CONCEPTS DE BASE DU LANGAGE PL/SQL

Enis BELHASSEN

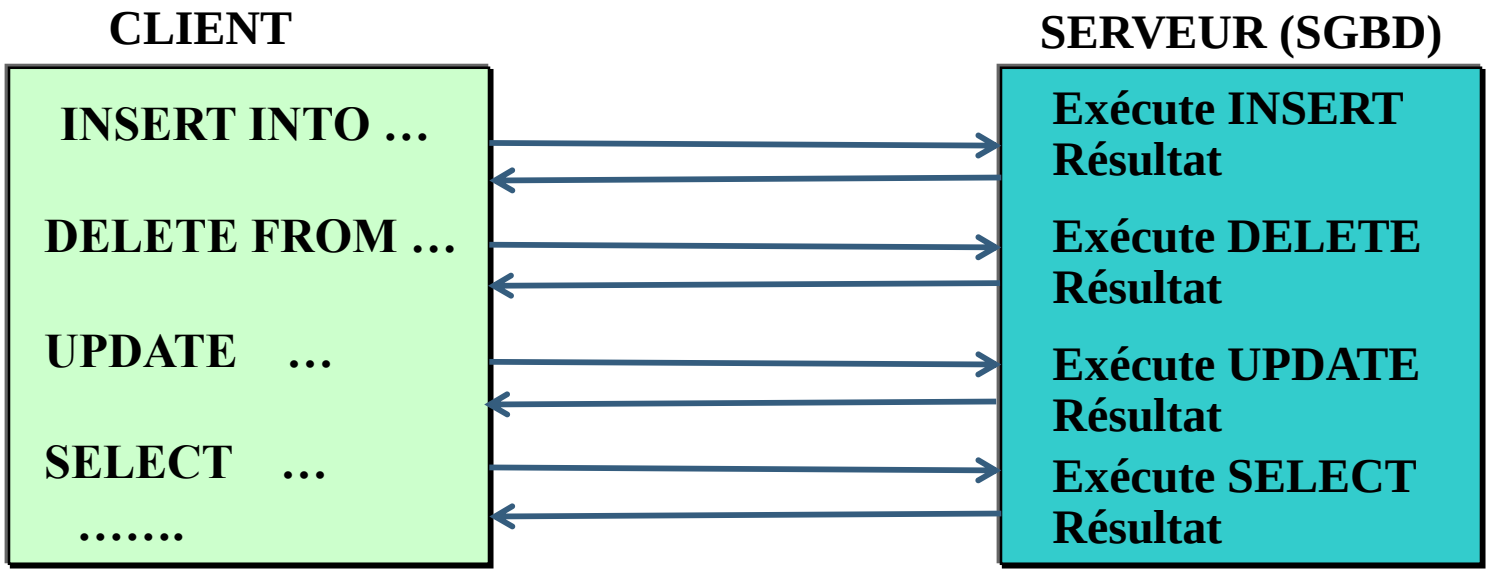
2019

Langage PL/SQL # SQL

- SQL : langage ensembliste non procédural
  - Ensemble de requêtes distinctes
  - Langage assertionnel de 4ème génération : on décrit le résultat sans dire comment il faut accéder aux données
  - Possibilité d'encapsulation dans un langage hôte de 3ème génération
- PL/SQL
  - 'Procédural Language' : surcouche procédurale à SQL (traitements itératifs, contrôles, affectations, exceptions, ….)
  - Un langage procédural pour lier plusieurs requêtes SQL avec des variables et dans les structures de programmation habituelles
  - L'intérêt du PL/SQL est de pouvoir mélanger la puissance des instructions SQL avec la souplesse d'un langage procédural

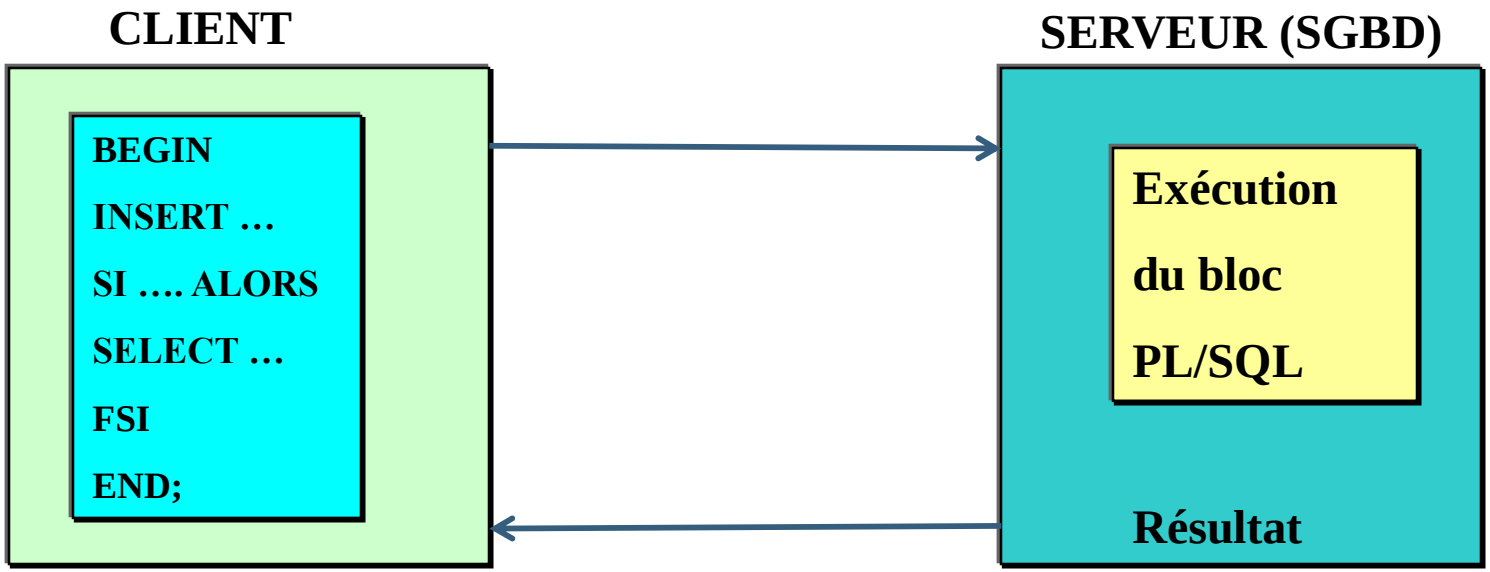
Requêtes SQL

- Chaque requête 'client' est transmise au serveur de données pour être exécutée avec retour de résultats

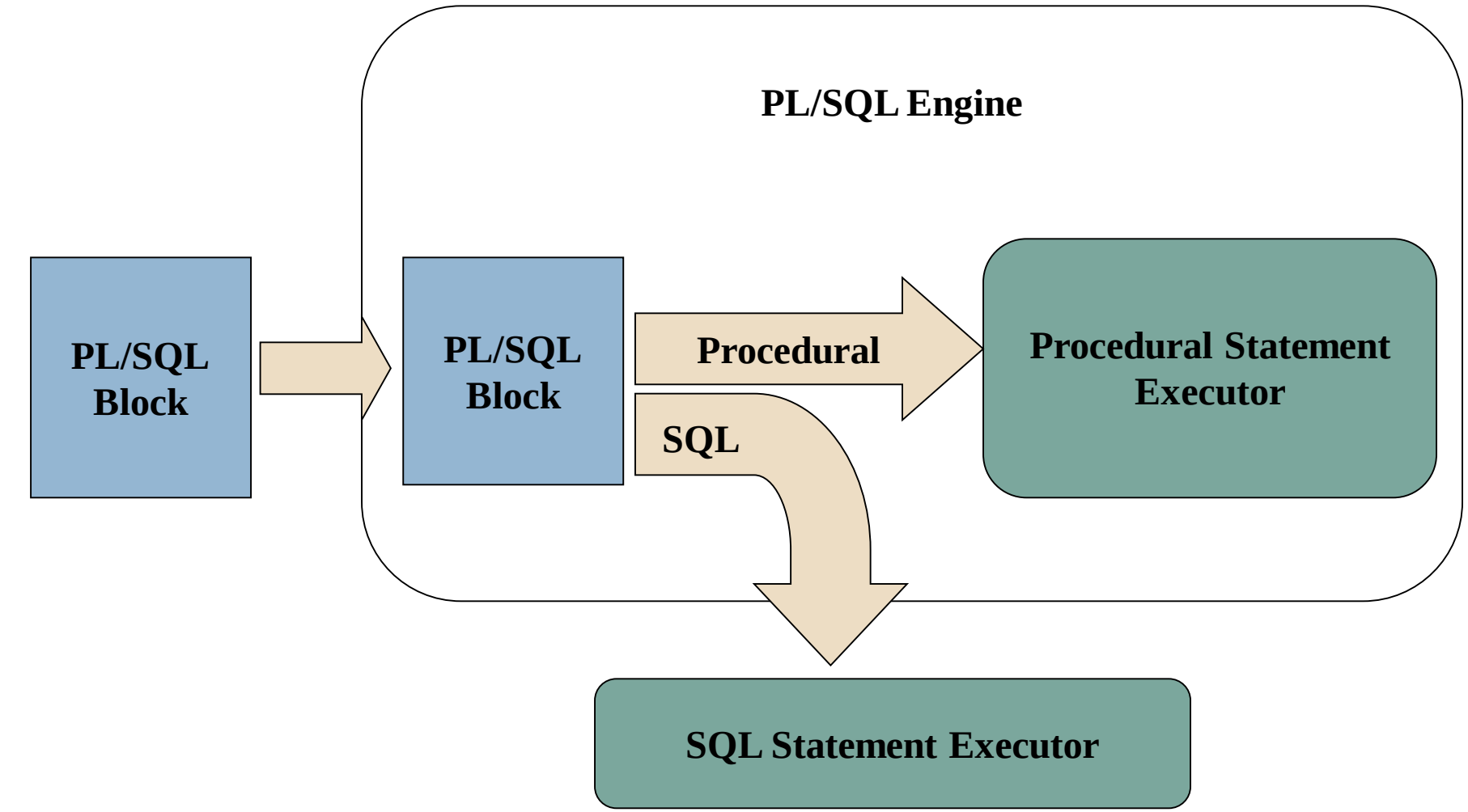


Bloc PL/SQL

- Le bloc de requêtes est envoyé sur le serveur. Celui-ci exécute le bloc et renvoie 1 résultat final.



Architecture



Avantages du PL/SQL

- Traitements procéduraux : la gestion des variables et les structures de contrôle (conditionnelles et itératives)
- Fonctionnalités supplémentaires : la gestion des curseurs et le traitement des erreurs (exceptions)
- Amélioration des performances : plusieurs instructions sont regroupées dans une unité (bloc) qui ne génèrera qu'un "accès" à la base (à la place d'un accès par instruction)

Avantages du PL/SQL

- Modularité : un bloc peut être nommé pour devenir une procédure ou une fonction cataloguée et réutilisable. Une procédure, ou fonction, cataloguée peut être incluse dans un paquetage (package)
- Portabilité : un programme PL/SQL est indépendant du système d'exploitation qui héberge le serveur Oracle. En changeant de système, les applicatifs n'ont pas à être modifiés

Avantages du PL/SQL

- Il s'agit d'une extension du langage SQL avec des caractéristiques propres aux langages de programmation :
  - Déclaration de variables et de constants
  - Types de données abstraits (collections, enregistrements, objets)
  - Traitements conditionnels, répétitifs
  - Traitement des curseurs
  - Gestion des erreurs à l'exécution (exceptions)
  - Modularité (sous-programmes, packages)

Structure d'un bloc PL/SQL

- Section DECLARE (section optionnelle) pour la déclaration des :
  - Variables locales simples
  - Variables tableaux
  - Curseurs / Exceptions
- Section BEGIN (section obligatoire)
  - Section des ordres exécutables
  - Ordres SQL
  - Ordres PL
- Section EXCEPTION (section optionnelle)
  - Traitement des erreurs interceptées
  - Exceptions SQL ou utilisateur

```
[DECLARE
        -- Déclarations]
BEGIN
        -- Instructions
[EXCEPTION
        -- Erreurs]
END;
/
```

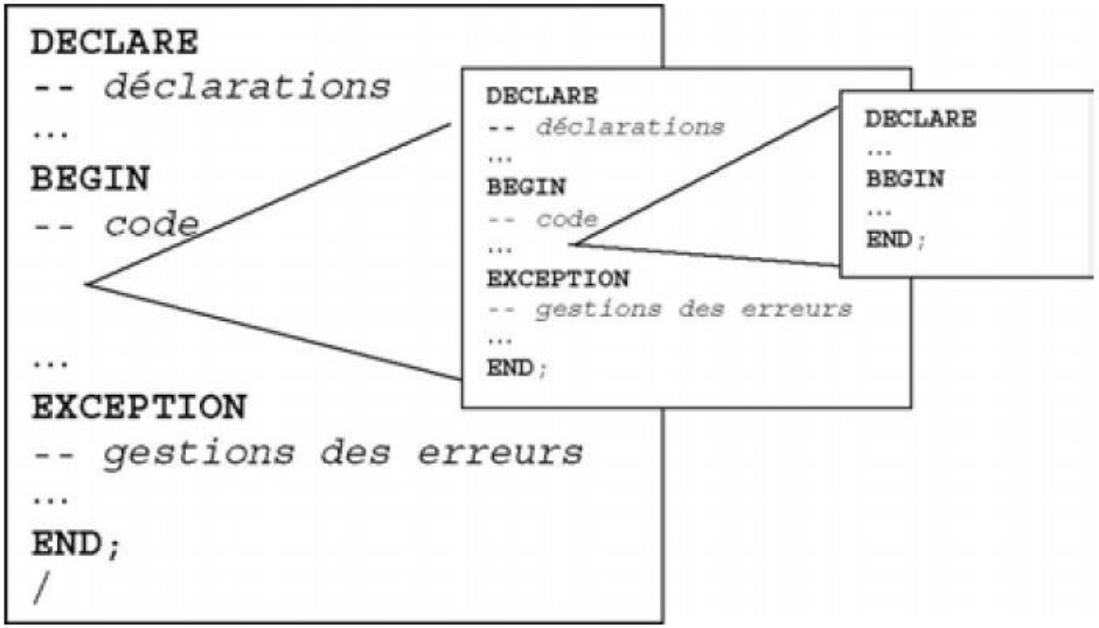
- Il est possible d'ajouter des commentaires à un bloc :

--	commentaire sur une seule ligne (mono-lignes)
/*... ... */	commentaire sur plusieurs lignes (multi-lignes)

Structure d'un bloc PL/SQL

- Un bloc peut être imbriqué dans le code d'un autre bloc : sous-bloc.
- Un sous-bloc peut aussi se trouver dans la partie des exceptions.
- Un sous-bloc commence par BEGIN et se termine par END.

PL/SQL n'est pas sensible à la casse (not case sensitive)



Types de blocs PL/SQL

- Bloc Anonyme
  - Stocké en-dehors de la base de données
  - Compilé et exécuté à la volée
- Procédure Stockée :
  - Compilée séparément
  - Stockée de façon permanente dans la BD
- Déclencheur (Trigger)
  - Procédure stockée associée à une table
  - Exécution automatique sur événement

Variables – SQL*Plus

- Variables lues par un ACCEPT .... PROMPT

SQL*Plus

PL/SQL

SQL*Plus

```
ACCEPT var PROMPT 'Entrer la valeur : '

DECLARE
        -- déclarations
BEGIN
        -- &var si numérique
        -- '&var' si caractère
END;
/
-- Ordre SQL .....
```

- Il existe deux types de variables :
  - *variable temporaire* : **&variable**
  - *variable permanente* : **&&variable**

Variables – SQL*Plus

- DEF : renvoie les valeurs de toutes les variables définies
- DEF variable = valeur : affecte une valeur à la variable d'une façon permanente
- UNDEF variable : supprime la définition de la variable

Oracle SQL*Plus

Fichier Edition Recherche Options Aide

```
SQL*Plus: Release 10.1.0.5.0 - Production on Mar. Oct. 8 08:15:51 2019

Copyright (c) 1982, 2005, Oracle.  All rights reserved.


Connecté à :
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, Real Application Clusters, OLAP, Advanced Analytics
and Real Application Testing options

SQL> def
DEFINE _DATE            = "" (CHAR)
DEFINE _CONNECT_IDENTIFIER = "dbdev11" (CHAR)
DEFINE _USER            = "USERFEC" (CHAR)
DEFINE _PRIVILEGE       = "" (CHAR)
DEFINE _SQLPLUS_RELEASE = "1001000500" (CHAR)
DEFINE _EDITOR          = "Notepad" (CHAR)
DEFINE _O_VERSION       = "Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Producti
With the Partitioning, Real Application Clusters, OLAP, Advanced Analytics
and Real Application Testing options" (CHAR)
DEFINE _O_RELEASE       = "1201000200" (CHAR)
SQL>
```

Commandes de dialogue – SQL*Plus

- PROMPT texte : afficher le texte
- ACCEPT variable [PROMPT texte] : forcer SQL*Plus à attendre une valeur pour définir une variable permanente

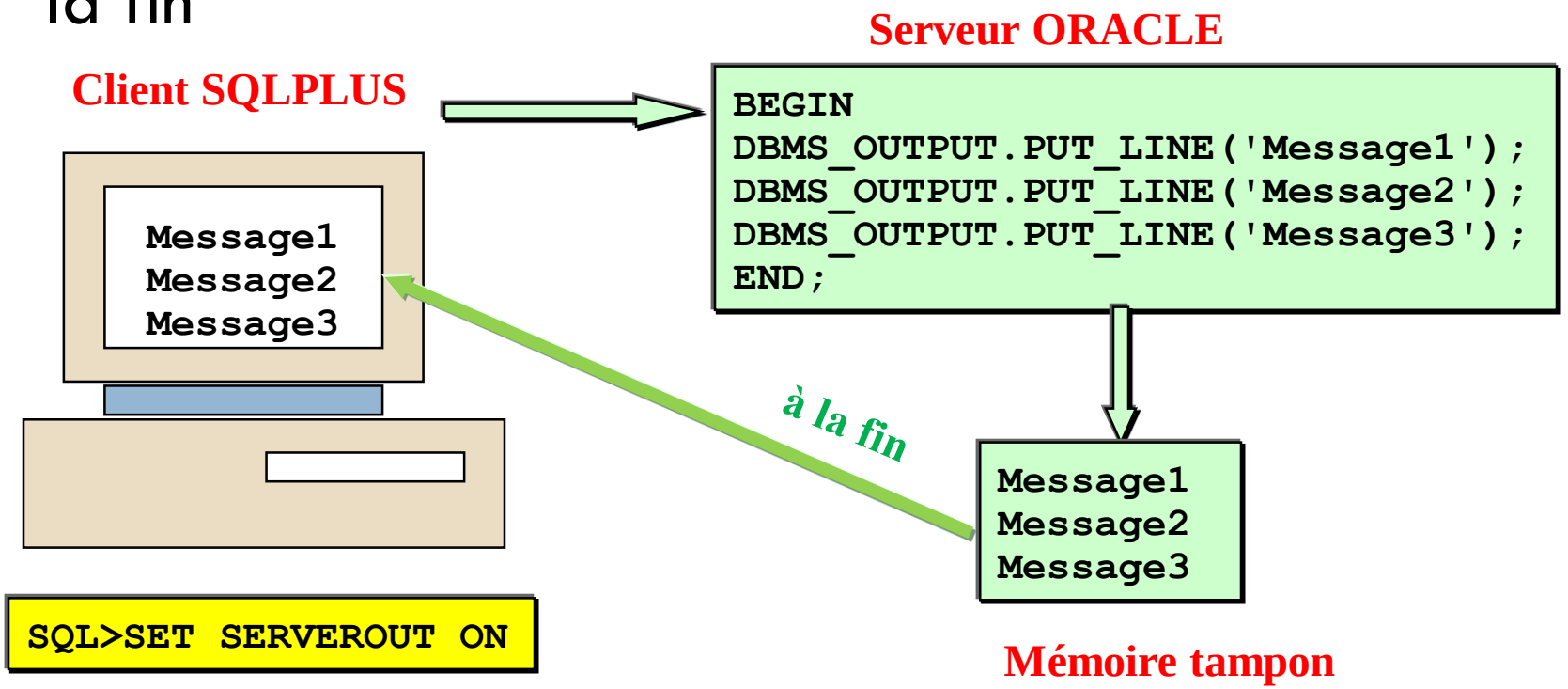
Oracle SQL*Plus

Fichier Edition Recherche Options Aide

```
SQL*Plus: Release 10.1.0.5.0 - Production on Mar. Oct. 8 08:26:35 2019

Copyright (c) 1982, 2005, Oracle.  All rights reserved.


Connecté à :
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, Real Application Clusters, OLAP, Advanced Analytics
and Real Application Testing options

SQL> ACCEPT nom_emp PROMPT 'Entrer le nom de l''employé : '
Entrer le nom de l'employé : Ali
SQL> DEF
DEFINE _DATE            = "" (CHAR)
DEFINE _CONNECT_IDENTIFIER = "dbdev11" (CHAR)
DEFINE _USER            = "USERFEC" (CHAR)
DEFINE _PRIVILEGE       = "" (CHAR)
DEFINE _SQLPLUS_RELEASE = "1001000500" (CHAR)
DEFINE _EDITOR          = "Notepad" (CHAR)
DEFINE _O_VERSION       = "Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Producti
With the Partitioning, Real Application Clusters, OLAP, Advanced Analytics
and Real Application Testing options" (CHAR)
DEFINE _O_RELEASE       = "1201000200" (CHAR)
DEFINE NOM_EMP          = "Ali" (CHAR)
SQL>
```

Package DBMS_OUTPUT – SQL*Plus

- Messages enregistrés dans une mémoire tampon côté serveur
- La mémoire tampon est affichée sur le poste client à la fin



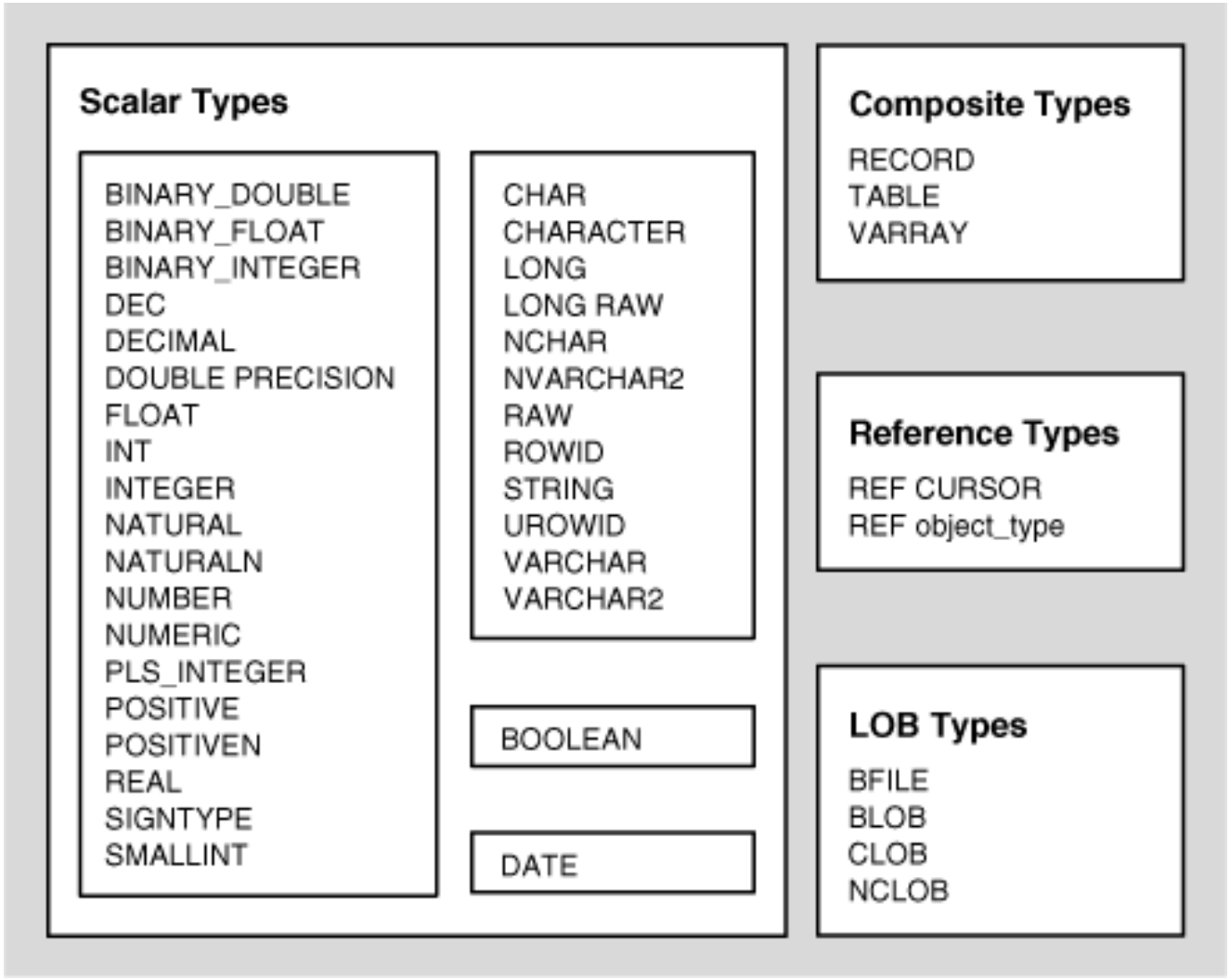
Package DBMS_OUTPUT – SQL*Plus

- Écriture dans le buffer avec saut de ligne
  - DBMS_OUTPUT.PUT_LINE(<chaîne caractères>);
- Écriture dans le buffer sans saut de ligne
  - DBMS_OUTPUT.PUT(<chaîne caractères>);
- Écriture dans le buffer d'un saut de ligne
  - DBMS_OUTPUT.NEW_LINE;

Types de Variables PL/SQL

- Variables locales:
  - De type simple: type de base ou booléen
  - Faisant référence à la métabase (dictionnaire de données de Oracle)
  - De type composé : Tableau, Record
- Variables Extérieures:
  - Variables d'un langage hôte (ex: C) (préfixés par :)
  - Paramètres (ex: SQL interactif préfixés par &)

Types de Variables



Opérateurs SQL

- Opérateur d'affectation **:=**
- Opérateurs arithmétiques **+ - / * ****
- Opérateur de concaténation **| |**
- Opérateurs de comparaison

  = < > <= >= <> != *IS NULL* *LIKE* *BETWEEN* *IN*
- Opérateurs logiques ***AND, OR, NOT***

Variables simples

- Déclaration :

```
variable_name [CONSTANT] datatype
  [NOT NULL] [:= | DEFAULT initial_value];
```

- Le nom d'une variable (identificateur) commence par une lettre suivie (optionnel) de symboles (lettres, chiffres, $, _ , #).
- Un identificateur peut contenir jusqu'à 30 caractères
- Les autres symboles sont interdits (& , - / espace ...)

Variables simples

- Variables de type SQL

```
Date_nais DATE;
nom       VARCHAR(30) DEFAULT 'Bonjour';
minimum   CONSTANT INTEGER := 5;
salaire   NUMBER(8,2);
debut     NUMBER NOT NULL := 10;
```

- Variables de type booléen (TRUE, FALSE, NULL)

```
fin      BOOLEAN;
reponse  BOOLEAN DEFAULT TRUE;
ok       BOOLEAN := TRUE;
```

Variables référençant la métabase

- Référence à une colonne (table, vue)

```
vsalaire        employe.salaire%TYPE;
vnom            etudiant.nom%TYPE;
```

- Référence à une ligne (table, vue)

```
vemploye        employe%ROWTYPE;
vetudiant       etudiant%ROWTYPE;
```

  - Variable de type ‘struct’
  - Contenu d’une variable : variable.colonne

```
vemploye.adresse
```

- Référence une variable précédemment définie

```
commi           number(7,2);
Salaire         commi%TYPE;
```

Tableaux dynamiques

- Déclaration d'un type tableau

```
TYPE  <nom du type du tableau>
IS TABLE OF <type de l'élément>
INDEX BY <type index>;
```

- Affectation (héritage) de ce type à une variable

```
<nom élément>  <nom du type du tableau>;
```

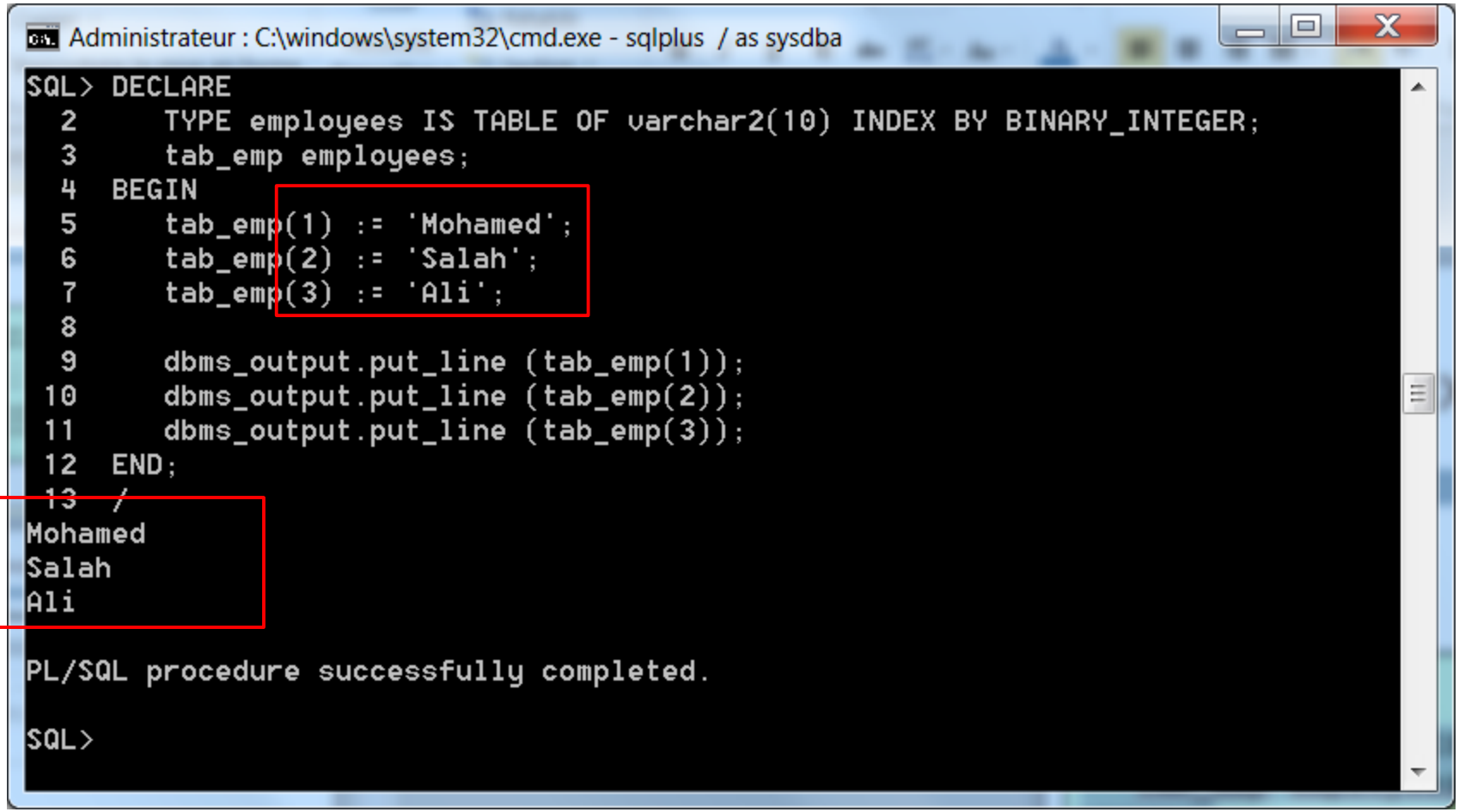
- Utilisation dans la section BEGIN : un élément du tableau :

```
<nom élément> (rang dans le tableau)
```

Tableaux dynamiques

```
DECLARE
   TYPE employees IS TABLE OF varchar2(10) INDEX BY BINARY_INTEGER;
   tab_emp employees;
BEGIN
   tab_emp(1) := 'Mohamed';
   tab_emp(2) := 'Salah';
   tab_emp(3) := 'Ali';

   dbms_output.put_line (tab_emp(1));
   dbms_output.put_line (tab_emp(2));
   dbms_output.put_line (tab_emp(3));
END;
/
```

Tableaux dynamiques



Tableaux dynamiques

- Fonctions pour les tableaux

Fonction	Description
`EXISTS(x)`	Retourne `TRUE` si le x^e élément du tableau existe.
`COUNT`	Retourne le nombre d'éléments du tableau.
`FIRST / LAST`	Retourne le premier/dernier indice du tableau (`NULL` si tableau vide).
`PRIOR(x) / NEXT(x)`	Retourne l'élément avant/après le x^e élément du tableau.

`DELETE`
`DELETE(x)`
`DELETE(x,y)`
	Supprime un ou plusieurs éléments au tableau.

Tableaux dynamiques

```
DECLARE
   TYPE salary IS TABLE OF NUMBER(4) INDEX BY VARCHAR2(10);
   salary_list salary;
   name   VARCHAR2(10);
BEGIN
   salary_list('Mohamed') := 2500;
   salary_list('Salah') := 2000;
   salary_list('Ali') := 3000;

   name := salary_list.FIRST;
   dbms_output.put_line ('Salaire '||name||'='||salary_list(name));
   name := salary_list.NEXT(name);
   dbms_output.put_line ('Salaire '||name||'='||salary_list(name));
   name := salary_list.LAST;
   dbms_output.put_line ('Salaire '||name||'='||salary_list(name));
END;
/
```

Tableaux dynamiques

Administrateur : C:\windows\system32\cmd.exe - sqlplus / as sysdba

```
SQL> DECLARE
  2     TYPE salary IS TABLE OF NUMBER(4) INDEX BY VARCHAR2(10);
  3     salary_list salary;
  4     name   VARCHAR2(10);
  5  BEGIN
  6     salary_list('Mohamed') := 2500;
  7     salary_list('Salah') := 2000;
  8     salary_list('Ali') := 3000;
  9
 10     name := salary_list.FIRST;
 11     dbms_output.put_line ('Salaire '||name||'='||salary_list(name));
 12     name := salary_list.NEXT(name);
 13     dbms_output.put_line ('Salaire '||name||'='||salary_list(name));
 14     name := salary_list.LAST;
 15     dbms_output.put_line ('Salaire '||name||'='||salary_list(name));
 16  END;
 17  /
Salaire Ali=3000
Salaire Mohamed=2500
Salaire Salah=2000

PL/SQL procedure successfully completed.

SQL> _
```

Tableaux dynamiques - Héritage

```
DECLARE
   TYPE noms_emp IS TABLE OF emp.ename%type INDEX BY BINARY_INTEGER;
   list_emp noms_emp;
BEGIN
   list_emp(1) := 'Mohamed';
   list_emp(2) := 'Salah';
   list_emp(3) := 'Ali';

   dbms_output.put_line (list_emp(1));
   dbms_output.put_line (list_emp(2));
   dbms_output.put_line (list_emp(3));
END;
/
```

Tableaux dynamiques - ROW

- Type ROW : chaque élément est une variable ‘struct’

```
DECLARE
  TYPE  type_dept IS TABLE OF dept%ROWTYPE INDEX BY BINARY_INTEGER;
  tab_depts type_dept;
BEGIN
  tab_depts(1).deptno := 100;
  tab_depts(1).dname := 'informatique';
  tab_depts(1).loc := 'tunis';

  tab_depts(2).deptno := 200;
  tab_depts(2).dname := 'marketing';
  tab_depts(2).loc := 'sfax';
END;
/
```

Tableaux dynamiques - RECORD

- Type RECORD : plusieurs variables dans un élément

```
TYPE <type_name> IS RECORD
   (<définition champ 1>, …, <définition champ n>);
```

```
DECLARE
  TYPE record_dept IS RECORD (nomDept VARCHAR(30), adrDept VARCHAR(80));
  TYPE  type_dept IS TABLE OF record_dept INDEX BY BINARY_INTEGER;
  tab_depts type_dept;
BEGIN
tab_depts(1).nomDept := 'informatique';
  tab_depts(1).adrDept := 'tunis';

  tab_depts(2).nomDept := 'marketing';
  tab_depts(2).adrDept := 'sfax';
END;
/
```

Conversion de type de données

- Fonctions de conversion :

TO_CHAR, TO_DATE, TO_NUMBER

```
v_date1 := to_date('01/01/1999', 'dd/mm/yyyy');
v_date2 := to_date('01-jan-1999', 'dd-mon-yyyy');
v_date_systeme := to_char(sysdate, 'dd/mm/yyyy');
v_heure_systeme := to_char(sysdate, 'hh24:mi:ss');
v_date_systeme_complete :=
      to_char(sysdate, 'dd/mm/yyyy hh24:mi:ss');
v_num_str := to_char(130);
v_num := to_number('140');
```

Traitements conditionnels

- Structure alternative ou conditionnelle
  - IF …. THEN ….. ELSIF …… ELSE ……END IF;

```
IF condition THEN
      instructions;
[ELSIF condition THEN
      instructions;]
[… ELSIF condition THEN
      instructions;]
[ELSE
      instructions;]
END IF;
```

  - Une instruction IF peut contenir plusieurs clauses ELSIF, mais une seule clause ELSE.

Structure alternative : CASE

- Choix selon la valeur d'une variable/une expression

```
CASE variable
      WHEN valeur1 THEN action1;
      WHEN valeur2 THEN action2;
      ……
      ELSE  action;
END CASE;
```

```
CASE
      WHEN expression1 THEN   action1;
      WHEN expression2 THEN   action2;
      ……
      ELSE  action;
END CASE;
```

Structure alternative : CASE

```
DECLARE
   note varchar2(1) := 'D';
BEGIN
   CASE note
      when 'A' then dbms_output.put_line('Excellent');
      when 'B' then dbms_output.put_line('Très Bien');
      when 'C' then dbms_output.put_line('Bien');
      when 'D' then dbms_output.put_line('Passable');
      when 'F' then dbms_output.put_line('Mauvais');
      else dbms_output.put_line('Valeur incorrecte');
   END CASE;
END;
/
```

Structure alternative : CASE

Administrateur : C:\windows\system32\cmd.exe - sqlplus / as sysdba

```
SQL> DECLARE
  2     note varchar2(1) := 'D';
  3  BEGIN
  4     CASE note
  5        when 'A' then dbms_output.put_line('Excellent');
  6        when 'B' then dbms_output.put_line('Très Bien');
  7        when 'C' then dbms_output.put_line('Bien');
  8        when 'D' then dbms_output.put_line('Passable');
  9        when 'F' then dbms_output.put_line('Mauvais');
 10        else dbms_output.put_line('Valeur incorrecte');
 11     END CASE;
 12  END;
 13  /
Passable

PL/SQL procedure successfully completed.

SQL>
```

Traitements itératifs

- LOOP

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

- FOR

```
FOR (indice IN [REVERSE] borne1..borne2) LOOP
      instructions;
END LOOP;
```

- WHILE

```
WHILE (condition) LOOP
      instructions;
END LOOP;
```

Structures itératives

```
------ Affichage des nombres divisibles par 4 compris entre 1 et 20
------ Boucle FOR
DECLARE
  max_nombre CONSTANT number(3) := 20;
BEGIN
  dbms_output.put_line ('Les nombres divisibles par 4 sont : ');
  FOR i IN 1..max_nombre LOOP
    IF (mod(i,4) = 0) THEN -- mod(i,4) : reste de la division de i par 4
       dbms_output.put_line (to_char(i) || ' - ');
    END IF;
  END LOOP;
END;
/
```

Structures itératives

Run SQL Command Line

```
SQL> ------ Affichage des nombres divisibles par 4 compris entre 1 et 20
SQL> DECLARE
  2    max_nombre CONSTANT number(3) := 20;
  3  BEGIN
  4    dbms_output.put_line ('Les nombres divisibles par 4 sont : ');
  5    FOR i IN 1..max_nombre LOOP
  6      IF (mod(i,4) = 0) THEN -- mod(i,4) : reste de la division de i par 4
  7         dbms_output.put_line (to_char(i) || ' - ');
  8      END IF;
  9    END LOOP;
 10  END;
 11  /
Les nombres divisibles par 4 sont :
4 -
8 -
12 -
16 -
20 -

PL/SQL procedure successfully completed.

SQL> _
```

Structures itératives

```
------ Affichage des nombres divisibles par 4 compris entre 1 et 20
------ Boucle WHILE
DECLARE
  max_nombre CONSTANT number(3) := 20;
  i number (2);
BEGIN
  dbms_output.put_line ('Les nombres divisibles par 4 sont : ');
  i := 1;
  WHILE (i <= 20) LOOP
    IF (mod(i,4) = 0) THEN -- mod(i,4) : reste de la division de i par 4
       dbms_output.put_line (to_char(i) || ' - ');
    END IF;
    i := i + 1;
  END LOOP;
END;
/
```

Structures itératives

```
------ Affichage des nombres divisibles par 4 compris entre 1 et 20
------ Boucle LOOP
DECLARE
  max_nombre CONSTANT number(3) := 20;
  i number (2);
BEGIN
  dbms_output.put_line ('Les nombres divisibles par 4 sont : ');
  i := 1;
  LOOP
    IF (mod(i,4) = 0) THEN -- mod(i,4) : reste de la division de i par 4
       dbms_output.put_line (to_char(i) || ' - ');
    END IF;
    i := i + 1;
    EXIT WHEN (i > 20);
  END LOOP;
END;
/
```

SELECT …. INTO … (mono – ligne)

- Toute valeur de colonne est rangée dans une variable avec INTO

```
DECLARE
   nom       emp.ename%type;
   fonction  emp.job%type;
   salaire   emp.sal%type;
BEGIN
   SELECT ename, job, sal
   INTO nom, fonction, salaire
   FROM emp
   WHERE empno = 7369;

   dbms_output.put_line ('Nom de emp 7369 : '||nom);
   dbms_output.put_line ('Fonction de emp 7369 : '||fonction);
   dbms_output.put_line ('Salaire de emp 7369 : '||salaire);
END;
/
```

SELECT …. INTO … (mono – ligne)

- Variable ROWTYPE

```
DECLARE
   vdept dept%rowtype;
BEGIN
   SELECT *
   INTO vdept
   FROM dept
   WHERE deptno = 40;

   DBMS_OUTPUT.PUT_LINE('Num dept 40 : '|| vdept.deptno);
   DBMS_OUTPUT.PUT_LINE('Nom dept 40 : '|| vdept.dname);
END;
/
```

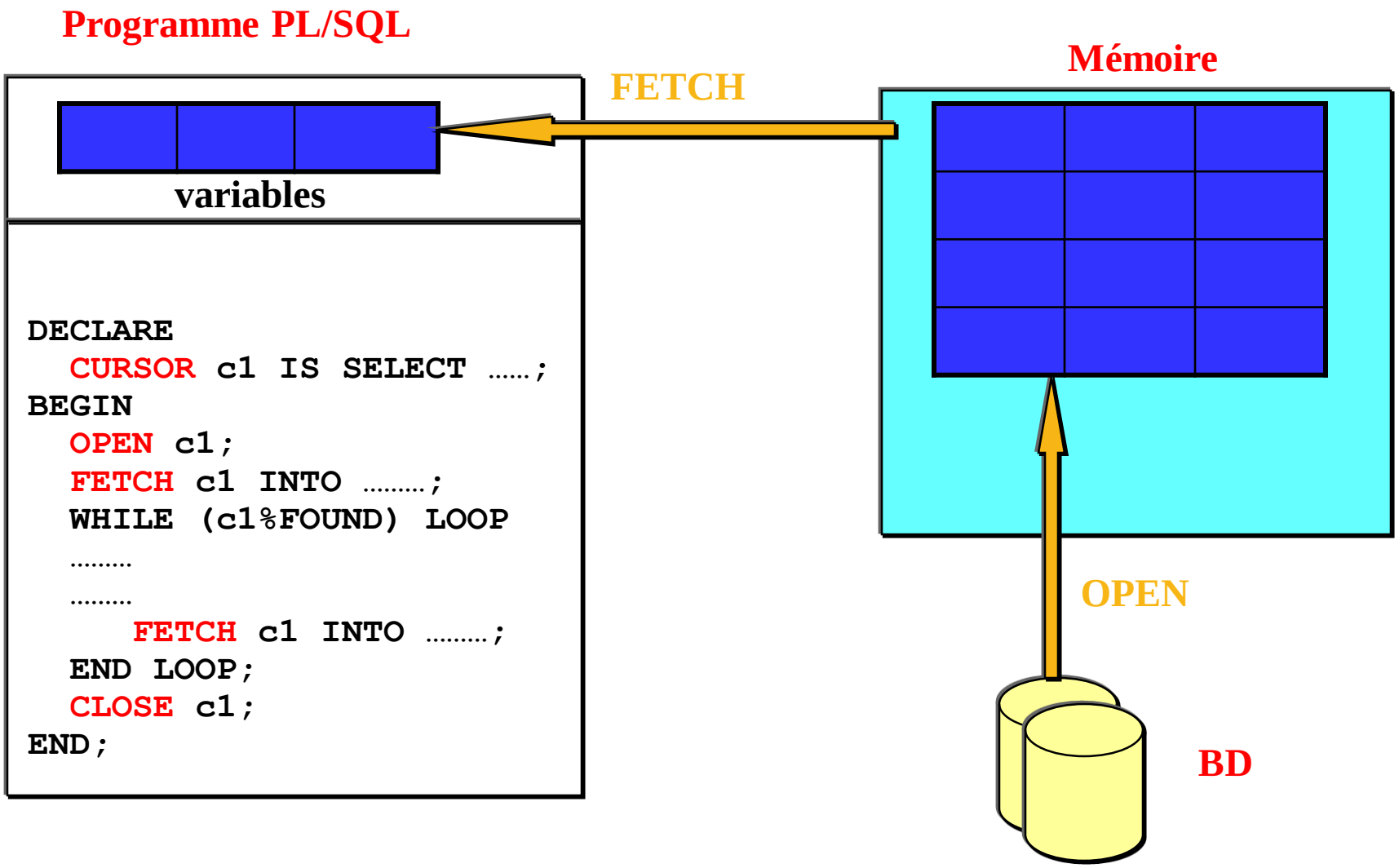
Curseurs (Sélection multi – ligne)

- Curseur : Structure de données permettant de stocker le résultat d'une requête qui retourne plusieurs lignes
- Curseur implicite : généré et géré par le noyau de Oracle pour chaque ordre SQL
  - SELECT t.* FROM table t WHERE ……
  - t est un curseur utilisé par SQL
- Curseur explicite : généré par l'utilisateur pour traiter un ordre SELECT qui ramène plusieurs lignes
  - Déclaration
  - Ouverture du curseur
  - Traitement des lignes
  - Fermeture du curseur

Démarche générale des curseurs

- Déclaration du curseur : DECLARE
  - Ordre SQL sans exécution
- Ouverture du curseur : OPEN
  - SQL ‘monte‘ les lignes sélectionnées en mémoire Verrouillage préventif possible
- Sélection d’une ligne : FETCH
  - Chaque FETCH ramène une ligne dans le programme client
  - Tant qu’il existe une ligne en mémoire
- Fermeture du curseur : CLOSE
  - Récupération de l’espace mémoire

Traitement d'un curseur



Attributs d'un Curseur

- Curseur%FOUND
  - Variable booléenne : est égal à TRUE si le dernier FETCH a retourné un résultat
- Curseur%NOTFOUND
  - Variable booléenne opposée au précédent : est égal à TRUE si le dernier FETCH n'a pas retourné un résultat
- Curseur%ROWCOUNT
  - Variable numérique : retourne le nombre de lignes lues
- Curseur%ISOPEN
  - Variable booléenne : est égal à TRUE si le curseur est ouvert

Gestion ‘classique’ d’un curseur

```
DECLARE
  CURSOR emp_cursor IS                    -- déclarer le curseur
        SELECT ename, dname FROM emp, dept
        WHERE emp.deptno = dept.deptno;
  emp_record emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;                        -- Ouvrir le curseur
  LOOP
   FETCH emp_cursor INTO emp_record;      -- avancer au tuple suivant
   EXIT WHEN emp_cursor%NOTFOUND;         -- sortir si la fin du curseur
                                          -- est détectée
   DBMS_OUTPUT.PUT_LINE('L''employé '|| emp_record.ename ||
            ' travaille dans le département ' || emp_record.dname);
  END LOOP;
  CLOSE emp_cursor;                       -- fermer le curseur
END;
/
```

Gestion ‘classique’ d’un curseur

Administrateur : C:\windows\system32\cmd.exe - sqlplus / as sysdba

```
SQL>
SQL> DECLARE
  2    CURSOR emp_cursor IS              -- déclarer le curseur
  3     SELECT ename, dname FROM emp, dept
  4     WHERE emp.deptno = dept.deptno;
  5    emp_record emp_cursor%ROWTYPE;
  6  BEGIN
  7    OPEN emp_cursor;                  -- Ouvrir le curseur
  8    LOOP
  9     FETCH emp_cursor INTO emp_record;       -- avancer au tuple suivant
 10     EXIT WHEN emp_cursor%NOTFOUND;  -- sortir si la fin du curseur
 11                                     -- est détectée
 12     DBMS_OUTPUT.PUT_LINE('L''employé '|| emp_record.ename ||
 13             ' travaille dans le département ' || emp_record.dname);
 14    END LOOP;
 15    CLOSE emp_cursor;                         -- fermer le curseur
 16  END;
 17  /
L'employé SMITH travaille dans le département RESEARCH
L'employé ALLEN travaille dans le département SALES
L'employé WARD travaille dans le département SALES
L'employé JONES travaille dans le département RESEARCH
L'employé MARTIN travaille dans le département SALES
L'employé BLAKE travaille dans le département SALES
L'employé CLARK travaille dans le département ACCOUNTING
L'employé SCOTT travaille dans le département RESEARCH
L'employé KING travaille dans le département ACCOUNTING
L'employé TURNER travaille dans le département SALES
L'employé ADAMS travaille dans le département RESEARCH
L'employé JAMES travaille dans le département SALES
L'employé FORD travaille dans le département RESEARCH
L'employé MILLER travaille dans le département ACCOUNTING

PL/SQL procedure successfully completed.

SQL>
```

Gestion ‘classique’ d’un curseur

```
DECLARE
  CURSOR emp_cursor IS                    -- déclarer le curseur
        SELECT ename, dname FROM emp, dept
        WHERE emp.deptno = dept.deptno;
  emp_record emp_cursor%ROWTYPE;
BEGIN
  OPEN emp_cursor;                        -- Ouvrir le curseur
  FETCH emp_cursor INTO emp_record;       -- avancer au premier tuple
  WHILE (emp_cursor%FOUND) LOOP           -- sortir si aucun tuple
                                          -- n’a été ramené par
                                          -- le dernier fetch
    DBMS_OUTPUT.PUT_LINE('L''employé '|| emp_record.ename ||
           ' travaille dans le département ' || emp_record.dname);

    FETCH emp_cursor INTO emp_record;-- avancer au tuple suivant
  END LOOP;
  CLOSE emp_cursor;                       -- fermer le curseur
END;
/
```

Gestion ‘automatique’ d’un curseur

```
DECLARE
  CURSOR emp_cursor IS                    -- déclarer le curseur
        SELECT ename, dname FROM emp, dept
        WHERE emp.deptno = dept.deptno;
BEGIN
   -- Ouverture, parcours et fermeture automatique du curseur
   FOR emp_record IN emp_cursor LOOP
      DBMS_OUTPUT.PUT_LINE('L''employé '|| emp_record.ename ||
           ' travaille dans le département ' || emp_record.dname);
  END LOOP;
END;
/
```

Gestion ‘automatique’ d’un curseur

```
BEGIN
   -- Ouverture, parcours et fermeture automatique du curseur
   FOR emp_record IN ( SELECT ename, dname FROM emp, dept
                       WHERE emp.deptno = dept.deptno)
   LOOP
      DBMS_OUTPUT.PUT_LINE('L''employé '|| emp_record.ename ||
           ' travaille dans le département ' || emp_record.dname);
  END LOOP;
END;
/
```

Gestion des Exceptions

- Toute erreur (SQL ou applicative) entraîne automatiquement un débranchement vers le paragraphe EXCEPTION :

```
BEGIN
      instruction1;
      instruction2;
      ……
      instructionn;
EXCEPTION
      WHEN exception1 THEN
      ……
      WHEN exception2 THEN
      ……
      WHEN OTHERS THEN
      ……
END;
```

Débranchement involontaire (erreur SQL) ou volontaire (erreur applicative)

Gestion des Exceptions

- Exceptions SQL déclenchée implicitement par une erreur Oracle
  - Déjà définies (pas de déclaration) :
    - DUP_VAL_ON_INDEX
    - NO_DATA_FOUND
    - OTHERS
  - Non définies
    - Déclaration obligatoire avec le n° erreur (sqlcode)

```
nomerreur EXCEPTION;
PRAGMA EXCEPTION_INIT(nomerreur,n°erreur);
```

- Exceptions applicatives déclenchée explicitement par le programme
  - Déclaration sans n°erreur (section DECLARE)

```
nomerreur EXCEPTION;
```

  - Lever l'exception (section BEGIN)

```
RAISE nomerreur;
```

Gestion des Exceptions

- Fonctions PL/SQL pour la gestion d'erreurs
  - SQLCODE : renvoie la valeur numérique associée à la dernière exception détectée
  - SQLERRM : renvoi le message associé au code de l'erreur
- Exceptions prédéfinies

Nom	Code erreur	Sqlcode	Description
NO_DATA_FOUND	ORA-01403	-1403	SELECT mono-ligne retournant 0 ligne
TOO_MANY_ROWS	ORA-01422	-1422	SELECT mono-ligne retournant plus d'1 ligne
DUP_VAL_ON_INDEX	ORA-00001	-1	Insertion d'une ligne en doublon
VALUE_ERROR	ORA-06502	-6502	Erreur arithmétique, conversion ou limite de taille
ZERO_DIVIDE	ORA-01476	-1476	Division par zéro
CURSOR_ALREADY_OPEN	ORA-06511	-6511	Ouverture d'un curseur déjà ouvert
INVALID_NUMBER	ORA-01722	-1722	Echec sur une conversion d'un chaîne de caractères vers un nombre

…

Exemple de gestion d'exception

```
DECLARE
   erreur        EXCEPTION;              ---- Déclaration exception
BEGIN

   SELECT ……

   IF …… THEN RAISE erreur;              ---- Levée exception
   ……
EXCEPTION
   WHEN NO_DATA_FOUND THEN               ---- Exception prédéfinie
        ……
   WHEN erreur THEN                      ---- Traitement exception
        ……
   WHEN OTHERS THEN                      ---- Exception prédéfinie
        ……
END;
```

Exemple de gestion d'exception

```
DECLARE
        c INTEGER;
        x_aucun_emp EXCEPTION;                    ----  Déclaration exception
BEGIN
   SELECT COUNT(*) INTO c FROM emp;
   IF c=0 THEN
      RAISE x_aucun_emp;                          ----  Levée exception
   ELSE
      DBMS_OUTPUT.PUT_LINE('La table EMP contient '||c||' employés');
  END IF;
EXCEPTION
   WHEN x_aucun_emp THEN                          ----  Traitement exception
       DBMS_OUTPUT.PUT_LINE('La table EMP est vide');
   WHEN OTHERS THEN                               ----  Exception prédéfinie
       DBMS_OUTPUT.PUT_LINE('Erreur inconnue numéro ' || SQLCODE);
END;
/
```

Exemple de gestion d'exception

```
DECLARE
        c INTEGER;
        x_aucun_emp EXCEPTION;              ----  Déclaration exception
BEGIN
   SELECT COUNT(*) INTO c FROM emp;
   IF c=0 THEN
      RAISE x_aucun_emp;                    ----  Levée exception
   .....
EXCEPTION
  WHEN x_aucun_emp THEN                     ----  Traitement exception
      RAISE_APPLICATION_ERROR(-20501, 'La table EMP est vide');
   .....
END;
/
```

La procédure **RAISE_APPLICATION_ERROR** permet de définir des messages d'erreur personnalisés en indiquant :

- **numero_erreur :** représente un entier négatif compris entre -20000 et -20999
- **message :** représente le texte du message d'une longueur maximum de 2048 octets

Exemple de gestion d'exception

```
DECLARE
      violation_cle_etrangere  EXCEPTION;
      PRAGMA EXCEPTION_INIT(violation_cle_etrangere,-2291);
BEGIN
       INSERT INTO fils VALUES ( …… );

EXCEPTION
       WHEN violation_cle_etrangere THEN
       ……
       WHEN OTHERS THEN
       ……
END;
```

Il est possible d'associer un code erreur Oracle à une variable exception à l'aide de **PRAGMA EXCEPTION_INIT**, dans la section déclarative