

# HAI901I : PL/SQL

I.Mougenot

UM

2021

# Surcouche procédurale Oracle

## Plusieurs rôles à son actif

- ➊ **renforcement de la sécurité des bases de données**
- ➋ **supervision et évaluation de la performance**
- ➌ gestion de l'applicatif au sein même du SGBD (cœur métier)
- ➍ langage support des développeurs Oracle : couche objet, fonctions élémentaires disponibles dans SQL
- ➎ **langage transactionnel (instructions en bloc : atomicité)**

## Pour ce module :

En faire le langage support qui va permettre d'aborder toutes les thématiques abordées dans le module : sécurité, performance, optimisation, architecture physique, transaction

# Ajouts à la base de données

Choix assumés de ne pas gérer que la donnée

- 1 procédures et fonctions
- 2 déclencheurs (ou triggers)
- 3 types de données utilisateur (UDT)
- 4 paquetages
- 5 ...

# Notion de programme principal

## Listing 1: Structure générale

```
[DECLARE  
-- declarations variables, curseurs,  
-- constantes, exceptions]  
BEGIN  
-- instructions  
[EXCEPTION -- gestion des erreurs ]  
END ;  
/
```

Tout bloc PL/SQL se termine par une ligne ne comportant qu'un point ou encore un slash (dans ce dernier cas, la vérification du code est alors suivie d'une exécution immédiate).

# Corps du programme

Un corps de programme PL/SQL peut comprendre :

- ordres SQL
- opérateurs de comparaison
- instructions de contrôle conditionnel
- instructions de contrôle itératif
- gestion des curseurs
- gestion des erreurs

# Types de données

## Types de données primitives ou complexes

- données scalaires (char, varchar, long, boolean, integer, number, float, ...)
- record et table de la surcouche objet/relationnel (SQL3)
- cursor (équivalent d'une collection de tuples)
- déclaration d'un type sur la base d'un attribut ou en encore d'un tuple d'une relation
  - `nom_table.nom_attribut%type`
  - `nom_table.nom_attribut%rowtype`

## Illustration : programme sans vrai fond

Listing 2: Exemple PP

```
DECLARE
x number ; y number ; compteur number ;
z constant integer := 4 ; --constante
aujourd'hui date := sysdate ; -- date du jour
BEGIN
x :=2 ; -- affectation
if x>0 then y :=x ; else y :=-x ;
end if ; -- traitement conditionnel
dbms_output.put_line('x et y valent '||x||' et '||y) ;
for compteur in 1..z -- traitement iteratif
loop
dbms_output.put_line('histoire d''essayer');
end loop ;
END ;
```



# Paquetage PL/SQL

Traiter la logique fonctionnelle comme une collection de fonctions et de procédures

PL/SQL donne accès à un ensemble de paquetages prédéfinis.  
L'utilisateur peut définir ses propres paquetages métiers

- DBMS\_OUTPUT
- DBMS\_METADATA
- DBMS\_STATS
- DBMS\_UTILITY
- DBMS\_TRACE
- ...

voir le contenu d'un paquetage : desc DBMS\_OUTPUT

## Exemple de DBMS\_OUTPUT

```
PROCEDURE PUT (A VARCHAR2)
PROCEDURE PUT (A NUMBER)
PROCEDURE PUT (A DATE)
PROCEDURE PUT_LINE (A VARCHAR2)
PROCEDURE PUT_LINE (A NUMBER)
PROCEDURE PUT_LINE (A DATE)
```

Listing 3: Signatures PUT

Pour afficher à l'écran divers messages ou variables, il est nécessaire d'utiliser la librairie qui fournit les méthodes d'écriture des diverses variables sur le périphérique de sortie standard. Les procédures PUT et PUT\_LINE permettent d'écrire dans le buffer (avec possibilité de retour à la ligne)

# Notion de curseur

Les curseurs (CURSOR) permettent de traiter les résultats d'une requête (équivalents de tables temporaires)

Deux types de curseurs :

- curseur implicite quand la requête ne renvoie qu'un tuple
- curseur explicite quand la requête renvoie plus d'un tuple

Indispensables à l'accès du contenu des tables

## Exemple de curseur implicite

"set serveroutput on" pour l'affichage et "show errors" si erreurs

```
declare
name emp.nom%type;
numero emp.num%type;
begin
select num, nom into numero, name from emp where num=79
    ;
dbms_output.put_line('j''affiche le numero et nom
    '||numero||' '||name) ;
end ;
/
```

Listing 4: Curseur implicite

## Exemple de curseur explicite

```
declare
cursor mon_curseur is
select num, nom, fonction
from EMP ;
begin
For emp_rec in mon_curseur
loop
dbms_output.put_line(emp_rec.num||' '||emp_rec.nom||'
    '||emp_rec.fonction);
end loop ;
end ;
/
```

Listing 5: Simplification

# Notion d'exception

La gestion des exceptions : indispensable à la prise en charge d'anomalies de comportement

La gestion des exceptions dans un programme PL/SQL peut se faire en trois temps :

- 1 Déclarer une exception :
- 2 Lever une exception
- 3 Traiter une exception

## Enrichissement exemple précédent

```
declare
name emp.nom%type; numero emp.num%type;
mon_exception exception;
begin
select num,nom into numero,name from emp where num=79 ;
if name is null then raise mon_exception ;
else dbms_output.put_line('j''affiche le numero et nom
    '||numero||' '||name) ;
end if ;
exception
when mon_exception then
    raise_application_error(-20100,'le salarie
est non reference');
end ;
/
```

# Des exceptions prédéfinies

Un certain nombre d'exceptions sont prédéfinies :

- NO\_DATA\_FOUND
- CURSOR\_ALREADY\_OPEN
- ZERO\_DIVIDE
- VALUE\_ERROR
- TOO\_MANY\_ROWS
- OTHERS

La fonction `RAISE_APPLICATION_ERROR`(code erreur dans les -20000, message à afficher) sert à bloquer toute opération indésirable.



## Usage d'exception prédéfinie

```
declare
name emp.nom%type; numero emp.num%type;
begin
select num,nom into numero,name from emp where num=79 ;
dbms_output.put_line('j''affiche le numero et nom
    '||numero||' '||name) ;
exception
when no_data_found then
    raise_application_error(-20100,'le salarie
est non reference');
end ;
/
```

Listing 7: Exception

# Les structures conditionnelles

Plusieurs types d'instructions sont disponibles :

- IF : La syntaxe sera de type : `if ... then ... else ... end if ;`  
Les if imbriqués sont possibles pour traiter les choix multiples (else if ou encore elsif)
- CASE : La syntaxe sera de type : `case [ expression ] when condition_1 then result_1 ; when condition_2 then result_2 ; ...when condition_n then result_n ; else result end`

## Exemple IF imbriqué

```
IF sal < 1000 THEN prime:=300;  
ELSIF sal < 1500 THEN prime:=300;  
ELSIF sal < 2000 THEN prime:=200;  
ELSE prime:=100;  
END IF;
```

Listing 8: Condition

# Les structures itératives

Plusieurs types d'instructions sont disponibles :

- boucle infinie loop séquence-de-commandes end loop avec une instruction exit pour sortir de la boucle
- for compteur in borneInf ... borneSup loop  
séquence-de-commandes end loop
- while condition-plsql loop séquence-de-commandes end loop

## Exemple While

```
declare
i integer :=20000 ;
begin
while i <= 20010
loop
INSERT INTO emp (nom,num) VALUES ('pierrot',i);
i :=i+1 ;
end loop ;
end ;
```

Listing 9: Loop While

## Exemple For

```
begin
for i in 20000..20010
loop
INSERT INTO emp (nom,num) VALUES ('pierrot',i);
end loop ;
```

Listing 10: Loop For

## Exemple Exit

```
loop  
exit when i>20010 ;  
INSERT INTO emp(nom,num) VALUES ('pierrot',i);  
i :=i+1 ;  
end loop ;
```

Listing 11: Loop Exit

# Les procédures et les fonctions

Un intérêt de PL/SQL : définir des objet fonctions et procédures cataloguées au niveau du méta-schéma et pouvant être appelées par les utilisateurs

- Consultables dans les vues user\_objects ou user\_procedures :  
select object\_name from user\_procedures;
- suppression d'une procédure : drop procedure nom\_procedure
- suppression d'une fonction : drop function nom\_fonction



## Schéma de construction d'une procédure (BNF)

```
create [or replace] procedure [schema.]nom_procedure  
    [(liste d'arguments)]  
{is|as}  
bloc pl/sql
```

Arguments :

- in : variable en entree
- out : variable en sortie
- in out : variable en entree, renseignee par la  
procédure puis renvoyee

Listing 12: Procédure PL/SQL

# Schéma de construction d'une fonction (BNF)

```
create [or replace] function [schema.]nom_fonction  
    [(liste d'arguments)]  
return type  
{is|as}  
bloc pl/sql
```

Listing 13: Fonction PL/SQL

## Illustration procédure de conversion monétaire

```
create or replace procedure P_CEF (euros IN number,  
    francs OUT number) is  
begin  
    francs :=euros*6.559;  
end ;  
/  
declare  
    temp number ;  
begin  
    P_CEF(100,temp);  
    DBMS_OUTPUT.PUT_LINE(temp);  
end ;  
/
```

Listing 14: Proc et PP

## Illustration fonction de conversion monétaire

```
create or replace function F_CEF (euros in number)
    return number
is
francs number ;
begin
francs :=euros*6.559;
return (francs) ;
end ;
/
declare
result number ;
begin
result := F_CEF(100) ;
DBMS_OUTPUT.PUT_LINE(result);
end ;
/
```

## Illustration fonction de conversion monétaire

La fonction F\_CEF peut être utilisée maintenant dans une requête SQL :

```
select salaire, f_cef(salaire) as salaire_en_francs  
from emp;
```

Listing 16: Fonction utilisateur dans SQL

## Contraintes et déclencheurs (triggers)

Mise à jour pouvant être à risque : erreur, donnée de mauvaise qualité, perte de cohérence, acte de malveillance ...

L'intégrité des BD est assurée par les contraintes intra ou inter-relations mises en place au niveau du schéma de BD :

- contrainte intra-table
  - clé primaire (double contrainte d'unicité et de non nullité)
  - contrainte sur les attributs ou sur les tuples (unicité, non nullité et autres contraintes de domaine)
- contrainte inter-table : clé étrangère

Aller plus loin dans la sécurité des données et s'orienter vers de la supervision d'activités : déclencheurs

# Trigger ou déclencheur

Programmation événementielle et action différée : un déclencheur est vu comme une règle événement-condition-action ou règle ECA. Le trigger s'active lorsqu'un événement portant sur la base de données survient

- ❶ déclencheur LMD (Langage de Manipulation de Données) : opérations de mise à jour des tuples d'une table
- ❷ déclencheur LDD (Langage de Définition de Données) : opérations survenant au niveau d'un schéma utilisateur de la base
- ❸ déclencheur d'instance : opérations survenant au niveau de la base

# Déclencheur LMD

Il possède les caractéristiques suivantes :

- l'action réalisée soit avant (**before**), soit après (**after**) l'évènement
- l'action réalisée pour chaque tuple concerné (**for each row**) ou bien pour l'ensemble des tuples concernés
- la réponse peut se rapporter aux anciennes (**old**) comme aux nouvelles valeurs (**new**) des attributs du tuple affecté lors de l'évènement
- pour une opération update, l'évènement peut ne concerner qu'un attribut ou un ensemble d'attributs
- Les conditions peuvent être spécifiées dans le corps du programme ou bien dans une clause WHEN.



## Syntaxe BNF déclencheur

```
create [or replace] trigger [schema.]nom trigger
before|after
delete|insert|update [of colonne [,colonne]...] [or
delete|insert|update [of colonne [,colonne]...]...]
on [schema.]table [[reference old [as] ancien new [as]
nouveau] |
new [as] nouveau old [as] ancien]]
[for each row]
[when (condition)]
bloc pl/sql
```

Listing 17: Un trigger en notation BNF

## Exemple d'un déclencheur d'alerte

```
create or replace trigger Alert
after delete or insert or update
on EMP for each row
begin
dbms_output.put_line('fin de l''Operation');
end ;
/
```

Listing 18: Message

## Exemple d'un déclencheur qui encadre les jours ouvrés

```
create trigger Ouvrable
before delete or insert or update on EMP
begin
if (to_char(sysdate,'DY')='SAT') or
   (to_char(sysdate,'DY')='SUN')
then
raise_application_error(-20010, 'Modification interdite
   le '||to_char(sysdate,'DAY') ) ;
end if ;
end ;
/
```

Listing 19: Ouvrable

## Exemple d'un déclencheur avec when, new et old

```
create trigger verif_salaire
before insert or update of salaire on EMP
for each row
when (new.fonction != 'president')
begin
if ( :new.salaire < 1000 ) then
raise_application_error
(-20022, :new.nom||' n''est pas assez paye.') ;
elsif ( :new.salaire > 5000 ) then
raise_application_error
(-20023, :new.nom||' est trop paye.') ;
end if ;
end ;
/
```

Listing 20: Salaire

## Illustration before

```
create or replace trigger Before_D
before insert on EMP for each row
begin
:new.nom := upper( :new.nom);
dbms_output.put_line('bravo vous venez d''insérer le
    tuple '|| :new.nom) ;
end ;
/
```

Listing 21: Before

## Illustration after : lève une erreur

```
create or replace trigger After_D
after insert on EMP for each row
begin
:new.nom := upper( :new.nom);
dbms_output.put_line('bravo vous venez d''insérer le
    tuple '|| :new.nom) ;
end ;
/
```

Listing 22: After

## Déclencheur global

```
create table log (  
txt varchar2(20), date_maj date, user_maj varchar2(15)  
    ) ;  
create or replace trigger Global_update  
before update on EMP  
begin  
insert into log values ('update trigger', sysdate,  
    user) ;  
end ;  
/  
update EMP set nom = nom || 't'  
where substr(nom,1,1) = 'M' ;  
  
-- un seul enregistrement dans log  
select * from log ;
```

## Déclencheur sur le type d'évènement

```
create or replace trigger monitor_historique
after insert or delete or update on emp for each row
declare
typeOp varchar(15);
BEGIN
if inserting then
typeOp := 'Insertion';
elsif updating then
typeOp := 'Modification';
elsif deleting then
typeOp := 'Suppression';
end if ;
insert into historique values (sysdate, typeOp,
user, :old.num, :new.num);
end ;
/
```



## Déclencheur LDD

Interdiction de toute suppression de table ou autre objet d'un schéma de la base le vendredi

```
set serveroutput on

create or replace trigger majRefusee before drop on
    isa.schema
begin
    if trim(to_char(sysdate,'DAY')) = 'FRIDAY' then
        raise_application_error
            (-20001,'forbidden on friday') ;
    else dbms_output.put_line(to_char(sysdate,'DAY'));
    end if ;
end ;
/
```

# Exemple de déclencheur d'instance

```
create table QuiSeConnecte
(
  c_user varchar2(30),
  os_user varchar2(60),
  c_date date
);
```

Listing 1: supervision au travers d'une table

# Exemple déclencheur instance

```
CREATE OR REPLACE TRIGGER logon_db
after logon ON DATABASE
declare
o_user varchar2(60);
begin
select sys_context('USERENV','OS_USER') into o_user
      from dual ;
INSERT INTO QuiSeConnecte VALUES (user, o_user,
      sysdate);
commit;
end;
/
```

Listing 2: trigger alimentant la table

# Vues du méta-schéma et déclencheurs

- Consulter : `select trigger_name from user_triggers`
- Supprimer : `drop trigger nom_declencheur`
- `select text from user_source where name = 'NOM_DECLENCHEUR';`
- désactiver : `alter trigger nom_declencheur disable;`
- réactiver : `alter trigger nom_declencheur enable;`