Cours

Systèmes de Gestion des Bases Données Avancées

Master 1, ISI.
2023-2024

Dilekh Tahar

tahar.dilekh@univ-batna2.dz

Département d'informatique. Faculté des mathématiques et informatique. Université Batna 2

Les systèmes Relationnel Objets : Application à ORACLE/Objet.

Introduction

- Fonctions orientées objets a partir de la version SQL3.
- Toute instruction de SQL classique est toujours valable en SQL3.
- Ce chapitre présente l'extension orientée objets de SQL3, telle qu'offerte par le système de gestion de bases de données Oracle : les **types** (avec structure complexe et méthodes) et les **oids**.

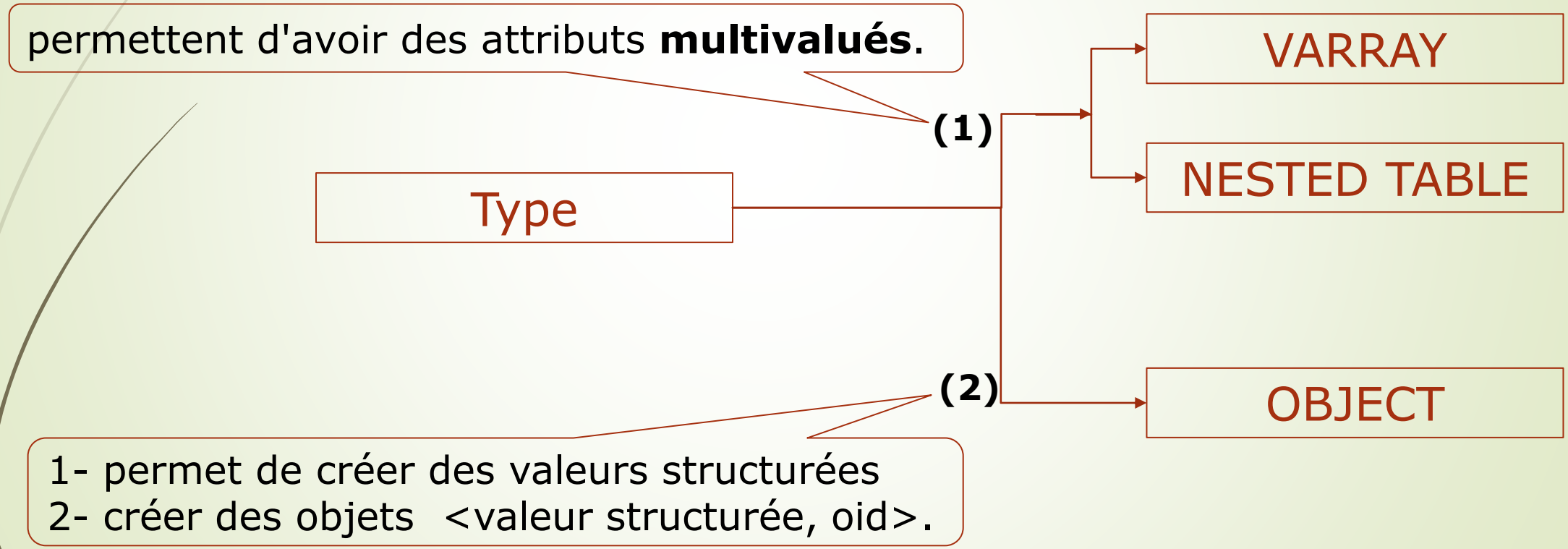
Introduction

Les nouvelles possibilités:

1. Définir de nouveaux domaines à structure complexe, appelés **types**,
2. Associer à chaque type des **méthodes**,
3. Créer des **hiérarchies** de types
4. Créer des **objets** qui sont composés d'une valeur structurée et d'un **oid**,
5. établir des **liens de composition** par des attributs référence qui contiennent l'oid de l'objet composant,
6. Créer des **tables contenant** soit des tuples normaux (en première forme normale), soit des tuples en non première forme normale (des valeurs structurées), soit des **objets**.

1. Types / multivalués / VARRAY

- Les domaines usuels de SQL tels que VarChar, Number, Date…
- Définir de nouveaux domaines (appelés **types**)



1. Types / multivalués / VARRAY

- **Types** pour les attributs **multivalués** :

1. **VARRAY** permettent de créer des attributs **multivalués** sous la forme de **tableaux** à **une dimension**.

   CREATE TYPE *nom-type* ***AS VARRAY*** *(nb-max)* ***OF*** *nom-type2 ;*

   dont chaque élément est un tableau à une dimension, type nom-type2 peut être n'importe quel type, prédéfini de SQL (VarChar, Date, Number…) ou défini par l'utilisateur.

1. Types / multivalués/ NESTED TABLE

- **Types** pour les attributs **multivalués** :

  2. **NESTED TABLE** (table imbriquée) permettent de créer des multivalués sous la forme de tables relationnelles qui vont être incluses dans les autres tables

CREATE TYPE nom-type **AS TABLE OF** nom-type2 ; **(1)**

NESTED TABLE nom-attribut-de-type-table-emboîtée **STORE AS** nom-table-annexe; **(2)**

(1) dont chaque élément est une table relationnelle dont les tuples sont tous du type nom-type2.

l'instruction **(1)** contient une clause supplémentaire **(2)**:

Lors de l'emploi de ce type, **nom-type**, comme domaine d'un attribut d'une autre table, par exemple de l'attribut **A** de la table **T**, SGBD Oracle crée physiquement une table annexe (**NESTED TABLE**) dans laquelle il stockera les tuples (les valeurs) de l'attribut **A**.

1. Types

Les différences entre les types VARRAY et NESTED TABLE:

- **NESTED TABLEs** peuvent être indexées, pas les **VARRAYs** ;
- **NESTED TABLEs** peuvent contenir un nombre quelconque de tuples , à la différence de les **VARRAYs** ont une taille maximale déclarée à leur création;
- les valeurs (tuples) des **NESTED TABLEs** ne sont pas ordonnées.

1. Types / OBJECT

OBJECT créent des valeurs structurées et/ou des objets

CREATE TYPE nom-type **AS OBJECT** (

nom-attr$_1$ nom-type$_1$,

nom-attr$_2$ nom-type$_2$,

… ,

nom-attr$_n$ nom-type$_n$)

- Les types nom-type$_i$ peuvent être n'importe quel type, prédéfini de SQL ou défini par l'utilisateur.
- Cette instruction permet aussi de déclarer un jeu de méthodes associées aux objets de ce type, en utilisant le mot clé **MEMBER**).

1. Types

VARRAY, **NESTED TABLE** et **OBJECT** définissent des structures contenant un seul constructeur.

c.-à-d. pas d'attribut complexe contenant des **attributs complexes** ou **multivalués**, pas d'attribut **complexe et multivalué**.

Solution :il faut définir un type **intermédiaire** par constructeur. **Exemple** :

```
CREATE TYPE T-Personne AS OBJECT (
    nom VARCHAR(20) ,
    prénoms T-ListePrénoms ,
    enfants T-ListeEnfants )
```

```
CREATE TYPE T-Enfant AS OBJECT (
    prénoms T-ListePrénoms ,
    sexe CHAR ,
    dateNais DATE )
```

```
CREATE TYPE T-ListePrénoms AS VARRAY OF VARCHAR(20) ;
```

```
CREATE TYPE T-ListeEnfants AS VARRAY OF T-Enfant ;
```

1. Types

- Les **types** peuvent être utilisés comme :
  - domaines d'attributs dans les relations ,
  - dans les types,
  - pour définir le format de relations.

Exemple : **CREATE TABLE** nom-table **OF** *nom-type* ;

Donc, les tables ne sont plus en *première forme normale*.

1. Types / manipulation des données

1. **Structures complexes** : (attributs complexes et attributs références)

   la notation pointée récursive est utilisée:

   composé.composant-du-premier-niveau. ….

2. **Structures de type collection** : (attributs multivalués, ensembles de tuples, d'objets, de valeurs ou d'oids)
   - Plus le SQL classique (élément IN collection, EXIST collection…)
   - Il faut dans la clause FROM préfixer toute déclaration de variable sur une **collection** qui *n'est pas une **table*** du mot clé **TABLE**

     SELECT … FROM nom-table t , TABLE t.nom-attribut-multivalué v …

- Création des valeurs structurées lors d'insertions ou de mises à jour : T-Voiture ('VD 1212', 'Toyota', 'Yaris', 2000)

T-Voiture (numéro: 'VD 1212', marque: 'Toyota', année: 2000) .

2. Tables

1. Table relationnelle classique :

CREATE TABLE nom-table (nom-attribut$_1$ nom-type$_1$, nom-attribut$_2$ nom-type$_2$, … , nomattribut$_n$ nom-type$_n$)

où les nom-type$_i$ sont des domaines prédéfinis de SQL, tels que VARCHAR, NUMBER ou DATE.

2. Table relationnelle en non première forme normale

CREATE TABLE nom-table (nom-attribut$_1$ nom-type$_1$,nom-attribut$_2$ nom-type$_2$, … , nomattribut$_n$ nom-type$_n$)

où au moins un des nom-typei est le nom d'un type construit (VARRAY, TABLE emboîtée, ou OBJECT)

3. Table d'objets

CREATE TABLE nom-table **OF** nom-type

où nom-type est le nom d'un type OBJECT

3. Identité et attributs-référence

- Chaque type **OBJECT** possède **Oid**, et peut donc être **référencé** par un ***lien de composition*** en utilisant la clause « **REF** nom-type ».

```
CREATE TYPE T-Personne AS OBJECT (
    AVS  NUMBER ,
    nom  VARCHAR(18) ,
    prénom  VARCHAR(18) ,
    conjoint  REF T-Personne,  /*attribut référence qui
                                contiendra un oid de type T-Personne*/
    …. )
```

3. Identité et attributs-référence / manipulation

- Trois nouvelles clauses permettent de manipuler les **oid**s et les valeurs des objets.

1. REF(objet):

```
CREATE TABLE LesPersonnes  OF T-Personne ;


INSERT INTO LesPersonnes (AVS, nom, prénom, conjoint)
    VALUES (17924457911, 'Rochat', 'Philippe',
        (SELECT REF(p) FROM LesPersonnes  p
            WHERE p.nom='Muller' AND p.prénom='Annie' )) ;
```

Insère un objet (Philippe Rochat) et initialise son lien de référence conjoint (sur Annie Muller).

3. Identité et attributs-référence / manipulation

2. **VALUE(objet):**

La clause VALUE(variable-objet) a l'effet complémentaire; elle fournit en résultat la valeur structurée de l'objet (avec le nom de son type).

3. **DEREF(oid):** passer à la valeur de l'objet

SELECT p.nom, **DEREF**(p.conjoint) **FROM** LesPersonnes p;

donnera pour chaque personne p, son nom, et la "valeur" de son conjoint, à savoir : son numéro AVS, son nom, son prénom et l'oid de son conjoint.

- **IS DANGLING :** Cette condition est vraie si et seulement si l'attribut référence n'a pas été initialisé. Par Exemple:

SELECT p.nom **FROM** LesPersonnes p **WHERE** p.conjoint **IS DANGLING**;

4. Hiérarchie de types OBJECT

```
CREATE TYPE T-Personne AS OBJECT (
    AVS  NUMBER ,
    nom  VARCHAR(18) ,
    prénom  VARCHAR(18) ,
    adresse  VARCHAR(200) )
    NOT FINAL /

CREATE TYPE T-Etudiant UNDER T-Personne (
    faculté VARCHAR(18) ,
    cycle  VARCHAR(18) )
    /
```

Permettre de déclarer des sous-types de T-Personne. Sans cette clause n'accepte pas un sous-type

Pour créer un sous-type du type non final T-Personne,

L'héritage multiple est interdit sous oracle.

4. Hiérarchie de types OBJECT

- **Exemple :**

CREATE TABLE *LesPersonnes* **OF** ***T-Personne*** ; **/* création d'une table d'objets */**

INSERT INTO *LesPersonnes* **VALUES** (***T-Person***(11111111, 'Rochat', 'Philippe', 'Rue des oiseaux, 1, Morges') **/* création et insertion d'un objet de type T-Personne */**

INSERT INTO *LesPersonnes* **VALUES** (***T-Etudiant***(22222222, 'Muller', 'Annie', 'Rue du marché, 22, Renens', 'HEC', 'master') **/* création et insertion d'un objet de type T-Etudiant */**

4. Hiérarchie de types OBJECT

- **Exemple (suite):**

CREATE TABLE LivresEmpruntés (livre VARCHAR(20), emprunteur ***T-Personne***); **/* création d'une table de tuples structurés (non 1NF) */**

INSERT INTO LivresEmpruntés **VALUES** ('SQL3', ***T-Person*** (11111111, 'Rochat', 'Philippe', 'Rue des oiseaux, 1, Morges'); **/* insertion d'un tuple structuré dont l'attribut emprunteur est de type T-Personne */**

INSERT INTO LivresEmpruntés **VALUES** ('Les Bases de Données', ***T-Etudiant*** (22222222, 'Muller', 'Annie', 'Rue du marché, 22, Renens', 'HEC', 'master') ; **/* insertion d'un tuple structuré dont l'attribut emprunteur est de type T-Etudiant */**

4. Hiérarchie de types OBJECT

Dans le cas d'une hiérarchie de types, une nouvelle condition permet de tester le type d'une valeur. La condition élémentaire :

<valeur> **IS OF** <nom-sous-type>

est vraie ssi la valeur appartient bien au type <nom-sous-type>.

5. Méthodes

- Chaque type OBJECT peut avoir des méthodes

CREATE TYPE nom-type AS OBJECT(

déclaration des **attributs** ,

déclaration des **signatures des méthodes**)

- Signature d'une méthode

MEMBER FUNCTION nom-méthode (

nom-paramêtre$_1$ [IN / OUT] type$_1$, ...)

RETURN type$_n$

MEMBER PROCEDURE nom-méthode (

nom-paramêtre$_1$ [IN / OUT] type$_1$, ...)

5. Méthodes

Corps de méthodes

- Le corps contient le code de ses méthodes et le code peut contenir des :
  - instructions PL/SQL (ou JAVA)
  - instructions SQL
  - appels de méthodes

5. Méthodes

CREATE TYPE BODY nom-type **AS**

MEMBER *FUNCTION/PROCEDURE* nom-méthode$_1$ (

nom-paramêtre$_{11}$ [IN / OUT] type$_{11}$,

....

)

BEGIN

code de-la-méthode$_1$

END

MEMBER *FUNCTION / PROCEDURE* nom-méthode$_2$ (

nom-paramêtre$_{21}$ [IN / OUT] type$_{21}$,

....

)

BEGIN

code de-la-méthode$_2$

END

....

5. Méthodes

Exemple :

```
create type Tpersonne as object (
      nom varchar2(25),
      prenoms Tliste_prenoms,
      date_naiss date,
      MEMBER function age(date_naiss date) return numeric
);
```

Le corps des méthodes est défini par :

```
create or replace type body nom_type
as
Member nom_methode is
begin
...
end ;
...
end;
```

5. Méthodes

Exemple :

```
Create type T_Formateur as object(
        Id integer,
        nom varchar(15),
        prénom varchar(15),
        Adresse T_Adresse,
        num_telephone T_num_telephone,
        date_naissance date,
        MEMBER FUNCTION age RETURN NUMBER
);
CREATE OR REPLACE TYPE BODY T_Formateur AS
        MEMBER FUNCTION age RETURN NUMBER AS
        BEGIN
                RETURN trunc(months_between(sysdate, date_naissance)/12);
        END age;
END;
```