PL/SQL : Stockage des traitements - 2

Procédures/Fonctions - Curseurs - Exceptions

Est-ce bien à moi de faire tout cela ?

Sommaire

Procédures et Fonctions

Procédures et Fonctions

- Déporter les traitements de manipulation de données vers le SGBD.
- Ensemble d'instructions procédurales et de requètes SQL.
- On les dit procédures ou fonction **stockées**.
- Traitement identique et commun à tous les applicatifs.
- Charge le SGBD.

Procédures et Fonctions

- Déclaration

```
CREATE [OR REPLACE] PROCEDURE nom_procédure (paramètres)
IS
    <bloc de variables>
BEGIN
    <bloc de traitements>
END;

CREATE [OR REPLACE] FUNCTION nom_fonction (paramètres)
RETURN type
IS
    <bloc de variables>
BEGIN
    <bloc de traitements>
    RETURN <result>;
END;
```

Procédures et Fonctions

- Paramètres

```
CREATE [OR REPLACE] PROCEDURE nom_procédure (
    p_1 IN <type1> DEFAULT <valeur>,
    p_2 OUT <type2>,
    p_3 IN OUT <type3>)
IS
    <bloc de variables>
BEGIN
    <bloc de traitements>
END;
```

- **ATTENTION** les types sont spécifiés sans leur taille !

```
CREATE OR REPLACE PROCEDURE ma_proc (
    p_1 IN NUMBER,
    p_2 IN VARCHAR2)
```

Procédures et Fonctions

- Variables, typage

```
CREATE [OR REPLACE] PROCEDURE nom_procédure ( <paramètres> )
IS
    v_1 NUMBER;
    v_2 <table>.<colonne>%type;
    v_3 <table>%rowtype;
    TYPE t_rec_nom IS RECORD (
        <att1 type1>[,
        <att2 type2>]);
    v_rec_1  t_rec_nom;

BEGIN
    <bloc de traitements>
END;
```

Procédures et Fonctions

- Dans un bloc de traitement PL/SQL :

```
BEGIN
    v_res := <f_nom>(<params>);
    ...
    <proc_nom>(<params>);
END;
```

- Depuis sqlplus :

```
SQL> SELECT <f_nom>(<params>) FROM DUAL;
SQL> EXEC <proc_nom>(<params>);
```

Procédures et Fonctions

- Quelques fonctions natives du langage :
  - chaînes : concat (| |), trim, length, substr, ...
  - numériques : ceil, floor, round, trunc, sqrt, avg, max, ...
  - date : to_date, to_char, day, month, year, next_day, ...
  - transformation : decode, coalesce, nvl, ...
- Il en existe bien plus, soyez curieux !

Curseurs

Curseurs

- Objet permettant de manipuler un ensemble de résultats d'une requète **SELECT**.
- Contient les données de la requète.
- 2 types de déclaration :
  - implicite : SELECT ... INTO ... FROM ...;
  - explicite : CURSOR IS ... ;

Curseurs implicites

- Exemples

```
BEGIN
     v_num := 100016;

     r_req := ' SELECT * ' ||
              ' FROM   users ' ||
              ' WHERE  uso_num   = :p_mun ';
     EXECUTE IMMEDIATE r_req INTO v_res USING v_num;

    SELECT nom, prenom INTO v_nom, v_prenom
    FROM clients
    WHERE id = p_id_client ;
END;
```

Curseurs

- Déclaration (**dans le bloc variables**)

```
CREATE [OR REPLACE] PROCEDURE nom_procédure
IS
    CURSOR <cur_nom> IS
    SELECT ... FROM ... ;
BEGIN
    <bloc de traitements>
END;
```

Curseurs

- Ouverture / Parcours / Fermuture (**dans le bloc traitements**)

```
CREATE [OR REPLACE] PROCEDURE nom_procédure
IS
    CURSOR <cur_nom> IS
    SELECT * FROM table ;
    v_item <cur_nom>%rowtype;
BEGIN
    OPEN <cur_nom>; -- ouverture
    LOOP            -- parcours
    FETCH <cur_nom> INTO v_item ;
    EXIT WHEN <cur_nom>%NOTFOUND; -- fin du parcours
        <traitements>
    END LOOP ;
    CLOSE <cur_nom> ; -- fermeture !
    <traitements>
END;
```

Curseurs

- Pseudos attributs pour connaître l'état d'un curseur
  - %FOUND : vrai si un enregistrement dans le curseur;
  - %NOTFOUND : vrai si aucun enregistrement dans le curseur;
  - %ISOPEN : vrai si le curseur est ouvert;
  - %ROWCOUNT : donne le nombre de FETCH en cours.

Curseurs

- Ouverture / Parcours / Fermuture automatiques (**dans le bloc traitements**)

```
CREATE [OR REPLACE] PROCEDURE nom_procédure
IS
    CURSOR <cur_nom> IS
    SELECT ... FROM ... ;
BEGIN
    FOR <v_ligne> IN <cur_nom>
    LOOP
        <traitements>
    END LOOP ;
    <traitements>
END;
```

Curseurs

- Ouverture / Parcours / Fermuture curseur implicite

```
CREATE [OR REPLACE] PROCEDURE nom_procédure (
IS
    <bloc variables>
BEGIN
    FOR <v_ligne> IN (SELECT ... FROM ...)
    LOOP
        <traitements>
    END LOOP ;
    <traitements>
END;
```

Curseurs

- Il est possible de paramétrer un curseur :

```
CREATE [OR REPLACE] PROCEDURE nom_procédure
IS
    CURSOR <cur_nom>(p_num NUMBER) IS
    SELECT * FROM table
    WHERE no = p_num ;
BEGIN
    ...
    OPEN <cur_nom>(12);
END;
```

Transactions

Transactions

- Permettent d'isoler un ensemble de modifications sur les données;
- Vérifient un ensemble de propriétés **ACID** :
  - Atomique : si une échoue, on annule l'ensemble des modifications;
  - Consistente : la transaction prend et rend la BDD dans un état consistent;
  - Isolée : l'effet d'une transaction n'est visible qu'après une validation explicite;
  - Durable : les modifications d'une transaction sont permanentes;

Transactions

- 4 composantes principales:
  - SAVEPOINT : permet de poser des points de sauvegarde;
  - COMMIT : permet de valider une transaction;
  - ROLLBACK : permet d'annuler une transaction. L'état revient au dernier état stable (validation ou point de sauvegarde) connu;
  - SET TRANSACTION [READ ONLY|READ WRITE] [NAME 'tr_nom']; : nomme une transaction,

```
BEGIN
    INSERT INTO ...;
    INSERT INTO ...;
    INSERT INTO ...;
    INSERT INTO ...;
    ROLLBACK; -- Annule les 4 commandes précédentes
    INSERT INTO ...;
    INSERT INTO ...;
    SAVEPOINT icionsauve;
    INSERT INTO ...;
    INSERT INTO ...;
    ROLLBACK TO icionsauve; -- Annule les 2 dernières commandes

END;
```

Transactions

```
BEGIN
    SET TRANSACTION NAME 'tr_nom'; --Définit un début de transaction
    DELETE FROM
    INSERT INTO ...;
    UPDATE ...;
    UPDATE ...;
    COMMIT; -- Sauvegarde les modifications de la transaction 'tr_nom'

END;
```

Transactions

- Une procédure stockée/fonction **devrait** ne pas réaliser de validation, charge au code appelant ;
- Possibilité de créer une transaction autonome dans la procédure/fonction/trigger **PRAGMA AUTONOMOUS_TRANSACTION** :

```
CREATE [OR REPLACE] PROCEDURE nom_procédure
IS
    PRAGMA AUTONOMOUS_TRANSACTION;
    <bloc variables>
BEGIN
    <traitements>
    COMMIT;
END;
```

Gestion des exceptions

Exceptions

- Bloc PL/SQL :

```
BEGIN
    <traitements à réaliser>
    EXCEPTION
    <traitements de gestion des erreurs>
END;
```

- Gestion multiple :

```
BEGIN
    <traitements à réaliser>
    EXCEPTION
        WHEN <ex1> THEN ...
        WHEN <ex2> THEN ...
        WHEN OTHERS THEN ...
END;
```

Exceptions

- Les classiques :
  - DUP_VAL_ON_INDEX (ORA-00001) : doublon clé primaire;
  - NO_DATA_FOUND (ORA-01403) : ...;
  - TOO_MANY_ROWS (ORA-01422) : soit il faut utiliser un curseur, soit on a un pb de restriction...
  - et plein d'autres ! (ZERO_DIVIDE, INVALID_NUMBER, EXCEPTION_PERSO, ...).
- Fonction récupérant les codes et messages d'erreur :
  - SQLCODE : retourne le code d'erreur (ORA-xxxx);
  - SQLERRM : retourne le message d'erreur explicite.

Exceptions

- Bloc PL/SQL :

```
BEGIN
    INSERT INTO ...;
    EXCEPTION
    WHEN DUP_VAL_ON_INDEX THEN
       DBMS_OUTPUT.PUT_LINE('Erreur no '||SQLCODE||' - '||SQLERRM) ;
        UPDATE ...;
END;
```

Exceptions

- Créer une exception :
  - Déclarer EXCEPTION
  - Associer un code et un message :
  - PRAGMA EXCEPTION_INIT(nom, code);
- Lever une erreur ponctuelle :
  - RAISE_APPLICATION_ERROR (code, message);
  - code [-21000,-20000];

```
CREATE OR REPLACE PROCEDURE verifier_age(p_age NUMBER)
IS
   e_age_invalide EXCEPTION;
   PRAGMA EXCEPTION_INIT(e_age_invalide, -20001);

BEGIN
   p_age < 18 THEN
      RAISE e_age_invalide;
   END IF;

   DBMS_OUTPUT.PUT_LINE('Âge valide');

EXCEPTION
   WHEN e_age_invalide THEN
      DBMS_OUTPUT.PUT_LINE('Erreur dans la procédure : âge invalide');
END;
```

Exceptions

- Exemple

```
c_num_monexc CONSTANT NUMBER := -20042;
ex_monexception EXCEPTION;
PRAGMA EXCEPTION_INIT(ex_monexception,-20042);

-- procédure levant notre exception
BEGIN
    ...
    EXCEPTION
    WHEN OTHERS THEN
       RAISE_APPLICATION_ERROR(c_num_monexc, 'Erreur id :'||p_id) ;
END;

....

-- procédure appelante gérant notre exception
BEGIN
    ...
    EXCEPTION
    WHEN ex_monexception THEN
       DBMS_OUTPUT.PUT_LINE('Je gère...') ;
END;
```

Packages

Packages

- Regroupe des fonctionnalités, traitements au sein d'une même entité
- Concerne les :
  - procédures / fonctions;
  - exceptions;
  - curseurs;
  - variables / constantes.

Packages

- 2 parties :
  - Déclaration : expose certaines des fonctionnalités à une visibilité publique;
  - Description : contient les définitions (code) de l'ensemble des fonctionnalités (d'exposition publique et privée).

Packages

- Déclaration :

```
CREATE [OR REPLACE] PACKAGE <nom_pkg>
[IS|AS]
    |[déclaration des variables/constantes ;]
    |[déclaration des excepions ;]
    {[déclaration de procédure ;]
    |[déclaration de fonctions ;]
    |...}
END <nom_pkg>;
```

Packages

- Description :

```
CREATE [OR REPLACE] PACKAGE BODY
<nom_pkg>
[IS|AS]
    |[déclaration de variable ;]
    {[déclaration et corps de procédure ;]
    |[déclaration et corps de fonctions ;]
    |...}
END <nom_pkg>;
```

Packages

- Exemple :

```
CREATE OR REPLACE PACKAGE <pkg_nom> IS
    -- Exemple de constante
    c_PI        CONSTANT   <table>.<col>%TYPE := 3.14159;
    -- Exemple variable globale
    vg_user     VARCHAR(10);
    PROCEDURE logue(p_str IN VARCHAR2(200));
    PROCEDURE init;
    FUNCTION rech(p_id IN user.id%TYPE) RETURN user.usr_no%TYPE;
END <nom_package>;
/
CREATE OR REPLACE PACKAGE BODY <pkg_nom> IS
    PROCEDURE logue(p_str IN VARCHAR2(200)) IS
    <variables>
    BEGIN
    ...
    END logue;
    PRODECURE init IS
    ...
    END init;
END <pkg_nom>;
/
```

À vous de jouer !