BIGDATA & Bases de Données Avancées

TELLI AbdelmoutiA
Maitre de Conférences Classe A
Département d'informatique
Université de Biskra

2021-2022

Intitulé du Master : Réseaux et Technologies de l'information et de la Communication

Semestre : S3

Intitulé de l'UE : UEM 3

Intitulée de la matière : BIGDATA & Bases de Données Avancées

Crédits : 5

Coefficients : 2

Connaissances préalables recommandées : Base de données et programmation orientée objet

Mode d'évaluation : Examen 50% + Contrôle Continu 50%

Références :

- Connolly T., Begg C., Strachan A. (2004) : Database Systems A Practical Approach to Design, Implementation and Management
- Gardarin, G. Bases de Données Objet et Relationnel, Eyrolles(2001)
- Godin, R : Systèmes de gestion de bases de données par l'exemple Loze- Dion, Montréal„ 2006
- Melton, J. Simon A. SQL 1999 : Understanding Relational Language Components Morgan Kaufmann Publishers, 2001

Objectifs de l'enseignement

Ce module est la suite du module Base de données et UML donné en M1. Il permet aux étudiants de maitriser les grands standard des bases de données Objets/relationnelles SQL3 et ODMG ainsi que la gestion des données non structurées temps réel avec Hadoop.

Table des matières

Chapitre 1

Rappel : Bases de données relationnelles

The hero is a remedy for the natural weakness of children, the relational wound of adults or the historic humiliation of a nation.
Boris Cyrulnik.

Introduction

Une base de données est une collection de données opérationnelles, enregistrées et utilisées par des systèmes d'application d'une organisation particulière. La collection de données est structurée indépendamment d'une application particulière. Elle est cohérente, redondance minimale et accessible simultanément par plusieurs utilisateurs.

Nous commençons ce cours par un rappel sur la notion de BD avec le fameux modèle relationnel qui a dominé pendant longtemps le domaine des BDs, ses composantes et ses fonctions. Ensuite, nous détaillons la notion de Système de Gestion des Bases de Données (SGBD) et le son langage de modélisation (langage SQL).

1.1 Le modele relationnel

La structure d'une BD relationnelle est fondée sur la théorie des ensembles. Elle concerne essentiellement deux problématiques complémentaires :

1. **La structuration des données :** une base de données relationnelle c'est un ensemble de tables associées les unes aux autres. La conception du schéma (structures des tables, contraintes, liens entre tables) doit obéir à certaines règles et satisfaire certaines proprietés. Une théorie solide, la normalisation a été développée qui permet de s'assurer que l'on a construit un schéma correct.
2. **Les langages d'interrogation :** deux approches se sont dégagées : la principale est une conception déclarative des langages de requêtes, basées sur la logique mathématique (on formule ce que l'on souhaite, et le système décide comment calculer le résultat), et la seconde est de nature plus procédurale, et identifie l'ensemble minimal des opérateurs dont le système doit disposer pour évaluer une requête.

1.1.1 Notions de base

— **Une relation** R de degré n sur les domaines $D_1, D_2, ...D_n$ est un sous-ensemble fini du produit cartésien $D_1 \times D_2 \times ... \times D_n$.

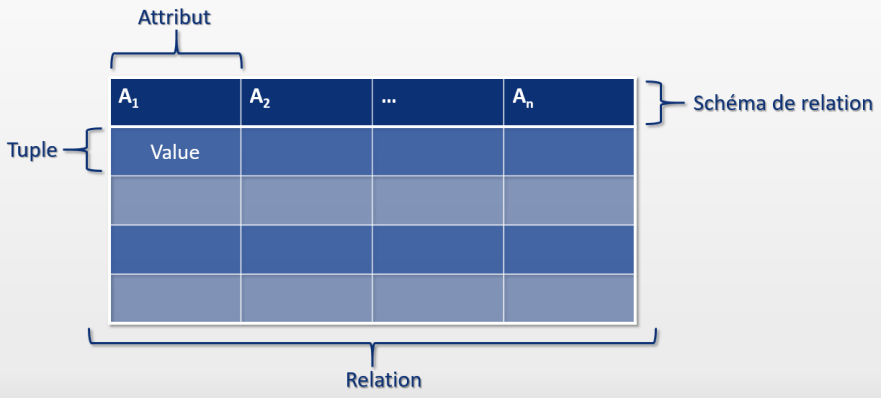
— **Le domaine** d'un attribut est un ensemble, fini ou infini, de valeurs admissibles.

— **Ue schéma** de relation est le type de données auquel correspondent les valeurs affectées aux attributs.

— **Une clé primaire** est un attribut ou un ensemble d'attribut qui caractérise de manière unique un tuple.

— **Une clé étrangère** c'est une clé primaire présente dans une relation ou une table dans laquelle elle n'est pas une clé primaire.

— **Un nuplet** $(a_1, a_2, ...a_n)$ est un élément de relation R avec les a_i les valeurs des attributs.



— Une relation est en **première forme normale (1NF)** si toutes les valeurs d'attribut sont connues et atomiques et si elle ne contient aucun doublon.

— Une relation R est en **deuxième forme normale (2NF)** si et seulement si elle est en 1NF et que tout attribut n'appartenant pas à une clé ne dépend pas d'une partie de la clé de R.

— Une relation est en **troisième forme normale (3NF)** si elle est en 2NF et si tout attribut n'appartenant pas à la clé ne dépend pas d'un attribut non clé.

1.2 Système de Gestion de Base de Données

Le SGBD est un ensemble d'outils logiciels permettant la création, la manipulation et le contrôle de bases de données. Il assure l'indépendance des programmes aux données (la possibilité de modifier les schémas conceptuel et interne des données sans modifier les programmes d'applications, et donc les schémas externes). Pour assurer ces fonctionnalités, un langage de base de données appellée "Structured Query Language" (SQL) est défini dans les SGBDs et il compose un Langage de Définition de Données (LDD) qui permet à l'utilisateur de décrire des entités avec leurs attributs et leurs liens selon des contraintes éventuelles sur ces entités. Aussi, il compose un Langage de Manipulation de Données (LMD) qui offre à l'utilisateur le moyen de rechercher, de créer, de modifier et de supprimer des informations. De plus, un Langage de Contrôle de Données (LCD) qui assure l'intégrité des donn'es ainsi que l'accès sécurisé à ces dernières.

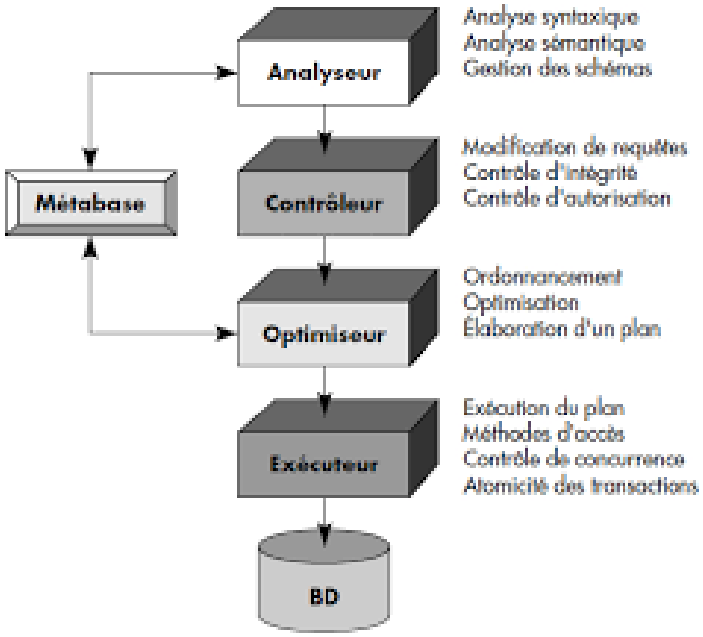


FIGURE 1.1 – Architecture général de SGBD

1.3 Conception d'une base de données relationnelle

Les grandes et les petites organisations utilisent des bases de données relationnelles pour stocker, gérer et analyser plus efficacement des informations essentielles dans différents domaines. La conception d'une base relationnelle se déroule en général selon un processus comportant plusieurs étapes (Voire Figure 1.3) :

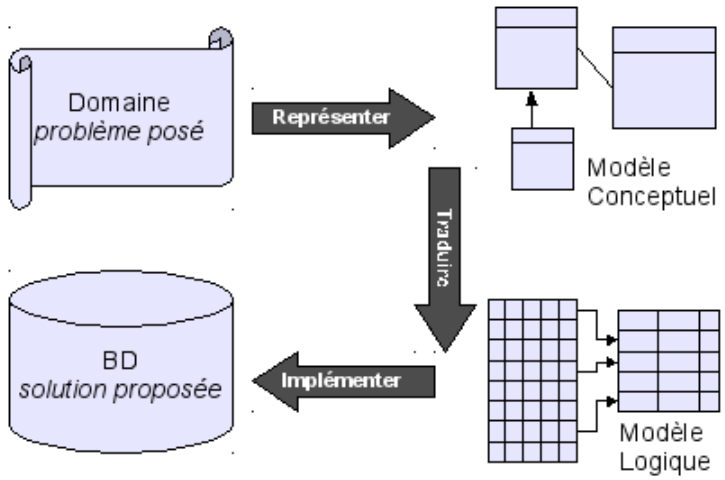


FIGURE 1.2 – Conception d'une base de données.

L'intégrité des données comprend trois aspects techniques spécifiques de la structure d'une base de données relationnelle :

- **Intégrité des entités** qui garantit qu'une table ne contient pas de records en double et que les valeurs des clés primaires de la table sont toutes uniques et non nulles.
- **Intégrité du domaine** qui garantit que l'objectif de chaque domaine est clair et identifiable et que les valeurs de chaque domaine sont valides, cohérentes et exactes.
- **Intégrité référentielle** qui garantit que les relations entre les paires de tables sont solides.

Il existe un processus systématique pour vous assurer la base de données relationnelle suit les bons principes en matière de design, qu'elle est adaptée aux besoins de l'organisation et qu'elle évite les pièges classiques :

1.3.1 Analyse des besoins

Cette étape de la conception repose sur l'analyse de l'existant et des besoins. Elle consiste à étudier le problème et à consigner dans un document, les informations sur les données, les informations sur les contraintes entre les données, les règles de gestion et de transformation de données.

1.3.2 Modèle conceptuel de données

Cette étape consiste à traduire le document d'analyse de besoins selon un modèle conceptuel. Elle décrit les objets principaux (les entités, les attributs, les relation,...), leurs caractéristiques et leurs relations grâce à une représentation schématique des données.

1.3.3 Modèle logique relationnelle

Transcription du modèle conceptuel dans un formalisme adapté à une implémentation ultérieure, au niveau physique, sous forme de base de données.

1.3.4 Modèle Physique de données

Le modèlle physique (ou interne) des données permet d'établir la manière concrète dont le système sera mis en place (pour un SGBD fixé). Il implémente la base de données dans un SGBD, à partir du modèle logique (et optimisation).

1.4 Langage de requêtes SQL

Le langage SQL quant à lui est destiné aux utilisateurs du SGBD et, à l'heure actuelle, est le langage standard d'interrogation. Il a été standardisé par l'ANSI en 1986 puis par l'ISO en 1989, 1992, 1999, 2003 et 2008. Ces standards successifs ont étendu les capacités du langage. En plus de ces standards, SQL possède de nombreux dialectes spécifiques à chaque SGBD.

Dans ce qui suit nous présentons deux sous-langages du SQL : Le langage de définition de données (DLL) et Le langage de manipulation de données (DML) en utilisant la norme SQL2 du langage largement disponibles dans les SGBDs :

1.4.1 Langage de définition de données (DDL)

Data Definition Langage (DDL) permet la modification du schéma d'une base de données. Il propose les opérations suivantes :

1. **Creation d'une base de données** :
   CREATE SCHEMA nom de la base des données ;
2. **Creation d'une table :**
   CREATE TABLE nom de table (<définitions de colonnes> <contraintes de table>) ;
3. **Ajouter d'une clé primaire** :
   CREATE TABLE nom de table **ADD PRIMARY KEY** (nom-clé) ;
4. **Suppression d'une table :**
   DROP TABLE nom de table ;
5. **Ajoute d'une colonne dans une table :**
   ALTER TABLE nom de table **ADD COLUM** nom de colonne(Taille) ;

6. **Elimination de colonne :**
   ALTER TABLE nom de table **DROP COLUM** nom de colonne.

Les contraintes d'intégrité sur la table supportés par le SQL sont :

— **CHAR :** chaînes de caractères de taille fixe.
— **VARCHAR :** chaînes de taille variable
— **INTEGER :** entiers sur 32 bits.
— **SMALLINT :** entier signé long sur 16 bits.
— **REAL :** nombres en virgule flottante simple précision.
— **NUMERIC :** nombre décimal prend par défaut la valeur de 0.
— **BIT :** chaîne de taille fixe.
— **BIT VARYING :** chaîne de taille variable.
— **DATE :** année, mois de jour.
— **TIME** : heure, minute, et seconde.
— **INTERVAL :** intervalle entre dates ou entre instants,

1.4.2 Langage de manipulation de données (MDL)

Data Manipulation Langage (MDL) permet d'ajouter les données proprement dites dans la BD. C'est le rôle du langage DML. Il propose les opérations suivantes :

1. **Extraction et modification de données d'une table :**
   SELECT <liste d'expressions> **FROM** <liste de tables>
   WHERE <conditions> **GROUP BY** <liste d'attributs>
   HAVING <conditions> **ORDER BY** <liste d'attributs>
   Sachant que les rôles des clauses sont les suivants :
   SELECT : spécifie le schéma de sortie (projection).
   FROM : précise les tables impliquées et leurs liens (produit cartésien et jointures).
   WHERE : fixe les conditions que doivent remplir les n-uplets résultats (sélection).

GROUP BY : comment regrouper des nuplets (agrégation) ?
HAVING : impose une condition sur les groupes.
ORDER BY : définit les critères de tris des résultats.

2. **Insertion de n-uplets :**
   INSERT INTO <table> [(<liste d'attributs>)] **VALUES** (<liste de valeursi) | <requete>
3. **Modifier les valeurs des n-uplets d'une table :**
   UPDATE <table> SET <liste d'affectations> [**WHERE** <condition>]
4. **Supprimer des n-uplets d'une table :**
   DELETE FROM <table> [**WHERE** <condition>]
5. **Intégrité des données au moment des mises à jour effectuées par les utilisateurs :**
   ON DELETE <NO ACTION | **CASCADE** | **SET NULL** | **SET DEFAULT**>

1.4.3 Le langage de contrôle de données (DCL)

Data Control Language (DCL) est l'ensemble de commandes de contrôle d'accès aux données. Il inclut :

1. **GRANT :** Autorisation à réaliser une opération.
2. **DENY :** Interdiction de réaliser une opération.
3. **REVOKE :** Annulation d'une commande de contrôle précédente.
4. **UPDATE :** Autorisation à modifier des enregistrements.
5. **READ :** Interdiction de modifier des enregistrements.
6. **DELETE :** Autorisation à supprimer des enregistrements.

Références du chapitre

J.L Hainaut, Bases de données : Concepts, utilisation et développement. ISBN : 9782100574100, Dunod, Paris, (2012).

A. Meier, D.H. Nguyen, Introduction pratique aux bases de données relationnelles. ISBN-10 : 2-287-25205-3, Collection IRIS, French Edition-Springer, (2006).

N. Boudjlida, Bases de données et systèmes d'information, le modèle relationnel : langages, systèmes et méthodes. ISBN : 2100070304 Dunod, Paris, 2002.

Gardarin G., "Bases de données objet et relationnel", Ed. Eyrolles, 1999 (ISBN : 2-212-09060-9).

Chapitre 2

Bases de données objet

Any unique image that you desire probably already exists on the internet or in some database ... the problem today is no longer how to create the right image, but how to find an already existing one.

Lev Manovich.

Introduction

La plupart des personnes qui travaillent régulièrement avec des bases de données utilisent des bases de données relationnelles à l'aide de systèmes de gestion de bases de données (SGBD) adaptés comme MySQL. Il existe pourtant des alternatives : les bases de données orientées objet (Bases de données objet) sont certes rarement utilisées, mais elles peuvent faire considérablement avancer certains projets.

Le modèle relationnel connait un très grand succès et sévère très adéquat pour les applications traditionnelles des bases de données. Mais, il est moins adapté aux nouvelles applications plus complexes :

— Computer-Aided Design (CAD)
— Computer-Aided Manufacturing (CAM)
— Images et de graphiques.
— Geographic Information Systems (GIS ou SIG)
— Multimédia (son, image, ...)

2.1 Définition de Bases de Données Objet

Les modèles de bases de données orientées objet est un ensemble de données est regroupé avec tous ses attributs pour former un objet. Les données sont interrogeables par ensemble au lieu de tout répartir dans différentes tables. Ainsi, les objets inclure des méthodes.

Les objets sont à nouveau regroupés en classes. Cette configuration engendre une hiérarchie entre classes et sous-classes. A l'intérieur de cette structure, les sous-classes adoptent les propriétés des classes du niveau supérieur et les complètent avec leurs propres attributs. Une base de données objets permet en plus d'interroger immédiatement tous les composants d'une unité, (ensembles de données plus complexes).

2.2 Avantages du base de données objet

— Le type de base de données le plus approprié à retenir dépend fortement de la nature du projet.
— Lorsque nous travaillons déjà avec des langages de programmation orientés objet, comme par exemple Java, une base de données objet est avantageuse.
— Les objets du code source peuvent simplement être repris dans la base de données.
— Les ensembles de données complexes s'enregistrent et s'interrogent rapidement et simplement.
— Les identifiants des objets sont attribués automatiquement.
— Bonne compatibilité avec les langages de programmation orientés objet.

2.3 Inconvénients du base de données objet

— Bien que ce modèle existe depuis les années 1980, jusqu'à présent, très peu de SGBD orientés objet ont vu le jour (la plupart des développeurs préfèrent donc utiliser les bases de données relationnelles, plus répandues, bien documentées et approfondies).
— Les bases de données orientées objet sont peu répandues.
— Les problèmes de performance (la grande complexité est engendré dans certaines situations).

2.4 Structure de Bases de Données Objet

Puisque tout objet peut être rendu persistant, on peut définir n'importe quelle structure de données. En comparaison, les BD relationnelles n'utilisent que des tuples dont les composantes sont des types primitifs. Les SGDBs relationnels étendus offrent quelques structures additionnelles (par exemple, tables imbriquées, stockage d'un objet dans un tuple), elles n'atteignent pas le degré de flexibilité offert par les SGBD OO.

Les structures suivantes sont très générales et très utiles :

— **set** : Ensemble.
— **bag / collection :** multi-ensemble similaire à un ensemble, mais permet plusieurs occurrences du même objet.
— **list :** ensemble d'éléments qui sont ordonnés
— **map :** ensemble de couples clé-valeur

2.5 Transmittance et Persistance

Objet transitoire : est un objet qui n'existe qu'en mémoire vive ; il disparaît à la fin de l'exécution du programme.

Objet persistant : est un objet qui est stocké sur disque et qui peut être chargé en mémoire vive. Il est créé durant l'exécution d'un programme et sauvegardé avant la fin de l'exécution du programme. Pour être stocké, l'objet doit avoir un nom persistant, ou bien il doit être accessible à partir d'un objet ayant un nom persistant.

Le concept de persistance transitive permet de stocker les objets persistants sur disque. Lorsqu'un objet ayant un nom persistant est sauvegardé, les objets qu'il référence sont aussi sauvegardés si leur classe est décrite comme étant admissible à la persistance ; aussi appelé persistance par accessibilité.

2.6 Hiérarchie

Une classe est un type et un ensemble d'instances de ce type. Dans les langages de programmation, une classe est juste un type. Nous utilisons le terme extension pour désigner l'ensemble des instances d'une classe.

2.6.1 Hiérarchie de types

Une classe est un type et chaque classe peut être une spécialisation d'une autre classe (la classe générale). La classe spécialisée hérite de tous les attributs et de toutes les méthodes définis dans la classe générale.

La déclaration suivante (en Java) définit la classe $C2$ comme une spécialisation de la classe $C1$ (Voire Figure 2.1).

CLASS $C2$ EXTENDS $C1$ {...}

2.6.2 Hiérarchie de classes

Une classe est un ensemble d'objets. Nous utilisons le concept de classe pour modéliser les données d'une application. Par exemple, la classe Livre dans un système de gestion d'une bibliothèque dénote l'ensemble des tous les livres de la collection de la bibliothèque.

L'expression *la classe C2 est une sous-classe de la classe C1* signifie que tous les objets de C2 appartiennent aussi à C1. En C++ ou en Java, on représente l'ensemble des objets d'une classe à l'aide d'un objet de type collection, soit un Set, un Bag, une List, ou bien un Map.

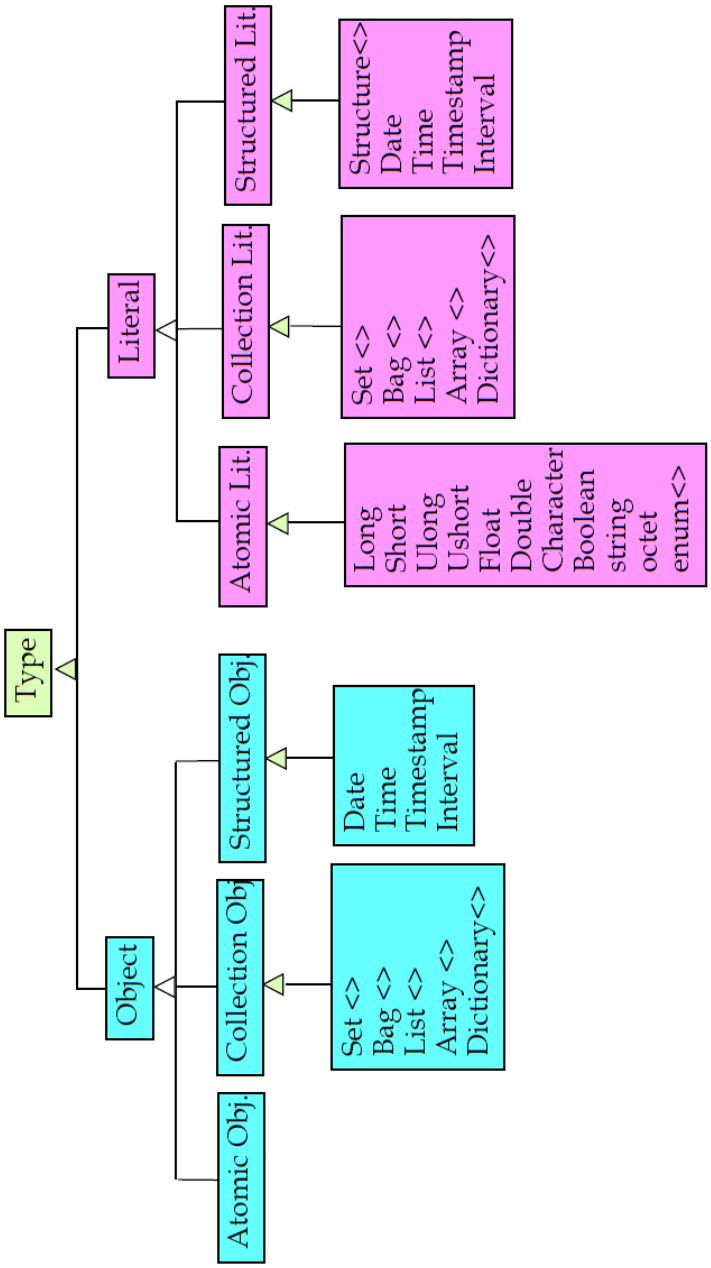


FIGURE 2.1 – Hiérarchie de Types

2.7 Le modèle objet de ODMG et JDO

Le JDO (Java Data Objects) est le norme pour la définition de SGBD orienté objet et qui remplace l'ancienne norme de l'ODMG (Object Data Management Group). Les normes ODMG permettent de propose un modèle objet et définit un langage de définition d'objets ODL (Object Definition Language). De plus, il définit un langage de d'interrogation des objets OQL (Object Query Language). Les objectifs des BD OO sont :

- Rendre les objets persistants (les objets persistent sont stockés dans un fichier) après la fin de l'exécution d'un programme.
- Partager des objets entre plusieurs programmes en parallèle.
- Lire des objets de manière transparente (persistance transitive).
- Ecrire des objets de manière transparente à l'aide du concept de transaction, qui permet aussi de préserver la cohérence des objets.

2.7.1 Architecture d'un SGBDO

L'architecture d'un SGBDO conforme à l'ODMG contient (Figure 2.2) :

- ODL (Object Definition Language) : Langage de définition de schéma des bases de données objet proposé par l'ODMG. (Equivalent des DDL - Data Definition Language des SGBD.)
- OQL(Object Query Language) : Langage d'interrogation de bases de données objet proposé par l'ODMG, basé sur des requêtes SELECT proches de celles de SQL.
- OML (Object Manipulation Language) : Langage de manipulation intégré à un langage de programmation objet permettant la navigation, l'intérrogation (OQL) et la mise à jour de collections d'objets persistants.

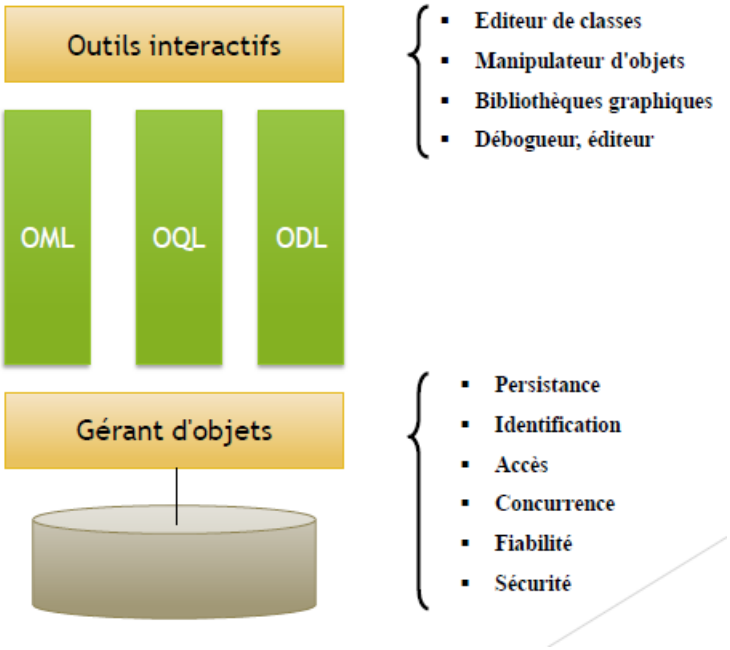


FIGURE 2.2 – Architecture d'un SGBDO conforme à l'ODMG

SGBD OO = SGBD + Lagauge de Porgramation OO

Représentation du réel	Développement
Persistence	Structure complexe
Gestion des disques	Identité
Partage des données	Encapsulation
Fiabilité des données	Classe
Sécurité	Héritage
Langages de requêtes	Redéfinition
Indépendance logique / physique	Bibliothèques de classes

La figure suivante représente un méta-modèle du modèle ODMG :

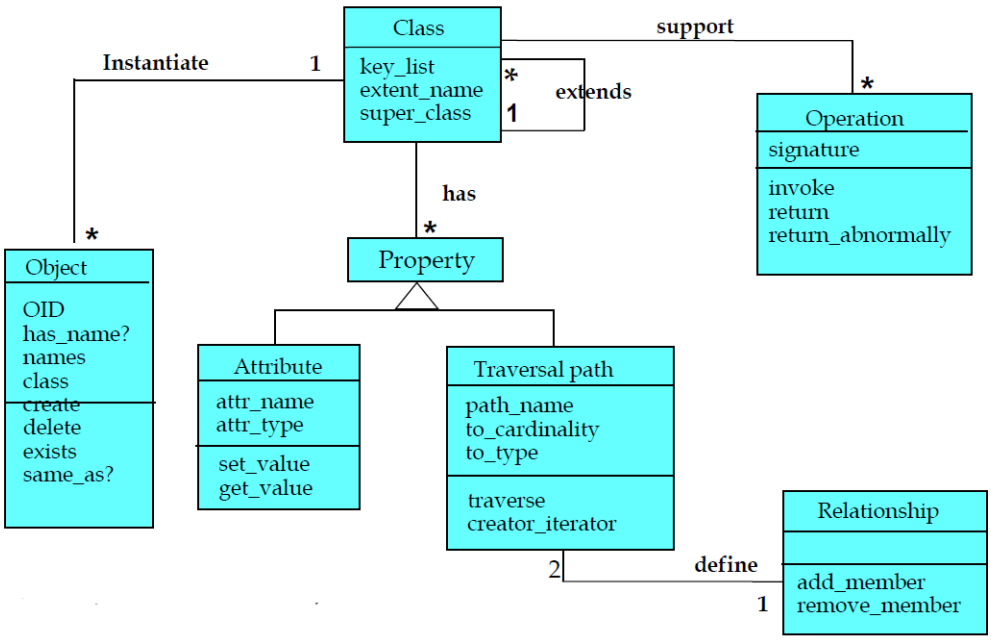


FIGURE 2.3 – Modèle ODMG

2.7.2 Les objets

Un objet est décrit selon quatre caractéristiques :

1. **Identificateur :** identifie de manière unique un objet ; c'est une valeur interne au SGBD ; il n'est pas accessible pour les programmes d'application.
2. **Nom :** c'est un nom persistant. Un programme d'application peut lire un objet en utilisant ce nom. Typiquement, nous associons un nom seulement aux collections d'objets.
3. **Durée de vie :** il y a deux valeurs possibles pour cette caractéristique : transitoire ou persistant (Voir la section 2.5).
4. **Structure :** un objet est soit atomique (défini avec les clauses interface et class) ou soit une collection d'objets. Une interface ne peut être instanciée. Une class peut être instanciée. Nous pouvons lui associer une extension dont le nom est spécifié avec le mot clé extent.

L'OMG a proposé un modèle standard pour les objets. Les BD objets nécessitent des adaptations/extensions : instances de classes, collections, associations, persistance et transactions.

2.7.3 Les collections

Pour appliquer les opérations suivantes aux collections, nous utilisons :

— **count(C)** : retourne le nombre d'éléments de la collection **C** :
select count(elements) **from** element in collection ;
— **avg(C) :** retourne la moyenne des valeurs de la collection **C** :
avg (select element.count(element) **from** element in collection)
— De même, on a aussi les fonctions **min, max, sum**.

2.7.4 Les littéraux

Un littéral est une donnée qui n'a pas d'identificateur d'objet. Il ne peut être stocké directement de manière persistante. Pour être persistant, il doit faire partie d'un objet. Nous dénotons trois types de littéraux :

1. **Atomic :** long, short, float, double, char (caractère), string (chaîne de caractères), boolean (booléen), enum (énumération).
2. **Composé :** date, interval (temps en jours, heures, minutes, secondes, milisecondes), time (heure, minute, seconde, miliseconde). Nous pouvons aussi définir ses propres structures avec l'opérateur struct.
3. **Collection :** Il existe plusieurs littéraux de type collection. Ce sont les mêmes types que pour les objets, sauf que leur nom commence par une lettre minuscule.

2.7.5 Interface

Une interface est une spécification d'un type :

- ***Super :*** héritages simple et multiple.
- ***Extent :*** clés candidates.
- ***Attributs :*** attribute <type> <nomattr>.
- ***Associations :*** relationship <type> <nomasso> <nom-interface>.
- ***Méthodes :***
  - <type-retourné> <nommeth> (<type-paramêtre> : <type>, . . .).
  - raise (<type-d-exception>).
  - <type-paramêtre> : in, out, inout
- ***Classe :*** interface + une implantation particulière du type ou dans un des LMD disponibles.

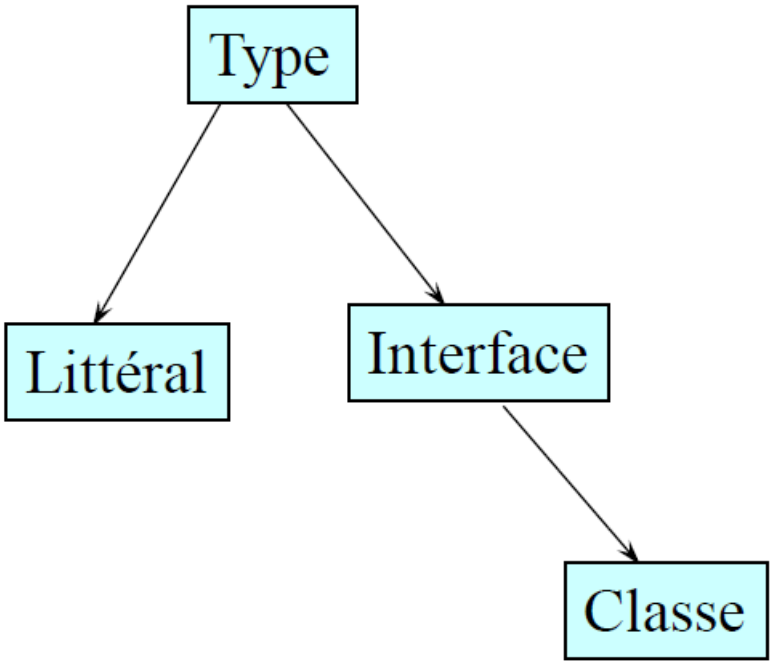


FIGURE 2.4 – Type, Interface et classe

2.8 Langage de définition des objets (ODL)

Il permet de définir un schéma de base de données objet. Il est composé de déclarations de classes et d'interfaces. Le langage supporte l'héritage (entre classes, avec le mot clé extends, et entre une classe et une interface, ou entre deux interfaces).

Une classe est un type et une collection d'objets. Sa définition comporte donc les éléments suivants :

- **Nom de classe :** Spécifié par le mot clé class.
- **Nom de la collection (optionnel) :** Spécifié par le mot clé extent.
- **Clés (optionnel) :** Spécifiées par le mot clé key. Nous pouvons associer des clés (une ou plusieurs) à un objet. Une clé est une liste d'attributs.
- **Attributs :** Spécifiés par le mot clé attribute.
- **Relations :** Spécifiées par le mot clé relationship. Pour une relation unidirectionnelle, on utilise le concept d'attribut. Si un programme d'application modifie une direction de la relation, le SGBD modifie l'autre direction, afin que les deux directions soient cohérentes.
- **Opérations :** Spécifiées en donnant la signature de l'opération.

Exemple

Interface Personne {

attribute string nom ;

attribute string prenom ;

relationship Appart habite inverse Appart : : loge ;

relationship Voiture Possede inverse Voiture : :Appartient ;

} ;

2.9 Langage de définition des requêtes (OQL)

Il permet d'interroger une base de données objet. La syntaxe est inspirée de SQL. L'énoncé *SELECT* de base retourne un bag d'objets ou de littéraux :

SELECT element **FROM** elements in collection **WHERE** condition ;

Le symbole element est une variable d'itération sur les éléments de la collection. Il est similaire à un alias de table en SQL. Cet énoncé retourne un bag <string> de tous les éléments de la collection dont le condition est vrai.

Nous pouvons utiliser le mot clé **DISTINCT** pour éliminer les doublons :

SELECT DISTINCT element **FROM** elements in collection **WHERE** condition ;

L'énoncé suivant retourne une collection d'objets :

SELECT element **FROM** elments in collection **WHERE** condition ;

Pour sélectionner un sous-ensemble d'attributs d'un objet nous utilisons l'opérateur **STRUCT** :

SELECT STRUCT (condition sur les attributs) **FROM** elemenst in collection **WHERE** condition ;

Pour ordonner les éléments du résultat ou le résultat est une liste :

SELECT STRUCT (élément d'attributs) **FROM** elements in collection **WHERE** condition **ORDER BY** critère d'ordonnancement ;

Dans la clause **FROM**, nous pouvons spécifier une expression qui est une collection.

2.9.1 Navigation

Pour accéder aux objets auxquels un objet est relié dans un énoncé SELECT et dans toutes leurs clauses (FROM et WHERE), en naviguant à travers les attributs, les relations et les opérations (Voire 2.5).

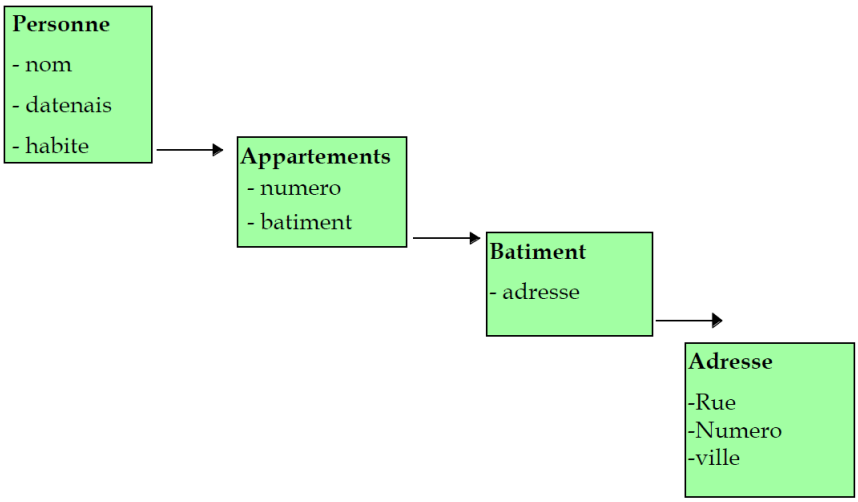


FIGURE 2.5 – Exemple de Navigation

Pour tester si un attribut est défini avec le prédicat is-defined (expresssion) :

SELECT STRUCT (élément d'attributs) **FROM** elements in collection **WHERE is-defined** (expresssion) ;

Pour omettre l'opérateur STRUCT s'il n'y a pas deux expressions qui ont le même nom. Le nom de chaque attribut du résultat est le dernier nom de l'expression :

SELECT (élément d'attributs) **FROM** elements in collection ;

— L'opérateur **.** appliqué à une valeur indéfinie retourne une valeur indéfinie.
— Les opérateurs de comparaison **=, !=, <, >, <= >=** retourne faux si une de leurs opérandes est indéfinie.
— Le prédicat **is-defined(expresssion)** retourne vrai si l'expresssion est définie.
— Le prédicat **is-undefined(expresssion)** retourne vrai si l'expresssion est indéfinie.

Pour inclure un SELECT dans un SELECT :

SELECT STRUCT (élément d'attributs : (**SELECT** élément d'attributs)) **FROM** elements in collection ;

De même, on peut naviguer aussi avec les attributs et les relations.

2.9.2 Les expressions booléennes

Trois types d'expressions booléennes quantifiées sont existe :

1. **in** collection : retourne vrai si element appartient à la collection.
2. **for all** element **in** collection : condition : retourne vrai si pour tout element de la collection et la condition est vraie.
3. **exists** element **in** collection : condition : retourne vrai si il existe un element de la collection tel que la condition est vraie.

2.9.3 Les groupes

Pour regrouper les tuples résultants de la clause **FROM-WHERE** d'un SELECT, nous utilisons l'énoncé suivant :

SELECT * FROM x in collection **group by** critère de regroupement ;

2.10 BD relationnelle vs modèle objet

Dans la modélisation des données, nous utilisons habituellement le diagramme entité-relation pour faire un modèle conceptuel des données d'un système. Il existe aussi la base de données non relationnelle, aussi appelée NoSQL. Alors que la première stocke les données les données dans des tables, la deuxième les stocke au format clé-valeur, dans des documents, en colonnes, en graphiques ou autres. Cependant, les bases de données NoSQL ne sont pas en train de supplanter les bases relationnelles mais viennent les compléter, dans un contexte de croissance exponentielle de données. Si nous choisissons d'implémenter le système avec une BD OO comme Object Store, nous pouvons utiliser les algorithmes de traduction suivants :

2.10.1 Algorithme de traduction des entités

1. Chaque entité ordinaire E est traduite en deux classes :
   - Une classe **Tuple-E** pour représenter les instances de l'entité.
   - Une classe **Coll-E** pour contenir l'ensemble des instances de l'entité.

   Nous allons choisir une collection appropriée en fonction du type d'accées requis. Typiquement, une entité est représenté par un Map ou la clé est une des clés candidates et la valeur la classe E représentant les instances de l'entité.
2. Chaque entité faible F est traduite de manière similaire à une entité ordinaire, sauf que la clé du Map de la collection est formée de la clé de l'entité père et de la clé partielle de l'entité faible.

   Une autre alternative consiste à utiliser la classe Tuple-E de l'entité père E de l'entité faible F et d'y ajouter un attribut Map<K,Tuple-F>, ou K désigne la clé de l'entité faible.

2.10.2 Algorithme de traduction des associations

1. Chaque association binaire ordinaire A entre les entités E1 et E2 (sans classe associative) est représentée en ajoutant des attributs aux classes Tuple-E1 et Tuple-E2.
2. Si une instance de E1 est associée a au plus une instance de E2, alors nous utilisons un attribut Tuple-E2.r, ou r est le nom de rôle
3. Si une instance de E1 est associée a au plus une instance de E2, alors il faut choisir une collection appropriée.
4. Comme on navigue généralement une association dans les deux directions, il faut aussi ajouter un attribut dans Tuple-E2 en pour permettre d'accéder aux instances de E1 associées à une instance de E2, en utilisant la même stratégie.
5. Pour chaque association binaire A entre les entités E1 et E2 et comprenant une classe associative est représentée comme suit. On crée une classe Tuple-A qui comprend ses attributs propres ainsi que des attributs vers les entités associées, selon la cardinalité de l'association, exactement comme dans une association binaire sans attributs. On ajoute dans les classes Tuple-E1 et Tuple-E2 des attributs pour accéder aux instances de la classe associatives.
6. Pour l'association n-aire il faut créer une classe associative et la traiter comme une classe associative binaire.

Pour avoir l'équivalent d'une table, il faut : définir une classe avec les attributs. Puis, rendre cette classe admissible à la persistance avec le post-processeur et définir une variable de type collection pour contenir tous les objets de la classe ; nous utilisons un Map avec les attributs de clé primaire comme clé du Map. Enfin, donner un nom persistant à cette variable.

Pour ajouter d'un tuple, il faut démarrer une transaction ; puis faire un getRoot sur la racine de la table pour lire la collection et ajouter l'objet dans la collection ; avant de commit de la transaction.

2.11 Les systèmes Relationnel-Objet (SQL3)

Oracle Database est considéré comme un système de gestion de base de données relationnelle (SGBDR) qui depuis l'introduction du support du modèle objet dans sa version 8 peut être aussi qualifié de système de gestion de base de données relationnel-objet (SGBDRO).

Les types d'objet Oracle sont des types définis par l'utilisateur qui permettent de modéliser des entités du monde réel telles que des clients et des bons de commande en tant qu'objets dans la base de données.

La technologie objet Oracle est une couche d'abstraction construite sur la technologie relationnelle Oracle. De nouveaux types d'objets peuvent être créés à partir de n'importe quel type de base de données intégré et de n'importe quel type d'objet, référence d'objet et type de collection précédemment créés. Les métadonnées des types définis par l'utilisateur sont stockées dans un schéma disponible pour SQL, PL/SQL, Java et d'autres interfaces.

2.11.1 Type de données abstrait

Type de données créé par le concepteur d'un schéma relationnel-objet, qui encapsule des données et des opérations sur ces données. Ce concept est à rapprocher de celui de classe d'objets :
TYPE nom-type : <attribut1 :typeAttribut1,

...,

attributN :typeAttributN,

=methode1 :(paramètres) typeRetourné1,

=...,

=methodeM :(paramètres) typeRetournéM >

2.11.2 Déclaration de type

Les utilisateurs ont à leur disposition les domaines usuels de SQL tels que Var, Char, 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 :

CREATE TYPE nom-type **AS OBJECT** (

nom-attribut1 type-attribut1

...

MEMBER FUNCTION nom-fonction1 (parametre1 **IN|OUT**

type-parametre1, ...)

RETURN type-fonction1 ...)

CREATE TYPE BODY nom-type **IS**

MEMBER FUNCTION nom-fonction1 (...)

RETURN type-fonction1 IS

BEGIN

...

END ;

MEMBER FUNCTION nom-fonctionn ...

...

END ;

END ;

2.11.3 Les types OBJECT

Pour créer des valeurs structurées et/ou des objets, nous utilisons l'instruction suivante :

CREATE TYPE nom-type **AS OBJECT** (nom-attribut1 nom-type1 ,..., 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.

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 O**F T-Enfant ;

2.11.4 Creation de 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).

Pour Créer une table d'objets nous utilisons l'instruction :

CREATE TABLE nom-table **OF** nom-type

Cette table contiendra des objets, identifiés par leurs oids et structurés selon le type "nomtype".

Pour associer aux tables quelle que soit leur sorte les contraintes usuelles de SQL : **PRIMARY KEY** (nom-col*), **UNIQUE** (nom-col*), **CHECK** (condition) et **FOREIGN KEY** (nom-col*) **REFERENCES** nom-table [(nom-col*)] [action]

2.11.5 Déclaration de METHODES

Chaque type OBJECT peut avoir des méthodes :

CREATE TYPE nom-type **AS OBJECT** (déclaration des attributs, déclaration des signatures des méthodes). Pour la signature d'une méthode :

MEMBER FUNCTION nom-méthode (nom-paramêtre1 [**IN / OUT**] type1 ,..., typen)

RETURN type **MEMBER PROCEDURE** nom-méthode (nom-paramêtre1 [**IN / OUT**] type1 ,..., typen)

2.11.6 Hiérarchie de type 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.

Il y a plusieurs définitions formelles possibles de la classe des objets complexes. Nous posons d'abord l'existence des éléments suivants :

- D_i : un ensemble fini de domaines ;
- Nous noterons $D = \cup\, D_i$;
- A : un ensemble dénombrable d'attributs ;
- ID : un ensemble dénombrable d'identificateurs.

Nous définissons ensuite l'ensemble des valeurs V subdivisé en :

- Valeurs de base (base values) : ce sont
  - nil : utilisé pour représenter une valeur non définie ;
  - Les éléments d $\in$ D.
- Valeurs-ensembles (set values) : ce sont les sous-ensembles de ID.
- Valeurs-tuples (tuple values) : ce sont les fonctions (partielles) de A vers ID.

Un objet est une paire (id, v) ou : id $\in$ ID est l'identificateur de l'objet, et v $\in$ V est la valeur de l'objet.

Dans tout ensemble d'objets considéré, deux objets distincts ne peuvent avoir le même identificateur.

2.12 Exercices

Exercice 00 : Nous désirons conserver dans une base de donnée géographique la description d'un réseau routier :

- Les points d'intersection qui sont les points ou plusieurs routes se croisent ou qui correspondent à un changement de caractéristiques d'une route. Un point d'intersection est caractérisé par ses coordonnées géographiques (latitude et longitude).
- Les segments de route qui sont des tronçons de route situés entre deux points d'intersection. Outre ses points d'origine et de destination, une information de catégorie (deux bandes, quatre bandes, . . .) caractérise chaque segment.
- Une route est désignée par un identifiant (N83, C07, . . .) et est décrite par un ensemble de segments. De plus pour chaque route on conserve la désignation de l'autorité (région, commune, . . .) qui la gère.

Cette information ne peut être représentée naturellement dans le modèle relationnel, car on ne peut y définir de relations entre éléments qui sont des tuples ou des ensembles. On est donc amené à introduire de relations auxiliaires et des identifiants d'objets que l'on doit gérer explicitement.
Question 0 : Quelle sont les caractéristiques des bases de données OO ?
Question 1 : Proposez un schéma possible pour ce modèle ?
Il faut noter que l'usage des identifiants permet par exemple d'avoir deux segments de route entre les mêmes points, ou de représenter sans ambiguïté qu'un même segment fait partie de plusieurs routes.
Question 2 : Représentez un contenu possible de cette base de données ?
Question 3 : Associez des procédures (Méthodes) de manipulation des objets complexes dans la BD de routes ?
Question 4 : Définir quelque sous-classes particulières de classe route ?
Question 5 : Définir les routes comme des objets complexes ?
Devoirs : Conception d'un modèle de BDO pour un domaine spécifique.

Exercice 01 : Nous désirons créer une base de donnée objet pour la description d'objet ETUDIANT.

0. Expliquer les différences entre un objet persistant, une objet transitant, et un objet à versions ?

1. Donner le code SQL3 qui permet d'implémenter le modèle t-etudiant ?

2. Créer une table objet relationnelle etudiant de type t-etudiant ?

3. Insérer par de deux manières un individu dans la table etudiant ?

4. Afficher et récupérer l'identifiant de l'objet etudiant ?

Exercice 02 : Nous désirons créer une base de donnée objet en utilisant un SGBDO pour la description d'objets VOITURE et PERSONNE.

0. C'est quoi un schéma de BDOO ?

1. Donner le code SQL3 qui permet d'implémenter les modèles t-voiture et t-personne ?

2. Créer les tables voiture de type t-voiture et personne de type t-personne ?

3. Insérer les individus suivantes dans la table voiture :
(2012, 'peugeot', '207') et (2021, 'renault', 'cleo')

4. Insérer les individus suivantes dans la table personne :
(1, 'Moutia', 'Telli', null) et (2, 'khaled', 'khaldi', (mat.voiture=400022016)

5. Afficher les tables voiture et personne ?

6. Afficher les numéros et les marques de voitures ?

7. Créer un table relationnelle (non pas objet relationnelle) ?

8. Insérer l'individu (1, 'Daniel', 'Telli', 777,'mercedes', 'e250') dans la table rpersonne.

9. Créer une table (liste de voitures) pour le type t-voiture ?

10. Créer une table liste de prénoms

11. Créer une table personnes inclut une liste de prénoms et liste de voitures ?

12. Créer une collection pour l'objet voiture ?.

13. Insérer l'individu ('Moutia', ('Karim', 'Amin'), ('11/11/1999'), ('Dacia', '01/01/2020', 128), ('Mégane', '01/01/1998', 179)) ?

14. Supprimer la personne 'Moutia' ayant la voiture 128 ?

15. Ajouter une fonction qui calcule l'age d'une personne ?

2.13 Solutions

Réponse 0 : Objets complexes, Identificateurs d'objet, Héritage, Encapsulation, Liaison tardive, Complétude au niveau du calcul (toutes les requêtes calculables doivent être expressibles), Le système doit être extensible (permettre la définition de nouvelles classes), Le traitement de grandes quantités de données doit être possible, Une gestion des transactions est nécessaire (parallélisme, récupération après erreur), et il doit être possible de poser aisément des requêtes simples.

Réponse 1 : un schéma possible serait :

POINTS(ID-P, LATITUDE, LONGITUDE)

SEGMENTS(ID-SEG, ID-P1, ID-P2, CATEGORIE)

ROUTES(ID-R, ID-LISTE SEG, AUTORITE)

LISTES-SEGMENTS(ID-LISTE-SEG, ID-SEG)

Réponse 2 : un contenu possible de la base de données :

POINTS

ID-P	LATITUDE	LONGITUDE
P1	lat1	long1
...	...	...

SEGMENTS

ID-SEG	ID-P1	ID-P2	CATEGORIE
SEG1	P11	P12	cat1
...	...	...	...

ROUTES

IDR	IDLISTE-SEG	AUTORITE
R1	LS1	autorite1
...	...	...

LISTES DE SEGMENTS

ID-LISTE-SEG	ID-SEG
LS1	SEG1
...	...

Réponse 3 : Les procédures de manipulation aux objets (méthodes) sont :

— Méthodes qui pourraient être associées à une route.

— Sélectionner un élément constituant (les segments, l'autorité).

— Dessiner la route sur une carte.

— Modifier la route (ajouter un segment).

Réponse 4 : Quelque sous-classes particulières de classe route :

— Routes nationales

— Autoroutes européennes

— Autoroutes nationales

Réponse 5 : Pour définir les routes en tant qu'objets complexes, nous considérons un ensemble de domaines de base contenant :

— Dlat : le domaine des latitudes,

— Dlong : le domaine des longitudes,

— Did : le domaine des identifiants de routes,

— Dcat : le domaine des catégories,

— Daut : le domaine des autorités.

Un réseau routier est représenté par des objets de la manière suivante :

— Un point d'intersection est un objet tuple de type :
Tpoint = (LATITUDE, Dlat), (LONGITUDE, Dlong).

— Un segment est un objet tuple de type :
Tseg = (POINT1, Tpoint), (POINT2, Tpoint), (CATEGORIE, Dcat).

Exercice 2 :

Réponse 0 : Les différences entre les type des objets :

Objet persistant : Objet stocké dans la BD dont la durée de vie est supérieure au programme qui le crée.

Objet transitant : Objet restant en mémoire, dont la durée de vie ne dépasse pas celle du programme qui le crée.

Objet à versions : Objet dont l'historique des instances crées (successivement ou simultanément) est gardé dans la base sous forme de versions consultables et modifiables. La gestion des versions permet de revenir à un état antérieur de l'objet avant modification.

Réponse 1 : Création d'un type objet ETUDIANT

CREATE TYPE t-etudiant **AS OBJECT** (

mat **number**(10),

nom **varchar**(50),

prénom **varcha**r(50)

) ;

Réponse 2 : Creation d'une table etudiant

CREAT TABLE etudiant of t-etudiant ;

Réponse 3 : Insertion d'un individu dans la table etudiant

1. **INSERT INTO** etudiant **VALUES** (1,'Moutia', 'Telli') ;
2. **INSERT INTO** etudiant **VALUES** (t-etudiant(2,'Nour', 'Hadi')) ;

Réponse 4 : Affichage et récupération de l'identifiant de l'objet etudiant

SELECT * **FROM** etudiant ;

SELECT ref(e), e.* **FROM** etudiant e ;

Exercice 3 :

Réponse 0 : Schéma de BDOO est la description d'une BD en termes de classes, d'attributs et d'opérations ainsi que d'associations entre les classes. Elle est généralement représentée dans un diagramme.

Réponse 1 : Création d'un type objet VOITURE et PERSONNE

CREATE TYPE t-voiture **AS OBJECT**(

mat **number**(15),

marque **varchar**(15),

modele **varchar**(15)) ;

CREATE or **REPLACE TYPE** t-personne **AS OBJECT**(

num **number**(10),

nom **varchar**(25),

prenom **varchar**(25),

veh ref t-voiture) ;

Réponse 2 : Creation d'un table voiture et personne
CREAT TABLE voiture of t-voiture (mat **primary key**) ;
CREAT TABLE personne of t-personne (num **primary key**) ;

Réponse 3 : Insertion des individus dans la table voiture
INSERT INTO voiture **VALUES**(2012, 'peugeot', '207') ;
INSERT INTO voiture **VALUES**(2021, 'renault', 'cleo') ;

Réponse 4 : Insertion des individus dans la table personne
INSERT INTO personne **VALUES** (1, 'Moutia', 'Telli', null) ;
INSERT INTO personne **VALUES** (2, 'khaled', 'khaldi', (select ref(v) **FROM** voiture v **WHERE** mat=400022016)) ;

Réponse 5 : Affichage de tables voiture et personne
SELECT * **FROM** voiture **UNION SELECT** * **FROM** personne ;

Réponse 6 : Affichage des numéros et des marques de voitures
SELECT p.num, p.veh.marque **FROM** personne p ;

Réponse 7 : Creation d'un table relationnelle (pas objet relationnelle)

CREAT TABLE rpersonne (

num **number**(10),

nom **varchar**(25),

prenom **varchar**(25),

veh t-voiture) ;

Réponse 8 : Insertion un individu dans la table rpersonne
INSERT INTO rpersonne **VALUES** (1, 'Daniel', 'Telli', t-voiture(777,'mercedes', 'e250')) ;

Réponse 9 : Création d'une table (liste de voiture) pour le type t-voiture
CREATE or **REPLACE TYPE** liste-voiture **AS TABLE** of t-voiture ;
Réponse 10 : Création d'une table (liste de prénoms) pour le type t-prénoms t-voiture
CREATE TYPE liste-prenom **AS VARRAY**(10) of **varchar**(15) ;

Réponse 11 : Création d'une table personnes

CREATE TABLE personnes (nom **varchar**(15)

prenoms liste-prenom,

date-naissance **Date**,

voitures liste-voiture) ;

Réponse 12 :

NESTED TABLE voiture **STORE AS** voiture ;

Réponse 13 :

INSERT INTO personne **VALUES**('Moutia', liste-prenom('Karim', 'Amin'), to-date('16/03/1975', 'dd/mm/yyyy'), liste-voiture(t-voiture('Dacia', '01/01/2020', 128), t-voiture('Mégane', '01/01/1998', 179))) ;

Réponse 14 :

DELETE FROM personne p **WHERE** p.nom='Moutia' and p.voitures.no=128 ;

Réponse 15 :

ALTER TYPE t-personne **ADD MEMBER FUNCTION** age

RETURN NUMBER IS

n **number** ;

BEGIN

n := **trunc**((sysdate-self.datenaiss)/365) ;

RETURN n ;

END age ;

Références du chapitre

S.W. Dietrich and S.D. Urban, Fundamentals of Object Databases : Object-Oriented and Object-Relational Design, ISBN : 9781608454761, Morgan & Claypool, (2011).

J.L. Harrington, Object-Oriented Database Design Clearly Explained, ISBN : 0123264286, Morgan Kaufmann, (1999).

Ramakrishnan et Gehrke, Database management systems, ISBN 0072465638, 3ème edition, McGraw-Hill, USA (2003).

Frappier, Marc. "Exploitation de bases de données relationnelles et orientées objet IFT287."

Chapitre 3

Bases de données réparties

Somewhere in that database my name sat in its own little niche, the name of a reject, undisciplined and worthless. Just the way I liked it.
Ilona Andrews.

Introduction

L'évolution de techniques informatiques depuis les vingt dernières années permet d'adapter les outils informatiques à l'organisation des entreprises. La puissance de ordinateurs et de stations de travail, la fiabilité et la souplesse des SGBD relationnels, les performances des réseaux permettent d'envisager une répartition des ressources informatiques tout en préservant l'essentiel (la cohérence des bases de données).

3.1 Problématique

Les pressions pour la distribution sont :

— Devient impératif de décentraliser l'information (des multinationales).
— Augmentation du volume de l'information (14 fois de 1990 à 2000).
— Augmentation du volume des transactions (10 fois dans les 5 prochaines années).
— Besoin de serveurs de BDs qui fournissent un bon temps de réponse sur des gros volumes de données.

D'autre part, pour améliorer le débit des E/S, il est nécessaire de :

— Partitionnement des données,
— Accès parallèle aux données,
— Utiliser plusieurs noeuds (avec un bon coût/ performance), et les faire communiquer par un réseau.

3.2 Définition

Une base de données répartie (BDR) est un ensemble structuré et cohérent de données, stocké sur des processeurs distincts, et géré par un système de gestion de bases de données réparties (SGBDR) (Voire Figure 3.1).

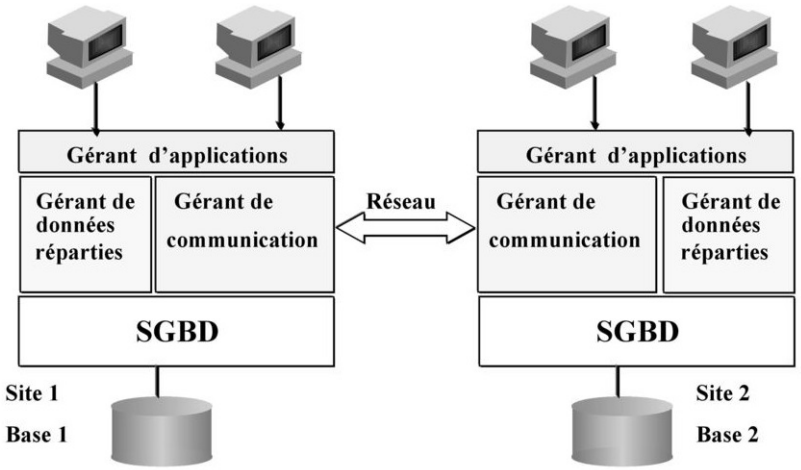


FIGURE 3.1 – Architecture de base de données répartie

BDR est un ensemble de bases de données (BD) manipulé comme s'il s'agissait d'une BD unique. Cependant, chaque BD est gérée par des systèmes de gestion de BD (SGBD) autonomes qui sont localisés sur des sites éventuellement distants.

Le SGBDR repose sur un système informatique réparti qui est constitué d'un ensemble de processeurs autonomes appelés sites (mini ou micro-ordinateurs, stations de travail,...) reliés par un réseau de communication (local ou public) qui leur permet d'échanger des données. Un SGBDR suppose en plus que les données soient stockées sur deux processeurs au moins, ceux-ci étant dotés de leur SGBD propre. Ainsi, dans l'exemple d'une configuration constituée de trois processeurs interconnectés, dont l'un est chargé de la gestion des données (serveur base de données), la base de données n'est évidemment pas répartie bien que l'on soit en présence d'un système réparti.

La BDR est décrite par un schéma global qui contient la localisation des données et qui permet ainsi d'aiguiller un traitement sur les processeurs détenant les données à traiter.

Lorsque les BD qui composent la BDR reposent sur des modèles de données différents (hiérarchique, réseau, relationnel, orienté objet), on parle de BDR hétérogène. Par conséquence, le SGBDR est également hétérogène puisqu'il est constitué de SGBD de types différents.

3.3 Fonctionnalités du SGBD répartie

SGBD répartie est un système de gestion de base assure : la définition des données locales/réparties, exécution des transactions (locales : accès aux données sur site et globales : accès sur plusieurs sites), contrôle de concurrence, reprise après panne et l'optimisation de question. Ce sont les fonctionnalités classiques des BD avec les fonctionnalités additionnelles suivantes :

1. Transparence de la localisation via des mécanismes de fragmentation ou d'agrégation.
2. Décomposition des requêtes et agrégation des réponses aux sous-requêtes générées.
3. Cohérence des mises à jour multibase grâce au protocole de validation à deux phases.
4. Réplication des données sur les sites pour optimiser leur accès.
5. Partage accru de données et ressources.
6. Fiabilité et disponibilité accrues.
7. Optimisation de requêtes réparties.

3.4 Objectifs de la répartition de bases de données

Les bases de données réparties ont une architecture plus adaptée à l'organisation des entreprises décentralisées. Elles assurent les objectifs suivants :

- **Autonomie** de chaque site avec l'absence de site privilégié.
- **Fiabilité :** les bases de données réparties ont souvent des données répliquées. La panne d'un site n'est pas très importante pour l'utilisateur, qui s'adressera à autre site.
- **Performances :** 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.
- **Accroissement :** l'accroissement se fait par l'ajout de machines sur le réseau.
- **Transparence :**
  - Vis à vis de la localisation des données.
  - Vis à vis de la fragmentation.
  - Vis à vis de la réplication.
- **Indépendance :**
  - Vis à vis du matériel.
  - Vis à vis du système d'exploitation.
  - Vis à vis du réseau.
  - Vis à vis du SGBD.

3.5 Système de Gestion de Bases de Données Réparti

Une base de données centralisée est gérée par un seul SGBD, est stockée dans sa totalité à un emplacement physique unique et ses divers traitements sont confiés à une seule et même unité de traitement. Par opposition, une base de données distribuée est gérée par plusieurs processeurs, sites ou SGBD.

Un système de bases de données réparties ne doit donc en aucun cas être confondu avec un système dans lequel les bases de données sont accessibles à distance. Il ne doit non plus être confondu avec une multibase ou une BD fédérée. Dans une multibase, plusieurs BDs interopérent avec une application via un langage commun et sans modèle commun. Dans une BD fédérée, plusieurs BDs hétérogènes sont accédées comme une seule via une vue commune.

Du point de vue organisationnel nous distinguons deux architectures :

1. **Architecture Client-Serveur :** les serveurs ont pour rôle de servir les clients. Dans l'environnement de BD client-serveur on a : le contexte centralisé où un client s'adresse à un serveur, le contexte multibase où un client s'adresse à plusieurs serveurs, et le contexte réparti où un SGBD est client des autres SGBD de l'architecture répartie.

2. **Architecture Pair-à-Pair (Peer-to-Peer, P2P) :** par ce terme on désigne un type de communication pour lequel toutes les machines ont une importance équivalente.

3.6 Conception d'une BD répartie

L'intérêt de cette démarche est de constituer une BDR à partir de BD existantes. Évidemment, c'est dans le cadre d'une telle démarche que se pose plus particulièrement le problème de la coopération de BD hétérogènes. Par exemple, on peut supposer qu'une BDR permette de faire coopérer certaines banques de données publiques

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.

Le problème de conception est cependant différent selon que l'on crée de toutes pièces une BDR (démarche descendante) ou bien que l'on constitue une BDR par agrégation de bases de données existantes (démarche ascendante).

3.6.1 Conception descendante (Top down design)

Nous commençons par définir un schéma conceptuel global de la base de données répartie. Puis, nous distribuons 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. Cette approche est intéressante quand on part du néant.

Aux niveaux conceptuel et externe, la BDR est perçue comme une base de données centralisée, les processus de conception classiques pour bases de données centralisées s'appliquent au niveau externe global et au niveau conceptuel global (Voire Figure 3.2).

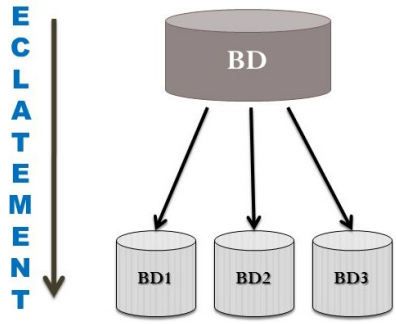


FIGURE 3.2 – Conception descendante d'une BDR.

Toute la difficulté réside dans le niveau interne global où, considérant la BDR comme un ensemble de relations : la fragmentation des relations en unités de localisation, et la localisation de ces fragments dans le réseau.

Fragmentation

Pour fragmenter une relation globale sans perte d'information, il suffit d'appliquer à cette relation l'opération algébrique de sélection (fragmentation horizontale) ou celle de projection (fragmentation verticale). Les opérations de jointure $\bowtie$ et d'union $\cup$ permettent ensuite de reconstituer la relation initiale.

Il est possible de spécifier des fragments en utilisant un langage de type SQL pour spécifier des vues comme suit :

CREATE FRAGMENT nom de frgment **AS SELECT** attribut1,...,
attributn **FROM** nom Table **WHERE** condution ;

NB : L'opérateur $\bowtie$ représente une jointure sur les clés primaires sans duplication d'une des deux clés dans le schéma résultat.

Localisation des fragments

Dans la démarche descendante, la localisation des fragments est transparente pour l'utilisateur. Ainsi, la répartition des fragments dans les différents sites du réseau répond à des critères essentiellement techniques tels que le coût de stockage, le coût de mise à jour, et la performance d'accès.

Par exemple, chaque fois qu'un fragment est dupliqué dans le réseau, les applications peuvent accéder plus aisément à ce fragment mais, en contrepartie, le coût de mise à jour est plus élevé (modification des différentes copies). Les SGBDR proposent généralement des modèles d'allocation des données qui sont paramétrables en fonction des contraintes liées à l'application (taux de mise à jour, temps de réponse, etc.).

3.6.2 Conception ascendante (Bottom up design)

Cette 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 (Voire Figure 3.3).

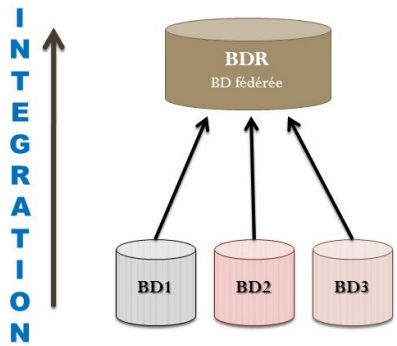


FIGURE 3.3 – Conception ascendante d'une BDR.

3.7 Niveaux de répartition des données

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

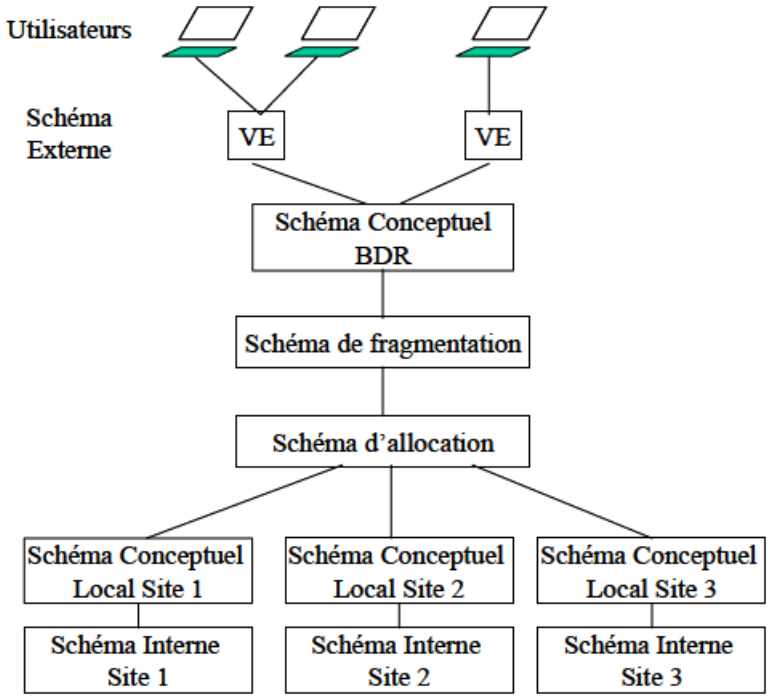


FIGURE 3.4 – Niveaux de répartition des données.

— **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.

3.8 Fragmentation

La fragmentation est le processus de décomposition d'une base de donnée en un ensemble de sous-bases de données. Cette décomposition doit être sans perte d'information. Elle peut être coûteuse s'il existe des applications qui possèdent des besoins opposés. Les règles de fragmentation sont les suivantes :

1. **La complétude :** pour toute donnée d'une relation R, il existe un fragment $R1$ de la relation R qui possède cette donnée.

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

Le principe est de baser la fragmentation sur l'ensemble des requêtes d'interrogation ou de mise à jour prédéfinies. Il faut extraire de ces requêtes toutes les conditions de sélections, les attributs projetés et les jointures.

3.8.1 Répartition des classes d'objet

Cette technique consiste en la répartition de classes qui peuvent être réparties sur différents fragments. Toutes les occurrences d'une même classe appartiennent ainsi au même fragment. L'opération de partitionnement est la définition de sous-schémas, et l'opération de recomposition est la réunion de sous-schémas.

3.8.2 Répartition des occurrences

Les occurrences d'une même classe peuvent être réparties dans des fragments différents : l'opérateur de partitionnement est la sélection (σ), et l'opérateur de recomposition est l'union ($\cup$)

3.8.3 Répartition des attributs

Toutes les valeurs des occurrences pour un même attribut se trouvent dans le même fragment. Une fragmentation verticale est utile pour distribuer les parties des données sur le site où chacune de ces parties est utilisée : l'opérateur de partitionnement est la projection (Π), et l'opérateur de recomposition est la jointure.

3.8.4 Répartition des valeurs (hybride)

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, et l'opération de recomposition est une combinaison de jointures et d'unions.

3.9 Les transactions dans une BDs réparties

Une trasaction est une séquence d'operations (lecture/ ecriture) qui doit être exécutée dans son intégralité, amenant la BD d'un état cohérent a un autre état cohérent. Si il n'est pas possible d'acceder a un etat coherent, rien ne doit etre fait.

Une transaction est une unité de travail définie dans une application et exécutée par rapport à une base de données. Elle est mise en œuvre via une suite d'opérations qui font passer la base de données d'un état A (antérieur à la transaction) à un état B postérieur et des mécanismes permettent d'obtenir que cette suite soit à la fois atomique, cohérente, isolée et durable (ACID).

Les propriétés ACID de transaction

Une transaction est donc composée d'une suite de requêtes dépendantes à la base qui doivent vérifier les propriétés suivants :

1. **Aatomicité : (Atomicity)** une transaction doit être traitée comme une seule opération, le SGBD doit assurer que toutes les actions de la transaction sont exécutées ou bien qu'aucune. Si l'exécution d'une transaction est interrompue à cause d'une panne quelconque, le SGBD doit être capable de décider des actions à entreprendre après la réparation de la panne : soit compléter la transaction en exécutant les actions restantes, soit défaire les actions qui avaient déjà été exécutées avant la panne (journal des actions).
2. **Cohérence (Consistency) :** La transaction doit faire passer la base de données d'un état cohérent à un autre. En cas d'échec, l'état cohérent initial doit être restauré. La cohérence de la base peut être violée par un programme erroné ou un conflit d'accès concurrent entre transactions.
3. **Isolation (Insulation) :** Les résultats d'une transaction ne doivent être visibles aux autres transactions qu'une fois la transaction validée, afin d'éviter les interférences avec les autres transactions. Les accès concurrents peuvent mettre en question l'isolation.
4. **Durabilité (Durability) :** c'est la propriété qui assure que lorsqu'une transaction a été confirmée (COMMIT), ses effets deviennent permanents Importance : après un arrêt externe (panne) le SGBD doit s'assurer que :
   — Les effets d'une transaction validée apparaissent dans la base de données après la reprise.
   — Les effets d'une transaction annulée n'apparaissent pas dans la base de données après la reprise.

3.10 Mise à jour de 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 jours 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 :

1. **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.
2. **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.
3. **Modification :** Rechercher les tuples, les modifier et les déplacer vers les bons fragments si nécessaire.

3.11 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.

3.12 Exercices

Exercice 1 : Soit ETUD la table relationnelles suivante :

ETUD

ID	NOM	AGE	GENRE
1	Ahmed	22	M
2	Mohamed	21	M
3	Daniel	20	
4	Lila	20	F
5	Meriem	19	F

0) Quelle sont les types de fragmentations d'une BD.

1) Faites une fragmentation (ETUD1, ETUD2) de la relation ETUD selon les prédicats p1 et p2 tels que : p1 : AGE <20 et p2 : AGE>20.

2) Pourquoi le résultat de la fragmentation (ETUD1, ETUD2) ne remplit pas les règles de la fragmentation correcte.

3) Modifiez donc les prédicats p1 et p2 pour qu'ils puissent partitionner la relation ETUD en respectant les règles de la fragmentation correcte.

Soit MOY la table relationnelles suivante :

MOY

ID	MOYENNE
1	9
2	15
3	12
4	7
5	10

4) Faites une fragmentation (MOY1, MOY2) de la relation MOY selon les prédicats p'1 et p'2 tels que : p'1 : MOY <10 et p'2 : MOY=>10.

5) Faites une fragmentation (ETUDM1, ETUDM2) de la relation ETUD tels que : ETUDM1 = ETUD x MOY1 et ETUDM2 = ETUD x MOY2.

6) Faites une fragmentation : ETUD'1 = (ID, NOM), ETUD'2 =(ID, AGE).

Exercice 2 : Soit ETUD la table relationnelles suivante :

ETUD

ID	NOM	AGE	GENRE
1	Ahmed	22	M
2	Mohamed	21	M
3	Danial	20	
4	Lila	20	F

0) Quelle est la différence entre la réplication synchrone et la réplication asynchrone ?

1) Faites une fragmentation (ETUD1, ETUD2) de la relation ETUD selon les prédicats p1 et p2 tels que : p1 : GENRE = 'F' et p2 : GENRE ='M'.

2) Quel type cette fragmentation (ETUD1, ETUD2) ? Est-elle correcte ? Justifier ?

3) Faites une fragmentation (ETUD11, ETUD12) de la relation ETUD1 tels que : ETUDE11= (ID, NOM) et ETUDE12=(ID, AGE) ?

4) Quel type de cette fragmentation (ETUD11, ETUD12) ? Est-elle correcte ? Justifier ?

5) Faites une fragmentation (ETUD21, ETUD22) de la relation ETUD2 tels que : ETUDE21= (ID, NOM, AGE) et ETUDE22=(ID, AGE, GENER) ?

6) Quel type de cette fragmentation (ETUD21, ETUD22) ? Est-elle correcte ? Justifier ?

Exercice 3 : Soit ARTICLE la table relationnelles suivante :

ARTICLE

ID	NomArticle	Prix	Qnt
01	Pantalon	220	500
02	Chaussures	210	220
03	Chemise	200	50
04	Manteau	200	120

0) 0. Définir la fragmentation correcte ?

1) Pour chaque type de fragmentation (Correcte ou non correcte), proposer un exemple appliqué sur la table ARTICLE ?

3.13 Solutions

Exercice 1.

Réponse 0 : Les types de fragmentations d'une BD sont : fragmentation verticale, fragmentation horizentale, et fragmentation hybride.

Réponse 1 : ETUD1 $=\sigma_{(Age<20)}$ ETUD et ETUD2 $= \sigma_{(Age>20)}$ ETUD

ID	NOM	AGE	GENRE
5	Meriem	19	F

ID	NOM	AGE	GENRE
1	Ahmed	22	M
2	Mohamed	21	M

Réponse 2 : Cette est incomplète. Donc, elle est incorrecte.

Réponse 3 : Modifier les prédicats p1 et p2 tels que : p1 : AGE <=20 et p2 : AGE>20.

Réponse 4 : MOY1= $\sigma_{(MOY<10)}$ MOY, MOY2 = $\sigma_{(MOY=>10)}$ MOY

ID	MOYENNE
1	9
4	7

ID	MOYENNE
2	15
3	12
5	10

Réponse 5 : ETUDM1 =ETUD x MOY1, ETUDM2 =ETUD x MOY2

ID	NOM	AGE	GENRE	MOY
1	Ahmed	22	M	9
4	Lila	20	F	7

ID	NOM	AGE	GENRE	MOY
2	Mohamed	21	M	15
3	Daniel	20	M	12
5	Meriem	19	F	10

Réponse 6 :ETUD'1 =$\Pi_{(ID,NOM)}$ ETUD ETUD'2 =$\Pi_{(ID,AGE)}$ ETUD

ID	NOM
1	Ahmed
2	Mohamed
3	Daniel
4	Lila
5	Meriem 19

ID	AGE
1	22
2	21
3	20
4	20
5	19

Exercice 2.

Réponse 0 : Ces deux techniques diffèrent sur la façon de mettre à jour les différentes répliques d'un objet :

— **Synchrone :** une transaction qui fait une mise-à-jour doit modifier toutes les répliques avant de valider.
Avantage : toutes les copies ont toujours la même valeur.
Inconvénient : cher et long.

— **Asynchrone :** les mises à jour sont faites périodiquement.
Avantage : moins cher.
Inconvénient : données pas toujours fraîches, des incohérences possibles, mais acceptables dans la plupart des cas.

Réponse 1 : ETUD1 $=\sigma_{(GENRE='F')}$ ETUD, ETUD2 $= \sigma_{(GENRE='M)}$ ETUD

ID	NOM	AGE	GENRE
3	Danial	20	F
4	Lila	20	F

ID	NOM	AGE	GENRE
1	Ahmed	22	M
2	Mohamed	21	M

Réponse 2 : Fragmentation horizontale correcte (Complète, Reconstructible, Disjointe).

Réponse 3 : ETUD11=$\Pi_{(ID,NOM)}$ ETUD1 ETUD12 =$\Pi_{(ID,AGE)}$)

ID	NOM
3	Danial
4	Lila

ID	AGE
3	20
4	20

Réponse 4 : Fragmentation hybride incorrecte. Elle est incomplète.

Réponse 5 : ETUD21=$\Pi_{(ID,NOM,AGE)}$ ETUD2 ETUD22 =$\Pi_{(ID,AGE,GENRE)}$)

ID	NOM	AGE
1	Ahmed	22
2	Mohamed	21

ID	AGE	GENRE
1	22	M
2	21	M

Réponse 6 : Fragmentation hybride incorrecte. Elle est non disjointe.

Exercice 3.

Réponse 0 : Une fragmentation R est correcte SSI :
Complète : Chaque élément de R doit se trouver dans un fragment.
Reconstructible : Doit pouvoir recomposer R à partir de ses fragments.
Disjointe : Chaque élément de R ne doit pas être dupliqué.

Réponse 1 :

- Fragmentation horizontale correcte :
  ARTICLE1=$\sigma_{(Qnt>200)}$ et ARTICLE2=$\sigma_{(Qnt<200)}$.
- Fragmentation horizontale incorrecte :
  ARTICLE1=$\sigma_{(Qnt>200)}$ et ARTICLE2=$\sigma_{(Qnt>300)}$.
- Fragmentation verticale correcte :
  ARTICLE1=$\Pi_{(ID,Qnt)}$, ARTICLE2=$\Pi_{(ID,NomArticle)}$, et ARTICLE3=$\Pi_{(ID,Prix)}$ et ARTICLE2=$\sigma_{(Qnt<200)}$.
- Fragmentation verticale incorrecte :
  ARTICLE1=$\Pi_{(ID,Qnt)}$, ARTICLE2=$\Pi_{(ID,NomArticle,Qnt)}$.

Références du chapitre

H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil and P. O'Neil. A critique of ANSI SQL isolation levels, In Proceedings of ACM SIGMOD International Conference on Management of Data, 1–10, ACM Press, (1995).

M.J. Cahill, U. Röhm, A.D. Fekete, Serializable Isolation for Snapshot Databases, SIGMOD '08 : Proceedings ACM SIGMOD international conference on Management of data, 729–738, doi.org/10.1145/1376616.1376690, (2008).

V. DESFONTAIN. Introduction aux bases de données réparties. Université de Technologie de Compiègne, (2000).

A. D. Jhingran, N. Mattos, and H. Pirahesh, Information integration : A research agenda. IBM Systems J., 41(4) :555–562 (2002).

L. Liu, M.T. Özsu (Eds.) Encyclopedia of Database Systems,Springer Science+Business Media, LLC 2009 (USA), ISBN : 978- 0-387-35544-3, (2009).

M.T. Özsu, P.Valduriez, Principles of Distributed Database Systems, 3ème Edition, ISBN 10.1007/978-1-4419-8833-8-5, Springer science Business Media, LLC (2011).

Chapitre 4

Bases de Données Parallèles

The security of computers and the Internet is a horrible and dangerous mess. Every week we hear about breaches of databases of Social Security numbers and financial information and health records, and about critical infrastructure being insecure.

Matt Blaze.

Introduction

L'approche centralisée permet à un ensemble d'applications distribuées d'accéder de façon efficace à un serveur de base de données unique. Toutefois, cette approche souffre des limitations traditionnelles des systèmes centralisés. Les développements remarquables faits dans des domaines aussi divers que les réseaux de communication, les mini et micro-ordinateurs ont permis à l'approche bases de données réparties de devenir une solution alternative à la centralisation.

Outre ce besoin de décentralisation, d'autres problèmes étaient posés par la suite. Il s'agit en particulier des volumes de données qui s'accroissent de jour en jour ainsi que des requêtes complexes. La solution qui s'imposait était alors de combiner la gestion de bases de données et le traitement parallèle afin d'augmenter la performance et la disponibilité des données. Et c'est ainsi que sont apparus les SGBDs parallèles.

Ce chapitre passe en revue les principaux concepts des systèmes de bases de données parallèles.

4.1 Architectures Matérielles

Le choix d'une architecture destinée à supporter un SGBD parallèle est guidé par le souci d'atteindre tous les objectifs de traitements parallèles des transactions et des opérations tout en garantissant le meilleur rapport prix/performances. Trois architectures ont été utilisées comme support pour les SGBDs parallèles. Nous décrivons dans ce qui suit ces trois architectures et les architectures hybrides qui tentent de combiner les avantages de chaque architecture :

4.1.1 Architecture à Mémoires Partagées (Shared Memory)

Dans cette architecture, la mémoire est accessible et partagée par l'ensemble des processeurs de la machine (Voire Figure 4.1).

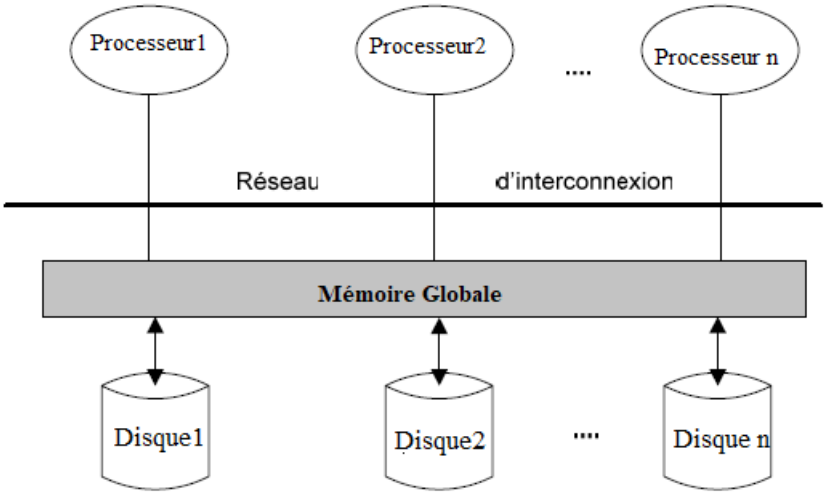


FIGURE 4.1 – Architecture à mémoires partagées.

Chaque processeur a un coût uniforme d'accès à la mémoire. Un bus relie entre eux les éléments matériels de la machine : processeurs, mémoires et disques. Cette architecture permet un accès rapide aux données. De plus, comme les informations de contrôle et les informations globales concernant les données sont accessibles par tous les processeurs, les principes de conception ne sont pas très différents de ceux d'un SGBD centralisé. Par contre, la complexité du réseau d'interconnexion due à la nécessité de relier chaque processeur à tous les modules augmente le coût d'une telle architecture et limite son nombre de processeurs.

Parmi les systèmes de gestion de bases de données qui utilisent ce type d'architecture : XPRS, DBS3, Oracle et Informix

4.1.2 Architecture à Disques Partagés (Shared Disk)

L'architecture à disques partagés est basée sur l'hypothèse que chaque processeur a sa mémoire centrale privée, mais les disques sont partagés par tous les processeurs. Ceci facilite la répartition de la charge de travail tandis que la mémoire reste distribuée pour ne pas pénaliser l'extensibilité (fig. 4.2).

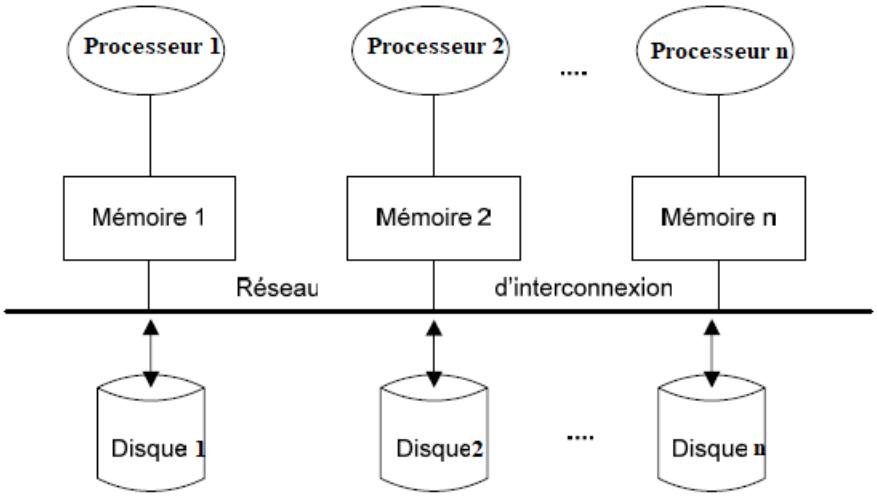


FIGURE 4.2 – Architecture à disques partagés.

Le disque partagé offre donc une excellente utilisation des ressources. De plus, il fournit un bon niveau de tolérance aux pannes avec l'ensemble des données qui restent accessibles tant qu'il y a au moins un noeud qui fonctionne. Cependant, comme la mémoire n'est pas partagée, une page disque peut être dupliquée sur plusieurs noeuds de la machine. Ceci nécessite la gestion de cohérence entre toutes les copies d'une page. Le coût de la maintenance de cette cohérence est bien sûr l'inconvénient majeur de cette architecture.

Parmi les systèmes de gestion de bases de données s'exécutant sur des machines à disques partagés : Oracle Parallel Server.

4.1.3 Architecture à Mémoires Distribuées (Shared Nothing)

Dans cette architecture, dite aussi à partage de rien, excepté le réseau d'interconnexion, rien n'est partagé entre les processeurs. Chaque processeur a sa propre mémoire centrale et son propre disque (Voire figure 4.3).

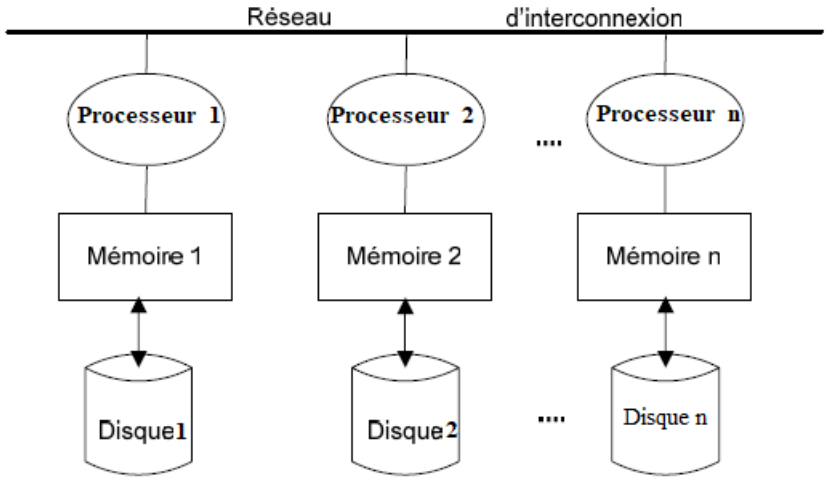


FIGURE 4.3 – Architecture à mémoires distribuées

Le partage de ressources matérielles et logicielles entre les différents processeurs est donc limité au minimum. Cette architecture est ainsi facilement extensible. De plus, une bonne disponibilité des données est assurée grâce à leur réplication au niveau de plusieurs noeuds. Cependant, pour équilibrer la charge de travail entre les processeurs, la redistribution des données est très complexe.

Parmis les systèmes de gestion de bases de données utilisent ce type d'architecture : NonStopSQL de Tandem, Gamma et PRISMA/DB

4.1.4 Architecture Hybride

Une architecture hybride est une combinaison des architectures à mémoires distribuées et à mémoires partagées (Voire Figure 4.4).

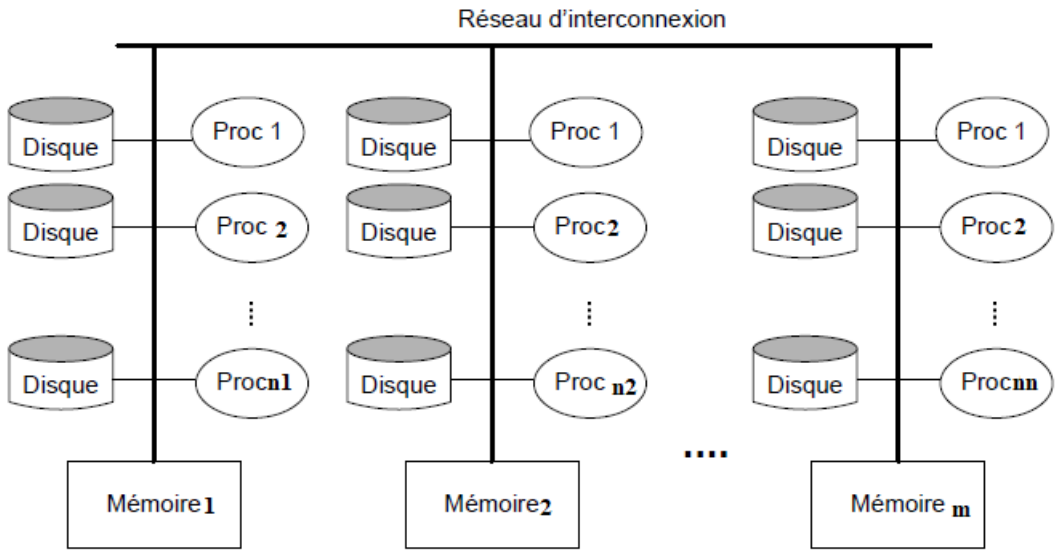


FIGURE 4.4 – Architecture Hybride

Une telle architecture combine les avantages de chaque architecture, et compensent leurs inconvénients respectifs. L'architecture hybride combine l'équilibre de charge des architectures à mémoires partagées et l'extensibilité des architectures à mémoires distribuées.

4.2 Partitionnement des Données

Cette section présente, le partitionnement de leurs données. Le partitionnement des données, appelé aussi fragmentation, est l'un des facteurs critiques qui font d'un SGBD un système performant ou non.

4.2.1 Schémas de Partitionnement

Le partitionnement d'une table permet à des transactions portant sur des tuples différents de cette table de s'exécuter de façon concurrente. Ainsi, les applications n'accèdent qu'à des sous ensembles de tables et non a des tables entières. Il existe trois schémas fondamentaux du partitionnement : le partitionnement horizontal et le partitionnement vertical.

1. **Partitionnement Horizontal :** Le partitionnement horizontal fragmente une table selon ses tuples. Chaque fragment a alors un sous ensemble de tuples de la table. Il existe deux manières de partitionner une table horizontalement :
   — Partitionnement primaire : est effectué en utilisant des prédicats définis sur la table elle même.
   — Partitionnement dérivé : est effectué en utilisant des prédicats définis sur une autre table.
2. **Partitionnement Vertical :** Le partitionnement vertical d'une table T produit un ensemble de fragments $(T_1, ..., T_n)$ contenant chacun un sous ensemble des attributs de T en plus de la clé primaire de la table T. Le partitionnement vertical est beaucoup plus complexe que le partitionnement horizontal. Ceci est essentiellement dû au grand nombre d'alternatives possibles.
3. **Partitionnement Hybride :** Dans la plupart des cas, un partitionnement horizontal ou vertical simple d'une base de données ne satisfait pas les demandes des applications. Donc, un partitionnement horizontal peut être suivi par un partitionnement vertical ou vice versa.

4.2.2 Stratégies de Partitionnement de Données

Après avoir présenté les schémas de partitionnement des données dans un SGBD parallèle. Plusieurs stratégies de partitionnement ont été proposées :

1. **Partitionnement Circulaire (Round Robin) :** est implanté dans les systèmes RAID (Redundant Arrays of Independant Disks). Il est considéré comme la stratégie la plus simple pour le partitionnement des données. Cette stratégie garantit une distribution uniforme des données. Elle permet de partitionner une table en fragments de même taille entre des noeuds distribués.

   Si N est le nombre de partitions d'une table, alors le i-ème tuple de cette table est affecté à la partition de numéro (i mod N). Les tuples sont placés en fonction de leur numéro d'ordre. La première donnée sera placée sur le premier processeur. La seconde sur le deuxième et ainsi de suite jusqu'à placer la nouvelle donnée sur le dernier processeur (Voire Figure 4.5).

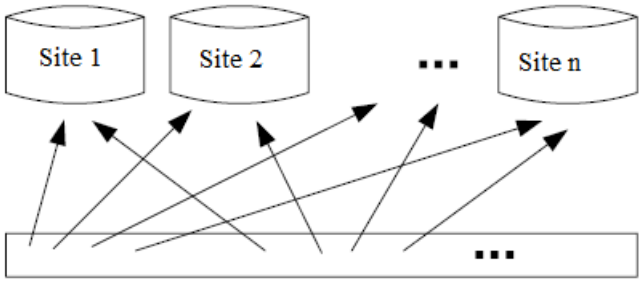


FIGURE 4.5 – Partitionnement circulaire

L'avantage majeur du partitionnement circulaire est l'optimisation du temps de réponse des requêtes tout en permettant un accès séquentiel et parallèle aux données. Cependant, cet accès nécessite le parcours de toutes les partitions d'une table, ce qui entraîne inévitablement une dégradation des performances. De plus, le problème avec cette technique est que les applications désirent souvent accéder aux tuples de façon associative, ce qui n'est pas possible avec cette technique.

2. **Partitionnement par Hachage (Hashage Partitioning) :** La stratégie de partitionnement par hachage distribue l'ensemble des tuples en utilisant une fonction de hachage h sur un ensemble d'attributs. La fonction de hachage retourne le numéro du serveur dans lequel le tuple sera rangé. Elle convient aux applications qui veulent accéder aux données de façon séquentielle et associative. L'accès associatif aux tuples ayant une valeur d'attribut spécifique peut être dirigé vers un seul disque, évitant ainsi le surcoût dû à un lancement de requêtes sur plusieurs disques (Voire Figure 4.6).

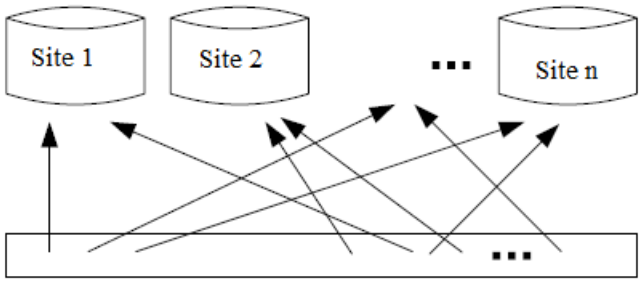


FIGURE 4.6 – Partitionnement par hachage

Il existe deux méthodes de hachage : hachage statique et hachage dynamique. Tant qu'il n'y a pas de débordements, le hachage statique est très simple et donne d'excellentes performances. Cependant, les débordements dégradent rapidement les performances. Le hachage dynamique permet de faire grandir progressivement un fichier haché saturé en distribuant les tuples dans de nouvelles régions allouées à une table. Deux méthodes du hachage dynamique s'avèrent importantes : le hachage extensible et le hachage linéaire.

— **Hachage Extensible :**

Le hachage extensible est une méthode de hachage dynamique qui associe à chaque fichier un répertoire des adresses de paquets. Au départ, M bits de la fonction de hachage sont utilisés pour adresser le répertoire.

A la première saturation d'un paquet, le répertoire est doublé, et un nouveau paquet est alloué au fichier. Le paquet saturé est distribué entre l'ancien et le nouveau paquet, selon le bit suivant (M+1) de la fonction de hachage. Ensuite, tout paquet plein est éclaté en deux paquets, lui-même est un nouveau paquet alloué au fichier. L'entrée du répertoire correspondant au nouveau paquet est mise à jour avec l'adresse de ce nouveau paquet si elle pointait encore sur le paquet plein. Sinon, le répertoire est à nouveau doublé.
Le hachage extensible consiste alors à éclater un paquet plein et à mémoriser l'adresse des paquets dans un répertoire adressé par les (M+P) premiers bits de la fonction de hachage, où P est le nombre d'éclatements maximal subi par les paquets.

— **Hachage Linéaire :** est une méthode de hachage dynamique nécessitant la gestion du débordement et consistant à éclater le paquet pointé par un pointeur courant quand un paquet est plein, et mémoriser le niveau d'éclatement du fichier afin de déterminer le nombre de bits de la fonction de hachage à appliquer avant et après le pointeur courant. L'avantage du hachage linéaire est alors la simplicité de l'algorithme d'adressage du répertoire : on utilise (M+P) bits de la fonction de hachage. Si on est positionné avant le pointeur courant, on utilise un bit de plus, sinon on lit l'adresse du paquet dans le répertoire. A chaque éclatement, le répertoire s'accroît d'une seule entrée. Le hachage linéaire est utilisé par plusieurs SGBDs : SQL Server, et MySQL.

3. **Partitionnement par Intervalle (Range Partitioning) :** La stratégie de distribution par intervalle distribue les tuples d'une table en fonction de la valeur d'un ou de plusieurs attributs, formant alors une valeur unique dite clé, par rapport à un ordre total de l'espace des clés. Les fonctions de hachage en général n'offrent pas ce type de partition. Les attributs formant la clé sont appelés alors aussi attributs ou clés de partitionnement. La table ou le fichier partagé sont dits ordonnés.

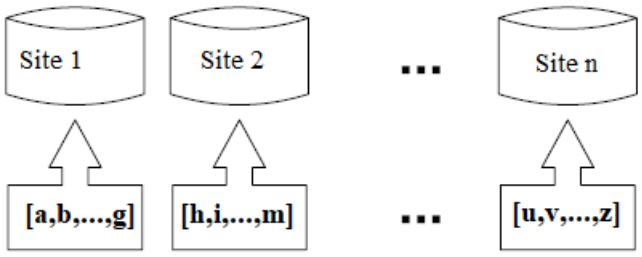


FIGURE 4.7 – Partitionnement par intervalle

La méthode générale du partitionnement par intervalle consiste en deux phases. La première phase permet de diviser l'espace des valeurs des attributs sélectionnés en intervalles puis chaque intervalle est affecté à un noeud. La deuxième phase assigne chaque tuple t au noeud n si les valeurs d'attributs spécifiés du tuple t appartiennent à l'intervalle de n. La stratégie de partitionnement par intervalle convient aux requêtes dont le prédicat implique les attributs de partitionnement et donc les requêtes par intervalle. Cependant, elle ne garantit pas un équilibre de charge entre les noeuds. Une des méthodes de partitionnement par intervalle les plus utilisées est la méthode des arbres B :

— **Arbres B :** Cette méthode générait une partition statique. Un fichier grandissant nécessitait périodiquement une réorganisation globale. Cette contrainte posait des problèmes évidents aux applications et usagers. Les clés d'un arbre B sont triées selon l'ordre post-fixé induit par l'arbre, afin de permettre les recherches en un nombre d'accès n'excédant pas le niveau de l'arbre B. L'utilisation des arbres B originaux pour réaliser des tables indexées conduit néanmoins à un taux d'accès aux disque durant le parcours des nœuds internes qui peut être aisément amélioré.

4.3 Traitement Parallèle des Requêtes

Pour atteindre de hautes performances, un SGBD exploite le parallélisme pour les requêtes exprimées dans un langage d'assertions. Le traitement parallèle est utilisé afin de maximiser le débit d'un système distribué en réduisant le temps total des requêtes d'une part, et de minimiser le temps de réponse des requêtes d'autre part. Le partitionnement de données fournit des opportunités de traitement parallèle.

Deux formes de parallélisme sont obtenues directement à partir du partitionnement des données : Le parallélisme inter-requête permet l'exécution parallèle de plusieurs requêtes, chacune agissant sur des partitions différentes, et le parallélisme intra-requête permet l'exécution parallèle de plusieurs opérations relationnelles à l'intérieur de la requête même. Une même opération peut être traitée par plusieurs processeurs. Il s'agit du parallélisme intra-opération.

Trois mécanismes de base pour réaliser le parallélisme des traitements :

1. **Parallélisme Indépendant (Independent Parallelism) :** permet d'exécuter en parallèle plusieurs opérations indépendantes d'une même tâche ou requête (Voire Figure 4.8).

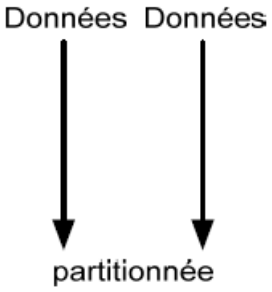


FIGURE 4.8 – Parallélisme indépendant

2. **Parallélisme en Tuyau (Pipelined Parallelism) :** est utilisé dans le cas où le résultat d'une opération constitue les données d'entrée pour l'opération suivante (Voire Figure 4.9).

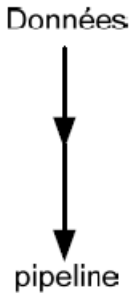


FIGURE 4.9 – Parallélisme en tuyau

3. **Parallélisme par Fragmentation (Partitioned Parallelism) :** permet de fragmenter la charge de travail entre plusieurs processus. Cela nécessite à la fin une fusion des résultats partiels pour produire le résultat final (Voire Figure 4.10).

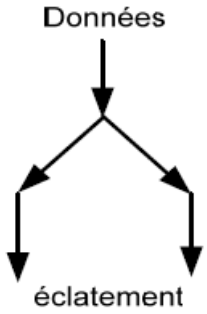


FIGURE 4.10 – Parallélisme par fragmentation

4.4 Exercices

Exercice 01 :

0) Expliquez la différence entre les architectures à mémoire partagée et à disque partagés ?

Soit une relation $R(A, B)$ tenant sur 2000 pages, R est initialement triée sur l'attribut A. On décide d'utiliser une base de données parallèle. Celle-ci est formée de 10 nœuds ($N_1 ... N_{10}$) ayant chacun une mémoire vive (RAM) de 100 pages. R est allouée aux nœuds disposant d'un disque de taille suffisante.

1) Montrer que le fragment de R alloué à chaque nœud est lui-même trié sur le site A.

2) Donner la stratégie pour évaluer la requête suivante, posée sur le nœud N_1 et évaluer son coût en entrée/sorties (nb de pages) et son coût en terme de pages de données transférées sur le réseau. Détailler les coûts étape par étape : SELECT avg(A)*max(A) FROM R ;

4.5 Solutions

0) Mémoire partagée vs Disque partagé

Mémoire partagée	Disque partagé
Mémoire est accessible par tous des processeurs	Les disques sont partagés par tous les processeurs
Un coût uniforme d'accès à la mémoire	Chaque processeur a sa mémoire centrale
Accès rapide aux données	Facilite la répartition de la charge de travail
Ne sont pas très différents de ceux d'un SGBD centralisé	Excellente utilisation des ressources
La complexité du réseau augmente	Bon niveau de tolérance aux pannes
XPRS, DBS3, Oracle et Informix	Oracle Parallel Server

1) Les nuplets sont envoyés dans l'ordre au différents sites, ils sont donc triés sur les sites.

2) la stratégie de round robin ne garantit pas que tous les sites ont exactement le même nombre de tuples. La stratégie est donc que chaque site fait une passe en lecture sur son fragment, pour calculer sum, max et count. Tous les sites envoie leurs résults à N_1 (une page transférée par site suffit). Enfin, N_1 fait le calcul final :

Coût en entrée/sorties : $10 * 2000/10 = 2000$ pages lues.

Coût en pages transférées : 9 pages transférées ($N_2 ... N_{10}$).

Références du chapitre

Litwin, W., Rich, T. and Schwarz, Th. Architecture for a scalable Distributed DBSs application to SQL Server 2000. 2nd Intl. Workshop on Cooperative Internet Computing (CIC 2002), August 2002, Hong Kong

Sahri, Soror. Conception et implantation d'un système de bases de données distribuée & scalable : SD-SQL Server. Diss. Paris 9, 2006.

Vingralek, R., Breitbart, Y., Weikum, G. & Snowball. Scalable Storage on Networks of Workstations with Balanced Load. Distributed and Parallel Databases 6(2) : 117-156 (1998).

Guillaume CABANAC, Claude CHRISMENT, Olivier TESTE, Michel TUFFERY, "Bases de données réparties", 10 févr. 2014

Moussa, Rim. "Systèmes de Gestion de Bases de Données Réparties & Mécanismes de Répartition avec Oracle." (2005).

Chapitre 5

Big Data

Beaucoup de projets Big Data échouent parce que leurs initiateurs voient trop grand. Il convient de rester humble, de privilégier les itérations et de redéfinir les objectifs en cours de projet.

Christophe Bourguignat.

Introduction

Ce chapitre permet l'utilisation d'une architecture Big Data pour réaliser des analyses de données. Il présente les points suivants :

— L'environnement Hadoop.
— L'architecture technique, les composants tels que les noeuds d'un cluster, les flux de données entre noeuds, etc.
— Le système de fichier distribué HDFS.
— Les principes de traitement distribué Map/Reduce.
— L'écosystème d'outils autour de Hadoop.

5.1 Définition et principe

Big Data est utilisée pour désigner aux ensembles de grosses bases de données. Il signifie big volume of data. Dans le Big Data, il y a l'idée qu'on ne gère pas de la même manière des bases de données classiques et des énormes volumes de données. A partir d'un certain seuil, la différence quantitative, volumétrique, se transforme en différence qualitative. Les process et traitements changent de nature, et les données ne peuvent plus être gérées de manière classique, dans des bases et des outils classiques.

Les dispositifs informatiques et plus largement les technologies utilisés pour gérer de gros volumes de données. Le Big Data ne renvoie pas qu'aux données en tant que telles, mais aussi aux technologies, aux stratégies, aux techniques utilisées pour gérer de gros volumes de données.

Les prérequis fondamentaux pour travailler avec des Big Data sont les mêmes que ceux nécessaires pour travailler avec des bases de données classiques. Dans les deux cas, il s'agit de gérer des données : stockage, transformations et traitements divers et variés.

Avec le Big Data, nous restons dans le monde plus large du Data Management. Néanmoins, l'échelle massive des données à traiter, la vitesse d'ingestion et de processing, les caractéristiques des données qui doivent être processées à chaque étape de traitement font émerger de nouveaux challenges technologiques. L'objectif principal du Big Data est de réussir à faire apparaître des enseignements (insights) et des connexions entre de gros volumes de données de nature hétérogène qui seraient impossible à obtenir avec les méthodes classiques d'analyse des données.

En 2001, Doug Laney, un analyste de chez Gartner, a donné une définition intéressante du Big Data. Pour expliquer ce qu'est le Big Data, il a présenté la théorie des 3 V. C'est un mode de présentation du Big Data simple et efficace. Selon Doug Laney, le Big Data peut se comprendre à partir de trois notions ayant tous la particularité de commencer par la lettre "V" (Voire Figure 5.1) :

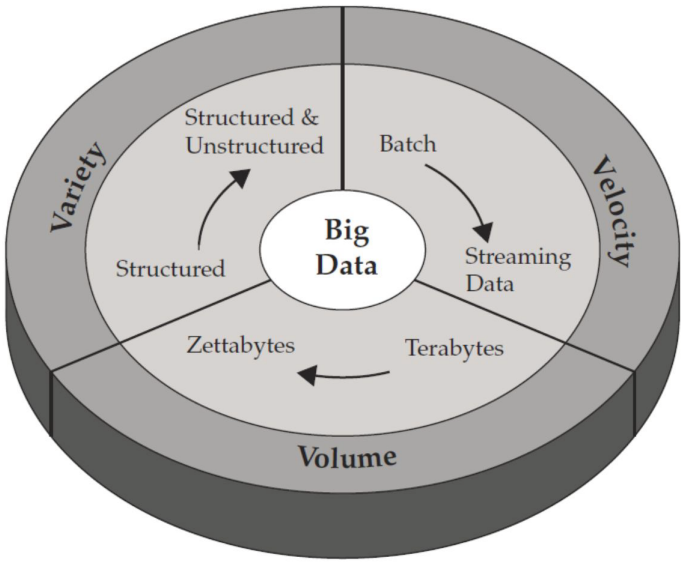


FIGURE 5.1 – Les Vs de Big Data

1. **Volume :** Un système Big Data se caractérise par le volume de données en jeu. Il traite un volume de données largement supérieur à ce que traitent les bases de données traditionnelles. Ce qui pose un défi technologique : les volumes de données en jeu excèdent les capacités de stockage d'un simple ordinateur, nécessitent des mises en réseau, l'utilisation du Cloud Computing.
2. **Vélocité :** Dans Big Data, il y a évidemment *Big.* Mais dans le Big Data, le volume n'est pas le seul sujet. Un système Big Data, c'est aussi un système dans lequel la donnée circule vite entre les outils, les bases, les applicatifs, les sources. Les données arrivent dans le système en provenance de sources multiples et sont processées souvent en temps réel pour générer des insights et mettre à jour le système. Dans le Big Data, l'approche orientée "batch" tend progressivement à céder sa place au streaming de données en temps réel ou quasi-temps réel. Dans certains cas d'usage du Big Data, le temps réel ou le quasi temps-réel sont nécessaires.
3. **Variété :** Les données sont en grand nombre et circulent vite dans le système. Mais ce n'est pas tout. Le Big Data se caractérise aussi par l'immense variété des données traitées. Les bases de données relationnelles ont affaire à des données structurées, bien définies, bien classées, bien normées. Un Data Warehouse organise de manière structurée des données structurées. Dans le Big Data, les données sont dans leur majorité non-structurées ou semi-structurées. C'est pour cette raison que Big Data rime davantage avec Data Lake qu'avec Data Warehouse.

5.2 Architecture

L'architecture de la Big data est composé de 4 grandes parties : Intégration, Data Processing et Stockage, Sécurité et Opération comme le montre le schéma si dessous (Voire figure 5.2) :

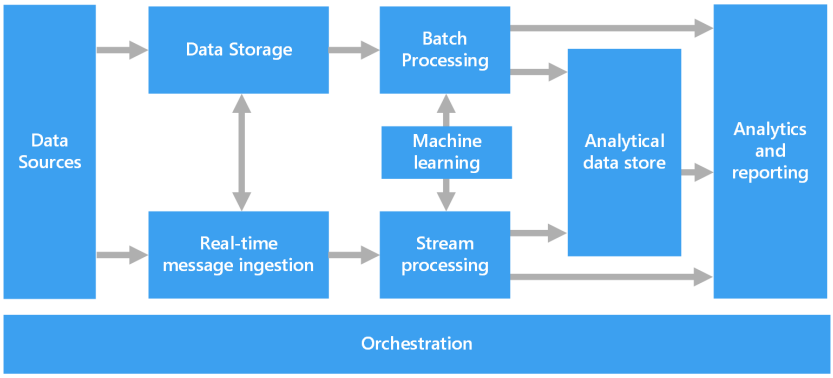


FIGURE 5.2 – Architecture de Big Data

— **Intégration :** Charger le volume de données au sein du stockage.
— **Stockage de données (Data Storage) :** Le stockage du volume de données s'appuie sur les bases de données NoSQL avec comme variété :

1. *Key Value Data Stores :* a donnée est stockée dans une clé, le data store ne connait pas le contenue de la clé mais effectue des opérations sur la valeur de la clé. Plusieurs types de Key value disponible tel que : Redis, Riak, Amazon S3 et Dynamo DB.
2. *Document Data stores :* Stockage de document pour les documents semistructuré. Les documents sont identifiés par une clé unique. Les différents types de document data stores sont les suivantes : Mongo DB, et Raven DB.
3. *Column Family Data Stores :* Stockage de données sous forme tabulaire. Exist plusieurs types comme Cassandra, et HBase.
4. *Graph data :* C'est un type de stockage de données orienté graphe sa particularité est l'exploitation des données de ce type. Chaque entité est accessible via un pointeur. Les types de graphes disponibles sont : Neo4j, OrientDB, et FlockDB.

— **Manipulation de données (Data Processing) :** Il s'agit de la manipulation et du traitement de donnés appelé Map Reduce.
— **Sécurié :** Autorisation, l'authentification et la protection des données.
— **Opérations** : Pour la gestion, le monitoring et les taches planifiés.

5.3 Domaines d'application

Une grande partie des cas d'usage du Big Data existaient déjà avant son émergence. Les nouvelles techniques permettent cependant d'aller plus vite et de traiter plus de données. La plupart des contextes d'utilisations actuelles du Big Data se résume en quelques domaines :

Marketing : Est un client pour le Big Data que ce soit pour de l'analyse prédictive ou de l'analyse de sentiment, que l'on peut définir rapidement pour l'interprétation automatisée de l'opinion exprimée d'un individu. Ce jugement peut-être caractérisé par une polarité (positive, neutre, un mélange des deux) et une intensité.

Protection de la population et prévention : En effet de nombreux moyens ont mis en œuvre par les états au nom de la défense du territoire et de la protection des citoyens contre toute menace ou attaque ; de ce fait des milliards de données non structurées sont ainsi collectées sous forme d'images, d'enregistrement audio ou vidéo. Etc. qu'il faut pouvoir stocker, trier en fonction de la pertinence et analyser afin d'en ressortir des informations critique.

Les médias numérique : Le ciblage publicitaire et l'analyse des sites web au plus détailler le numérique grçâce au Big Data et la démocratisation d'internet, il est possible de créer une campagne de publicité ciblée. En effet, sur Google et Facebook, on peut cibler une personne en particulier, et ce en fonction de ces préférences et de sa situation géographique.

La santé : Le Big Data permet à la science de réaliser des avancées importantes. En effet, dans le domaine de la santé, les technologies du Big Data permettent des évolutions impressionnantes dans l'analyse du génome humain, on passe de dix ans et de 2.3 milliards d'euros pour la réalisation du premier séquençage humain complet, à quelques jours et 760 euros. Ces avancées ont permis de mieux comprendre le développement de pathologies, d'améliorer les protocoles de soins et les mesures de prévention.

5.4 Les techniques de Big Data

Les bases de données classiques ne permettant plus de gérer de tels volumes, les grands acteurs du web (Facebook, Google, Yahoo,..) ont créé des Framework Big Data permettant de gérer et traiter des grandes quantités de données à travers. Ces données sont ensuite "splittées" ou séparées pour être traitées parallèlement afin d'alléger les processus de calcul. Puis réassemblées pour donner le résultat final. C'est cette technologie qui permet des vitesses de traitement aussi rapides sur de gros volumes de données. Nous citons quelques techniques utilisées pour manipuler le Big Data :

5.4.1 Les systèmes de fichiers distribués

Le stockage dans la base Big Data est énorme de quantité, vélocité d'agrandir la capacité de manière progressive, et variété de système de fichier qui permet de stocker n'importe quel genre de donnée. Généralement, sur du matériel et/ou OS propriétaire qui ne permet pas de lancer ses propres tâches. Donc, Système "scale-up" plutôt que "scale-out"

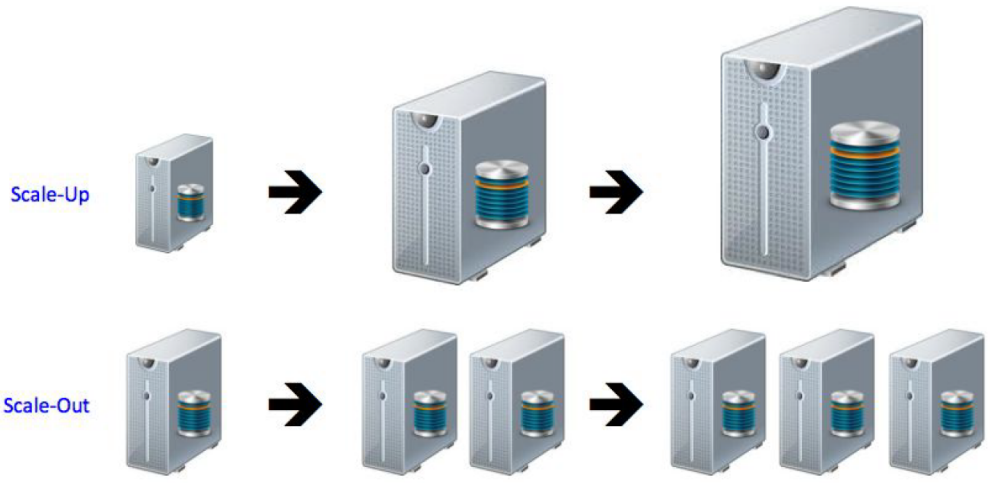


FIGURE 5.3 – Scale up vs. Scale Out

Hadoop Distributed File System (HDFS)

Est un système de fichier distribué permettant de stocker et de récupérer des fichiers en un temps record. Il s'agit de l'un descomposants basiques du framework Hadoop Apache, et plus précisément de son système de stockage.

Hadoop est un framework crée par Doug CUTTING 2009, qui permet le traitement distribué de grands ensembles de données à travers des grappes d'ordinateurs utilisant des modèles simples de programmation. Elle est libre et open source écrit en Java destiné à faciliter la création d'applications distribuées (au niveau du stockage des données et de leur traitement) et échelonnables (scalables) permettant aux applications de travailler avec des milliers de nœuds et des pétaoctets de données. Ainsi chaque nœud est constitué de machines standard regroupées en grappe. Tous les modules de Hadoop sont conçus dans l'idée fondamentale que les pannes matérielles sont fréquentes et qu'en conséquence elles doivent être gérées automatiquement par le Framework. Hadoop a été inspiré par la publication de MapReduce, GoogleFS et BigTable de Google. Le noyau d'Hadoop est constitué d'une partie de stockage :

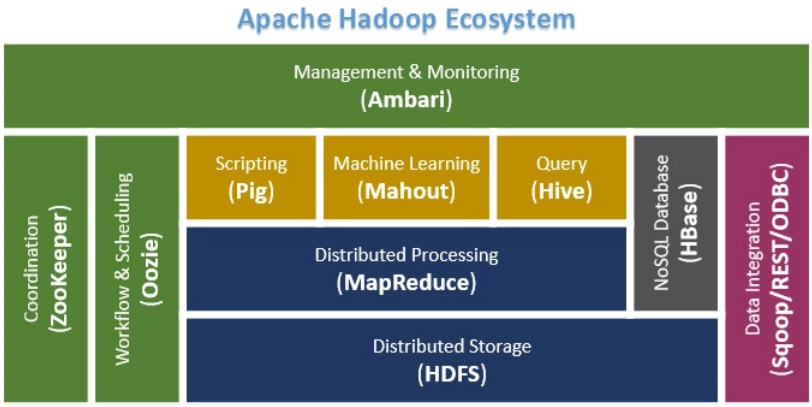


FIGURE 5.4 – Architecture de Framework Hadoop

HDFS est un système de stockage très adapté au Big Data. En combinaison avec YARN, ce système est capable de gérer des milliers de nœuds sans intervention d'un opérateur. Il existe quelques remarques importantes sur le format HDFS sont les suivantes :

— Les données ne sont pas structurées et sont stockées sous forme de paires (clé, valeur) : un morceau de données correspond à une valeur et à chaque valeur est associé une clé.

— Des surcouches existent pour traiter des données structurées : Hbase, Hive/Pig langage pour interroger une base Hadoop avec une syntaxe du type SQL.

— Les donées dans un HDFS sont enregistrées en blocs de grande taille de 64 MB (par défaut) ou plus pour des raisons de performance (réduire le temps d'accés).

— Dans un Cluster, on dispose d'un nœud maître NameNode qui gère les nœuds de données DataNodes. La haute disponibilité est assurée par la couche de données, car les blocs sont répliqués (par défaut) en 3 copies ou plus.

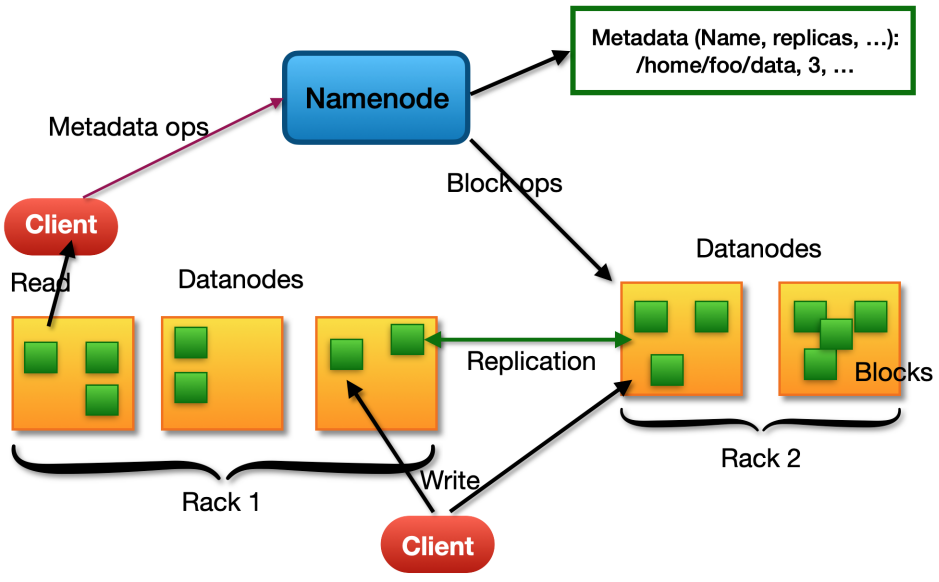


FIGURE 5.5 – Architecture HDFS

5.4.2 Les algorithmes de clustering

Le clustering est défini comme une méthode dans laquelle les données sont divisées en groupes de manière à ce que les objets de chaque groupe partagent plus de similarité entre eux avec d'autres objets dans d'autres groupes. Est une technique bien connue dans divers domaines de l'informatique, mais il est largement utilisé dans d'autres domaines d'étude tels que la bioinformatique, les réseaux, la reconnaissance des formes et donc beaucoup de travaux de recherche ont été fait dans ce domaine.

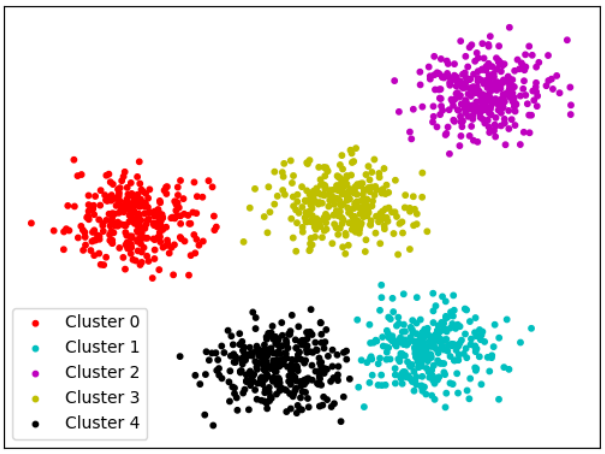


FIGURE 5.6 – Principe de clustering

Plusieurs algorithmes de clustering : Kmeans, Fuzzy K-means, PSO, ...etc.

MapReduce

MapReduce est un modèle de programmation créé par Google pour le traitement et la génération de larges ensembles de données sur des clusters d'ordinateurs dans lequel sont effectués des calculs parallèles, et souvent distribués, de données potentiellement très volumineuses, typiquement supérieures en taille à 1 téraoctet.

Il s'agit d'un composant central du Framework logiciel Apache Hadoop, qui permet le traitement résilient et distribué d'ensembles de données non structurées massifs sur des clusters d'ordinateurs, au sein desquels chaque nœud possède son propre espace de stockage. Concrètement, le framework propose deux fonctionnalités principales.

Il répartit le labeur sur les différents nœuds du cluster (map), puis les organise et réduit les résultats fournis par chaque nœud en une seule réponse cohérente à une requête. Cela est rendu possible grâce à son système de fichiers distribués HDFS.

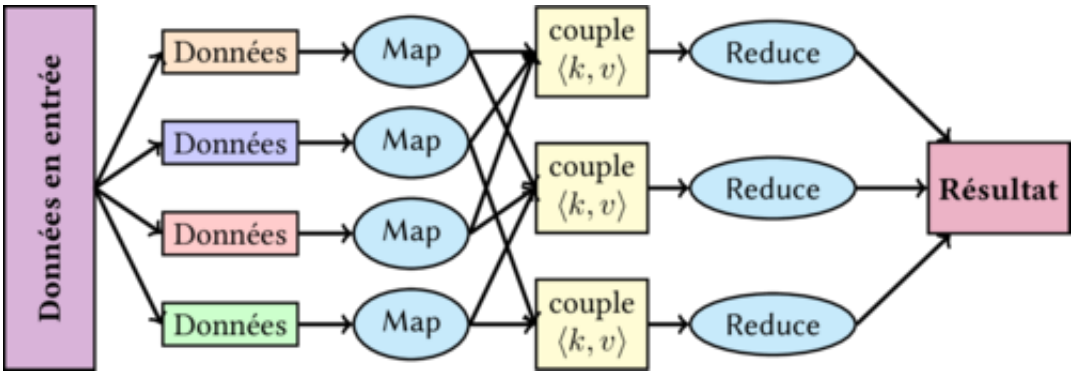


FIGURE 5.7 – Architecture de MapReduce

MapReduce est composée de deux processus essentiels sont "JobTracker" et des nœuds appelés "TaskTracker". Le JobTracker doit savoir l'état du cluster et l'ensemble des nœuds qui sont actifs et c'est lui qui recevra les différentes tâches à effectuer pour les distribuer par la suite aux autres nœuds ou TaskTracker qui vont réaliser ces tâches. Un TaskTracker est chargé d'exécuter des tâches de Map ou de Reduce. Deux fonctions de MapReduce sont :

1. **Fonction Map :** est une fonction appliquée à chaque partie des données d'entrée, pour générer une série de couples (clef ; valeur). Elle s'écrit de la manière suivante : map (clé1, valeur1) $\longrightarrow$ List (clé2, valeur2). À partir d'un couple clé/valeur, la fonction map retourne un ensemble de nouveaux couples clé/valeur. Cet ensemble peut être vide, d'une cardinalité un ou plusieurs.

2. **Fonction Reduce : (shuffle)** est une fonction qui permet le regroupement des couples (clef ; valeur) par clef distincte. Elle s'écrit de la manière suivante : reduce (clé2, List (valeur2)) $\longrightarrow$ List (valeur2). À partir des groupes de valeurs associées à une clé, la fonction reduce retourne généralement une valeur ou aucune, bien qu'il soit possible de retourner plus d'une valeur.

Le principe le plus importants de fonctionnement de MapReduce est comme suite :

Après l'accès aux données sur HDFS, le système Hadoop charge de splitter les données en blocs. Puis, sur chaque bloc est appliqué la fonction map que nous devons programmer. Le système réorganise les données d'une manière à regrouper les objets qui ont une clé unique et pour chaque type de données possédant la même clé est appelé la fonction Reduce. En sortir, on a une liste des éléments calculés.

5.5 Les technologies de Big Data

L'accroissement notable du déluge de données dans le Big Data apporte d'énormes défis sur l'acquisition de données, le stockage, la gestion et l'analyse. La gestion traditionnelle de données et les systèmes d'analyse sont basés sur le système de gestion de base de données relationnelle (SGBDR). Cependant, de tels SGBDRs appliquent uniquement aux données structurées. En outre, ces SGBDRs utilisent de plus en plus de matériels coûteux. Il est clair que les SGBDRs traditionnels ne pouvaient pas gérer l'énorme volume et l'hétérogénéité des Big data. Une plateforme de Big Data doit posséder un ensemble de caractéristiques pour répondre aux besoins et satisfaire aux exigences posées par ces nouvelles technologies, elle doit être scalable pour absorber l'accroissement continu du volume des données. Ainsi, elle doit répondre rapidement aux requêtes qui sont très complexes. Nous décrivons dans ce chapitre les plus connus :

5.5.1 Les systèmes de BD-NoSQL

Une BD NoSQL est distribuée offre un paradigme d'accès ou stockage non relationnel et une certaine forme de capacité "scale-out". Il utilise un design simple et une architecture sans point de défaillance unique. Généralement, une BD NoSQL peut soutenir un taux de requête très grand, survivre à des défaillances réseau ou de noeud, et offrir une capacité très grande de stockage.

Le NoSQL n'est pas magique, de nombreux problèmes :

— Projets immatures : nombreux bugs, perte de données en production.

— Défaillance architecturales fondamentales.

— Installation et opérations parfois très complexe.

— Très peu, voire aucun support pour les entreprises.

Selon le théorème de Brewer ou théorème CAP = un système distribué ne peut garantir en même temps les trois propriétés suivantes :

— 1. Cohérence (Consistency) : tous les noeuds du système voient la même information au même moment

— 2. Disponibilité (Availability) : toute requête reçoit une réponse.

— 3. Résistance au morcellement (Partition tolerance) : fonctionnement autonome en cas de morcellement du réseau.

Généralement, les SGBD non-relationnels privilégient la résistance au morcellement (systèmes AP/CP) :

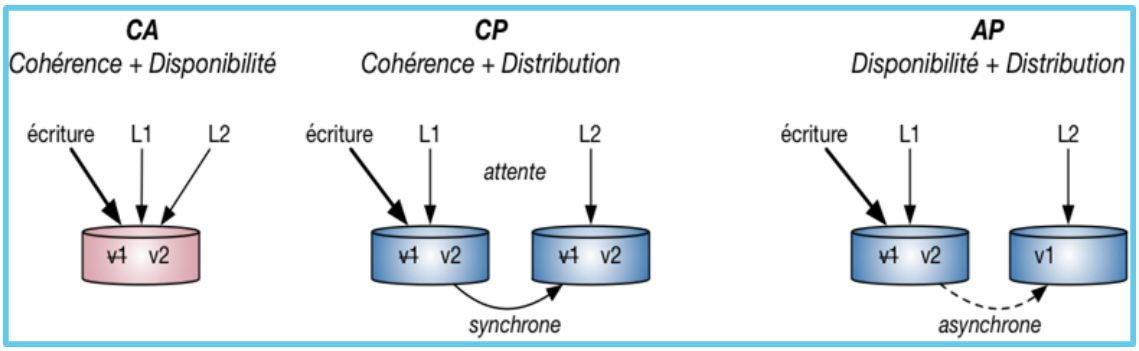


FIGURE 5.8 – Théorème CAP

5.5.2 Les systèmes de BD - Document

Le paradigme est généralement une collection de document JSON. Les documents peuvent être arbitrairement larges les documents d'une même collection ne sont pas nécessairement homogènes (schemaless). nons accédons au document sous une clé primaire. Nous interrogons par la clé primaire ou une clé secondaire (le cas échéant). Il offre généralement un langage pour chercher parmi les documents et il se combine généralement facilement avec des applications web (JSON + HTTP/REST).

5.5.3 Les systèmes de BD - HBase

Inspiré de BigTable (Google). La distribution des clés est divisée en régions ou chaque région est prise en charge par un noeud et un même noeud peut prendre en charge plusieurs régions. Si un noeud n'est pas disponible, les clés dans cette région sont également non disponibles. IL nécessite une infrastructure HDFS et utilise Zookeeper comme service de coordination. Il est implémenté en Java.

5.5.4 Les systèmes de BD - Cassandra

Créé chez Facebook, inspiré de DynamoDB (Amazon). Chaque clé est hashée et ensuite assignée à plusieurs noeuds (réplication). Pour relire la valeur, on hash la clé et on interroge un (ou plusieurs) noeud qui devrait la posséder. Cassandra utilise le timestamp dépend de la précision des horloges sur les noeuds. Il offre des familles de colonnes et un langage d'interrogation simili-SQL : le CQL. Ainssi, la capacité d'étendre un même cluster Cassandra dans plusieurs centres. L'installation et opérations très simples. Le très gros utilisateur est Netflix.

5.5.5 Les systèmes de BD - Graphe

Paradigme d'accès est celui de noeuds et liens (nodes and edges) avec les noeuds peuvent représenter des concepts hétérogènes et les liens représentent les relations entre les noeuds. Il est spécialisé différent types de réseaux (sociaux, télécommunication, etc.), transport, routes. Généralement, il implémenté sur un paradigme plus simple (clé-valeur ou autre) mais il permet des interrogations très complexes. Ce système peu de cas d'utilisation publique tel que Google, Facebook et Twitter utilisent des BD graph.

5.5.6 Les systèmes de BD - Structurées

Le paradigme tabulaire ou même relationnel offre généralement le concept de table et de colonnes typées. Certaines offrent même les garanties des BD relationnelles avec le SQL ou d'autres un langage simili-SQL.

5.6 Exercices

00 : C'est quoi le Big Data (mégadonnées) ?

01 : NOSQL vs. SGBDR

03 Choisir les bonnes réponses :

a) Hadoop a besoin de matériel spécialisé pour traiter les données

b) *Hadoop permet le traitement en temps réel des données en temps réel*

c)*Hadoop est idéal pour la charge de travail analytique, post-opérationnelle, d'entrepôt de données*

d) HDFS s'exécute sur un petit groupe de nœuds

e)...NameNode est utilisé lorsque le NameNode primaire ne fonctionne plus.

e1) Rack

e2) Data

e3) *Secondaire*

f) La machine ..... est un point d'échec unique pour un cluster HDFS

f1) DataNode

f2) *NameNode*

f3) ActionNode

g) peut-être décrit comme un modèle de programmation utilisé pour développer des applications basées sur Hadoop qui peuvent traiter des quantités massives de données.

g1) *MapReduce*

g2) Cassandra

g3) HBase

h) *MapReduce place les données et le calcul le plus proche dans le temps*

i) *La tâche Map du MapReduce est exécutée à l'aide de la fonction Mapper*

j) *Réduire la tâche dans MapReduce est effectuée en utilisant la fonction Map*

k) ....est responsable de la consolidation des résultats produits par les Map()

k1) *Reduce*

k2) Map

k3) Reducer

5.7 Solutions

00 : Le Big Data (mégadonnées) représente les collections de données caractérisées par un volume, une vélocité et une variété si grands que leur transformation en valeur utilisable requiert l'utilisation de technologies et de méthodes analytiques spécifiques.

01 : Le choix de NOSQL en opposition aux bases de données relationnelles conduits par les contraintes du marché et les besoins techniques

NOSQL	SGBDR
Consistance éventuelle	Consistance forte
Enorme quantité de données	Grandes quantités de données
Evolutivité facile	Évolutivité possible
Map-Reduce	SQL
Très haute disponibilité	Bonne disponibilité

03 : Les bonnes réponses sont : b, c, e3, f2, g1, h, i, j, k1.

Références du chapitre

Cointot, Jean-Charles, and Yves Eychenne. La révolution Big data. Dunod, 2014.

Monino, Jean-Louis, and Soraya Sedkaoui. Big Data, Open Data et valorisation des données. Vol. 4. ISTE Group, 2016.

Karoui, Myriam, Grégoire Davauchelle, and Aurélie Dudezert. "Big data. Mise en perspective et enjeux pour les entreprises." Ingénierie des Systèmes d Inf. 19.3 (2014) : 73-92.

Leonelli, Sabina. La recherche scientifique à l'ère des Big Data : Cinq façons dont les Big Data nuisent à la science et comment la sauver. Editions Mimésis, 2019.