Bases de Données Distribuées

Support de cours
Destiné aux étudiants de Master 2 GL

Dr. SEBAA Abderrazak
Enseignant au Département d'Informatique
Faculté des Sciences Exactes
Université Abderrahmane Mira de Bejaïa

Année scolaire : 2018/2019

Avant propos

Ce document est un support de cours destiné aux étudiants de la deuxième année Master de la spécialité Génie Logiciel. Les étudiants de cette formation ayant déjà acquis les connaissances de base sur les concepts de base sur les bases de données au cours de leur formation à travers le module de bases de données en deuxième année Licence et le module de bases de données avancées en première année Master, seront confrontés à de nouveaux concepts du domaine.

Objectifs du cours :

La gestion des données complexes et distribuées est devenue l'un des plus importants champs d'étude et de recherche. Boosté par l''avènement de l'Internet et du Web, ce domaine fait émerger quotidiennement de nouvelles solutions technologiques. Avec l'émergence, au cours des dernières années, de différentes formes d'informatique distribuées illustrées par les flux de données, le Cloud, et les réseaux sociaux, un regain d'intérêt est alors suscité pour la gestion des données complexes et distribuées. Le premier objectif de ce cours est de familiariser les étudiants avec les concepts fondamentaux liés aux aspects complexes de données, notamment aux aspects structurels, temporels et spatiaux. Il vise également à appréhender les notions, les techniques, et les technologies de base de données distribuées. Ceci à travers les problèmes liés à l'aspect de distribution particulièrement la conception, l'optimisation, les transactions, et la sécurité des bases de données distribuées. L'autre objectif est d'initier les étudiants aux principales questions de recherche liées à ce domaine.

Structure et contenu du cours:

La structure et le contenu des chapitres de ce document sont synchronisés avec le nouveau contenu du programme établi dans l'arrêté n°1346 de l'offre de formation validée le 09/08/2016.

Le présent cours est subdivisé en deux parties et au total de neuf chapitres, tel décrit ci-après :

Partie 1 : Types complexes de données

Chapitre 1 : Les données semi-structurées.

Chapitre 2 : Les données spatiales.

Chapitre 3 : Les données temporelles.

Chapitre 4 : Les données multimédias.

Partie 2 : La distribution de données

Chapitre 5 : Concepts et architectures des bases de données réparties.

Chapitre 6 : Conception des bases de données réparties.

Chapitre 7 : Traitement et optimisation de requêtes centralisées et réparties.

Chapitre 8 : Gestion des transactions centralisées et réparties.

Chapitre 9 : La sécurité des bases de données.

Connaissances préalables recommandées :

Des connaissances en base de données, particulièrement des connaissances sur le modèle relationnel, les dépendances fonctionnelles, les formes normales, le langage de requêtes SQL, et des notions sur le modèle objet sont nécessaires pour une compréhension adéquate de ce support de cours. Ces connaissances sont logiquement déjà acquises pour les étudiants de cette spécialité au cours de leur formation à travers le module de base de données durant la deuxième année Licence et du module de base de données avancé' en première année Master.

Table de Matière

Table de figures

Partie 1 - Types données complexes

Les modèles de données traditionnels notamment le modèle relationnel et objet souffrent de plusieurs insuffisances. En effet, le modèle relationnel n'est pas adéquat aux objets complexes tels que les documents structurés et ne supporte pas les applications non standards telles que la représentation des endroits et des places, la représentation du temps, les fichiers multimédias, la gestion des statistiques et des versions des valeurs de chaque attribut. En plus de ces insuffisances, les langages qui accompagnent ce modèle souffrent aussi de plusieurs problèmes tels que les limitations du langage SQL1 (mauvais support de l'imbrication) et du langage SQL3 (complexité de requêtes). De son côté, le modèle objet constitué des concepts d'objet, de classe et d'héritage, permet de modéliser des objets complexes à travers l'héritage, l'encapsulation et l'extensibilité, mais ne couvre pas toutes les applications non standards et il y a très peu de SGBD objet sur le marché. Afin de pallier toutes ces insuffisances, les chercheurs ont tenté alors de trouver des solutions alternatives pour représenter d'autres types de données plus complexes. L'objectif de cette première partie de ce support de cours est de décrire quelques nouveaux modèles.

Chapitre 1: Les données semi-structurées

Objectif du chapitre

Pour remédier aux insuffisances des modèles de données traditionnels tels que le modèle relationnel, la communauté scientifique a proposé plusieurs alternatives. L'une de ces alternatives est le modèle semi-structuré. Ce dernier n'a pas de structure contraignante par un schéma spécifique, mais plutôt une structure indicative. L'objectif de ce chapitre est de définir les concepts de base et le principe du modèle semi-structuré. Nous allons nous intéresser particulièrement au langage XML, son objectif et sa modélisation à travers plusieurs exemples.

1. Concepts du modèle semi-structuré

1.1 Définition du modèle semi-structuré

Le modèle semi-structuré est un modèle dans lequel les données sont principalement auto-descriptives et non conformes à un schéma particulier et fixe. En effet, les données semi-structurées n'ont pas de structure contraignante, mais une structure indicative. On dit alors que sa structure est implicite. Elles sont généralement formalisées sous forme de graphiques étiquetés. Des exemples de données semi-structurées sont les lettres, les documents, les systèmes d'information Web, et les bibliothèques numériques.

1.2 Objectifs du modèle semi-structuré

L'objectif initial du modèle semi-structuré est de prendre en compte les données du Web qui sont rangées généralement dans des fichiers de format variés (HTML, SGML, latex, …) et caractérisées par l'évolution fréquente et par l'absence de structure bien définie et surtout indexées par de nombreux moteurs de recherche. Le modèle semi-structuré doit alors faciliter l'interrogation et la recomposition dynamique de partie de documents. Il doit autoriser aussi la navigation intelligente dans une base de documents. Notant aussi que ce modèle vient pour pallier à la rigidité des modèles de données relationnel et objet.

1.3 Le principe de fonctionnement du modèle semi-structuré

Le principe de base du modèle semi-structuré est de partir des documents existants de trouver une structure commune, suffisamment souple pour prendre en compte les irrégularités, les valeurs manquantes et les évolutions de ces données. En effet, les modèles semi-structurés utilisent des graphes annotés pour représenter les données. Plusieurs variantes ont été utilisées et la différence entre ces variantes réside dans :

– L'endroit où sont situées les annotations (arêtes et/ou nœuds)

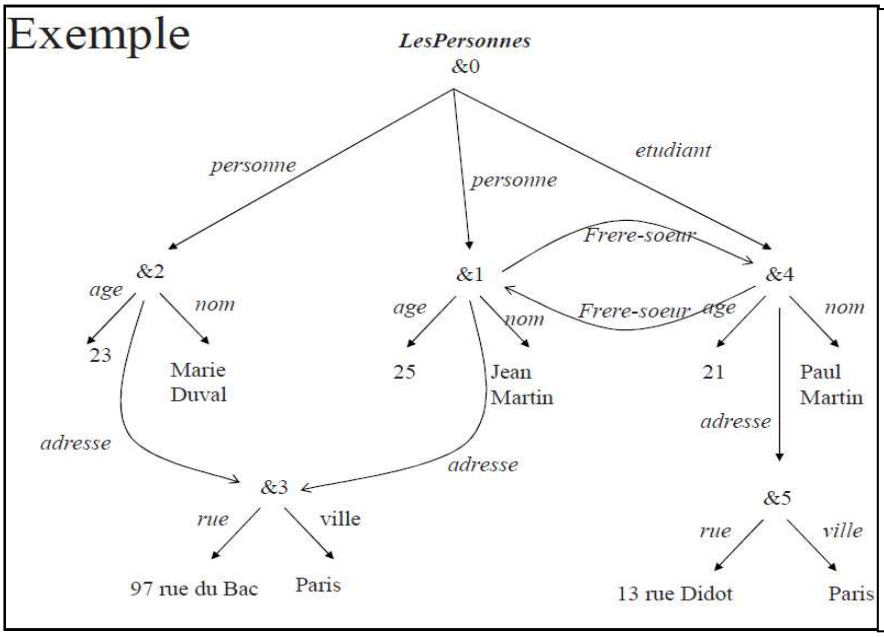
– L'existence ou non d'un ordre sur les fils d'un nœud.

– La façon de représenter le partage d'information.

Exemple

- OEM utilise des annotations sur arcs et feuilles, et ne support pas d'ordre
- UnQL utilise des annotations sur les arcs, et ne support pas d'ordre
- XML utilise des annotations sur les nœuds et feuilles, et support l'ordre.

Dans l'exemple ci-après, nous utilisons le modèle XML.




```
<LesPersonnes id=&0
<Personne id=&1
nom="Jean Martin"
age="25" frere-soeur=&4
adresse=&3 />
<Personne id=&2
nom="Marie Duval"
age="23" adresse=&3 />
<Etudiant id=&4
nom="Paul Martin"
age="21" frere-soeur=&1
adresse=&5 />
<Adresse id=&3
rue="97 rue du Bac"
ville="Paris" />
<Adresse id=&5
rue="13 rue Dutot"
ville="Paris" />
/>
```

Figure 1 Exemple de données semi-structurées

2. Le modèle XML

2.1 Définition d'XML

XML (Extensible Markup Language), est un ensemble de règles **(**format ou langage standard) pour encoder des documents sous une forme lisible par machine**.** Il est défini dans la spécification XML de sa première version (1.0) produite par le W3C en

1998, et d'autres spécifications connexes, toutes les normes ouvertes gratuites. Il est essentiellement destiné pour la publication et surtout l'échange de documents sur le Web. Son origine vient de HTML, héritier de SGML (HTML est une grammaire particulière de SGML). XML est aussi général que SGML, mais simplifié.

2.2 Objectif d'XML

L'objectif de l'utilisation de XML est de mettre en pratique la simplicité, la généralité et la convivialité des données Web. Ceci à travers un format de données textuel avec un support puissant. Bien qu'il se concentre sur les documents, il est largement utilisé pour la représentation, l'échange, le partage, la diffusion et publication de documents. Notant aussi que XML est exploité dans des applications de commerce électronique et pour la recherche d'information à travers les moteurs de recherche généralisés.

2.3 Caractéristiques

Le modèle XML est caractérisé par :

• *Structure irrégulière* : une modélisation est nécessaire pour la structure.

• *Structure implicite* : un document électronique comporte un texte et une grammaire. On doit donc parser pour détecter des informations et des relations entre ces informations, mais cela nécessite des outils spéciaux.

• *Structure partielle* : veut dire qu'il manque des informations, certaines informations sont stockées hors de la base et ne sont pas structurées.

• *Structure indicative* : veut dire qu'il n'y a pas de typage de données ni de schéma.

• *Un schéma à posteriori et non à priori* : en semi-structuré, le schéma est souvent postérieur aux données, mais parfois, on peut suggérer ou guider pour une présentation cohérente de données (exemple de pages personnelles qui comportent les mêmes informations).

• *Un schéma très vaste* : c'est-à-dire qu'on ne possède pas toutes les informations.

• *Un schéma ignoré* : c'est-à-dire qu'on ne prend pas en considération le schéma, mais l'inconvénient est qu'on parcourt tout le document à la recherche d'une chaîne. Ainsi il faut trouver d'autres langages à cause de l'impossibilité d'utiliser le langage SQL.

• Un schéma qui évolue rapidement et qu'il faut prendre en compte dans le langage de requêtes.

• La distinction entre schéma et données est peu claire.

2.4 Représentation de données XML

Un document XML peut se représenter sous deux formes:

1– La forme sérialisée est la forme courante (contenu marqué par des balises). Elle est utilisée pour stocker un document dans un fichier et pour échanger des documents.

2– La forme arborescente met en évidence la structure du document et facilite la conception des traitements. Elle permet de spécifier des manipulations de données XML et elle peut être utilisée par certaines applications qui gèrent les documents en mémoire (éditeurs XML).

Pour mieux connaître XML, la compréhension de l'ensemble des concepts sont nécessaires. Dans la suite, nous décrivons ces concepts :

Balisage et contenu. Les caractères qui constituent un document XML sont divisés en balisage et contenu. Le balisage et le contenu peuvent être distingués par l'application des règles syntaxiques simples. Toutes les chaînes qui constituent la balise commencent par le caractère "<" et se terminent par un ">" où bien par le caractère "&" et se terminent par un ";". Les chaînes de caractères qui ne sont pas de balisage sont des contenus.

Balise. C'est une construction de balisage qui commence par "<" et se termine par ">". Il existe trois types de balises: balises de début, par exemple <section>, balises de fin, par exemple </ section> et balises d'élément vide, par exemple < saut_de_ligne />.

Élément. Un composant logique d'un document qui commence par une balise de début, et se termine par sa correspondante balise de fin, ou consiste uniquement en une balise d'élément vide. Les caractères entre les balises de début et de fin correspondent au contenu de l'élément et peuvent contenir d'autres balises, y compris d'autres éléments, appelés éléments fils.

Exemple. Soit l'élément Salut suivant : <Salut>
Bonjour le monde.
</Salut>

On aura le résultat : (bonjour le monde).

Attribut: Construction de balisage consistant en une paire (nom/valeur) existante dans une balise de début ou une balise vide.

Exemple

Dans l'exemple ci-après, l'élément image a deux attributs, src et alt:

<image src = "photo.jpg" alt = 'de karim' > une photo </image >

2.5 Structure d'un document XML

Un document XML peut être découpé en plusieurs *entités* enregistrées dans un ou plusieurs fichiers. Il est généralement composé trois parties suivantes :

A– Un *prologue* (facultatif)

B – Un arbre *d'éléments* qui décrit le contenu (obligatoire)

C – De *commentaires* et *instructions* de traitement (facultatif)

A) Prologue

Tout document XML peut commencer par un prologue qui comporte une déclaration XML, des instructions de traitements (utilisées par les moteurs et les navigateurs) et une déclaration de type de documentes, comme dans l'exemple suivant :

<*?xml version="1.0" encoding="UTF-8" standalone=''yes'' ?*>

- *xml version= ''1.0''* : décrit la version utilisée
- *encoding=' UTF-8'*: le nom du codage de caractères utilisés
- *standalone=''yes''* : existence/non de déclarations extérieures (Yes = toutes les déclarations nécessaires au document sont incluses)

Notant aussi que le prologue peut contenir la déclaration d'une DTD.

B) Arbre d'éléments contenu

Un document est formé d'une hiérarchie d'éléments montrant la structure sémantique De son contenu. L'élément racine est obligatoirement unique et contient tous les autres éléments, tout élément fils est complètement inclus dans son père. De plus, un élément peut contenir d'autres éléments, des données, des références à des entités, des sections littérales et des instructions de traitement. Il peut aussi être vide. Enfin notant que tout élément doit avoir une balise ouvrante et une balise fermante et le nom de balise doit commencer par une lettre, un _, ou : mais pas de chiffre, ni de caractères réservés tel que ?, /, %, … Il ne doit pas contenir aussi d'espace. Par exemple <nom> contenu </nom>

Remarque un élément peu contenir des sections littérales (CDATA) dont son contenu est interprétable. <![CDATA [contenu non interprété]]>

C) Les commentaires

Les commentaires en XML sont formés par une balise qui commence par !-

Par exemple <!—commentaire 1 -->

Exemple complet

Ci-après un petit document XML complet, qui utilise plusieurs concepts.

```
<?xml version="1.0" encoding='ISO-8859-1' standalone='yes'>
<personne>
        <nom>Martin</nom>
        <prenom>Jean</prenom>
        <adresse>
                <rue>rue du Bac</rue>
                <ville>Paris</ville>
        </adresse>
<!-- pas d'autre information disponible -->
</personne>
```

2.6 DTD XML (Document Type Definition)

La DTD ou (définition de type de document) est un ensemble de déclarations de balisage définissant un type de document permettant de définir précisément la structure d'un document. Il s'agit des contraintes que doit respecter un document pour être *valide*. Les DTD étaient un précurseur du schéma XML et ont une fonction similaire, bien que différentes capacités. La DTD utilise une syntaxe formelle concise qui déclare précisément quels éléments et quelles références peuvent apparaître dans le document de type particulier, ainsi que le contenu et les attributs de ces éléments. La DTD déclare également des entités pouvant être utilisées dans le document d'instance.

A) Type des DTDs. Il existe trois types de DTD

DTD interne : La spécification de la DTD interne arrive dans l'entête du même document XML, on trouve d'abord le mot-clef DOCTYPE suivi de l'élément servant de racine au document et enfin la DTD elle-même entre crochets :

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE cv [...]>
<cv> . . . </cv>
```

DTD externe : Une DTD externe est écrite dans un fichier séparé (fichier.dtd), et on se contente d'y faire référence dans l'entête du document XML. On retrouve le mot-clef DOCTYPE suivi de l'élément servant de racine, puis le mot-clef SYSTEM suivi d'une URI menant au fichier DTD. À noter également que la première ligne doit faire apparaître l'attribut standalone avec la valeur no.

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no" ?>
<!DOCTYPE cv SYSTEM "cv.dtd">
<cv> . . . </cv>
```

```
<?xml version="1.0" encoding="iso-8859-1" standalone="no" ?>
<!DOCTYPE cv SYSTEM "cv.dtd" [...] >
<cv> . . . </cv>
```

DTD mixte : il est possible de mélanger les deux notations pour avoir une partie de la DTD dans un fichier séparé et une autre partie embarquée dans le document XML :

Remarque : Un document XML est dit valide si ce document est bien formé et conforme à une définition. Cela signifie que ce document respecte toutes les règles qui lui sont imposées par la DTD.

B) Déclarations

La DTD décrit la structure d'une classe de documents via des déclarations d'élément et de liste d'attributs.

Les déclarations d'éléments nomment l'ensemble d'éléments autorisés dans le document et spécifient comment déclarer les éléments et les suites de données de caractères qui peuvent être contenus dans chaque élément.

Les déclarations de liste d'attributs nomment l'ensemble d'attributs autorisés pour chaque élément déclaré, y compris le type de chaque valeur d'attribut, s'il ne s'agit pas d'un ensemble explicite de valeur (s) valide (s).

Les déclarations de balisage DTD déclarent quels types d'élément, listes d'attributs, entités et notations sont autorisées dans la structure de la classe correspondante des documents XML

1) Déclarations de type d'élément

Une déclaration de type d'élément définit un élément et son éventuel contenu. Un document XML valide contient uniquement des éléments définis dans la DTD. Les des règles suivantes doivent être respectées dans la déclaration de type d'élément :

- La structure d'une déclaration d'élément est la suivante :

<*!ELEMENT nom_element (modele contenu)* > tel que

nom_element sera le nom donné à la balise, modèle contenu désigne les sous éléments autorisés, leur ordre, si du contenu textuel est autorisé, . . .

- Déclaration de données textuelles parsées : <*!ELEMENT element (#PCDATA)*>

Par exemple : <!ELEMENT nom (#PCDATA)>

- Déclaration d'un sous élément : <*!ELEMENT element (souselement)*>

Par exemple : <!ELEMENT fax (numero)>

- Déclaration de séquence de sous éléments (ordonnées) : <*!ELEMENT element (souselement1, souselement2, ...)*>

Par exemple : <!ELEMENT identite (prenom, nom)>

Pour spécifier le nombre d'occurrences de sous éléments, on utilise les symboles suivants :

? : L'élément (ou groupe d'éléments) est optionnel.

+ : L'élément (ou groupe d'éléments) apparaît au moins 1 fois.

* : L'élément (ou groupe d'éléments) apparaît de 0 à n fois.

Par exemple : <!ELEMENT identite (nom, nomjeunefille?, prenom+, surnom*)>

- Déclaration de choix exclusif entre plusieurs sous éléments possibles: <*!ELEMENT element (souselement1 | souselement2 | ...)*>

Par exemple : <!ELEMENT situation (celibataire | marie | divorce)>

- Déclaration de groupe (parenthèses) permet de combiner les opérateurs précédents

<*!ELEMENT cercle (centre, (rayon | diametre))*>

Par exemple :<!ELEMENT contact (nom, prenom?, adresse, (telfixe |telmobile | telbureau)*)>

- Le symbole «,» indique la concaténation.

Par exemple <!ELEMENT element (nom, prenom)> signifie que le nom doit suivre un prenom.

- Déclaration de contenu mixte c-à-d mélange de données textuelles et de sous éléments, la seule possibilité est de combiner #PCDATA, | et * :

*<!ELEMENT element (#PCDATA | souselement1 | ...)**>*

Par exemple :<!ELEMENT paragraphe (#PCDATA | citation)*>

Sachant que **#PCDATA** doit toujours être en tête

- Déclaration de l'élément vide ie un élément qui n'a pas de contenu :

<!ELEMENT element EMPTY>

- Déclaration de contenu indéterminé (que pour la mise au point de DTD)

<!ELEMENT element ANY>

2) Déclarations d'attributs

La structure qui permet la déclaration des attributs est la suivante :

<!ATTLIST element attribut$_1$ type déclaration par défaut

attribut$_2$ type déclaration par défaut

……

attribut$_n$ type déclaration par défaut >

Les types d'attributs existants sont :

- *CDATA* : n'importe quelle chaine de caractères.
- *ID* : nom XML unique dans le document (un élément ne peut avoir qu'un seul attribut de ce type).
- *IDREF* : fait référence à la valeur d'un attribut de type ID et permet de créer des relations entre des éléments. Par exemple des projets, des personnes ⇨ des personnes qui travaillent dans un ou plusieurs projets.
- *IDREFS* : série de références à des ID séparées par des blancs.
- ENTITY : contient le nom d'une entité non parsée.
- ENTITIES : série de noms d'entités non parsés, séparés par des blancs.

Déclaration par défaut :

#*IMPLIED* : l'attribut est optionnel

#*REQUIRED* : l'attribut est obligatoire

#*FIXED* : la valeur de l'attribut est fixe et non modifiable et l'attribut est présent dans l'élément même si il est omis

littéral : la valeur par défaut de l'attribut est spécifiée

Exemple soit la liste d'attributs d'une image :

```
<!ATTLIST image  src      CDATA    #REQUIRED
                          width    CDATA    #REQUIRED
                          height   CDATA    #REQUIRED
                          alt      CDATA    #IMPLIED>
```

Exemple

Ci-après un exemple de DTD XML externe très simple pour décrire le schéma d'une liste de personnes :

```
<! ELEMENT personne_list (personne) *>
<! ELEMENT personne (nom, date_naissance?, Sexe?, Numéro_sécurité_sociale?)>
< !ELEMENT Nom (#PCDATA)>
<! ELEMENT Date_naissance (#PCDATA)>
<! ELEMENT Sexe (#PCDATA)>
<! ELEMENT Numéro_sécurité_sociale (#PCDATA)>
```

3. Bases de données et XML

Deux méthodes permettant aux bases de données de gérer des documents XML :

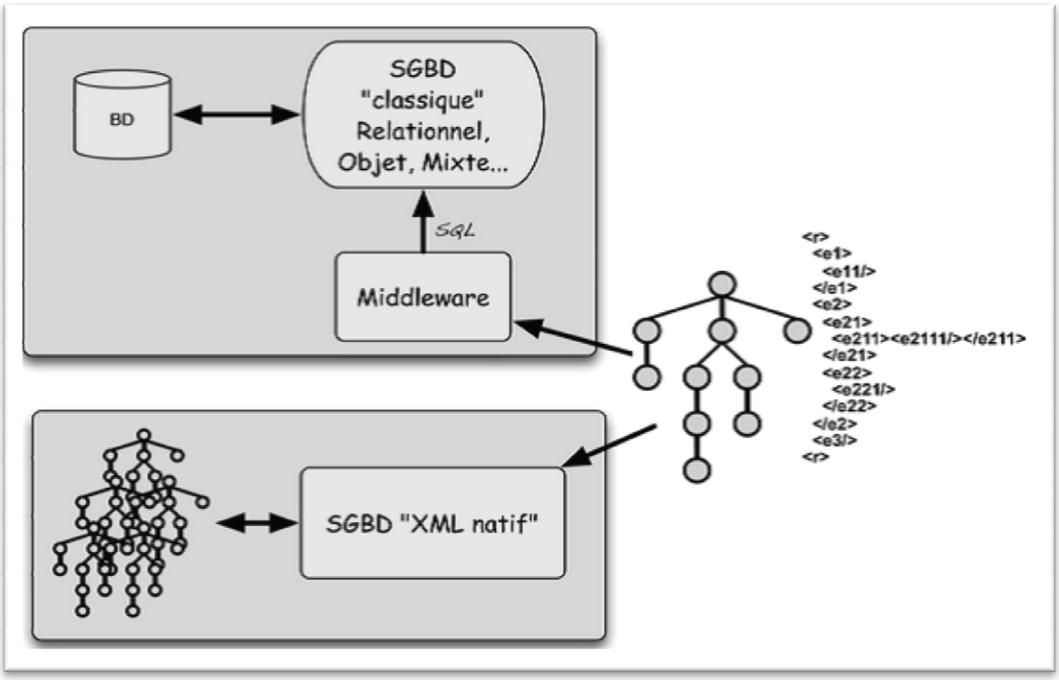


Figure 2. Les deux familles de stockage de données XML.

(1) L'extension des bases de données relationnelles "classiques" et (2) Des bases de données XML natives. Dans la première méthode, les données sont transformées et

mémorisées dans un SGBD relationnel. Dans la dixième méthode, les données sont intégrées directement en XML. Le schéma de la figure 2 illustre ces deux méthodes.

3.1 Utilisation d'une base de données relationnelles

Le problème, c'est comment mémoriser des données XML dans une base de données relationnelles ? Une solution consiste à créer une couche intermédiaire (middleware) qui jouera le rôle d''intermédiaire entre données ou requêtes XML et la base de données relationnelles. Cette couche intermédiaire appelé généralement «mapping». Le mapping complet de données XML n'est pas nécessaire, car la plupart des SGBD ont intégré des types XML, voire même des fonctions pour les manipuler (XPath ou autres). L'idée est donc de ne transformer qu'une partie des données, en laissant le reste en XML. On a ainsi un mapping partiel.

A) Mapping Relationnel/XML

Le mapping relationnel/XML se base sur deux phases : (1) la détermination des schémas XML et du schéma relationnel et (2) la mise en œuvre des transformations dans les deux sens. L'opération de détermination des schémas doit prendre en compte non seulement les schémas généraux, mais les schémas liés aux différentes requêtes de mise à jour et de recherche aussi. L'opération de transformation s'effectue en deux phases : une série de transformations du modèle XML (en utilisant XSLT, ou XQuery) puis une transformation entre XML et le modèle relationnel (Andreas & Lars, 2000).

B) Le passage du Relationnel vers le XML

Pour passer du modèle relationnel pour construire le modèle XML (sous forme de DTD). La méthodologie de construction du schéma XML est la suivante :

- ✓ Pour chaque table **"ti"** est généré un élément XML **"ei"** ;
- ✓ Pour chaque attribut de la table, créer un attribut ("CDATA", en le mettant "#IMPLIED" s'il peut être null et "#REQUIRED" sinon) ou un élément #PCDATA (avec "?" s'il peut être null) inclus uniquement dans l'élément associé à la table.
- ✓ créer un élément "k_ei ID #REQUIRED", obtenu en appliquant une fonction (bijective et compatible avec les noms XML) sur les attributs clés de la table.

- ✓ Pour chaque clé étrangère (clé d'une autre table "tj"), procéder de la même manière en créant "fk_ej IDREF #REQUIRED" ;
- ✓ Créer un élément "liste_ei" défini par "ei*" ;
- ✓ Créer l'élément racine contenant chacun des éléments "liste_ei".

C) Le passage du modèle XML vers le modèle relationnel

Le passage du modèle XML au modèle relationnel est un peu plus complexe à mettre en œuvre du fait de l'approche très souple (n'est pas rigide) d'XML. Plusieurs approches sont possibles. Dans tous les cas il faut tout d'abord de s'assurer de ne pas perdre d'informations et de faire attention aux modifications directes de la base de données. En effet, toute modification directe de la base risque de produire un résultat "ne respectant plus le schéma XML".

Un document XML est par nature une structure hiérarchique (un arbre), aussi les méthodes de construction du modèle relationnel cherchent à mémoriser cette structure. Cette structuration permet de mettre en place des liens souvent représentés par des clés (étrangères) dans le modèle relationnel. Il est donc souvent nécessaire, lors du passage XML vers le Relationnel de créer des clés. Cependant, il ne faut pas non plus perdre de vue le fait que l'arbre XML est ordonné, c'est-à-dire que les fils d'un élément ont un ordre qui peut avoir son importance et soit porteur d'informations. Cette dernière remarque est à prendre en compte, car le modèle relationnel ne possède, lui, pas de notion d'ordre (Andreas & Lars, 2000).

Pour représenter un document XML, deux tables sont nécessaires :

(1) La première table pour stocker la structure arborescente avec le numéro du nœud, sa balise, son type, son père et le numéro d'ordre parmi ses frères.

(2) La deuxième table pour stocker les valeurs des feuilles.

3.2 Bases de données XML native

L'utilisation de bases de données relationnelles a pour l'avantage d'utilisation d'SGBD communs et des performances connues. Cependant, le passage de l'XML vers le relationnel nécessite un coûteux important. De plus, durant la mise en place du mapping, il y a possibilité de perte d'informations, nécessitant de nombreux contrôles par le middleware. Pour remédier à ça, les SGBD XML natifs (ou SGBD-

XML) sont apparus comme des SGBD traitant directement des données XML sans faire une transformation de modèle (Andreas & Lars, 2000). Dans ce type de système, des "forêts" d'arbres XML sont mémorisés. Les SGBD-XML permettent d'optimiser l'accès aux informations, en particulier par la mise en place, de manière automatique ou manuelle plusieurs stratégies d'indexations des chemins, de sous-arbres, de nœuds, etc. Comme dans les SGBD-R, les SGBD-XML possèdent un langage de recherche et un langage de modification nommé XQuery. Ce langage est assez utilisé dans les bases de données natives et il utilise XPath pour décrire les localisations des modifications. Les opérations autorisées sont des opérations : (1) d'insertion : "insert-before" et "insert-after", (2) d'ajout : "append" et (3) de mise à jour : "update", "rename" et "remove" (Andreas & Lars, 2000).

3.3 Langages & XML

XML s'accompagne de standards spécifiques permettant la manipulation des informations qu'il véhicule : XPath, XQuery, SiXDML ou XUpdate. Aucun de ces langages ne couvre pas l'ensemble des fonctionnalités du SQL. Ainsi, on peut les répartir, selon la classification langage de manipulation (DML), langage de définition (DDL) et langage de contrôle (DCL).

	DML *(manipulation de données)*	***DDL*** *(définition de données)*	***DCL*** *(administration de données)*
XPath	interrogation	non	non
XQuery	interrogation	non	non
SiXDML	oui	oui	non
XUpdate	suppression, mise à jour	non	non

répartition des différents langages de requêtes selon la classification DML, DDL, DCL

Chapitre 2: Les données spatiales

Objectif du chapitre

Les modèles de données spatiales jouent un rôle de plus en plus important dans les applications métiers telles que la planification du développement urbain, le contrôle des transports et de la circulation, la prévention des incendies et des inondations, etc. À travers ce chapitre, nous allons donner et expliquer la problématique des BD spatiales, puis présenter les différentes modélisations possibles portant sur les bases de données spatiales. Ce chapitre couvre aussi les concepts et outils de base des bases de données spatiales.

1. Définition d'une base de données spatiales

Une base de données spatiales est une base de données optimisée pour stocker et interroger des objets spatiaux (ensemble organisé d'objets géographiques). Chaque objet -appelé aussi composante spatiale- est un couple formé d'une description qualitative ou quantitative et d'une localisation spatiale. Les données sont organisées souvent en couches thématiques ou les objets géographiques appartenant à un même thème forment une seule couche thématique.

Exemple de thèmes: hydrologie, rue et bâtiment.

Notant que la plupart des bases de données permettent une représentation de géométrie simple telle que des points, lignes et polygones. Certaines prennent en charge des structures plus complexes telles que les objets 3D.

2. Objectifs d'une base de données spatiales

Les objectifs de base de données spatiales permettent de :

- Modéliser les objets géographiques (une ville, un fleuve, une route…) par une description (attributs), une composante spatiale: sa géométrie (forme) et sa topologie (localisation par rapport à d'autres objets).
- Stocker et de manipuler les objets spatiaux comme tout autre objet de base de données
- Bénéficier de la cohérence du modèle de données et de l'organisation en thèmes.
- Centralisation des données spatiales.

- Interpolation possible des données spatiales avec les données attributaires avec les requêtes SQL.
- Offrir de nombreuses fonctions spatiales.
- Étendre la représentation logique aux données géométriques
- Intégrer au langage de requêtes les nouvelles fonctions assurant les opérations applicables aux objets géométriques
- Représentation physique efficace des objets spatiaux.
- Elaboration d'algorithmes d'indexation plus efficace puisque les arbres B+ ne sont pas appropriés aux données spatiales, ainsi que de nouveaux algorithmes de jointure.

3. Type de données spatiales

Pour modéliser la composante spatiale, on utilise des types de base et des opérations spatiales :

3.1 Types de base

Les types de bases de données spatiales sont utilisés pour représenter des entités géographiques complexes et pour permettre l'accéder à des propriétés de l'entité géographique telles que les contours et la dimension. Ces entités doivent avoir la capacité d'héritage de la structure et du comportement ainsi que la hiérarchie de type.

Ils sont :

- **Point**: pour représenter par exemple une maison, ou un monument.
- **Ligne**: pour représenter par exemple une route ou un réseau routier
- **Polygone**: pour représenter par exemple une commune ou une zone.

3.2 Les Opérations et fonctions spatiales

Les types de base sont manipulés à travers des opérations et fonctions spatiales, qui sont de deux types :

A) Opérations unaires. Les principales opérations unaires sont :

- Existence d'une propriété spatiale (résultat booléen)
- Calcul d'une longueur, d'une surface (résultat scalaire)

• Transformation spatiale : changement d'échelle, extraction d'objets,… (Résultat spatial)

B) Opérations binaires. Les principales opérations binaires sont :

• Prédicats topologiques, métriques (résultat booléen)

• Calcul de distance (résultat scalaire)

• Opérations ensemblistes (résultat spatial)

C) Fonctions spatiales

Parmi les fonctions les plus utiles, on peut trouver celles-ci, classées en trois groupes :

• Fonctions de récupération : ce sont des fonctions qui permettent de récupérer les propriétés et les mesures d'une géométrie.

• Fonctions de comparaison : ce sont des fonctions qui permettent de comparer deux géométries en respectant leurs relations spatiales.

• Fonctions de construction: ce sont des fonctions qui permettent de construire de nouvelles géométries à partir d'autres.

3.3 Métadonnées géographiques

C'est l'ensemble des méta-informations sur les données géographiques telles que l'échelle, l'emprise, le référentiel géographique (système de projection), qualité (incertitude de localisation et des attributs) et la datation.

4. Utilisation du modèle relationnel pour décrire les données spatiales

4.1 Modélisation

Il est possible d'utiliser le modèle relationnel pour décrire les données spéciales, c'est-à-dire en utilisant des tables relationnelles pour stocker un objet géographique sous forme d'un n-uplet. Les représentations spatiales sont composées ainsi en plusieurs relations. Pour l'interrogation des données spatiales, on utilise les requêtes traditionnelles. L'avantage de cette méthode est de pouvoir utiliser les outils existants tels que les SGBD relationnels et le langage SQL. Cependant, ses inconvénients sont nombreux tels que les mauvaises performances, l'interrogation très lourde (beaucoup de jointures), pas de calculs géométriques (tests d'adjacence, extractions de fenêtres, etc.), l'interface peu conviviale (manipulation de tables de points) et enfin la difficulté à décrire les nouveaux types.

4.2 Comparaison entre une BD relationnel et BD spatiale

	BD RELATIONNEL	BD SPATIAL
Données	Entier, Réel, Texte, ...	Plus complexes: Point, Ligne, Région…
Prédicats et calculs	Tests : =, >,... Calculs : +, /,... et fonctions simples	Prédicats et calculs géom. et topologiques: Tests : intersecte, adjacent à,... Fonctions géom. : intersection, surface..
Manipulation	Opérateurs de l'algèbre : Sélection, Projection, Jointure... Agrégats : Count, Sum, Avg...	Manipulation par thème ou inter-thèmes Sélection et jointure sur critère spatial Agrégats : fusion d'objets adjacents
Lien s entre objets	Par clés de jointures.	Liens spatiaux (souvent) implicites.
Méthodes d'accès	Index B-tree, hachage	Index R-tree, quad-tree, etc

4.3 Avantages et inconvénients du modèle relationnel pour décrire les données spatiales

L'avantage d'utiliser le modèle relationnel -pour décrire les données spatiales- est de pouvoir utiliser des solutions existantes telles que le moteur SGBD et le langage SQL. Cependant, cette méthode présente toutefois plusieurs inconvénients, notamment, les mauvaises performances, les difficultés à décrire les nouveaux types, interrogation très lourde à cause du nombre important de jointures, n'offre pas de calculs géométriques tels que tests d'adjacence, extractions de fenêtres et la non-convivialité de son interface.

Exemple: Soit les tables : *Pays, Frontière, Contour, et Point* suivants :

Pays (nom, capitale, population, id-frontiere)

Frontière (id-frontiere, id-contour)

Contour (id-contour, ordre-point, id-point)

Point (id-point, x, y)

Une requête : *Quels sont les contours de la France ?*

SELECT F.id-contour, x, y *FROM* Pays P, Frontiere F, Contour C, Point PT

WHERE Nom='France' *AND* P.id-frontiere = F.id-frontiere *AND* F.id-contour = C.id-contour

AND C.id-point = PT.id-point *ORDER BY* F.id-contour, ordre-point.

5. Modèles spatiaux non relationnels

L'objectif d'un nouveau modèle de données spéciales est d'étendre la représentation logique des données aux données géométriques pour des objectifs de simplicité, représentation proche du monde réel, l'indépendance aux données, l'intégration au

langage de requêtes des nouvelles fonctions assurant les opérations applicables aux objets géométriques, une représentation physique plus efficace, une indexation plus efficace et de nouveaux algorithmes de jointure plus souple. D'un point de vue théorique, les modèles spatiaux sont classés en deux grandes familles :

- **Les modèles spaghetti** qui offrent uniquement des primitives géométriques, telles que les points ou les lignes pour la représentation des objets géographiques.
- **Les modèles topologiques** qui conservent les relations topologiques entre les objets en plus de primitives géométriques de représentation de la forme.

6. Modèle Spaghetti

Le modèle spaghetti propose uniquement des primitives géométriques à travers les poly-lignes et des libellés. Il est caractérisé par la simplicité**,** l'homogénéité et la non-restriction de la représentation, mais aussi la redondance de données (les frontières adjacentes sont représentées deux fois). Ce modèle offre des avantages comme le faible coût de digitalisation et la non-nécessité à une opération de transformation pour le stockage des données puisqu'elles sont issues de la saisie. En contrepartie, ce modèle est très peu structuré. Il ne conserve aucune relation entre les objets, rendant la recherche selon des critères spatiaux difficiles et n'offre aucun contrôle au niveau des données. Il laisse donc la place à plusieurs incohérences, tant au niveau de la forme des objets que des relations entre les objets.

Nous détaillons deux modèles Spaghetti qui sont le modèle anarchique et le modèle polygonal unifié.

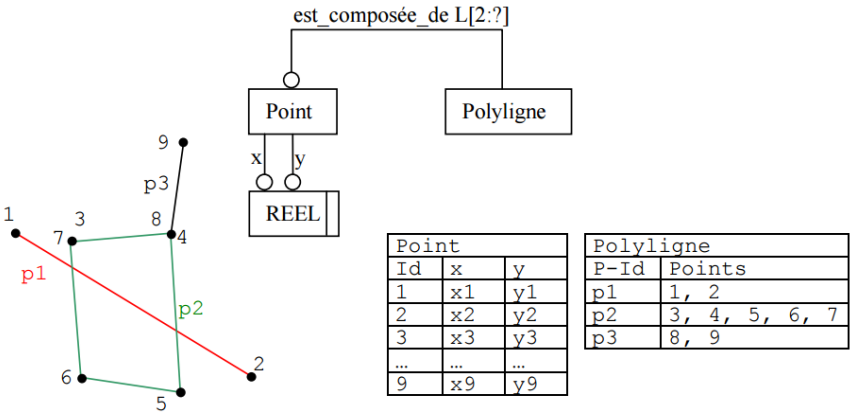
6.1 Modèle Spaghetti anarchique

Ce modèle ne gère que les points et les poly-lignes (c-à-d des lignes composées d'une liste de points). Dans ce modèle, il n'existe aucun lien entre les différentes poly-lignes, elles sont uniquement positionnées sur le plan. La notion de polygone est également absente. Les objets géographiques de type surfacique sont donc décrits uniquement par des éléments linéaires qui composent leurs contours. Et les requêtes faisant intervenir une notion de surface (par exemple, quels sont les bâtiments dans une certaine parcelle) ne sont pas adaptées à ce type de données. Ce modèle est

totalement anarchique puisqu'il ne tient même pas compte des points qui sont susceptibles de servir plusieurs fois dans des poly-lignes différentes.

Exemple

Dans l'exemple suivant, les points 4, 8 et 3, 7 sont dupliqués.



Point		
Id	x	y
1	x1	y1
2	x2	y2
3	x3	y3
…	…	…
9	x9	y9

Polyligne	
P-Id	Points
p1	1, 2
p2	3, 4, 5, 6, 7
p3	8, 9

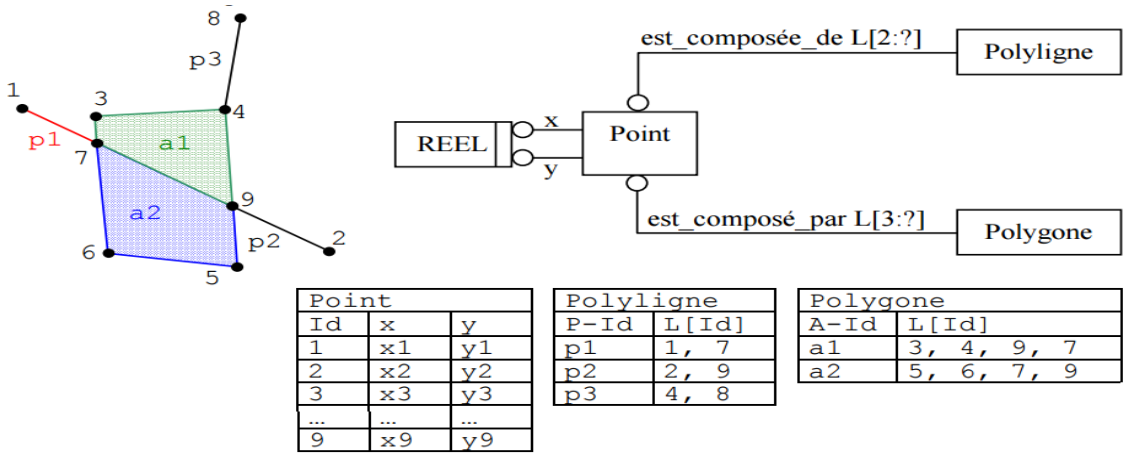
Figure 3. Exemple de modèle spaghetti anarchique

6.2 Modèle Spaghetti polygonal unifié

Dans ce modèle, un objet est représenté par un ensemble de points, mais les relations topologiques entre les objets sont calculées à la demande. Le modèle Spaghetti polygonal unifié s'appuie sur une couche géométrique unique partagée entre tous les objets. Deux polygones -utilisant un même point dans la définition de leur contour- font référence au même objet point de la base. Cette caractéristique permet d'éviter les duplications de points et de poly-lignes. Ce modèle, nettement plus cohérent que le modèle anarchique, ceci par la création de points à chaque intersection de poly-lignes et permet beaucoup plus de calculs géométriques et de raisonnements spatiaux, mais n'est pas encore très bien adapté aux requêtes faisant intervenir des notions topologiques comme adjacence, en raison de la nécessité de parcourir la totalité des listes d'objets.

Exemple

Dans l'exemple précédent, on remarque que le point 7 appartient à la poly-ligne p1 ainsi qu'aux polygones a1 et a2 définis par leurs contours linéaires. Ce qui sera évité dans ce modèle polygonal unifié.



Point		
Id	x	y
1	x1	y1
2	x2	y2
3	x3	y3
…	…	…
9	x9	y9

Polyligne	
P-Id	L[Id]
p1	1, 7
p2	2, 9
p3	4, 8

Polygone	
A-Id	L[Id]
a1	3, 4, 9, 7
a2	5, 6, 7, 9

Figure 4. Modèle spaghetti polygonal unifié (basé sur les points)

7. Modèles topologiques

Contrairement au modèle de données spaghetti, le modèle topologique est caractérisé par l'inclusion d'informations topologiques dans le jeu de données, c'est-à-dire, qu'il ne se contente pas de stocker les éléments qui constituent les entités, mais il stocke également les relations topologiques entre ces entités. La topologie est un ensemble de règles qui modélisent les relations entre les points, les lignes et les polygones voisins et détermine la manière dont ils partagent la géométrie.

Nous considérons par exemple deux polygones adjacents. Dans le modèle de spaghetti, la limite commune des deux polygones voisins est définie comme deux lignes séparées et identiques. L'inclusion de la topologie dans le modèle de données permet à une seule ligne de représenter cette limite partagée avec une référence explicite indiquant quel côté de la ligne appartient à quel polygone. Le modèle topologique considère une classe d'entités non plus comme un ensemble d'entités complètement autonomes, mais plutôt comme un graphe (c-à-d un ensemble d'arcs connectés). Notant enfin, qu'au sein d'une classe d'entité, les entités ne peuvent pas se chevaucher**.**

Trois concepts topologiques sont la base du modèle de données topologiques et qui sont :

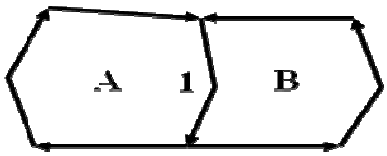
(1) Un arc est constitué d'un ou deux sommets (ou nœuds) et des points annexes. *Un arc orienté* a un sommet initial et un sommet final.

(2) Un polygone est délimité par des arcs.

(3) Zone (Arc – Polygones) : un arc étant orienté, il a un polygone gauche et un polygone droit.

Exemples :

1) Dans l'exemple ci-après, l'arc 1 a pour polygone gauche le polygone B et pour polygone droit le polygone A.



2) Dans cet exemple pour les arcs I et II, les trois tables ci-après sont nécessaires

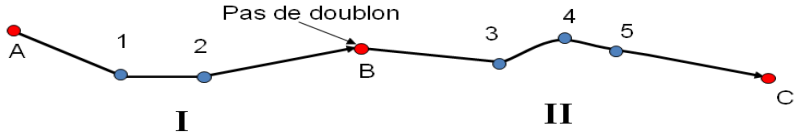


Tableau des arcs

	Topologie			
	Sommet Initial	Sommet final	Point annexe début	Point annexe fin
I	A	B	1	2
II	B	C	3	5

	XS	YS
A	XS_A	YS_A
B	XS_B	YS_B
C	XS_C	YS_C

Tableau des sommets

	X	Y
1	X_1	Y_1
2	X_2	Y_2	
3	X_3	Y_3
4	X_4	Y_4
5	X_5	Y_5

Tableau des points annexes

Dans cet exemple pour les polygones P1 et P2 et les arcs, les trois tables ci-après sont nécessaires

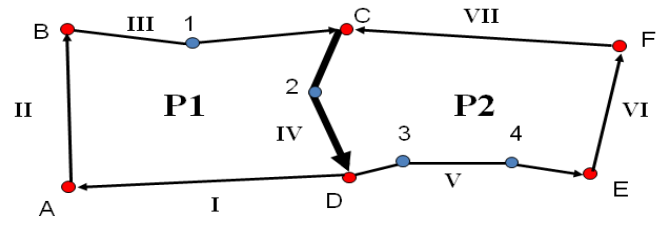


Tableau des arcs

	Topologie					
	Sommet Initial	Sommet final	Polygone gauche	Polygone droit	Point annexe début	Point annexe fin
I	D	A				
II	A	B				
III	B	C			1	1
IV	C	D	P2	P1	2	2
V	D	E			3	4
VI	E	F				
VII	F	C				

	XS	YS
A	XS_A	YS_A
B	XS_B	YS_B
C	XS_C	YS_C
D	XS_D	YS_D
E	XS_E	YS_E
F	XS_F	YS_F

Tableau des sommets

	X	Y
1	X_1	Y_1
2	X_2	Y_2
3	X_3	Y_3
4	X_4	Y_4

Tableau des points annexes

Figure 5. Exemple Arc – Polygones

8. Comparaison entre le modèle topologique et le modèle spaghetti

Modèle topologique	Modèle spaghetti
Avantages - Pas de redondance - Accélération très significative des traitements faisant appel aux relations de voisinage - Très strict (pas de trous, pas de chevauchements) - Excellent pour les traitements sur gros volumes de données.	**Avantages** - Très simple à comprendre et à manipuler
Inconvénients - Très lourd à gérer	**Inconvénients** - Redondance - Pas assez strict - Mises à jour graphiques souvent laborieuses - Très pénalisant pour les traitements sur gros volumes de données

Chapitre 3: Les données temporelles

Objectif du chapitre

De nos jours, les SGBD relationnels sont conçus pour gérer les informations du présent. Par contre, si l'utilisateur désire interroger ses bases de données en considérant le facteur temps, il est personnellement responsable de la gestion et de la maintenance des données historiques et futures étant donné que le SGBD ne lui offre aucun support. L'objectif du présent chapitre est d'introduire et d'expliquer la notion du temps dans les bases de données, sa modélisation et sa granularité. Ensuite, nous décrivons les mécanismes d'intégration du temps dans les bases de données relationnelles. Nous allons terminer ce chapitre par la description des différents types de données temporelles.

1. Notion du temps

De nos jours, les SGBD relationnels sont conçus dans le but de gérer les informations du présent (informations courantes). Si, en revanche, l'utilisateur désire interroger et analyser ses bases de données relationnelles en considérant le facteur temps, il doit lui assurer la gestion et de la maintenance des données historiques et futures. Le système de bases de données ne lui offre aucun support pour le stockage, la recherche ou l'analyse des informations comprenant des aspects temporels. Cependant, la totalité des applications ont besoin du temps ce qui explique l'intérêt de leur fournir des fonctionnalités temporelles directement dans le SGBD. Par exemple la table ci-dessous illustre l'intérêt du temps pour une base de données.

Nom	Salaire	Titre	DateNais	Start	Stop
Toto	60000	assistant	12 05 65	01 01 91	31 09 96
Toto	70000	assistant	12 05 65	01 10 96	31 09 97
Toto	70000	MCF	12 05 65	01 10 97	31 09 98
Toto	70000	Prof	12 05 65	01 10 98	01 01 00

En effet, les requêtes simples deviennent compliquées, ainsi dans l'exemple ci-dessus comment répondre aux requêtes suivantes :

1) quel est le salaire actuel de Toto ?
2) quel est l'historique du salaire de Toto ?

La requête SQL correspondante à la première est la suivante :

SELECT Salaire FROM Employés WHERE Nom = 'Toto'
and Start <= CURRENT_DATE and Stop >= CURRENT_DATE

En effet la deuxième partie de la requête devrait être implicite

Pour la deuxième requête, il faut trouver tous les intervalles de temps "fusionnables" pour lesquels le salaire de Toto est le même, (Cette requête est très compliquée et coûteuse directement en SQL)

1.1 Définition du temps

Le temps introduit une *grandeur physique unidimensionnelle,* dont les valeurs sont globalement ordonnées. Ceci permet donc de comparer deux valeurs quelconques sur l'axe du temps pour déterminer si l'une est «inférieure» (antérieure) ou «supérieure» (postérieure) à l'autre. Les données temporelles qui nous intéressent peuvent non seulement désigner un jour ou un instant précis, mais encore une durée sous la forme d'un intervalle de temps. Par exemple, l'ancienneté d'un employé s'exprime en nombre d'années. Il convient de noter qu'une donnée temporelle peut être interprétée comme un point temporel ou comme une durée selon le point de vue de l'utilisateur.

1.2 Modélisation du temps

A) Structure: La structure utilisée pour décrire le temps est une structure linéaire, alternative, ou périodique. Les limites du temps peuvent être connues ou inconnues. Dans le premier cas, la structure possède une origine (limite inf.), et peut être limité (limites sup).

B) Comportement temporel des données: En général, le comportement des entités temporelles peut être classé dans l'une des quatre catégories de base, à savoir: (1) discret, (2) continu, (3) Constant pas-à-pas et (4) basé sur la période.

- **Continu :** Un comportement continu est observé lorsqu'une valeur d'attribut est enregistrée de manière constante dans le temps, de sorte qu'elle puisse changer constamment. On trouve souvent un comportement continu dans les systèmes de surveillance enregistrant des caractéristiques continues. Par exemple, l'indicateur de vitesse d'un véhicule automobile.

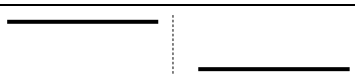
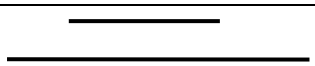
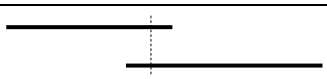
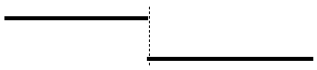
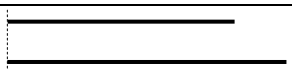
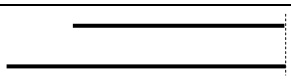
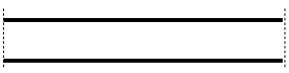
- **Discret:** les attributs de données discrètes sont enregistrés à des moments précis, mais ne sont pas définis à aucun autre moment. Les données discrètes sont associées à des événements individuels. Par exemple « un contrôle complet a eu lieu à une date donnée».
- **Constant pas-à-pas: l**es données constantes pas-à-pas sont constituées de valeurs qui changent à un moment donné, puis restent constantes jusqu'à ce qu'elles soient à nouveau modifiées. Par exemple, la mesure de la pression artérielle.
- **Périodique:** les données basées sur des périodes modélisent le comportement des événements qui se produisent sur une certaine période, mais à la fin de cette période deviennent indéfinis. Un exemple de données basées sur la période serait les enregistrements d'utilisation du médicament du patient, où un patient prend un médicament pendant une période prescrite, puis cesse de le prendre.

C) Granularité : lorsque nous considérons le temps en unités discrètes, par objectif ou par contrainte de l'application, des unités de tailles différentes peuvent être utilisées. La taille des unités de temps utilisées pour se référer à un scénario particulier s'appelle granularité des unités temporelles : les petits grains temporaux désignent des unités courtes (jours, heures, secondes, millisecondes, etc.) et de gros grains temporaux se référer à des unités de temps plus longues (mois, années, décennies, etc.). Les concepts relatifs au ''temps'' sont :

- **Chronon:** pour une situation particulière, il représente la plus petite unité indivisible de temps pouvant être enregistrée ou raisonnée.
- **Instant**: c'est un point temporel dans le monde réel.
- **Intervalle temporel : c'**est une période de temps ayant un début et une fin.
- **Élément temporel** : c'est un ensemble d'intervalles temporels. Par exemple les vacances.
- **Période de vie** : c'est une période pendant laquelle un objet existe.

1.3 Prédicats d'Allen (1983)

Afin de pouvoir manipuler et comparer les intervalles de temps, Allen (1983) a proposé un ensemble de prédicats et d'opérations sur les données temporelles. Le résultat de ces prédicats est un élément temporel.

E1 before E2 (←after) «précède»	E1 E2	
E1 during E2 (←contains) «pendant»	E1 E2	
E1 overlaps E2 (←overlapped_by) «chevauche»	E1 E2	
E1 meets E2(←met_by) «jouxte»	E1 E2	
E1 starts E2(←started_by) «commence»	E1 E2	
E1 finishes E2(←finished_by) «finit»	E1 E2	
E1 equal E2 «égale»	E1 E2	

Prenons deux événements, E1 et E2. Chaque événement possède un point de départ dans le temps et un point de fin dans le temps - chaque événement se déroule donc comme un intervalle dans le temps. Les relations possibles entre deux intervalles ou événements sont décrites dans le tableau précédent.

2. Intégration du temps dans les bases de données relationnelles

Plusieurs moyens par lesquels les données temporelles peuvent être enregistrées et interrogées à l'aide d'extensions du modèle relationnel. Nous allons détailler les approches les plus importantes.

2.1 Archivage

L'une des premières méthodes de gestion des données temporelles pour une base de données consistait à sauvegarder et archiver toutes les données stockées dans la base de données à intervalles réguliers. c-à-d que toute la base de données a été copiée avec un horodatage spécifique (hebdomadaire ou quotidien). Cependant, les informations entre les sauvegardes sont perdues et la récupération des informations obtenues est lente et lourde, puisqu'une version complète de la base de données doit être chargée.

2.2 Découpage temporel

La méthode de découpage temporel fonctionne si la base de données est stockée sous forme de tables, comme dans le modèle relationnel. En cas de modification de la base de données, au moins un attribut d'au moins un tuple -d'une table particulière-

est modifié. L'approche par découpage temporel stocke simplement la table entière avant l'événement et lui donne un horodatage. Ensuite, une copie dupliquée, mais mise à jour est créée et devient une partie de l'état de la base de données «en direct». Le découpage temporel est plus efficace et plus facile à implémenter que l'archivage, car seules les tables modifiées sont copiées avec un horodatage. Cependant, la redondance de données reste importante dans l'approche de la chronologie. Cette redondance de données résulte de la duplication d'une table entière lorsque, par exemple, une seule valeur -d'attribut d'un tuple -a été modifiée. Avec le découpage temporel, il n'est pas possible de connaître la durée de vie d'un état particulier de base de données.

2.3 Horodatage des tuples

L'horodatage des tuples signifie que chaque relation est complétée par deux attributs temporels représentant un intervalle de temps, (les deux attributs sont le temps de transaction ou le temps valide). Ainsi, la table entière n'a pas besoin d'être dupliquée, mais que les nouveaux tuples sont simplement ajoutés à la table existante lorsqu'un événement se produit. Ces nouveaux tuples sont ajoutés à la fin de la table.

3. Type de bases de données

En considérant l'aspect du temps, quatre types de bases de données peuvent être identifiés, en fonction de la capacité d'une base de données à prendre en charge le temps de validité et / ou du temps de transaction et du degré de mise à jour de la base de données.

3.1 Base de données instantanée

Une base de données d'instantanée peut prendre en charge soit le temps valide, soit le temps de transaction, mais pas les deux en même-temps. Elle forme un 'instantané' de l'état de la base de données à un moment donné. Le terme "instantané" est utilisé pour désigner l'état de calcul d'une base de données à un moment donné. Si les données stockées dans la base représentent un modèle correct du monde à ce moment actuel, l'état de la base de données représente le temps de transaction, par contre, si les données stockées dans la base sont valides ou vraies à ce moment actuel, l'état de la base de données représente une heure valide. Une base de données

instantanée est une base de données sans enregistrement des modifications ou des valeurs de données antérieures.

3.2 Base de données de restauration

Appelée aussi base de données d'annulation, elle est composée d'une séquence d'états indexés par heure de transaction, c'est-à-dire l'heure à laquelle l'information a été stockée dans la base. Une base de données de restauration préserve les états passés de la base de données, mais pas ceux du monde réel. Par conséquent, elle ne peut pas corriger les erreurs dans les informations passées, ni indiquer que certains faits seront valables pour une période future. Dans ce type de base de données, à chaque fois qu'une modification est apportée à la base de données, les états avant et après sont enregistrés avec l'horodatage de la transaction au moment où la modification a eu lieu. Ce mécanisme permet à la base de données de revenir ultérieurement à un état antérieur.

3.3 Base de données historique

Ce type de base de données ne prend en charge que le temps valide. En effet, la sémantique du temps de validité est étroitement liée à la réalité et, par conséquent, les bases de données historiques peuvent conserver l'histoire de l'information et représentent les connaissances actuelles sur le passé. Cependant, les bases de données historiques ne peuvent pas visualiser la base de données telle qu'elle était à un moment donné dans le passé.

3.4 Base de données temporelle

Ce type de base de données représente à la fois le temps de transaction et le temps valide, la base de données est donc à la fois historique et de restauration. Ainsi, les bases de données temporelles enregistrent des données relatives à la date de validité et au moment de la transaction lorsque de telles données ont été entrées dans la base de données. Cela signifie qu'à moment donné, nous pouvons restaurer notre base de données temporelle pour déterminer quelles étaient nos convictions en matière de temps valides pour un temps de transaction antérieur donné. Les bases de données temporelles permettent une mise à jour rétroactive, c'est-à-dire qu'elle entre en vigueur après l'heure à laquelle les données ont été référencées. Les bases de données

temporelles prennent également en charge la mise à jour proactive, c'est-à-dire qu'elle entre en vigueur avant l'heure à laquelle les données ont été référencées.

Dans une *base de données temporelle,* les valeurs de données, les tuples ou des tables entières sont définies *par rapport à l'axe du temps*. Une donnée temporelle peut revêtir plusieurs significations différentes pour un objet particulier dans la base. Ainsi, un *temps valide* peut désigner soit un instant précis auquel se produit un événement déterminé, soit un intervalle de temps qui définit la durée de validité des valeurs de données associées. Par exemple, l'adresse d'un employé est valable jusqu'au prochain changement de domicile. D'autre part, le *temps transactionnel* désigne le point temporel auquel un objet spécifique a été introduit, modifié ou détruit dans une base de données. C'est le SGBD qui gère les temps transactionnels au moyen d'un journal.

Pour enregistrer les dates de validité, la plupart des SGBD relationnels supportent deux types de granules de temps: le type DATE *permet de représenter une date* composée de l'année, du mois et du jour, le type TIME *définit le temps en heures, minutes et secondes*. Pour représenter une durée, aucun type particulier de donnée n'est requis, car les nombres entiers et décimaux suffisent. Le calcul avec des données temporelles s'effectue donc de manière naturelle.

Pour exprimer la *validité d'une entité*, il faut introduire généralement deux attributs. Le point temporel « valable dès » (*valid from*, en anglais) détermine l'instant à partir duquel un tuple ou une valeur de données est valide. L'attribut « valable jusqu'à » (*valid to*, en anglais) indique le point temporel où la période de validité prend fin. Des fois, au lieu de définir deux instants VALID_FROM et VALID_TO, l'attribut VALID_FROM suffit sur l'axe du temps. En effet, chaque instant VALID_TO est implicitement déterminé par le prochain instant VALID_FROM, car les intervalles de validité d'une entité ne se chevauchent pas.

Chapitre 4: Les données multimédias

Objectif du chapitre

Actuellement, les bases de données multimédias attirent l'attention de beaucoup de chercheurs et de développeurs, puisqu'elles sont convoitées par beaucoup applications modernes et permettent l'amélioration de nombreux aspects techniques, tels que de l'utilisation de la bande passante. Ce chapitre est une introduction à ce type de base de données. Il décrit tout d'abord les différents types de données multimédias. Par ailleurs, il décrit les SGBD multimédias, leurs fonctionnements, leurs caractéristiques et les outils qu'ils utilisent. Les annotations et métadonnées dans les bases de données multimédia sont évoquées dans ce chapitre.

1. Données multimédia

Les contenus multimédias peuvent être de différents types, en particulier, document texte, image, son, vidéo et peuvent aussi être combinés.

A) Document texte : Les modèles de données classiques ne permettent pas le traitement efficace de documents. En effet, il est difficile de définir avec ces modèles classiques un schéma, prenant en compte les spécificités de chaque document à cause de la structure hiérarchique, la profondeur arbitraire et la structure hypertextuelle d'un document. Par ailleurs, les problèmes de l'interrogation de ces documents sont comment parcourir, filtrer, extraire et reconstruire, transformer et les présenter. La solution pour décrire un document texte est d'utiliser SGML, XML pour la représentation et XQuery pour la manipulation.

B) Image : Une image est décrite par ses propres caractéristiques et par des relations sémantiques, fonctionnelles et spatiales avec d'autres objets. Pour modéliser les images, on utilise la matrice de pixels. Cependant, l'interrogation des images concerne sur ces attributs descriptifs (couleur, texture, forme), la similarité avec d'autres images, les propriétés spatiales et la détection du contour.

C) Sons : Le son est une donnée continue où le temps est pris en compte. Il existe deux façons principales de décrire le son.

(1) Techniques de traitement du signal c-à-d séquence de cellules codant la fréquence ou l'amplitude du signal.

(2) Métadonnées décrivant le contenu et souvent se fait manuellement.

Pour indexer le contenu du son, on extrait des informations telles que l'intensité, le volume et le rythme.

D) Vidéo: plusieurs problèmes devraient être traités afin de pouvoir stocker et manipuler les données de type vidéo. Parmi ces problèmes nous cotons l'utilisation ou non de la compression, les normes utilisées, la modélisation, l'indexation, la recherche et l'interrogation de la vidéo.

Pour modéliser la vidéo, on utilise la structuration hiérarchique. Par exemple, un film est un ensemble de scènes, une scène à son tour est un ensemble de séquences, puis plans et ce dernier est un ensemble d'images. L'annotation manuelle et semi-automatique des segments vidéo sont largement utilisées. Pour interroger les données vidéo, on utilise des langages déclaratifs tels que le SQL et l'OQL et un ensemble de techniques spécifiques pour interroger sur la nature visuelle, auditive, temporelle et spatiale de ces données. Par exemple, rechercher des images telles que le coucher de soleil dans une vidéo. Recherche de séquences (audio, vidéo), d'objets, de mouvements, de propriétés dans une vidéo.

Exemple: Comparaison par rapport au volume unitaire de stockage

	Stockage brut	**Compression sans perte**	**Compression avec perte**
Marceau Texte	125 Ko	Gzip: 42 Ko	N/A
Photo(6 Mpixels)	18 Mo	Gzip*: 12 Mo	1 Mo
Album musical	750 Mo	FLAC: 406 Mo	75 Mo
Film de 100 mn	180 Go	X	700 Mo

2. SGBD multimédia

2.1 Définition d'un SGBD multimédia

C'est un SGBD capable de créer, stocker, manipuler et interroger des objets multimédia (documents, sons, images, vidéos.) de grandes tailles, en très grande quantité (10^{12}, 10^{15}, 10^{24} octets). Il utilise un modèle et un langage spécifique aux données multimédia. Il capable aussi d'indexer ces données et garantira des

performances acceptables. Il est également capable de gérer les aspects temporels et spatiaux.

2.2 Caractéristiques

Un SGBD multimédia doit être capable d'offrir les cinq fonctionnalités suivantes:

- Gérer des types de données multimédias tels que les documents texte, les images, le son et la vidéo, en offrant la possibilité de la composition ou la décomposition des objets multimédias (une 1 piste vidéo = image + son), la possibilité de synchronisation des composants, la gestion de la dimension temps et de nouveaux opérateurs spécifiques : 'zoom', 'rotate', 'play', 'avance rapide', 'compresse', etc.
- Offrir les fonctionnalités traditionnelles des bases de données, c'est-à-dire l'existence d'un langage d'interrogation non procédural permettant la recherche par contenu, la mise à jour, la persistance, la concurrence et la fiabilité.
- Assurer la gestion de larges volumes de données, pouvant atteindre les péta-octets (10**15) et assurer une gestion efficace de l'espace mémoire.
- Supporter des structures de recherche efficaces comme les Quadtree, les Rtree et leurs variantes et permettant une recherche rapide sur le contenu ou la sémantique.
- Etre capable de récupérer des informations à partir de sources hétérogènes.

L'interrogation d'objets multimédias passe par la recherche classique dans une BD structurée à partir d'attributs décrivant les objets (exact-match retrieval), mais surtout par la recherche basée sur le contenu des objets. Une requête typique est la recherche des k objets les plus similaires à un objet donné. On récupère d'abord un ensemble de résultats classés par ordre de pertinence et pour raffinement, il doit être interrogeable à nouveau.

2.3 Architecture d'un SGBD multimédia

Les SGBD multimédias sont basés généralement sur l'architecture physique ci-dessous :

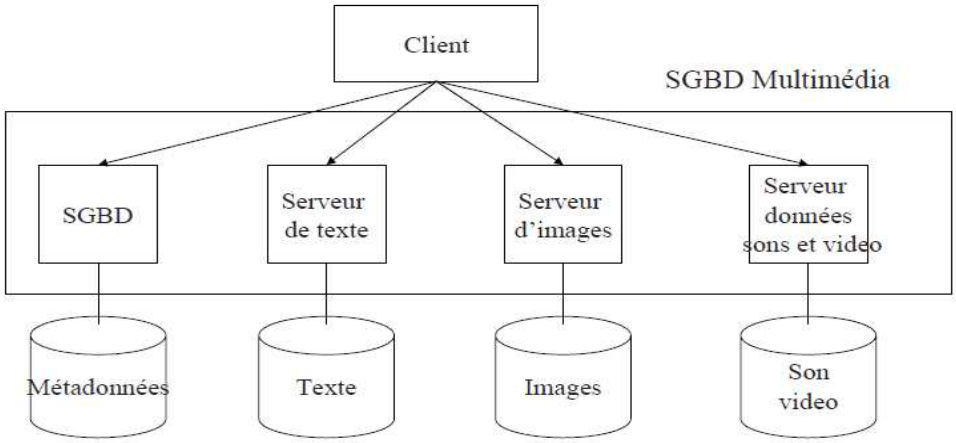


Figure 6. Architecture d'un SGBD multimédia.

Un SGBD multimédia nécessite une interface utilisateur permettant la manipulation des objets existants, la présentation de résultats, le retour de pertinence. Il doit posséder deux modes :

(1) Gestion du contenu (ajout, modification et suppression des objets présents) et,

(2) Interrogation de la base. Il doit permettre

- Le processus de requête est interactif
- Le processus de requête peut utiliser plusieurs modes de représentation (ex: texte, image)
- Ne pas reposer (uniquement) sur des annotations manuelles

2.5 Gestion du volume d'objets multimédias par les SGBDM.

Les objets multimédias sont volumineux et nécessitent des supports et des politiques de stockage appropriés à leurs caractéristiques. On trouve essentiellement trois types d'approches pour pallier ces problèmes.

*A. **Stockage par numérisation et utilisation de serveurs dédiés**.*

La numérisation du texte: s'opère simplement par codage de chaque caractère en une suite de 0 et de 1. Le code ASCII (American Standard Code for Information Inter-change) sur 7 bits permet de coder 128 caractères usuels. Comme les ordinateurs travaillent usuellement sur des mots qui sont des multiples de mots de 8 bits (octets), on peut rajouter un 0 devant le code ASCII, ce qui correspond au code normalisé. Ainsi le caractère "A" est codé 01000010.

Numérisation des images : Le pixel est la taille du plus petit élément de l'image. Il désigne aussi un point de la matrice image. Un pixel possède une valeur qui peut être un scalaire, représente un niveau de gris ou un vecteur représentant une couleur.
1) Les images dites en "*noir et blanc*" sont composées de pixels binaires noirs ou blancs (chaque point est codé sur 1 bits (0 pour noir, 1 pour blanc).
2) Les images en *niveaux de gris* sont composées de pixels de valeurs scalaires représentant la luminosité, Pour une image avec 256 nuances de gris allant du noir total au blanc total, chaque point sera codé sur 8 bits (de 00000000 pour noir à 11111111 pour blanc).
3) Pour une image *couleur*, on exprime la couleur comme une super position du système RVB (Rouge, du Vert et du Bleu).Si on permet une variation de 0 à 255 pour chaque rang de couleur, alors une couleur quelconque sera codée comme combinaison de trois octets, soit 24 bits. Donc 2^24 ou 16777216 couleurs possibles.
Pour les applications multimédias, les images sont converties en matrices de points. Chaque point est codé selon sa "couleur" et les codes résultants sont placés séquentiellement dans un fichier, ligne par ligne, colonne par colonne.
Numérisation du son : Les technologies d'acquisition du son ont été longtemps analogiques. Le son était représenté par les variations d'une grandeur physique, une tension électrique par exemple. Les techniques actuelles permettent d'obtenir directement un son numérisé: c'est notamment le cas des magnétophones produisant un enregistrement sur cassettes D.A.T. (Digital Audio Tapes, digitalisation du son à une fréquence de 48 KHz).
Pour numériser un son enregistré de manière analogique, on procède en trois étapes:

- Echantillonnage: l'amplitude du signal analogique est mesurée à une fréquence d'échantillonnage f. On obtient ainsi une collection de mesures.
- Quantification: une échelle arbitraire allant de 0 à 2^{n-1} est employée pour convertir les mesures précédentes. Une approximation est faite de manière à ce que chaque mesure coïncide avec une graduation de l'échelle (cette approximation, qui modifie légèrement le signal, est appelée bruit de quantification).

- Codage: suivant sa grandeur dans cette nouvelle échelle, chaque mesure est codée sur n bits et placée séquentiellement dans un fichier binaire.

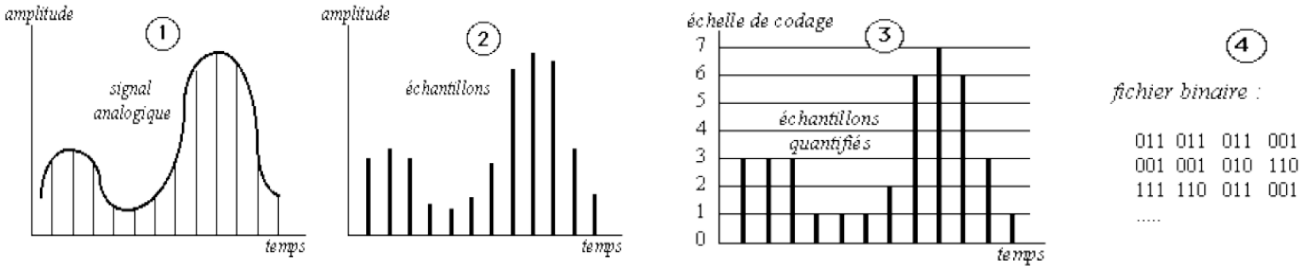


Figure 7. Phases de numérisation du son.

Numérisation de la vidéo: Comme pour le son, la vidéo s'exprime par des signaux de nature analogique qu'il faut numériser pour une utilisation multimédia. Il existe ici quatre signaux à numériser: trois pour l'image et un pour le son. Les signaux d'image se répartissent en signal de luminance Y et deux signaux de chrominance Db et Dr, reliés aux composantes R,V,B (Rouge, Vert, Bleu) par les relations linéaires.

* Y=0.30R+0.59V+0.11B * Db=B-Y * Dr=R-Y.

La composante Y est échantillonnée à une fréquence de 13,5 MHz, tandis que les composantes de chrominance sont échantillonnées à une fréquence de 6,75 MHz. La quantification pour les trois signaux d'image s'opère sur 8bits.

Les données peuvent être simultanément ou ultérieurement compressées.

Ensuite, la technique des **serveurs dédiés** est utilisée pour restituer des données (numérisées) continues telles que l'audio ou la vidéo qui nécessitent un flot continu de données avec une certaine qualité de service. Des techniques spécifiques de bufférisation sont aussi utilisées.

B. Compression de données.

Actuellement, la taille des données multimédia augmente plus vite que les capacités de stockage et largement plus que la bande passante des réseaux (car cela imposerait d'énormes changements dans les infrastructures de télécommunication). Par conséquent, il faut réduire la taille de ces données multimédia en exploitant la puissance des processeurs (traitement) c-à-d une méthode de compression. En effet,

la compression consiste à réduire la taille physique de blocs d'informations en compressant des données, (soit en plaçant plus d'informations dans le même espace de stockage, ou en utilisant moins de temps pour le transfert). Plusieurs méthodes de compression ont été définies permettant d'une part de diminuer l'espace de stockage requis et d'autre part de permettre l'acquisition ou la restitution de données continues avec un débit adapté. Les méthodes de compression dépendent du type de données à compresser car un fichier texte, une image, un fichier audio, ou une vidéo ne représentent pas le même type de données.

Les différentes méthodes de compression sont basées sur les critères :

- Le taux de compression: c'est le rapport de la taille du fichier compressé sur la taille du fichier initial. (Débit initial/ débit après compression)
- La qualité de compression: sans ou avec pertes (pourcentage de perte).
- La qualité du signal comprimé
- La vitesse de compression et de décompression.
- La complexité du système : coût calcul, mémoire requise

La compression peut être symétrique ou asymétrique. Une méthode de compression symétrique utilise le même algorithme et demande la même capacité de calcul, aussi bien pour la compression que pour la décompression. Par exemple, une application de transmission de données où la compression et la décompression sont les deux faits en temps réels sera généralement implémentée avec un algorithme symétrique si l'on veut atteindre la plus grande efficacité. Par contre, *la compression asymétrique* demande plus de travail pour l'une des deux opérations, la plupart des algorithmes requiert plus de temps de traitement pour la compression que pour la décompression. Des algorithmes plus rapides en compression qu'en décompression peuvent être nécessaires lorsque l'on archive des données pour des raisons de sécurité par exemple.

La compression peut être physique ou logique. La compression *physique* est basée sur un algorithme capable de comprimer des données (textes, sons, images et vidéo) dans un minimum de place. Le résultat de la compression d'un bloc de données est un bloc plus petit que l'original, car l'algorithme de compression physique a retiré la redondance qui existait entre les données elles-mêmes. Mais on peut également

adopter une approche de compression *logique* basée sur un algorithme de recodage des données dans une représentation différente plus compacte contenant la même information. La compression logique est réalisée par le processus de substitution logique qui consiste à remplacer un symbole alphabétique, numérique ou binaire en un autre. La substitution logique ne fonctionne qu'au niveau du caractère ou plus haute test basée exclusivement sur l'information contenue à l'intérieur même des données.

La compression peut être statistique ou numérique. Certains algorithmes *statiques* c-à-d non-adaptatifs (ex codage de Huffmann) sont conçus pour compresser seulement des types spécifiques de données. Ces algorithmes contiennent un dictionnaire statique de chaînes de caractères prédéfinies qui sont connues comme apparaissant à de grandes fréquences dans les données à encoder. Par exemple, un codeur conçu spécifiquement pour compresser la langue française contiendra un dictionnaire avec des chaînes de caractères telles que "et"," mais", "de", "le", car ces chaînes apparaissent très fréquemment dans les textes en français. Pour les algorithmes qui opèrent au niveau *numérique* (adaptatif), à l'inverse n'intégrera pas de données relatives à la fréquence d'apparitions des données à compresser. Des compresseurs adaptatifs comme LZW ou Huffmann dynamique déterminent la dépendance des données en construisant leur dictionnaire à la découvert. Ils n'ont pas de listes prédéfinies de chaînes de caractères par exemple, mais les construisent dynamiquement à l'encodage.

La compression peut être sans ou avec perte, A l'inverse de la compression avec perte, la compression sans pertes signifie que lorsque des données sont compressées et ensuite décompressées, l'information originale contenue dans les données a été préservée. Aucune donnée n'a été perdue ou oubliée. Les données n'ont pas été modifiées. Il existe deux types d'algorithmes de compression sans perte (1) Codage statistique dont le but est de réduire le nombre de bits utilisés pour le codage des caractères fréquents et augmenter ce nombre pour des caractères plus rares. (2) Substitution de séquences qui consiste à comprimer les séquences de caractères identiques. Cependant, le taux de compression des algorithmes sans perte est

insuffisant pour les données de type multimédia. Il faut donc utiliser la compression avec perte. Les algorithmes avec pertes s'appliquent généralement aux données ayant de forts taux de redondance, comme les images, ou les sons. Le principe de compression avec perte, est basé sur l'étude précise de l'œil et de l'oreille humaine. Les signaux audio et vidéo contiennent une part importante de données que l'œil et l'oreille ne peuvent pas percevoir et une part importante de données redondantes. Les objectifs de la compression avec pertes sont d'éliminer les données non pertinentes pour ne transmettre que ce qui est perceptible.

C. Indexation

L'objectif de l'indexation est de structurer la base pour ne pas avoir à la parcourir entièrement à chaque recherche, comme dans un dictionnaire. La difficulté de l'indexation réside dans les médias en question sont représentés par des vecteurs de grande dimension.

Les principales techniques d'indexation sont basées sur les R-Tree qui sont des extensions des index B-arbres ont été utilisées dans le contexte des présentations multimédias.

3. Annotations et métadonnées dans les bases de données multimédia

L'annotation consiste à associer une métadonnée et (un sens) aux données multimédia.

En effet les annotations peuvent aussi impliquer une description technique (non sémantique) des données multimédia, telle que la date de la prise de vue, et peuvent concerner la structure de données multimédia complexes. Les annotations peuvent être acquises automatiquement (difficile et risque d'erreur -vocabulaire différent pour signifier la même chose par différentes personnes-) ou générées manuellement (trop restrictive, coût élevé). L'approche la plus utilisée reste l'annotation textuelle et manuelle (Tag dans xml). Les annotations permettent une recherche indépendante du type de media et l'utilisation des techniques de base de données classiques. Trois *classes de métadonnées existent qui sont :* classe de données indépendantes du contenu, classe qui décrit le contenu et classe qui dépend du contenu.

Média	**Indépendant du contenu**	**Décrit le contenu**	**Dépend du contenu**

Texte	Date de mise-à-jour	Mots clés, format	Frontière des sujets
Son	Durée	Personne qui parle	Reconnaissance vocale
Imag	Titre, date	Format	Caractéristiques
Vidéo	Distributeur, durée	Angles des prises de vues	Frontières des séquences

4. Approches d'interrogation

Il existe trois approches d'interrogation d'une BD multimédia

(1) Approche ABR (***Attribute Based Retrieval***) consiste à utiliser un ensemble d'attributs (champs d'une table) comme dans les SGBD traditionnels pour l'interrogation. Cette approche est efficace pour des données textuelles mais n'utilise pas la richesse du contenu des images. Exemple: "Afficher les images des voitures de la marque "Renault""

(2) Approche TBR (Text Based Retrieval) : consiste à ajouter des annotations (courtes descriptions dans des attributs structurées). Les annotations générées et saisies manuellement par des professionnels, mais les requêtes formulées par des utilisateurs. Cette approche est difficile à réaliser en pratique. Exemple: « Afficher les images du :"match Chelsea vs Manchester United " « Description des vidéos sur Youtube !!! »

(3) Approche CBR (Content Based Retrieval) : c'est une approche alternative au TBR, ou l'extraction de plusieurs éléments d'information peut être automatisée. Le repérage dans cette approche est plus intuitif car utilise des éléments en relation naturelle avec les objets: texture, couleur, forme (attributs physiques du média).
•Exemple : afficher les images des "voitures rouges", forme + couleur.

4.2 Niveaux d'interrogation: Il existe trois niveaux d'interrogation d'une BD multimédia,

- **Niveau1**: il consiste à extraire automatique des caractéristiques physiques tels que la couleur, la forme, la texture, la localisation spatiale et le mouvement des objets. Exemple de La requête : « Trouver des objets qui possèdent du violet»,

- **Niveau 2**: il consiste à extraire automatique des caractéristiques logiques qui sont reliées à l'identité d'un objet dans le média. Exemple de requête: « Trouver une vidéo d'un avion en approche », « Trouver une image d'une baleine bleue»
- **Niveau3**: il consiste à extraire automatique des attributs abstraits associés à la compréhension (sens) de la nature et l'objectif de l'objet dans le média.
  Exemple de requête: «Trouver une photo d'un acte terroriste»

PARTIE 2: La distribution de données

Dans un contexte des applications multiutilisateurs, plusieurs utilisateurs peuvent utiliser simultanément un même fichier. L'un des mécanismes consiste à utiliser un disque partagé d'un serveur pour stocker le fichier et il sera accessible par différents postes de travail simultanément. Cette solution pose un certain nombre de problèmes quand le nombre d'utilisateurs devient important à savoir: l'augmentation du nombre d'accès disque sur le serveur. Une forte sollicitation du réseau. Une augmentation du nombre d'accès concurrents. Ces différents problèmes entraînent la chute des performances de la solution du « disque partagé» si on dépasse 30 utilisateurs.

Afin de résoudre ces problèmes (éviter la saturation du réseau et le nombre très important de requêtes à traiter par une seule machine), il y a nécessité de rapprocher les données vers les utilisateurs, ce qui a fait émerger le concept de répartition des données sur plusieurs machines. L'utilisation des bases de données réparties est l'une des solutions à ce problème.

Chapitre 5: Concepts et architectures des BD réparties

Objectif du chapitre

Dans ce cinquième chapitre, nous décrivons les concepts de base et l'architecture des bases de données réparties. Ce chapitre présente les architectures distribuées, leurs forces et leurs faiblesses de chacune et les mécanismes de mise en œuvre correspondants. Ce chapitre examine également différentes typologies des SGBD répartis et incite le lecteur à comprendre quel type est le plus approprié à un besoin et une situation donnée.

1. Concepts des bases de données réparties

Une base de données répartie est un assemblage de différentes bases de données stockées dans plusieurs ordinateurs mis en relation les unes avec les autres à travers une interconnexion réseau. Elle permet de combiner et rassembler des données plus ou moins hétérogènes via une interconnexion réseau sous forme d'une base de données globale, homogène et intégrée.

Dans les bases de données réparties, on peut distinguer entre la base de données logique et les bases de données physiques.

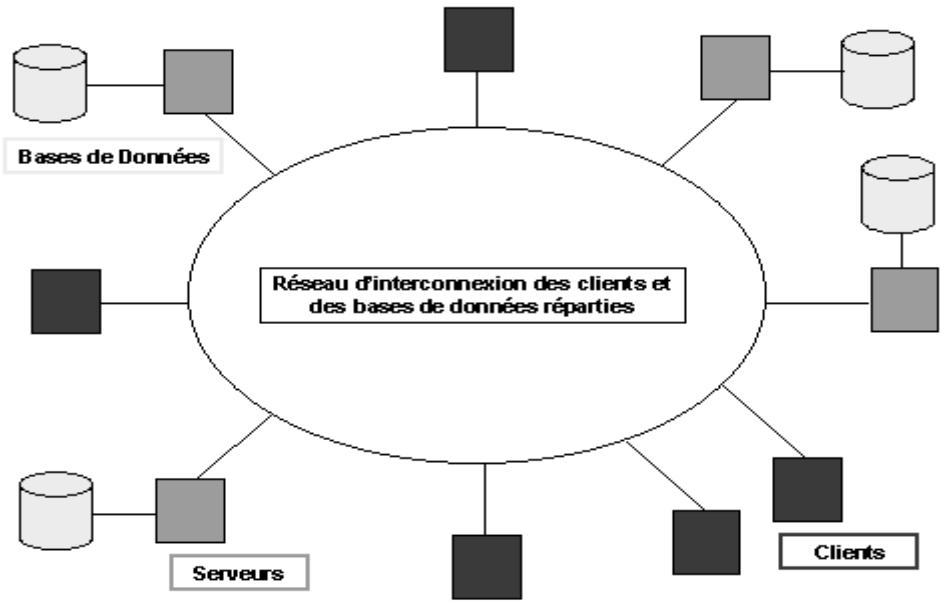


Figure 8. Base de données répartie

Base logique : est la base de données vue par l'utilisateur comme une seule et même base de données**.**

Base physique : représente chacune des bases de données regroupées au sein d'une base logique.

Une base de données répartie (Figure 8) ne doit donc pas être confondue avec un système dans lequel les bases de données sont accessibles à distance (selon le principe client serveur). Elle ne doit non plus être confondue avec un système multi-base ou chaque utilisateur accède à différentes bases de données en spécifiant leur nom et adresse (plusieurs BDs hétérogènes), et le système se comporte alors comme un serveur de BD et n'apporte aucune fonctionnalité particulière à la répartition. Au contraire, un système de base de données répartie est suffisamment complet pour décharger les utilisateurs de tous les problèmes de concurrence, fiabilité, optimisation de requêtes ou transaction sur des données.

Exemple: une banque possède des agences à Alger et à Bejaia. Dans le cas d'une BD centralisée, le siège social de la banque gérerait tous les comptes des clients et les agences de cette banque devraient communiquer avec le siège social pour pouvoir accéder aux données des clients. Par contre, dans le cas d'une BD répartie, les informations sur les comptes sont distribuées dans les agences et celles-ci sont interconnectées (entièrement ou partiellement) afin qu'elles puissent avoir accès aux données externes. Cependant, la répartition de la base de données bancaire est invisible aux agences en tant qu'utilisateurs et la seule conséquence directe pour elles est que l'accès à certaines données est beaucoup plus rapide.

2. Les règles des bases de données réparties

Quatre principales règles existent dans base de données réparties à savoir :

(1) Indépendance à la localisation. (3) Indépendance à la fragmentation/réplication

(2) Indépendance aux SGBD. (4) Autonomie des sites.

2.1 Indépendance à la localisation des données

Une des règles clés des BD repartie est de permettre d'écrire des programmes d'application sans connaître la localisation physique des données. Ainsi, les noms des objets de base de données doivent être indépendants de leur localisation. De la

même manière, les requêtes des utilisateurs exprimées en SQL sont réécrites avec des requêtes locales de façon transparente. Les avantages de la transparence à la localisation sont : de simplifier la vue utilisateur, la réécriture des requêtes, et particulièrement d'introduire la possibilité de déplacer les objets sans modifier les requêtes.

Les utilisateurs accèdent à la base soit directement par le schéma conceptuel soit indirectement au travers de vues externes (Figure 9). Mais en aucun cas ils n'ont les moyens d'accéder aux schémas locaux ni de préciser le site. C'est le principe de transparence de localisation. Dans l'exemple précédent, un client peut ouvrir un compte à Alger et effectuer régulièrement des opérations à Bejaia. C'est le système qui recherche le site où sont mémorisées ces informations et non l'utilisateur qui doit l'indiquer.

2.2 Indépendance à la fragmentation \ réplication des données

Une relation dans une base de données répartie est constituée de différents fragments (parties), localisés dans des sites différents. La relation principale ne doit pas dépendre de la manière de sa décomposition et doit pouvoir être modifiée sans modifier les programmes.

Les utilisateurs ne doivent pas savoir si une telle information est fractionnée et ne doivent donc pas se préoccuper de la réunifier. C'est le système qui gère les partitionnements et les modifie en fonction de ses besoins et c'est donc lui qui doit rechercher toutes les partitions et les intégrer en une seule information logique présentée à l'utilisateur.

Les utilisateurs n'ont pas à savoir si plusieurs copies d'une même information sont disponibles. C'est le principe de transparence de duplication. La conséquence directe est que lors de la modification d'une information, c'est le système qui doit se préoccuper de mettre à jour toutes les copies. Dans l'exemple précédent, il est fort possible que lorsqu'un client possède des comptes (courant, épargne logement, ...) à Alger mais, effectue régulièrement des retraits d'argent à Bejaia, les informations le concernant soient réparties entre Alger et Bejaia, l'historique des opérations du compte courant est conservé à Bejaia, la gestion de ses autres comptes est assurée à Alger et le solde du compte courant est dupliqué à Alger et Bejaia.

2.3 Indépendance aux SGBD

Une base de données répartie ne doit pas être dépendante des différents systèmes de gestion de bases de données. La relation globale doit pouvoir être exprimée dans un langage normalisé indépendant des constructeurs.

2.4 Autonomie des sites

Cette règle vise à garder une administration locale séparée et indépendante pour chaque serveur participant à la base de données répartie, et ainsi d'éviter la nécessité d'une administration centralisée. La reprise après une panne et les mises à niveau des logiciels doivent être réalisées localement et ne doivent pas avoir d'impacts sur les autres sites. Même-si chaque base travaille étroitement avec les autres bases, les gestions de schémas doivent rester indépendantes. Chaque base conserve son dictionnaire local contenant les schémas locaux.

3. Systèmes de gestion de bases de données répartis (SGBD Réparti)

En général, un SGBD réparti doit être capable d'offrir les mêmes services qu'un SGBD centralisé, en déchargeant les utilisateurs de tous les problèmes de concurrence, de fiabilité et d'optimalisation de requêtes. Ainsi, un SGBD réparti doit disposer de :

- Dictionnaire de données réparties
- Traitement de requêtes réparties
- Gestion des transactions réparties
- Communication de données inter-site
- Gestion de la cohérence et de la sécurité

Le SGBD réparti reçoit des requêtes référençant des tables d'une base de données réparties. Il assure la réécriture des requêtes distribuées en plusieurs sous-requêtes locales envoyées à chaque site. La réécriture de requête est une décomposition qui prend en compte les règles de localisation.

Pour ce qui concerne les mises à jour, le SGBD doit assurer la gestion des transactions réparties, en prenant en compte la vérification des règles d'intégrité multi-bases, le contrôle des accès concurrents et surtout la gestion de l'atomicité des transactions distribuées. Le SGBD réparti peut utiliser les fonctions locales de gestion de transactions pour accomplir les fonctions globales.

Dans le cas de données hétérogènes, le SGBD réparti doit être capable d'assurer la traduction des requêtes exprimées dans un langage de médiation (pivot) en requêtes compréhensibles par le SGBD local.

4. Architecture

Comme illustré dans la figure 9, l'architecture d'une base de données répartie est constituée des composants suivants.

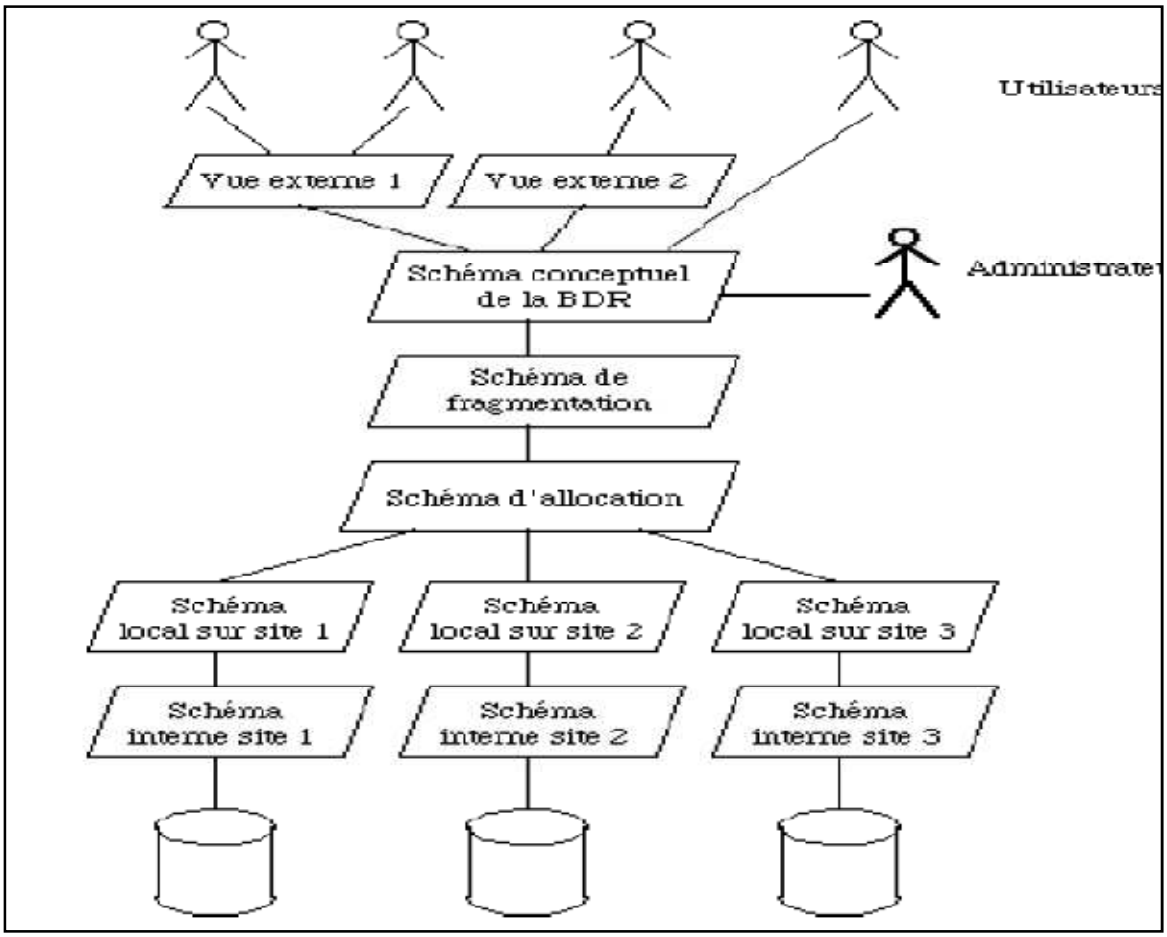


Figure 9. Niveaux d'une base de données répartie

4.1 Schéma local

Une base de données locale comporte un schéma géré par le SGBD local. Lors de la constitution d'une base de données répartie, chaque base locale rend visible une partie de la base aux sites clients.

4.2 Schéma global

Comme toute base de données, une base de données répartie possède un schéma appelé schéma global qui permet de définir l'ensemble des types de

données de la base. Le schéma global ignore les concepts d'implémentation. À ce titre, il est souvent appelé schéma conceptuel. Dans une base de données répartie, le schéma global n'est pas forcément matérialisé. Chaque base locale en implémente une partie. Ces parties locales sont les seules matérialisées sur les disques.

4.3 Niveaux de couplages

Il existe différents niveaux de couplages dans un système distribué. La littérature propose différents modèles et terminologies, mais les différents auteurs se contredisent parfois. Certains critères restent déterminants, soit la non-possibilité d'accès aux systèmes locaux, la matérialisation du schéma global, l'interdépendance des différents participants pour assurer la cohérence de l'ensemble, etc.

Par exemple la base maîtresse ne contient que des méta-données et les accès aux sites locaux sont prohibés. D'autres modèles, plus réalistes proposent une autonomie des sites locaux et instancient des données sur le site maître. Dans la pratique, on trouve encore plus fréquemment des systèmes faiblement couplés, ou les différents systèmes locaux communiquent, mais où il n'existe pas réellement de schéma global.

5. Typologie des SGBD répartis

La typologie des SGBD répartie concerne l'étude de l'autonomie, la relation entre les sites et la distribution de données (où et comment placer les données et comment les répartir).

5.1 Autonomie

L'autonomie se rapporte au degré avec lequel une des bases locales peut travailler indépendamment des autres. On peut distinguer trois types d'alternatives dans l'autonomie que peuvent avoir les bases locales.

La première est une intégration très poussée des bases de données locales. Une image unique de la base de données globale est offerte aux différents utilisateurs. Du point de vue utilisateur, la base de données est centralisée. Dans cette configuration, un SGBD, contrôle de bout en bout une requête d'un utilisateur même si la requête met

en jeu différentes bases locales et donc différents SGBD locaux. L'autonomie n'est donc pas bien importante.

La seconde est une autonomie intermédiaire. Les SGBD locaux peuvent opérer indépendamment, mais ils participent à une collection de bases qui coopèrent afin de partager leurs données. Une dernière alternative est l'isolation totale. Cette méthode pose de nombreux problèmes, car une base locale ne connaît ni l'existence des autres bases ni la façon de communiquer avec elles. Il ne peut donc pas y avoir de contrôle global quant à l'exécution d'une requête sur les différentes bases locales.

5.2 Relations entre machines

Dans l'architecture client-serveur, un groupe de machines, les serveurs, ont pour rôle de servir un autre groupe, les utilisateurs, que l'on nomme les clients (on prend ici une définition au niveau machine, on pourrait également en donner une au niveau processus). Par servir, on désigne la réalisation d'une tâche demandée par le client. Sur la machine cliente, les utilisateurs disposent de l'interface. Sur les serveurs, c'est la gestion des bases de données qui est effectuée (analyse et optimisation des requêtes, répartition). On peut distinguer deux types de clients.

Client-Serveur : Par ce terme on désigne un type de communication pour lequel une machine offre de service appelée serveur et des machines -qui demandent des services- appelées clients. On distingue entre deux types de clients, (1) le client lourd, où l'utilisateur est obligé de se connecter explicitement à tous les serveurs dont il a besoin pour la requête qu'il veut formuler. (2) le client léger, il offre plus de transparence. L'utilisateur ne se connecte qu'à la base de données via un unique serveur. Le système de gestion des bases données se charge alors de gérer les différentes connexions que nécessitera la requête de l'utilisateur.

Pair-à-Pair : Par ce terme on désigne un type de communication pour lequel toutes les machines ont une importance équivalente. Il n'y a pas de machine qui a une importance hiérarchique par rapport aux autres. C'est aussi ce qu'on peut désigner comme l'architecture totalement répartie.

Chacune de ces architectures possède des avantages et des inconvénients. Le client-serveur avec sa structure plus hiérarchique est très sensible aux problèmes de pannes des serveurs, bloquant les utilisateurs (les clients). En revanche, la prise de décision

des serveurs est rapide. Pour l'architecture pair-à pair, comme les machines sont équivalentes, une panne d'une machine ne fait que rendre le système un peu plus lent. Mais cette architecture engendre énormément de communication pour toute décision.

5.3 Hétérogénéité

L'hétérogénéité peut intervenir à plusieurs niveaux. On peut penser aux problèmes matériels, aux protocoles réseaux, mais ce qui nous importe ici, ce sont les problèmes spécifiques aux bases de données. L'hétérogénéité peut exister au niveau de la représentation des données, au niveau du langage de requête ou au niveau du modèle des différentes bases de données (bases de données relationnelles, bases de données objets).

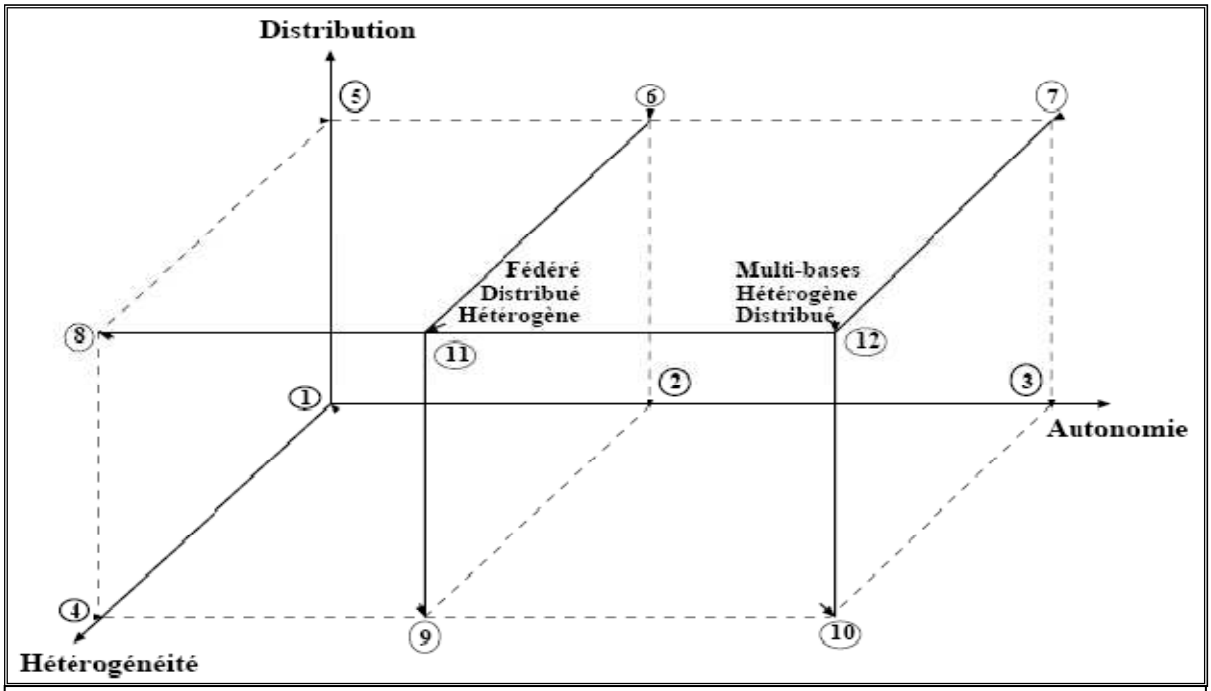


Figure 10. Typologie SGBD reparti

6. Inconvénients de la répartition

L'inconvénient majeur de la répartition des données d'une base entre plusieurs sites provient de la complexité résultant de leur coordination. Le coût de développement du système, la potentialité des pannes et la coordination de l'ensemble constituent des servitudes qui n'existent pas sur les systèmes centralisés.

Chapitre 6: Conception d'une base de données répartie

Objectif du chapitre

Dans les BDD réparties, les données sont délibérément distribuées pour tirer parti de toutes les ressources informatiques disponibles. Une approche de conception descendante prend en compte les besoins en données de l'ensemble de l'organisation. Une approche de conception ascendante prend en compte les données existantes distribuées au sein d'une organisation et utilise l'intégration de schéma pour créer au moins un schéma unifié**.** Ce chapitre étudie les différentes alternatives de conception et de la distribution de données pour un système de gestion de base de données distribué. Nous décrivons les alternatives de fragmentation, en particulier la fragmentation horizontale, verticale et hybride. Nous discutons de l'impact de la réplication des données et des problèmes de placement de données.

1. La conception des bases de données réparties

La conception d'une base de données répartie peut-être le résultat de deux approches totalement différentes, soit d'une part la nécessité de connecter une multitude de base de données existantes, ainsi une globalisation des systèmes informatiques est attendue, soit de composer une base de données en plusieurs ''sous'' bases de données. Deux types de conception des bases de données réparties existent qui sont:

1.1 Conception descendante

Cette méthode de conception est utilisée lors de la constitution de nouvelles bases de données. Sa démarche consiste à partir du schéma global, de construire des schémas locaux. Cette approche est généralement guidée par l'objectif de performances à obtenir par la mise à proximité des données aux utilisateurs potentiels.

1.2 Conception ascendante

La conception ascendante permet l'intégration de bases de données locales existantes dans une base distribuée. Il s'agit cette fois de construire un schéma

global à partir de schémas locaux, généralement existants. Cette démarche est la plus difficile puisqu'en plus des problèmes techniques identiques à ceux inhérents à une conception descendante, il faudra également résoudre des problèmes d'hétérogénéité des systèmes ou même sémantiques des informations.

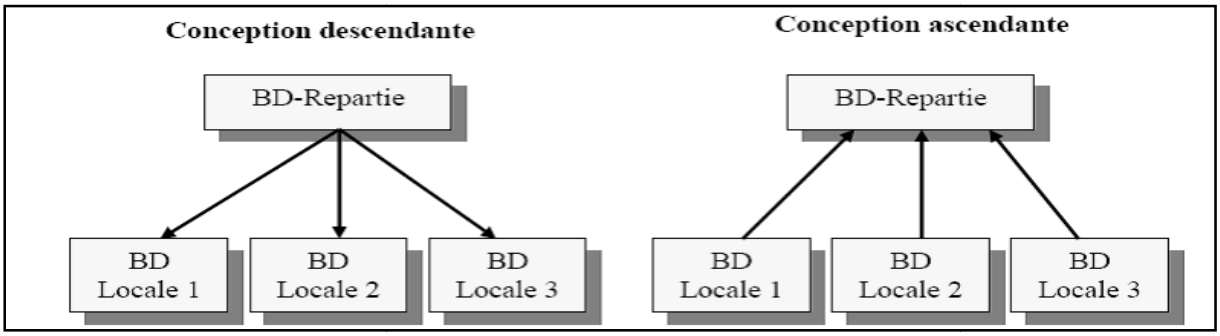


Figure 11. Type de conception d'une base de données repartie.

2. Fragmentation

La fragmentation 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 :

- ✓ La **complétude**: pour toute donnée d'une relation R, il existe un fragment R_i de la relation R qui possède cette donnée.
- ✓ La **reconstruction**: pour toute relation décomposée en un ensemble de fragments Ri, il existe une opération de reconstruction.

À partir d'une relation globale, la fragmentation consiste à isoler les sous-relations placées sur un même site. Elle peut s'effectuer par sélection de lignes (fragmentation horizontale) ou par projection sur des colonnes, ce qui représente la fragmentation verticale.

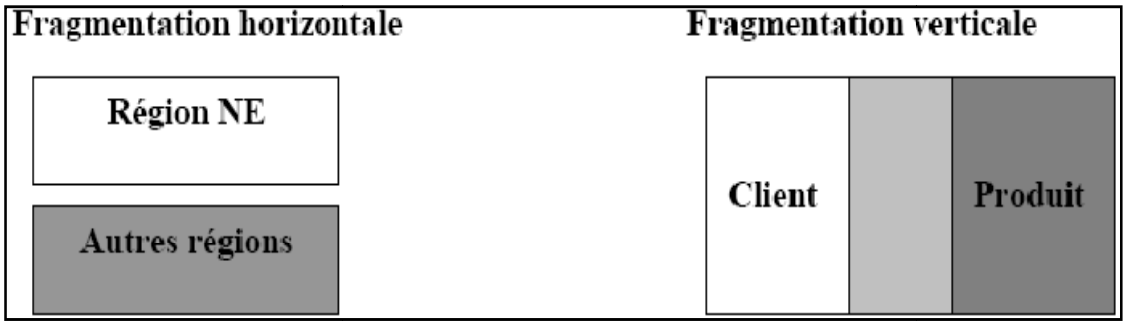


Figure 12. Type de fragmentation de tables.

Exemple

Relation Compte

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

Relation Agence

Agence	Adresse
Lausanne	Rue du lac, 3. 1002 Lausanne
Genève	Avenue du Mont Blanc, 21. 1200 Genève

Relation Client

No client	Nom client	Prènom	Age
174723	Villard	Jean	29
177498	Cattell	Blaise	38
201639	Tsellis	Alan	51
203446	Kowalsky	Valdimir	36

2.1. Fragmentation verticale

La fragmentation verticale est le découpage d'une table en sous-tables par des projections permettant de sélectionner les colonnes qui composent chaque fragment. Afin de ne pas perdre d'informations, la relation initiale doit pouvoir être recomposée par jointure des fragments.

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)
- L'opérateur de recomposition est la jointure (⋈)

Exemple

Soit le partitionnement de la relation précédente Client en deux relations Cli1 et Cli2:

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

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

Relation Cli1

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

Relation Cli2

La reconstruction de la relation de base se fait par :

```
CREATE VIEW V_commande
AS SELECT C1.numero, C1client, C2.produit, C2.qte
FROM commande1@Site1 C1,commande2@Site2 C2
WHERE C1.numero= C2.numero;
```

Vue Commande

numéro	client	produit	qté
1	1	54	10
2	1	52	12
3	2	64	25
4	4	18	51

Commande1=Commande(numéro,client)
Commande2=Commande(numéro,produit,qté)

Commande=numéro,client,produit,qté
WHERE Commande1.numéro=Commande2.numéro

Commande1

numéro	client
1	1
2	1
3	2
4	4

Commande2

numéro	produit	qté
1	54	10
2	52	12
3	64	25
4	18	51

2.2 Fragmentation horizontale

La fragmentation horizontale est un découpage d'une table en sous-tables par utilisation de prédicats permettant de sélectionner les lignes appartenant à chaque fragment. La relation initiale sera obtenue par union des fragments.

Les occurrences d'une même classe peuvent être réparties dans des fragments différents.

- L'opérateur de partitionnement est la sélection (σ)
- L'opérateur de recomposition est l'union (U)

Exemple

Dans l'exemple précédent, la relation Compte peut être fragmentée (horizontalement) en Compte1 et Compte2 avec la fragmentation suivante,

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

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

La reconstruction de la table Compte est réalisée par: Compte1 U Compte2

2.3 Fragmentation mixte

La fragmentation mixte consiste en l'application successive d'opérations de fragmentation horizontale et de fragmentation verticale sur une relation globale. Elle peut donc être définie comme une vue à partir des relations composant les fragments.(Utilisation de la fermeture de SQL). C'est la combinaison des deux fragmentations précédentes, horizontale et verticale.

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

Exemple

La relation Client est obtenue avec : (Cli1 ∪ Cli2) ⋈ Cli3 ⋈ Cli4 tel que :

Cli1= $\pi_{[NoClt, NomClt]} \sigma_{[Age < 38]}$Client, Cli2 = $\pi_{[NoClt, NomClt]} \sigma_{[Age \geq 38]}$Client

Cli3 = $\pi_{[NoClt, Prénom]}$Client et Cli4 = $\pi_{[NoClt, Age]}$Client

2.4 Fragmentation horizontale dérivée

La fragmentation horizontale dérivée (FHD) consiste à utiliser les fragments d'une table pour la fragmentation d'une autre table.

Dans l'exemple précédent des tables d'agences bancaires, on peut fragmenter les tables «Agence» avec leurs clients avec leurs comptes. On obtient alors deux réseaux d'occurrences liées. Le premier est relatif à la ville de Genève, et le deuxième est relatif à la ville de Lausanne.

3. Définition des fragments

Le principe est de fragmenter la relation de base sur l'ensemble des requêtes d'interrogation ou de mise-à-jour (Workload). Pour cela, il faut extraire de ces requêtes toutes les conditions de sélections, les attributs projetés et les jointures. Les opérations de sélection sont à la base des fragmentations horizontales, les opérations de projection sont à la base des fragmentations verticales.

Pour éviter le risque d'émietter la base de données, on peut restreindre le nombre de requêtes prises en considération en considérant uniquement qu'aux requêtes les plus fréquentes ou les plus complexes.

3.1 Définition des fragments horizontaux

Soient c_1, c_2, ..., c_n les conditions de sélection qui ont été extraites des requêtes (workload). Comme les fragments horizontaux doivent être exclusifs, on produit l'ensemble des 2^n conjonctions de condition où chaque condition élémentaire est

prise dans sa forme positive ou dans sa forme négative : CC= {$\wedge$ C_i^* $_{i=1,n}$ tel que C_i^* soit c_i soit $\neg c_i$ };

À la fin, on ôte de cet ensemble les conjonctions de condition qui sont toujours fausses, et on simplifie les autres.

3.2 Définition des fragments verticaux

Les fragments verticaux sont exclusifs, sauf en ce qui concerne le (ou les) attribut(s) de jointure qui sont communs à tous les fragments, et ce pour que la décomposition soit sans perte d'information. On procède comme suit,

1. Extraire toutes les expressions de projection concernées par les requêtes.
2. Ajouter à chacune d'elles le(s) attribut(s) de jointure si elle ne les possède pas.
3. Générer le complément de chaque expression (c'est à dire les autres attributs) en ajoutant le (ou les) attribut(s) de jointure.

L'étape suivante consiste, pour chaque fragment Fi produit par la fragmentation horizontale, à rechercher toutes les requêtes qui concernent ce fragment et à prendre les expressions de projection de ces requêtes.

À partir des n expressions de projection qui concernent Fi, il faut produire l'ensemble des 2^n intersections des expressions de projection.

4. Schéma de répartition

Chaque fragment est placé sur un site. Un schéma doit être élaboré afin de déterminer la localisation de chaque fragment et sa position dans le schéma global (comme indiqué dans la figure 13). Pour déterminer le ou les sites à qui envoyer une opération, il faut prendre en compte les règles de localisation des données. Celles-ci permettent de localiser des fragments de relations. Lors de l'exécution d'une requête distribuée, le SGBDR doit décomposer sa requête globale en plusieurs requêtes locales en utilisant son schéma de répartition.

5. Schéma d'allocation

L'affectation des fragments sur les sites est décidée en fonction de l'origine (sites d'émission) des requêtes. Le but est de placer les fragments sur les sites où ils sont le plus utilisés, et ce pour minimiser les transferts de données entre les sites.

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

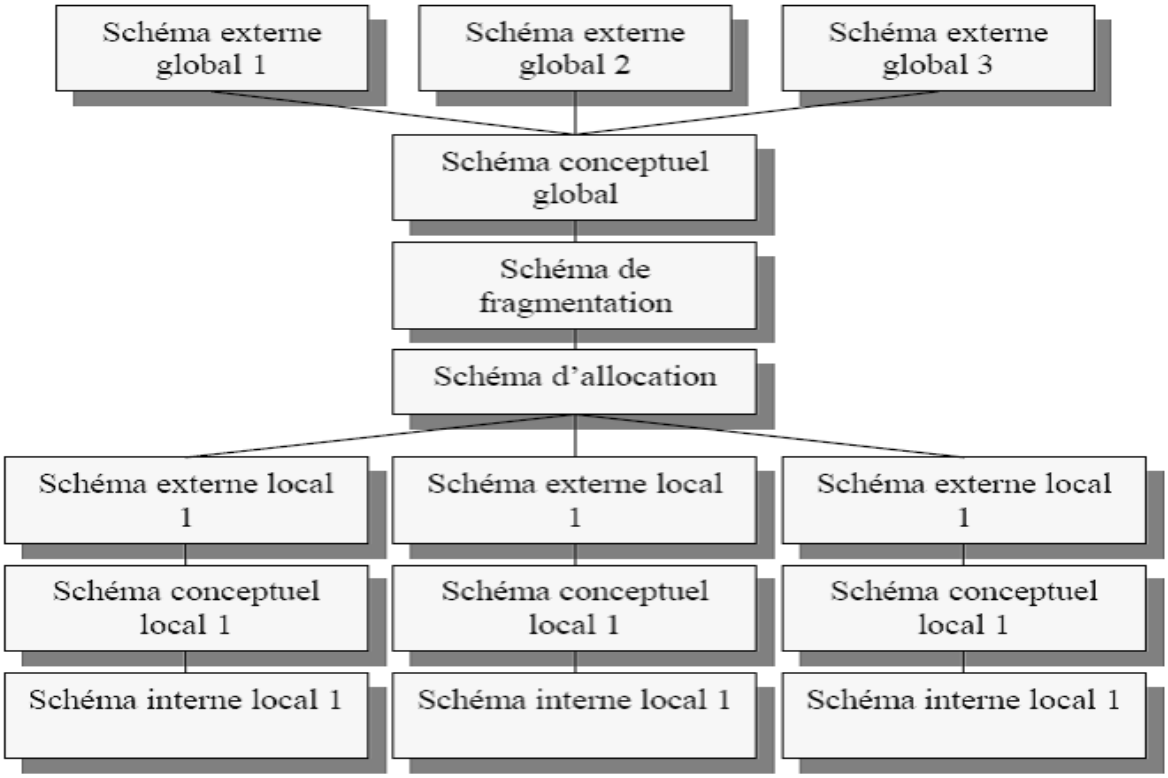


Figure 13. Schéma de répartition dans une BD répartie.

6. Réplication

Un cliché ou snapshot représente un état de la base de données à un instant donné. La pertinence d'un cliché diminue au fur et à mesure que le temps passe. L'utilisation des clichés permet l'amélioration des performances d'accès et la réduction du trafic sur le réseau.

On peut se passer de l'information exacte si les données ne sont pas fréquemment mises à jour (nom de famille, l'adresse…).

Les deux critères qui sont à prendre en compte pour définir l'intérêt d'un cliché sont d'une part l'ancienneté du cliché, et d'autre part le temps d'attente qui serait nécessaire avant d'obtenir l'information originale (à jour). Ces deux informations, l'ancienneté et le temps d'attente, peuvent être pondérées par un taux de satisfaction.

Chapitre 7: Traitement et optimisation de requêtes réparties

Objectif du chapitre

En effet, pour une requête donnée, il existe de nombreuses alternatives d'évaluation. La raison de l'existence d'un grand nombre d'alternatives est le grand nombre de facteurs qui affectent l'évaluation des requêtes. Ces facteurs incluent le nombre de relations dans la requête, le nombre d'opérations à effectuer, le nombre de prédicats appliqués, la taille de chaque relation dans la requête, l'ordre des opérations à effectuer, l'existence d'index ou non, etc. Ainsi, une requête exécutée dans un environnement de base de données devra forcément passer par une phase d'optimisation de requête. Dans ce chapitre, nous présentons un aperçu des différentes phases de traitement des requêtes mettant l'accent sur l'optimisation des requêtes dans des environnements de bases de données centralisées et distribuées.

1. Traitement de requêtes centralisées

L'exécution d'une requête est une série d'opérations permettant d'extraire des données de la base. Cependant, l'optimisation de requête est l'activité permettant de choisir la meilleure stratégie d'exécution d'une requête.

Le traitement d'une requête dans une BD centralisée passe par les trois étapes suivantes :

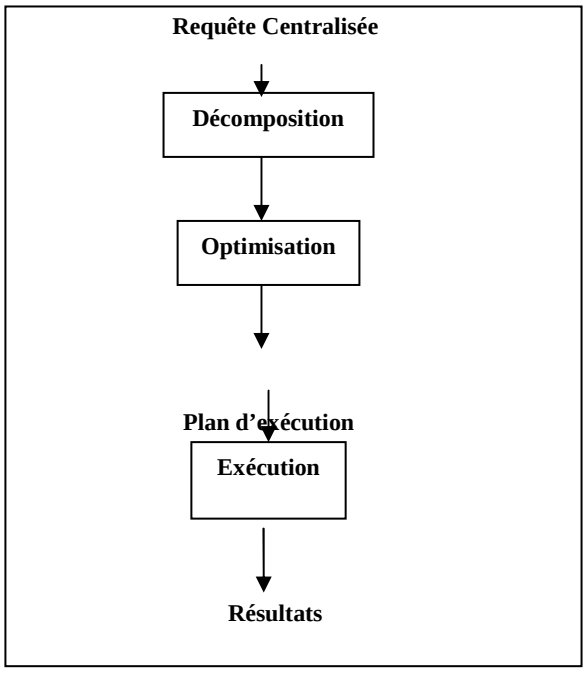


Figure 14. Évaluation d'une requête dans une BD centralisée.

A) Décomposition *:* Durant l'étape de décomposition, la requête SQL est transformée en arbre algébrique.

B) Optimisation : Durant cette étape, des optimisations sont appliquées sur l'arbre issu de la phase précédente, afin de déterminer l'ordonnancement optimal des opérations relationnelles, ainsi que l'application des algorithmes d'accès aux données. L'optimisation utilise :

- Les propriétés des opérateurs algébriques telles que l'associativité, la commutativité et la distributivité.
- Les heuristiques : par exemple en évaluant en premier les opérations les moins coûteuses, et en faisant descendre *les sélections,* ce qui réduit le nombre de tuples, ainsi que *les projections* qui réduit le nombre de colonnes et taille de données.

C) Exécution : c'est la dernière étape, qui consiste en l'exécution de la requête optimisée afin d'élaborer son résultat.

2. Traitement des requêtes réparties

Dans les bases de données réparties, une requête passe par quatre étapes pour être traitée, qui sont :

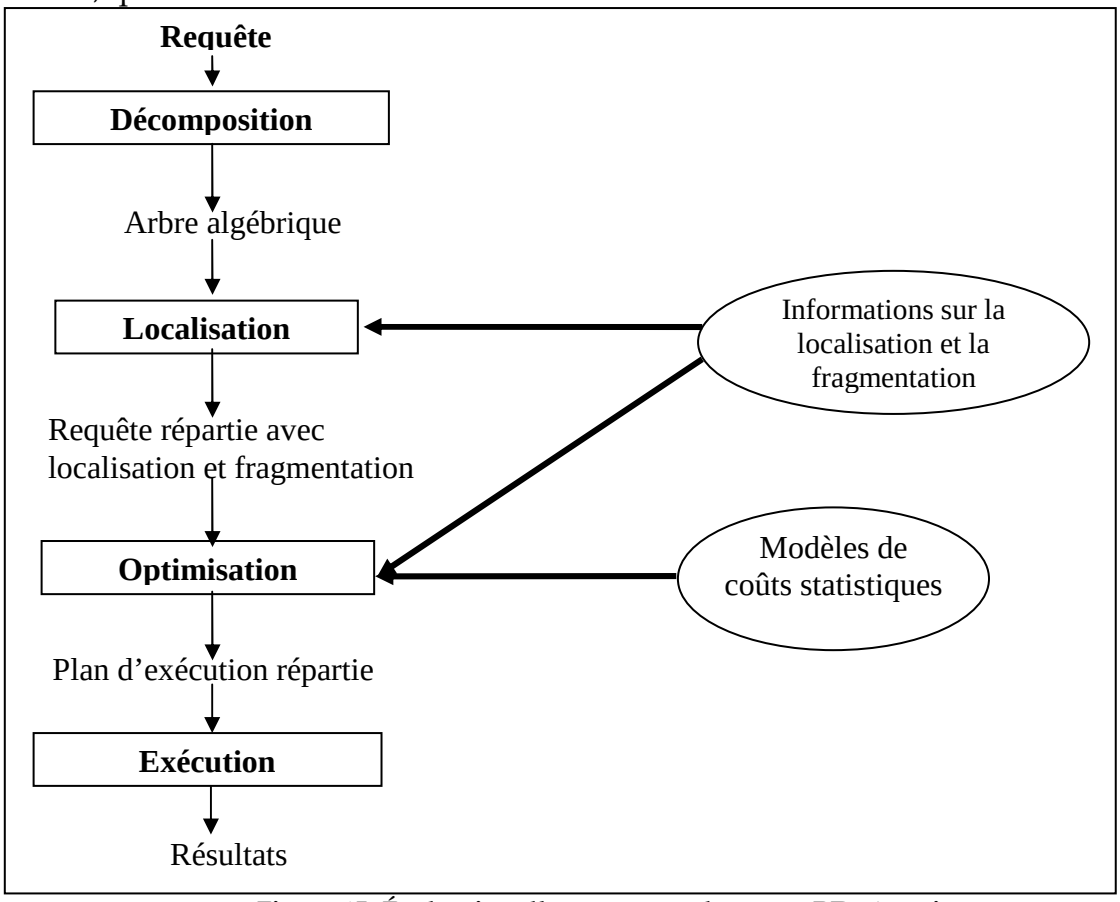


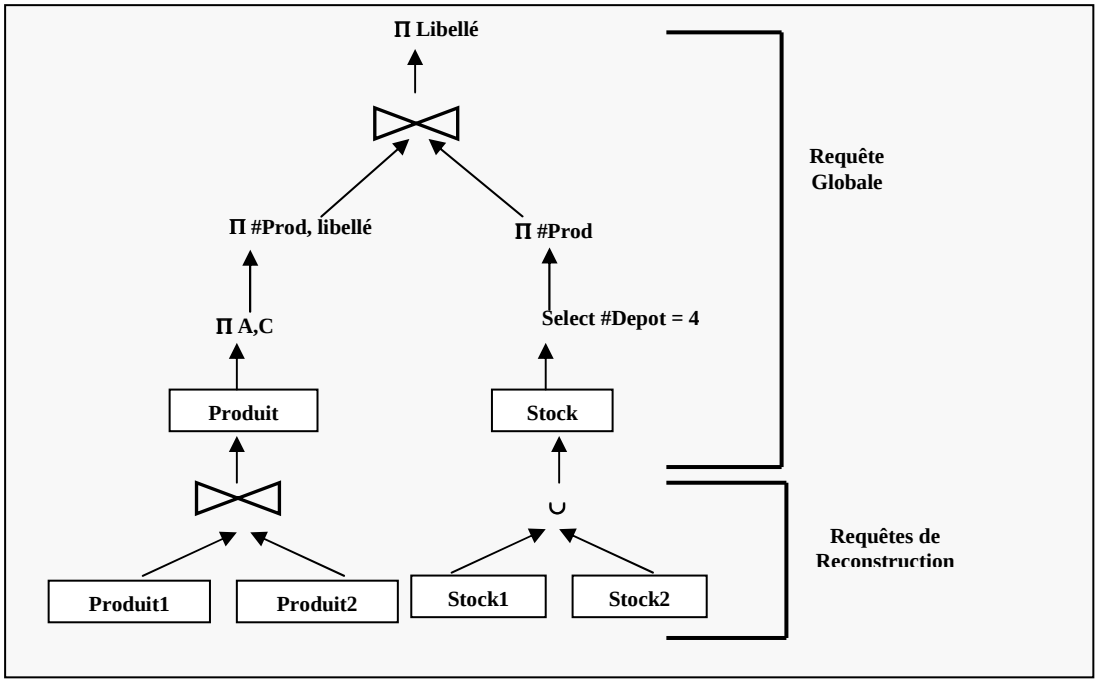
Figure 15. Évaluation d'une requête dans une BD répartie.

2.1 La décomposition

Durant l'étape de décomposition, la requête globale est décomposée en sous-requêtes locales.

Exemple : ***Relation :*** *Produit (Prod, Libellé, Pu…,) ; Stock (Dépôt,…)*

La requête globale Req **=** *Quels sont les produits du dépôt n°4 ?*



Site1 **Site2**

Produit1= π#prod,libellé (Produit)

Produit2 = π#prod,Pu (Produit)

Stock1 = θ# Dep=4 (Stock)

Stock2 = θ # Dep <> 4 (Stock)

2.2 La localisation

Durant cette étape, la localisation d'une requête distribuée s'effectue en deux étapes :

A) *Génération de requête canonique :* la requête ''canonique'' est obtenue en remplaçant chaque relation de la requête distribuée globale par la requête de reconstruction correspondante.

B) *La simplification :* cette opération est basée sur des règles de simplifications, qui permettent de déterminer les opérations inutiles dans une requête canonique. Sachant que les opérations inutiles sont celles qui produisent un résultat vide, ou identique à l'opérande. Parmi ces règles, on cite :

– Une opération de restriction sur un fragment horizontal, dont le prédicat est en contradiction avec le prédicat de fragmentation, produit un résultat vide.

– Éliminer les conditions de sélections *inutiles,* quand la condition est identique à celle de la fragmentation.

– Une opération de projection sur un fragment vertical, dont tous les attributs projetés- à l'exception, de l'attribut commun de reconstruction- n'appartiennent pas au fragment, donne un résultat vide.

2.3 Optimisation

Le rôle de cette étape d'optimisation est de déterminer une stratégie d'exécution de la requête distribuée qui minimise une fonction de coût, ceci évitera des stratégies qui conduisent à de mauvaises performances.

Les fonctions de coût : les fonctions de coût à minimiser sont généralement les suivantes :

- ***La fonction de coût*** *«temps total d'exécution»* : dans le cas d'un SGBD centralisé, ce temps est la somme, du temps des entrées-sorties (accès disque) et du temps du calcul en unité centrale. Dans le cas distribué, on ajoute le temps de communication nécessaire, à l'échange des données entre différents sites participants à la requête distribuée.
- ***La fonction de coût*** *«le temps de réponse» :* ce cout prend en considération les traitements exécutés en parallèle.

L'objectif de la phase d'optimisation est de minimiser la les temps d'exécution des sous requêtes sur chaque site et de tenir compte de :

– Du parallélisme,
– Des coûts de transferts.
– Des profils des fragments (taille, nombre de n-uplets, taille du domaine des attributs…)
– De la taille des résultats intermédiaires.
– De la topologie du réseau

2.4 L'exécution

Le processus d'exécution de requêtes distribué consiste en plusieurs processus d'exécution de requêtes locales et d'un processus d'exécution global.

Le plan de l'exécution : c'est l'ensemble des traitements locaux et les opérations de communication de données intermédiaires nécessaires pour les exécutions locales et la synchronisation globale. Les plus impotentes décisions à prendre dans le plan d'exécution sont les suivantes

- L'ordre et les stratégies des jointures.
- Sélection des copies de données (par exemple le site le plus proche ou le moine engorgé).
- Choix des sites d'exécution (fonction des coûts de communication).
- Choix des algorithmes d'accès répartis.
- Choix du mode de transfert (soit par tuple ou par paquets)

Exemple

Soit la base de données des fournisseurs suivante :

Fournisseur (nf, nomf, codf, villef),

Produit (np, désignation, villep),

Four-prod (nf, np, qte)

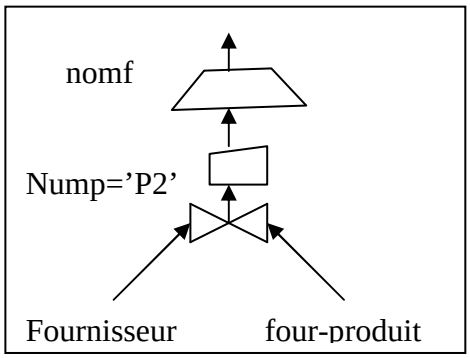
Avec les hypothèses suivantes : Cardinalité de la relation fournisseurs = 100, et cardinalité de la relation four-produit = 10000

Soit la requête suivante : « donner les noms des fournisseurs de la pièce P2 »

La requête SQL est la suivante :

SELECT nomf
FROM fournisseur, four-produit
Where nump = 'P2'
and fournisseur.numf= four-produit.numf

Un arbre algébrique non optimisé de la requête est le suivant



A) Stratégie non optimisée

L'évaluation de l'arbre non optimisé de la requête précédente est comme suit :

La 1ère étape consiste en la jointure des relations fournisseur et four-produit comme suit : Lecture des 10000 tuples de four-produit, puis lecture de chacun des 100 fournisseurs 10000 fois, construction du résultat intermédiaire de jointure (100 ×10000 tuples joints). Ce résultat est volumineux et (peut-être) ne sera pas gardé en mémoire.

La 2ème étape consiste en la restriction du résultat de l'étape précédente :les seuls tuples vérifiant nump='P2' seront gardés. Nous supposons qu'ils sont 50 et peuvent être gardés en mémoire.

La 3ème étape consiste en la projection sur le nomf par le parcours des 50 tuples du résultat précédent afin de ne garder que nomf.

* Nous constatons qu'il y a une perte de temps en effectuant la jointure en premier du fait des lectures répétées.

B) Stratégie optimisée

Une autre stratégie beaucoup plus efficace est la suivante :

- La 1ère étape consiste à effectuer la restriction de four-produit à nump= 'P2' :lecture des 10000 tuples de four-produit pour ne garder en mémoire qu'une table de 50 tuples.
- La 2ème étape consiste à effectuer la jointure du résultat précédent avec fournisseur, cette étape entraîne l'extraction des 100 fournisseurs. Le résultat alors contient 50 tuples.
- La 3ème étape consiste à effectuer la projection.

Ainsi nous constatons que la commutation de la jointure et la restriction réduit considérablement le nombre de transferts de la mémoire secondaire vers la mémoire centrale, donc implique un gain de temps et donc une réduction du coût.

3. Ordonnancement des opérations

L'ordonnancement des opérations relationnelles est complexe, elle est basée sur une restructuration algébrique. L'optimisation dépend en grande partie de l'ordre d'exécution des opérateurs apparaissant dans l'arbre algébrique utilisé.

3.1 Stratégie générale d'une optimisation

Une stratégie d'optimisation consiste tout d'abord à traduire les sous-arbres booléens en une opération ayant un prédicat composé. Puis, effectuer les opérations unaires de restriction et projection le plus tôt possible car elles réduisent respectivement le degré

et la cardinalité d'une relation. Ensuite, pré-traiter les relations avant d'effectuer la jointure soit en créant des index soit en effectuant le tri. Rechercher les sous expressions communes à plusieurs sous-arbres pour les pré-évaluer. Et enfin, traiter en une seule fois les opérations de restriction et projection sur une même relation.

3.2 Les règles de transformation des arbres

Afin de transformer l'arbre d'une requête, un ensemble de regèles suivantes sont utiliser.

Règles de regroupement

Règle 1 : Regroupement des projections

π (π (R/ A1, A2, …, An)**) /** B1, B2, …, Bp**) = π (**R / B1, B2, …, Bp**),** Tel que **{**Bj**} ⊂ {**Ai**}**

Règle 2 : regroupement des restrictions

σ(σ (R / Xi θ a**)/** Xj θ b **) = σ (** R / Xi θ a ∧ Xj θ b**)**

Règles de commutativité

Règle 3 : Commutativité

R1×R2=R2×R1, R1⋈R2 = R2⋈ R1, R1∪R2 = R2∪R1, R1∩R2=R2∩R1

Règles de commutation

La distributivité d'un opérateur Θ sur un autre opérateur θ s'exprime comme suit :

Θ (R1θ R2) = (Θ R1) θ (Θ R2)

Règle 4: Commutation des restrictions et projections

1er cas : l'argument de restriction fait partie des attributs de projection

π/(X1,…, ,Xi) **(σ** (R/ (Xp θ a)**)** = **σ**/ (Xp θ a) **(π(**R / (X1,…, Xp…,Xi)**)**

2ième cas : Sinon

π /(X1,…, ,Xi) **(σ** (R/ (Xp θ a)**)** = **π**)/ X1,…,Xi **(σ**/ Xp θ a **(π** (R/(X1,…,Xi, Xp))**)**

Règle 5 : Commutation des restrictions avec l'union

σ (Union (R1, R2) / Xp θ a) = **Union (σ(**R1/ Xp θ a**), σ(** R2/ Xp θ a**))**

Règle 6 : Commutation des restrictions avec la différence

σ (Minus (R1, R2) / Xp θ a) = **Minus (σ(**R1/ Xp θ a**), σ(** R2/ Xp θ a**))**

Règle 7 : Commutation des restrictions avec le produit cartésien

1er cas : la restriction porte sur l'argument d'une seule relation :

les deux relations n'ont pas d'attributs communs : R1 (…., Xi, …), R2 (…., Yj, …)

σ (R1 ⊗ R2**) /** (Xi θ a) **) = σ (**R1**/** (Xi θ a)**) ⊗** R2

2ième cas : la restriction porte sur deux arguments respectifs de R1 et R2 , les deux relations n'ont pas d'attributs communs : R1 (…., Xi, …), R2 (…., Yj, …)

σ (R1 ⊗ R2**) /** (Xi θ a ∧ Yj θ b) **) = σ (**R1**/** (Xi θ a)**) ⊗ σ (**R2**/** (Yj θ b)**)**

3ième cas : la restriction porte sur deux arguments respectifs de R1 et R2 , les deux relations ont un attribut commun Xp : R1 (…., Xi, Xp, …), R2 (…., Xp, …)

σ (R1 ⊗ R2**) /** (Xi θ a ∧ Xp θ b) **) = σ ((σ (**R1**/** (Xi θ a)**) ⊗**R2 **)/** (Xp θ b) **)**

Règle 8 : Commutation des projections et produit cartésien

Soient R1 (…., Xi, …) et R2 (…., Yj, …)

π((R1 ⊗ R2**) /** X1, …, Xp, Y1, …Yp **) = π(**R1 / X1, …, Xp**) ⊗π(**R2 / Y1,…,Yp**)**

Règle 9 : Commutation des projections avec l'union

π((R1 **Union** R2**) /** X1, …, Xp **) = π(**R1 / X1, …, Xp**) Union π(**R2 / X1,…,Xp**)**

Règle 10 : Cas de la jointure naturelle

Une jointure naturelle de R1 (X1…., Xi, …,Xn), R2 (Y1,…., Xi, …,Ym) est une restriction d'un produit cartésien à Xi = Xi suivie de la projection sur R1R2

Commutation avec la projection :

π((R1 ⋈ R2 **)** / X1….XpY1…Yq **) = π(π(**R1 / X1, …, Xp,Xi**) ⋈ π(**R2 / Y1,…,Yp,Xi**)/** X1..XpY1..Yq **)**

Remarque : la dernière projection n'est nécessaire que si Xi ∉ à X1,…Xp ou Y1,..,Yq

***Commutation avec la restriction* :** R1 (…,Xi ,…) et R2 (…,Xi,..Yj,…)

σ((**R1 ⋈ R2**) **/ (Yj θ b)**) = **R1 ⋈** (σ(**R2 / (Yj θ b)**)

A toutes ces règles peuvent être rajoutées des **règles d'associativité.** Tel que l'associativité est définie comme suit : AΘ (BΘ C) = (AΘ B) Θ C.

L'union, l'intersection, et la jointure sont toutes associatives mais pas la différence.

3.3 Algorithme simple de génération d'un plan de requête

Nous décrivons dans la suite un algorithme qui génère des plans de requêtes en utilisant les règles précédentes appelé *Miranda* :

- Appliquer les deux premières règles pour regrouper les restrictions et projections
- Faire descendre les restrictions au plus bas possible en utilisant la règle R4 (pour passer une projection), et en utilisant les règles R5 et R6 (pour passer une union ou une différence) et la règle R7 (pour passer un produit ou une jointure) et combiner éventuellement des restrictions avec R2.

- Faire descendre les projections à travers les projections en utilisant la règle R1, à travers les restrictions (R4), à travers les produits (R8), à travers les jointures (R10).
- Supprimer les projections redondantes au niveau des feuilles de l'arbre de requêtes si tous les attributs sont cités et au niveau des nœuds par l'utilisation de la règle de regroupement R1.

Plusieurs plans de requêtes peuvent être générés par transformations d'expressions. L'optimiseur devra choisir alors le moins coûteux par l'étape d'estimation des coûts d'exécution qui permettra de choisir.

4. Estimation des coûts d'exécution

L'une des solutions pour choisir le meilleur plan d'exécution consiste à générer tous les cas possibles et puis estimer le coût d'exécution de chacun et enfin choisir celui ayant le coût minimal. Cependant, cette démarche n'est pas très réalisable car le nombre de plans peut être très grand et aussi la taille des résultats intermédiaires pris en compte est un surcoût.

La solution consiste alors à utiliser des connaissances statistiques stockées et des connaissances sur les coûts d'implantation des opérateurs, afin de choisir la meilleure stratégie d'exécution de la requête.

4.1 Utilisation de données statistiques

Dans cette approche, l'estimation du coût d'exécution est effectuée en utilisant les statistiques -sur la base de données- enregistrés dans les catalogues. En effet, certaines informations comme la cardinalité des relations existent déjà dans les catalogues, certaines d'autres dites ''informations complémentaires'' sont disponibles comme le nombre de pages occupées par une table, le nombre distinct de valeurs dans une colonne, le nombre de niveaux d'index, le nombre de pages de chaque niveau.

4.2 Utilisation de procédures d'implantation des opérateurs

Dans cette approche, c'est le programme d'optimisation qui aura à sa disposition un ensemble de procédures d'implantation prédéfinies des opérateurs. Chaque procédure aura une formule de coût paramétrée qui pourra ainsi guider l'optimiseur dans son choix. Par exemple, l'opérateur de restriction pourra être implémenté selon

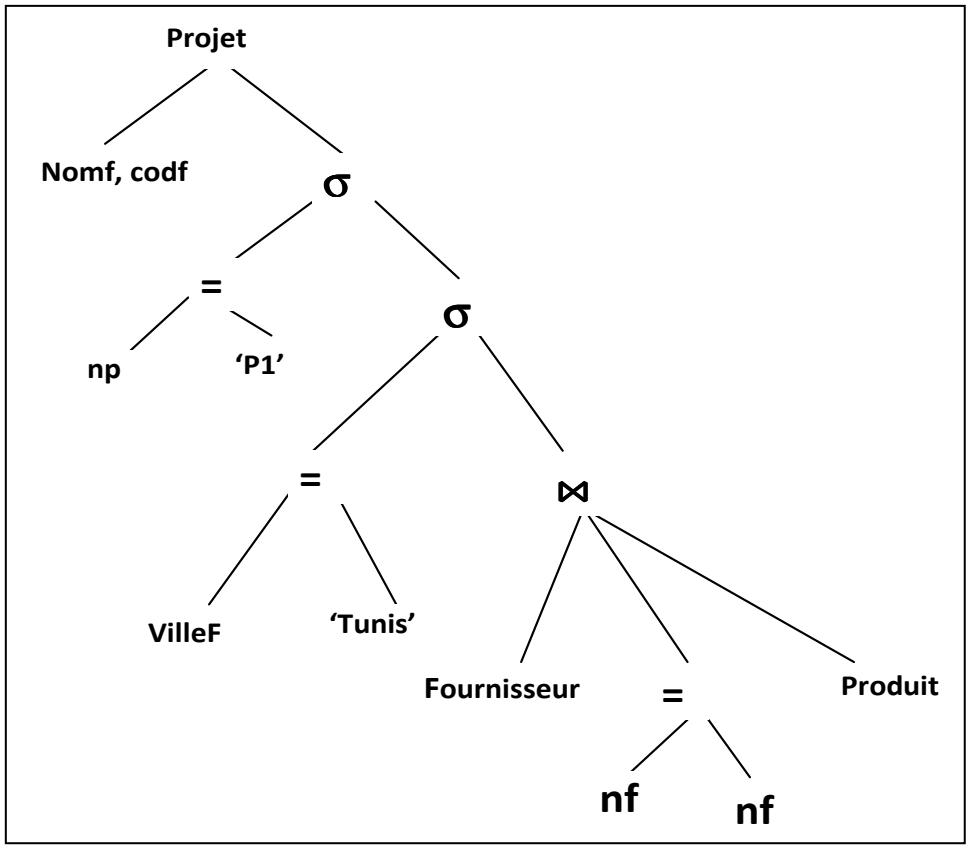
la formule de qualification, si elle fait référence à une clé, ou à un attribut non clé, mais indexé, ou tout simplement à un attribut non clé non indexé. Un autre exemple, est l'opérateur de jointure de deux relations R1 et R2, qui pourra être implémentée brutalement à l'aide de deux boucles imbriquées (une boucle externe complète sur les occurrences de R1 et une boucle interne complète sur les occurrences de R2), ou s'il existe un index sur l'attribut de jointure en effectuant un accès direct au tuple (recherche indexée) sans la deuxième boucle, ou encore une implémentation par fusion en triant les deux relations suivant l'attribut de jointure, et ainsi le parcours des deux relations sera synchronisé et ne fournira qu'un seul parcours de données sans redondance.

Exemple : soit la requête suivante sur la base de données des fournisseurs

SELECT *nomf, codf* **FROM** Fournisseur, Four-Prod

WHERE Fournisseur.nf = Four-Prod.nf **AND** VilleF = 'Tunis' and np = 'P1'

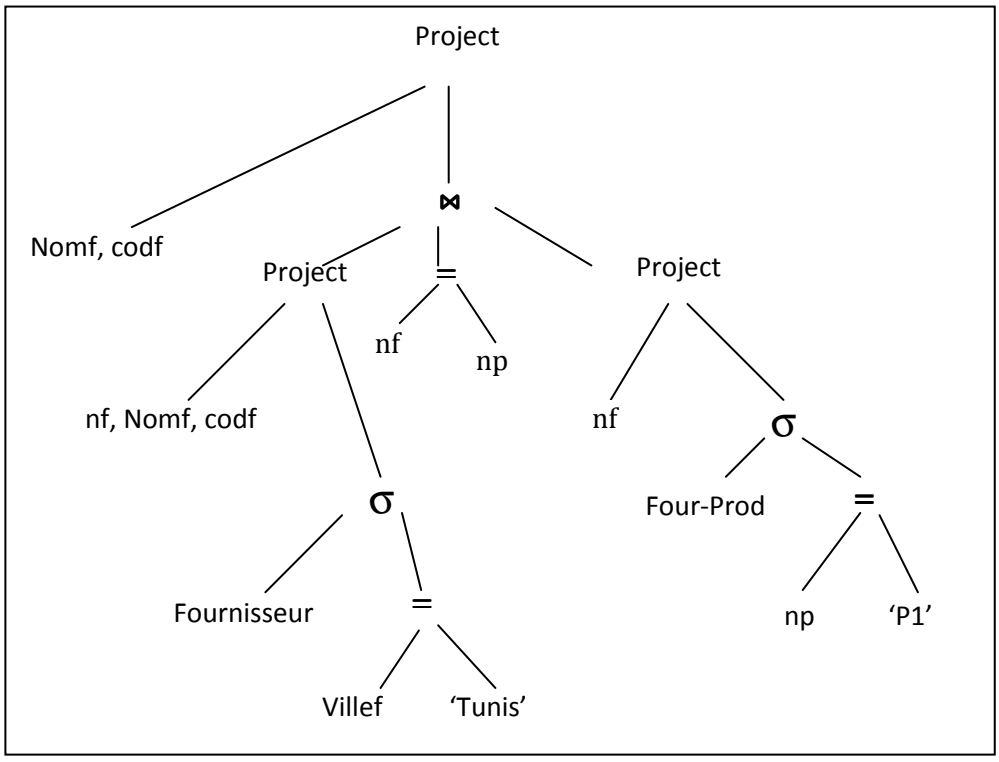
L'arbre canonique est le suivant:



Une séquence SQL peut donner lieu à différentes interprétations qui peuvent conduire à différents temps d'exploitation. Donc, l'étape d'optimisation consiste à :

- grâce aux différents catalogues qui décrivent la base (appelés méta-bases), le système peut faire la correspondance entre les noms symboliques (noms de relations, d'attributs) et les objets effectivement gérés par le système (analyse sémantique).
- Le système vérifie que l'usager qui a émis la requête va manipuler les objets qu'il est effectivement autorisé à manipuler (gestion des droits d'accès et de la confidentialité).
- Un ordre peut faire référence à une relation de base ou à une vue. La vue est définie par référence à des relations de base. Il convient alors de remplacer la référence à la vue par une référence aux relations de base (phase de substitution).
- Déterminer les chemins d'accès disponibles et la meilleure stratégie d'accès aux données. Pour cela, le système dispose dans ces catalogues des index définis sur chaque relation, mais aussi d'informations statistiques sur les relations à manipuler.

Résultat de l'étape : arbre optimisé à partir duquel il y a exécution. L'interpréteur SQL fournit le résultat de la requête en exécutant les opérateurs contenus dans l'arborescence à l'aide de procédures associées à ces différents opérateurs.



CHAPITRE 8 : Gestion des transactions réparties

Objectif du chapitre

Le concept de transaction est essentiel pour maintenir la cohérence des données. Une transaction est une unité de mise à jour composée de plusieurs opérations successives qui doivent être toutes exécutées ou pas du tout.

Dans ce chapitre, nous allons introduire les concepts de base liés à la gestion de transactions dans les SGBDs. Ensuite, nous étudions les problèmes liés à la gestion de transactions dans les systèmes de bases de données distribuées, notamment les problèmes de concurrence. Nous terminons ce chapitre par l'explication de quelques algorithmes de gestion de la concurrence.

1. Concept de la transaction

Une transaction est un ensemble d'opérations menées sur une BD, qui la transforment depuis un état cohérent en un autre état cohérent. Soit la transaction est validée par un commit, soit annulée par un rollback, soit interrompue par un abort. Dans les deux derniers cas, c-à-d la base de données reviendra à son état initial cohérent. Les opérations peuvent être en lecture et/ou écriture. Elles sont atomiques, c'est donc une unité indivisible de traitement,

Chaque transaction possède une marque de début (Begin Of Transaction - BOT), et une marque de fin (End Of Transaction -EOT).

L'**ACID**ité d'une transaction, c-à-d, la cohérence et la fiabilité d'une transaction sont garanties par les quatre propriétés suivantes: **A**tomicité, **C**ohérence, **I**solation, et **D**urabilité.

- ✓ **Atomicité** : L'atomicité d'une transaction veut dire qu'elle est traitée comme une seule opération, et toutes les actions sont exécutées à bien ou aucune d'entre elles.
- ✓ **Cohérence** : La cohérence veut dire qu'une transaction mène la base de données d'un état cohérent à un autre état cohérent de façon que toutes les contraintes d'intégrité restent vérifiées.
- ✓ **Isolation** : L'isolation impose à chaque transaction de voir la base de données cohérente. Si une transaction est en cours d'exécution, elle ne peut révéler ses résultats à d'autres transactions concurrentes qu'une fois le commit est effectué.

✓ **Durabilité** : la durabilité est garantie lorsqu'une transaction a effectué son commit, le résultat sera permanent et ne pourra être effacé de la base de données quelques soient les pannes du système rencontrées.

Exemple

Dans cet exemple, nous décrivons le cas de transfert d'argent d'un compte épargne vers un compte courant d'un client d'une banque.

Cette transaction comporte des instructions qui retirent de l'argent 'solde' dans le compte d'épargne du client et des instructions qui rajoutent la somme retirée vers son compte courant. La transaction commence par :

Begin Transaction Transfert_C_Epargne/C_Courant

Le contenu de la transaction

```
Begin
        Input(somme, numClt)
        EXEC SQL UPDATE ComptesEpargne
                        SET solde = solde – somme
                        WHERE numClient = numClt ;
        EXEC SQL UPDATE ComptesCourant
                        SET solde = solde + somme
                        WHERE numClient = numClt ;
        Output ("Le transfert de la somme de votre compte épargne vers votre compte courant est
                effectué")
End
```

2. Problèmes des transactions réparties

Garantir une transaction répartie revient à assurer les quatre propriétés d'acidités suivantes en considérant l'aspect de la répartition:

• **Atomicité** : Dans une base de données répartie, les mises à jour doivent être effectuées sur tous les sites, ou sur aucun site, d'où la nécessité de coordonner entre les sites.

• **Isolation** : pour garantir l'isolation plusieurs problèmes doivent être résolus :

Le problème des items locaux (physique) exécutés dans chaque site et les items globaux (logique). Le problème de réplication de données où plusieurs données (items physiques) représentent une même donnée (l'item logique). Le problème de transaction globale et transaction locale, c-à-d le système doit garantir la sérialisabilité globale. Enfin le problème d'inter-blocage intersites qui est difficile à détecter et à prévention.

3. Contrôle de concurrence

Dans les bases de données, plusieurs utilisateurs tentent d'accéder simultanément à la BD. Les mécanismes d'accès concurrent permettent de partager les ressources et d'améliorer les performances d'accès aux données. Le contrôle de concurrence est un mécanisme propre à SGBD, qui permet de contrôler l'exécution simultanée de transactions de manière à produire les mêmes résultats qu'une exécution séquentielle permettant ainsi d'assurer la propriété de **sérialisabilité**. En effet, une exécution d'un ensemble de transactions est dite sérialisable si elle donne pour chaque transaction participante, le même résultat que l'exécution en séquentiel de ces mêmes transactions. Pour garantir la propriété de sérialisabilité, plusieurs mécanismes sont utilisés *:*

3.1 Les estampilles

La technique des estampilles permet d'installer un ordre total entre les actions qui s'exécutent dans une base de données et ainsi de les sérialiser. Ainsi, une estampille est un numéro unique attribué à une transaction permettant de l'ordonner strictement par rapport aux autres transactions. En général, l'estampille utilisée à une transaction est son horodate de lancement concaténée avec le numéro du site sur lequel elle est lancée. Alors, l'estampille est le couple (horloge locale, numéro du site). L'inconvénient majeur des estampilles est la difficulté de synchroniser des sites de différentes horloges.

3.2 Verrouillage

L'une des techniques la plus répandue pour sérialiser les transactions est basée sur l'utilisation de verrous. Cette technique impose que l'accès aux données se fasse de manière mutuellement exclusive.

Remarque : Toute méthode de sérialisabilité basée sur le verrouillage peut donner lieu à des inter-blocages lorsque deux transactions s'entre-attendent. Pour illustrer ce cas, on peut prendre un exemple d'une transaction (T_1) qui détient un verrou en lecture ou en écriture sur la donnée (x), et une transaction (T_2) qui détient un verrou en lecture ou en écriture sur la donnée (y). La transaction (T_1) attend un verrou en

écriture sur la donnée (y) et la transaction (T_2) attend un verrou en écriture sur la donnée (x). Il y a dans ce cas un inter-blocage.

Notant aussi qu'un outil permettant l'analyse des inter-blocages est le graphe des attentes. Les nœuds du graphe des attentes sont des transactions simultanées et les arcs orientés qui les relient représentent l'attente d'une transaction pour une autre. Grâce à cette représentation de graphe, il est facile de localiser les inter-blocages puisqu'il apparaît comme des cycles sur le graphe. Dans un environnement réparti, les inter-blocages impliquent différents sites, ainsi seul le graphe des attentes local est insuffisant pour illustrer les inter-blocages, il faut également un graphe au niveau global.

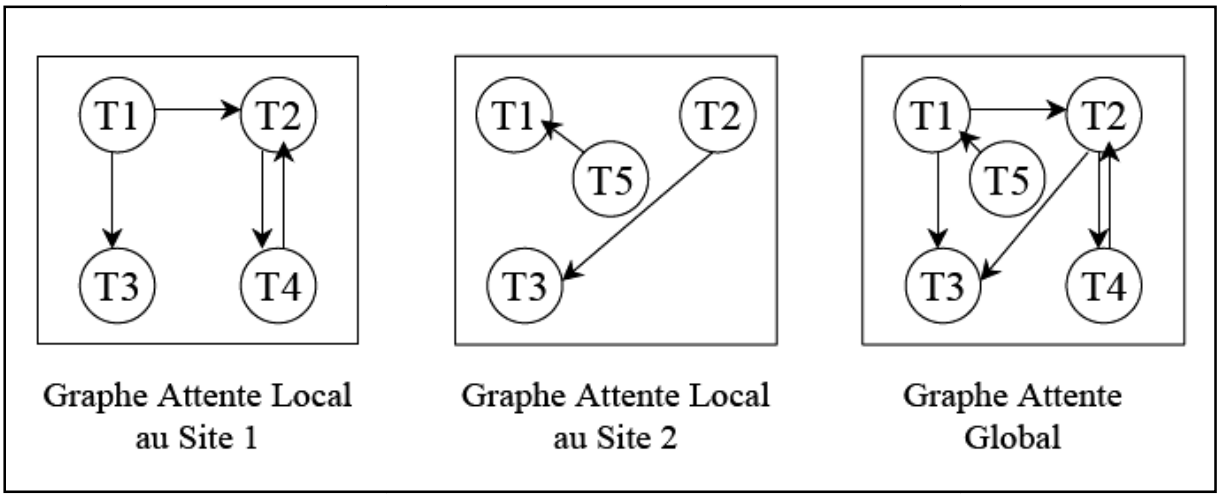


Figure 16. Utilisation des graphes pour l'analyse des inter-blocages

Pour résoudre le problème des inter-blocages, la solution consiste à annuler des transactions concurrentes jusqu'à la suppression de cycles et de les reprendre plus tard.

3.4 Protocole de validation à deux phases

Dans les systèmes répartis, un ensemble de processus collabore dans une transaction. Afin de coordonner la décision de valider une transaction, un protocole de validation en deux étapes est généralement utilisé. Ce protocole a été proposé par Lampson en 1976. Le protocole exige un site coordinateur pour contrôler et des sites participants, et il est exécuté par chaque site. Il assure l'atomicité (sauf dans le cas de la destruction du journal log). Dans ce protocole, une transaction globale est validée si et seulement si toutes les transactions locales qui la composent sont valides aussi. En

effet, si au moins une transaction locale ne valide pas, toutes les transactions sont annulées. Il se compose de deux phases :

- Phase de préparation : dans cette phase, le coordinateur demande à chaque participant de se préparer pour valider la transaction locale.
- Phase de validation : dans cette phase, le coordinateur ordonne à tous les participants de valider ou d'annuler leur transaction. La décision est prise par le coordinateur en tenant compte de la réponse de chaque participant.

Les transactions distribuées présentent deux avantages : (i) les données situées sur d'autres serveurs, peuvent être mises à jour et les instructions UPDATE peuvent être gérées comme étant une seule unité. (ii) utilisation du 2-PC transparente à l'utilisateur.

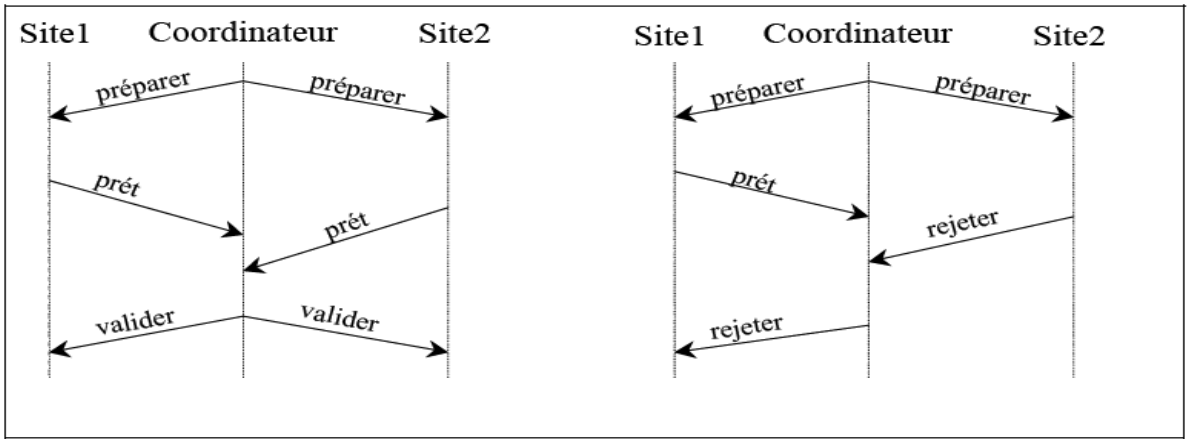


Figure 17. Exemples du protocole de validation en deux étapes

Chapitre 9 : La sécurité des données C/D

Objectif du chapitre

La sécurité est un aspect clé de tout déploiement de base de données. Il est essentiel d'authentifier les utilisateurs de la base de données, et surtout de permettre uniquement aux utilisateurs autorisés d'accéder aux informations et de maintenir l'intégrité globale des données du système. Dans ce chapitre, nous donnons un aperçu sur les différents concepts de la sécurité des bases de données. Nous commençons par les menaces et les failles de sécurité. Ensuite, nous examinerons les piliers de la sécurité des SGBDs. Enfin, nous détaillons **l**es principaux mécanismes de mise en œuvre de la sécurité dans un SGBD**.**

1. Les menaces et les failles de sécurité

La sécurité de données est aspect très important. En effet, les menaces sont nombreuses, en plus des attaques accidentelles ou les défaillances du système, les mauvais comportements des gestionnaires et attaques 'malveillantes' sont aussi plus dangereuses aux données. Ci-après quelques types de plus importantes menaces:

Les virus informatiques : sont des programmes informatiques généralement capables de se reproduire (se recopier), et des fois se déplacer à travers un réseau en utilisant les mécanismes réseau et d'endommager les données informatiques. Ils visent à paralyser, ou détruire tout ou partie du système d'information.

Les configurations et installation 'par défaut' : la configuration et l'installation des SGBD tout en laissant les valeurs de paramètres par défaut risque de mettre les données en danger dans la mesure ou les communications par défaut ne sont pas chiffrées (ftp, telnet, pop) et les valeurs de paramètres par défaut sont connues, par exemple port : 80 / 1525, mot de passe par défaut : SYS et SYSTEM, les services superflus sont accessibles tels que SRY FTP, SRY SAMBA, SNMP,

La mauvaise politique de gestion des droits : l'administrateur de l'SGBD qui ne maitrise pas la gestion des droits de son SGBD et de ses bases de données pour pouvoir donner les privilèges aux bons utilisateurs ayant le droit d'accès à ces données. De plus, l'administrateur doit éviter une utilisation abusive de l'héritage de donner le droit d'accès à tout le monde ou de ne le donner à personne. Par exemple,

l'administrateur n'assure pas une installation confortable où tous les SGBD et logiciels sont installés sous root et tous les utilisateurs des applications ont tous les privilèges.

Mauvaise gestion : la mauvaise gestion du matériel informatique qui entraînera des bugs et qui peuvent provoquer un déni de service, voir l'arrêt total du système. Notant aussi que le mauvais codage des paramètres en clair dans les URLs et les connexions non chiffrées sont des sources de menaces.

2. Les piliers de la sécurité des SGBDs

La sécurité des données ne se résume pas dans la confidentialité de ces données, mais plutôt plusieurs autres concepts sous-tendent la sécurité. Ils sont pratiquement tous importants aux ***SGBDs***.

La confidentialité: toutes les données ne sont pas accessibles à tous les utilisateurs. Dans les bases de données, la confidentialité est le fait de donner un certain nombre de droits et de ressources en fonction d'un profil défini est le rôle de l'administrateur de la base. La granularité d'accès peut aller jusqu'à la vision unique d'un champ d'un enregistrement d'une table particulière.

La disponibilité : la disponibilité consiste en la capacité de livrer correctement un service ou des données en termes de délai et de qualité à l'utilisateur. On peut mesure et évaluer la disponibilité en pourcentage du temps de fonctionnement total. Une disponibilité de 100% veut dire que le temps de reprise nul.

La fiabilité : pour assurer la fiabilité, plusieurs mécanismes de sauvegarde physique, logique, offline, online, totale, partielle, ainsi que des mécanismes de journalisation et de reprise permettent de restaurer des informations sans aucune perte. Dans tous les cas de problème matériel ou logiciel.

L'intégrité : veut dire que les données doivent être cohérentes c-à-d d'une part que les accès concurrents d'utilisateurs, notamment lors de mises à jour, ne doivent pas compromettre la cohérence des données et d'autre part que ces dernières satisfassent aux contraintes d'intégrité du modèle et aux règles de gestion de l'entreprise.

La traçabilité : la traçabilité veut dire qu'en cas de problème important ou d'attaque du système, on peut recourir à l'analyse de traces ou de logs. Le niveau de détail de

ces traces est paramétrable et concerne soit les aspects système, le réseau, ou l'accès aux données élémentaires elles-mêmes.

La maintenabilité : La maintenabilité veut dire que l'aptitude à la réparation, évolution, maintenance du SGBD. La maintenabilité peut être mesurée et évaluée en temps de reprise après panne.

3. Les mécanismes de mise en œuvre de la sécurité

Les SGBD centralisés et distribués doivent fournir un certain nombre de mécanismes internes ou de fonctionnalités assurant un niveau satisfaisant de sécurité. Les fonctionnalités qu'ils doivent assurer sont :

L'authentification : c'est le processus qui contrôle l'identité de l'utilisateur c-à-d le processus qui permet de vérifier qu'un utilisateur demandant un accès est bien celui qu'il prétend être. Cette action se fait en général via la fourniture du couple nom d'utilisateur (login) et mot de passe. Dans certains cas l'authentification peut être implicite et héritée d'une authentification précédente, ou reconnue automatiquement (@IP du user sur le réseau par exemple), bien que la simplification des accès aux données est un choix qui peut s'avérer dangereux.

Les droits et privilèges : en effet, l'utilisateur doit pouvoir accéder aux informations et aux ressources auxquelles il a droit, une fois correctement identifié (et uniquement à celle-la et non pas aux autres données). Ce problème est résolu simplement avec la gestion de droits élémentaires accordés à un individu, ou plus efficacement avec des rôles et profils affectés à des groupes d'individus.

Les LOGs ou traces : permet d'enregistrer tout ou une partie des informations concernant les accès. Cette trace pourra être atténuée et son volume étroitement surveillé. Ainsi on l'utilisera seulement de manière ciblée sur des périodes de temps spécifiques.

La tolérance aux pannes : la tolérance aux pannes permet aux SGBDs ou aux logiciels de supporter de manière partiellement ou complètement transparentes les différents types de pannes, au niveau du client, du réseau, ou du serveur. Une tolérance totale a bien sûr un coût de réalisation.

La sauvegarde et restauration: la sauvegarde c'est la possibilité de stocker des données sur des supports externes tels que les disques et de pouvoir les restituer,

avec les mises à jour les plus récentes. Le but de cette opération est de ne pas perdre de données suite à un problème matériel (panne disque), logiciel (bug) ou une fausse manipulation d'un utilisateur.

Les mécanismes transactionnels : par définition l'atomicité des transactions assure la cohérence des données, dans des environnements centralisés et distribués. L'image avant modification, stockée de manière redondante dans ou hors de la base de données permet de faire d'éventuels retours en arrière pour retrouver le dernier état cohérent, ou de s'assurer qu'il n'y pas eu d'opérations partielles ou incomplète.

Bibliographe

[1] TamerOzsu, Patrick Valduriez. Principles of Distributed Database Systems. 3ème Edition, Springer Science & Business Media, 2011. ISBN 978-1-4419-8833-1

[2] Georges Gardarin. Base de données, 5ème Édition, Ed. Eyrolles, 2003. ISBN-10: 221211281.

[3] Saeed K. Rahimi, Frank S. Haug. Distributed Database Management Systems: A Practical Approach, 1ère Edition. Wiley-IEEE Computer Society Pr. 2010. ISBN-10: 047040745X

[4] Ullman, J. D., Wildom, J. A First Course in Database Systems., 3rd edition Ed. Pearson, 2008. ISBN-10**:** 9780136006374

[5] Raghu Ramakrishnan, Johannes Gehrke. Database Management Systems, 3ème Edition, McGraw-Hill Higher Education. 2002. ISBN-10: 0072465638

[6] Elmasri R. & Navathe S.B. Fundamentals of database systems. 7ème Édition, Ed. Pearson, 2015. ISBN-10: 0133970779

[7] Didier Deleglise, la Bible de l'administrateur et du développeur Oracle, Independently Published, 2018. ISBN: 1717833039

[8] Boudjlida Nacer. Gestion et Administration des Bases de Donnees: Application a Sybase et Oracle. Dunod, Paris, 2003. ISBN10 : 2100058479

[9] XML précis et concis, Simon St. Laurent et Michael Fitzgerald, 0'Reilly, 2005. ISBN-10: 2841771040

[10] Anne Doucet, Modèles et Langages pour les Bases de Données Avancés, support de cours, Master Informatique Spécialité IAD, http://www-poleia.lip6.fr/~doucet/MABD/MABD5-semistructure.pdf

[11] Souris, M. Les principes des systèmes d'information géographique –Principes, algorithmes et architecture du système Savane. 2002

[12] Rim Moussa, Systèmes de Gestion de Bases de Données Réparties & Mécanismes de Répartition avec Oracle. Support de cours, l'ISSATM –Université du 7 Nov. de Carthage. 2006/2017.

[13] Hervé Jégou & Matthijs Douze. Bases de données multimédia. Support de cours, Ecole d'ingénieur ENSIMAG. 2014/2015.

[14] Esteban Z. Spatial Databases, Support de cours, Department of Computer & Decision Engineering (CoDE). Bruxelles University. 2012/2013

[15] Emmanuel Desmontils, Annie Tartier, Ingénierie XML : concepts de base, support de cours, Master MIAGE, de Master CCI de Université de Nantes. 2017.

[16] Anne Doucet. Modèles de données avancés, Support de cours du module BDIA, Master d'Informatique Spécialité IAD.

[1] Nassim Dennouni, Polycopié du cours: Base de données avancées. Master 1 Spécialités : IL & ISIA Université Hassiba Benbouali de Chlef. 2015/2016