1 Déclarer des variables

ORACLE

Objectifs

- reconnaître un bloc PL/SQL de base et ses différentes sections
- décrire la signification des variables en PL/SQL
- déclarer des variables PL/SQL
- exécuter un bloc PL/SQL

ORACLE

Structure d'un bloc PL/SQL

`DECLARE` (facultatif)

Variables, curseurs, exceptions définies par l'utilisateur

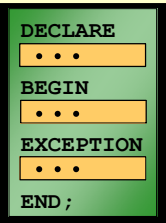
`BEGIN` (obligatoire)

- Instructions SQL
- Instructions PL/SQL

`EXCEPTION` (facultatif)

Actions à réaliser lorsque des erreurs se produisent

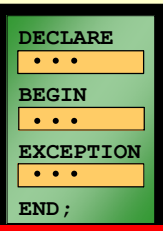
`END;` (obligatoire)



ORACLE

Exécuter des instructions et des blocs PL/SQL

```
DECLARE
  v_variable  VARCHAR2(5);
BEGIN
  SELECT column_name
  INTO v_variable
  FROM table_name;
EXCEPTION
  WHEN exception_name THEN
  ...
END;
```



ORACLE

Types de bloc

Anonyme

```
[DECLARE]

BEGIN
  --statements

[EXCEPTION]

END;
```

Procédure

```
PROCEDURE name
IS

BEGIN
  --statements

[EXCEPTION]

END;
```

Fonction

```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;
[EXCEPTION]

END;
```

ORACLE

Utiliser des variables

Les variables peuvent être utilisées pour les raisons suivantes :

- stockage temporaire de données
- manipulation de valeurs stockées
- possibilité de réutilisation
- facilité de maintenance

ORACLE

Traiter les variables en PL/SQL

- Déclarer et initialiser les variables dans la section déclarative
- Affecter de nouvelles valeurs aux variables dans la section exécutable
- Transmettre des valeurs aux blocs PL/SQL via des paramètres
- Afficher les résultats via des variables de sortie

ORACLE

Utiliser des variables SQL*Plus dans des blocs PL/SQL

- Le langage PL/SQL ne permet pas les entrées/sorties
- Vous pouvez référencer des variables de substitution dans un bloc PL/SQL en les faisant précéder d'un signe esperluette
- Les variables attachées ou variables hôte SQL*Plus peuvent être utilisées pour transmettre des valeurs d'exécution du bloc PL/SQL vers l'environnement SQL*Plus

ORACLE

Déclarer des variables PL/SQL

Syntaxe :

```
identifier [CONSTANT] datatype [NOT NULL]
     [:= | DEFAULT expr];
```

Exemples :

```
DECLARE
  v_hiredate      DATE;
  v_deptno        NUMBER(2) NOT NULL := 10;
  v_location      VARCHAR2(13) := 'Atlanta';
  c_comm          CONSTANT NUMBER := 1400;
```


Règles relatives à la déclaration de variables PL/SQL

- Suivre les conventions d'appellation
- Initialiser les variables `NOT NULL` et `CONSTANT`
- Déclarer un identificateur par ligne
- Initialiser les identificateurs en utilisant l'opérateur d'affectation (`:=`) ou le mot réservé `DEFAULT`

```
identifier := expr;
```


Règles d'appellation

- Deux variables peuvent porter le même nom si elles sont dans des blocs distincts
- Le nom de la variable (identificateur) doit être différent du nom des colonnes de table utilisées dans le bloc.

```
DECLARE
  employee_id  NUMBER(6);
BEGIN
  SELECT    employee_id
  INTO      employee_id
  FROM      employees
  WHERE     last_name = 'Kochhar';
END;
/
```

Adopter une convention d'appellation pour les identificateurs PL/SQL : par exemple, v_employee_id

Initialisation de variables et mots-clés

- Opérateur d'affectation (`:=`)
- Mot-clé `DEFAULT`
- Contrainte `NOT NULL`

Syntaxe :

```
identifier := expr;
```

Exemples :

```
v_hiredate := '01-JAN-2001';
```

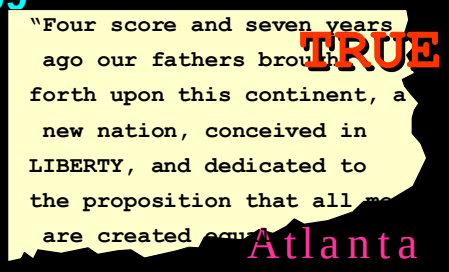
```
v_ename := 'Maduro';
```

Types de données scalaires

- Ils stockent une seule valeur
- Ils n'ont pas de composant interne

25-OCT-99

256120.08



Types de données scalaires de base

- CHAR [(*maximum_length*)]
- VARCHAR2 (*maximum_length*)
- LONG
- NUMBER [(*precision, scale*)]
- BINARY_INTEGER
- PLS_INTEGER
- BOOLEAN
- DATE

Déclaration de variables scalaires

Exemples :

```
DECLARE
  v_job          VARCHAR2(9);
  v_count        BINARY_INTEGER := 0;
  v_total_sal    NUMBER(9,2) := 0;
  v_orderdate    DATE := SYSDATE + 7;
  c_tax_rate     CONSTANT NUMBER(3,2) := 8.25;
  v_valid        BOOLEAN NOT NULL := TRUE;
  ...
```


Attribut %TYPE

- Déclarer une variable d'après :
  - une définition de colonne de base de données
  - une autre variable précédemment déclarée
- Faire précéder %TYPE :
  - de la table et de la colonne de base de données
  - du nom de la variable précédemment déclarée

Déclarer des variables avec l'attribut %TYPE

Syntaxe :

```
identifier      Table.column_name%TYPE;
```

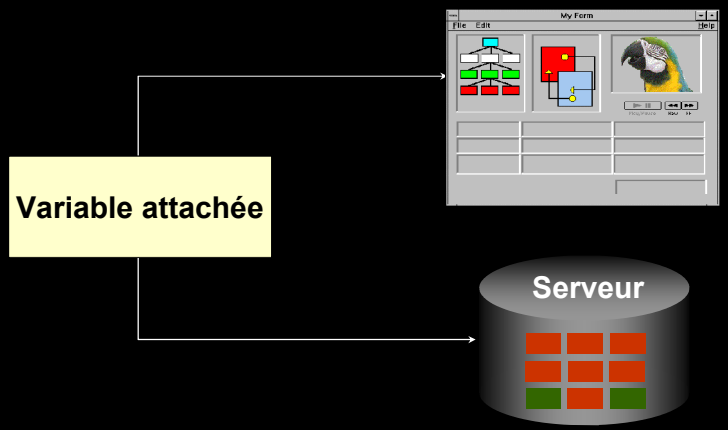
Exemples :

```
...
  v_name            employees.last_name%TYPE;
  v_balance         NUMBER(7,2);
  v_min_balance     v_balance%TYPE := 10;
...
```


Déclarer des variables booléennes

- Seules les valeurs TRUE, FALSE et NULL peuvent être affectées à une variable booléenne
- Les variables sont comparées en utilisant les opérateurs logiques AND, OR et NOT
- Les variables renvoient toujours TRUE, FALSE ou NULL
- Des expressions de type arithmétique, caractère ou date peuvent être utilisées pour renvoyer une valeur booléenne.

Variables attachées



Utiliser des variables attachées

En PL/SQL, pour référencer une variable attachée, il faut faire précéder son nom du signe deux-points (:)

Exemple :

```
VARIABLE      g_salary NUMBER
BEGIN
   SELECT     salary
   INTO       :g_salary
   FROM       employees
   WHERE      employee_id = 178;
END;
/
PRINT g_salary
```

Référencer des variables non PL/SQL

Stocker le salaire annuel dans une variable hôte SQL*Plus.

```
:g_monthly_sal := v_sal / 12;
```

- Référencer des variables non PL/SQL en tant que variables hôte
- Faire précéder les références à l'aide d'un signe deux-points (:)

DBMS_OUTPUT.PUT_LINE

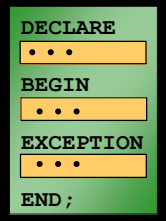
- Procédure de package fournie par Oracle
- Autre méthode d'affichage des données d'un bloc PL/SQL
- Doit être activé dans SQL*Plus via `SET SERVEROUTPUT ON`

```
SET SERVEROUTPUT ON
DEFINE p_annual_sal = 60000
```

```
DECLARE
  v_sal NUMBER(9,2) := &p_annual_sal;
BEGIN
  v_sal := v_sal/12;
  DBMS_OUTPUT.PUT_LINE ('The monthly salary is ' ||
                        TO_CHAR(v_sal));
END;
/
```


Synthèse

- Les blocs PL/SQL sont composés des sections suivantes :
  - déclarative (facultative)
  - exécutable (obligatoire)
  - traitement des exceptions (facultative)
- Un bloc PL/SQL peut être un bloc, une procédure ou une fonction anonyme.



Synthèse

- Les identificateurs PL/SQL :
  - sont définis dans la section déclarative
  - peuvent être initialisés
- Les variables déclarées dans un environnement externe tel que SQL*Plus sont appelées variables hôte
- `DBMS_OUTPUT.PUT_LINE` permet d'afficher les données d'un bloc PL/SQL.

Présentation de l'exercice 1

Dans cet exercice, vous allez :

- déterminer la validité de quelques déclarations
- déclarer un bloc PL/SQL simple
- exécuter un bloc PL/SQL simple

ORACLE

2 Ecrire des instructions exécutables

ORACLE

Objectifs

A la fin de ce chapitre, vous pourrez :

- comprendre l'utilité de la section exécutable
- utiliser correctement les identificateurs
- écrire des instructions dans la section exécutable
- décrire les règles des blocs imbriqués
- exécuter et tester un bloc PL/SQL
- utiliser des conventions de codage

ORACLE

Syntaxe et remarques relatives aux blocs PL/SQL

- Les instructions peuvent s'étendre sur plusieurs lignes
- Les unités lexicales se répartissent en plusieurs catégories :
  - délimiteurs
  - identificateurs
  - littéraux
  - commentaires

ORACLE

Identificateurs

- Peuvent contenir jusqu'à 30 caractères
- Doivent commencer par une valeur alphabétique
- Peuvent contenir des valeurs numériques, des traits de soulignement, ainsi que les signes dollar et dièse
- Ne doivent pas contenir de caractères tels que les traits d'union et les barres obliques, ni d'espaces
- Ne doivent pas porter le même nom qu'une colonne de table de la base de données
- Ne doivent pas correspondre à des mots réservés

Syntaxe et remarques relatives aux blocs PL/SQL

- Littéraux
  - Les littéraux de type caractère et date doivent être mis entre apostrophes.

```
v_name := 'Henderson';
```

  - Les nombres peuvent correspondre à des valeurs simples ou à une notation scientifique.
- Une barre oblique (/) permet d'exécuter le bloc PL/SQL dans un fichier script ou dans certains outils tels que SQL*PLUS.

Commenter le code

- Faire précéder les commentaires monolignes de deux tirets (--)
- Placer les commentaires multilignes entre les symboles /* et */

Exemple :

```
DECLARE
...
  v_sal NUMBER (9,2);
BEGIN
  /* Compute the annual salary based on the
     monthly salary input from the user */
  v_sal := :g_monthly_sal * 12;
END;     -- This is the end of the block
```


Fonctions SQL en PL/SQL : exemples

- Créer l'adresse postale d'une société

```
v_mailing_address := v_name||CHR(10)||
                     v_address||CHR(10)||v_state||
                     CHR(10)||v_zip;
```

- Convertir le nom d'un employé en minuscules

```
v_ename       := LOWER(v_ename);
```

Conversion de type de données

- Convertir des données en types de données comparables
- Mélanger les types de données peut provoquer des erreurs ou nuire aux performances
- Fonctions de conversion :
  - TO_CHAR
  - TO_DATE
  - TO_NUMBER

```
DECLARE
   v_date DATE := TO_DATE('12-JAN-2001', 'DD-MON-YYYY');
BEGIN
   . . .
```


Conversion de type de données

L'instruction suivante génère une erreur de compilation si la variable `v_date` est déclarée en tant que type de données `DATE` .

```
v_date := 'January 13, 2001';
```


Conversion de type de données

Pour corriger cette erreur, il faut utiliser la fonction de conversion `TO_DATE` .

```
v_date := TO_DATE ('January 13, 2001',
                   'Month DD, YYYY');
```


Blocs imbriqués et portée des variables

- Les blocs PL/SQL peuvent être imbriqués partout où une instruction exécutable est permise
- Un bloc imbriqué devient une instruction
- Une section de traitement des exceptions peut contenir des blocs imbriqués
- La portée d'un identificateur correspond à la région d'un programme (bloc, sous-programme ou package) à partir de laquelle vous pouvez référencer l'identificateur

Blocs imbriqués et portée des variables

Exemple :

```
...
  x  BINARY_INTEGER;
BEGIN
  ...
  DECLARE
    y  NUMBER;
  BEGIN
      y:= x;
  END;
  ...
END;
```

Portée de *x*

Portée de *y*

Portée de l'identificateur

Un identificateur est visible dans les régions à partir desquelles vous pouvez le référencer sans devoir le qualifier :

- un bloc peut effectuer une recherche dans le bloc englobant
- un bloc ne peut pas effectuer de recherche dans les blocs qu'il englobe

Qualifier un identificateur

- Le qualificatif peut correspondre à l'étiquette d'un bloc englobant
- Qualifier un identificateur en utilisant l'étiquette du bloc en tant que préfixe

```
<<outer>>
   DECLARE
      birthdate DATE;
   BEGIN
         DECLARE
             birthdate DATE;
         BEGIN
            ...
            outer.birthdate :=
                          TO_DATE('03-AUG-1976',
                                  'DD-MON-YYYY');
         END;
    ....
    END;
```


Déterminer la portée d'une variable

Exercice

```
<<outer>>
DECLARE
  v_sal       NUMBER(7,2) := 60000;
  v_comm      NUMBER(7,2) := v_sal * 0.20;
  v_message   VARCHAR2(255) := ' eligible for commission';
BEGIN
  DECLARE
        v_sal           NUMBER(7,2) := 50000;
        v_comm          NUMBER(7,2) := 0;
        v_total_comp    NUMBER(7,2) := v_sal + v_comm;
  BEGIN
        v_message := 'CLERK not'||v_message;
        outer.v_comm := v_sal * 0.30;

  END;
     v_message := 'SALESMAN'||v_message;
END;
```

1

2

Opérateurs en PL/SQL

- Opérateur logique
- Opérateur arithmétique
- Opérateur de concaténation
- Parenthèses permettant de contrôler l'ordre des opérations

} Identiques en SQL

- Opérateur exponentiel (**)

Opérateurs en PL/SQL

Exemples :

- Incrémenter le compteur d'une boucle.

```
v_count       := v_count + 1;
```

- Définir la valeur d'un indicateur booléen.

```
v_equal       := (v_n1 = v_n2);
```

- Vérifier si un numéro d'employé contient une valeur.

```
v_valid       := (v_empno IS NOT NULL);
```


Remarques relatives à la programmation

Faciliter la maintenance du code en :

- commentant le code
- développant une convention d'utilisation des majuscules et des minuscules
- développant des conventions d'appellation pour les identificateurs et les autres objets
- réalisant des indentations pour améliorer la clarté

Indenter le code

Pour plus de clarté, indenter chaque niveau du code

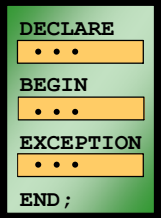
Exemple :

```
BEGIN
  IF x=0 THEN
     y:=1;
  END IF;
END;
```

```
DECLARE
  v_deptno       NUMBER(4);
  v_location_id  NUMBER(4);
BEGIN
  SELECT  department_id,
          location_id
  INTO    v_deptno,
          v_location_id
  FROM    departments
  WHERE   department_name
          = 'Sales';
...
END;
/
```

Synthèse

- syntaxe et remarques relatives aux blocs PL/SQL
- utilisation correcte des identificateurs
- structure des blocs PL/SQL : imbrication des blocs et règles de portée
- programmation en PL/SQL :
  - fonctions
  - fonctions de conversion de type de données
  - opérateurs
  - conventions et remarques



Présentation de l'exercice 2

Dans cet exercice, vous allez :

- revoir les règles de portée et d'imbrication
- développer et tester des blocs PL/SQL

3 Interagir avec le serveur Oracle

Objectifs

- écrire une instruction SELECT correcte en PL/SQL
- écrire des instructions LMD en PL/SQL
- contrôler des transactions en PL/SQL
- déterminer le résultat de l'exécution d'instructions SQL LMD (Langage de manipulation de données)

Instructions SQL en PL/SQL

- Extraire une ligne de données à partir d'une base de données en utilisant la commande SELECT
- Modifier des lignes de la base de données en utilisant des instructions LMD
- Contrôler une transaction avec la commande COMMIT, ROLLBACK ou SAVEPOINT
- Déterminer le résultat de l'exécution d'une instruction LMD avec des attributs de curseur implicite

Instructions SELECT en PL/SQL

Extraire les données de la base de données à l'aide d'une instruction SELECT

Syntaxe :

```
SELECT  select_list
INTO    {variable_name[, variable_name]...
        | record_name}
FROM    table
[WHERE  condition];
```


Instructions SELECT en PL/SQL

- La clause INTO est obligatoire
- Les interrogations doivent renvoyer une et une seule ligne

Exemple :

```
DECLARE
  v_deptno           NUMBER(4);
  v_location_id      NUMBER(4);
BEGIN
  SELECT     department_id, location_id
  INTO       v_deptno, v_location_id
  FROM       departments
  WHERE      department_name = 'Sales';
  ...
END;
/
```


Extraire des données en PL/SQL

Extraire la date d'embauche et le salaire de l'employé indiqué

Exemple :

```
DECLARE
  v_hire_date   employees.hire_date%TYPE;
  v_salary      employees.salary%TYPE;
BEGIN
  SELECT   hire_date, salary
  INTO     v_hire_date, v_salary
  FROM     employees
  WHERE    employee_id = 100;
  ...
END;
/
```

Extraire des données en PL/SQL

Renvoyer la somme des salaires de tous les employés du service indiqué

Exemple :

```
SET SERVEROUTPUT ON
DECLARE
  v_sum_sal    NUMBER(10,2);
  v_deptno     NUMBER NOT NULL := 60;
BEGIN
  SELECT       SUM(salary)  -- group function
  INTO         v_sum_sal
  FROM         employees
  WHERE        department_id = v_deptno;
  DBMS_OUTPUT.PUT_LINE ('The sum salary is ' ||
                        TO_CHAR(v_sum_sal));
END;
/
```


Conventions d'appellation

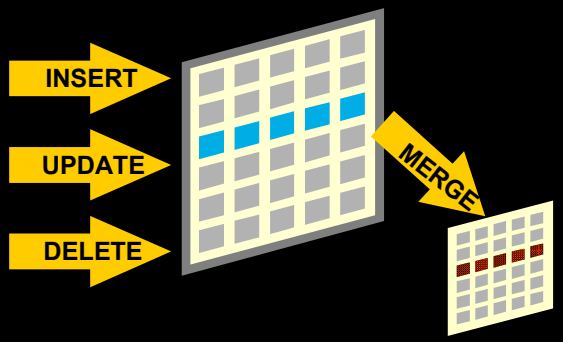
```
DECLARE
  hire_date       employees.hire_date%TYPE;
  sysdate         hire_date%TYPE;
  employee_id     employees.employee_id%TYPE := 176;
BEGIN
  SELECT       hire_date, sysdate
  INTO         hire_date, sysdate
  FROM         employees
  WHERE        employee_id = employee_id;
END;
/
```

```
DECLARE
*
ERROR at line 1:
ORA-01422: exact fetch returns more than requested number of rows
ORA-06512: at line 6
```


Manipuler les données en PL/SQL

Modifier des tables de base de données en utilisant les instructions LMD suivantes :

- INSERT
- UPDATE
- DELETE
- MERGE



Insérer des données

Ajouter les informations relatives à un nouvel employé à la table EMPLOYEES

Exemple :

```
BEGIN
  INSERT INTO employees
  (employee_id, first_name, last_name, email,
   hire_date, job_id, salary)
  VALUES
   (employees_seq.NEXTVAL, 'Ruth', 'Cores', 'RCORES',
    sysdate, 'AD_ASST', 4000);
END;
/
```

Mettre à jour des données

Augmenter le salaire de tous les employés chargés du contrôle des stocks

Exemple :

```
DECLARE
  v_sal_increase   employees.salary%TYPE := 800;
BEGIN
  UPDATE      employees
  SET         salary = salary + v_sal_increase
  WHERE       job_id = 'ST_CLERK';
END;
/
```

ORACLE

Supprimer des données

Supprimer les lignes appartenant au service 10 à partir de la table `EMPLOYEES`

Exemple :

```
DECLARE
  v_deptno   employees.department_id%TYPE := 10;
BEGIN
  DELETE FROM   employees
  WHERE         department_id = v_deptno;
END;
/
```

ORACLE

Fusionner des lignes

Insérer ou mettre à jour des lignes dans la table `COPY_EMP`, pour correspondre à la table `EMPLOYEES`

```
DECLARE
     v_empno employees.employee_id%TYPE := 100;
BEGIN
MERGE INTO copy_emp c
     USING employees e
     ON (e.employee_id = v_empno)
   WHEN MATCHED THEN
     UPDATE SET
       c.first_name     = e.first_name,
       c.last_name      = e.last_name,
       c.email          = e.email,
       . . .
   WHEN NOT MATCHED THEN
     INSERT VALUES(e.employee_id, e.first_name, e.last_name,
          . . .,e.department_id);
END;
```

ORACLE

Conventions d'appellation

- **Utiliser une convention d'appellation pour éviter toute ambiguïté dans la clause `WHERE`**
- **Les colonnes de base de données et les identificateurs doivent porter des noms différents**
- **Des erreurs de syntaxe peuvent survenir car PL/SQL recherche en premier lieu une colonne de table dans la base de données**
- **Les noms des variables locales et les paramètres formels ont priorité sur les noms des tables de la base de données**
- **Les noms de colonne des tables de la base de données ont priorité sur les noms des variables locales**

ORACLE

Curseur SQL

- Un curseur est une zone de travail réservée à SQL
- Il existe deux types de curseur :
  - curseurs implicites
  - curseurs explicites
- Le serveur Oracle utilise des curseurs implicites pour analyser et exécuter les instructions SQL
- Les curseurs explicites sont déclarés de manière explicite par le programmeur

Attributs d'un curseur SQL

Grâce aux attributs d'un curseur SQL, vous pouvez tester le résultat lié à l'exécution d'instructions SQL

SQL%ROWCOUNT	Nombre de lignes affectées par la dernière instruction SQL (valeur entière)
SQL%FOUND	Attribut booléen qui prend la valeur TRUE si la dernière instruction SQL affecte une ou plusieurs lignes
SQL%NOTFOUND	Attribut booléen qui prend la valeur TRUE si la dernière instruction SQL n'affecte aucune ligne
SQL%ISOPEN	Prend toujours la valeur FALSE car PL/SQL ferme les curseurs implicites immédiatement après leur exécution

Attributs de curseur SQL

Supprimer les lignes possédant l'ID d'employé indiqué dans la table EMPLOYEES et afficher le nombre de lignes supprimées

Exemple :

```
VARIABLE rows_deleted VARCHAR2(30)
DECLARE
  v_employee_id employees.employee_id%TYPE := 176;
BEGIN
  DELETE FROM  employees
  WHERE        employee_id = v_employee_id;
  :rows_deleted := (SQL%ROWCOUNT ||
                    ' row deleted.');
END;
/
PRINT rows_deleted
```


Instructions de gestion des transactions

- Initialiser une transaction avec la première instruction LMD suivant COMMIT ou ROLLBACK
- Utiliser les instructions SQL COMMIT et ROLLBACK pour mettre fin explicitement à une transaction

Synthèse

- intégrer du code SQL dans un bloc PL/SQL en utilisant `SELECT`, `INSERT`, `UPDATE`, `DELETE` et `MERGE`
- intégrer des instructions de gestion des transactions dans un bloc PL/SQL en utilisant `COMMIT`, `ROLLBACK` et `SAVEPOINT`

Synthèse

- il existe deux types de curseur : implicites et explicites
- les attributs d'un curseur implicite permettent de vérifier le résultat de l'exécution d'instructions LMD :
  - `SQL%ROWCOUNT`
  - `SQL%FOUND`
  - `SQL%NOTFOUND`
  - `SQL%ISOPEN`
- les curseurs explicites sont définis par le programmeur

Présentation de l'exercice 3

Dans cet exercice, vous allez créer un bloc PL/SQL pour :

- sélectionner des données dans une table
- insérer des données dans une table
- mettre à jour des données dans une table
- supprimer un enregistrement d'une table

4 Ecrire des structures de contrôle

Objectifs

- identifier les types de structure de contrôle et leurs utilisations
- écrire une instruction IF
- utiliser des expressions CASE
- écrire et identifier différents types d'instruction LOOP
- utiliser des tables logiques
- contrôler le flux de blocs à l'aide de boucles imbriquées et d'étiquettes

Contrôler le flux d'exécution PL/SQL

- Vous pouvez modifier l'exécution logique des instructions en utilisant des instructions conditionnelles IF et des structures de contrôle LOOP
- Instructions conditionnelles IF :
  - IF-THEN-END IF
  - IF-THEN-ELSE-END IF
  - IF-THEN-ELSIF-END IF

Instructions IF

Syntaxe :

```
IF condition THEN
  statements;
[ELSIF condition THEN
  statements;]
[ELSE
  statements;]
END IF;
```

Si le nom de l'employé est Gietz, lui affecter l'ID de manager 102.

```
IF UPPER(v_last_name) = 'GIETZ' THEN
  v_mgr := 102;
END IF;
```


Instructions IF simples

Si le nom de famille est Vargas :

- affecter la valeur SA_REP au code de poste
- affecter la valeur 80 au numéro du service

```
. . .
IF v_ename      = 'Vargas' THEN
   v_job        := 'SA_REP';
   v_deptno     := 80;
END IF;
. . .
```

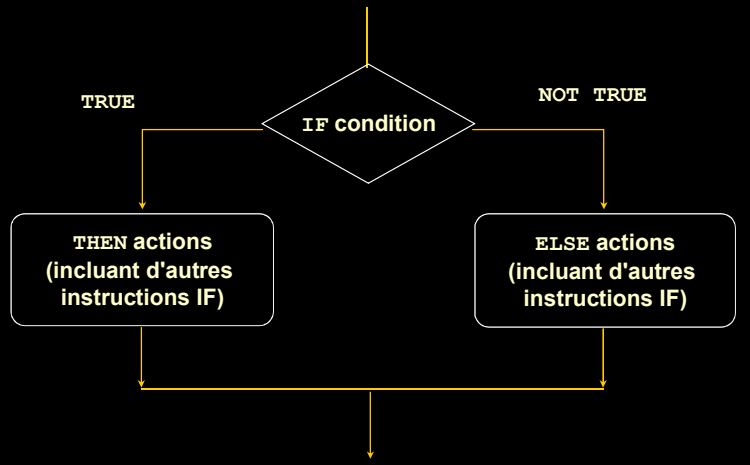
Instructions IF composées

Si le nom de famille est Vargas et que le salaire est supérieur à 6 500 :

affecter la valeur 60 au numéro du service

```
. . .
IF v_ename  = 'Vargas' AND salary > 6500 THEN
   v_deptno := 60;
END IF;
. . .
```


Flux d'exécution des instructions IF-THEN-ELSE

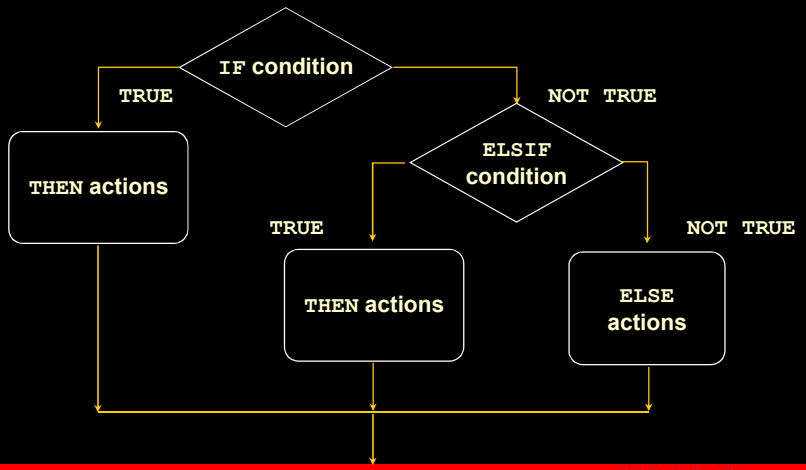


Instructions IF-THEN-ELSE

Affecter la valeur TRUE à un indicateur booléen si la date d'embauche est de plus de cinq ans ; sinon, affecter la valeur FALSE.

```
DECLARE
   v_hire_date  DATE := '12-Dec-1990';
   v_five_years BOOLEAN;
BEGIN
. . .
IF MONTHS_BETWEEN(SYSDATE,v_hire_date)/12 > 5 THEN
   v_five_years := TRUE;
ELSE
   v_five_years := FALSE;
END IF;
...
```


Flux d'exécution des instructions IF-THEN-ELSIF



Instructions IF-THEN-ELSIF

Calculer le pourcentage d'une valeur donnée en fonction d'une condition

Exemple :

```
. . .
IF    v_start > 100 THEN
      v_start := 0.2 * v_start;
ELSIF v_start >= 50 THEN
      v_start := 0.5 * v_start;
ELSE
      v_start := 0.1 * v_start;
END IF;
. . .
```

ORACLE

Expressions CASE

- Une expression CASE sélectionne un résultat et le renvoie
- Pour que le résultat soit sélectionné, l'expression CASE utilise une expression dont la valeur permet d'effectuer un choix parmi plusieurs possibilités

```
CASE selector
   WHEN expression1 THEN result1
   WHEN expression2 THEN result2
   ...
   WHEN expressionN THEN resultN
  [ELSE resultN+1;]
END;
```

ORACLE

Exemple d'expressions CASE

```
SET SERVEROUTPUT ON
DECLARE
   v_grade CHAR(1) := UPPER('&p_grade');
   v_appraisal VARCHAR2(20);
BEGIN
    v_appraisal :=
      CASE v_grade
         WHEN 'A' THEN 'Excellent'
         WHEN 'B' THEN 'Very Good'
         WHEN 'C' THEN 'Good'
         ELSE 'No such grade'
      END;
DBMS_OUTPUT.PUT_LINE ('Grade: '|| v_grade || '
                      Appraisal ' || v_appraisal);
END;
/
```

ORACLE

Traiter les valeurs NULL

Lorsque vous utilisez des valeurs NULL, vous pouvez éviter certaines erreurs fréquentes en gardant à l'esprit les règles suivantes :

- les comparaisons simples impliquant des valeurs NULL renvoient toujours une valeur NULL
- l'application de l'opérateur logique NOT à une valeur NULL renvoie une valeur NULL
- dans les instructions de contrôle conditionnelles, si la condition renvoie une valeur NULL, la séquence d'instructions associée n'est pas exécutée

ORACLE

Tables logiques

Créer une condition booléenne simple avec un opérateur de comparaison

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	
TRUE	FALSE
FALSE	TRUE
NULL	NULL

Conditions booléennes

Quelle est la valeur de `V_FLAG` dans chaque cas ?

```
v_flag := v_reorder_flag AND v_available_flag;
```

V_REORDER_FLAG	V_AVAILABLE_FLAG	V_FLAG
TRUE	TRUE	?
TRUE	FALSE	?
NULL	TRUE	?
NULL	FALSE	?

Contrôle d'itération : instructions LOOP

- Les boucles permettent d'exécuter plusieurs fois une instruction ou une séquence d'instructions
- Il existe trois types de boucle :
  - boucle de base
  - boucle `FOR`
  - boucle `WHILE`

Boucles de base

Syntaxe :

```
LOOP                                    -- delimiter
  statement1;                           -- statements
  . . .
  EXIT [WHEN condition];                -- EXIT statement
END LOOP;                               -- delimiter
```

condition est une variable ou une expression booléenne (TRUE, FALSE, ou NULL);

Boucles de base

Exemple :

```
DECLARE
  v_country_id    locations.country_id%TYPE := 'CA';
  v_location_id   locations.location_id%TYPE;
  v_counter       NUMBER(2) := 1;
  v_city          locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO v_location_id FROM locations
  WHERE country_id = v_country_id;
  LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_location_id + v_counter),v_city, v_country_id);
    v_counter := v_counter + 1;
    EXIT WHEN v_counter > 3;
  END LOOP;
END;
/
```


Boucles WHILE

Syntaxe :

```
WHILE condition
LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

Condition is evaluated at the beginning of each iteration.

Utiliser la boucle WHILE pour répéter des instructions tant qu'une condition renvoie TRUE.

Boucles WHILE

Exemple :

```
DECLARE
  v_country_id      locations.country_id%TYPE := 'CA';
  v_location_id     locations.location_id%TYPE;
  v_city            locations.city%TYPE := 'Montreal';
  v_counter         NUMBER  := 1;
BEGIN
  SELECT MAX(location_id) INTO v_location_id FROM locations
  WHERE country_id = v_country_id;
  WHILE v_counter <= 3
  LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_location_id + v_counter), v_city, v_country_id);
    v_counter := v_counter + 1;
  END LOOP;
END;
/
```


Boucles FOR

Syntaxe :

```
FOR counter IN [REVERSE] lower_bound..upper_boun
LOOP
  statement1;
  statement2;
  . . .
END LOOP;
```

- **Utiliser une boucle FOR pour simplifier le contrôle du nombre d'itérations**
- **Ne pas déclarer le compteur (sa déclaration est implicite)**
- **La syntaxe requise est 'lower_bound .. upper_bound'**

Boucles FOR

Insérer trois nouveaux ID d'emplacement pour le code de pays CA et la ville de Montréal.

```
DECLARE
  v_country_id    locations.country_id%TYPE := 'CA';
  v_location_id   locations.location_id%TYPE;
  v_city          locations.city%TYPE := 'Montreal';
BEGIN
  SELECT MAX(location_id) INTO v_location_id
    FROM locations
    WHERE country_id = v_country_id;
  FOR i IN 1..3
  LOOP
    INSERT INTO locations(location_id, city, country_id)
    VALUES((v_location_id + i), v_city, v_country_id );
  END LOOP;
END;
/
```

ORACLE

Boucles FOR

Remarques

- Ne référencer le compteur qu'à l'intérieur de la boucle, il n'est pas défini en dehors
- *Ne pas référencer* le compteur en tant que cible d'une affectation

ORACLE

Remarques relatives à l'utilisation des boucles

- Utiliser la boucle de base lorsque ses instructions doivent s'exécuter au moins une fois
- Utiliser la boucle WHILE si la condition doit être évaluée au début de chaque itération
- Utiliser une boucle FOR si le nombre d'itérations est connu

ORACLE

Boucles imbriquées et étiquettes

- Imbriquer des boucles à plusieurs niveaux
- Utiliser des étiquettes pour différencier les blocs des boucles
- Quitter la boucle externe en utilisant l'instruction EXIT qui référence l'étiquette

ORACLE

Boucles imbriquées et étiquettes

```
...
BEGIN
  <<Outer_loop>>
  LOOP
    v_counter := v_counter+1;
  EXIT WHEN v_counter>10;
    <<Inner_loop>>
    LOOP
      ...
      EXIT Outer_loop WHEN total_done = 'YES';
      -- Leave both loops
      EXIT WHEN inner_done = 'YES';
      -- Leave inner loop only
      ...
    END LOOP Inner_loop;
    ...
  END LOOP Outer_loop;
END;
```


Synthèse

Modifier l'enchaînement logique des instructions en utilisant des structures de contrôle

- Instructions conditionnelles (`IF`)
- Expressions `CASE`
- Boucles :
  - boucle de base
  - boucle `FOR`
  - boucle `WHILE`
- Instructions `EXIT`

Présentation de l'exercice 4

Dans cet exercice, vous allez :

- exécuter des actions conditionnelles en utilisant l'instruction `IF`
- écrire des schémas itératifs en utilisant la structure de la boucle