

Les blocs anonymes

- Les programmes écrits en PL / SQL respectent une structure en blocs.

- Structure: (excp)

DECLARE

 v last name varchar(2) ;

BEGIN

 Select last name into v last name from employees where employee id = 124;

END;

/ ↗ Exécuter un bloc PL / SQL anonyme

- Le type Record:

* Déclarer l'enregistrement

* Initialiser les composants interne de ce type d'enregistrement.

(excp):

```
TYPE emp-second-type IS RECORD
  (last-name VARCHAR2(25),
   job-id VARCHAR(10),
   salary employees.salary%type);
emp-second emp-second-type;
```

- Initialisation des variables:

Initialisation

* Par l'opérateur d'affectation :=

* par le mot réservé **DEFAULT**

* Affectation des variables de la même manière (avec :=) dans la section exécutable (**BEGIN..END**).

- Les structures de contrôle:

* **IF** Condition **THEN**
instruction

ELSIF Condition2 THEN
instruction

ELSE
instruction

END IF;

- L'identifiant ne doit pas dépasser 30 caractères.

- Le premier caractère doit être une lettre.

(excp):

DECLARE

 v-char varchar(30) NOT NULL := 'SGBD'

 v-date date DEFAULT '01-JANU-1997';

 v-constant CONSTANT integer := 10;

(excp):

BEGIN

 IF v1 < v2 THEN

 dbms_output.put_line('v1 < v2');

 ELSE

 dbms_output.put_line('v2 < v1');

 END IF;

END;

/

* CASE expression

```
WHEN valeur1 THEN  
    instruction 1;  
    :  
ELSE  
    instructionN;  
END CASE;
```

(exp):

```
BEGIN  
CASE v1  
WHEN 1 THEN  
    dbms-output.put-line('A');  
ELSE  
    dbms-output.put-line('X');  
END CASE;  
END;  
/
```

* CASE

```
WHEN exp1 THEN  
    instruction 1;  
    :  
ELSE  
    instructionN;  
END CASE;
```

(exp):

```
BEGIN  
CASE  
WHEN v1 < 5 THEN  
    dbms-output.put-line('A');  
ELSE  
    dbms-output.put-line('X');  
END CASE;  
END;  
/
```

* WHILE condition LOOP

```
instructions;  
END LOOP;
```

(exp):

```
BEGIN  
WHILE v1 < 10 LOOP  
    dbms-output.put-line(v1);  
    v1 := v1 + 1;  
END LOOP;  
END;  
/
```

* LOOP

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

(exp):

```
BEGIN  
LOOP  
    dbms-output.put-line(v1);  
    EXIT WHEN v1 = 10;  
    v1 := v1 + 1;  
END LOOP;  
END;  
/
```

* FOR compteur IN [REVERSE]inf..sup Loop

```
instructions;  
END LOOP;
```

(exp):

```
BEGIN  
FOR v1 IN 1..10 LOOP  
    dbms-output.put-line(v1);  
END LOOP;  
END;  
/
```

- L'instruction SELECT :

SELECT Col1 Col2 **INTO** v_Col1,v_Col2 FROM table1,table2
 \ identifier les variables PL/SQL

(ex):

```
DECLARE
  salaire-moy employees.salary %TYPE;
BEGIN
  Select avg(salary) into salaire-moy FROM employees WHERE
  department-id = 100 ;
  dbms-output.put-line ('Le salaire moyen des employés est : ' ||
    to-char (salaire-moy, '99999.99'));
END;
/
```

DECLARE

```
v-emp employees %ROWTYPE;
BEGIN
  Select * into v-emp FROM employees WHERE employee-id=124;
  dbms-output.put-line ('Nom employé : ' || v-emp.last-name || chr(10) ||
    'Fonction: ' || v-emp.job-id || chr(10) || 'Département: ' ||
    to-char (v-emp.hire-date, 'dd/mm/yyyy') || chr(10) || 'Salaire: ' ||
    v-emp.salary);
END;
/
```

Les curseurs

* Curseur : variable qui pointe vers le résultat d'une requête.

- **Implicites**: déclarés implicitement et manipulés par SQL.

* pour les requêtes du CID et les interrogations retournant un seul enregistrement.

- **Explicites**: * déclarés et manipulés par l'utilisateur.

* pour les interrogations qui retournent plus d'un enregistrement.

- Manipulation du curseur:

Ouverture du curseur (open) (et allocation mémoire pour stocker le résultat)

Extraction d'une ligne (fetch)

Fermeture du curseur (close) (et libération de la zone mémoire)

- Déclaration:

CURSOR nom IS instruction select ;

(expt):

DECLARE

CURSOR cur_emp IS

SELECT employee_id, first_name, last_name FROM employees
WHERE department_id = 100;

CURSOR cur_dept IS

SELECT * FROM departments ORDER BY department_id ;

- Ouverture :

BEGIN

OPEN cur_emp;

END;

} suite

- Exécution:

. Lecture de la zone pointée par le curseur.

. Affectation des valeurs de la ligne courante dans les variables de sortie.

. Passage à l'enregistrement suivant.

(exp) :

DECLARE

```

CURSOR cur-emp IS
SELECT employee_id, first-name, last-name FROM employees
WHERE department_id = 100;
V-no employees.employee_id %TYPE;
V-Fname employees.first-name %TYPE;
V-Lname employees.last-name %TYPE;
BEGIN
IF NOT cur-emp %IS OPEN THEN
OPEN cur-emp;
END IF;
FETCH cur-emp INTO V-no, V-Fname, V-Lname;
WHILE cur-emp %FOUND
Loop
dbms_output.put_line ('Employé n°' || V-no || 'Nom:' || V-Fname ||
'Prénom:' || V-Lname);
FETCH cur-emp INTO V-no, V-Fname, V-Lname;
END Loop;
CLOSE cur-emp;
END;

```

- Attributs des curseurs:

nom-attribut % ATTRIBUT

- %IS OPEN : vrai si ouvert
- %FOUND : vrai si FETCH réussie
- %NOTFOUND : inverse de FOUND
- %ROWCOUNT : renvoie le nb de lignes

- Utilisation simplifiée des curseurs:

* Utilisation du type RECORD : (dont les éléments sont du même type que les colonnes rentrées par le curseur)

Exemple:

```

DECLARE
CURSOR cur-emp IS SELECT employee_id FROM employees;
rec-emp cur-emp %ROWTYPE;
BEGIN
OPEN cur-emp;
Loop
FETCH cur-emp INTO rec-emp;
EXIT WHEN cur-emp %NOTFOUND;
dbms_output.put_line ('Employee n°' || rec-emp.employee_id || );
END Loop;
CLOSE cur-emp;
END;

```

* Utilisation de la structure FOR..IN : (pour éviter de déclarer l'enregistrement hôte dans la partie déclarée)

Exemp:

```
DECLARE
    CURSOR cur-emp IS SELECT employee-id FROM employees;
BEGIN
    FOR rec-emp IN cur-emp
    LOOP
        dbms-output.put-line ('Employé n°: ' || rec-employee-id);
    END LOOP;
END;
```

* Utilisation de sous-requête dans la requête FOR..IN : (éviter de déclarer le curseur dans la partie DECLARE)

Exemp:

```
BEGIN
    FOR rec-emp IN (select employee-id from employees)
    LOOP
        dbms-output.put-line ('Employé n°! ' || rec-emp.employee-id || );
    END LOOP;
END;
```

- Curseurs paramétrés :

- Paramétrer la requête associée à un curseur pour éviter de multiplier les curseurs similaires.
- Il faut fermer le curseur avant de l'appeler avec d'autres valeurs pour les paramètres (sauf pour la boucle FOR qui ferme automatiquement)

Exemp:

```
DECLARE
    CURSOR cur-emp (v-dept number, v-sal employees.salary%TYPE) IS
        Select employee-id, last-name, salary FROM employees WHERE
        department-id = v-dept and salary > v-sal;
    rec-emp cur-emp%ROWTYPE;
BEGIN
    OPEN cur-emp(80, 6000);
    LOOP
        FETCH cur-emp INTO rec-emp;
        EXIT when cur-emp%NOTFOUND;
        dbms-output.put-line ('Employé n: ' || rec-emp.employee-id ||
        ' Nom: ' || rec-emp.last-name || ' Salaire: ' || rec-emp.salary#);
    END LOOP;
    CLOSE cur-emp;
END;
```

Les sous-programmes: procédures & fonctions

- Les procédures et fonctions stockées (autonomes):

sous programme PL/SQL conservé dans une BD ORACLE et appelé par un utilisateur, directement ou indirectement.

Efficacité: minimiser l'appel des requêtes SQL côté client en les remplaçant par des appels de procédures côté serveur.

Réutilisabilité: peuvent être réutilisées.

Portable: une procédure est indépendante de la version du SE ou du compilateur.

Maintenabilité: en appelant la même procédure à partir de plusieurs outils.

- Les procédures stockées :

3 modes de passage de paramètre:

IN, OUT, IN OUT

Exemple:

```
CREATE OR REPLACE PROCEDURE add-dept (dept-id IN
    department.department_id%TYPE, nbre OUT number) IS
    déclaration variables locales (sans écrire DECLARE)
Begin
    insert into departments (department_id) values (dept-id);
    commit;
    select count(*) into nbre from departments;
    dbms_output.put_line ('Le nbre de départements est : ' || nbre);
END;
```

- Les fonctions stockées :

tous les paramètres en mode IN (on l'écrit pas)

Exemple:

```
CREATE OR REPLACE FUNCTION fn-check-sal (empno
    employees.employee_id%TYPE) RETURN Boolean IS
    dept-id employees.department_id%TYPE;
    sal employees.salary%TYPE;
    aug-sal employees.salary%TYPE;
BEGIN
    Select salary, department_id into sal, dept-id from employees where
        employee_id = empno;
    select aug(salary) into aug-sal from employees where department_id = dept-id;
    IF sal > aug-sal THEN
        Return TRUE;
    ELSE RETURN FALSE;
END IF;
END;
```

Appel de fonctions stockées:

Exemple:

```
BEGIN  
  IF (fn-check-sal(124)) THEN  
    dbms-output.put-line ('salary > average');  
  ELSE  
    dbms-output.put-line ('salary < average');
```

Remarques :

- Lors de la création d'un objet, des entrées sont créées dans la table user-objects. Pour la consulter:
`SELECT object-name, object-type FROM user-objects;`
- Le code source d'une procédure ou fonction est enregistré dans la table user-source. Pour consulter:
`SELECT * FROM user-source WHERE name = 'FN';`
- Consulter les arguments et le type renvoyé : `DESCRIBE FN-Check;`
- Les procédures et fonctions non stockées (non autonomes) :

- Les procédures non stockées:

Exemple:

```
DECLARE  
  a number := 10;  
  b number := 30;  
  s number;  
  p number;  
PROCEDURE myproc (a in number, b in  
  number, s out number,  
  p out number) IS  
BEGIN  
  surf := a * b;  
  p := (a+b)^2;  
END myproc;  
BEGIN  
  myproc(a, b, s, p);  
  dbms-output.put-line('surface' || s);  
  dbms-output.put-line('perimetre' || p);  
END;
```

- Les fonctions non stockées:

Exemple:

```
DECLARE  
  a number := 10;  
  b number := 30;  
  c number;  
FUNCTION myfunc (a number, b number)  
RETURN number IS  
  surf number;  
BEGIN  
  surf := a * b;  
  return surf;  
END;  
BEGIN  
  c := myfunc(a, b);  
  dbms-output.put-line('Surface' || c);  
END;
```

Les exceptions

- Exceptions déclenchées implicitement :

Exception

```
WHEN exception1 [OR exception2] THEN
    instruction1;
WHEN OTHERS THEN
    instruction;
```

« Exceptions prédefinies :

- 6511 . **CURSOR_ALREADY_OPEN** : Ouverture d'un curseur déjà ouvert.
- 1 . **DUPLICATE_ON_INDEX** : Insertion d'une clé dupliquée.
- 1722 . **INVALID_NUMBER** : Echec sur une conversion de chaîne en nombre.
- 100 . **NO_DATA_FOUND** : Select retournant 0 ligne.
- 1422 . **TOO_MANY_ROWS** : Select retournant plus d'une ligne.
- 6502 . **VALUE_ERROR** : Erreur arith. de conversion, de troncature ou limite de taille.
- 1476 . **ZERO_DIVIDE** : Division par zéro.

Exp :

```
DECLARE
    v_emp employees%ROWTYPE;
BEGIN
    BEGIN
        select * into v_emp from employees;
    EXCEPTION
        WHEN TOO_MANY_ROWS THEN dbms_output.put_line('Plusieurs lignes');
    END;
END;
```

« Exceptions non prédefinies :

Exemple :

```
DECLARE a Nom
    erreur_valeur exception;
    pragma exception_init(erreur_valeur, -6502);
    vnom varchar2(10);
BEGIN
    vnom := 'quelbani foud';
EXCEPTION
    WHEN erreur_valeur THEN
        dbms_output.put_line('Erreur de troncature');
END;
```

- Exceptions ~~définies~~ déclenchées explicitement:

* Exceptions définies par l'utilisateur :

Excp:

DECLARE

valeur-positive exception;
valeur-negative exception;
valeur-nulle exception;
x number := 0;

BEGIN

case

when $x > 0$ then raise valeur-positive;
when $x < 0$ then raise valeur-negative;
else raise valeur-nulle;

end case;

EXCEPTION

WHEN valeur-positive THEN dbms_output.put_line ('Positif');
WHEN valeur-negative THEN dbms_output.put_line ('Négatif');
WHEN OTHERS THEN dbms_output.put_line ('Nul');

END;

Déclencher explicitement
l'exception avec RAISE

- Fonctions d'interception des erreurs:

* SQLCODE: Renvoie la valeur numérique associée au code de l'erreur.

* SQLERRM: Renvoie le message ~~numérique~~ associé au code de l'erreur.

Excp:

DECLARE

code-erreur number;
message-erreur varchar2(55);
v-emp employees%ROWTYPE;

BEGIN

SELECT * into v-emp FROM employees WHERE employee_id = 1;

EXCEPTION

WHEN OTHERS THEN

dbms_output.put_line ('Code err: ' || SQLCODE || 'Message:' || SQLERRM);

END;

- Procedure RAISE_APPLICATION_ERROR :

- * Permet de définir des erreurs utilisateurs.
- * Annule la transaction en cours.
- * Peut être utilisée dans la section Executable ou Exception.

Exemple :

```
drop table x1
create table x1 (id number primary key, nom varchar2(25))
```

DECLARE

```
    uid number;
    unom varchar2(25);
```

BEGIN

```
    For uid in 1..50
```

Loop

```
        insert into x1 values (uid, 'nom'||uid);
```

End Loop;

```
    update x1 set nom = upper(nom);
```

Commit;

```
    For uid in reverse 40..60
```

Loop

```
        insert into x1 values (uid, 'nom'||uid);
```

End loop;

EXCEPTION

```
    WHEN DUP_VAL_ON_INDEX THEN
```

```
        raise_application_error (-20001, 'duplicé');
```

END;

Les déclencheurs (TRIGGERS)

- Les déclencheurs permettent de :

- * Intégrité des données.
- * Intégrité référentielle.
- * Calcul de données dérivées.
- * Journalisation des événements.
- * Sécurité.

- Déclencheur de niveau instruction :

s'exécute une seule fois qdq le nb de lignes traitées.

Excp :

drop table emp1

create table emp1 as select * from employees;

CREATE OR REPLACE TRIGGER trig-emp1-before-insert-global
BEFORE INSERT ON emp1

BEGIN

dbms_output.put_line('Début insertion' || To_char(sysdate,
'Day dd-mm-yyyy hh:mm:ss'));

END;

CREATE OR REPLACE TRIGGER trig-emp1-after-insert-global
AFTER INSERT ON emp1

BEGIN

dbms_output.put_line('Fin d"insertion'" || To_char(sysdate,
'Day dd-mm-yyyy hh:mm:ss'));

END;

BEGIN

insert into emp1 select * from employees;

dbms_output.put_line(SQL%ROWCOUNT || ' insertion effectuée');

END;

- Déclencheur de niveau ligne:

s'exécute une fois pour chaque ligne.

- Variables Record OLD et NEW:

: NEW est utilisée dans une instruction INSERT ou UPDATE.

: OLD est utilisée dans une instruction UPDATE ou DELETE.

Excp:

```
CREATE OR REPLACE TRIGGER emp1_before_insert BEFORE INSERT
ON emp1
```

FOR EACH ROW

WHEN (new.employee_id between 120 and 130)

BEGIN

: New.employee_id := : New.employee_id + 1000;

: New.lastname := upper(: New.lastname);

: New.firstname := lower(: New.firstname);

END;

BEGIN

insert into emp1 select * from employees;

dbms_output.put_line('SQL % RowCount || ' insertion effectuée');

END;

- Predicats INSERTING, UPDATING & DELETING :

~~Le~~ - Le trigger se déclenche par une instruction LID.

- utilisés pour identifier l'instruction en cours.

Excp:

.drop table x1

.create table x1 (c1 number primary key, c2 varchar2(50))

```
CREATE OR REPLACE TRIGGER trig_x1 BEFORE INSERT OR UPDATE
OR DELETE ON x1
```

BEGIN

IF INSERTING THEN dbms_output.put_line('INSERTION'); END IF;

IF UPDATING THEN dbms_output.put_line('MODIF'); END IF;

IF DELETING THEN dbms_output.put_line('SUPPR'); END IF;

END;

BEGIN

FOR i IN 1..1000

Loop

insert into x1 values (i, 'nom' || i);

END Loop;

update x1 set c2 = upper(c2) where c1 <= 100;

delete x1 from x1 where c1 > 800;

END;

- Activation / Désactivation des triggers:

ALTER TRIGGER nom [ENABLE | DISABLE]

ALTER TABLE nom [ENABLE | DISABLE] ALL TRIGGERS

- Recherche des triggers:

Informations relatives aux triggers:

- USER-TRIGGERS
- ALL-TRIGGERS
- DBA-TRIGGERS

Exp:

SELECT * FROM USER-TRIGGERS WHERE trigger_name like emp%;