

---

## Les aspects objet-relationnels d'Oracle (de la V8 à la 11g)

Christian Soutou  
<http://www.soutou.net/christian>



---

## Plan

- Généralités
- Définition des objets
  - Types abstrait de données, Héritage
  - Références
  - Collections
- Manipulation des objets
- Interrogation des objets
- Méthodes
- Vues objet



---

## Généralités



---

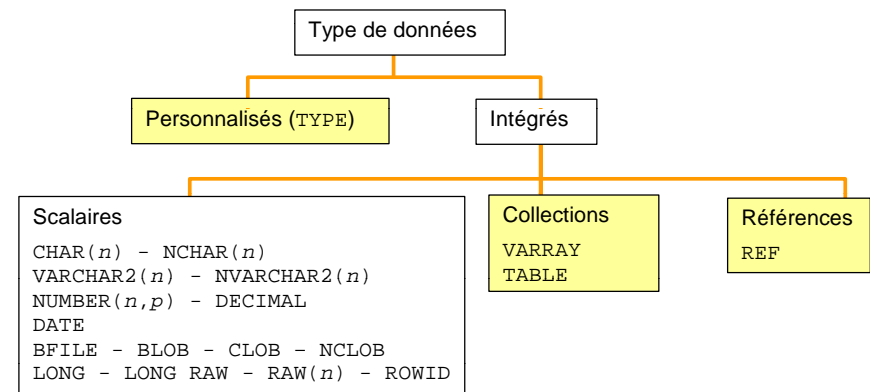
## Modèle de données

- Extension à l'objet du modèle relationnel :  
types abstraits de données (TAD)
  - Collections
  - Identité des objets par l'utilisation de références
  - Héritage
  - Encapsulation par la programmation de méthodes  
(procédures ou fonctions)

# Oracle

- Oracle8 (Juin 1997)
  - Premières fonctions objet (limitées), Partitionnement, Java (JDBC, JSQL, Servlets, Java Server Pages, Entreprise Java Beans)
  - Oracle8i 8.2 (8.1.5) Juillet 1999
  - Oracle8i 8.3 (8.1.7) Août 2000 Améliorations Java, FS, XML, sécurité
- Oracle9i, 9.1 (Novembre 2000), 9.2 (2002)
  - **Héritage**
- Oracle10g, (2004), *Grid Computing* (ressources en *clusters*)
  - **Collections multi-niveaux**
- Oracle11g, (2007) ...
- Compatibilité ascendante avec des bases SQL2 (versions 7)

# Types de données



# Catégories d'objet

- objets persistants qui sont stockés dans des tables objets (*row objects*)
- objets qui composent une colonne d'une table relationnelle (*column objects*)
- objets non persistants qui n'appartiennent pas aux deux précédentes classification et n'existent que durant l'exécution d'un programme

# Catégories d'objet

- Un type Oracle peut être utilisé :
  - pour la construction d'autres types (type composite)
  - pour définir une ou plusieurs tables objet (*object tables*)
  - pour définir une colonne d'une table relationnelle (*column objects*)
  - pour la construction de vues objets (*object views*)

## Définition des objets

## Création d'un type (TAD)

### Syntaxe

```
CREATE [OR REPLACE] TYPE schéma.nomType [AS OBJECT | UNDER schéma.nomSurType]
(
  colonne1 type1, colonne2 type2... ,
  méthode1(paramètres1), méthode2(...)...
)
[[NOT] INSTANTIABLE]
[[NOT] FINAL]
```

Structure  
Comportement  
Relatif à l'héritage

### Limitations

- 30 caractères pour les noms de colonnes
- Pas de LONG, LONGRAW, ROWID, %TYPE

## Type composite

```
CREATE TYPE etat_civil_type AS OBJECT
(nom VARCHAR(30), datenais DATE)
/
CREATE TYPE adresse_type AS OBJECT
(nrue NUMBER(3), rue VARCHAR(40), ville VARCHAR(30))
/
CREATE TYPE Pilote_type AS OBJECT
(brevet CHAR(6), etat_civil_t etat_civil_type,
  adresse_t adresse_type, paye NUMBER(6,2))
/
```

## Table objet

### Syntaxe

```
CREATE TABLE [schéma.]nomTableObjet OF [schéma.] nomType
[(colonne [DEFAULT expression]
  [ contrainteEnLigne [contrainteEnLigne]...
  | contrainteREFEnLigne ]
  | { contrainteHorsLigne | contrainteREFHorsLigne }
  [,colonne... ])]
[OBJECT IDENTIFIER IS { SYSTEM GENERATED | PRIMARY KEY }];
```

### Exemple

```
CREATE TABLE Pilote OF Pilote_type
(CONSTRAINT pk_Pilote PRIMARY KEY(brevet),
 CONSTRAINT df_paye paye DEFAULT 3000,
 CONSTRAINT nn_paye CHECK (paye IS NOT NULL),
 CONSTRAINT ck_paye CHECK (paye BETWEEN 2000 AND 5000),
 CONSTRAINT nn_nom CHECK (etat_civil_t.nom IS NOT NULL),
 CONSTRAINT un_nom UNIQUE (etat_civil_t.nom));
```



# Héritage

---

```
CREATE TYPE Employe_type AS OBJECT
  (codemp CHAR(6), etat_civil etat_civil_type,
   adresse adresse_type, paye NUMBER(6,2))
  INSTANTIABLE NOT FINAL
/
CREATE TYPE Pilote_type UNDER Employe_type
  (nbHvol NUMBER, compagnie VARCHAR(6))
  NOT INSTANTIABLE NOT FINAL
/
CREATE TYPE Pil_instructeur_type UNDER Pilote_type
  (nbHvolIns NUMBER, expireQualif DATE)
  INSTANTIABLE FINAL
/
```



# Les références



# Syntaxe

---

## ■ Dans un type

```
CREATE TYPE type2 {AS OBJECT | UNDER nomSurType2}
  (... , colonne REF type1, ...)
/
```

## ■ Dans une table

```
CREATE TABLE nomTableRelationnelle
  (... , colonne REF type1, ...);
```



# Caractéristiques

---

- pointeur d'un objet (persistant ou non) vers un objet persistant (*row object*)
- Jointures implicites dans le SELECT et le WHERE
- Récursivité des types possible via REF
- Intégrité référentielle (FOREIGN KEY, NOT NULL, REFERENCES)

## Exemple

```
CREATE TYPE Siege_social_type AS OBJECT
  (nrue NUMBER(3), rue VARCHAR(20), ville VARCHAR(15))
/
CREATE TYPE Compagnie_type AS OBJECT
  (comp VARCHAR(4), siege_social_t siege_social_type,
  nomComp VARCHAR(15))
/
CREATE TYPE Avion_type AS OBJECT
  (immat VARCHAR(6), typeAvion VARCHAR(10),
  capacite NUMBER(3), ref_Compagnie REF Compagnie_type)
/
```

## Exemple

```
CREATE TYPE Siege_social_type AS OBJECT
  (nrue NUMBER(3), rue VARCHAR(20), ville VARCHAR(15))
/
CREATE TYPE Compagnie_type AS OBJECT
  (comp VARCHAR(4), siege_social_t siege_social_type, nomComp VARCHAR(15))
/
CREATE TYPE Avion_type AS OBJECT
  (immat VARCHAR(6), typeAvion VARCHAR(10),
  capacite NUMBER(3), ref_Compagnie REF Compagnie_type)
/
CREATE TABLE Compagnie OF Compagnie_type
  (CONSTRAINT pk_Compagnie PRIMARY KEY(comp));

CREATE TABLE Avion OF Avion_type
  (CONSTRAINT pk_Avion PRIMARY KEY(immat),
  CONSTRAINT nn_ref_Compagnie CHECK (ref_Compagnie IS NOT NULL),
  CONSTRAINT refer_Avion ref_Compagnie REFERENCES Compagnie);
```

## Opérateurs pour les références (étudiés dans les parties *Manipulation-Interrogation*)

- `REF(alias)` renvoie un OID
- `DEREF(reference)` extrait le contenu du type à l'adresse passée en paramètre
- `VALUE(alias)` s'applique à tout type de données et extrait les valeurs du type renvoyé
- `DANGLING` (renvoie `TRUE` si la référence est perdue)

## Collections NESTED TABLE et VARRAY

# Les collections

- Principe
  - Collection = { éléments de même type }
- Collections Oracle
  - NESTED TABLE : non ordonnée et non limitée (pas denses)
  - VARRAY : ordonnée et limitée mais extensible (v comme *Variable*)
- Imbrications possibles (multi-niveaux)
  - NESTED TABLE de VARRAY
  - VARRAY de NESTED TABLE
  - VARRAY de VARRAY
  - NESTED TABLE de NESTED TABLE

# Création d'une *nested table*

`pilotes_elt_nt_type`

brevet	nom	age
--------	-----	-----

`pilotes_nt_type`

{pilotes_nt}		
brevet	nom	age

`compagnie_type`

comp	nomComp	{pilotes_nt}		
		brevet	nom	age

`Compagnie`

comp	nomComp	{pilotes_nt}			
		brevet	nom	age	

pilotes\_tabnt

# Syntaxe

- Syntaxe
 

```
CREATE TYPE nomcollection_nt_type
    IS TABLE OF nomelement_elt_nt_type [NOT NULL]
/
CREATE TABLE nomdetable
    [ OF nomType / (coll type1, ... collection_nt collection_nt_type ...) ]
    NESTED TABLE collection_nt STORE AS collection_tabnt;
```
- Exemple

# Contraintes

- Au niveau des colonnes de la collection
  - Pas possible de définir CHECK, NOT NULL et UNIQUE sur une colonne d'une collection *nested table* lors de la création de la table (CREATE TABLE...)
  - Solution : ALTER TABLE, appliquée à la table de stockage (directives ADD, DROP, DISABLE, ENABLE)
- Exemples
  - L'âge d'un pilote doit être compris entre 18 et 60 ans
  - Il n'y a pas d'homonymie

# Contraintes

- Au niveau de la collection elle-même (sur l'objet colonne dans sa globalité)
  - La contrainte NOT NULL assure que toute collection sera toujours non nulle (ne pas confondre avec une collection vide qui ne stocke aucun élément)
  - Solution : ALTER TABLE, appliquée à la table principale (directives ADD, DROP, DISABLE, ENABLE)

```
ALTER TABLE Compagnie
  ADD CONSTRAINT nn_pilotes_nt
    CHECK (pilotes_nt IS NOT NULL);
```

# Restrictions



- Pas possible de définir un déclencheur (*trigger*) sur la table de stockage d'une collection *nested table*
- Pas possible de définir une clé étrangère sur une colonne d'une collection *nested table*
- La table de stockage n'est pas accessible directement par SELECT, INSERT, UPDATE ou DELETE (elle permet simplement de gérer des contraintes ou des index)

# Caractéristiques des *varrays*

- Un *varray* (tableau pré-dimensionné) est une collection ordonnée et limitée
- Les éléments d'un *varray* sont enregistrées au sein de la table (*in line*) où ils sont stockés dans une colonne BLOB
- Chaque élément d'un *varray* ne peut être accéder que par son indice
- L'espace de stockage n'est pas précisé pour collection *varray*
- Une collection *varray* de taille *n*, pourra être initialisée à taille *m* ( $m < n$ ). Il sera possible par la suite d'étendre la collection (sans toutefois excéder la taille *n*).

# Composition

*pilotes\_elt\_vry\_type*

brevet	nom	age
--------	-----	-----

*pilotes\_vry\_type*(3)

{pilotes_vry}		
brevet	nom	age

*Compagnie\_type*

comp	nomComp	{pilotes_vry}		
		brevet	nom	age

*Compagnie*

comp	nomComp	{pilotes_vry}		
		brevet	nom	age

max 3

## Syntaxe

- Syntaxe

```
CREATE TYPE nomcollection_vry_type
    IS VARRAY(n) OF elt_vry_type [NOT NULL]
/
```

- Exemple

## Collection simple

- Syntaxe

```
CREATE TYPE collection_simple_type IS TABLE OF type_oracle
/
```

- Restrictions

*type\_oracle* ne doit pas être

- BOOLEAN
- LONG, LONG RAW
- NATURAL
- REF CURSOR
- STRING

- Exemples

```
CREATE TYPE listeINSEE_nt_type IS TABLE OF VARCHAR(13)
/
CREATE TYPE calendrier_vry_type IS VARRAY(365) OF DATE
/
```

## Recommandations

<i>Syntaxe</i>	<i>Nature</i>	<i>Exemple</i>
----------------	---------------	----------------

<b>xxx_type</b>	type abstrait
<b>ref_xxx</b>	colonne référence
<b>xxx_t</b>	colonne structurée

<b>avion_type</b>
<b>ref_comp</b>
<b>papiers_avion_t</b>

<b>xxx_elt_nt_type</b>	type des éléments	<b>derniers_vols_elt_nt_type</b>
<b>xxx_nt_type</b>	type de la <i>nested</i>	<b>derniers_vols_nt_type</b>
<b>xxx_nt</b>	collection <i>nested</i>	<b>derniers_vols_nt</b>
<b>xxx_tabnt</b>	table de stockage	<b>derniers_vols_tabnt</b>
<b>xxx_elt_vry_type</b>	type des éléments	<b>equipage_elt_vry_type</b>
<b>xxx_vry_type</b>	type du <i>varray</i>	<b>equipage_vry_type</b>
<b>xxx_vry</b>	collection <i>varray</i>	<b>equipage_vry</b>

## Manipulation des objets

## Instanciation

- **Objet non persistant**

```
DECLARE
  nonPersistant Pilote_type;
BEGIN
  nonPersistant := NEW Pilote_type('PL-11',
    etat_civil_type('Peyrard', '05-02-1970'),
    adresse_type(1, 'G. Brassens', 'Blagnac'), 3500);
```

- **Objet persistant (*row object, column object*)**

```
INSERT INTO Pilote
VALUES (Pilote_type('PL-11',
  etat_civil_type('Peyrard', '05-02-1970'),
  adresse_type(1, 'G. Brassens', 'Blagnac'), 3500));
```

## Modification / Suppression

- **Objet non persistant**

```
DECLARE
  nonPersistant Pilote_type;
BEGIN
  ...
  nonPersistant.adresse_t.ville := 'Castanet';
```

- **Objet persistant (*row object*)**

```
UPDATE Pilote p
SET p.etat_civil_t.ville = 'Castanet'
WHERE ...;

DELETE FROM Pilote p
WHERE p.brevet=...;
```

## Instanciation d'une référence

- **Opérateur REF(*alias*) pour relier un objet à un objet persistant**

```
INSERT INTO Compagnie
VALUES ('AF', Siege_social_type(124, 'Port Royal', 'Paris'),
  'Air France');

INSERT INTO Avion VALUES ('C-LARA', 'A320', 235,
  (SELECT REF(c) FROM Compagnie c WHERE c.comp='AF'));
```

## Instanciation d'une collection

- **Utilisation de constructeurs (éléments et collection)**

```
INSERT INTO departement
VALUES (departement_type('D1', 253000, employes_nt_type()));

INSERT INTO departement
VALUES (departement_type('D2', 2000, NULL));

INSERT INTO departement
VALUES (departement_type('D3', 3000,
  employes_nt_type(
    employe_elt_nt_type('I1', 34, 'Pascal'),
    employe_elt_nt_type('I2', 38, 'Agnès'),
    employe_elt_nt_type('I3', 18, 'Paul')
  ));
```



## Ajout d'un élément

- *Nested table* : directive TABLE (une seule collection doit être concernée)

```
INSERT INTO TABLE
(SELECT employes_nt FROM departement WHERE numdep = 'D1')
VALUES (employe_elt_nt_type('I4',42,'René'));
```

- *varray* : programmation PL/SQL



## Modification d'un élément

- *Nested table* : directive TABLE (une seule collection doit être concernée) + alias

```
UPDATE TABLE
(SELECT...
```

- *varray* : programmation PL/SQL



## Suppression d'un élément

- *Nested table* : directive TABLE (une seule collection doit être concernée) + alias

```
DELETE FROM TABLE
(SELECT...
```

- *varray* : programmation PL/SQL avec TRIM + programmation pour faire "remonter" les éléments restants.



## Interrogation des objets

## Extraction d'un objet persistant

- Objet persistant : comme en relationnel + notation pointée

Table Compagnie Type Compagnie\_type

comp	siege_social_t			nomComp
	nrue	rue	ville	
AF	124	Port Royal	Paris	Air France
SING	7	Camparols	Singapour	Singapore AL

## Extraction *via* une référence

- Référence : Alias nécessaire pour les jointures implicites

Avion

immat	typeAvion	capacite	ref_Compagnie
C-LARA	A320	235	
R-OMAN	A319	180	
F-WTSS	Concorde	90	
F-PAUL	A380	560	

Compagnie

comp	siege_social_t			nomComp
	nrue	rue	ville	
AF	124	Port Royal	Paris	Air France
SING	7	Camparols	Singapour	Singapore AL

## Opérateurs pour l'extraction

- `DEREF(reference)` extrait le contenu du type à l'adresse passée en paramètre  
Ex : Type de la compagnie propriétaire de l'avion F-WTSS
- `VALUE(alias)` s'applique à tout type de données et extrait les valeurs du type renvoyé  
Ex : Objet compagnie de nom 'Air France'
- `DANGLING` (renvoie TRUE si la référence est perdue)  
Ex : Immatriculation des avions qui ne sont rattachés à aucune compagnie

## Extraction d'éléments de collections

- Extraire d'une seule collection : `TABLE (SELECT...)`
- Extraire de plusieurs collections sous forme tabulaire : `TABLE(at.coll) ant`
- Extraire de plusieurs collections (résultat sous une forme « imbriqué ») : `CURSOR`

## Interrogation d'une seule collection

- Utilisation de : TABLE ( SELECT... )

Compagnie

comp	nomComp	{pilotes}		
		brevet	nom	age
ALIT	Air-Littoral	PL-1	Lamothe	36
		PL-2	Albaric	37
		PL-3	Peyrard	33
ABLA	Air-Blagnac	PL-6	Miranda	55
ACAS	CastaLines	PL-7	Payrissat	60
		PL-8	Bidal	38
		PL-9	Périssel	36

## Interrogation de plusieurs collections

- Résultats sous la forme tabulaire : TABLE(at.coll) ant
- Pseudo-jointure automatique

Compagnie

comp	nomComp	{pilotes}		
		brevet	nom	age
ALIT	Air-Littoral	PL-1	Lamothe	36
		PL-2	Albaric	37
		PL-3	Peyrard	33
ABLA	Air-Blagnac	PL-6	Miranda	55
ACAS	CastaLines	PL-7	Payrissat	60
		PL-8	Bidal	38
		PL-9	Périssel	36

## Interrogation de plusieurs collections

- Résultats sous une forme « imbriqué » : CURSOR

```
SELECT aliast.col1, CURSOR (SELECT aliascollection.col2, [CURSOR...]  
FROM TABLE(nomCollection) aliascollection  
WHERE aliascollection.col3 = valeur ...)  
FROM nomTable aliast  
WHERE aliast.col4 = valeur ... ;
```

## Fonctions pour les collections

- Fonctions
  - EXISTS (IF collection.EXISTS(i) THEN ...)
  - COUNT (FOR i IN 1..collection.COUNT LOOP ...) pour les *varray* COUNT = LAST
  - LIMIT (n := collection.LIMIT) (NULL pour *nested*, nombre max d'éléments qu'un *varray* peut contenir)
  - FIRST et LAST (i := collection.FIRST) retournent les 1er et dernier indices de la collection.  
Pour les *varray* FIRST=1 et LAST=COUNT. Pour les *nested*, FIRST<=LAST
  - PRIOR(x) et NEXT(x) retournent l'indice avant/après le x<sup>e</sup> élément de la collection
- Procédures
  - EXTEND / (i) / (i,j) pour ajouter un/des éléments à une collection
  - TRIM pour diminuer la taille de la collection
  - DELETE / (i) / (i,j) pour supprimer un/des éléments à une collection DELETE(i) pas possible pour *varray*

## Collections *varray*

- Directive `TABLE` inopérante : programmation PL/SQL pour la modification d'éléments

```
DECLARE
  variable_coll collection_type;
BEGIN
  SELECT collection INTO variable_coll
    FROM ... WHERE ...;

  variable_coll(i).FONCTION(PARAMETRES);

  UPDATE ...
    SET collection = variable_coll
    WHERE ...;
END;
/
```

## Exemples de fonctions

```
INSERT INTO departement
  VALUES (departement_type('D1'), employees_vry_type(
    employe_elt_vry_type('I1', 'Pascal'),
    employe_elt_vry_type('I2', 'Agnès'),
    employe_elt_vry_type('I3', 'Paul') ));
```

**VARRAY(5)**

```
DECLARE
  var_emp employees_vry_type;
BEGIN
  SELECT employees_vry INTO var_emp FROM departement WHERE numdep='D1';
  DBMS_OUTPUT.PUT_LINE('count : ' || TO_CHAR(var_emp.COUNT));
  var_emp.EXTEND(1);
  var_emp(4) := elt_employes_vry_type('I4', 'Fabrice');
  DBMS_OUTPUT.PUT_LINE('first/last : ' || TO_CHAR(var_emp.FIRST) || ' / '
    || TO_CHAR(var_emp.LAST));
END;
/
count : 3
first/last : 1 / 4

Procédure PL/SQL terminée avec succès.
```

## Les méthodes

## Méthodes

- Fonction ou procédure (PL/SQL, Java, C...)
- Surcharge possible
- Trois types
  - MEMBER (s'applique aux objets d'un table)
  - STATIC (s'applique au type seul)
  - CONSTRUCTOR (création d'instances d'objets non persistants)
- Appel
  - PL/SQL (bloc, procédure, fonction, méthode)
  - Requête `SELECT` (pour les méthodes de type fonction)
  - Programme C, Java...
  - Implicite à la création d'un objet

## Exemple d'une procédure MEMBER

Departement				
numdep	budget	(employees)		
		nmsee	age	nom
D1	253000	N1	26	Smith
D2	325000	N2	25	Jones
		N3	33	Codd
D3	120000			
		N5	35	Ellison
D4	125000	N6	32	Gates
		N7	33	Chen

- Augmenter le budget d'un département `augmenteBudg(n)`

```

...
CREATE TYPE departement_type AS OBJECT
  (numdep varchar2(11), budget number, employes emps_type,
  )
/
CREATE TABLE Departement OF departement_type
  (CONSTRAINT pk_dept PRIMARY KEY(numdep))
  NESTED TABLE employes STORE AS tabemp;

CREATE TYPE BODY departement_type

```

## Exemple d'une fonction MEMBER

Departement				
numdep	budget	(employees)		
		nmsee	age	nom
D1	253000	N1	26	Smith
D2	325000	N2	25	Jones
		N3	33	Codd
D3	120000			
		N5	35	Ellison
D4	125000	N6	32	Gates
		N7	33	Chen

- Compter le nombre d'employés d'un département `nbEmp()`

```

...
CREATE TYPE departement_type AS OBJECT
  (numdep varchar2(11), budget number, employes emps_type,
  )
/
CREATE TABLE Departement OF departement_type
  (CONSTRAINT pk_dept PRIMARY KEY(numdep))
  NESTED TABLE employes STORE AS tabemp;

CREATE TYPE BODY departement_type

```

## Invoquer une méthode MEMBER

- Principe : charger l'objet appelant de la table dans un objet temporaire par la fonction `VALUE` puis invoquer la méthode sur cet objet
- Appel d'une procédure: `obj.methode(paramètres);`
- Appels d'une fonction
  - `SELECT a.methode(paramètres) FROM table a ...`
  - `resultat = obj.methode(paramètres);`

## Exemples de méthodes STATIC

- `plusRicheDept()` département le mieux doté
- `ajouteDep(...)` création d'un département

# Appel d'une méthode STATIC

- Syntaxe

- `nom_type.methode(paramètres);`

# Méthode CONSTRUCTOR

```
ALTER TYPE stagiaire_type REPLACE AS OBJECT
(numero NUMBER, nom VARCHAR(40), datenaiss DATE,
MEMBER PROCEDURE change_nom(n IN VARCHAR),
STATIC PROCEDURE insere_st(p1 IN VARCHAR, p2 IN DATE),
CONSTRUCTOR FUNCTION stagiaire_type(p IN VARCHAR) RETURN SELF AS RESULT,
CONSTRUCTOR FUNCTION stagiaire_type(p IN DATE) RETURN SELF AS RESULT)
/
CREATE OR REPLACE TYPE BODY stagiaire_type AS
...
CONSTRUCTOR FUNCTION stagiaire_type(p IN VARCHAR) RETURN SELF AS RESULT IS
BEGIN
    SELF.numero := 1;
    SELF.nom := p;
    SELF.datenaiss := SYSDATE;
    RETURN;
END;
CONSTRUCTOR FUNCTION stagiaire_type(p IN DATE) RETURN SELF AS RESULT IS
BEGIN
    SELF.numero := 10;
    SELF.nom := 'Jean';
    SELF.datenaiss := p;
    RETURN;
END;
END;
/
```

# Appel d'une méthode CONSTRUCTOR

- Exemple

```
DECLARE
s1 stagiaire_type;
s2 stagiaire_type;
s3 stagiaire_type;
BEGIN
s1 := NEW stagiaire_type(0, 'Christian', '05-02-1965');
s2 := NEW stagiaire_type('Paul');
s3 := NEW stagiaire_type(TO_DATE('13-05-1995'));
END;
/
```

# Surcharge (*overloading*)

- Principe

- Codage différent de méthodes MEMBER
- Choix du code automatique à l'exécution

- Exemple

```
CREATE TYPE poste_travail_type AS OBJECT
(nserie VARCHAR(10), adrIP CHAR(5), dateachat DATE,
MEMBER PROCEDURE modif_poste(nouveau_nserie IN VARCHAR),
MEMBER PROCEDURE modif_poste(nouvelle_adr IN CHAR),
MEMBER PROCEDURE modif_poste(achat IN DATE),
MEMBER PROCEDURE modif_poste(num IN varchar2, achat IN DATE)
)
/
```

# Évolution de type

## ■ Principes

- Ajout/suppression/modification d'attributs
- Ajout/suppression de méthodes
- Modification de la nature du type FINAL, INSTANTIABLE
- Répercussion sur les objets des tables CASCADE, INVALIDATE...

# Évolution de type

## ■ Syntaxe de ALTER TYPE

```
ALTER TYPE xxx_type
  {COMPILE SPECIFICATION | BODY}
  {DROP colonne}
  {ADD / MODIFY} ATTRIBUTE colonne type
  {ADD / DROP} | methode(parametres) ;
```

## ■ Exemples

```
ALTER TYPE pil_type
  MODIFY ATTRIBUTE nom VARCHAR(35);
ALTER TYPE pil_type
  DROP ATTRIBUTE prime_hvol_supp;
ALTER TYPE pil_type
  ADD MEMBER PROCEDURE ajouter_vol(n IN NUMBER);
ALTER TYPE pil_type
  DROP MEMBER PROCEDURE affiche_pil;
ALTER TYPE pil_type
  COMPILE SPECIFICATION;
```

# Surcharge : OVERRIDING

## ■ Principe

- Redéfinition d'une méthode dans un sous-type

```
CREATE TYPE pil_type AS OBJECT
  (brevet VARCHAR(10), nom VARCHAR(30), nbhvol NUMBER)
  NOT INSTANTIABLE NOT FINAL
/
CREATE TYPE pil_Europe_type UNDER pil_type
  (prime_hvol_supp NUMBER(6,2),
   FINAL MEMBER PROCEDURE ajouter_vol(n IN NUMBER),
   NOT FINAL MEMBER PROCEDURE affiche_pil)
  NOT FINAL
/
CREATE TYPE pil_Français_type UNDER pil_Europe_type
  (syndicat VARCHAR(20),
   OVERRIDING NOT FINAL MEMBER PROCEDURE affiche_pil)
/
```

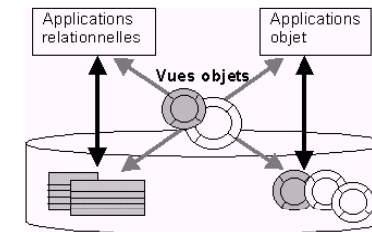
# Surcharge : OVERRIDING

```
CREATE OR REPLACE TYPE BODY pil_Europe_type AS
  FINAL MEMBER PROCEDURE ajouter_vol(n IN NUMBER) IS
  BEGIN
    nbhvol := nbhvol +n;
  END ajouter_vol;
  NOT FINAL MEMBER PROCEDURE affiche_pil IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(brevet || nom || TO_NUMBER(nbhvol+prime_hvol_supp));
  END affiche_pil;
END;
/
CREATE OR REPLACE TYPE BODY pil_Français_type AS
  OVERRIDING NOT FINAL MEMBER PROCEDURE affiche_pil IS
  BEGIN
    DBMS_OUTPUT.PUT_LINE(nom || 'Affilié à ' || syndicat ||
      TO_NUMBER(nbhvol*(1+(prime_hvol_supp/100))) );
  END affiche_pil;
END;
```

## Vues objet-relationnelles

## Généralités

- Migration en douceur vers la l'objet
  - OID avec la clause `WITH OBJECT OID`
  - Collections (`NESTED TABLE`, `VARRAY` avec `CAST` et `MULTISET`)
  - Pointeurs (`REF` avec `MAKE_REF`)



## Définition

- A partir
  - d'une ou plusieurs tables objet-relationnelles
  - d'une ou plusieurs tables relationnelles
  - d'une ou plusieurs vues objet-relationnelles
- Principe
  - Définir un type de données (`CREATE TYPE...`)
  - Écrire la requête (`AS SELECT...`) qui décrit les objets de la vue
  - Spécifier un identifiant (`OID`) basé sur un attribut de la vue pour pouvoir référencer chaque objet de la vue (`REFs`). Se servir de préférence d'une clé primaire
  - Programmer des déclencheurs `INSTEAD OF` (optionnel)

## Exemple 1 : vue monotable

```
SQL> SELECT * FROM pil_table;
      NUM NOM                SALAIRE JOB
-----
      1 San Filippo          45000 Pilote
      2 Sigaudes             40000 Copilote
      3 Soutou               35000 Copilote
      4 Miranda              25000 Pilote
      5 Dupond               15000 Stagiaire

CREATE TYPE copi_type AS OBJECT
  (empno NUMBER(5), nom VARCHAR2(20), paye NUMBER (9,2),
   fonction VARCHAR2 (20))
/
CREATE VIEW copi_vueOR OF copi_type
  WITH OBJECT IDENTIFIER (empno)
  AS SELECT p.*
     FROM   pil_table p
     WHERE  salaire > 30000;
```

## Exemple 1 : vue monotable

```
SQL> SELECT REF(c), c.* FROM copi_vueOR c;
```

```
REF(C)
```

```
-----  
EMPNO NOM                PAYE FONCTION  
-----  
00004A038A0046B1C34F72707545188B1B9B559443F7CE00000014260100010001002900000  
1 San Filippo            45000 Pilote  
00004A038A0046B1C34F72707545188B1B9B559443F7CE00000014260100010001002900000  
2 Sigaudes              40000 Copilote  
00004A038A0046B1C34F72707545188B1B9B559443F7CE00000014260100010001002900000  
3 Soutou                 35000 Copilote
```

```
SQL> INSERT INTO copi_vueOR
```

```
2 VALUES (6,'BigChief', 55000,'ChefPil');
```

```
1 ligne créée.
```

## Exemple 2 : structure

```
SQL> SELECT * FROM compagnie_table;
```

```
COMP NOM                RUE                VILLE    ETAT  
-----  
AF Air France           Maignon         Paris    FR  
ALIB Air Lib            Brassens        Blagnac  FR
```

```
CREATE TYPE adresse_comp_type AS OBJECT  
(rue VARCHAR2(20),ville VARCHAR2(10),etat CHAR(5))
```

```
/
```

```
CREATE TYPE compagnie_type AS OBJECT  
(numcomp CHAR(4), nomcomp VARCHAR2(20),  
siegesocial_t adresse_comp_type)
```

```
/
```

```
CREATE VIEW compagnie_vueOR OF compagnie_type  
WITH OBJECT IDENTIFIER (numcomp)  
AS SELECT c.comp, c.nom,  
adresse_comp_type(c.rue,c.ville,c.etat) AS siege  
FROM compagnie_table c;
```

## Exemple 2 : structure

```
SQL> SELECT REF(c), c.* FROM compagnie_vueOR c;
```

```
REF(C)
```

```
-----  
NUMC NOMCOMP  
-----  
SIEGESOCIAL_T(RUE, VILLE, ETAT)  
-----  
00003B038A003728FC2F14BAB6455A9335FECAAE77B95E00000017260100010001002900000...  
AF Air France  
ADRESSE_COMP_TYPE('Maignon', 'Paris', 'FR ' )  
  
00003B038A003728FC2F14BAB6455A9335FECAAE77B95E00000017260100010001002900000...  
ALIB Air Lib  
ADRESSE_COMP_TYPE('Brassens', 'Blagnac', 'FR ' )
```

## Exemple 3 : construction d'une collection

- CAST convertit une collection dans un autre type de collection
- Typage du résultat d'une requête (collection qui n'est pas nommée) ou d'une collection nommée (VARRAY ou NESTED TABLE)
- si le résultat de la requête retourne plusieurs enregistrements, il est alors nécessaire de coupler CAST avec MULTISSET

## Exemple 3 : construction d'une collection

```
SQL> SELECT * FROM compagnie;
```

```
COMP NOM_COMP
-----
AF Air France
TAT Transport Air Touraine
AL Air Libert 
AOM Air Outre-Mer
```

```
SQL> SELECT * FROM avion;
```

```
NA TYPE CAP COMP
-----
A1 T2 170 AF
A2 T4 140 TAT
A3 T1 400 AF
A4 T4 120 AF
A5 T3 200 TAT
```

## Exemple 3 : construction d'une collection

```
CREATE TYPE elt_nt_avion_type AS OBJECT
  (na VARCHAR(4),type VARCHAR(4), cap NUMBER(3))
/
CREATE TYPE avion_nt_type AS TABLE OF elt_nt_avion_type
/
CREATE TYPE compagnie_type AS OBJECT
  (compa VARCHAR(4), nom_comp VARCHAR(30), avion_nt avion_nt_type)
/

CREATE VIEW comp_vuerOR OF compagnie_type
  WITH OBJECT IDENTIFIER (compa)
  AS SELECT c.comp, c.nom_comp,
           CAST( MULTISET (SELECT a.na, a.type, a.cap
                           FROM avion a
                           WHERE a.comp = c.comp)
              AS avion_nt_type)
  AS flotte
  FROM compagnie c;
```

## Exemple 3 : construction d'une collection

```
SQL> SELECT * FROM comp_vuerOR;
```

```
COMP NOM_COMP
-----
AVION_NT(NA, TYPE, CAP)
-----
AF Air France
AVION_NT_TYPE(ELT_NT_AVION_TYPE('A1', 'T2', 170), ELT_NT_AVION_TYPE('A3', 'T1',
400), ELT_NT_AVION_TYPE('A4', 'T4', 120))

TAT Transport Air Touraine
AVION_NT_TYPE(ELT_NT_AVION_TYPE('A2', 'T4', 140), ELT_NT_AVION_TYPE('A5', 'T3',
200))

AL Air Libert 
AVION_NT_TYPE()

AOM Air Outre-Mer
AVION_NT_TYPE()
```

```
SQL> SELECT nt.* FROM TABLE (SELECT avion_nt FROM comp_vuerOR WHERE compa='AF') nt;
```

```
NA TYPE CAP
-----
A1 T2 170
A3 T1 400
A4 T4 120
```

## Exemple 4 : construction de r f rences

```
SQL> SELECT * FROM compagnie;
```

```
COMP NOM_COMP
-----
AF Air France
TAT Transport Air Touraine
AL Air Libert 
AOM Air Outre-Mer
```

```
SQL> SELECT * FROM avion;
```

```
NA TYPE CAP COMP
-----
A1 T2 170 AF
A2 T4 140 TAT
A3 T1 400 AF
A4 T4 120 AF
A5 T3 200 TAT
```

## Exemple 4 : construction de références

```
CREATE TYPE compagnie_type AS OBJECT
  (comp VARCHAR(4), nom_comp VARCHAR(30))
/
CREATE TYPE avion_type AS OBJECT
  (nav VARCHAR(4), type VARCHAR(4), cap NUMBER(3),
   ref_comp REF compagnie_type)
/

CREATE VIEW comp_vueOR
  OF compagnie_type
  WITH OBJECT IDENTIFIER(comp)
  AS SELECT * FROM compagnie;

CREATE VIEW avion_vueOR OF avion_type
  WITH OBJECT IDENTIFIER(nav)
  AS SELECT a.nav, a.type, a.cap, MAKE_REF(comp_vueOR, a.comp)
  FROM avion a;
```

## Exemple 4 : construction de références

```
SQL> SELECT a.nav, a.type, a.ref_comp.nom_comp FROM avion_vueOR a;

NAV  TYPE REF_COMP.NOM_COMP
-----
A1   T2   Air France
A2   T4   Transport Air Touraine
A3   T1   Air France
A4   T4   Air France
A5   T3   Transport Air Touraine

--autre solution sans MAKE REF

CREATE VIEW avion_vueOR2 OF avion_type
  WITH OBJECT IDENTIFIER(nav)
  AS SELECT a.nav, a.type, a.cap, REF(c)
  FROM avion a, comp_vueOR c
  WHERE a.comp = c.comp;
```