République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Abbess Laghrour Khenchela
Faculté Des Sciences Et De La Technologie
Département De Mathématiques Et Informatique

Cours de bases de données avancées

Dr. Ouassila HIOUAL

Novembre 2015

Table des matières

Introduction 01

Chapitre 1 : Rappel sur les bases de données

Partie 1. Introduction aux bases de données (BD) 02

1.1. Qu'est-ce qu'une donnée ? 02

1.2. Qu'est –ce donc une BDD ? 02

1.3. Qu'est-ce qu'un Système de Gestion de Base de Données (SGBD) ? 02

1.4. Fonctions d'un SGBD 02

1.5. Objectifs des systèmes de gestion de bases de données 02

1.6. Types d'utilisateurs de base de données 03

1.7. Cycle de vie d'une base de données 04

1.8. Evolution des bases de données 05

Partie 2. Modèle Entité/Association 06

2.1. Entité et type-entité 06

2.2. Attribut, propriété 06

2.3. Association, relation 07

2.4. Identifiant 07

2.5. Cardinalité 08

2.6. Dimension 09

Partie 3. Modèle relationnel 09

3.1. Notion de domaine 09

3.2. Attribut 09

3.3. Relation (table) 09

3.4. Clé d'une relation 10

3.5. Population 10

3.6. Règle de passage du Modèle E/A au modèle relationnel 10

Série d'exercices sur les bases de données 12

Chapitre 2 : Les bases de données orientées objets

1. Limites des SGBDs relationnels et nouveaux besoins 14
2. Structure de données 16
   2.1. Structure complexe 16
   2.2. Identité d'objet 17
   2.3 Liens de composition 19
   2.4. Graphe de généralisation/spécialisation des classes 21
   2.5. Populations et persistance 25
3. La dynamique 27
   3.1. Les méthodes 27
   3.2. Polymorphisme et liaison dynamique 29
4. Interface navigationnelle de manipulation des données 30

Série d'exercices sur les bases de données orientées objets 33

Chapitre 3 : Le relationnel étendu

1. Introduction 35
   1.1 Points forts du modèle relationnel 35
   1.2 Points faibles du modèle relationnel 35
2. Le Relationnel étendu (Relationnel Objet (RO)) 36
3. Types 37
   3.1. Les types VARRAY 37
   3.2. Les types table emboîtée ("nested TABLE") 37
   3.3. Les types OBJECT 38
4. Tables 39
   4.1. Création d'une table relationnelle classique 39
   4.2. Création d'une table relationnelle en non première forme normale 39
   4.3. Création d'une table d'objets 39
5. Identité et attributs-référence 39
   5.1. REF (objet) 40
   5.2. VALUE (objet) 40
   5.3. DEREF (oid) 40
6. Méthodes 40

6.1. Signature d'une méthode 41
6.2. Corps d'un type OBJECT 41
7. Hiérarchie de types OBJECT 41
7.1. Nouvelle condition élémentaire 42
Série d'exercices sur le relationnel étendu 43

Chapitre 4 : Les bases de données déductives

1. Concept et Motivation 47
2. Définitions préliminaires 47
3. Exemple 47
4. Bases de données déductives s'appuyant sur des SGBDs relationnels 47
4.1. Bases de données déductives et langages du 1er ordre 48
4.1.1 Langages du 1er ordre 48
4.1.2 Interprétation d'un ensemble de clauses 49
4.1.3 Base de données relationnelle comme modèle d'un ensemble de clauses 51
4.1.4 Calcul du 1er ordre 51
4.1.5 Bases de données déductives utilisant le calcul du 1er ordre 52
4.2 Bases de données déductives et règles de production 52
4.3 Stratégies de construction 53
4.3.1 Stratégie par couplage 53
4.3.2 Stratégie par intégration de systèmes 54
4.4. Exemples de SGBD déductives relationnels 54
Série d'exercice sur les bases de données déductives 56

Chapitre 5 : Les bases de données réparties

1. Les bases de données réparties: émergence, avantages, définition et problèmes 58
1.1. Emergence de la répartition des données 58
1.2. Avantages de la répartition des bases de données 58
1.3. Définition d'une base de données répartie 59
1.4. Objectifs visés lors de la conception d'une BD répartie 59
1.5. Problèmes liés à la répartition des données 59
2. Différentes architectures 59
2.1. Architecture Client/ Serveur 59

2.2. Architecture Pair à Pair 60
3. Conception d'une BD répartie 60
3.1. Conception descendante (Top Down) 61
3.2. Conception ascendante (Bottom up) 61
4. Fragmentation 62
4.1. Règles de fragmentation 62
4.2. Techniques de fragmentation 62
4.2.1. Fragmentation horizontale (répartition des occurrences ou tuples) 62
4.2.2. Fragmentation verticale (répartition des attributs) 64
4.2.3. Fragmentation hybride ou mixte (répartition des valeurs) 65
5. Allocation des fragments 65
5.1. Fragmentation avec réplication 66
5.1.1. Techniques de réplication 66
5.2. Fragmentation sans réplication 67
6. Fragmentation et transparence 67
7. Traitement & Optimisation de Requêtes Réparties 68
7.1. Mise à jour des BD réparties 68
7.2. Requêtes sur les BDs réparties 69
7.2.1. Transferts de données 69
7.2.2. Traitement de requêtes réparties 70
7.2.3. Optimisation dynamique des requêtes 70
7.2.4. Semi-jointure 70
Série d'exercice sur les bases de données réparties 72
ANNEXE : Solutions des exercices 77

Introduction

Ce cours s'adresse aux étudiants du cycle 2 (Master en Informatique). Il a l'objectif ambitieux de permettre aux étudiants la découverte des différents aspects et techniques liés aux nouvelles tendances dans les bases de données. L'étudiant est censé avoir acquis comme compétences après le succès à cette matière :

- Savoir rendre persistants les objets manipulés par les langages objets.
- Connaître les diverses possibilités, leurs avantages, leurs inconvénients.
- Avoir un aperçu de différents types de bases de données avancées.
- Connaître les objectifs et les principes de base des systèmes relationnel-objet, être en mesure de les appliquer sur une application type (système d'information géographique, par exemple)
- Connaître les principes de base des bases de données distribuées / réparties,
- Ouverture vers d'autres types de données : objet, réparties, déductives, ….
- …

Quelques références bibliographiques :

- Bases de données objet & relationnel. Paris: Eyrolles. Gardarin, G. (1999).
- Omran A. Bukhres, Ahmed K. Elmagarmid : Object Oriented Multidatabase Systems : A solution for advanced applications Prentice Hall 1996
- Bases de données orientées-objet : Concepts, mise en œuvre et exercices résolus. Lavoisier. Chrisment. C et al. (2011)
- Bases de données oreintées objets. Vuibert. Catell. R.G.G. (1997)
- Cours bases de données réparties: http://ceria.dauphine.fr/Rim/SupportBDR.pdf. Dernière date de consultation: 01/11/2015.
- Le Client-Serveur. Gardarin G. « Ed. Eyrolles, ISBN 2-212-08876-0, 1996
- Client-Serveur : Concepts, moteurs, SQL et architectures parallèles. Miranda & Ruols, Ed. Eyrolles, ISBN 2-212-08816-7, 1994
- Principles of Distributed DB Systems. T. Ozsu, P. Valduriez (Prentice Hall)
- Réplication et Durabilité dans les systèmes répartis. Lambert SONNA MOMO**,** « Edition, Février 2001 ».

Chapitre 1 : Rappel sur les bases de données

Partie 1. Introduction aux bases de données (BD)

Les bases de données (BD) sont actuellement au cœur du système d'information des entreprises et au site web, elles peuvent être vues comme un réservoir de fichier de données informatiques.

1.1. Qu’est-ce qu’une donnée ?

C’est une information quelconque comme, par exemple : voici une personne, elle s’appelle X. C’est aussi une relation entre des informations comme, par exemple : X est un étudiant.

1.2. Qu’est –ce donc une BDD ?

Définition informelle

On peut considérer une BD comme une grande quantité de données centralisées ou non, permettant de répondes aux besoins d'une ou plusieurs applications, interrogeables et modifiables par un groupe d'utilisateurs.

Définition formelle

Une base de données est un ensemble d'informations sur un domaine qui est : exhaustif, non redondant, structuré, persistant.

1.3. Qu’est-ce qu’un Système de Gestion de Base de Données (SGBD) ?

Un système de gestion de base de données est un logiciel qui permet de : décrire, modifier, interroger et administrer les données d'une base de données.

1.4.Fonctions d’un SGBD :

- Décrire les données qui seront stockées
- Manipuler ces données (ajouter, modifier, supprimer des informations)
- Consulter les données et traiter les informations obtenues (sélectionner, trier, calculer,...)
- Définir des contraintes d'intégrité sur les données (contraintes de domaines, d'existence,...)
- Définir des protections d'accès (mots de passe, autorisations,...)
- Résoudre les problèmes d'accès multiples aux données (blocages, interblocages)
- Prévoir des procédures de reprise en cas d'incident (sauvegardes, journaux,...)

1.5. Objectifs des systèmes de gestion de bases de données

Des objectifs principaux ont été fixés aux systèmes de gestion de bases de données dès l'origine de ceux-ci et ce, afin de résoudre les problèmes causés par la démarche classique. Ces objectifs sont les suivants :

Indépendance par rapport :

- aux traitements : Pour faciliter la maintenance, un SGBD doit favoriser l'indépendance des traitements

– à l'implantation physique des données :(codage, support d'enregistrement, ordre dans lequel les données sont enregistrées,…)

– à l'implantation logique des données (existence d'index, décomposition en "fichiers logiques",…)

Manipulations des données par des non informaticiens

Il faut pouvoir accéder aux données sans savoir programmer, ce qui signifie des langages "quasi naturels".

Efficacité des accès aux données

Ces langages doivent permettre d'obtenir des réponses aux interrogations en un temps "raisonnable". Ils doivent donc être optimisés et, entre autres, il faut un mécanisme permettant de minimiser le nombre d'accès disques. Tout ceci, bien sur, de façon complètement transparente pour l'utilisateur.

Administration centralisée des données

Des visions différentes des données (entre autres) se résolvent plus facilement si les données sont administrées de façon centralisée.

Non redondance des données

Afin d'éviter les problèmes lors des mises à jour, chaque donnée ne doit être présente qu'une seule fois dans la base.

Cohérence des données

Les données sont soumises à un certain nombre de contraintes d'intégrité qui définissent un état cohérent.

1.6. Types d'utilisateurs de base de données

- **L'administrateur** de la base est chargé de (s):
  - contrôler la base de données, en particulier, permettre l'accès aux données aux applications ou individus qui y ont droit.

- conserver de bonnes performances d'accès à ces données.
- sauvegardes et des procédures de reprise après les pannes.

- **Le programmeur**
  - écrit des applications qui utilisent la base de données.
  - crée les tables et les structures associées (vues, index,...) utilisées par ses applications.
- **L'utilisateur final**
  - n'a accès qu'aux données qui lui sont utiles par l'intermédiaire d'applications en interrogeant directement les tables ou vues sur lesquelles l'administrateur lui a accordé des droits.

1.7. Cycle de vie d'une base de données

1.7.1. Conception

On appelle conception d'une base de données la phase d'analyse qui aboutit à déterminer le futur contenu de la base.

Lorsqu'une entreprise décide, pour son informatisation, d'adopter une approche base de données, le premier problème à résoudre, peut-être le plus difficile, est de déterminer les informations qu'il conviendra de mettre dans la base de données.

1.7.2. Implémentation

Ceci sera fait au moyen d'un langage symbolique, spécifique du SGBD choisi, que l'on appelle langage de description de données (LDD). Une fois que le SGBD aura pris connaissance de cette description, il sera possible aux utilisateurs de faire entrer les données, c'est-à-dire de constituer la première version de la base de données.

1.7.3. Manipulation

Une fois l'implantation terminée, on peut commencer l'utilisation de la base de données. Celle-ci se fait au moyen d'un langage, dit langage de manipulation de données (LMD), qui permet d'exprimer aussi bien les requêtes d'interrogation (pour obtenir des informations contenues dans la base) que des requêtes de mise à jour (pour ajouter de nouvelles informations, supprimer des informations périmées, modifier le contenu des informations).

1.8. Evolution des bases de données

- **50 - 60** Organisation classique en fichiers et méthodes d'accès (séquentiel, direct, séquentiel indexé).
- **62 - 63** Apparition du concept de Base de Données.
- **65 - 70** Conception des SGBD de 1ère Génération (modèles hiérarchique et réseau) :
  - IMS d'IBM (hiérarchique),
  - IDS de General Electric (réseau) qui a servi de modèle de base aux propositions du groupe CODASYL.
- **70 - 85** 2ème Génération des SGBD organisés sur le modèle relationnel. Donc, on a plus de spécification des moyens d'accès aux données. Exemples de systèmes commercialisés dans les années 1980 :
  - MRDS de Honeywell diffusé par CII-HB,
  - QBE (Query By Example),
  - SQL/IDS d'IBM,
  - INGRES de Relational Technology,
  - ORACLE de Relational Software.

Aujourd'hui : Les données sont plus variées (textes, sons, images, parole, ..),

- Bases de Données réparties,
- Bases de Données orientées objets,
- Bases de Connaissances et Systèmes Experts,
- Bases de données déductives,
- Accès intelligent multimodal et naturel (langage naturel écrit, graphique, parole, etc.).

Partie 2. Modèle Entité/Association

Le modèle *Entité-Association* (notation : EA) est aussi fréquemment nommé *Entité-Relation*. Le modèle EA propose des concepts (principalement les entités, les associations et les attributs) permettant de décrire un ensemble de données relatives à un domaine défini afin de les intégrer ensuite dans une BD.

- Le modèle *relationnel* est pauvre en capacité de représentation sémantique. Cependant, le concepteur de la BD a besoin de certaines informations sémantiques que le modèle EA lui permet de décrire sa base.

- Il est difficile de modéliser un domaine sous une forme directement manipulable par un SGBD. Une ou plusieurs modélisations intermédiaires sont donc utiles, le modèle EA constitue l'une des premières et des plus courantes modélisations.

2.1. Entité et type-entité

Définition 1 : Une **entité** est un objet, une chose concrète ou abstraite qui peut être reconnue distinctement.

Définition 2 : Un **type-entité** est un ensemble d'entités qui possèdent les mêmes caractéristiques.

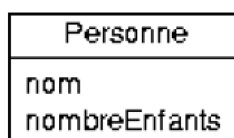
Attention ! Par abus de langage, on utilise souvent le mot entité à la place du mot type-entité, il faut cependant prendre garde à ne pas confondre les deux concepts.

- Exemples de type-entité : Humain
- Exemples d'entités : Femme et Homme
- Exemples d'occurrences ou instances : Fatima et Ali

2.2. Attribut, propriété

Définition : Un **attribut** (ou une **propriété**) est une caractéristique associée à un type entité. Au niveau du type-entité, chaque attribut possède un domaine qui définit l'ensemble des valeurs possibles qui peuvent être choisies pour lui (entier, chaîne de caractères, booléen…).

- Exemples d'attribut : l'âge d'une personne, le code d'un fournisseur, le numéro d'un produit…
- Exemple de représentation graphique d'un type-entité avec deux attributs :



2.3. Association, relation

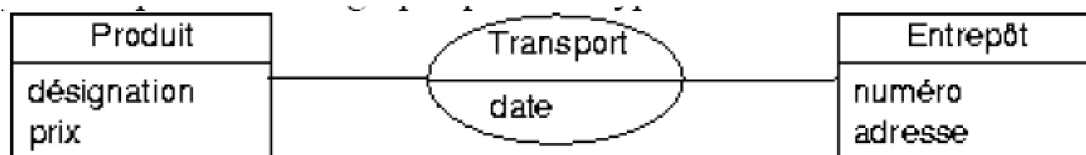
Définition 1 : Une **association** (ou une **relation**) est un lien entre plusieurs entités.

Définition 2: Un **type-association** (ou un **type-relation**) est un ensemble de relations qui possèdent les mêmes caractéristiques.

Le type-association décrit un lien entre plusieurs types-entités. Les associations de ce type-association lient des entités de ces types-entités.

Attention ! Par abus de langage, on utilise souvent le mot association à la place du mot type-association, il faut cependant prendre garde à ne pas confondre les deux concepts.

- Exemples de type-association : le transport d'un produit vers un entrepôt, l'affectation d'un employé à un service…
- Exemples d'association : le transport de la Clio 3333 XR 06 vers le dépôt de xxxx, le fait que Ali travaille au service Marketing…
- Exemple de représentation graphique d'un type-association :

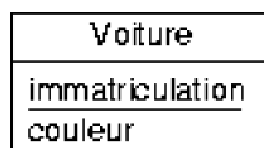


Un type-association peut ne pas posséder d'attribut et cela est relativement fréquent.

2.4. Identifiant

Définition : Un **identifiant** d'un type-entité ou d'un type-association est constitué par un ou plusieurs de ses attributs qui doivent avoir une valeur unique pour chaque entité ou association de ce type.

- Exemples d'identifiant : le numéro d'immatriculation d'une voiture, le code-barre d'un produit…
- Exemple de représentation graphique d'un identifiant (immatriculation) :



o On parle parfois de *clé* plutôt que d'identifiant.

o Chaque type-entité possède au moins un identifiant, éventuellement formé de plusieurs attributs. Chaque type-entité possède au moins un attribut qui, s'il est seul, est donc forcément l'identifiant.

o L'identifiant, qu'il soit explicite ou non, d'un type-association doit être la concaténation des identifiants des types-entités liés.

2.5. Cardinalité

Définition : La **cardinalité** d'un type-association est le nombre de fois minimal et maximal qu'une entité peut intervenir dans une association de ce type. La cardinalité minimale doit être inférieure ou égale à la cardinalité maximale.

o Exemple de cardinalité : un client peut commander entre 1 et n produits.

o Exemple de représentation graphique de cardinalité :

L'expression de la cardinalité est obligatoire pour chaque patte d'un type-association. La cardinalité minimale peut-être :

- 0 Cela signifie qu'une entité *peut exister* tout en étant impliquée dans *aucune association*.
- 1 Cela signifie qu'une entité *ne peut exister que* si elle est impliquée dans *au moins une association*.
- n Cela signifie qu'une entité *ne peut exister que* si elle est impliquée dans *plusieurs associations*.

La cardinalité maximale peut-être :

- 0 Cela signifie qu'une entité *ne peut pas être impliquée* dans une association.
- 1 Cela signifie qu'une entité peut être impliquée dans *au maximum une association*.
- n Cela signifie qu'une entité peut être impliquée dans *plusieurs associations*.

Attention ! En toute logique, le cas 0 ne doit pas exister : il démontre un problème de conception puisque le type-entité est inutile au type-association. Il faut alors reconsidérer la cardinalité ou retirer la liaison entre le type-entité et le type-association.

2.6. Dimension

Définition : La **dimension** (ou **nombre de pattes**) d'un type-association est le nombre de types-entités qui y participent. Si un type-entité participe plusieurs fois au type-association, il est compté autant de fois que de participations. On distingue :

Les associations binaires: qui ne relient que 2 entités.

Les associations trinaires: qui ne relient que 3 entités.

Les associations n-aires: qui relient plus de trois entités.

Les associations réflexives: qui relient 1 seule entité.

Partie 3. Modèle relationnel

En 1969, un mathématicien appelé CODD, travaillant au centre de recherche de IBM à SAN JOSE CALIFORNIE, publié un article qui donne les bases du modèle relationnel.
Ce modèle n'a été réellement implanté que vers la fin des années 70. C'est le modèle le plus utilisé par les SGBDs actuellement disponibles sur le marché.

Définition: Les objets et associations du monde réel sont représentés par un concept unique : les relations ; les relations sont des tableaux à deux dimensions appelés des tables.

3.1. Notion de domaine

Un domaine est un ensemble de valeurs que peut prendre un attribut. Exemple de domaines:

Dnom : chaînes de caractères de longueur maximale 30

Dnum : entiers compris entre 0 et 99999

Dsexe : {"f", "m"}

Dniveau:{1ère, 2ème, 3ème , 4ème , 5ème} .

3.2. Attribut

Il représente une colonne d'une relation caractérisé par un nom : nom, num, sexe…sont des attributs de la relation étudiant.

3.3. Relation (table): est un sous ensemble du produits cartésien de domaines, ce sous ensemble sera désigné par un nom qui sera le nom de la relation

Exemple :

Dnum : {0001/05,0002/05,0003/5….0050/05}

Dsexe : {"f", "m"}

Dniveau:{1ère, 2ème , 3ème , 4ème , 5ème} .

Dnum*Dsexe*Dniveau={(0001/05, f,1ère),(0001/05, m,1ère),(0001/05, f,2ème)……..(0050, m, 5ème)}.

Soit par exemple la relation étudiant décrite de la manière suivante :

Etudiant={(0001/05,m,1ere),(0002/05,f,1ere),(0003/05,f, 1ere),..(0050/05,m,1ere)}

Etudiant	num	sexe	niveau
	0001/05	m	1ere
	0002/05	f	1ere
	*	*	*
	0050/05	m	1ere

Chaque ligne appelée un tuple. Etudiant est le nom de la relation et les entêtes de colonnes sont les attributs.

3.4. Clé d'une relation

Une clé est constituée d'un ou plusieurs attributs dont les valeurs définissent de manière unique les tuples de la relation. Une relation est définie par :

- son nom
- liste des attributs
- son (ses) identifiant(s)

Les trois informations constituent le **schéma** de la relation.

Exemple : schéma de la relation Etudiant : Etudiant (<u>N° Etud</u>, Nom, Prénom, Age).

3.5. Population

La **population** d'une relation est constituée de l'ensemble des tuples de la relation. C'est un ensemble; il n'y a donc ni doubles, ni ordre (les nouveaux tuples sont rajoutés à la fin de la relation).

On appelle **schéma d'une base de données relationnelle** l'ensemble des schémas de ses relations. On appelle base de données relationnelle, la population de toutes ses relations.

3.6. Règle de passage du Modèle E/A au modèle relationnel

La modélisation EA des données étant effectuée, il faut implanter la structure obtenue en machine, par exemple sous forme d'un SGBD relationnel. Nous allons donc transformer notre structure sous une forme relationnelle. On dit aussi que l'on transforme le diagramme EA en schéma relationnel.

Règle 1 : Toute entité devient une relation.

L'identifiant de l'entité devient clé primaire de la relation.

Les attributs de l'entité deviennent attributs de la relation.

E1
P1 P2 P3

E1(P1,P2,P3, …)

Règle 2 : Association de type: (*,N)(*,N)/ *={0,1}.

Cette association devient une relation dont la clé de cette relation est la concaténation des clés des entités participantes et les attributs de l'association sont insérés dans la relation.

Règle 3 : Association de type : 1,1 *,N

Chaque entité devient une relation. La clé de l'entité père devient un attribut dans la relation fils.

Règle 4 : Association de type : (1,1)(1,1) :

Chaque entité devient une relation, la clé d'une entité devient un attribut dans l'autre.

Règle 5 : Association de type :(0,1) (*N)

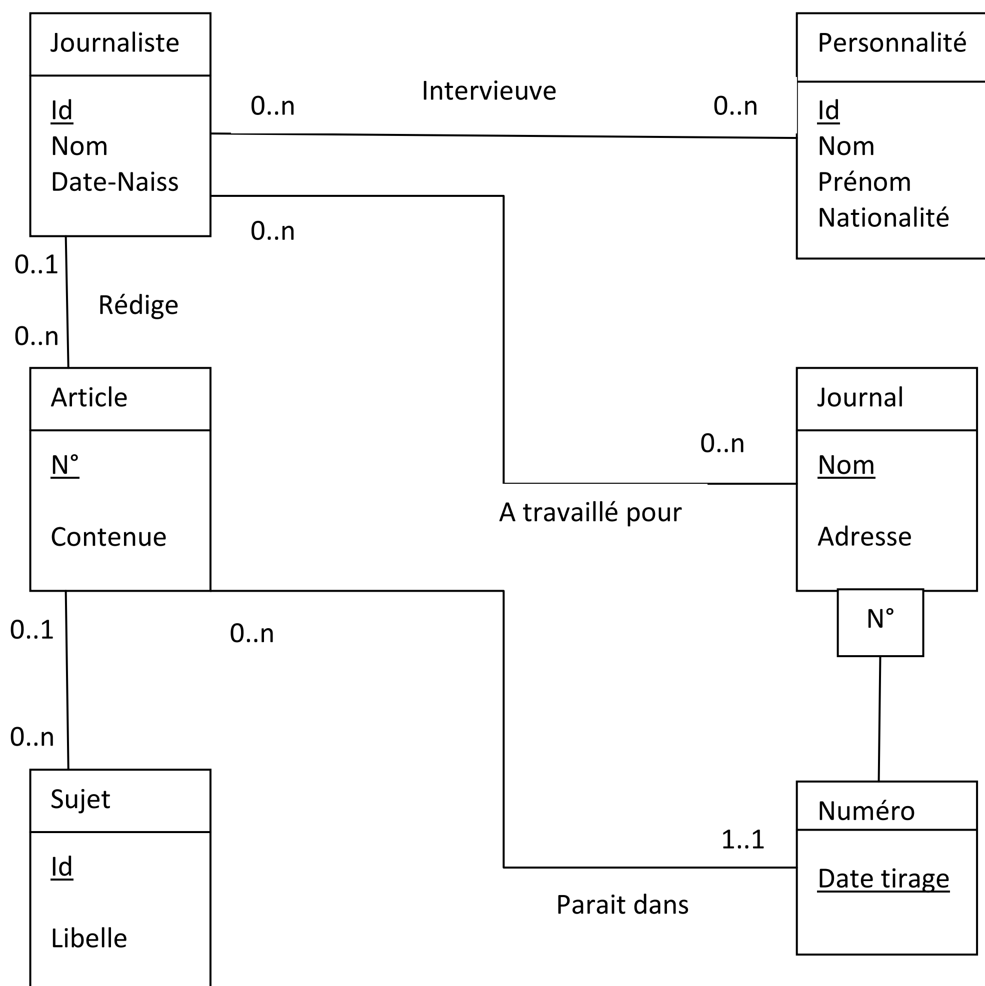
Elle est traitée comme étant une association de type : *N *N

Exercices sur le relationnel (Rappel)

Exercice 1 :

Etant donné le shéma Entité/ Association (E/A) du Système d'Information (SI) très simplifié d'un quotidien.

1- Un article peut-il être rédigé par plusieurs journalistes ?
2- Un article peut-il être publié plusieurs fois ?
3- Peut-il avoir plusieurs articles sur le même sujet dans le même numéros ?
4- Connaisant un article, est ce qu'on connait le journal où il est paru ?



Exercice 2 :

La relation suivante est-elle conforme à la définition ? si non, citez les anomalies

Titre	Année	Metteur en scéne	Acteur
Cyrano	1992	Rapelo	Departdieu , Perez
Les oiseaux	1963	Hitchcock	Taylor
Les oiseaux	1963	Hitchcock	Taylor
Titanic	1999	Camero	Dicaprio

Exercice 3 :

Des editeurs se réunissent pour créer une base de données sur leurs publications scientifiques. Dans de telle publications, plusieurs auteurs se regroupent pour écrire un livre en se répartissant les chapitres à rédiger. Après discution, voici le schéma obtenu :

Livres (Titre-Livre, année, éditeur, chiffre-affaire)

Chapitre (Titre-Livre, Titre-Chapitre, Nombre-Pages)

Auteur (Nom-Auteur, Prénom, Année-Naissance)

Rédaction (Nom-Auteur, Titre-Livre, Titre-Chapitre)

Les clés primaires sont les soulignés, mais les clès étrangéres ne sont pas signalés.

1- Donnez le schéma E/A correspondant au schéma relationnel.

2- Donnez les ordres Create- Table pour le schéma, en spécifiant soignesement les clés primaires et étrangéres avec le syntaxe SQL. Le type de données est secondaire, choisissez ce qui vous semble logique.

3- Sur quelle clès étrangère devrait-on mettre l'option On delette Cascade ?

Exercice 4 :

On trouve dans un SGBD relationnel les relations suivantes. Les clès primaires sont soulignés et les clès étrangères ne sont pas signalèes.

Immeuble (Nom, Adresse, Nombre-Etage, Année-Construction, Nom-Gérant)

Appartement (Nom-Immeuble, Numéro-Appartement, Type, Superficié, étage)

Personne (Nom, Prénom, Age, Code-Profession)

Occupant (Nom-Immeuble, Numéro-Appartement, Nom-Occupant, Année-Arrivée)

Propriètè (Nom-Immeuble, Nom-Propriètère, Quot-Part)

TypeAppart (Code, Libellé)

Profession (Code, Libellé)

1- Identifier les clès étrangères dans chaque relation, et reconstruire le schéma E/A

Chapitre 2 : Les BDs Orientées Objets

1. Limites des SGBDs relationnels et nouveaux besoins

Les SGBD relationnels ont à leur avantage:

- le modèle de données est très simple et donc facile à comprendre pour les utilisateurs;
- le modèle repose sur une base formellement définie; ce qui a permis de définir des méthodes de conception de schémas (théorie de la normalisation) et des langages de manipulation de données (LMD) déclaratifs et standardisés (SQL, QUEL...).

Cependant de nouvelles applications, différentes des applications de gestion classiques et qui ont de nouveaux besoins, se multiplient. Ce sont, par exemple, la conception ou production assistée par ordinateur, le génie logiciel, la bureautique, la recherche d'informations sur la toile, les systèmes d'informations géographiques, etc. qui manipulent des objets plus complexes et plus volumineux, comme les textes, graphiques, cartes, images, dessins multidimensionnels, vidéos, sons… De plus le coût des mémoires secondaires continuant à baisser, les applications classiques de gestion évoluent et veulent aussi manipuler des informations géographiques et des images. Par exemple en géomarketing il faut tenir compte de la localisation des clients actuels et potentiels ainsi que de celle des entrepôts et des points de vente. Autre exemple, les compagnies d'assurance veulent conserver les croquis, les photographies et la localisation géographique des accidents routiers qu'elles couvrent.

Ces applications ont révélé les limites du modèle relationnel, notamment:

- le modèle de données est trop simple et ne permet pas de représenter facilement les entités du monde réel qui sont souvent plus complexes qu'une relation. Dans les bases de données relationnelles, les entités du monde réel sont éclatées en plusieurs relations, ce qui multiplie les jointures dans les requêtes des utilisateurs et fait que les performances des SGBD relationnels pour ces applications sont mauvaises. De plus le modèle relationnel ne permet pas de représenter explicitement les différents types de liens sémantiques qui peuvent lier des entités : composition, généralisation / spécialisation, association… Il ne permet pas non plus au concepteur de base de données de définir de nouveaux domaines de valeurs (autres que les domaines standards des SGBD, Integer, Char, Date…), avec leurs fonctions de manipulation, et qui seraient adaptés à leur application.
- l'incompatibilité des LMD relationnels et des langages de programmation:
    - les LMD sont déclaratifs et fournissent en résultat un ensemble de tuples, alors que les langages de programmation sont impératifs et travaillent sur un élément à la fois;
    - les types de données manipulés par les langages de programmation sont plus variés et peuvent être plus complexes que les domaines de valeurs des LMD relationnels.
- le développement d'applications n'est pas satisfaisant: lenteur du développement, résultat souvent décevant (ne correspond pas ou mal aux besoins), applications difficilement maintenables.
- le mécanisme classique de gestion des transactions, ACID (Atomicity: transactions entièrement exécutées ou pas du tout, Consistency: base de données laissée toujours cohérente après la transaction, Isolation: indépendance de chaque transaction qui peut se croire seule, Durability: les

modifications effectuées par la transaction ne peuvent pas être perdues) est efficace pour les transactions courtes qui sont la règle dans les applications de gestion traditionnelle. Mais il se révèle inadapté aux transactions longues qui sont fréquentes dans ces nouvelles applications.

Les bases de données orientées objets (BDO) qui cherchent à répondre à ces nouveaux besoins, sont nées de la convergence de deux domaines:

- les bases de données;
- les langages de programmation orientés objets, tels que Eiffel, Smalltalk, C++, Java… L'objectif de ces langages est d'accroître la productivité des développeurs en permettant de créer des logiciels structurés, extensibles, réutilisables et de maintenance aisée. Leurs principes essentiels sont:
    - les objets, qui comportent deux parties: leur valeur, et les opérations, appelées méthodes, qui permettent de les manipuler. Selon le principe d'encapsulation des données, la valeur des objets est cachée. L'accès et la mise à jour des objets se fait par appel de leurs méthodes. Cela rend plus facile la maintenance des logiciels construits selon ce paradigme.
    - l'héritage, qui permet à une classe d'objets d'avoir les mêmes propriétés (structure de données et méthodes) qu'une autre classe sans avoir à les redéfinir. C'est l'héritage qui permet d'étendre et de réutiliser facilement des logiciels.

Les bases de données orientées objets sont caractérisées par quatre points essentiels:

- un modèle de données qui permet de représenter des structures de données complexes;
- les données et les traitements ne sont plus séparés. La dynamique (les méthodes) fait partie de la déclaration des classes d'objets;
- l'héritage;
- tout objet possède une identité qui le distingue de tout autre objet, même s'ils ont la même valeur.

D'autres critères sont souvent ajoutés à cette liste, notamment le suivant:

- il n'y a plus d'incompatibilité majeure entre le langage de programmation et le langage de manipulation des données.

Les SGBDO sont un domaine en évolution et il n'y a pas encore de consensus, tant sur le modèle de BDO que sur le LMD objet, alors que de nombreux produits sont proposés sur le marché. Il y a actuellement deux propositions rivales de norme :

- ODMG, défini en 1993, est une spécification d'interfaces pour SGBDO "purs". ODMG rassemble, entre autres, un groupe de constructeurs de SGBDO qui se sont engagés à respecter ces spécifications dans leurs produits.
- SQL3 est une norme pour les SGBD relationnels-objets, définie par le comité international de normalisation ISO. C'est une extension du SQL relationnel aux fonctionnalités orientées objet et à d'autres fonctionnalités, telles que les règles actives, le multi-média, le spatial ou le temporel.

Cette extension est compatible avec l'ancien SQL, c'est-à-dire que les bases de données relationnelles traditionnelles peuvent être gérées par SQL3 et que toute requête SQL classique peut s'exécuter sur un des nouveaux systèmes offrant SQL3. SQL3 est offert par les dernières versions des SGBD relationnels des grands constructeurs comme IBM, Oracle ou Sybase.

Pour le moment, les deux propositions sont incompatibles, mais des discussions ont lieu entre les deux comités pour essayer de faire converger les deux normes en une seule.

2. Structure de données

La structure de données des modèles orientés objets est caractérisée par le fait qu'il n'y a qu'un seul concept principal: l'objet. Les objets sont décrits par des attributs, et sont regroupés en classes.
Chaque objet a une identité qui lui est propre et qui le distingue de tous les autres. Cette identité est permanente et immuable. On l'appelle l'oid de l'objet (de l'anglais "object identity"). Il existe deux types de liens entre les objets :
- les liens de composition: "tel objet est composé de tel(s) objet(s)". Ces liens sont décrits par des attributs particuliers, appelés attributs référence, qui ont pour domaine une classe d'objets;
- les liens de généralisation / spécialisation: "tel objet de la base décrit sous un autre point de vue (plus général ou au contraire plus détaillé) la même entité du monde réel que tel autre objet de la base". Ce sont les liens IS-A des modèles sémantiques auxquels est associé un mécanisme d'héritage des propriétés de la classe d'objets génériques par la classe d'objets spécialisés.

Il n'existe pas un seul modèle de données orienté objets, mais un par SGBDO existant. Nous présentons ici les points essentiels et signalons les variantes principales. Pour les exemples nous employons –sauf indication contraire– une syntaxe très proche de celle d'ODMG.

2.1. Structure complexe

Les SGBDO permettent à leurs utilisateurs de définir des structures quelconques et qui peuvent être complexes. Les attributs peuvent être, comme dans le modèle entité association vu en cours ou comme en Codasyl (un ancien SGBD de type réseau), complexes et multivalués. La terminologie, cependant, est différente. A ces structures peuvent être associées des fonctions permettant de les manipuler (voir paragraphe 3). Ces structures sont utilisées par les concepteurs de bases de données dans deux cas:
- pour définir la structure des objets, c'est-à-dire celle de leur classe;
- pour étendre l'ensemble des domaines (appelés aussi types) prédéfinis existants dans le SGBDO (tels que STRING, INT, DATE, MONEY…) en définissant des domaines spécifiques à leur application.

Les structures sont définies en employant des constructeurs:
- le constructeur de structure complexe (l'équivalent d'un attribut complexe), "STRUCT", qui crée une structure composée d'une suite d'attributs: STRUCT {nomatt1: dom1; nomatt2: dom2; ...} où "domi" est le domaine de l'attribut de nom "nomatti", c'est :
    - soit un domaine prédéfini (STRING, FLOAT, INT, DATE...);
    - soit un type défini par un constructeur;
    - soit le nom d'une classe (voir paragraphe 2.3).
- les constructeurs de collection (l'équivalent des attributs multivalués). Par exemple, le constructeur "SET", crée un type ensemble d'éléments: *SET domaine*
où "domaine" a la même définition que ci-dessus.

Les autres constructeurs usuels de collection sont: la liste (LIST), le multi-ensemble ou ensemble pouvant contenir des doubles (BAG) et le tableau (ARRAY).

Exemple: Classe contenant des attributs complexes et multivalués:

```
CLASS Etudiant                                   /* déclaration d'une classe d'objets */
        { num : INT ;
        nom : STRING ;
        prénoms : LIST STRING ;                  /* attribut multivalué de type liste */
        date-nais : DATE ;
        adresse : STRUCT {n° : INT ;             /* adresse: attribut complexe */
                         rue : STRING ;
                         ville : STRING ;
                         pays : STRING } ;
        cours-suivis : SET STRUCT { nom-cours : STRING ;    /* cours-suivis: attribut
                                    note : FLOAT} }            complexe et multivalué */
```

Exemple: Seconde version du même exemple, mais avec déclaration d'un nouveau domaine, T_Adresse.

```
TYPEDEF T_Adresse                                           /* déclaration d'un domaine
        STRUCT {n° : INT ;                                     de type complexe */
                rue : STRING ;
                ville : STRING ;
                pays : STRING }
CLASS Etudiant                                       /* déclaration d'une classe d'objets */
        {num : INT ;
         nom : STRING ;
         prénoms : LIST STRING ;                     /* attribut multivalué de type liste */
        date-nais : DATE ;
        adresse : T_Adresse;                         /* attribut de domaine complexe */
        cours-suivis : SET STRUCT {nom-cours : STRING ;     /* cours-suivis: attribut
                                    note : FLOAT} }           complexe et multivalué */
```

Dans ce second exemple, la structure de la classe Etudiant est exactement la même que dans le premier exemple. L'intérêt d'avoir déclaré un domaine T_Adresse est qu'il peut être réutilisé comme domaine dans plusieurs classes et même lors de la définition d'autres domaines.

2.2. Identité d'objet

L'objectif est de pouvoir identifier tout objet indépendamment de sa valeur (ce qui implique, entre autres, de pouvoir gérer des objets de même valeur mais distincts). Cet identifiant doit être permanent (qui existe pendant au moins toute la durée de vie de l'objet), fixe (qui ne change pas durant la vie de l'objet) et unique dans la base et dans le temps (deux objets distincts de la même base, même s'ils n'existent pas en même temps n'auront jamais la même identité).

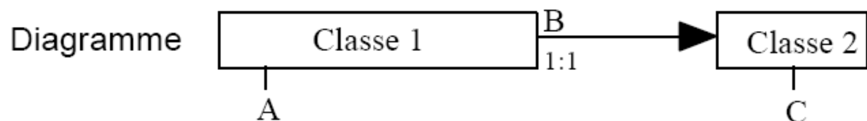
L'identité des objets est implémentée par un identifiant système géré par le SGBDO. Dès la création d'un objet, un identifiant système lui est attaché. La valeur de cet identifiant n'est ni affichable, ni imprimable, ni modifiable. On appelle cet identifiant, la référence de l'objet, ou son **oid**.

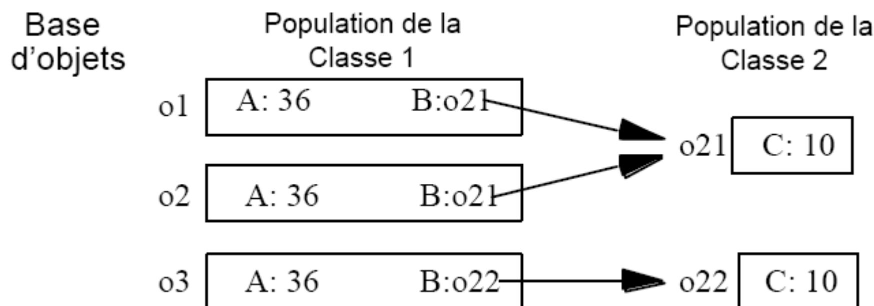
L'égalité de deux objets, o1 et o2, peut se mesurer à plusieurs niveaux:

- **l'identité** d'objets, notée:
o1 == o2
et qui signifie: "est-ce le même objet (le même oid)?"
- **l'égalité de valeur**, notée:
o1 = o2
et qui signifie: "les objets o1 et o2 ont-ils la même valeur ? C'est-à-dire tous leurs attributs ont la même valeur, que ce soient des attributs-valeur ou des attributs-référence.
- **l'égalité de valeur après fermeture**, notée:
o1 =f o2
et qui signifie: les objets o1 et o2 ont-ils la même "valeur après fermeture" ? C'est-à-dire:
    - les objets o1 et o2 ont des graphes de composition isomorphes (leurs attributs-référence constituent des graphes équivalents), et
    - les attributs-valeur correspondants, à tous les niveaux, sont égaux.

    Pour deux objets, o1 et o2, de la même classe, on a les propriétés suivantes:
    o1 == o2 $\Rightarrow$ o1 = o2
    o1 = o2 $\Rightarrow$ o1 =f o2

Exemple: Soient les deux classes d'objets représentées ci-dessous. La classe 1 possède un attribut A de domaine entier, et un attribut référence B qui contient l'oid d'un objet de la classe 2 (voir les liens de composition au paragraphe suivant). La classe 2 possède un attribut C de domaine entier.





On a alors:

o1.B == o2.B mais o1 == o2 est faux
o1 = o2 mais o1 = o3 est faux
o1 =f o3
o21 = o22 mais o21 == o22 est faux

Comparaison: C'est le même principe que dans les langages de programmation où chaque élément d'un programme possède une identité (nom de la variable ou adresse physique).
C'est différent du relationnel qui est basé sur les valeurs. En relationnel, l'utilisateur doit inventer des identifiants (ou clés) et les gérer (donner des valeurs différentes aux clés des différents tuples), ce qui pose problème en cas de mise à jour des attributs de la clé.
Les SGBD Codasyl gèrent aussi l'identité des objets: tout record a un identifiant système appelé "database key".

2.3 Liens de composition

Un lien de composition relie une classe C1 à une classe C2 si les objets de C1 (appelés "objets composés") sont composés d'objets C2 (appelés "objets composants"). C'est un lien orienté de cardinalité quelconque (0:1 ou 1:1 ou 1:N ou N:M ou 0:N…). Par exemple, les objets bicyclettes sont composés d'objets roues, cadre, pédalier... Il y a un lien de composition entre la classe bicyclette et la classe roue, un second entre la classe bicyclette et la classe cadre, un troisième entre la classe bicyclette et la classe pédalier... Les liens de composition sont représentés sur les diagrammes par des flèches fines.

Ces liens de composition sont implantés par des attributs-référence qui pointent sur les objets composants. Leur valeur est l'oid de l'objet référencé. On peut avoir des objets partagés qui sont composants de plusieurs objets composés. La déclaration d'un lien de composition est faite à l'intérieur de la description de la classe composée par un attribut-référence dont le domaine est celui de la classe composante.

Pratiquement, ces liens de compositions servent à décrire le fait qu'un objet est composé d'autres objets, mais aussi tout autre lien de type association, car il n'existe pas dans le modèle orienté-objets de lien d'association générique. Comme par exemple dans l'exemple suivant où la classe Cours est modélisée comme étant une classe composante de la classe Etudiant2 :

```
CLASS Etudiant2
      {num : INT ;
       nom : STRING ;
       prénoms : LIST STRING ;
       cours-suivis : SET STRUCT {cours : Cours ;
       note : FLOAT } }
CLASS Cours
      {nomC : STRING ; .... }
```

Les modèles de BDO existants proposent différentes variantes du lien de composition et parfois des contraintes d'intégrité qui peuvent lui être associées. Ce sont essentiellement:

• Un objet composant peut être, ou non, **partagé** entre plusieurs objets de la même classe composée. Par exemple, dans une base de données décrivant des voitures en réparation dans un garage, Moteur est une classe d'objets composants non partagés de la classe Voiture. Par contre, dans une base de données décrivant les modèles de voitures, Moteur serait une classe d'objets composants partagés de la classe Voiture, puisque deux modèles de voitures peuvent avoir le même type de moteur.

• Une classe composante peut être, ou non, **dépendante** de sa (d'une de ses) classe composée. Un objet o1 composant est dépendant de son objet père o2 si l'existence de o1 dépend de celle de o2, c'est-à-dire si la destruction de o2 entraîne automatiquement celle de o1. Par exemple, dans la base de données décrivant les voitures en cours de réparation, si le garage ne récupère aucune pièce sur les voitures envoyées à la casse, alors Moteur serait une classe d'objets composants dépendants de Voiture. Si au contraire, le garagiste récupère des pièces, et en particulier certains moteurs, alors Moteur serait une classe composante non dépendante de Voiture.
Certains SGBDO assurent la contrainte inverse : l'objet composé dépend de ses composants.
Les contraintes de dépendance définissent l'ordre selon lequel les objets doivent être créés (et détruits): d'abord les composants ou d'abord le composé.

• Parfois des **cardinalités** peuvent être associées au lien de composition. C'est l'équivalent des cardinalités des rôles des types d'association du modèle entité association. Par exemple, la contrainte "C2 est une classe composante non partagée de la classe C1", qui signifie qu'un objet de C2 ne peut être lié par un lien de composition qu'à un seul objet de la classe C1, peut alors s'exprimer de la façon suivante : "la cardinalité maximale du lien de composition C1-C2 du côté de C2 est 1".

• Le lien de référence est orienté de la classe composée vers la classe composante, à cause de son implémentation (pointeur logique sur les objets de la classe composante). Cela implique que, dans les requêtes, il est beaucoup plus facile et rapide d'aller de l'objet composé aux objets composants que l'inverse. Pour avoir un accès aisé dans les deux sens dans la base de données, il est utile de décrire le lien de composition et son **inverse**.

Exemple :

```
CLASS Cours2
        {nomC : STRING ;
         étudiants : SET Etudiant2 }
```

L'attribut étudiants de Cours2 est un attribut référence inverse de l'attribut cours-suivis de la classe Etudiant2.

Pour que le SGBDO soit à même d'assurer la cohérence de ces attributs référence inverses, certains systèmes permettent de déclarer l'existence des attributs inverses par une clause "INVERSE". L'exemple s'écrit alors:

```
CLASS Cours2
        {nomC : STRING;
         étudiants: SET Etudiant2 INVERSE Etudiant2.cours-suivis.cours }
```

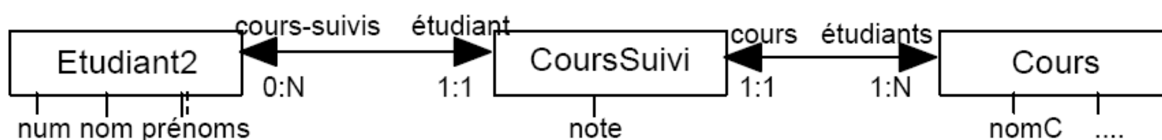
ODMG: Certains SGBDO, parmi les plus récents, se rapprochent du modèle entité association. La norme ODMG par exemple, n'autorise les attributs référence qu'en attributs du premier niveau des classes; les attributs composants d'attributs complexes ne peuvent pas être des attributs référence. L'exemple précédent de la classe Etudiant2:

```
CLASS Etudiant2
        {num : INT ;
         nom : STRING ;
         prénoms : LIST STRING ;
         cours-suivis : SET STRUCT {cours : Cours ;
                                    note : FLOAT } }
```

serait impossible tel quel en ODMG, car l'attribut référence cours est un attribut composant. Pour décrire cette situation en ODMG, il faudrait introduire une classe intermédiaire (qui représente une association du schéma entité association équivalent) de la façon suivante:

```
CLASS Etudiant2
        {num : INT ;
         nom : STRING ;
         prénoms : LIST STRING ;
         cours-suivis : SET CoursSuivi }
CLASS CoursSuivi
        {cours : Cours ;
         étudiant : Etudiant2 INVERSE Etudiant2.cours ;
         note : FLOAT }
```

Ce qui serait représenté graphiquement de la façon suivante en ODMG:



La norme ODMG offre aussi les attributs référence inverses. Ainsi déclarer un attribut du premier niveau comme référence avec son attribut référence inverse, lui aussi au premier niveau, est équivalent à déclarer une association binaire. Cependant ODMG ne permet pas d'attacher des attributs à ces pseudo-associations binaires, ni de définir directement des associations n-aires. Pour représenter en ODMG ces deux derniers cas (associations avec attributs et associations n-aires) il faut introduire des classes supplémentaires comme dans l'exemple ci-dessus.

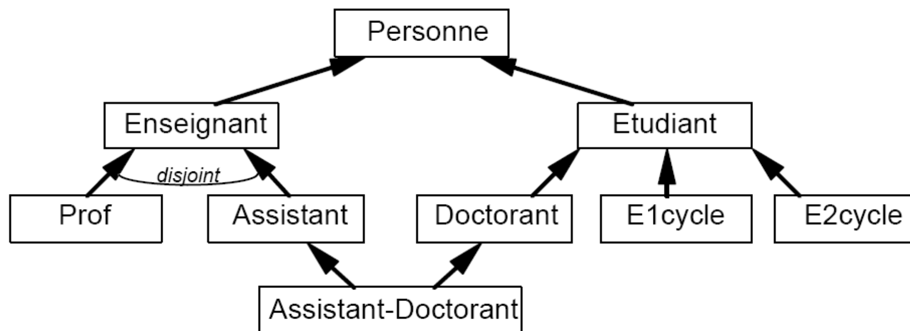
2.4. Graphe de généralisation/spécialisation des classes

Certains ensembles d'objets du monde réel ayant des caractéristiques communes peuvent être perçus comme comprenant un ou plusieurs sous-ensembles d'objets, chaque sous-ensemble ayant des propriétés particulières. Les modèles de données orientés objets permettent de décrire ce type de situation à l'aide du concept de généralisation/spécialisation (ou lien IS-A des modèles sémantiques) auquel ils ont ajouté un mécanisme d'héritage.

Un lien de généralisation/spécialisation, **CS –>CG,** est un lien binaire orienté entre deux classes, CS appelée sous-classe (ou classe spécifique) et CG appelée sur-classe (ou classe générique). Il est noté graphiquement sur les diagrammes par une flèche épaisse ; et il défini par les trois règles suivantes :

- l'inclusion des populations : CS représente un sous-ensemble des entités du monde réel décrites par CG. Du point de vue conceptuel, la population de CS est incluse dans celle de CG. (En pratique, au niveau interne, la solution retenue est souvent différente.)
- la substituabilité des objets de CG par des objets de CS : lors de la manipulation des objets, partout où l'on peut employer un objet de type CG, on peut aussi employer un objet de type CS;
- et l'héritage : CS hérite des attributs et méthodes (voir paragraphe 3) de CG. C'est ce concept d'héritage de la structure et des méthodes qui permet la réutilisation des classes.

Exemple : graphe de généralisation des classes décrivant les personnes dans une université



Déclaration partielle de ce graphe de généralisation (en ODMG, le lien de généralisation/spécialisation se déclare dans le schéma textuel par le caractère "**:**"):

Class Personne
{nom : **String** ;
prénoms : **List String**;
adresse : **String** ;
afficher() } ;
Class Etudiant **:** Personne
{n°E : **Int** ;
dateN : **Date** ;
études : **Set Struct** {année : **Int** ;
diplôme : **String** } ;
cours-obtenus : **Set Struct** {cours : Cours ;
année : **Int** ;
notes : **Set Float** } ;
cours-suivis : **Set** Cours ;
afficher() ; /* nouvelle définition de la méthode, particulière aux objets de type Etudiant */
inscrire(cours : Cours) ;
aobtenu(cours : Cours ; notes : **Set Int**) } ;

```
Class Enseignant : Personne
        {tél : Int ;
         statut : Enum: ("prof"; "assistant") ;
         rens.banc : Struct {banque : String ;
                            agence : String ;
                            compte : Int } ;
        coursdonnés : Set Cours ;
        afficher() ; /* nouvelle définition de la méthode, particulière aux objets de type
                   Enseignant*/
        assure (cours : Cours ) ;
        nassureplus (cours : Cours ) } ;
```

Dans cette base de données, tout professeur est à la fois un Prof, un Enseignant et une Personne. En d'autres termes, tout objet Prof "a" les attributs et méthodes de ces trois classes. Plus précisément, la classe Enseignant possède des attributs et méthodes qui lui sont propres: tel, statut, rens.banc, coursdonnés, assure, nassureplus. Elle hérite des attributs et méthodes de sa sur classe Personne: nom, prénoms, adresse, afficher (voir ci-dessous pour l'héritage avec redéfinition de afficher).

Le graphe engendré par la hiérarchie des classes ne doit pas contenir de cycle. Cette interdiction des cycles découle directement de la définition des liens IS-A avec inclusion de population. En effet si l'on décrivait par exemple les deux liens IS-A suivants:

C1 → C2 et C2 → C1

Cela signifierait que la population de C1 est contenue dans celle de C2 et vice versa. Donc, les deux classes devraient décrire exactement le même ensemble d'objets, elles devraient donc être confondues en une seule et même classe.

Des contraintes d'intégrité de couverture, de disjonction et de partition peuvent être associées aux sous-classes d'une classe.

- un sous-ensemble, S, des sous-classes immédiates d'une sur-classe C est **disjoint** si toute occurrence de C peut être occurrence d'au plus une des classes de S. Dans l'exemple du graphe de généralisation des personnes d'une université, un assistant ne peut pas être aussi un professeur, on a donc une contrainte de disjonction entre ces deux sous-classes.

- un sous-ensemble, S, des sous-classes immédiates d'une sur-classe C est **couvrant** si toute occurrence de C est aussi occurrence d'une (au moins) des classes de S. Dans l'exemple du graphe de généralisation des personnes d'une université, si les seules personnes enregistrées dans la base sont les enseignants et les étudiants, alors il y a une contrainte de couverture: Enseignant et Etudiant constituent une couverture de Personne. Mais il n'y a pas de disjonction entre Enseignant et Etudiant, car il peut y avoir des assistant-doctorants (assistants inscrits en thèse).

- enfin, un sous-ensemble, S, des sous-classes immédiates d'une sur-classe C constitue une **partition** de C, si ce sous-ensemble est à la fois disjoint et couvrant.

Restrictions: La plupart des SGBDO commercialisés limitent fortement les possibilités théoriquement offertes par la hiérarchie de généralisation / spécialisation. Souvent ils imposent aux sous-classes issues d'une même sur-classe soit d'être disjointes, soit que leur recouvrement ait été explicitement déclaré par une sous-classe commune. Cette restriction est due au fait que ces systèmes se basent sur le principe de l'instanciation unique des objets: chaque objet est stocké par le système dans une seule classe, qui est la classe la plus spécialisée le contenant (cette classe étant nécessairement unique).
Une seconde restriction, très fréquente elle aussi, est que la hiérarchie de généralisation / spécialisation soit statique: une fois qu'un objet a été créé dans une classe, il ne peut plus en changer, ni acquérir de nouvelles sous-classes. Il ne peut pas être spécialisé, ni "dé-spécialisé", ni passer dans une autre sous-classe. Par exemple dans la hiérarchie des personnes de l'université, pour ces systèmes, un objet de E1cycle ne peut pas devenir un objet de E2cycle. Pour cela, l'utilisateur devrait détruire l'objet de E1cycle puis créer un nouvel objet (avec un nouvel oid!) dans E2cycle.

Raffinement et redéfinition: Certaines propriétés peuvent avoir des caractéristiques particulières propres à une sous-classe. Par exemple, l'attribut statut de Enseignant aura toujours la valeur « prof » dans la sous-classe Prof, et la valeur "assistant" dans la sous-classe Assistant. De même, il est parfois utile de définir pour une méthode héritée un nouveau comportement spécifique de la sous-classe. Par exemple, la méthode afficher() appliquée à une Personne affichera les nom, prénoms et adresse de la personne, appliquée à un Etudiant elle affichera en plus les cours suivis. Pour cela, certains SGBDOs offrent la possibilité de raffiner un attribut ou redéfinir une méthode héritée :

- raffiner dans une sous-classe un attribut hérité signifie déclarer que dans la sous-classe son domaine de définition (ou ses cardinalités) est réduit à un sous domaine de celui qu'il a dans la sur-classe. Cette réduction respecte le principe de substituabilité.
- redéfinir dans une sous-classe une méthode héritée signifie déclarer dans la sous-classe un nouveau comportement (code) tout en conservant sa signature (voir paragraphe 3).
- D'autres variantes, telles le changement de signature d'une méthode, peuvent aussi être offertes.

L'**héritage multiple** est possible: une classe peut avoir plusieurs sur-classes. C'est le cas ci-dessus pour la classe Assistant-Doctorant. Les assistants doctorants sont à la fois assistants (et donc enseignants) et doctorants (et donc étudiants). Dans le cas d'héritage multiple, un conflit peut survenir lors de l'héritage des attributs et des méthodes: deux attributs ou méthodes de deux surclasses peuvent avoir le même nom et n'être pas identiques. Par exemple, pour Assistant-Doctorant, quelle est la bonne méthode afficher qu'il faut employer? Celle de Enseignant ou celle de Etudiant?

Face à une telle situation, les SGBDO réagissent différemment:

- soit ils ne savent pas résoudre ce conflit, et refusent la définition d'un tel schéma;
- soit ils demandent au concepteur, lors de la définition d'un tel schéma, qu'il choisisse la sur-classe "dominante" dont la sous-classe héritera;
- soit ils appliquent une règle déterministe pour ce choix (par exemple la première classe citée comme sur-classe dans la définition textuelle de la sous-classe).

L'héritage conduit parfois, à l'image de ce qui se fait dans les langages de programmation, à créer des classes sans population propre, uniquement pour pouvoir hériter de leurs méthodes. C'est ce que font les SGBDO en créant une bibliothèque de classes prédéfinies contenant des méthodes utiles à toutes leurs sous-classes. Par exemple, la méthode delete() qui détruit l'objet appelé et la méthode same_as(o:OID)→b:BOOLEAN qui teste si l'objet appelé et l'objet o passé en argument sont un seul et même objet (même oid, test d'identité), sont valables pour toute classe et sont donc associées à une classe prédéfinie. Ces classes prédéfinies constituent la racine du graphe de généralisation/spécialisation qui regroupe toutes les classes. Un exemple en est donné au paragraphe 4.

Avantages / inconvénients:

La hiérarchie de généralisation/spécialisation permet de faire une description plus proche du monde réel (comme dans les modèles sémantiques) et de réutiliser par héritage des déclarations de structures et des codes de procédures.
Cependant, le principe de substituabilité et le mécanisme d'héritage qui en découlent conduisent à des techniques de raffinement et de redéfinition complexes à maîtriser et pour lesquelles il n'existe pas encore de solution universelle.

2.5. Populations et persistance

Dans les SGBD classiques, un type (une relation en relationnel, un type d'entité en entité association, un record type en Codasyl) comporte deux aspects:
- la définition de la structure des occurrences potentielles du type. C'est par exemple pour une table relationnelle, la liste des attributs avec leur domaine;
- la déclaration d'un récipient contenant toutes les occurrences existantes du type, appelé population du type. Par exemple en relationnel, dans les LMD le nom de la relation représente l'ensemble de ses tuples. De plus, toutes les occurrences sont, par définition, permanentes.

Dans les langages de programmation au contraire, un type ne décrit que la structure (et les opérations associées) des occurrences potentielles. Par exemple, le type INTEGER n'a pas de population associée: l'ensemble des entiers n'est pas stocké quelque part. Ainsi, les SGBDO, selon qu'ils ont été conçus par des chercheurs du monde des bases de données ou par ceux du monde des langages de programmation orientés objets, ont une approche différente de la population des classes.
- Les SGBDO issus du monde des bases de données, associent généralement à chaque classe sa population qui est permanente. Tout objet créé est automatiquement stocké par le système dans le récipient associé à la classe et qui a pour nom celui de la classe. C'est le cas du SGBDO Orion. ODMG a choisi une solution similaire : le concepteur, lors de la déclaration de la classe, a le choix entre définir ou pas un récipient pour la population de la classe grâce à la clause facultative "Extent nom-population".
- Les SGBDO issus du monde des langages de programmation orientée objets, séparent nettement les deux fonctions : d'un côté la définition de la structure, de l'autre le récipient de stockage. Les classes définissent la structure uniquement. Le SGBDO ne leur associe aucune population. C'est aux utilisateurs de créer des récipients dans lesquels stocker les objets.

L'utilisateur va donc créer une variable ou un objet de type collection (SET, LIST, BAG ou ARRAY) et y stocker les oids des objets qu'il veut regrouper.

Par exemple, pour conserver les différentes promotions d'étudiants du paragraphe 2.4, une solution est de déclarer une variable (promo) dans laquelle les utilisateurs stockeront eux-mêmes les objets créés de type Etudiant, de la façon suivante:

```
Program
promo: Struct {année: Int ; */ déclaration d'une variable structurée, promo /*
                    étudiants: Set Etudiant } ;
lucie : Etudiant ; */ déclaration d'une variable de type Etudiant, lucie /*
..........
promo.année := 2004 ; */ initialisation de promo (avec un ensemble d'étudiants vide) /*
......
lucie := Etudiant( nom:"Dupont"; prénoms:list("Lucie"); n°E:15344; dateN:1/1/83 ) ;
                          */ création d'un objet Etudiant, dont l'oid est rendu en réponse /*
promo.étudiants.insert_element(lucie) ;
                    */ insertion de l'objet lucie dans l'ensemble des étudiants de promo /*
……
```

Dans ce programme, un récipient pour les étudiants a été créé: promo. Dans ce récipient l'objet lucie a été stocké. On pourra donc par balayage (ou autre recherche) de promo.étudiants retrouver les étudiants de la promotion.

Il reste à rendre ce récipient et son contenu permanents. En effet, ce type de SGBDO gère deux sortes d'objets: les objets temporaires qui sont détruits automatiquement à la fin du programme utilisateur qui les a créés, et les objets permanents qui sont conservés jusqu'à ce que l'utilisateur veuille les détruire. Les SGBDO offrent là aussi plusieurs solutions telles que les suivantes.

- Une instruction permet de rendre permanent un objet. Par exemple en ODMG, on peut donner à tout objet un ou plusieurs noms permanents, qui pourront servir de point d'entrée dans la base. Pour cela, il suffit de déclarer un nom grâce à l'instruction "Name". Par exemple, l'instruction de déclaration: **Name** directeur : Personne ; déclare un nom permanent, directeur, de type Personne. Il suffit alors d'affecter à cette variable permanente l'objet que l'on veut rendre permanent.

- La règle suivante est instituée : "Est permanent tout objet qui est accessible à partir d'un objet permanent". C'est-à-dire que tous les objets composants d'un objet composé permanent sont eux-mêmes permanents, et ceci récursivement. D'autre part, le SGBD offre une "racine de persistance" (ou le moyen de la créer), c'est un premier objet permanent dont tous les autres objets permanents seront des composants, par exemple une liste, PersistList, dans laquelle l'utilisateur insérera les objets qu'il veut rendre permanents.

Dans l'exemple ci-dessus de la promotion d'étudiants, il faut pour rendre cet ensemble d'étudiants permanent, rajouter une instruction qui insère promo dans la liste PersistList. Cette dernière solution est celle qui a été retenue par les SGBDO Gemstone et O2.

Avantages / inconvénients :

L'intérêt de l'approche des SGBDO issus du monde des langages de programmation est de faire de la permanence une propriété indépendante des autres. Les objets temporaires et les objets permanents peuvent être manipulés de la même manière par l'utilisateur. La même classe peut servir à créer des objets permanents et des objets temporaires de même type.
A l'opposé, l'approche suivie par les SGBDO issus du monde des bases de données, a l'avantage de décharger entièrement l'utilisateur de la gestion des populations et de la persistance des objets.

3. La dynamique

Lors de la conception du schéma d'une base de données orientée objets, la structure des données et les opérations (les méthodes) qu'on peut effectuer sur ces données sont définies en même temps.
L'accès aux données des objets se fait ensuite à travers ces opérations: on dit que les données sont "encapsulées". Plus précisément, le principe d'**encapsulation** consiste à ne laisser connaître aux utilisateurs des objets d'une classe que l'interface d'appel des méthodes de la classe. La structure des objets et le code des méthodes leur sont cachés. Seules les méthodes de la classe et de ses sous-classes peuvent accéder directement à la valeur des objets.

Ce principe d'encapsulation répond aux objectifs suivants:
- simplifier le travail des programmeurs d'application en ne leur demandant que de connaître le minimum;
- faciliter la maintenance des applications: on peut facilement changer la structure d'un objet, le code d'une méthode ou rajouter une nouvelle méthode sans avoir à connaître les autres classes, ni à modifier les programmes d'application;
- réutiliser au maximum le code des programmes: les méthodes sont écrites une fois pour toutes et utilisées par tous les programmes d'application dont la taille (en nombre de lignes de code) diminue d'autant.

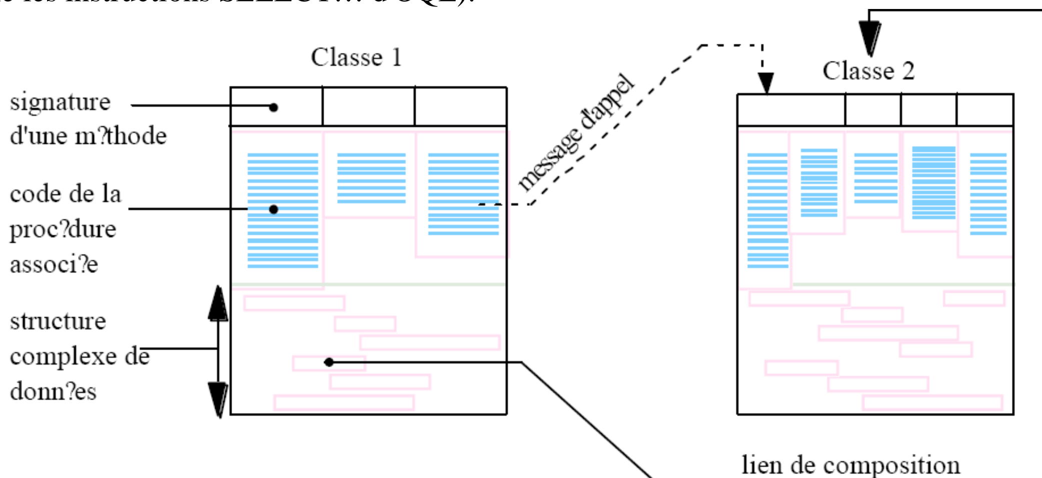
3.1. Les méthodes

Déclaration:
La définition d'une classe d'objets (tout comme celle d'un domaine) comporte donc la définition de la structure des objets et celle des méthodes servant à manipuler les objets. La déclaration d'une classe d'objets est composée de deux parties:
- l'interface, qui est visible aux utilisateurs: définition de la signature de chaque méthode (nom de la méthode, liste des paramètres d'appel avec leur type, et type du résultat s'il en existe un),
- l'implantation, qui est invisible aux utilisateurs: description du type d'objet (cf. paragraphe 2), et du corps de chaque méthode (code de la procédure associée).

Exemple :

Class Personne
{nom : **String** ;
prénoms : **Set String** ;
adr : **String** ;
Void afficher() ; /* signature de la méthode */
} ;

En ODMG, le mot clé "Void" indique que la méthode ne rend pas de résultat. La méthode afficher() déclarée ci-dessus ne rend pas de résultat et n'a pas de paramètre.
Le code des méthodes est défini par ailleurs en utilisant un langage de programmation orienté objets comme C++ ou Java. Le code des méthodes peut contenir, en plus des instructions du langage de programmation, des appels de méthodes standard du SGBD (par exemple pour des création et destructions d'objets), des appels de méthodes des classes d'objet de la base de données, et des requêtes du langage de manipulation de bases de données offert par le SGBD (par exemple les instructions SELECT… d'OQL).



Représentation schématique de l'encapsulation

Degrés d'encapsulation :
Les SGBDO n'appliquent pas tous strictement l'encapsulation. L'encapsulation stricte est lourde à appliquer, notamment elle rend difficile la définition d'un LMD déclaratif de type SQL. On peut définir différents degrés d'encapsulation:

- encapsulation stricte: tout accès depuis un objet o aux données d'un autre objet o' se fait par appel d'une méthode de l'objet o';
- encapsulation partielle: les classes d'objets ont deux types de données, les données publiques accessibles directement, et les données privées accessibles uniquement par l'appel de méthodes.
- encapsulation des écritures: tout accès en écriture se fait par appel d'une méthode, les accès en lecture peuvent être faits directement (sans passer par l'appel d'une méthode);
- encapsulation selon le type d'accès: les accès faits à partir d'un langage de manipulation de données déclaratif type SQL étendu aux objets peuvent ne pas respecter l'encapsulation, les accès faits depuis une méthode et un langage de programmation doivent respecter l'encapsulation.

Par exemple, le SGBDO O2 propose trois niveaux d'encapsulation pour les attributs et méthodes:

- `Private`: (option par défaut) qui signifie l'encapsulation stricte;

- `Read`: (option possible pour un attribut uniquement) qui signifie l'encapsulation des écritures;
- `Public`: aucune encapsulation, accès possible par tous en lecture et écriture.

Appel d'une méthode:

Lorsqu'un programme d'application (ou une méthode) demande à un objet d'exécuter une de ses méthodes, on dit qu'il envoie un message d'appel de méthode à l'objet. Par exemple, si p est un objet de type Personne:

- p.afficher() est un message d'appel qui déclenche l'exécution de la méthode afficher() de l'objet p;
- p.delete() est un message d'appel qui déclenche l'exécution de la méthode delete() (méthode générique héritée) et dont le résultat est la destruction de l'objet p.

3.2. Polymorphisme et liaison dynamique

Un concept important de la programmation orientée objet est le polymorphisme des méthodes. Ce concept existait déjà dans les langages traditionnels. Il consiste à permettre de donner le même nom à plusieurs opérations différentes. Par exemple le signe + est utilisé pour l'addition de deux entiers et pour celle de deux réels, parfois aussi pour deux booléens ou pour concaténer deux chaînes. Le même nom, +, appellera l'exécution de codes différents selon le type des arguments. En orienté objets le choix de la bonne méthode est fait en fonction de la classe de l'objet appelé. L'association du nom de l'opérateur au "bon" code peut être faite:

- soit lors de la compilation. C'est le cas des langages de programmation typés tels que Pascal ou C;
- soit lors de l'exécution. C'est le cas des langages de programmation non typés tels que Smalltalk. Dans ce cas on parle de liaison dynamique ou tardive ("dynamic binding" ou "late binding").

Les SGBDO, même s'ils ont un langage de programmation typé, utilisent le mécanisme de liaison dynamique pour le choix du corps des méthodes dans le cas de redéfinition du code des méthodes dans une sous-classe. En effet, dans un graphe de généralisation/spécialisation, un même oid peut appartenir à plusieurs classes: une sur-classe et une, voire plusieurs, sous-classes. Si le corps d'une méthode est redéfini dans une ou plusieurs sous-classes et que l'objet appelé appartient à plusieurs classes et/ou sous-classes, il y a un problème de choix.

La plupart des SGBO ont suivi la solution simplificatrice des langages de programmation orientés objets qui consiste à restreindre les possibilités de la hiérarchie de généralisation/spécialisation de façon à ce que chaque entité réelle ne soit représentée que par un objet dans une seule classe (cf. principe d'instanciation unique vu au paragraphe 2.4). Pour cela, la hiérarchie de généralisation/spécialisation doit respecter les deux contraintes suivantes : 1/ elle a une racine unique, et 2/ les sous-classes non disjointes ont nécessairement leur intersection décrite explicitement par une autre sous-classe (cf. la classe AssistantDoctorant). Ainsi, pour tout oid, il existe toujours une classe unique qui est la plus spécialisée et qui contient l'oid. En conséquence pour résoudre le polymorphisme dû à la redéfinition des méthodes, ces SGBDO choisissent la méthode de la classe la plus spécialisée contenant l'oid.

En reprenant l'exemple de la méthode afficher() du paragraphe 2.4, supposons que la méthode afficher() de Personne affiche les nom, prénoms et adresse, et que la méthode afficher() de Enseignant affiche les nom, prénoms, adresse, tel et coursdonnés. Soient deux objets de la base de données: pierre de la classe Personne de numéro AVS 123123123, et annie de la classe Enseignant de numéro AVS 45454545. On peut alors écrire le programme suivant :

```
p : Personne; */ déclaration d'une variable de type Personne /*
p := SELECT p FROM p IN Personne WHERE AVS=123123123 ; */affecte l'oid pierre/*
p.afficher() ; */ affiche les nom, prénoms et adresse de pierre /*
p := SELECT p FROM p IN Personne WHERE AVS=45454545 ;
*/affecte l'oid annie. Cette instruction est valide grâce au principe de substituabilité.
En effet, le type d'annie (Enseignant) est un sous-type de celui de p (Personne) /*
p.afficher() ; */ affiche les nom, prénoms, adresse, tel et coursdonnés de annie /*
```

Avantages / inconvénients :

La liaison dynamique donne une grande souplesse et une grande puissance à la programmation. Les SGBDO permettent ainsi de combiner les avantages de la liaison dynamique et ceux des langages typés (détection plus facile des erreurs). Mais les mécanismes employés pour résoudre la liaison dynamique conduisent à limiter fortement les possibilités de la hiérarchie de généralisation / spécialisation, notamment la multi-instanciation (la possibilité de décrire une entité réelle par plusieurs objets de même oid appartenant à des classes différentes qui ne sont pas toujours sur/sousclasse l'une de l'autre).

4. Interface navigationnelle de manipulation des données

L'interface naturelle des SGBDO est procédurale et navigationnelle. Les utilisateurs écrivent des programmes dans lesquels les accès et mises à jour des objets de la base se font par envoi de messages aux objets (appels de méthodes). Ce type d'interface possède un double avantage:

- les utilisateurs disposent de la puissance d'expression d'un langage de programmation, ce qui n'est pas le cas lors de l'emploi d'un LMD type SQL seul;
- le problème d'incompatibilité qui existe en relationnel entre les LMD et les langages de programmation est réduit au minimum: les mêmes structures de données peuvent être utilisées dans le langage de programmation et dans les méthodes des classes, le même type d'interface que celui des langages de programmation (impératif, un élément à la fois) peut être employé pour les méthodes.

Les méthodes définies par les concepteurs de bases de données utilisent dans leur code les méthodes prédéfinies des classes prédéfinies. Ces méthodes prédéfinies permettent d'effectuer des manipulations élémentaires sur la base de données, telles que la création d'un objet, le stockage d'un objet, le passage d'un objet composé à un de ses composants, la recherche dans un objet de type collection d'un élément de cette collection.... Nous présentons ci-dessous, à titre d'exemple, la hiérarchie des classes d'ODMG et quelques unes de ses méthodes prédéfinies.

Hiérarchie des classes prédéfinies d'ODMG:

```
Denotable_Object
Object
Atomic_Object
Type
Iterator
…
Structured_Object
Collection(<T>)
Set(<T>)
List(<T>)
Bag(<T>)
Array(<T>)
Structure<e1:T1; …; en:Tn>
Literal
Atomic_Literal
Integer
Float
Boolean
String
Structured_Literal
Immutable_Structure<e1:T1; …; en:Tn>
Date
Time
…
Immutable_Collection
Enumeration
…
```

L'indentation représente les liens de généralisation / spécialisation. Par exemple, Denotable_Object est la racine du graphe et a pour sous-classes Object et Literal. Les classes créées par le concepteur de bases de données sont automatiquement sous-classes de Object.

Méthodes associées à la classe Object:

Object est la racine de la hérarchie des classes définies par les utilisateurs.

- create([nom_attribut:valeur, …]) → o:Oid
  création d'un objet avec initialisation éventuelle de sa valeur et passage en réponse de l'oid de l'objet créé.
- delete()
  suppression de l'objet
- same_as(o:Oid) → b:Boolean
  test si l'objet appelé et l'objet passé en paramètre sont identiques (test d'identité).

Exemples de méthodes associées à la classe Collection<T>:

- create(<T>) → c:Collection<T> */ méthode redéfinie/*

création d'une collection dont les éléments seront de type T. Pratiquement, on ne peut pas créer de collection générique, mais une instance d'un de ses sous-types.

- insert_element(o:Denotable_Object)
- remove_element(o:Denotable_Object)
- select_element(predicate:String) → element:T
- select (predicate:String) → c:Collection<T>

Exemples de méthodes associées à la classe Set<T>:

- union(s2:Set<T>) → s3:Set<T>
- intersection(s2:Set<T>) → s3:Set<T>
- difference(s2:Set<T>) → s3:Set<T>
- is_subset?(s2:Set<T>) → b:Boolean

Exemples de méthodes associées à la classe List<T>:

- insert_element_before(o:T, position:Integer)
- insert_first_element(o:T)
- retrieve_element_at(position:Integer) → element:T
- ….

Exemples de méthodes associées à la classe Structure<e1:T1, … , en:Tn>:

- create(<nomatt1:T1, … , nomattn:Tn>) → Structure<nomatt1:T1, … , nomattn:Tn>
- get_element_value(nomatt) → value:Denotable_Object
- set_element_value(nomatt, value: Denotable_Object)
- clear_element_value(nomatt)
- ….

Avantages / inconvénients :

Ce type d'interface navigationnelle est très puissant, mais il est nettement moins efficace quant à la productivité des programmeurs que les langages déclaratifs tels SQL auxquels les utilisateurs de base de données sont habitués. En effet un langage déclaratif est plus simple à utiliser qu'un langage de programmation, et l'optimisation des requêtes sont effectuée par le système et non par l'utilisateur.

Aussi les SGBDO actuels offrent-ils aujourd’hui des LMD orientés objets plus ou moins déclaratifs et plus ou moins proches de SQL. Par exemple, ODMG propose OQL, un langage de requêtes purement objet, et la norme SQL3, extension de SQL au relationnel-objet, a été définie.

Série d'exercices sur les BDOOs

Exercice n° 01.

Un site Web peut être vu comme un objet complexe. Pour cela, on considère l'ensemble de domaines de base ci-dessous.

- D_{html}=*html* : le domaine des fichiers html.
- D_{image}=*image*$_i$: le domaine des fichiers images.
- D_{URL}=*url*$_i$: le domaine des URL.

Une *page Web* peut être vue comme un objet ayant 3 attributs : une URL, un fichier html et un ensemble d'images associées.
Finalement, un *site Web* est constitué d'une page Web d'accès et d'un ensemble de pages Web secondaires.
Donner le type des objets *page Web* et *site Web*.

Exercice n° 02.

L'information contenue dans un lecteur de fichiers musicaux est un objet structuré comme suit :

- L'objet est composé d'un ensemble d'albums.
- Chaque album est associé à un chanteur et un ensemble de chansons.
- Un chanteur est caractérisé par un nom, un prénom et une date de naissance.
- Pour chaque chanson d'un album, on conserve le nom, la position au sein de l'album et une séquence de bits correspondant au contenu musical.

Donner le type de cet objet et préciser les domaines des valeurs de base.

Exercice n° 03.

Un gérant de salons-lavoir désire conserver dans une base de données orientée objet les informations concernant les différents modèles de machine à laver disponible sur le marché.
Définir une base de données orientée-objet satisfaisant les propriétés suivantes, i.e. les types des différents objets figurant dans la base de données :
1. On associe à chaque modèle de machine un nom de modèle, un fabricant, un lieu de fabrication et la liste des programmes de lavage disponible ainsi que le prix d'achat et le nombre de mois de garantie.
2. Les fabricants sont caractérisés par nom, un pays, le nombre d'employés et l'année de création.
3. Un programme de lavage contient au maximum 5 phases, chacune des phases étant décrite par sa durée (en minutes), sa description, sa consommation d'eau et d'électricité ainsi que la vitesse de rotation du tambour durant la phase.

Exercice n°04

Un centre de recherche veut stocker dans ***une base orientée objets*** tous les documents qu'il produit: rapports, articles scientifiques, livres… Un document est caractérisé par son type (rapport, article scientifique ou livre). Il est composé de:

- Son nom (identifiant),
- Un entête comprenant le titre, les auteurs (nom, affiliation), les mots clés, la date de création,
- Un résumé (une suite de lignes de texte, 15 au maximum),
- Le corps du document,
- La bibliographie du document.

Certains documents étant en cours d'écriture, certaines parties peuvent ne pas être terminées ou être complètement absentes, par exemple, le résumé et les mots clés n'existent pas, le corps n'est pas fini…

Le corps est composé d'une suite de paragraphes. Chaque paragraphe a un numéro, un titre et un contenu. Le contenu d'un paragraphe est constitué d'une suite d'alineas qui sont soit du texte, soit un sous-paragraphe. La description d'un sous-paragraphe est identique à celle d'un paragraphe. Le texte peut contenir des figures.

Une figure est composée d'un numéro, d'un titre, d'une image (qui peut être un dessin, un graphique ou une photo), et éventuellement d'une légende et d'une courte description.

Les lignes de texte peuvent contenir des renvois à des références de la bibliographie, à d'autres paragraphes et à des figures du document.

La bibliographie est composée d'une suite de références à des ouvrages, sous la forme: nom de l'auteur (ou des auteurs), titre, date et lieu de parution. Ces ouvrages peuvent être eux-mêmes des documents de la base.

Un même paragraphe, une même figure et une même référence bibliographique peuvent être communs à plusieurs documents de la base.

Proposer une Base de données Orientée Objets répondant à des questions du type :

- lister les titres des documents qui contiennent une référence bibliographique à tel auteur (ou qui ont tel mot clé),
- lister les noms des documents qui contiennent telle figure (ou tel paragraphe),
- lister le Xième paragraphe (ou la Xième figure) de tel document,
- imprimer tout le document de tel nom,
- lister les titres (ou les résumés) des documents écrits par tel auteur,
- lister les dessins employés par tel auteur,
- lister les images correspondant à telle description…

Chapitre 3 : Le relationnel étendu

1. Introduction

Le modèle relationnel a fait ses preuves et il a démontré, surtout depuis les années 80, qu'il avait des points forts indiscutables. Il a aussi un certain nombre de points faibles, auxquels le modèle objet répond assez bien, d'où l'intérêt d'une intégration en douceur des deux modèles.

Cette intégration permet aussi aux utilisateurs de ne pas devoir abandonner deux décennies de développement de leurs applications sur des bases de données relationnelles et facilite donc la mise en place et l'adoption de cette nouvelle technologie.

1.1 Points forts du modèle relationnel

Le modèle relationnel est complètement adapté aux applications de gestion grâce à son système de modélisation formé de tables dont les colonnes prennent des valeurs alphanumériques. Les concepts du modèle relationnel (tables, enregistrements, relations, champs, etc…) sont simples et aisément compréhensibles par n'importe qui. De plus, la longue expérience dont bénéficient maintenant les SGBD relationnels a permis de les optimiser et donc de les rendre très performants.

Un autre point fort du modèle relationnel est l'existence de SQL, un langage standardisé de création et de manipulation des bases de données. Le langage SQL est bien adapté aux architectures client-serveur, il intègre la gestion des transactions, indispensable lors d'accès concurrents aux données ainsi que pour leur sécurité.

1.2 Points faibles du modèle relationnel

Le modèle relationnel, dont la conception est bien antérieure à l'avènement du multimédia et d'Internet, présente certaines faiblesses qui le pénalisent lourdement aujourd'hui:

1.2.1. Pas de support des objets complexes

Cette contrainte, imposée par le respect de la première forme normale de Boyce- Codd, est inadaptée à la gestion des objets complexes, de plus en plus nombreux dans les applications actuelles.
L'introduction des types de données binaires comme les CLOB (Character Large Object) ou les BLOB (Binary Large Object) ne résoud pas le problème, car la nature du stockage de ces types dans les SGBD relationnels ne permet pas des opérations comme la comparaison de deux LOB. Ils ne sont pas structurés, ce qui impose une lecture séquentielle de l'objet et interdit, par exemple, l'accès direct à un chapitre d'un livre, stocké dans une colonne de table.

1.2.2. Non intégration des traitements

Le modèle relationnel sépare complètement les données des traitements, ce qui rend impossible l'encapsulation des données.
L'encapsulation des données permet de cacher certaines informations à l'utilisateur et de les rendre accessibles uniquement par des opérations ou fonctions manipulant ces informations, permettant un contrôle de la consistance des données par le programme.

1.2.3. Absence de pointeurs visibles

Les pointeurs permettent un parcours très rapide de grandes quantités de données, avec des gains importants par rapport aux jointures par valeurs utilisées avec le modèle relationnel. Il est aussi possible de les utiliser pour partager des références sur des objets volumineux et ainsi économiser de l'espace de stockage.
La majorité des critiques liées aux pointeurs sont justifiées, mais il serait dommage de se priver de leurs avantages au niveau des SGBD, uniquement à cause de ces critiques, d'autant plus que les problèmes qui leurs sont liés sont connus, et donc évitables.

=> Ces limites du modèle relationnel ont conduit à de nombreuses recherches sur de nouveaux modèles de bases de données et, actuellement, deux types d'alternatives sont proposés : les ***modèles relationnels étendus*** et ***les modèles orientés objets***.
Dans les deux cas, l'enrichissement du modèle relationnel est fait par l'approche "***objet***", tirée des concepts des nouveaux langages de programmation dits "orientés objets".

Principe du relationnel étendu : le modèle relationnel étendu permet d'ajouter des fonctionnalités de type objets aux SGBD relationnels classiques.

2. Le Relationnel étendu (Relationnel Objet (RO))

Le modèle Relationnel-Objet (RO) reprend le modèle relationnel en ajoutant quelques notions qui comblent les plus grosses lacunes du modèle relationnel.

Pourquoi étendre le modèle relationnel ?

- La reconstitution d'objets complexes éclatés sur plusieurs tables relationnelles est coûteuse car elle occasionne de nombreuses jointures
- L'échappement aux éclatements-jointures, le RO réhabilite les références qui permettent d'implanter des structures complexes
- La représentation des attributs multi-valués (tableaux, ensembles ou listes).
- L'utilisation de références facilite aussi l'utilisation des données très volumineuses du multimédia en permettant leur partage simplement et à moindre coût (sans jointure)

Les nouvelles possibilités offertes aux utilisateurs sont les suivantes :

- Définir de nouveaux domaines à structure complexe, appelés types,
- Associer à chaque type des méthodes,
- Créer des hiérarchies de types
- Créer des objets qui sont composés d'une valeur structurée et d'un OID,

- Etablir des liens de composition par des attributs référence qui contiennent l'OID de l'objet composant,
- Créer des tables contenant soit des tuples normaux (en première forme normale), soit des tuples en non première forme normale (des valeurs structurées), soit des objets.

3. Types

Les utilisateurs ont à leur disposition les domaines usuels de SQL tels que : *VarChar*, *Number*, *Date*… Ils peuvent aussi définir de nouveaux domaines (appelés ***types***) propres à leur base de données grâce au constructeur **TYPE** dont il existe plusieurs formes. Les deux premières (***VARRAY*** et ***nested TABLE***) permettent d'avoir des attributs multi-valués. La troisième forme (***OBJECT***) a une double fonction: elle permet de créer des valeurs structurées (notamment pour les attributs complexes), et de créer des objets, c'est à dire des couples ***<valeur structurée, oid>***.

3.1. Les types VARRAY

Ce type permet de créer des attributs multi-valués sous la forme de tableaux à une dimension. L'instruction suivante:

CREATE TYPE nom-type AS VARRAY (nb-max) OF nom-type2 ;

déclare un nouveau type de nom "nom-type" dont chaque élément est un tableau à une dimension (un vecteur) de taille maximale "nb-max", et dont les éléments sont tous du type "nom-type2".
Le type nom-type2 peut être n'importe quel type, prédéfini de SQL (VarChar, Date, Number…) ou défini par l'utilisateur.

3.2. Les types table emboîtée ("nested TABLE")

Ce type permet de créer des multi-valués sous la forme de tables relationnelles qui vont être incluses dans les autres tables où elles serviront de domaine à un attribut. L'instruction suivante:

CREATE TYPE nom-type AS TABLE OF nom-type2 ;

déclare un nouveau type de nom "nom-type" dont chaque élément est une table relationnelle dont les tuples sont tous du type "nom-type2". Le type nom-type2 peut être n'importe quel type, prédéfini de SQL ou défini par l'utilisateur.

Lors de l'emploi de ce type, nom-type, comme domaine d'un attribut d'une autre table, par exemple de l'attribut A de la table T, le système Oracle crée physiquement une table annexe (la nested TABLE) dans laquelle il stockera les tuples (les valeurs) de l'attribut A. L'utilisateur doit, lors de la création de la table T, définir le nom de cette table annexe. Pour cela, l'instruction ***CREATE TABLE*** contient une clause supplémentaire afin que l'utilisateur puisse nommer cette table annexe :

CREATE TABLE ….
NESTED TABLE nom-attribut-de-type-table-emboîtée STORE AS nom-table-annexe ;

Accès aux valeurs multi-valuées (Types VARRAY et TABLE)

- Opérateurs usuels de SQL
    - condition élémentaire : élément IN collection
    - condition élémentaire : EXIST collection

- Nouveau : déclaration d'une variable sur une collection
-SELECT … FROM nom-table t , TABLE (t.attribut-multivalué) v …
=> Accès et mise à jour soit de toute la collection, soit d'un de ses éléments

- Exemples

```
CREATE TYPE Tprénoms AS VARRAY(4) OF VARCHAR(20);
CREATE TABLE Personne ( AVS CHAR(11) , nom VARCHAR(20) , prénoms
Tprénoms , adr Tadresse )
```

-Personnes dont un prénom est Annie :

```
SELECT p FROM Personne p, TABLE(p.prénoms) pr
WHERE pr.COLUMN_VALUE = 'Annie'
SELECT p FROM Personne p
WHERE 'Annie' IN (SELECT * FROM TABLE (p.prénoms))
```

3.3. Les types OBJECT

Ce type crée des valeurs structurées et/ou des objets. L'instruction suivante:

CREATE TYPE nom-type AS OBJECT (nom-attribut1 nom-type1 , nom-attribut2 nom-type2 , … , nom-attributn nom-typen)

déclare un nouveau type de nom "nom-type" dont chaque élément est un objet structuré comportant les attributs nom-attribut1 (de domaine nom-type1), nom-attribut2 (de domaine nom-type2),… et nom-attributn (de domaine nom-typen). Les types ***nom-typei*** peuvent être n'importe quel type, prédéfini de SQL ou défini par l'utilisateur.

Cette instruction permet aussi de déclarer un jeu de méthodes associées aux objets de ce type (en préfixant la déclaration de la méthode du mot clé MEMBER). Chacun de ces types, VARRAY, TABLE emboîtée et OBJECT, ne permet de définir que des structures contenant un seul constructeur:

Pour créer une structure contenant plusieurs constructeurs, il faut définir un type intermédiaire par constructeur. Par exemple, pour représenter une personne avec ses enfants, et pour chaque enfant ses prénoms, on devra créer les types suivants:

```
CREATE TYPE T-Personne AS OBJECT ( nom VARCHAR(20) , prénoms T-
ListePrénoms , enfants T-ListeEnfants )
CREATE TYPE T-ListePrénoms AS VARRAY OF VARCHAR(20) ;
CREATE TYPE T-Enfant AS OBJECT( prénoms T-ListePrénoms, sexe CHAR ,dateNais
DATE )
CREATE TYPE T-ListeEnfants AS VARRAY OF T-Enfant ;
```

Accès aux valeurs complexes (type OBJECT)

- Accès aux composants via la notation pointée

CREATE TYPE Tadresse AS OBJECT (rue VARCHAR(20), numéro VARCHAR(4), localité VARCHAR(20), NPA CHAR(4)) ;
CREATE TABLE Personne (AVS CHAR(11) , nom VARCHAR(20), prénoms Tprénoms , adr Tadresse)
SELECT p.adresse.localité FROM Personne p WHERE p.AVS = 12345123451

4. Tables

Les tables sont les récipients qui stockent les valeurs ou les objets. Les tables peuvent être :

- soit des tables du relationnel classique (contenant des valeurs de type tuple en première forme normale).
- soit des tables à structure complexe contenant des valeurs structurées (en non première forme normale).
- soit enfin des tables contenant des objets (qui eux-mêmes peuvent être structurés en non première forme normale).

4.1. Création d'une table relationnelle classique

On emploie l'instruction CREATE TABLE du relationnel classique:

CREATE TABLE nom-table (nom-attribut1 nom-type1 , nom-attribut2 nom-type2 , .. , nomattributn nom-typen)

Cette table contiendra des valeurs de type tuple en première forme normale.

4.2. Création d'une table relationnelle en non première forme normale

On emploie l'instruction CREATE TABLE du relationnel classique mais en utilisant un ou plusieurs types créés précédemment par un constructeur ***CREATE TYPE***.

CREATE TABLE nom-table (nom-attribut1 nom-type1 ,nom-attribut2 nom-type2 , … , nomattributn nom-typen)

où au moins un des ***nom-typei*** est le nom d'un type construit (VARRAY, TABLE emboîtée, ou OBJECT).

4.3. Création d'une table d'objets

On emploie l'instruction:

CREATE TABLE nom-table OF nom-type

Où *nom-type* est le nom d'un type OBJECT (qui doit avoir été créé auparavant). Le format de la table est celui de ***nom-type***.
Cette table contiendra des objets, identifiés par leurs OIDs et structurés selon le type "***nom-type***".
On peut associer aux tables quelle que soit leur sorte les contraintes usuelles de SQL :

-PRIMARY KEY (nom-col*)
-UNIQUE (nom-col*)
-FOREIGN KEY (nom-col*) REFERENCES nom-table [(nom-col*)] [action]
-CHECK (condition)

5. Identité et attributs-référence

Tout ce qui est de type OBJECT possède une identité (un ***OID***) et peut donc être référencé par un lien de composition. Pour référencer un objet, on utilise la clause "REF nom-type", comme dans l'exemple ci-dessous:

CREATE TYPE T-Personne AS OBJECT
(AVS NUMBER, nom VARCHAR(18) , prénom VARCHAR(18) , conjoint REF T-Personne)

Attention, le type suivant ne contient pas d'attribut référence:

CREATE TYPE T-Personne2 AS OBJECT
(AVS NUMBER , nom VARCHAR(18) , prénom VARCHAR(18) , voiture T-Voiture...)

Impact sur le langage de manipulation des données

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

5.1. REF (objet): La clause REF(variable-objet) donne en résultat l'OID de l'objet. C'est souvent utile pour les mises à jour. Par exemple, soit la table d'objets LesPersonnes:

CREATE TABLE LesPersonnes OF T-Personne ;

L'instruction:

INSERT INTO LesPersonnes (AVS, nom, prénom, conjoint) VALUES
(17924457911, 'Rochat', 'Philippe', (SELECT REF(p) FROM LesPersonnes p
WHERE p.nom='Muller' AND p.prénom='Annie')) ;

insère un objet (Philippe Rochat) et initialise son lien de référence conjoint (sur Annie Muller).

5.2. VALUE (objet): La clause VALUE (variable-objet) a l'effet complémentaire; elle fournit en résultat la valeur structurée de l'objet (avec le nom de son type).

5.3. DEREF (oid): Dans le cas où une requête fournit un oid, on peut passer à la valeur de l'objet par l'emploi explicite de la clause **DEREF(oid)**.
Par exemple, l'instruction :

SELECT p.nom, p.conjoint FROM LesPersonnes p;

donnera le nom des personnes avec l'OID de leur conjoint. Et l'instruction :

SELECT p.nom, DEREF(p.conjoint) FROM LesPersonnes p;

donnera pour chaque personne *p*, son nom, et la "valeur" de son conjoint, à savoir : son numéro AVS, son nom, son prénom et l'OID de son conjoint (c'est à dire l'OID de la personne *p* elle-même).
Un nouveau type de condition permet de tester si un attribut référence a été (ou non) initialisé :

attribut-référence IS DANGLING

Cette condition est vraie si et seulement si l'attribut référence n'a pas été initialisé.

Exemple:

SELECT p.nom FROM LesPersonnes p WHERE p.conjoint IS DANGLING;

6. Méthodes

Chaque type OBJECT peut avoir des méthodes :

CREATE TYPE nom-type AS OBJECT (déclaration des attributs , déclaration des signatures des méthodes)

6.1. Signature d'une méthode

MEMBER FUNCTION nom-méthode (nom-paramêtre1 [IN / OUT] type1 , ….) RETURN type

MEMBER PROCEDURE nom-méthode (nom-paramêtre1 [IN / OUT] type1 , ….)

6.2. Corps d'un type OBJECT

Contient le code de ses méthodes. Il peut contenir des :
- instructions PL/SQL (ou JAVA)
- instructions SQL
- appels de méthodes

Exemple :

CREATE TYPE BODY nom-type AS
MEMBER FUNCTION / PROCEDURE nom-méthode1 (nom-paramêtre11 [IN / OUT] type11 , ….)
BEGIN code de-la-méthode1 END
MEMBER FUNCTION / PROCEDURE nom-méthode2 (nom-paramêtre21 [IN / OUT] type21 , ….)
BEGIN code de-la-méthode2 END ;

7. Hiérarchie de types OBJECT

Il est possible de créer des hiérarchies de types OBJECT, puis, d'utiliser ces types et leurs sous-types lors des créations de tables. Pour créer un type pour lequel on va déclarer des sous types, il faut ajouter la clause NOT FINAL à son instruction CREATE TYPE; sinon, par défaut le type créé sera considéré par Oracle comme "final", c.à.d. sans sous-type.

Pour créer un sous-type d'un type non final, il faut rajouter la clause UNDER nom-sur-type. Le sous-type héritera alors des attributs et méthodes du sur-type. Attention, l'héritage multiple est interdit : un type ne peut avoir qu'un seul sur-type.

Exemple :

CREATE TYPE T-Personne AS OBJECT (AVS NUMBER , nom VARCHAR(18) , prénom VARCHAR(18), adresse VARCHAR(200))
NOT FINAL
CREATE TYPE T-Etudiant UNDER T-Personne (faculté VARCHAR(18) ,
cycle VARCHAR(18))

Le type T-Etudiant est un type OBJECT, qui contient 6 attributs, AVS, nom, prénom, adresse, faculté et cycle.
Dans le cas d'une table d'objets de type T-Personne et dans celui d'un attribut de type TPersonne, suivant le principe de substituabilité, l'utilisateur peut y insérer des valeurs de type T-Personne ou de type T-Etudiant.

Exemples :

CREATE TABLE LesPersonnes OF T-Personne ; */* création d'une table d'objets */*
INSERT INTO LesPersonnes VALUES (T-Person (11111111, 'TTT', 'SSSS', 'Rue des oiseaux, 1, MMMM') */* création et insertion d'un objet de type T-Personne */*
INSERT INTO LesPersonnes VALUES (T-Etudiant (22222222, 'Muller', 'Annie', 'Rue du marché, 22, RRRR', 'FS', 'master') */* création et insertion d'un objet de type T-Etudiant */*

CREATE TABLE LivresEmpruntés (livre VARCHAR(20), emprunteur T-Personne) ;
/ création d'une table de tuples structurés (non 1NF) */*
INSERT INTO LivresEmpruntés VALUES ('SQL3', T-Person (11111111, 'TTT', 'SSSS', 'Rue des oiseaux, 1, MMMM')*/* insertion d'un tuple structuré dont l'attribut emprunteur est de type T-Personne */*
INSERT INTO LivresEmpruntés VALUES ('Les Bases de Données', T-Etudiant (22222222, 'Muller', 'Annie', 'Rue du marché, 22, RRRR', 'FS', 'master')*/* insertion d'un tuple structuré dont l'attribut emprunteur est de type T-Etudiant */*

7.1. Nouvelle condition élémentaire

VALUE(objet) **IS OF** nom-sous-type: condition vraie ssi l'objet est du type "nom-soustype"

Exemple :

SELECT p FROM LesPersonnes p
WHERE VALUE(p) IS OF Tétudiant
=> rend les objets de LesPersonnes qui sont de type Tétudiant

Série d'exercices sur le relationnel étendu

Exercice 1

Soit le schéma relationnel de base de données suivant :

APPARTEMENT (Num_Appart, Adresse, Type, Nb_Pièces)
CLIENT (Num_Client, Nom, Prénom, Adresse, Tél, Type_Appart_Préféré, Loyer_Max)
PHOTO (Num_Photo, Num_Appart, Date, Commentaire, Photo)

1. Indiquer les problèmes ou insuffisances du modèle relationnel en termes de :
   pouvoir d'expression du modèle,
   pouvoir d'expression des requêtes,
2. Définir le domaine de définition des attributs, et notamment les attributs "non standards".
3. Donner des exemples de requêtes mettant en évidence les insuffisances du modèle relationnel.
4. Proposer un schéma de type relationnel-objet.

Exemples de requêtes
Quels sont les clients habitant en dehors du département de Charente-Maritime ?
Quels sont les types d'appartements préférés de M. Martin ?
Quels sont les appartements avec des photos des chambres ?

Exercice 2

On souhaite concevoir une base de données décrivant des pièces et leurs composants. Une pièce P est composée de plusieurs composants C, chacun des composants pouvant apparaître plusieurs fois dans une pièce. Une pièce peut elle-même être composante d'une autre pièce. Par exemple, un moteur est un composant d'une voiture, et les bougies sont des composants du moteur.

1. On demande de modéliser ces informations sous forme d'un schéma UML ou entité association, puis de donner le schéma relationnel.
Les schémas demandés doivent éviter les redondances et permettre de répondre efficacement au moins aux deux questions suivantes :
   Q1 : Etant donné une pièce, retrouver ses composants directs (c'est-à-dire sans récursivité), et le nombre d'exemplaires de chaque composant.
   Q2 : Étant donnée une pièce, retrouver toutes les pièces dont elle est un composant.
2. Expliquer comment on obtient ces réponses dans le cas relationnel.
3. Que pourrait apporter une représentation en relationnel objet ? (Donner les scripts de créations, d'insertion et d'extraction de données).

Exercice 3

Soit une base de données concernant des bâtiments de guerre. On souhaite en exprimer le schéma en représentation de type relationnel-objet avec des types abstraits de données.

Chaque bateau de guerre a les informations suivantes qui lui sont associées : son nom, son tirant en tonnes, son type (destroyer, canonnier,...).

Il y a certains bâtiments qui ont des spécificités : les canonniers sont des bateaux qui transportent de grands canons, tels les croiseurs. Pour ce type de bâtiments, on souhaite enregistrer le nombre et le calibre des principaux canons.

Les porteurs, pour lesquels nous enregistrons la longueur de la piste d'envol et les ensembles de groupes aériens (escadrilles) qui lui sont assignés.

Les sous-marins, pour lesquels la profondeur de la plongée maximum est connue. Aucun sous-marin n'est canonnier, ni même porteur!

Les cuirassés sont porteurs et canonniers en même temps, et donc on enregistre pour eux les données pour ces deux types de bâtiments.

1. Donner le diagramme E/R et le schéma relationnel étendu.
2. Montrer comment on pourrait représenter le cuirassé “Ise”, qui avait un tirant de 36000 tonnes, des canons de 20 cm, une piste de 700 m et les groupe d'envols 1 et 2.

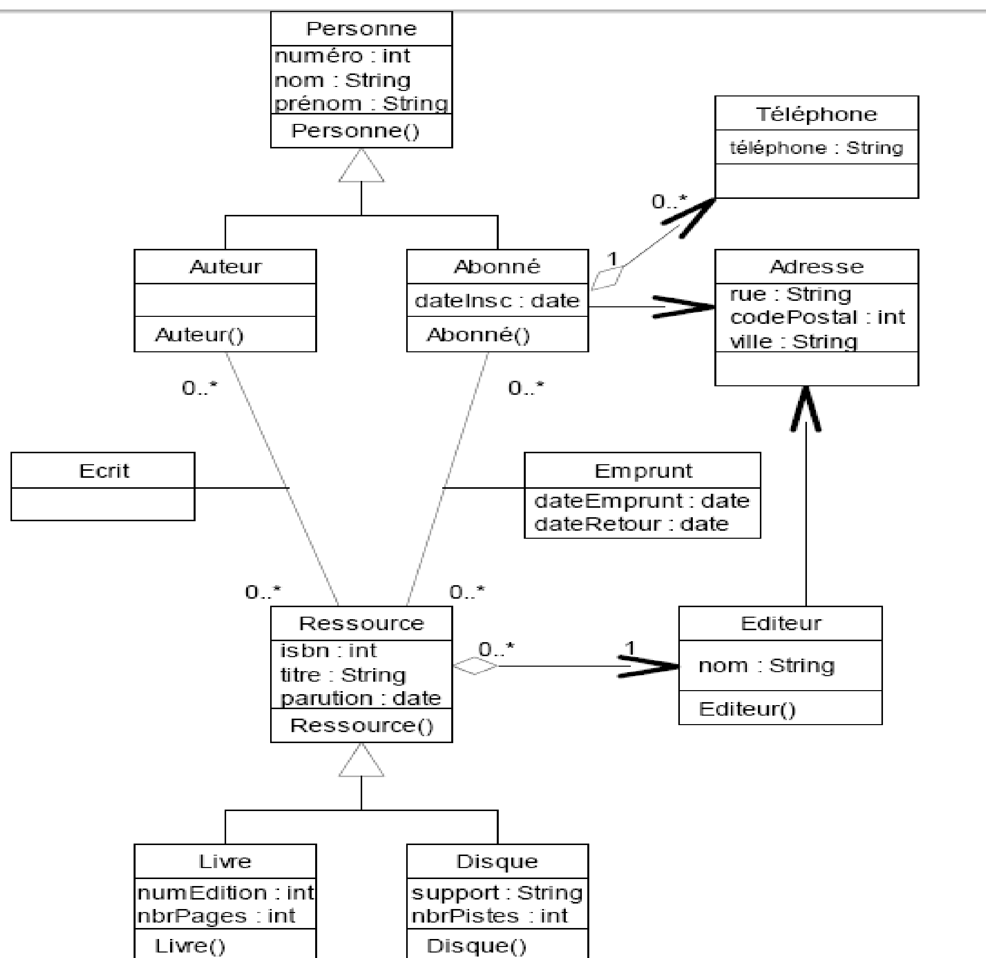
Exercice 4

Soit une base de données concernant des bâtiments de guerre. On souhaite en exprimer le schéma en représentation de type relationnel-objet avec des types abstraits de données.
Chaque bateau de guerre a les informations suivantes qui lui sont associées : son nom, son tirant en tonnes, son type (destroyer, canonnier,...).
Il y a certains bâtiments qui ont des spécificités : les canonniers sont des bateaux qui transportent de grands canons, tels les croiseurs. Pour ce type de bâtiments, on souhaite enregistrer le nombre et le calibre des principaux canons.
Les porteurs, pour lesquels nous enregistrons la longueur de la piste d'envol et les ensembles de groupes aériens (escadrilles) qui lui sont assignés.
Les sous-marins, pour lesquels la profondeur de la plongée maximum est connue. Aucun sous-marin n'est canonnier, ni même porteur!
Les cuirassés sont porteurs et canonniers en même temps, et donc on enregistre pour eux les données pour ces deux types de bâtiments.
1. Donner le diagramme E/R et le schéma relationnel étendu.
2. Montrer comment on pourrait représenter le cuirassé “Ise”, qui avait un tirant de 36000 tonnes, des canons de 20 cm, une piste de 700 m et les groupe d'envols 1 et 2.

Exercice 5

Les ressources empruntées peuvent être des disques ou des livres; toutes sont identifiées par leur numéro ISBN et, possèdent un titre et une date de parution. Chaque ressource est éditée par un éditeur, dont on stocke le nom et l'adresse, et est écrite par un ou plusieurs auteurs. Les livres ont un numéro d'édition et un nombre de pages. Les disques peuvent être de support « vinyle », « CD » ou « DVD » et possèdent un nombre de pistes. On stocke pour chaque abonné et chaque auteur son numéro identifiant, son nom et son prénom. On mémorise également la date d'inscription des abonnés, ainsi que leur(s) numéro(s) de téléphone et leur adresse postale afin de pouvoir les contacter. Lorsqu'un abonné emprunte une ressource, on mémorise la date d'emprunt et la date de restitution. Les emprunts d'une même ressource par un même abonné sont distingués par la date d'emprunt.

Le schéma UML de la base de données est représenté comme suit :



Question 1 : On souhaite dans un premier temps stocker les données en utilisant un SGBD relationnel-objet. Ecrivez les requêtes de créations de tables de la base de données relationnel-objet correspondant aux objets Abonné, Livre, Disque et Emprunt du schéma UML.

Question 2 : Pour chacune des tables correspondant aux objets Abonné, Livre, Disque et Emprunt de la base de données, donnez un exemple de requêtes d'insertion d'une instance. On suppose que les tuples correspondants aux auteurs et éditeurs du livre et du disque que vous insérez sont déjà enregistrés dans la base de données.

Question 3 : On souhaite maintenant proposer une solution pour stocker cette base de données dans un SGBD relationnel et non pas relationnel-objet. Proposez un schéma relationnel pour cette base de données

Chapitre 4 : Les BDs déductives

1. Concept et Motivation

L'idée est de coupler une base de données à un ensemble de règles logiques qui permettent d'en déduire de l'information.

Cela pourrait se faire dans le contexte d'un langage de programmation, mais on souhaite un couplage plus direct qui permet de manipuler les données au niveau de la base de données.

Les questions qui se posent sont les suivantes.

- Quelles est la forme des règles que l'on souhaite utiliser ? Comment les interpréter ?
- Les requêtes expressibles en SQL peuvent-elles être exprimées par des règles ?
- Les règles peuvent-elles exprimer plus ?

2. Définitions préliminaires

Une base de données déductive (BDD) est constituée :

- d'un ensemble de prédicats (relations), dits de base ou extensionnels, dont l'extension est conservée explicitement dans la base de données : la base de données *extensionnelle*.
- d'un ensemble de prédicats (relations), dits dérivés ou intentionnels, dont l'extension est définie par des règles déductives : la base de données *intentionnelle*.

3. Exemple :

Base de données *extensionnelle* : Prédicat parent à 2 arguments (ou relation parent à 2 attributs)

parent	
anna	bob
bob	chris
bob	dan
dan	eric

Base de données *intentionnelle* : Prédicat *grandparent* à 2 arguments (ou relation *grandparent* à 2 attributs)

$$grandparent(X,Y) \leftarrow parent(X,Z) \text{ AND } parent(Z,Y)$$

4. Bases de données déductives s'appuyant sur des SGBDs relationnels

Les SGBD relationnels ont servi de support aux premières bases de données déductives. Deux familles de systèmes ont été développées : les BD déductives s'appuyant sur *le cadre formel des langages du 1er ordre* et celles qui utilisent le formalisme général *des règles de production*.

4.1. Bases de données déductives et langages du 1er ordre

4.1.1 Langages du 1er ordre

Comme tous les langages formels, les langages du 1er ordre reposent sur des règles de construction syntaxique de formules bien formées du langage auxquelles on associe une interprétation.

Les règles de construction syntaxiques s'appuient sur :

- un alphabet composé de :

— constantes : a, b, c... ;
— variables : X, Y, Z... ;
— symboles de prédicat : P, Q, R... ;
— fonctions : f, g, h... ;
— les parenthèses ;
— les connecteurs logiques usuels : négation (¬), conjonction (∧), disjonction (∨), proposition conditionnelle (→), équivalence (↔) ;
— les quantificateurs universel (∀) et existentiel (∃) ;
— des règles de construction de formules du langage composé de :

• termes : une constante ou une variable constitue un terme. Si t1 , t2 , ..., tn sont des termes f (t1 , t2 , ..., tn) est un terme,
• formules atomiques : si t1 , t2 , ... tn sont des termes et P est un symbole de prédicat n-aire, alors P (t1 , t2 , ... tn) est une formule atomique ;

— des formules bien formées (fbf) :

• toute formule atomique est une fbf,
• si w1 et w2 sont des fbf, alors :

¬w1
w1 ∨ w2
w1 ∧ w2
w1 → w2
w1 ↔ w2
∃x w1 (x)
∀ x w1 (x)

Sont des fbf

Exemple :

∀ X (Facture (X)) → ∃ Z (Commande (Z, X)) est une fbf construite sur les symboles de prédicat Facture et Commande. Il s'agit d'une construction syntaxique qui peut s'interpréter : « À toute facture X correspond une commande Z ».

Les fbf peuvent subir une suite de transformations pour aboutir à des représentations équivalentes dont la manipulation est plus facile (en particulier la forme clausale qui est celle utilisée dans les langages de type Prolog) :

— la forme normale Prenex correspond à un regroupement de tous les quantificateurs en tête de la formule :

∀ X ∃ Z (Facture (X) → Commande(Z, X))

— la skolémisation permet de remplacer toutes les variables quantifiées existentiellement par une fonction f des variables quantifiées universellement qui les précèdent ; f est un symbole de Skolem d'arité k qui n'apparaît pas dans la formule initiale :

$$\forall X\ (\text{Facture}\ (X) \rightarrow \text{Commande}(f(X), X))$$

Il n'est donc plus nécessaire de faire figurer le quantificateur existentiel. Implicitement toutes les variables sont quantifiées universellement.

Remarque : la formule précédente peut aussi s'écrire :

$$\forall X\ (\neg \text{Facture}\ (X) \vee \text{Commande}\ (f\ (X), X))$$

— la forme clausale consiste à exprimer toute fbf sous forme d'une disjonction de formules atomiques positives ou négatives skolé misées :

$$\neg A1 \vee \neg A2 \ldots \vee \neg Am \vee B1 \vee B2 \ldots \vee Bn$$

qui peut aussi s'écrire :

$$A1 \wedge A2 \ldots \wedge Am \rightarrow B1 \vee B2 \ldots \vee Bn$$

Une telle expression constitue une clause.

Exemple :

$$\text{Parent}\ (X, Y) \rightarrow \text{Mère}\ (X, Y) \vee \text{Père}\ (X, Y)$$

On se limite la plupart du temps à l'expression de clauses ne contenant qu'un littéral positif en partie de droite (n = 1) : ce sont les clauses de Horn pour lesquelles il existe des mécanismes de preuves formelles. Le langage Prolog repose sur les clauses de Horn.
On peut considérer qu'un programme Prolog est analogue à un ensemble de clauses de Horn définies.

Dans la suite, nous nous intéresserons donc à des ensembles de clauses de Horn.

Exemple :

Père (X, Y) → Parent (X, Y)
Mère (X, Y) → Parent (X, Y)
Mère (X, Y) ∧ Parent (X, Z) → Gmère (X,Z)
Père (X, Y) ∧ Parent (Y, Z) → Gpère (X, Z)
Gmère (X, Y) → Gparent (X, Y)
Gpère (X, Y) → Gparent (X, Y)

4.1.2 Interprétation d'un ensemble de clauses

Les clauses n'étant qu'une construction syntaxique, nous donnons une signification à un ensemble de clauses en mettant en correspondance les éléments de ces clauses avec des éléments du monde réel appelé domaine d'interprétation : cette mise en correspondance s'appelle une interprétation. On choisit parmi toutes les interprétations celles qui rendent vraies toutes les clauses ; les éléments du monde réel satisfaisant ces clauses constituent un modèle pour ces clauses.

* **Interprétation**

Soit un ensemble de clauses. On se donne un domaine d'interprétation E non vide à partir duquel on va instancier les constantes et les variables des clauses. À chaque symbole Q de prédicat n-aire, on fait correspondre un nom de relation R de E^n.

Exemple : soit l'ensemble des clauses :
1. $P(X) \wedge Q(Y) \rightarrow R(X, Y)$
2. $\rightarrow P(a)$
3. $\rightarrow Q(c)$
4. $\rightarrow R(a, b)$
5. $\rightarrow R(b, c)$

Remarquons que les clauses 2 à 5 n'ont pas de partie gauche, elles constituent ce qu'on appelle des faits.
Soit le domaine d'interprétation constitué :
— des individus {jean, pierre, loto, tiercé} ;
— des relations :

a-gagné (jean),
a-gagné (pierre),
jeu (loto),
jeu (tiercé),
a-joué (jean, loto),
a-joué (jean, tiercé),
a-joué (pierre, loto).

La mise en correspondance suivante constitue une interprétation de l'ensemble des clauses :

jean → a	a-gagné → P
pierre → b	jeu → Q
loto → c	a-joué → R

Valeur de vérité d'une interprétation :
- S (e1, e2 ..., ep) sera vrai si (e1, e2 ...ep) ∈ R où R est l'image de S.
- → S (x) est vrai si, pour tous les éléments de E, R (x) est vrai où R est l'image de S.
- S ⇒ T est faux si T est faux et S est vrai, sinon il est évalué à vrai. (⇒ désigne le symbole d'implication).

Un modèle d'un ensemble de clauses est tout ensemble de formules atomiques constituant une interprétation qui rend vraies toutes les clauses.

Exemple : dans l'exemple précédent, l'instanciation proposée conduit aux formules atomiques :

a-gagné (jean) ;
jeu (loto) ;
a-joué (jean, loto) ;
a-joué (pierre, loto).

Ces formules constituent un modèle car elles rendent vraies à l'évidence les clauses 2 à 5. Quant à la clause 1, elle est également vraie quelles que soient les instanciations de X et Y par les constantes jean, pierre et loto.
On peut vérifier que l'interprétation suivante :

pierre → a	a-gagné → P
jean → b	jeu → Q
tiercé → c	a-joué → R

ne conduit pas à un modèle.

4.1.3 Base de données relationnelle comme modèle d'un ensemble de clauses

Exemple : considérons la base de données suivante :

père	
nomp	nomf
a	b
a	c
b	d
c	c

grand-père	
nomgp	nompf
a	d
a	c

et l'ensemble de clauses :

père (X, Y) ∧ père (Y, Z) → grandpère (X, Z)
→ père (a, b)
→ père (a, c)
→ père (b, d)
→ père (c, e)
→ grandpère (a, d)
→ grandpère (a, e)

Les faits de la base de données constituent un modèle pour l'ensemble des clauses. Celles-ci sont toutes vérifiées dans l'interprétation qui consiste à mettre en correspondance les éléments qui ont le même nom.

Cette démarche n'est cependant pas très productive car la première clause n'est pas utilisée comme règle de déduction et la base de données correspond à tout ce qu'on peut déduire de l'ensemble des clauses. Pour avoir une approche déductive, il faut définir des règles d'inférence permettant d'affirmer qu'une clause w est une conséquence logique d'un ensemble d'autres clauses W. C'est-à-dire que tout modèle de W satisfait w. On n'a pas alors à exprimer les faits associés à w.

4.1.4 Calcul du 1er ordre

On peut définir un calcul du 1er ordre sur un ensemble de fbf en utilisant des règles d'inférences, par exemple le Modus ponens :

Si P est vrai et si P → Q est vrai ALORS Q est vrai

et la particularisation :

Si ∀ X P(X) est vrai ALORS P (a) est vrai

Une formule w est dérivable d'un ensemble W de fbf si elle peut être déduite de W par application finie des règles d'inférences.

Le calcul du 1er ordre repose sur le principe de résolution de Robinson qui permet de dériver une nouvelle clause à partir de deux clauses données.

Ainsi, de P → Q et Q → R, on peut dériver P → R.

Un ensemble de fbf muni de règles d'inférences constitue une théorie du 1er ordre.

4.1.5 Bases de données déductives utilisant le calcul du 1er ordre

C'est une base de données dans laquelle une partie des faits est dérivée de faits physiquement (stockés) dans la base. Elle est donc constituée de deux parties :

— des données gérées par un SGBD relationnel représentant les faits de base d'une BD logique ;

— une théorie du 1er ordre (des clauses de Horn sans fonction) permettant de déduire de nouveaux faits.

Les clauses dans la théorie peuvent être partiellement instanciées.
Des clauses sans conclusion sont interprétées comme des contraintes d'intégrité (exprimées sous forme négative puisque P ∧Q → peut s'écrire ¬P ∨ ¬Q).

Exemple : soient les relations Père (nomP, nomE) et Mère (nomM, nomE) et l'ensemble de clauses :

1. Père (X, Y) → Parent (X, Y)
2. Mère (X, Y) → Parent (X, Y)
3. Mère (X, Y) ∧ Parent (Y, Z) → Gmère (X, Z)
4. Père (X, Y) ∧ Parent (Y, Z) → Gpère (X, Z)
5. Parent (X, Y) → Ancêtre (X, Y)
6. Parent (X, Y) ∧ Ancêtre (Y, Z) → Ancêtre (X, Z)

Ancêtre est un prédicat récursif alors que les autres prédicats sont (des prédicats) hiérarchiques.
Ces prédicats permettent de calculer les extensions correspondantes à partir des relations Père (nomP, nomE) et Mère (nomM, nomE)

Ces extensions sont calculées par un moteur d'inférences utilisant le formalisme clausal, les règles de déduction et la base de données.
D'une façon pratique, il s'agit le plus souvent d'un démonstrateur PROLOG connecté à un SGBD dont il exploite les données.
Exemple : si on demande les grand-mères de jean (?Gmère (X, jean)), le système utilise la clause 3 contenant la définition du prédicat Gmère. En propageant la constante jean dans la clause, il remplace le but à résoudre par le nouveau sous-but Mère (X, Y) ∧Parent (Y, jean).
L'utilisation des clauses 1 et 2 l'amène à résoudre deux nouveaux sous-buts :

Mère (X, Y) ∧ Père (X, jean)
Mère (X, Y) ∧ Mère (Y, jean)

dont la résolution va faire appel à la base de données. Le système construit donc un arbre de résolution qui a comme racine le but initial à démontrer et comme branches les sous-buts successifs obtenus par remplacement d'un but par des sous-buts associés dans une clause.
Les feuilles sont les faits (prédicats) de base. On dit que la résolution se fait par chaînage arrière.

4.2 Bases de données déductives et règles de production

Une règle de production est de la forme :

Si conditions ALORS actions

où les actions représentent un ensemble d'opérations à effectuer (opérations arithmétiques, entrées-sorties, génération d'un nouveau fait...). Ces opérations seront effectuées si l'ensemble des conditions est satisfait.

Exemple : les règles précédentes pourraient s'exprimer sous la forme :

Si X = Père (Y) et Y = Père (Z) ALORS créer X = Gpère (Z)
Si X = Père (Y) ALORS créer X = Ancêtre (Y)
Si X = Père (Y) et Y = Ancêtre (Z) ALORS créer X = Ancêtre (Z)

Le moteur d'inférences va alors exploiter les règles en cherchant celles dont les prémisses sont satisfaites à une étape donnée.
Celles-ci peuvent générer de nouveaux faits qui peuvent entraîner le déclenchement de nouvelles règles. Le mécanisme de déduction se fait donc par chaînage avant principalement. Ce type de fonctionnement est bien adapté à des problèmes dans lesquels on veut connaître les conséquences d'un ensemble d'hypothèses, comme par exemple les problèmes de diagnostic. La base de données contient l'ensemble des données décrivant les hypothèses.

4.3 Stratégies de construction

Les bases de données déductives relationnelles (BDDR) sont les bases de données construites sur l'idée que le modèle de données sous-jacent est le modèle relationnel. La transition d'une base de données relationnelle classique vers une BDDR est relativement simple et naturelle car le modèle relationnel repose sur des expressions simples, bien formalisées en logique des prédicats du premier ordre.

La stratégie adoptée dans le développement d'une BDDR est caractérisée par la conception d'une base de données relationnelle comme un outil de stockage de faits dans un environnement de démonstrateur de théorèmes. Deux principales stratégies ont été utilisées pour construire des BDDR : la stratégie par couplage et la stratégie par intégration de systèmes.

4.3.1 Stratégie par couplage

Elle consiste à équiper un langage logique, généralement PROLOG, de fonctionnalités de gestion de faits en mémoire secondaire de façon la plus transparente possible. Ainsi, à chaque fois que le système déductif a besoin de faits en mémoire centrale pour poursuivre ses déductions, le SGBD est activé pour l'alimenter.

La stratégie par couplage peut être affinée pour distinguer deux types de couplage : le couplage lâche (faible) et le couplage serré (fort).

Dans une **approche couplage lâche**, le système déductif accède à la base de données en générant des appels *via* le langage de requêtes du SGBD. Cette tâche est celle du programmeur. À la suite d'un appel du système déductif au SGBD, celui-ci retrouve tous les n-uplets satisfaisant la requête SELECT et les dépose dans la mémoire de travail du système déductif, et ce dernier peut alors continuer ses inférences.

Dans une **approche couplage fort**, le système déductif accède à la base de données en générant des appels directement aux méthodes d'accès du SGBD. Dès que le SGBD a retrouvé un n-uplet satisfaisant les conditions de sélection, il le charge dans la mémoire de travail du système déductif. Le système déductif continue ses inférences et le SGBD continue de façon asynchrone à générer des n-uplets satisfaisant la requête SQL, préparant ainsi des réponses à d'éventuels appels à la même requête SELECT.

4.3.2 Stratégie par intégration de systèmes

Cette approche consiste à intégrer en un seul système les fonctions de gestion de données permanentes (fonctions propres à un SGBD) et les fonctions de déduction (fonctions du moteur d'inférences). L'attention est souvent portée sur trois langages : un langage de programmation logique utilisé dans le système déductif, un langage de définition de données et un langage de manipulation de données.
L'intérêt d'une telle approche est de disposer d'un seul langage déclaratif pour les requêtes, les règles de déduction et les contraintes d'intégrité. Cette stratégie a été rarement utilisée dans la pratique car elle nécessite la réécriture de systèmes complets.

4.4. Exemples de SGBD déductives relationnels

Parmi les SGBD déductifs opérationnels, nous pouvons citer :

— SGBD déductifs relationnels (par couplage) : BIMProlog/Oracle, BIM-Prolog/Ingres, BIM-Prolog/UNIFY, Delphia-Prolog/Oracle.

De très nombreux logiciels permettant d'avoir des fonctionnalités de systèmes déductifs existent. La plupart du temps, il s'agit de systèmes proposant un langage de règles dont la manipulation est assurée par un moteur d'inférence. Une passerelle entre ces logiciels et des SGBD permet d'obtenir des fonctionnalités de SGBD déductifs.
La facilité d'accès aux données et de leur manipulation dépend beaucoup de la qualité de cette passerelle. Elle peut nécessiter une écriture explicite des requêtes SQL par l'utilisateur pour l'accès aux données par le moteur d'inférence ou, au contraire, être totalement transparente à l'utilisateur. Les requêtes au SGBD sont dans ce cas générées dynamiquement par le système déductif au fur et à mesure de ses besoins.

Exemple d'un SGBD déductif : la passerelle STORIA associée au langage PROLOG proposé par la société DELPHIA. Son intérêt réside dans la possibilité de la mettre en œuvre suivant trois modes différents : un premier mode dans lequel des requêtes SQL statiques, sous forme de chaînes de caractères, sont écrites dans le programme PROLOG comme paramètres de prédicats prédéfinis. Deux prédicats sont disponibles : « query/2 » qui est à utiliser pour les SELECT (il s'agit d'un prédicat binaire dont le premier paramètre est le SELECT SQL et le second le résultat de la requête exécutée par le SGBD) et « statement/1 », prédicat unaire pour toutes les requêtes SQL de mise à jour qui ne retournent pas de résultat.

Le second mode de communication PROLOG-SGBD est un paraphrasage de SQL qui permet l'utilisation de variables. Ces variables sont dynamiquement instanciées au moment de l'exécution du prédicat, ce qui permet une bonne intégration de SQL à PROLOG.

Le troisième mode de communication réalise un accès totalement transparent à la base de données. Les n-uplets stockés dans la base sont vus par PROLOG comme des faits de sa propre base. Pour ce faire, la passerelle se construit une image des schémas de relations par consultation du dictionnaire de données du SGBD.

Ces trois modes peuvent être utilisés de manière indépendante ou simultanée.

Exemple d'application de STORIA : il s'agit d'une application de contrôle et de maintenance en temps réel de centrales hydrauliques développée à l'aide de DELPHIA-PROLOG et STORIA. Les données de fonctionnement des centrales sont captées, transmises au site de contrôle et rangées dans une base ORACLE. Un système expert analyse en permanence l'évolution des paramètres et décide des modifications à effectuer sur les réglages des différentes centrales.
Les ordres correspondants sont formulés puis envoyés aux centrales par télématique. Ce système permet donc l'exploitation optimale des centrales difficiles d'accès.

Série d'exercices sur les BDs Déductives

Exercice n° 01.

Soit une base de données déductive concernant un ensemble de réseaux informatiques et contenant les règles suivantes :
Accessible (h1, h2) :-réseau (r, h1) & réseau (r, h2)
Accessible (h1, h2) :-réseau (r1, h1) & gateway (r1, r2) & réseau (r2, h2)
Accessible (h1, h2) :-réseau (r1, h1) & gateway (r2, r1) & réseau (r2, h2)
Accessible (h1, h2) :-Accessible (h1, h3) & Accessible (h3, h2)

Le prédicat *réseau (r, h)* est un prédicat extensionnel qui exprime le fait que la machine *h* se trouve sur le réseau *r*. Le prédicat *gateway (r1, r2)* est un prédicat extensionnel qui exprime le fait qu'il existe une connexion directe entre le réseau *r1* et le réseau *r2*.

1) Cet ensemble de règles est-il récursif ? Justifiez.
2) Construisez le modèle minimum du prédicat *accessible* en considérant les extensions ci-dessous.

reseau

8	candy
16	spirou
16	gaston
24	extra
32	yahoo

gateway

8	16
16	24

Exercice n°02.

La base de données déductive d'une compagnie aérienne est constituée d'un prédicat extensionnel *vol_direct* et d'un prédicat intentionnel *vol. Vol_direct (ville1, Pays1, Villes2, Pays2)* exprime le fait qu'il existe un vol direct reliant la *ville1* du *Pays1* à la *ville2* du *Pays2*.
Vol (Ville1, Pays1, Ville2, Pays2) exprime la possibilité d'atteindre la *ville2* du *Pays2* en partant de la *ville1* du *Pays1*, en une ou plusieurs étapes.

1. Donnez la (les) règle (s) permettant de définir le prédicat *Vol*.
2. Calculez l'extension du prédicat *Vol* en utilisant l'extension du prédicat *vol_direct* décrite ci-dessous. Expliquez brièvement la technique que vous utilisez pour ce calcul.

vol_ direct

Paris	*France*	*Bruxelles*	*Belgique*
Paris	*France*	*Londre*	*Royaume-uni*
Bruxelles	*Belgique*	*Paris*	*France*
Londre	*Royaume-uni*	*Rome*	*Italie*

Exercice n° 03.

La base de données déductive d'une société contient les prédicats extensionnels suivants :

– *Employé(X)* qui exprime le fait que *X* est un employé de la société.

– *SupDirect(X, Y)* qui exprime le fait que l'employé *X* est le supérieur direct de l'employé *Y* dans la hiérarchie de la société.

On suppose que tous les employés, sauf le directeur de la société, on toujours un et un seul supérieur direct.

(a) Définir les prédicats suivants intensionnels :

i. *sup(X, Y)* qui exprime le fait que l'employé *X* est un supérieur de l'employé *Y* dans la hiérarchie de la société. Un employé *X* est un supérieur de l'employé *Y* s'il est le supérieur direct de celui-ci ou s'il est le supérieur direct d'un supérieur de l'employé *Y*.

ii. *supCommun(X, Y, Z)* qui exprime le fait que l'employé *X* est un supérieur commun aux employés (différents) *Y* et *Z*.

iii. *ppSupCommun(X, Y, Z)* qui exprime le fait que l'employé *X* est l'employé le plus bas dans la hiérarchie qui est un supérieur commun aux employés (différents) *Y* et *Z*.

iv. *mêmeNiveau(X, Y)* qui exprime le fait que les employés *X* et *Y* sont au même niveau dans la hiérarchie de la société c-à-d qu'ils ont le même nombre de supérieurs.

(b) Calculer l'extension des prédicats *sup* et *mêmeNiveau* si les extensions des relations *employé* et *supDirect* sont celles données ci-dessous :

employé
Jean
Pierre
Albert
Joseph

supDirect

Jean	*Pierre*
Jean	*Albert*
Pierre	*Joseph*

Chapitre 5 : Les BDs réparties

1. Les bases de données réparties: émergence, avantages, définition et problèmes

1.1. Emergence de la répartition des données

Les BDR (Bases de données réparties) sont d'abords des bases de données normales. En fait, elles sont issues de l'évolution de ces dernières. En effet, la gestion de bases de données avec le temps, s'est confrontée à divers problèmes qui sont :

- L'augmentation du volume de données
- l'augmentation du volume de traitements
- l'augmentation du volume de transactions
- etc.

Cela a entraîné la lenteur des applications, car les périphériques de stockage submergés, ne répondant pas assez vite. Aussi, il a été noté que les débits des liaisons réseaux évoluaient beaucoup plus vite que les capacités des périphériques de stockage.
Ainsi, l'idée est venue de multiplier les sources de données et les faire communiquer par réseau, afin de bénéficier de traitements parallèles, minimisant ainsi les temps de réponses. Aujourd'hui, les BDRs sont de plus en plus répondus, et comblent largement les lacunes des bases de données classiques.

1.2. Avantages de la répartition des bases de données

Les avantages d'une base de données répartie sont nombreux. On peut citer comme principaux :

- *Gain en performances* : les traitements se font en parallèles donc, l'architecture d'une BDR est plus adaptée à l'organisation des entreprises décentralisées.
- *Fiabilité* : les bases de données réparties ont souvent des données répliquées. Si un site a une panne, un autre peut le remplacer valablement.
- *Transparence des données* : les développeurs et les utilisateurs n'ont pas à se préoccuper de la localisation des données qu'ils utilisent.
- *Meilleures performances en temps et en espace* : réduire le trafic sur le réseau est une possibilité d'accroître les performances. Le but de la répartition des données est de les rapprocher de l'endroit où elles sont accédées. Répartir une base de données sur plusieurs sites permet de répartir la charge sur les processeurs et sur les entrées/sorties.
- *Facilite l'accroissement*: l'accroissement se fait par l'ajout de machines sur le réseau.

1.3. Définition d'une base de données répartie

Une base de données répartie (distribuée) est une base de données logique dont les données sont distribuées sur plusieurs SGBD et visibles comme un tout. Les données sont échangée par messages.

⇨ Une base de données est décentralisée ou répartie lorsqu'elle est modélisée par un seul schéma logique de base de données, mais implémentée dans plusieurs fragments de tables physiques sur des sites géographiquement dispersés et reliés par un réseau. L'utilisateur d'une base de données répartie se focalise sur sa vue logique des données et n'a pas besoin de se préoccuper des fragments physiques. C'est le SGBD qui se charge d'exécuter les opérations, soit localement, soit en les distribuant sur plusieurs ordinateurs en cas de besoin.

Remarque : Si les données sont dupliqué on parle plutôt de BD répliquée, donc il ne faut pas confondre.

1.4. Problèmes liés à la répartition des données

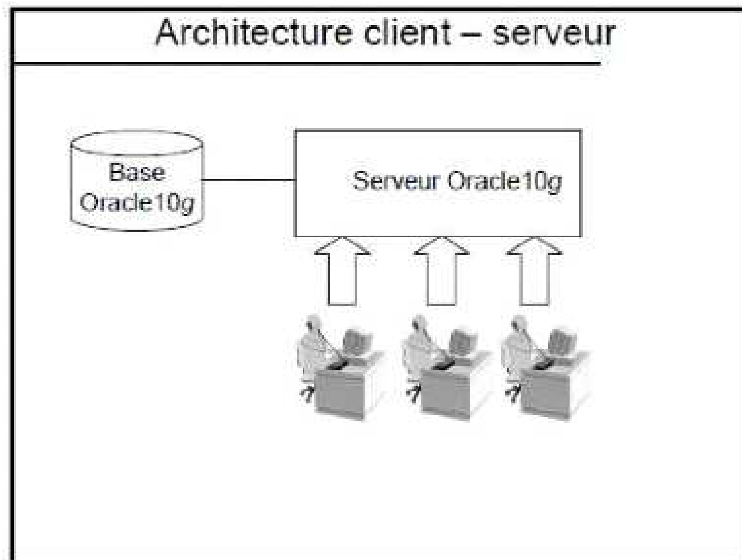
- *Coût* : la distribution entraîne des coûts supplémentaires en terme de communication, et en gestion des communications (hardware et software à installer pour gérer les communications et la distribution).
- *Problème de concurrence.*
- *Sécurité* : la sécurité est un problème plus complexe dans le cas des bases de données réparties que dans le cas des bases de données centralisées.

2. Différentes architectures

Il existe deux principaux types d'architectures qui sont :

2.1. Architecture Client/ Serveur

Dans cette architecture, l'application client se connecte au serveur de base de données (par exemple : Oracle). Ce dernier à son tour, leur renvoie des réponses en fonction de leurs requêtes.



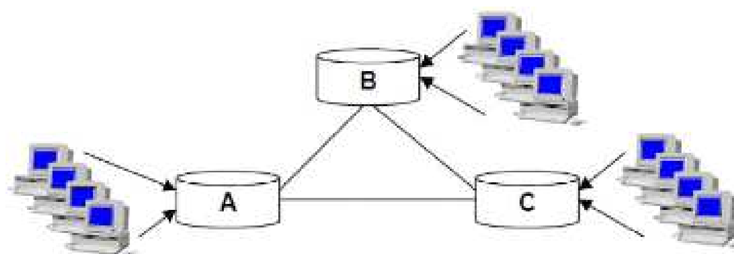
Architecture Client/Serveur

2.2. Architecture Pair à Pair

Dans un système de bases de données réparties, il existe en général plusieurs serveurs de données qui ont le même niveau d'importance. Chaque serveur gère sa base de données et échange les informations avec les autres. Le tout est vu comme une seule base de données logique.

Remarque :

De façon générale, les clients se connectent à leurs serveurs respectifs, et ces derniers s'échangent les informations si nécessaires (voir figure ci-dessous).



Connexion des clients aux serveurs d'une BDR

3. Conception d'une BD répartie

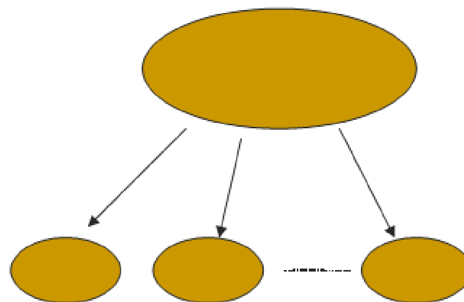
La définition du schéma de répartition est la partie la plus délicate de la phase de conception d'une BDR car il n'existe pas de méthode miracle pour trouver la solution optimale. L'administrateur doit donc prendre des décisions en fonction de critères techniques et organisationnels avec pour objectif de minimiser le nombre de transferts entre sites, les temps de

transfert, le volume de données transférées, les temps moyens de traitement des requêtes, le nombre de copies de fragments, etc...

3.1. Conception descendante (Top Down)

On commence par définir un schéma conceptuel global de la base de données répartie, puis on distribue sur les différents sites en des schémas conceptuels locaux.

La répartition se fait donc en deux étapes, en première étape la fragmentation, et en deuxième étape l'allocation de ces fragments aux sites.



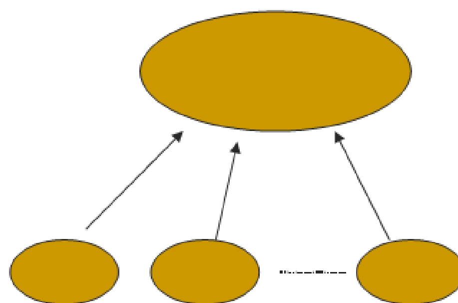
Approche descendante

Remarque :

L'approche *top down* est intéressante quand on part du néant. Si les BDs existent déjà la méthode *bottom up* est utilisée.

3.2. Conception ascendante (Bottom up)

L'approche se base sur le fait que la répartition est déjà faite, mais il faut réussir à intégrer les différentes BDs existantes en une seule BD globale. En d'autres termes, les schémas conceptuels locaux existent et il faut réussir à les unifier dans un schéma conceptuel global.



Approche ascendante

La répartition d'une base de données intervient dans les trois niveaux de son architecture en plus de la répartition physique des données :

Niveau externe: les vues sont distribuées sur les *sites utilisateurs*.

Niveau conceptuel: le schéma conceptuel des données est associé, par l'intermédiaire du schéma de répartition (lui même décomposé en un schéma de fragmentation et un schéma d'allocation), aux schémas locaux qui sont réparties sur plusieurs sites, les *sites physiques*.

Niveau interne: le schéma interne global n'a pas d'existence réelle mais fait place à des schémas internes locaux répartis sur différents sites.

4. Fragmentation

La fragmentation est le processus de décomposition d'une base de données en un ensemble de sous bases de données, c'est-à-dire que chaque table T est partitionnée en un nombre minimum de sous-tables disjointes T1,T2,..,Tn. Cette décomposition doit être sans perte d'information, c'est-à-dire que chaque fragment Ti doit contenir assez d'information pour reconstruire la table T.

NB. La fragmentation peut être coûteuse s'il existe des applications qui possèdent des besoins opposés.

4.1. Règles de fragmentation

Les **règles de fragmentation** sont les suivantes :

1. La **complétude** : pour toute donnée d'une relation *R*, il existe un fragment *Ri* de la relation *R* qui possède cette donnée.

2. La **reconstruction** : pour toute relation décomposée en un ensemble de fragments *Ri*, il existe une opération de reconstruction.

3. **La disjonction** : permet de contrôler la redondance au niveau d'allocation, il est souhaitable d'avoir de fragment disjoint

4.2. Techniques de fragmentation

Il existe plusieurs techniques de fragmentation. Chaque technique diffère de l'autre par l'unité de fragmentation.

4.2.1. Fragmentation horizontale (répartition des occurrences ou tuples)

Les tuples d'une même relation peuvent être réparties dans des fragments différents.

- Sélection selon un prédicat de qualification (càd, L'opérateur de partitionnement est la *sélection* (σ)),
- Réversible par union (càd, l'opérateur de recomposition est l'*union* (∪))

Exemple 1 :

Etant donné la relation *Compte,* comme décrit ci-dessous :

Relation Compte

No client	Agence	Type compte	Somme
174723	Lausanne	courant	123345
177498	Genève	courant	34564
201639	Lausanne	courant	45102
201639	Lausanne	dépôt	325100
203446	Genève	courant	274882

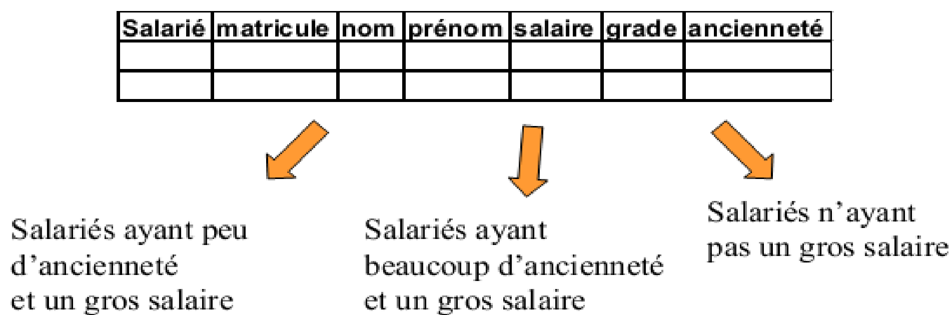
La relation *Compte* peut être fractionnée en *Compte1* et *Compte2* avec la fragmentation suivante

Compte1 = $\sigma_{[\text{TypeCompte = 'courant'}]}$ Compte

et Compte2 = $\sigma_{[\text{TypeCompte = 'dépôt'}]}$ Compte

La recomposition de *Compte* est *Compte1* ∪ *Compte2*

Exemple 2 :



Exemple3 :

Fragments définis par sélection

```
create table Client1 as
   select * from Client where ville = 'Paris'

create table Client2 as
   select * from Client where ville <> 'Paris'
```

Reconstruction

```
create view Client as
  select * from Client1
  union
  select * from Client2;
```

Client

nclient	nom	ville
C 1	Dupont	Paris
C 2	Martin	Lyon
C 3	Martin	Paris
C 4	Smith	Lille

Client1

nclient	nom	ville
C 1	Dupont	Paris
C 3	Martin	Paris

Client2

nclient	nom	ville
C 2	Martin	Lyon
C 4	Smith	Lille

4.2.2. Fragmentation verticale (répartition des attributs)

Toutes les valeurs des occurrences pour un même attribut se trouvent dans le même fragment.

- L'opérateur de partitionnement est la projection (π) (càd, Subdivision des attributs de T en groupes (en dupliquant une clé commune) et projection sur les groupes),
- L'opérateur de recomposition est la jointure (Réversible par jointure)

NB. Une fragmentation verticale est utile pour distribuer les parties des données sur le site où chacune de ces parties est utilisée.

Exemple 1 :

Soit le partitionnement de la relation précédente *Client* en deux relations :

Cli1 = $\pi_{[\text{NoClient, NomClient}]}$ Client
et Cli2 = $\pi_{[\text{Noclient, Prénom, Age}]}$ Client

NoClient	NomClient
174 723	Villard
177 498	Cattell
201 639	Tsellis
203 446	Kowalsky

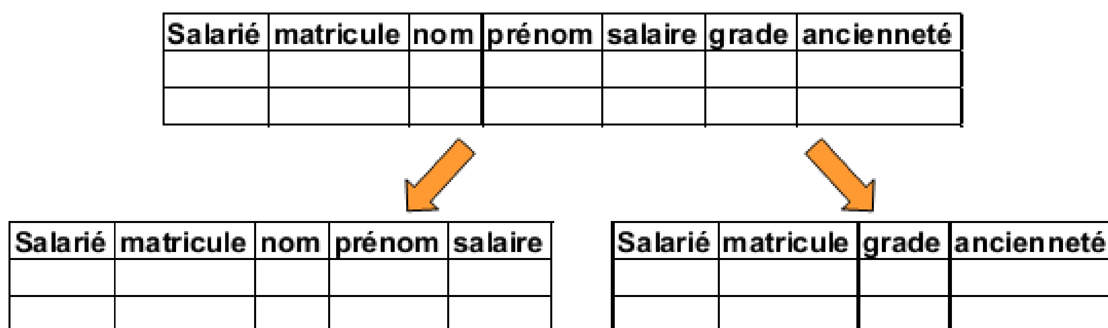
Relation Cli1

NoClient	Prénom	Age
174 723	Jean	29
177 498	Blaise	38
201 639	Alan	51
203 446	Vladimir	36

Relation Cli2

La relation d'origine est obtenue avec la recomposition suivante : Client = Cli1 * Cli2

Exemple 2 :



Salarié	matricule	nom	prénom	salaire	grade	ancienneté

Salarié	matricule	nom	prénom	salaire

Salarié	matricule	grade	ancienneté

Exemple 3 :

Fragments définis par projection
Cde1 = Cde (ncde, nclient)
Cde2 = Cde (ncde, produit, qté)

Reconstruction
Cde = [ncde, nclient, produit, qté]
where Cde1.ncde = Cde2.ncde

Cde

ncde	nclient	produit	qté
D 1	C 1	P 1	10
D 2	C 1	P 2	20
D 3	C 2	P 3	5
D 4	C 4	P 4	10

Cde1

ncde	nclient
D 1	C 1
D 2	C 1
D 3	C 2
D 4	C 4

Cde2

ncde	produit	qté
D 1	P 1	10
D 2	P 2	20
D 3	P 3	5
D 4	P 4	10

4.2.3. Fragmentation hybride ou mixte (répartition des valeurs)

C'est la combinaison des deux fragmentations précédentes, horizontale et verticale. Les occurrences et les attributs peuvent donc être répartis dans des partitions différentes.

- L'opération de partitionnement est une combinaison de projections et de sélections.
- L'opération de recomposition est une combinaison de jointures et d'unions.

NB. *Problèmes liés à ce type de fragmentation :* ***trop de sensibilité aux mises à jour et trop de jointures***

Exemple :

La relation *Client* est obtenue avec : $(Cli3 \cup Cli5) * Cli4 * Cli6$

- Relation *Cli3* $\pi_{[NoClient,\ NomClient]}\ (\sigma_{[Age < 38]}Client)$
- Relation *Cli5* $\pi_{[NoClient,\ NomClient]}\ (\sigma_{[Age \geq 38]}Client)$
- Relation *Cli4* $\pi_{[NoClient,\ Prénom]}Client$
- Relation *Cli6* $\pi_{[NoClient,\ Age]}Client$

5. Allocation des fragments

L'affectation des fragments sur les sites est décidée en fonction de l'origine prévue des requêtes qui ont servi à la fragmentation. Le but est de placer les fragments sur les sites où ils sont le plus utilisés, et ce pour minimiser les transferts de données entre les sites.

L'allocation peut se faire ***avec réplication*** ou ***sans réplication***. Sachant que la réplication favorise les performances des requêtes et la disponibilité des données, mais est coûteuse en considérant les mises à jour des fragments répliquas.

5.1. Fragmentation avec réplication

Ce type de fragmentation est réalisé en appliquant l'une des deux méthodes suivantes :

- ✓ Déterminer l'ensemble de tous les sites dont l'importance d'allouer une copie est d'intérêt plus élevé que le coût de transfert puis allouer une copie de fragment à chaque élément de cet ensemble.
- ✓ Déterminer premièrement la solution du problème qui n'est pas la réplication et introduire progressivement les copies en commençant par celles qui sont plus avantageuses. Le processus prend fin si aucune réplication additionnelle n'est avantageuse.

Exemple :

Dans un milieu bancaire les retraits d'argent par un client ne devraient être autorisés que si le compte est suffisamment approvisionné. Comme les ordres de virement et chèques émis au cours des heures écoulées ne sont pas encore connus et donc non débités, la situation réelle du compte peut être bien différente de l'état correspondant dans la base de données.
Par conséquent, la décision d'autoriser un retrait d'argent peut être prise en fonction d'un cliché légèrement ancien plutôt que d'accéder à l'information originale qui ne sera que légèrement plus fiable. Cependant, une information exacte permettrait de contrecarrer des retraits d'argent successifs dans deux distributeurs à quelques minutes d'intervalle.

5.1.1. Techniques de réplication

Dans le cas où la méthode classique d'allocation des fragments ne s'avèrent pas satisfaisante, des techniques plus puissantes mais aussi complexes à mettre en oeuvre doivent être envisagées :

- Allocation avec duplication des fragments
- Allocation dynamique des fragments
- Fragmentation dynamique
- Clichés

A. Allocation avec duplication

Certains fragments peuvent être dupliqués sur plusieurs sites (éventuellement sur tous les sites) ce qui procure l'avantage d'améliorer les performances en termes de temps d'exécution des requêtes (en évitant certains transferts de données). Elle permet aussi une meilleure disponibilité des informations (connues de plusieurs sites), et une meilleure fiabilité contre les pannes. Par contre, l'inconvénient majeur est que les mises à jour doivent être effectuées sur toutes les copies d'une même donnée. En conséquence, moins un fragment est sujet à des modifications, plus il est prédisposé à la duplication.

B. Allocation dynamique

Avec cette technique, l'allocation d'un fragment peut changer en cours d'utilisation de la BDR. Ce peut être le cas suite à une requête par exemple. Dans ce cas, le schéma d'allocation et les

schémas locaux doivent être tenus à jour. Cette technique est une alternative à la duplication qui se révèle plus efficace lorsque la base de données est sujette à de nombreuses mises à jour.

C. Fragmentation dynamique

Dans le cas où le site d'allocation peut changer dynamiquement, il est possible que deux fragments complémentaires (verticalement ou horizontalement) se retrouvent sur le même site. Il est alors normal de les fusionner. A l'inverse, si une partie d'un fragment est appelé sur un autre site, il peut être intéressant de décomposer ce fragment et de ne faire migrer que la partie concernée. Ces modifications du schéma de fragmentation se répercutent sur le schéma d'allocation et sur les schémas locaux.

D. Clichés

Un cliché (snapshot) est une copie figée d'un fragment. Il représente l'état du fragment à un instant donné et n'est jamais mis à jour contrairement aux vues et aux copies qui répercutent toutes les modifications qui ont lieu sur le fragment original.
L'intérêt d'un cliché diminue donc au fur et à mesure que le temps passe. L'utilisation des clichés est intéressante lorsque l'on juge que la gestion de copies multiples se révélerait trop lourde pour la base de données considérée alors que des copies même peu anciennes et non à jours seraient largement suffisantes.

Remarques :

- On peut en effet se passer de l'information exacte pour diverses raisons. D'une part l'information contenue dans la base de données peut ne pas refléter tout à fait la réalité (cas d'un changement d'adresse non signalé, par exemple).
- D'autre part, certaines informations ne subissent pas souvent de modification (comme le nom de famille, l'adresse ou le nombre d'enfants des employés) et par conséquent une copie même ancienne de ces informations est, dans sa grande majorité, encore exacte.

5.2. Fragmentation sans réplication

L'allocation **sans réplication** est facile à réaliser, il suffit d'associer à chaque allocation une mesure et à chaque site une meilleure mesure. C'est une solution qui ne tient pas compte de l'effet naturel, il faut placer un fragment dans un site donné si un autre fragment apparenté est aussi dans ce site.

6. Fragmentation et transparence

La transparence est la caractéristique principale d'un système distribué dans lequel l'utilisateur doit se voir travailler sur un énorme ordinateur personnel constitué de tous les ordinateurs connectés.
Nous distinguons plusieurs niveaux de transparence de répartition qui sont indépendantes du programme d'application de la répartition :

6.1. Transparence globale

La transparence globale définit toutes les données contenues dans la base de données réparties comme si cette base était définie exactement comme dans une base de données non réparties.

6.2. Transparence de fragmentation

Une relation globale peut être répartie en plusieurs fragments, une transparence de fragmentation définit une fonction entre la relation globale et les fragments.
Cette fonction est multivaluée, c'est à dire plusieurs fragments correspondent à une relation globale, mais une seule relation globale correspond à un seul fragment.

6.3. Transparence d'allocation

Les fragments sont des portions logiques des relations globales qui sont uniquement situées dans un ou plusieurs sites du réseau. La transparence d'allocation définit le site dans lequel est situé un fragment. La relation définit dans la transparence d'allocation détermine si la base de données répartie est redondante ou pas.

6.4. Transparence conceptuelle locale

La transparence conceptuelle locale définit une fonction qui associe chaque image physique aux objets qui sont manipulés par les systèmes de gestion de base de données locaux. Cette transparence dépend du type de système de base de données locale.

7. Traitement & Optimisation de Requêtes Réparties

Les règles d'exécution et les méthodes d'optimisation de requêtes définies pour un contexte centralisé sont toujours valables, mais il faut prendre en compte d'une part la fragmentation et la répartition des données sur différents sites et d'autre part le problème du coût des communications entre sites pour transférer les données. Le problème de la fragmentation avec ou sans duplication concerne principalement les mises à jours tandis que le problème des coûts des communications concerne surtout les requêtes.

7.1. Mise à jour des BD réparties

La principale difficulté réside dans le fait qu'une mise à jour dans une relation du schéma global se traduit par plusieurs mises à jour dans différents fragments. Il faut donc identifier les fragments concernés par l'opération de mise à jour, puis décomposer en conséquence l'opération en un ensemble d'opération de mise à jour sur ces fragments.

- ***Insertion***

Retrouver le fragment horizontal concerné en utilisant les conditions qui définissent les fragments horizontaux, puis insertion du tuple dans tous les fragments verticaux correspondants.

- ***Suppression***

Rechercher le tuple dans les fragments qui sont susceptibles de contenir le tuple concerné, et supprimer les valeurs d'attribut du tuple dans tous les fragments verticaux.

- ***Modification***

Rechercher les tuples, les modifier et les déplacer vers les bons fragments si nécessaire.

7.2. Requêtes sur les BDs réparties

Comme pour le traitement de requêtes en Bases de données centralisées, on produit l'arbre algébrique de la requête. Chaque feuille de l'arbre représente une relation, et chaque nœud représente une opération algébrique. On enrichit l'arbre avec les informations sur la répartition des données sur les différents sites, en particulier sur le site où chaque opération de la requête doit être exécutée.

La complexité d'une requête dans une base de données répartie est définie en fonction des facteurs suivants :

– Entrées/ Sorties sur les disques (*disks I/Os*), c'est le coût d'accès aux données.
– Coût CPU : c'est le coût des traitements de données pour exécuter les opérations algébriques (jointures, sélections ...).
– Communication sur le réseau : c'est le temps nécessaire pour échanger un volume de données entre des sites participant à l'exécution d'une requête.

Dans une base de données centralisée, seuls les facteurs E/Ss et CPU déterminent la complexité d'une requête.

Notons que nous faisons la distinction entre le *coût total* et le *temps de réponse global* d'une requête:

– ***Coût total*** : c'est la somme de tous les temps nécessaires à la réalisation d'une requête. Dans ce coût, les temps d'exécution sur les différents sites, les accès aux données et les temps de communication entre les différents sites qui entrent en jeu.
– ***Temps de réponse global*** : c'est le temps d'exécution d'une requête. Comme certaines opérations peuvent être effectuées en parallèle sur plusieurs sites, le temps de réponse global est généralement inférieur au coût total.

7.2.1. Transferts de données

Le temps de transmission d'un message tient compte du temps d'accès et de la durée de la transmission (volume des données / débit de transmission). Le temps d'accès est négligeable sur un réseau local, mais peut atteindre quelques secondes pour des transmissions sur de longues distances ou via satellite. Dans ces conditions, un traitement ensembliste des données s'impose. L'unité de transfert entre sites est une relation ou un fragment, et non une occurrence.

7.2.2. Traitement de requêtes réparties

Le but est d'affecter de manière optimale un site d'exécution pour chacune des opérations algébriques de l'arbre. Pour cela, on associe à chacune des feuilles le site sur lequel la relation va être puisée. Lorsqu'une relation est dupliquée, le choix du site de départ est un élément d'optimisation. On cherche ensuite à associer à chaque nœud de l'arbre le site sur lequel l'opération algébrique associée à ce nœud sera exécutée. Généralement, il faut faire localement tous les traitements qui peuvent y être faits.

Ainsi, lorsque toutes les opérandes d'une même opération algébrique sont situées sur le même site, la solution la moins coûteuse pour exécuter cette opération est le plus souvent de l'exécuter sur ce site. Ceci est notamment toujours vrai pour les opérateurs unaires qui font diminuer le volume d'informations (sélection, projection).

Pour diminuer le volume de données transmis d'un site à un autre, il faut limiter les transferts d'information aux seules informations utiles. Pour cela il faut systématiquement rajouter des projections dans l'arbre algébrique pour abandonner les attributs inutiles. Il faut aussi noter que les parties de requêtes indépendantes peuvent être exécutées en parallèle sur des sites différents et donc baisser le temps total d'exécution.

7.2.3. Optimisation dynamique des requêtes

Après avoir généré un arbre de requête, la stratégie adoptée pour l'exécution est ascendante. C'est à dire que l'affectation de chaque nœud de l'arbre à un site peut être décidée en cours d'exécution en fonction des différents volumes de données intermédiaires obtenus sur les sites. On part donc des feuilles, où l'on connaît les données (type, volume, statistiques sur la répartition des occurrences dans les domaines de valeurs...) pour remonter au niveau supérieur et prendre une décision sur le site d'affectation de l'opérateur. Et ainsi de suite pour les niveaux supérieurs jusqu'à la racine. Il faut cependant noter que cette stratégie n'est pas optimale si elle est effectuée en aveugle.

D'une manière générale, il faut tenir compte des volumes de données de chaque relation, et de leur composition. Il est en effet parfois intéressant de tenir compte du calcul effectué par un site avant que les autres sites se mettent à travailler.

7.2.4. Semi-jointure

Les BDR ont abouti à la définition d'un nouvel opérateur, la semi-jointure, qu'il est parfois intéressant d'utiliser. Il s'agit en fait d'une double jointure : le principe est d'effectuer deux petites jointures plutôt qu'une grosse; c'est à dire deux petites transmissions de données plutôt qu'une seule beaucoup plus volumineuse.

La semi-jointure réduit la taille des opérandes des relations. Elle permet de réduire la taille des données à transmettre.

A. *Algorithme de semi-jointure*

Soient *R1* et *R2* deux relations se trouvant respectivement sur les sites *S1* et *S2*.

But : Evaluer $R1 \infty R2$ sur le site *S1*.

L'algorithme de semi-jointure se déroule comme suit,

S1> $temp1 \leftarrow \pi_{R1 \cap R2}(R1)$

S1> envoi de *temp1* vers *S2*

S2> $temp2 \leftarrow R2 \bowtie temp1$

S2> envoi de *temp2* vers S1

S1> $R1 \bowtie temp2$ (équivalent à $R1 \bowtie R2$)

Série d'exercices sur les BDs réparties

Exercice 1 :

1- Expliquez la différence entre une BD répartie et une BD parallèle. Donnez 2 avantages des BD réparties en comparaison avec une architecture centralisée.

2- Décrivez le principe de la stratégie d'optimisation par semi-jointure. Précisez comment cette stratégie permet d'accélérer la jointure dans le contexte des BD réparties.

3- Expliquez le rôle de la répartition cyclique par bloc employée dans certaines architectures RAID.

4- Expliquez la différence entre la fragmentation horizontale et verticale d'une table.

5- Donnez deux avantages de la fragmentation dans le contexte des bases de données réparties.

6- Donnez deux avantages de la duplication dans les BD réparties.

7- Expliquez brièvement la différence entre la duplication répartie synchrone et asynchrone.

8- Précisez le rôle des vues matérialisées (MATERIALIZED VIEW) dans les BD réparties.

9- Donnez deux différences entre l'optimisation de requêtes dans les BD centralisées et dans les BD réparties.

10- Expliquez la différence entre les architectures RAID1 et RAID5. Dites comment ces architectures se comparent en terme de fiabilité et de performance.

11- Dites comment l'opération de sélection peut être accélérée dans les BD parallèles.

12- Expliquez la différence entre les architectures à mémoire partagée et à disque partagés. Donnez un avantage et un inconvénient pour chacune d'elles.

Exercice 2 :

Soit le schéma global de la base de données hospitalière de la région Alsacea:

Service (Snum, nom, hôpital, bât, directeur)

le directeur d'un service est un docteur désigné par son numéro

Salle (Snum, SAnum, surveillant, nbLits)

le numéro de salle est local à un service, i.e., il peut y avoir des salles avec le même numéro dans des services différents d'un même hôpital.

nbLits est le nombre total de lits d'une salle,

un surveillant de salle est un infirmier désigné par son numéro

Employé (Enum, nom, adr, tél)

Docteur (Dnum, spéc) -- spéc est la spécialité du médecin

Infirmier (Inum, Snum, rotation, salaire)

Un employé est soit infirmier soit docteur (Inum et Dnum font référence à Enum).

Patient(Pnum, Snum, SAnum, lit, nom, adr, tél, mutuelle, pc)

L'attribut pc est la prise en charge par la mutuelle

Acte (Dnum, Pnum, date, description, coef) -- coef est le coefficient de l'acte médical

Question 1

Exprimer en SQL la question suivante: “Donner le nom des cardiologues qui ont traité un ou plusieurs patients hospitalisés dans un service de gérontologie.”

Répartition des données

La base est répartie sur trois sites informatiques,“Strasbourg”, “Colmar” et “Régional”, correspondant aux valeurs “Ambroise Paré”, “Colmar” et "autre" de l'attribut hôpital de Service.

Question 2

Proposer (et justifier) une bonne décomposition de la base hospitalière sur ces trois sites. On pourra utiliser la fragmentation horizontale et/ou verticale ainsi que la réplication des données, en se basant sur les hypothèses suivantes (H1 à H5) :

- H1: Les sites Strasbourg et Colmar ne gèrent que les hôpitaux correspondants.
- H2 : Les infirmiers sont employés dans un servicedonné.
- H3 : Les docteurs travaillent le plus souvent surplusieurs hôpitaux (ou cliniques).
- H4 : La gestion des lits d'hôpitaux est locale à chaque hôpital.
- H5 : On désire regrouper la gestion des frais d'hospitalisation au centre régional.

Pour chaque fragment, on donnera sa définition en algèbre relationnelle à partir du schéma global.

Question 3

Indiquer comment se calcule chaque relation de la base globale à partir de ses fragments.

Question 4

Proposer un plan d'exécution réparti pour la requête SQL vue en Question 1, sachant maintenant que les données sont réparties sur les trois sites selon la décomposition proposée à la Question 2.

Exercice 3 :

Le schéma global d'une base de données cinématographique est le suivant :

Film (F, titre, année, durée, réalisateur). un film est identifié par son numéro F

Artiste (A, nom, prénom, nationalité) un artiste est identifié par son numéro A

Rôle (nom, F, A) l'artiste numéro A joue le rôle Nom dans le film numéro F.

Cinéma (C, arrond, ville) un cinéma est identifié par son nom C.

Salle (C, Sa, P, clim) la salle numéro Sa du cinéma numéro C a une capacité de P places. L'attribut clim, valant 'oui' ou 'non', indique si la salle est climatisée.

Séance (C, Sa, H, F) le film numéro F est projeté dans la salle numéro Sa du cinéma C à partir de l'heure H.

Les attributs soulignés forment la clé d'une relation.

Question 1: Un spectateur Si possède une partie de la base sur son site personnel. Un spectateur cinéphile ne s'intéresse qu'aux films anciens réalisés jusqu'à 2000 inclus, un spectateur tendance ne s'intéresse qu'aux films récents réalisés en 2001 et après. Un spectateur ne s'intéresse qu'à des films projetés dans sa ville: Paris, Lyon ou Nice. Un spectateur s'intéresse àtoutes les données liées aux films qui l'intéressent.
Par contre, un spectateur ne s'intéresse jamais à la capacité d'une salle.

Remarque : On peut déduire de l'énoncé qu'un spectateur cinéphilene s'intéresse pas aux films récents dans lesquels jouent des acteurs qui jouent aussi dans des films anciens.

Définir une fragmentation de la base cinématographique qui permette à un spectateur de construire sa base personnelle en répliquant des fragments sur son site, en respectant les conditions suivantes :

- un fragment est répliqué soit entièrement, soit pas du tout, sur le site d'un spectateur,
- la base d'un spectateur doit contenir toutes les données qui l'intéressent, et seulement celles-ci.

Pour chaque relation du schéma global :

a) Les *n* fragments d'une relation R sont notés Ri avec i $\in[1, n]$. Donner le nombre de fragments et écrire les expressions algébriques de définition des fragments.

b) La fragmentation proposée est-elle complète ? Est elle disjointe ? Si non expliquez pourquoi.

c) Définir le schéma de reconstruction de la relation à partir des ses fragments. Donner son expression algébrique.

Exercice 4 :

Une compagnie Microdur ayant deux succursales indépendantes au Québec et en France décide de bâtir un portail Web permettant aux clients d'acheter des items provenant de ces ceux endroits. La compagnie aimerait permettre aux usagers de consulter simultanément les catalogues des deux succursales tout en minimisant les délais lors de la consultation de ces catalogues.

La compagnie vous donne l'information suivante :

- L'application Web réside au même endroit que la BD de la succursale québécoise;
- Les serveurs BD sont respectivement, serveur-bd.microdur.quebec.com et serveur-bd.microdur.france.com;
- Une succursale ne doit pas pouvoir modifier le catalogue de l'autre, mais doit voir instantanément les changements faits à celui-ci.

1. Proposez une solution à la compagnie Microdur satisfaisant les requis décrits ci-haut.
2. Expliquez brièvement comment cette solution pourrait être implémentée à l'aide de la technologie Oracle.

Exercice 5 :

On considère la relation

Employé(E, nom, D, salaire)

Un n-uplet représente un employé identifié par son numéro E. Le numéro D fait référence au directeur de l'employé. Le directeur est aussi un employé.

Soit la requête R1 :

select a.nom
from Employé a, Employé b
where a.D = b.E and a.salaire > b.salaire

Question 1. Traduire la requête R1 en une phrase, en français :

Question 2. Donner l'expression algébrique de la requête ***R1*** en fonction de la relation globale Employé, et telle que les opérations les plus réductrices sont traitées le plus tôt possible.

Question 3. La relation ***Employé*** est fragmentée en 2 fragments ***E1*** et ***E2*** tels que:

E1 contient tous les employés dont le salaire est inférieur ou égal à 1000,

E2 contient tous les employés dont le salaire est supérieur à 1000.

- Donner l'expression algébrique de la requête ***R1***, en fonction des fragments ***E1*** et ***E2***, et telle que l'expression soit de la forme :

$$R1 \Leftrightarrow \pi_{nom} (T1 \cup T2 \cup \dots \cup Tn)$$

où les sous expressions Tine contiennent pas d'union et peuvent ne pas être vides, les opérations les plus réductrices sont traitées le plus tôt possible.

- Préciser combien il y a de sous expressions Ti.

ANNEXE : Solution des exercices

Rappel sur les bases de données

Exercice1 :

1- Non (selon la cardinalité : 1-1 côté journaliste)
2- Non (voir cardinalité)
3- Oui
4- Oui

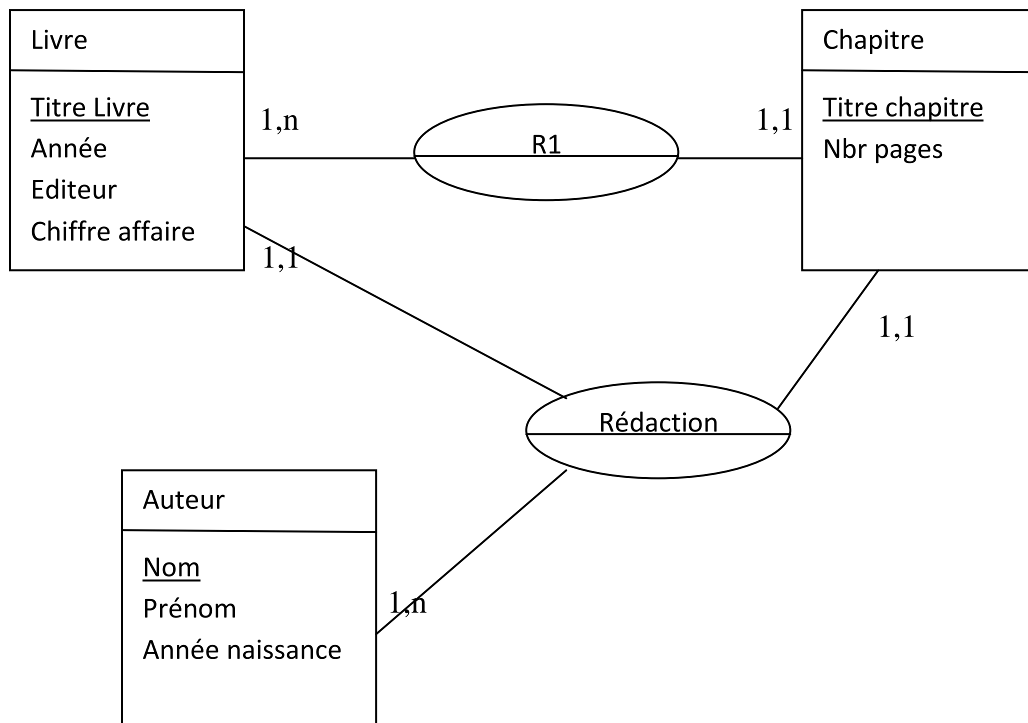
Exercice2 :

Elle n'est pas conforme à la définition car :

- Il existe des tuples en doubles
- Dans le premier tuple, l'attribut acteur a deux valeurs différentes (voir 1ére FN, 2éme FN et 3émé FN)

Exercice3 :

-a.



-b.

Create Table Livre (Titre Livre : String(60), Année : Integer, Editeur : String (30), Chiffre affaire : Double, Primary Key (Titre Livre) Unique).

Create Table Chapitre (Titre chapitre : String(30), Nbr pages : Integer, Titre Livre : String(60), Foreign Key (Titre Livre) References Livre)
Create Table Auteur (Nom : String(20), Prénom : String(30), Année naissance : Date, Primary Key (Nom))

Create Table Rédaction (Nom: String(20), Titre Livre : String(60), Titre chapitre : String(30), Foreign Key (Nom) References Auteur, Foreign Key (Titre Livre) References Livre, Foreign Key (Titre chapitre) References Chapitre, Primary key(Nom, Titre Livre, Titre chapitre))

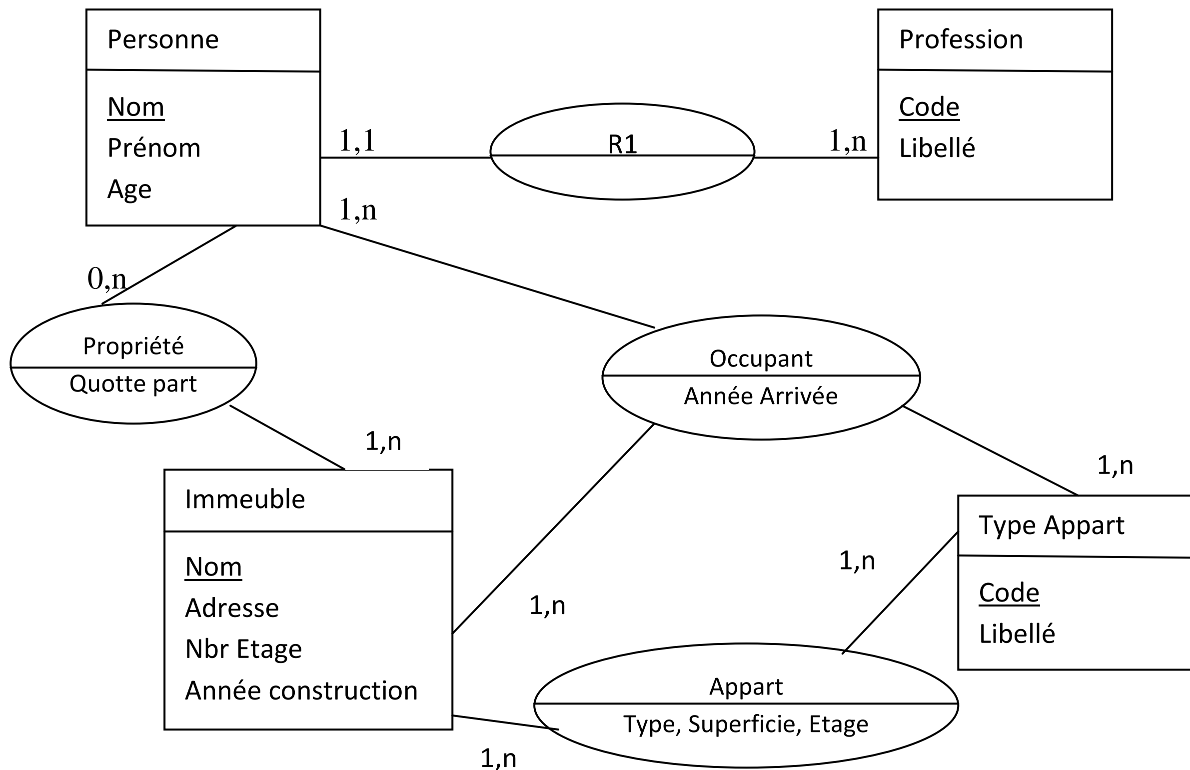
-c. On met DELETE CASCADE sur la clé étranger Titre Livre de la relation Chapitre

Exercice4:

-1. Les clés étrangers dans chaque relation :

- Immeuble : Aucune clé étranger
- Appart : NomImm, NomApp
- Personne : CodeProfession
- Occupent : NomImm, N°App, NomOccupent
- Propriété : NomImm, NomPropriétaire
- TypeAppart : Aucune clé étranger
- Profession : Aucune clé étranger

-2. Le Schéma E/A



Exercice1 :

Class Page_w

{URL : D_{URL};

F_html: D_{html};

Image8assoc:List D'image;

Affiche();

Acceder (U:D_{URL});

…..

}

Class Page_w_home: Page_w

{P_précédente: D_{URL} Null (Un champ spécifique pour dire que la page home est la première page du site donc elle n'a pas de pages precedents c-à-d discuter avec les étudiants et vous pouvez aboutir à d'autres solutions)

}

Class Site_w

{home_p : Page_w_home ;

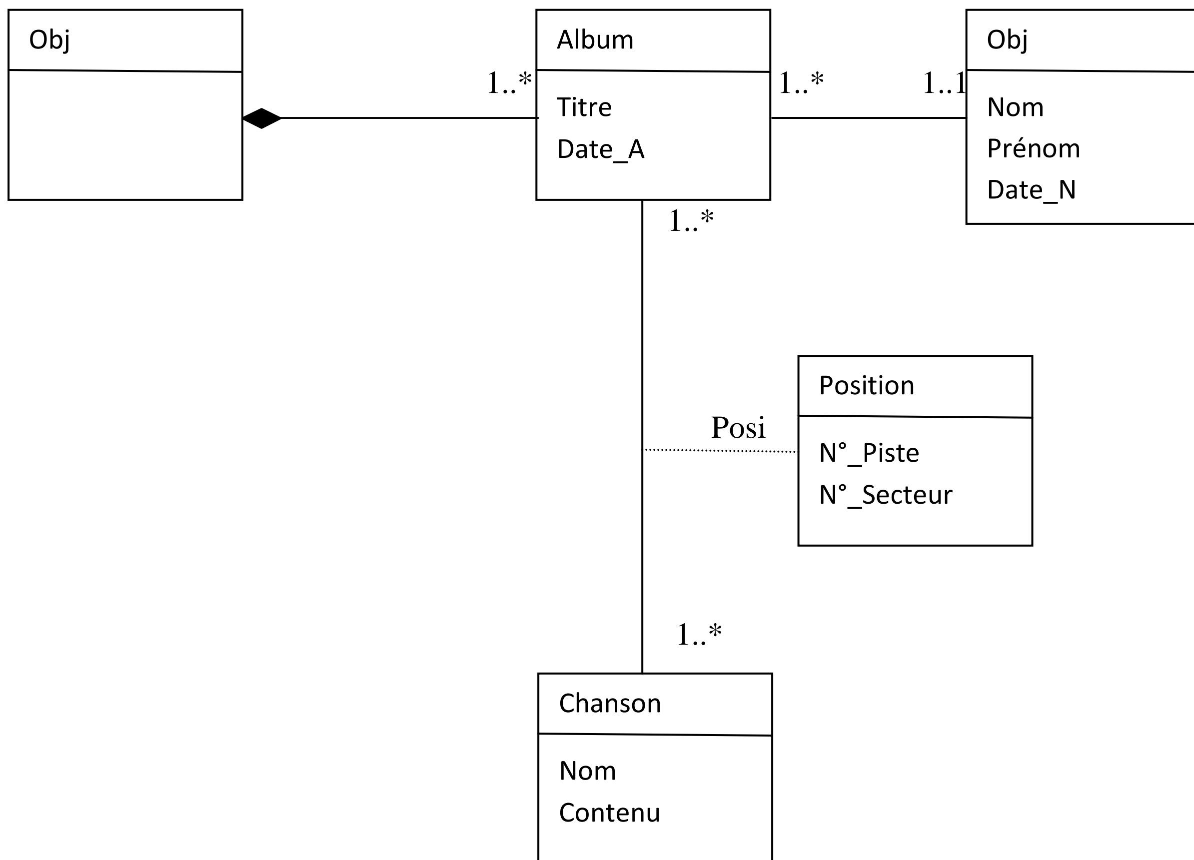
Page_sec: List page_w;

}

Vous pouvez proposer un ensemble de méthodes à votre choix ; Exemple accéder_site()

Exercice2 :

On peut proposer le diagramme de classe UML associé à ce problème comme suit :



Donc, le type de cet objet sera comme suit :

Class Obj

{Album : List album

}

Class Album

{Titre: String;

Date_A: Date;

chanteur: Chanteur;

Chanson : List Chanson ;

}

Class Chanteur

{Nom : String ;

Prénom : List String ;

```
 Date_N: Date;
 Albu: Set Album Inverse Album.chanteur;
 }
Class Chanson
 {Nom: String;
 Contenu: String;
 Alb: List Album Inverse Album.chanson
 Pos: Position;
 }
Class Position
 {Albu: Album
 Chansons: Chanson Inverse Chanson.pos;
 N°_Secteur : Integer ;
 N°_Piste : Integer ;
 }
```

Exercice3:

```
Class Modèle_Machine
 {Nom_M : String ;
 fabricant : Fabricant;
 Lieu_F : String;
 Prog_lavage: : Set Prog_Lavage ;
 Prix_Achat: Float;
 Nbr_M_Garantie : Integer ;
 }
Class Fabricant
 {Nom : String ;
 Pays : String;
```

```
        Nbr_Emp : Integer;
        Nbr_Emp : Integer ;
        Année_Création: Integer;
        Modèle_Mach : Set Modèle_Machine Inverse Modèle_Machine.fabricant ;
        }
Class Phase
        {Durée : Integer ;
        Desciption : String;
        Consommation_Eau :Float;
        Consommation_Elect :Float;
        Vitesse_Rotation_Tombour: Integer;
        Prog_lavage :  Set Prog_Lavage Inverse Prog_Lavage.Ph;
        }
Class Prog_Lavage
        {Ph:  Array[0..4] of Phase;
        }
```

Exercice4:

```
Class Document
        {Nom : String ;
        Type : String;
        Entête : Struct {Titre : String;
                      Auteur : Set Struct {Nom : String ;
                                          Affiliation : String ;
                                          }
                      }
         Résumé : List Array[1..15] of Ligne_txt ;
         Corps: Corps_doc;
```

```
 Bibli: Bibliographie;
 }
Class Corps_doc
   {Suite_P : Set Paragraphe ;
   }
Class Paragraphe
   {Num: Integer;
    Titre: String;
    Contenu: List Alineas;
   }
Class Alineas
  {texte: Set Texte;
   Sous_P: Paragraphe;
  }
Class Texte
  {Ligne_T : List Ligne_Txt
   Fig : Set Figure ;
  }
Class Figure
  { Num_Fig : Integer ;
   Titre_Fig: String;
   Image: Struct { Type: String;
                   Taille: Integer;
                  }
   Légende: String;
   Desciption: String;
  }
```

```
Class Ligne_Txt
        {Réf: List Référence;
         Para : List Paragraphe ;
         Fig : List Figure ;
        }
Class Bibliographie
        {Réf_Ouvrage : Set Référence ;
        }
Class Référence
       {Nom : String ;
        Titre : String ;
        Date_P : Date ;
        Lien_P: String;
       }
```

Ajouter les attributs références Inverse aux différents classes concernées

Le Relationnel Etendue

Exercice1 :

1- Quelques problèmes liés au modèle relationnel

a) Pouvoir d'expression du modèle :

➔ L'attribut adresse est un attribut composé donc dans cet exercice soit on le considère comme un attribut atomique de type chaîne de caractère, soit on éclate les relations qui le contient en deux relations : la relation mère et la relation adresse, ce qui engendra d'autres types de problèmes (jointure volumineuse par exemple).

Remarque : Si on considère que « Adresse » est composée, on doit l'éclater dans une nouvelle relation, sinon les relations **Appartements** et **Clients** ne seront pas en 1FN

➔ L'attribut type_Appart_préféré est un attribut multi-value et ce type d'attribut ne peut pas être exprimé en relationnel (même chose pour l'attribut **photo**).

b) Pouvoir d'expression des requêtes :

Les types de requête qui peuvent engendrer des problèmes dans le modèle relationnel :

- Donner les types d'appartement préférés d'un client
- Donner les clients qui habitent la même ville
- Donner le différentes photos d'un appartement donné
- Donner les appartements qui possèdent des photos de considération, etc…

2- Définition du domaine des attributs

Num_Appart : Numérique ou VarChar

Adresse : c'est un produit cartésien (Num, Rue, Ville, Pays)

Type : Numérique

Nb_Pièces : Numérique

Num_client : Numérique ou VarChar

Nom : VarChar

Prénom : VarChar

Tél : VarChar si on considére l'indicatif du pays

Type_Appart_Préféré : Numérique

Loyer_Max : Numérique ou VarChar si on considère DA

Num_Photo : Numérique ou VarChar

Num_Appart : Numérique ou VarChar

Date : Date

Commentaire : String

Photo : VarChar puisque il s'agit du chemin où se trouve la photo

3- Revoir la question 1- b

4- Proposition d'un schéma de type relationnel objet :

Create Type Adr As Objet (Num : Integer, Rue : VarChar(20), Ville : VarChar(20), Pays : VarChar(20))

Create Type TAP As Varray(4) of Integer

Create Type Tphoto As Varray(5) of VarChar(50)

Create Table Appartement (Num_Appart VarChar(10), Adresse: Adr, Type: Integer, Nb_Pièces: Integer, Primary Key(Num_Appart))

Create Table Clients(Num_Client : Integer, Nom : VarChar(20, Prénom : VarChar(20), Adresse : Adr, Tél : VarChar(15), Type_Appart_Préféré : TAP, Loyer_Max : Float(double), Primary Key(Num_Client))

Create Table Photo(Num_Photo : VarChar(10), Num_Appart : VarChar(10), Date : Date, Commentaries : String, Photo : Tphoto, Primary Key(Num_appart, Num_Photo))

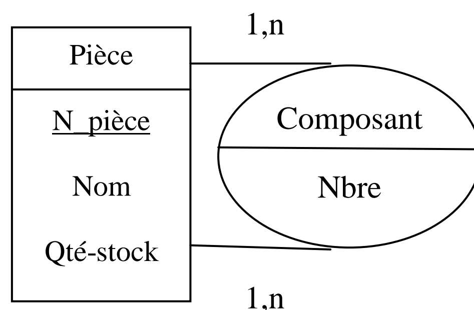
Remarque: On peut considerer que Tél est un attribut composé de :

- Indicatif_pays : Integer
- Indicatif_Ville : Integer
- Numéro : Integer

Exercice 2 :

1.

A. Modélisation en entité-association



B. **Le schéma relationnel**

Create table Pièce (N_pièce integer (ou varchar (5)), Nom string, Qté_stock interger primarykey (N_pièce))

Create table composant (N_composé interger, N_composant integer, Nbre integer primarykey (N_comosé, N_composant))

Remarque.

- N_composé et N_composant sont les mêmes que N_pièce. C'est-à-dire ils prennent leurs valeurs de l'attribut N_pièce.
- Nbre est pour spécifier le nombre de fois d'apparition d'un composant dans une pièce. Exemple : (1, 2, 4) où 1 est le numéro de la pièce table, 2 est le numéro de la pièce pied de table et 4 est le nombre d'apparition de pieds dans une table.

2.

Q1. Select N_composant, Nbre From Composant Where N_composé=10001

Q2. Select N_composé From Composant Where N_composant=25400

3. Représentation en relationnel objet

Create type Tpièce As Object (N_pièce integer, Nom String, Qté_stock integer)

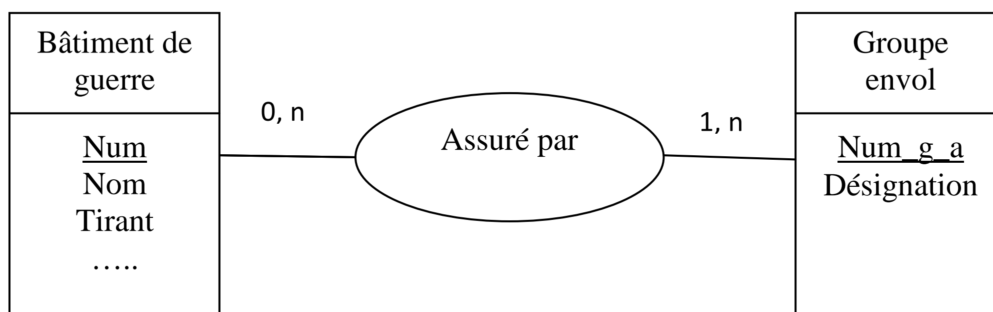
Create table Pièces of Tpièce

Create table Composant (N_pièce integer, composant set of ref (Tpièce))

Laisser l'étudiant de développer quel est l'apport du Relationnel étendu dans ce cas.

Exercice 4 :

1. **Une proposition d'un diagramme E/R**



2.

Create ype T_bateau_g As Object (Num Number, Nom String, Tirant varchar(30))

Not Final

Create Type T_cannonier Under T_bateau_g (nbr Number, Calibre String)

Create Type T_porteur Under T_bateau_g (long_vol Number, groupe_a set of T_g_a)

Create Type T_sous_marin Under T_bateau_g (prof_plong_max Number)

Create Type T_cuirassé Under T_bateau_g (X ref T_porteur, Y ref T_cannonier)

Create Type T_g_a As Object (Num_g_a Number, Désign String)

Create Table les_batimênts_guerre of T_bateau_g

Create Table les_g_a of T_g_a

3.

Insert into les_bâteaux_guerre (T_porteur (10, ISE, 36000 tonnes, 700, {1, 2})

Insert into les_bâteaux_guerre (T_cannonier (11, ISE, 36000 tonnes, 20, 20cm)

Insert into les_bâteaux_guerre (10, 11)

Les Bases de données Déductives

Exercice1 :

-1. La règle quatre (4) est une règle récursive car le prédicat accessible est apparent dans la partie gauche et la partie droite.

-2. Un modèle du prédicat « accessible » :

Accessible

Spiron	Gaston	r = 16

La 1ére règle

Accessible

Candy	Spiron	r1 = 8
Candy	Gaston	r2 = 16
Spiron	Extra	r1 = 16
Gaston	Extra	r2 = 24

La 2éme règle

Accessible

Spiron	Candy	r1 = 16
Gaston	Candy	r2 = 8
Extra	Spiron	r1 = 24
Extra	Gaston	r2 = 16

La 3éme règle

Accessible

Candy	Extra	h1 = Candy
Extra	Candy	h3 = Spiron
		h2 = Extra
Spiron	Extra	h1 = Candy
Spiron	Candy	h3 = Gaston
		h2 = Extra

La 4éme règle

Un modèle pour le prédicat « accessible » :

Si on suppose que h1 et h2 des quatre (4) règles ne sont pas les mêmes alors un modèle pour ce prédicat est :

Accessible(Spiron, Gaston)
Accessible(Spiron, Extra)

Exercice2 :

-1. Les règles pour définir le prédicat Vol :

Vol(Ville1, Pays1, Ville2, Pays2) :-- Vol_Direct (Ville1, Pays1, Ville2, Pays2)

Vol(Ville1, Pays1, Ville2, Pays2) :-- Vol_Direct (Ville1, Pays1, Ville3, Pays3)
Et Vol_Direct (Ville3, Pays3, Ville2, Pays2)

Vol(Ville1, Pays1, Ville2, Pays2) :-- Vol (Ville1, Pays1, Ville3, Pays3)
Et Vol (Ville3, Pays3, Ville2, Pays2)

-2.

Vol Direct

Ville1	**Pays1**	**Ville2**	**Pays2**
Paris	France	Rome	Italie
Bruxelle	Belgique	Londre	Grande Bretagne

Exercice3 :

-a.

1) Sup (x,y) :-- Sup Direct (x,y)
   Sup (x,y) :-- Sup Direct (x,z) ∧ Sup(z,y)
2) Sup Commun(x,y,z) :-- Sup(x,y) ∧ Sup(x,z)
3) Pb Sup Commun(x,y,z) :-- Sup Direct(x,y) ∧ Sup Diect(x,z)
4) Même Niveau (x,y) :-- Pb Sup Commun (a,x,y)

-b.

Sup

x	**Y**
Ali	Mohamed
Ali	Ahmed
Mohamed	Youcef
Ali	Youcef

Même Niveau

x	**Y**
Mohamed	Ahmed

Les bases de données réparties

Exercice1 :

1-

BD parallèles:

Les données peuvent être distribuées sur plusieurs disques d'un même site, et l'exécution des requêtes peut être parallélisée sur les différentes unités de traitement (CPU) du site.

BD réparties:

Les données sont distribuées et/ou dupliquées sur différents sites du réseau (ex: internet)qui possèdent un certain degré d'autonomie. Chaque site peut comporter une BD parallèle.

Avantages des BD réparties:

• *Performance* : En rapprochant les données des applications utilisant ces données (ex : stockant les comptes des clients montréalais dans un site à localisé à Montréal), on peut réduire les coûts de transfert sur le réseau et, ainsi, augmenter la performance des requêtes sur ces données.

• *Fiabilité* : En dupliquant certaines données importantes sur plusieurs sites, on minimise l'impact d'une panne sur un site. De même, en cas de panne, on peut rediriger le traitement d'une requête vers un autre site disponible.

• *Extensibilité* : Si les besoins en espace de stockage et en puissance de traitement augmentent on peut facilement rajouter un nouveau nœud(site), sans avoir à remplacer le serveur(ex : approche Google).

2-

Stratégie de la semi-jointure :

La stratégie par semi-jointure permet de réduire le coût d'une jointure en limitant la quantité de données transférées sur le réseau.

Supposons que l'on veuille calculer T1 ⋈ T2 où la table Ti est située sur le site i. Au lieu de transférer une table complète d'un site à un autre, on envoie seulement les colonnes nécessaires à la jointure(la clé). Par exemple, on envoie $\Pi_{clé}$(T2) au site 1 et on fait la jointure avec T1: R = T1 ⋈ $\Pi_{clé}$(T2)

Ceci correspond à faire la semi-jointure entre T1et T2. Ensuite, on envoie le résultat R au site 2 pour faire la jointure avec T2: T = R ⋈ T2= T1 ⋈ T2

Les données transférées sont celles de $\Pi_{clé}$ (T2) et de R, et ont une taille potentiellement moins grande que celle de T1ou de T2.

3-

Répartition cyclique par bloc : Au lieu de disposer les blocs d'une table séquentiellement sur un même disque, la répartition cyclique les dispose en alternance sur plusieurs disques. Par exemple :

Disque 1	Disque 2	Disque 3
bloc 1	bloc 2	bloc 3
bloc 4	bloc 5	bloc 6
...	...	...

Le but de cette stratégie est de permettre la lecture / écriture de plusieurs blocs en parallèle (un dans chaque disque).

3-

Fragmentation horizontale : Chaque fragment contient un sous-ensemble de lignes de la table. Par exemple, on découpe la table Client selon la provenance(ex : province, état, etc.) d’un client.

Fragmentation verticale :Chaque fragment contient un sous-ensemble de colonnes de la table. En pratique, ce type de fragmentation est rarement employé.

4-

Avantages de la fragmentation :

- La fragmentation horizontale permet de répartir les lignes d'une table sur les sites où le traitement de ces lignes est souvent fait, réduisant ainsi les temps de transfert sur le réseau.

- En cas de panne d’un site, l’information stockée sur les autres sites reste disponible.

5-

Avantages de la duplication :

- Réduit les coûts de transfert en dupliquant sur les différentes l’information globale à tous les sites. Par exemple, les codes et les frais associés aux transactions bancaires.

- Assure la disponibilité des données dupliquées dans le cas où un ou plusieurs Sites tombent en panne.

6-

Duplication synchrone :

Une transaction modifiant des données de plusieurs sites n'est confirmée qu'au moment ou tous les sitesont confirmés les changements.

Duplication asynchrone :

Les mises à jour sont d'abord faites sur la copie primaire des tables, et les autres copies sont mises à jour en différé.

7-

Vues matérialisées :

Permet de créer une copie locale d'une table située sur un autre site distant. En somme, elles permettent d'implémenter le concept de la duplication (synchrone ou asynchrone).

8-

Optimisation dans les BD réparties

- Coût de communication: Contrairement aux BD centralisées, l'optimisation de requêtes utilisant des données sur plusieurs sites doit également tenir compte du coût de transfert sur le réseau.

- Ressources multiples: L'optimiseur doit également tenir compte de la localisation des données et des diverses ressources à sa disposition. Par exemple, plusieurs sites peuvent contribuer en parallèle à répondre à la requête selon les données qu'ils renferment.

9-

RAID1:

Le niveau RAID 1 est basé sur la duplication des données sur des disques miroirs. Cette architecture est robuste aux pannes survenant sur un ou plusieurs disques. De plus, elle permet la lecture en parallèle sur les différents disques(mais pas l'écriture). Par contre, cette architecture est plutôt gourmande en terme d'espace.

RAID 5:

Contrairement au niveau RAID 1, le niveau RAID 5 ne duplique pas les données. En revanche, ce niveau emploi la répartition cyclique par bloc ce qui permet de faire des lectures ET des écritures en parallèle. Par ailleurs, elle permet une certaine forme de fiabilité à l'aide de bits de parité stockés séparément des données.

10-

Sélection dans BD parallèles: En supposant que la table sur laquelle opère la sélection est fragmentée, on peut effectuer en parallèle une recherche sur chacun des fragments et ensuite combiner les résultats de ces recherches. Par ailleurs, si la fragmentation est faite selon la clé de sélection, on peut limiter la recherche aux fragments correspondants.

11-

Mémoire partagée: Plusieurs processeurs(CPU)partagent la même mémoire vive(RAM). L'avantage est que les processeurs peuvent communiquer efficacementà travers la mémoire RAM. Cependant, la mémoire RAM constitue un goulot d'étranglement qui limite le nombre de CPU possibles.

Disques partagés: Contrairement à la précédente, les CPU de cette architecture ont chacun leur propre RAM. Cela facilité l'extension de l'architecture (ajout de nouveau CPU) mais complexifie un peu la communication entre les processeurs. En pratique, cette architecture est celle employée le plus souvent.

Exercice 4 :

- Puisque l'application Web réside du côté de la succursale québécoise, pour accéder au catalogue français, il faut créer un lien entre la BD québécoise et la BD française :

CREATE PUBLIC DATABASE LINK serveur-bd.microdur.france.com

CONNECT TO nom_schema IDENTIFIED BY mot_de_passe;

Par ailleurs, puisque les délais doivent être minimisés, on choisit de une approche de duplication qui crée une copie locale du catalogue français dans la BD situé au Québec. Cette approche peut être implémentée à l'aide du vue matérialisée :

CREATE MATERIALIZED VIEW CatalogueFrance

REFRESH FAST ON COMMIT AS SELECT * FROM Catalogue@serveur-bd.microdur.france.com

Le paramètre ON COMMIT est employé dans ce cas pour que les changements faits au catalogue français soient immédiatement visibles à l'application. Le paramètre FAST assure une mise à jour incrémentale rapide.

- Pour rendre transparent à l'application la localisation des items, on crée une vue regroupant tous les produits :

CREATE VIEW CatalogueGlobal AS

(SELECT * FROM Catalogue) --le catalogue local

UNION

(SELECT * FROM CatalogueFrance) --le catalogue francais