Module MLBDA
Master Informatique
Spécialité DAC

Cours 3 – SQL3

Rev 4/10/2017

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>, …)]
```

Méthodes (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;
```

Héritage des méthodes

```
create type personne as object
   (nom varchar2(10),
    adresse varchar2(30),
    datenais date,
    not instantiable member function statut()return
   varchar2(10))
    not final not instantiable;

create type étudiant under personne
   (université varchar2(20),
    no-étudiant number(10),
    overriding member function statut() return
   varchar2(10)) ;
```

Not instantiable : type sans instance, méthode abstraite
Overriding : spécialisation d'une méthode

Langage de requête

Standard SQL 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.

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 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

- Un chemin permet de naviguer à travers les objets.
- Syntaxe d'une expression de chemin : **v.a_1.a_2. ... .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→1 (mais pas 1→N ni N→M)
- Un chemin peut aboutir sur une collection d'objets

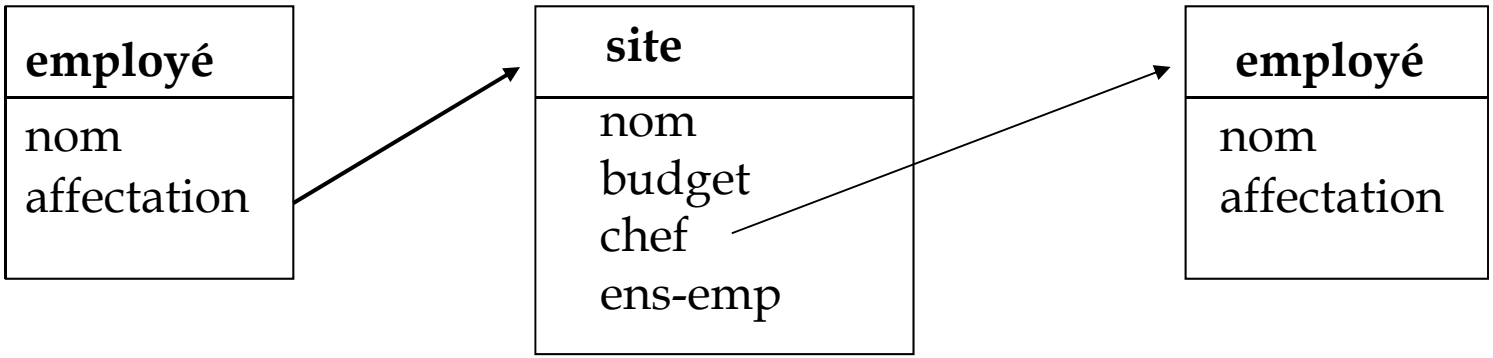
Exemple

```
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 **REF site**
e.affectation.chef est de type **REF 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),
   datenais Date,
   member function age return Number) ;
create table LesPersonnes of personne;
```

- Appel de méthode

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

SELECT p.nom  FROM LesPersonnes p
WHERE p.age < 20;
```

Obtenir la REF d'un objet

```
Create table LesCours  of type 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;
```

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

Fonction Deref

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;

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

Fonction Deref

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

```
p1 Personne;
Begin
Select deref(p.conjoint) into p1 from dual;
% …  utilisation des attributs de p1…
End;
```

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 ensEnfant as table of personne;
create type classe as Object (
  niveau varchar2(10),
  responsable varchar(20),
  enfant ensEnfant) ;
create table LesClasses of classe
  nested table enfant store as t1;

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

Création d'instances dans les collections(2)

```
create type musiciens as varray(10) of
  personne;
create type Stage (
  lieu varchar(10),
  date date,
  participants musiciens);

create table LesStages of Stage;

insert into LesStages values (
  Stage( 'sarlat', '20 mars 15',
      musiciens (personne('zaza', '12-06-07'),
                 personne('lulu', '05-01-07')
                )
        )
);
```

Interroger des collections

```
create type ensEnfant as table of personne;

create type classe as Object (
      niveau varchar2(10),
      responsable varchar(20),
      enfant ensEnfant) ;     % collection

create table LesClasses of classe
      Nested table enfant store as t1;

create table LesPersonnes of personne;

Select p.responsable from LesClasses c
where c.niveau = 'CP';
```

Interroger les varray

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


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

Interroger des collections imbriquées

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 enfants sous la forme imbriquée :

```
enfant (nom, datnais)
-----------------------------
Ens-enfant(personne(zaza,12-06-10),personne(lulu,05-
  01-10))
Ens-enfant(personne(zoe,12-12-09),personne(léa,13-
  01-11))
...
```

Navigation dans les collections

Pour parcourir les collections, 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.enfant) e;
```

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
```

Expression Table

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

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

Renvoie la collection des noms d'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 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

- Oracle ne permet pas ce type de modification sur les colonnes de type **`VARRAY`**. Seules les modifications atomiques sont autorisées.

Exemple

- Insérer un enfant dans la classe de CM1

```
Insert into table (select c.enfant from
  LesClasses c where c.niveau = 'CM1')
  values (Personne('Paul', 05-01-2009));

Insert into table (select c.enfant from
  LesClasses c where c.niveau = 'CM1')
  values (select p from LesPersonnes p where
  p.nom='Paul');
```

Exemple (2)

- Supprimer un enfant de la classe de CM1

```
Delete from table (select c.enfant
                   from LesClasses c
                   where c.niveau= ‘CM1’)e
where e.nom = ‘martin’;
```

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

```
Update table (select c.enfant from
  Lesclasses c where c. niveau =‘CP’) e
Set e.dateNaissance = '12-01-11'
where e.nom=‘Léa’’;
```

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)

Collections de collections

```
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),
   agglomérations 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
   agglomérations store as tabv);
```

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

Interrogation de collections de collections

Nom de toutes les villes d'Auvergne:

```
Select v.nom
From LesPays p,
   table(p.reg) r,
  table(r.agglomerations) v
Where r.nom = 'Auvergne';
```

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.

```
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

Ajouter une ville à une région.

```
INSERT INTO TABLE (SELECT r.agglomérations
   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.agglomérations
    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)

Si la ville est déjà stockée dans la base (par ex. dans la table LesVilles) et si on change la définition des types Villes et Région

```
create type villes as table of REF Ville;
create type region as object (
   nom varchar(20),
   agglomérations villes);


INSERT INTO TABLE (SELECT r.villes
    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 ‘atomique’ d’une collection de collection

```
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;
```

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

Fonction Value

- La fonction **`value`** prend comme paramètre une variable déclarée dans la clause **`from`** et renvoie l'instance de l'objet stocké dans la table.

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

La fonction **`value`** retourne un type identique à celui de la variable.

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';
…
```

Exemple

```
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;
```

Méthode reseau (d number)

```
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

- 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);
```

Procedure

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;
```

Insertion

Instruction équivalente :

```
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

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