Bases de données et sites WEB

Cours 3 : SQL3 Langage

Plan

- Références
- Requêtes simples
- Expressions de chemin
- Appels de fonctions et d’opérateurs
- Création d’instances
- Opérations sur les collections

Références

```
CREATE TYPE employe AS OBJECT (
   Nom VARCHAR2(10),
   Directeur REF employe );
```

L'attribut `Directeur` fait référence à un objet de type `employe`.

Le type REF est un pointeur logique sur un objet tuple. Il permet de référencer un objet par son oid. L'objet peut être consulté, modifié, indépendamment de son appartenance à l'objet employe.

Utilisation des REF

```
CREATE TABLE employes OF employe;
```

% les n-uplets de cette table sont des objets, et ont un oid par défaut.

```
INSERT INTO employes
VALUES (employe('Martin', NULL));
```

%constructeur d'objet = nom du type

```
INSERT INTO employes values
employe('Vincent', (select ref (e) from
  employes e where e.nom='Martin'));
```

% insertion de Vincent, dont le chef est Martin

Utilisation des REF

- Les références permettent de naviguer entre les objets.
- Les associations entre deux types se modélisent à l'aide d'attributs ajoutés aux types. On utilise le type REF pour faire référence à un objet.
- Association 1-1 :

```
CREATE TYPE ordi AS OBJECT
   (num NUMBER, …);
CREATE TYPE personne AS OBJECT
   (nom VARCHAR2(10),
    ordinateur REF ordi,…);
```

Une personne utilise un ordinateur, un ordinateur est utilisé par une personne.

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,
   enf ens-enfant, …);
```

Association N:M

- Le type des attributs doit être une collection

```
CREATE TYPE Voiture;  % pour que le nom soit reconnu
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, …);
```

Pour déréférencer, on utilise la fonction **`deref.`**

Obtenir une REF à un objet

```
CREATE TABLE personnes OF TYPE personne;

DECLARE personne_ref REF personne;
BEGIN
SELECT REF(p) INTO personne_ref
FROM personnes p
WHERE p.nom = ‘Martin’;

... utilisation de personne_ref ...
END;
```

La requête doit renvoyer exactement un tuple.

Langage de requête SQL3

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 adr AS OBJECT
  (num number, rue varchar2(20), ville varchar2(20));
CREATE TYPE personne AS OBJECT
  (nom varchar2(10), adresse adr, datenais date) ;
CREATE TYPE employé UNDER personne
  (affectation varchar2(10),repos jour-ouvré);
CREATE TABLE employés OF employé

SELECT * FROM employés e WHERE e.nom='Martin';

SELECT e.nom FROM employés e
WHERE e.adresse.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 mais ne peut donc pas la traverser

Exemple

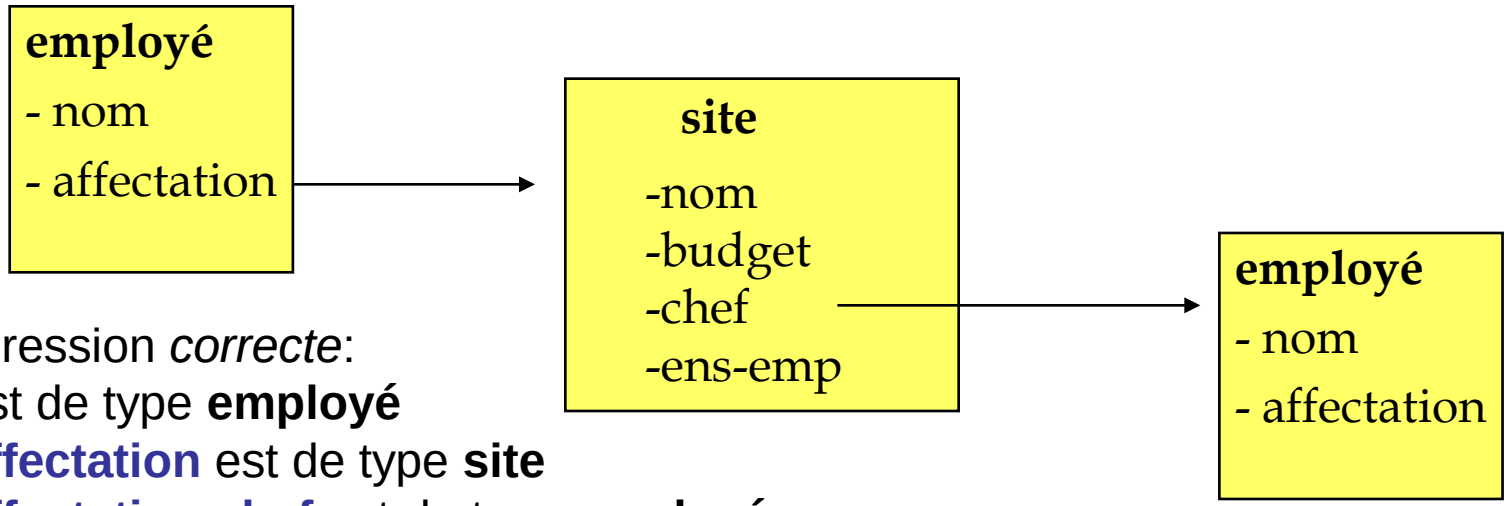
```
CREATE TYPE site;   % déclarer le type
CREATE TYPE employé AS OBJECT
   (nom varchar2 (10), affectation REF site) ;
CREATE TYPE emps AS TABLE OF REF employé ;

CREATE TYPE site AS OBJECT
   (nom varchar2(10), budget number, chef REF
   employé, ens-emp emps);

CREATE TABLE LesEmployés OF employé ;
CREATE TABLE LesSites OF site ;

SELECT nom                    % noms des employés affectés à Dupont
FROM LesEmployés e
WHERE e.affectation.chef.nom = "dupont" ;
```


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

Expression de chemin

Un chemin permet de naviguer à travers les objets :

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



Expression *correcte*:
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**

Expression *incorrecte*:
~~**e.affectation.ens-emp.nom**~~

Appels de méthode

- Type avec méthode

```
CREATE TYPE Employe AS OBJECT (
    nom varchar2(10), date_naissance …
    MEMBER FUNCTION age RETURN Number
);
CREATE TABLE emp OF Employe;
```

- Appel de méthode

```
SELECT j.nom
FROM emp j
WHERE j.age < 40 ;
```

Stockage des données

Les objets sont stockés dans des relations (table).

- une table peut posséder des attributs d'un type abstrait
- un tuple contient des références ou des valeurs complexes
- un attribut peut être de type référence (REF <type>)

Exemples :

```
CREATE TABLE LesPersonnes OF personne;
```

La table **LesPersonnes** stocke les objets de type **personne** (une instance par tuple).

```
CREATE TABLE LesVoitures (marque Varchar2(15),
type Varchar2(10), proprietaire Personne);
```

Le champ proprietaire de la relation **LesVoitures** stocke un objet de type **personne** (l'objet est stocké comme attribut d'un tuple).

Si on a **ref Personne**, alors l'objet personne est stocké dans **LesPersonnes**.

Stockage des données (2)

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-
   imbriquee>;

CREATE TYPE Children AS TABLE OF personne;
CREATE TYPE employe AS OBJECT (
   nom varchar2(10),
   nss numsecu,
   datenais Date,
   enfant Children) ;
CREATE TABLE LesEmployes OF employe
NESTED TABLE enfant store AS les-enfants;
```

Remarque : pas de nested table pour le type ensembliste varray, car taille fixe

Création d'instances

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

```
INSERT INTO <table> VALUES
      (<constructeur>( <valeur>,<valeur> …));

CREATE TYPE employe AS OBJECT(
      id number,
      nom varchar2(10),
      datenais Date) ;


CREATE TABLE LesEmployes of employe;
  INSERT INTO LesEmployes
      VALUES (employe(1,'Martin','13-3-60'));
```

Et si on rajoutait un attribut chef ref employe ?

Création d'instances dans les collections(1)

```
Create type Children as table of personne;
create type emp as Object (
  nom varchar2(10),
  nss Number,
  datenais Date,
  enfant Children) ;
Create table LesEmployes of emp
  Nested table enfant store as les-enfants;

Insert into LesEmployes values (
   emp('Martin',123456789012,'12 juin 60',
  Children(personne(1,'Max','15 mai 95'),
           personne(1,'Jim','12 mai 96'))));
```

Création d'instances dans les collections(2)

```
CREATE TYPE employe AS OBJECT (id number,nom
  varchar2(10), datenais Date);
CREATE TYPE directeurs as VARRAY(10) OF
  employe;
CREATE TABLE stage (
  lieu varchar(10),
  date Date,
  participants directeurs);

INSERT INTO stage VALUES ('Disney','20 mars
  05', directeurs (employe(1,'Max','12 juin
  60'), employe(2,'Toto' '15 mai 65')));
```

Interroger des collections

```
CREATE TYPE emps as TABLE OF personne;
CREATE TYPE service AS OBJECT (
        numserv number,
        nom varchar(5),
        directeur personne,
        membres emps;
CREATE TABLE LesServices OF service
        NESTED TABLE membres STORE AS membres_tab;
```

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

```
Select s.membres  from LesServices s;
```

Renvoie la collection des membres sous la forme imbriquée :

```
Membres (id, nom, datnais)
---------------------------
emps(personne(1,max,12-06-60),personne(2,toto,15-05-65))
emps(personne(3,lulu,12-12-63),personne(4,joe,13-01-82))
...
```

Navigation dans les collections

Pour naviguer dans des collections, il faut désimbriquer (ou aplatir) la collection.

L'expression **TABLE** permet d'interroger une collection dans la clause **FROM** comme une table.

```
SELECT m.*
    FROM LesServices s, TABLE(s.membres) m;
```

% s représente un service, s.membres une collection, m un employé

Renvoie la collection des membres, sous forme désimbriquée :

```
Id      Nom             date
-----------------------------
1       Max             12-06-60
2       Toto            15-05-65
3       Lulu            12-12-63
4       Joe             13-01-82
```

Expression Table

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

```
Select *
From table (select s.membres
            from LesServices s where numserv=
  432)
```

Renvoie la collection des membres du service 432.

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.

Mise à jour atomique d’attribut varray

```
CREATE TYPE ProjectList AS VARRAY(50) OF Project;
CREATE TABLE depart (dept_id  NUMBER(2), name VARCHAR2(15),budget NUMBER(11,2), projects ProjectList);
INSERT INTO depart
     VALUES(60, 'Security', 750400,
       ProjectList(Project(1, 'Issue New Employee Badges', 13500),
                   Project(2, 'Find Missing IC Chips', 2750),
                   Project(3, 'Upgrade Alarm System', 3350),
                   Project(4, 'Inspect Emergency Exits', 1900)));
END;
DECLARE
  new_projects ProjectList :=
    ProjectList(Project(1, 'Issue New Employee Badges', 13500),
                Project(2, 'Develop New Patrol Plan', 1250),
                Project(3, 'Inspect Emergency Exits', 1900),
                Project(4, 'Upgrade Alarm System', 3350),
                Project(5, 'Analyze Local Crime Stats', 825));
BEGIN
  UPDATE department
    SET projects = new_projects WHERE dept_id = 60;
END;
```

Exemples

Insertion d'un nouveau membre dans le service 123 :

```
INSERT INTO TABLE (SELECT s.membres
                   FROM LesServices s
                   WHERE numserv= 123)
VALUES (1,'martin','12 juin 60');
```

Modification d'un membre du service 123 :

```
UPDATE TABLE (SELECT s.membres
              FROM LesServices s
              WHERE numserv= 123)m
SET VALUE(m) = personne(1,'martin','12 mai 60')
  WHERE m.id = 1;
```

Exemples

Supprimer un membre du service 123 :

```
DELETE FROM TABLE (SELECT s.membres
                   FROM LesServices s
                   WHERE numserv= 123)m
WHERE m.id = 1;
```

Collections multiniveaux

- Types collection dont les éléments sont eux-mêmes des collections :
  - Nested tables of nested tables
  - Nested tables 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 multiniveaux

```
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)
   nested table reg store as reg-tab
   (nested table agglomerations store as villes-tab);
```

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

Collections multiniveaux

L'expression **TABLE** permet de désimbriquer des collections multiniveaux (table et varray) :

La requête suivante donne le nom de toutes les villes :

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

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))
                          )
                    ));
```

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.**

Mise à jour d'éléments d'une collection multiniveau

Insertion dans une 'nested table' qui comprend une 'nested table'.
On insère une nouvelle région, comprenant une nested table (les villes) dans la table des régions.

```
INSERT INTO TABLE(SELECT reg  FROM pays
  WHERE nom='France')
VALUES ('Ile de France', villes
  (ville('Paris',75)));
```

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.agglomérations
   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.agglomérations
   FROM pays p, table(p.reg) r
   WHERE p.nom = 'France' and r.nom = 'Rhône-Alpes')
VALUES ('Annecy', 74)
```