CADI Zouheir

ORACLE 8 OBJET

Exposé CNAM
Vendredi 10 janvier 2003

SOMMAIRE

INTRODUCTION

Ce rapport est le fruit d'un aperçu bibliographique effectué pour un exposé présenté au CNAM. L'objectif en est de présenter les fonctionnalités d'Oracle en matière de base objet.

Nous évoquerons dans un premier temps le positionnement des principaux éditeurs de SGBD dans le domaine de l'objet relationnel. Nous verrons ensuite quelles sont les possibités d'Oracle sur les points suivants :

- type abstrait de données
- imbrication d'objets (implantation par valeur d'un lien entre 2 objets)
- pointeurs (implantation par référence d'un lien entre 2 objets)
- tableaux prédimensionnés
- tables imbriquées
- méthodes associées à des objets
- vues objets relationnelles

ARCHITECTURE

Technologie objet-relationnelle : combinaison d'un moteur relationnel et d'un système objet.

Le figure 1 ci-dessous représente un schéma de l'architecture d'un SGBD Objet-relationnel.

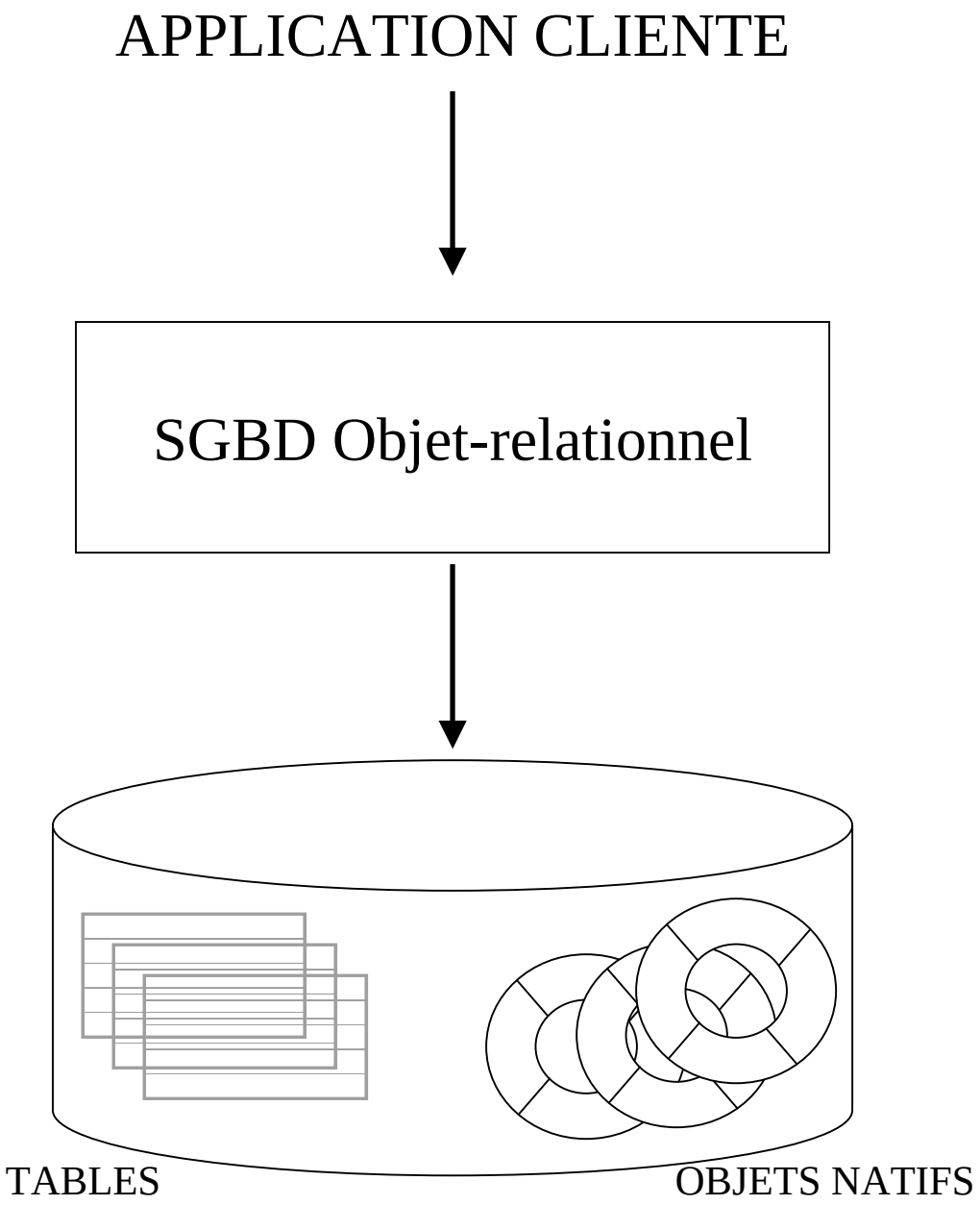


Fig1. Architecture d'un SGBD objet relationnel

POSITIONNEMENT DES PRINCIPAUX EDITEURS DU MARCHE

Les principaux acteurs du marché sont representés dans le tableau ci-dessous

Tableau 1. Revenus provenant de la vente de nouvelles licences de SGBD

Société	2000 Part de marché (%)	1999 Part de marché (%)
Oracle	33.8	31.4
IBM	30.1	29.9
Microsoft	14.9	13.1
Sybase	3.2	3.3
Informix	3.0	5.0
Autres	15.0	17.3
Total Market	**100.0**	**100.0**

Source: Gartner Dataquest (May 2001 – GAR1)

IBM

- 1991, Object Relational Server né dans les laboratoires de l'entreprise
- Juillet 1995, IBM sort la version objet-relationnelle de son serveur DB
- Après la sortie du produit concurent d'informix, IBM s'engage avec DB2 Universal Server 2 puis 3, dans la course aux serveurs universels.

Microsoft

- Développement conjoint avec Sybase jusqu'en 1994
- 1992, Version beta
- 1993 SQL-Server 4.2 sur NT
- 1994 SQL-Server 6.0
- 1996 SQL-Server 6.5 Windows NT 4.0
- 1998 SQL-Server 7.0
- 2000 Microsoft .NET

Microsoft sember éviter la fusion des deux technologies préférant plutot les fédérer au sein de COM et DCOM. La version 7.0 n'offre pas de fonctionnalités objet-relationel

INFORMIX

- 1996 Adopte la technologie Objet relationnel avec l'achat du SGBD d'Illustra
- 2001 Racheté par IBM

SYBASE

- 1994 Séparation avec Microsoft après avoir développé conjointement le noyau du SGBD SQL Server

- A rejoint le camp de l'universalité avec Adaptive Server Entreprise. Pas de documents sur l'intégration réelle de l'objet.

Computer Associates Ingres

- 1996, Prototype Jasmine fruit des travaux de C.A. et Fujitsu
- 1997 produit commercialisé sous les plates-formes Windows NT et Unix

Computer Associates propose 2 moteurs distincts : Ingres pour le relationel, Jasmine pour les nouvelles technologies et le multimedia (Jasmine ODB).

PostgreSQL

Cet éditeur est cité car l'outil qu'il commercialise est dérivé du SGBD objet Posgres développé à l'université de Berkley en 1986.

A l'heure actuelle, et parmi les acteurs leaders du marché, seul Oracle et IBM sont présent dans la technologie objet-relationnel. CA opte pour deux moteurs distincts.

L'intégration du moteur objet s'inscrit dans une orientation plus général comprenant plusieurs axes technologiques : evolution interne du moteur du SGBD, connexion entre le moteur relationnel et les moteurs spécialisés pour la manipulation de certains types de données complexes, ...

MISE EN OEUVRE AVEC ORACLE 8

1. Environnement de test

L'environnement sur lequel ont été testé les requêtes proposées est le suivant :

- Ordinateur personnel
  - processeur Intel Pentium III
  - 533 MHz
  - 640 MO RAM
- Système : Windows XP Professionnel (Version 2002)
- Oracle 8i Entreprise Edition (Version 8.1.7.0.0)

2. Type Abstrait de données

2.1. Définition

Un type abstrait de données (TAD) est un nouveau type d'attribut défini par l'utilisateur qui enrichit la collection existante.

Il peut correspondre à une structure de données partagées ; dans ce cas, il peut être utilisé dans une ou plusieurs tables et entrer dans la composition d'un ou plusieurs autres types. Il peut également decrire la structure et le comportement des objets stockés dans des tables objet-relationnelle.

2.2. Exemple d'implémentation

Création d'un type

```
CREATE TYPE article_objtyp AS OBJECT (
num         NUMBER,
lib         VARCHAR2(10),
prxUni      NUMBER(6,2),
delApr      NUMBER(3)
)
/
```

Type utilisé pour la déclaration d'une table objet relationnelle

```
CREATE TABLE article OF article_objtyp
/
```

Chaque ligne de la table représentera un objet de type client_objtyp. Ces objets sont référençables : d'autres objets peuvent référencer une ligne objet en utilisant l'OID qui lui correspond. Oracle ne fournit pas de documentation concernant l'accès à la structure interne de cet identifiant. Ce dernier est utilisé pour construire les attributs de type REF et les adresses des enregistrement des tables imbriquées.

Options de création de tables

```
CREATE TABLE client OF client_objtyp (num PRIMARY KEY)
OBJECT ID SYSTEM GENERATED
/
```

Généré par le système. Par défaut

```
CREATE TABLE client OF client_objtyp (num PRIMARY KEY)
OBJECT ID PRIMARY KEY
/
```

La clé primaire sert d'OID. Plus efficace si la clé primaire est plus petite en taille qui la place prise par defaut lorsque l'OID est généré par le système.

3. EMBEDED OBJECTS – Table avec des objets comme attribut

3.1. Définition

Une table avec un objet comme attribut correspond à une implatation par valeur. Une implantation par valeur est toujours une agrégation (VUI-091). Les cycles de vie des objets CLIENT et ADRESSE sont liés.

3.2. Exemple d'implémentation

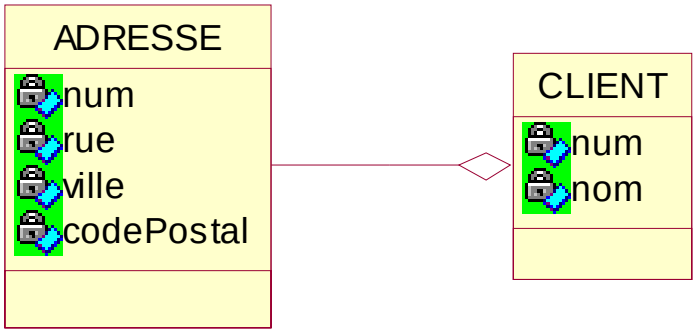


Fig 2. Modèle UML à la base de l'exemple (implantation par valeur)

```
/* Destuction des types et des objets crees */
DROP TABLE client ;
DROP TYPE client_objtyp ;
DROP TYPE adresse_objtyp ;


CREATE TYPE adresse_objtyp AS OBJECT (
num             NUMBER(4),
rue             VARCHAR2(20),
ville           VARCHAR2(20),
codePostal      NUMBER(5)
)
```

```
/

CREATE OR REPLACE TYPE client_objtyp AS OBJECT(
num             NUMBER(6),
nom             VARCHAR2(20),
adresse         adresse_objtyp
)
/

CREATE TABLE client OF client_objtyp (num PRIMARY KEY)
    OBJECT ID PRIMARY KEY
/
```

Si nous faisons un desc à l'issue de ces requêtes, nous obtenons le résultat suivant :

```
DESC client
Nom                                       NULL ?          Type
 ----------------------------------------- -------- ----------------------------
 NUM                                      NOT NULL        NUMBER(6)
 NOM                                                      VARCHAR2(20)
 ADRESSE                                                  ADRESSE_OBJTYP
```

3.3. Insert

Il est obligatoire d'utiliser le constructeur de type de l'objet manipulé comme attribut. Les commandes suivantes insèrent des objets (objet principal et objet attribut).

```
INSERT INTO client VALUES (
1,
'entreprise 1',
adresse_objtyp(100,'rue de la bastille','Paris','75010')
)
/

INSERT INTO client VALUES (
2,
'entreprise 2',
adresse_objtyp(50,'rue du soleil','Marseille','13000')
)
/

INSERT INTO client VALUES (
3,
'entreprise 3',
adresse_objtyp(20,'rue des cathares','Narbonne','11100')
)
/
```

num	nom	adresse			
		num	rue	ville	codePostal
1	entreprise 1	100	rue de la bastille	Paris	75010
2	entreprise 2	50	rue du soleil	Marseille	13000
3	entreprise 3	20	rue de cathares	Narbonne	11100

3.4. Select, update, delete

On utilise la notation pointée pour acceder à l'objet manipulé comme attribut d'un objet principal

Exemple :

```
SELECT c.nom, c.adresse.ville
FROM client c
/

Résultat de la requête précédente
NOM                 ADRESSE.VILLE
-------------------- --------------------
entreprise 1        Paris
entreprise 2        Marseille
entreprise 3        Narbonne
```

4. Les pointeurs

4.1. Définition

Une implantion par référence correspond à un objet de type pointeur sur l'objet lié. Avec Oracle 8, les pointeurs sont mis en oeuvre via les attributs de type REF. Ils servent à manipuler les objets stockés par leur OID et répondent ainsi à la notion d'identité d'objet (SOU). Nous n'aborderons ci-dessous que la référence simple comme il est decrit dans le schéma ci-dessous.

4.2. Exemple d'implémentation

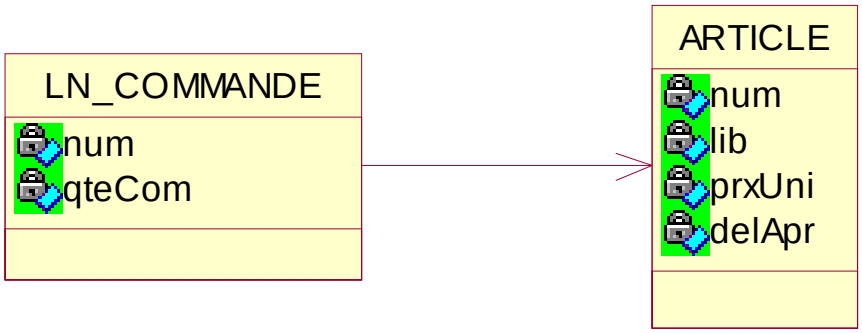


Fig3. Modèle UML à la base de l'exemple (implantation par référence)

```
/* Destruction des objets et des types */
DROP TABLE ln_commande ;
DROP TABLE article;
DROP  type article_objtyp FORCE;
DROP type ln_commande_objtyp ;


CREATE TYPE article_objtyp AS OBJECT(
num         NUMBER,
lib         VARCHAR2(10),
prxUni      NUMBER(6,2),
delApr      NUMBER(3)
)
/

CREATE TYPE ln_commande_objtyp AS OBJECT(
num         NUMBER,
qtCom       NUMBER,
article     REF article_objtyp
)
/

CREATE TABLE article OF article_objtyp
```

```
        (CONSTRAINT pk_num PRIMARY KEY (num))
/

CREATE TABLE ln_commande OF ln_commande_objtyp (PRIMARY KEY
(num))
/

DESC ln_commande ;
Nom                     NULL ?          Type
---------------------------------------- -------- ----------------------------
NUM                     NOT NULL        NUMBER
QTCOM                                   NUMBER
ARTICLE                                 REF OF ARTICLE_OBJTYP



/* jeu dessai de la table article */
INSERT INTO article VALUES
        (1,'gombo',5.00,10);

INSERT INTO article VALUES
        (2,'piment esp',10.00,10);

INSERT INTO article VALUES
        (3,'piment oig',7.50,10);

INSERT INTO article VALUES
        (4,'riz basma',3.75,10);

INSERT INTO article VALUES
        (5,'riz thaï',3.75,10);
```

4.3. Insertion dans la table contenant le pointeur

4.3.1. Insert sans affectation de pointeur

Les commandes suivantes inserent des objets dans la table ln_commande avec initialisation du pointeur article à NULL.

```
INSERT INTO ln_commande VALUES (11,500,NULL);

INSERT INTO ln_commande VALUES (12,100,NULL);

INSERT INTO ln_commande VALUES (13,1500,NULL);

INSERT INTO ln_commande VALUES (21,1500,NULL);

INSERT INTO ln_commande VALUES (31,750,NULL);
```

4.3.2. Insert avec affectation de pointeur

Les commandes suivantes inserent des objets dans la table ln_commande. Chaque objet inséré est rataché à l'article sur lequel il pointe. Le format général de la requête est : INSERT suivie de SELECT REF qui permet de ratacher l'objet inséré à l'objet d'une autre table sur lequel il pointe.

```
INSERT INTO ln_commande
SELECT 32,  800, REF(a)
FROM article a WHERE a.lib='gombo'
/
```

4.4. Update

4.4.1. Affectation de pointeur

La commande UPDATE suivie de SELECT REF permet de rattacher un objet d'une table à celui sur lequel il pointe.

```
UPDATE ln_commande ln
SET ln.article=
        (SELECT REF(a) FROM article a WHERE a.num=1)
WHERE ln.num=11
/

UPDATE ln_commande ln
SET ln.article=
        (SELECT REF(a) FROM article a WHERE a.num=2)
WHERE ln.num=12
/

UPDATE ln_commande ln
SET ln.article=
        (SELECT REF(a) FROM article a WHERE a.num=4)
WHERE ln.num=31
/
```

4.4.2. Modification d'un pointeur

La modification d'un pointeur s'effectue de la même façon que pour la commande précédente : UPDATE suivie de SELECT REF.

4.4.3. Suppression de pointeur

La suppression d'un pointeur est mise en oeuvre par l'affectation de la valeur NULL au pointeur.

```
UPDATE ln_commande ln
SET ln.article=NULL
WHERE ln.num=33
/
```

4.5. Delete

Il est possible d'utiliser des expressions de chemins à base de pointeur dans la commande DELETE. Supprimons les objets de la table ln_commande qui contiennent un pointeur vers la table article

```
DELETE FROM ln_commande ln
WHERE ln.article IS NOT NULL
```

4.6. Select

Quelles sont les lignes de commandes pour lequel il y a achat de gombo ?

```
SELECT ln.num , ln.qtCom
FROM ln_commande ln
where ln.article.lib='gombo'
```

```
NUM          QTCOM
----------   ----------
11           500
32           200
```

4.6.1. Value

La fonction VALUE s'applique à tout type. Dans le cas exposé ci-dessous, elle restitue les valeurs des pointeurs si le type n'est pas nul.

```
SELECT VALUE(ln)
FROM ln_commande ln
WHERE ln.num=11

VALUE(LN)(NUM, QTCOM, ARTICLE)
--------------------------------------------------------------------------------
LN_COMMANDE_OBJTYP(11, 500,
0000220208920D367B3D734C1392B91B79FCF4FBBCAEEE6E3E5B
FF4EAB936828B95FD4728B)
```

4.6.2. Déréférencement

La fonction DEREF() s'applique à un pointeur et extrait le contenu du type à l'adresse du pointeur. Un DEREF affiche l'intégralité du contenu du type à l'adresse du pointeur. Dans l'exemple cité, il n'est pas possible d'obtenir uniquement le numéro de l'article. Toutefois, la valeur retournée peut être typé et exploité dans un programme PL/SQL.

Quels sont les articles de la ligne de commande 12

```
SELECT DEREF(ln.article)
```

```
FROM ln_commande ln
WHERE ln.num=32
/

DEREF(LN.ARTICLE)(NUM, LIB, PRXUNI, DELAPR)
--------------------------------------------------------------------------------
ARTICLE_OBJTYP(1, 'gombo', 5, 10)
```

5. Tableaux prédimensionnés

5.1. Définition

Un VARRAY est une collection ordonnée et limitée d'éléments de même type (SOU)

5.2. Exemple d'implémentation

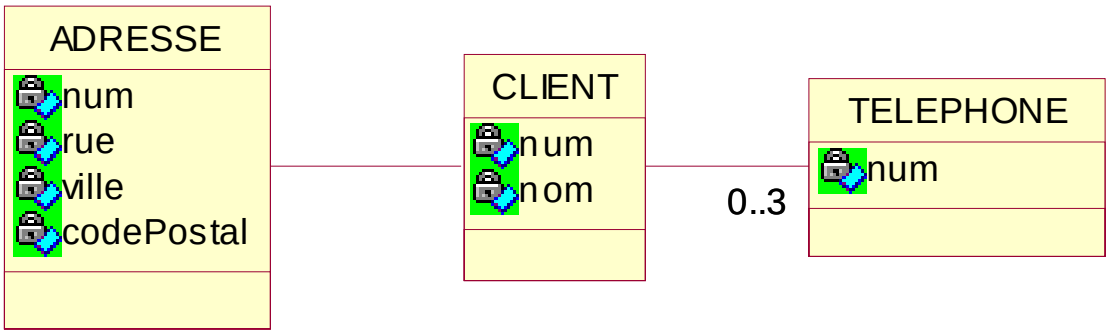


Fig 4. Modèle UML à la base de l'exemple

5.2.1. Création d'un VARRAY à partir d'un autre objet

```
CREATE TYPE tel_objtyp AS OBJECT (
numtel                  VARCHAR2(20))
/
CREATE TYPE telLst_vartyp AS VARRAY(3) OF tel_objtyp
/
```

5.2.2. Création d'un VARRAY à partir d'un type standard

```
CREATE TYPE telLst_vartyp AS VARRAY(5) OF VARCHAR2(20)

CREATE OR REPLACE TYPE client_objtyp AS OBJECT(
numCli          NUMBER,
nom             VARCHAR2(20),
adresse         adresse_objtyp,
telLst          telLst_vartyp)
/
```

5.2.3. Jeu d'essai

```
/* Destuction des types et des objets crees */
DROP TABLE client ;
DROP TYPE client_objtyp ;
DROP TYPE adresse_objtyp ;
DROP TYPE telLst_vartyp ;

CREATE TYPE adresse_objtyp AS OBJECT (
num             NUMBER(4),
rue             VARCHAR2(20),
ville           VARCHAR2(20),
codePostal      NUMBER(5)
)
/

CREATE TYPE tel_objtyp AS OBJECT (num VARCHAR2(10))
/

CREATE TYPE telLst_vartyp AS VARRAY(3) OF tel_objtyp
/

CREATE OR REPLACE TYPE client_objtyp AS OBJECT(
num             NUMBER(6),
nom             VARCHAR2(20),
adresse                 adresse_objtyp,
telLst          telLst_vartyp
)
/

CREATE TABLE client OF client_objtyp (num PRIMARY KEY)
        OBJECT ID PRIMARY KEY
/

DESC client ;
```

Résultat du desc

```
Nom                                       NULL ?       Type
----------------------------------------- -------- ----------------------------
NUM                                       NOT NULL     NUMBER(6)
NOM                                                    VARCHAR2(20)
ADRESSE                                                ADRESSE_OBJTYP
TELLST                                                 TELLST_VARTYP
```

5.3. Insert d'une ligne entière (objet principal + VARRAY)

Il est nécessaire d'utiliser le constructeur de type pour les insertions.

```
INSERT INTO client VALUES (
1,
'entreprise 1',
adresse_objtyp(100,'rue de la bastille','Paris','75010'),
telLst_vartyp(tel_objtyp('0140404040'),
                tel_objtyp('0140404245'),
                tel_objtyp('0130303030')
                )
)
/

INSERT INTO client VALUES (
2,
'entreprise 2',
adresse_objtyp(50,'rue du soleil','Marseille','13000'),
                telLst_vartyp()
)
/

INSERT INTO client VALUES (
3,
'entreprise 3',
adresse_objtyp(20,'rue des cathares','Narbonne','11100'),
telLst_vartyp(
                tel_objtyp('0540125478'),
                tel_objtyp( '0554528565'),
                NULL)
)
/
```

Tableau 2. Résultat des insertions

num	nom	adresse				{telLst}
		num	rue	ville	codePostal	num
1	entreprise 1	100	rue de la bastille	Paris	100	0140404040
						0140404245
						0130303030
2	entreprise 2	50	rue du soleil	Marseille	13000	NULL
						NULL
						NULL
3	entreprise 3	20	rue de cathares	Narbonne	11100	0540125478
						0554528565
						NULL

Manipulation
Pour manipuler un enregistrement particulier d'un VARRAY, il est necessaire d'utuliser un programme PL/SQL.

5.4. Insert dans le VARRAY

Une insertion d'enregistrements dans un VARRAY se programme en PL/SQL à l'aide d'un UPDATE dans la table objet-relationnelle qui contient le VARRAY. La procédure suivante affecte 2 numéros de téléphone à l'entreprise 2.

```
DECLARE
    telLst telLst_vartyp := telLst_vartyp(tel_objtyp('0611252525'),
                                          tel_objtyp('0540404025'),
                                          NULL
                                          ) ;
BEGIN
    UPDATE client
    SET telLst = telLst
    where num=2;
END ;
/
```

5.5. Update

Pour modifier un enregistrement particulier dans un VARRAY, il faut programmer en PL/SQL l'opération UPDATE dans la table objet-relationnelle qui contient le VARRAY.

```
DECLARE
    atelLst telLst_vartyp;
BEGIN
    SELECT telLst into atelLst
    FROM client
    WHERE num=1;
    atelLst(3).num :='0658525252';
    UPDATE client
    SET telLst=atelLst
    where num=1;
END ;
/
```

5.6. Delete

Pour supprimer un enregistrement particulier dans un VARRAY, il faut programmer en PL/SQL l'opération UPDATE dans la table objet-relationnelle qui contient le VARRAY.

```
DECLARE
    atelLst telLst_vartyp;
BEGIN
    SELECT telLst into atelLst
    FROM client
    WHERE num=1;
    atelLst(2).num :=NULL;
```

```
        UPDATE client
        SET telLst=atelLst
        where num=1;
    END ;
    /
```

Pour supprimer tous les enregistrements d'un VARRAY, on utilise la méthode DELETE associée à tout VARRAY. Dans la procédure précédente, on remplace : atelLst(2).num:=NULL par atelLst.DELETE.

5.7. Select

Il nest pas possible d'accéder directement à un element d'un VARRAY. Un select restitue le tableau dans son integralité.

6. Tables imbriqués

6.1. Définitions

Collection non ordonnée et non limitée d'éléments de même type. Oracle 8.0 autorise l'implantation d'un seul niveau d'imbrication (SOU).
Une table imbriquée est représentée par une colonne dans la description de la table principale. Pour chaque ligne de la table principale, il peut exister un nombre illimité de lignes dans la table imbriquée (CHA).

6.2. Exemple d'implémentation

L'exemple se base sur le modèle représenté sur la figure ci-dessous

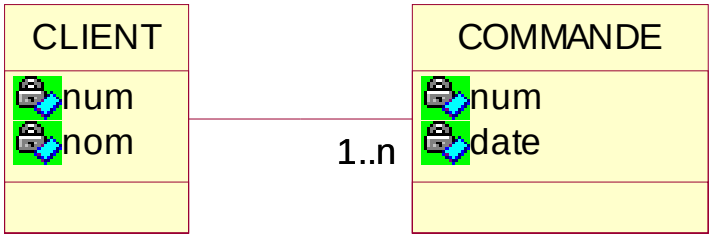


Fig 5. Modèle UML à la base de l'exemple

```
/* Destruction des types et objets */
DROP TABLE client ;
DROP TYPE client_objtyp ;
DROP TYPE commandeLst_tabTyp ;
DROP TYPE commande_objtyp ;
```

```
CREATE TYPE client_objtyp
/

CREATE TYPE commande_objtyp AS OBJECT(
num    NUMBER(6),
datCom        DATE
)
/

CREATE TYPE commandeLst_tabtyp AS TABLE OF commande_objtyp
/

CREATE OR REPLACE TYPE client_objtyp AS OBJECT(
num                    NUMBER(6),
nom                    VARCHAR2(20),
commandeLst            commandeLst_tabTyp
)
/

CREATE TABLE client OF
```

6.3. Insert

6.3.1. Insert avec initialisation sans insertion de valeurs dans la table imbriquée

La commande insert avec le constructeur de type de la nested table stocke un objet dans la table et initialise à vide la table imbriquée associée.

```
/* insertion dans la table adresse -nested table vide*/
        INSERT INTO client VALUES (
                1,
                'entreprise 1',
                commandeLst_tabTyp()
                                )
        /

        INSERT INTO client VALUES (
                2,
                'entreprise 2',
                commandeLst_tabTyp()
                                )
        /
```

num	nom	commande	
		num	datCom
1	entreprise 1		
2	entreprise 2		

6.3.2. Insert avec initialisation et insertion de valeurs dans la table imbriquée

La commande insert suivante, avec le constructeur de type de la nested table stocke un objet dans la table et initialise la table imbriquée associée avec des enregistrements.

```
        INSERT INTO client VALUES (
        3,
        'entreprise 3',
        commandeLst_tabTyp( commande_objtyp(31,SYSDATE-40),
                            commande_objtyp(32,SYSDATE-30),
                            commande_objtyp(33,SYSDATE-20))
        )
```

num	nom	commande	
		num	datCom
1	entreprise 1		
2	entreprise 2		
3	entreprise 3	31	date1
		32	date2
		33	date3

6.3.3. Insert avec THE

La commande INSERT INTO THE (SELECT ...) stocke un enregistrement dans la table imbriquée désignée par THE. Le select après le THE doit retourner un seul objet, ce qui permet de sélectionner la table imbriquée associée. Ces commandes ne peuvent être utilisées que si la table imbriquée a été initialisée.

Insertion d'enregistrements dans la table imbiquée de l'entreprise 1

```
INSERT INTO
THE (SELECT c. commandeLst FROM client c
        WHERE c.num=1)
VALUES(11,SYSDATE-10)
/

INSERT INTO
THE (SELECT  c. commandeLst FROM client c
        WHERE c.num=1)
VALUES(12,SYSDATE-5)
/

INSERT INTO
THE (SELECT  c. commandeLst FROM client c
        WHERE c.num=2)
VALUES(21,SYSDATE-60)
/
```

num	nom	commande	
		num	datCom
1	entreprise 1	11	date11
		12	date12
2	entreprise 2	21	date21
3	entreprise 3	31	date31
		32	date32
		33	date33

6.3.4. Insert dans la table principale uniquement

L'instruction suivante n'initialise pas à vide la table imbriquée

```
INSERT INTO client (num,nom)
VALUES(4,'entreprise 4')
/
```

Il n'est pas possible d'inserer des enregistrements dans la table imbriquée correspondant à cette ligne car elle n'a pas été initialisée. Pour pouvoir le faire, il faudrait supprimer la ligne de la table principale et la réinserer en en initialisant la table imbriquée avec le constructeur de type.

num	nom	commande	
		num	datCom
1	entreprise 1	11	date11
		12	date12
2	entreprise 2	21	date21
3	entreprise 3	31	date31
		32	date32
		33	date33
4	entreprise 4		

6.4. Update

Il est possible de modifier :

- les attributs de la table principale
- les attributs des enregistrements de la table imbriquée

6.4.1. Update dans la table principale

Un Update pour des objets de la table principale se fait de façon classique

```
UPDATE client c
SET c.nom=entreprise 11
WHERE num=1
/
```

6.4.2. Update dans la table imbriquée

La commande UPDATE THE (SELECT ...) modifie un enregistrement dans la table imbriquée

```
UPDATE THE
        (SELECT c.commandeLst FROM client c
        WHERE c.num=1) nt
SET nt.datCom=(sysdate-400)
 WHERE nt.num=12;
```

6.5. Delete

6.5.1. Delete dans la table principale comme en relationnel pur

```
DELETE FROM client c
WHERE c.attribut=?
```

6.5.2. Delete dans la table principale avec critère sur la table imbriquée

```
DELETE FROM client c
WHERE EXISTS (SELECT * FROM
                    THE (SELECT cl.commandelst
                    FROM client  cl
```

```
                    WHERE cl.num=c.num) nt
            WHERE nt.num=12)
/
```

6.5.3. Delete dans la table imbriquée

DELETE THE (SELECT ...) supprime un enregistrement dans la table imbriquée.

Comme dans un insert ou un update, le select qui figure après le the doit retourner un seul objet, ce qui permet de sélectionner la table imbriquée associée.

```
DELETE THE
        (SELECT c.commandeLst FROM client c
        WHERE c.num=1) nt
WHERE nt.num=12
```

6.6. Select

6.6.1. Select sur la table imbriquée associée à un objet de la table principale

La requête qui suit le mot THE doit obligatoirement retourner un seul objet de la table principale.

Le mot THE désigne la table imbriquée. Il est necessaire d'utiliser un alias qui fait référence à la table imbriquée de manière à extraire des attributs contenus dans la table en question.

```
SELECT nt.num, TO_CHAR(nt.datCom)
FROM THE(SELECT c.commandelst
            FROM client c
            WHERE c.num=1)nt
/
```

6.6.2. Select sur les tables imbriquées associées à plusieurs objets de la table principale

6.6.2.1. <u>utilisation de l'opérateur ensembliste UNION</u>

```
SELECT nt.num, TO_CHAR(nt.datCom)
FROM THE(SELECT c.commandelst
            FROM client c
            WHERE c.num=1)nt
UNION
SELECT nt.num, TO_CHAR(nt.datCom)
FROM THE(SELECT c.commandelst
            FROM client c
            WHERE c.num=2)nt
```

6.6.2.2. Nested Cursor

```
SELECT c.nom, CURSOR    (SELECT nt.num, TO_CHAR(nt.datCom)
                         FROM TABLE(commandeLst) nt
                         )
FROM client c
WHERE c.num=1 OR c.num=2
/
```

Restriction sur l'utilisation des nested cursors

Les curseurs imbriqués ne peuvent pas figurer dans un SELECT imbriqué (avec l'utilisation de THE) sauf si l'imbrication elle même est un curseur imbriqué

Les curseurs imbriqués ne peuvent pas intervenir dans la déclaration d'une vue, ni figurer dans les clauses FROM ou WHERE d'une requête SELECT.

7. Comparaison NESTED TABLE et VARRAY

Nested Table :

Nombre d'élément illimté

possibilité de définir un index (SOU)

VARRAY

Pas de possibilité de définir un index

Nombre limité d'élément

impossibilité d'accéder directement aux enregistrements via une requête de type SELECT (SOU)

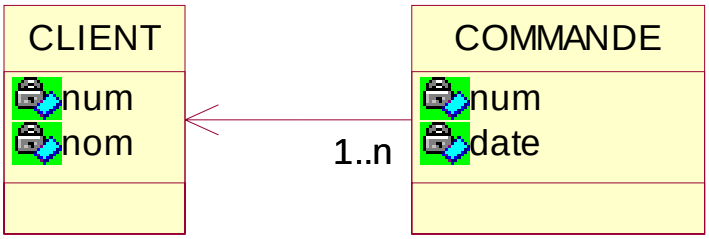
Cette description est représentée dans le tableau ci-dessous.

Tab xxx. Comparaison NESTED TABLE et VARRAY

	NESTED TABLE	**VARRAY**
ORDRE	NON	OUI
INDEX	OUI	NON
ACCES DIRECT (SELECT)	OUI	NON

Si on souhaite effectuer des requêtes sur la collection, il faut utiliser une table imbriquée. Si on souhaite extraire la collection comme un tout, il faut utiliser un varray.

8. Implémentation d'un association 1-N




```
DROP TABLE client;
DROP TABLE commande ;
DROP TYPE commandeRef_typ FORCE;
DROP TYPE client_objtyp FORCE;
DROP TYPE CommandeLst_typ ;
DROP TYPE commande_objtyp FORCE;

CREATE TYPE client_objtyp
/

CREATE TYPE commande_objtyp AS OBJECT(
num         NUMBER,
datCom      DATE,
resp        REF client_objtyp
)
/

CREATE TYPE commandeRef_typ AS OBJECT
(commandeRef      REF commande_objtyp)
/

CREATE TYPE CommandeLst_typ AS TABLE OF commandeRef_typ
/


CREATE OR REPLACE TYPE client_objtyp AS OBJECT(
num                 NUMBER,
nom                 VARCHAR2(20),
commandeLst         CommandeLst_typ
)
/

CREATE TABLE client OF client_objtyp (num PRIMARY KEY)
        NESTED TABLE commandeLst STORE AS tabCommandes
/

CREATE TABLE commande OF commande_objtyp
(CONSTRAINT pk_UV PRIMARY KEY (num), resp NOT NULL)
/
```

ENCAPSULATION DES OBJETS – LES METHODES

1. Definition

Un type abstrait de données peut inclure des méthodes qui manipulent les objets du type . Celles-c doivent être declarées dans la déclaration du type et inplémentées à l'aide de l'instruction CREATE TYPE BODY.

2. Accessibilité

Il est impossible d'affecter un caractère public ou privé à une méthode. Les restrictions sur l'accessibilité ne peuvent être faites que sur un type dans son ensemble, et donc sur ces méthodes. La manipulation d'un type est restreinte par le biais de la commande :

GRANT EXECUTE nomtype TO user

3. Les fonctions

3.1. Exemple

L'exemple cité reprend le cas utilisé pour le paragraphe 4 du chapitre mise en oeuvre sous oracle.

```
CREATE TYPE ln_commande_objtyp AS OBJECT(
num         NUMBER,
qtCom       NUMBER,
article     REF article_objtyp,
MEMBER FUNCTION qtTotale RETURN NUMBER,
PRAGMA RESTRICT_REFERENCES(qtTotale,WNDS, WNPS)
)
/
```

Supposons que nous souhaitons compter la quantité totale commandée. Nous définissons pour cela la méthode qtTotale. La fonction est implémenté comme suit :

```
CREATE OR REPLACE TYPE BODY ln_commande_objtyp AS
MEMBER FUNCTION  qtTotale RETURN number IS qtTot NUMBER;
        BEGIN
                SELECT SUM(ln.qtCom) INTO qtTot
                FROM ln_commande ln;
                RETURN qtTot;
        END qtTotale;
```

```
END;
/
```

Cette méthode peut être utilisée par des objets du type dans un programme PL/SQL ou par des objets de la table ln_commande dans un programme PL/SQL ou une requête.

3.2. Appel dans une requête

```
SELECT ln.qtTotale()
FROM ln_commande ln
```

3.3. Purity levels

Le type est déclaré à l'aide de la clause PRAGMA RESTRICT_REFERENCES qui permet de définir les caractéristiques de la méthode. Les options possibles sont détaillées dans le tableau ci-dessous.

tableau xxx. Purity levels

Option	Interprétation
WNDS (Writes No Database State)	La méthode ne modifie pas l'état de la base de donnée
WNPS (Writes No Package State)	La méthode ne modifie pas les variables des paquetages
RNDS (Reads No Database State)	La méthode n'interroge pas la base de données
RNPS (Reads No Package State)	La méthode ne lit pas les variables des paquetages

4. Les procédures

Il est possible d'associer à un type une méthode qui correspond à une procédure. L'encapsulation d'une procédure obeït aux mêmes règles que les fonctions décrites précédemment : déclaration lors de la création du type, implémentation dans le corps comme suit :

```
CREATE OR REPLACE TYPE BODY untype  AS
        MEMBER PROCEDURE maprocedure(argument IN
        typeArgument, ...) IS

        END maprocedure;
END;
```

Les procédures ne peuvent être appelées que dans des programmes PL/SQL.

5. Surcharge des méthodes (overloading))

6. Comparaison des objets

Oracle 8 permet la comparaison d'objet de même classe avec les fonctions membres MAP ou ORDER. Un type Oracle 8 peut contenir la déclaration d'une fonction MAP ou d'une fonction ORDER mais pas les deux.

6.1. Fonction MAP

Une fonction MAP ne comporte pas de paramètre. Les types de données autorisés dans le retour de la fonction MAP sont DATE, NUMBER, VARCHAR2, CHAR et REAL.

Une fonction MAP doit être implémentée lorsqu'on souhaite comparer deux objets de même type sur la base d'un critère ad hoc. Cette fonction est appelée de manière implicite lorsque deux objets sont comparés dans un programme PL/SQL. L'appel peut aussi être explicite dans des ordres SQL quand il y a des prédicats de comparaison (ORDER BY, GROUP BY, ...).

6.2. Fonction ORDER

Une fonction MAP ne comporte pas de paramètre. Les types de données autorisés dans le retour de la fonction MAP sont DATE, NUMBER, VARCHAR2, CHAR et REAL.

Une fonction MAP doit être implémentée lorsqu'on souhaite comparer deux objets de même type sur la base d'un critère ad hoc. Cette fonction est appelée de manière implicite lorsque deux objets sont comparés dans un programme PL/SQL. L'appel peut aussi être explicite dans des ordres SQL quand il y a des prédicats de comparaison (ORDER BY, GROUP BY, ...).

VUE OBJET-RELATIONNELLE

1. Vues objet-relationnelle d'objets

1.1. Définition

CREATE [OR REPLACE] VIEW mavue OF typedemavue
AS SELECT ....

Une vue peut faire intervenir un nouveau type qui lui est propre.

1.2. Option

WITH CHECK OPTION : interdit l'insertion d'enregistrement dans la vue

WITH OBJECT OID : affecte des OID à chacun des enregistrements de la vue.

WITH READ ONLY : vue non modifiable

2. Vues objet-relationnelle de tables relationnelles

2.1. Définition

```
CREATE OR REPLACE TYPE typedemavue AS OBJECT
(attribut1 Domaine1, Attribut 2 Domaine2, ...);
/

CREATE VIEW mavue OF typedemavue
WITH OBJECT OID (attribut1)
AS SELECT * FROM matablerelationnelle
/
```

BIBLIOGRAPHIE

Livres

SOUTOU C., Objet-relationnel sous Oracle8, Ed. Eyrolles, 1999

CHAPUIS R., Les bases de données Oracle 8i, Ed. Dunod, 2001

Liens

http://www.gartner.com/5_about/press_room/pr20010523b.html

http://otn.oracle.com/