Université de Badji Mokhtar
Département Informatique

Base de Données Avancées

Master Réseau RSI

C. Cours: Djellali H. **Octobre 2019**

Chapitre 2: **L'OBJET-RELATIONNEL (OR) ET SQL3**

Ce chapitre

- Le modèle objet-relationnel
- Le Langage de requêtes associé SQL3.

INTRODUCTION

- Le développement des SGBD objet s'est rapidement heurté à la nécessité de conserver la compatibilité avec l'existant. En effet, la fin des années 80 a vu le déploiement des SGBD relationnels et des applications client-serveur. Les utilisateurs n'étaient donc pas prêts à remettre en cause ces nouveaux systèmes dès le début des années 90.
- Les tables relationnelles ne permettaient guère que la gestion de données alphanumériques. Avec le Web au milieu de la décennie 90, la nécessité de supporter des données complexes au sein de la base de données s'est amplifiée.
- Les Données Multimedia: Ces données complexes peuvent être des documents textuels, des données géométriques ou géographiques, audiovisuelles ou soniques.

Modèle objet-relationnel

Le modèle objet-relationnel (OR) reprend le modèle relationnel en ajoutant quelques notions qui comblent les plus grosses lacunes du modèle relationnel

La compatibilité est ascendante : les anciennes applications relationnelles fonctionnent dans le monde OR

- La norme SQL99 (SQL3) reprend beaucoup d'idées du modèle OR

2. POURQUOI INTEGRER L'OBJET AU RELATIONNEL ?

- Le modèle relationnel présente des points forts indiscutables, mais il a aussi des points faibles. L'objet répond à ces faiblesses. D'où l'intérêt d'une intégration douce.
- **2.1 Forces du modèle relationnel**
- Le relationnel permet tout d'abord une modélisation des données adaptée à la gestion à l'aide de tables dont les colonnes prennent valeurs dans des domaines alphanumériques.

Forces du modèle relationnel

- Grâce au langage assertionnel puissant que constitue le standard universel SQL2, à sa bonne adaptation aux architectures client-serveur de données, à sa théorie:
- La conception des bases (normalisation), l'optimisation de requêtes (algèbre, réécriture, modèle de coûts)
- La gestion de transactions (concurrence, fiabilité) intégrée, le relationnel s'est imposé dans l'industrie au cours des années 80.

2.2 Faiblesses du modèle relationnel

- Le relationnel présente cependant des faiblesses qui justifient son extension vers l'objet. Tout d'abord, les règles de modélisation imposées pour structurer proprement les données s'avèrent trop restrictives

Pourquoi étendre le modèle relationnel ? (1)

La reconstitution d'objets complexes éclatés sur plusieurs tables relationnelles est coûteuse car elle occasionne de nombreuses jointures

Pour **échapper aux éclatements-jointures**, l'OR réhabilite les références qui permettent d'implanter des structures complexes

Les attributs multivaluées (tableaux, ensembles

- ou listes)

Pourquoi étendre le modèle relationnel ? (2)

Même si ce problème n'est pas inhérent au modèle relationnel, SQL92 ne permet pas de créer de nouveaux types, ce qui implique un manque de souplesse et une interface difficile avec les applications orientées objet.

L'OR (et SQL99) permet de définir de nouveaux types utilisateur simples ou complexes (*User data type), avec des fonctions* ou procédures associées comme dans les classes des langages objet

Pourquoi étendre le modèle relationnel ? (3)

- L'OR supporte l'héritage de type pour profiter du polymorphisme et faciliter la réutilisation

Pourquoi ne pas passer directement aux SGBD Objet ? (1)

Le relationnel a ses avantages, en particulier sa grande facilité et efficacité pour effectuer des recherches complexes dans des grandes bases de données.

La facilité de spécifier des contraintes d'intégrité sans programmation une théorie solide et des normes reconnues.

Pourquoi ne pas passer directement aux SGBD Objet ? (2)

Inertie de l'existant : de très nombreuses bases relationnelles en fonctionnement

Manque de normalisation pour les SGBDO ;

Trop de solutions propriétaires

- SGBDOO moins souple que le relationnel pour s'adapter à plusieurs applications et à l'augmentation de charge
- Peu d'informaticiens formés aux SGBDO
- Le modèle OR peut permettre un passage en douceur

Faiblesses du modèle relationnel

- L'**absence de pointeurs visibles par l'utilisateur est très pénalisante.**
- Dans les architectures modernes de machine où la mémoire à accès directe est importante, il devient intéressant de chaîner directement des données. Le parcours de références est rapide et permet d'éviter les jointures par valeurs du relationnel.
- Des enregistrements volumineux représentant de gros objets peuvent aussi être stockés une seule fois et pointés par plusieurs lignes de tables : le partage référentiel d'objets devient nécessaire.

Exemple

- Une image sera stockée une seule fois, mais pointée par deux tuples, un dans la table Employés, l'autre dans la table Clients.

Faiblesses du modèle relationnel

- Le **non-support de domaines composés** imposé par la première forme normale de Codd est inadapté aux objets complexes,
- Exemple aux documents structurés. L'introduction de champs binaires ou caractères longs (**Binary Large Object — BLOB, Character Large Object — CLOB***) et plus généralement de champs longs (Large Object — LOB) est insuffisante.*

Objet long (*Large Object)*

- Domaine de valeurs non structurées constitué par des chaînes de caractères (CLOB) ou de bits (BLOB) très longues (pouvant atteindre plusieurs giga-octets) **permettant le stockage d'objets multimédia** dans les colonnes de tables relationnelles.
- Les objets longs présentent deux inconvénients majeurs :
- Ils ne sont pas structurés et ne supportent pas les recherches associatives.
- La seule possibilité est de les lire séquentiellement. Ils sont donc particulièrement insuffisants pour le stockage intelligent de documents structurés, où l'on souhaite accéder par exemple à une section sans faire défiler tout le document.
- **En clair, il faut pouvoir lever la contrainte d'atomicité des domaines, serait-ce que pour stocker des attributs composés**
- Exemple l'adresse composée d'une rue, d'un code postal et d'une ville) ou multivalués (par exemple les points d'une ligne).

Faiblesses du modèle relationnel

- La **non intégration des opérations dans le modèle relationnel constitue un manque.**
- L'approche traditionnelle sépare les traitements des données**. Certes, les procédures stockées ont été introduites dans le relationnel pour les besoins des architectures client-serveur,** mais celles-ci restent séparées des données.
- **Elles ne sont pas intégrées dans le modèle.** *Il est par exemple impossible de spécifier des attributs cachés de l'utilisateur, accessibles seulement via des opérations manipulant ces attributs privés.*

Faiblesses du modèle relationnel

- Tout ceci conduit à un mauvais support des applications non standard telles que la CAO, la CFAO, les BD géographiques et les BD techniques par les SGBD relationnels.
- En effet, ces applications gèrent des objets à structures complexes, composés d'éléments chaînés souvent représentés par des graphes.
- Le relationnel pur est finalement mal adapté.
- Il en va de même pour le support d'objets multimédia que l'on souhaite pouvoir rechercher par le contenu, par exemple des photos que l'on souhaite retrouver à partir d'un portrait robot.

Nouvelles possibilités de l'OR

- Définir **de nouveaux types complexes** avec des fonctions pour les manipuler
- Une colonne peut contenir une collection

(**ensemble, liste**)

- Ligne considérée comme un objet, avec un

identificateur (*Object Identifier OID)*

- Utilisation de références aux objets
- Extensions du langage SQL (SQL3) pour la recherche et la modification des données

2.3 L’apport des modèles objet

- Les modèles objet sont issus des langages objets. Ils apportent des concepts nouveaux importants qui répondent aux manques du relationnel,
- comme l’identité d’objets, l’encapsulation,
- l’héritage et le polymorphisme
- L’**identité d’objet permet l’introduction de pointeurs invariants pour chaîner les objets entre eux. Ces possibilité de chaînage rapide sont importantes pour les applications nécessitant le support de graphes d’objets. On pourra ainsi parcourir rapidement des suite d’associations, par navigation dans la base.**
- **Par exemple, passer d’un composé à ses composants puis à la description de ces composants nécessitera simplement deux parcours de pointeurs, et non plus des jointures longues et coûteuses.**

L’**encapsulation des**

- L’**encapsulation des données** permet d’isoler certaines données dites privées par des opérations et de présenter des interfaces stables aux utilisateurs.
- Facilite l’évolution des structures de données qui peuvent changer sans modifier les interfaces publiques seules visibles des utilisateurs. Au lieu d’offrir des données directement accessibles aux utilisateurs, le SGBD pourra offrir des services cachant ces données, beaucoup plus stables et complets. Ceux-ci pourront plus facilement rester invariants au fur et à mesure de l’évolution de la base de données.

L'**héritage d'opérations et de structures**

- L'**héritage d'opérations et de structures** facilite la réutilisation des types de données. Il permet plus généralement l'adaptation de composants à son application.
- Les composants pourront être des tables, mais aussi des types de données abstraits, c'est-à-dire un groupe de fonctions encapsulant des données cachées.
- Il sera d'ailleurs possible de définir des opérations abstraites, qui pourront, grâce au polymorphisme, porter le même nom mais s'appliquer sur des objets différents avec pour chaque type d'objet un code différent.
- Cela simplifie la vie du développeur d'applications qui n'a plus qu'à se soucier d'opérations abstraites sans regarder les détails d'implémentation sur chaque type d'objet.

2.4 Le support d'objets complexes

- L'apport essentiel de l'objet-relationnel est sans doute le support d'objets complexes au sein du modèle. Ceci n'est pas inhérent à l'objet, mais plutôt hérité des premières extensions tendant à faire disparaître la première forme normale du relationnel.
- On parle alors de base de données **en non première forme normale (NF).**

Non première forme normale (*Non First Normal Form - NF2)*

- Forme normale tolérant des domaines multivalués.
- Un cas particulier intéressant est le modèle relationnel imbriqué [Scholl86].

Modèle relationnel imbriqué (*Nested Relational Model)*

- Modèle relationnel étendu par le fait qu'un domaine peut lui-même être valué par des tables de même schéma.
- Par exemple, des services employant des employés et enregistrant des dépenses pourront directement être représentés par une seule table externe imbriquant des tables internes :
- **SERVICES (N° INT, CHEF VARCHAR, ADRESSE VARCHAR, {EMPLOYES (NOM, AGE)}, {DEPENSES (NDEP INT, MONTANT INT, MOTIF VARCHAR)})**

Exemple

- Employés correspond à une table imbriquée (ensemble de tuples) de schéma (Nom, Age) pour chaque service, et de même, Dépenses correspond à une table imbriquée pour chaque service.
- Notons que le modèle relationnel permet une imbrication à plusieurs niveaux, par exemple, Motif pourrait correspondre à une table pour chaque dépense.

Services

N°	Chef	Adresse	Employés		Dépenses		
			Nom	Age	NDep	Montant	Motif
24	Paul	Versailles	Pierre	45	1	2600	Livres
			Marie	37	2	8700	Mission
					3	15400	Portable
25	Patrick	Paris	Eric	42	5	3000	Livres
			Julie	51	7	4000	Mission

3. LE MODELE OBJET-RELATIONNEL

- Le modèle objet-relationnel est fondé sur l'idée d'extension du modèle relationnel avec les concepts essentiels de l'objet (voir figure XIII.3). Le coeur du modèle reste donc conforme au relationnel, mais l'on ajoute les concepts clés de l'objet sous une forme particulière pour faciliter l'intégration des deux modèles.

Extensions apportées au relationnel

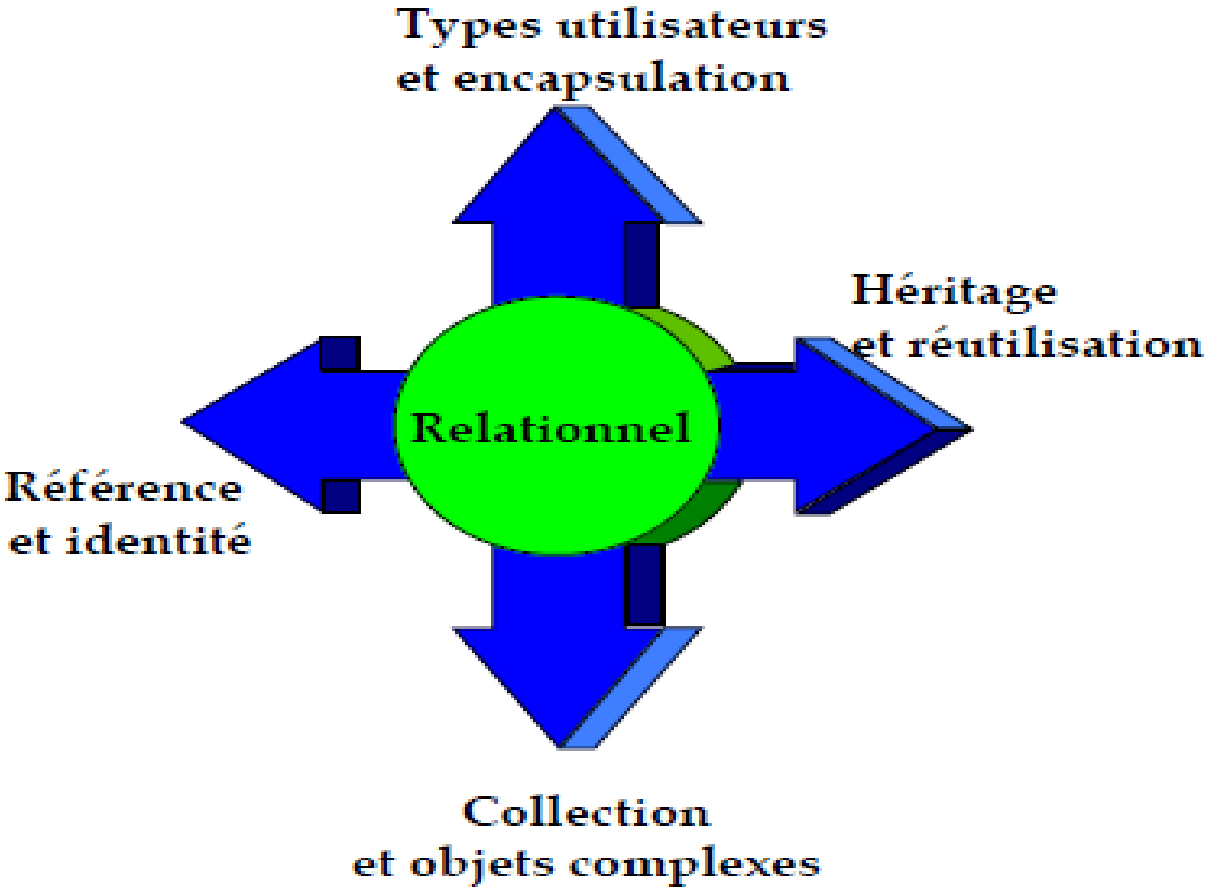

Figure XIII.3 — Les extensions apportées au relationnel

3.1 Les concepts additionnels essentiels

- Les types utilisateur permettent la définition de types extensibles utilisables comme des domaines, supportant l'encapsulation.

- **Type de données utilisateur (*User data type)***
- Type de données définissable par l'utilisateur composé d'une structure de données et d'opérations encapsulant cette structure.
- Le système de type du SGBD devient donc extensible, et n'est plus limité aux types alphanumériques de base, comme avec le relationnel pur et SQL2. On pourra par exemple définir des types texte, image, point, ligne, etc. Les types de données définissables par l'utilisateur sont appelés **types abstraits (ADT) en SQL3. La notion de type abstrait fait référence au fait que l'implémentation est spécifique de l'environnement : seule l'interface d'un type utilisateur est visible.**

Les concepts additionnels: **Patron de collection (*Collection template***

- Type de données générique permettant de supporter des attributs multivalués et de les organiser selon un type de collection.
- Les SGBD objet-relationnels offrent différents types de collections, tels que le tableau dynamique, la liste, l'ensemble, la table, etc. Le patron LISTE(X) pourra par exemple être instancié pour définir des lignes sous forme de listes de points : LIGNE LISTE(POINT).

Le modèle Objet Relationnel

- Deux vues sont possibles : la vue relationnelle dans laquelle les relations sont des relations entre objets (voir figure XIII.5), chaque colonne étant en fait valuée par un objet ; la vue objet (voir figure XIII.6) où une ligne d'une table peut correspondre à un objet simple (un tuple) ou complexe (un tuple avec des attributs complexes).

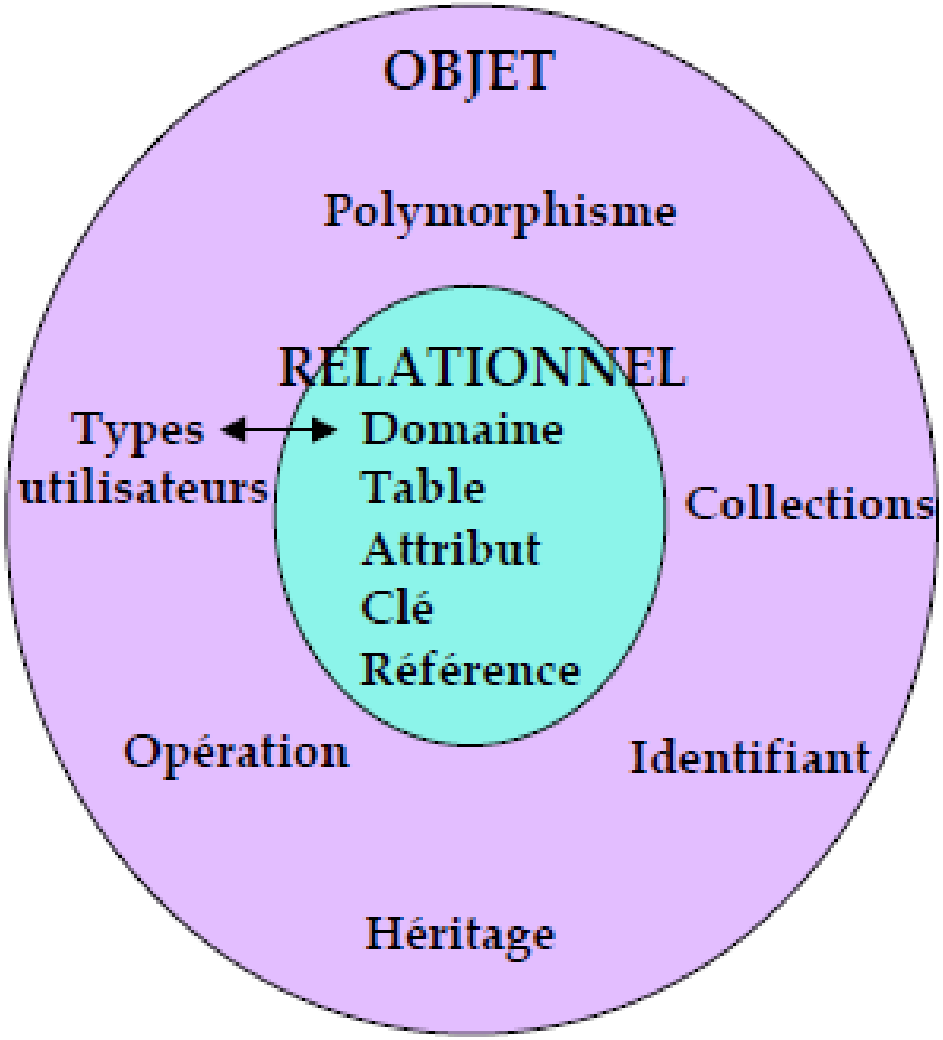

Figure XIII.4 — Les concepts de l'objet-relationnel

Les concepts additionnels:

Référence d'objet (*Object Reference)*

- Type de données particulier permettant de mémoriser l'adresse invariante d'un objet ou d'un tuple.
- Les références sont les identifiants d'objets (OID) dans le modèle objet-relationnel. Elles sont en théorie des adresses logiques invariantes qui permettent de chaîner directement les objets entre eux, sans passer par des valeurs nécessitant des jointures.

Les concepts additionnels: Héritage de type (*Type inheritance)*

- Forme d'héritage impliquant la possibilité de définir un sous-type d'un type existant, celui-ci héritant alors de la structure et des opérations du type de base.
- L'héritage de type permet donc de définir un sous-type d'un type SQL ou d'un type utilisateur. Une table correspondant à un type sans opération, l'objet-relationnel permet aussi l'**héritage de table.**

Les concepts additionnels: Héritage de table (*Table inheritance)*

- Forme d'héritage impliquant la possibilité de définir une sous-table d'une table existante, celle-ci héritant alors de la structure et des éventuelles opérations de la table de base.

En résumé

- le modèle objet-relationnel conserve donc les notions de base du relationnel (domaine, table, attribut, clé et contrainte référentielle) auxquelles il ajoute :
- Les concepts de type utilisateur avec structure éventuellement privée et opérations encapsulant, de **collections supportant des attributs multivalués (structure, liste, tableau, ensemble, etc.),**
- Héritage sur relations et types, et d'identifiants d'objets permettant les références inter-tables. Les domaines peuvent dorénavant être des types utilisateurs.

Accident	Rapport	Photo
134		
219		
037		

Figure XIII.5 — Une relation entre objets

Police	Nom	Adresse	Conducteurs	Accidents
24	Paul	Paris	Conducteur / Age: Paul 45; Robert 17	Accident / Rapport / Photo: 134, 219, 037

Objet Police

Figure XIII.6 — Un objet complexe dans une table

SQL3

- SQL3 comporte beaucoup d'aspects nouveaux, l'essentiel étant sans doute son orientation vers une intégration douce de l'objet au relationnel.
- Ce chapitre traite en détail de cette intégration et survole les autres aspects de SQL3.

SQL3

- SQL3 est une extension de SQL2, développée par le groupe de normalisation ANSI X3 H2 et internationalisée au niveau de l'ISO par le groupe ISO/IEC JTC1/SC21/WG3.

VUE D'ENSEMBLE DE SQL3

- SQL3 est une spécification volumineuse, qui va bien au-delà de l'objet-relationnel. Nous examinons dans cette section ses principaux composants.
- **4.1 Le processus de normalisation**
- Au-delà du groupe ANSI nord-américain très actif, la normalisation est conduite par un groupe international dénommé ISO/IEC JTC1/SC 21/WG3 DBL, DBL signifiant *Database Language.*
- *L'essentiel du travail est bien préparé au niveau du groupe ANSI X3H2 (http://www.ansi.org) . Ce groupe a déjà réalisé le standard SQL2 présenté dans la partie relationnelle. Les standards acceptés étaient contrôlés aux Etats-Unis par le NIST (http://ncsl.nist.gov) qui définissait des programmes de tests de conformité. Ceux-ci existent partiellement pour SQL2 mais pas pour SQL3.*

4.2 Les composants et le planning

- Alors que SQL2 était un unique document [SQL92], SQL3 a été divisé en composants pour en faciliter la lisibilité et la manipulation, l'ensemble faisant plus de 1.500 pages. Les composants considérés sont brièvement décrits dans le tableau représenté figure XIII.7.

4.3 La base

- Le composant 2 contient l'essentiel du langage SQL3, en particulier les fonctionnalités de base pour définir les autres composants (*Basic SQL/CLI capabilities , Basic SQL/PSM capabilities), les déclencheurs (Triggers), les types utilisateurs appelés types abstraits (Abstract Data Types) et les fonctionnalités orientées objets que nous allons étudier en détail ci-dessous.* les déclencheurs ne sont normalisés que dans SQL3. Il s'agit là de combler une lacune de la normalisation.

5. LE SUPPORT DES OBJETS

- Dans cette section, nous allons présenter l'essentiel des commandes composant les fonctionnalités objet de SQL3 :
- définition de types abstraits,
- support d'objets complexes,
- héritage de type et de table.
- Pour chaque commande, nous donnons la syntaxe simplifiée et quelques exemples.

Types définis par l'utilisateur

- Nouveaux types prédéfinis
- Le relationnel objet ajoute des types prédéfinis à la norme SQL (étudiés plus loin
- dans le cours) :
- **Référence**
- **Collection**
- **LOB (lié aux objets de grande taille)**

Les types utilisateur

- Le développeur peut aussi créer ses propres
- Types de données :
- Types « distincts »
- Types structurés

Création d'un type de données

La syntaxe est semblable à celle de la création

- d'une table :
- **CREATE TYPE departement_type AS OBJECT (numDept integer, nomD varchar(30), lieu varchar(30));**
- Les types créés par l'utilisateur peuvent être utilisés comme les types natifs SQL (integer, varchar,…)

Exemple

1. créez un type_adresse avec un numero de rue, un nom de rue et un nom de ville?
2. Créez un type departement_type sur le même modèle que la table département?
3. Créez un type employe_type avec un matricule , un nom, une adresse(de type adresse_type), un salaire, une reference à un superieur, une référence à un departement?

Exemple Création des Types

- **CREATE TYPE** adresse_type **AS OBJECT**

(numero integer, rue varchar(30), ville varchar(20))

- **CREATE TYPE** departement_type **AS OBJECT**

(numDept smallint, nomDept varchar(20), lieu varchar(50))

CREATE TYPE employe_type **AS OBJECT**

(matricule smallint, nom varchar(30),

adresse adresse_type, salaire decimal(8,2), superieur ref employe_type,

departement ref departement_type)

Création d'un type de données

- Un type ne peut contenir de contrainte d'intégrité.
- La commande « **create or replace type »**

  permet de redéfinir un type s'il existe déjà
- Une colonne ne peut avoir le nom d'un type ; il est donc conseillé de suffixer les noms de type avec « _type » ou « _t »

Fonctions dans les types

- **CREATE TYPE departement_type AS OBJECT**

(numDept integer,
nomD varchar(30),
lieu varchar(30),
MEMBER FUNCTION getLieu RETURN varchar);
CREATE TYPE BODY departement_type AS
MEMBER FUNCTION getLieu RETURN varchar IS
begin
return lieu;
end;
end;

Héritage

- Les types supportent l'héritage multiple avec le mot-clé UNDER :
- **create** type employe_type **as object**

(matr integer, nom varchar(30),

sal numeric(8,2)) not final;

- **create type commercial_type under employe_type (comm numeric(8,2)) not final;**
- Un type est **final par défaut**

Ajout d’un attribut dans un type

- **alter type** employe_type
- **add** attribute date_naissance **date cascade;**

Ajout d'une méthode/fonction à un type

- **alter type** employe_type

add member

function age return integer

cascade;

Supprimer un type

- **drop type employe_type;**

Types « distincts »

- Ces types permettent de mieux différencier les domaines des colonnes ; ils sont formés à partir des types de base :
- **CREATE TYPE codePays as char(2);**
- **CREATE TYPE matricule as integer;**
- Par exemple, pour différencier les domaines des colonnes **matricule et numDept**
- Ces types s'utilisent avec les mêmes instructions que le type de base sous-jacent
- Pas supporté par Oracle

Types structurés

- Correspondent aux classes des langages objets
- Ils peuvent contenir des constructeurs, attributs (≈ variables d'instances), fonctions et procédures (≈ méthodes)
- Les membres peuvent être public, protected ou Private
- Les fonctions et procédures peuvent être écrites en SQL ou en un autre langage
- Supportent l'héritage

5.1 Les types abstraits

- **5.1.1 Principes**
- La première nouveauté de SQL3 est l'apparition d'une commande **CREATE TYPE**.
- Au-delà des types classiques (numériques, caractères, date) de SQL, il devient possible de créer des types dépendant de l'application, multimédia par exemple .
- Une instance d'un type peut être un objet ou une valeur. Un objet possède alors un OID et peut être référencé . Une valeur peut simplement être copiée dans une ligne d'une table.
- Un type possède en général des colonnes, soit directement visibles, soit cachées et accessibles seulement par des fonctions encapsulant le type. Les opérateurs arithmétiques de SQL (+, *, -, /) peuvent être redéfinis au niveau d'un type.
- Par exemple, on redéfinira l'addition pour les nombres complexes. Enfin, un type peut posséder des clauses de comparaison (=, <) permettant d'ordonner les valeurs et peut être représenté sous la forme d'un autre type, une chaîne de caractères par exemple.

5.1.2 Syntaxe

- La syntaxe de la commande de déclaration de type est la suivante :
- **CREATE [DISTINCT] TYPE <NOM ADT> [<OPTION OID>] [<CLAUSE SOUS-TYPE>]**
- **[AS] (<CORPS DE L'ADT>)**
- Le mot clé DISTINCT est utilisé pour renommer un type de base existant déjà, par exemple dans la commande :

CREATE DISTINCT

- **CREATE DISTINCT TYPE EURO AS (DECIMAL(9,2)**
- La clause :
- **<OPTION OID> ::= WITH OID VISIBLE**
- permet de préciser la visibilité de l'OID pour chaque instance (objet). Par défaut, une instance de type est une valeur sans OID. Cette clause semble plus ou moins avoir disparue de la dernière version de SQL3.

- La clause :
- **<CLAUSE SOUS-TYPE> ::= UNDER <SUPERTYPE>[,<SUPERTYPE>]…**
- permet de spécifier les super-types dont le type hérite. Il y a possibilité d'héritage multiple avec résolution explicite des conflits.

Exemple de création de types

- Voici quelques exemples de création de types :
- 1. Type avec OID pouvant être utilisé comme un objet :

- **CREATE TYPE PHONE WITH OID VISIBLE (PAYS VARCHAR, ZONE VARCHAR,**
- NOMBRE **INT, DESCRIPTION CHAR(20))**

Exemple de création de types

- 2. Type sans référence :
- **CREATE TYPE PERSONNE (NSS INT, NOM VARCHAR, TEL PHONE)**

- 3. Sous-type :
- **CREATE TYPE ETUDIANT UNDER PERSONNE (CYCLE VARCHAR, ANNEE INT)**

- 4. Type énuméré :
- **CREATE TYPE JOUR-OUVRE (LUN, MAR, MER, JEU, VEN);**

5. Type avec OID et fonction

- **CREATE TYPE EMPLOYE WITH OID VISIBLE**
- **(NOM CHAR(10), DATENAIS DATE, REPOS JOUR-OUVRE, SALAIRE FLOAT,**
- **FUNCTION AGE (E EMPLOYE)RETURNS (INT)**
- { CALCUL DE L'AGE DE E }
- **END FUNCTION;);**

6. Autre type sans OID avec fonction

- **CREATE TYPE ADRESSE WITHOUT OID**
- **(RUE CHAR(20), CODEPOST INT, VILLE CHAR(10),**
- **FUNCTION DEPT(A ADRESSE) RETURNS (VARCHAR) END FUNCTION);**

Exemple de création

- 7. Création de procédure SQL de mise à jour associée au type employé :
- **CREATE procedure** AUGMENTER **(E EMPLOYE, MONTANT MONEY)**
- **{ UPDATE EMPLOYE** SET SALAIRE = SALAIRE + MONTANT WHERE EMPLOYE.OID = E**}**
- **END PROCEDURE**

Création d’une table à partir d’un type

- Les données d’un type ne sont persistantes

que si elles sont rangées dans une table

- On peut créer des tables comme en SQL92
- On peut aussi créer des tables à partir d’un
- type de données

Création d'une table à partir d'un type

Soit le type employe_type :

- **CREATE TYPE employe_type AS OBJECT**

(matricule integer, nom varchar(30),

- **. . .**
- **dept integer);**

- On peut créer une table à partir de ce type et indiquer des contraintes d'intégrité :
- **create table employe OF employe_type**
- **(primary key (matricule));**

Héritage de tables

- Une table peut hériter d’une ou plusieurs

Tables

- Pas supporté par Oracle 10g

Création de table à partir d'un type dérivé

- **create** table commerciaux **of commercial_type (constraint** pk_com **primary key(**matr**));**

Caractéristiques d'une table objet-relationnelle

- Une table est une table objet-relationnelle si

elle a été construite à partir d'un type

(**create table … OF)**

- Les lignes de ces tables sont considérées

comme des objets avec un identifiant (OID,

Object IDentifier)

- On peut utiliser des références pour désigner

les lignes de ces tables (pas possible pour les autres tables)

Insertion de données

- On ajoute des données comme avec une table normale :
- **insert into** commerciaux **(matr, nom, sal, comm) values** (234, 'TITI', 3200, 600);

Insertion avec constructeur

- On peut aussi utiliser le « constructeur du type »

avec lequel la table a été construite :

- **insert into employe values (employe_type(125, 'Dupond', ...));**
- Si le type est un type utilisé par un autre type,

l'utilisation du constructeur du type est obligatoire :

- **insert into employe (matr, nom, sal, adresse)**

values (1, 'Toto', 12000, adresse_type(12, 'Victor Hugo', 'Nice'))

Afficher les valeurs des types

- **select** nom, e.adresse.rue **from** employe e

Modifications

Utiliser la notation pointée comme en SQL92 mais avec un alias si un type est concerné :

- **update employe e set salaire = 12000,**
- **e.adresse.numero = 23**
- **where nom = 'Dupond';**

- SQL99 utilise la notation « .. » pour désigner un attribut d'une colonne de type structuré :
- **update employe e**
- **set e.adresse..numero = 12**
- **where nom = 'Dupond';**

Appel de procédure ou fonction

select nom, age(e)

from employe e

where age(e) < 40

- Sous Oracle :

select nom, e.age()

from employe e

where e.age() < 40

Références

- On peut indiquer dans la définition d'un type

qu'un attribut contient des références (et non des valeurs) à des données d'un autre type ;

- La syntaxe est « REF nom-du-type » :

create type employe_type **as object**

(matricule integer, nom varchar(30),

. . .

dept **REF** dept_type**);**

Exemple de select avec référence

- La notation pointée permet de récupérer les attributs d'un type dont on a un pointeur
- Lieu de travail des employes (avec Oracle) :

select nom, e.dept.lieu from employe e

- En SQL99 :

select nom, e.dept->lieu from employe e

- Attention, l'alias **e est indispensable**

Insertions avec référence

- **insert into employe values (**

1230, 'Durand', …, NULL); (pointeur NULL)

insert into employe(matricule, nom, dept)

select 1240, 'Dupond', REF(d)

from dept d

where d.numDept = 10; (reference vers departement N°10)

DEREF

- La fonction **DEREF renvoie un objet dont on a** la référence (penser à tester si la référence n'est pas **NULL)**
- Exemple :

select deref(dept) from emp
where matricule = 500

- Affiche

DEPARTEMENT_TYPE(10,'Finances','Nice')

Collections

- Types de collections
- Pour représenter une colonne multivaluée, on peut utiliser les collections ou les tableaux :
- tableaux de taille fixe (array)
- ensembles, au sens mathématiques ; pas de doublons (set)
- sacs, avec des doublons (bag ou multiset)
- listes, ordonnées et indexées par un entier (list)
- D'autres types de collections peuvent être

ajoutées par les SGBD

Exemple de collection

- **create type employe_type**

(matricule integer, nom varchar(30),

prenoms LIST(varchar(15)),

enfants SET(personne),

. . .);

Utilisation d'une collection

- On peut utiliser une collection comme une table en la faisant précéder par le mot-clé TABLE :
- **select nom from employe E**

where nom in

(select * from TABLE(E.prenoms))

On peut aussi faire afficher une collection

comme un tout :

select nom, prenoms from employe

Les collections avec Oracle 10g

- Oracle 10g n'offre que 2 types de collections :
- Table imbriquée (NESTED TABLE) qui est

une collection non ordonnée et non limitée en nombre d'éléments

- Tableau prédimensionné (VARRAY) qui est

une collection d'éléments de même type, ordonnée et limitée en taille

- Tableaux dimensionnés
- Un VARRAY est une collection ordonnée et
- limitée en nombre, d’éléments d’un même
- type
- On peut imbriquer plusieurs tableaux
- dimensionnés en utilisant des pointeurs sur
- des tableaux

Exemple de VARRAY

- **create type telephones_type as**

VARRAY(3) OF varchar(10)

create type personne_type **as object**

(nom varchar(30), telephones telephones_type)

- **insert into personne** (nom, telephones)

values('Dupond',telephones_type('0492077987', '0492074567'))

Référence

- Programmer objet avec Oracle

de Christian Soutou

Vuibert