
dbms_output.put_line
(vcur.idlig || ' ' ||vcur.idproduit||' '||vcur.quantiteprod||' '||vmontant);
end loop ;
-- on affiche le pied de la facture
dbms_output.put_line('-----------------------------------------') ;
```

```
dbms_output.put_line('Montant total de la commande : ' || vmontanttot );
exception
when no_data_found then
dbms_output.put_line('erreur dans le numéro de la commande') ;
end ;
/
```