Le modèle relationnel-objet

Année universitaire 2019-2020

Génèse de la technologie objet-relationnelle

- La technologie objet-relationnelle est née en 1992,
- l'exemple le plus flagrant étant peut-être la base de données DB2 qualifiée d' "Universal Database" par IBM, qui tendent à intégrer les nouvelles technologies comme l'objet-relationnel tout en conservant toutes les anciennes fonctionnalités de leurs SGBD.

Pourquoi intégrer l'objet au relationnel?

- Le modèle relationnel a fait ses preuves , a des points forts indiscutables et aussi un certain nombre de points faibles, auxquels le modèle objet répond bien.
- L'intégration E/O permet aux utilisateurs de ne pas abandonner 3 décennies de développement de leurs applications sur des BDD relationnelles.
- Points forts du modèle relationnel :
  - concepts (tables, enregistrements, relations, champs, etc…) simples et aisément compréhensibles.
  - SQL : langage standardisé pour la création et manipulation des BDR.
- Parmi les points forts du modèle objet :
  - L'identité de l'objet
  - L'encapsulation
  - L'héritage
  - Support d'objets complexes

Pourquoi faire évoluer le modèle relationnel?

Limitations du modèle relationnel:

- Pas de support des objets complexes:
  - **Structure de données trop simple:**
    - **relation: {tuple} , 1FN, identifiant: valeur**
- Pas d'intégration des opérations avec les données :
  - **pas de méthodes et pas d'encapsulation**
- le nombre de types de base restreint et non extensible.

Les Apports du modèle objet

- L'identité d'objet : identifiant d'objet
- L'encapsulation des données.
- L'héritage.
- Le support d'objets complexes:
  - Non-1FN,

Les apports du modèle objet-relationnel

- Le modèle objet-relationnel se fonde sur l'extension du modèle relationnel par les concepts essentiels de l'objet.
- Le cœur du système reste donc relationnel, mais tous les concepts clés de l'objet peuvent être pris en charge.
- Deux éléments constituent les véritables fondements des apports du modèle objet-relationnel : les **objets complexes** et les **types abstraits** de données.

Exemple : table Personne

nom	{prenoms}	Date_naissance	{voitures}			...
	prenom		modele	annee	no	
Douib	Zoubir	16-05-1963	Clio	2010	128	
	Ahmed		Mégane	2012	371	
	Amine					
Layeb	Mohamed	29-02-1984	Twingo	2015	17	
	Rafik					

Les apports du modèle objet-relationnel

- **Les objets complexes**: permettent la définition de domaines multivalués et en **non première forme normale.**

Exemple :

PERSONNE (np INT, nom VARCHAR, prénom VARCHAR, {ADRESSES (rue VARCHAR, loc VARCHAR, Num INT)})

//pour chaque personne, correspondra une table imbriquée *ADRESSES.*

- **Autres objets complexes :** les collections génériques (l'ensemble, la liste...), appelés également "collection template", qui peuvent être imbriqués pour représenter des objets très complexes.

Type de données utilisateur

- Type de données définissables par l'utilisateur composé d'une structure de données et d'opérations encapsulant cette structure.
- Le système de types du SGBD devient extensible et n'est plus limité aux types alphanumériques de base. L'utilisateur est en mesure de définir ses propres types.
- Il devient possible de définir des types image, point, vecteur, vidéo etc. avec les opérations adéquates qui les encapsulent. Ils sont appelés types abstraits (ADT, Abstract Data Type).

Les apports du modèle objet-relationnel

- un TAD est un ensemble de fonctions qui cache l'implémentation d'un objet.
- Les types abstraits : sont utilisés pour :
  - définir des structures de données partagées qui peuvent être utilisées pour définir une ou plusieurs tables, ou encore un ou plusieurs attributs dans une ou plusieurs tables,
  - définir des nouveaux types qui vont enrichir la collection existante de types disponibles par défaut.

Vue sur SQL3

Les Concepts De Base de SQL3 (1)

- SQL3 est une extension de SQL , qui comporte un langage de requêtes et étend le modèle relationnel pour supporter les notions objet
- SQL3 donne aux utilisateurs les possibilités suivantes :

  - définir de nouveaux domaines à structure complexes, appelés **types**,
  - associer à chaque type des méthodes,
  - créer des hiérarchies de types,
  - créer des objets qui sont composés d'une valeur structurée et d'un OID,

Les Concepts De Base de SQL3 (2)

- établir des liens de composition par des attributs référence,
- créer des tables contenant soit des tuples normaux soit des tuples en non première forme normale.

Les Types (1)

- Les types classiques (numériques, caractères, date…) de SQL,
- Les types définis par l'utilisateur (varray, table, et object).
- VARRAY<longueur> : créer des attributs multivalués sous la forme de tableaux à une dimension limitée.
- TABLE : créer des attributs multivalués sous la forme de tableaux à une dimension non limite.

Les types(2)

- **Les Types VARRAY ou TABLE**

Syntaxe:

CREATE **TYPE** <nom-type> AS (**VARRAY** <longueur> | **TABLE**) OF <nom-type2>;

Exemple

Create **TYPE** Prénom as **Varray** (3) of varchar(10) ;

Create **TYPE** Telephone as **Table** of varchar(10);

Les types (3)

- **Les Types OBJECT**

Syntaxe :

CREATE **TYPE** <nom-type> AS **OBJECT** (nom-attrbt1 nom-type1, nom-attrbt2 nom-type2, …, nom-attrbt n nom-type n) ;

Exemple

Create **TYPE** Employé as **Object** (nom varchar2(10), nss number, datenais Date) ;

Les types (4)

Remarque

- On peut créer une structure contenant plusieurs constructeurs, pour cela on définit un type intermédiaire par constructeur.

Exemple

CREATE **TYPE** T-Personne AS **OBJECT** (nom VARCHAR(20), prénoms T-Liste_Prénoms, enfants T-Liste_Enfants);

CREATE **TYPE** T-Liste_Prénoms AS **VARRAY** (20) OF VARCHAR;

CREATE **TYPE** T-Liste-Enfants AS **VARRAY** OF T-Enfant ;

CREATE **TYPE** T-Enfant AS **OBJECT** (prénoms T-Liste_Prénoms, sexe CHAR, date nais DATE) ;

Les types (5)

- Un type ne peut pas contenir des contraintes d'intégrité.

- La commande « CREATE or REPLACE TYPE » permet de redéfinir un type s'il existe déjà.

Les types (6)

- **L'Ajout d'un attribut dans un type**

Syntaxe:

Alter **type** <nom_type> **add attribute** <nom_attribute> <type_attribute> cascade ;

Exemple

Alter **type** Employe_Type **add attribute** date_naissance date cascade;

Cascade : propage aux tables construites à partir du type.

Les types (7)

- **L'Ajout d'une méthode ou fonction à un type**

Syntaxe:

Alter **type** <nom_type> **add member** Function <nom_fonction> [(<nom_parametre>, …)] **return** <type_de retour> ;

Exemple

Alter **type** Employe_Type **add member** function age

return integer cascade;

Les Tables (1)

- **Création d'une table relationnelle classique**

Syntaxe:

CREATE **TABLE** nom-table (nom-attribut1 nom-type1, nom-attribut2 nom-type2,…, nom attribut n nom-type n) ;

Exemple

CREATE **TABLE** Personne (NUM Char(11),

nom Varchar (20), prénom Varchar (20), rue Varchar

(20), numVarchar (4), localité Varchar (20)) ;

Les Tables (2)

- **Création d'une table relationnelle en non première forme normale**

Syntaxe:

CREATE **TABLE** nom-table (nom-attribut1 nom-type1,

nom-attribut2 nom-type2,…, nom attribut n nom-type n) ;

Exemple

CREATE **TABLE** Personne (NUM Char(11), nom Varchar (20), prénom T_prenom, adr T_adresse) ;

CREATE **TYPE** T-prenom AS **VARRAY** (3) OF VARCHAR;

CREATE **TYPE** T-adresse AS **OBJECT** (rue Varchar(20), numero Varchar (4), localité Varchar (20));

Les Tables (3)

- **Création d'une table d'objet**

Syntaxe: CREATE **TABLE** nom-table OF nom-type ;

Exemple

Create **type** Employe_Type AS **OBJECT** (Matricule integer, Nom varchar (30), Dept integer);

Create **table** Employe OF Employe_Type (primary key(matricule));

Remarque:

On peut créer une table à partir d'un type et indiquer des contraintes d'intégrité comme dans la forme classique :

(primary key (nomcol*) |Unique (nomcol*) |(Check (condition))

Foreign key (nomcol*) références nom_tabl(nomcol*)

Les Tables d'objet

- Une table d’objet est:
  - Uniquement créable avec la commande :

    create table nom_table of nom_TAD;

  - Toute instance d’une telle table possède un oid unique, ce sont des n-uplets objet.
  - La portée de cet oid est globale.
- Attention:
  - Les autres tables (rel. "classique", 1) et (rel. NF^2,…..) ne sont pas des tables d’objets.
  - Leurs instances n’ont pas d’oid

Résumé

(1) Table relationnelle "classique"

- CREATE TABLE Personne (
  nom VARCHAR(20),
  telephone number(10))

(2) Tables relationnelles NF1

- CREATE TABLE Personne (
  nom VARCHAR(20),
  telephone Ttelephones))

(3) Tables d'objet

- CREATE TABLE Personne OF TPerson;

Les références

- **Chaque instance (n-uplet) d'une table d'objets possède un identificateur d'objet (OID) 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

- **Référence: Pointeur vers une instance d'une table objet**
  - **Attention, impossible de référencer une collection**
- **Création d'une référence:**
  - **Créer le TAD t dont on veut référencer les instances,**
  - **Créer la table contenant ces instances de t,**
  - **Créer le TAD qui référence t (mot-clé REF)**
  - **Créer la table associée ou**
  - **Créer directement la table.**

Exemple

- **Référencer la voiture d'une personne :**

```
Create type Tvoiture as object (
                modele varchar2(15), annee date,
                No integer)

/

Create table voitures of Tvoiture;

Create type Tpersonne as object(nom varchar2(15), prenoms liste_prenom, date_naissance Date, voiture REF Tvoiture)

/

Create table personnes of Tpersonne;
```

Identité et attributs-référence (1)

➢Tout ce qui est de type OBJECT possède une identité (un OID) et **peut donc être référencé par un lien de composition.**

➢Les lignes d'une table sont considérées comme des objets avec un identifiant (OID);

- Pour référencer un objet, on utilise la clause "REF nom-type » : donne comme résultat le oid de l'objet

- Pour faire référencer une colonne à un objet, on utilise la clause: «nom_colonne» **REF** « nom-type ».

Identité et attributs-référence (2)

Exemples :

1) **CREATE TYPE T-Personne AS OBJECT (code NUMBER, Nom VARCHAR(18), Prénom VARCHAR(18), Conjoint REF T-Personne ,...);**

//attribut référence contiendra un OID de type T-Personne. Conjoint fait une référence à un objet de type T-personne.

Create table les_personnes of t_personnes;

2) Create **Type** Voiture as **object** (Numero Char(9), Couleur Varchar, Propriétaire **Ref** (personne)) ;

Identité et attributs-référence (2)

Deux nouvelles clauses permettent de manipuler les OIDs et les valeurs des objets.

- **REF (objet):** La clause **REF (variable-objet)** donne en résultat l'OIDS de l'objet. C'est souvent utile pour les mises à jour.

Exemple: (Insertion d'un objet avec le lien de référence Conjoint)

Create **Table** Les_Personnes OF T-Personne ;

INSERT INTO Les_Personnes (code, nom, prénom, conjoint) VALUES (171, 'labed', 'said', (SELECT **REF** (**p**) FROM Les_Personnes p WHERE p.nom ='mouras' AND p.prénom ='Asma'));

Identité et attributs-référence (3)

- **VALUE (objet):** Pour obtenir la valeur d'un objet à partir de sa référence, on utilise la fonction value, qui prend la référence d'un objet et renvoie sa valeur.

Exemple

```
SELECT p.nom, p.conjoint  FROM  Les_Personnes  p;
  /*Renvoie le nom + l'OID du conjoint */

SELECT p.nom, value (p.conjoint) FROM Les_Personnes p;
 /*Renvoie le nom + la valeur du conjoint */
```

Hiérarchie de types Object (1)

possibilité de créer des hiérarchies de types **OBJECT**, et de les utiliser lors des créations de tables. Pour cela :

- Il faut ajouter la clause **NOT FINAL** à CREATE TYPE;
- créer un sous-type d'un type not final en rajoutant la clause **UNDER** nom-sur-type

Exemple

- CREATE **TYPE** T-Personne AS **OBJECT** (code NUMBER, nom VARCHAR(18), prénom VARCHAR(18), adrs T_adresse) **NOT FINAL.**
- CREATE **TYPE** T-Etudiant **UNDER** T-Personne (faculté VARCHAR(18),cycle VARCHAR(18)) ; //héritage
- L'utilisateur peut insérer des valeurs de type T-Personne ou de type T-Etudiant

Les méthodes (1)

❑ Déclaration d'une méthode

➢ ***Déclaration d'une fonction***

Member function < nom_fonction > [(<nom_paramètre> in <type>, ...)] **return** <type_du_resultat>

➢ ***Déclaration d'une procédure***

Member Procedure <nom_procedure>[(<nom_paramètre> in <type>, ...)]

❑ Définition du corps de la méthode:

Create **type body** <type-objet> as < declaration- méthode (fonction | procedure) > **is**

<Declaration var-locales>

Begin

<Corps de la méthode>

End;

Les méthodes (3)

méthodes: fonctions ou procédures écrites en PL/SQL ou Java et stockées dans la BD ou en C et stockées à l'extérieur de la BD

Exemple

Create **Type** Personne As Object (

nom Varchar2(10), nss Number, datenais Date,

member function age **return** Number) ;

Create **Type Body** Personne as

member function age **return** Number is
begin
return sysdate – datenais;
end;/

Les collections (1)

- Support de la construction d'objets complexes à partir de type de **collection** paramétrés.
- Les constructeurs de collections peuvent être appliqués sur tout type déjà défini.
- Pour représenter une colonne multivaluée, on peut utiliser les types de collection suivants:
  - L'ensemble (**SET**) POUR définir des collections non ordonnées sans doubles,
  - le sac (**BAG**) POUR définir des collections non ordonnées avec doubles,
  - la liste (**LIST**) POUR définir des collections ordonnées mais avec doubles.

Les collections (2)

Exemple

CREATE **TYPE** Adresse as **Object** (rue VARCHAR, code INT, localité VARCHAR);

CREATE **TYPE** Personne AS (nom VARCHAR,

prénom **LIST**(VARCHAR), adresses **SET**(Adresse));

modèle relationnel objet d'Oracle 9i et plus

- **Concepts NF^2 : structure complexe avec collection,**
- **Possibilité de référencer les données (OID)**
- **Programmation incluse dans le SGBD avec les méthodes (langage : PL/SQL)**

⇒ **Moins de jointure, références directes,**

⇒ **Meilleures performances (en théorie)**

- **héritage dans v. 9i et plus**