Les Objets dans SQL3

et dans Oracle et Informix

Didier DONSEZ

Université Joseph Fourier

IMA –IMAG/LSR/ADELE

Didier.Donsez@imag.fr,

Didier.Donsez@ieee.org

Attention

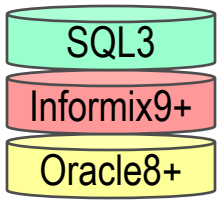
Ce cours sur SQL3 (SQL:1999) a été écrit en 1998 et n'a pas été mis au goût du jour (SQL:2003, Oracle 10, IBM DB2, …)

Il reste néanmoins une première approche de SQL3 que l'article suivant sur SQL:2003 complète

- Andrew Eisenberg and Jim Melton and Krishna G. Kulkarni and Jan-Eike Michels and Fred Zemke, SQL: 2003 has been published, SIGMOD Record, 33 (1) 2004, pages 119-126,
  - http://www.sigmod.org/sigmod/record/issues/0403/E.JimAndrew-standard.pdf

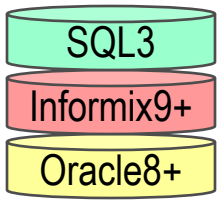
Sommaire

- Abstract Data Types
- Litéraux et Objets
- Méthodes et Surcharge
- Héritage, Sous-Type et Sous-Table
- Conversion
- Vue Objet-Relationnelle
- Extension de Types
- SQL3 et JDBC2.0



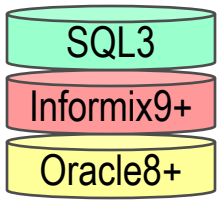
Motivations

- SQL3 étend le SQL aux concepts Objet.
  - Cependant la « relation » reste fondamentale dans la manipulation des données
- Les SGBDs basés sur SQL3 sont appelés **Objet-Relationnels**
  - Informix, Oracle, Sybase, IBM/DB2, CA-OpenIngres, PostGres ...
- Attention, peu respectent SQL3 !!!!



Principe

- Typage fort
  - Type = Données + Méthodes
- La création de type ne crée pas d ’objets
- Les objets d ’un type sont persistants que lors ils sont insérés dans des tables déclarées (CREATE TABLE)



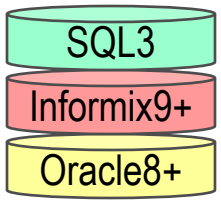
Les Types Atomiques (SQL-92)

- Build-In Type
  - SQL-92
    - DECIMAL, FLOAT, (NUMBER), … VARCHAR, DATE
- LOB Long OBject
  - CLOB (Character LOB)
  - BLOB (Binary LOB)
    - Chaîne d ’octets

      Le noyau du SGBD ne connaît pas la structure du BLOB

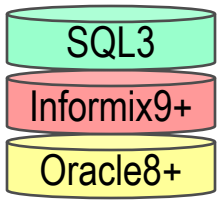
      Usage : Stockage Multimédia (MPEG2, MP3, …)
  - BFILE
    - Stockage externe

      SGF Multimédia (QoS, …)



UDT - User Defined Type

- Type Distinct
  - Type atomique avec typage fort
    - Conversion implicite et explicite, Fonction de conversion
- ADT Abstract Data Type
  - Type complexe
    - Attributs de type Atomique ou Complexe
    - Méthodes définies en PL/SQL ou un autre langage.
- 2 types d 'ADT
  - Type de Littéral
    - typage des colonnes d 'une table
  - Type d 'Objet
    - typage d 'un tuple-objet d 'une table



Type Distinct

```
CREATE DISTINCT TYPE frfranc_t AS DOUBLE PRECISION
CREATE DISTINCT TYPE euro_t AS DOUBLE PRECISION

CREATE TABLE PrixProduit2 (
    descrprod       VARCHAR(20),
    prixeuro        euro_t,
    prixfrfranc     frfranc_t
);

INSERT INTO PrixProduit2
VALUES ( 'yaourt', 3.00::euro_t, 12.50::frfranc_t)
```

SQL3

ADT Litéral - Exemple 1

- Type de Colonne

```
CREATE TYPE codpostal_t ( dept NUMBER(2), ext NUMBER(3) );
CREATE TYPE adresse_t (
        rue VARCHAR(20), ville        VARCHAR(20),      codpost codpostal_t
);
CREATE TABLE Person ( nom VARCHAR(20), adr adresse_t );
INSERT INTO Person VALUE ( ' Paul Dupont ',
        ROW(      ' 1 rue Danton ', 'Paris ',
                  ROW( 75, 1 )::codpostal_t
                )::adresse_t   );
SELECT p.nom, p.adr
FROM Person p
WHERE p.adr.codpost.dept = 26;
```

SQL3

ADT Litéral - Exemple 2

- Type de Colonne

```
CREATE TYPE doublePrix_t (
        eneuro          euro_t,
        enfranc         frfranc_t
);

CREATE TABLE Produit ( descr VARCHAR(20), prix doublePrix_t);

SELECT p.descr, p.prix.eneuro
FROM Produit p
WHERE p.prix.enfranc > 10;
```

SQL3

ADT d 'Objet - Exemple 1

- Type d 'Objets (Ligne d 'une table)

```
CREATE ROW TYPE person_t (
        nom     VARCHAR(20),
       adr      adresse_t
);

CREATE TABLE Person OF person_t (PRIMARY KEY nom);

SELECT p.nom, p.adr
FROM Person p
WHERE p.adr.codpost.dept = 59;
```

SQL3

ADT d 'Objet - Exemple 2

- Type d 'Objets (Ligne d 'une table)

```
CREATE ROW TYPE produit_t (
        descr VARCHAR(20),
        prix doublePrix_t
);

CREATE TABLE Produit OF produit_t (PRIMARY KEY descr);

SELECT p.descr, p.prix.eneuro
FROM Produit p
WHERE p.prix.enfranc > 10;
```

ADT Oracle - Litéral et Objet Exemple 1


```
CREATE TYPE person _t AS OBJECT (
                                        nom VARCHAR(20), adr adresse_t  );

CREATE TABLE Person OF person_t;

INSERT INTO Person VALUE ( ‘ Paul Dupont ’,
      adresse_t( ‘ 1 rue Danton ’, ’Paris ’, codpostal_t( 75, 1 ) ) );
```

Instanciation de valeurs d ’ADT

```
CREATE TABLE Employe( emp person_t , sal frfranc_t );

SELECT e.emp.nom, e.emp.adr    FROM Employe e
WHERE e.emp NOT IN (SELECT VALUE(p) FROM Person p);
```

ADT Oracle - Litéral et Objet Exemple 2


```
CREATE TYPE produit_t AS OBJECT (
      descr VARCHAR(20), prix doublePrix_t );

CREATE TABLE Produit OF produit_t;
CREATE TABLE TarifConcurrence(
      concurrent VARCHAR(20), produit produit_t );

SELECT     p.descr, t.concurrent
FROM       Produit p, TarifConcurrence t
WHERE      p.prix.enfranc > t.produit.prix.enfranc
     AND   p.descr = t.produit.descr
```

ADT Informix - Litéral et Objet Exemple 1

Informix9+

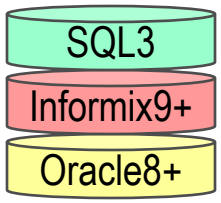
```
CREATE ROW TYPE person _t AS OBJECT (
        nom VARCHAR(20), adr adresse_t  );

CREATE TABLE Person OF TYPE person_t;
INSERT INTO Person VALUE ( ' Paul Dupont ',
          ROW(      ' 1 rue Danton ', 'Paris ',
                    ROW( 75, 1 )::codpostal_t )::adresse_t );
```

Instanciation de valeurs d 'ADT

```
CREATE TABLE Employe( emp person_t , sal frfranc_t );

SELECT e.emp.nom, e.emp.adr
FROM Employe e
WHERE e.emp NOT IN (SELECT * FROM Person);
```

ADT Informix - Litéral et Objet Exemple 2

```
CREATE ROW TYPE produit_t AS OBJECT (
      descr VARCHAR(20), prix doublePrix_t );

CREATE TABLE Produit OF TYPE produit_t;
CREATE TABLE TarifConcurrence(
      concurrent VARCHAR(20), produit produit_t );

SELECT     p.descr, t.concurrent
FROM       Produit p, TarifConcurrence t
WHERE      p.prix.enfranc > t.produit.prix.enfranc
      AND  p.descr = t.produit.descr
```

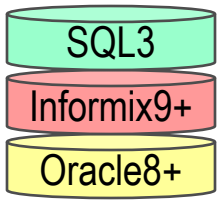


Compléments sur les ADT

- Pas de récursion de type
- Pas de valeur par défaut des champs
- Aucune contrainte CHECK
  - Elles sont déclarées dans les tables
- Contrainte NOT NULL
  - autorisée

```
CREATE TYPE codpostal_t (
        dept NUMBER(2) NOT NULL,    ext NUMBER(3) );
```

  - obligatoire dans les collections



Définition des Tables (i)

- Compatible SQL-92

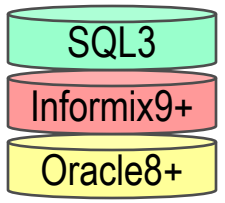
```
CREATE TABLE Employe(
        numemp INT, nom VARCHAR(20), sal DOUBLE PRECISION,
                PRIMARY KEY numemp, CHECK(sal>0) );
```

- Utilisation de Constructeurs

```
CREATE TABLE Employe(
        numemp INT,
        pers        ROW( nom VARCHAR(20), ville VARCHAR(20), dept NUMBER );
        postes      LIST( ROW(          intitule VARCHAR(10),
                                        debut DATE NOT NULL, fin DATE) ),
                PRIMARY KEY numemp, CHECK(pers.dept BETWEEN 1 AND 98)
);
```

Définition des Tables (ii)

- Utilisation des types

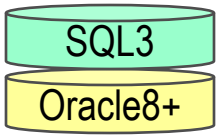


CREATE TABLE Employe(
numemp INT, pers person_t, sal frfranc_t
PRIMARY KEY numemp, CHECK(sal>0)
);

- Table d 'Objets

CREATE ROW TYPE employe _t
(numemp INT, pers person_t, sal frfranc_t);

CREATE TABLE Employe **OF** employe _t
(*PRIMARY KEY numemp, CHECK(sal>0)*);

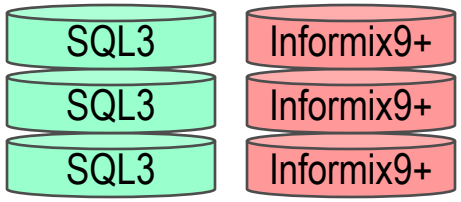


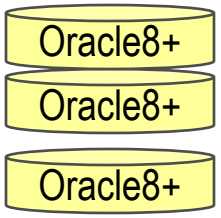
CREATE TABLE Employe **OF TYPE** employe _t
(*PRIMARY KEY numemp, CHECK(sal>0)*);

Constructeurs de Type

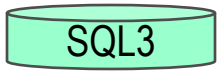
- Ligne ROW

  - plusieurs champs nommés de type différent
  - nombre fixe de champs
- Collections
  - plusieurs éléments de même type
  - avec/sans doublon, ordonné/sans ordre, fixé/variable
    - SET
    - LIST
    - MULTISET
    - VARRAY
    - Table imbriquée (Nested Table)

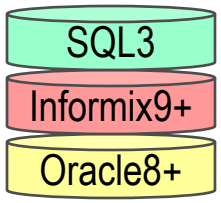




- Référence REF



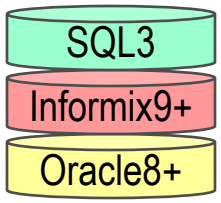
  - référence vers des objets de table
  - navigation par déréferenciation



Constructeur Ligne (ROW) Définition

- Structure
  - plusieurs champs nommés de type différent
  - nombre fixe de champs

```
CREATE TYPE person_t (
        nom                 VARCHAR(20),
        adresse             ROW(
                                rue         VARCHAR(40),
                                ville       VARCHAR(20),
                                ROW(
                                    dept    NUMBER(2),
                                    ext     NUMBER(3)
                                )
                            )
);
```



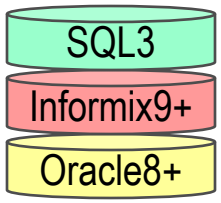
Constructeur Ligne (ROW) Instanciation et Accès

- Instanciation

```
CREATE TABLE Person OF person_t;

INSERT INTO Person VALUE
   ( ‘ Paul Dupont ’, ROW( ‘ 1 rue Danton ’, ’Paris ’, ROW( 75, 1 ) ) );
```

- Accès aux Champs
  - opérateur . (DOT)

```
SELECT p.nom, p.adresse.ville
FROM Person p
WHERE p.nom LIKE ‘ Pa%  ’ AND p.adresse.codpostal.dept = 75;
```



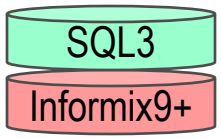
Constructeurs Collections

- plusieurs éléments de même type
- avec/sans doublon, ordonné/sans ordre, fixé/variable

Collection	Duplicata (Doublon)	Ordonné	Variable	Syst/Std
SET			✓	SQL3 Informix9+
MULTISET	✓		✓	SQL3 Informix9+
LIST	✓	✓	✓	SQL3 Informix9+
Nested Table	✓		✓	Oracle8+
VARRAY	✓	✓	fixé mais extensible	Oracle8+

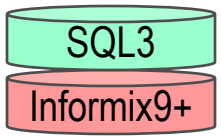
Remarque:

- une collection ne peut contenir de valeurs nulles
- ROW(NULL, NULL, ROW (NULL,NULL)) est nulle
- ROW('1 av Foch ', NULL,ROW (NULL,NULL)) n 'est pas nulle



Instanciation dans les Collections

```
CREATE TABLE CarnetAdresse (
        nom                     VARCHAR(20),
        adresse   s             LIST ( adresses_t ) NOT NULL,
        amis                    SET ( VARCHAR(20) ) NOT NULL
);

INSERT INTO CarnetAdresse VALUE (
    'Paul Dupont',
    LIST(
      ROW('1 rue Danton','Paris', ROW(75,1)::codpostal_t)::adresse_t,
      ROW(NULL,'Paris', ROW(75,15)::codpostal_t)::adresse_t
    ),
    SET( 'Pierre Durant ', ' Jacques Martin ')
);
```



Opérations sur les Collections

- Opérateurs IN et NOT IN *(en SQL)*

  SELECT nom FROM CarnetAdresse
  WHERE 'Jacques Martin' **IN** (amis)

- Opérateur CARDINALITY *(en SQL)*

  SELECT nom, **CARDINALITY**(adresses)
  FROM CarnetAdresse
  WHERE **CARDINALITY**(adresses) > 1;

- Autres Opérations
  - *possibles mais en SQL Procédural*

VARRAY dans Oracle

Oracle8+

- Ensemble ordonné et fixé d ’éléments de même type
  - mais la taille peut être étendue (méthode EXTEND(N))
- .COUNT =< taille fixée du VARRAY(tailledéclaration)
- autres méthodes ((), .FIRST, .LAST)

```
CREATE TYPE listAdresses_t AS VARRAY(4) OF adresse_t ;
DECLARE adressesPaul listAdresses_t ;
BEGIN
     adressesPaul = listAdresses_t(
       adresse_t( ‘1 rue Danton ’, ‘Paris’,  codpostal_t(75, 001)  ),
       adresse_t( ‘8 rue LouisXVI ’, ‘Paris’,  codpostal_t(75, 016)  )  );
For i IN 1.... adressesPaul.LAST LOOP
     INSERT INTO CarnetAdresse(‘Paul Dupont’, adressesPaul(i) );
END LOOP; END;
```

Tables Imbriquées (Nested Table)

Oracle8+

- Ensemble non ordonné et variable *(MultiSet)* d 'éléments de même ADT Objet

```
CREATE TYPE projet_t AS OBJECT(
    numproj INT, nomproj VARCHAR(20), budget DEC(10,2) ) ;
CREATE TYPE projets_t AS TABLE OF projet_t ;

CREATE TABLE Departement (
   numdept INT, nomdept VARCHAR(20),
   projets projets_t
) NESTED TABLE projets STORE AS TableDesProjets;
```

- Aucune opération directe sur la table **TableDesProjets**

Table Imbriquée - Manipulation (i)

Oracle8+

- TABLE() : Flattened Subquery

TABLE(SELECT d.projets FROM Department d WHERE d.numdept = 600 **)**

- le SELECT doit retourner un et un seul tuple collection
- **TABLE()** est anciennement nommé **THE()**

```
SELECT p.numproj, p.nomproj FROM
    TABLE( SELECT d.projets FROM Department d WHERE d.numdept = 600 ) p
                WHERE p.budget > 10000;
INSERT INTO
      TABLE( SELECT d.projets FROM Department d WHERE d.numdept = 600 )
                VALUES (projet_t(110, 'Téléportation', 1000000));
UPDATE
      TABLE( SELECT d.projets FROM Department d WHERE d.numdept = 600 )
                SET budget = 500000 WHERE numproj = 110;
```

Table Imbriquée - Manipulation (iii)

Oracle8+

- Nested Cursors

Liste des projets de plus d ’1M (avec les départements)

```
SELECT     numdept, nomdept,
           CURSOR(
                SELECT      p.numproj, p.nomproj, p.budget
                FROM        TABLE(Department.projets) p
                WHERE       p.budget > 1000000
           )
FROM Department ;
```

Department.projets : table projets dans le schéma Departement
TABLE(Department.projets) facilite la résolution de noms

Instanciation des Collections

Oracle8+

```
create type tel_t      as object ( num varchar2(10), poste varchar2(6) );
create type listtel_t  as varray(10) of tel_t;
```

- Constructeur

```
insert into ovPers values ( 1390120989, 'Dupont', 'Jean',
    listtel_t( tel_t('0327141234','8520 '), tel_t('0320445962',NULL) ) );
```

- Opérateur MULTISET( )
  - permet l 'instanciation d 'une collection (VARRAY, NT) à partir d 'une requête SELECT

```
select    numss, nom, prenom,
          cast( multiset( select tel_t(numtel,numposte)
                          from Tel t where t.numss=p.numss)
          as listtel_t) from Pers p;
```

Opérations sur les Collections (i)

- Remarque : Ce sont des méthodes !

En SQL

`.EXISTS(i)`

- Teste si le ième élément existe

`.COUNT()`

- Compte le nombre d'éléments non vide (inférieur ou égal à LAST())

`.LIMIT()`

- Retourne la taille maximale d'un VARRAY (NULL pour une nested table)

Opérations sur les Collections (ii)

Oracle8+

- En PL/SQL

`(i)`
  - ième élément (peut être vide)

`.FIRST(), .LAST()`
  - Indice du premier ou dernier élément (non vide).
  - NULL si la collection est vide

`.NEXT(i), .PRIOR(i)`
  - élément non vide suivant (précédent) l 'élément i

`.EXTEND, .EXTEND(n), .EXTEND(n,i)`
  - étend de 1 ou n éléments, avec des copies de l 'élément i

`.DELETE`
  - Supprime tous les éléments

`.DELETE(i), .DELETE(i,j)`
  - Supprime l'élément i / Supprime les éléments i à j

`.TRIM`
  - supprime l'élément à l'indice LAST()

`.TRIM(n)`
  - supprime les n derniers éléments

Exemple

Oracle8+

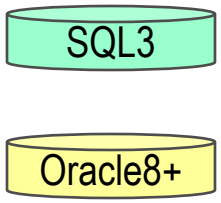
- Soit une collection

```
DECLARE C VARRAY(6) OF NUMBER;
C = (10,20,30,40,50,60);
    C.FIRST()=1, C.LAST()=6, C.LIMIT()=6, C.COUNT()=6
C.EXTEND(1);
    C = (10,20,30,40,50,60,NULL)
    C.LAST()=6, C.LIMIT()=7, C.COUNT()=6
C.EXTEND(3,2);
    C = (10,20,30,40,50,60,NULL,20,20,20)
    C.LAST()=10, C.LIMIT()=10, C.COUNT()=9
C.DELETE(1); C.DELETE(4);
    C = (NULL,20,30,NULL,50,60,NULL,20,20,20)
    C.FIRST()=2, C.NEXT(3)=5, C.PRIOR(5)=3, C.COUNT()=7
C.TRIM(8);
    C = (NULL,20)
```

Exemple de parcours d ’une collection

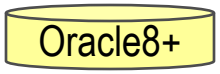
Oracle8+

```
DECLARE carnetAdressesDeJean listeadresses_t; i INTEGER;
BEGIN
carnetAdressesDeJean = listeadresses_t(
                        addresse_t(‘1 rue du Vignoble’, ‘Bordeaux’,  33000   ),
                        addresse_t(‘6 av de la Cave’,   ‘Reims’, 51000) );
i := carnetAdressesDeJean.FIRST;
While i <= carnetAdressesDeJean.LAST loop
            DBMS_OUTPUT.PUT(  ' Adresse ' || i || ': ');
            DBMS_OUTPUT.PUT_LINE(carnetAdressesDeJean(i));
            i := carnetAdressesDeJean.NEXT(i);
end loop;
For i IN 1.... carnetAdressesDeJean.LAST    loop
            INSERT INTO TabCarnetsAdresse(      ‘Jean Dupont ’,
                                                carnetAdressesDeJean(i) );
end loop;
END;
```



Références

- Notion d ’OID (Object IDentifier)
  - Immuable
  - Globalement Unique
  - Non réutilisable après destruction de l ’objet
  - *généré par le système (performance en navigation)*
- Chaque objet possède un OID
  - sauf si c ’est une colonne d ’une table



- Si l ’objet est de type T,
  sa référence est de type REF(T)
- Les références ont une valeur affichable
  - *ce qui n ’est pas le cas dans les SGBD-OO*

SQL3

Déclaration de Références

- Déclaration dans un type

```
CREATE TYPE person_t;
CREATE OR REPLACE BODY TYPE person_t (
      nom VARCHAR(20), prenom VARCHAR(10), nais DATE,
      conjoint          REF(person_t)
);
CREATE TABLE Person OF person_t
```

- Déclaration dans une table

```
CREATE TABLE Ami (
      ami1    REF(person_t),
      ami2    REF(person_t)
);
```

Déclaration de Références

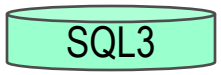
Oracle8+

- Déclaration dans un type

```
CREATE TYPE person_t;
CREATE OR REPLACE BODY TYPE person_t (
    nom VARCHAR(20), prenom VARCHAR(10), nais DATE,
    conjoint        REF     person_t
);
CREATE TABLE Person OF person_t;
```

- Déclaration dans une table

```
CREATE TABLE Ami (
    ami1    REF     person_t,
    ami2    REF     person_t
);
```



Limitation de la portée (SCOPE FOR)

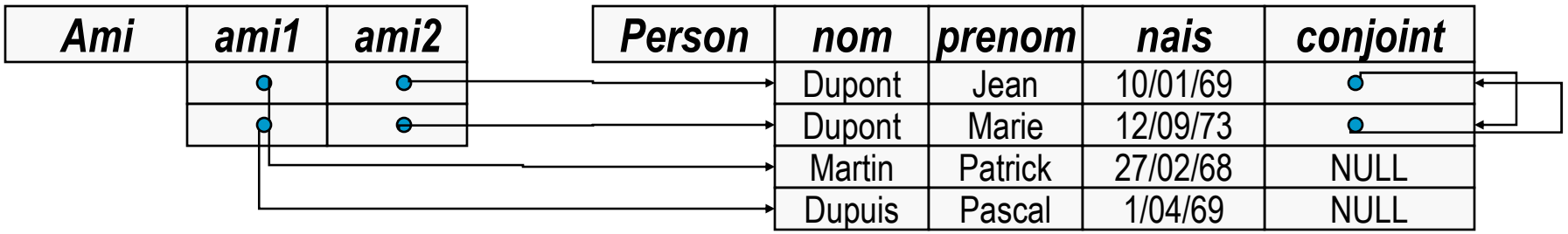
- Limite la portée de la référence à une table particulière
  - renforce l 'intégrité référentielle
- Exemple

```
CREATE TABLE Person2 OF person_t
            (SCOPE FOR conjoint IS Person) ;

CREATE TABLE Ami2 (
    ami1    REF     person_t,
    ami2    REF     person_t,
    SCOPE FOR ami1 IS Person,
    SCOPE FOR ami2 IS Person
);
```

SQL3

Oracle8+

Référence d ’Objets



SQL3

Oracle8+

Déréférenciation

- Opérateurs de Déréférenciation -> et .
  - R->A, R.A : accès à la valeur d 'un attribut A d 'un objet référencé par la référence R
- Exemple

```
SELECT a.ami2->conjoint->nom FROM Ami a
WHERE a.ami1->nom = 'Martin';
```

- équivalent à

```
SELECT x.nom FROM Ami a, Person x, Person y, Person z
WHERE     z.nom = 'Martin'
    AND   REF(z)=a.ami1
    AND   a.ami2=REF(y)
    AND   y.conjoint=REF(x);
```

Références dans ORACLE

- Déclaration

  DECLARE refpers REF person_t; en PL/SQL

- Référence d 'un objet REF()

  SELECT **REF(p)** INTO refpers FROM Person p
  WHERE p.nom='Martin ';

- Déréferenciation vers un objet DEREF()

  UPDATE Employe
  SET personne = **DEREF(refpers)**
  WHERE numemp = 1003;

- Test d 'Accessibilité des Références
  - IS [NOT] DANGLING

  SELECT e.nom,e.chef.nom,e.chef.age() FROM Employe e
  WHERE e.numemp = 1003 AND e.chef **IS NOT DANGLING**;

Références dans ORACLE

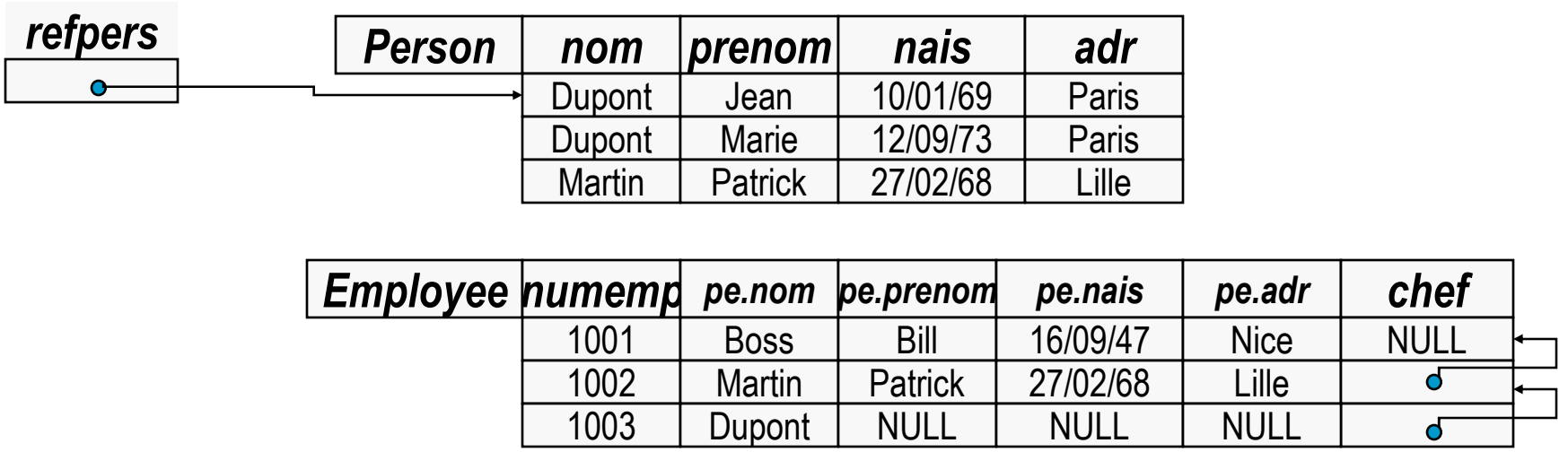
Oracle8+

Déclaration

DECLARE refpers REF person_t; en PL/SQL

Référence d 'un objet REF()

SELECT **REF(p)** INTO refpers FROM Person p
WHERE p.nom='Dupont ' AND p.prenom=' Jean ';

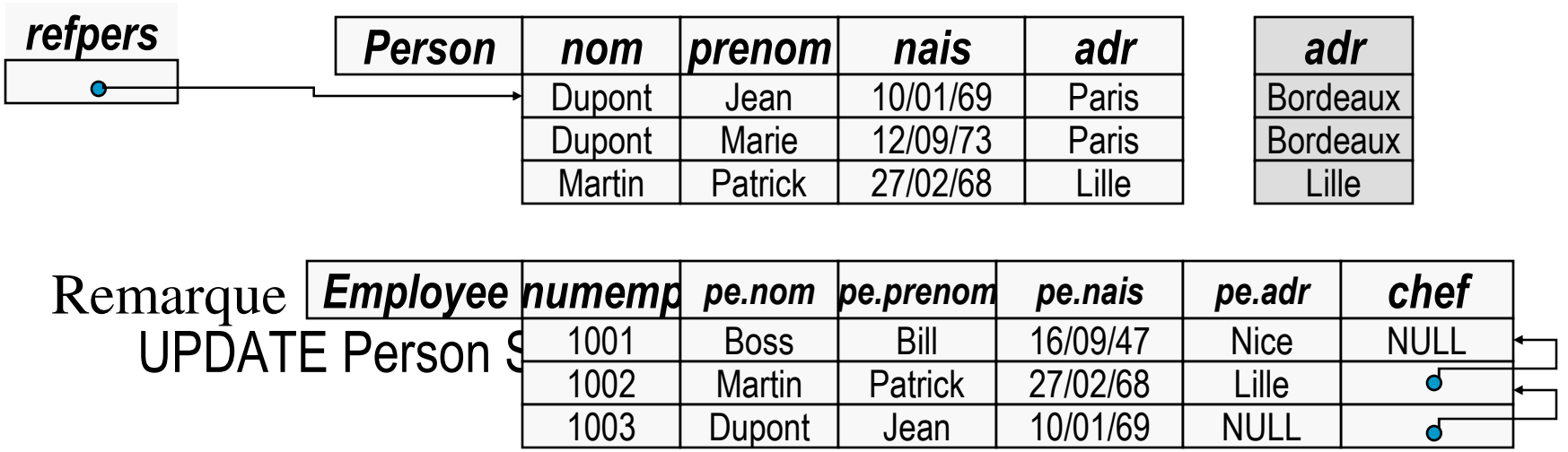


Déréferenciation

Oracle8+

DEREF() : Déréferenciation vers un objet

```
UPDATE Employe SET pe = DEREF(refpers)
WHERE numemp = 1003;
```



Remarque

```
UPDATE Person S
```

Employee	numemp	pe.nom	pe.prenom	pe.nais	pe.adr	chef
	1001	Boss	Bill	16/09/47	Nice	NULL
	1002	Martin	Patrick	27/02/68	Lille	
	1003	Dupont	Jean	10/01/69	NULL	

Références Pendantes (Dangling)

Oracle8+

Références sur un objet n 'existant plus

Exemple

```
SELECT e.pe.nom,e.chef.pe.nom,e.chef.pe.age() FROM Employe e
WHERE e.numemp = 1003;
```

Employee	numemp	pe.nom	pe.prenom	pe.nais	pe.adr	chef
	1001	Boss	Bill	16/09/47	Nice	NULL
	1003	Dupont	Jean	10/01/69	NULL	

Test d 'Accessibil

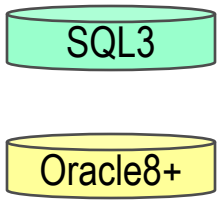
IS [NOT] DANGLING

```
SELECT e.pe.nom,e.chef.pe.nom,e.chef.pe.age()
FROM Employe e
WHERE e.numemp = 1003 AND e.chef IS NOT DANGLING;
```

Références dans Informix

- Présentes
  - colonne virtuelle ROWID
- MAIS pas de déclaration, ni déréférenciation
- Usage : comparaison de lignes

```
SELECT x.rowid, x.numclient, y.rowid
FROM Client x, Client y
WHERE x. numclient = y. numclient
AND x.rowid != y.rowid
```

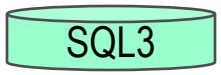


Fonctions et Procédures d 'ADT

- Encapsulation
  - « méthodes » = fonctions et procédures membres
- Déclaration
  - à la création ou à la modification du type

```
CREATE TYPE nomtype (
        attributs
        déclarations ou définitions des méthodes
    );
```

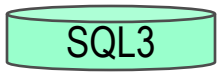
- Définition
  - à la création ou à la modification du type
    - les méthodes sont définies dans un langage procédural interne (PL/SQL) ou externe (C, C++)



Déclaration et Définitions des fonctions membres (i)

```
CREATE TYPE Vendeur_t ( num NUMBER, nom CHAR(20),
   FUNCTION nouvvendeur( :num CHAR(20), :nom DATE)
   RETURNS Vendeur_t;
      :v Vendeur_t ; /* local decl. */
   BEGIN
      :v := Vendeur_t (); /* built-in constructor */
      :v.num := :num; :v.nom := :nom;
      RETURN :v;
   END;
   FUNCTION chiffreVente(:v Vendeur_t ) RETURNS FLOAT; );
```

- nouvvendeur() est une définition()
- chiffreVente() est seulement une déclaration
  - cette méthode doit être définie quelque part



Déclaration et Définitions des fonctions membres (ii)

```
FUNCTION chiffreVente(:v Vendeur_t ) RETURNS FLOAT;
   :r FLOAT;
BEGIN
   SELECT SUM(qte*prixunit) INTO :r
   FROM Vente WHERE numvendeur = :v.num;
   RETURN :r;
END;
```

Déclaration et Définitions des fonctions membres (i)

Oracle8+

- Définition

```
CREATE TYPE Vendeur_t AS OBJECT (
                num NUMBER, nom CHAR(20),
                MEMBER FUNCTION nouvvendeur(
                num IN CHAR(20), nom IN DATE) RETURN Vendeur_t
    );
```

- Déclaration

```
CREATE OR REPLACE TYPE BODY Vendeur_t AS
MEMBER FUNCTION nouvvendeur( num IN CHAR(20), nom IN DATE)
      RETURN Vendeur_t RETURN Vendeur_t IS
   DECLARE          v Vendeur_t ;
   BEGIN            v.num := num; v.nom := nom; RETURN v;
   END;
END;
```

Déclaration et Définitions des fonctions membres (ii)

Oracle8+

- Modification

```
ALTER TYPE Vendeur_t REPLACE AS OBJECT (
    num NUMBER, nom CHAR(20),
    MEMBER FUNCTION nouvvendeur(
                num IN CHAR(20), nom IN DATE) RETURN Vendeur_t ,
    MEMBER FUNCTION chiffreVente() RETURN NUMBER,
);
CREATE OR REPLACE TYPE BODY Vendeur_t AS
MEMBER FUNCTION chiffreVente RETURN FLOAT IS
    DECLARE  r NUMBER;
    BEGIN
                SELECT SUM(qte*prixunit) INTO r
                FROM Vente WHERE numvendeur = self.num;
                RETURN r;
    END;        END;
```

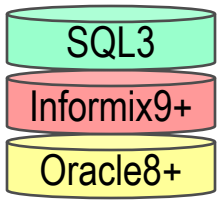
Surcharge des fonctions membres

Oracle8+

```
CREATE TYPE Person_t AS OBJECT(
    name VARCHAR2(100),
    dob DATE,
    MEMBER FUNCTION age() RETURN number,
    MEMBER FUNCTION print() RETURN varchar2
) NOT FINAL;

CREATE TYPE Employee_t UNDER Person_t(
    salary NUMBER,
    bonus NUMBER,
    MEMBER FUNCTION wages() RETURN number,
    OVERRIDING MEMBER FUNCTION print() RETURN varchar2
);
```

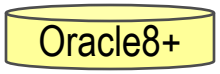
Fontions et Procédures d 'ADT dans Informix

- Pas de « méthodes »
  - ADT en argument des fonctions et des procédures
  - Surcharge



Fonctions « BuildIn » d 'ADT

- Fonctions de Comparaison (Built in)
  - EQUAL, LESSTHAN peuvent être déclarées
  - ces fonctions sont invoquées lors des comparaisons
    dans la clause WHERE =, <, =<, …, < ANY, > ALL, ...
  - ORDER retourne -1 (<), 0 (=), 1 (>)



- Ordre
  - RELATIVE
  - HASH
  - fonction de hachage MAP
    - donne un NUMBER qui établit un ordre relatif e… …stances de l 'ADT

- Conversion CAST

SQL3

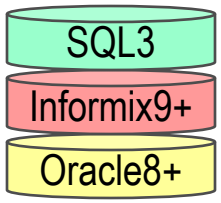
Fonctions de Comparaison

```
FUNCTION LESSTHAN( :v1 Vendeur_t, :v2 Vendeur_t )
    RETURNS BOOLEAN;
  BEGIN              IF chiffreVente(:v1) > chiffreVente(:v2)
                     RETURN FALSE ELSE RETURN TRUE;
  END;
FUNCTION EQUAL ( :v1 Vendeur_t, :v2 Vendeur_t )
    RETURNS BOOLEAN;
  BEGIN              IF chiffreVente(:v1) = chiffreVente(:v2)
                     RETURN TRUE ELSE RETURN FALSE;
  END;

SELECT v.num FROM Vendeur v
  WHERE v >= ALL Vendeur;
```

Fonctions de Comparaison

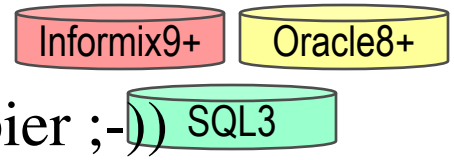
Oracle8+

```
FUNCTION ORDER( v1 Vendeur_t, v2 Vendeur_t )
    RETURNS NUMBER;
  DECLARE        cv1, cv2 : FLOAT;
  BEGIN   cv1:=v1.chiffreVente();
          cv2:=v2.chiffreVente()
          IF cv1 > cv2      THEN  RETURN      1
          ELSIF cv1 = cv2 THEN  RETURN      0
          ELSE                  RETURN      -1;
  END;

SELECT v.num FROM Vendeur v
  WHERE v >= ALL Vendeur;
```

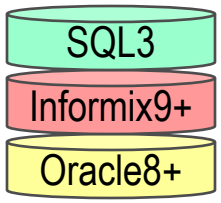


Héritage

- Héritage Simple vs Multiple
  - simple pour Oracle8+ et Informix9+
  - multiple spécifié dans SQL3 (qui est du papier ;-))



- Héritage de Type
  - Un type T peut avoir 1 (ou plusieurs) super-type ST
- Héritages de Table
  - Une table R peut avoir 1 super-table SR



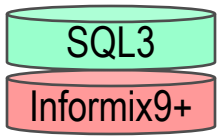
Héritage de Type

```
CREATE ROW TYPE person_t
    (nom CHAR(20), sexe CHAR(1),age INT);

CREATE ROW TYPE employe_t UNDER person_t
    (salaire FLOAT);

CREATE ROW TYPE manager_t UNDER employe_t
    (bonus FLOAT);

CREATE ROW TYPE client_t UNDER person_t
    (compte FLOAT);
```

Héritage de Type

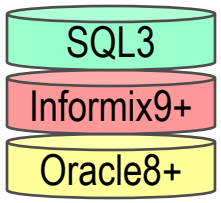
Oracle8+

```
CREATE ROW TYPE person_t
     (nom CHAR(20), sexe CHAR(1),age INT)
NOT INSTANTIABLE          -- pas d'instance du type
NOT FINAL;                -- sous-typage possible

CREATE ROW TYPE employe_t UNDER person_t
     (salaire FLOAT);

CREATE ROW TYPE client_t UNDER person_t
     (compte FLOAT)
FINAL;                    -- pas de sous-typage possible
```



Héritage de Table

```
CREATE TABLE Person
    (nom CHAR(20), sexe CHAR(1),age INT);

CREATE TABLE Employe UNDER Person
    (salaire FLOAT);

CREATE TABLE Manager UNDER Employe
    (bonus FLOAT);

CREATE TABLE Client UNDER Person
    (compte FLOAT);
```



Héritage des propriétés des tables

- Clause ADD/DROP/MODIFY
  - Attributs
  - Fonctions et Procédures
    - Surcharge des fonctions et procédures
  - Contraintes
    - pas de retrait, mais ajout possible
  - Triggers
    - pas de retrait, mais « surcharge » possible
  - Index
    - pas de retrait, mais ajout possible

Informix9+

Relation entre héritage de table et héritage de type

- Contrainte sur l 'héritage de table

  CREATE TABLE ***R*** OF TYPE ***T*** UNDER ***ST***;

  - le type T de table R doit avoir pour super-type ST,
    le type de la super-table SR de R

```
CREATE TABLE Person     OF TYPE person_t;
CREATE TABLE Employe    OF TYPE employe_t    UNDER Person;
CREATE TABLE Manager    OF TYPE manager_t    UNDER Employe;
CREATE TABLE Client     OF TYPE client_t     UNDER Person;
```

Informix9+ Consultation de l 'héritage de table

SELECT * FROM ONLY(Person);

- ONLY réduit la portée du SELECT/UPDATE dans un héritage de tables

SELECT * FROM Person;

SELECT Person FROM Person;

Person	Nom	Sexe	Age
	Paul	M	12
	Pierre	M	34

Employe	Nom	Sexe	Age	Salaire
	Jacques	M	45	10000
	Anne	F	56	90000

Client	Nom	Sexe	Age	Compte
	Henri	M	21	2000
	Marie	F	65	100000

Manager	Nom	Sexe	Age	Salaire	Bonus
	Bill	M	36	700000	100000

Opération sur les types

Oracle8+

- IS OF(*type1,type2, …*)
  - Test d'appartenance à un type ou à un sous-type

```
SELECT VALUE(c) FROM Company c
WHERE c.employee IS OF(manager_t, engineer_t);
```

- IS OF(ONLY *type1,type2, …*)
  - Test d'appartenance strict à un type

```
SELECT VALUE(c) FROM Company c
WHERE c.employee IS OF(manager_t); --REM exclut les director_t
```

- TREAT(*col_supertype* AS *type*)
  - coercition vers un sous type

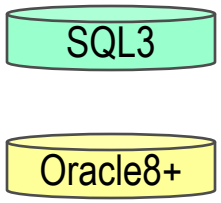
```
SELECT TREAT(c.employee AS manager_t)  FROM Company c;
-- seuls les rows contenant des manager_t sont affichées
```

- SYS_TYPEID(*type*)
  - Retourne un numéro entier de type dans une hiérarchie de type

```
SELECT SYS_TYPEID(c.employee)  FROM Company c;
```

Surcharge de Fonction et de Procédure

```
CREATE FUNCTION CalculeSalaire(e employe_t)
   RETURNING integer
   RETURN e.salaire;
END FUNCTION;

CREATE FUNCTION CalculeSalaire(m manager_t)
   RETURNING integer
   RETURN m.bonus + CalculeSalaire(m::employe_t);
END FUNCTION;

SELECT MOY(CalculeSalaire(e)) FROM Employe e; # 300000
```

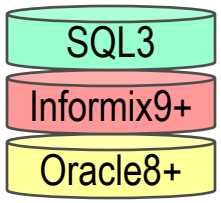


Accès aux « Membres » d ’un ADT

- Protection de l ’accès des attributs et des méthodes d ’un ADT

3 niveaux

- PUBLIC (par défaut)
  - accessible par tous
- PROTECTED
  - accessible que par les méthodes des types dérivés
- PRIVATE
  - accessible que par les méthodes de l ’ADT



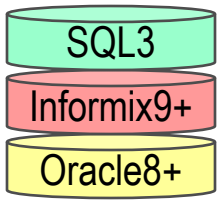
Conversion (CAST)

- Typage fort
  - conversion de type
    - explicite / implicite
- Opérateur de Conversion
  - CAST(attribut AS type)

```
SELECT titre
FROM projet
WHERE CAST(ecrivain AS editeur_t) = editeur
```

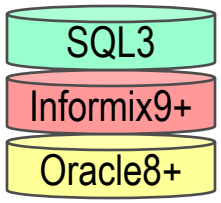
  - Opérateur de conversion ::

```
SELECT titre
FROM projet
WHERE ecrivain:: editeur_t = editeur
```



Conversion

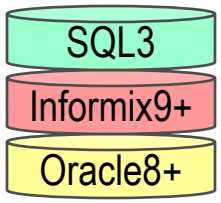
```
CREATE ROW TYPE info_t (x BOOLEAN, y BOOLEAN)
CREATE TABLE customer (cust_info info_t)
CREATE TABLE retailer (ret_info ROW (a CHAR(1), b CHAR(1)))

SELECT cust_info
FROM customer, retailer
WHERE cust_info = ret_info::info_t

SELECT cust_info
FROM customer, retailer
WHERE cust_info::ROW(a CHAR(1), b CHAR(1)) = ret_info
```



Conversion automatique

- Pour les types « BuildIn »,
  certaines conversion sont automatiques

```
CREATE TABLE prices (
    col1 ROW(a SMALLINT, b FLOAT)
    col2 ROW(x INT, y REAL)
);
SELECT * FROM prices WHERE col1 = col2
        Conversion automatique de SMALLINT vers INT
        Conversion automatique de REAL vers FLOAT
SELECT * FROM prices
WHERE ROW(col1.a::INT,col1.b) = ROW(col2.x, col2.y::FLOAT)
```

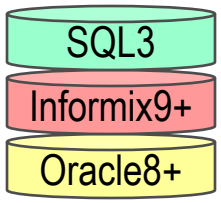


Conversion de Types Distincts (i)

```
CREATE DISTINCT TYPE frfranc_t AS DOUBLE PRECISION
CREATE DISTINCT TYPE euro_t AS DOUBLE PRECISION
CREATE TABLE PrixProduit(prix ROW(d VARCHAR(20), f frfranc_t))
```

Conversion Explicite

```
INSERT INTO PrixProduit VALUES(ROW( 'yaourt', 12.50::frfranc_t))
```

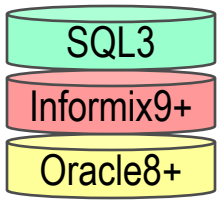


Conversion de Types Distincts (ii)

```
CREATE DISTINCT TYPE frfranc_t AS DOUBLE PRECISION
CREATE DISTINCT TYPE euro_t AS DOUBLE PRECISION
CREATE TABLE PrixProduit(prix ROW(d VARCHAR(20), f frfranc_t))
```

Conversion Implicite

```
CREATE CAST (frfranc_t AS euro_t)
SELECT prix::ROW(d VARCHAR(20), f euro_t) FROM PrixProduit

CREATE TABLE PrixProduit2 (
  descrprod VARCHAR(20), prixeuro euro_t, prixfrfranc frfranc_t );

INSERT INTO PrixProduit2
VALUES ( 'yaourt', 3.00::euro_t, 12.50::frfranc_t)
```



Fonctions de Conversion

```
CREATE TABLE PrixProduit2 (          descrprod VARCHAR(20),
                                     prixeuro euro_t,
                                     prixfrfranc frfranc_t
);
```

Fonction de Conversion

```
CREATE FUNCTION frfranc_to_euro(f frfranc_t)        RETURNS euro_t
    RETURN CAST((f::DOUBLE PRECISION / 6.33) AS euro_t);
END FUNCTION;
```

Déclaration

```
CREATE CAST(frfranc_t AS euro_t WITH frfranc_to_euro);
```

Usage

```
SELECT CAST(prixfrfranc AS euro_t ), prixeuro
FROM PrixProduit2
WHERE CAST(prixfrfrancs AS euro_t ) < prixeuro
```

SQL3

Oracle8+

Object View

- Motivation
  - Objets métiers, récupération de l 'existant, ...
- Exemple

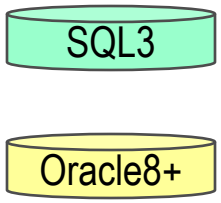
Vue Objet de la Base Relationnelle

ovPers	numss	nom	prenom	listtel
	136123	Dupont	Jean	(0327141234,8520),(0320445962,NULL)
	25688	Durant	Marie	NULL

Base Relationnelle

Pers	numss	nom	prenom
	136123	Dupont	Jean
	25688	Durant	Marie

Tel	numss	numtel	numposte
	136123	0327141234	8520
	136123	0320445962	NULL



Exemple de vue OR

- Base relationnelle

```
create table Pers( numss number, nom varchar2(10), prenom varchar2(10) );
create table Tel( numss number, numtel varchar2(10), numposte varchar2(6) );
```

- Vue Objet

```
create type tel_t          as object ( num varchar2(10), poste varchar2(6) );
create type listtel_t      as varray(10) of tel_t;
create type ovPers_t as object (
    numss number, nom varchar2(10), prenom varchar2(10), listtel listtel_t);
create view ovPers of ovPers_t with objectoid(numss) as
    select      numss, nom, prenom,
                cast( multiset(     select tel_t(numtel,numposte)
                                    from Tel t where t.numss=p.numss) as listtel_t)
    from Pers p;
```

SQL3

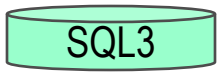
Oracle8+

Modification au travers d 'une vue OR

- Trigger INSTEAD OF
  - permet les modifications sur une vue OR
- Syntaxe

```
CREATE TRIGGER nom_du_trigger
        INSTEAD OF    type_ordre
                ON nom_de_la_table
FOR EACH ROW
        action
```

  - La modification (INSERT, UPDATE, DELETE)sur la **vue** est substituée par l 'action
  - les variables :old et :new sont utilisées dans l 'action comme si l 'événement avait lieu



Création de Trigger INSTEAD OF pour les modifications sur une vue OR

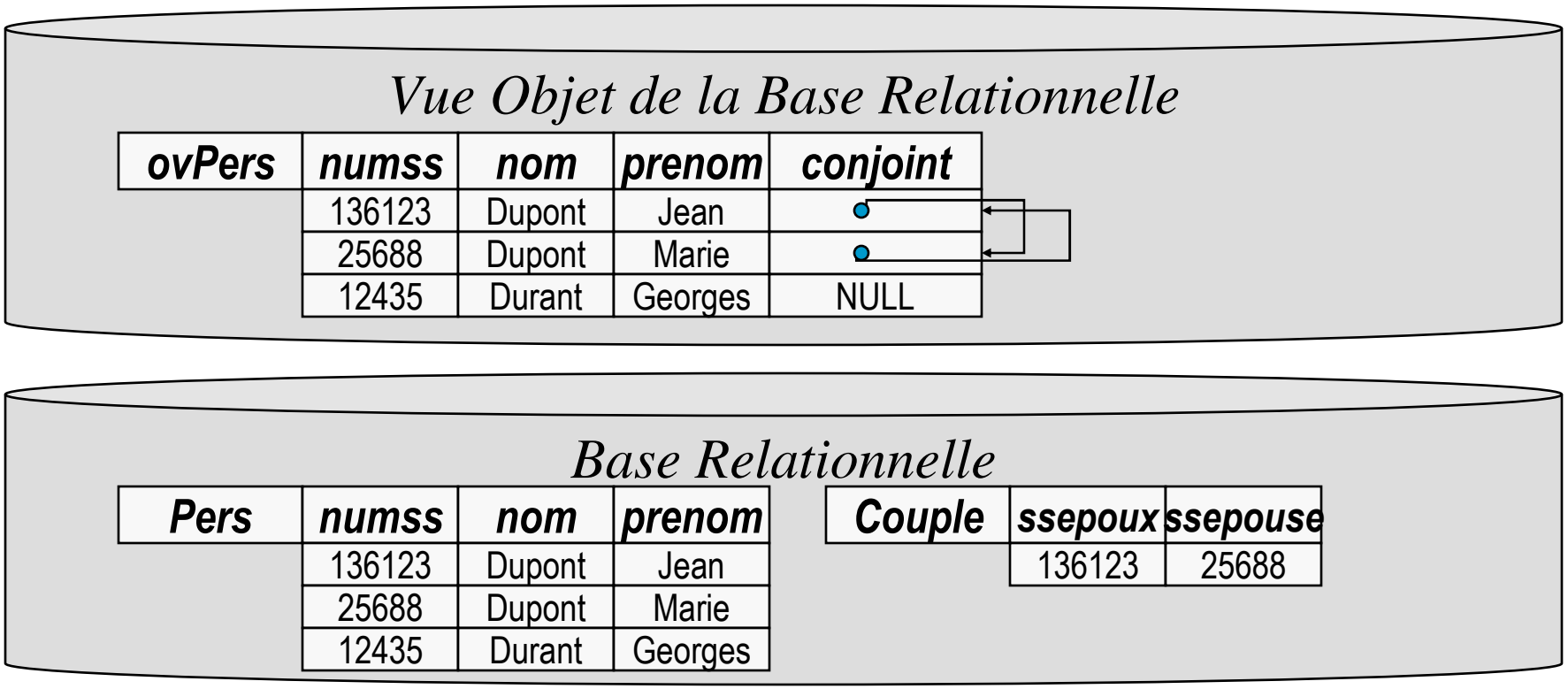
```
create trigger tg_ins_ovPers instead of insert on ovPers
for each row declare i: integer;
begin
    insert into Pers values(:new.numss, :new.nom, :new.prenom);
    if :new.listtel is not null and :new.listtel.count > O then
      for i in :new.listtel.first  … :new.listtel.last loop
      insert into Tel
                values (:new.numss, :new.listtel(i).tel, :new.listtel(i).poste);
      end loop;
    end if; end;
> insert into ovPers values ( 1390120989, 'Dupont', 'Jean',
      listtel_t( tel_t('0327141234','8520 '), tel_t('0320445962',NULL) ) );
```

SQL3

Oracle8+

Références dans une vue OR (i)

- Motivation
  - Navigation entre les « objets » de la vue
- Exemple



Références dans une vue OR (ii)

Oracle8+

Construction de référence à partir des clés

- Déclaration de l 'OID

  create view nomvueor **of** typeor **with objectoid(** clé **)** as ...

- Exemple

```
create table Pers( numss number, nom varchar2(10), prenom varchar2(10),
                   primary key(numss) );
create table Couple( ssepoux number, ssepouse number
                   primary key(ssepoux,ssepouse),
                   foreign key (ssepoux) references Pers,
                   foreign key (ssepouse) references Pers );

create view ovPers of ovPers_t with objectoid(numss) as ...
```

Références dans une vue OR (iii)

Oracle8+

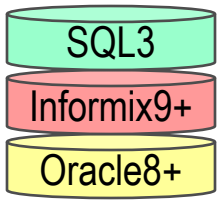
- MAKEREF(*attribut clé*)
  - Construction d 'un OID pour les « objets » de la vue à partir des clés primaires des tables

```
create view ovPers of ovPers_t with objectoid(numss) as
((select        p.numss, p.nom, p.prenom, MAKEREF(c.ssepouse)
      from      Pers p, Couple c where      p.numss=c.ssepoux)
      union
(select         p.numss, p.nom, p.prenom, MAKEREF(c.ssepoux)
      from      Pers p, Couple c where      p.numss=c.ssepouse))
      union
(select         p.numss, p.nom, p.prenom, NULL
      from      Pers p
      where     p.numss not in ((select ssepoux from Couple)
                                union (select ssepouse from Couple))
);
```

Références dans une vue OR (iii)

Oracle8+

```
create trigger tg_ins_ovPers instead of insert on ovPers for each row
begin
   insert into Pers    values(
                                :new.numss,
                                :new.nom,
                                :new.prenom
   );
   insert into Couple values(
                                :new.numss,
                                DEREF(:new.conjoint).numss
   );
end;
```



Gestion des Types et des Tables

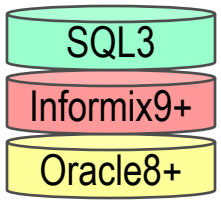
- CREATE
  - déclaration et définition de type et de table
- CREATE OR REPLACE
  - définition de type ou de membre de type
- ALTER
  - modification d ’une table
- DROP
  - suppression d ’une table ou d ’un type
  - à condition qu ’il n ’est ai pas de dépendance
    - dépendance d ’héritage, d ’ agrégation, de référence

Dictionnaires d'ORACLE

- Tables de la métabase

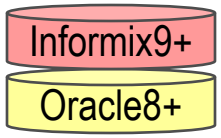
TABS	*décrit les tables*
COLUMNS	*décrit les colonnes*
USER_CONSTRAINTS	*décrit les contraintes par table*
USER_CONS_COLUMNS	*décrit contraintes par colonne*
USER_OBJECT_TABLES	décrit les tables objets
USER_TYPES	décrit les types
USER_TYPES_ATTRS	décrit les attributs des types
USER_TYPES_METHODS	décrit les méthodes des types
USER_TYPES_VERSIONS	décrit les versions des types
USER_DEPENDENCIES	décrit les dépendances entre types
USER_COLL_TYPES	décrit les collections
USER_INDEX_TYPES	décrit les index sur les types

- Usage
  - SQL> DESC USER_TYPES
  - SQL> SELECT * FROM USER TYPES



Contrôle d ’Accès

- Les types, les tables et les vues
- ne peuvent être utilisées
- que par les utilisateurs qui en droit
  - GRANT et REVOKE



Les extensions de Types

- Les SGBD-ORs offrent des mécanismes pour ajouter de nouveaux types atomiques
  - Sons, Vidéo, Image, Données Spatiales, Séries Temporelles, Séquences d ’ADN, Empreintes digitales, Fonds d ’œil …
- Ces types sont dit OPAQUES
  - **quand** le système n ’a aucun connaissance de la structure interne des valeurs de ces types
- Ces types et leurs méthodes peuvent être fournis par des sociétés tiers sous la forme de
  - Data Cartridges pour Oracle
  - Data Blades pour Informix
  - Extenders pour IBM/DB2
  - Pluggins pour Sybase

Exemple d ’extension de type

```
-- Bienvenu à GATACA
create table SUSPECTS (     NOM             VARCHAR (20),
                            GENE            VARGENOMIC(1000000000)
);
create table INDICES (      NUM_AFFAIRE     INTEGER,
                            LIEU            VARCHAR(30),
                            EMPREINTE       VARGENOMIC(1000000000)
);

Select S.NOM AS COUPABLE
From SUSPECTS S, INDICES I
Where I.EMPREINTE Similar S.GENE;
```

Les Données multimédia

- Applications
  - Infini !
- Type
  - Non temporelle
    - Images
  - Temporelle
    - Audio, Vidéo, Sous-titrage
- Opérations
- Indexation par rapport la position
  - Segmentation en objets (voir MPEG4 et 7)

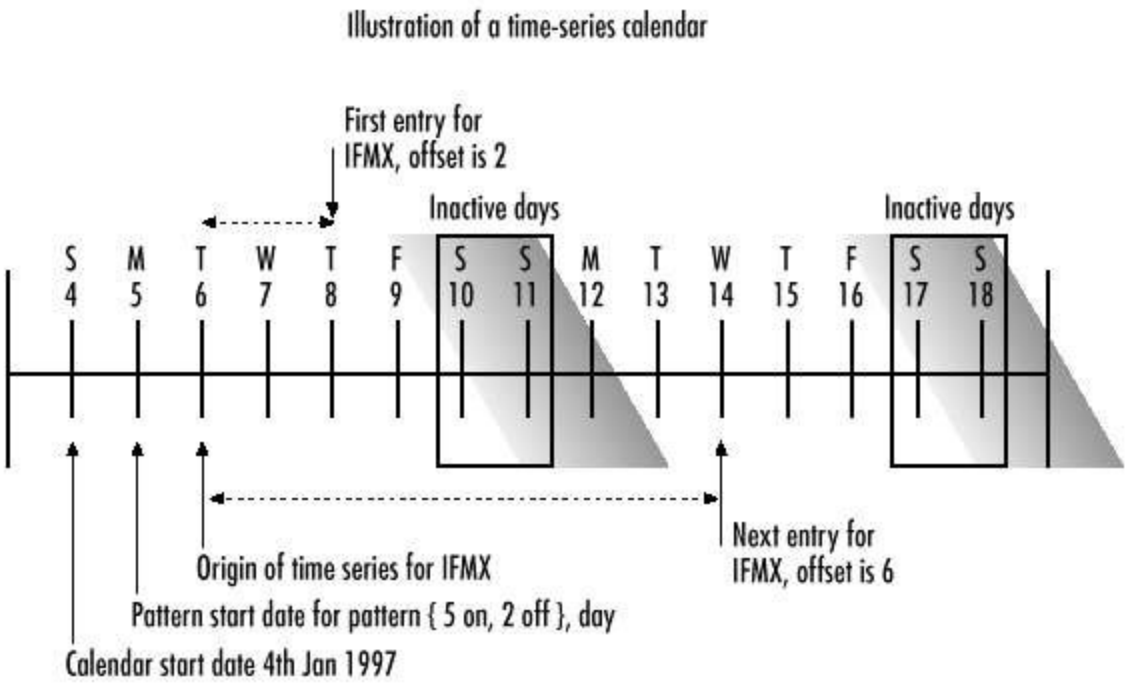
Les Données Spatiales

- Applications
  - Systèmes d ’Information Géographique (SIG/*GIS*)
    - Cadastre, Environnement, Voies de Circulation, Conduites, ..
- Type
  - Points, Ensembles (clusters) de Points
  - Lignes et Multilignes
  - Polygones complexes incluant 0 ou plusieurs « trous »
- Opérations
  - **ANYINTERACT, CONTAINS, COVEREDBY, COVERS, DISJOINT, EQUAL, INSIDE, OVERLAPBDYDISJOINT, OVERLAPBDYINTERSECT, TOUCH**
- Indexation par rapport la position
  - Techniques : Quad-Tree, …

Les Séries Temporelles

- Définition
  - Suite de couple (Valeur, estampille de temps)
- Applications
  - Finance (stock value), Santé (épidémiologie), Capteurs (prévision des risques, sécurité, sureté, …), …
- Type
  - calendar, ...
- Opérations
- Index
  - par rapport au temps

Les Séries Temporelles



Les Types Opaques dans Oracle 8

- Data Cartridges

```
REM une procédure de comparaison de séquence d 'ADN
Create or Replace Library libSmithAndWaterman
      as  'c:\datacartridges\libraries\smithwat.dll';

Create or Replace Procedure libSmithAndWaterman (
      adnseq VARCHAR, adnstr VARCHAR, pos NUMBER
) as External
      Library libSmithAndWaterman Name "dllsw" Language C
      Parameters(CONTEXT, adnseq String, adnstr String, pos Long
) With CONTEXT;
```

SQL3 et JDBC2.0

Correspondance de types

Type	getXXX()	setXXX()	updateXXX()
BLOB	getBlob()	setBlob()	updateBlob()
CLOB	getClob()	setClob()	updateClob()
ARRAY	getArray()	setArray()	updateArray()
Structured Type	getObject()	setObject()	updateObject()
REF	getRef()	setRef()	updateRef()

SQL3 et JDBC2.0

- ARRAY

```
ResultSet rs = stmt.executeQuery(
    "SELECT SCORES FROM STUDENTS WHERE ID = 2238"); rs.next();
Array scores = rs.getArray("SCORES");
```

- CLOB

```
Clob notes = rs.getClob("NOTES");
PreparedStatement pstmt = con.prepareStatement(
"UPDATE MARKETS SET COMMENTS = ? WHERE SALES < 1000000",
    ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
pstmt.setClob(1, notes);
```

- Structured Type

```
// CREATE TYPE PLANE_POINT ( X FLOAT,Y FLOAT)
ResultSet rs = stmt.executeQuery("SELECT POINTS FROM PRICES WHERE PRICE > 3000.00");
while (rs.next()) {  Struct point = (Struct)rs.getObject("POINTS");
            // do something with point
 }
```

Bibliographie

Objet-Relationnel

- Jeffrey D. Ullman, Jennifer Widom, "A First Course in Database Systems", 1ère édition, Ed. Prentice Hall Engineering, Science & Math, Avril 1997, ISBN 0-13-861337-0, 470 pp.
- G. Gardarin, "Bases de Données Objet et Relationnel", Ed Eyrolles, 1999, ISBN 2-212-09060-9
- Chris Date, « Introduction aux Bases de Données », 2000, 7ème édition, ISBN 2-7117-8664-1, Annexe B

SQL3

- Jim Melton, "Understanding new SQL", 1999, Ed Harcourt Pub. Ltd.
- Chris Date, « Introduction aux Bases de Données », 2000, 7ème édition, ISBN 2-7117-8664-1, Annexe B
- Nelson Mattos, "An Overview of the SQL3 Standard", presentation foils, Database Technology Institute, IBM Santa Teresa Lab., San Jose, CA, July 1996, ftp://jerry.ece.umassd.edu /isowg3/dbl/BASEdocs/descriptions/SQL3_foils.ps

Bibliographie

Informix 9 et +

- Informix Guide to SQL: Tutorial, Version 9.1
  http://www.informix.com/pub/pdf/3856.pdf

Oracle 8 et +

- Scott Urman , « Oracle8 PL/SQL Programming », ed Osborne-McGraw-Hill, Oracle Press Series, ISBN 0-07-882305-6.
  - PL/SQL au complet
- Christian Soutou, "Objet-Relationnel sous Oracle8, Modélisation avec UML", Ed Eyrolles, 1999, ISBN 2-212-09063-3
  - décrit bien les alternatives de conception des associations avec l 'objet-relationnel et avec le relationnel
- http://www.oracle.com/st