M1 Informatique -Sorbonne Université

MU4IN801 - **MLBDA**

Modèles et Langages Bases de Données Avancées

Bernd Amann

COURS 3 – SQL-99

SQL-99 : Interrogation

Requêtes SQL-99

Standard SQL-92 étendu à l'objet-relationnel :

SELECT [distinct] … **FROM** … [WHERE …]

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

À partir de SQL-99 : Les clauses **FROM** et **WHERE** peuvent contenir des requêtes imbriquées.

Exemple : schéma et requêtes

SCHÉMA

```
create type Adresse as object (
        num number,
        rue varchar2(20),
        ville varchar2(20));
/
create type Personne as object (
        nom varchar2(10),
        habite Adresse,
        datenaiss date) ;
/
create table LesPersonnes of Personne;
/
```

REQUÊTES

```
select p.nom, p.habite, p.datenaiss
from LesPersonnes p
where p.nom = 'martin';


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

Expression de chemin

Expression de chemin

Une expression de chemin permet de naviguer à travers les objets.

Syntaxe d'une expression de chemin : **v.a1.a2. … .ak.f**

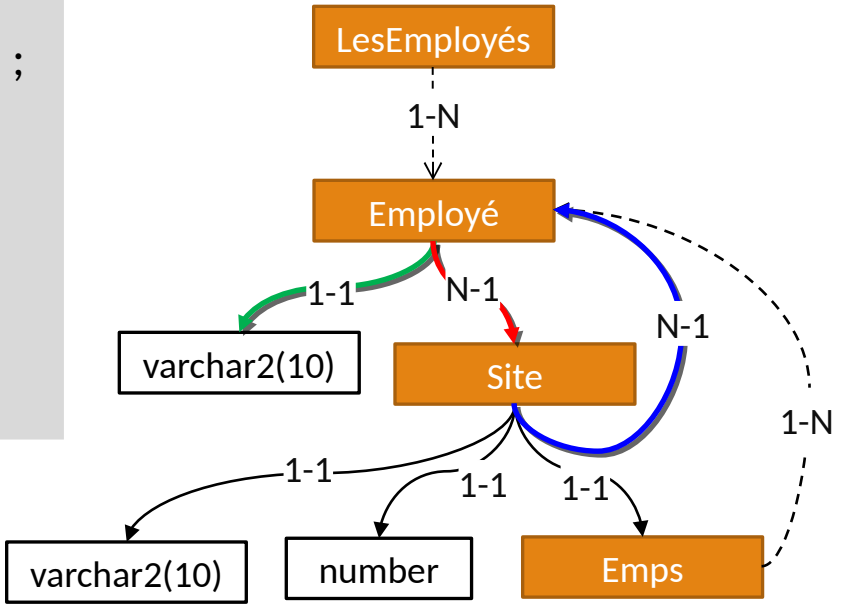
- Un chemin commence par une variable **v** de type objet ou référence à un objet (ref)
- Les termes intermédiaires **ai** sont des noms d'attributs de type objet ou référence à un objet (ref)
- Le mot final **f** est un nom d'attribut de type atomique, objet, référence ou collection.

Un chemin traverse des objets intermédiaires en suivant des associations 1-1 ou N-1 (objet, ref), mais pas 1-N ni N-M (collections).

Exemple

```
create type Site;
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 ;
```

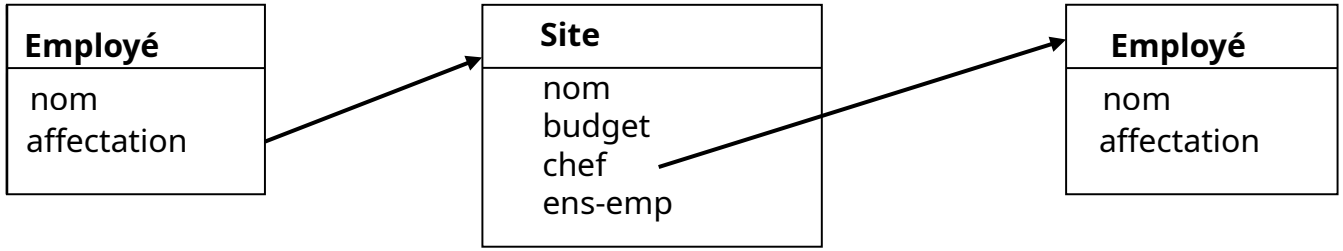
```
select nom
from LesEmployés e
where e.affectation.chef.nom = 'dupont' ;
```



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 REF Site

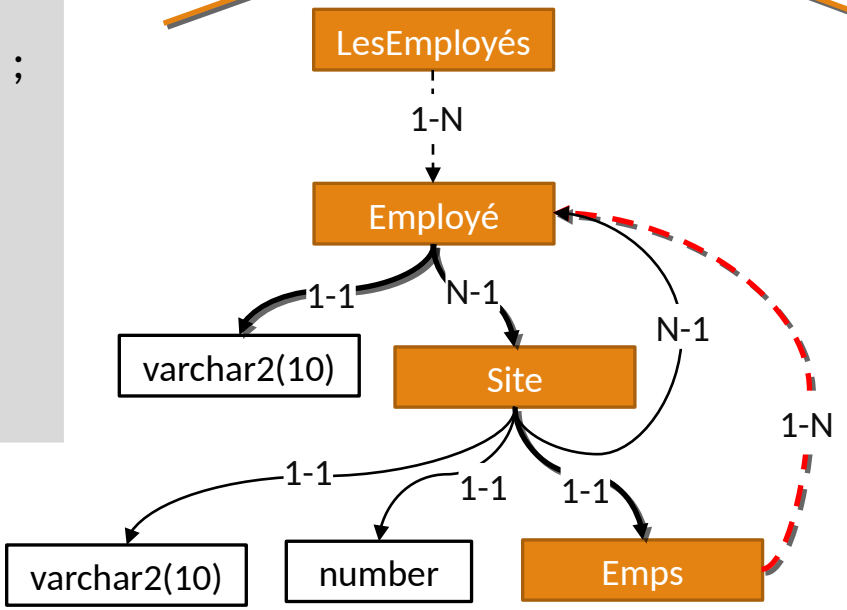
e.affectation.chef est de type REF Employé

e.affectation.chef.nom est de type varchar2

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
from LesEmployés e
where e.affectation.ens_emp.nom = 'dupont' ;
```



Obtenir la référence d'un objet o: ref(o)

Procédure PLSQL/Oracle

```
create table LesCours  of Cours;
/
create or replace procedure test as
   cours_ref ref Cours;
begin
  select ref(c) into cours_ref
  from LesCours c
  where c.titre = 'MLBDA';
  … utilisation de cours_ref dans la procédure …
end;
/
```

L'objet déréférencé doit exister dans une table d'objets (ici LesCours)

Obtenir l'objet d'une référence r: deref(r)

La fonction **deref** prend une expression de chemin dont le type est une référence à un objet et retourne un type objet.

```
create type Personne as object (
        nom varchar2 (10),
        conjoint ref Personne);
create table LesPersonnes of Personne;
```

Le résultat est de type **table of ref** Personne :

```
select p.conjoint
from LesPersonnes p;
```

Le résultat est de type **table of** Personne :

```
select deref(p.conjoint) from
LesPersonnes p;
```

Exemple

Schéma Ecole

```
create type ensEnfant as table of Personne;
create type Classe as object (
        niveau varchar2(10),
        responsable varchar(20),
        enfants ensEnfant) ;
create table LesClasses of Classe
        nested table enfants store as t1;
```

Interroger des collections imbriquées

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

```
select c.enfants
from LesClasses c;
```

Renvoie la *collection des enfants* (EnsEnfant) sous la forme imbriquée :

enfants
EnsEnfant(personne(zaza,12-06-10),personne(lulu,05-01-10))
EnsEnfant(personne(zoe,12-12-09),personne(léa,13-01-11))

Interroger les varray

Schéma Musique

```
create type Musiciens as varray(10) of Personne;
create type Stage (
        lieu varchar(10),
        date date,
        participants Musiciens);
create table LesStages of Stage;
```

Requête

```
select s.participants
from LesStages s
where s.lieu='Sarlat';
```

Le résultats est une table avec un attribut *participants* de type *varray(10) of Personnes*.

Navigation dans les collections (1-N) : **table**

Pour parcourir les collections (associations 1-N), il faut les *désimbriquer* (ou aplatir). L'expression **table** permet d'interroger une collection dans la clause **from** comme une table.

```
select e.nom, e.dateNaissance
from LesClasses c, table(c.enfants) e;
```

~~**select** c.enfants.nom,~~
~~c.enfants.dateNaissance~~
~~**from** LesClasses c;~~

Renvoie la collection des membres, sous forme désimbriquée :

Nom	dateNaissance
Zaza	12-06-10
Lulu	05-01-10
Zoé	12-12-09
Léa	13-01-11

Interrogation de collections de collections

Schéma Régions

```
create type Ville as object (
        nom varchar(20),
        departement number(2);
create type Villes as table of Ville;
create type Region as object (
        nom varchar(20), aggloms Villes);
create type Regions as table of Region;
create type Pays as object(
        nom varchar(15), reg Regions)
create table LesPays of pays
   nested table reg store as tabr (nested table aggloms store as tabv);
```

Nom de toutes les villes d'Auvergne:

```
select v.nom
from LesPays p, table(p.reg) r,   table(r.aggloms) v
where r.nom = 'Auvergne';
```

Expression **table**

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

```
select e.nom
from table(select c.enfants
                 from LesClasses c
                where niveau='CM1') e
```

```
create type ensEnfant as table of Personne;
create type Classe as object (
        niveau varchar2(10),
        responsable varchar(20),
        enfants ensEnfant) ;
create table LesClasses of Classe
        nested table enfants store as t1;
```

Requête : collection des noms d'enfants de CM1.

Sous-requête : une collection d'objets de type Enfant

Remarques :

- la sous-requête doit renvoyer *une seule valeur* de type collection
- la clause **select** de la sous-requête doit contenir *un seul attribut*

value et **is of type**

p.* retourne le **nuplet** qui *représente* l'objet de type Personne :

```
select p.*
from LesPersonnes p;
```

NOM	CONJOINT
marie	oracle.sql.REF@b49ebc6e
martin	

value(p) retourne **l'objet** de type Personne:

```
select value(p)
from LesPersonnes p;
```

VALUE(p)
PERSONNE('???', NULL)
PERSONNE('???', oracle.sql.REF@b49ebc6e)

is of type vérifie le type d'un objet : tous les étudiants parmi les personnes:

```
select value(p)
from LesPersonnes p
where value (p) is of type (Etudiant);
```

La fonction **value** et les références

```
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 ;
create table LesSites of Site
   nested table ens_emp store as t456;
```

```
select value(e).nom, s.nom
from LesSites s, table(s.ens_emp) e
where value(e).affectation.chef.nom = 'dupont' ;
```

Correct :

- **value(e).nom** : value(e) est de type *ref Employé*
- **deref(value(e)).nom** : deref(value(e)) est de type *Employé*
- **s.nom** : s est un *nuplet qui représente un objet* de type *Employé* (pas une référence)
- **value(s).nom** : value(s) est de type *Site*
- **ref(value(s)).nom** : ref(value(s)) est de type *ref Employé*

Incorrect :

- e.nom : e est un *nuplet qui représente une référence vers un objet de type Employé*

Les fonctions **value** et **ref** : égalité d'objets

```
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 ;
create table LesSites of Site
   nested table ens_emp store as t456;
```

Correct (égalité objet)

```
select value(e).nom
from LesSites s, table(s.ens_emp) e,
        Les Employés e2
where value(e).affectation.chef.nom = 'dupont'
   and deref(value(e))=value(e2);
```

Correct (égalité référence)

```
select value(e).nom
from LesSites s, table(s.ens_emp) e,
        Les Employés e2
where value(e).affectation.chef.nom = 'dupont'
   and value(e)=ref(e2);
```

Incorrect

```
select value(e).nom
from LesSites s, table(s.ens_emp) e,
        Les Employés e2
where value(e).affectation.chef.nom = 'dupont'
   and e=ref(e2);
```

Appels de méthode

Type avec méthode

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

Appel de méthode dans les requêtes

```
select p.age()
from LesPersonnes p
where p.nom = 'Joe';

select p.nom  from LesPersonnes p
where p.age() < 20;
```

PL/SQL et SQL-99 : **bulk collect into**

L'instruction **bulk collect into** permet d'affecter à une variable l'ensemble des objets retournés par une requête SQL.

L'instruction se place dans la clause select, juste avant la clause from de la requête.

```
create type ensNoms as table of varchar2(20);
…
resultat ensNoms;
begin
        select p.nom bulk collect into resultat
        from LesPersonnes p
        where p.adresse='Paris';
…
end;
```

Exemple : réseaux social

```
create type Personne;
create type Amis as table of ref Personne;
create type Personne as object (
        nom varchar2(30),
        cercle Amis,
        member function entourage return Amis,
        member function reseau(d number) return Amis
);
create table lesPersonnes of Personne
        nested table cercle store as tabCercle;
```

Méthode **entourage** (amis d'amis)

```
create type body Personne as
        member function entourage return Amis is
resultat Amis;
begin
        select distinct value(e) bulk collect into resultat
        from table (self.cercle) a,
                table (value(a).cercle) e
        where deref(value(e)) <> self;
return resultat;
end;
```

Explications:

- self est de type **Personne**
- e est un *nuplet* avec une référence vers Personne
- value(e) est de type **ref Personne**
- deref(value(e)) est de type **Personne**

Méthodes / requêtes récursives

```
create type body Personne as
member function reseau(d number) return Amis is
 resultat Amis;
 begin
       if (d > 1) then
          select value(a) bulk collect into resultat
          from table (cercle) a
          union
          select value(e)
          from table (self.cercle) a, table (value(a).reseau (d-1)) e
          where deref(value(e)) <> self;
       else resultat := cercle;
       end if;
       return resultat;
end;
```

Requêtes avec méthodes récursives

Entourage de Max:

```
select value(e).nom
from LesPersonnes p, table(p.entourage()) e
where p.nom= 'Max';
```

Paires de noms de personnes ayant au moins un ami en commun dans leur réseau avec une distance 3 (reseau(3)).

```
select distinct value(p1).nom, value(p2).nom
from LesPersonnes p1, table (p1.reseau(3)) r1,
        LesPersonnes p2, table (p2.reseau(3)) r2,
where  value(p1) <> value(p2)
    and value(r1) = value(r2);
```

PL/SQL : Fonction **deref**

Remarque : PL/SQL ne supporte pas les expressions de chemins avec traversée de références. Il faut « déréférencer » explicitement chaque référence dans une requête.

La fonction **deref** prend une expression de chemin dont le type est une référence à un objet et retourne un type objet.

PLSQL/Oracle

```
declare
        p1 Personne;
        p_ref REF Personne;
begin
        select deref(p_ref) into p1 from dual;
        % …  utilisation des attributs de p1…
end;
```

Requêtes de Mises-à-Jour

Mises à jour d'éléments d'une collection imbriquée

Pour mettre à jour des éléments d'une collection imbriquée, 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

VARRAY :

- Oracle ne permet pas d'insertion, de suppression et de mise-à-jour d'éléments sur les colonnes de type VARRAY
- seules les modifications atomiques (« remplaçant tout le varray ») sont autorisées.

Création d'instances

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

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

Exemple :

Insertion

```
create type Personne as object(
                nom varchar2(10),
                datenais date) ;
/
create table LesPersonnes of Personne;
/
insert into LesPersonnes values (
        Personne('martin',to_date('13-3-2020', 'DD-MM-YYYY'));
/
```

Création d'instances avec références

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

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

Insertion

```
create type Personne as object (
        nom varchar2 (10),
        conjoint ref Personne);
insert into LesPersonnes values (Personne('Martin',NULL));
insert into LesPersonnes values (
        Personne(' Marie',(select ref(m) from LesPersonnes m
                                where m.nom='Martin')));
```

Création d'instances dans les collections(1)

Schéma Ecole

```
create type ensEnfant as table of Personne;
create type Classe as object (
        niveau varchar2(10),
        responsable varchar(20),
        enfants ensEnfant) ;
create table LesClasses of Classe
        nested table enfant store as t1;
```

Insertion

```
insert into LesClasses values (
        classe('CM1',
                'Martin',
                ensEnfant(Personne('Max','5-05-2008'))));
```

Création d'instances dans les collections(2)

Schéma Musique

```
create type Musiciens as varray(10) of Personne;
create type Stage as object (
        lieu varchar(10),
        date date,
        participants Musiciens);
create table LesStages of Stage;
```

Insertion

```
insert into LesStages values (
        Stage( 'sarlat',
                to_date('30/10/2020', 'DD/MM/YYYY'),
            Musiciens (Personne('zaza', to_date( '12-06-07' 'DD/MM/YYYY')),
                    Personne('lulu', to_date('05-01-07', 'DD/MM/YYYY'))));
```

Exemple

Schéma Ecole

```
create type ensEnfant as table of Personne;
create type Classe as object (
        niveau varchar2(10),
        responsable varchar(20),
        enfants ensEnfant) ;
create table LesClasses of Classe
        nested table enfants store as t1;
```

Insérer un enfant dans la classe de CM1

```
insert into table (select c.enfants from LesClasses c where c.niveau = 'CM1')
                values (Personne('Paul', 05-01-2009));
insert into table (select c.enfants from LesClasses c where c.niveau = 'CM1')
                select value(p) from LesPersonnes p where p.nom='Marie';
```

Exemple (2)

Supprimer un enfant de la classe de CM1

Suppression

```
delete from table (select c.enfants
                   from LesClasses c
                   where c.niveau= ‘CM1’) e
where e.nom = ‘Marie’;
```

Changer la date de naissance de Léa, en classe de CP.

Mise-à-jour

```
update table (select c.enfants
              from Lesclasses c
              where c. niveau ='CM1') e
set e.nom = ‘Léa'
where e.nom=‘Marie';
```

Collections de collections

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)

Mise à jour des collections de collections

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

Schéma Régions

```
create type Ville as object (
        nom varchar(20),
        departement number(2));
create type Villes as table of Ville;
create type Region as object (
        nom varchar(20),
        aggloms Villes);
create type Regions as table of Region;
create type Pays as object(
        nom varchar(15),
        reg Regions)
create table LesPays of pays (primary key nom)
   nested table reg store as tabr (nested table
      aggloms store as tabv);
```

Insertion

```
INSERT INTO LesPays VALUES(
 Pays( 'France',
        Regions (
              Region('Auvergne',
                 Villes(Ville('Clermont',63),
                        Ville('Moulins',03))),
              Region('Rhône-Alpes',
                   Villes(Ville('Chambéry',73),
                          Ville('Lyon',69))))));
```

Insertion dans une collection de collection (1)

Ajouter une ville à une région.

Insertion de la ville d'Anneey

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

Requête équivalente :

```
INSERT INTO TABLE (SELECT r.aggloms
                     FROM LesPays p, table(p.reg) r
                    WHERE p.nom= 'France'
                      AND  r.nom = 'Rhône-Alpes')
  VALUES (Ville('Annecy', 74));
```

Insertion dans une collection de collection (2)

Schéma

```
create table LesVilles of Ville;

create type Villes as table of ref Ville;
create type Region as object (
  pays varchar(15),
  nom varchar(20),
  aggloms Villes);
```

Insertion

```
INSERT INTO TABLE (
        SELECT r.aggloms
        FROM TABLE (SELECT p.reg
                    FROM LesPays p
                    WHERE p.nom = 'France') r

        WHERE r.nom = 'Rhône-Alpes')
VALUES ((SELECT ref(v)
             FROM LesVilles v
             WHERE v.nom= ' Annecy'));
```

Mise à jour d'une collection de collection

Utilisation de update pour l'insertion d'une région de France avec le contenu de la variable v_regions.

```
declare v_regions  Regions;
begin
        v_regions := Regions( Region('PACA',
                                                Villes(Ville('Marseille',13),
                                                        Ville('Nice',06))));
        UPDATE LesPays p
        SET p.regions = v_regions
        WHERE p.nom = 'France';
end;
```

Procédure **insertion**

Insérer une personne nommée Lucie dans le cercle d'amis de Max.

```
create or replace procedure insertion as
 p1 ref(Personne);
 begin
    select ref(p) into p1 from LesPersonnes p where value(p).nom='Lucie';
    insert into table (select p.cercle  from LesPersonnes p
                          where value(p).nom='Max')
     values (p1);
 end;
```

Ou:

```
insert into table (select p.cercle from LesPersonnes p
                          where value(p).nom='Max')
values (select ref(p) from LesPersonnes p
          where value(p).nom='Lucie');
```

Conclusion

SQL-99, 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.