ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR

Membre de HONORIS UNITED UNIVERSITIES

École Marocaine des Sciences de l'Ingénieur

Filière: IIR

Les Bases de Données

Prof. Zakaria KHATAR

Partie 2 : PL/SQL

Les boucles et les conditions

1) Les boucles et les conditions en PL/SQL :

Les boucles et les conditions sont des éléments de base de tous les langages de programmation et PL/SQL n'est pas différent.

Les boucles permettent de répéter un certain nombre d'instructions plusieurs fois et Les conditions vous permettent de tester si une expression est vraie ou fausse et d'exécuter les instructions en question.

PL/SQL offre plusieurs types de boucles, dont les plus couramment utilisées sont :

- **La condition IF**
- **La condition CASE**
- **La boucle LOOP**
- **La boucle FOR**
- **La boucle WHILE**

2) Les conditions en PL/SQL :

Les conditions vous permettent de tester si une expression est vraie ou fausse et d'exécuter les instructions en question.

PL/SQL propose plusieurs instructions de contrôle de flux qui peuvent être utilisées avec les conditions, notamment :

- **La condition IF**
- **La condition CASE**

2-1) La condition IF :

Elle exécute une instruction ou un ensemble d'instructions si la condition est vraie, et éventuellement exécute une autre instruction ou un autre ensemble d'instructions si la condition est fausse.

Syntaxe :

```
DECLARE
        Déclaration des variables
BEGIN
        IF condition1 THEN traitement1;
        ELSIF condition2 THEN traitement2;
        …
        ELSE traitementN;
        END IF;
END;
```

2-1) La condition IF :

Exemple :

```
DECLARE
    x number := 5;
BEGIN
    IF x > 10 THEN
        dbms_output.put_line('x est plus grand que 10');
    ELSIF x = 10 THEN
        dbms_output.put_line('x est égal à 10');
    ELSE
        dbms_output.put_line('x est plus petit que 10');
    END IF;
END;
/
```

2-1) La condition IF :

Vous pouvez utiliser autant de clauses **ELSEIF** que vous le souhaitez dans une instruction **IF**. Cependant, vous ne pouvez utiliser qu'une seule clause **ELSE** qui sera exécutée si aucune des conditions précédentes n'est vraie.

2-2) La condition CASE :

Elle exécute une instruction ou un ensemble d'instructions en fonction de la valeur d'une expression.

Syntaxe :

```
DECLARE
        Déclaration des variables
BEGIN
        CASE expression
            WHEN condition1 THEN traitement1;
            WHEN condition2 THEN traitement2;
            …
            ELSE traitementN;
        END CASE;
END;
```

2-2) La condition CASE :

Dans cette syntaxe:

Expression : est une expression qui sera évaluée et dont la valeur sera comparée à chaque condition **WHEN**.

condition1, condition2, etc. : sont des expressions qui seront évaluées et qui doivent être vraies ou fausses.

traitement1, traitement2, etc. : sont des instructions qui seront exécutées si la condition correspondante est vraie.

La clause ELSE est **facultative** et permet d'exécuter une instruction ou un ensemble d'instructions si aucune des conditions **WHEN** n'est vraie.

2-2) La condition CASE :

Exemple :

```
DECLARE
    command varchar2(20) := 'PRINT';
BEGIN
    CASE command
      WHEN 'PRINT' THEN
          dbms_output.put_line('Impression en cours...');
      WHEN 'SAVE' THEN
          dbms_output.put_line('Sauvegarde en cours...');
      WHEN 'DELETE' THEN
          dbms_output.put_line('Suppression en cours...');
    END CASE;
END;
```

2-2) La condition CASE :

Dans cet exemple, l'instruction **dbms_output.put_line('Impression en cours...')** sera exécutée car la variable command vaut **'PRINT'**.

En résumé, l'instruction **CASE** peut être utilisée de différentes manières pour prendre des décisions et exécuter des actions en fonction de la valeur d'une expression ou d'une variable. Elle peut être particulièrement utile lorsque vous avez besoin de tester plusieurs conditions de manière successive.

2-3) CASE dans SELECT :

```
SELECT
    employee_id, salary,
        CASE
            WHEN salary <= 3000 THEN 'Faible'
            WHEN salary > 3000 AND salary <= 7000 THEN 'Moyen'
            WHEN salary > 7000 THEN 'Élevé'
        END AS categorie_salaire
FROM
    employees;
```

3) Les Boucles en PL/SQL :

Les boucles permettent de répéter un certain nombre d'instructions plusieurs fois. PL/SQL offre trois types de boucles :

- **La boucle LOOP (de base)**
- **La boucle FOR**
- **La boucle WHILE**

3-1) La boucle LOOP :

La boucle **LOOP** en PL/SQL est utilisée pour exécuter un ensemble d'instructions indéfiniment jusqu'à ce qu'une instruction **EXIT** soit **trouvée**. Voici comment déclarer et utiliser une boucle LOOP :

Syntaxe :

```
DECLARE
        Déclaration des variables
BEGIN
    LOOP
        instruction1;
        instruction2;
        ...
        EXIT WHEN condition;
    END LOOP;
END;
```

3-1) La boucle LOOP :

Exemple :

```
DECLARE
    X number := 0;
BEGIN
    LOOP
        X := X + 1;
        dbms_output.put_line(X);
        EXIT WHEN X = 10;
    END LOOP;
END;
```

Dans cet exemple, la boucle **LOOP** incrémente la variable **X** et affiche sa valeur à chaque itération jusqu'à ce que la variable **X** atteigne la valeur 10, moment où la boucle est terminée grâce à l'instruction **EXIT**.

3-1) La boucle FOR :

La boucle **FOR** permet de répéter un ensemble d'instructions d'une manière définie en incrémentant ou décrémentant une variable de boucle dans une intervalle de valeurs spécifiées. Il est possible de définir un pas pour contrôler l'intervalle entre chaque itération de la boucle.

Syntaxe :

```
DECLARE  Déclaration des variables
BEGIN
    FOR variable IN borne_inférieure..borne_supérieure BY [Nombre de pas]
    LOOP
        instruction1;
        instruction2;
        …
    END LOOP;
END;
```

3-1) La boucle FOR :

Exemple 1 :

```
DECLARE
        X number;
BEGIN
        FOR X IN 1..10
        LOOP
          DBMS_OUTPUT.PUT_LINE(X);
        END LOOP;
END;
```

Dans cet exemple, la boucle **FOR** est définie avec la variable de boucle **X** et un intervalle de valeurs de **1 à 10**. Le bloc de code associé à la boucle affiche chaque valeur de **X** à chaque itération. La boucle s'exécutera donc **10** fois, affichant les valeurs de **1 à 10**.

3-1) La boucle FOR :

Exemple 2 :

```
DECLARE
        X number;
BEGIN
        FOR X IN 1..10 BY 2
        LOOP
          DBMS_OUTPUT.PUT_LINE(X);
        END LOOP;
END;
```

L'exemple 2 ci-dessus joue le même rôle que celui de **l'exemple 1**, sauf que dans cet exemple, **on a ajouté un nombre de pas qui est de 2**. donc il va nous afficher les valeurs suivantes : 1, 3, 5, 7, 9

3-1) La boucle FOR :

Exemple 1 :

```
DECLARE
        X number;
BEGIN
        FOR X IN REVERSE 1..10
        LOOP
          DBMS_OUTPUT.PUT_LINE(X);
        END LOOP;
END;
```

Dans cet exemple 3, on a utilisé la clause **REVERSE** pour inverser les valeurs et commencer à afficher les valeurs de 10 à 1.

3-2) La boucle WHILE :

La boucle **WHILE** permet d'exécuter un bloc de code de manière répétée tant qu'une condition spécifiée est vraie. La boucle WHILE est définie en spécifiant une condition qui est vérifiée avant chaque itération du bloc. Si la condition est vraie, le bloc est exécuté, sinon la boucle s'arrête.

Syntaxe :

```
DECLARE  Déclaration des variables
BEGIN
    WHILE condition
    LOOP
        instruction1;
        instruction2;
        …
    END LOOP;
END;
```

3-2) La boucle WHILE :

Exemple :

```
DECLARE
    X number := 1;
BEGIN
    WHILE X <= 10
    LOOP
        DBMS_OUTPUT.PUT_LINE(X);
        X := X + 1;
    END LOOP;
END;
```

L'exemple ci-dessus définit une boucle WHILE qui affiche les nombres de 1 à 10.

Exemple de FOR avec SELECT :

```
BEGIN
   FOR emp IN (SELECT first_name, salary FROM employees)
   LOOP

      IF emp.salary > 10000 THEN
         DBMS_OUTPUT.PUT_LINE(emp.first_name || ' - High Earner');

      ELSE
         DBMS_OUTPUT.PUT_LINE(emp.first_name);
      END IF;

   END LOOP;
END;
```