Systèmes de Gestion de Bases de Données - 2e

PL-SQL - Chapitre 2 - Les types de données et les variables

Daniel Schreurs

7 novembre 2021

Haute École de Province de Liège

Table des matières du chapitre i

Table des matières du chapitre ii

Types de données scalaires

Table des matières de la section : Types de données scalaires i

Table des matières de la section : Types de données scalaires ii

Table des matières de la section : Types de données scalaires iii

Types de données scalaires

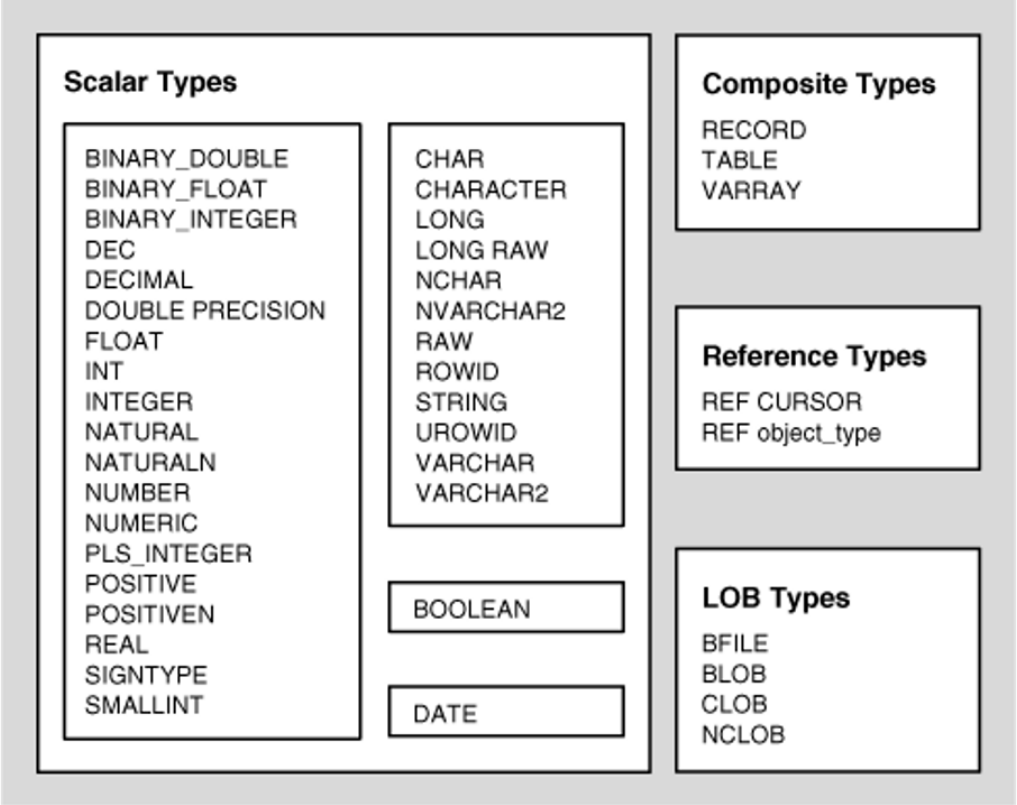

FIGURE 1 – Types de données scalaires

Types de données scalaires : Quatre catégories

Scalaires	Ces types sont atomiques : ce sont les types utilisés pour définir une colonne d'une table
Composés	Ces types comprennent plus d'un élément ou composant
Références	Ces types permettent de définir des références vers d'autres types
Grands Objets (LOB = Large Binary Object)	Ces types de données spécifient la localisation d'un "grand objet" binaire comme une image stockée dans la base de données ou un fichier externe.

Types de données scalaires : Quatre catégories

1. Types de données numériques
2. Types de données ”caractères”
3. Type de données booléen
4. Types de données `DATE`, `TIME` et `INTERVAL`

Types de données scalaires : Types de données numériques

- `NUMBER[(P, S)]`
- `BINARY_INTEGER`
- `PLS_INTEGER`
- `BINARY_FLOAT` et `BINARY_DOUBLE`

Types de données scalaires : Types de données numériques NUMBER[(P, S)]

- P : nombre de chiffres significatifs
- S : nombre de décimales
- `NUMBER` possède quelques sous-types permettant une compatibilité ANSI/ISO dont, entre autres :
- `DEC` ou `DECIMAL` : nombre, virgule fixe, de 38 chiffres max
- `INTEGER`, `INT` ou `SMALLINT` : entier de 38 chiffres max

Types de données scalaires : Types de données numériques BINARY_INTEGER

- `BINARY_INTEGER` Nombres entiers signés compris entre -2.147.483.647 et +2.147.483.647
- Types dérivés :
  - `NATURAL` ou `POSITIVE` : Permettent de restreindre uniquement aux valeurs non négatives (pour `NATURAL`) ou positives (pour `POSITIVE`)
  - `NATURALN` ou `POSITIVEN` : Idem que pour NATURAL et POSITIVE, mais n'acceptent pas la valeur nulle
  - `SIGNTYPE` : Retreint un entier aux valeurs -1, 0, 1

Types de données scalaires : Types de données numériques PLS_INTEGER

- Il est **plus performant** que `NUMBER` et `BINARY_INTEGER` qui utilisent des librairies arithmétiques
- Même s'il possède le même rang que `BINARY_INTEGER`, il ne se comporte pas toujours de la même manière
- Alors que `BINARY_INTEGER` ne génère pas une **exception** lorsqu'un dépassement de capacité intervient lors d'un calcul si le résultat est transféré dans un `NUMBER`, `PLS_INTEGER` génère bien une exception
- Pour des raisons de compatibilité, il est toujours possible d'utiliser `BINARY_INTEGER`, mais il est conseillé d'utiliser `PLS_INTEGER` dans les nouvelles applications

Types de données scalaires : Types de données numériques BINARY_FLOAT

- `BINARY_FLOAT` et `BINARY_DOUBLE`
- Ces 2 types de données sont nouveaux depuis Oracle 10g
- Il s'agit de nombres floating-point de format IEEE-754 simple et double précision
- Ils permettent d'augmenter la performance d'applications nécessitant certains types de calculs notamment certaines applications manipulant des données scientifiques.

Types de données scalaires : Types de données numériques CHAR

- `CHAR [(maximum size [CHAR | BYTE])]`
- Chaînes de longueur fixe
- Stockage dépendant du jeu de caractères de DB
- Taille exprimée par défaut en bytes (maximum 32767)

ATTENTION : Aux jeux de caractères multi byte. On préfère `CHAR(20 CHAR)`

Types de données scalaires : Types de données numériques VARCHAR2

- VARCHAR2 (maximum size [CHAR | BYTE])
- Chaînes de longueur variable
- Stockage dépendant du jeu de caractères de DB
- Taille exprimée par défaut en bytes (maximum 32767)

Types de données scalaires : Types de données numériques LONG

- `LONG` et `LONG RAW`
- Le type `LONG` est similaire à `VARCHAR2` sauf que le nombre maximum de bytes est 32760
- `LONG RAW` est similaire mais n'est pas interprété par le PL/SQL
- Les variables de type `LOB` sont destinées à remplacer les types `LONG` et `LONG RAW`

Types de données scalaires : Types de données numériques RAW

- `RAW (maximum size)`
- Ce type de données permet de traiter des données binaires ou des caractères en binaire.

Types de données scalaires : Types de données numériques ROWID

- ROWID [language=xml]!UROWID!
- De manière interne, chaque table de base de données possède une pseudo-colonne ROWID qui comprend une valeur binaire. Chacune de ces valeurs (rowid) représente l'adresse de stockage de la ligne correspondante.
- ROWID permet uniquement de traiter des rowids physiques UROWID (rowids universels) permet de traiter les rowids physiques, logiques et "étranger", c'est-à-dire vers des tables étrangères (non Oracle).

La fonction CAST permet de convertir une donnée de type rowid

```
1 SELECT CAST(ROWID AS CHAR(30))
2 FROM DUAL;
```

Types de données scalaires : Types de données numériques NCHAR

- `NCHAR[(maximum size)]` et `NVARCHAR2` (maximum size)
- Les types de données NCHAR et NVARCHAR2 sont identiques au types CHAR et VARCHAR2 mais les données sont traitées et stockéeS au format du jeu de caractères national.

Types de données scalaires : Types de données DATE, TIME et INTERVAL

- Type `DATE`
- Type `TIMESTAMP [(précision)]`
- Type `INTERVAL DAY [(précision)] TO SECOND [(précision)]`
- Type `INTERVAL YEAR [(précision)] TO MONTH`

Types de données LOBS

Types de données LOBS

- Sont regroupés sous cette appellation tous les large objects
- Ils peuvent contenir des données non structurées pouvant aller jusqu'à 4 gigabytes.
- Nous ne les aborderons pas dans le cadre de ce cours.

Structure d’un bloc

Structure d’un bloc

Structure d’un bloc

```
1 DECLARE
2   -- Une section de déclaration de variables
3 BEGIN
4   -- Une section d'instructions
5 EXCEPTION
6   -- Une section d'exception
7 END;
```

Structure d’un bloc

- Une section de déclaration de variables
  - Permet de stocker des données récupérées par des instructions SQL dans les colonnes des tables. Elles peuvent également conserver des données internes au bloc. Les variables peuvent être scalaires ou composées Les variables doivent être obligatoirement déclarées avant leur utilisation
- Une section d’instructions
- Une section d’exception

Déclaration de variables

Déclaration de variables

Déclaration de variables

```
1 Nom_Variable [CONSTANT] TYPE [NOT NULL]
2   [{DEFAULT | := ] VALEUR];
```

- `Nom_variable` : ce nom de variable doit être unique dans le bloc
- `TYPE` : le type de la variable qui peut être un type scalaire ou un type composite
- `CONSTANT` : La valeur de la variable est une constante qui ne sera pas modifiable dans le bloc
- `NOT NULL` : la variable doit obligatoirement être assignée, sinon une erreur est générée
- `:= VALEUR` : la variable est initialisée avec la valeur

Déclaration de variables

Une variable déclarée NOT NULL doit avoir une affectation d'initialisation

```
1  DECLARE
2      Vuser           VARCHAR2(35) := USER;
3      -- PLS-00218: a variable declared NOT NULL must have an
           initialization assignment
4      VNotNull        INTEGER NOT NULL;
5      Vdefaut         INTEGER DEFAULT 1;
6      Vconst CONSTANT INTEGER      := 1;
7  BEGIN
8      DBMS_OUTPUT.PUT_LINE('Vuser : ' || Vuser);
9      DBMS_OUTPUT.PUT_LINE('VNotNull : ' || VNotNull);
10     DBMS_OUTPUT.PUT_LINE('Vdefaut : ' || Vdefaut);
11     DBMS_OUTPUT.PUT_LINE('Vconst : ' || Vconst);
12 EXCEPTION
13     WHEN OTHERS THEN
14         DBMS_OUTPUT.PUT_LINE(SQLERRM);
15 END;
```

Déclaration de variables

L'expression 'VCONST' ne peut pas être utilisée comme cible d'une affectation

```
1  DECLARE
2    VNotNull  INTEGER NOT NULL := 1;
3    Vconst  CONSTANT INTEGER := 1;
4  BEGIN
5    DBMS_OUTPUT.PUT_LINE('VNotNull : ' || VNotNull);
6    DBMS_OUTPUT.PUT_LINE('Vconst : ' || Vconst);
7    VNotNull := 0;
8    -- PLS-00363: expression 'VCONST' cannot be used as an assignment
         target
9    Vconst := 0;
10 EXCEPTION
11   WHEN OTHERS THEN
12     DBMS_OUTPUT.PUT_LINE (SQLERRM);
13 END;
```

Types composites ou composés

Types composites ou composés

- Ces types de données font partie des types de données définis par le programmeur PL/SQL
- Parmi ces types, on retrouve également les types dérivés ou sous-types, objets de la section suivante.

Types composites ou composés : Type RECORD

Le type de données RECORD permet de déclarer des types de variables composites.

- Un RECORD est l'équivalent de la définition d'un enregistrement ou d'une structure de données d'un langage de 3ème génération
- Il doit faire l'objet d'une déclaration d'un type de données et ensuite une variable peut être définie à l'aide de ce type

Types composites ou composés : Type RECORD

Donc, pour déclarer un RECORD, on doit passer par deux étapes distinctes :

- Déclarer un TYPE correspondant à la structure voulue
- Utiliser ce TYPE pour déclarer une variable

Types composites ou composés : Type RECORD

Exemple : (cfr schéma SCOTT table emp)

```
1  DECLARE
2     TYPE TupleEmploye IS RECORD (
3      EmpNo NUMBER(4),
4      Ename VARCHAR2(10),
5      Job   CHAR(9),
6      Mgr   NUMBER(4),
7      HireDate  DATE,
8      Sal   NUMBER(7,2),
9      Comm  NUMBER(7,2),
10     DeptNo  NUMBER(2));
11     UnEmploye TupleEmploye; -- On se sert du record
12 BEGIN
13
14 EXCEPTION
15
16 END;
```

Types composites ou composés : Type RECORD

Exemple : Initialisation d'un RECORD

```
1  DECLARE
2      TYPE TupleEmploye IS RECORD
3                           (
4                               EmpNo    NUMBER(4),
5                               Ename    VARCHAR2(10),
6                               Job      CHAR(9),
7                               Mgr      NUMBER(4),
8                               HireDate DATE,
9                               Sal      NUMBER(7, 2),
10                              Comm     NUMBER(7, 2),
11                              DeptNo   NUMBER(2)
12                          );
13     UnEmploye TupleEmploye;
14 BEGIN
15     SELECT * INTO UnEmploye FROM emp WHERE empno = 7788;
16     DBMS_OUTPUT.PUT_LINE('Le client 7788 : ' || UnEmploye.Ename
17         || ' ' || UnEmploye.Job);
18 END;
```

Types composites ou composés : Utilisation de %TYPE

- L'attribut `%TYPE` permet de déclarer une variable dont le type est basé sur le type d'une colonne ou d'une autre variable
- Les variables déclarées à l'aide de l'attribut `%TYPE` héritent du type de données de la colonne ou de la variable référencée mais non automatiquement des contraintes (comme `NOT NULL` ou `DEFAULT`)

Types composites ou composés : Utilisation de %TYPE

Utilisation de %TYPE

```
1  DECLARE
2      TYPE TupleEmploye IS RECORD
3                           (
4                               EmpNo    Emp.Empno%TYPE,
5                               Ename    Emp.Ename%TYPE,
6                               Job      Emp.Job%TYPE,
7                               Mgr      Emp.Mgr%TYPE,
8                               HireDate Emp.HireDate%TYPE,
9                               Sal      Emp.Sal%TYPE,
10                              Comm     Emp.Comm%TYPE,
11                              DeptNo   Emp.DeptNo%TYPE
12                          );
13
14     UnEmploye TupleEmploye;
15 BEGIN
16     SELECT * INTO UnEmploye FROM emp WHERE empno = 7788;
17     DBMS_OUTPUT.PUT_LINE('Le client 7788 : ' ||
18                          UnEmploye.Ename || ' ' || UnEmploye.Job);
19 END;
```

Types composites ou composés : Utilisation de %TYPE

ma_variable hérite de la contrainte NOT NULL

```
1 DECLARE
2     VariableDeBase NUMBER NOT NULL := 0;
3     ma_variable    VariableDeBase%TYPE;
4 BEGIN
5     NULL;
6 END;
```

Types composites ou composés : Utilisation de %TYPE

ma_variable n'hérite pas de la clause DEFAULT

```
1 DECLARE
2     VariableDeBase NUMBER DEFAULT 20;
3     ma_variable    VariableDeBase%TYPE;
4 BEGIN
5     DBMS_OUTPUT.PUT_LINE('Variable de base ' ||
6                          VariableDeBase);
7     DBMS_OUTPUT.PUT_LINE('Variable dérivée ' ||
8                          ma_variable);
9 END;
```

Types composites ou composés : Utilisation de %ROWTYPE

Utilisation de %ROWTYPE

```
1 DECLARE
2     UnEmploye Emp%ROWTYPE;
3 BEGIN
4     SELECT * INTO UnEmploye FROM emp WHERE empno = 7788;
5     DBMS_OUTPUT.PUT_LINE('Le client 7788 : ' ||
6                          UnEmploye.Ename || ' ' || UnEmploye.Job);
7 END;
```

Il n'est **pas** nécessaire de **connaître le nom de toutes les colonnes** d'une table ni le nombre et l'ordre de ces colonnes pour effectuer une sélection.

Types composites ou composés : Utilisation de %ROWTYPE

Portée des variables

```
1 DECLARE
2     Ename Emp.Ename%TYPE := 'SCOTT';
3 BEGIN
4     -- Le nom de la colonne l'emporte sur le nom de la variable.
5     DELETE FROM Emp WHERE Ename = Ename; -- toujours VRAI
6 END ;
```

Types composites ou composés : Utilisation de %ROWTYPE

Portée des variables : solution

```
1 <<Main>>
2 DECLARE
3     Ename Emp.Ename%TYPE := 'SMITH';
4 BEGIN
5     --On précise avec la portée de la variable. On est dans Main.
6     DELETE FROM Emp WHERE Main.Ename = Ename;
7 END;
```

Conversion de types

Conversion de types : Conversions explicites

Fonction normalisée CAST disponible depuis Oracle 9iR2.

Exemple

```
1 SELECT CAST('15/11/2021' AS DATE)
2 FROM DUAL;
```

Conversion de types : Conversions implicites

Fonction normalisée CAST disponible depuis Oracle 9iR2.

Exemple : conversions implicites

```
1  DECLARE
2      TimeStart   CHAR(5);
3      TimeFinish  CHAR(5);
4      TimeElapsed NUMBER(5);
5  BEGIN
6      -- recherche du nb de secondes écoulées depuis minuit
7      SELECT TO_CHAR(CURRENT_DATE, 'SSSSS') INTO TimeStart FROM DUAL;
8      -- …
9      -- recherche du nb de secondes écoulées depuis minuit
10     SELECT TO_CHAR(CURRENT_DATE, 'SSSSS') INTO TimeFinish FROM DUAL;
11     -- La soustraction est effectuée car convertion automatique
12     TimeElapsed := TimeFinish - TimeStart;
13     DBMS_OUTPUT.PUT_LINE('Temps écoulé = ' || TimeElapsed);
14 END;
```

Types REF

Types REF

- Les types REF (référence) représentent des valeurs correspondant à des pointeurs qui permettent de référencer d'autres objets du code.
- Certaines de ces utilisations relèvent des extensions objet-relationnelles. [1]

1. Nous ne les aborderons pas.

Visibilité des variables

Visibilité des variables

- Une variable est référencée par son nom et est visible dans le bloc PL/SQL où elle a été déclarée.
- Elle est visible également dans tous les blocs imbriqués dans celui où la déclaration a été faite pour autant qu'elle n'ait pas été redéfinie.

Visibilité des variables

Visibilité des variables

```
1  DECLARE
2      VLoc  VARCHAR2(20);
3      Vuser VARCHAR2(50) := USER;
4  BEGIN
5      VLoc := 'Bloc Principal';
6      DECLARE
7          VLoc VARCHAR2(50);
8      BEGIN
9          VLoc := 'Bloc imbriqué';
10         DBMS_OUTPUT.PUT_LINE('VLoc = ' || VLoc ||
11                              ', Vuser = ' || Vuser);
12     END;
13     -- Ici VLoc du bloc imbriqué n'est plus connu
14     DBMS_OUTPUT.PUT_LINE('VLoc = ' || VLoc ||
15                          ', Vuser = ' || VUser);
16 END;
17 -- VLoc = Bloc imbriqué, Vuser = USER_SGBD_2
18 -- VLoc = Bloc Principal, Vuser = USER_SGBD_2
```

Opérateurs et expressions

Opérateurs et expressions

- Les expressions PL/SQL sont construites en appliquant des opérateurs à des opérandes.
- Une opérande peut être une variable, une constante ou par exemple un littéral
- Les expressions sont évaluées suivant la préséance des opérateurs

Opérateurs et expressions

Opérateur	Opération
**	Exponentiation
-	Opposé
*, /	Multiplication, division
+, -, ||	Addition, soustraction concaténation
=, <, >, <=, >=, <>, !=, ~=, ^=, IS NULL, LIKE, BETWEEN, IN	Comparaison
NOT	Négation logique
AND	Conjonction (ET Logique)
OR	OU Logique

Table 1 – Opérateurs et expressions

<>, !=, ~=, ^= : 4 manière d'exprimer la différence.

Opérateurs et expressions

Important

PL/SQL utilise des évaluations d'expression appelées "short-circuit"

Dès que le résultat de l'expression peut être déterminé, l'évaluation s'arrête

Ne générera pas d'erreur

```
1 Vcontinue BOOLEAN := FALSE;
2 IF ((NOT Vcontinue) OR ((41/0) > 0))
```

Logique trivalente et valeur NULL

Logique trivalente et valeur NULL

- VRAI
- FAUX
- INCONNU

Logique trivalente et valeur NULL

Norme SQL2 : `COALESCE (expr [, expr] ...)` Fonctions Oracle

- NVL
- NULLIF
- NVL2

Séq. et pseudo-colonnes du PL/SQL

Séq. et pseudo-colonnes du PL/SQL

Le PL/SQL reconnaît les pseudo-colonnes de SQL telles que ROWID, CURRVAL, NEXTVAL, LEVEL et ROWNUM

Exemple

```
1  DECLARE
2      VRowid ROWID;
3      VSal   Emp.Sal%TYPE;
4  BEGIN
5      SELECT Sal, ROWID
6      INTO VSal, VRowid
7      FROM emp
8      WHERE Empno = 7369;
9      VSal := 1000;
10     UPDATE Emp SET sal = VSal WHERE rowid = VRowid;
11 END;
```

Séq. et pseudo-colonnes du PL/SQL

Exemple

```
1  CREATE SEQUENCE seqCours START WITH 1;
2  CREATE TABLE Cours
3  (
4      Nr  NUMBER,
5      Nom VARCHAR2(20)
6  );
7
8  DECLARE
9      VNr Cours.Nr%TYPE;
10 BEGIN
11     INSERT INTO Cours VALUES (SeqCours.NEXTVAL, 'SGBD');
12     SELECT Nr INTO VNr FROM Cours WHERE RTRIM(Nom) = 'SGBD';
13     DBMS_OUTPUT.PUT_LINE('VNr = ' || VNr);
14     INSERT INTO Cours VALUES (SeqCours.NEXTVAL, 'C#');
15     SELECT Nr INTO VNr FROM Cours WHERE RTRIM(Nom) = 'C#';
16     DBMS_OUTPUT.PUT_LINE('VNr = ' || VNr);
17     SELECT SeqCours.CURRVAL INTO VNr FROM DUAL;
18     DBMS_OUTPUT.PUT_LINE('VNr =' || VNr);
19 END;
```

Séq. et pseudo-colonnes du PL/SQL

- `ROWNUM` permet de spécifier l'ordre dans lequel une ligne est sélectionnée dans une table. Le premier tuple sélectionné porte le ROWNUM 1, le deuxième le
- `ROWNUM2` et ainsi de suite.
- `ROWNUM` est attribué avant l'opération de tri (`ORDER BY`)

Séq. et pseudo-colonnes du PL/SQL

Exemple

```
1  DECLARE
2      TYPE TableEmployes IS TABLE OF Emp%ROWTYPE;
3      LesEmployes TableEmployes;
4  BEGIN
5      SELECT * BULK COLLECT
6      INTO LesEmployes
7      FROM Emp
8      WHERE ROWNUM < 3
9      ORDER BY Ename;
10     DBMS_OUTPUT.PUT_LINE('Nbre Employes : '
11         || LesEmployes.COUNT);
12     DBMS_OUTPUT.PUT_LINE('ROWNUM 1 : '
13         || LesEmployes(1).Ename);
14     DBMS_OUTPUT.PUT_LINE('ROWNUM 2 : '
15         || LesEmployes(2).Ename);
16 END;
```

Séq. et pseudo-colonnes du PL/SQL

Exemple

```
1  DECLARE
2      TYPE TableEmployes IS TABLE OF Emp%ROWTYPE;
3      LesEmployes TableEmployes;
4  BEGIN
5      SELECT * BULK COLLECT
6      INTO LesEmployes
7      FROM (SELECT * FROM Emp ORDER BY Ename)
8      WHERE ROWNUM < 3;
9      DBMS_OUTPUT.PUT_LINE('Nbre Employes : '
10         || LesEmployes.COUNT);
11     DBMS_OUTPUT.PUT_LINE('ROWNUM 1 : '
12         || LesEmployes(1).Ename);
13     DBMS_OUTPUT.PUT_LINE('ROWNUM 2 : '
14         || LesEmployes(2).Ename);
15 END;
```