

Principes d'utilisation des systèmes de gestion de bases de données

PL/SQL : triggers, exceptions et erreurs BD
Cours 5

M1 - Ecole Informatique d'Orsay
2008/09

Emmanuel Waller, LRI, Orsay

notion d'erreur

notion d'erreur

- un ordre SQL

envoyé depuis un environnement donné au serveur

peut ne pas être exécuté par le serveur (ou non parf.)

- pourquoi ? serveur considère ordre pas acceptable
- ex :
 - `select a from t : t n'existe pas`
 - `update u set b = 1 : il y a contrainte $b > 0$`
- on dit que cet ordre est en erreur

notion d'erreur

- réaction serveur ?
- environnement averti ? si oui comment ? quelles informations transmises par le serveur à l'environnement ?
- quelles sont les causes d'erreurs ?

outils fournis par le SGBD

- notion d'erreur
- ordre non exécuté
- réaction du serveur envers le client selon l'environnement de demande de l'ordre
- environnement toujours averti
- compte-rendu fourni par le serveur :
 - message d'erreur
 - code d'erreur (identifie l'erreur)
 - etc.

catégories d'erreurs : intuition

- syntaxe incorrecte : `select a where b = 1 from t`
- sémantique incorrecte :
 - schéma : table or view does not exist
 - instance :
 - constraint violated
 - `insert into t values ('toto') :`
type incorrect : `ex : integer, varchar2(3)`
- rem : surviennent en interactif ou en mode programme (PL/SQL, PHP, Java, etc.)

ex : en mode interactif

- réaction serveur :
 - affichage compte-rendu dans console
 - rend la main à l'entrée standard du client interactif (qui peut continuer à taper des ordres)
- ex : ORA-942: table or view does not exist
- rem : c'est à l'expert BD qui tape d'adapter les prochains ordres qu'il va taper
(ex : pas de insert into t)

triggers

triggers

- définition
- motivation
- exemples
- syntaxe générale

qu'est-ce que c'est ?

- « déclencheur » = bloc PL/SQL
 - Associé à une table
 - s'exécute lors d'un ordre DML (mise à jour d'instance) sur cette table

à quoi ça sert ?

- vérifier contraintes non exprimables en SQL
(ex : ancêtres)
- « tenir un journal » pour certaines mises à jour
(quand une mise à jour a lieu dans t, le trigger permet de mémoriser dans une autre table qu'elle a eu lieu, avec informations)

Exemple 1

- Créer un trigger qui prévienne d'une suppression :

```
create trigger avant_supp_cli
```

```
before delete
```

```
on client
```

```
begin
```

```
    dbms_output.put_line('je supprime un client');
```

```
end;
```

Exemple 2

- Agir en cas de baisse du prix :

```
create trigger post_maj_prix
  after update
  on article
  for each row
  when (old.prixht > new.prixht)
  begin
    dbms_output.put_line('ancien prix' || :old.prixht);
  end;
```

Syntaxe

Create [or replace] trigger nom

before | after

insert | update [of col] | delete | insert or delete

| ... or ...

on table

[for each row]

[when (. . .)]

bloc PL/SQL

- rem : détails chaque clause à préciser

PL/SQL

- notion d'erreur (hors PL/SQL)
- le langage :
 - partie programmation : exceptions
 - partie BD : gestion des erreurs BD
- triggers

exceptions

exceptions

- But :
 - associer un traitement
 - à des exceptions définies par l'utilisateur
 - survenues lors de l'exécution d'un bloc PL/SQL
- Même principe qu'en Java
- indépendant de la partie BD de PL/SQL
- Exemple :
 - Afficher tous les articles
 - Si l'un dépasse 10 000 euros, le dire et arrêter
 - (possible exemple sans BD)

```
declare cursor c1 is select . . . ; art c1%rowtype;
    depassement exception;

begin
    open c1; fetch c1 into art;
    while c1%found loop
        dbms_output.putline(c1.refart || ' ' || c1.prixht);
        if c1.prixht > 10 000 then raise depassement;
        fetch c1 into art;
    end loop; close c1;

exception
    when depassement then dbms_output.putline('c'est trop');
    -- when others then . . .

end;
```

Exemple

fonctionnement

- si l'exception n'est pas traitée (rattrapée) dans le bloc
- alors elle est transmise (levée) automatiquement dans le contexte appelant :
 - procédure appelante
 - session interactive de l'appel

PL/SQL

- notion d'erreur (hors PL/SQL)
- le langage :
 - partie programmation : exceptions
 - partie BD : curseurs, SQL dynamique, gestion des erreurs BD
- triggers

gestion des erreurs BD

gestion des erreurs BD en PL/SQL

- rappel : erreur BD (hors PL/SQL)
- erreurs BD en PL/SQL
- associations d'exceptions aux erreurs BD
- exemple

rappel : notion d'erreur BD

- il y a erreur quand ordre SQL pas exécuté
- serveur réagit
- environnement appelant averti
- compte-rendu disponible pour l'appelant (ex : message, code)
- ex : mode interactif : réaction serveur = affichage compte-rendu

situation d'erreur en PL/SQL

- c'est quand un ordre SQL d'un bloc est en erreur
- réaction serveur : lève une exception PL/SQL
- compte-rendu : sqlcode, sqlerrm
- conséquence : le programmeur peut la gérer comme toute exception PL/SQL :
 - il peut choisir de la rattrapper ou non
 - s'il la rattrape il peut choisir de :
 - afficher le compte-rendu
 - adapter les prochains ordres à envoyer au serveur

le compte-rendu

- `sqlcode`, `sqlerrm` : les affecter à variables locales pour utilisation dans ordres SQL

association d'exceptions PL/SQL aux erreurs SQL

- une exception PL/SQL doit avoir un nom
- une vingtaine d'erreurs nommées par Oracle :
 - DUP_VAL_ON_INDEX : -1
 - NO_DATA_FOUND : +100 (mode ANSI) (select into ou curseur)
 - TOO_MANY_ROWS : -1422 (select into)
 - etc.

- le programmeur peut nommer l'erreur :
 - identifier le code `c` de l'erreur (dans documentation)
 - associer un nom `n` à ce code :
`pragma_init (n, c)`
 - declare l'exception `n`
 - `n` est une exception normale : elle se traite normalement

Exemple

```
declare cursor c1 is select . . .; art c1%rowtype;

    pasdetable exception;

    pragma exception_init(pasdetable, -942);

begin

    open c1; fetch c1 into art;

    while c1%found loop

        dbms_output.putline(c1.refart || ' ' || c1.prixht);

        fetch c1 into art; end loop; close c1;

exception

    when pasdetable

        then dbms_output.putline('table n'existe pas !');

    when others then dbms_output.putline('autre erreur !');

end;
```

définir ses propres messages d'erreur

- programmeur : lève erreur avec message
- but : prévenir l'application appelante (interactif, PL/SQL, PHP, etc.)
- `raise_application_error(n, msg)`
 - `n` : -20000 à -20999
 - `msg` : chaîne

erreurs BD : résumé

- nommées :
 - Oracle : une vingtaine
 - programmeur : via exception
- non nommées :
 - Oracle : les autres
 - utilisateur : `raise_application_error`

Packages

- but : regrouper des procédures stockées
- Éléments cachés : permet encapsulation (style objet, modules)
- Deux parties :
 - En-tête
 - Corps
- ex : déjà connu : `dbms_output` est un package prédéfini
- Nous : packages créés par le programmeur

create or replace package EWPCK as

 procedure p (... paramètres ...); ...

 function f (...) type;

 exception e ...;

 type t ... ;

end;

create or replace package body EWPCK as

 procedure p (...) begin ... end; ...

 function f (...) begin ... end;

end;

- Appel : comme procédure hors package + nom package : ewpck.p(... paramètres ...)

Confidentialité et procédures stockées

- procédure stockée est un objet de la base
- appel d'une procédure stockée est une « action »
- autorisation = (utilisateur, exécuter, procédure)
- grant execute

on p

to toto

- rem : le code de p accède à des tables : c'est le propriétaire de la procédure qui doit avoir les droits (ex : encapsulation) (il existe cas général)

indépendance des niveaux et procédures stockées

- procédures peuvent être considérées niveau externes par rapport aux tables
- doivent respecter indépendance par rapport niveau inférieur : utiliser vues
- (mises à jour vues hors programme => pas de vues si mises à jour)

délégués ?