

Chapitre 1: Modèle relationnel-objet

Contenu

- Faiblesses du modèle relationnel
- Types abstraits de données: type OBJET, type COLLECTION (VArray, Nested Table)
- Tables: table relationnelle classique, table relationnelle NF2, table d'objets
- Références
- Requêtes
 - insertion dans une table NF2, dans une table d'objets
 - manipulation de collection
 - manipulation de références

Faiblesses du modèle relationnel

- Structure de données trop simple:
 - relation: tuples, attributs simples et monovalués, identifiants par valeur
 - besoin de représenter: objets complexes, attributs complexes et multivalués, identifiants d'objet
- Pas d'intégration des opérations avec les données:
 - pas de méthodes, pas d'encapsulation
- Pas de possibilité de référencer directement un objet (pointeur) par oid
 - jointures sur les identifiants de relations
- Pas de fonctions spécifiques à certaines données de type spatial, multimédia, ...
 - très peu adapté à ces types de données

Modèle objet

- Caractéristiques du modèle objet:
 - Identité d'objet (oid)
 - Objet (structure) complexe
 - Collection
 - Composition
 - Encapsulation
 - Spécialisation et héritage

- Deux possibilités:
 - créer de nouveaux SGBD orientés objets: O2, Versant, ...
 - étendre les SGBD relationnels existants: Oracle, PostGrès, Informix, ...

Modèle relationnel-objet (Oracle 9i)

■ Extensions:

- objets possédant une valeur et un identifiant d'objet système (oid)
- objets avec structure complexe: attributs complexes et multivalués (collections)
- références à un objet (par oid)
- types avec méthodes (en PL/SQL)
- spécialisation et héritage

Types abstraits de données

- En relationnel: types prédéfinis
 - CHAR, VARCHAR2, NUMBER, DATE, ...
- En relationnel-objet: nouveaux types abstraits de données (TAD) définis par les utilisateurs
 - ordre CREATE TYPE ...
- Plusieurs catégories de types:
 - types OBJET
 - types COLLECTION: VARRAY, NESTED TABLE

Type OBJET

- Pour représenter des objets complexes
- Type objet décrit par:
 - **nom**: unique dans le schéma
 - **attributs**
 - **méthodes**: fonctions ou procédures
 - ▶ écrites en PL/SQL ou Java et stockées dans la BD
 - ▶ écrites en d'autres langages (ex. C) et stockées à l'extérieur de la BD
- Syntaxe:

```
CREATE [OR REPLACE] TYPE nom_type AS OBJECT
    (att1 DATATYPE,
      att2 DATATYPE,
      ...,
      attn DATATYPE ,
      [définitions de méthodes] )
/
```

Type OBJET (suite)

■ Exemple:

```
CREATE TYPE T_Contact AS OBJECT (  
    nom          VARCHAR2(30),  
    telephone    VARCHAR2(20) )
```

■ Un type objet peut être utilisé comme:

● le type d'une colonne d'une table

```
▶ CREATE TABLE Annuaire (contact    T_Contact,  
                           date_entree DATE)
```

● le type d'un attribut d'un autre TAD

```
▶ CREATE TYPE T_Annuaire (contact    T_Contact,  
                           date_entree DATE)
```

● le type d'une table

```
▶ CREATE TABLE LesContacts OF T_Contact
```

▶ chaque ligne de la table *LesContacts* est de type *T_Contact*

Types COLLECTION

- 2 types:
 - type tableau: VARRAY
 - type table imbriquée: NESTED TABLE
- un type collection permet de créer des attributs multivalués
- un type collection a un constructeur de même nom que le nom du type
 - `nom_type(val1, val2, ..., valn)`

Type VARRAY

- Tableau à une dimension, i.e. ensemble ordonné d'éléments de même type
- Tableau de taille variable (variable-array)

- Syntaxe:

```
CREATE [OR REPLACE] TYPE nom_type  
AS VARRAY (nb_max) OF nom_type2
```

- définit un type tableau nommé *nom_type* de maximum *nb_max* éléments de type *nom_type2*
- *nom_type2* peut être n'importe quel type prédéfini ou créé par l'utilisateur

- Exemple:

- ```
CREATE TYPE T_liste_prix AS
 VARRAY(2) OF NUMBER(12,2)
```

# Type VARRAY (suite)

- Un type VARRAY peut être utilisé comme:
  - le type d'une colonne d'une table
    - ▶ `CREATE TABLE` `Produit` (`no_produit` `INTEGER`,  
`prix` `T_liste_prix`)
  - le type d'un attribut d'un autre TAD
    - ▶ `CREATE TYPE` `T_Produit` (`no_produit` `INTEGER`,  
`prix` `T_liste_prix`)
  - aussi dans une autre collection (collection de collection)

■ Example:

- CREATE TYPE T\_liste\_prenoms AS VARRAY(5) OF VARCHAR2(15)
- CREATE TYPE T\_Etudiant AS OBJECT (  
nom VARCHAR2(15),  
prenoms T\_liste\_prenoms,  
date\_n DATE)
- CREATE TABLE LesEtudiants OF T\_Etudiant

# Type VARRAY (suite)

## ■ Exemple plus complexe:

- CREATE TYPE `T_adresse_client` AS OBJECT (  
    rue                VARCHAR2(40),  
    code\_postal      INTEGER,  
    ville              VARCHAR2(30),  
    code\_pays        CHAR(2) )
- CREATE TYPE `T_liste_adresses` AS  
    VARRAY(3) OF `T_adresse_client`
- CREATE TABLE Client (no\_client INTEGER,  
                          adresses `T_liste_adresses`)

# Type NESTED TABLE

- Ensemble non ordonné d'éléments de même type

- Syntaxe:

```
CREATE [OR REPLACE] TYPE nom_type
AS TABLE OF nom_type2
```

- définit un type table imbriquée nommé *nom\_type* (table relationnelle) dont chaque tuple est de type *nom\_type2*
- *nom\_type2* peut être n'importe quel type prédéfini ou créé par l'utilisateur

- Exemple:

- ```
CREATE TYPE T_enfant AS OBJECT (  
    nom          VARCHAR2(30),  
    prenom       VARCHAR2(20) )
```
- ```
CREATE TYPE T_ens_enfants AS TABLE OF T_enfant
```

# Type NESTED TABLE (suite)

- Un type NESTED TABLE peut être utilisé comme:

- le type d'une colonne d'une table

- ▶ `CREATE TABLE` Employe (nom VARCHAR2(20),  
enfants T\_ens\_enfants)  
`NESTED TABLE` enfants `STORE AS` Enfant\_tab;

- ▶ Oracle crée physiquement une `table annexe` pour la "Nested Table" (que l'utilisateur doit nommer, ex. ici, Enfant\_tab)

- le type d'un attribut d'un autre TAD

- ▶ `CREATE TYPE` T\_Employe (nom VARCHAR2(20),  
enfants T\_ens\_enfants)

- aussi dans une autre collection (collection de collection)

# Type NESTED TABLE (suite)

- Remarques sur la table imbriquée (Nested Table):
  - génération automatique d'un identifiant *Nested\_Table\_Id* pour chaque collection
  - possibilité de créer un index dans la table imbriquée
  - attention: une table imbriquée n'est pas une table objet (pas d'oid associé à un élément de collection)

# Type NESTED TABLE (suite)

## ■ Exemple:

- CREATE TYPE T\_voiture AS OBJECT  
(modele VARCHAR2(15), annee DATE, no INTEGER)
- CREATE TYPE T\_ens\_voitures AS TABLE OF T\_voiture
- CREATE TABLE Personne (nom VARCHAR2(15),  
prenoms T\_liste\_prenoms, date\_naissance DATE,  
voitures T\_ens\_voitures)  
NESTED TABLE voitures STORE AS ToutesVoitures

Personne

| nom    | prenoms | date_n | voitures |
|--------|---------|--------|----------|
| Dupont | Pierre  | ...    | ld1      |
|        | Paul    |        |          |
| Durand | Jeanne  | ...    | ld2      |

ToutesVoitures

| Nested_table_Id | modele | annee | no |
|-----------------|--------|-------|----|
| ld1             | 2ch    | 1975  | 12 |
| ld1             | 206    | 2000  | 3  |
| ld2             | 2ch    | 1975  | 12 |
| ld1             | 607    | 2002  | 1  |

# Tables

- En Oracle 9i, on peut créer:
  - des tables relationnelles "classiques"
  - des tables relationnelles en non 1<sup>ère</sup> forme normale (NF<sup>2</sup>)
  - des tables d'objets
  
- Tables relationnelles classiques:
  - les types des colonnes sont des types prédéfinis d'Oracle, e.g. VARCHAR2, NUMBER, DATE,...
  - pas d'oid
  - ex: 

```
CREATE TABLE Departement (code CHAR(5),
 nom VARCHAR2(20))
```

# Tables (suite)

- Tables relationnelles en non 1<sup>ère</sup> forme normale (NF<sup>2</sup>):
  - les types des colonnes sont des TAD (varray, nested table ou objet)
  - pas d'oid
  - ex:
    - ▶ CREATE TYPE T\_liste\_telephones  
AS VARRAY(3) OF VARCHAR2(20)
    - ▶ CREATE TABLE Departement (  
nom VARCHAR2(20),  
telephones T\_liste\_telephones)

# Tables (suite)

## ■ Tables d'objets:

- uniquement créée avec la commande:

```
CREATE TABLE nom_table OF nom_TAD
```

- tout ligne d'une telle table possède un **oid** unique, et représente un n-uplet objet
- ex:

```
▶ CREATE TYPE T_Departement AS OBJECT (
 code CHAR(5),
 nom VARCHAR2(20),
 telephone VARCHAR2(20))
```

```
▶ CREATE TABLE LesDepartements OF T_Departement
```

- possible d'associer des contraintes (PK, unique, check) aux tables objet

```
▶ CREATE TABLE LesDepartements OF T_Departement(
 code PRIMARY KEY)
```

# Références

- Chaque ligne d'une table objet possède un **identificateur d'objet** qui:
  - identifie l'objet stocké dans ce n-uplet de façon unique,
  - c'est une colonne cachée, générée par le système,
  - sert à référencer l'objet,
  - Oracle lui associe un index.
  
- Référence: pointeur vers une ligne d'une **table objet**
  - OID utilisé pour faire des références vers des objets
  - Attention, impossible de référencer une collection !

# Références (suite)

- Création d'une référence:
  - ▶ Créer le TAD  $t$  dont on veut référencer les instances,
  - ▶ Créer la table contenant ces instances de  $t$ ,
  - Puis :
    - ▶ Créer le TAD qui référence  $t$  (mot-clé **REF**),
    - ▶ Créer la table associée.
- ou
  - ▶ Créer directement la table.

# Références (suite)

## ■ exemple 1: référencer la voiture d'une personne

- ```
CREATE TYPE T_voiture AS OBJECT (  
    modele  VARCHAR2(15),  
    annee   DATE,  
    no      INTEGER)
```
- ```
CREATE TABLE lesVoitures OF T_voiture;
```
- ```
CREATE TYPE T_personne2 AS OBJECT (  
    nom                VARCHAR2(15),  
    prenoms            T_liste_prenoms,  
    date_naissance     DATE,  
    voiture             REF T_voiture)
```
- ```
CREATE TABLE lesPersonnes OF T_personne2 (
 FOREIGN KEY (voiture) REFERENCES lesVoitures);
```

# Références (suite)

## ■ exemple 2: référencer la voiture d'une personne (création directe de la table)

- ```
CREATE TYPE T_voiture AS OBJECT (  
    modele  VARCHAR2(15),  
    annee   DATE,  
    no      INTEGER)
```
- ```
CREATE TABLE lesVoitures OF T_voiture; -- table d'objets
```
- ```
CREATE TABLE lesPersonnes (  
    nom          VARCHAR2(15),  
    prenom       T_liste_prenoms,  
    date_naissance DATE,  
    voiture      REF T_voiture)
```

 - ▶ *lesPersonnes* n'est pas directement construite avec un TAD, donc n'est pas une table d'objets.

Insertion dans une table NF²

■ exemple 1: colonne de type objet

- CREATE TYPE **T_Contact** AS OBJECT (
 nom VARCHAR2(30),
 telephone VARCHAR2(20))

CREATE TABLE Annuaire (**contact** **T_Contact**,
 date_entree DATE)

● INSERT INTO Annuaire VALUES (
 T_contact('Dupont', '022-1234567'),
 '12-MAR-07')

Annuaire

<i>contact</i>		<i>date_entree</i>
nom	telephone	
Dupont	022-1234567	12-MAR-07

Insertion dans une table NF² (suite)

■ exemple 2: colonne de type varray

- CREATE TYPE `T_liste_prix` AS VARRAY(2) OF NUMBER(12,2)
CREATE TABLE Produit (no_produit INTEGER,
 prix T_liste_prix)
- INSERT INTO Produit VALUES (1,
 T_liste_prix(20.50, 19.00))

Produit

<i>no_produit</i>	<i>prix</i>
1	20.50
	19.00

Insertion dans une table NF² (suite)

■ exemple 3: colonne de type varray

- CREATE TYPE `T_adresse_client` AS OBJECT (
 rue VARCHAR2(40), code_postal INTEGER,
 ville VARCHAR2(30), code_pays CHAR(2))

CREATE TYPE `T_liste_adresses` AS
 VARRAY(3) OF T_adresse_client

CREATE TABLE Client (no_client INTEGER,
 adresses T_liste_adresses)

● INSERT INTO table Client VALUES (1,
 T_liste_adresses(
 T_adresse_client('24 Dufour', 1204, 'Genève', 'CH'),
 T_adresse_client('20 Rhône', 1204, 'Genève', 'CH')
))

Client

no_client	adresses			
	rue	code_postal	ville	code_pays
1	24 Dufour	1204	Genève	CH
	20 Rhône	1204	Genève	CH

Insertion dans une table NF² (suite)

■ exemple 4: colonne de type nested table

- CREATE TYPE **T_voiture** AS OBJECT
(modele VARCHAR2(15), annee DATE, no INTEGER)

CREATE TYPE **T_ens_voitures** AS TABLE OF **T_voiture**

CREATE TABLE Personne (nom VARCHAR2(15),
prenoms T_liste_prenoms, date_naissance DATE,
voitures T_ens_voitures)

NESTED TABLE voitures STORE AS ToutesVoitures
- INSERT INTO Personne VALUES ('Dupont',
T_liste_prenoms('Pierre', 'Paul'), '16-MAR-80',
T_ens_voitures(
 T_voiture('2ch', '1975', 12),
 T_voiture('206', '2000', 3),
 T_voiture('607', '2002', 1)
));

Insertion dans une table NF² (suite)

Personne

<i>nom</i>	<i>prenoms</i>	<i>date_naissance</i>	<i>voitures</i>		
Dupont	Pierre	16-MAR-80	<i>modele</i>	<i>annee</i>	<i>no</i>
	Paul		2ch	1975	12
			206	2000	3
			607	2002	1

Insertion dans une table NF² (suite)

- mise à jour d'une nested table (exemple 4):

- supprimer une nested table:

```
UPDATE Personne SET voitures = NULL
WHERE nom = 'Dupont'
```

- créer un ensemble vide:

```
UPDATE Personne SET voitures = T_ens_voitures()
WHERE nom = 'Dupont'
```

- créer un singleton :

```
UPDATE Personne SET voitures = T_ens_voiture(
    Tvoiture('Mégane', '1998', 179))
WHERE nom= 'Dupont'
```

Insertion dans une table d'objets

■ exemple:

- ```
CREATE TYPE T_Departement AS OBJECT (
 code CHAR(5),
 nom VARCHAR2(20),
 telephone VARCHAR2(20))

CREATE TABLE LesDepartements OF T_Departement
```

- insertion comme dans une table relationnelle:

```
INSERT INTO LesDepartements VALUES('INFOR',
 'Informatique', '022-1234567')
```

LesDepartements

| <i>code</i> | <i>nom</i>   | <i>téléphone</i> |
|-------------|--------------|------------------|
| INFOR       | Informatique | 022-1234567      |

# Insertion dans une table d'objets (suite)

Interroger la table d'objets:

- exemple: récupérer l'ensemble de valeurs des attributs d'une table objet

- `SELECT * FROM LesDepartements;`

| ▶ code | nom           | telephone   |
|--------|---------------|-------------|
| -----  | -----         | -----       |
| INFOR  | Informatique  | 022-1234567 |
| MATHS  | Mathematiques | 022-6666666 |

- exemple: récupérer la valeur structurée des objets, opérateur **VALUE**

- `SELECT VALUE(d) FROM LesDepartements d;`

| ▶ value(d)                                             |
|--------------------------------------------------------|
| -----                                                  |
| T_Departement('INFOR', 'Informatique', '022-1234567')  |
| T_Departement('MATHS', 'Mathematiques', '022-6666666') |

# Manipulation de collection

## ■ Insertion et mise à jour d'une nouvelle valeur pour la collection entière

- pour varray et nested table

- ▶ UPDATE Personne SET prenom =  
T\_liste\_prenoms('Pierre', 'Michel')  
WHERE nom = 'Dupont'
- ▶ UPDATE Personne SET voitures = T\_ens\_voitures(  
Tvoiture('Mégane', '1998', 179))  
WHERE nom = 'Dupont'

## ■ Insertion, mise à jour, suppression d'éléments d'une collection

- pour nested table uniquement
- opérateur **TABLE**: permet de voir une collection comme une table

# Manipulation de collection (suite)

## ■ exemple 1: ajouter un élément à une nested table

- ```
INSERT INTO TABLE(SELECT voitures
                     FROM   Personne
                     WHERE  nom = 'Dupont')
VALUES ('Megane', '1998', 179)
```

■ exemple 2: mäj un élément dans une nested table

- ```
UPDATE TABLE(SELECT voitures
 FROM Personne
 WHERE nom = 'Dupont') p
set VALUE(p) = T_voiture('607', '2003', 1)
WHERE p.no = 1
```

(set p.annee = 2003)

| <i>nom</i> | <i>prenoms</i> | <i>date_n</i> | <i>voitures</i> |              |           |
|------------|----------------|---------------|-----------------|--------------|-----------|
| Dupont     | Pierre         | 16-MAR-80     | <i>mod</i>      | <i>annee</i> | <i>no</i> |
|            | Paul           |               | 2ch             | 1975         | 12        |
|            |                |               | 206             | 2000         | 3         |
|            |                |               | 607             | 2002         | 1         |

# Manipulation de collection (suite)

- exemple 3: supprimer un élément d'une nested table

```
delete FROM TABLE(SELECT voitures
 FROM Personne
 WHERE nom = 'Dupont') p
WHERE p.no = 1
```

| <i>nom</i> | <i>prenoms</i> | <i>date_n</i> | <i>voitures</i> |              |           |
|------------|----------------|---------------|-----------------|--------------|-----------|
| Dupont     | Pierre         | 16-MAR-80     | <i>mod</i>      | <i>annee</i> | <i>no</i> |
|            | Paul           |               | 2ch             | 1975         | 12        |
|            |                |               | 206             | 2000         | 3         |
|            |                |               | 607             | 2002         | 1         |

# Manipulation de collection (suite)

## ■ exemple 4: interroger une collection

- ```
SELECT    nom, prenom  
FROM      Personne
```

- ▶

NOM	PRENOMS
-----	---------

- | | |
|-------|-------|
| ----- | ----- |
|-------|-------|

- | | |
|--------|--------------------------------|
| Dupont | T_liste_prenoms(Pierre, Paul) |
|--------|--------------------------------|

- | | |
|--------|-------------------------|
| Durand | T_liste_prenoms(Jeanne) |
|--------|-------------------------|



retourne une
collection pour
chaque tuple

- ```
SELECT p.nom, pr.*
FROM Personne p, TABLE(p.prenoms) pr;
```

- ▶ 

| NOM | PRENOMS |
|-----|---------|
|-----|---------|

- |       |       |
|-------|-------|
| ----- | ----- |
|-------|-------|

- |        |        |
|--------|--------|
| Dupont | Pierre |
|--------|--------|

- |        |      |
|--------|------|
| Dupont | Paul |
|--------|------|

- |        |        |
|--------|--------|
| Durand | Jeanne |
|--------|--------|

- ▶ l'expression **TABLE(alias.collection)** peut utiliser tout alias déclaré à sa gauche dans la clause FROM

# Manipulation de collection (suite)

## ■ exemple 5: interroger une collection

- chercher les personnes dont un des prénoms est Paul

```
SELECT p.*
FROM Personne p, TABLE(p.prenoms) pr
WHERE pr.COLUMN_VALUE = 'Paul';
```

- chercher les personnes ayant une voiture 206

```
SELECT p.*
FROM Personne p, TABLE(p.voitures) v
WHERE v.modele = '206';
```

- chercher le nom des personnes et le numéro de leurs voitures

```
SELECT p.nom, v.no
FROM Personne p, TABLE(p.voitures) v;
```

# Manipulation de référence

- Deux opérateurs:
  - **REF**: permet d'obtenir la référence associée à un objet (son oid) à partir du tuple
  - **DEREF**: permet d'obtenir les valeurs de l'objet à partir de la référence à cet objet
  
- exemple: insérer une ligne dans la table LesPersonnes, et donc référencer une voiture
  - ```
CREATE TYPE T_personne2 AS OBJECT (  
    nom VARCHAR2(15), prenom T_liste_prenoms,  
    date_naissance DATE,  
    voiture REF T_voiture)  
  
CREATE TABLE lesPersonnes OF T_personne2 (  
    FOREIGN KEY (voiture) REFERENCES lesVoitures);
```

Manipulation de référence (suite)

- récupérer l'adresse (oid) de la voiture: utiliser REF qui renvoie l'adresse (oid) d'une ligne dans une table
- ```
INSERT INTO lesPersonnes
 SELECT 'Durand', T_liste_prenoms('Pierre', 'Paul'),
 '16-MAR-80', REF(v)
FROM lesVoitures v
WHERE v.no = 3;
```

LesPersonnes

| <i>nom</i> | <i>prenoms</i> | <i>date_n</i> | <i>voiture</i> |
|------------|----------------|---------------|----------------|
| Durand     | Pierre         | 16-MAR-80     |                |
|            | Paul           |               |                |

LesVoitures

| <i>modele</i> | <i>annee</i> | <i>no</i> |
|---------------|--------------|-----------|
| 2ch           | 1975         | 12        |
| 206           | 2000         | 3         |
| 607           | 2002         | 1         |

# Manipulation de référence (suite)

## ■ exemple: interroger

- `SELECT * FROM LesPersonnes;`

| <i>nom</i> | <i>prenoms</i> | <i>date_n</i> | <i>voiture</i>                                                          |
|------------|----------------|---------------|-------------------------------------------------------------------------|
| Durand     | Pierre         | 16-MAR-80     | 03F43970135A39847C...<br>-- adresse du pointeur.<br>Ne veut rien dire ! |
|            | Paul           |               |                                                                         |

- `SELECT p.nom, p.prenoms, p.date_n, Deref(p.voiture)`  
`FROM LesPersonnes p`  
`WHERE Deref(p.voiture).année > 1990;`

| <i>nom</i> | <i>prenoms</i> | <i>date_n</i> | <i>voiture</i>          |
|------------|----------------|---------------|-------------------------|
| Durand     | Pierre         | 16-MAR-80     | T_voiture(206, 2000, 3) |
|            | Paul           |               |                         |

# Manipulation de référence (suite)

- référence vide: référence qui n'existe plus ("dangling")
  - exemple: si on supprime la voiture no 3

LesPersonnes

| <i>nom</i> | <i>prenoms</i> | <i>date_n</i> | <i>voiture</i> |
|------------|----------------|---------------|----------------|
| Durand     | Pierre         | 16-MAR-80     |                |
|            | Paul           |               |                |

LesVoitures

| <i>modele</i> | <i>annee</i> | <i>no</i> |
|---------------|--------------|-----------|
| 2ch           | 1975         | 12        |
| 206           | 2000         | 3         |
| 607           | 2002         | 1         |

- mettre à NULL les références vides

```
UPDATE LesPersonnes
set voiture = NULL
WHERE voiture IS DANGLING
```