SQL3 - l'objet-relationnel

Syntaxe (Oracle 10&11)

➢ LA DEFINITION DES OBJETS

Type abstrait

```
CREATE TYPE nomTypeAbstrait AS OBJECT
( nomAtt1 nomType, .. ,
   nomAttn nomType )
/
```

Type abstrait utilisé par un autre type

```
CREATE TYPE nomTypeAbstrait AS OBJECT
( nomAtt1 nomType, .. ,
   nomAttn nomTypeAbstraitM )
/
```

Type abstrait utilisé par une table TOR

TOR : table objet-relationnelle

```
CREATE TABLE nomTOR OF nomTypeAbstrait
( CONSTRAINT contrainte, ..)
/
```

Tableaux prédimensionnés

```
CREATE TYPE listefinieType
AS VARRAY(nombreMax) OF elementType
/
CREATE TYPE nomType AS OBJECT
        ( … , nomAtt listefinieType )
/
```

La référence d'objet

```
CREATE TYPE nomTypeAbstrait AS OBJECT
( nomAtt nomType, .. ,
   nomAtt REF nomTypeAbstrait )
/
```

Soit un lien de TOR1 vers TOR2 :

```
CREATE TABLE nom_TOR1 OF nomTypeAbstrait
(CONSTRAINT contrainte,
nomAtt SCOPE IS nom_ TOR2) ;
```

Avoir défini auparavant :

```
CREATE TABLE nom_TOR2 …
( CONSTRAINT … )
OIDINDEX nomdelindex ;
```

Tables imbriquées

```
CREATE TYPE elementType AS OBJECT
(   …   )
/
CREATE TYPE listeType AS TABLE OF elementType
/
CREATE TYPE nomTypeAbstrait AS OBJECT
    (…, table_imbriquee listeType)
/
CREATE TABLE nomTOR OF nomTypeAbstrait
( CONSTRAINT contraintes de relation)
NESTED TABLE table_imbriquee STORE AS tabimbric
/
```

TAD & sous-types

-- Le super-type

```
CREATE TYPE superType AS OBJECT
      (nomAtt1 nomType,
          …
          )
          NOT FINAL
/
```

-- Un sous-type

```
CREATE TYPE sousType UNDER superType
 ( nomAttsupp nomType,
          …
          )
/
```

NOT FINAL NOT INSTANTIABLE pour une classe qui n'hébergera pas d'instances
Pas de hiérarchie des tables, ni d'inclusion des objets d'un table dans l'autre.

Le dictionnaire

-- Visualisation des structures à plusieurs niveaux

```
SET DESCRIBE DEPTH 3
DESC nomTOR
DESC nomType
```

-- Dictionnaire, mes types abstraits

```
SELECT type_name
FROM user_types;
```

-- Dictionnaire, mes tables objets-relationnelles

```
SELECT table_name
FROM user_object_tables ;
```

-- D'autres sources d'information

```
user_type_methods…
```

TAD & méthodes

```
CREATE TYPE nomType AS OBJECT
    (nomAtt1 NOMTYPE,
       …
       MEMBER FUNCTION nomFonction ( [nomArgs] ) RETURN nomType ),
       et/ou
       MEMBER PROCEDURE nomProc ( [nomArgs] )    )
       NOT FINAL
```

Les méthodes seront définies à l'aide de procédures ou de fonctions selon leur nature.

```
CREATE TYPE BODY nomType AS
    MEMBER FUNCTION nomFonction ( [nomArgs nomTypes] ) RETURN AutreType IS
        BEGIN
        ..
            Code PL/SQL dont SELF pour l'objet courant
            SELF.attribut est possible
            ..
            RETURN … ;
        END;
        /
    et/ou MEMBER PROCEDURE nomProc ( [nomArgs nomTypes] )
END nomProc ;

END nomType ;
/
```

Possible redéfinition des méthodes

```
CREATE TYPE sousType UNDER superType
(  nomAttributs supplémentaires,
OVERRIDING MEMBER FUNCTION nomFonction RETURN nomType
  // ou MEMBER PROCEDURE nomProc..)
```

➢ LA MANIPULATION DES OBJETS

Manipuler les compositions typeAbstrait ()

Composition dans son ensemble

ALIAS.NOMATT = **typeAbstrait** (, ,)

Notation pointée : *alias.nomAtt.composant*

Manipuler les références REF ()

```
SELECT REF(OI )  FROM nom_TOR OI
WHERE OI .PK=…
```

Notation pointée : alias.nomAttREF.nomUnAttCible

Et aussi **DEREF** (..), **IS DANGLING**

Manipuler les listes limitées TABLE ()

```
FROM nom_TOR aliasT,
        TABLE(aliasT.nomAttListeLim) aliasA
```

Réutilisation aliasA.nomAttribut ou aliasA.COLUMN_VALUE

Manipuler les tables imbriquées THE ()

```
THE  (  SELECT nomTableIMB  FROM nom_TOR
                WHERE nomAttPK = valeur)
```

Manipuler un objet VALUE ()

Utile pour appliquer une méthode :

```
SELECT VALUE(n).nomMethodeF ( .. )
FROM nom_TOR n ;
```

Interrogation sur l'appartenance à un (sous-)type

```
FROM nom_TOR n
WHERE VALUE(n)  IS OF ( nomType ) ;
```

Accès aux objets, attributs et méthodes du sous-type

```
TREAT (VALUE (c) AS nomType)
        .nomMethodeF ou n
```