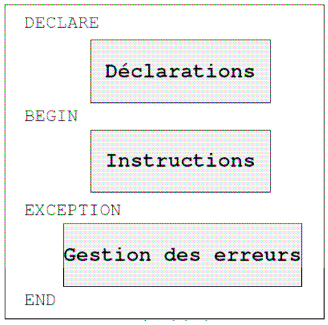
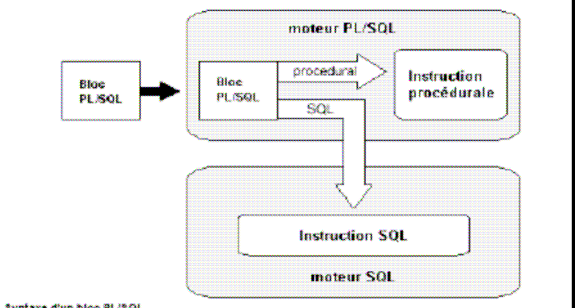
Extension procédurale dans Oracle

PL/SQL version 10g

Hala Skaf-Molli
Université Henri Poincaré, Nancy1
skaf@loria.fr
www.loria.fr/~skaf

Structure d'un Bloc d'instructions PL/SQL





Syntaxe d'un bloc PL/SQL

Exemple

```
SQL >
begin
delete  produit where prod_id ='p1';
end;
```

PL/SQL procedure successfully completed.

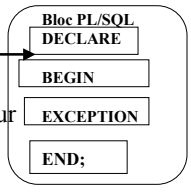
Sortie à l'écran

```
> SQL
set serveroutput on
begin
dbms_output.put_line('Bonjour utilisateur id :  '|| UID  || 'nom est:  ' ||
    USER || 'on est:  ' || to_char(sysdate,'dd month yyyy'));
end;
```

- Bonjour utilisateur id : 69nom est: SKAFon est: 11 january 2006
  PL/SQL procedure successfully completed.

Déclarations

- Variables et constantes
  - Types de base
  - types définis par l'utilisateur
- Curseurs
- Exceptions



- Pas de différences entre minuscules et majuscules pour les identifiant

Types de base

<type>::= CHAR | BOOLEAN | DATE | NUMBER(t,d) | VARCHAR2(s)

avec

- **t** nombre total de chiffres
- **d** nombre de chiffres après la virgule
- **s** nombre maximum de caractères dans la chaîne

7

Syntaxe déclaration

- Nom_variable [CONSTANTE] TYPE {NOT NULL}[{DEFAULT | :=} VALEUR];

```
SQL> set serveroutput on
declare
User_id number := UID;
nom Varchar2(30) :=USER;
trouve boolean;
begin
dbms_output.put_line('Bonjour utilisateur id: '|| User_id || 'nom est:' || nom || 'on est' || to_char(sysdate,'dd month yyyy'));
end;

Bonjour utilisateur id: 69nom est:SKAFon est11 january 2006
PL/SQL procedure successfully completed.
```

8

Exemple

```
SQL>
declare
tcon constant number := 5;
begin
dbms_output.put_line('CONSTANT: ' || tcon );
end;

CONSTANT: 5
PL/SQL procedure successfully completed.
```

9

Exemple

```
SQL> declare
tcon constant number := 5;
begin
tcon:= 10;
dbms_output.put_line('CONSTANT: ' || tcon );
end;
ERROR at line 4:
ORA-06550: line 4, column 1:
PLS-00363: expression 'TCON' cannot be used as an assignment target
ORA-06550: line 4, column 1:
PL/SQL: Statement ignored
```

10

Variables basées

- Faire référence à une entité existante
  - %TYPE: référence une colonne d'une table ou une variable

```
Nom-variable {nom_table.colonne |nom_varaible}%TYPE
Exemple:
Sql> Declare
        date_commande Commande.date_com%TYPE;
…..
```

11

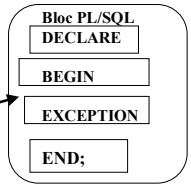
Variables basées

%ROWTYPE:

```
Nom_variable {nom_table |nom-varaible}%ROWTYPE;
Sql> declare
        unProduit Produit%ROWTYPE;
Begin
    unProduit.prod_id := 'p1';
    unProduit.nom := 'cryon';
    unProduit.pu := 200;
dbms_output.put_line(unProduit.prod_id || unProduit.nom || unProduit.pu);
End;
p1cryon200
PL/SQL procedure successfully completed.
```

12

Instructions



- Requête SQL
- Affectation
  - Deux façons
- Instruction de contrôle conditionnel
- Instruction de contrôle itératif
- Gestion des curseurs

Syntaxe

- Les **instructions** sont séparées par des ;
- Les commentaires se mettent entre **/*** mes commentaires ***/**
ou après **--** mes commentaires
- **Affectation** : := …. Ou bien **select into** ..
- Exemple:

```
declare
unProduit Produit%ROWTYPE;  /*Une variable */
Begin
Select * into unProduit from produit where prod_id ='p2';
dbms_output.put_line(unProduit.prod_id || unProduit.nom || unProduit.pu);
End;
```

```
p2toto200
    PL/SQL procedure successfully completed.
```


Conditionnelle

if condition **then** instructions;
[elseif condition **then** instructions; ..]
[else
instructions; ..]
END IF;

Expressions de comparaison

- AND, OR, NOT
- = (égalité), <, >, <=, >=, !=(différent)
- is NULL

Exemples conditionnelles

```
Sql > begin
if  MOD(UID,2) <> 0  then
dbms_output.put_line(UID || 'est un nombre impaire');
end if;
end;
69est un nombre impaire
    PL/SQL procedure successfully completed.
```

```
Sql>
begin
if  MOD(UID,2) <> 0  then
dbms_output.put_line(UID || 'est un nombre impaire');
else
dbms_output.put_line(UID || 'est un nombre pair');
end if;
end;
```


Traitements itératifs

- Loop
- while loop
- For loop

LOOP

[<<NOM_BOUCLE>>]
LOOP <instructions> END LOOP ;

Sql>
```
declare
 x number := 0;
Begin
loop
x:=x+1;
end loop;
End;
```

- Boucle infinie !

19

LOOP

[<<NOM_BOUCLE>>]
LOOP
  <instructions>
EXIT [<<NOM_BOUCLE>>] WHEN
  <condition>;
<instructions>
END LOOP ;

20

```
declare
 x number := 0;
Begin
<<incrt_boucle>>
Loop
     x:=x+1;
       dbms_output.put_line('la valeur de x est dans la boucle:' || x);
     exit incrt_boucle when x>5;
 end loop;
dbms_output.put_line('la valeur de x à la fin est :' || x);
```

```
la valeur de x est dans la boucle:1
     la valeur de x est dans la boucle:2
     la valeur de x est dans la boucle:3
     la valeur de x est dans la boucle:4
     la valeur de x est dans la boucle:5
     la valeur de x est dans la boucle:6
     la valeur de x à la fin est :6
     PL/SQL procedure successfully completed.
```

21

WHILE

[<<NOM_BOUCLE>>]
WHILE <condition> LOO P
<instructions>
END LOOP [NOM_BOUCLE]

22

```
declare
 x number := 0;
Begin
<<boucle_while>>   /* pas obligatoire */
while x <= 5 loop
     x:=x+1;
       dbms_output.put_line('la valeur de x est dans la boucle:' || x);
      end loop boucle_while;
dbms_output.put_line('la valeur de x à la fin est :' || x);
end;
```

```
la valeur de x est dans la boucle:1
     la valeur de x est dans la boucle:2
     la valeur de x est dans la boucle:3
     la valeur de x est dans la boucle:4
     la valeur de x est dans la boucle:5
     la valeur de x est dans la boucle:6
     la valeur de x à la fin est :6
     PL/SQL procedure successfully completed.
```

23

FOR LOOP

[<<NOM_BOUCLE>>]
FOR indice IN [REVERSE] inf..sup LOOP
<instructions>;
END LOOP [NOM_BOUCLE];

24

Exemple

```
declare
 x number := 0;
Begin
for  x in 1..6 loop
       dbms_output.put_line('la valeur de x est dans la boucle:' || x);
      end loop;
dbms_output.put_line('la valeur de x à la fin est :' || x);
end;

la valeur de x est dans la boucle:1
     la valeur de x est dans la boucle:2
     la valeur de x est dans la boucle:3
     la valeur de x est dans la boucle:4
     la valeur de x est dans la boucle:5
     la valeur de x est dans la boucle:6
     la valeur de x à la fin est :0
     PL/SQL procedure successfully completed.
```


Intégration des requêtes

- Requêtes retournant une seule ligne
- Requêtes retournant plusieurs lignes (curseurs)

Requête à ligne unique

Mot clé **into**

- Exemple:

```
declare
unProduit Produit%ROWTYPE;   /*Une variable */
Begin
Select * into unProduit from produit where prod_id ='p2';
dbms_output.put_line(unProduit.prod_id  ||  unProduit.nom   ||
unProduit.pu);
End;
```


Requêtes à lignes multiples

- Utilisation d'une structure de données appelée **CURSEUR**
- Qu'est-ce que c'est un curseur?
  - Un **curseur** est un nom symbolique associé à une instruction **select**
- A quoi ça sert?
  - Manipuler chaque ligne individuellement d 'un **select**
  - Accès aux donner ligne par ligne.

Les curseurs

- Comment l'utiliser?

Déclare, ouvrir, parcourir et fermer.

- Déclaration du curseur

```
CURSOR nom_cursor [[nom_arg type
   :=valeur_defalut[……])] is reqte;
```


Exemple

- Sql> declare
  - Cursor c_produit_sans is
    select * from produit order by nom;
- Sql> declare
  - Cursor c_produit_avec(p_nom Produit.nom%TYPE := 'Titi') is
    select * from Produit where nom =p_nom;

Ouverture d'un curseur

- Open nom_curseur [(valeur_arg[..,..])
- Exemple
  - Open c_produit_sans ;
  - Open c_produit_avec('crayon') ; -- la valeur de param p_nom est affectéé à 'crayon'
  - Open c_produit_avec(p_nom =>'crayon') ;

31

Traitement des lignes d'un curseur

- Fetch nom_curseur into nom_enregistrement |nom_variable[..]};
- Status d'un curseur:
  - %FOUND: attr de type booléen…vari si exécution correcte de la reqte SQL
  - %NOTFOUND
  - %ISOPEN
  - %ROWCOUNT
- La syntaxe de consultation:
  - Nom_curseur%ATTRIBUT
- Exemple : c_produit_sans%ISOPEN

32

```
Sql> Declare
unProduit Produit%ROWTYPE;
Cursor  c_prod is select  prod_id, nom,pu  from Produit
      order by nom;
begin
open c_prod;
if c_prod%ISOPEN then
 dbms_output.put_line('la valeur de rowcount: ' || c_prod%ROWCOUNT);
loop
 FETCH c_prod  INTO unProduit;
 EXIT WHEN c_prod%NOTFOUND;
  IF unProduit.pu > 10 then
  dbms_output.put_line(unProduit.nom ||  unProduit.pu || ' cher');
end if;
dbms_output.put_line('la valeur de rowcount: ' || c_prod%ROWCOUNT);
End loop;
end if;
close c_prod;
End;

la valeur de rowcount: 0
      la valeur de rowcount: 1
      toto200 pas cher
      la valeur de rowcount: 2
      PL/SQL procedure successfully completed.
```

33

Avec while:

```
Set serveroutput on;
Declare
unProduit Produitt%ROWTYPE;
Cursor  c_prod is select  prod_id, nom,pu  from Produit   order by nom;
begin
open c_prod;
 dbms_output.put_line('la valeur de rowcount  ' || c_prod%ROWCOUNT);
 FETCH c_prod  INTO unProduit;
while c_prod%FOUND loop
  IF unProduit.pu> 10 then
  dbms_output.put_line(unProduit.nom ||  unProduit.pu || ' cher');
end if;
dbms_output.put_line('la valeur de rowcount: ' || c_prod%ROWCOUNT);
FETCH c_prod  INTO unProduit;
end loop;
close c_prod;
End;
```

34

Les boucles FOR et les curseurs

- Curseur est ouvert et fermé automatiquement
- La variable unProduit est définie automatiquement comme une variable de type enregistrement en lecture seule.

```
 Sql> Declare
Cursor  c_prod is select  prod_id, nom,pu  from Produit
      order by nom;
begin
for unProduit in c_prod loop
  if  unProduit.pu > 10 then
  dbms_output.put_line(unProduit.nom ||  unProduit.pu || '  cher');
  end if;
End loop;
End;
```

35

Les boucles FOR et les curseurs

- Sans déclarer le curseur:

```
begin
for unProduit in (select prod_id, nom,pu  from Produit  order by nom)  loop
  if  unProduit.pu > 10 then
  dbms_output.put_line(unProduit.nom ||  unProduit.pu || '  cher');
  end if;
End loop;
End;
```

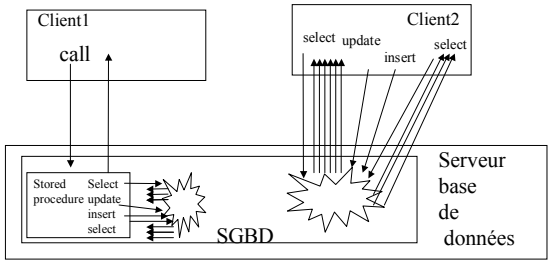
36

Procédures stockées

- Instructions SQL + le langage de contrôle de flux dans une procédure stockée afin d'améliorer les performances de SQL Server.
- Les procédures stockées sont précompilées.
  - 1ère fois:
    - le processeur de requêtes de SQL Server l'analyse et prépare un plan d'exécution
    - le nom, text sont stockés dans le dictionnaire
    - La procédure est exécutée selon ce plan stocké: exécution presque instantanément.

37

Procédures stockées et trafic réseau



38

Procédures stockées

- Déclaration

Create or replace <nom procedure>(paramètres) {is |as}
<bloc pl/sql>
End [nom_procedure] ;

- Suppression
  - drop procedure nom_procédure ;

39

```
select * from produit
      p2  toto  200
      p1  A3  10
show errors;
create or replace procedure addProduit
is
Begin
 insert into Produit values('p3','toto',100);
end addProduit;
No errors.
    Procedure created.
```

Appel :

```
SQL> call addProduit();
      p2  toto  200
      p1  A3  10
      p3 toto 100
Sql >
begin
addProduit();
end;
```

40

En cas de problème

« Attention : procédure créée avec erreurs de compilation »

> show errors

LINE/COL ERRO R

---------- ----------

XX/YY m essage d 'erreur

41

Procédure avec paramètres formels

- Les paramètres sont IN, OUT ou IN OUT
- Exemple:

```
create or replace procedure addProduit ( prod_id in char(2),
    nomp in  varchar2(20) Produit.nom%type , pup float)  is
begin
insert into Produit values (prod_id,nomp,pup);
end addProduit;
Warning: Procedure created with compilation errors.
```

- Attention: dans la déclaration de procédure, une contrainte de longueur ne peut pas être appliquée aux arguments char, varchar.. De même qu'une contrainte de précision aux arguments number

42

Exemple

```
create or replace procedure addProduit (
   prod_id in Produit.prod_id%type, nomp in
   Produit.nom%type , pup in
   Produit.pu%type)  is
begin
insert into Produit values (prod_id,nomp,pup);
end addProduit;
```


Exemple

```
create or replace procedure updatePuProduit(
   montant in produit.pu%type)  is
begin
Update  Produit set pu = pu +montant ;
Montant := 200 ;
end updatePuProduit;
```

- Errors .. Pourquoi ?

Exemple

```
create or replace procedure updatePuProduit(
   montant in Produit.pu%type)  is
begin
Update  Produit set pu = pu +montant ;
end updatePuProduit;
```


Exemple

```
create or replace procedure nombreProduits(nb_produits out  number)  is
begin
Dbms_output.put_line('la valeur est :' || nb_produits);
Select count(*) into nb_produits from Produit;
end nombreProduits;
```

Appel:

```
declare
res number;
begin
nombreProduits(res);
dbms_output.put_line(res);
end;
```

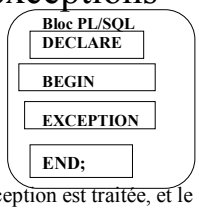

Functions

```
create or replace function nbProduits return number is
vnb number;
Begin
select count(*) into vnb from produit;
return vnb;
end nbProduits;
```

Appel:

```
begin
dbms_output.put_line('nombre de produits:'|| nbProduits());
end;
```


Gestion des exceptions



- Les exceptions provoquent l'arrêt du déroulement normal du bloc PL/SQ L.
- Si un gestionnaire d'exception a été mis en place l'exception est traitée, et le programme peut éventuellement reprendre son cours
- Si aucun gestionnaire n'est mis en place l'exception provoque l'affichage d'un message d'erreur (SQ L*PLUS) et l'arrêt de l'exécution du bloc.

Les types d'exceptions

- 4 types:
  - Les exceptions systèmes nommées :suite à une erreur PL/SQL ou oracle.
  - Exceptions systèmes anonymes: suite à une erreur PL/SQL ou oracle. Il est possible de donner un nom (PRAGMA EXCEPTION_INIT)
  - Les exceptions utilisateur nommées: suite à une erreur dans le code applicatif
  - Exceptions utilisateur anonymes : code compris entre –20 000 et –20 999, déclenche avec RAISE_APPLICATION_ERROR ..

Quelques exceptions prédéfinies

- Dup_val_on_index:
  - essai d'insertion d'une ligne dont la clé existe déjà dans la table
- Invalid_number:
  - conversion d'une chaîne en nombre qui échoue
- no_data_found:
  - aucune ligne retournée par la dernière requête
- Too_many_rows:
  - le dernier select a retourné plus d'une ligne
- Zero_divide:
  - division par 0
- Others:
  - pour récupérer toutes les exceptions non traitées explicitement

Exception

```
Exception
 when nom_exception [OR nom_exception ..] then
 instructions;
[when others then instructions ;]
End;
```


Exemple

```
declare
    unProduit Produit%ROWTYPE;   /*Une variable */
    Begin
    Select * into unProduit from produit where prod_id ='p9';
    dbms_output.put_line(unProduit.prod_id  ||  unProduit.nom   ||
    unProduit.pu);
    End;

declare * ERROR at line 1:
    ORA-01403: no data found
    ORA-06512: at line 4
```


Exemple

```
declare
    unProduit Produit%ROWTYPE;   /*Une variable */
Begin
    Select * into unProduit from produit where prod_id ='p9';
    dbms_output.put_line('Vous ne verrez pas cette ligne ');
 exception
    When NO_DATA_FOUND then
    dbms_output.put_line('pas de produit avec ce numero ');
end;
```

pas de produit avec ce numero
PL/SQL procedure successfully completed.

Packages

- Un package est un regroupement d'un ensemble d'objets (curseurs, variables, procédures et de fonctions) logiquement associés.
- Deux parties distinctes :
  - le corps contient le code et déclarations privées
  - la spécification: interface publique

Avantages

- Protection de données:
  - Public: peut-être référencé à l'extérieur du package
  - Privé: dans le package lui-même
  - Accès aux spécification et masquer le code
- Conception descendante
  - Écrire l'interface avant de coder le corps
  - Compiler l'interface seules ..

Avantages

- Données globales:
  - Ex: une procédure packagée ouvre un curseur, celui-ci reste ouvert et utilisable par d'autres routines packagées durant toute la session…
  - Les variables de packages sont liées à la session et non pas à une transaction
- Amélioration des performances:
  - Appel d'un objet 1ère fois, tout le package est chargé en mémoire…..

Spécification de package

```
Create or replace package gererProduit
As
v_produit   Produit%ROWTYPE;
e_pasDeproduit  EXCEPTION;
function nbProduits return number;
procedure addProduit (prod_id in Produit.prod_id%type,
    nomp in Produit.nom%type , pup in Produit.pu%type);
end gererProduit;
```

Attention: Dans l'interface c'est **public** dans le corps est **privé**

Corps de package

```
Create or replace package body gererProduit
Is
function nbProduits return number is
vnb number;
Begin
select count(*) into vnb from produit;
return vnb;
end nbProduits;
procedure addProduit ( prod_id in Produit.prod_id%type, nomp in
    Produit.nom%type, pup in Produit.pu%type)  is
begin
insert into Produit values (prod_id,nomp,pup);
end addProduit;
End gererProduit;
```

Exécution: call gererProduit.addProduit('p7', 'titi',300);

Corps de package

```
Create or replace package body gererProduit
Is
function nbProduits return number is
vnb number;
Begin
select count(*) into vnb from produit;
return vnb;
end nbProduits;
procedure addProduit ( prod_id in Produit.prod_id%type, nomp in Produit.nom%type, pup in Produit.pu%type)  is
begin
insert into Produit values (prod_id,nomp,pup);
end addProduit;
Begin
Exception
 when Dup_VAL_ON_INDEX then
     dbms_output.put_line('Erreur le  produit est là ');
End gererProduit;
```

Exécution: call gererProduit.addProduit('p7', 'titi',300);