SUPPORT DE COURS
SYSTEME DE GESTION DE BASE DE DONNEES

Public cible: DSI2

Charge horaire semestrielle par semaine:

1,5h cours
1,5h TD
1,5h TP

Objectifs du cours:

Ecrire des programmes modulaires en utilisant les procédures, les fonctions et les packages.
Contrôle des événements à l'aide des déclencheurs.

Pré-requis du cours:

Algorithmique
Base de données
Le langage SQL

SOMMAIRE

1. INTRODUCTION

Pourquoi PL/SQL ?

- SQL est un langage non procédural

- Les traitements complexes sont parfois très difficiles à écrire si on ne peut utiliser des variables et des structures de programmation comme les boucles et les alternatives

- On ressent vite le besoin d'un langage Procédural pour lier plusieurs requêtes SQL avec des variables et des structures de programmation habituelles

Principales caractéristiques de PL/SQL

- Extension de SQL : des requêtes SQL cohabitent avec les structures de contrôle habituelles de la programmation structurée (blocs, alternatives, boucles)

- Un programme est constitué de procédures et de fonctions

- Des variables permettent l'échange d'information entre les requêtes SQL et le reste du programme

Utilisation de PL/SQL

- PL/SQL peut être utilisé pour l'écriture des procédures stockées et des triggers

- Il convient aussi pour écrire des fonctions utilisateurs qui peuvent être utilisées dans les requêtes SQL (en plus des fonctions prédéfinies)

2. STRUCTURE D'UN PROGRAMME

```
DECLARE

  -- définitions de variables

BEGIN

  -- Les instructions à exécuter

EXCEPTION

  -- La récupération des erreurs

END;
/
```

LES VARIABLES

Identificateurs Oracle :
- 30 caractères au plus
- commence par une lettre
- peut contenir lettres, chiffres, _, $ et #

Pas sensible à la casse
Doivent être déclarées avant d'être utilisées

Commentaires :
-- Pour une ligne
/* Pour plusieurs lignes */

Types de variables

Les types habituels correspondants aux types Oracle : number, char,…

Types composites adaptés à la récupération des colonnes et lignes des tables SQL : %TYPE, %ROWTYPE

Déclaration d'une variable

identificateur [CONSTANT] type [:= valeur];

Exemples :

```
age number;
nom varchar(30);
dateNaissance date;
ok boolean := true;
```

Déclarations multiples **interdites** :

```
i, j number;
```

Déclaration %TYPE

On peut déclarer une variable du même type qu'une colonne d'une table

```
nom emp.nom%TYPE;
```

Déclaration %ROWTYPE

Une variable peut contenir toutes les colonnes d'une table

```
employe emp%ROWTYPE;
```

Exemple d'utilisation

```
declare
   employe emp%ROWTYPE;
begin
   select * into employe from emp
   where empno=7369;
   insert into emp(empno,ename) values
   (44,employe.ename);
end;
/
```

Type RECORD

Equivalent à struct du langage C

```
TYPE nomRecord IS RECORD
(
champ1 type1,
champ2 type2,
…);
```

Utilisation du type RECORD

```
declare
   type employes is record
     (empno number(4),
      ename char(10)
     );
   employe employes;
begin
   select empno, ename into employe
   from emp
   where empno=7369;
   insert into emp(empno,ename) values
   (45,employe.ename);
end;
 /
```

Type TABLE

TYPE nomTable IS TABLE of type INDEX BY BINARY_INTEGER;

Utilisation du type TABLE

```
declare
   type employes is table of varchar2(25) index by
   binary_integer;
   employe employes;
begin
   select ename,job into employe(1), employe(2)
   from emp
   where empno=7369;
   insert into emp(empno,ename,job) values
   (48,employe(1), employe(2));
end;
```

Affectation

deux façons d'affecter une valeur à une variable :

- :=
- par la directive INTO de la requête SELECT

Exemples :

- wdate := '10/10/2004';
- select ename INTO wnom
  from emp
  where empno = 7369;

3. STRUCTURES DE CONTRÔLE

Structures alternatives

- IF condition THEN
  instructions;
  END IF;

Exemple:

```
declare
        wsal number(10,3);
begin
        select sal into wsal
        from emp where empno=7369;
        if wsal<500 then
                update emp set sal=sal*1.1
                where empno=7369;
        end if;
end;
/
```

❖ IF condition THEN
    instructions1;
  ELSE
    instructions2;
  END IF;

Exemple:

```
declare
        wsal number(10,3);
 begin
        select sal into wsal
        from emp where empno=7369;
        if wsal<500 then
                update emp set sal=sal*1.1 where empno=7369;
        else
               update emp set sal=sal*1.05
               where empno=7369;
        end if;
end;
/
```

- IF condition1 THEN
    instructions1;
  ELSIF condition2 THEN
    instructions2;
  ELSIF …
    …
  ELSE
    instructionsN;
  END IF;

```
declare
        wsal number(10,3);
begin
        select sal into wsal
        from emp
         where empno=7369;

        if wsal<500 then
                update emp set sal=sal*1.1
                where empno=7369;
        elsif wsal<900 THEN
                update emp set sal=sal*1.05
                where empno=7369;
        else
                update emp set sal=sal*1.02
                where empno=7369;
        end if;
end;
```

Structures répétitives

Boucle « tant que »

```
WHILE condition LOOP
        instructions;
END LOOP;
```

Exemple:

```
 declare
        i number:=0;
        s number :=0;
 begin
         while i<=9 loop
            s:=s+i;
            i:=i+1;
         end loop;
   dbms_output.put_line(s);
 end;
/
```

Boucle « jusqu'à »

```
LOOP
        instructions;
EXIT [WHEN condition];
END LOOP;
```

Exemple:

```
declare
         i number:=0;
         s number :=0;
begin
        loop
                s:=s+i;
                i:=i+1;
                exit when i=10;
        end loop;
Dbms_output.put_line(s);
end;
/
```

Boucle « pour »

```
FOR compteur IN [REVERSE] inf..sup LOOP
        instructions;
END LOOP;
```

Exemple :

```
declare
        s number :=0;
begin
        for i in 0..9  loop
                s:=s+i;
        end loop;
Dbms_output.put_line(s);
end;
 /
```

```
declare
        s number :=0;
begin
        for i in reverse 1..9 loop
                s:=s+i;
         end loop;
Dbms_output.put_line(s);
end;
 /
```

Mise au point

Pour la mise au point il est utile de faire afficher les valeurs des variables

On peut le faire en activant sous SQL*PLUS la sortie sur l'écran et en utilisant le paquetage DBMS_OUTPUT

Un paquetage est un regroupement de procédures et de fonctions

Exemple:

```
set serveroutput on
declare
        s number :=0;
begin
        for i in reverse 1..9 loop
                s:=s+i;
        end loop;
 dbms_output.put_line (s);
 end;
/
```

4. LES CURSEURS

Fonctionnalités

Toutes les requêtes SQL sont associées à un Curseur

❖ Ce curseur représente une zone mémoire utilisée pour exécuter une requête

❖ Le curseur peut être implicite ou explicite (déclaré par l'utilisateur)

❖ Les curseurs explicites servent à retourner plusieurs lignes

Attributs des curseurs

Tous les curseurs ont des attributs que l'utilisateur peut utiliser

- %ROWCOUNT : nombre de lignes traitées par la requête
- %FOUND : vrai si au moins une ligne a été Traitée
- %NOTFOUND : vrai si aucune ligne n'a été traitée
- %ISOPEN : vrai si le curseur est ouvert

Curseur implicite

❖ Un curseur implicite est toute commande SQL (select, insert, update, delete) située dans la partie BEGIN.

❖ Un curseur implicite ne possède pas de nom

❖ Son nom est toujours SQL.

Exemple:

```
declare
        wempno  emp.empno%type;
        wename emp.ename%type;
begin
       select empno, ename into wempno, wename
       from emp
       where empno=7369;
       dbms_output.put_line(wempno||wename);
 end;
/
```

Exception du curseur implicite *select into*:

- NO_DATA_FOUND: aucune ligne n'est traitée
- TOO_MANY_ROWS : résultat formé de plusieurs lignes

Conséquence d’exception

En cas d’exception non gérée par le développeur, le système engendre un échec en abandonnant le reste de l’exécution du bloc.

Remarque:

Avec les curseurs implicites de mise à jour (update, delete et insert) le problème ne se pose pas en cas de:

- ✓ aucune ligne n'est traitée
- ✓ résultat formé de plusieurs lignes

Exemple1:

```
begin
        delete from dept where deptno=345;
        dbms_output.put_line(sql%rowcount);
end;
 /
```

Exemple2:

```
begin
        delete from a;
        dbms_output.put_line(sql%rowcount);
        if (sql%notfound) then
                dbms_output.put_line('pas de lignes');
        end if;
end;
 /
```

Curseur explicite

- Pour traiter les select qui renvoient plusieurs lignes
- Ils doivent être déclarés
- Ils doivent être utilisés explicitement avec les ordres OPEN, FETCH et CLOSE ou bien avec FOR
- Le plus souvent on les utilise dans une boucle dont on quitte quand l'attribut *NOTFOUND* du curseur est vrai

Syntaxe

DECLARE

CURSOR nom_curseur[(liste de paramètres)] **IS**

Commande select

[FOR UPDATE OF COLONNE1,…?COLONNE N];

…

BEGIN

OPEN nom_curseur[(liste de paramètres)];
LOOP
FETCH nom_curseur INTO variables;
EXIT WHEN nom_curseur%NOTFOUND;
END LOOP;
CLOSE nom_curseur.

FOR enreg IN nom_curseur[(liste de paramètres)]LOOP

…..

;
END LOOP;

END;

Exemple avec fetch(curseur non paramétré)
Affichage de la liste des fonctions (job)

```
  declare
        cursor fonctions is
        select distinct job
        from emp;
        v_job fonctions%rowtype;
   begin
        open fonctions;
        loop
              fetch fonctions into v_job;
              exit when fonctions%notfound;
              dbms_output.put_line(v_job.job);
        end loop;
        close fonctions;
   end;
/
```

Exemple avec fetch (curseur paramétré)

```
declare
        cursor fonctions(wdept dept.deptno%type) is
        select distinct job
        from emp
        where deptno=wdept;

        v_job fonctions%rowtype;
begin
        open fonctions(10);
        loop
                fetch fonctions into v_job;
                exit when fonctions%notfound;
                dbms_output.put_line(v_job.job);
        end loop;
        close fonctions;
end;
/
```

Exemple avec for (curseur non paramétré)

```
declare
        cursor fonctions is
        select distinct job
        from emp;
begin
        for  enreg in fonctions loop
                dbms_output.put_line(enreg.job);
         end loop;
end;
/
```

Exemple avec for (curseur paramétré)

Affichage de la liste des fonctions (job) pour un département donné

```
declare
        cursor fonctions(wdept dept.deptno%type) is
        select distinct job
        from emp
        where deptno=wdept;

        begin
        for enreg in fonctions(&wd) loop
                dbms_output.put_line(enreg.job);
        end loop;
end;
/
```

Modification d’une ligne courante d’un curseur

- Pour modifier une ligne courante d’un curseur lors de son parcours on peut utiliser la clause « **WHERE CURRENT OF** » pour désigner cette ligne.

- Pour utiliser cette technique il faut déclarer le curseur avec la clause **FOR UPDATE OF.**

Exemple avec FETCH

```
declare
        cursor employes is select empno, ename, sal, comm
        from emp where deptno = 10
        for update of emp.sal;
        enreg_emp employes %rowtype;
begin
        open employes ;
        loop
                fetch employes into Enreg_emp;
                exit when employes %notfound;

                if enreg_emp.comm is null then
                    update emp set sal=sal*1.1
                    where current of employes ;
                end if;
        end loop;
        close employes ;
end;
```

Exemple avec FOR

Mettre à jour les salaires des employés du département 10 en cas de commission égale à null

```
declare
        cursor employes is select empno, ename, sal, comm
        from emp where deptno = 10
        for update of emp.sal;
begin
        for Enreg_emp in employes loop
                if enreg_emp.comm is null then
                        update emp set sal=sal*1.1
                        where current of employes ;
                end if;
        end loop;
end;
/
```

5. LES EXCEPTIONS

Une exception est une erreur rencontrée au moment de l'exécution d'un bloc PL/SQL

❖ On distingue 2 types d'exception :

- ➢ prédéfinies par Oracle (internes)
- ➢ définies par le programmeur (externes)

Liste des exceptions prédéfinies :

Exception prédéfinie	Erreur Oracle	Valeur de SQLCODE
ACCESS_INTO_NULL	ORA-06530	-6530
CASE_NOT_FOUND	ORA-06592	-6592
COLLECTION_IS_NULL	ORA-06531	-6531
CURSOR_ALREADY_OPEN	ORA-06511	-6511
DUP_VAL_ON_INDEX	ORA-00001	-1
INVALID_CURSOR	ORA-01001	-1001
INVALID_NUMBER	ORA-01722	-1722
LOGIN_DENIED	ORA-01017	-1017
NO_DATA_FOUND	ORA-01403	+100
NOT_LOGGED_ON	ORA-01012	-1012
PROGRAM_ERROR	ORA-06501	-6501

Exception prédéfinie	Erreur Oracle	Valeur de SQLCODE
ROWTYPE_MISMATCH	ORA-06504	-6504
SELF_IS_NULL	ORA-30625	-30625
STORAGE_ERROR	ORA-06500	-6500
SUBSCRIPT_BEYOND_COUNT	ORA-06533	-6533
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	-6532
SYS_INVALID_ROWID	ORA-01410	-1410
TIMEOUT_ON_RESOURCE	ORA-00051	-51
TOO_MANY_ROWS	ORA-01422	-1422
VALUE_ERROR	ORA-06502	-6502
ZERO_DIVIDE	ORA-01476	-1476

Toutes les autres exceptions doivent être traitées par leur code erreur numérique.

Exemple sans exception:

```
declare
        wename emp.ename%type;
begin
        select ename into wename
        from emp
        where empno=1;
        dbms_output.put_line(wename);
 end;
/
```

ORA-01403: Aucune donnée trouvée

Exemple(1) avec exception:

```
declare
        wename emp.ename%type;
begin
        select ename into wename
        from emp
        where empno=1;
        dbms_output.put_line(wename);
exception
        When no_data_found then
        Dbms_output.put_line('Numéro inexistant');
end;
/
```

Numéro employé inexistant
Procédure PL/SQL terminée avec succès.

Exemple(2) avec exception:

```
declare
        wename emp.ename%type;
begin
      select ename into wename
      from emp
      where empno=1;
      dbms_output.put_line(wename);
exception
      When others then
      dbms_output.put_line( 'Code erreur : ' ||SQLCODE );
      dbms_output.put_line( 'Code erreur : ' ||SQLERRM );
end;
/

Code    erreur : 100
Code    erreur : ORA-01403: Aucune donnée trouvée
Procédure PL/SQL terminée avec succès.
```

Exemple(3) avec exception:

```
declare
        wename emp.ename%type;
begin
        begin
                select ename into wename
                from emp where empno=1;
                exception
                        when no_data_found then
                        dbms_output.put_line( 'Code : ' ||SQLCODE );
                        dbms_output.put_line( 'Code : ' ||SQLERRM );
        end;
        begin
                select ename into wename
                from emp where empno=7369;
                dbms_output.put_line(wename);
        exception
        when others then null;
        end;
end;
```

PRAGMA EXCEPTION_INIT

Nous pouvons associer un code erreur Oracle
à nos propres variables exception à l'aide du mot clé PRAGMA EXCEPTION_INIT:

Nom_exception EXCEPTION ;
PRAGMA EXCEPTION_INIT(nom_exception, -code_error_oracle);

Exemple :
Lorsque on insère une valeur plus longue que la taille d’une chaine, Oracle déclenche une erreur -6502.
Nous allons "nommer" cette erreur en Val_trop_grand

Exemple:

```
Declare
    V varchar2(3) ;
    Val_trop_grand exception ;
    pragma exception_init(Val_trop_grand, -6502 ) ;
Begin
    v := 'smith';
Exception
    when Val_trop_grand then
    dbms_output.put_line( 'Chaîne trop longue') ;
End ;
/
```

```
Chaîne trop longue
Procédure PL/SQL terminée avec succès.
```

Exception de l'utilisateur

En plus des erreurs Oracle, nous pouvons traiter nos propres erreurs en déclarant des variables de type **exception** et qui vont être déclenchées par l'instruction **RAISE**

Syntaxe

```
DECLARE
        Nom_exception Exception ;
        …
BEGIN
        …..
        Raise Nom_exception ;
        …
EXCEPTION
        WHEN Nom_exception Then…….
END ;
```

Exemple:

```
Declare
        v varchar2(3) ;
        x varchar2(10) ;
        slim exception;
Begin
        x := &w;
        if length(x)>3 then
                raise slim;
        else
                v:=x;
        end if;
Exception
        when slim then
        raise_application_error(-20001, 'Chaîne trop longue') ;
 End ;
```

6. LES SOUS PROGRAMMES

LES FONCTIONS STOCKÉES

```
create or replace FUNCTION(<liste paramètres>)RETURN <type retour> IS
-- déclaration des variables
BEGIN
-- code de la fonction
END;
```

Exemple1:

```
Create or replace function somme(n1 number, n2 number) return number is
        S number;
Begin
        S:=n1+n2;
        Return(s);
End;
```

Exemple2:

```
Create or replace function majorer(wsal number) return number is
        sal_maj number;
Begin
        sal_maj:=wsal*1.1;
        Return(sal_maj);
End;
/
```

Utilisation

- Update emp set sal=majorer(sal);
- Select majorer(10) from dual;

LES PROCEDURES STOCKÉES

```
create or replace procedure(<param1 [in/out/in out] type [:=val/default val],
Param_i…) is
-- déclaration des variables
BEGIN
-- code de la procédure
END;
```

Exemple1:

```
Create or replace procedure somme(n1 in number, n2 in number, s out
number) is

Begin

        s:=n1+n2;

End;
```

Utilisation

```
declare
        s number;
begin
        somme(10,12,s);
        dbms_output.put_line(s);
end;
```

Exemple de fonction non stockée

```
declare
   w number;
   function somme2(n1 number,n2 number) return number is
   s number;
   begin
        s:=n1+n2;
        return (s);
   end;
begin
   w:=somme2(5,6);
   dbms_output.put_line(w);
end;
```

Exemple de procédure non stockée

```
Declare

    w number;
    procedure somme2(n1 number,n2 number, s out number) is
    begin
          s:=n1+n2;
    end;

begin
    somme2(5,6,w);
    dbms_output.put_line(w);
end;
/
```

LES PACKAGES

Un paquetage est un ensemble de procédures, de fonctions, de variables, de constantes…regroupées dans un objet nommé.
Ces éléments sont regroupés car ils sont souvent utilisés ensemble.

Un paquetage est organisé en deux parties distinctes:

- **Une partie spécification**

Permet de spécifier à la fois des fonctions et des procédures ainsi que des déclarations des types, variables, constantes, exceptions et curseurs utilisés dans le paquetage.

- **Une partie corps**

Contient les blocs et les spécifications de tous les objets listés dans la partie spécification.

Création de la partie spécification

```
create [or replace] package nom_de_package is/as
Function f1(.....) return type_de_base;
…..
Function fn(…..) return type_de_base;
…..
Procedure p1(…..);
…..
Procedure pn(…..);
…..
End [nom_de_package]
```

Exemple:

```
Create or replace package  p1 is
function majorer(wsal number) return number;
Procedure somme(n1 in number, n2 in number, s out number);
End p1;
```

Création de la partie corps

```
create [or replace] package body nom_de_package is/as

{Declaration de variables};
Function f1(.....) return type_de_base is…;
…..
Function fn(…..) return type_de_base is …;
…..
Procedure p1(…..) is …;
…..
Procedure pn(…..)is …;
…..
[Begin
…]
End [nom_de_package]
```

Exemple:

```
Create or replace package body p1 is

function majorer(wsal number) return number is
sal_maj number;
Begin
        sal_maj:=wsal*1.1;
        Return(sal_maj);
End;

Procedure somme(n1 in number, n2 in number, s out number) is
Begin
        s:=n1+n2;
End;

End p1;

/
```

Utilisation:

```
select p1.majorer(5) from dual;
variable x number;
execute p1.somme(3,4,:x);
```

LES DECLENCHEURS

Un trigger est un bloc PL/SQL qui s'exécute implicitement à chaque fois qu'un événement spécifique se produit.

❖ Temporisation du trigger : BEFORE ou AFTER

❖ Evénement déclencheur : INSERT,UPDATE ou DELETE

❖ Nom de la table : la table sur laquelle le trigger est défini

❖Syntaxe de création :

```
CREATE [OR REPLACE] TRIGGER trigger_name
Before/after       delete/update/insert
On table_name
For each row
Declare
trigger_body
End;
```

Au moment de déclenchement d'un trigger, ORACLE offre deux pseudo records : OLD et NEW de même structure que la table du déclencheur, contenant respectivement les anciennes et les nouvelles valeurs de l'enregistrement en cours.

Exemple1:

```
create or replace trigger archive_emp
before delete  /* ou bien before*/
on emp
for each row
Begin
Dbms_output.put_line('suppression du client :'||:old.ename);
 insert into archive
values(:old.empno,:old.ename,:old.job,:old.MGR,:old.HIREDATE,
:old.SAL,:old.COMM,:old.DEPTNO);
end;
/
```

Exemple2:

```
  create or replace trigger insert_dept
   before  insert
   on dept
   for each row
   Declare
 Wdept number;
    X number;
    Cursor c is select deptno from dept where deptno =:new.deptno;
    Begin
    Open c;
    Fetch c into x;
    If c%found then
     raise_application_error(-20009,'erreur d insertion');
   End if;
Close c;
 end;
/
```

Exemple3:

```
CREATE OR REPLACE TRIGGER comm
BEFORE INSERT OR UPDATE
OF comm
on emp
FOR EACH ROW
BEGIN
IF (:NEW.JOB ='ANALYST')
THEN RAISE_APPLICATION_ERROR (-20010,
'Analyste ne peut pas avoir une commission');
END IF;
END;
/
```

```
create or replace trigger del_emp
before delete or update or insert
on emp
for each row
begin
if deleting then
insert into EMP_ARCHIVE
values(:old.empno,:old.ENAME,:old.JOB,:old.MGR ,:old.HIREDATE
,:old.SAL ,:old.COMM ,:old.DEPTNO,sysdate,'delete',USER);
end if;
if updating then
insert into EMP_ARCHIVE
values(:old.empno,:old.ENAME,:old.JOB,:old.MGR ,:old.HIREDATE
,:old.SAL ,:old.COMM ,:old.DEPTNO,sysdate,'update',USER);
end if;
if inserting then
if (:new.deptno=9) then
raise_application_error(-20009,'erreur insertion');
end if;
end if;

end;
```

Désactivation ou réactivation d'un trigger de base de données :

ALTER TRIGGER trigger_name DISABLE | ENABLE

Désactivation ou réactivation de tous les triggers d'une table :

ALTER TRIGGER table_name DISABLE | ENABLE ALL TRIGGERS

Recompilation d'un trigger pour une table :

ALTER TRIGGER trigger_name COMPILE

Pour supprimer un trigger de la base de données:

DROP TRIGGER trigger_name