مخبر هندسة الأنظمة المعقدة

LABORATOIRE D'INGÉNIERIE DES SYSTÈMES COMPLEXES

جامعة باجي مختار-عنابة

Université Badji Mokhtar-Annaba

Administration des bases de données avancées

Master I: Ingénierie des Logiciels Complexes (ILC)

Dr Kamilia MENGHOUR

Laboratoire d'Ingénierie des Systèmes Complexes

Université Badji Mokhtar-Annaba

K_menghour@yahoo.fr

Année Universitaire : 2024- 2025

Chapitre 2

Les modèles avancés de bases de données

Plan du cours

Chapitre 2 : *Les modèles avancés de bases de données*

- Les bases de données orientées objet.
- Bases de données actives.
- Bases de données déductives.
- Bases de données géographiques.
- Les entrepôts de données.

Introduction: limites des SGBDR (1/3)

- ***Force du Modèle Relationnel***
- Le modèle de données est très simple (tables, relations…) et donc facile à comprendre pour les utilisateurs.
- Le modèle repose sur une base formellement définie (normalisation, algèbre, …), ce qui a permis de définir des langages de manipulation de données (LMD) standardisés tels que (QUEL, SQL)
- Efficace pour les applications de gestion classique

Introduction: limites des SGBDR (2/3)

- ***Faiblesses du Modèle Relationnel***
- Modèle de données simple : difficile de modéliser des structures complexes :
  - pas d'attribut complexe, ni multivalué
  - Données alphanumériques uniquement (pas d'images, de sons, de vidéo, …)
  - un seul type de lien (clé étrangère) : jointure indispensable
  - le système de types figé (non extensible)
  - Les objets sont éclatés en plusieurs relations, ce qui multiplie les jointures dans les requêtes des utilisateurs

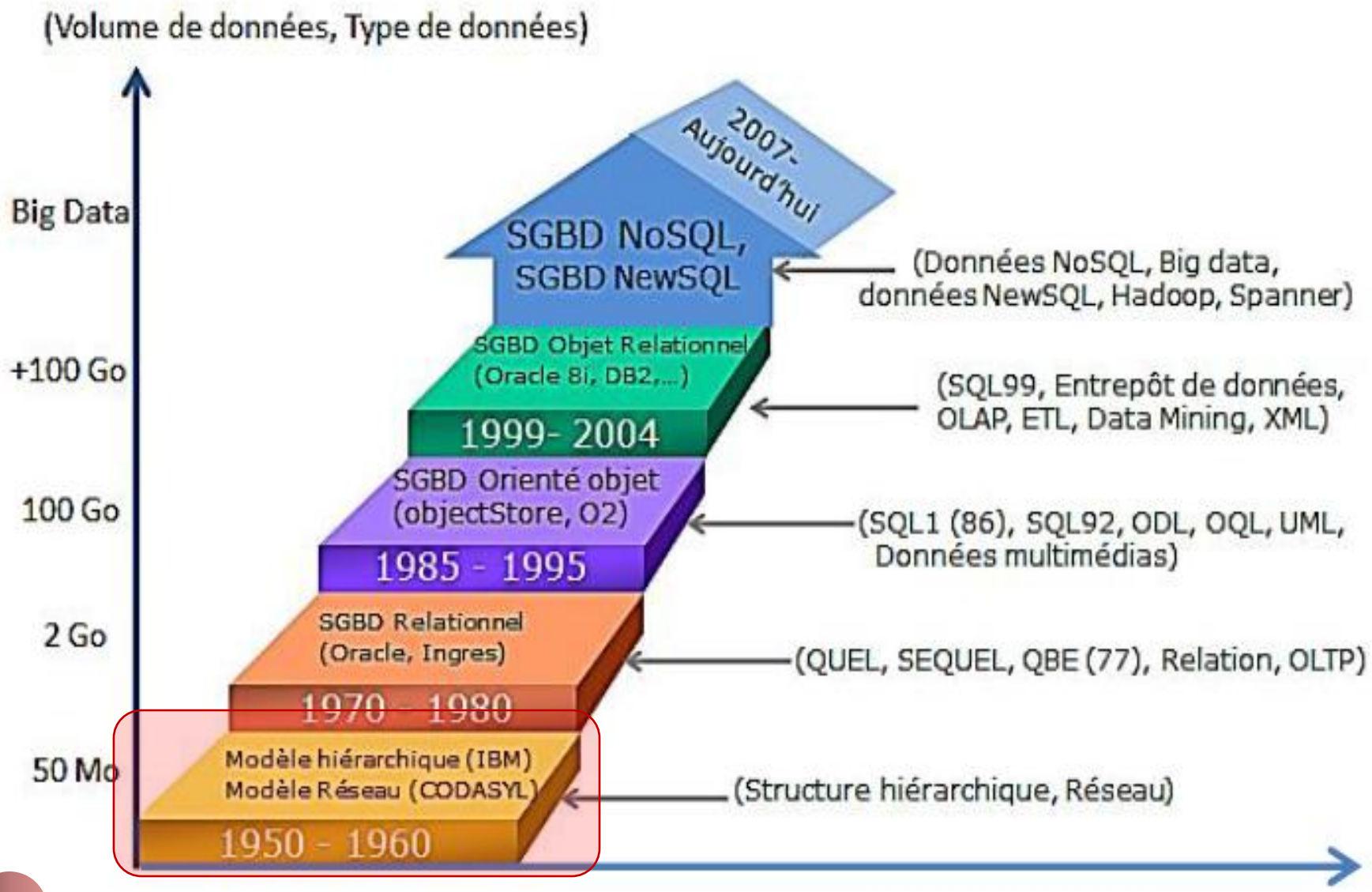
Introduction: limites des SGBDR (3/3)

- ***Faiblesses du Modèle Relationnel***
- L'incompatibilité des LMD relationnels et des langages de programmation
- Le développement d'applications est très lent et les applications sont difficilement maintenables.
- Mal adapté à :
  - de nouvelles applications : aide à la décision, conception, géographie, bureautique, multimédia, …
  - de nouvelles techniques : IHM, programmation orientée objet, programmation en logique, architectures réparties, …

Introduction: Nouveau besoins

- Les nouvelles applications ont besoin de gérer des objets complexes comme : texte, graphiques, cartes, images, … Pour cela, les bases de données objets sont nées, ils regroupent:
  - les bases de données
  - et les langages de programmation orientés objets (Simula, Eiffel, Smalltalk, C++, JAVA). Ces langages permettent d'augmenter la productivité des développeurs, ils se basent sur:
    - Les objets (valeurs, méthodes)
    - L'héritage, permet de donner à une classe fille les même propriétés de la classe mère sans avoir à les redéfinir.
- Deux solutions ont été proposés afin de pallier les lacunes de modèle relationnel:
  1. Modèle Orienté Objet.
  2. Modèle Objet-Relationnel.

Introduction: Évolution des SGBD (Rappel)



Modèle Orienté Objet (1/2)

- Premier SGBD objet : 1983 (Gemstone)

- Le ***modèle Orienté objet*** est une ensemble de méthodologies et d'outils pour concevoir et réaliser des logiciels structurés et réutilisables, par composition d'éléments indépendants.

- Les ***SGBDOO*** possèdent certaines caractéristiques propres au paradigme objet qui les distinguent des autres applications.

- Ces concepts sont : **l'identité d'objet** appelé OID (Object Identifier), **l'héritage,** l'**encapsulation**, les **classes** et **types**, l'**agrégation**, le **polymorphisme** et l'**extensibilité**.

Modèle Orienté Objet (2/2)

- **Objectif** : productivité des programmeurs
  - Moyen : réutilisation
- **Concepts essentiels:**
  - Objet encapsulé
    - interface visible : opérations (méthodes)
    - implémentation cachée : structure et code
  - Héritage
- **Langages de programmation OO :**
  - C++, Java, Smalltalk, …
- **Produits**
  - O2, ObjectStore, Ontos, Objectivity, Jasmine, Versant

Règles d'or d'un SGBD Objet (1/4)

- Pour être qualifié d'objet, un SGBD doit respecter treize règles:

1. ***Données persistantes***
   - Les données persistantes et non persistantes (temporaires) sont manipulées de la même façon par un programme
2. ***Grande quantité de données***
   - Techniques de regroupement physique, d'indexation, d'optimisation de requêtes et de gestion de cache
3. ***Fiabilité des données***
   - Transactions et privilèges
4. ***Partages de données***
   - multi-utilisateur
   - mécanismes de verrous

Règles d'or d'un SGBD Objet (2/4)

5. ***Facilité d'interrogation***
   - Langage de requêtes.
   - Le résultat d'une requête n'est pas forcément un objet d'une classe existante.
   - Le langage offre en outre la puissance d'un langage de programmation
6. ***Encapsulation***
   - Les données sont accessibles par des méthodes à différents degrés (visibilité)
7. ***Objets composites***
   - Les structures de données à définir et à manipuler peuvent être complexes

Règles d'or d'un SGBD Objet (3/4)

8. ***Identificateur d'objet***
   - Identifier d'une manière unique dans la base de données n'importe quel objet de cette base
   - Accès aux objets directement ou par des liens inter-objets via les OID
   - Objets partagés
9. ***Classes, types et méthodes***
   - Types abstraits de données
   - Concepts de classes et de méthodes de l'approche objet
10. ***Héritage***
   - Permet la réutilisation des types de données, car il permet la définition d'objets génériques (tables, types de données abstraits) pouvant être spécialisés suivant les utilisations.

Règles d'or d'un SGBD Objet (4/4)

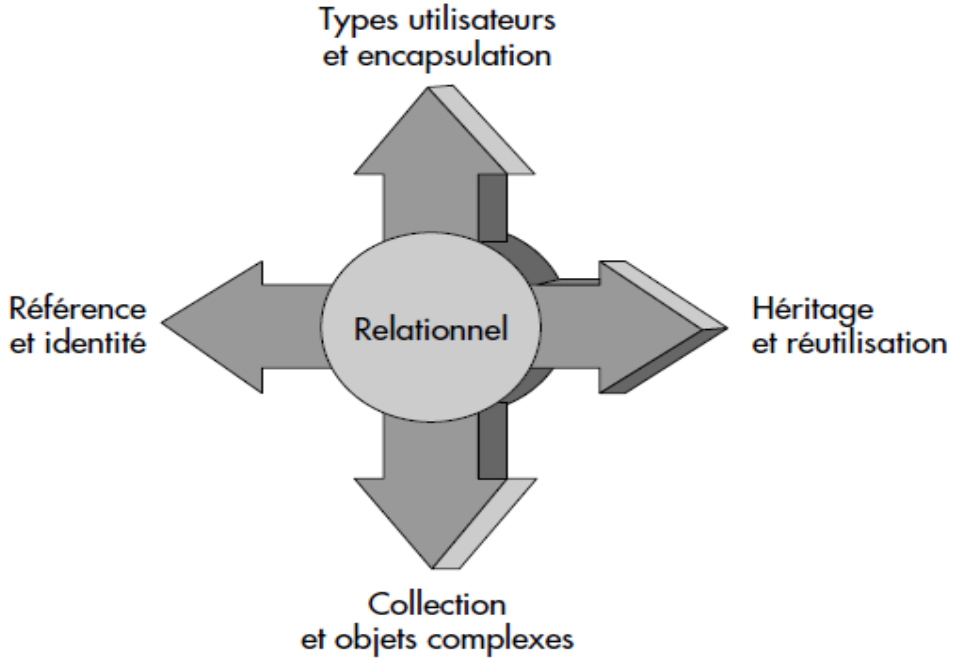
11. ***Surcharge et liaison dynamique***
    - Les méthodes peuvent être surchargées. La liaison dynamique est la capacité d'établir la correspondance entre le nom d'une méthode et son implantation lors de l'exécution et non lors de la compilation
12. ***Langage de programmation complet***
    - Le système dispose d'un langage de programmation qui ne nécessite pas d'opérateur externe pour écrire une application
13. ***Extensibilité***
    - Le système autorise l'ajout dynamique de nouveaux types abstraits de donnée, de nouvelles classes, de nouvelles méthodes, etc.

SGBDOO ou SGBDRO?

- les ***SGBDOO*** sont plus "propres" du point de vue objet et les mieux adaptés pour traiter les objets mais ils sont complètement absents du monde professionnel.
- les ***SGBDRO*** sont basés sur des SGBD robustes et éprouvés répandus dans le monde professionnel mais qui ne sont pas prévus pour gérer l'objet.
- Jusqu'à présent, les améliorations qui ont été développées pour les SGBD se sont fondus dans les SGBD existants. Dans le monde professionnel, les concepteurs et les utilisateurs de bases de données ne sont pas prêts à remettre en cause leurs savoirs et à redévelopper toutes leurs applications sur de nouveaux systèmes.
- On va privilégier les **SGBDRO**.

Modèle Objet-Relationnel

- Le modèle ***objet-relationnel*** est fondé sur l'idée d'extension du modèle relationnel avec les concepts essentiels de l'objet.



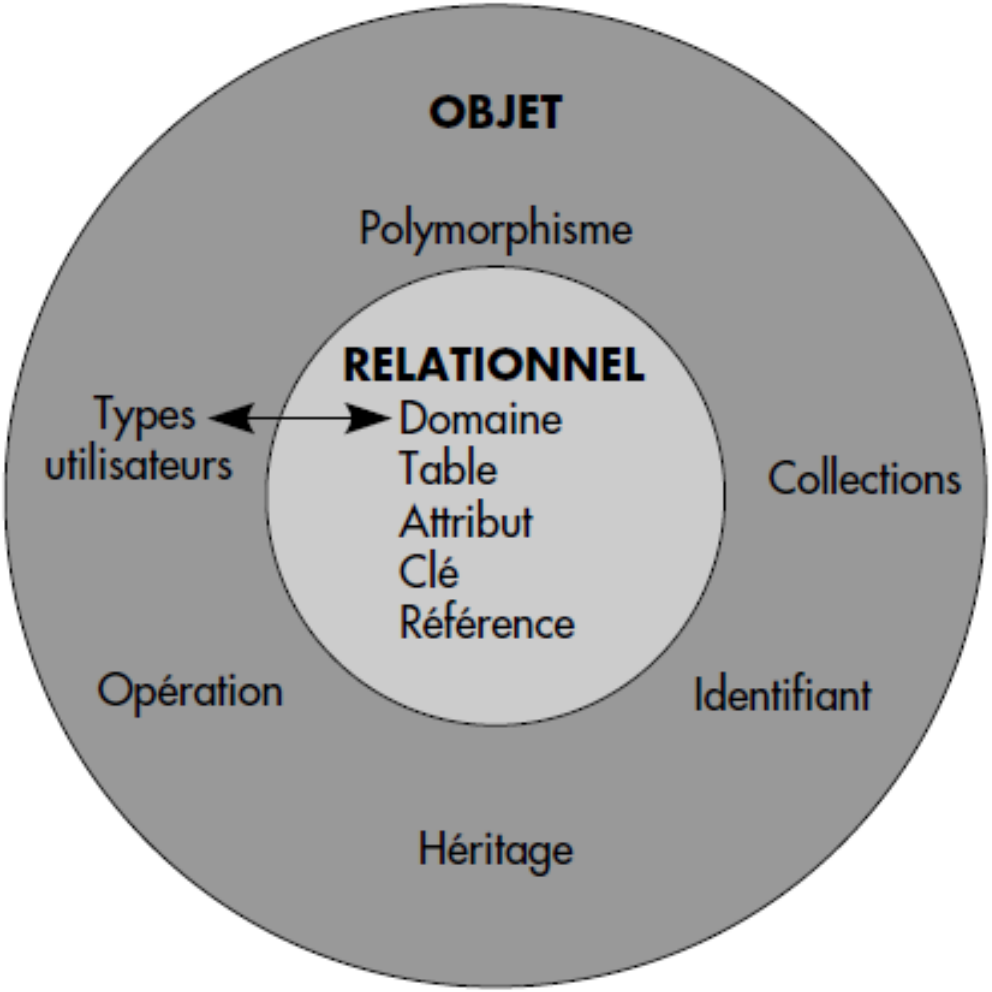
- Le cœur du modèle reste donc conforme au relationnel, mais l'on ajoute les concepts clés de l'objet sous une forme particulière pour faciliter l'intégration des deux modèles.

Modèle Objet-Relationnel

- Le modèle ***objet-relationnel*** conserve :
  - Les notions de base du ***relationnel*** :domaine, table, attribut, clé et contrainte référentielle
  - Les concepts de l'***objet***: de type utilisateur avec structure éventuellement privée et opérations encapsulant, de collections supportant des attributs multivalués (structure, liste, tableau, ensemble, etc.), d'héritage sur relations et types, et d'identifiants d'objets permettant les références inter-tables. Les domaines peuvent dorénavant être des types utilisateurs.

Modèle Objet-Relationnel

- ***Les concepts de l'objet-relationnel :***



Modèle Objet-Relationnel

- Deux vues sont possibles :
- la vue **relationnelle** dans laquelle les relations sont des relations entre objets chaque colonne représente un objet ;
- la vue **objet** où une ligne d'une table peut correspondre à un objet simple (un tuple) ou complexe (un tuple avec des attributs complexes).
- ***Exemple***:

Objet conducteur

Objet police

Police	Nom	Adresse	Conducteurs		Accidents		
			Conducteur	Âge	Accident	Rapport	Photo
24	Paul	Paris	Paul	45	134		
			Robert	17	219		
					037		

Objet Police

- Les relations dans les BDOR ne suivent pas la 1ère forme normale

Extension du langage SQL

- Au-delà du modèle OR, il est nécessaire d'étendre le langage de manipulation de données pour supporter les nouvelles avancées du langage de définition.
- **SQL 3** (SQL 99) étend le SQL aux concepts Objet.
- ***Principe***
  - Typage fort:

    Type = Données + Méthodes
  - La création de type ne crée pas d'objets
  - Les objets d'un type sont persistants que lors ils sont insérés dans des tables déclarées (CREATE TABLE)

Le support des objets

Les Fonctionnalités Objet De SQL3 :

- Définition De Types Abstraits,
- Support D'objets Complexes,
- Héritage De Type Et De Table.

Types définis par l'utilisateur (UDT)

- Le développeur peut créer ses propres types de données.
- Équivalent à une classe en OO
- Un UDT possède
  - Des attributs (simples ou autres types)
  - Des méthodes
- Utilisations d'un UDT
  - Comme colonne d'une table ordinaire,
  - Comme attribut d'un autre type,
  - Comme une ligne d'une table d'objets

Les Types Abstraits

- ✓Une instance d'un **type** peut être un ***objet*** ou une ***valeur***.
- ✓Un *objet* possède alors un ***OID*** et peut être *référencé*.
- ✓Une *valeur* peut simplement être copiée dans une ligne d'une table.
- ✓Un **type** possède en général des *colonnes*.
- ✓Les opérateurs arithmétiques de SQL (+, *, –, /) peuvent être redéfinis au niveau d'un type. Par exemple, on redéfinira l'addition pour les nombres complexes.
- ✓Enfin, un type peut posséder des clauses de comparaison (=, <) permettant d'ordonner les valeurs et peut être représenté sous la forme d'un autre type (une chaîne de caractères par exemple).

Le Type objet

✓Un **type objet** est un type de données *composé* défini par l'utilisateur, qui encapsule une structure de données ainsi que les fonctions et procédures nécessaires à la manipulation des données.

✓Les variables qui forment la structure de données sont appelées des ***attributs*** et les fonctions/procédures qui caractérisent le comportement du **type objet** sont les ***méthodes***.

✓Un objet a donc des attributs et un comportement.

Les Types distincts

✓Ces types permettent de mieux différencier les domaines des colonnes; ils sont formés à partir des types de base:

```
CREATE TYPE codePays as char(2);

CREATE TYPE matricule as integer;
```

✓Par exemple, pour différencier le domaine des colonnes **matricule** et **numDepat.**

✓Ces types s'utilisent exactement avec les mêmes instructions que le type de base sousjacent.

Les Types structurés

- ✓Correspondent aux classes des langages objets.
- ✓Ils peuvent contenir des constructeurs, attributs(≈ variables d'instances), fonctions et des procédures (≈ méthodes)
- ✓Les membres peuvent être **public, protected** ou **private**.
- ✓Les foctions et les procédures peuvent être écrites en SQL ou en un autre langage.
- ✓Supportent l'héritage.

Création d'un type de données

✓La syntaxe est semblable à celle de la création d'une table:

```
CREATE TYPE departement_type AS OBJECT
(numDept integer, nomD varchar(30),
lieu varchar(30));
```

✓Un type ne peut contenir des contrainte d'intégrité.

✓La commande « CREATE OR REPLACE TYPE » permet de redéfinir in type s'il existe déjà.

Les fonctions dans les types

```
CREATE TYPE departement_type AS OBJECT
(numDept integer,
nomD varchar(30),
lieu varchar(30),
MEMBER FUNCTION getLieu RETURN varchar);
```

```
CREATE TYPE BODY departement_type AS
MEMBER FUNCTION getLieu RETURN varchar IS
begin
return lieu;
end;
end;
```

l'héritage

- ✓ Les types supportent l'héritage multiple avec le mot-clé **UNDER**

```
create type employe_type AS OBJECT
(matr integer, nom varchar(30),
sal numeric(8,2))
not final;
```

```
create type commercial_type
under employe_type
(comm numeric(8,2))
not final;
```

- ✓ Un type est **final** par défaut

La modification d'un type: Ajout d'un attribut

✓ L'ajout d'un attribut dans un type

```
alter type employe_type
add attribute date_naissance date
cascade;
```

Propage aux tables déjà construites à partir du type

La modification d'un type: Ajout d'une fonction

- ✓ L'ajout d'une méthode dans un type

```
alter type employe_type

add member

function age return integer

cascade;
```

- ✓ Supprimer un type

```
drop type employe_type;
```

Type de ligne

✓SQL99 possède aussi la notion de type de ligne qui correspond aux structures du C: c'est un ensemble non encapsulé d'attributs.

✓Type de ligne non nommé

```
create table Employe
(nomE varchar(35),
adresse ROW(numero integer,
rue varchar(30),…))
```

✓Type de ligne nommé

```
CREATE ROW TYPE adresse_t
(numero integer,
rue varchar(30),…)
```

✓On peut ensuit utiliser ce type pour une déclaration d'attribut ou même pour créer une table à partir de ce type (comme pour les autres types

Vues dictionnaire de données

- ✓Sous Oracle :
- ✓USER_TYPES pour les types(et les collections)
- ✓USER_TYPE_ATTRS pour les attributs des types
- ✓USER_TYPES_METHODS pour les méthodes des types
- ✓USER_OBJECT_TABLES pour les tables objet-relationnelles
- ✓Sous SQLPLUS d'oracle la commande *describe* permet d'afficher la structure d'un type:

```
Describe departement_type
```

Création d'une table à partir d'un type

- ✓Les données d'un type ne sont pas persistantes que si elle sont rangées dans une table.
- ✓On peut créer des tables comme en SQL92 .
- ✓On peut aussi créer des tables à partir d'un type de données.

Création d'une table à partir d'un type

✓Soit le type employe_type:

```
CREATE TYPE employe_type AS OBJECT
(matricule integer,
nom varchar(30),
. . .
dept integer);
```

✓On peut créer une table à partir de ce type et indiquer des contraintes d'intégrité:

```
create table employe OF employe_type
(primary key (matricule));
```

Héritage d'une table

- ✓ Une table peut hériter d'une ou plusieurs tables.
  - ✓ Pas supporté par Oracle10g et 11g.
- ✓ Création de table à partir d'un type dérivé:

```
create table commerciaux of commercial_type
(constraint pk_com primary key(matr));
```

Les caractiréstiques d'une table objet-relationnelle

✓Une table est une table objet relationnelle si elle a été construite à partir d'un type (create table…of…)

✓Les lignes de ces tables sont considérées comme des objets avec un identifiant(OID, Object Identifier)

✓On peut utiliser des références pour désigner les lignes de ces tables (pas possibles pour les autres tables)

✓Vues du dictionnaire des données:

✓USER_OBJECT_TABLES pour les tables objet-relationnelles.

Insertion de données

✓On ajout des données comme avec une table relationnelle normale:

```
Insert into commerciaux
(matr, nom, sal, comm)
values (234, 'TITI ', 3200,600);
```

Insertion avec constructeur

✓On peut aussi utiliser le « constructeur du type » avec lequel la table a été construite:

```
Insert into employe values(
employe_type(125, 'Dupond', …))
```

✓Si le type est un type utilisé par un autre type, l'utilisation du constructeur du type est obligatoire:

```
Insert into employe (matr, nom, sal, adresse)
Values (1,'Toto', 12000, adresse_type(12,'Victor Hugo ', 'Nice'))
```

Afficher les données d'une table relationnelle

```
Select nom, e.adresse.rue
from employe e
```

Modifications

✓Utiliser la notion pointée comme en SQL92 mais avec un alias si un type est concerné:

```
Update employe e
Set salaire =12000, e.adresse.numero=23
Where nom= 'Dupond';
```

✓SQL99 fournit aussi la notation « .. » pour déisgner un attribut d'une colonne de type structuré:

```
Update employe
Set employe.adresse..numero=12
Where nom= 'Dupond';
```

Appel de procédure ou fonction

```
select e.nom, age(e)
from employe e
where age(e) < 40
```

Le « this » est passé en paramètre

- Sous Oracle:

```
select e.nom, e.age()
from employe e
where e..age() < 40
```

Références

- On peut indiquer dans la définition d'un type qu'un attribut contient des références (et non des valeurs) à des données d'un autre type ; la syntaxe est « REF nom-du-type » :

```
create type employe_type as object
(matricule integer,
nom varchar(30),
. . .
dept REF dept_type);
```

Exemple de select avec référence

- La notation pointée permet de récupérer les attributs d'un type dont on a un pointeur
- Lieu de travail des employes (avec Oracle) :

```
select nom, e.dept.lieu
from employe e
```

- En SQL99 :

```
select nom, e.dept->lieu
from employe e
```

- Attention, l'alias ***e*** est indispensable

Insertion avec références

```
insert into employe values (
1230, 'Durand', …, NULL);
```

Pointeur NULL

```
insert into employe(matricule, nom, dept)
select 1240, 'Dupond', REF(d)
from dept d
where dept.numDept = 10;
```

référence
vers le dept
de numéro 10

Modification avec références

```
update employe
set dept =
(select REF(d)
from dept d
where numDept = 10)
where matricule = 7500;
```

Attention, cette instruction peut très bien mettre la valeur **NULL** dans la colonne **dept** car le select renvoie **NULL** si le département de numéro 10 n'existe pas !

Contrainte NOT NULL

- Pour éviter le problème de l'exemple précédent, il faut ajouter la contrainte NOT NULL sur la colonne dept :

```
dept REF dept_type NOT NULL
```

Les types de collections

- Pour représenter une colonne multivaluée, on peut utiliser les collections ou les tableaux :
  - tableaux de taille fixe (array)
  - ensembles, au sens mathématiques ; pas de doublons (set)
  - sacs, avec des doublons (bag ou multiset)
  - listes, ordonnées et indexées par un entier (list)
- D'autres types de collections peuvent être ajoutées par les SGBD

Exemple de collection

✓ Exemple :

```
create type employe_type
(matricule integer,
nom varchar(30),
prenoms list(varchar(15)),
enfants SET(personne),
. . .);
```

Les collections avec Oracle 11g

- ✓ Oracle 11g n'offre que 2 types de collections:
  - ✓ Table imbriquée (NESTED TABLE) qui est une collection non ordonnée et non limitée en nombre d'éléments.
  - ✓ Tableau prédimensionné (VARRAY) qui est une collection d'élément de même type, ordonnée et limitée en taille.

Stockage des collections libres (tables impriquées)

✓ Construction d'une NESTED TABLE:

1. Création du type de la collection, avec:

   Create type *nom* as table of *type_elt*,

2. Création du type contenant la collection puis 3, ou création directement de la table, par exemple personne.

3. Création de la table de ce dernier type (s'il y a lieu).

➢ Génération automatique d'un identifient superficiel NESTED_TABLE_ID pour chaque collection

➢ Possibilité de créer un index dans la NESTED TABLE

Créer des collections libres

```
Create type Tvoiture as object (modele varchar(15),
                annee date, no integer)

Create type ens_voiture as Table of Tvoiture

Create table Personnes (nom varchar2(15),
        prenoms liste_prenom,
        date_naissance Date,
        voitures ens_voiture);
Nested table voitures Store As ToutesVoitures
```

Personnes

Nom	liste-prénoms	age	voitures
'Liera'	('Pierre','Paul')	32	Id1
'Liera'	('Zoé','Marie')	27	Id2

Nested_table_Id	Tvoiture		
Id1	'2ch'	'1975'	12
Id1	'206'	'2000'	3
Id2	'2ch'	'1975'	12
Id1	'607'	'2002'	1

index

Parcourir une collection

- ✓Pour parcourir des collections libres ou liées : voir une collection comme une table.
  - ✓Utilisation de l'operateur TABLE (liste ou 'sous_table' -> table).
  - ✓Puis utilisation d'un alias pour parcourir les tables ainsi obtenues.

Exemple de parcours de listes

- ✓Numéro des voitures des personnes:

```
Select v.NO
From personnes p, table (p.voitures) v;
```

- ✓Numéro des voitures de piona:

```
Select v.no
From personnes p, table (p.voitures) v
Where p.nom ='Piona';
```

Stockage des collections liées

✓Un VARRAY est une collection ordonnée et limitée en nombre d'éléments d'un même type

✓Construction d'un VARRAY:

1. Création du type de la collection, avec

   Create type *nom* as Varray *(nbr_max)* of *type_elt,*

2. Création du type contenant la collection, *Tpersonne* par exemple (ou création directe de la table)

3. Création de la table de ce dernier type?

Exemple de collections liées

✓ Plusieurs prénoms pour une personne:

Create type *liste_prénom* as Varray(3) of varchar(15)

- Varray (3): Varray est le mot clé pour définir une collection et 3 el le nombre maximum d'éléments da cette liste.
- Type des éléments de la collection indiqué après le mot-clé 'of'.

Create or replace type *Tpersonne* as object
(*nom* varchar(15), *Date_n* Date, *prenoms* liste_prenom);
Create table *personnes* of *Tpersonne ;*

- Personnes est une table d'objets.

Exemple de table avec collections liées

- Insert into personnes values (Tpersonne('Duran', liste_prenom('Pierre', 'Paul', 'Jacques'), '16-05-1963'));

Table personnes :

(ordre SQL : select * from personnes)

Nom	{prenoms}	Date_naissance
	Liste_prenom	
Duran	Pierre	16-05-1963
	Paul	
	Jacques	
Piona	Marie	29-02-1944
	Jeanne	

Merci de votre attention

Des Questions

K_menghour@yahoo.fr