REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE 08 MAI 1945 -GUELMA-
FACULTE DES MATHEMATIQUES, INFORMATIQUE ET SCIENCES DE LA MATIERE
DEPARTEMENT D'INFORMATIQUE

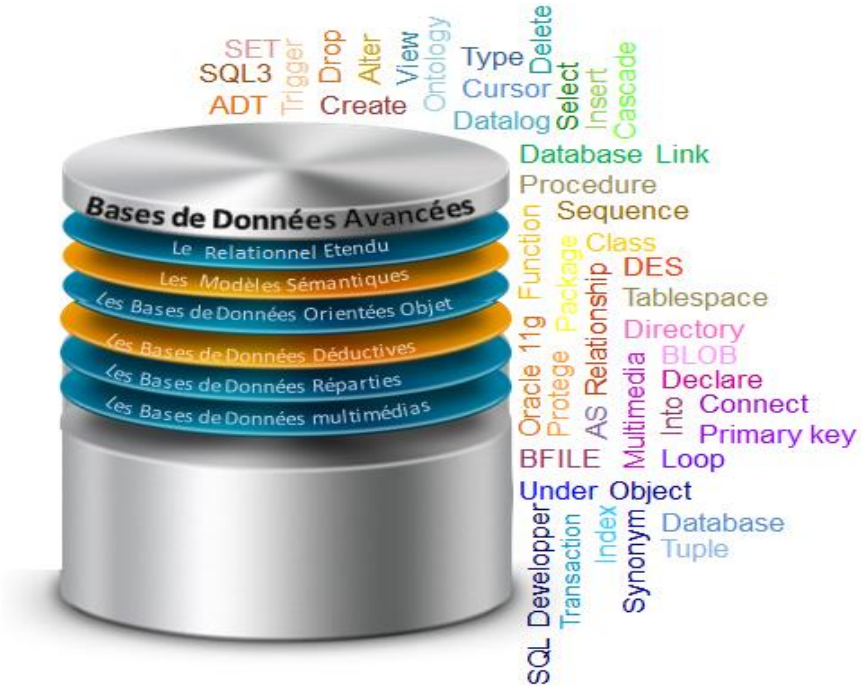
Polycopié de cours

Bases de Données Avancées

Niveau : Master 2 Systèmes Informatiques

Réalisé par :

Dr. Aicha AGGOUNE



2019-2020

Préface

Ce polycopié du cours de bases de données avancées s'adresse aux étudiants de deuxième années master systèmes informatiques qui est offert au département d'informatique de la faculté de mathématiques, informatique et sciences de la matières à l'université 08 Mai 1945, Guelma, Algérie.

Le module de bases de données avancées est programmé dans l'unité d'enseignement fondamentale UEF3. Il est caractérisé par un coefficient égal à 3 et un crédit soit 5. Ce module s'organise selon trois séances par semaine : un cours magistral, une séance de travaux dirigés (TD) et une séance de travaux pratiques (TP), complétées par un travail personnel de 5 heures qui englobe principalement sur la micro interrogation et la validation de projet final du TP sur le SGBD Oracle. La durée d'enseignement du module de bases de données avancées est environ $67^{h}30$. *La pondération de ce module est comme suit : 60% Examen final, 20% TD (inclus une micro interrogation, l'assiduité et la participation), 20% TP (inclus un projet final, l'assiduité et la moyenne des TPs fourni durant les séances de TP).*

Les bases de données sont en effet des structures de stockage de données inévitable en informatique et servant de support à une application de gestion. La création et la manipulation de bases de données sont effectuées par un système de gestion de base de données (SGBD) qui peut être vu comme un ensemble de logiciels et de langages permettant aux utilisateurs de manipuler efficacement des données de la base de données. L'évolution croissante des matériels informatique et de la taille de données avec leur complexité ont induit des évolutions au niveau de modélisation et de fonctionnement de bases de données. Les bases de données avancées sont apparues pour d'une part, cohabiter avec l'évolution de technologie informatique et de la croissance de taille et de complexité de données, et d'autre part, de combler les limites des bases de données relationnelles.

L'objectif principal du cours de bases de données avancées est d'étudier les notions et les fonctions avancées sur les bases de données. Il vise également à connaître des nouveaux modèles de données tels que le modèle relationnel étendu, le modèle orienté objet, le modèle sémantique, le modèle déductif, etc. Ce cours requiert des prérequis suivants : une connaissance préalable au domaine de base de données

relationnelle, l'approche orientée objets, la programmation logique et les systèmes répartis.

Le polycopié de ce cours est structuré en six chapitres, chacun se termine par des séries d'exercices TD et TP. Ces chapitres sont décrits ci-après :

Le premier chapitre s'intitule « Le relationnel étendu » représente une extension du modèle relationnel proposé par Edgar Frank CODD afin de combler ses limites. Cette extension du modèle relationnel vise à intégrer les propriétés de l'approche orientée objet dans le modèle relationnel. Les bases de données sont dites dans ce cas « les bases de données objet-relationnelles » ou encore les bases de données relationnelles-objet fondées sur le langage SQL3. Le SGBD Oracle 11g Express Edition va être étudié dans les séances de TP.

Le deuxième chapitre « Les modèles sémantiques» présente les différents modèles sémantiques utilisés pour définir des connaissances dans les bases de données. Les ontologies est le modèle sémantique à apprendre pour exprimer la sémantique aux bases de données. Ce chapitre présente quatre langages ontologiques : RDF, RDFS, OWL et SPARQL. La manipulation des ontologies via l'éditeur Protégé sera l'objet de série TP.

Le troisième chapitre « Les bases de données orientées objet» présente une deuxième solution aux limites du modèle relationnel. Cette solution est purement orientée objet. Deux types de modélisation vont être décrits : UML et ODMG. Le modèle ODMG va être étudie avec ses deux langages de définition et de manipulation de données qui sont respectivement ODL et OQL.

Le quatrième chapitre « Les bases de données déductives » introduit un nouveau type de base de données capable de définir des faits et un ensemble de règles déductives qui peuvent inférer des nouvelles connaissances à partir des faits. Ce chapitre fait la connaissance au langage de manipulation de BD déductives appelé Datalog. Une série de TP est consacrée à la manipulation de SGBD déductif DES.

Le cinquième chapitre « Les bases de données réparties» illustre comment manipuler les bases de données logiquement liées et physiquement localisées sur différents sites et perçues par l'utilisateur comme une base unique. Ces bases de données sont appelées "Bases de données réparties". Ce chapitre présent deux approches de conception de BD réparties : une approche ascendante et une approche descendante. Il

focalise sur l'approche descendante par la présentation de deux principales fonctions : la fragmentation et la réplication. Le chapitre termine par la définition de traitement des requêtes réparties et la gestion des transactions réparties. Le SGBD Oracle 11g va être étudié dans les séances de TP pour la manipulation de bases de données réparties.

Le sixième chapitre « Les bases de données multimédias » traite les bases de données multimédias (BDMM). Nous nous focalisons sur l'utilisation de l'excellent SGBD Oracle pour la définition et la manipulation de ce type de données par les types large objet (LOB) tels que BLOB et BFILE.

Le présent polycopié se termine par une section qui regroupe les solutions de toutes les séries de TD et TP proposées dans ce cours.

Aggoune.aicha@univ-guelma.dz

AVANT-PROPOS

Le présent polycopié du cours de bases de données avancées a été rédigé par Dr. Aicha AGGOUNE, Maître de conférences à l'université 08 Mai 1945, Guelma, et membre du laboratoire LabSTIC, équipe TWSI, Guelma. Dr. Aicha AGGOUNE commence à enseigner ce cours dans l'année universitaire 2012/2013 au niveau du département d'informatique.

Ce polycopié est le fruit d'une expérience des années d'enseignement du module bases de données avancées destiné particulièrement aux étudiants de deuxième année master systèmes informatiques (anciennement master académique). Il aborde toutes les dernières tendances dans le domaine de bases de données suivant le canevas d'enseignement fourni par le département d'informatique. Chaque chapitre traite un modèle de données particulier. Les exemples donnés dans ce cours sont tous validés par un SGBD le plus important est le SGBD Oracle. De même, les solutions des séries de TD sont aussi validées et présentées par des captures d'écrans.

Finalement, ce polycopié de cours répondra très certainement à l'attente des étudiants qui se destinent à l'informatique, aux ingénieurs, aux informaticiens, aux enseignants en informatique et aux professionnels utilisateurs et concepteurs des systèmes d'information avec une expérience pratique et théorique en matière de bases de données avancées.

Table des Matières

CHAPITRE 1. LE RELATIONNEL ÉTENDU

CHAPITRE 2. LES MODÈLES SÉMANTIQUES

LES SOLUTIONS DES SÉRIES DE TD ET DE TP

Liste des Figures

CHAPITRE 01

LE RELATIONNEL ÉTENDU

1. Introduction

Dans ce premier chapitre, nous étudierons le modèle objet-relationnel qui vise à remédier les limites du modèle relationnel. Il s'agit d'une extension du modèle relationnel intégrant les caractéristiques de l'approche orientée objet.

Ce chapitre présente dans un premier temps un bref rappel sur les bases de données. Il présente également les limites du modèle relationnel. Le reste du chapitre est consacré à la présentation du modèle relationnel étendu (ou objet-relationnel). Ce cours est enrichi par des exemples en utilisant le meilleur SGBD professionnel "Oracle" version 11g Express Edition.

2. Bref rappel sur les bases de données

Georges Gardarin, définit une base de données comme étant « *un ensemble de données modélisant les objets d'une partie du monde réel avec le moins de redondance possible et servant de support à une application informatique*» [4].

Le modèle le plus populaire pour la description de bases de données (BD) est le modèle relationnel qui a été proposé en 1970, par l'informaticien britannique Edgar Frank Codd [5].

Le modèle relationnel est fondé essentiellement sur la théorie des ensembles dont les tables relationnelles sont vues comme des ensembles et l'utilisation des opérateurs ensemblistes sur ces tables (intersection, union, produit cartésien et la différence). Il utilise également des opérateurs propres aux bases de données telles que la sélection, la projection, la jointure et la division [6].

Le SQL (Structured Query Language) est un langage déclaratif de manipulation de bases de données, doté de quatre sous langages [7, 8] :

1. **Langage de définition de données (LDD) :** il permet de créer et de modifier la structure de données (BD, table relationnelle, vues, index, etc.) par l'utilisation de trois commandes qui sont: CREATE (création), ALTER (modification) et DROP (suppression).

2. **Langage de manipulation de données (LMD) :** il permet de rechercher (SELECT), d'ajouter (INSERT), de modifier (UPDATE) ou de supprimer des données (DELETE).
3. **Langage de contrôle de données (LCD) :** il permet d'autoriser (GRANT) ou d'interdire (REVOKE) l'accès à certaines données aux personnes précises.
4. **Langage de contrôle des transactions (LCT) :** il vise à valider ou annuler une transaction (une suite d'opérations sur les données) en cours par les commandes COMMIT et ROLLBACK respectivement.

Le SQL3 est une extension de SQL2 développée par le groupe de normalisation ANSI X3 H2 et internationalisé au niveau de l'ISO par le groupe ISO/IEC JTC1/SC21/WG3 [9]. Il est adopté depuis 1999 des nouveaux aspects, tels que l'intégration de l'objet au relationnel et la représentation de données multimédias via les types de larges objets, etc. [9].

3. Evolutions des SGBD

Un SGBD est un ensemble de programmes assurant le stockage, la modification, l'accès et la maintenance de données de la BD indépendamment des programmes d'application [4].

Avec la croissance importante du volume de données et de la diversité de type de données, des évolutions ont été apportées sur les SGBD sont illustrées dans la figure suivante.

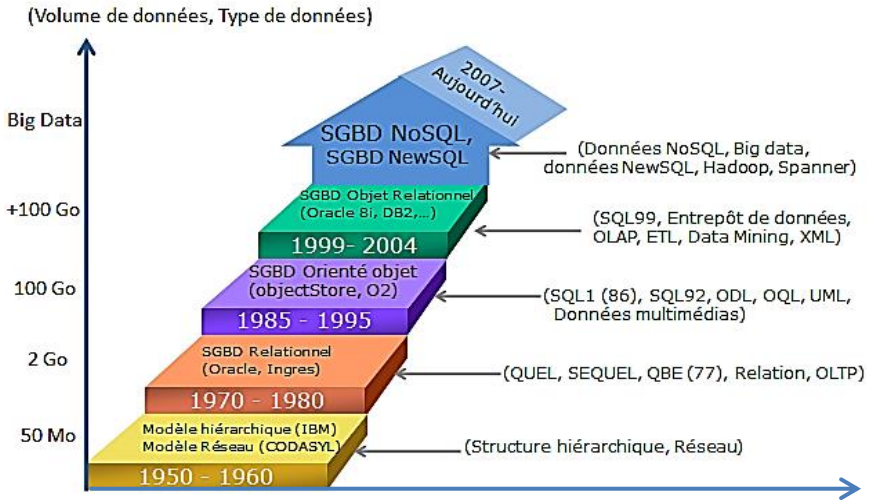


Figure 1.1. Evolutions des SGBD

4. Limites du modèle relationnel

Le modèle relationnel bien que reste le modèle le plus puissant et le plus utiliser dans le monde professionnel, des limites ont été relevées, les plus importantes sont [7, 9, 10]:

1. ***Structure de données simple* :** une BD relationnelle est un ensemble de tables reliées entre elles par des relations et qui suivent la 1ère forme normale de Codd.
2. ***Pas de référence entre tables :*** le modèle relationnel est basé sur l'utilisation des tables reliées entre elles via l'opération de jointure.
3. ***Pauvreté du système de typage :*** le modèle relationnel ne dispose pas un moyen de définir des types de données propres à un utilisateur.
4. ***Programmation interdite dans le SGBD:*** le modèle relationnel est doté du langage SQL qui ne permet pas de réaliser des programmes.
5. ***Sémantique insuffisante :*** La seule solution existante pour définir la sémantique de données est d'utiliser les contraintes d'intégrités.

Pour pallier les lacunes du modèle relationnel, deux solutions ont été proposés : la première appelée modèle relationnel étendu et la deuxième solution appelée le modèle orientée objet qui sera présentée dans le chapitre 03.

5. Le modèle relationnel étendu

Le modèle relationnel étendu, souvent appelé modèle objet-relationnel, est une extension du modèle relationnel qui a pour but de remédier aux limites du modèle relationnel. Il vise à combiner les avantages du modèle relationnel avec ceux de l'approche orientée objet.

5.1. Avantages du modèle relationnel

Le modèle relationnel présente des limites, mais il a aussi des avantages, en particulier [7, 9]: Bon support théorique, Simplicité des concepts et du schéma, Langage d'interrogation déclaratif, Haut degré d'indépendance des données, Optimisation des accès à la BD, Bonnes performances et la gestion de contraintes d'intégrité.

5.2. Définition d'une base de données objet-relationnelle

Une base de données objet-relationnelle (BDOR) se définit comme suit [4]:

- Les tables ou relations ne suivent pas la 1ère forme normale.
- Création de nouveaux types de données défini par l'utilisateur appelés ADT.
- Association aux types de données ADT des programmes.
- Identité d'objet et utilisation de référence.
- Héritage entre types objets.
- Compatibilité ascendante conservée.

6. Les types abstraits de données (ADT)

Le modèle relationnel étendu introduit la notion de type abstrait de données baptisé ADT (Abstract Data Type) pour définir de nouveaux types dépendant de l'application [4, 9].

```
CREATE [OR REPLACE] TYPE [nom schema.]<nom ADT> <corps>;/
```

Le mot clé **REPLACE** est utilisé pour modifier un type existant sans le supprimer et le recréer à nouveau. Le corps du type est représenté d'une manière générale comme suit :

```
AS <Type ADT> | UNDER <super type>
 (<nom attribut1>   type d'attribut1, <nom attribut2>   type
d'attribut2, …
 [<méthodes >
 Spec sous-programmes, Spec  constructeurs, Spec fonction
d'ordre map,  Spec pragma ])
<FINAL | NOT FINAL> <INSTANTIABLE | NOT INSTANTIABLE>;
```

Il existe en général quatre types d'ADT : objet (OBJECT), tableau (VARRAY), table imbriquée (NESTED TABLE) et type référence (REF).

6.1. Type OBJECT

Le type OBJECT permet de créer des valeurs structurées sous forme des couples <valeur, oid> dont la valeur est une données et l'oid désigne l'identificateur d'objet [9].

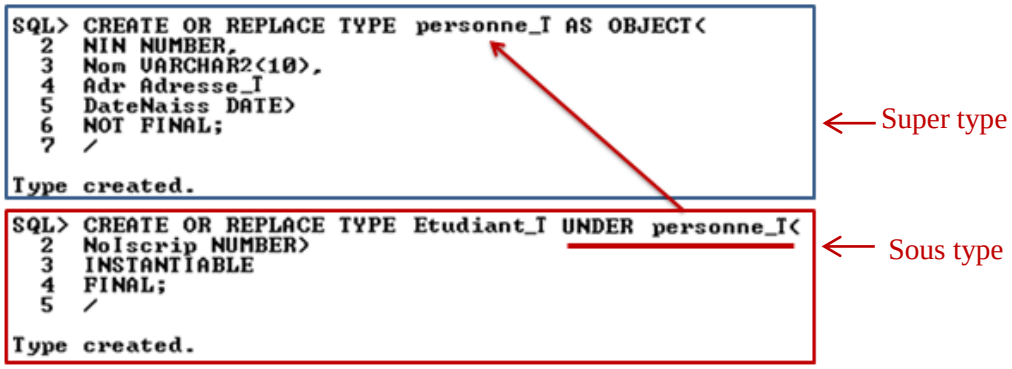
La figure suivante illustre la création de type adresse.

Adresse_T			
Num	**Rue**	**Ville**	**Pays**

```
SQL> CREATE OR REPLACE TYPE Adresse_T AS OBJECT(
  2  Num INTEGER,
  3  Rue VARCHAR2(40),
  4  Ville VARCHAR2(40),
  5  Pays VARCHAR2(40));
  6  /

Type created.
```

Figure 1.2. Exemple de création du type OBJECT

Un super type d'un sous type (Héritage), par exemple le type "Etudiant_T" hérite le type "Personne_T".



Remarque :

- Le type abstrait ne possède qu'un seul super type (héritage simple).
- Un type hérite les méthodes et les structures de données de son super type.
- On peut assurer des imbrications d'héritage de types (type2 hérite type1 et type 3 hérite le type 2, etc.)

6.2. Type VARRAY

Le type VARRAY (Variable size Array) ou tableau dynamique, permet de définir des collections de taille limitée de données ordonnées, indexées avec les doubles [11]:

```
CREATE [OR REPLACE] TYPE <nom-type1> AS VARRAY (nb-max)
```

Exemple : La création de type Prenoms de chaîne de caractères est faite comme suit:

```
SQL> CREATE OR REPLACE TYPE Prenoms AS VARRAY (4) OF VARCHAR2(20);
  2  /

Type created.
```

6.3. Type NESTED TABLE

Le type NESTED TABLE ou table imbriquée, correspond à des collections non ordonnées et non limitées en taille avec des doubles [4, 11]. La création de type NESTED TABLE est faite par le mot clé TABLE OF.

```
CREATE TYPE nom_type1 AS TABLE OF nom_type2
```

Exemple 01: on crée le type nested table "ens_diplomes" à partir le type "diplomes".

```
SQL> CREATE TYPE Diplome AS OBJECT(
  2  Specialite VARCHAR2(60),
  3  Type VARCHAR2(40),
  4  Mension VARCHAR2(40), Annee NUMBER(4));
  5  /

Type created.

SQL> CREATE TYPE ens_diplomes AS TABLE OF Diplome;
  2  /

Type created.
```

Exemple 02: création de table d'objets avec la table imbriquée Nested table.

```
SQL> CREATE TYPE Etudiant_T AS OBJECT(
  2  Num NUMBER, Nom VARCHAR2(40), Prenom Prenoms, diplomes ens_diplomes);
  3  /

Type created.

SQL> CREATE TABLE Etudiant  OF Etudiant_T
  2  NESTED TABLE diplomes STORE AS Lesdiplomes;

Table created.
```

Figure 1.3. Exemple de création du type NESTED TABLE

Remarque :

Une Nested Table n'est pas une table objet ; elle ne possède pas d'OID.

6.4. Type REF

Le type REF d'un attribut dit "attribut référence" ayant comme valeur l'identificateur d'objet OID. Il permet de référencer une ou plusieurs instances d'un type objet afin de réduire les jointures entre tables [4, 11]. Ce type conserve dans les tables une référence plutôt qu'une clé étrangère.

Exemple :

```
SQL> CREATE OR REPLACE TYPE Poste_T AS OBJECT(
  2  Num INTEGER,
  3  SE VARCHAR2(40),
  4  AdresseIP VARCHAR2(15));
  5  /

Type created.

SQL> CREATE OR REPLACE TYPE  Etudiant_T AS OBJECT(
  2  NO INTEGER, Nom VARCHAR2(50), Prenom VARCHAR2(50),
  3  Niveau VARCHAR2(10), Groupe INTEGER, PC REF Poste_T);
  4  /

Type created.
```

Figure 1.4. Exemple de création du type REF

Dans la définition de l'attribut de type REF, on peut limiter la portée de référence à une table particulière en utilisant la commande SCOPE IS ou REFERENCES le nom de la table référencée [3].

7. Le schéma relationnel étendu

Le schéma relationnel étendu décrit les types ADT et les relations à travers le langage SQL [4]. Nous distinguons deux modes d'associations [9, 10]:

- **Association symétrique :** assure une *imbrication partielle* entre types. Dans ce mode, deux relations sont dominantes. Il produit deux tables avec des références entre elles.
- **Association agrégation :** dans ce mode d'association, l'un des types ADT domine l'autre. Ce mode assure l'imbrication totale qui produit une seule table.

Dans ce cours, nous restreignons le choix d'une telle association (symétrique ou agrégation) selon le problème et les données à modéliser.

Exemple : soit le modèle Entité/Association modélisant une personne et sa possession d'une voiture. La figure suivante illustre la création de ce schéma étendu qui se définit par une association symétrique entre une personne et sa voiture (création de deux tables).

Création des ADT

Création des Tables

```
SQL> CREATE OR REPLACE TYPE Voiture_T;
  2  /

Type created.

SQL> CREATE OR REPLACE TYPE Ens_Voitures AS TABLE OF REF Voiture_T;
  2  /

Type created.

SQL> CREATE TYPE Personne_T AS OBJECT(Num INTEGER, Nom VARCHAR2(40),
  2  DateP DATE, Voitures Ens_Voitures);
  3  /

Type created.

SQL> CREATE TYPE Voiture_T AS OBJECT (Matricule INTEGER,
  2  Modele VARCHAR2(40), Marque VARCHAR2(40), Possesseur REF Personne_T);
  3  /

Type created.

SQL> CREATE TABLE Personne OF Personne_T(Num Primary key)
  2  NESTED TABLE Voitures STORE AS Lesvoitures;

Table created.

SQL> CREATE TABLE Voiture OF Voiture_T(Matricule Primary key);

Table created.
```

Figure 1.5. Exemple du schéma relationnel étendu

8. Langages de définition et de manipulation de données objet-relationnelles

8.1.Langage de définition de données

Dans le modèle relationnel étendu, nous distinguons trois types de tables de données :

- ***Table relationnelle :*** une table classique qui contient des données simple (sans ADT).

- ***Table NF²:*** une table NF² (Not First Normal Form) contenant des attributs complexes.
- ***Table d'objets :*** est une table créée à partir de type OBJECT.

La création d'une table d'objet est faite par la commande suivante [4] :

```
CREATE TABLE nom_table OF Type_objet (Contraintes d'intégrité)
[NESTED TABLE attribut-multivalué STORE AS Nom-Table-Annexe];
```

La modification d'une table relationnelle classique ou table NF² est faite par la commande **ALTER TABLE** nom-table <Action>. Sachant que <Action> peut être : renommer, supprimer (drop), ajouter (ADD) et modifier (MODIFY) les éléments de la table.

Modifier les attributs et méthodes d'une table d'objet revient à modifier son type d'objet par la commande **ALTER TYPE** nom-type <Action>. Avec <Action> permet de : **Ajouter, Modifier, Supprimer** et **Mettre à jour** des attributs, méthodes [4].

Exemple : ajouter au type adresse_T deux attributs "CodePostal" et "Code".

```
SQL> ALTER TYPE Adresse_T ADD ATTRIBUTE(CodePostal NUMBER(5), Code VARCHAR2(50)) CASCADE;

Type altered.
```

La suppression d'une table est faite par la commande :

```
DROP TABLE nom_table ;
```

La suppression d'un type est faite par la commande :

```
DROP TYPE nom_type ;
```

8.2. Langage de manipulation de données

Le langage de manipulation de données LMD vise à effectuer les opérations d'insertion, de suppression, de modification et de recherche de données [4, 6, 9].

8.2.1. Insertion de données

La syntaxe générale d'insertion de données est la suivante :

```
INSERT INTO nom_table1
[(nom-attribut1, nom-attribut2, ….)]
VALUES (valeur1, valeur2, …)| SELECT *| attributs
FROM nom-table2 WHERE conditions
```

Exemple : L'insertion de l'employé FAREH Ahmed dans la table employe est :

```
SQL> INSERT INTO Employe
  2  VALUES (Employe_T(115, 'FAREH', 'Ahmed',
  3  Adresse_T(01, 'Rue 08 MAI 45', 'Guelma')));

1 row created.
```

Dans cette expression, on peut insérer un nouvel objet sans utiliser son constructeur du type d'objet (utilisation facultative) comme le constructeur Employe_T. Par contre, l'utilisation du constructeur de type objet d'un attribut est obligatoire (cas de l'attribut Adresse de type Adresse_T).

Dans le cas de l'insertion d'un tuple dans la table imbriquée, on utilise la clause **TABLE**.

Exemple : Ajouter à la personne "Djaber", une nouvelle voiture.

```
SQL> INSERT INTO TABLE(SELECT P.Voitures FROM Personne P
  2  WHERE P.Nom='Djaber') V
  3  VALUES(123456789, 'TOYOTA', 'GT86');

1 row created.
```

Dans le cas d'insertion d'un tuple dans une table contenant un attribut de type VARRAY, on procède de la même manière que le NESTED TABLE.

Exemple :

```
SQL> INSERT INTO Person VALUES(593392, 'FATEHI', Prenoms('Mohammed', 'Omar'),
  2  '25-07-1955', Adresse(1, 'Che guevara', 'Guelma', 'Algerie', 24000, 24));

1 row created.
```

L'insertion d'un n-uplet d'une table contenant un attribut référence (REF) sert à récupérer l'OID de l'objet de la table référencée par l'utilisation d'une requête SELECT.

8.2.2. Suppression de données

La suppression de données est exprimée par la syntaxe générale suivante :

```
DELETE [FROM] [Nom-user.] nom_table1 [alias]
[WHERE conditions];
```

Exemple 01:

```
SQL> DELETE BDA.Fonctionnaire F
  2  WHERE F.Num=1;

1 row deleted.
```

Exemple 02 : supprimer de personne Djaber les voitures de modèle "yaris".

```
SQL> DELETE FROM TABLE(SELECT P.Voitures FROM Personne P WHERE P.Nom='Djaber')V
  2  WHERE V.Modele='yaris';

1 rows deleted.
```

8.2.3. Modification de données

La modification ou la mise à jour de valeur d'un tuple d'une table est faite comme suit :

```
UPDATE [Nom-user.] nom_table1 [alias] SET nom-attribut1=valeur1
[, nom-attribut2=valeur2,...] [WHERE condition];
```

Exemple :

```
SQL> UPDATE Employe
  2  SET Nom='HADRI', Prenom='Mohamed'
  3  WHERE NumE=115;

1 row updated.
```

8.2.4. Recherche de données

Le langage de manipulation de données est muni d'un langage d'interrogation de données LID qui permet de rechercher et d'interroger les bases de données par l'utilisation de requêtes SPJ (SLEECT PROJECT JOIN) exprimées par la syntaxe générale suivante :

```
SELECT [DISTINCT] {*|<Alias>.<nom type ADT>.<nom type>,[…]}
FROM <nom de table> <Alias> [,…]
[WHERE <conditions>]
[GROUP BY attribut [,…]
[HAVING <conditions>]
[{UNION [ALL]} |INTERSECT|MINUS requête SELECT]
[OREDER BY {attribut|numéro colonne} [{ASC|DESC}][,…]
```

Exemple : afficher la marque des voitures de toutes les personnes.

```
SQL> SELECT V.column_value.Marque FROM Personne P, TABLE(P.Vehicule) V;

COLUMN_VALUE.MARQUE
--------------------------------------------------
Renault
VW
```

Pour la manipulation des attributs références, nous distinguons deux opérations :

- **REF** (objet) : permet de récupérer l'identificateur OID de cet objet.
- **DEREF** (OID) : opération inverse de REF, elle récupère la valeur d'objet à partir de son identificateur OID.

9. Manipulations avancées de données objet-relationnelles

9.1. Utilisation des méthodes

Un type OBJECT dispose d'un sous-programme par défaut, permettant l'initialisation et l'instanciation d'objet appelé constructeur qui ayant le même nom que le nom du type d'objet. Le constructeur est composé de deux parties : la signature décrit le type du constructeur, son nom et ses paramètres avec le type SELF comme résultat. Le corps (ou body) du constructeur représente le code de ce constructeur. :

```
CREATE [OR REPLACE] TYPE BODY <nom type ADT>AS
Signature du méthode 1 IS
BEGIN
<code>
End Nom_méthode1;
Signature du méthode2 IS
BEGIN
<code>
End Nom_méthode2;
……
End;
```

La définition des méthodes se fait de la même façon que les constructeurs.

```
MEMEBER|STATIC FUNCTION nom_fonction [(paramètres)]
RETURN type_résultat
```

La spécification MEMBER permet de faire appel à un objet instancié en utilisant le paramètre SELF, tandis que STATIC définit l'appel à un type d'objet sans besoins d'instancier un objet appelant. De plus, l'utilisation de SELF dans le corps d'une méthode STATIC est interdite puisqu'il n'y'a pas d'objet appelant.

Remarque :

⚠ Lors de définition des paramètres des méthodes, on utilise le mode IN pour les fonctions et IN OUT pour les procédures.

Exemple :

```
SQL> CREATE TYPE Voiture_T AS OBJECT(Matricule INTEGER,
  2  Marque VARCHAR2(40), Modele VARCHAR2(40),
  3  Carburant VARCHAR2(20),
  4  MEMBER FUNCTION Get_Modele(Matricule IN INTEGER) RETURN VARCHAR2);
  5  /

Type created.
```

Le corps de la fonction Get_Modele est défini comme suit :

```
SQL> CREATE TYPE BODY Voiture_T AS
  2  MEMBER FUNCTION Get_Modele(Matricule IN INTEGER) RETURN VARCHAR2 IS
  3  T VARCHAR2(40);
  4  BEGIN
  5  SELECT V.Modele INTO T
  6  FROM Voiture V
  7  WHERE V.Matricule=Matricule;
  8  RETURN T;
  9  END Get_Modele;
 10  END;
 11  /

Type body created.
```

Exemple : afficher la marque et le modèle de voiture du matricule 1234567 en invoquant la fonction Get_Modele pour récupérer le modèle de voiture à partir de son matricule.

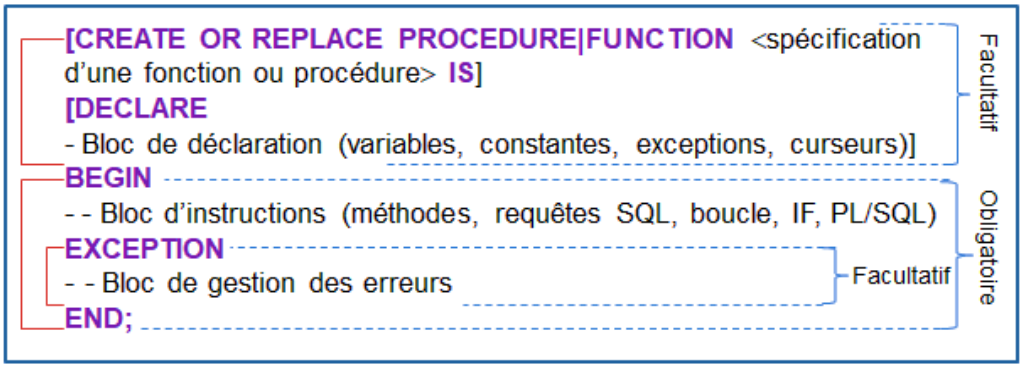
```
SQL> SELECT V.Marque, V.Get_Modele(1234567)
  2  FROM Voiture V;

MARQUE
----------------------------------------
V.GET_MODELE(1234567)
----------------------------------------
Mercedes
Merce Classe C Coupé
```

Pour supprimer une méthode, on utilise dans la commande ALTER TYPE et la clause DROP suivie par la signature d'une méthode. Comme on peut supprimer le corps d'un type objet par la commande suivante :

```
DROP TYPE BODY nom_Type objet;
```

9.2. Programmation PL/SQL

Le langage PL/SQL (Procedural Language/SQL) permet de manipuler une base de données en utilisant des expressions de programmation procédurale telles que les boucles, les conditions, la récursivité, etc. La syntaxe générale d'un programme PL/SQL est la suivante :



Exemple : un programme PL/SQL permet d'afficher les éléments d'un VARRAY nommée "Numeros". Ces éléments sont des numéros de téléphones de type NUMBER.

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
  2  T Numeros;
  3  BEGIN
  4  T:=Numeros(053211111, 06559434, 073333399, 0374433333);
  5  DBMS_OUTPUT.PUT_LINE('Mes Numéros de téléphones sont: ');
  6  FOR i IN T.First..T.LAST LOOP
  7  DBMS_OUTPUT.PUT_LINE('(' || to_char(i)||') ' || to_char(T(i)));
  8  END LOOP;
  9  END;
 10  /
Mes Numéros de téléphones sont:
(1) 53211111
(2) 6559434
(3) 73333399
(4) 374433333

PL/SQL procedure successfully completed.
```

9.3. Les curseurs

Le SGBD ORACLE dispose la notion de curseur permettant de créer des zones de travail pour stocker et exécuter les requêtes SQL, puis parcourir leurs résultats ligne par ligne.

```
DECLARE CURSOR <nom de curseur> IS
Requête SELECT…FROM…WHERE
Declaration des variables
BEGIN
OPEN <nom de curseur> ;
LOOP
FETCH <nom de curseur> INTO Variables;
Traitement iteratif
EXIT WHEN <nom de curseur>%NOTFOUND
END LOOP;
CLOSE <nom de curseur> ;
END; /
```

Exemple :

```
SQL> DECLARE CURSOR curseur IS
  2  SELECT * FROM Personne;
  3  Tampon  curseur%ROWTYPE;
  4  BEGIN
  5  OPEN curseur;
  6  LOOP
  7  FETCH curseur INTO Tampon;
  8  DBMS_OUTPUT.PUT_LINE(Tampon.Preno);
  9  EXIT WHEN  curseur%NOTFOUND;
 10  END LOOP;
 11  CLOSE curseur;
 12  END;
 13  /
NADA
Mohamed
Naim
PL/SQL procedure successfully completed.
```

9.4. Les vues objet-relationnelles

La vue est une table virtuelle créée à partir d'une requête.

```
CREATE VIEW <nom de VOR> OF type_objet [WITH OBJECT OID
(att_clé primaire)]
[UNDER nom_vue_mère] AS
SELECT  *| att1, att2
[,CAST(MULTISET(<requête>) AS <type complexe>) AS Alias]
[, type_nuplet (att_np1, att_np2, …)]
[, REF(alias)]
[, MAKE_REF(nom_autre_vue, att_oid)]
FROM  nom_table alias [, nom_table2 alias2]
WHERE conditions
[WITH CHECK OPTION];
```

Une vue objet-relationnelle offre la possibilité de créer des colonnes de types complexes tels que le type NESTED TABLE, VARRAY et REF. Les opérateurs CAST et MULTISET permettent de construire une colonne complexe (NESTED TABLE et VARRAY) à partir des tables relationnelles ou NF^2 par la syntaxe suivante :
CAST(MULTISET(SELECT…) AS <type>**)** **AS** <Alias>.

Exemple : soit la vue objet-relationnelle nommée Virtual, créée à partir d'une table relationnelle nommée Personne. Nous voulons créer une colonne complexe (NESTED TABLE) formée des matricules d'une deuxième table nommée Voiture pour récupérer pour chaque personne la liste des matricules de ses voitures. La création de vue est comme suit :

```
SQL> CREATE VIEW Virtual AS
  2  SELECT P.NUM, CAST(MULTISET(SELECT V.Mat FROM Voiture V
  3  WHERE V.NUMP=P.NUM) AS Ens_Mat) AS Matricules
  4  FROM Personne P;

View created.
```

9.5. Les triggers

Un trigger ou déclencheur *est une procédure stockée dans la base de données qui est déclenchée automatiquement par des événements spécifiés par le programmeur et ne s'exécutant que lorsqu'une condition est satisfaite.*

```
CREATE [OR REPLACE] TRIGGER nom_trigger BEFORE | AFTER
INSERT| DELETE| UPDATE [OF attributs] ON nom_table
[REFERENCING [OLD [AS] Nom_old] [NEW [AS] Nom_new]]
[FOR EACH ROW] [WHEN (condition)]]
bloc_pl/sql (code, requêtes SQL, affichage, etc.) ;
```

Remarque :

Un seul trigger par évènement sur une table de la base de données.

On peut manipuler simultanément l'ancienne et la nouvelle valeur d'un attribut en utilisant la clause REFRENCING OLD nom_ancien NEW nom_nouveau.

Exemple: création de trigger permettant de calculer et afficher le nombre de lignes de la table "Personne" après l'insertion dans cette table.

```
SQL> CREATE OR REPLACE TRIGGER MonTrigger AFTER INSERT ON Personne
  2  DECLARE N NUMBER;
  3  BEGIN
  4  SELECT COUNT(*) INTO N FROM Personne;
  5  DBMS_OUTPUT.PUT_LINE('le nombre de lignes de la table Personne est: '|| TO_
CHAR(N));
  6  END;
  7  /

Trigger created.

SQL> INSERT INTO Personne VALUES(545432, 'BADRI', 'MARIA', Mestel(TEL(23494949))
,null, 40000);
le nombre de lignes de la table Personne est: 5

1 row created.
```

Figure 1.6. Exemple de création et de validation de trigger

Finalement, plusieurs SGBD objet-relationnels ont été utilisés les plus importants sont: Oracle, PostgreSQL, Informix et DB2. Une série d'exercices de TD et de TP en utilisant Oracle 11g va être présentée dans les pages suivantes. Les corrections de ces exercices seront décri à la fin de ce polycopié.

Série TD N°1

(Bases de données Objet-Relationnelles)

Exercice 01.

Un centre de recherche veut stocker dans une base de données OR tous les documents qu'il produit. Un document est caractérisé par un numéro, un titre, un type (rapport, article ou livre), la date de création et le corps du document. Le corps est composé d'une suite de paragraphes. Chaque paragraphe a un numéro, un titre et un contenu. Un document a été rédigé par au moins un auteur. Un auteur est représenté par un identificateur, un nom, une affiliation et l'adresse électronique.

Questions :

Définir le schéma objet relationnel (création des ADT et les tables).

Exercice 02. *Requêtes SQL3*

En se basant sur la base de données objet-relationnelle de l'exercice 1. Ecrire les requêtes SQL3 suivantes :

1. Ajouter l'auteur Jiang Wang du département d'informatique et qui s'identifie par le numéro 617. Cet auteur dispose d'une adresse e-mails suivante : jiang.wa@yahoo.fr.
2. Le document 5491 est un livre intitulé « Les bases de données ». Ce document a été rédigé par l'auteur numéro 617 en 23/07/2010.

Exercice 03. *Programmation PL/SQL, curseur*

1. Ajouter une méthode qui donne le nombre de documents.
2. Ecrire un programme PL/SQL qui affiche le titre du livre de référence 5491.
3. Créer un curseur qui permet d'afficher le nom et l'email des auteurs.

Exercice 04. *Les vues objets et les triggers*

1. Créer la vue objet relationnelle qui donne les caractéristiques du document dont l'attribut corps est une table imbriquée.
2. Afficher le titre de document après l'attribution de nouveau titre.

Corrigé page 75.

TP 01.
(Création d'une base de données ORACLE)

Objectifs

L'objectif principal du TP est d'apprendre comment créer manuellement une base de données oracle. Une configuration du système d'exploitation doit être effectuée avec une préparation importante des paramètres de la BD.

1. Caractéristiques générales de la base de données :

La base de données objet-relationnelle que nous voulons créer permet de modéliser l'appartenance d'étudiant dans son université. Un étudiant ayant un numéro, un nom, un prénom et un niveau d'étude. Une université est identifiée par le nom et la ville. Ainsi, elle est caractérisée par l'année de création.

Cette BD s'appelle BDEU (ce nom est stocké dans DB_NAME), il ne doit pas dépasser 8 caractères. Il est préférable de choisir un nom d'instance et un nom de base de données identiques.

Remarque : L'instance par défaut du serveur est définie par la variable d'environnement ORACLE_SID.

2. Configuration du système d'exploitation :

Une BD ORACLE comporte deux principaux répertoires : un répertoire ADMIN pour les fichiers d'administration et un répertoire ORADATA contient les fichiers de base de données (*.dbf). Pour faciliter la création de BDEU avec ses répertoires, on définit dans un premier temps les variables d'environnement : ORACLE_HOME et ORACLE _BASE.

- Consulter ces deux variables sous Windows.
- Définir ces deux variables par la commande SET de l'invite de commande (CMD). Ces variables prennent les chemins : oraclexe…Server et oraclexe…oracle respectivement.

La deuxième étape sert à créer les répertoires de BD. Au niveau d'ORACLE _BASE, on crée deux répertoires :
ORADATA/BDEU et ADMIN/BDEU/. Ce dernier contient les répertoires suivants : ADUMP (pour les fichiers d'audit), BDUMP, DPDUMP (pour les fichiers export), UPDUMP et le répertoire PFILE qui contient les fichiers de paramètres.

Création du service windows pour le fonctionnement de l'instance :
Dans l'invite de commande : créer l'instance par la commande : ORADIM -NEW -SID BDEU.
Vérifier l'état du service oracle via les outils windows.

3. Création de la Base BDEU :

Préparation de fichiers oracle :

1. Préparer le fichier d'initialisation initBDEU.ORA dans le répertoire PFILE.
2. Pour se connecter à l'instance inactive (BDEU), utiliser le privilège SYSDBA par la commande / as SYSDBA.
3. Démarrer l'instance par :

   startup nomount pfile='%ORACLE_BASE%/ADMIN/BDEU/pfile/initBDEU.ora' ;
4. Créer la base de données BDEU : dans l'invite de commande SQL>
5. Installer le dictionnaire de données et les packages par la commande : @%ORACLE_HOME%/rdbms/admin/catalog.sql
6. Installer les objets du schéma : @%ORACLE_HOME%/rdbms/admin/catproc.sql

4. Travail à faire :

La base de données BDEU est prête pour accueillir des objets de schéma.

1. Dans SQL*PLUS, assurer la connexion de la base BDEU.
2. A partir de la base BDEU, créer le schéma d'objet modélisant la relation entre étudiant et l'université.
3. Remplir les tables obtenues et effectuer les requêtes de recherche, de suppression et de mise à jour.

Remarque :

Les autres manipulations de données vues en cours (PL/SQL, trigger, séquences, view, cursor, etc.) vont être étudiés dans le projet final de TP de ce module Bases de données avancées.

Corrigé page 77.

CHAPITRE 02

LES MODÈLES SÉMANTIQUES

1. Introduction

Le modèle Objet Relationnel représente le modèle de bases de données le plus répandu. Néanmoins, il comporte des critiques importants notamment la difficulté d'intégrer la sémantique aux données. Plusieurs modèles sémantiques ont été proposés pour exprimer le sens de données, le plus important est les ontologies du web sémantique.

Dans ce chapitre nous évoquerons brièvement les différents modèles sémantiques et nous focalisons sur l'utilisation du modèle sémantique le plus populaire appelé les ontologies afin d'exprimer la sémantique aux bases de données. Ces bases de données sont dites Bases de données à base ontologique (BDBO). Par la suite, nous définirons les approches de conception de BDBO. Dans le reste de ce chapitre, nous présenterons les trois langages de représentation des ontologies (RDF, RDFS et OWL) ainsi que le langage d'interrogation d'ontologie SPARQL. Une série d'exercices sera présentée à la fin du chapitre avec une série de travaux pratiques en utilisant l'éditeur Protégé.

2. Lacunes du modèle conceptuel

La définition du modèle relationnel (ou relationnel étendu) est basée sur des règles de passage du modèle conceptuel vers le modèle logique [5]. Une grande partie de la sémantique portée par le modèle conceptuel est disparu dans le modèle logique (cardinalités, la spécialisation, le nom des associations, etc.). Le modèle relationnel (ou logique) résultant est alors devenu incompréhensible surtout pour des grands modèles conceptuels.

Le modèle conceptuel étant une représentation de connaissance qui dépend à la fois les besoins applicatifs pour lesquels il a été défini et les concepteurs qui ne produisent pas le même modèle conceptuel adressé au même domaine. Les bases de données qui résulteront de différents modèles conceptuels seront alors hétérogènes et s'engendrerons des conflits sémantiques lors de toute tentative d'intégration des nouvelles données. Les principaux conflits sont [12, 13] : les conflits de nommage (par exemple une table voiture et une table véhicule), les conflits de structures (par exemple la table voiture peut être définie par des

propriétés de base et d'autre décrit par des propriétés détaillées), les conflits de granularité des attributs (par exemple l'attribut adresse peut se définir par un type complexe ou une chaîne de caractères), les conflits de type (par exemple l'attribut mois peut être défini par une chaîne de caractères ou un entier entre 1 et 12), etc.

Pour pallier ces lacunes du modèle conceptuel, une solution consiste à attribuer aux bases de données décrivant le même domaine, une description sémantique par l'utilisation d'un modèle sémantique.

3. Les modèles sémantiques pour la description des connaissances

Pour tenir compte de la sémantique de données, des modèles sémantiques ont été proposés, les plus connus sont [14]: les réseaux sémantiques, les graphes conceptuels, les thésaurus, les topic maps et les ontologies.

3.1. Les réseaux sémantiques

Les réseaux sémantiques représentent le premier modèle sémantique qui montre comment l'information pourrait être représentée en mémoire et comment on pourrait accéder à ces informations [15]. Un réseau sémantique est un graphe orienté et étiqueté, composé d'un ensemble de nœuds typés, dénotant des concepts du domaine modélisé, et d'arcs orientés étiquetés représentant les relations sémantiques entre les concepts [15].

3.2. Les graphes conceptuels

Les graphes conceptuels ont été introduits pour pouvoir exprimer la langue naturelle avec le formalisme de la logique par des nœuds qui peuvent être des concepts ou individus, des relations conceptuelles entre nœuds. Des arcs entrants dans les relations et d'autres sortants de nœuds sont représentés par des flèches [16].

3.3. Les thésaurus

Un thésaurus est une ressource sémantique qui garantit que la même terminologie est utilisée d'une manière cohérente [17].

3.4. Les topic maps

Un topic map ou carte topique permet de regrouper les concepts concernant un sujet donné « topic » [18]. Les relations entre les sujets sont représentées par des associations ainsi les occurrences représentent les relations entre les sujets et les ressources informationnelles qui s'y rapportent.

3.5. Les ontologies

Les ontologies représentent une technique primordiale du web sémantique servant un vocabulaire standard pour le partage et la réutilisabilité des connaissances [14].

En philosophie, l'ontologie (ontos=être et logos= études) désigne l'étude de ce qui existe en général [19]. Ce terme est utilisé dans le domaine de l'ingénierie de connaissances pour exprimer et structurer les connaissances du domaine par un ensemble de concepts [19]. La définition la plus complète de l'ontologie est celle de Studer qui combine la définition de Gruber avec celle de Borst. Selon Studer, l'ontologie se définit comme «*une spécification formelle et explicite d'une conceptualisation partagée*» [20].

La spécification formelle et explicite: signifie que la conceptualisation est représentée par un langage formel interprétable par une machine et elle définit d'une manière explicite [20].

La conceptualisation: représente les connaissances consensuelles d'une partie du monde réel par un modèle abstrait de données [20].

Le partage: les connaissances consensuelles doivent être partagées par un groupe d'individus [20].

Exemple: nous présentons une petite partie de l'ontologie ONTARIS (Ontology of Alimentation Risks) [1]. Les concepts Fruit et Vegetable héritent le concept Food. Apple est un type de fruit tandis que tomato est un type de vegetable.

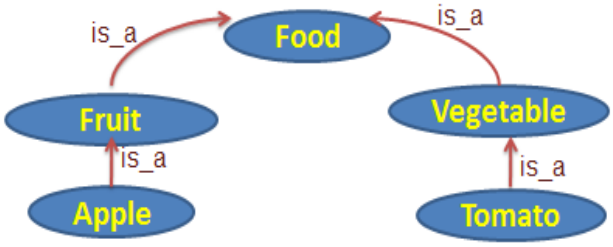


Figure 2.1. Exemple d'une partie d'ontologie ONTARIS [1]

Une ontologie est donc une hiérarchie des concepts relient entre eux par des relations prédéfinies telle que le lien de généralisation/spécialisation *is-a*, et d'autres relations particulières à un domaine donné [21, 22]. Chaque concept se définit par un nom et un ensemble de propriétés (ou slots) qui décrivent leur caractéristiques intéressantes [21, 22]. Ces propriétés sont dotées des valeurs qui peuvent être restreindre par des restrictions appelées facettes. Chaque concept est décrit par un ensemble d'individus (ou instances) représente les connaissances.

4. Approches de conception de Base de données à base ontologique

Une base de données à base ontologique (BDBO) est une source de données composée de deux structures différentes : une ou plusieurs ontologies décrivant la description sémantique de données et une base de données contenant les instances ou individus des classes de ses ontologies [23, 24].

Une BDBO est représentée par une table relationnelle contenant le schéma d'ontologie et une ou plusieurs tables contenant les données tout en assurant le lien entre une donnée et sa représentation sémantique à travers l'URI (Uniform Resource Identifier) de chaque élément d'ontologies [2]. La figure suivante illustre la structure d'une BDBO.

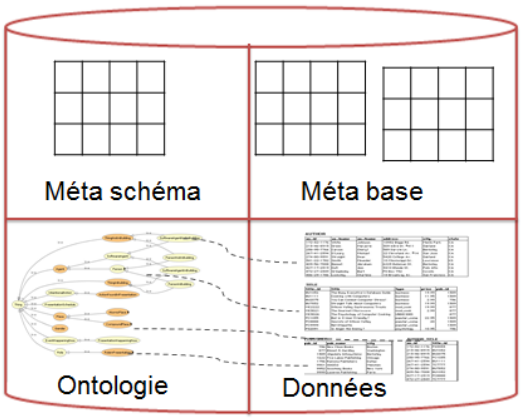


Figure 2.2. Structure d'une BDBO [2]

4.1. Approche à posteriori

Cette approche commence par la définition d'un modèle conceptuel initial d'un domaine donné et on essaye de mettre en correspondance entre ce modèle et des ontologies du domaine d'application précédemment construites [25]. Dans cette étape, le concepteur peut mettre à jour le modèle conceptuel initial par ajout des composants manquants ou suppression des composants superflus. Le concepteur doit vérifier la consistance entre le modèle conceptuel amélioré et les ontologies existantes [25].

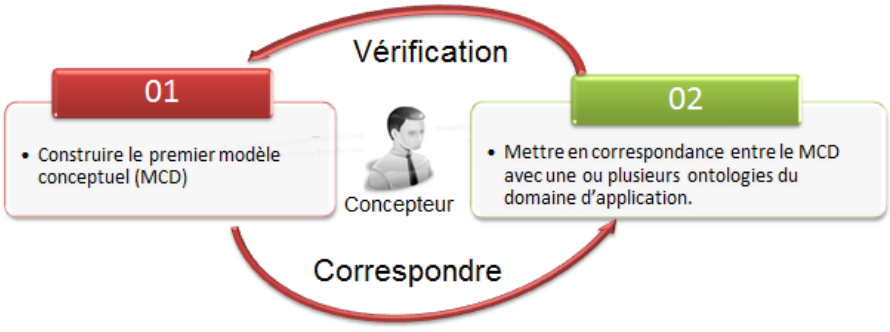


Figure 2.3. Approche à Posteriori

4.2. Approche à priori

A l'inverse de l'approche à postériori, l'approche à priori permet de construire le modèle conceptuel de données à partir d'une ou plusieurs ontologies. Plusieurs concepteurs peuvent intervenir pour définir une ontologie locale à partir d'un ensemble d'ontologies [26]. Cette ontologie locale sert ensuite à générer le modèle conceptuel de la future base de données [26].

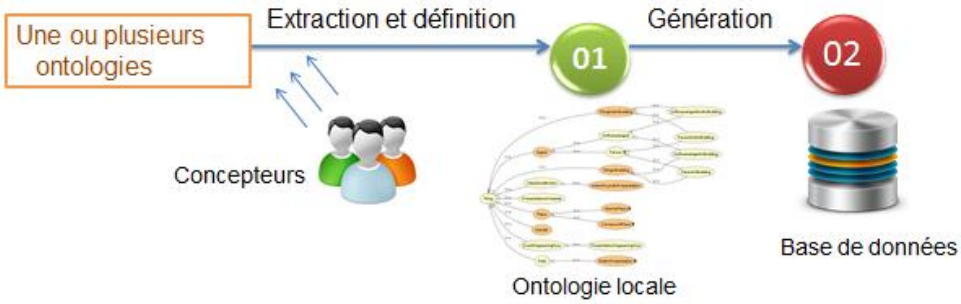


Figure 2.4. Approche à Priori

5. Langages d'ontologie

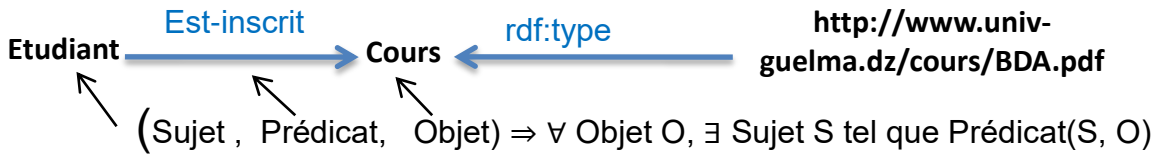
Plusieurs langages de représentation des ontologies ont été proposés afin d'en faire des ontologies formelles facilement exploitables et partageables par des utilisateurs.

5.1. RDF

RDF (Resource Description Framework) est un langage de représentation d'ontologie qui a été créé par le W3C[1] pour décrire des ressources Web (document, site web, partie de page web, page web, etc.) à l'aide d'une structure à base d'XML [27]. Ces ressources sont représentées sous forme de triplet sujet-prédicat-objet, sachant que le sujet décrit la ressource identifiée par son URI, le prédicat est une propriété ou relation entre deux sujets et l'objet est une donnée (littéral) ou une autre ressource [27]. Un Sujet peut être objet dans un autre triplet.

Un ensemble de triplets forme un graphe RDF où les nœuds peuvent être des sujets ou des objets et les arcs sont des prédicats [27].

Exemple: le graphe RDF modélisant la relation entre étudiant et le cours BDA est le suivant :



[1] W3C (World Wide Web Consortium) une communauté internationale de définition des standards pour les technologies liées aux web.

Le prédicat Est_inscrit décrit une relation sémantique entre le sujet "Etudiant" et l'objet Cours, tandis que le prédicat rdf: type est un prédicat prédéfini par RDF appelé aussi constructeur. Le rdf: type présente l'appartenance d'une instance à une classe ou même la définition de relation d'héritage entre classes.

La représentation RDF/XML de ce graphe RDF est représentée dans la figure suivante.

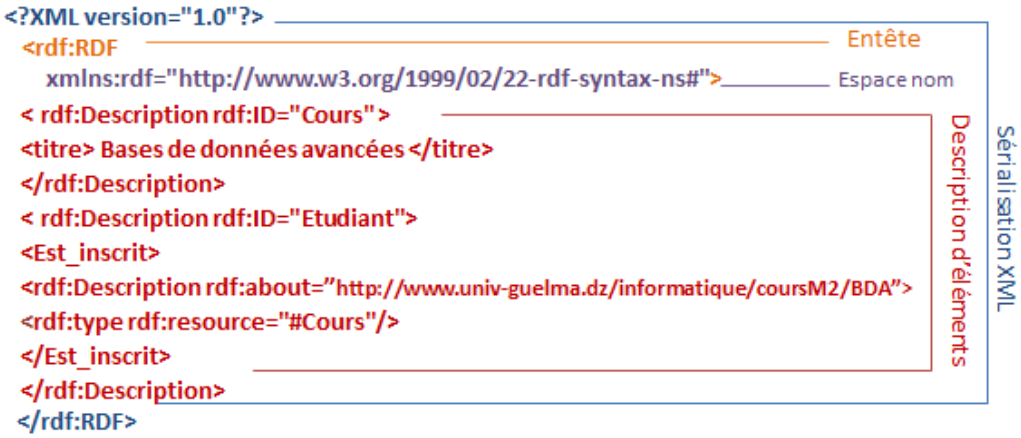


Figure 2.5. Exemple de document RDF/XML

Le document RDF/XML est une séquence d'éléments de type rdf :description permettant de décrire les triplets d'ontologie sachant que [27]:

- Rdf:type décrit la relation entre une classe et son instance. Il peut avoir comme valeur rdf:Resource ou rdf:Property.
- Rdf:resource permet de référencer un objet décrit dans le document.
- rdf:Property décrit une propriété d'une classe.
- Rdf:about : permet de définir des propriétés d'une ressource existante.
- Rdf:ID : définit une nouvelle ressource locale avec ses propriétés.
- Des balises définit par l'utilisateur par exemple, <Titre>BDA</Titre>.
- Rdf:datatype : décrit le domaine des valeurs en XSD, par exemple, le type chaîne de caractères est exprimé par rdf :datatype='&XSD :String', '&xsd:Positiveinteger ' pour les entiers positifs, '&xsd:Boolean' le type booléen, '&xsd:NonNegativeinteger' pour les entiers non négatifs, etc.

5.2. RDFS

Le RDFS (RDF Schema) ou RDF Schéma est une extension du RDF qui utilise ses constructeurs et offre la possibilité de définir les classes par rdfs:Class, l'héritage entre classes par rdfs:subClassOf, les propriétés par rdfs:Property, etc. [28]. Il offre aussi d'autre

constructeurs tels que rdfs:comment pour documenter une ressource, et rdfs:label pour définir un nom à une ressource [28].

Exemple :

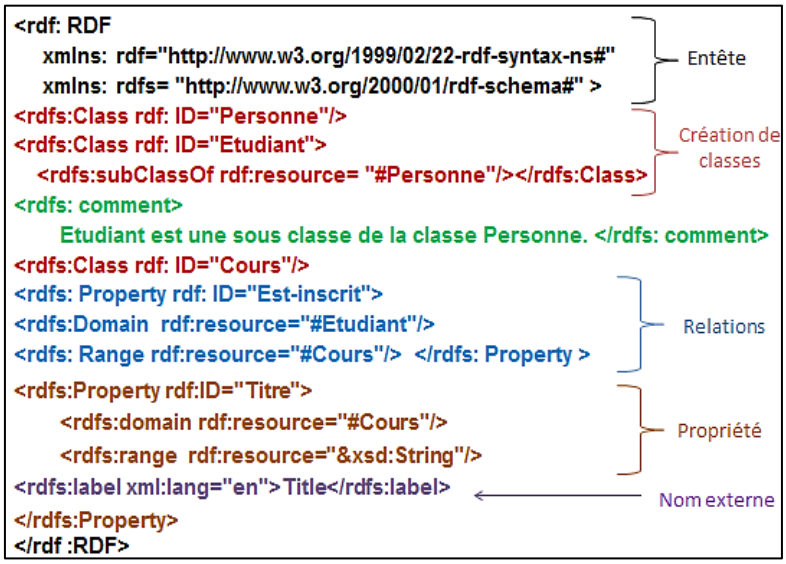


Figure 2.6. Exemple de document RDFS

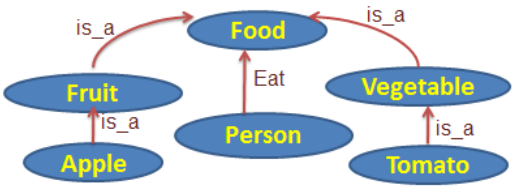
5.3. OWL

Le langage OWL (Web Ontology Language) a été recommandé par le groupe WebOnt de W3C dont le but d'étendre le RDFS par l'ajout des constructeurs permettant de décrire des ontologies complexes tels que la définition des cardinalités entre relations, les relations d'équivalence, de la symétrie, de l'inverse, etc. [29]. Le contenu du document OWL est défini par les triplets RDF avec une description utilisant des mots-clés préfixés OWL [29]. OWL se compose de trois sous-langages [29]: ***OWL-Lite*** (une version légère et moins complexe d'OWL), ***OWL-DL*** (un vocabulaire riche basé sur la logique de description DL) et ***OWL-Full*** (une version la plus complète et la plus expressive).

OWL demeure le langage le plus représentatif d'ontologies. Le graphe OWL contient :

- Une super classe de tout appelée owl:Thing
- Une classe vide (sans instances) définit par owl:Nothing
- ObjectProperty sont des relations entre les ressources uniquement.
- DatatypeProperty ont pour valeur un littéral possiblement typé.
- Relation d'inverse, par exemple, <rdfs:Property rdf:ID="hasChild"> <owl:inverseOf rdf:resource="#hasParent"/> </rdfs:Property>
- D'autres constructeurs tels que la définition des restrictions, l'union, l'intersection, etc.

Exemple: le document OWL du graphe RDF ci-dessous est le suivant.




```
< owl:Class rdf: ID="Person"/>
< owl:Class rdf:ID="Food"/>
< owl:Class rdf:ID= " Fruit">
<rdfs:subClassOf rdf:resource="# Food"/>
</owl:Class>
< owl:Class rdf:ID= "Vegetable">
<rdfs:subClassOf rdf:resource="# Food "/>
</owl:Class>
<!-- Propriétés d'objet -->
< owl:ObjectProperty rdf:ID= "Eat">
<rdfs:domain rdf:resource="# Person"/>
<rdfs:range rdf:resource="# Food"/>
</owl:ObjectProperty>
<!-- Propriétés de type de donnée -->
<owl:DataTypeProperty
  rdf: ID="nom">
<rdfs:domain rdf:resource="# Food"/>
<rdfs:domain rdf:resource="#Person"/>
<rdfs:range
rdf:resource="&xsd:String"/>
</owl: DataTypeProperty>
<!– les instances -->
< Fruit  rdf:ID=" Apple">
< Nom> Apple </Nom> </Fruit>
< Vegetable rdf:ID=" Tomato">
< Nom> Tomato </Nom>
 </ Vegetable>
```

Figure 2.7. Exemple de document OWL

Les ontologies sont comme les bases de données, possèdent un langage de manipulation appelé SPARQL permettant d'interroger le contenu d'ontologie.

5.4. SPARQL

SPARQL (SPARQL Protocol And RDF Query Language) est un langage proche de SQL, permet de manipuler les ontologies à travers des requêtes [30].

Une requête SPARQL est composée de deux parties principales [30]:

- *Partie déclaration :* est composée d'un ensemble des espaces de noms (name space) ou encore des adresses IRI (Internationalized Resource Identifiers) qui généralisent et internationalisent les adresses URI. Chaque espace de nom est préfixé par le mot-clé PREFIX pour simplifier son utilisation dans la partie interrogation.
- *Partie interrogation :* comprend un ensemble de commandes à effectuer sur les motifs de triplet (triple pattern).

Les requêtes SPARQL sont définies par la clause SELECT qui contient soit le symbole * pour sélectionner tous les composants du graphe ou bien un certain nombre de variables préfixées par le symbole '?' [30]. Elles peuvent contenir la clause WHERE délimitée par des accolades et contient un ensemble de motifs séparés par un point et un certain nombre des

conditions telles que: UNION entre deux conditions, FILTER pour filtrer les réponses selon un critère donné, ORDER BY pour trier selon une variable, LIMIT pour limiter le nombre de réponses, OFFSET indique à partir de quelle position dans la séquence on démarre, les fonctions d'agrégat (COUNT, SUM, MIN, MAX, etc.), et la négation par le symbole '!', etc.

Exemple : chercher dix étudiants âgés de plus de 20 ans des facultés MISM et STECH, triés par nom. Ces étudiants vont être sélectionnés à partir de 16ème réponse.

La requête SPARQL qui corresponde à cette question est illustrée dans la figure suivante.

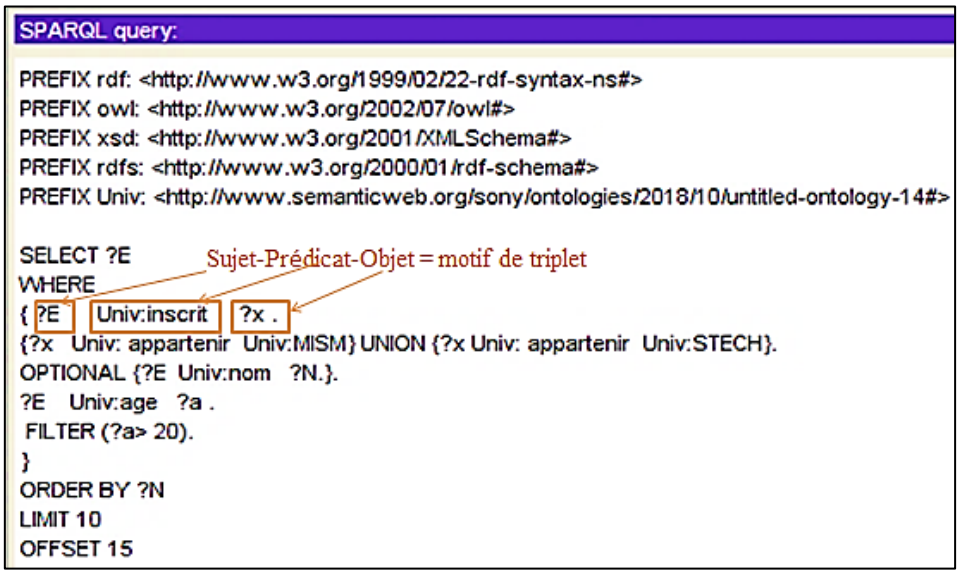


Figure 2.8. Exemple d'une requête SPARQL

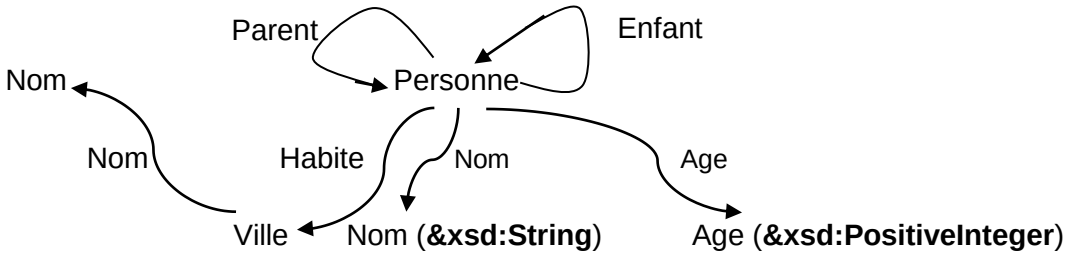
Il existe plusieurs outils de gestion d'ontologies le plus important est Protégé 2000. Pour implémenter et utiliser les BDBO, des outils doivent mettre en place dans des SGBD offrant le stockage des métadonnées des ontologies dans les bases de données. Parmi ces outils, on cite [2]: Jena, Sesame, RDFSuite, etc. Dans les séances de TP, nous nous focalisons sur l'utilisation de l'éditeur Protégé 4.3.0 pour la création et la manipulation des ontologies.

Série TD N°2

(Les modèles sémantiques)

Exercice 01. RDF et RDFS

Soit donné le graphe RDF suivant :



Questions :

Traduisez ce graphe

1. En code RDF-XML.
2. En code RDFS.

Exercice 02. OWL et SPARQL

En se basant sur le graphe RDF de l'exercice .1.

Questions :

1. Créez le document OWL correspondant.

2. Exprimez les requêtes suivantes en SPARQL :

2.1. Les personnes qui habitent à Constantine.

2.2. L'âge de Said.

Corrigé page 78

TP 02.
(Manipulation des ontologies via Protégé 4.3.0)

Objectifs

L'objectif de ce TP est de créer à l'aide du l'éditeur Protégé 4.3.0 un modèle sémantique de type ontologie. Ce TP permet d'apprendre comment construire un schéma ontologique et comment l'interroger par SPARQL.

1. Création de l'ontologie OntoTP :

En exploitant les menus de protégé, créer les classes d'OntoTP suivantes : Personne, Ville, Conducteur et Voiture.

Créer les propriétés suivantes :

- Dans la classe Personne : Nom, Age et genre (Masculin ou Féminin).
- Dans la classe Ville : NomVille.
- Dans la classe Conducteur : NumP (Numéro du permis de conduite).
- Dans la classe Voiture : Immatricule, Type (Type de voiture).

Créer les relations suivantes :

- Une personne habite dans une ville.
- Une personne est un parent d'une autre personne.
- Une personne est un enfant d'une autre personne. Cette relation étant l'inverse de la relation Parent.
- Un conducteur conduit une voiture.

2. Travail demandé :

1.1. Le Protégé permet de vérifier la cohérence et la consistance de l'ontologie créée via des raisonneurs tels que FaCT++ et HermiT. Vérifier que l'ontologie OntoTP ne contient pas des définitions contradictoires à l'aide de raisonneur HermiT.

1.2. Définir le name space OntoTP qui contient l'URI de l'ontologie OntoTP et traduire les questions suivantes en requêtes SPARQL :

a. Les parents avec leurs enfants.

b. Les personnes avec la ville où elles habitent.

Corrigé page 79.

CHAPITRE 03

LES BASES DE DONNÉES ORIENTÉES OBJET

1. Introduction

Dans le premier chapitre de ce polycopié de cours, nous avons dit qu'il existe deux solutions pour combler les limites du modèle relationnel ; la première a été présentée dans le premier chapitre appelée modèle relationnel étendu. La deuxième solution est purement orientée objet qui sera l'objet de ce chapitre.

Nous définirons dans un premier temps la notion de base de données orientée objet (BDOO) et les caractéristiques des SGBD orientés objet puis nous présenterons les éléments d'une BDOO. Nous présenterons la modélisation ODMG qui dispose deux langages de définition et de manipulations d'objets : ODL et OQL.

2. Définitions

Le modèle relationnel étendu bien qu'il conserve les caractéristiques du modèle relationnel, son inconvénient majeur réside dans le fait qu'il présent deux niveaux d'abstraction différents: le niveau relation et le niveau objet. Ces deux niveaux apportent des problèmes notamment la perte de compatibilité relationnelle et le manque de compatibilité avec les langages de programmation, etc.

2.1. Base de données orientée objet

Une base de données orientée objet (BDOO) est une base de données dont son contenu est un ensemble d'objets ou instances des classes plutôt que des n-uplets des relations [31]. Elle est caractérisée par des propriétés dérivées de l'approche orientée objet telles que:

1. La représentation des structures de données complexes.
2. L'encapsulation de données et de traitements.
3. L'héritage des classes.
4. Le polymorphisme de méthodes.
5. L'identificateur d'objet pour différencier entre objets mêmes s'ils ont la même valeur.

6. La compatibilité avec les langages de programmation orienté objet tels que Java, C++.

2.2.Système de gestion de base de données orienté objet

La gestion des BDOO est faite à travers un SGBD orienté objet (SGBDOO) qui est obtenu par la fusion de SGBD et de langage de programmation orienté objet (LPOO) [32]. Le SGBDOO offre un ensemble de fonctionnalités qui n'existent pas dans le LPOO [32, 33]:

- *La persistance des données :* un objet continue d'exister après que son créateur ait cessé d'exister ou que l'emplacement où l'objet a été créé change l'adresse.
- *Langage de manipulation déclaratif :* un langage de manipulation de données LMD permet d'interroger les objets d'une façon déclarative.
- *Optimisation par le SGBD* : le SGBOO permet d'optimiser ses requêtes.
- *Intégrité des données* : les objets persistants respectent les contraintes d'intégrité.
- *Gestion de transactions :* le SGBDOO assure le partage, la gestion de concurrence et de transaction d'objets avec toute confidentialité, fiabilité et sécurité.

3. Les éléments d'une BDOO

Ce cours emploie une syntaxe tirée de celle d'ODMG[2] (cf. section 4). Nous nous basons sur les références [32, 33] pour présenter ces éléments.

3.1. Objet

L'objet est la structure principale qui définit les données d'une BDOO. Il est composé d'un identificateur unique OID (Object Identifier), un ensemble d'attributs décrivant l'état d'objet et une collection de méthodes forme le comportement d'objet.

Objet = OID + Etat (attributs) + Comportement (méthodes).

Un objet atomique est défini par des types littéraux atomiques. Un littéral est une donnée qui n'a pas d'identificateur d'objet. Il ne peut être stocké directement de manière persistante. Les littéraux atomiques sont: Long, Short, Float, Double, Char, String, Boolean, Enum (énumération).

Un objet collection représente des attributs multivalués tels que ; Set<t>, Bag<t>, List<t>, Array<t>, Dictionary<t, k>. Avec : t, k sont des types d'objets ou des types de littéraux.

Un objet structuré est un objet qui peut utiliser des types littéraux structurés tels que Date, Interval, Time ou une série d'attributs définie par le constructeur "STRUCT" comme suit :

[2] ODMG (Object Database Management Group) groupe de normalisation des systèmes de gestion de base de données orientés objet.

```
STRUCT {Type_attribut1 Nom_attribute1, Type_attribut2 Nom_attribute2, …}
```

On peut créer un type d'objet structuré par la commande TYPEDEF suivante :

```
TYPEDEF Nom_type-structuré STRUCT
{Type_attribut1 Nom_attribute1, Type_attribut2 Nom_attribute2, …} ;
```

Exemple :

TYPEDEF T_Affiliation STRUCT {STRING Depart, STRING Univ, STRING Lab, STRING Pays, SHORT BP};

3.2. Classe d'objets

Un ensemble d'objets ayant les mêmes états et comportements forme une classe d'objets. La classe permet de créer des objets appelés souvent instances. La création de classe d'objets se fait par la commande suivante :

```
CLASS Nom_class [: super_class] (EXTENT Nom_collection [KEY Nom_attributs])
{ATTRIBUTE Type_attribut1 Nom_attribute1 ; ATTRIBUTE Type_attribut2 Nom_attribute2 ; … ; Relationships entre classes ; Signature des Méthodes et constructeurs;}
```

Exemple : on utilisant le type structuré T_Affiliation présenté dans la sous-section précédente pour la création de classe Chercheur suivante :

CLASS Chercheur
(EXTENT LesChercheurs **KEY** ORCID**)**
{**ATTRIBUTE LONG** ORCID;
ATTRIBUTE STRING Nom ;
ATTRIBUTE STRING Prenom;
ATTRIBUTE T_Affiliation Affiliation;

ATTRIBUTE ENUM Grade {'Attaché', 'Chargé', 'Maitre', 'Directeur'};
ATTRIBUTE STRING Email;
}

Remarque :

Une interface est une classe particulière qui ne peut être instanciée. Elle est composée d'un ensemble d'attributs et une liste de méthodes sans implémentation.

3.3. L'héritage

L'héritage des classes permet de construire une hiérarchie des classes relient entre elles par une relation de généralisation/spécialisation ou le lien IS_A ou encore relation de subsomption.

Remarque :

La majorité des SGBD OO n'autorise que l'héritage simple (une seule classe mère).

Exemple : la classe "Chercheur-permanent" hérite la classe "Chercheur" qui a été présentée dans la sous-section 3.2.

Class Chercher_permanent **:** Chercheur

EXTENT LesChercheursPermanents

{**ATTRIBUTE DOUBLE** Salaire**;**}

3.4. Les constructeurs et les méthodes

Chaque classe possède au moins une méthode appelée Constructeur permettant de créer des instances des classes.

```
<Type de résultat>/VOID <nom de méthode> (type_param1 param1, …)
```

Une méthode peut être une procédure avec le mot clé VOID qui indique que la méthode ne rend pas de résultat ou une fonction en spécifiant le type de résultat.

Exemple : la méthode NouveauChercheur de la classe "Chercheur" permet d'ajouter un nouveau chercheur. VOID NouveauChercheur(Chercheur C) ;

Le corps d'une méthode peut contenir des instructions de LPOO, des requêtes, invocation de méthodes sur d'autres objets.

La création du corps d'une méthode se fait par la syntaxe suivante :

```
METHOD BODY <Type de résultat>/VOID <nom de méthode>
(type_param1 param1, …) IN CLASS <Nom_class> {
Code LPOO, requêtes, appels méthodes ;}
```

Exemple : le corps de la méthode NouveauChercheur de l'exemple précédent est la suivante :

METHOD BODY VOID NouveauChercheur(Chercheur C) IN CLASS Chercheur

{Chercheur Nchercheur;

Nchercheur = Chercheur (Nom: C. Nom, Prenom: C.Prenom, Affiliation: C.Affiliation, Grade: C.Grade, Email:C.Email, ORCID: C. ORCID);}

3.5. Le polymorphisme

Le polymorphisme permet de surcharger un nom de méthode dans le sens où plusieurs méthodes ayant le même nom mais de corps et des types d'arguments différents [34].

3.6. L'association "Relationship"

L'association Relationship est une relation binaire, nommée, sans attributs, avec de cardinalité qui peut être (1:1), (1:N), (N:M) [35]. La définition de relationship se fait comme suit :

```
RELATIONSHIP [SET|LIST] nom-classe nom-de-la-relation INVERSE
nom-classe :: nom de la relation inversée;
```

Exemple :

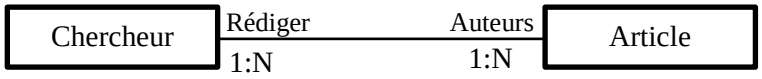


Figure 3.1. Exemple d'une Relationship entre classes

La définition de relationship entre ces deux classes est exprimée comme suit :

Class Chercheur (Extent LesChercheurs) { ... , RELATIONSHIP SET Article Rédiger INVERSE Article::Auteurs; }	**Class** Articles (Extent LesArticles) { ..., RELATIONSHIP SET Chercheur Auteurs INVERSE Chercheur::Rédiger ; }

3.7. Le lien de composition

Le lien de composition est un lien orienté de la classe composée vers la classe composante avec des cardinalités. La destruction d'une classe composée implique également la destruction de la classe composante.

Exemple : la classe Article est composée de plusieurs sections.

Figure 3.2. Exemple d'un lien de composition

La représentation textuelle de ce diagramme est la suivante :

Class Articles (Extent LesArticles) {ATTRIBUTE SHORT ID ; ATTRIBUTE STRING Titre ; **ATTRIBUTE Section Sections**;}	**Class** Section (Extent LesSections) {ATTRIBUTE SHORT N°; ATTRIBUTE STRING Titre;}

4. Modélisation des BDOO par ODMG

L'OMG[3] propose un standard permettant la portabilité des schémas de base de données et des programmes développés sur des SGBDOO appelé ODMG (Object Database Management Group) [36]. L'ODMG dispose un langage graphique pour modéliser les bases de données orientées objet dont les classes sont représentées par un rectangle associé avec le nom de collection et la clé de classe. Les attributs et les méthodes sont attachés dans les classes par des tirés.

Exemple : les classes Chercheur-permanent, Chercheur-Doctorant héritent la classe chercheur. La classe 'Article' est liée avec la classe 'Chercheur'. La classe 'Article' est composée de plusieurs objets de la classe 'Section'.

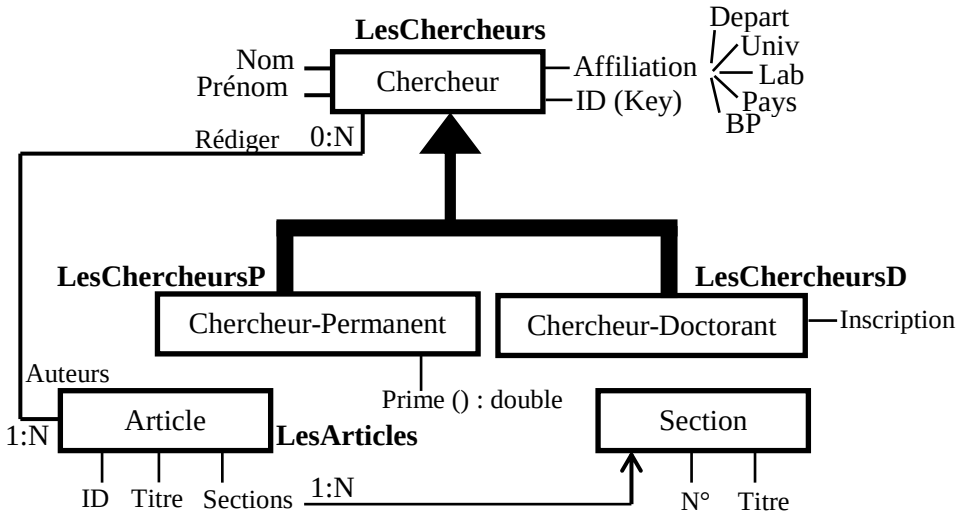


Figure 3.3. Exemple de diagramme ODMG

5. Langages de définition et de manipulation des BDOO

La norme ODMG dispose des langages de définition et de manipulation de données les plus importants sont: ODL et OQL. Nous nous basons sur les références [36-38].

5.1. ODL

Le langage ODL (Object Definition Language) permet de créer un schéma ODL décrivant une BDOO par la création des classes, des interfaces, des objets et implémentations des méthodes. Il permet aussi de compiler le schéma d'objets et de générer les données de la BDOO.

[3] OMG : Object Management Group

Exemple : le schéma ODL du diagramme ODMG de la figure 3.5 est le suivant.

Class Chercheur **(Extend** LesArticles **Key** ID**) {**ATTRIBUTE SHORT ID;
ATTRIBUTE STRING Nom;
ATTRIBUTE STRING Prenom;
ATTRIBUTE STRUCT Affiliation_type {STRING Depart, STRING Univ, STRING Lab, STRING Pays, SHORT BP} Affiliation;
RELATIONSHIP SET Article Rédiger INVERSE Article::Auteurs;**}**
Class Chercheur-Permanent: Chercheur **(Extend** LesChercheursP**)**
{DOUBLE Prime(); **}**
Class Chercheur-Doctorant: Chercheur **(Extend** LesChercheursD**)**
{Attribute DA TE Inscription; **}**
Class Article **(Extend** LesArticles **Key** ID**)**
{ATTRIBUTE SHORT ID;
ATTRIBUTE STRING Titre;
ATTRIBUTE SET SECTION Sections ;
RELATIONSHIP SET Chercheur Auteurs INVERSE Chercheur::Rédiger;**}**

5.2. OQL

L'OQL (Object Query Language) est un langage déclaratif de requêtes proche de SQL mais non compatible avec SQL [37]. L'OQL fournit des primitives de haut niveau qui manipulent les collections et les types structurés.

Exemple : on crée un objet persistant nommé de la classe Chercheur:

NAME SERIDI : Chercheur ; //déclaration d'une variable permanente nommée SERIDI

SERIDI=Chercheur (ID :12754324, Nom : 'Séridi', Prenom : 'Hamid', Affiliation : STRUCT (Depart: 'Informatique', Univ: '08 mai 45 Guelma', Lab : 'LabSTIC Guelma', Pays : 'Algérie', BP : 401), Rédiger : SET (Article(….), Article(….))) ;

Dans cet exemple, nous avons créé une variable permanente nommée "SERIDI" de type "Chercheur" en spécifiant ses valeurs par le constructeur de classe. Concernant les attributs de type structuré comme dans le cas de l'attribut "affiliation", on utilise le mot clé STRUCT suivi par des valeurs associées à ses attributs.

On peut créer une valeur structurée comme dans l'attribut Affiliation:

Affiliation_Séridi=STRUCT (Depart: 'Informatique', Univ: '08 mai 45 Guelma', Lab : 'LabSTIC Guelma', Pays : 'Algérie', BP : 401).

L'attribut "Rédiger" de la classe chercheur est une relation avec plusieurs objets de type "Article".

L'OQL permet d'accéder directement aux données de BDOO à partir d'un objet nommé ou collection d'objets, par exemple, SERIDI.Nom ; SERIDI.Prenom ;

On peut accéder aux données structurées par l'utilisation de la notation pointée, par exemple, SERIDI. Affiliation.Lab ;
L'OQL fournit un appel direct aux méthodes à travers l'objet nommé, par exemple SERIDI.Age() ;.

De plus, l'OQL supporte l'affichage de tous les objets d'une collection via son nom, par exemple, afficher tous les chercheurs par le nom de collection : LesChercheurs ;

Remarque :

La notation pointée est interdite pour accéder aux attributs d'une relation. L'accès se fait via des requêtes OQL.

Exemple :
SERIDI. Rédiger. Titre N'EST PAS CORRECT

La requête OQL interroge via la clause FROM les collections d'objets définies par le mot clé EXTENT plutôt que des classes d'objets et cela par l'utilisation du mot clé "IN".

Exemple : afficher le nom de tous les chercheurs
SELECT C.Nom
FROM C IN LesChercheurs ;
==>Littéral BAG <String>

Le SGBDOO affiche automatiquement le type de résultat de requête OQL.

Exemple 01: Afficher le nom et le prénom de tous les chercheurs.
SELECT C.Nom, C.Prenom
FROM C IN LesChercheurs;
==>Littéral BAG<STRUCT<Nom: STRING, Prenom:STRING>>

Exemple 02: afficher la prime de chercheur Chantal Reynaud sachant que Prime() est une fonction. L'invocation de cette méthode s'effectue par la notation pointée.
SELECT C.Prime()
FROM C IN LesChercheurs;
WHERE C.Nom='Reynaud' AND C.Prenom='Chantal';
==> <DOUBLE>

L'OQL offre un mécanisme de nommage de requêtes par la commande DEFINE suivante :

```
DEFINE <nom> AS requête;
```

Exemple : on définit une requête qui retourne l'ensemble des chercheurs algériens.

DEFINE Chercheur-Algérien AS
SELECT C
FROM C IN LesChercheurs
WHERE C. Affiliation.Pays='Algérie'.

Remarque :

On peut utiliser l'opérateur IN dans la clause WHERE comme dans le SQL.

Les quantificateurs logiques : on peut utiliser dans une requête OQL les quantificateurs logiques qui sont : le quantificateur existentiel EXISTS…IN… et le quantificateur universel FOR ALL…IN.... Ces opérateurs sont utilisés dans une clause conditionnelle telle que la clause WHERE.

Exemple 01: afficher les titres des articles s'il y'a au moins une section dénommée Résumé.

SELECT A.Titre
FROM A IN LesArticles
WHERE **EXISTS** S **IN** A.Section**:** S.Titre='Résumé';

==>Littéral SET <STRING>

Exemple 02 : afficher les chercheurs bénéficiaires d'une prime supérieure à 50000DA.

SELECT C
FROM C IN LesChercheurs
WHERE **FOR ALL** C **IN** LesChercheurs**:** C. Prime()>50000;

==>Littéral SET <Chercheur>

L'OQL supporte la fonction GROUP BY de SQL qui permet de subdiviser le résultat de requête en groupes appelés PARTITION selon un certain nombre de critères. Le type de résultat d'une requête avec GROUP BY est toujours SET STRUCT(….). Le mot clé PARTITION va être utilisé comme une collection d'objet et il peut être utilisé dans une clause FROM d'une sous-requête.

Exemple : Afficher le nom et le nombre de chercheurs par département.

SELECT department, NomC: (P.Nom FROM P IN **PARTITION)**, Nbre: (SELECT COUNT(P) FROM P IN **PARTITION**)
FROM C IN Chercheur
GROUP BY (department: C.Affiliation.Depart);

==>Littéral SET<STRUCT<departement: STRING, NomC: STRING: Nbre:SHORT>>

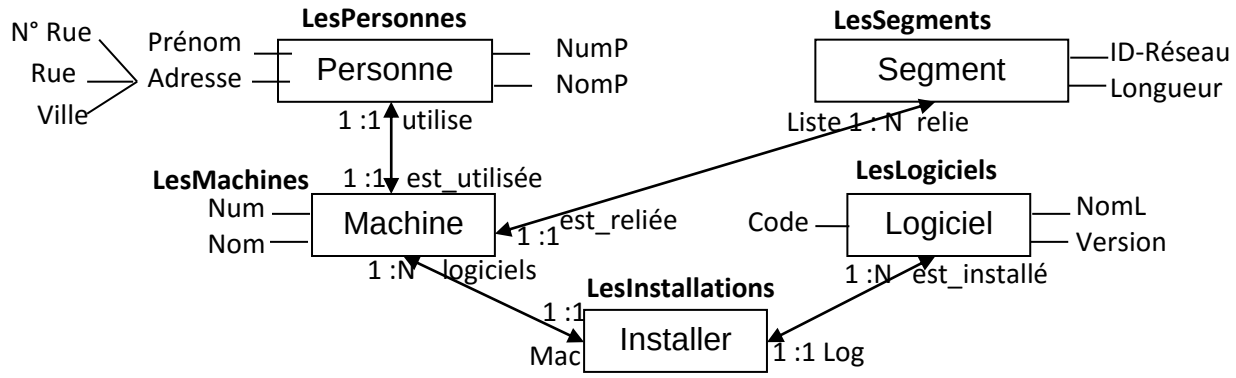
D'autres fonctions sont disponibles dans l'OQL. Dans ce cours, nous avons étudié les fonctions les plus importantes citées précédemment. L'utilisation de SGBDOO est limitée dans des applications du point de vue objet (CAO, SIG, etc.) et ils sont complètement absents du monde professionnel. De plus, la majorité des SGBDOO sont commerciaux tels que ObjectStore, Versant Object Technology, etc.

Série TD N°3

(Base de données orientée objet)

Exercice 01. Schéma ODMG, ODL

La modélisation orientée-objet d'une base de données de gestion d'un Cyber Café est donnée par le diagramme ODMG suivant :



Question :

Traduire ce diagramme ODMG en schéma ODL.

Exercice 02. Méthodes et OQL

En utilisant le schéma ODL de l'exercice 01 pour la manipulation d'objets via le langage OQL.

Questions :

1. Ecrire le corps des méthodes suivantes :

1.1. *NbrSeg* permet de connaitre le nombre de segments existants.

1.2. Initialiser la longueur de segment par 1.

2. Ecrire en OQL les requêtes suivantes :

2.1. Affecter la personne suivante à la machine nommée MacJeux du numéro 18: Omar Tahi du numéro 19, habitant à Guelma, rue 19 Juin, numéro 25.

2.2. Les noms des logiciels qui sont installés sur la machine 5.

Corrigé page 80.

CHAPITRE 04

LES BASES DE DONNÉES DÉDUCTIVES

1. Introduction

Le succès des modèles relationnels résident dans le fait qu'ils reposent sur deux théories mathématiques : la théorie des ensembles et la logique mathématique. Néanmoins, utiliser le modèle relationnel pour faire des raisonnements et exprimer des règles déductives demeure l'une de ses limites. De ce fait, une base de données déductive est apparue pour la modélisation des bases de données relationnelles par la logique du premier ordre.

Ce chapitre se présente dans un premier temps un bref rappel sur la logique du premier ordre puis il étudie les BD déductives en introduisant le langage standard appelé DATALOG. Finalement, nous présenterons des séries de TD et TP en utilisant le SGBD déductif DES, version 6.1.

2. Bref rappel sur la logique du premier ordre

La logique du premier ordre ou calcul de prédicats permet de représenter les connaissances par la spécification des relations entre objets et de déduire des nouvelles connaissances en appliquant des règles d'inférence [39]. Un prédicat de longueur n, est une fonction P à n arguments décrivant une relation particulière entre les objets qui peut être vraie ou fausse [40]. Les arguments sont des termes qui peuvent être des constantes généralement notées a, b, c, etc. ou des variables notées x, y, z, etc. [40]. Le prédicat P à n termes P(t1,t2,…tn) est appelé une formule atomique, par exemple, Père (x, y) signifie que x est le père de y [41]. Un littéral est une formule atomique ou la négation d'une formule atomique [41].

La logique du premier ordre utilise des connecteurs logiques entre prédicats qui sont [41]: la conjonction "$\wedge$", la disjonction "$\vee$" et l'implication "$\Rightarrow$". On peut aussi définir la négation d'un prédicat par le symbole "$\neg$".

De plus, une formule de la forme : $P1 \wedge P2 \wedge P3 \wedge \ldots \wedge Pn \Rightarrow Q1 \wedge Q2 \wedge Q3 \wedge \ldots \wedge Qm$ avec Pi et Qj sont des littéraux positifs (sans négation) est appelée clause [40]. Les Pi sont appelés les

antécédents (les littéraux situés avant l'implication) et les Qj sont les conséquents ou la tête de la clause.

Une clause ayant un seul littéral en tête est dite clause de Horn de la forme [42]: $P1 \wedge P2 \wedge P3 \wedge \ldots \wedge Pn \Rightarrow Q$

Exemple : soient x, y deux employés et s est un service. Si l'employé x est responsable à l'employé y et que ces deux employés travaillent dans le même service alors x est le chef service. Cette phrase est exprimée comme suit :

Responsable (x, y) $\wedge$ Service (s, x) $\wedge$ Service (s, y) $\Rightarrow$ Chef_Service(x).

On peut transformer l'implication d'une clause par la disjonction des négations de littéraux qui forment la tête de clause [42]. La clause de Horn devient donc: $P1 \wedge P2 \wedge P3 \wedge \ldots \wedge Pn \vee \neg Q$.

Une règle d'inférence aussi appelée règle déductive est une application qui à partir de n>=1 expressions on déduit une formule bien formé. Elle est de la forme [43]:

$\forall t\ (h(t) \leftarrow \exists y(b1\ (y) \wedge \cdots \wedge bn\ (y))$

Exemple : la clause précédente s'écrit sous forme d'une règle déductive comme suit :

$\forall x$ Chef_Service(x) $\leftarrow \exists y\ \exists s$ (Responsable (x, y) $\wedge$ Service (s, x) $\wedge$ Service (s, y)).

Nous terminons cette section par quelques propriétés entre prédicats [43]:

- P dépend directement de Q si Q est un antécédent et P est un conséquent.
- P est récursif si P dépend de P.
- P dépend strictement de Q si P dépend de Q et Q ne dépend pas de P.
- Une règle déductive est récursive si son antécédent contient un prédicat qui dépend du conséquent.
- Une règle déductive est stricte si les littéraux figurants dans l'antécédent sont tous positifs.
- Une règle déductive stricte est équivalente à une clause de Horn : $h(t) \vee \neg b1 \vee \cdots \vee \neg bn$.

3. Base de données déductive

Les bases de données déductives sont vues comme le résultat du couplage entre deux domaines ; les bases de données et l'intelligence artificielle [43]. L'idée est d'intégrer dans une base de données un ensemble de règles logiques (règles déductives) qui permettent d'en déduire de l'information [43]. Une BD déductive se compose de deux types de prédicats :

- *Les prédicats extensionnels :* ils correspondent aux relations du modèle relationnel comme la relation Personne est un prédicat extensionnel. Les extensions des prédicats sont souvent appelées les faits (Ex. Omar) qui sont des tuples de BD. L'ensemble de

prédicats extensionnels avec leurs extensions forment une base de données extensionnelle (BE).

- *Les prédicats intentionnels :* sont des règles déductives exprimées par une programmation logique sous forme d'une suite de clause de Horn en utilisant un langage logique comme Datalog. Ils correspondent aux vues dans le modèle relationnel. L'ensemble de prédicats intentionnels forme une base de données intentionnelle (BI).

Alors, on peut définir une base de données déductive à partir d'une base de données relationnelle comme suit : les prédicats extensionnels et leurs extensions sont le contenu de cette base de données relationnelle avec la création des vues relationnelles qui vont être vues comme des prédicats intentionnels sous forme des clauses de Horn permettant de déduire ou d'inférer des informations à partir de faits stockés dans une base de données.

L'ensemble de règles déductives forme un graphe de dépendances dont les nœuds sont des prédicats et les arcs entre prédicats représentent les règles déductives. S'il existe une règle récursive alors le graphe de dépendances dispose un cycle. L'exemple le plus connu est certainement la description des relations de parenté :

Ancêtre (x, y) ←Parent (x, y).

Ancêtre (x, y) ←Parent (x, z)∧ Ancêtre (z, y).

Le graphe de dépendance qui correspond ces règles d'inférences est :

4. SGBD déductif

A la fin des années 70, Gallaire, Minker et Nicolas ont fait des travaux visant à rendre les SGBD capables de : Définir des faits, Faire des raisonnements et Exprimer des règles déductives en logique du premier ordre. Il s'agit d'un SGBD déductif qui comporte les fonctionnalités d'un SGBD classique pour gérer les faits avec des opérateurs dérivés de la logique du premier ordre pour assurer la définition et la manipulation de connaissances [44]. Il possède trois langages [44]:

- Un langage de définition de base de données extensionnelle.
- Un langage de requêtes SQL.
- Un langage de règles pour définir la base de données intensionnelles.

Le SGBD déductif intègre donc un moteur d'inférence et un langage intégré de définition et manipulation de connaissances [45].

Les SGBD déductifs sont utiles dans des applications avancées nécessitant la gestion efficace de connaissance telles que la fouille de données (data mining) et l'extraction de connaissances dans les données (ECD). Ces SGBD déductifs sont souvent basés sur le modèle relationnel pour gérer les connaissances à titre d'exemple, le SGBD DES (Datalog Educational System) fondé sur l'utilisation de langage logique Datalog et le SQL [10].

5. Le langage DATALOG

Le langage DATALOG (Data base Logic) signifie la logique pour les données, est un langage déclaratif de requête et de règles pour les bases de données déductives.
DATALOG est un sous ensemble de langage de programmation logique PROLOG, dédié à la description et à la manipulation de connaissances. Le DATALOG est composé des symboles suivants [10, 46, 47]:

- Des variables en majuscule (obligatoire) dénotées X, Y, Z … ;
- Des constantes qui peuvent être les tuples de la BDR, des entiers, etc.;
- Des prédicats ;
- Les opérateurs de comparaison arithmétiques : <, ≤, >, ≥, =, ≠.
- Les opérateurs de comparaison lexicographiques : = = (l'égalité), \= = (la différence).
- Les connecteurs logiques "et" dénoté par une virgule "," et "ou" exprimé par deux clauses de Horn ou on utilise le symbole " ;" dans une seule règle. L'implication est désigné par le symbole ":-" qui s'interprète de la droite vers la gauche.
- La négation de prédicat par le mot clé "not".

5.1. Définition de données

La déclaration de prédicats extensionnels se fait comme suit :

```
:-type (nom-prédicat (argument 1 : type 1, …)).
```

A partir de cette déclaration, on peut créer les extensions de prédicats comme suit :

```
nom-prédicat (valeur1, valeur2, …)).
```

Exemple : on utilise la BD déductive de SGBD DES qui décrit les relations de parenté.
La définition de prédicats Father et Mother se résume dans la figure suivante :

```
% Optionally declare types
:-type(father(father:string,child:string)).
father(tom,amy).
father(jack,fred).
father(tony,carolII).
father(fred,carolIII).

% Optionally declare types
:-type(mother(mother:string,child:string)).
mother(grace,amy).
mother(amy,fred).
mother(carolI,carolII).
mother(carolII,carolIII).
```

Figure 4.1. Exemple de définition de prédicats extensionnels et les faits sous DES

Remarque :

On peut définir des commentaires dans un programme DATALOG en utilisant le symbole "%".

La création des prédicats intensionnels sous forme des clauses de Horn se fait comme suit :

```
nom-prédicat :-nomPredicat1 (variables), nomPredicat2
(variables), opérations de comparaison.
```

Exemple : la figure suivante présente la définition de deux prédicats intentionnels Parent (x,y) et Ancestor (x,y).

```
parent(X,Y) :-
  father(X,Y)
  ;
  mother(X,Y).
% The above clause for parent is equivalent to:
% parent(X,Y) :-
%   father(X,Y).
% parent(X,Y) :-
%   mother(X,Y).

ancestor(X,Y) :-
  parent(X,Y).
ancestor(X,Y) :-
  parent(X,Z),
  ancestor(Z,Y).
```

Figure 4.2. Exemple de définition de prédicats intentionnels sous DES

Comme dans le SQL, on peut définir des contraintes d'intégrité en DATALOG par le prédicat INCORRECT_DB qui contient des règles incorrectes qui ne doivent pas figurer dans le programme DATALOG.

Exemple : soit le prédicat personne défini comme suit :

% personne avec trois arguments.

:-type (Personne (Num: integer, Nom: String, Prénom : String).

Pour assurer qu'une personne ne doit pas avoir deux numéros, on définit cette contrainte (clé primaire) comme suit :

INCORRECT_DB:- Personne (N1, Nom, Pre), Personne (N2, Nom, Pre), N1\== N2.

5.2. Evaluation de requêtes

L'évaluation ou l'exécution de requêtes se fait suivant la stratégie de chaînage avant "bottom-up" [48], où on part des faits de la base de données extensionnelle BE pour déduire de nouveaux faits en utilisant les règles déductives de la base de données intentionnelle BI.

L'évaluation de requêtes s'appuie sur la théorie du point fixe et le calcul des ensembles de faits en appliquant les opérateurs de l'algèbre relationnelle (sélection, projection, etc.).

La théorie du point fixe consiste à continuer de déduire des nouveaux faits à partir d'une requête jusqu'à ne plus pouvoir générer aucun fait nouveau.

5.2.1. Evaluation de requêtes non récursives

L'évaluation de requêtes non récursives se fait par la transformation de la requête sous forme d'algèbre relationnelle puis le calcul des opérations de l'algèbre relationnelle suivantes :

La sélection : si la relation personne définit par trois attributs: nom, prénom et âge. Nous voulons afficher les personnes âgées de 50 ans.

En SQL: SELECT * FROM Personne P WHERE P.Age=50.

En DATALOG : Personne(X, Y, 50) ou bien Personne(X, Y, Z), Z=50 avec les variables : X signifie le nom, Y le prénom et Z l'âge. Le résultat de l'évaluation de cette requête DATALOG est une suite de valeurs de X et Y par exemple :

```
X= Seridi Y=Hamid ;
X=Kolladi Y= Khiereddine ;
no
```

La projection : par exemple: quels sont les noms et les prénoms des personnes.

En SQL: SELECT Distinct P.Nom, P.Prenom FROM Personne P.

En DATALOG : Personne(X, Y, _).

L'intersection: Supposons qu'on a deux relations ayant le même schéma: Etudiant et Employé. Ces deux relations possèdent les attributs suivants : Nom, Prénom, Age, Salaire. Nous voulons afficher les étudiants salariés comme suit :

En DATALOG : on définit un nouveau prédicat intentionnel nommé "Intersection".

Intersection (X, Y, Z, T):- Etudiant (X, Y, Z, T), Employé (X, Y, Z, T).

L'union: on utilise les relations de l'exemple précédent pour afficher tous les étudiants salariés ou non.

En DATALOG, l'union se traduit par l'opération de disjonction présentée par ";".

Union (X, Y, Z, T):- Etudiant (X, Y, Z, T)**;** Employé (X, Y, Z, T).

On peut aussi créer deux règles déductives ayant la même tête nommé "Union".

Union (X, Y, Z, T):- Etudiant (X, Y, Z, T).

Union (X, Y, Z, T):- Employé (X, Y, Z, T).

La différence: par exemple, quels sont les étudiants non-salariés ?

On crée un prédicat intentionnel nommé "Différence" avec :

Différence (X, Y, Z, T) :- Etudiant (X, Y, Z, T), not (Employé (X, Y, Z, T)).

La jointure: soient Personne (num, nom, prenom) et Voiture (immatricule, marque, modèle, num) deux relations. Soit la question suivante : qui possédait de voiture ?.

On crée un prédicat intentionnel dénommé "Joint" :

Joint (X, Y, Z, T, V, W) :- Personne (X, Y, Z), Voiture (T, V, W, X). Sachant que x désigne l'argument numéro de personne.

Le produit cartésien: soient deux relations : Thésard (nom, prenom) et Thèse (intitulé, domaine). Afficher tous les thésards et les thèses existants :

On crée un prédicat intentionnel dénommé "Produit" :

Produit (X, Y, Z, T) :- Thésard (X, Y), Thèse (Z, T).

Par ailleurs, le DATALOG supporte l'utilisation des fonctions d'agrégats telles que SUM, MAX, MIN, AVG, COUNT, COUNT-DISTINCT. Pour résoudre une requête non récursive, on construit l'arbre de recherche comme dans le PROLOG.

Exemple : Le but est : Parent(X, Fred). L'arbre de recherche donné par la figure suivante :

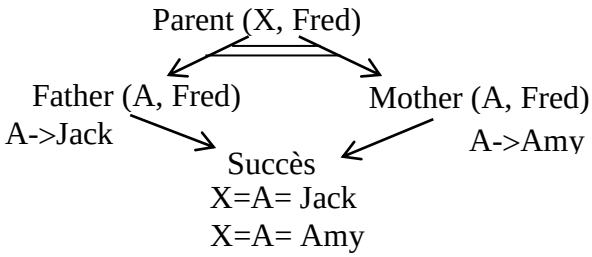


Figure 4.3. Exemple d'arbre de recherche

5.2.2. Evaluation de requêtes récursives

Le principe utilisé pour évaluer les règles récursives consiste à créer successivement des tables intermédiaires en commençant par une table vide puis en appliquant la règle déductive jusqu'à l'arrive au point fixe.

Exemple : La règle récursive Ancestor est la suivante :

Ancestor (X, Y):-Parent (X, Y). ……...…R1

Ancestor (X, Y):-Parent (X, Z), Ancestor (Z, Y). … ...…..…R2

Etape 1: création d'une table vide Ancestor 0:

Ancestor 0	
X	**Y**

Etape 2: création d'une table Ancestor 1 en appliquant la règle R1 avec :

Parent (X, Y):-Father (X, Y); Mother(X, Y).

Ancestor 1	
X	**Y**
Tom	Amy
Jack	Fred
Tony	CaroIII
Fred	CaroIIII
Grace	Amy
Amy	Fred
CaroII	CaroIII
CaroIII	CaroIIII

Etape 3: création d'une table Ancestor 2 en utilisant la règle R2.

Ancestor 2	
X	**Y**
Tom	Fred
Jack	CaroIIII
Tony	CaroIIII
Grace	Fred
Amy	CaroIIII
CaroII	CaroIIII

Etape 4: création d'une table Ancestor 3 en utilisant la règle R2

Ancestor 3	
X	**Y**
Tom	CaroIIII
Grace	CaroIIII

Cet étape représente le point fixe car il n'y'a plus de possibilités d'appliquer la règle récursive. Le résultat final est la fusion de toutes les tables intermédiaires.

Concernant les autres manipulations de données telles que l'ajout, la suppression et la mise à jour, le DATALOG dispose deux fonctions : ASSERT pour l'ajout d'un fait et RETRACT

pour la suppression d'un fait. Pour la mise à jour, DATALOG ne dispose pas une fonction de mise à jour mais elle peut être interprétée par une suppression suivie par un ajout.
Exemple : Pour ajouter au père Tom le fils Anne, on exécute la règle suivante :

/ASSERT(Father (Tom, Anne)).

Finalement, malgré que le DATALOG permet de faire des raisonnements et déduire des nouveaux faits à partir d'une grande base de données, il avait montré des limites telles que l'expression des clauses GROUP BY et HAVING, etc. Le SQL reste toujours le langage le plus professionnel et le plus performant pour la gestion efficace de bases de données.

Série TD N°4

(Bases de données déductives)

Exercice 01. Création d'une BD déductive

Soit le schéma relationnel relatif à la gestion de commandes :

Fournisseur (**Id**, Nom, Prénom)

Commande (**Num**, Date, Total, Num-fournisseur)

Article (**Code**, Désignation, Prix, commande)

Questions :

Supposons que cette BD relationnelle dispose les n-uplets suivants :

Les n-uplets de fournisseur	Les n-uplets de commande	Les n-uplets d'Article
123, Fathi, Karim	0001, 01/09/2018,40.000, 456	485, bureau, 20.000, 0001
456, Ramdani, Said	0013, 19/10/2018,80.000, 101	361, imprimante, 30.000, 0013
789, Dridi, Yacine	0151, 22/08/2018,35.000, 123	211, PC, 80.000, 501
101, Kara, Hamza	501, 30/05/2018,97.000, 789	432, Onduleur, 10.000, 0151

1. Ecrire un programme DATALOG qui permet de créer une BD déductive sachant que cette base contient le prédicat "Acceptation" décrivant les commandes acceptées par les fournisseurs.

2. Assurer-vous que : Id du fournisseur, Num de commande et Code de l'article sont uniques.

Exercice 02. Interprétation et évaluation de requêtes

Soient deux prédicats extensionnels baptisés "Arc" et "Chemin". Ces deux prédicats ayant les mêmes arguments (NumSource, NumDestination). Un chemin est un ensemble d'arcs.

1. Définir le prédicat intentionnel Chemin.

2. Evaluer le prédicat Chemin si les extensions du prédicat Arc sont représentées ci-dessous.

3. Quel est le nombre d'extensions du prédicat Chemin ?

Arc	
1	2
1	3
2	4
4	5
3	6

Corrigé page 81.

TP 03.
(Manipulation de base de données déductive par le SGBD DES version 6.1)

Objectifs

L'objectif du TP est de vous amener à manipuler les bases de données déductives via le SGBD DES, d'afficher le graphe de dépendance, de créer des tables relationnelles et d'exprimer des requêtes en SQL et en DATALOG.

1. Travail demandé :

Le projet TPBDA comporte la base de données déductive de relations de parenté.

1. Afficher les tables relationnelles, les vues et les contraintes d'intégrités s'il existe.
2. Afficher le graphe de dépendance du fichier Family.dl (sachant que dl est l'extension du fichier datalog).
3. Ajouter les faits suivants : Father(peter, marck), Father (Marck, Bob), Mother (Jina, Marck), Mother(Lila, bob).
4. Calculer les extensions de prédicats father et mother.
5. Exprimer les questions suivantes en DATALOG :
   - 5.1. Qui sont les parents de marck ?
   - 5.2. Quelles sont les familles existantes dans cette base ?

Corrigé page 82

CHAPITRE 05

LES BASES DE DONNÉES RÉPARTIES

1. Introduction

Avec la croissance exponentielle du volume de bases de données rend leur gestion un travail fastidieux. En effet, les utilisateurs ont besoin de partager les données d'une base de données sur plusieurs sites tout en assurant la transparence et la disponibilité de données. L'objectif de ce chapitre est donc de présenter les bases de données réparties en utilisant le SGBD Oracle.

2. Base de données répartie

Une base de données Répartie (Distributed database) est un ensemble de bases de données logiquement liées et physiquement localisées sur différents sites (machines) et perçues par l'utilisateur comme une base unique. Chaque base de données possède son schéma local. Le schéma de la BD répartie constitue le schéma global qui n'est généralement pas matérialisé et il peut servir du support de conception et d'expression de requêtes [49, 50].

Remarque :

Une base de données répartie ce n'est pas une base de données centralisée accessible via le réseau.

3. SGBD Réparti

Un SGBD réparti permet de gérer les bases de données réparties en fournissant un moyen d'accès rendant la distribution transparente [66]. Les principales fonctionnalités fournissant par le SGBD réparti sont [49, 50]:

- La fragmentation d'une base de données en plusieurs fragments.
- La distribution et la localisation de fragments sur plusieurs sites.
- La réplication (ou duplication) de données pour assurer la disponibilité en cas de panne.
- La reconstruction de fragments.
- Le traitement de requêtes réparties.
- La gestion de transactions réparties.

- L'indépendance vis à vis du système d'exploitation, du réseau et du SGBD.

Prenons l'exemple de l'architecture client-serveur où chaque machine d'un réseau est un nœud qui pouvant héberger une ou plusieurs bases de données. Chaque nœud peut agir en tant que client, serveur ou les deux, en fonction de la situation [3]. Dans ce cours, nous nous focalisons sur l'utilisation de SGBD Oracle pour faire nos exemples.

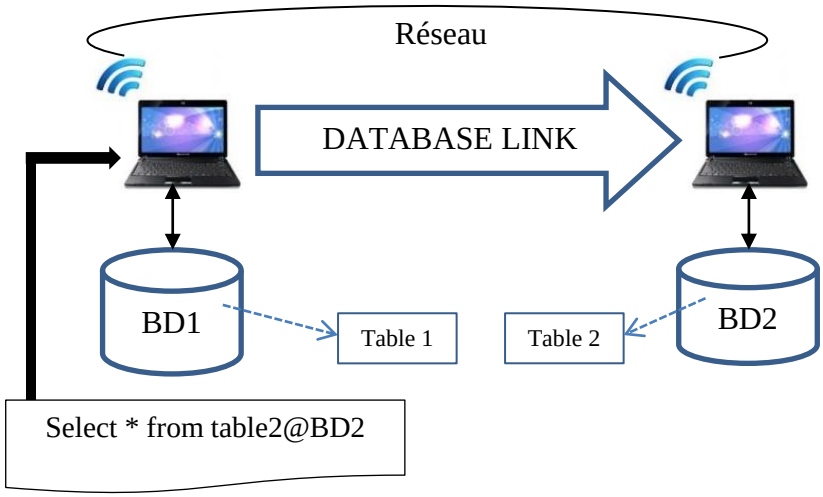


Figure 5.1. Architecture Client-Serveur d'un SGBD réparti [3]

Le SGBD réparti oracle assure la liaison des bases de données par la création d'un lien unidirectionnel "DATABASE LINK" d'un serveur à un client via la commande suivante :

```
CREATE [SHARED|PUBLIC|PRIVATE] DATABASE LINK Nom_du_Lien
CONNECT TO {CURRENT_USER | Nom-utilisateur IDENTIFIED BY
mot_de_passe}
USING nom_de_service;
```

Le mot clé USING nom_de_service spécifie le nom de service d'une base distante qui se trouve dans le fichier de configuration tnsnames.ora du serveur distant.

Exemple : Sur le site Serveur, on crée un DATABASE LINK nommé ServeurClient qui lie le Serveur avec le client du compte Client avec le mot de passe C20182019 et le nom du service est SITE1.

```
SQL> CREATE PUBLIC DATABASE LINK ServeurClient
  2  CONNECT TO Client IDENTIFIED BY C20182019
  3  USING 'SITE1';

Database link created.
```

Pour référencer une base de données distante, on utilise le nom global ou le lien de base de données défini par database link ou rendre la localisation purement transparente par l'utilisation des synonymes.

Exemple : création du synonyme de la table distante Projet@ServeurClient.

```
SQL> CREATE OR REPLACE SYNONYM ProjetS
  2  FOR Projet@ServeurClient;

Synonym created.
```

Pour supprimer un lien entre deux bases de données, on exécute la commande suivante :

```
DROP DATABASE LINK nom_du_lien;
```

4. Conception d'une base de données répartie

Il existe deux approches de conceptions de base de données réparties : une approche ascendante (Bottom up design) ou approche par intégration et une approche descendante (top down design) aussi appelée approche par décomposition.

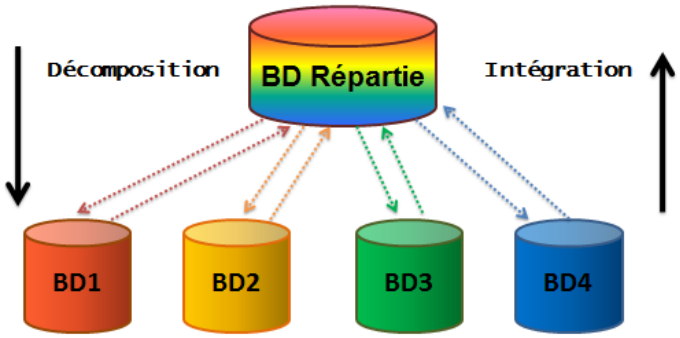


Figure 5.2. Approches de conception de base de données répartie

4.1. Approche de conception ascendante

Cette approche se base sur le fait que la répartition est déjà faite, et on intègre les schémas locaux de différentes BD existantes en un seul schéma conceptuel global [51]. Cette approche nécessite des traitements d'hétérogénéités sémantiques et syntaxiques des schémas locaux pour assurer leur intégration dans un seul schéma [51].

4.2. Approche de conception descendante

Cette approche assume que le schéma global de BD répartie est préexistant et à partir de ce schéma, on définit les schémas locaux des bases de données locales [51].

Le schéma global est pris en charge de complexité de la répartition (fragmentation et allocation sur sites avec réplication). Le choix d'une telle approche de conception dépend le problème à étudier. On utilise la conception descendante si on part de néant sinon on utilise la conception ascendante.

5. Fragmentation

La fragmentation est le processus de décomposition d'une base de données en un ensemble de petites bases de données dites bases de données locales [51]. Le processus de fragmentation doit assurer la réplication (ou la duplication) de fragments sur plusieurs sites afin de favoriser les accès locaux, diminuer du coût imposé par les transmissions, assurer l'équilibrage de la charge de travail entre les sites et augmenter la disponibilité des données.

5.1. Fragmentation par répartition des relations

Ce type de fragmentation consiste à distribuer les relations (ou classes) sur différents sites. La reconstruction du schéma global est basée sur la réunion de différents schémas locaux.

Exemple : le schéma global contient deux relations : Etudiant et Groupe.

En appliquant ce type de fragmentation, on obtient :

Sur le site 1 : fragment 1= Etudiant (**nEtudiant**, nom, prénom, âge, NGroupe).

Sur le site 2 : fragment 2= Groupe (**Numéro**, Intitule).

5.2. Fragmentation horizontale

Ce type de fragmentation consiste à partitionner les données d'une relation selon un certain nombre de critères. Ce type de fragmentation se fait par l'opération de sélection et la reconstruction est donc faite par l'union de fragments.

Exemple : la relation Etudiant peut être fragmentée en deux fragments selon l'attribut âge:

En algèbre relationnelle : Etudiant20= $\sigma_{âge<20}$ (Etudiant) et Etudiantsup20= $\sigma_{âge\geq20}$ (Etudiant).

La reconstruction de la relation Etudiant est comme suit :

Etudiant = Etudiant20$\cup$ Etudiantsup20.

5.3. Fragmentation verticale

La fragmentation verticale consiste à partitionner les données d'une relation en fragments selon un certain nombre d'attributs par la projection et l'utilisation d'un attribut commun entre les fragments pour permettre la reconstruction de relation initiale en utilisant la jointure entre fragments. Cet attribut commun est souvent la clé primaire de la relation.

Exemple : Etudiant1= $\pi_{nEtudiant, nom, NGroupe}$ (Etudiant).

Etudiant2= $\pi_{nEtudiant, Prénom, âge}$ (Etudiant).

La reconstruction de la relation Etudiant est comme suit : Etudiant= Etudiant1$\bowtie$ Etudiant2.

5.4. Fragmentation hybride

Elle combine la fragmentation horizontale et la fragmentation verticale. La répartition est donc faite par la combinaison de projection et de sélection. Ainsi, la reconstruction se fait par la combinaison de jointure et d'union.

5.5. Fragmentation horizontale dérivée

Cette fragmentation consiste à définir des fragments d'une relation par une semi jointure avec un fragment d'une autre relation.

6. Réplication

La réplication consiste à dupliquer et maintenir les fragments sur différents sites pour augmenter la disponibilité de données, réduire la charge du réseau [51, 52].

Il existe plusieurs techniques de réplication de fragments telles que la commande Copy, la création de clichés (Snapshot) et les vues matérialisées. Dans ce cours, nous nous focalisons sur l'utilisation de vues matérialisées pour la réplication de fragments.

Les vues matérialisées (materialized views) est la technique la plus récente et la plus utilisée pour la réplication de fragments [53].

```
CREATE MATERIALIZED VIEW <nom de vue matérialisée>
[TABLESPACE … STORAGE …]
[REFRESH FAST|COMPLETE|FORCE]
START WITH sysdate
NEXT sysdate [+valeur]  // fixer les dates de mise à jour
WITH PRIMARY KEY // si possible …
ENABLE QUERY REWRITE  …
AS <requête SELECT> ;
```

7. Traitement de requêtes réparties

Il existe trois types d'accès aux bases de données réparties [51-53] :

1. **SGBD Réparti :** Le SGBD réparti pris en charge à l'exécution et à l'optimisation de requêtes réparties d'une façon implicite sans intervention d'utilisateur.
2. **Les vues réparties :** Au niveau du serveur, on peut créer des vues réparties dont leurs tables sont réparties sur différents sites.
3. **Les connecteurs:** l'accès aux BD réparties peut être fait par le biais des connecteurs tels que JDBC (Java Database Connection), ODBC (Open Database Connectivity) et RDA (Remote Data Access).

8. Gestion des transactions concurrentes

Une transaction est une suite d'opérations menée sur une base de données qui doit être soit validée par un COMMIT, soit annulée par un ROLLBACK.

Remarque :

Les opérations de mise à jour de données (insert, delete et update) démarrent par défaut une transaction implicite avec un commit, sans une annulation rollback.

Les transactions permettent de résoudre les problèmes de la concurrence d'accès aux BD en rendant invisible aux clients le partage simultané des données.

Remarque :

La gestion des transactions concurrentes permet d'assurer que l'exécution simultanée des transactions produit le même résultat que leur exécution séquentielle

Un ensemble de transactions concurrentes {T1, …, Tn} pose des conflits sur les opérations de Read et Write qui sont:

- Perte de mise à jour lorsque deux transactions modifient simultanément la même valeur.
- Lecture impropre lorsque une transaction annule l'écriture. Ex : r1(x), w1(x), r2(x), w2(y), Rollback1, commit2. La transaction 2 lit une valeur de x mise à jour par la transaction 1 mais qui ne sera finalement pas validée (Rollback1).
- Lecture non reproductible lorsque une transaction lit deux valeurs différentes pour la même variable. Ex : r1(x), r2(x), w1(x), r2(x). T2 lit deux valeurs de A différentes.
- Objets fantômes destinées aux variables de type collection (liste, ensemble, table, etc.): après la lecture deux fois de même collection, on observe des objets supplémentaires à cause de mise à jour.

T1	T2	A=10
R1(a1←A)		A=10
	R2(a2←A)	A=10
a1←a1+15		
W1(A←a1)		A=25
	a2←a2+30	
	W2(A←a2)	A=40

Exemple : Les transactions T1 et T2 modifient simultanément la valeur de A initialement égale à 10

Les modifications effectuées par T1 sont perdues.

Pour éviter les anomalies présentées précédemment, plusieurs solutions ont été proposées telles que *le verrouillage* à travers un Verrou qui représente la technique la plus classique et *l'ordonnancement (*scheduling) des transactions. Nous nous focalisons dans ce cours sur la technique d'ordonnancement.

Un ordonnancement O est un ordre d'exécution des opérations aij des transactions Ti.

Exemple : Soit l'ordonnancement O suivant:

O=R1(A) W2(A)R2(A)R1(A) W1(A).

Donner le résultat de O pour la valeur initiale A=10 et W=A+1.

Réponse :

1) T1: A=10
2) T2: A=11
3) T2: A=11
4) T1: A=11
5) T1: A=12 Alors le résultat de O est A=12.

Remarque :

Un ordonnancement est **Sérialisable** si elle donne pour chaque transaction participante, le même résultat que l'exécution en série de ces mêmes transactions.

Dans l'exemple précédent, O n'est pas sérailisable car A=11 (T1) et A=12 (T2) alors que dans O A=12 (T1) et A=11 (T2).

Dans ce cours, nous avons présenté les notions fondamentales sur les bases de données réparties et les SGBD réparti notamment le SGBD Oracle. Des exemples des SGBD répartis comme : DB2, SQL Server, Sybase, Informix, Ingres.

Malgré les avantages des bases de données réparties notamment l'accroissement de vitesse de traitement et la disponibilité de données, on trouve des inconvénients les plus importants sont : administration complexe, distribution du contrôle, difficulté de migration et coût important.

Série TD N°5

(Bases de données réparties)

Exercice 01. Conception descendante d'une BD répartie

Soit le schéma global de la base de données répartie de gestion des projets de recherche scientifique des enseignants sur trois universités algériennes: Guelma, Constantine, Annaba.

Enseignant (**Enum**, Nom, Prénom, Grade, Université, Pcode).

Projet (**Pcode**, Titre, description, PUniversité, année).

Questions :

1. Proposer une bonne conception descendante (top down design) de la base de données sur ces trois universités en s'appuyant sur les deux hypothèses suivantes :

   H1. Chaque université gère ses propres projets de recherche.

   H2. Le numéro d'enseignant, son nom et son prénom sont des informations connues dans toutes les universités comme informations d'un chercheur par contre les attributs le grade, l'université et le code du projet sont privés.

2. Assurez-vous qu'on peut reconstruire le schéma global à partir de ces fragments.
3. Créer la vue matérialisée Ens-projet pour rendre disponible les données suivantes : pour chaque titre de projet de l'université de Guelma, afficher la description du projet et la liste des enseignants (le nom, le prénom). Assurer la mise à jour de données tous les trois jours.

Exercice 02. Traitement de requêtes réparties

Reprendre la base de données répartie de l'exercice 01 et traduire la question suivante en requêtes réparties selon les types d'accès suivant : Vues réparties et le connecteur RDA.

Au niveau de l'université de Guelma, chercher tous les projets faits en 2018.

Exercice 03. Gestion des transactions concurrentes

Soient les transactions T1, T2 suivantes :

T1	**T2**
R1(a1←A)	
a1← a1-2	
	R2(a2←A)
	A2← a2+3
W1(A← a1)	
R1(b1←B)	
	W2(A← a2)
b1← b1+2	
W1(B← b1)	

1. Etant donnés A=4 B=8, quel est le résultat correct d'exécution simultané de T1T2.

2. Soit l'ordonnancement O suivant avec W(A)=A-2, W1(B)=B+2 et W2(B)=B+1:
R1(A)R2(B)W1(A)W2(B)W1(B)R2(A)R1(B).
   a) Donner le résultat de O.
   b) O est-il sérialisable?

Corrigé page 83.

TP 04.
(Manipulation de base de données répartie sous ORACLE)

Objectifs

L'objectif principal du TP est de manipuler les bases de données réparties sur plusieurs machines. De ce fait, on utilise le logiciel de virtualisation de machines VMWare pour la création des sites clients. Une configuration correcte du réseau entre machines est un travail principal pour réussir l'accès à une base de données distante (remote database).

1. Les outils nécessaires :

1.1. SGBD Oracle database 11g Express Edition.

1.2. VMware : un logiciel de virtualisation qui permet d'émuler plusieurs systèmes d'exploitation sur la même machine.

2. Configuration entre un serveur et un client:

2.1. Utiliser VMware et créer une machine virtuelle baptisée Site1 (assurez-vous que l'option du BIOS intel (R) virtualization technology est Enable).

2.2. Installer Oracle 11g sur les deux machines (machine physique « Serveur » et machine virtuelle « client » Site1.

2.3. Sur le site1, récupérer l'adresse IPv4 et rendre cette adresse statique.

2.4. Tester par la commande Ping la bonne connexion entre les machines.

2.5. Le processus d'écoute Oracle (LISTENER) est un service permettant à des clients d'utiliser le protocole TCP pour accéder à une base de données distante.

- Sur le site 1, aller au répertoire $ORACLE_HOME/Network/Admin/ et ouvrir le fichier listener.ora. Ajouter dans SID_LIST_LISTENER le code suivant :

```
(SID_DESC =
(SID_NAME = XE)
(ORACLE_HOME=
C:\oraclexe\app\oracle\product\11.2.0\server))
```

2.6. Configuration du fichier tnsnames.ora du serveur : ouvrir le fichier tnsnames.ora qui se trouve dans le répertoire : $ORACLE_HOME/Network/Admin/ et ajouter le code suivant:

```
SITE1 =
(DESCRIPTION =
(ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.1.5)(PORT =
1521))
   (CONNECT_DATA =(SID = XE)))
```

3. Création de DATABASE LINK entre le serveur et le client:

3.1. Sur le SITE1: Créer un compte utilisateur avec username= Client et le mot de passe est C20182019 et l'accorder tous les privilèges.

3.2. Sur le Serveur: Créer le compte utilisateur username=Serveur et le mot de passe est S20182019 et l'accorder tous les privilèges.

3.3. Sur le Serveur: Créer un database link nommé ServeurClient entre le serveur et le client. Sur le SITE1: Créer un database link nommé ClientServeur entre le client et le serveur.

4. Création de base de données répartie :

4.1. Connecter en tant qu'utilisateur et créer sur le serveur la table Projet vue en TD qui contient les projets de recherche de l'université de Guelma.
Projet (**Pcode**, Titre, description, PUniversité, année).

- Créer sur le site1, la table Projet qui contient les projets de Constantine.
- Insérer 7 projets sur le serveur et 2 projets sur le client.

4.2. Sur le site Serveur, afficher le nombre de projets du site Client.

4.3. Sur le site Serveur, créer une réplication par une vue matérialisée des projets de site 1 avec une mise à jour chaque semaine.

Corrigé page 84.

CHAPITRE 06

LES BASES DE DONNÉES MULTIMÉDIAS

1. Introduction

L'utilisation quotidienne de données multimédias et le besoin de les stocker, les organiser et les manipuler ont fait apparaître une nouvelle génération de bases de données appelées base de données multimédias (BDMM).

Ce dernier chapitre de ce polycopié de cours présente dans un premier temps la notion de multimédia puis nous allons définir la base de données multimédia par l'utilisation de type large object (LOB). Par la suite, nous présenterons les bases de données NoSQL, plus précisément les bases de données orientées document de MongoDB.

2. Les données multimédias

Le média désigne un moyen de transmission, de stockage ou de présentation des informations [54]. De telles données comportent plusieurs types de média utilisés conjointement sont appelées données multimédias [55]. Ces données peuvent être rendues accessibles via le web ou stockées dans des bases de données.

Les données multimédias sont par nature volumineuses qui sont caractérisées par les propriétés suivantes [54]: Capacité de stockage importante, Utilisation de formats de représentations différents, Utilisation des outils particuliers de production et de création, Interrogation particulière et la recherche par le contenu, Utilisation des outils de traitement (indexation, segmentation, compression), Besoins de la distribution (serveur de streaming).
Il existe deux catégories de médias [54]: les médias statiques (texte et image) et les médias dynamiques (audio et vidéo).

3. Définition de base de données multimédia

Les bases de données multimédias (BDMM) s'appuient le plus souvent sur des architectures de base de données existantes, les plus utilisées étant le modèle relationnel et le modèle

orienté objet. Elles ont la capacité de stocker les descripteurs de données multimédias à savoir les histogrammes de couleur pour les images.

Exemple : Dans une BD des films définie par des données classiques, telles que les titres de films, des noms des acteurs, le nom du producteur, année de sortie, etc. et les films eux-mêmes stockés dans un répertoire et représentés par le chemin de stockage de ces films. Cette BD permet de représenter ce qu'on veut d'une manière atomique, par contre dans une BDMM, ce qu'on veut, c'est travailler sur des éléments multimédia, par exemple, dans quel film on a telle image d'acteur?

Le SGBD multimédias doit offrir les mêmes caractéristiques qu'un SGBD classique (persistance, gestion des transactions, interrogation …) avec les fonctions suivantes [4]:

1. Le stockage de tous les types de données multimédia.
2. La manipulation de données multimédias.
3. La gestion efficace de données volumineuses.
4. La recherche par le contenu dont la requête peut contenir des objets multimédias.

4. Types de données multimédias sous Oracle

C'est à partir de SQL3 qu'on peut manipuler les données multimédias par les types de large objets, appelés les types LOB (Large OBject). Selon la localisation des LOB au niveau de la BDMM, nous pouvons distinguer deux types de LOB [4, 12, 55] :

- *Les LOB internes* sont stockés directement dans la table de la BDMM et référencés par un pointeur logique appelé *Locator* qui pointe vers les données. Le type binaire BLOB (Binary Large OBject) pour stocker les données binaires (vidéo, image et audio) et le type de caractères longs CLOB (Character Large Object) pour stocker les données textuelles. Le NCLOB (National CLOB) pour les chaînes de caractères Unicode.
- *Les LOB externe* comme son nom l'indique, les données sont stockées dans un fichier externe et référencées à l'aide d'un pointeur. Le BFILE (Binary File) est un LOB externe où les données se trouvent dans un fichier externe à la base de données (sur disque dure, etc.) et relient avec elle à travers Locator qui pointe vers ce fichier.

5. Manipulation de BDMM sous Oracle

La déclaration d'un attribut de type LOB se fait comme une déclaration d'attribut de type simple.

Exemple : Soit la table Fruit définit par un identificateur id, un nom du fichier image et l'image de type Blob. La création de cette table se fait comme suit :

```
SQL> CREATE TABLE Fruit (
  2  id INTEGER PRIMARY KEY,
  3  Nom VARCHAR2(80),
  4  Image BLOB);

Table created.
```

Dans l'attribut de type BLOB (attribut image), les valeurs qui sont des fichiers images sont stockés dans un répertoire nommé DirFruit qui a été créé comme suit :

```
SQL> connect /as sysdba
Connected.
SQL> CREATE OR REPLACE DIRECTORY DirFruit AS 'E:\Fruit\';

Directory created.

SQL> GRANT ALL ON DIRECTORY DirFruit to PUBLIC;

Grant succeeded.
```

Quel que soit le type de LOB (LOB externe ou interne), sa manipulation nécessite l'utilisation de programme PL/SQL et des méthodes du package DBMS_LOB telles que *empty_blob()* pour initialiser le type blob, *Loadfromfile()* pour le chargement de données multimédias à partir de fichier BFILE, etc.

Exemple. L'insertion de données BFILE se fait comme suit :

```
SQL> INSERT INTO Plante VALUES(200, 'Fleure blanche',
  2  BFILE('DIRFLOWERS', 'FleurB.jpg'));

1 row created
```

La recherche d'image par le contenu **CBIR** (**C**ontent-**B**ased **I**mage **R**etrieval) consiste à rechercher des images à partir de leurs descripteurs (signatures ou vecteurs de caractéristiques) représentant leurs contenus visuels.

Oracle dispose la bibliothèque ORDVIR (Object Relational Data Visual Information Retrieval) pour la recherche par le contenu de base d'images définie par le type ORDImage.

6. Les bases de données NoSQL

Les bases de données actuelles sont souvent volumineuses et qui ne suivent pas le même modèle de données (XML, BDR, etc.). La gestion de ces grosses données (Big data) est impossible via les SGBD existants. Le Bases de données NoSQL (Not Only SQL) ou non seulement SQL, représente une alternative de bases de données SQL pour le stockage et le traitement de Big data [56].

Les données NoSQL ne suivent pas tous le même schéma « SchemaLess ou Schema free ». Le stockage de données est faite d'une manière souple contrairement aux données SQL qui faite avec précaution selon un schéma fixé avec contraintes.

Propriétés de données NoSQL:

- **Consistency:** tous les nœuds voient la même version de données.
- **Availability**: chaque requête obtient une réponse par la réplication de données sur le réseau.
- **Partition tolerance**: la perte de messages n'empêche pas le système de continuer à fonctionner.
- **Scalability :** La possibilité d'ajouter des nœuds sans problème.
- **Gestion des informations incomplètes.**
- **Traitement parallèle de données**

Il existe quatre types de Bases de données NoSQL [57] :

1. **Clé / valeur:** (Redis, Riak et Voldemort de LinkedIn…).

   Domaines: gestion de fichiers log, gestion de cache, de fraude, etc.

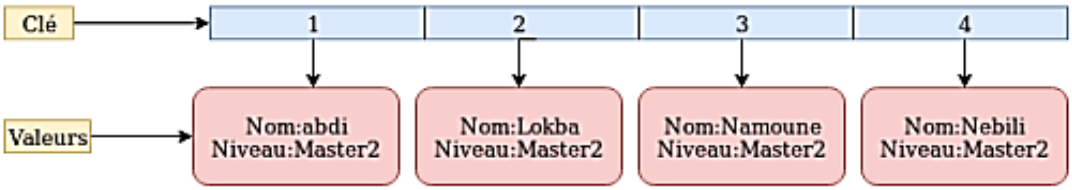


Figure 6.1. Exemple de données NoSQL orientées Clé-Valeur.

2. **Document:** (MongoDB, CouchBD , DynamoDB ...).

   Domaines: manipulation de données web (document json, javascript, etc.).

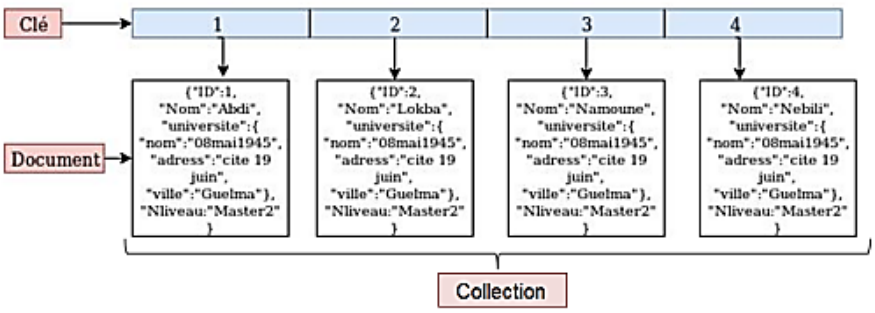


Figure 6.2. Exemple de données NoSQL orientées Document.

3. **Colonne:** (Cassandra, HBASE de Hadoop, Accumulo...).

Domaines: comptage, moyennes, gestion de stocks, etc.

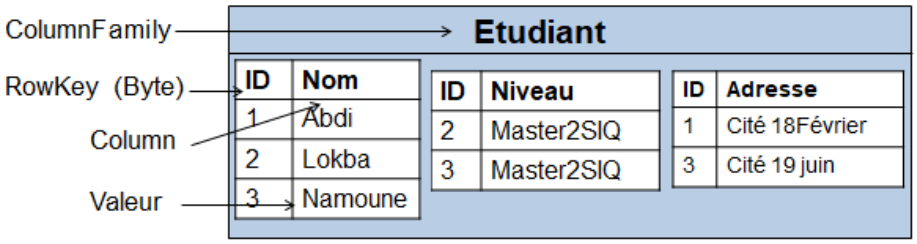


Figure 6.3. Exemple de données NoSQL orientées Colonne.

4. Orientées graphe: (Neo4J, ArangoDB, Hyper GraphDB...)

Domaines: gestion de réseaux sociaux, réseau électrique, recommandation, etc.

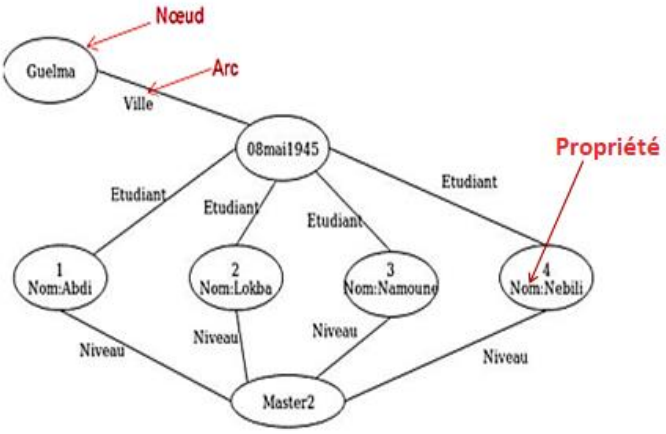


Figure 6.4. Exemple de données NoSQL orientées Graphe.

Dans ce polycopié de cours, nous nous focalisons sur l'une des bases de données NoSQL les plus populaires qui est la base de données orientée document du système MongoDB.

6.1. Les bases de données MongoDB

MongoDB est un SGBD orienté documents qui se compose de:

- Mongo: interpréteur des commandes
- Mongod: le moteur de base de données.
- Mongos: est le service de partitionnement (sharding).
- MongoDB est très pratique lorsqu'on travaille sur le web. Il n'a pas un schéma fixé comme SGBDR.

Une BD orientée documents est un ensemble de collections. Une collection est composée d'un ensemble de documents JSON identifiés par un identifiant unique _id de type ObjectId (12-byte BSON).

La gestion de BD MongoDB est faite via des commandes shell, par exemple, Afficher la BD courante en écrivant le mot clé db.

6.2. Manipulation de BD MongoDB

- Créer ou sélectionner une BD: use <name>. Exemple :

```
> use Module
switched to db Module
```

- Suppression d'une BD: db.dropDatabase(). Exemple de suppression de la base de données nommée BDD.

```
> db.dropDatabase()
{ "dropped" : "BDD", "ok" : 1 }
```

- La création d'une collection de BD courante:

```
> db.createCollection("modules")
```

- Création d'un document: Exemple de création de document module1:

```
> module1={"title":"BDA", "coeff": 3, "Credit": 5, "Examen": 60, "TD":20, "TP":20}
```

- Insertion des documents par la commande db.collection.insert ou insertMany([<doc>]*). Exemple : insertion de document nommé module1 est faite comme suit :

```
> db.modules.insert(module1)
WriteResult({ "nInserted" : 1 })
```

- Afficher les collections par la commande show collections. Exemple :

```
> show collections
modules
```

- Modification de données : db.collection.update({condition}, {$Set: {field:valeur}})

```
> db.modules.update({"title":"BDA"}, {$set: {"coeff":4}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

- Suppression d'un document: db.collection.remove(condition)

```
> db.modules.remove({"title":"IA"})
WriteResult({ "nRemoved" : 1 })
```

- Interrogation:
  1. db.collection.find() pour afficher tous les documents.
  2. db.collection.find().pretty() est comme 1 avec une amélioration d'affichage.

- Opérateur in:

```
> db.modules.find({"title": {"$in":["BDA", "IA"]}})
{ "_id" : ObjectId("5de2dfe9e98f4949085fbe6f"), "title" : "BDA", "coeff" : 4, "C
redit" : 5, "Examen" : 60, "TD" : 20, "TP" : 20 }
{ "_id" : ObjectId("5de575bb8265543774b8ab60"), "title" : "IA", "coeff" : 2, "cr
edit" : 2 }
```

- Condition:

```
> db.modules.find({"coeff": {$lt:3}})
{ "_id" : ObjectId("5de575bb8265543774b8ab60"), "title" : "IA", "coeff" : 2, "cr
edit" : 2 }
```

- Recherche avec OR :

```
> db.modules.find({$or:[{"coeff":2}, {"title": "IA"}]})
{ "_id" : ObjectId("5de575bb8265543774b8ab60"), "title" : "IA", "coeff" : 2, "cr
edit" : 2 }
```

Série TD N°6

(Bases de données multimédias)

Exercice 01. Manipulation de Base de données multimédias

Nous voulons construire une base d'image botanique contenant la description des plantes et des fleurs. Une plante se définit par un code unique et un nom. Elle peut être de type fleure, plante d'intérieur ou plante d'extérieur. Chaque plante ayant une image de plante de type BFILE.

Questions :

1. Donner le schéma objet-relationnel de cette base d'images et assurez-vous que l'attribut Type ne doit prendre que les valeurs suivantes : Fleure, Plante externe et Plante interne.
2. Créer le répertoire RMM du chemin C:\botanique pour stocker les images de plantes.
3. Ecrire une requête SQL3 qui permet d'insérer la plante d'intérieur AleoVera ayant le code 113 avec un fichier image baptisé AleoVera.jpg.
4. Soient Q une image exemple donnée en requête représentée par le vecteur Q= (2, 0, 3, 5) et B1 et B2 deux images représentées par les vecteurs D1 = (2.6, 0, 3, 5) et D2 = (2, 0, 2.75, 5) respectivement. En utilisant la distance Euclidienne, quelle est l'image la plus similaire à Q ?

Exercice 02. Manipulation de Base de données NoSQL

Soit donné le schéma objet-relationnel suivant:

```
CREATE TYPE Size_t AS OBJECT(W FLOAT, H FLOAT);
CREATE TYPE Piece_t AS OBJECT(Id INTEGER, Name VARCHAR2(40), Sizee Size_t,Quantity INTEGER);
CREATE TABLE Piece OF Piece_t(PRIMARY KEY(Id));
```

Questions :

1. Créer la base de données orientée documents nommée BDP, qui comporte la collection Pièces. Cette collection contient deux pièces.
2. Traduire les requêtes ci-après en des commandes Mongo shell:

   2.1. Les pièces ayant une quantité plus de 1000.

   2.2. Le nom des pièces dont la largeur est entre 15 et 25 et la hauteur est égal à 50.

Corrigé page 85.

TP 05.
(Manipulation de base de données multimédia sous ORACLE JDeveloper)

Objectifs

Le but de ce TP est d'apprendre comment manipuler les bases de données multimédias en utilisant le SGBD Oracle 11g et l'IDE Oracle JDeveloper 11g. Nous nous focalisons sur les données images afin de montrer comment visualiser ce type de données dans un navigateur par le biais du connecteur JDBC qui permet de connecter une base de données avec une application, un applet ou servlet.

1. Création de base de données multimédia sous SQL Developer:

Utiliser l'outil Oracle SQL Developer et créer la table relationnelle nommée Botanique vue en TD qui contient la description des plantes et des fleurs avec leurs images.

Botanique (**Code**, Nom, Type, Photo). Avec l'attribut Photo de type BLOB.

2. Création de code java de la page JSP:

2.1. Double clic sur le bouton upload image et ajouter une classe java nommée uploadImage et écrire le code qui se trouve dans le fichier codeupload.txt.

2.2. Dans la propriété Value d'input file, sélectionner Expression Builder puis choisir ADF Managed Bean et sélectionner UploadImageBean puis la variable file.

2.3. Dans le bouton Commit, supprimer le text qui se trouve dans la propriété Disable..

2.4. Dans l'arborescence ViewController, cliquer avec le bouton droit sur View puis New et ajouter une nouvelle Servlet baptisée VisualisationBLOB. Dans les propriétés de l'image, ajouter l'URL : /visualisationblob.

2.5. Dans le fichier visualisationblob.java, écrire le code dans le fichier codeimage.txt.

2.6. Dans la propriété d'image Source URL, ajouter la source suivante : ?codej=#{bindings.Code.inputValue}.

2.7. Exécuter la page JSP par le bouton Run.

Corrigé page 86.

Solution de Série TD N°1

Exercice 01.

```
SQL> CREATE OR REPLACE TYPE Auteur_t;
  2  /

Type created.

SQL> CREATE OR REPLACE TYPE Ens_Auteurs AS TABLE OF REF Auteur_t;
  2  /

Type created.

SQL> CREATE OR REPLACE TYPE Document_t;
  2  /

Type created.

SQL> CREATE OR REPLACE TYPE Ens_documents AS TABLE OF REF Document_t;
  2  /

Type created.

SQL> CREATE OR REPLACE TYPE Paragraphe_t AS OBJECT(NumP NUMBER, TitreP VARCHAR(4
0), Contenu VARCHAR2(1000), Doc REF Document_t);
  2  /

Type created.

SQL> CREATE OR REPLACE TYPE Ens_paragraphe AS TABLE OF REF Paragraphe_t;
  2  /

Type created.

SQL> CREATE OR REPLACE TYPE Document_t AS OBJECT (NumD NUMBER, TitreD VARCHAR2(1
00), Type VARCHAR2(40), Date_creation DATE, Corps Ens_paragraphe, Auteurs Ens_auteurs) ;
  2  /

Type created.

SQL> CREATE OR REPLACE TYPE Auteur_t AS OBJECT (NumA NUMBER, NomA VARCHAR2(50),
  2  Affiliation VARCHAR2(80), Email VARCHAR2(80), Docs Ens_Documents);
  3  /

Type created.


SQL> CREATE TABLE Auteur OF Auteur_t(Primary key(NumA))
  2  NESTED TABLE Docs STORE AS axdocs;

Table created.

SQL> CREATE TABLE Document OF Document_t(Primary key(NumD), CHECK(Type IN('rappo
rt', 'Article', 'Livre')))
  2  NESTED TABLE Corps STORE AS axcorps
  3  NESTED TABLE Auteurs STORE AS axauteurs;

Table created.

SQL> CREATE TABLE Paragraphe OF Paragraphe_t(Primary key(NumP));

Table created.
```

Exercice 02.

1.
```
SQL> INSERT INTO Auteur VALUES (617, 'Jiang wang', 'informatique',
  2  'Jiang.wa@yahoo.fr', Ens_documents());

1 row created.
```

2.
```
SQL> INSERT INTO Document VALUES (5491, 'Les bases de données', 'Livre',
  2  '23-07-2010',
  3  Ens_paragraphes((SELECT REF(P) FROM Paragraphes P WHERE P.Num=1)),
  4  Ens_auteurs((SELECT REF(1) FROM Auteur A WHERE A.Num=617)));

1 row created.
```

Exercice 03.

1.
```
SQL> ALTER TYPE Document_t ADD MEMBER FUNCTION NB RETURN INTEGER CASCADE;

Type altered.

SQL> CREATE TYPE BODY Document_t AS
  2  MEMBER FUNCTION NB RETURN INTEGER IS
  3  N INTEGER;
  4  BEGIN
  5  SELECT COUNT(D.NumD) INTO N
  6  FROM Document D;
  7  RETURN N;
  8  END;
  9  End;
 10  /

Type body created.
```

2.
```
SQL> DECLARE
  2  T VARCHAR2(40);
  3  BEGIN
  4  SELECT D.TtitreD INTO T
  5  FROM Document D
  6  Where D.NumD=5491;
  7  DBMS_OUTPUT.PUT_LINE('le titre du document est '|| T);
  8  END;
  9  /
```

3.
```
SQL> DECLARE CURSOR C1 IS
  2  SELECT E.NomA, E.Email
  3  FROM Ecrivain E;
  4  V C1%ROWTYPE;
  5  BEGIN
  6  OPEN C1;
  7  LOOP
  8  FETCH C1 INTO V;
  9  DBMS_OUTPUT.PUT_LINE('les noms des auteurs sont: '|| V.NomA);
 10  DBMS_OUTPUT.PUT_LINE('**********************************');
 11  DBMS_OUTPUT.PUT_LINE('les emails des auteurs sont: '|| V.EMAIL);
 12  EXIT WHEN C1%NOTFOUND;
 13  END LOOP;
 14  CLOSE C1;
 15  END;
 16  /
les noms des auteurs sont: JIANG WANG
**********************************
les emails des auteurs sont: jiang@yahoo.fr
PL/SQL procedure successfully completed.
```

Exercice 04.

1.
```
SQL> ALTER TYPE Paragraphe_t ADD ATTRIBUTE NUM NUMBER CASCADE;

Type altered.

SQL> ALTER TABLE Paragraphe ADD CONSTRAINT FK FOREIGN KEY(NUM) REFERENCES
  2  Document(NumD);

Table altered.

SQL> CREATE VIEW Vue OF Document_t WITH OBJECT OID(NumD) AS
  2  SELECT D.NumD, D.TitreD, D.Type, D.Date_creation, CAST(MULTISET(
  3  SELECT REF(P) FROM Paragraphe P WHERE P.Num=D.NumD) AS Ens_paragraphes), au
teurs
  4  FROM Document D;

View created.
```

2.
```
SQL> CREATE TRIGGER T1 AFTER UPDATE OF TitreD ON Document
  2  FOR EACH ROW
  3  BEGIN
  4  DBMS_OUTPUT.PUT_LINE('Le nouveau titre est: '|| :New.TitreD);
  5  END;
  6  /

Trigger created.
```

Solution de Série TP N°1

Configuration du système d'exploitation :

- Consultation de variables d'oracle : Regedit – Software -- Oracle --
- Définition de variables d'orcale:

```
C:\Users\Aicha>SET ORACLE_HOME=C:\oraclexe\app\oracle\product\11.2.0\server
C:\Users\Aicha>SET ORACLE_BASE=C:\oraclexe\app\oracle
```

- Création de l'instance BDEU :

```
C:\Windows\system32>ORADIM -NEW -SID BDEU
Instance created.
```

Création de la Base BDEU :

```
SQL> CREATE DATABASE BDEU
  2  USER SYS IDENTIFIED BY wX#12
  3  USER SYSTEM IDENTIFIED BY az#78
  4  LOGFILE GROUP 1 ('%ORACLE_BASE%/oradata/BDEU/redo1.log') SIZE 20M,
  5             GROUP 2 ('%ORACLE_BASE%/oradata/BDEU/redo2.log') SIZE 20M,
  6             GROUP 3 ('%ORACLE_BASE%/oradata/BDEU/redo3.log') SIZE 20M
  7     DATAFILE '%ORACLE_BASE%/oradata/BDEU/system.dbf' SIZE 100M AUTOEXTEND ON
 NEXT 10M MAXSIZE UNLIMITED EXTENT MANAGEMENT LOCAL
  8     SYSAUX DATAFILE '%ORACLE_BASE%/oradata/BDEU/sysaux.dbf' SIZE 50M AUTOEXT
END ON NEXT 10M MAXSIZE UNLIMITED
  9     UNDO TABLESPACE UNDO DATAFILE '%ORACLE_BASE%/oradata/BDEU/undo.dbf' SIZE
 10M AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED
 10     DEFAULT TEMPORARY TABLESPACE TEMP TEMPFILE '%ORACLE_BASE%/oradata/BDEU/u
ndoTBS1.dbf' SIZE 10M AUTOEXTEND ON NEXT 10M MAXSIZE UNLIMITED;

Database created.
```

Connexion avec la base de données BDEU :

```
SQL> startup nomount pfile='%ORACLE_BASE%/ADMIN/BDEU/PFILE/initBDEU.ora';
ORACLE instance started.
```

Création du schéma d'objets :

```
SQL> CREATE OR REPLACE TYPE Etudiant_t;
  2  /

Type created.

SQL> CREATE OR REPLACE TYPE Ens_etudiants AS TABLE OF REF Etudiant_t;
  2  /

Type created.

SQL> CREATE OR REPLACE TYPE Universite_t AS OBJECT(Nom VARCHAR2(100),
  2  Ville VARCHAR2(60), annee NUMBER(4), etudiants Ens_etudiants);
  3  /

Type created.

SQL> CREATE TYPE Etudiant_t AS OBJECT(Num NUMBER,
  2  Nom VARCHAR2(60), Prenom VARCHAR2(60), Niveau VARCHAR2(20),
  3  Univ REF Universite_t);
  4  /

Type created.

SQL> CREATE TABLE Etudiant OF Etudiant_t(Primary key(Num));

Table created.

SQL> CREATE TABLE Universite OF Universite_t(Primary key(Nom, ville))
  2  NESTED TABLE etudiants STORE AS lesetudiants;

Table created.
```

Remplissage des tables :

```
SQL> INSERT INTO Etudiant VALUES(124566, 'Jiang', 'Wang', '2 master SIQ',
  2  (SELECT REF(U) FROM Universite U WHERE U.Nom='universite 8mai45'));

1 row created.

SQL> INSERT INTO Universite VALUES('universite 8mai 45', 'Guelma', 1986,
  2  Ens_etudiants((select ref(E) FROM Etudiant E where E.Num=124566)));

1 row created.

SQL> INSERT INTO Universite VALUES('abdelhamid mahri', 'constantine', 2011,
  2  Ens_etudiants((select ref(E) FROM Etudiant E where E.Num=57895)));

1 row created.

SQL> INSERT INTO Etudiant VALUES(57895, 'Aggoune', 'aicha', 'doctorat',
  2  (SELECT REF(U) FROM Universite U WHERE U.Nom='abdelhamid mahri'));

1 row created.
```

Solution de Série TD N°2

Exercice 01.

1. Le code RDF-XML:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:
rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 <rdf:Description rdf:ID="Ville">
 <nom rdf :datatype="&xsd :string"/>
 </rdf:Description>
<rdf:Description rdf:ID="Personne ">
 <Parent>
<rdf:Description rdf:Resource="#Personne">
 </Parent>
 <Enfant>
<rdf:Description rdf:Resource="#Personne">
 </Enfant>
 <nom rdf :datatype="&xsd :string"/>
 <Age rdf :datatype="&xsd :PostiveInteger"/>
 <Habite>
<rdf:Description rdf:Resource="#Ville">
 </Habite>
</rdf:Description>
 </rdf:RDF>
```

2. Le code RDFS

```
<rdf: RDF
xmlns: rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns: rdfs= "http://www.w3.org/2000/01/rdf-schema#" >
<rdfs: comment>  définition des classes </rdfs: comment>
<rdfs:Class rdf:ID="Ville"/>
<rdfs:Class rdf:ID="Personne"/>
<rdfs: comment>  définition des propriétés </rdfs: comment>
<rdf:Property rdf:ID="Nom">
<rdfs:domain rdf:resource="#Personne"/>
<rdfs:domain rdf:resource="#Ville"/>
<rdfs:range rdf:resource="&xsd :string"/>
</rdf:Property>
<rdf:Property rdf:ID="Age">
<rdfs:domain rdf:resource="Personne"/>
<rdfs:range rdf:resource="&xsd :PostiveInteger"/>
</rdf:Property>
<rdfs: comment>  définition des relations </rdfs: comment>
<rdf:Property rdf:ID="Parent">
<rdfs:domain rdf:resource="#Personne"/>
<rdfs:range rdf:resource='#Personne'/>
</rdf:Property>
<rdf:Property rdf:ID="Enfant">
<rdfs:domain rdf:resource="#Personne"/>
<rdfs:range rdf:resource="#Personne"/>
</rdf:Property>
<rdf:Property rdf:ID="Habite">
<rdfs:domain rdf:resource="#Personne"/>
<rdfs:range rdf:resource="#Ville"/>
</rdf:Property>
</rdf:RDF>
```

Exercice 02.

1. Document OWL

```
< owl:Class rdf:ID="Personne"/>
< owl:Class rdf:ID="Ville"/>
<!-- Propriétés d'objet -->
< owl:ObjectProperty rdf:ID="Habite">
<rdfs:domain rdf:resource="#Personne"/>
<rdfs:range rdf:resource="#Ville"/>
</owl:ObjectProperty>
< owl:ObjectProperty rdf:ID="Parent">
<rdfs:domain rdf:resource="#Personne"/>
<rdfs:range rdf:resource="# Personne"/>
</owl:ObjectProperty>
< owl:ObjectProperty rdf:ID="Enfant">
<owl : InverseOF rdf:resource="#Parent"/>
</owl:ObjectProperty>
<!-- Propriétés de type de donnée -->
<owl:DataTypeProperty rdf:ID="nom">
<rdfs:domain rdf:resource="#Personne"/>
<rdfs:domain rdf:resource="#Ville"/>
<rdfs:range rdf:resource="&xsd :string"/>
</owl: DataTypeProperty>
<owl:DataTypeProperty rdf:ID="Age">
<rdfs:domain rdf:resource="#Personne"/>
<rdfs:range
rdf:resource="&xsd :PositiveInteger"/>
</owl: DataTypeProperty>
```

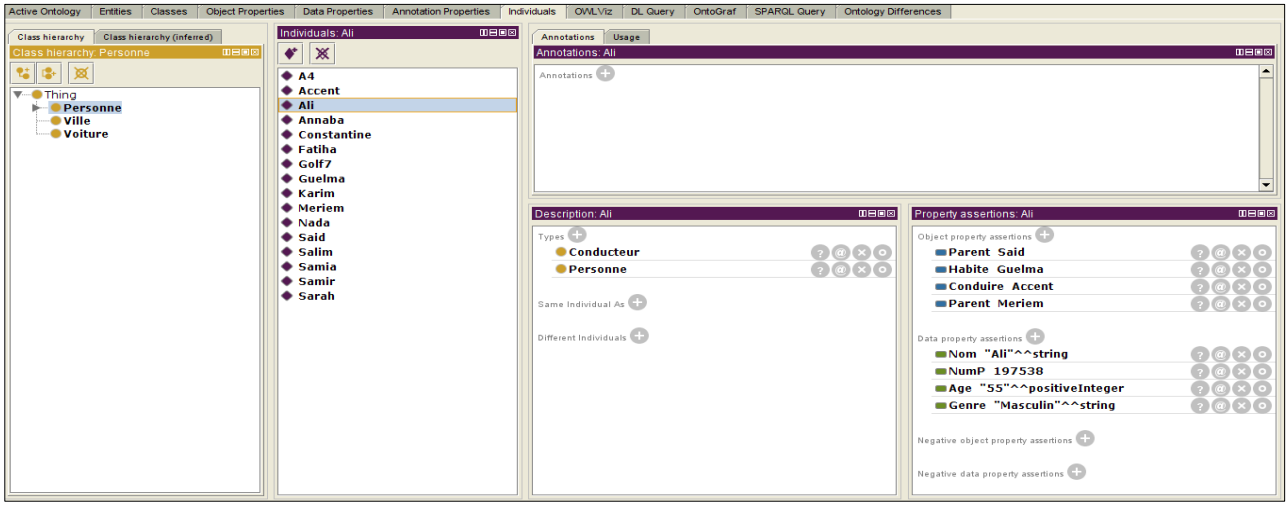
2. Les requêtes SPARQL

2.1
```
SELECT ?P ?Y
WHERE { ?P :Habite ?V.  ?V :Nom ?Y.
        Filter(?Y="Constantine").}
```

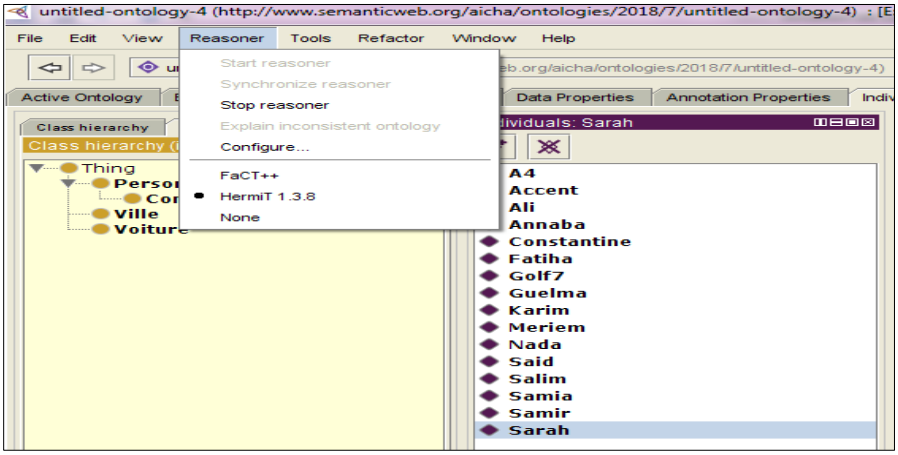
2.2
```
SELECT ?AgeK
WHERE {?A :Nom ?X . ?A :Age ? AgeK.
Filter(?X ="Said").}
```

Solution de Série TP N°2

1. Création d'ontologie OntoTP :



2. Vérification de consistance d'ontologie OntoTP via le raisonneur HermiT.



3. Les requêtes SPARQL

1.

```
SPARQL query:
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX OntoTP: <http://www.semanticweb.org/aicha/ontologies/2018/7/untitled-ontology-4#>
SELECT ?x ?y
            WHERE { ?x  OntoTP:Enfant  ?y}
```

2.

```
SPARQL query:
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX OntoTP: <http://www.semanticweb.org/aicha/ontologies/2018/7/untitled-ontology-4#>
SELECT ?x ?A
WHERE
 {
  ?x  OntoTP:Habite  ?A

 }
```

Solution de Série TD N°3

Exercice 01. Schéma ODMG, ODL

CLASS Personne (EXTENT Les personnes KEY NumP)
{ATTRIBUTE SHORT NumP ; ATTRIBUTE STRING NomP; ATTRIBUTE STRING Prénom ;
ATTRIBUTE STRUCT {SHORT N°Rue, STRING Rue, STRING Ville} Adresse
RELATIONSHIP Machine utilise INVERSE Machine :: Est_utilisée;}

CLASS Machine (EXTENT Lesmachines KEY Num)
{ATTRIBUTE SHORT Num; ATTRIBUTE STRING Nom;
RELATIONSHIP Segment est_reliée INVERSE Segment:: relie;
RELATIONSHIP SET(Installer) logiciels INVERSE Installer :: Mac;
RELATIONSHIP Personne Est_utilisée INVERSE Personne:: utilise; }

CLASS Installer (EXTENT Lesinstallations)
{ATTRIBUTE DATE Date_Installation ;
RELATIONSHIP Machine Mac INVERSE Machine :: logiciels ;
RELATIONSHIP Logiciel Log INVERSE Logiciel:: est_installé; }

CLASS Logiciel (EXTENT Leslogiciels KEY code)
{ATTRIBUTE SHORT code; ATTRIBUTE STRING NomL; ATTRIBUTE STRING Version;
RELATIONSHIP SET (Installer) est_installé INVERSE Installer ::Log; }

CLASS Segment (EXTENT LesSegments KEY ID_Réseau)
{ATTRIBUTE STRING ID_Réseau; ATTRIBUTE SHORT Longueur;
RELATIONSHIP LIST (Machine) relie INVERSE Machine::est_reliée; }

Exercice 02. Méthodes et OQL

1. Les méthodes :

1.1 METHOD BODY SHORT NbrSeg() IN CLASS Segment
{SHORT N ;
SELECT COUNT (S) INTO N
FROM S IN LesSegments;
RETURN N ;
}

1.2 METHOD BODY VOID Segment() IN CLASS Segment
{Segment S;
S. Longueur:=1;
}

2. Les requêtes OQL :

2.1 NAME OMAR=Personne ;
OMAR=Personne (NumP : 19, NomP :'Tahi', Prénom : 'Omar', Adresse : STRUCT(N° Rue :25, Rue : '19 Juin', Ville : 'Guelma'), Utilise : Machine(Num : 18, Nom : 'MacJeux', Logiciels : SET(Installer()), est_reliée : Segment())) ;

2.2 SELECT L.NomL FROM I IN Lesinstallations, L IN I.Log, M IN I.Mac
WHERE M.Num=5
==>Littéral BAG<STRING>

Solution de Série TD N°4

Exercice 01.

1. Programme DATALOG :

:-type (Fournisseur (Id : Integer, Nom : String, Prénom : String)).

Fournisseur (123, Fathi, Karim). Fournisseur (456, Ramdani, Said). Fournisseur (789, Dridi, Yacine). Fournisseur (101, Kara, Hamza).

:-type (Commande (Num: Integer, Date : Date, Total : double, Num-fournisseur : integer)).

Commande (0001, 01/09/2018, 40.000, 456). Commande (0013, 19/10/2018, 80.000, 101). Commande (0151, 22/08/2018, 35.000, 123). Commande (501, 30/05/2018, 97.000, 789).

:-type (Article (Code : Integer, Désignation : String, Prix : double, Commande : Integer)).

Article (485, bureau, 20.000, 0001). Article (361, imprimante, 30.000, 0013). Article (211, PC, 80.000, 501). Article (432, Onduleur, 10.000, 0151).

% prédicat intentionnel

Acceptation (x, y, z, t) :- Commande (x, y, z, t), Fournisseur (t, v, w).

2. Contraintes d'unicité:

INCORRECT_DB:- Fournisseur (Id1, Nom, Prénom), Fournisseur (Id2, Nom, Prénom), Id1\== Id2.

INCORRECT_DB:- Commande (Num1, Date, Total, Num-fournisseur), Commande (Num2, Date, Total, Num-fournisseur), Num1\== Num2.

INCORRECT_DB:- Article (Code1, Désignation, Prix, Commande), Article (Code2, Désignation, Prix, Commande), Code1\== Code2.

Exercice 02.

1. Définition du prédicat Chemin.

Chemin (S, D):- Arc (S, D).

Chemin (S, D):- Arc (S, A), Chemin (A, D).

2. Evaluation du prédicat récursif Chemin.

Etape 01 : table vide

Chemin 0	
S	D

Etape 2 : appliquer la règle 1.

Chemin 1	
S	D
1	2
1	3
2	4
4	5
3	6

Etape 3 : appliquer la règle 2.

Chemin 2	
S	D
1	4
1	6
2	5

Etape 4 : appliquer la règle 2.

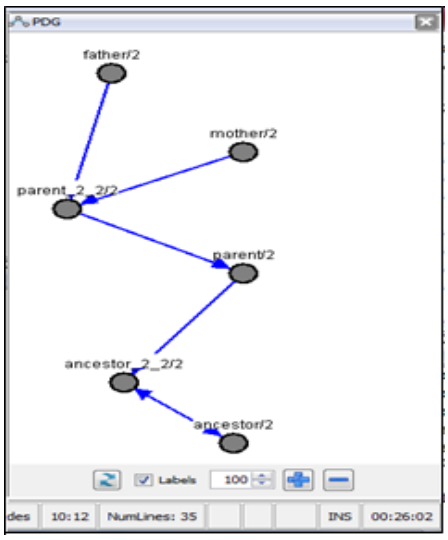
Chemin 3	
S	D
1	5

Le nombre d'extension du prédicat Chemin est : 09

Solution de Série TP N°3

1. Affichage de tables relationnelles, vues et les contraintes d'intégrités existantes.

2. Affichage de graphe de dépendance de la BD déductive



3.Ajouter des faits

```
Console
DES> insert into father values('peter', 'marck');

Info: 1 tuple inserted.

DES> insert into father values('marck', 'Bob');

Info: 1 tuple inserted.

DES> insert into mother values('jina', 'marck');

Info: 1 tuple inserted.

DES> insert into mother values('lila', 'bob');

Info: 1 tuple inserted.
DES>
```

4. Calcul des extensions de prédicats father et mother

```
Console
DES> ext(M):-count(father(S,_),S,M).

Info: Processing:
  ext(M) :-
    count(father(S,_),S,[],M).
{
  ext(7)
}
Info: 1 tuple computed.

DES> ext(M):-count(mother(S,_),S,M).

Info: Processing:
  ext(M) :-
    count(mother(S,_),S,[],M).
{
  ext(6)
}
Info: 1 tuple computed.
```

5. Les requêtes DATALOG :

5.1

```
DES> parent (X, marck).

{
  parent(jina,marck),
  parent(peter,marck)
}
Info: 2 tuples computed.
```

5.2

```
DES> famille(X,Y,Z):-father (X, Z), mother(Y, Z).

Info: Processing:
  famille(X,Y,Z) :-
    father(X,Z),
    mother(Y,Z).
{
  famille(fred,carolII,carolIII),
  famille(jack,amy,fred),
  famille(peter,jina,marck),
  famille(tom,grace,amy),
  famille(tony,carolI,carolII)
}
Info: 5 tuples computed.
```

Solution de Série TD N°5

Exercice 01.

1. Les fragments nécessaires :

Selon H1 : fragmentation horizontale :

Projet $_{Guelma}$=$\partial_{Luniversité= Guelma}$ (Projet).

Projet $_{Constantine}$ =$\partial_{Luniversité= Constantine}$ (Projet).

Projet $_{Annaba}$ =$\partial_{Luniversité= Annaba}$ (Projet).

Selon H2 : fragmentation verticale :

Chercheur= $\Pi_{(Enum, Nom, Prénom)}$ (Enseignant).

Fragmentation hybride :

Prof $_{Guelma}$ = $\Pi_{Enum, grade, université, Pcode}$ ($\partial_{Luniversité= Guelma}$ (Enseignant)).

Prof $_{Constantine}$ = $\Pi_{Enum, grade, université, Pcode}$ ($\partial_{Luniversité= Constantine}$ (Enseignant)).

Prof $_{Annaba}$ = $\Pi_{Enum, grade, université, Pcode}$ ($\partial_{Luniversité= Annaba}$ (Enseignant)).

2. Reconstruction du schéma global:

Projet= Projet $_{Guelma}$ ∪ Projet $_{Constantine}$ ∪ Projet $_{Annaba}$.

Enseignant= Chercheur ⋈ (Prof $_{Guelma}$ ∪ Prof $_{Constantine}$ ∪ Prof $_{Annaba}$)

3. La vue matérialisée:

```
CREATE MATERIALIZED VIEW Ens-projet
REFRESH COMPLETE START WITH Sysdate NEXT Sysdate + 3
Enable QUERY REWRITE
AS
SELECT P.titre, P. Description, C.Nom, C. NOM, C.Prénom
FROM Projet@site1 P, Chercheur@site1 C
GROUP BY P.titre;
```

Exercice 02.

a. Les vues réparties :

CREATE VIEW V1 (Pcode, Titre, description, PUniversité, année) AS SELECT * FROM Projet $_{Guelma}$ UNION SELECT * FROM Projet $_{Constantine}$@site2 UNION SELECT * FROM Projet $_{Annaba}$@site3;

La requête répartie est: SELECT * FROM V1 WHERE Année=2018;

b. Le connecteur RDA:

Sur le site 2 : SELECT * INTO temp1 FROM Projet $_{Constantine}$;

Sur le site 3 : SELECT * INTO temp2 FROM Projet $_{Annaba}$;

Sur le site 1: la requête repartie: SELECT P1.*, P2.*, P3.* FROM Projet $_{Guelma}$ P1, temp1 P2, temp2 P3;

Exercice 03. Gestion des transactions concurrentes

T1	T2	A	B
R1(a1←A)		4	
a1← a1-2		2	
	R2(a2←A)	4	
	A2← a2+3	7	
W1(A← a1)		2	
R1(b1←B)			8
	W2(A← a2)	7	
b1← b1+2			10
W1(B← b1)			10

Le résultat obtenu est : A=7 et B=10 et le résultat correct est A=5 et B=10

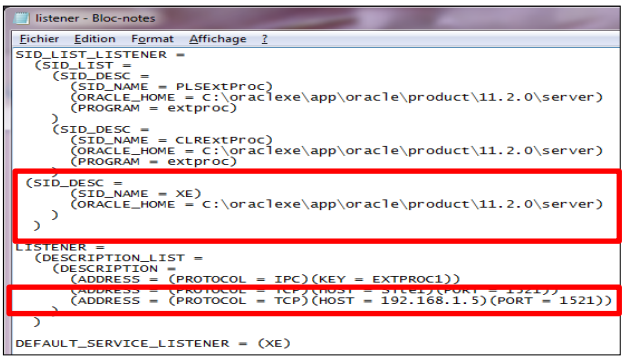
2.A)

T1 :A=4
T2 :B=8
T1 :A-2=2
T2 :B+1=9
T1 :B+2=10
T2 :A=2
T1 :B=10

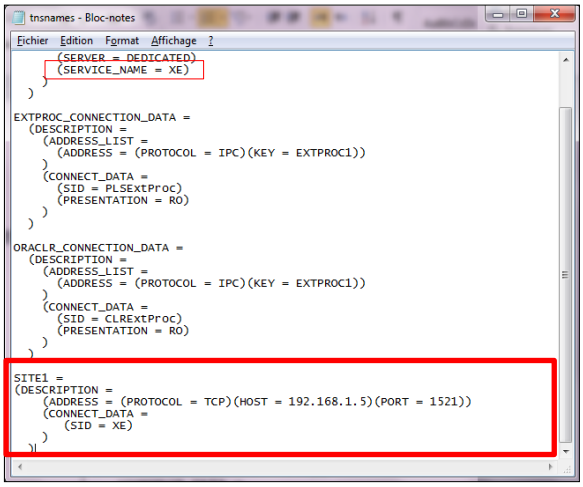
2.B) si on applique T1;T2 : on obtient : T1 : A=4, A=2, B=10, T2 :B=10, B=11, A=2 Donc O n'est pas sérialisable.

Solution de Série TP N°4

1. Configuration de Listener d'ORACLE :



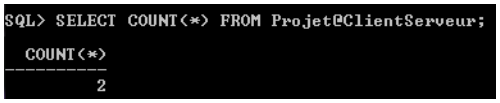
2. Configuration du fichier tnsnames du Serveur



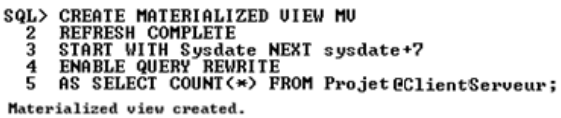
3. Création de lien DATABASE LINK ServeurClient et ClientServeur.

4. Création des tables et des tuples :

4. Requête répartie :



5. La vue matérialisée



Solution de Série TD N°6

Exercice 01. Manipulation de données LOB

1. La création du schéma relationnel.

```
SQL> CREATE TYPE Plante_t AS OBJECT(CU INTEGER, Nom VARCHAR2(60),
  2  Type VARCHAR2(50), Photo BFILE);
SQL> CREATE TABLE Plante OF Plante_t (PRIMARY KEY(CU), CHECK (Type IN ('Fleur',
  2  'Plante interne', 'Plante externe')));
Table created.
```

2. Création du répertoire RMM.

```
SQL> CREATE OR REPLACE DIRECTORY RMM AS 'C:\botanique';
Directory created.
SQL> GRANT ALL ON DIRECTORY RMM TO PUBLIC;
Grant succeeded.
```

3. Insertion de l'attribut Image de type BFILE.

```
SQL> INSERT INTO Botanique VALUES(113, 'Aloevera', 'PlanteINT', BFILENAME('RMM',
  2  'AleoVera.jpg'));
1 row created.
```

4. Calculer la similarité entre deux images B1 et B2 :

Distance (Q, B1)= $\sqrt{\sum_{i=1}^{4}(qi - bi)^2}$ = $\sqrt{(2-2.6)^2 + (0-0)^2 + (3-3)^2 + (5-5)^2}$ = $\sqrt{0.16} = 0.4$

Distance (Q, B2)= $\sqrt{(2-2)^2 + (0-0)^2 + (3-2.75)^2 + (5-5)^2} = \sqrt{0.625} = 0.25$

Alors : Similarité (Q, B1)= 1-0.4 = 0.6 et Similarité (Q, B2)= 1-0.25 = 0.75

L'image B2 est la plus similaire à Q.

Exercice 02. Manipulation de Base de données NoSQL

1. Création de BDP

```
> use BDP
switched to db BDP
> db.pièces.insertMany([{"_id":1, "Name": "PX", "Sizee":{"W":25, "H":40},"Quanti
ty": 40},{"_id": 2, "Name": "PY"}])
{ "acknowledged" : true, "insertedIds" : [ 1, 2 ] }
> db.pièces.find()
{ "_id" : 1, "Name" : "PX", "Sizee" : { "W" : 25, "H" : 40 }, "Quantity" : 40 }
{ "_id" : 2, "Name" : "PY" }
```

2. Les requêtes :

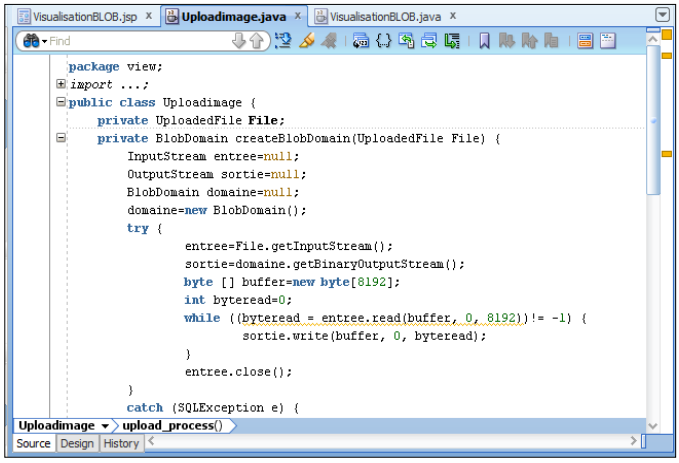
2.1. `> db.pièces.find({"Quantity": {$gt:1000}})`

2.2.

```
> db.pièces.find({},{"Name":1, "_id":0}, {"Sizee":{"W":{"$in":[15, 25]}}, "H":50
}})
{ "Name" : "PX" }
{ "Name" : "PY" }
```

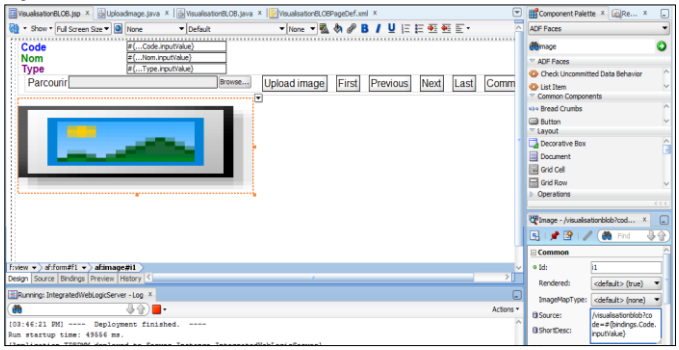
Solution de Série TP N°5

Ajout de code Upload :

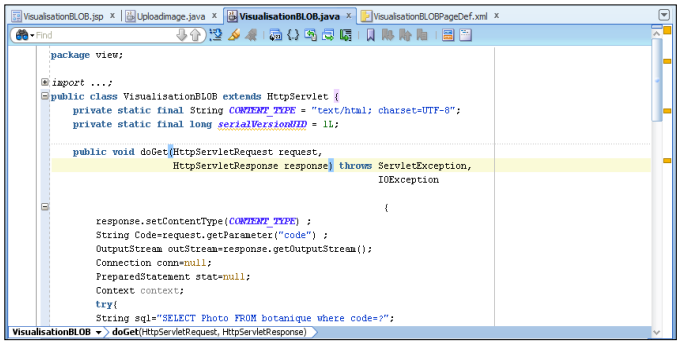


```
package view;
import ...;
public class Uploadimage {
    private UploadedFile File;
    private BlobDomain createBlobDomain(UploadedFile File) {
        InputStream entree=null;
        OutputStream sortie=null;
        BlobDomain domaine=null;
        domaine=new BlobDomain();
        try {
                entree=File.getInputStream();
                sortie=domaine.getBinaryOutputStream();
                byte [] buffer=new byte[8192];
                int byteread=0;
                while ((byteread = entree.read(buffer, 0, 8192))!= -1) {
                        sortie.write(buffer, 0, byteread);
                }
                entree.close();
        }
        catch (SQLException e) {
```

Ajout de Servlet VisualisationBLOB

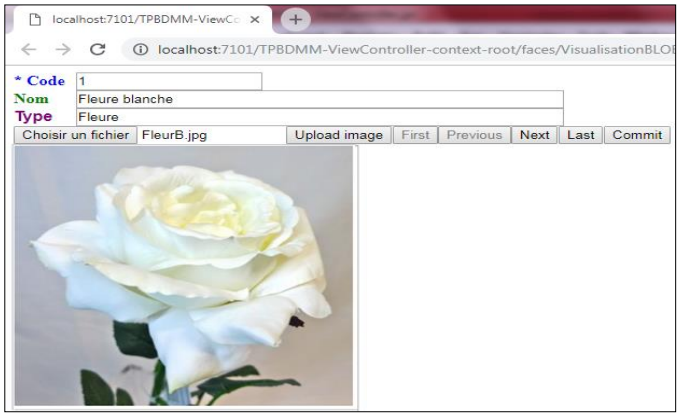


Code java :



```
package view;
import ...;
public class VisualisationBLOB extends HttpServlet {
    private static final String CONTENT_TYPE = "text/html; charset=UTF-8";
    private static final long serialVersionUID = 1L;

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException,
                                                          IOException
                                                          {
        response.setContentType(CONTENT_TYPE) ;
        String Code=request.getParameter("code") ;
        OutputStream outStream=response.getOutputStream();
        Connection conn=null;
        PreparedStatement stat=null;
        Context context;
        try{
        String sql="SELECT Photo FROM botanique where code=?";
```

Exécution de page JSP



Références Bibliographiques

1. Aggoune, A., A. Bouramoul, and M.K. Kholladi, *Mediation system for dealing with semantic problems in databases.* International Journal of Data Mining, Modelling and Management, 2017. **9**(2): p. 99-121.
2. Dehainsala, H., *Explicitation de la sémantique dans les bases de données: Base de données á base ontologique et le modéle OntoDB*. 2007, Université de Poitiers.
3. Oracle. 25/07/2018]; Available from: https://docs.oracle.com/cd/B28359_01/appdev.111/b28370/create_type.htm#LNPLS01375.
4. Gardarin, G., *Bases de données*. 2003: Editions Eyrolles.
5. Codd, E.F., *A relational model of data for large shared data banks.* Communications of the ACM, 1970. **13**(6): p. 377-387.
6. Connolly, T.M. and C.E. Begg, *Database systems: a practical approach to design, implementation, and management*. 2005: Pearson Education.
7. Elmasri, R. and S. Navathe, *Fundamentals of database systems*. 2010: Addison-Wesley Publishing Company.
8. Nijssen, G.M. and T.A. Halpin, *Conceptual Schema and Relational Database Design: a fact oriented approach*. 1989: Prentice Hall Sydney.
9. Delmal, P., *SQL2-SQL3: applications à Oracle*. 2000: De Boeck Supérieur.
10. Gardarin, G. and P. Valduriez, *SGBD avancés: bases de données objets, déductives, réparties*. 1990: Eyrolles.
11. Loney, K., *Oracle Database 11g The Complete Reference*. 2008: McGraw-Hill, Inc.
12. AGGOUNE, A., *Traitement de L'hétérogénéité Sémantique pour L'exploration des Sources des Donnés Multimédias*. 2017, Abdelhamid Mahri Constantine 2 -Algérie -.
13. Kim, W. and J. Seo, *Classifying schematic and data heterogeneity in multidatabase systems.* Computer, 1991. **24**(12): p. 12-18.
14. Berners-Lee, T., J. Hendler, and O. Lassila, *The semantic web.* Scientific american, 2001. **284**(5): p. 34-43.
15. Woods, W.A., *WHAT'S IN A LINK: Foundations for Semantic Networks*, in *Representation and Understanding*, D.G. Bobrow and A. Collins, Editors. 1975, Morgan Kaufmann: San Diego. p. 35-82.
16. Sowa, J.F., *Conceptual structures: information processing in mind and machine.* 1984.
17. Roberts, N., *The pre-history of the information retrieval thesaurus.* Journal of documentation, 1984. **40**(4): p. 271-285.
18. Caussanel, J., et al., *Les Topic Maps sont-ils un bon candidat pour l'ingénierie du Web Sémantique?* Actes des 13e journées francophones d'ingénierie des connaissances (IC). Prix AFIA de la meilleure présentation, 2002.
19. Welty, C. and N. Guarino, *Supporting ontological analysis of taxonomic relationships.* Data & Knowledge Engineering, 2001. **39**(1): p. 51-74.
20. Studer, R., V.R. Benjamins, and D. Fensel, *Knowledge engineering: principles and methods.* Data and knowledge engineering, 1998. **25**(1): p. 161-198.
21. Guarino, N., D. Oberle, and S. Staab, *What is an ontology?*, in *Handbook on ontologies*. 2009, Springer. p. 1-17.
22. Teitsma, M., et al., *Engineering ontologies for question answering.* Applied Ontology, 2014. **9**(1): p. 1-25.
23. Dehainsala, H. *Base de données à base ontologique*. in *Proc. du 23ème congrès Inforsid*. 2004.
24. Pierra, G., et al. *Base de données à base ontologique: le modèle OntoDB*. in *BDA*. 2004.

25. Roldan-Garcia, M., I. Navas-Delgado, and J.F. Aldana-Montes. *A design methodology for semantic web database-based systems*. in *Information Technology and Applications, 2005. ICITA 2005. Third International Conference on*. 2005. IEEE.
26. Sugumaran, V. and V.C. Storey, *The role of domain ontologies in database design: An ontology management and conceptual modeling environment.* ACM Transactions on Database Systems (TODS), 2006. **31**(3): p. 1064-1094.
27. Lassila, O. and R.R. Swick, *Resource description framework (RDF) model and syntax specification.* 1998.
28. McBride, B., *The resource description framework (RDF) and its vocabulary description language RDFS*, in *Handbook on ontologies*. 2004, Springer. p. 51-65.
29. McGuinness, D.L. and F. Van Harmelen, *OWL web ontology language overview.* W3C recommendation, 2004. **10**(10): p. 2004.
30. Pérez, J., M. Arenas, and C. Gutierrez. *Semantics and Complexity of SPARQL*. in *International semantic web conference*. 2006. Springer.
31. Khoshafian, S. and R. Abnous, *Object orientation: concept, analysis and design, languages, databases, graphical user interfaces, standards*. 1995: John Wiley & Sons, Inc.
32. Amiel, E., et al., *Etude de la persistance dans les SGBDOO*. 1992, INRIA.
33. Adiba, M. and C. Collet, *Objets et bases de données: le SGBD O2*. 1993, Editions Hermes.
34. Zdonik, S.B. and D. Maier, *Readings in object-oriented database systems*. 1990: Morgan Kaufmann.
35. Rumbaugh, J., et al., *Object-oriented modeling and design*. Vol. 199. 1991: Prentice-hall Englewood Cliffs, NJ.
36. ODMG. 16/06/2018]; Available from: http://www.odbms.org/odmg-standard/.
37. Jordan, D., *C++ Object Databases: Programming with the ODMG standard*. Vol. 456. 1998: Addison-Wesley.
38. Cattell, R.G.G., et al., *The object database standard: ODMG 2.0*. Vol. 131. 1997: Morgan Kaufmann Publishers Los Altos, CA.
39. Gallaire, H., J. Minker, and J.M. Nicolas, *An overview and introduction to logic and data bases*, in *Logic and Data Bases*. 1978, Springer. p. 3-30.
40. Cori, R. and D. Lascar, *Logique mathématique: cours et exercices. Calcul propositionnel, algèbres de Boole, calcul des prédicats*. 1993: Masson.
41. Gochet, P. and P. Gribomont, *" Logique", méthodes pour l'informatique fondamentale.* Revue de Métaphysique et de Morale, 1992. **97**(4): p. 573.
42. Horn, A., *On sentences which are true of direct unions of algebras.* The Journal of Symbolic Logic, 1951. **16**(1): p. 14-21.
43. Gallaire, H., J. Minker, and J.-M. Nicolas, *Logic and databases: A deductive approach*, in *Readings in Artificial Intelligence and Databases*. 1988, Elsevier. p. 231-247.
44. HACID, M.-S. and J. KOULOUMDJIAN, *Bases de données déductives.* Techniques de l'ingénieur. Informatique, 1997(H2048): p. H2048. 1-H2048. 11.
45. Nicolas, J.-M. and K. Yazdanian, *Integrity checking in deductive data bases*, in *Logic and data bases*. 1978, Springer. p. 325-344.
46. Sagiv, Y., *Optimizing datalog programs, Foundations of deductive databases and logic programming*. 1988, Morgan Kaufmann Publishers Inc., San Francisco, CA.
47. Gottlob, G., E. Grädel, and H. Veith, *Datalog LITE: A deductive query language with linear time model checking.* ACM Transactions on Computational Logic (TOCL), 2002. **3**(1): p. 42-79.
48. Rolston, D.W., *Principles of artificial intelligence and expert systems development.* 1988.

49. Bernstein, P.A., et al., *Query processing in a system for distributed databases (SDD-1).* ACM Transactions on Database Systems (TODS), 1981. **6**(4): p. 602-625.
50. Ceri, S., *Distributed databases*. 2017: Tata McGraw-Hill Education.
51. Özsu, M.T. and P. Valduriez, *Principles of distributed database systems*. 2011: Springer Science & Business Media.
52. CABANAC, G., et al., *Bases de données réparties.* 2014(15/08/2018).
53. Gabillaud, J., *Oracle 11g: SQL, PL/SQL, SQL* Plus*. 2009: Editions ENI.
54. Bulterman, D.C.A., *SMIL 2.0. 2. Examples and comparisons.* IEEE MultiMedia, 2002. **9**(1): p. 74-84.
55. Nwosu, K.C., B. Thuraisingham, and P.B. Berra, *Multimedia Database Systems: design and implementation strategies*. 2012: Springer Science & Business Media.
56. Sadalage, P.J. and M. Fowler, *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. 2013: Pearson Education.
57. AGGOUNE, A. and M.S. NAMOUNE. *A Method for Transforming Object-relational to Document-oriented Databases*. in *2020 2nd International Conference on Mathematics and Information Technology (ICMIT)*. 2020. IEEE.