



Module MLBDA
Master Informatique
Spécialité DAC

Cours 3 – Modèle objet-relationnel et SQL3

L'objet-relationnel

- **Relationnel** (tables, attributs, domaine, clé) + **Objet** (collections, identifiants, héritage, méthodes, types utilisateurs, polymorphisme)
- **Extension du modèle relationnel**
 - attributs multivalués : structure, liste, tableau, ensemble, ...
 - héritage sur relations et types
 - domaine type abstrait de données (structure cachée + méthodes)
 - identité d'objets
- **Extension de SQL**
 - définition des types complexes avec héritage
 - appels de méthodes en résultat et qualification
 - imbrication des appels de méthodes
 - surcharge d'opérateurs

Exemple de table et objet

NSécu	Nom	Adresse	Enfants		Voitures		
1234	Paul	Paris	Prénom	Age	Marque	Type	Photo
			Léa	7	Ferrari	F355	
			Max	5	Citroën	2CV	

Objet Personne

Les concepts

- Extensibilité des types de données
 - Définition de types abstraits
 - Possibilité de types avec ou sans OID
- Support d'objets complexes
 - Constructeurs de types (tuples, set, list, ...)
 - Utilisation de référence (OID)
- Héritage
 - Définition de sous-types
 - Définition de sous-tables

Le modèle SQL3 (ANSI99)

Extension objet du relationnel, inspirée de C++

- **type** = type abstrait de données avec fonction
- **objet** = instance de type référencée par un OID (défaut)
- **collection** = constructeur de type
 - **table, tuple, set, list, bag, array**
- **fonction**
 - associée à une base, une table ou un type
 - écrite en SQL, SQL3 PSM (Persistent Stored Module) ou langage externe (C, C++, Java, etc.)
- **sous-typage et héritage**
- **association** = contrainte d'intégrité inter-classe

Types atomiques (SQL3 Oracle)

- Types atomiques
 - `Varchar2(<longueur>)`
 - `Number(<longueur>)`
 - `Date`

Ex :

```
create type nsecu varchar2(15);  
create type taille number(3) ;  
create type datenais Date;
```

Types objet (SQL3 Oracle)

Types objet

```
Create type <nom-type> as Object (  
    (<nom-attribut> [ref] <type>, )+  
    (<declaration-methodes> ,)*  
);
```

Ex :

```
create type personne as Object (  
nom Varchar2(10),  
nss nsecu,    % type défini par l'utilisateur  
datenais Date) ;
```

Types ensemblistes (SQL3 Oracle)

Types ensemblistes :

- Table (ensemble avec doublons)
- Varray (collection ordonnée avec doublons)

```
Create type <nom-type> as  
    (Table | Varray(<longueur>)) of <type>;
```

```
create type retraités as Table of personne;  
create type centenaires as Varray (50) of  
    personne ;
```


Méthodes (déclaration)

- Fonctions ou procédures associées à un type d'objet.
- Modélisent le comportement d'un objet
- Ecrites en PL/SQL ou JAVA, et sont stockées dans la base.

```
Member function <nom-fonction>
```

```
    [ (<nom-para> in <type>, ...) ] return  
    <type-resultat>
```

```
Member procedure <nom-proc>
```

```
    [ (<nom-para> in <type>, ...) ]
```

Méthode (implémentation)

Le corps de la méthode est défini dans la classe du receveur.

```
Create type body <type-objet> as  
    <declaration-methode> is  
    <declaration var-locales>  
begin  
    <corps de la methode>  
end;
```

Exemple

```
create type personne as Object (  
    nom Varchar2(10),  
    nss nsecu,  
    datenais Date,  
    member function age return Number) ;
```

```
create type body personne as  
    member function age return Number is  
    begin  
        return sysdate - datenais;  
    end;
```

Stockage des données

Les objets sont stockés dans des relations (**table**),

- comme n-uplet,

```
Create table <nom-table> of <nom-type>;
```

- ou comme attribut d'un n-uplet.

```
Create table <nom-table> (  
    (<nom-attribut> [ref] <type>, )+  
);
```

Exemple

Stockage d'objet comme n-uplet :

```
Create table LesPersonnes of personne;
```

La table `LesPersonnes` stocke les objets de type `personne` (une instance par n-uplet).

Stockage d'objet comme attribut d'un n-uplet :

```
Create table LesFamilles (  
    nom Varchar2(10),  
    pere personne,  
    mere personne);
```

Le champ `père` (`mere`) de la relation `LesFamilles` stocke un objet de type `personne` (l'objet est stocké comme attribut d'un n-uplet).

Stockage des collections

On définit une table imbriquée (**nested table**) pour les attributs ensemblistes :

```
Create table <nom-table> of <nom-type>  
  nested table <nom-attribut> store as <nom-table-  
  imbriquée>;
```

Remarques :

- On déclare des 'nested table' pour le type ensembliste table uniquement;
- Pas de 'nested table' pour le type ensembliste Varray.
- On déclare une 'nested table' pour chaque champ ensembliste de la table.

Exemple

```
Create type ens-enfants as table of personne;
```

```
Create type personne as Object (  
    nom varchar2(10),  
    nss nsecu,  
    datenais Date,  
    enfants ens-enfants) ; % collection
```

```
Create table LesPersonnes of personne  
Nested table enfants store as les-enfants;
```

Références (type REF)

- Le type **REF** est un pointeur logique sur un objet. Il permet de référencer un objet par son OID (permet le partage d'objets). L'objet peut être consulté, modifié.

```
Create type couple as object (  
    nom varchar2(10),  
    nss nsecu,  
    conjoint ref personne );
```

Le champ **conjoint** fait référence à un objet de type **personne**.

Utilisation des REF

- Les associations entre deux types se modélisent à l'aide d'attributs. On utilise le type **REF** pour faire référence à un objet.
- Association 1-1 :

```
Create type personne as object  
  (nom Varchar2(10),  
   conjoint REF personne,...) ;
```

Une personne a un conjoint, qui est une personne (qui a un conjoint)

Association 1: N

- Le type de l'attribut doit être une collection (table ou varray)

```
Create type personne;
```

```
Create type ens-enfant as table of ref  
personne;
```

```
Create type personne as object  
(nom varchar(10),  
père ref personne,  
enfants ens-enfant, ...);
```

Association N:M

- Le type des attributs doit être une collection

```
Create type Voiture;
```

```
Create type ens-voitures as table of ref  
Voiture;
```

```
Create type Personne as object  
(conduit ens-voitures... );
```

```
Create type ens-pers as table of ref  
personne;
```

```
Create type Voiture as object  
(conduite-par ens-pers, ...);
```

Obtenir une REF à un objet

```
Create table LesPersonnes  of type personne;
```

```
Declare personne_ref ref personne;
```

```
Begin
```

```
Select ref(p) into personne_ref
```

```
From LesPersonnes p
```

```
Where p.nom = 'martin';
```

```
... utilisation de personne_ref ...
```

```
End;
```

La requête doit renvoyer exactement un objet (un tuple)
de la table LesPersonnes.

Fonction Value

- Pour obtenir la valeur d'un objet à partir de sa référence, on utilise la fonction value, qui prend la référence d'un objet et renvoie sa valeur.

```
Select p.nom, p.conjoint from  
LesPersonnes p
```

renvoie le nom et l'oid du conjoint

```
Select p.nom, value(p.conjoint) from  
LesPersonnes p
```

renvoie pour chaque personne p le nom et la valeur de son conjoint

Fonction Deref

- Il n'est pas possible de naviguer via les références dans les procédures PL/SQL. On utilise alors la fonction **deref**, qui prend une référence à un objet et renvoie sa valeur.

Declare

p1 Personne;

Begin

**Select deref(p.conjoint) into p1 from
dual;**

...

End;

Sous-typage et héritage

```
create type personne  
  (nom varchar2(10), adresse varchar2(30), datenais  
  date) ;
```

```
create type salarié under personne  
  (affectation varchar2(10), repos jour-ouvré);
```

```
create type étudiant under personne  
  (université varchar2(20) no-étudiant integer) ;
```

```
create table LesPersonnes of personne  
  (primary key (nom)) ;
```

```
create table LesSalariés under LesPersonnes of  
  salarié ;
```

Langage de requête

Standard SQL étendu à l'objet-relationnel :

```
SELECT [distinct] ... FROM ... [WHERE ...]
```

La clause **SELECT** peut contenir, une variable, un attribut, un nom de fonction, un chemin (notation pointée).

Les clauses **FROM** et **WHERE** peuvent contenir des requêtes imbriquées.

Exemple

```
create type adresse as object
  (num number, rue varchar2(20), ville
   varchar2(20));

create type personne as object
  (nom varchar2(10), habite adresse,
   datenais date) ;

create table LesPersonnes of personne;

select * from LesPersonnes p where
  p.nom='martin';

select p.nom from LesPersonnes p
where p.habite.ville = "paris";
```

Expression de chemin

- Un chemin permet de naviguer à travers les objets.
- Syntaxe d'une expression de chemin : $v.a_1.a_2 \dots .a_k.f$
 - Un chemin commence par une **variable**
 - Les mots intermédiaires sont des **noms d'attributs** de type objet ou REF à un objet
 - Le mot final est un **nom d'attribut** de type atomique, objet, REF ou collection
- Un chemin traverse des objets intermédiaires en suivant des associations 1-1 ou $N \rightarrow 1$ (mais pas $1 \rightarrow N$ ni $N-M$)
- Un chemin peut aboutir sur une collection d'objets

Exemple

```
create type site;    % déclarer le type
create type employé as object
    (nom varchar2(10), affectation ref site);
create type emps as table of ref employé ;
create type site as object
    (nom varchar2(10), budget number,
     chef ref employé, ens-emp emps);

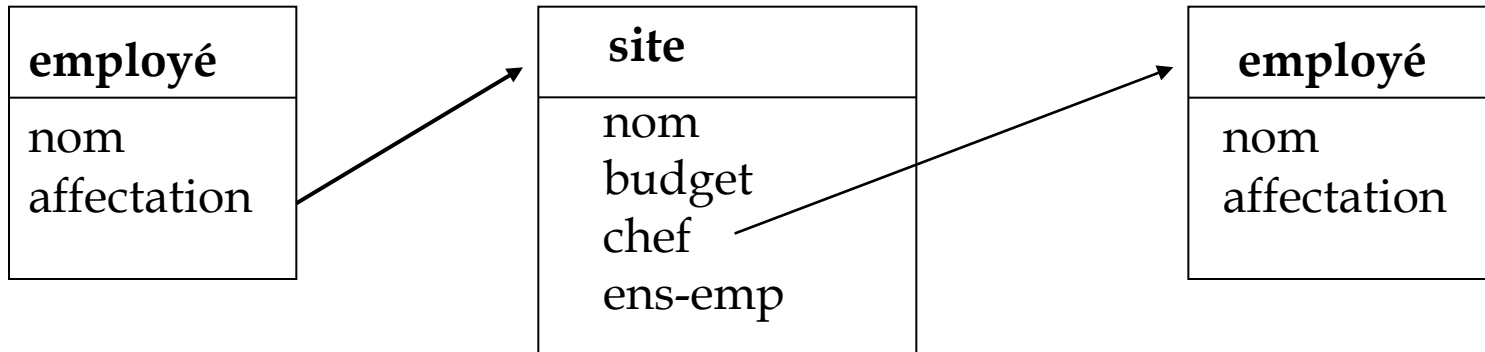
create table LesEmployés of employé ;

select nom                % noms des employés affectés à dupont
from LesEmployés e
where e.affectation.chef.nom = "dupont" ;
```

Expression de chemin

Un chemin permet de naviguer à travers les objets :

Ex. **e.affectation.chef.nom**



e est de type **employé**

e.affectation est de type **site**

e.affectation.chef est de type **employé**

e.affectation.chef.nom est de type **varchar2**

Remarque : la notation pointée ne peut être utilisée que pour les associations 1-1
(pas de collection)

Appels de méthode

- Type avec méthode

```
create type personne as Object (  
    nom varchar2(10),  
    nss nsecu,  
    datenais Date,  
    member function age return Number) ;  
create table LesPersonnes of personne;
```

- Appel de méthode

```
SELECT p.age  
FROM LesPersonnes p  
WHERE p.name = 'Joe';
```

Création d'instances

Les instances sont créées avec des instructions SQL (insert ou update).

```
insert into <table> values  
    (<constructeur>( <valeur>, <valeur> ...)) ;
```

Ex :

```
create type personne as object(  
    nom varchar2(10),  
    datenais date) ;  
create table LesPersonnes of personne;
```

```
insert into LesPersonnes  
    values (personne('martin', '13-3-60')) ;
```

Création d'instances dans les collections(1)

```
create type ens-enfant as table of personne;  
create type classe as Object (  
    niveau varchar2(10),  
    responsable varchar(20),  
    enfant ens-enfant) ;  
create table LesClasses of classe  
    Nested table enfant store as les-enfants;  
  
Insert into LesClasses values (  
    classe('CM1', 'Martin',  
    ens-enfant(personne('Max', '5 mai 2000'))));
```

Création d'instances dans les collections(2)

```
create type musiciens as varray(10)  
  of personne;
```

```
create table stage (  
  lieu varchar(10),  
  date date,  
  participants musiciens);
```

```
insert into stage values ('sarlat','20  
mars 15', musiciens  
(personne('zaza','12 juin 07'),  
personne('lulu' '05 janvier 07')));
```


Collections multiniveaux

- Types collection dont les éléments sont eux-mêmes des collections :
 - Nested table of nested table
 - Nested table of varray
 - Varray of nested table
 - Varray of varray
 - Nested table (ou varray) d'un type défini (par l'utilisateur), qui possède un attribut collection (varray ou nested table)

Interroger des collections

```
create type ens-enfant as table of personne;  
create type classe as Object (  
    niveau varchar2(10),  
    responsable varchar(20),  
    enfant ens-enfant) ;  
create table LesClasses of classe  
    Nested table enfant store as les-enfants;
```

Interroger une collection dans la clause SELECT imbrique les éléments de la collection dans le n-uplet résultat :

```
Select c.enfant from LesClasses c;
```

Renvoie la collection des membres sous la forme imbriquée :

```
enfant (nom, datnais)
```

```
-----
```

```
Ens-enfant (personne(zaza,12-06-01),personne(lulu,05-01-01))
```

```
Ens-enfant (personne(zoe,12-12-03),personne(léa,13-01-04))
```

```
...
```

Navigation dans les collections

Pour naviguer dans des collections, il faut désimbriquer (ou aplatis) la collection. L'expression **TABLE** permet d'interroger une collection dans la clause **FROM** comme une table.

```
SELECT e.*  
      FROM LesClasses c, TABLE(c.enfant) e;
```

Renvoie la collection des membres, sous forme désimbriquée :

Nom	date
Zaza	12-06-07
Lulu	05-01-07
Zoé	12-12-09
Léa	13-01-10

Expression Table

L'expression table peut contenir une sous-requête d'une collection.

```
Select *  
From table (select c.enfant  
             from LesClasses c where niveau='CM1')
```

Renvoie la collection des enfants du CM1

Remarques :

la sous-requête doit renvoyer un type collection

la clause select de la sous-requête doit contenir un seul attribut

la sous-requête doit renvoyer une seule collection

Mises à jour d'éléments d'une collection

- Pour mettre à jour des éléments d'une collection de type **table of**, il faut utiliser l'expression **TABLE** dans l'instruction du DML (**insert**, **update**, **delete**);
 - Insertion de nouveaux éléments dans une collection
 - Suppression d'un élément
 - Mise à jour d'un élément
- Oracle ne permet pas ce type de modification sur les colonnes de type **VARRAY**. Seules les modifications atomiques sont autorisées.

Exemples

Insertion d'un nouvel élève dans le niveau CM1 :

```
INSERT INTO TABLE (SELECT c.enfant
                      FROM LesClasses c
                      WHERE niveau= 'CM1')
VALUES ('martin','21 avril 07');
```

Suppression d'un élève du niveau CM1 :

```
DELETE FROM TABLE (SELECT c.enfant
                      FROM LesClasses c
                      WHERE niveau= 'CM1')e
WHERE e.nom = 'martin';
```

Interrogation de collections imbriquées

```
create type ville as object (  
    nom varchar(20),  
    population number);  
create type villes as table of ville;  
create type region as object (  
    nom varchar(20),  
    agglomérations villes);  
create type regions as table of region;  
create table pays (nom varchar(15), reg regions, ...);
```

La relation **pays** contient une table imbriquée de régions, chaque région ayant une table imbriquée de villes.

La requête suivante donne le nom de toutes les villes :

```
Select v.nom  
From pays p, table(p.reg) r,  
         table(r.agglomérations) v;
```

Mise à jour des collections multiniveaux

Les modifications dans les collections multiniveaux peuvent être faites de façon atomique, sur la collection en entier, ou sur des éléments sélectionnés.

```
INSERT INTO pays
  VALUES ('France',
    regions (region('Auvergne',
      villes (ville('Clermont',63),
        ville('Moulins',03))),
      region('Rhône-Alpes',
        villes(ville('Chambéry',73),
          ville('Lyon',69))
        )
    ));
```


Insertion dans une table imbriquée d'une table imbriquée

Ajouter une ville à une région.

On sélectionne la table imbriquée au niveau interne à l'aide d'une sous-requête dans l'expression **TABLE**.

```
INSERT INTO TABLE (SELECT r.villes
                     FROM TABLE (SELECT p.reg
                                   FROM pays p WHERE p.nom = 'France') r
                     WHERE r.nom = 'Rhône-Alpes')
VALUES ('Annecy', 74)
```

```
INSERT INTO TABLE ( SELECT r.villes
                     FROM pays p, table(p.reg) r
                     WHERE p.nom= 'France' and r.nom = 'Rhône-Alpes')
VALUES ('Annecy', 74)
```

Mise à jour 'atomique' d'une collection multiniveau

Soit **v_regions** une variable de type **regions**

```
UPDATE pays p  
  SET p.regions = :v_regions  
 WHERE p.nom = 'France';
```

Met à jour les régions de France avec le contenu
de la variable **v_regions**.

Conclusion

SQL3, standard en évolution, proposé par tous les grands constructeurs (Oracle, Sybase, IBM, etc.)

De nombreuses extensions :

- gestion de données temporelles et spatiales
- data mining
- données multidimensionnelles et requêtes décisionnelles
- ...

Compatibilité avec le relationnel.

ODMG et SQL3 : deux propositions complémentaires