Le Langage PL/SQL de Oracle (Brève Introduction)

Najib Tounsi
Ecole Mohammadia d'Ingénieurs, Rabat

Année 2010/2011
2e année Génie Informatique http://www.emi.ac.ma/~ntounsi/COURS/DB/Polys/SQL/PLSQL/PLSQL.html

Sommaire

Introduction

- *PL/SQL* est un langage procédural (*Procedural Language/SQL*) qui permet en plus de SQL d'avoir les mécanismes des langages algorithmiques.
- L'unité de programmation PL/SQL est le *bloc*.
- Un bloc contient des déclaration de variables ou de constantes et des instructions (qui peuvent être des commandes SQL).
- Des variables peuvent être utilisées pour mémoriser des résultats de requêtes.

Introduction (suite)

- Parmi les instructions d'un bloc PL/SQL on a:
  - commandes SQL donc,
  - instructions de boucle (**loop**),
  - instructions conditionnelles (**if-then-else**),
  - traitement des exceptions,
  - appels à d'autres blocs PL/SQL.
- Un bloc PL/SQL peut-être une fonction (ou une procédure).

Introduction (suite)

- Les blocs PL/SQL qui définissent des fonctions ou procédures, peuvent être groupés en packages.
- Un package est comme un module, il a une partie interface et une partie implémentation.
- PL/SQL permet de traiter le résultat d'une requête *tuple par tuple*.
- La notion de **CURSOR** sert à cet effet.

Structure des blocs PL/SQL

- Un bloc est l'unité de programme en PL/SQL.
- Un bloc a un nom quand il définit une fonction, une procédure ou un package.
- Les blocs peuvent être imbriqués.
- Un Bloc a:
  - une partie déclarations (optionnelle),
  - une partie instructions,
  - et une partie (optionnelle) de traitement d'exceptions.

Structure des blocs PL/SQL (suite)

- Structure d'un bloc PL/SQL

```
[<Enête de bloc>]
[declare
        <Constantes>
        <Variables>
        <Cursors>
        <Exceptions utilisateurs>]
begin
        <Instruction PL/SQL>
        [exception <Traitement d'exception>]
end;
```

- Entête de bloc: si fonction, procédure ou package. Sinon, bloc anonyme.

Déclarations de variables

- Exemple de déclarations

```
declare
        dateEmbauche date; /* initialisation implicite with null */
        nom varchar2(80) := 'Benali';
        trouve boolean; /* initialisation implicite with null */
        incrSalaire constant number(3,2) := 1.5; /* constante */
        . . .
begin . . . end;
```

- *null* est valeur par défaut, si pas d'initialisation (sauf si `NOT NULL` spécifié)
- Types usuels de ORACLE.

- Usage du type de donné par référence à une colonne.

```
declare numero Employee.enum%TYPE ;
```

- Utile pour garantir la compatibilité des affectations (e.g. dans clause INTO)

Déclarations de variables ou de curseurs (suite)

- On peut déclarer une structure de tuple, par référence au schéma d'une relation.

```
declare emp Employee%ROWTYPE ;
```

- Le mot *cursor* sert à déclarer un "curseur" qui recevra le résultat d'un *select* pour un parcours tuple par tuple.

```
declare cursor empCurseur is
select * from EMPLOYEE where DEPT = 123;
```

- ou bien avec paramètres formel

```
declare cursor empCurseur (dno number ) is
select * from EMPLOYEE where DEPT = dno;
```

- Le paramètre *dno* sera passé lors de open.

```
open empCurseur (123);
```

- Si un curseur est utilisé pour mettre à jour un tuple de ralation, on le signale avec *for update* en fin de déclaration

```
declare cursor empCurseur is
select * EMPLOYEE where DEPT = 123
for update (SAL);
```

-

Instructions

- PL/SQL offre la plupart des constructions des langages de programmation: affectation de variables, structures de contrôle de boucle (*loop*) et de teste (*if-then-else*), appels de procédure et de fonction, etc.

```
declare
        quantite integer := 0;
        . . .
begin
        quantite := quantite + 5;
end
```

- PL/SQL ne permet pas les commandes SQL langage de définition comme la création de table
- PL/SQL permet tous les autres types de comandes SQL (*insert*, *delete*, *update*,

commit ...)
- La commande *select* (sauf si imbriqué) est toujours utilisé avec *into*, pour affecter les attributs retrouvés aux variables PL/SQL.

Instructions (suite)

- PL/SQL permet d'affecter chaque tuple réultat d'un select à une structure (record) ou à ne liste de variable (cf. *Select ... into ...*)

```
declare
 employeeRec employee%ROWTYPE;
 maxSal employee.SAL%TYPE;
begin
 select EMPNO, ENAME, JOB, MGR, SAL, COMM, HIREDATE, DEPTNO
 into employeeRec
 from EMPLOYEE where EMPNO = 5698;
 ...
 select max(SAL) into maxSal from EMPLOYEE;
 ...
end;
```

- Dans *into* on peut utiliser une liste de variables (comme ESQL/C) ou un record.

Structure de contrôle *if-then-else*

- Sémantique analogue aux autres langages

```
if <condition> then <sequence d'instructions>
[elsif ] <condition> then <sequence d'instructions>
. . .
[else] <sequence of statements>
end if ;
```

- Usage de `elsif` pour suite de tests et `endif` pour finir le `if`.

Structures de contrôles (boucle *loop*)

- Boucle *while*

```
[<label name>]
while <condition> loop
        <sequence of statements>;
end loop [<label name>] ;
```

- On peut nommer une boucle pour, en cas de boucles imbriquées, s'y référer avec *exit* par exemple.

Structures de contrôles (boucle *loop*)

- Boucle *for*

```
[<label name>]
```

```
for <index> in [reverse] <lower bound>..<upper bound> loop
        <sequence of statements> ;
end loop [<label name>] ;
```

- L'index est déclaré implicitement.

Structures de contrôles (boucle *loop*)

- Une autre forme de boucle (infinie) est

```
loop ... end loop;
```

- arrêt avec

```
exit when ...
```

- Utilisée surtout avec curseur.

Structures de contrôles (boucle *loop* avec *cursor*)

- La forme *loop ... end loop*, est utilisée avec un curseur.

```
declare
        cursor empCurseur is select * from EMPLOYEE;
        employeeRec employee%ROWTYPE;
        maxSal employee.SAL%TYPE;
begin
        open empCurseur;
        loop
        fetch empCurseur into employeeRec;
        exit when empCurseur%NOTFOUND;
        maxSal := employeeRec.sal;
        ...
        end loop;
        close empCurseur;
        . . .
end;
```

- Après *open*, `emp_cur%NOTFOUND` est évaluée à *null*. Après un `fetch` elle est évaluée à *false* si un tuple est retrouvé, *true* sinon.
- La boucle finit donc dès que la condition `exit when` est vérifiée (aucun tuple retrouvé par `fetch`).

Structures de contrôles (boucle *for (each)*)

- Une autre forme plus simple de parcourir un curseur.

```
[<label name> ]
for <record name> in <cursor name>[(<list of parameters>)] loop
        <sequence of statements>
```

```
end loop [<label name>];
```

- La boucle est exécutée **pour chaque** tuple dans le cursor (mécanisme d'itérateur abstrait):
  - La variable de contrôle record name est implicitement déclarée du type du corsor.
  - Cette boucle exécute un fetch à chaque itération (A chaque itération, un seul tuple est retrouvé.)
  - Ce for exécute aussi un open avant d'entrer en boucle et un close en fon de boucle.
  - La boucle se termine automatiquement (sans exit) dès qu'aucun tuple n'est trouvé.

Structures de contrôles (variante de *for (each)*)

- Requête directe au lieu de curseur

```
for <record name> in (<select statement>) loop
        <sequence of statements>
end loop;
```

- Exemple:

```
for salRec in (select SAL * 1.07 nouveau from EMP) loop
        ... ;
end loop;
```

- *nouveau* est un alias pour l'expression calculée.
- A chaque itération, un tuple est retrouvé. La variable *salRec*, implicitement déclarée, reçoit le résultat accessible par *salRec.nouveau*.

Cursor avec mise à jour

- Les commandes SQL *update* et *delete* peuvent être utilisés avec un curseur (déclaré avec la clause *with update of*)
- Elles affectent alors seulement le tuple courant de *fetch*.
- La clause *where current of* curseur, est alorsajoutée à la commande

```
declare
        cursor empCureur is select SAL from EMPLOYEE
        where DEPT = 123 for update of SAL;
begin
        for empRec in empCureur loop
        update EMPLOYEE set SAL = empRec.sal * 1.05
        where current of empCureur ;
        end loop;
        commit;
end;
```

- pour augmenter de 5% les salaires des employés du département 123

Traitement d'Exceptions

- Une erreur ou avertissement PL+SQL peut survenir en cours d'exécution et soulever une exception.
- Une exception peut être predéfinie par le système ou déclarée par l'utilisateur.
- Le traitement d'une exception se fait par la règle *when*

```
when <nom d'exception> then <sequence d'instructions>;
```

- où la séquence d'instructions est exécuté quand l'exception donnée est soulevée.

Traitement d'Exceptions (suite)

- Les exeptions systèmes sont automatiquement soulevées lors de l'apparition de l'erreur ou de l'avertissement correspondant.
- Exemples d'exceptions système.

```
CURSOR_ALREADY_OPEN: tentative d'ouverture de curseur déjà ouvert
INVALID_CURSOR: par exemple fetch sur un curseur déjà fermé
NO_DATA_FOUND: aucun tuple retourné (select into ou fetch)
TOO_MANY_ROWS: select into retourne plus d'un tuple
...
ZERO_DIVIDE: tentative de division par zéro
```

- Exemple d'usage

```
when NO DATA FOUND then rollback;
```

- Les exception utilisateurs sont soulevées par *raise*

```
raise <nom d'exception>
```

Exemple complet

- Augmenter de 5% les salaires des employés du département '123' sans toutefois dépasser 4000

```
declare
        empSal EMP.SAL%TYPE;
        empNo EMP.EMPNO%TYPE;
        tropGrand exception;
begin
        select EMPNO, SAL into empNo, empSal
        from EMPLOYEE where DEPT = '123';
        if empSal * 1.05 > 4000 then raise tropGrand
        else update EMPLOYEE set . . .
        end if ;
exception
        when NO DATA FOUND then rollback;
        when tropgrand then insert into RICHES values(empNO, empSal);
        commit;
end;
```

- Les traitements d'exceptions sont définis à la fin du bloc instructions par la clause

`exception`.
- La variable `tropGrand` est déclarée de type `exception`, pour être utilisée dans `raise`.

En savoir plus

- http://en.wikipedia.org/wiki/PL/SQL
- www.mathcs.emory.edu/~cheung/Courses/377/Others/tutorial.pdf
- http://www.plsql-tutorial.com/
- http://www.plsqltutorial.com/
- http://www.java2s.com/Tutorial/Oracle/CatalogOracle.htm
- http://infolab.stanford.edu/~ullman/fcdb/oracle/or-plsql.html (1998)