

La gestion des erreurs


Les exceptions

Principes

- En PL/SQL, les cas d'erreurs sont gérés par un mécanisme ***d'EXCEPTIONS***
- Lorsqu'une erreur d'exécution est détectée (erreur sql, données absentes ...), une exception est générée, puis elle est traitée dans une partie séparée du programme :

```
DECLARE
BEGIN
... Instructions
< ERREUR >

EXCEPTION
    WHEN nom_exception THEN Instructions _pl/sql
END ;
```

A red curved arrow originates from the text '< ERREUR >' and points towards the 'EXCEPTION' block, indicating the flow of control when an error occurs.

Principes

- Chaque nom d'exception correspond à un type d'erreur particulier
- Le langage possède des exceptions prédéfinies correspondant à des cas d'erreurs d'exécution,
- Le langage permet au programmeur de définir ses propres exceptions correspondant à des erreurs de l'application

Exceptions prédéfinies

- **DUP_VAL_ON_INDEX** : Valeur dupliquée pour un index unique
 - Exemple : insertion d'une clé primaire déjà existante
- **NO_DATA_FOUND** : Select ne retourne aucune ligne
- **TOO_MANY_ROWS** : Select simple renvoie plus d'une ligne dans un
`Select .. Into`
- **INVALID_NUMBER** : nombre invalide
- **VALUE_ERROR** : Erreur de troncature ou de conversion
- **OTHERS** : les autres erreurs

DECLARE

vemp emp%rowtype ;

vid emp.id%type;

BEGIN

vid := :id-employe ;

select sal into vemp from emp where emp.id = vid;

IF vemp.sal > 5000 then insert into gros_salaire
values (vemp.id, vemp.nom, vemp.sal)

END IF;

EXCEPTION

when no_data_found then .. -- cas select retourne 0 lignes

when too_many_rows then .. -- cas select retourne + lignes

when dup_val_on_index then .. -- cas insert emp. Déjà inséré

when others then .. -- gérer les autres erreurs

END ;

Autres cas d'erreurs d'exécution

- Les exceptions prédéfinies ne couvrent pas toutes les erreurs d'exécution, notamment les erreurs SQL
- Il faut les traiter avec l'exception OTHERS et en utilisant 2 variables prédéfinies :
 - SQLCODE : code de la plus récente erreur Oracle
 - SQLERRM : libellé de la plus récente erreur Oracle

```
...  
EXCEPTION  
when others then  
    dbms_output.put_line('erreur: ' || SQLCODE || SQLERRM);  
END ;
```

Erreurs applicatives

- Cas des erreurs qui ne sont pas des erreurs d'exécution mais des erreurs liées à la logique de l'application
- Exemple : employé avec un salaire = 0
- Ces erreurs peuvent être traitées avec des exceptions :
 - L'exception doit être déclarée,
 - L'exception doit être explicitement déclenchée lorsque le cas d'erreur est détecté,
 - L'exception doit être traitée dans la partie EXCEPTION

DECLARE

vemp emp%rowtype ;

vid emp.id%type;

salaire_zero EXCEPTION;

BEGIN

vid := :id-employe ;

select sal into vemp from emp where emp.id = vid;

IF vemp.sal = 0 then

RAISE salaire_zero ;

END IF;

EXCEPTION

when salaire_zero then

dbms_output.put_line('erreur : bénévole détecté') ;

END ;

Les séquences Oracle

- Une séquence est un objet stocké dans la base de donnée permettant de générer une suite (séquence) de valeurs entières successives
- On utilise une séquence pour générer des identifiants uniques, souvent utilisés comme clé primaire
- exemple : IDFILM, IDACTEUR

Créer une séquence :

```
CREATE SEQUENCE SEQ_IDFILM START WITH 30 INCREMENT by 1 ;
```

Utiliser une séquence :

```
insert into FILM (idfilm, titre, genre )  
values ( SEQ_IDFILM.NEXTVAL,  
        'le retour de la vengeance masquée 2' ,  
        'serie Z' ) ;
```

```
select SEQ_IDFILM.NEXTVAL from DUAL ;  
select SEQ_IDFILM.CURRVAL from DUAL ;
```

/**

*** NOTES :**

* CURRVAL n'est utilisable qu'après au moins 1 appel NEXTVAL

* NEXTVAL ne génère qu'une seule valeur par ligne : insert into T values (S1.netxval, S1.nextval) ;

* produira la même valeur dans les 2 colonnes de la table T

*/

Fonctions et Procédures (stockées)

- PL/SQL permet de définir des fonctions et des procédures
- Une **procédure** est un bloc PL/SQL **nommé** et pouvant recevoir des **paramètres**
- Une **fonction** est un bloc PL/SQL **nommé**, pouvant recevoir des **paramètres** et retournant un **résultat** d'un certain type
- Procédures et fonctions permettent de définir des modules ré-utilisables

Example

DECLARE

FUNCTION **specialite**(p_nomski char) RETURN varchar2 IS

v_special varchar2(25) ;

begin

select specialite into v_special from skieur where nomski=p_nomski;

return v_special ;

Exception

when NO_DATA_FOUND then **return** 'nom inconnu' ;

END **specialite**;

BEGIN

dbms_output.put_line('specialite: '|| **specialite**('worley'));

END ;

Fonctions et procédures stockées

- Il existe 2 formes de procédures et fonctions :
- Les **procédures et fonctions déclarées dans un script PL/SQL** client : elles ne sont utilisables que dans ce script
 - Intérêt et utilisation limités
- Les **procédures et les fonctions stockées** sur le serveur : elles sont utilisables à partir d'un script quelconque, d'une autre fonction/procédure stocké, ou d'un programme applicatif écrit dans un autre langage
 - Elles permettent de programmer des fonctionnalités de gestion des données réutilisables dans différents contextes et dans différents programmes et applications

Déclaration d'une procédure stockée

```
CREATE [ OR REPLACE ] PROCEDURE nom_proc [ (param1..[,paramN]) ]  
IS  
    [declaration_variables_locales]  
BEGIN  
    --instructions exécutables  
    [EXCEPTION]  
    --gérer les exceptions  
END [nom_proc] ;
```

- La commande CREATE PROCEDURE déclenche la compilation et le stockage de la procédure sur le serveur
- La procédure peut être supprimée : DROP PROCEDURE nom_proc;

Paramètres d'une procédure

- Une procédure peut recevoir des paramètres ; chaque paramètre est spécifié par :
- Un **nom**, qui permet de l'utiliser dans le code de la procédure,
- Un **usage** : IN, OUT, IN OUT
 - IN : **paramètre en entrée** : reçoit une valeur à l'appel et ne peut pas être modifié par la procédure
 - OUT : **paramètre en sortie** : la procédure valorise le paramètre pour renvoyer une valeur au programme appelant,
 - IN OUT : **paramètre en entrée et sortie** : la procédure reçoit une valeur qu'elle peut modifier pour la retourner au programme appelant
- Un **type** PL/SQL dans lequel on ne précise pas sa taille
 - VARCHAR, NUMBER au lieu de VARCHAR(64), NUMBER(6)

CREATE OR REPLACE

PROCEDURE classt_extremes(p_nomski char, p_min out number,
p_max out number) IS

begin

Select min(rang), max(rang) into p_min,p_max from classement
where nomski=p_nomski ;

END classt_extremes ;

DECLARE

v_min number(3); v_max number(3);

v_nom skieur.nomski%type := :nom_skieur ;

BEGIN

classt_extremes (v_nom, v_min, v_max);

dbms_output.put_line(classt mini : ' || v_min) ;

dbms_output.put_line(classt maxi : ' || v_max) ;

END ;

Utilisation et programmation des procédures

- Les procédures stockées permettent de programmer des **fonctionnalités** de gestion de données **réutilisables** dans différentes applications ou différents modules d'un système applicatif
- Elles sont utiles pour toutes les fonctionnalités nécessitant d'enchaîner plusieurs requêtes SQL
- Pour faciliter leur réutilisation, elles doivent **communiquer uniquement aux travers des paramètres**
- Sauf en phase de mise au point, il faut **éviter d'utiliser les instructions d'entrée/sortie** :
 - Variables de substitutions : totalement inutiles, puisqu'elles ne sont pas connues du serveur
 - `Dbms_output.put_line()` : produit des affichages qui sont rarement pertinents dans tous les contextes

Déclaration d'une fonction stockée

- Une fonction retourne un résultat dont on doit préciser le type

```
CREATE [ OR REPLACE] FUNCTION nom_func [ (param1,...[,paramN]) ]  
    RETURN TYPE_RESULTAT  
IS  
    [declaration_variables_locales]  
BEGIN  
    --instructions exécutables  
    [EXCEPTION]  
    --gérer les exceptions  
END [nom_func] ;
```

Exemple

```
CREATE OR REPLACE
FUNCTION nb_courses(p_nomski char) RETURN Number
IS
nb number(4);
begin
  select count(*) into nb  from classement
  where nomski=p_nomski ;
  return nb ;
END nb_courses ;
```

Utilisation

```
DECLARE
```

```
v_course number(3) ;
```

```
Cursor skieurs( nmin NUMBER ) is
```

```
select * from skieurs where nb_courses(nomski) > nmin
```

```
BEGIN
```

```
v_courses := nb_courses( 'worley' );
```

```
select nb_courses( nomski ) into v_courses  
from skieur where nomski = 'schifftrin' ;
```

```
END ;
```

Programmation et utilisation des fonctions

- Une fonction stockée doit être programmée de manière à pouvoir être utilisée comme une fonction oracle dans une requête SQL
- Elles permettent de programmer des fonctionnalités spécifiques à une application (des fonctions métier) utilisables dans une requête SQL : **c'est une façon d'étendre SQL**
- Une fonction ne doit jamais avoir d'effets autres que celui de calculer son résultat : **pas d'effet de bord**
- Sauf en phase de mise au point, il faut **éviter d'utiliser les instructions d'entrée/sortie** :