Chapitre 2
Le modèle objet-relationnel et SQL3

Master II STIC

Année universitaire 2025/2026

2.1. Introduction

Le **modèle objet-relationnel** est une évolution du modèle relationnel classique.

Il vise à combiner les avantages des **bases de données relationnelles** (maturité, normalisation, SQL) avec ceux de la **programmation orientée objet** (encapsulation, héritage, réutilisabilité).

Cette évolution a été normalisée avec la norme **SQL:1999**, aussi appelée **SQL3**, qui étend le langage SQL traditionnel pour prendre en charge ces nouveaux concepts.

2.2. Le modèle objet-relationnel

- **2.2.1. Objectifs du modèle objet-relationnel**
- Réduire le fossé entre le **monde des objets** (applications OO) et le **monde relationnel** (base de données).
- Permettre une **modélisation plus naturelle** des données complexes.
- Améliorer la **réutilisabilité** et la **cohérence** des structures de données.

2.2. Le modèle objet-relationnel

2.2.2. Caractéristiques principales

a) Types de données complexes

- Possibilité de définir des **types structurés** composés (par exemple, une adresse avec rue, ville, code postal).
- Les champs peuvent eux-mêmes être des objets ou des collections.

2.2. Le modèle objet-relationnel

2.2.2. Caractéristiques principales

b) Identité des objets (OID)

- Chaque tuple peut avoir une **identité unique** indépendante de ses valeurs.
- Cela facilite la référence à un objet même si ses attributs changent.

2.2. Le modèle objet-relationnel

2.2.2. Caractéristiques principales

c) Encapsulation

- Les **méthodes** peuvent être définies avec les données (comme en programmation objet).
- Ces méthodes sont appelables depuis des requêtes SQL.

2.2. Le modèle objet-relationnel

2.2.2. Caractéristiques principales

d) Héritage

- Un type peut hériter des propriétés et méthodes d'un autre type.
- Permet une modélisation hiérarchique naturelle.

e) Collections

- Possibilité de stocker des **ensembles**, **listes**, ou **tableaux** dans un attribut.

2.3. SQL3 (SQL:1999)

SQL3 est la norme qui introduit la plupart des concepts objet-relationnels dans SQL.

2.3.1. Évolutions majeures par rapport à SQL2

- Support des **types définis par l'utilisateur (UDT)**
- Méthodes associées aux types
- **Constructeurs** pour instancier des objets
- **Héritage** de types
- Gestion des **collections** (ARRAY, MULTISET, etc.)
- Intégration des **objets et références**

2.3.2. Types définis par l'utilisateur (UDT)

Exemple de définition d'un type objet :

```
CREATE TYPE Adresse AS Object (
   rue VARCHAR(50),
   ville VARCHAR(30),
   code_postal VARCHAR(10)
);
```

2.3.2. Types définis par l'utilisateur (UDT)

Ce type peut ensuite être utilisé dans une table :

```
CREATE TABLE Personne (
   id INTEGER PRIMARY KEY,
   nom VARCHAR(30),
   prenom VARCHAR(30),
   adresse Adresse
);
```

2.3.3. Méthodes et encapsulation

```
CREATE TYPE Cercle AS Object (
   rayon FLOAT
)
METHOD getSurface() RETURNS FLOAT;
```

Et de les implanter :

```
CREATE METHOD getSurface FOR Cercle
BEGIN
   RETURN 3.14159 * SELF.rayon * SELF.rayon;
END;
```

2.3.4. Héritage

Un type peut hériter d’un autre type :

```
CREATE TYPE Employe UNDER Personne AS (
   salaire FLOAT
);
```

👉 Ainsi, **Employe** hérite des attributs de **Personne** et peut en ajouter.

2.3.5. Références et objets

SQL3 introduit des **références** pour établir des liens entre objets :

```
CREATE TABLE Departement (
   idDept INTEGER PRIMARY KEY,
   nomDept VARCHAR(30)
);
```

2.3.5. Références et objets

```
CREATE TABLE Employe OF Personne (
   refDept REF(Departement)
);
```

REF permet de stocker une référence vers un autre objet.

C’est l’équivalent d’un pointeur objet.

2.3.6. Collections

SQL3 autorise les attributs de type collection :

```
CREATE TABLE Classe (
   idClasse INTEGER PRIMARY KEY,
   etudiants VARRAY [10] of VARCHAR(50))
);
```

2.4. Avantages de SQL3 et du modèle objet-relationnel

- Modélisation plus naturelle de données complexes.
- Meilleure intégration avec les langages objets.
- Réduction de l'« object-relational impedance mismatch ».
- Réutilisation et extensibilité.
- Compatibilité ascendante avec SQL2.

2.5. Exemples de SGBD supportant SQL3 et l'objet-relationnel

- PostgreSQL : support avancé des types utilisateurs, fonctions et héritage.
- Oracle (Object-Relational features)
- IBM DB2 (Object-Relational extensions)
- Informix (Universal Server)

2.6. Limites et défis

- Plus grande complexité de conception.
- Difficulté de migration à partir de modèles relationnels classiques.
- Support variable selon les SGBD.
- Performance parfois inférieure pour certains cas simples.

Partie pratique : Exemples CRUD avec SQL3

1. Définition d’un type objet (UDT)

CREATE

```
-- Création d’un type objet
CREATE TYPE Adresse AS OBJECT (
   rue VARCHAR2(50),
   ville VARCHAR2(30),
   code_postal VARCHAR2(10)
);
```

```
CREATE TYPE Personne AS OBJECT (
   id NUMBER,
   nom VARCHAR2(30),
   prenom VARCHAR2(30),
   adresse Adresse
);
```

2. Table d’objets

CREATE

```
-- Table d’objets de type Personne
CREATE TABLE Personnes OF Personne (
   PRIMARY KEY (id)
);
```

2. Table d’objets

INSERT

```
INSERT INTO Personnes VALUES (
  Personne(1, 'Dupont', 'Jean', Adresse('1 rue A', 'Paris', '75000'))
);

INSERT INTO Personnes VALUES
  (2, 'Martin', 'Claire', Adresse('2 rue B', 'Lyon', '69000'));
```

2. Table d'objets

SELECT

```
SELECT p.nom, p.adresse.ville
FROM Personnes p;
```

UPDATE

```
UPDATE Personnes p
SET p.adresse = Adresse('10 rue Z', 'Marseille',
 '13000')
WHERE p.id = 1;
```

DELETE (supprimer l'adresse)

```
UPDATE Personnes p SET p.adresse= NULL WHERE p.id =
 2;
```

3. Table imbriquée (Nested Table)

```
-- Définir un type TABLE
CREATE OR REPLACE TYPE competence_table AS TABLE OF VARCHAR2(50);
/

-- Type objet
CREATE OR REPLACE TYPE employe_type AS OBJECT (
    id_employe   NUMBER,
    nom          VARCHAR2(50),
    competences  competence_table
);
/
```

3. Table imbriquée (Nested Table)

```
-- Table objet avec table imbriquée
CREATE TABLE employe_obj OF employe_type
(
    CONSTRAINT pk_employe PRIMARY KEY (id_employe)
)
NESTED TABLE competences STORE AS emp_comp_table;
/
```

Insertion initiale

```
INSERT INTO employe_obj VALUES (1, 'Nadia',
  competence_table('Java','SQL','Python'));
INSERT INTO employe_obj VALUES (2, 'Ali', competence_table('C++','Docker'));
```

Ajouter un élément dans une table imbriquée

```
INSERT INTO TABLE(
    SELECT e.competences
    FROM employe_obj e
    WHERE id_employe = 1
) VALUES ('Machine Learning');
```

Supprimer un élément dans une table imbriquée

```
DELETE FROM TABLE(
     SELECT e.competences
     FROM employe_obj e
     WHERE id_employe = 1
)
WHERE COLUMN_VALUE = 'SQL';
```

Modifier un élément dans une table imbriquée

```
UPDATE TABLE(
     SELECT e.competences
     FROM employe_obj e
     WHERE id_employe = 1
) c
SET c.COLUMN_VALUE = 'PL/SQL'
WHERE c.COLUMN_VALUE = 'Python';
```

4. VARRAY

CREATE

```
CREATE TYPE VarrayTel AS VARRAY(5) OF VARCHAR2(15);

CREATE TYPE Client AS OBJECT (
   id NUMBER,
   nom VARCHAR2(30),
   telephones VarrayTel
);

CREATE TABLE Clients OF Client (PRIMARY KEY (id));
```

4. VARRAY

INSERT

```
INSERT INTO Clients VALUES (
  Client(1, 'Ahmed', VarrayTel('0770000001',
 '0770000002'))
);
```

SELECT

```
SELECT c.nom, t.COLUMN_VALUE AS telephone
FROM Clients c,
     TABLE(c.telephones) t;
```

4. VARRAY

UPDATE

```
UPDATE Clients c
SET c.telephones = VarrayTel('0770000003',
 '0770000004', '0770000005')
WHERE c.id = 1;
```

DELETE

```
DELETE FROM Clients c WHERE c.id = 1;
```

5. Références REF

CREATE

```
CREATE TYPE Departement AS OBJECT (
   idDept NUMBER,
   nomDept VARCHAR2(30)
);

CREATE TABLE Departements OF Departement (
   PRIMARY KEY (idDept)
);
```

5. Références REF

```
CREATE TYPE Employe AS OBJECT (
   idEmp NUMBER,
   nom VARCHAR2(30),
   dept REF Departement
);

CREATE TABLE Employes OF Employe (PRIMARY KEY
  (idEmp));
```

5. Références REF

INSERT

```
INSERT INTO Departements VALUES (Departement(1,
 'Informatique'));
INSERT INTO Departements VALUES (Departement(2,
 'Finance'));


INSERT INTO Employes
SELECT Employe(1, 'Sami', REF(d))
FROM Departements d
WHERE d.idDept = 1;
```

5. Références REF

SELECT avec DEREF

```
SELECT e.nom, DEREF(e.dept).nomDept AS
 departement
FROM Employes e;
```

5. Références REF

UPDATE

```
UPDATE Employes e
SET e.dept = (SELECT REF(d) FROM Departements d
 WHERE d.idDept = 2)
WHERE e.idEmp = 1;
```

DELETE

```
DELETE FROM Employes WHERE idEmp = 1;
```

6. Héritage de types

CREATE

```
CREATE OR REPLACE TYPE personne AS OBJECT (
    id_personne NUMBER,
    nom         VARCHAR2(50)
) NOT FINAL;
/

CREATE OR REPLACE TYPE etudiant UNDER personne (
    matricule  VARCHAR2(20),
    moyenne    NUMBER
);
/
```

6. Héritage de types

CREATE

```
CREATE TABLE personne_tab OF personne
(
    CONSTRAINT pk_personne_tab PRIMARY KEY
  (id_personne)
);

CREATE TABLE etudiant_tab OF etudiant;
```

6. Héritage de types

INSERT

```
INSERT INTO personne_tab VALUES (1, 'Ali');
INSERT INTO etudiant_tab VALUES (2, 'Nadia',
 'MAT2025', 15.5);
```

SELECT

```
SELECT VALUE(p) FROM personne_tab p;
SELECT VALUE(e) FROM etudiant_tab e;
```

6. Héritage de types

UPDATE

```
UPDATE etudiant_tab e
SET e.moyenne = 16
WHERE e.id_personne = 2;
```

DELETE

```
DELETE FROM etudiant_tab WHERE id_personne = 2;
```