

02

Le modèle objet-relationnel et SQL3

25 % du module · Le modèle objet-relationnel (50 %) · SQL3 (50 %)

Le modèle objet-relationnel

Il étend le relationnel avec des concepts objet, sans abandonner les tables ni SQL. Il lève la 1FN (modèle NF², « non first normal form ») et est normalisé par SQL:1999, dit **SQL3**.

Types utilisateur (UDT)

Types structurés composés d'attributs et de méthodes.

Objets et OID

Chaque ligne d'une table d'objets possède un identifiant unique.

Références (REF)

Pointeurs vers des objets, qui remplacent souvent les jointures.

Collections

Attributs multivalués : VARRAY et NESTED TABLE.

Héritage

Sous-types (UNDER) et substituabilité des instances.

Méthodes

Comportement attaché au type : MEMBER, STATIC, MAP, ORDER.

SQL3 : définir des types objet

```
CREATE TYPE Adresse_t AS OBJECT (  
    rue    VARCHAR2(50),  
    ville  VARCHAR2(30),  
    cp     NUMBER(5)  
);  
/  
CREATE TYPE Personne_t AS OBJECT (  
    num     NUMBER,  
    nom     VARCHAR2(30),  
    adresse Adresse_t,  
    dateNais DATE,  
    MEMBER FUNCTION age RETURN NUMBER  
) NOT FINAL;  
/
```

CREATE TYPE ... AS OBJECT déclare un type structuré : attributs puis signatures des méthodes.

Un type peut être utilisé comme type d'attribut : **adresse** est une valeur composite.

NOT FINAL autorise la création de sous-types. Par défaut un type est FINAL.

Un constructeur est créé automatiquement : Adresse_t('R1', 'Mila', 43000)

Tables d'objets, OID et références

```
CREATE TABLE Personnes OF Personne_t
(PRIMARY KEY (num));

CREATE TYPE Departement_t AS OBJECT (
  code VARCHAR2(10),
  chef REF Personne_t
);
/
CREATE TABLE Departements OF Departement_t
(chef SCOPE IS Personnes);

INSERT INTO Departements
SELECT 'INFO', REF(p) FROM Personnes p
WHERE p.num = 1;
```

Table d'objets (OF type) : chaque ligne est un objet avec un OID géré par le SGBD.

REF type : attribut qui pointe vers un objet d'une table d'objets.

SCOPE IS limite les références à une table donnée.

REF(p) donne la référence d'un objet ; **DEREF(r)** renvoie l'objet pointé ; **VALUE(p)** renvoie l'objet d'une ligne.

Référence vers un objet supprimé : **dangling**, testée par IS DANGLING.

CRUD : type objet et table d'objets

Module_t AS OBJECT (code VARCHAR2(8), titre VARCHAR2(40), coef NUMBER) · Modules OF Module_t

C — Créer (INSERT)

```
INSERT INTO Modules
  VALUES (Module_t('BDA', 'BD avancées', 3));
-- sans constructeur : valeurs à plat
INSERT INTO Modules VALUES ('RES', 'Réseaux', 2);
```

R — Lire (SELECT)

```
SELECT m.code, m.titre FROM Modules m
WHERE m.coef >= 3;
-- l'objet entier et sa référence (OID)
SELECT VALUE(m), REF(m) FROM Modules m;
```

U — Modifier (UPDATE)

```
UPDATE Modules m SET m.coef = 4
WHERE m.code = 'BDA';
-- remplacer l'objet entier
UPDATE Modules m
SET   VALUE(m) = Module_t('BDA', 'BDA', 4)
WHERE m.code = 'BDA';
```

D — Supprimer (DELETE)

```
DELETE FROM Modules m
WHERE m.code = 'RES';
-- l'OID disparaît : les REF vers cet
-- objet deviennent pendantes (dangling)
```

CRUD : type composé (attribut objet)

Personne_t(num, nom, adresse Adresse_t(rue, ville, cp), dateNais) · Personnes OF Personne_t

C — Créer (constructeurs imbriqués)

```
INSERT INTO Personnes VALUES (  
  Personne_t(1, 'Ali',  
    Adresse_t('R. 1er Nov', 'Mila', 43000),  
    DATE '1990-05-10'));
```

R — Lire (notation pointée)

```
-- l'alias p est obligatoire  
SELECT p.nom, p.adresse.ville  
FROM   Personnes p  
WHERE  p.adresse.cp = 43000;
```

U — Modifier (un champ ou tout l'attribut)

```
UPDATE Personnes p  
SET    p.adresse.ville = 'Constantine'  
WHERE  p.num = 1;  
UPDATE Personnes p  
SET    p.adresse = Adresse_t('R2', 'Mila', 43000)  
WHERE  p.num = 2;
```

D — Supprimer

```
DELETE FROM Personnes p  
WHERE  p.adresse.ville = 'Mila';  
-- vider seulement l'attribut composé  
UPDATE Personnes p SET p.adresse = NULL  
WHERE  p.num = 2;
```

CRUD : références (REF)

Departement_t(code, chef REF Personne_t) · Departements OF Departement_t (chef SCOPE IS Personnes)

C — Créer (récupérer la référence)

```
INSERT INTO Departements
  SELECT Departement_t('INFO', REF(p))
FROM   Personnes p
WHERE  p.num = 1;
```

R — Lire (navigation, Deref)

```
-- jointure implicite
SELECT d.code, d.chef.nom FROM Departements d;
-- l'objet pointé complet
SELECT Deref(d.chef) FROM Departements d
WHERE  d.code = 'INFO';
```

U — Modifier (changer la cible)

```
UPDATE Departements d
SET    d.chef = (SELECT REF(p) FROM Personnes p
                  WHERE p.num = 2)
WHERE  d.code = 'INFO';
```

D — Supprimer, références pendantes

```
DELETE FROM Departements d
WHERE  d.chef.nom = 'Ali';
-- après suppression d'une personne
UPDATE Departements d SET d.chef = NULL
WHERE  d.chef IS Dangling;
```

Les collections : VARRAY et NESTED TABLE

```
CREATE TYPE Tels_t AS VARRAY(3)
  OF VARCHAR2(15);
/
CREATE TYPE Cours_t AS OBJECT (
  code VARCHAR2(8), titre VARCHAR2(40));
/
CREATE TYPE ListeCours_t AS TABLE OF Cours_t;
/
CREATE TABLE Enseignants (
  num    NUMBER PRIMARY KEY,
  tels   Tels_t,
  cours  ListeCours_t)
NESTED TABLE cours STORE AS cours_tab;

-- « Désimbriquer » une collection
SELECT e.num, c.titre
FROM   Enseignants e, TABLE(e.cours) c;
```

VARRAY

Taille maximale fixée, éléments ordonnés, stocké avec la ligne.

NESTED TABLE

Taille non bornée, non ordonnée, stockée dans une table séparée (STORE AS). Chaque élément peut être modifié par SQL.

CRUD : VARRAY

```
Tels_t AS VARRAY(3) OF VARCHAR2(15) · Enseignants(num, tels Tels_t, cours ListeCours_t)
```

C — Créer (constructeur de collection)

```
INSERT INTO Enseignants (num, tels)
VALUES (1, Tels_t('0555112233', '0661334455'));
-- au plus 3 éléments (taille maximale)
```

R — Lire (collection ou éléments)

```
SELECT e.tels FROM Enseignants e
WHERE e.num = 1;      -- un seul bloc
SELECT e.num, t.COLUMN_VALUE AS tel
FROM Enseignants e, TABLE(e.tels) t;
```

U — Modifier (remplacement complet)

```
UPDATE Enseignants e
SET e.tels = Tels_t('0555112233', '0770556677')
WHERE e.num = 1;
-- un élément seul : en PL/SQL
-- (v.EXTEND; v(3) := '...'; puis UPDATE)
```

D — Supprimer

```
UPDATE Enseignants e SET e.tels = Tels_t()
WHERE e.num = 1;      -- collection vide
UPDATE Enseignants e SET e.tels = NULL
WHERE e.num = 2;      -- pas de collection
```

CRUD : NESTED TABLE (TABLE OF)

`Cours_t(code, titre) · ListCours_t AS TABLE OF Cours_t · NESTED TABLE cours STORE AS cours_tab`

C — Créer (la ligne, puis un élément)

```
INSERT INTO Enseignants (num, cours) VALUES (2,  
  ListCours_t(Cours_t('BDA', 'BD avancées'),  
    Cours_t('XML', 'Semi-structuré')));  
INSERT INTO TABLE(SELECT e.cours  
  FROM Enseignants e WHERE e.num = 2)  
VALUES (Cours_t('SEC', 'Sécurité'));
```

R — Lire (désimbriquer)

```
SELECT e.num, c.code, c.titre  
FROM   Enseignants e, TABLE(e.cours) c  
WHERE  c.code = 'BDA';
```

U — Modifier un élément

```
UPDATE TABLE(SELECT e.cours  
  FROM Enseignants e WHERE e.num = 2) c  
SET    c.titre = 'Bases de données avancées'  
WHERE  c.code = 'BDA';
```

D — Supprimer un élément

```
DELETE FROM TABLE(SELECT e.cours  
  FROM Enseignants e WHERE e.num = 2) c  
WHERE  c.code = 'XML';  
-- la sous-requête doit renvoyer une seule ligne
```

Héritage de types

```
CREATE TYPE Etudiant_t UNDER Personne_t (  
    numInscription NUMBER,  
    specialite      VARCHAR2(20)  
);  
/  
-- Substituabilité : un étudiant est une personne  
INSERT INTO Personnes VALUES (  
    Etudiant_t(5, 'Amine',  
        Adresse_t('R1', 'Mila', 43000),  
        DATE '2001-03-12', 2025017, 'STIC'));  
  
SELECT TREAT(VALUE(p) AS Etudiant_t).specialite  
FROM   Personnes p  
WHERE  VALUE(p) IS OF (Etudiant_t);
```

UNDER crée un sous-type qui hérite des attributs et méthodes. Le supertype doit être **NOT FINAL**.

Substituabilité : une table de Personne_t peut contenir des Etudiant_t.

IS OF teste le type d'une instance ; **TREAT** la convertit vers un sous-type.

NOT INSTANTIABLE définit un type abstrait ;
OVERRIDING redéfinit une méthode.

CRUD : héritage de types

Etudiant_t UNDER Personne_t (numInscription, specialite) · Personnes OF Personne_t

C — Créer (sous-type dans la table mère)

```
INSERT INTO Personnes VALUES (  
  Etudiant_t(5, 'Amine',  
    Adresse_t('R1', 'Mila', 43000),  
    DATE '2001-03-12', 2025017, 'STIC'));
```

R — Lire (polymorphisme, TREAT)

```
SELECT p.nom FROM Personnes p; -- tous  
SELECT p.nom,  
  TREAT(VALUE(p) AS Etudiant_t).specialite  
FROM   Personnes p  
WHERE  VALUE(p) IS OF (ONLY Etudiant_t);
```

U — Modifier

```
UPDATE Personnes p SET p.nom = 'Amine B.'  
WHERE  p.num = 5; -- attribut hérité  
-- attribut du sous-type : remplacer l'objet  
UPDATE Personnes p SET VALUE(p) = Etudiant_t(5,  
  'Amine', p.adresse, p.dateNais, 2025017, 'RSD')  
WHERE  p.num = 5;
```

D — Supprimer selon le type

```
-- seulement les étudiants  
DELETE FROM Personnes p  
WHERE  VALUE(p) IS OF (ONLY Etudiant_t);
```

Méthodes des types

```
CREATE TYPE BODY Personne_t AS
  MEMBER FUNCTION age RETURN NUMBER IS
  BEGIN
    RETURN TRUNC(
      MONTHS_BETWEEN(SYSDATE, SELF.dateNais) / 12);
  END;
END;
/
-- Appel d'une méthode dans une requête
SELECT p.nom, p.age()
FROM   Personnes p
WHERE  p.age() > 25;
```

Type	Rôle
MEMBER	méthode d'instance, accède à SELF
STATIC	méthode de type, sans SELF
CONSTRUCTOR	constructeur redéfini
MAP	renvoie un scalaire pour comparer et trier
ORDER	compare SELF à un autre objet (-1, 0, 1)

Un type a au plus une méthode MAP ou une méthode ORDER.

Interroger une base objet-relationnelle

Mécanisme	Exemple	Effet
Notation pointée	p.adresse.ville = 'Mila'	accès aux attributs composites
Navigation par REF	SELECT d.chef.nom FROM Departements d	jointure implicite
DEREF	SELECT Deref(d.chef) FROM Departements d	objet pointé complet
VALUE	SELECT VALUE(p) FROM Personnes p	la ligne vue comme un objet
TABLE()	FROM Enseignants e, TABLE(e.cours) c	parcours d'une collection
Mise à jour	INSERT INTO TABLE(SELECT e.cours ...)	ajout dans une NESTED TABLE