République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université d'IBN KHALDOUN de TIARET
Faculté des Mathématiques et de l'Informatique
Département d'Informatique

Polycopié de cours

Bases de Données Avancées

Pour 1ère Année Master
des spécialités Génie Informatique
et Génie Logiciel

Réalisé par
Dr MERATI Medjeded

Expertisé par

BENYAMINA Abou el Hassan – Professeur – Université Oran 1, Ahmed Ben bella
BELARBI Mustapha – MCA – Université Ibn Khaldoun, Tiaret

Année universitaire
2019/2020

Table des matières

Figures 8
Tableaux 9
Introduction 10

C H A P I T R E 1

Système d'Information

1.1. Définition d'un SI 13
1.2. Composantes d'un SI 14
1.3.1. Sous-systèmes d'un SI 15
1.3.1. Sous-système opérationnel 15
1.3.2. Sous-système décisionnel 15
1.3.3. Sous-système d'information (ou de communication) 15
1.4. Processus de développement d'un système d'information 16
1.4.1. Cycle de développement 16
1.4.2. Cycle de vie 17
1.4.3. Les modèles de cycle de vie 17
1.4.3.1. Modèles linéaires 17
1.4.3.2. Modèles itératifs 18
1.4.3.3. Modèles agiles 19
1.5. Notion de base de données (BD) 21
1.5.1. Définition d'une BD 21
1.5.2. Fonctions d'une base de données 21
1.6. Système de Gestion de base de données (SGBD) 22
1.6.1. Définition d'un SGBD 22
1.6.2. Niveaux d'abstraction 23
1.6.2.1. Niveau conceptuel 23
1.6.2.2. Niveau interne 24
1.6.2.3. Niveau externe 24
1.6.3. Historique des SGBDs 25
1.6.4. Avantages des SGBDs 26
1.7. En résumé 28
1.8. Exercices 28
Exercice 1 : 28

Exercice 2 : 28
1.9. Références du chapitre 29

C H A P I T R E 2

Bases de données relationnelles

2.1. Modèle relationnel : description des données 31
2.2. Normalisation par décomposition 32
2.2.1. Contraintes d'intégrité 32
2.2.1.1. Définition de la clé de relation 33
2.2.1.2. Définition de la dépendance fonctionnelle 33
2.2.2. La décomposition d'une relation 34
2.2.3. Formes normales 34
2.2.3.1. Première forme normale 35
2.2.3.2. Deuxième forme normale 35
2.2.3.3. Troisième forme normale 35
2.2.3.4. Forme normale de Boyce and Codd 36
2.3. Langage de requêtes SQL 37
2.3.1. Data Definition Langage (DLL) 37
2.3.1.1. Création d'un schéma 37
2.3.1.2. Création d'une table 37
2.3.1.3. Les clés de relation 39
2.3.1.4. Opérations sur les tables 39
2.3.2. Data Manipulation Langage (DML) 41
2.3.2.1. Requêtes simples 41
2.3.2.2. Données dérivées 42
2.3.2.3. Fonctions statistiques 43
2.3.2.4. Condition par sous requêtes 44
2.3.2.5. Extraction de données de plusieurs tables 45
2.3.2.6. Extraction de données groupées 45
2.3.2.7. Modification des données 46
2.3.2.8. Mise à jour et contraintes référentielles 47
2.4. En résumé 48
2.5. Exercices 48
Exercice1 : 48
Exercice2 : 49

Exercice3 : 49
Exercice4 : 49
2.6. Références du chapitre 50

C H A P I T R E 3

Base de données objet

3.1. Historique des bases de données objet 52
3.2. Concepts fondamentaux 53
3.2.1. Objet 53
3.2.2. Classe 53
3.2.3. Objets complexes 54
3.2.4. Identité de l'objet 54
3.2.5. Encapsulation 54
3.2.6. Extensibilité 54
3.2.7. Classes hiérarchiques et Héritage 54
3.2.8. Redéfinition et surcharge 55
3.3. Base de données Orientées Objet (OODB) 55
3.3.1. Langage de définition objet ODL 55
3.3.1.1. Les classes 55
3.3.1.2. Les attributs et les contraintes 56
3.3.2. Langage de requête objet OQL 59
3.3.2.1. Les résultats d'une requête OQL 59
3.3.2.2. Utilisation des méthodes dans les requêtes 60
3.3.2.3. Les requêtes embarquées 60
3.3.2.4. La quantification existentielle et universelle 61
3.3.2.5. Ordonner le résultat 61
3.3.2.6. Collection 62
3.3.2.7. Agrégation et groupement 62
3.4. Bases de données Relationnelles-Objet (BDRO) 63
3.4.1. Les types de données intégrés 63
3.4.1.1. Le type ligne 63
3.4.1.2. Les types collections 64
3.4.2. Types de données définis par l'utilisateur (UDTs) 65
3.4.2.1. Les types distincts 65
3.4.2.2. Les types structurés 66

3.4.2.3. Méthode définie par l'utilisateur 66
3.4.3. Types, tables et hiérarchie 66
3.4.4. Relation 68
3.5. En résumé 68
3.6. Exercices 69
Exercice 1: 69
Exercice 2: 69
Exercice 3: 70
3.7. Références du chapitre 71

C H A P I T R E 4

Base de Données Distribuées

4.1. Motivation 72
4.2. Définition d'une BDD 72
4.3. Définition d'un SGBDD 73
4.4. Avantages des SGBDD 73
4.4.1. Indépendance des données 73
4.4.2. Transparence de la distribution 73
4.4.3. Transparence de la fragmentation 73
4.4.4. Fiabilité garantie par les transactions distribuées 73
4.4.5. Performance améliorée 73
4.5. Architectures des BDDs 74
4.5.1. Modèle Client/Server 74
4.5.2. Modèle Peer to Peer (P2P) 74
4.5.3. Modèle Multi-BDs 75
4.6. Conception d'une BDD 75
4.6.1. Conception descendante 75
4.6.1.1. Fragmentation 76
4.6.1.1.1. Fragmentation horizontale 77
a) Fragmentation horizontale primaire 78
b) Fragmentation horizontale dérivée 78
4.6.1.1.2. Fragmentation verticale 79
4.6.1.1.3. Fragmentation hybride 80
4.6.1.2. Allocation 81
4.6.2. Conception ascendante 82

4.7. Réplication 82
4.7.1. Objectifs de la réplication 82
4.7.2. Types de réplication 83
4.8. Traitement et optimisation des requêtes 83
4.8.1. Complexité et mesures 83
4.8.2. Décomposition des requêtes et localisation des données 84
4.8.3. Optimisation des requêtes 85
4.9. Gestion des transactions distribuées 86
4.9.1. Propriétés ACID 86
4.9.2. Contrôle de concurrence 86
4.9.2.1. Approche d'isolation multiversion 87
4.9.2.2. Approche à verrouillage 87
4.10. En résumé 88
4.11. Exercices 88
Exercice 1 : 88
Exercice 2 : 89
4.12. Références du chapitre 90

CHAPITRE 5

Data Warehouse

5.1. Motivation et objectifs des DWs 92
5.2. Définition d'un Data Warehouse 94
5.3. L'approche Data Warehouse 94
5.4. Architecture des DWs 97
5.4.1. Le niveau préparation des données 98
a) Extraction 98
b) Transformation 98
c) Loading (chargement) 98
5.4.2. Le niveau Data Warehouse 98
5.4.3. Le niveau OLAP 99
5.4.4. Le niveau frontal 99
5.5. Data Warehouse vs Datamart 99
5.6. Modèle multidimensionnel 100
5.7.1. Cubes 101
5.7.2. Dimensions 102

5.7.3. Faits 102
5.7.4. Mesures 103
5.7.5. Opérations OLAP 103
5.8. ROLAP, MOLAP et HOLAP 105
5.9. En résumé 105
5.10. Exercices 105
Exercice 1 : 105
Exercice 2 : 106
Exercice 3 : 106
5.11. Références du chapitre 107
Conclusion 108

Figures

Figure 1 : Composantes d'un système d'information (Rivard 2001) 14
Figure 2: Interaction entre les sous-systèmes d'une organisation 16
Figure 3 : Modèle en cascade 17
Figure 4 : Modèle en V 18
Figure 5 : Modèle spirale (Boehm 2000) 19
Figure 6 : Modèle incrémental 19
Figure 7 : La boucle de planification et de feedback dans XP 20
Figure 8: Niveaux d'abstraction d'un SGBD 23
Figure 9: Exemple de schéma conceptuel 24
Figure 10: Exemples de schémas externes 25
Figure 11: Les formes normales (Meier 2006) 34
Figure 12: La relation LIVRE 36
Figure 13 : Diagramme UML d'un exemple de gestion des films publicitaires 57
Figure 14: Démarche de la conception descendante d'une BDD 76
Figure 15: Fragmentation horizontale primaire de la relation ETUD 78
Figure 16 : Fragmentation horizontale dérivée de la relation ETUD 79
Figure 17: Fragmentation verticale de la relation ETUD 80
Figure 18: Fragmentation hybride de la relation ETUD 81
Figure 19 : Niveaux des données d'un DW 96
Figure 20 : Les niveaux de données pour un exemple simple : un employé 97
Figure 21: Architecture typique d'un DW (Vaisman 2014) 97
Figure 22: Un cube en trois dimensions pour les données de vente avec les dimensions : 101
Figure 23: Hiérarchie de la dimension *Client* 102

Tableaux

Tableau 1: Dépendance fonctionnelle dans une relation 33
Tableau 2: La relation avion (Boudjlida 2002) 35
Tableau 4: Table des étudiants 77
Tableau 5: Table des moyennes des étudiants 79
Tableau 6 : Différences entre données primitives et données dérivées (Inmon 2005) 95
Tableau 7: Résumé des opérations OLAP (Vaisman 2014) 104

Introduction

Depuis le début de l'automatisation de la gestion d'organisation, l'évolution de la notion de base de données (BD) a influencée considérablement de façon directe dans la gestion de données et de façon indirecte dans la performance des organisations et des entreprises en particulier.

Dans le présent polycopié de cours nous traçons le parcours d'évolution de la notion de BD depuis l'ère où les premiers essais de stockage et de partage de données ont vu le jour, jusqu'à l'ère de l'exploitation des données dans des processus d'analyse dans le but d'aide à la prise de décision ou la prédiction en générale dans une organisation.

Afin de suivre cette évolution de la notion de BD de manière pédagogique, nous avons opté pour une démarche qui consiste en premier lieu à décrire le contexte d'une organisation à travers la notion de Système d'Information (SI) dans le but de montrer à quel endroit du SI, le stockage et le partage des données est nécessaire et à quel endroit du SI, les données sont par la suite exploitées dans la prédiction. La suite de la démarche consiste à discuter les différents modèles de BDs qui ont été développé durant cette évolution.

Les parties composant ce polycopié sont issues du programme du module « Base de données avancées » de la promotion Master I des deux spécialités Génie Informatique et Génie Logiciel.

Le polycopié est organisé de la manière suivante :

Le chapitre 1 commence d'une part par définir la notion de SI et ses composantes. Ensuite, nous détaillons les sous-systèmes constituant un SI où les différents modèles de BDs peuvent être utilisés dans son développement. Ainsi, les différents cycles de vie de développement d'un SI sont brièvement discutés. De l'autre part, le chapitre définit la notion de BD et ses fonctions. Ensuite, la notion de Système de Gestion des Bases de Données (SGBD) et le langage SQL sont définis et La modélisation des SGBDs en niveaux d'abstraction proposée par ANSI est détaillée. Le chapitre est finalisé par un historique résumant l'évolution du développement des SGBDs.

Le chapitre 2 lance la discussion sur les différents modèles de BD avec le fameux modèle relationnel qui a dominé pendant longtemps le domaine des BDs. Le chapitre est entamé par la description de la manière avec laquelle les données sont décrites dans le modèle relationnel. Ensuite, le concept de normalisation est discuté en focalisant sur les concepts : contraintes d'intégrité, décomposition des relations et formes normales. Le chapitre comprend aussi des définitions détaillées sur les deux modules du langage SQL à savoir le langage de définition des données (DDL) et le langage de manipulation des données (DML) appuyées par des exemples illustratifs.

Le chapitre 3 enchaîne avec le modèle des Bases de Données Objet (BDO) qui a suivi deux chemins dans son développement dans la littérature. Le premier concerne le modèle des Bases de Données Orientées Objet (BDOO) et le second concerne le modèle des Bases de Données Relationnelles-Objet (BDRO). Après la définition des principaux concepts de l'orienté objet (tels que l'objet, l'encapsulation, …), les deux langages utilisés dans les BDOOs à savoir le langage de définition objet (ODL) et le langage de requête objet (OQL) sont définis avec des exemples illustratifs. Ensuite, une section est consacrée aux BDROs où une discussion détaillée de l'adaptation du SQL pour intégrer les concepts orientés objet est également donnée avec des exemples illustratifs.

Le chapitre 4 aborde le modèle des Bases de Données Distribuées (BDDs) en définissant la notion de BDD ainsi que le Système de Gestion des Bases de Données Distribuées (SGBDD). Après l'explication des principaux avantages de l'utilisation des SGBDDs (tels que l'indépendance des données et la transparence de la distribution et de la fragmentation), L'architectures des BDDs est discutée. Ensuite, la conception des BDDs est détaillée en focalisant sur le type de conception descendante alors qu'une brève définition est donnée pour le type de conception ascendante puisqu'elle concerne le modèle Data Warehouse abordé dans le chapitre 5. Dans la conception descendante, il est question de définir la notion de fragmentation des données, y compris ses types (horizontale, verticale et hybride), ainsi que la notion d'allocation des données. Finalement, des concepts très importants pour les BDDs sont discutés en détail tels que la réplication, l'optimisation des requêtes et la gestion des transactions distribuées.

Le chapitre 5 présente le modèle Data Warehouse (DW) en expliquant les principaux objectifs des DWs. Nous expliquons par un exemple l'idée derrière l'utilisation des DWs. Puis, une description détaillée d'une architecture typique d'un DW est donnée. L'important concept du modèle multidimensionnel est

abordé ainsi que les modèles basés sur ce dernier à savoir ROLAP, MOLAP et HOLAP.

CHAPITRE 1

Système d'Information

Le traitement de l'information est un moyen de création de la valeur dans une organisation ou en particulier dans une entreprise. Plus que ça, une bonne exploitation de l'information et sa mise en disponibilité contribuent pleinement dans l'atteinte des objectifs de l'entreprise. A partir des années 70, les systèmes d'information ont pu rendre les bases de données en véritables gisements d'informations. Les bases de données ont été utilisées à des fins décisionnelles à partir des années 80 grâce aux travaux de Ralph Kimball. En 1990, le concept d'entrepôt de données ou « Data Warehouses » a vu le jour grâce aux travaux de William H. Inmon.

Dans ce chapitre nous présentons l'essentiel des concepts relatifs aux systèmes d'informations (SIs) à savoir : la définition du SI, ses composantes ainsi que les différents cycles de vie des SIs. Les notions de Base de Données (BD) et de Système de Gestion des Bases de Données (SGBD) seront étudiées en particulier.

1.1. Définition d'un SI

Le système d'information (SI) est un ensemble organisé de ressources qui permet de collecter, stocker, traiter et distribuer de l'information (De Courcy 1992).

Les ressources peuvent être humaines, matérielles, logicielles (i.e. programmes et procédures de traitement), de données (i.e. bases de données) et de réseaux (i.e. voies de communications).

Les fonctions d'un SI peuvent être détaillées comme suit :

Collecter : il s'agit d'acquérir les informations à partir de l'environnement interne ou externe.

Stocker : il s'agit de conserver l'information acquise et de la rendre disponible.

Traiter : il s'agit de transformer l'information en modifiant le fond ou la forme et la rendre adaptée au traitement.

Distribuer : il s'agit de diffuser l'information dans l'environnement interne ou externe du système.

1.2. Composantes d'un SI

Comme illustré dans la figure 1, tout système d'information est constitué de quatre types de composantes : les inputs, les traitements, les dépôts de données et les outputs.

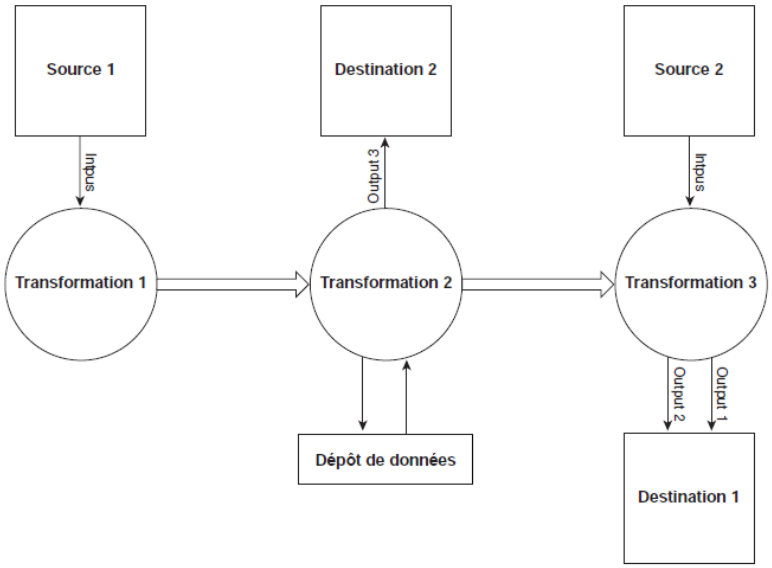


Figure 1 : Composantes d'un système d'information (Rivard 2001)

Des inputs (données) sont émis par une ou plusieurs sources et traités par le système, lequel utilise aussi des données entreposées préalablement. Les résultats du traitement (outputs) sont transmis à une ou plusieurs destinations ou mettent à jour des données entreposées (Rivard 2001).

1.3.1. Sous-systèmes d'un SI

Dans toute organisation, le périmètre d'un système d'information couvre trois sous-systèmes qui interagissent entre eux : le sous-système opérationnel, le sous-système décisionnel et le sous-système d'information.

Le sous-système d'information se positionne au cœur de l'organisation en assurant la communication des informations collectées et modifiées du système opérationnel au système décisionnel qui se charge de contrôler et prendre des décisions (Figure 2).

1.3.1. Sous-système opérationnel

Les systèmes opérationnels également appelés OLTP (On-Line Transactional Processing) regroupe les tâches de base et répétitives (insertion, modification et suppression) réalisées quotidiennement par les membres (ex. employés) de l'organisation (ex. entreprise). Certaines entreprises, assurent la gestion de ses métiers par des Progiciels de Gestion Intégrée (PGIs) ou Entreprise Ressource Planning (ERPs) qui intègrent les données et les processus d'une entreprise (finance, ressources humaines, ventes, stock,…) dans un même système. Les ERPs utilisent une base de données unique qu'est accessible par tous les modules du système.

1.3.2. Sous-système décisionnel

Les systèmes décisionnels également appelés OLAP (On-Line Analytical Processing) se focalisent sur le management de l'organisation (ex. entreprise) dans le sens d'aider les décideurs dans leur pilotage en leur offrant une vision transversale sur l'organisation. Certaines entreprises mettent en place des entrepôts de données (Data Warehouses) comme système décisionnel.

1.3.3. Sous-système d'information (ou de communication)

Ce système se charge de collecter, stocker, transformer et diffuser des informations dans les sous-systèmes opérationnel et décisionnel. Il assure donc l'échange d'informations avec les deux autres sous-systèmes.

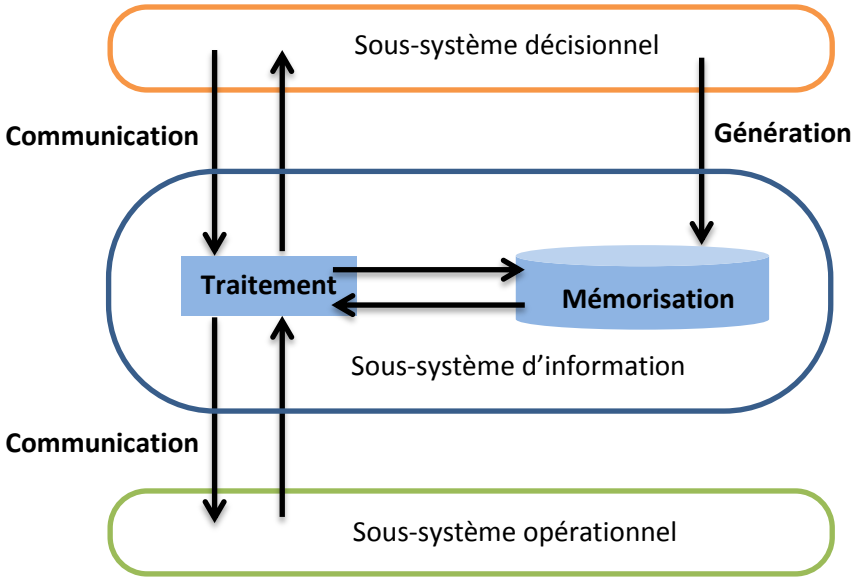


Figure 2: Interaction entre les sous-systèmes d'une organisation

1.4. Processus de développement d'un système d'information

1.4.1. Cycle de développement

Un cycle de développement comprend généralement les phases suivantes :

- **La définition des objectifs :**
  C'est à ce niveau, que la décision de fabriquer, de développer ou d'acheter un nouveau produit (SI) est prise.
- **Analyse des besoins et faisabilité :**
  La formalisation détaillée des besoins à satisfaire et les contraintes auxquelles le produit doit obéir
- **Planification et gestion des projets :**
  Planification des tâches du projet dans le temps ainsi qu'une estimation des coûts.
- **La conception globale :**
  La définition de l'architecture du logiciel ainsi que l'interface entre ses différents modules.
- **Codage et test unitaire :**
  Codage de chaque module du produit et son test indépendamment des autres (test unitaire).
- **Intégration :**
  Chaque module est intégré avec les autres modules du produit et testé.

- **Qualification :**
  Le test du produit est réalisé en vrai grandeur dans les conditions normales d'utilisation.

- **Maintenance :**
  La maintenance accompagne le produit jusqu'à son retrait en exploitant la documentation faite pendant les phases précédente.

1.4.2. Cycle de vie

Afin de maîtriser les risques, les délais, les coûts et offrir un produit de qualité conforme aux besoins, un cycle de vie est établit en décrivant les phases allant de la création à la disparition du produit y compris son distribution sur le marché. Le cycle de développement s'insère donc dans le cycle de vie.

1.4.3. Les modèles de cycle de vie

On peut regrouper les modèles de cycle de vie en trois catégories : les modèles linéaires, les modèles itératifs et les modèles agiles (Suki 2018).

1.4.3.1. Modèles linéaires

Parmi les modèles linéaires nous citons les suivants :

- **Modèle en cascade :** Introduit en 1970, ce modèle est le plus ancien. Puisqu'il est séquentiel (Figure 3), on ne passe à une étape qu'après l'achèvement de l'étape qui lui précède et les conséquences en amont du cycle ont un impact important sur les coûts en aval (Royce 1970).

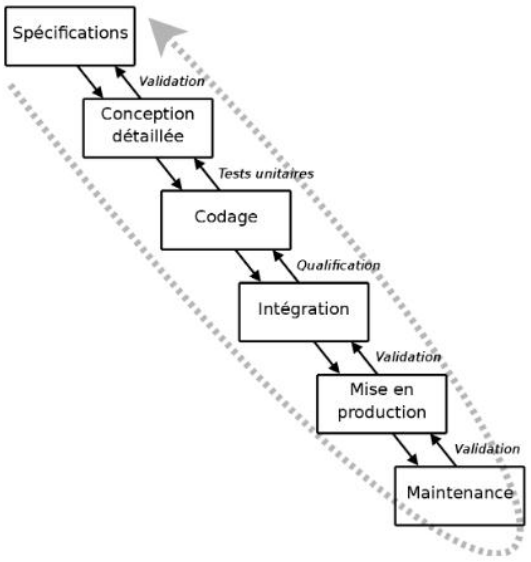


Figure 3 : Modèle en cascade

- **Modèle en V :** Comme dans le modèle en cascade, dans le modèle en V qui a été introduit par Goldberg en 1986, tout changement opéré dans une étape a un impact important sur les étapes suivantes (Figure 4). Néanmoins, le modèle en V est une amélioration du modèle en cascade dans le sens où il permet en cas d'anomalie de limiter un retour aux étapes précédentes. La conformité du produit aux spécifications doit être respectée dès les phases de conception (Firesmith 2013).

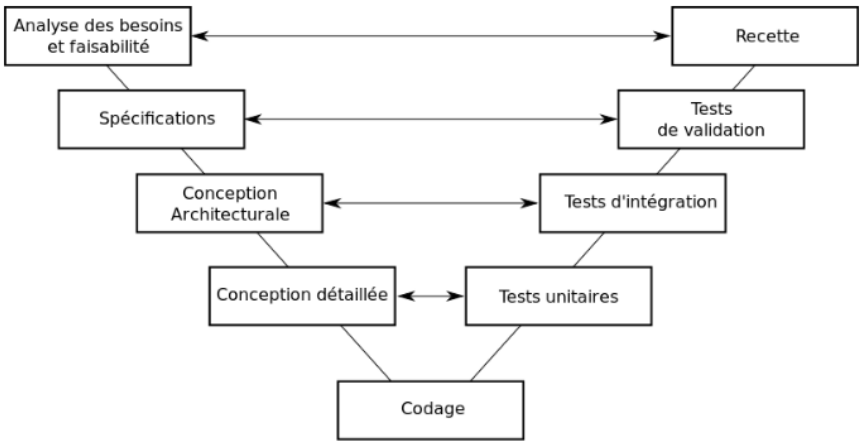


Figure 4 : Modèle en V

1.4.3.2. Modèles itératifs

Le modèle spirale et le modèle incrémental sont des exemples des modèles itératifs

- **Modèle spirale :** Défini par Barry Boehm en 1988, le modèle spirale reprend les étapes du modèle en V tout en mettent l'accent sur la gestion des risques **(**Boehm 1988). Le modèle spirale peut être déroulé sur quatre phases à savoir : (i) détermination des objectifs, des alternatives et des contraintes, (ii) analyse des risques, (iii) développement et vérification de la solution retenue, et (iv) revue des résultats et vérification du cycle suivant (Figure5).

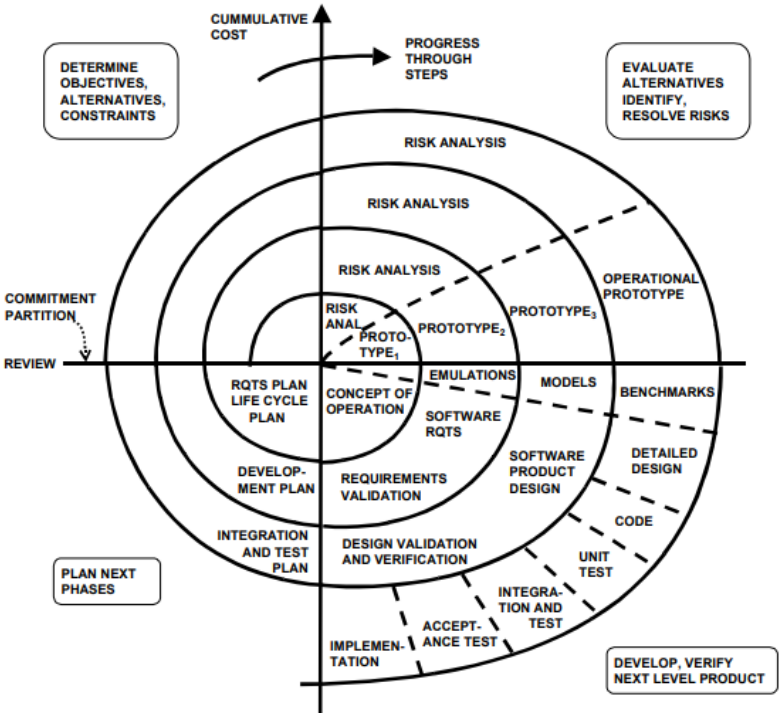


Figure 5 : Modèle spirale (Boehm 2000)

- **Modèle incrémental :**

  Dans le modèle incrémental, les modules (fonctionnalités) du produit une fois développés, sont greffés à un noyau du logiciel. Cette manière de faire du modèle incrémental (Figure 6) est adaptée aux cas où les fonctionnalités ou la plupart d'entre elles sont connues au début d'un projet de longue durée (Suki 2018).

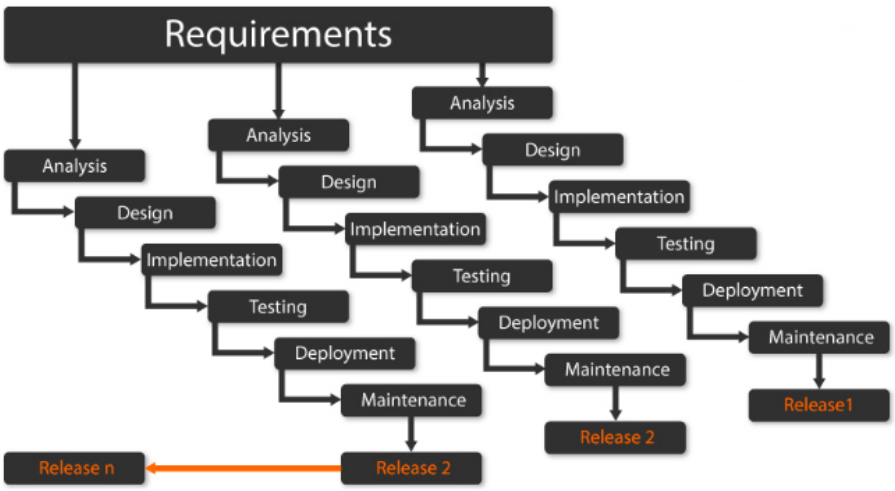


Figure 6 : Modèle incrémental

1.4.3.3. Modèles agiles

En réalité, c'est des modèles basés sur les modèles itératifs mais ils se démarquent par rapport à ces derniers par le fait qu'ils impliquent au maximum

le client où des versions fonctionnelles du produit sont régulièrement livrées et des feedbacks sur ces versions sont recueillis du client jusqu'à ce que ce dernier décide qu'il est satisfait de la version de test livrée. Ils sont donc gérés d'une manière adaptative, incrémentale et itérative.

Nous citons parmi les modèles agiles les plus populaires : XP, Scrum et FDD.

- **eXtreme Programming (XP) :**
  Ken Beck, de la société Chrysler, a conçu le modèle XP dans le but d'assurer la collaboration étroite de tous les acteurs du projet en optant pour des itérations de développement très courtes (Figure 7). Beck a défini quatre activités principales pour XP: codage, test, écoute et conception (Beck 2001).

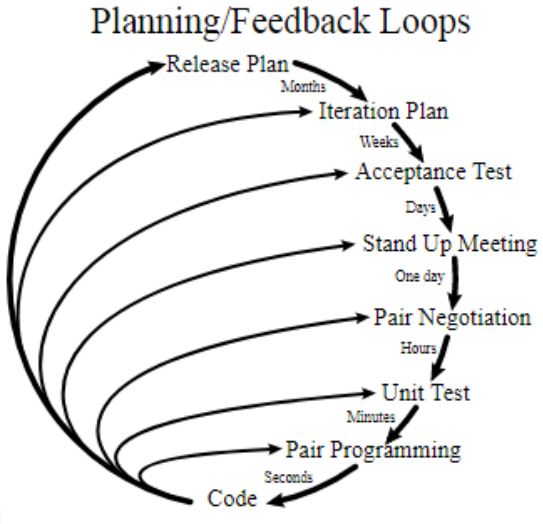


Figure 7 : La boucle de planification et de feedback dans XP

- **Scrum:**
  Le mot Scrum est repris du monde du rugby qui veut dire que le projet peut modifier sa direction au fur et à mesure de son avancement. Si le projet arrive à un stade où sa réussite ne soit pas évidente. Il est alors permet de réorienter le projet pour repartir sur de meilleurs bases. La méthode est bien adaptée au cas où le client n'a pas encore défini toutes les fonctionnalités dont il a besoin. Proposée en 1996 par Ken Scwaber, Scrum n'a pas été défini par ce dernier comme une méthode au sens strict du mot mais plutôt comme étant un cadre de travail fondée sur le changement, les résultats, la transparence et la communication, le respect des utilisateurs et l'esprit de l'équipe (Collignon 2016).

- **Feature Driven Development (FDD) :**
  Introduit pour la première fois en 1999, FDD pour Développement Dirigée par les Fonctionnalités est une méthode de gestion de projet basée sur les itérations courtes autour de fonctionnalités (Coad 1999).
  La FDD procède par les étapes suivantes : (i) la définition d'un modèle générale du produit qui détermine le périmètre globale de la réalisation, (ii) détermination de la liste complète des fonctionnalités à réaliser, (iii) conception technique des fonctionnalités (iv) et réalisation.

1.5. Notion de base de données (BD)

Les systèmes de gestion des fichiers (SGF) ont montré leurs limites dans la gestion des données notamment quand ces dernières deviennent gigantesques. En effet, de telles représentations souffrent, d'un côté de la redondance des données qui souvent devienne plus importante notamment lorsque les applications ne sont pas réalisées dans le cadre d'une démarche globale de développement. De l'autre côté, elles souffrent aussi de la forte dépendance des programmes aux données dans le sens où la modification de l'organisation des données mène d'office à la modification des programmes existants.

L'avènement du concept de base de données a été justement vu comme la solution adéquate aux problèmes liés aux SGFs.

1.5.1. Définition d'une BD

Boudjlida définit une base de données comme une collection de données opérationnelles, enregistrées (sur un support adressable) et utilisées par des systèmes d'application (les programmes) d'une organisation (humaine) particulière. En outre, la collection de données est structurée indépendamment d'une application particulière, elle est cohérente, de redondance minimale et accessible simultanément par plusieurs utilisateurs (Boudjlida 2002).

La structuration de données a pour principal objectif d'éliminer les redondances de données dans la base. Alors que la mémorisation des informations permet d'un côté d'assurer une indépendance entre les programmes et les données et de l'autre côté elle permet d'intégrer toutes les données de l'entreprise dans un même support.

1.5.2. Fonctions d'une base de données

Une base de données étant contenant des données structurées peut assurer les fonctions fondamentales suivantes :

- **Stocker** l'information d'une manière permettant la restitution facile de cette dernière.
- **Traiter** de grandes masses de données.
- Traiter les données de manière optimale en temps d'accès et espace de stockage.
- **Contrôler** l'accès aux données.
- **Partager** des données par le fait que ces dernières sont séparées des programmes.

1.6. Système de Gestion de base de données (SGBD)

1.6.1. Définition d'un SGBD

Un système de gestion de bases de données (SGBD) est un ensemble d'outils logiciels permettant la création, la manipulation et le contrôle de bases de données.

Il inclut donc : les données, le matériel, les applications et les utilisateurs (Date 2004).

Gardarin précise que le principal objectif d'un SGBD est d'assurer l'indépendance des programmes aux données, c'est-à-dire la possibilité de modifier les schémas conceptuel et interne des données sans modifier les programmes d'applications, et donc les schémas externes vus par ces programmes (Gardarin 1994).

Afin d'assurer ces fonctionnalités, un langage de base de données « Structured Query Language » (SQL) est défini dans les SGBDs et qu'est composé des sous-langages suivants :

- Un langage de Définition de Données (LDD) qui permet à l'utilisateur de décrire des objets (étudiants, formation,…), leurs attributs (le nom des étudiants, le libellé des formations,…), leurs liens (une personne *est inscrite* à une formation) et des contraintes éventuelles sur ces objets.
- Un Langage de Manipulation de Données (LMD) qui offre à l'utilisateur le moyen de rechercher, de créer, de modifier et de supprimer des informations.
- Un Langage de Contrôle de Données (LCD) qui assure l'intégrité des données ainsi que l'accès sécurisé à ces dernières.

1.6.2. Niveaux d'abstraction

Les SGBD assurent une abstraction des données sauvegardées sur les supports physiques afin de simplifier la vision des utilisateurs. Cette abstraction a été proposée par le groupe ANSI/X3/SPARC (ANSI 1978) qui consiste à modéliser la description des données en trois niveaux d'abstraction : Conceptuel, Interne et Externe (Figure 8).

Cette manière de conception en trois niveaux d'abstraction est derrière l'importante caractéristique qui donne de la puissance au SGBD à savoir l'indépendance des données en séparant les niveaux logiques (schémas conceptuel et externe) du niveau interne (physique).

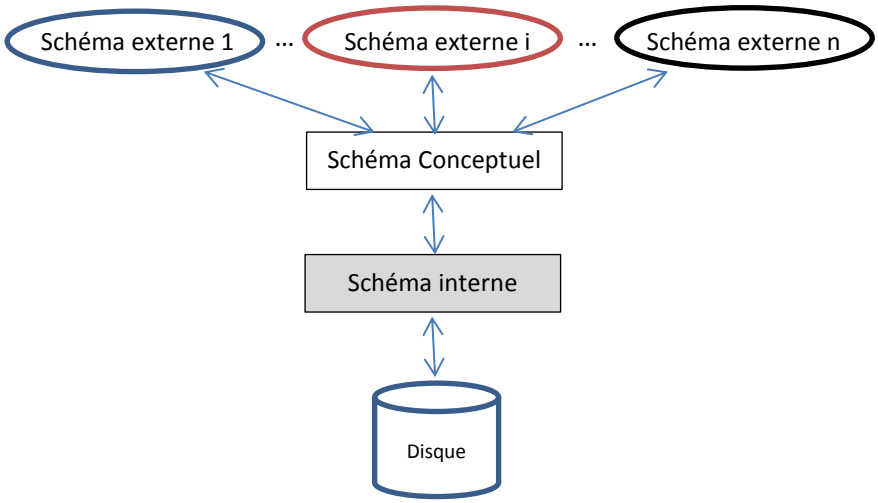


Figure 8: Niveaux d'abstraction d'un SGBD

1.6.2.1. Niveau conceptuel

A ce niveau, un schéma conceptuel décrit les données d'une entreprise ou d'une partie d'une entreprise en terme de types indépendants de toute représentation en machine, correspondant à une vue canonique globale de la portion d'entreprise modélisée (Gardarin 1994).

Comme illustré dans la figure 9, le schéma conceptuel (ou logique) permettra par exemple de définir:

- Les types de données élémentaires (ex. Nom, Prénom, Titre, …)
- Les types de données composées (ex. Etudiant, Formation, Inscription)

Les règles que devront suivre les données au cours de leur vie (ex. tout étudiant doit avoir une inscription, une durée de formation est 3 ans pour le niveau Licence ou 2 ans pour le niveau Master)

Types= d'objets :

Etudiant(Nom, Prénom, Adresse)

Formation(Titre, Niveau, durée)

Type de relations :

Inscription(Etudiant, formation, Année)

Figure 9: Exemple de schéma conceptuel

1.6.2.2. Niveau interne

A ce niveau, un schéma interne décrit des données d'une base au niveau de la représentation physique en machine correspondant à une spécification des structures de stockage et des méthodes d'accès utilisées pour stocker les données sur disque (Gardarin 1994).

Le schéma interne résume, donc, comment les données décrites dans le schéma conceptuel sont enregistrées dans les équipements de stockage. Cela se fait en déterminant les types de fichiers dans lesquels seront enregistrées les données et en définissant des structures de données supplémentaires, appelées indexes, qui aident à effectuer les opérations de recherche de données.

1.6.2.3. Niveau externe

A ce niveau, un schéma externe décrit une partie de la base de données correspondant à la vision d'un programme ou d'un utilisateur, donc à un arrangement particulier de certaines données (Gardarin 1994).

Le schéma externe permet, donc, de personnaliser l'accès des utilisateurs aux données. Ainsi, contrairement aux schémas conceptuel et interne qui décrivent toute une base de données, on peut avoir plusieurs schémas externes chacun décrivant la partie des données présentant un intérêt pour un utilisateur ou un groupe d'utilisateurs.

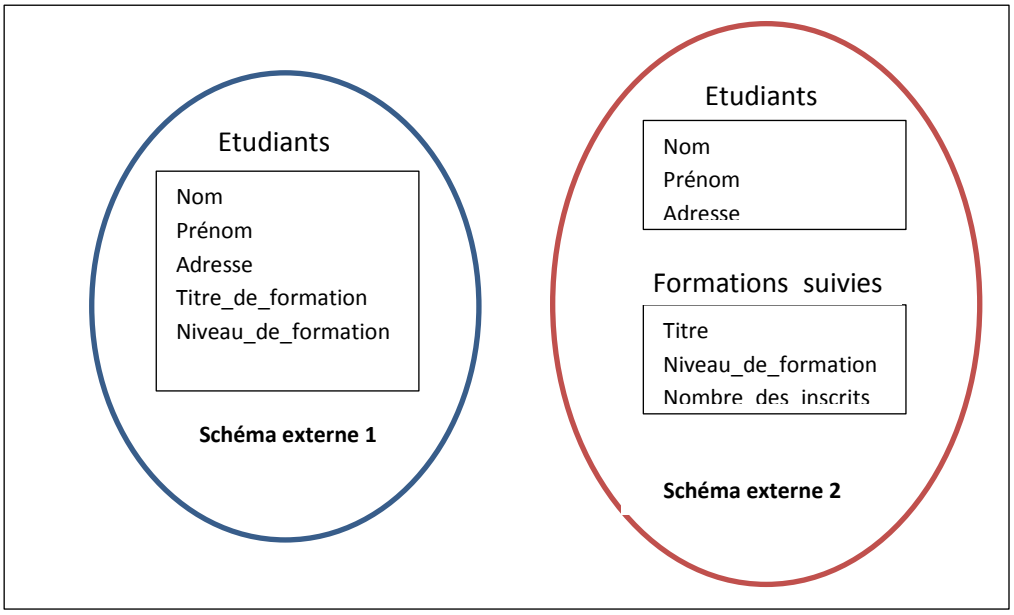


Figure 10: Exemples de schémas externes

Un schéma externe est une collection de vues considérées conceptuellement comme des relations, dont les enregistrements ne sont pas stockés mais plutôt calculés à partir de relations définies dans le schéma conceptuel (Figure 10).

1.6.3. Historique des SGBDs

Depuis les premiers jours de l'informatique, le stockage et la manipulation des données ont suscité beaucoup d'attention. Ramakrishnan et al, ont résumé l'historique de l'évolution des SGBDs de la façon suivante (Ramakrishnan 2003):

- En 1960, Charles Bachman conçut le premier SGBD au sein de la firme General Electric en USA, et le nomma Integrated Data Store.
- A la fin de l'année 1960, IBM développa « Information Management System » (IMS). Presque au même moment, le système SABRE fut développé conjointement par American Airlines et IBM pour faire les réservations aériennes et il permettait à plusieurs utilisateurs d'accéder aux mêmes données à l'aide d'un réseau.
- En 1970, Edgar Codd proposa au sein d'IBM une nouvelle représentation des données appelée modèle relationnel de données « Relational Data Model » (Codd 1970). L'avènement de ce modèle constitua un tournant dans le développement des systèmes de bases de données. En effet, ces systèmes devinrent une discipline académique et

l'utilisation des SGBDs pour la gestion des données dans les entreprises devint une pratique standard.

- En 1980, le modèle relationnel des données consolida sa position dominante de paradigme dans le domaine des systèmes de bases de données par l'avènement du langage des requêtes SQL développé par IBM.
- A la fin des années 1980 et 1990, des progrès ont été réalisés dans différents domaines relatifs aux systèmes de bases de données notamment, le perfectionnement d'un langage de requêtes plus puissant et l'enrichissement des modèles de données. Plusieurs firmes ont pu étendre leurs systèmes en les rendant capables de stocker de nouveaux types de données (tels que les images et les textes) et d'interroger les bases de données par des requêtes plus complexes. Ainsi, des systèmes spécialisés ont été développés pour la création des entrepôts de données (Data Warehouses) et l'intégration des données à partir de différentes bases.

  Une branche intéressante s'est développée par l'émergence de plusieurs progiciels de gestion intégrée « Entreprise Resource Planning » (ERPs) qui ajoutent principalement des couches de fonctionnalités au-dessus des SGBDs.

  Après l'avènement de l'Internet, plusieurs SGBDs ont pu remplacer les systèmes opérationnels dans le stockage des données utilisées par les sites Internet.

1.6.4. Avantages des SGBDs

Beaucoup d'avantages ont été cités dans (Date 2004) (Ramakrishnan 2003) (Gardarin 1994). Dans ce qui suit, nous donnerons un résumé sur chaque avantage cité dans une des références mentionnées ci-dessus:

- **Indépendance physique :**
  Les SGBDs fournissent une vue abstraite des données qui cachent les détails des données aux applications. Grâce à cette propriété très importante, il est possible de modifier le schéma interne sans avoir à modifier le schéma conceptuel malgré que les schémas interne et conceptuel représentent les mêmes données.

- **Indépendance logique :**
  L'indépendance logique permet de modifier un schéma externe sans avoir à modifier le schéma conceptuel. Cela assure aussi une

indépendance entre les différents utilisateurs à travers leurs schémas externes correspondant.

- **Manipulation des données par des langages non procéduraux :**
  Les SGBDs sont dotés de langages non procéduraux tels que SQL qui permettent aux utilisateurs (utilisateurs interactifs ou des programmes) de manipuler les données sans préciser d'algorithmes d'accès.

- **Accès plus efficace aux données :**
  Afin d'assurer aux utilisateurs un accès efficace et rapide aux données, les SGBDs fournissent des techniques sophistiquées telles que l'utilisation des mémoires caches pour que les accès aux données se fassent en mémoire du fait que les Entrées/Sorties disques représentent un goulot d'étranglement.

- **Intégrité et sécurité des données :**
  Afin d'assurer l'intégrité des données, les SGBDs appliquent des contraintes lors des mises à jour effectuées sur les données. Par exemple, lors d'une inscription d'un étudiant dans une formation, le SGBD peut contrôler si l'étudiant n'est pas déjà inscrit.
  La protection des données est aussi assurée par les SGBDs en appliquant des mécanismes adéquats pour autoriser, contrôler ou enlever des droits d'accès des utilisateurs à tout ensemble de données.

- **Administration centrale des données :**
  La centralisation de l'administration des données dans les SGBDs a pour but de minimiser les redondances et adapter le stockage de données dans le sens d'obtenir une recherche de données efficace.

- **Accès simultané et récupération après incident :**
  Dans les SGBDs, les utilisateurs jouissent d'un accès simultané aux données de telle manière qu'ils pensent que les données sont accédées par un seul utilisateur à la fois. En plus, les données sont protégées lors d'une panne de système.

- **Réduction de temps dans le développement d'application :**
  Le développement des applications est plus facile et rapide avec les SGBDs de fait que ces derniers intègrent beaucoup de fonctions communes aux applications en plus d'une interface de haut niveau avec les données.

1.7. En résumé

Dans ce chapitre, nous avons donné la définition d'un système d'information et sa composition. Ensuite, nous avons défini les sous-systèmes qui existent dans toute organisation ainsi que l'interaction qui les relie.

Le développement de pareils systèmes n'étant pas toujours évident, nous avons dressé brièvement quelques cycles de vie qui ont été défini dans le domaine de gestion des projets.

Finalement, nous avons terminé par discuter en particulier les concepts de base de données et des systèmes de gestion de bases de données.

1.8. Exercices

Exercice 1 : [1]

Déterminer, pour chaque traitement ci-dessous, s'il prépare spécifiquement une prise de décision en précisant, le cas échéant, des décisions qui peuvent en découler.

a) Préparation d'un bon de livraison.

b) Analyse des ventes saisonnières.

c) Étude de devis pour la rénovation d'un entrepôt.

d) Préparation du bilan comptable annuel.

e) Simulation des effets d'une augmentation des salaires.

Exercice 2 :

Tout système d'information est scindé en trois niveaux : sous-système opérationnel, sous-système de communication et sous-système décisionnel.

Donner des exemples du rôle du système à chaque niveau.

[1] Cet exercice est cité dans (Sornet 2017)

1.9. Références du chapitre

[ANSI 1978] ANSI/X3/SPARC Study Group on Data Base Management Systems, Framework Report on Database Management Systems, Information Systems, vol. 3, n° 3, 1978.

[Beck 2001] K. Beck, Extreme Programming, Computerworld, (2001).

[Boehm 1988] B. Boehm, A Spiral Model of Software Development and Enhancement, Computer, IEEE, 21(5):61-72, (988).

[Boehm 2000] B. Boehm, Spiral Development: Experience, Principles and Refinements, Special Report CMU/SEI-2000-SR-008, July 2000.

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

[Coad 1999] P. Coad, E. Lefebvre, J. De Luca, Java Modeling in Color with UML, Enterprise Components and Proces, Pearson PTR Pub, 15 juin 1999, ISBN: 0-13-011510-X, (1999).

[Codd 1970] E.F. Codd, A relational model of data for large shared databanks. CACM, vol. 13, n6, 1970.

[Collignon 2016] A. Collignon, J. Schöpfel, Méthodologie de gestion agile d'un projet. Scrum – les principes de base, I2D – Information, données & documents, Volume 53, p 12-15, (2016).

[Date 2004] C.J. Date, An introduction to database systems, Pearson 8ème edition, (2004).

[De Courcy 1992] R. De Courcy, Les systèmes d'information en réadaptation, Québec, Réseau international CIDIH et facteurs environnementaux, no 5 vol. 1-2 p. 7-10, (1992).

[Firesmith 2013] D. Firesmith, Using V Models for Testing, Carnegie Mellon University - Software Engineering Institute Blogs - sur insights.sei.cmu.edu, 11 décembre 2013, consulté le 11/12/2019.

[Gardarin 1994] G. Gardarin, Maîtriser les bases de données : modèles et langages, 2ème édition, Groupe Eyrolles (1994).

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

[Rivard 2001] S. Rivard, J. Talbot, Le développement de systèmes d'information: une méthode intégrée à la transformation des processus, 3ème édition, presses de l'université du Québec, (2001).

[Royce 1970] W. W. Royce, Managing the Development of Large Software Systems. IEEE Wescon, p. 1-9, (1970).

[Sornet 2017] J. Sornet, Systèmes d'information de gestion : l'essentiel en fiches, ISBN : 978-2-10-076243-9, Dunod, (2017).

[Suki 2018] C. T. Suki, Modélisation des systèmes d'information, Publié le 04/07/2018, https://www.supinfo.com/articles/single/7087-modelisation-systemes-information#idm45266072536864, visualisé le 10/12/2019.

CHAPITRE 2

Bases de données relationnelles

Historiquement, durant trois générations d'évolution des bases de données à savoir : (i) fichiers systèmes, (ii) bases de données hiérarchiques, (iii) bases de données CODASYL, le développement de ces modèles a connue des insuffisances qui persistaient. Effectivement, le manque d'indépendance des données et la difficulté d'accès à la base de données ont justement donné naissance à la quatrième génération des bases de données à savoir la technologie des Bases de Données Relationnelles (BDRs) ou Relationnel Databases caractérisées principalement par la notion de requête déclarative (Kim 1990).

Dans ce chapitre, nous allons définir les principaux concepts sur lesquels sont basées les BDRs à savoir la description des données, la normalisation par décomposition ainsi que le langage SQL.

2.1. Modèle relationnel : description des données

La structuration des données dans le modèle relationnel repose sur deux principaux concepts : le domaine et la relation (Hainaut 2012).

Un *domaine* est un ensemble nommé de valeurs donné a priori. Chaque valeur est supposée désigner un objet concret ou abstrait du monde réel.

Une *relation* contienne des informations sur les objets et les liens. La relation peut être définit comme un ensemble d'agrégats de n valeurs, chacune

appartenant à un domaine. Un agrégat est appelé *ligne* et les composants de même rang des lignes forment un *attribut* de la relation.

Théoriquement une relation est définie comme une partie du produit cartésien d'une suite de domaines. Soient D_1, D_2, …, D_n des domaines non nécessairement distincts. Une relation $R(A_1 :D_1, A_2 :D_2, \ldots, A_n :D_n)$ ou $R(A_1, A_2, \ldots, A_n)$ est définie comme :

$$R \subseteq D_1 \times D_2 \times \ldots \times D_n$$

Finalement, la description des données en terme de modèle est appelée un *schéma* qui spécifie pour chaque relation son nom, le nom de chaque attribut (champ ou colonne) et son type.

2.2. Normalisation par décomposition

La redondance d'informations dans une base de données cause énormément de problèmes dans la conception des bases de données tels que la redondance de stockage, les anomalies de mise à jour, les anomalies d'insertion et de suppression.

Le concept de normalisation a été justement défini comme une solution à ce phénomène dans le sens où elle permet de réaliser un raffinement de la base de données.

Plusieurs procédés de normalisation ont été adoptés pour pallier aux problèmes liés à la redondance d'informations à savoir la normalisation par décomposition, par synthèse, par dépendances multivaluées et par dépendances de jointure.

Dans ce manuscrit, nous ne pouvons se permettre de détailler tous les types de normalisations que le lecteur peut les trouver en détails dans la littérature (Ramakrishnan 2003) (Boudjlida 2002) (Hainaut 2012). Néanmoins, nous expliquerons le premier type de normalisation proposé par Codd (1970) à savoir, la normalisation par décomposition. Les autres procédés sont aussi bons et peuvent parfois révéler des redondances qui ne peuvent être détectées par la normalisation par décomposition (Ramakrishnan 2003).

2.2.1. Contraintes d'intégrité

Le raffinement de la base de données qui vise à éliminer au maximum la redondance de données doit être guidé par le strict respect des contraintes

d'intégrité. Dans ce sens, nous nous focalisons sur deux concepts importants: il s'agit de la clé de relation (ou l'identifiant) et la dépendance fonctionnelle.

Justement, la notion de normalisation par décomposition se base principalement sur les rapports qui existent entre ces deux concepts.

2.2.1.1. Définition de la clé de relation

La clé de relation est définie comme un ensemble d'attributs dont les valeurs permettent de distinguer les n-uplets de la relation.

Toutefois, une relation peut avoir plusieurs clés. Nous en choisirons une qui sera alors appelée *clé primaire* et les autres seront appelées *clés candidats*.

Exemples : On indique une clé de relation en soulignant ses composants.

VENTE(<u>ARTICLE</u>, MAGASIN, PRIX) : Un article n'est vendu que dans un seul magasin et à prix fixe.

VENTE(<u>ARTICLE, MAGASIN</u>, PRIX) : Un article peut être vendu par plusieurs magasins, son prix dépend du magasin qui le vend.

VENTE(<u>ARTICLE, MAGASIN, PRIX</u>) : Un magasin peut vendre un même article à différents prix.

2.2.1.2. Définition de la dépendance fonctionnelle

Dans une relation R(A, B, C, D), il existe une dépendance fonctionnelle A$\rightarrow$B si, à tout instant, deux lignes de R qui ont même valeur de A ont aussi même valeur de B.

ACHAT		
CLIENT	**PRODUIT**	**PRIX**
Mohamed	Café	700
Abdelkader	Café	700
Abdelkader	Vinaigre	30
Amine	Lit	25
Amine	Vinaigre	30

Tableau 1: Dépendance fonctionnelle dans une relation

Le Tableau 1 montre que quelles que soient les valeurs des attributs de la relation ACHAT, le PRIX ne dépond que du PRODUIT. On dit qu'il existe une dépendance fonctionnelle (DF) de PRODUIT vers PRIX, ce qu'on notera :

PRODUIT → PRIX

On dira que :

PRODUIT est le déterminant et PRIX est le déterminé de la dépendance fonctionnelle

2.2.2. La décomposition d'une relation

Selon Delobel et al, une la relation R(A, B, C, D), qui a une dépendance fonctionnelle A→B peut être décomposée en deux relations R1(A, B) et R2(A, C, D) (Delobel 1973).

Pour l'exemple de la relation ACHAT, la décomposition sera comme suit :

ACHAT(CLIENT, PRODUIT, PRIX), PRODUIT→PRIX peut être décomposée en deux relations TARIF(PRODUIT, PRIX) et ACHAT(CLIENT, PRODUIT)

2.2.3. Formes normales

La dépendance fonctionnelle est à la base des trois premières formes normales et celle de Boyce and Codd (Figure 11).

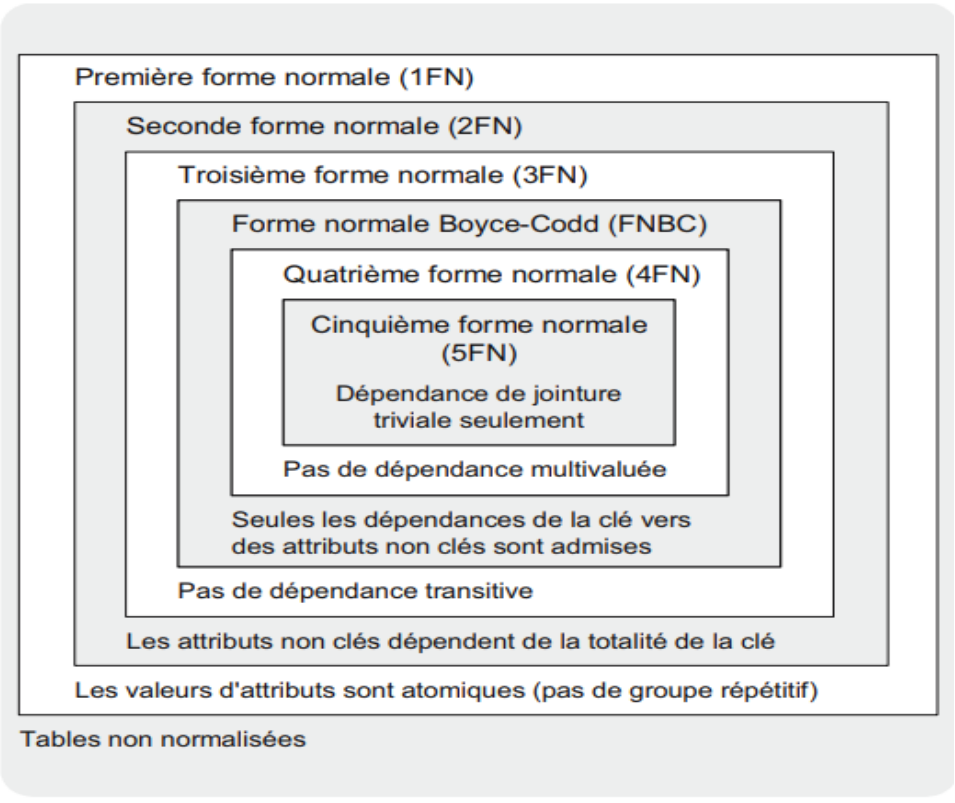


Figure 11: Les formes normales (Meier 2006)

2.2.3.1. Première forme normale

Une relation est dite en première forme normale (1NF), si chacun de ses attributs a un domaine atomique mono-valué.

C'est-à-dire, les relations avec attributs dont les valeurs sont des ensembles ou des listes ne sont pas en 1NF. Toutefois, on signale ici que les travaux menus dans le cadre des modèles orientés objet et des modèles relationnels-objet visent justement à surpasser cette contrainte de 1NF.

2.2.3.2. Deuxième forme normale

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

Exemple : la relation EVALUATION(MODULE, FORMATION, LIBELLE, NOTE) où la dépendance FORMATION → LIBELLE n'est pas en 2NF car l'attribut LIBELLE ne dépend que de la partie FORMATION de la clé MODULE, FORMATION.

2.2.3.3. Troisième forme normale

Une relation est en troisième forme normale (3NF) si (i) elle est en 2NF et si (ii) tout attribut n'appartenant pas à la clé ne dépend pas d'un attribut non clé (ou encore ne dépend pas transitivement de la clé).

Exemple : La relation AVION(NO_AVION, CONSTRUCTEUR, TYPE, CAPACITE, PROPRIETAIRE) du Tableau 2 n'est pas en 3NF puisque les attributs CONSTRUCTEUR et CAPACITE n'appartiennent pas à la clé et dépendent de l'attribut non clé TYPE. Par contre les relations AVION1(NO_AVION, TYPE, PROPRIETAIRE) et MODELE(TYPE, CONSTRUCTEUR, CAPACITE) obtenues par décomposition sont en 3NF.

AVION				
NO_AVION	**CONSTRUCTEUR**	**TYPE**	**CAPACITE**	**PROPRIETAIRE**
AH321	Boeing	B747	C1	Air Algérie
AF564	Airbus	A320	C2	Air France
BA777	Boeing	B747	C1	British Airways

Tableau 2: La relation avion (Boudjlida 2002)

2.2.3.4. Forme normale de Boyce and Codd

Une relation R est en forme normale de Boyce and Codd (BCNF) si et seulement si les seules dépendances fonctionnelles élémentaires qu'elle comporte sont celles où une clé détermine un attribut.

Exemple (Figure 12): Malgré que la relation LIVRE(TITRE, PREMIER_AUTEUR, NATIONALITE_AUTEUR, PAGES, ANNEE_EDITION, NUM_EDITION) avec les dépendances fonctionnelles TITRE → PREMIER_AUTEUR et TITRE → PREMIER_AUTEUR, NATIONALITE_AUTEUR est en 3NF mais les redondances ne sont pas toutes éliminées. La forme normale de Boyce and Codd permet d'éliminer ce type de redondance.

LIVRE					
Titre	**Premier Auteur**	**Nationalite Auteur**	**Pages**	**Annee Edition**	**Num Edition**
Database Management Systems.	Raghu Ramakrishnan	Américaine	1065	2003	3
Bases de données et systèmes d'information, le modèle relationnel : langages, systèmes et méthodes.	Nacer Boudjlida	Française	279	2002	1
Bases de données : Concepts, utilisation et développement.	Jean-Luc Hainaut	Belge	702	2012	2
Le data Warehouse: Guide de conduite de projet.	Ralph Kimball	Américaine	575	2005	1
The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling.	Ralph Kimball	Américaine	564	2013	3

Livre				
Titre	**Premier Auteur**	**Pages**	**Annee Edition**	**Num Edition**
Database Management Systems	Raghu Ramakrishnan	1065	2003	3
Bases de données et systèmes d'information, le modèle relationnel : langages, systèmes et méthodes	Nacer Boudjlida	279	2002	1
Bases de données : Concepts, utilisation et développement	Jean-Luc Hainaut	702	2012	2
Le data Warehouse: Guide de conduite de projet.	Ralph Kimball	575	2005	1
The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling.	Ralph Kimball	564	2013	3

Auteurs	
Premier Auteur	**Nationalite Auteur**
Raghu Ramakrishnan	Américaine
Nacer Boudjlida	Française
Jean-Luc Hainaut	Belge
Ralph Kimball	Américaine

Figure 12: La relation LIVRE

Les relations LIVRE(TITRE, PREMIER_AUTEUR, PAGES, ANNEE_EDITION, NUM_EDITION) et AUTEURS(PREMIER_AUTEUR, NATIONALITE_AUTEUR) obtenues de la décomposition de la relation LIVRE sont en BCNF tout en constatant que la dépendance TITRE → PREMIER_AUTEUR, NATIONALITE_AUTEUR a disparu.

2.3. Langage de requêtes SQL

Dans ce qui suit nous présentons les deux sous-langages du SQL (Structured Query Langage) à savoir : Data Definition Langage (DLL) et Data Manipulation Langage (DML) en utilisant la norme SQL2 du langage largement disponibles dans les SGBDs (Hainaut 2012).

2.3.1. Data Definition Langage (DLL)

Dans cette section nous décrivons les principales commandes de définition et de manipulation des structures : créer, supprimer et modifier une table, un domaine ou une contrainte.

2.3.1.1. Création d'un schéma

Une base de données est définie par son schéma.

```
create schema ETUDEVAL
```

Les schémas sont rassemblés dans un catalogue qui représente un ensemble de base de données. Un site peut contenir plusieurs catalogues.

2.3.1.2. Création d'une table

Par la requête create table on spécifie le nom de la table et la description de ses colonnes (nom de la colonne et le type de ses valeurs).

```
create table ETUDIANT ( NETUD        char(10),
                        NOM, PRENOM  char(40),
                        GENRE        char(1),
                        DATE_NAISS   DATE,
                        LIEU_NAISS   char(30),
                        ADRESSE      char(80) ) ;

create table MODULE ( CODE_MOD   char(10),
                      LIBELLE_MOD char(60) ) ;
```

SQL propose des types de base dans la déclaration de colonne. On citera les principaux :

SMALLINT : entier signé long (par ex. 16 bits),

INTEGER ou (INT) : entier signé long (par ex. 32 bits),

NUMERIC(p, q) : nombre décimal de p chiffres dont q après le point décimal (q prend par défaut la valeur de 0),

FLOAT(p) ou (FLOAT) : nombre en virgule flottante d'au moins p bits significatifs,

CHARACTER(p) ou (CHAR) : chaîne de longueur fixe de p caractères,

CHARACTER VARYING ou (VARCHAR(p)) : chaîne de longueur variable d'au plus p caractères,

BIT(p) : chaîne de longueur fixe de p bits,

BIT VARYING: chaîne de longueur variable d'au plus p bits,

DATE: date (année, mois de et jour),

TIME: instant (heure, minute, seconde, éventuellement 1000^{e} de seconde),

TIMESTAMP: date + temps,

INTERVAL: intervalle en années/mois/jours entre dates ou en heures/minutes/secondes entre instants,

Il est possible de déclarer des domaines de valeurs pour les types de base comme suit :

```
create domain NOTE decimal(2,2)

create domain MATRICULE char(10)

create domain LIBELLE char(60)

create table ETUDIANT (NETUD        MATRICULE,
                       NOM, PRENOM  LIBELLE,
                       GENRE        char(1),
                       DATE_NAISS   DATE,
                       LIEU_NAISS   char(30),
                       ADRESSE      char(80)) ;

create table MODULE ( CODE_MOD      MATRICULE,
                      LIBELLE_MOD   LIBELLE) ;
```

2.3.1.3. Les clés de relation

Elle peut être primaire :

```
create table ETUDIANT (NETUD        MATRICULE,
                       NOM, PRENOM  LIBELLE,
                       GENRE        char(1),
                       DATE_NAISS   DATE,
                       LIEU_NAISS   char(30),
                       ADRESSE      char(80)
                       primary key (NETUD)) ;
```

Comme elle peut être secondaire en utilisant le prédicat unique:

```
create table HEBERGEMENT ( NUM_HEBERG    MATRICULE,
                           NETUD         MATRICULE,
                           NUM_CHAMBRE   char(5),
                           primary key (NUM_HEBERG),
                           unique (NETUD)) ;
```

Comme elle peut être étrangère en référençant la table par la clause foreign key:

```
create table EVALUATION( NETUD      MATRICULE,
                         CODE_MOD   MATRICULE,
                         NOTE_EXAM,
                         NOTE_TD,
                         NOTE_TP    NOTE,
                         DATE_EXAM  DATE,
                         primary key (NETUD,CODE_MOD),
                         foreign key (NETUD) reference ETUDIANT,
                         foreign key (CODE_MOD) reference MODULE) ;
```

2.3.1.4. Opérations sur les tables

Une table peut être supprimée et son contenu sera perdu.

```
drop table EVALUATION
```

La commande suivante ajoute la colonne NATIONALITE à la table ETUDIANT :

```
alter table ETUDIANT add column NATIONALITE char(30);
```

La commande suivante élimine la colonne DATE_EXAM de la table EVALUATION :

```
alter table EVALUATION drop column DATE_EXAM;
```

Une caractéristique d'une colonne peut être modifiée :

```
alter table ETUDIANT alter column GENRE set default 'M';

alter table ETUDIANT alter column GENRE drop default;
```

Un domaine peut être modifié ou supprimé :

```
alter domain NOTE set default 0.0;

drop domain MATRICULE ;
```

Il est possible d'ajouter une clé primaire :

```
alter table ETUDIANT add primary key (NETUD);
```

ou étrangère:

```
alter table EVALUATION add foreign key (NETUD) references ETUDIANT;
```

Lors de la déclaration d'une contrainte, Il est possible de la nommée:

```
create table EVALUATION ( NETUD     MATRICULE,
                          CODE_MOD  MATRICULE,
                          NOTE_EXAM
                          NOTE_TD,
                          NOTE_TP   NOTE,
                          DATE_EXAM DATE,
        constraint CT1 primary key (NETUD,CODE_MOD),

        constraint CT2 foreign key (CODE_MOD) reference MODULE) ;
```

Il est possible d'ajouter une contrainte à une table existante

```
alter table EVALUATION add constraint CT3 foreign key (NETUD)
references ETUDIENT;
```

Il est aussi possible de supprimer une contrainte nommée existante:

```
alter table EVALUATION drop constraint CT3;
```

2.3.2. Data Manipulation Langage (DML)

Dans cette section nous examinons les principes de l'extraction et de la modification de données d'une table.

L'extraction est réalisée par une seule commande : la requête select qui contient trois parties principales.

- La clause select précise les colonnes à extraire
- La clause from précise la ou les table(s) à partir desquelles les données seront extraites
- La clause where précise les conditions à respecter lors de la sélection des lignes.

2.3.2.1. Requêtes simples

Afficher toutes les colonnes de la table ETUDIANT

```
select * from ETUDIANT;
```

Afficher certaines colonnes NETUD, NOM, PRENOM de la table ETUDIANT

```
select NETUD, NOM, PRENOM from ETUDIANT;
```

Afficher certaines lignes sélectionnées de la table ETUDIANT

```
select NETUD, NOM, PRENOM from ETUDIANT where LIEU_NAISS = 'Tiaret';
```

Pour éliminer les lignes en double, on peut utiliser la clause distinct comme suit :

```
select distinct LIEU_NAISS from ETUDIANT where DATE_NAISS = '21/12/1971';
```

Au lieu de l'égalité, on peut utiliser d'autres relations de comparaison (>, <, <>, >=, <=) qui correspondent respectivement à (plus grand que, plus petit que, différent de, plus grand que ou égal, plus petit que ou égal).

Une condition peut porter sur l'existence de la valeur NULL :

```
DATE_NAISS is null

DATE_NAISS is not null
```

Une condition peut également porter sur l'appartenance à un ensemble :

GENRE in ('F', 'M') ou GENRE not in ('F', 'M')

Ou à un intervalle:

NOTE_EXAM between 0.0 and 20.00

NOTE_EXAM not between 0.0 and 9.99

Ou encore sur la présence de certains caractères dans une valeur

PRENOM like 'M_STAPHA'

ADRESSE like '%Tiaret%' ou ADRESSE not like '%Tiaret%'

DATE_NAISS like '%1971%'

Le signe "_" désigne un caractère quelconque et "%" désigne toute suite de caractères.

Pour utiliser les deux signes ensemble dans une même recherche, il suffit de les préfixer par un caractère spécial comme suit :

ADRESSE like '%$_chaine%'

Les conditions peuvent êtres composées :

select NOM, NOTE_EXAM from EVALUATION where NOM ='M%' and NOTE_EXAM>=10 ;

2.3.2.2. Données dérivées

En plus des données extraites de la base de données, il est possible de spécifier des données dérivées, des constantes et des alias de colonnes:

select 'TVA de', NPROD as Produit, '=', 0,17*PRIX*QSTOCK;

select 'Moyenne de', NETUD as ETUDIANT, '=', 0.6*NOTE_EXAM+0.2*NOTE_TD+0.2*NOTE_TP as moy;

from EVALUATION

where moy>=10.00

Cette requête affiche le tableau suivant :

Moyenne de	NETUD	=	Moy
Moyenne de	3827	=	16.5
Moyenne de	5827	=	11.05
Moyenne de	9827	=	13.75
Moyenne de	21827	=	10.00

Encore, il est possible de dériver des données en utilisant des fonctions avec la clause select ou la clause where et aussi avec les instructions update et insert.

char_length(*ch*) : donne le nombre de caractrères de la chaîne *ch*,

position(*ch1* in *ch2*) : donne la position de la chaîne *ch1* dans la chaîne *ch2*, *1* si *ch1* est vide et *0* si *ch1* n'apparaît pas dans *ch2* ;

ch1 || *ch2* : concatène les deux chaînes *ch1* et *ch2* ;

lower(*ch*) : transforme les caractères de *ch* en minuscules;

upper(ch) : transforme les caractères de *ch* en majuscules;

substring(*ch* from *I* for *L*) : donne une chaîne formée de *L* caractères de la chaîne *ch* partant de la position *I* ;

trim(*e c* from *ch*) : supprime les caractères *c* à l'extrémité *e* de la chaîne *ch* ; les valeurs de *e* sont *leading*, *trailing* et *both*

Il est aussi possible d'utiliser une fonction de sélection comme suit :

```
select NOM, PRENOM,
        case GENRE
                when 'M' then 'Masculin'
                when 'F' then 'Féminin'
                else 'inconnu'
        end, ADRESSE
from ETUDIANT;
```

A chaque fois que GENRE retourne 'M' ou 'F', il est affiché respectivement 'Masculin' ou 'Féminin'.

Des valeurs d'environnement de l'utilisateur peuvent êtres fournit :

current_user : l'identificateur de l'utilisateur courant,

current_date : la date courante,

current_time : l'instant courant,

current_timestamp : date+instant courant,

2.3.2.3. Fonctions statistiques

count(*) : retourne le nombre de lignes trouvées,

count(nom-colonne) : retourne le nombre de valeurs de la colonne,

avg(nom-colonne) : retourne la moyenne des valeurs de la colonne,

sum(nom-colonne) : retourne la somme des valeurs de la colonne,

min(nom-colonne) : retourne la minimum des valeurs de la colonne,

max(nom-colonne) : retourne la maximum des valeurs de la colonne.

2.3.2.4. Condition par sous requêtes

La requête suivante donne les étudiants qui ont passé les examens des modules dont leurs libellés comportent la chaine de caractère 'Algo'.

```
select *
from ETUDIANT
where NETUD in
        (select NETUD
        from EVALUATION
        where (DATE_EXAM is not null) and
        (CODE_MODULE in
                (select CODE_MOD
                from MODULE
                where LIBELLE_MOD = 'Algo%')));
```

Dans la requête suivante on utilise des sous requêtes corrélées pour sélectionner les étudiants dont leur note d'examen est supérieure à la moyenne des notes d'examens du même module.

```
select NETUD
from EVALUATION A
where NOTE_EXAM > (select avg(NOTE_EXAM)
                        from EVALUATION
                        where CODE_MOD = A.CODE_MOD);
```

La requête suivante montre comment utiliser une condition d'association quantifiée pour rechercher les étudiants qui ont passé au moins deux examens.

```
select NETUD, NOM, PRENOM
from ETUDIANT A
where (select count(*)
         from EVALUATION
        where NETUD = A.NETUD) >=2);
```

2.3.2.5. Extraction de données de plusieurs tables

La requête suivante réalise une jointure entre les tables ETUDIANT et EVALUATION où les lignes obtenues sont formées d'une ligne de chacune des deux tables. Les lignes couplées ont la même valeur de la colonne NETUD qui constitue une clé étrangère dans la table EVALUATION et primaire dans la table ETUDIANT.

```
select EVALUATION.NETUD, EVALUATION.DATE_EXAM,
       ETUDIANT.NETUD, ETUDIANT.NOM, ETUDIANT.ADRESSE
from EVALUATION, ETUDIANT
where EVALUATION.NETUD = ETUDIANT.NETUD
```

On peut fusionner les lignes obtenues d'une requête à celles obtenues d'une autre requête en utilisant l'opérateur union comme suit :

```
select ADRESSE from ETUDIANT where LIEU_NAISS <> 'Tiaret'
union
select ADRESSE from ETUDIANT where GENRE='F'
```

Cependant, l'utilisation de l'opérateur union all empêche l'élimination des lignes en double.

2.3.2.6. Extraction de données groupées

Par exemple, on peut obtenir le nombre d'étudiants pour chaque groupe d'étudiants classés en utilisant la clause group by pour une colonne donnée comme suit :

```
select NOM, PRENOM, count(*) as NOMBRE_ETUDIANTS
from ETUDIANT
group by GENRE
```

Pour imposer des conditions sur les groupes, la clause having est dédiée à de telles situations afin d'éviter toute confusion avec la clause where qui porte uniquement sur les lignes. La requête suivante donne les étudiants qui ont obtenu une note d'examen inférieure à 10.00 au moins une fois :

```
select NETUD, count(*)
from ETUDIANT
where NETUD in(select NETUD
               from EVALUATION
               where NOTE_EXAM < 10.00)
group by NETUD
having count(*) >= 1;
```

La requête suivante montre comment combiner la jointure et le groupement pour, par exemple, obtenir pour chaque étudiant le nombre d'examens auxquels a assisté.

```
select A.NETUD, A.NOM, A.GENRE, count(*) as nombre_examens
from ETUDIANT A, EVALUATION B
where A.NETUD = B.NETUD and B.NOTE_EXAM is not null
group by A.NETUD, A.NOM, A.GENRE;
```

On signale l'utilisation de NOM et GENRE au niveau de la clause group pour permette leur présence au niveau de la clause select et non pour le groupement lui-même. En d'autres termes, on pourrait ignorer la mention de NOM et GENRE au niveau de la clause group si on n'a pas besoin d'afficher le NOM au niveau de la clause select.

On peut aussi grouper par des expressions de calcul comme la requête suivante qui constitue des groupes d'étudiants selon la première lettre de leur nom.

```
select substring(NOM from 1 for 1) as NOM, count(*) as N
from ETUDIANT
group by substring(NOM from 1 for 1);
```

Les données sélectionnées dans une requête peuvent être classées par ordre croissant ou décroissant sur une colonne ou plusieurs en utilisant la clause order by avec la mention asc ou desc pour spécifié respectivement un ordre ascendant ou descendant.

Par exemple, la requête suivante classe les lignes par NOM en ordre décroissant puis dans chaque NOM elle les classe par PRENOM en ordre décroissant.

```
select *
from ETUDIANT
order by NOM, PRENOM desc;
```

Il est à noter que la clause order by n'admet pas l'utilisation d'un alias de colonne, par contre on peut utiliser l'ordre de la colonne : order by 2 desc.

2.3.2.7. Modification des données

On peut ajouter des lignes à une table existante par l'instruction insert. L'ordre des valeurs doit respecter l'ordre des colonnes lors de leurs créations. Toute colonne non spécifiée prend la valeur null ou la valeur par défaut.

```
Insert into EVALUATION values ('1201', 'Algo1', 14.5, 16.00, 15.00, '21/12/2019');
```

On peut ajouter à la table ETUDIANT_MASCULIN(NUMERO, NOM, ADRESSE) des données extraites d'une ou plusieurs tables comme illustré dans la requête suivante :

```
insert into ETUDIANT_MASCULIN
select NETUD, NOM, ADRESSE
from ETUDIANT
where GENRE = 'M';
```

On peut supprimer des lignes d'une table désignées par la clause where comme suit :

```
delete from ETUDIANT
where NETUDIANT = '1201';
```

La requête suivante supprime de la table ETUDIANT les lignes qui spécifient un étudiant qui n'a passé aucun examen :

```
delete from ETUDIANT
where NETUD not in (select NETUD
                    from EVALUATION);
```

De la même façon, des modifications peuvent être effectuées sur les lignes d'une table spécifiées dans la clause where comme suit :

```
update ETUDIANT
set ADRESSE = 'Rue el Amir Abdelkader, Tiaret'
where NETUD = '1201';
```

2.3.2.8. Mise à jour et contraintes référentielles

Tous les SGBDs garantissent l'intégrité des données en respectant les contraintes d'intégrité au moment des mises à jour effectuées par les utilisateurs.

La définition de la structure de la table EVALUATION dans les requêtes suivante comporte la mention on delete **no action** pour refuser la suppression d'une ligne de la table ETUDIANT s'il existe une ou plusieurs lignes de EVALUATION dépendantes (c'est-à-dire dont EVALUATION.NETUD = ETUDIANT.NETUD).

```
create table ETUDIANT( NETUD        MATRICULE not null,
                       primary key (NETUD)     ) ;

create table EVALUATION ( CODE_MOD    MATRICULE not null,
                          NETUD       MATRICULE not null,
```

```
primary key (CODE_MOD),
foreign key (NETUD) references ETUDIANT on
delete no action) ;
```

Par contre, avec la mention on delete **cascade** et on delete **set null** la suppression est acceptée, mais la première (cascade) entraîne la suppression conjointe des lignes de EVALUATION dépendantes et la deuxième (set null) attribue la valeur null à la colonne NETUD des lignes dépendantes et rend ces dernières indépendantes.

Les mêmes actions sont préconisées pour la modification de la valeur de l'identifiant NETUD de ETUDIANT avec la mention on update. Refus s'il existe des lignes de EVALUATION dépendantes (no action). Propagation par modification des clés étrangères pour les évaluations dépendantes (cascade) et mise à null des clés étrangères (set null).

2.4. En résumé

Les BDRs sont connues principalement par l'indépendance des données et la facilité d'accéder à ces données ainsi que l'utilisation d'un langage déclaratif facilitant leur manipulation. Dans ce chapitre, les principaux concepts ont été discutés tels que la structuration des données en se basant sur la notion du domaine ainsi que la notion de relation. Un autre concept très important a été abordé en détail, est celui de la normalisation des données par décomposition. Cette dernière est basée sur les notions de contraintes d'intégrité, décomposition des relations et formes normales. Le célèbre langage déclaratif SQL a été décrit en expliquant, par des exemples de requêtes, ses deux modules à savoir le langage de définition des données (DLL) et le langage de manipulation des données (DML).

2.5. Exercices

Exercice1 :

Décomposer la relation suivante :

EVALUATION (NETUD, CODE_MOD, LIBELLE_MOD, NOTE_EXAM)

NETUD $\rightarrow$ CODE_MOD, NOTE_EXAM

CODE_MOD $\rightarrow$ LIBELLE_MOD

Exercice2 :

Décomposer la relation suivante :

ENSEIGNE (NENSEIGNANT, NOM, PRENOM, SALLE, HEURE, CODE_MOD, LIBELLE_MOD, CODE_FORMATION, LIBELLE_FORMATION)

NENSEIGNANT → CODE_MOD, CODE_FORMATION, SALLE, HEURE

NENSEIGNANT → NOM, PRENOM

CODE_MOD → LIBELLE_MOD

CODE_FORMATION → LIBELLE_FORMATION

Exercice3 :

Soit les relations suivantes:

EMPLOYE(NEMP, NOM, AGE, SALAIRE)
AFFECTATION(NEMP, NSERV)
SERVICE(NSERV, LIBELLE_SERVICE, NCHEF_SERVICE)

a) Donnez les requêtes SQL nécessaires à la création des tables correspondantes aux relations précédentes incluant les contraintes d'intégrité sur les clés primaires et étrangères.

b) Définissez la relation SERVICE en SQL de manière à garantir un chef de service pour chaque service.

c) Ecrivez la requête SQL qui ajoute l'employé BOUSLIMAN ABDELKADER avec NEMP = 71, AGE = 29 et SALAIRE= 30000.

d) Ecrivez la requête qui ajoute à tous les employés une augmentation de salaire de 10%.

Exercice4 :

Soit les informations suivantes relatives à une base de données pour gérer des équipes de Football participant à une compétition :

- Chaque équipe a un identificateur, un nom, 22 joueurs, un entraineur, une couleur.
- Chaque joueur a un identificateur, un nom et un prénom, un poste, une date de naissance.

- Dans la compétition, il y a 16 équipes qui sont divisées en quatre groupes pour jouer un $16^{ème}$ de final dont les deux premiers du classement de chaque groupe joueront un $8^{ème}$ de final. les gagnants du $8^{ème}$ final joueront en quart de final. les gagnants de ce dernier joueront en demi final et les gagnants de ce dernier joueront la finale.
- Chaque match a un numéro, équipe 1, équipe 2, une date, heure de début et heure de fin, un score en temps règlementaire, éventuellement un score en temps supplémentaire, éventuellement un score de tirs au but, et une phase ($16^{ème}$, $8^{ème}$, quart de $8^{ème}$, demi final ou la finale).

Répondez aux questions suivantes :

a) Définissez les relations qui expriment les informations précédentes

b) Ecrivez les requêtes SQL qui définissent les tables relatives aux relations définies dans la réponse à la question précédente tout en incluant les contraintes d'intégrité nécessaires aux clés primaires et étrangères.

c) Ecrivez une requête qui affiche tous les matchs nuls en temps règlementaires.

Ecrivez les requêtes qui affichent pour chaque équipe : le nombre de matchs gagnants, perdants et nuls, le nombre de buts marqués et le nombre de buts encaissés.

2.6. Références du chapitre

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

[Codd 1970] E.F. Codd, A relational model of data for large shared databanks. CACM, vol. 13, n6, 1970.

[Date 2004] C.J. Date, An introduction to database systems, Pearson 8ème edition, (2004).

[Delobel 1973] C. Delobel et R.G. Casey, Decomposition of a database and the theory of Boolean switching functions, IBM Journal of research and Development, Vol. 17, No 5, pp. 374-386, (1973).

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

[Kim 1990] W. Kim, Introduction to object-oriented databases, ISBN 0-262-11124-1, (1990).

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

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

CHAPITRE 3

Base de données objet

Le terme Bases de Données Objet (BDOs) « Object DataBases en anglais » fait référence aux bases de données avec des caractéristiques d'objet. Les BDOs se sont développés suivant deux chemins distincts : les Bases de Données Orientées-Objet (BDOOs) « Object-Oriented DataBases en anglais» et les Bases de Données Relationnelles-Objet (BDROs) « Object-Relational DataBases en anglais ».

3.1. Historique des bases de données objet

Au moment où l'intérêt pour l'utilisation de la technologie des bases de données commençait à se répandre, les bases de données relationnelles (BDRs) ont été utilisées pour une variété de nouvelles applications telles que l'ingénierie de conception, les systèmes d'information géographiques et les systèmes de télécommunication. Cependant, l'utilisation des données complexes (spatiales, multimédia et vocales) par ces applications a posé des problèmes de représentation pour les BDRs étant donné que ces derniers exigent des valeurs simples et atomiques (chaines de caractères, entiers, réels et logiques) pour leurs attributs.

Le développement d'un modèle de BD en réponse au besoin de traiter des données non adaptées aux tables traditionnelles du modèle relationnel fût devenu une nécessité pressante.

L'intérêt grandissant pendant les années 80 pour les langages de programmation orientés objet « Object-Oriented Programming Languages en anglais » (OOPls) tels que C++ et Simula a beaucoup influencé sur le développement des BDOOs. D'un côté, les OOPLs ont introduit une nouvelle approche dans le développement des applications basé principalement sur l'utilisation de l'objet avec encapsulation de sa structure et son comportement. De l'autre côté, la communauté des BDs a commencé à penser à la fusion entre la persistance des données objets avec les OOPLs ce qui a conduit directement à l'émergence des BDOOs.

Donc, les BDOOs ont été développées dans le but d'intégrer la persistance aux OOPLs donnant par conséquent une solution au fameux problème connue par le terme erreur d'impédance « impedance mismatch en anglais » qui résume les difficultés rencontrées lorsqu'une BDR est desservi par une application écrite en langage orienté objet.

En réponse au développement des BDOOs, la communauté des BDRs a développé les BDROs dans le but d'étendre les BDRs en prenant en charge de nombreux concepts orientés objet (Dietrich 2011).

3.2. Concepts fondamentaux

3.2.1. Objet

L'objet est un concept abstrait pour représenter une entité décrivant son état et son comportement. Considérant une personne comme un objet, son état peut contenir des informations telles qu'un identifiant, un nom et une adresse. Son comportement peut contenir les méthodes de création, de suppression de l'objet et de modification de son état. Un objet peut aussi avoir des méthodes qui le relient à d'autres objets, ex. l'objet personne peut avoir la méthode inscription qui le relie à l'objet cours. Toute méthode a une signature décrivant son nom ainsi que les noms et les types de ses paramètres.

3.2.2. Classe

Une classe est l'abstraction de caractéristiques communes, dans le sens où elle décrit des objets qui ont les mêmes signatures. Un objet constitue donc une instance de la classe.

3.2.3. Objets complexes

Les objets simples sont définis par des constructeurs qui combinent des objets simples pour créer une forme de haut niveau d'un objet. Ex. un avion peut être considéré comme un objet de haut niveau formé à partir d'objets de niveau inférieur tels que la carrosserie de l'avion, l'aile, la queue et le moteur. Il est aussi possible de construire des collections telles que (i) les ensembles « sets »: des collections d'éléments non ordonnés (ii) les multi-ensembles « bags »: des ensembles contenant des éléments dupliqués, (iii) les listes « lists » : des collections d'éléments ordonnés.

3.2.4. Identité de l'objet

Un identifiant est affecté à l'objet « Objet IDentifier en anglais » (oid) au moment de son création. Un oid reste inchangé durant la vie de l'objet. Par contre, l'état de l'objet peut changer en modifiant les valeurs de ses propriétés sans pour autant que l'identité de l'objet change.

3.2.5. Encapsulation

L'encapsulation consiste à rassembler les données et les méthodes (interfaces) au sein d'une structure en cachant l'implémentation de l'objet. Ce mécanisme empêche l'accès aux données par un autre moyen que les interfaces proposées. L'encapsulation permet donc de garantir l'intégrité des données contenues dans l'objet.

3.2.6. Extensibilité

Il est possible de définir de nouveaux types de données correspondant à la sémantique de l'application en plus des types prédéfinis par le système sans qu'il y ait de distinction entre l'utilisation de ces derniers et ceux définis par l'utilisateur.

3.2.7. Classes hiérarchiques et Héritage

Les concepts de classe et d'héritage sont inséparables. Les classes peuvent être modélisées sous forme hiérarchique en utilisant l'héritage. La hiérarchie peut être en sous-classes et en superclasses ou en d'autres termes en spécialisation et en généralisation. La spécialisation focalise sur les attributs et les relations des sous-classes qui n'existent pas dans la superclasse. La généralisation focalise sur les attributs et les relations communs des sous-classes pour former une superclasse.

3.2.8. Redéfinition et surcharge

Redéfinition ou overriding et le fait de redéfinir un comportement d'une méthode défini dans une classe supérieure pour une méthode défini au niveau d'une sub-classe.

Quant au concept de surcharge ou overloading, il s'agit de permettre à une ou plusieurs classes d'avoir des méthodes portant le même nom mais qui se distinguent par le nombre et le type de leurs paramètres.

3.3. Base de données Orientées Objet (OODB)

La norme (ODMG) « de l'anglais Object Data Management Group » a été développée par la communauté BDOO dans le but d'aider à la spécification d'une interface objet commune. Deux langages faisant partie d'ODMG à savoir ODL « Object Definition Language » pour spécifier le schéma de la base de données et OQL « Object Query Language » qu'est un langage déclaratif pour récupérer les données à partir de la BDOO.

3.3.1. Langage de définition objet ODL

ODL est utilisé dans la déclaration des classes y compris les propriétés et les signatures des opérations. L'implémentation des opérations requière en revanche un langage de programmation spécifique tel que C++ ou Java (Harrington 1999).

3.3.1.1. Les classes

La déclaration suivante résume la définition d'une classe.

```
class NomClasse
( extent NomExtent
  key NomCle )
{ attribute TypeAttribut NomAttribut;
relationship TypeRelation NomRelation
inverse SpecificationInverse;
// SignaturesMethode;
};
```

Une classe a une extension NomExtent pour gérer les instances de la classe et une clé NomCle pour référencier l'ensemble de propriétés (attributs et relations) qui peuvent identifier les instances de manière unique. La clé ne peut être déclarée qu'on présence d'une extension.

Un attribut décrit les caractéristiques d'un objet et la relation décrit une association entre deux classes. La classe NomClasse possède un attribut

NomAttribut de type TypeAttribut et une relation NomRelation de type TypeRelation. Afin d'assurer l'intégrité de la relation, la spécification de la relation inverse est décrite dans la déclaration de la relation pour maintenir la correspondance de l'identifiant de l'objet dans les deux côtés de la relation. Ex. si un employé est lié à ses enfants, il est obligatoire qu'il y ait un moyen d'assurer que lorsque l'identifiant d'un enfant est inséré dans l'objet employé, l'identifiant du même employé doit être inséré dans l'objet enfant, ce qu'est similaire en quelques sortes à l'intégrité référentielle dans le modèle relationnel.

Le comportement d'une classe est donné par les signatures des méthodes définies dans la classe.

L'héritage des classes est proposé par ODL sous deux formes :

(i) héritage de l'état et du comportement de l'objet qui peut être spécifié par la clause extends comme suit,

```
class subclasse extends superclasse
( . . . )
{
// Propriétés et comportement spécifiques de la subclasse sont définis ici
};
```

(ii) héritage du comportement seulement à travers l'interface décrivant le comportement abstrait du type en utilisant le symbole ' :' comme suit,

```
interface NomInterface
{ // SignaturesMethode; };
class NomClasse: NomInterface ( . . . ) { . . . };
```

Les deux formes d'héritage peuvent être combinées comme suit :

```
class NomSubclasse extends NomSuperclasse: NomInterface ( . . . ) { . . . };
```

3.3.1.2. Les attributs et les contraintes

L'exemple suivant montre comment ODL permet de déclarer des attributs de type composé en utilisant la clause struct ou de type collection (set, bag ou list)

```
struct StructureComposite
{ string attrA1;
string attrA2;
string attrA3;
};
class C {
attribute string s1;
```

```
attribute string s2;
attribute string s3;
attribute StructureComposite s4;
attribute set<string> s5;

constraint<notnull> on s1;
constraint<notnull> on s2;
constraint<unique> on s2;
constraint<unique, propagate=off > on s3;
};
```

Il est possible d'appliquer deux contraintes sur les objets : not null et unique. L'exemple précédent montre aussi comment les utiliser.

L'attribut s1 doit être non nul, l'attribut s2 doit être non nul et unique et l'attribut s3 doit être unique.

La contrainte sur l'attribut s3 n'est pas propagée aux classes inférieures puisqu'elle est déclarée avec la mention propagate=off.

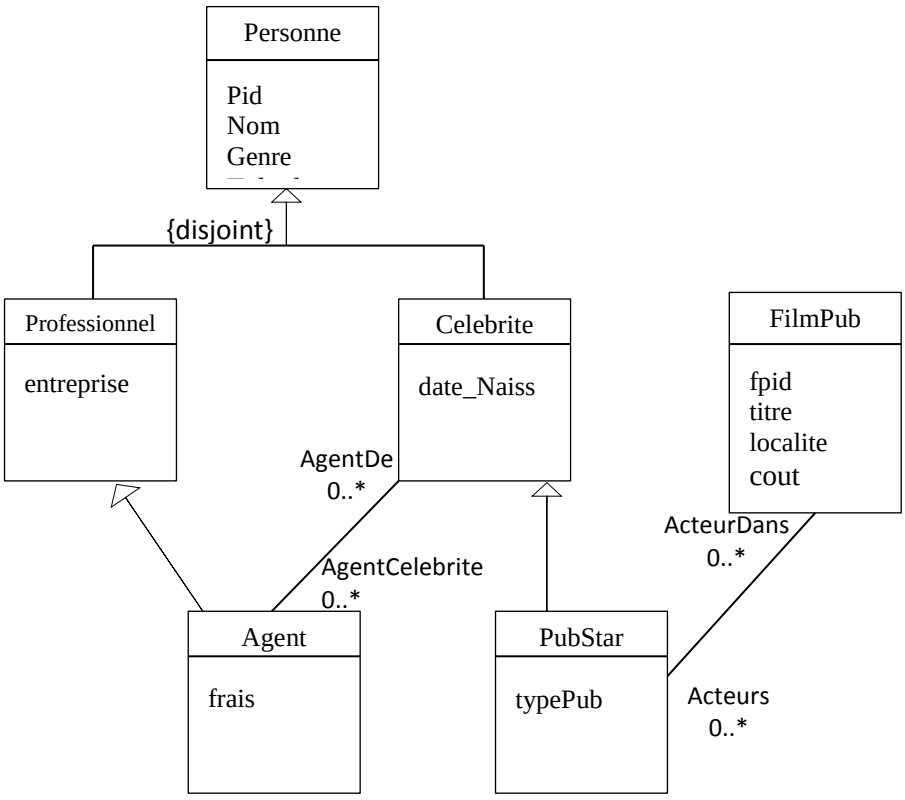


Figure 13 : Diagramme UML d'un exemple de gestion des films publicitaires

Pour les besoins d'illustration dans cette section nous allons considérer l'exemple de la figure 13. Ce dernier ainsi que ses requêtes ODL et OQL ont été adapté principalement à partir d'un exemple cité dans (Dietrich 2011).

Les requêtes suivantes constituent le schéma ODL de l'exemple schématisé dans la figure 13.

```
class Personne
( extent personnes
key pId )
{ attribute string pId;
attribute string nom;
attribute string genre;
attribute string telephone;
attribute string addresse;
};
class Professionel extends Personne
( extent professionels)
{ attribute string entreprise;
} ;
class Celebrite extends Personne
( extent celebrites )
{ attribute date date_Naiss;
relationship Agent AgentCelebrite inverse Agent::agentDe;
};
class PubStar extends Celebrite
( extent pubStars )
{ attribute string typePub;
attribute float salaire;
relationship set<FilmPub> ActeurDans inverse FilmPub::Acteurs;
};
class Agent extends Professionnel
( extent agents )
{ attribute float frais;
relationship set<Celebrite> agentDe inverse Celebrite:: AgentCelebrite;
};
class FilmPub
( extent filmPubs
key filmPubId )
{ attribute string filmPubId;
attribute string localite;
attribute float cout;
relationship set<PubStar> Acteurs inverse PubStar::ActeurDans;
relationship Sponsor sponsorePar inverse Sponsor::FilmPubsSponsores;
};
```

3.3.2. Langage de requête objet OQL

OQL utilise le format de la clause select-from-where utilisé par SQL du modèle relationnel. Cependant, puisque OQL est appliqué sur un modèle objet, la clause from spécifie les collections d'objets pertinents pour la réponse à la requête. De même, la clause select en OQL spécifie également la structure du résultat de la requête mais elle est plus expressive qu'en SQL.

La requête suivante retrouve les stars qui apparaissent dans les films publicitaires réalisés à "*Alger*" :

```
select a.nom
from f in filmPubs, a in f.Acteurs
where f.localite = "Alger";
```

La propriété Acteurs dans cette requête est un ensemble d'objets PubStar qui participent dans le film publicitaire. Puisque PubStar est une personne; la clause select renvoie le nom de l'acteur.

3.3.2.1. Les résultats d'une requête OQL

Le résultat d'une requête SQL est une collection (set, multiset ou bag) de tuplets ou de structures. Tandis que, dans une requête OQL le résultat peut être des atomes, structures, collections et littéraux.

Le résultat de la requête précédente est une bag <string>. Il est possible qu'une requête OQL retourne une collection de structures. La requête suivante retourne le nom de l'acteur et son agent pour chaque acteur participant au film publicitaire nommé "*FilmPublicitaire1*".

```
select struct( PubStarNom: a.nom,
agentNom: a.AgentCelebrite.nom )
from f in filmPubs, a in f.Acteurs
where f.titre = " FilmPublicitaire1";
```

La requête utilise des labels pour chaque structure de champ (PubStarNom : spécifie le nom de la star du film publicitaire et agentNom : spécifie le nom d'agent de l'acteur). Le résultat est donné comme suit :

```
bag < struct (PubStarNom : string, agentNom : string) >.
```

Une requête peut être définie par un nom en utilisant le mot clé define. Par exemple, la requête qui affiche les agents avec une date de naissance après le 31/13/2000 est définie par le nom jeunesAgents

```
define jeunesAgents as
select ja
from ja in agents
```

```
where ja.date_naiss >= "31/12/2000";
```

Le type de jeunesAgents est
bag < Agent >.

Avec define, Il est aussi possible de définir un objet par un nom :

```
define Film1 as
element ( select f
from f in FilmPubs
where f.titre = "film1");
```

Il est donc possible d'obtenir n'importe qu'elle valeur de l'objet film1 comme suit:

```
film1.localite
```

3.3.2.2. Utilisation des méthodes dans les requêtes

Supposant que la classe Celibrite a une méthode age qui calcule l'âge d'une célébrité en utilisant l'attribut Date_Naiss. La requête suivante donne les noms des célébrités ayant moins de 30 ans.

```
select c.nom
from c in celebrites
where c.age < 30;
```

3.3.2.3. Les requêtes embarquées

Les clauses, select et from peuvent embarquer des sous-requêtes. La requête suivante retourne le titre et le coût des films publicitaires localisés à "*Alger*" ayant un coût supérieur à 10 million en une variable d'itération alg qui itère dans la sous-requête embarquée dans la clause from :

```
select struct(titre: alg.titre, cout: alg.cout )
from alg in (select f
             from f in filmPubs
             where f.localite = "Alger" )
where alg.cout > 10,000,000;
```

Le résultat est donné comme suit :

```
bag < struct (titre : string, cout : float) >.
```

L'intégration de la sous requête peut être effectuée au niveau de la clause select comme la requête suivante qui renvoie les noms des célébrités gérés par un agent :

```
select struct(  NomAgent: a.nom,
```

```
                agentDeCelebrites: (select c.nom from c in a.agentDe))
    from a in agents;
```

La requête retourne le nom de chaque agent et les noms des célébrités gérés par cet agent. Le résultat sortira sous la forme suivante.

bag < struct (NomAgent :string, agentDeCelebrites :bag < string >) >.

3.3.2.4. La quantification existentielle et universelle

La quantification existentielle et universelle est effectuée respectivement par les clauses exists et for all.

Par exemple, la requête suivante utilise le quantificateur existentiel exists pour identifier les stars de cinéma possédant au moins un film publicitaire réalisé à "Alger".

```
select NomFilm: m.nom
from m in pubStars
where exists p in m.ActeurDans:
p.localite = "Alger";
```

Tandis que la requête suivante utilise le quantificateur universel for all pour donner les stars de cinéma pour lesquelles tous leurs films publicitaires coûtent plus de 20 millions.

```
select m.nom
from m in pubStars
where for all fp in m.ActeurDans: fp.cout > 20,000,000;
```

3.3.2.5. Ordonner le résultat

OQL utilise la même clause order by utilisée en SQL avec la mention asc ou desc pour obtenir respectivement un ordre croissant ou décroissant.

La requête suivante donne un ordre alphabétique des stars de cinéma pour chaque film publicitaire réalisé à "*Alger*";

```
select struct(  algTitre: fp.titre,
                algPubStars: ( select a.nom
                               from a in fp.acteurs
                               order by a.nom ))
from fp in FilmPubs
where fp.localite = "Alger"
order by fp.titre asc;
```

Le résultat est donné sous la forme suivante :

list < struct (algTitre : string, algPubStars : list < string >) >.

3.3.2.6. Collection

Avec OQL on peut extraire le premier ou le dernier élément d'une collection indexée ainsi que des éléments indexés dans cette collection.

La requête suivante donne le dernier élément d'une collection de nom des Stars dans les films publicitaires réalisés à "Alger" par ordre décroissant.

```
last (select struct(StarNom: m.nom)
    from m in pubStars, p in m.ActeurDans
    where p.localite = "Alger"
    order by StarNom asc);
```

Il est possible d'extraire les dix premiers éléments de la liste précédente en utilisant l'expression indexée [0 :9] comme suit :

```
(select struct(StarNom: m.nom)
    from m in PubStars, p in m.ActeurDans
    where p.localite = "Alger"
    order by StarNom asc)[0:9];
```

3.3.2.7. Agrégation et groupement

OQL utilise les mêmes opérateurs d'agrégation que ceux utilisés en SQL à savoir (min, max, count, sum, avg).

La requête suivante utilise l'opérateur count pour compter le nombre des films publicitaires dans lesquels une star de cinéma a joué :

```
select struct (  nom: m.nom,
                 NbrFilms: count(m.ActeurDans) )
from m in PubStars
order by nom asc;
```

La requête suivante calcule le cout total des films publicitaires de chaque star de cinéma:

```
select struct ( nom: m.nom,
coutFilms: sum(select p.cout from p in m.ActeurDans) )
from m in PubStars
order by nom asc;
```

La clause group by en OQL fournit une référence à une collection d'objets de chaque groupe comme la requête suivante qui retourne le nombre et l'ensemble de célébrités représentés par un agent en regroupant les célébrités en fonction du nom de l'agent de la célébrité :

```
select struct(  agentName: m. AgentCelebrite.nom,
                NbrCelebrite: count(partition),
                NomCelebrites: (select p.m.nom from p in partition))
from m in PubStars
group by NomAagent: m.AgentCelebrite.nom;
```

Le mot clé partition désigne les groupes de noms des agents dans lesquels est compté le nombre des célébrités.

On peut restreindre le résultat de la requête précédente aux agents ayant plus de trois célébrités, en ajoutant la clause having à la fin de la requête précédente comme suit :

```
having count(partition) > 3
```

3.4. Bases de données Relationnelles-Objet (BDRO)

Les applications non traditionnelles génèrent beaucoup de données sous formes de vidéos, audio, données spatiales, données complexes, données hiérarchiques structurées qui ne sont pas supportées par les BDRs. L'idée d'intégrer les concepts orientés objet dans les BDRs a pris le chemin auprès des chercheurs et a abouti à la création des BDROs.

Le SQL standard supporte un grand nombre de concepts orientés objet comme extensions du relationnel en se basant essentiellement sur l'utilisation des types de données définies par l'utilisateur « User-Defined Data Types en anglais» (UDTs).

3.4.1. Les types de données intégrés

En plus des types de données atomiques telles que le caractère, l'entier, le réel et le booléen, le SQL standard supporte de nouveaux types de données structurées telles que les ensembles et les collections. Les ensembles peuvent contenir des valeurs de différents types alors que les collections doivent êtres composées de valeurs homogènes.

3.4.1.1. Le type ligne

Dans l'exemple suivant l'adresse (location) de la classe footballClub contient trois valeurs (street, town, prefecture). Avec le type row, location est considérée conceptuellement comme une colonne avec des valeurs non atomiques où chaque élément représente un champ. Ce qui signifie que la forme normale 1NF n'est pas respectée.

```
create table footballClub
( fcid varchar(10),
nom varchar(50) not null,
acronyme varchar(10)
localite row (rue varchar(30), ville varchar(5), prefecture varchar(5)),
phone varchar(12),
entraineurClub varchar(11) references entraineur (pId),
primary key (cId));
```

L'insertion d'une occurrence dans la table footballClub se fait comme suit :

```
insert into footballClub values
("FC1",
"Jeunesse Sportive Musulmane de Tiaret ",
"JSMT",
row("Avenue elAmir abdelkader", "Tiaret", "Tiaret"),
"C1");
```

L'accès aux différents champs du type row se fait à l'aide de la notation point comme suit :

```
select c.localite.rue, c.localite.ville, c.localite.prefecture
from footballClub c
where c.nom = "Mouloudia Club Alger";
```

3.4.1.2. Les types collections

Le SQL standard propose le type de collection dans sa forme de tableau seulement (les formes list, set et bag ne sont pas supportées) en utilisant la mention array comme suit :

```
create table footballClub
( fcid varchar(10),
nom varchar(50) not null,
acronyme varchar(10),
localite row (rue varchar(30), ville varchar(5), prefecture varchar(5)),
telephone varchar(12),
entraineurClub varchar(9) references entraineur(pId),
joueurs varchar(11) array[22] references joueur(pId),
primary key (cId));
```

La requête suivante montre comment réserver de l'espace d'un tableau vide pour l'attribut, joueurs:

```
insert into footballClub values
("FC1",
"Jeunesse Sportive Musulmane de Tiaret ",
"JSMT",
row("Avenue elAmir abdelkader", "Tiaret", "Tiaret"),
"C1",
array[ ]);
```

On peut utiliser la clause update pour ajouter des joueurs à un club en utilisant un tableau :

```
update footballClub
set joueurs = array["J1", "J2", "J3"]
where nom = " Jeunesse Sportive Musulmane de Tiaret ";
```

On peut aussi utiliser une position dans le tableau pour effectuer une modification :

```
update footballClub
set joueurs[1] = "J4"
where nom = "Jeunesse Sportive Musulmane de Tiaret ";
```

L'utilisation de la position dans le tableau permet aussi de retourner l'identifiant du premier élément du tableau : joueurs

```
select joueurs[1]
from footballClub
where nom = "Jeunesse Sportive Musulmane de Tiaret ";
```

Il est possible de connaître la taille du tableau en utilisant la fonction cardinality :

```
cardinality(c.joueurs)
```

3.4.2. Types de données définis par l'utilisateur (UDTs)

Les UDTs sont des donnés abstraites qui encapsulent leurs structures de données internes. Cependant, les UDTs se distinguent par le fait que leurs attributs et les méthodes de ces derniers sont publics et ne peuvent pas être protégés ou privés. Dans cette catégorie on peut distinguer deux types : types distincts et types structurés.

3.4.2.1. Les types distincts

Les types distincts donnent une nouvelle définition aux types atomiques en leurs associant une sémantique spéciale

L'exemple suivant définie deux types différents, taille et poids sur la base du type atomique integer en utilisant la clause as de l'instruction create type:

```
create type taille as integer final;
create type poids as integer final;

create table personne
( personneId varchar(3),
personneTialle taille,
personnePoids poids,
primary key (personneId));
```

Les trois types taille, poids et integer sont conceptuellement différents et par conséquent ne sont pas comparables. Cependant, il est quand même possible de les comparer après les avoir transformé à l'aide de la fonction cast comme illustré dans la requête suivante qui retourne toutes les personnes ayant un poids qui dépasse la norme recommandée par rapport à la taille.

```
select personneId
from personne
where cast (personnePoids as integer) > cast (personneTaille as integer)-
100;
```

3.4.2.2. Les types structurés

Reprenant l'exemple précédent dans lequel nous avons défini l'adresse d'un club de football. Au lieu de déclarer l'adresse comme un ensemble à l'aide de row, il est possible de la déclarer comme un type structuré comme suit :

```
create type localiteUDT as
(rue varchar(30),
 ville varchar(5),
 prefecture varchar(5))
not final;
```

Le mot not final est toujours mentionné afin de permettre la déclaration des sous-types.

3.4.2.3. Méthode définie par l'utilisateur

L'utilisateur peut définir dans un type structuré sa propre méthode. En plus des paramètres implicites auxquels on peut accéder par la mention self et qui représentent l'instance du type dans lequel est définie la méthode, il est possible de définir des paramètres explicites dans la signature de la méthode. L'exemple suivant illustre cet aspect:

```
create type rectangle as
(longueur real,
 largeur real)
not final
method surface() returns real;
create method surface() returns real for rectangle
begin
return self.longueur * self.largeur;
end
```

3.4.3. Types, tables et hiérarchie

L'exemple suivant définit un type structuré comme superclasse et sous-classe:

```
create type personneUDT as
( pId varchar(11),
nom varchar(20),
prenom varchar(20),
date_naiss date)
```

```
instantiable not final ref is system generated;
```

Une table basée sur un type structuré a en plus une colonne à référence automatique contenant un identifiant d'objet unique, appelé référence, pour chaque ligne de la table. Dans le cas de la table personneUDT la référence unique d'objet pour les instances du type est générée automatiquement par le SGBD puisque la mention system generated est spécifiée. Si la mention user defined ou derived est mentionnée, alors l'utilisateur est responsable d'assurer l'unicité de la référence.

```
create type entraineurUDT under personneUDT as
(rang varchar(20))
instantiable
not final;
create type joueurUDT under personneUDT as
( poste varchar(20))
instantiable
not final;
```

Remarquons qu'une hiérarchie de types est définie en déclarant les deux types entraineur et joueur comme sous-types du type personneUDT avec la clause under.

La clause instantiable est recommandée pour une éventuelle déclaration de tables basées sur ces types structurés.

En effet, on peut déclarer des tables en se basant sur les types structurés comme suit :

```
create table personne of personneUDT
( primary key (pId),
ref is personneID system generated);
```

Il faut noter ici que la clause ref is doit nommer la colonne à référence pour qu'il soit possible de manipuler cette dernière en cas où la référence est déclarée user defined.

On peut aussi définir une hiérarchie de tables. La requête suivante déclare les tables entraineur et joueur comme sous-tables de la super-table personne.

```
create table entraineur of entraineurUDT under personne;
create table joueur of joueurUDT under personne;
```

3.4.4. Relation

Dans l'exemple précédent, il existe d'un côté, une relation entre la table footballClub et la table entraineur où un club a un seul entraineur et ce dernier ne peut entrainer qu'un seul club, et de l'autre côté, il y a une relation entre la table footballClub et la table joueur où un club a plusieurs joueurs et chaque joueur ne peut jouer que pour un seul club. Pour modéliser ces deux relations on peut utiliser ce qu'on appelle le type référence en utilisant la clause ref.

```
create type footbalClubUDT as
( cId varchar(10),
nom varchar(50) not null,
acronyme varchar(10),
localite locationUDT,
telephonne varchar(12),
entraineurClub ref(entraineur), /* Référence à l'objet entraineur */
joueurs ref(joueur) array[22]) /* Référence à l'objet joueur */
instantiable not final ref is system generated);
create table footballClub of footballClubUDT
( primary key (cId),
ref is footballClubId system generated);
```

La définition du type de entraineurClub en ref(entraineur) est appelé type référence. Cette définition indique que la valeur stockée dans l'attribut entraineurClub est un objet référençant une ligne au lieu d'une seule valeur d'une clé étrangère dans le modèle relationnel.

Le type référence permet d'effectuer une jointure implicite entre les deux tables footballClub et entraineur dans une requête select comme le montre la requête suivante :

```
select entraineur → nom
from footballClub c
where c.localite.prefecture = 'Tiaret';
```

La requête interroge la table footballClub mais elle ramène les valeurs à partir de la table entraineur sans utiliser une condition de jointure explicite entre les deux tables.

Il est possible de retourner toute la structure du type associé à la valeur de la référence en utilisant la fonction deref() avec la clause select :

```
select deref(entraineur)
```

3.5. En résumé

Historiquement, le développement des BDOs a pris deux chemins : le chemin des BDOOs ainsi que le chemin des BDROs. Le premier s'est basé

exclusivement sur une conception objet alors que le deuxième a été basé sur une conception mixte combinant le modèle relationnel et le modèle orienté objet. Les principaux concepts des bases de données BDOOs ont été expliqués. Les deux langages pour la définition et la manipulation des données des BDOOs à savoir respectivement ODL et OQL ont été expliqués avec des exemples de requêtes. Par la suite, l'approche des bases de données BDROs a été détaillée en expliquant, par des exemples de requêtes, comment les concepts orientés objet ont été intégrés dans les bases de données relationnelles.

3.6. Exercices

Exercice 1:

Reconsidérez l'exemple schématisé dans la figure 13 et répondez sur les questions suivantes :

a) Ecrivez une requête OQL qui retourne les noms des célébrités qui ont un agent.

b) Ecrivez une requête OQL qui affiche une liste ordonnée sur le coût des films publicitaires réalisés à "Alger" en incluant le titre du film.

Exercice 2: [2]

Soient la définition des deux classes film et theatre relatives à la gestion de projection des films dans les théâtres :

```
interface film
( extent films key nomFilm )
{ attribute string nomFilm;
attribute date debut;
attribute date fin;
relationship Set(theatre) projeteDans inverse theatre :: projection;
};
interface theatre
( extent theatres key nomTheatre )
{ attribute string nomTheatre;
attribute string adresse;
attribute integer prixTicket;
relationship Set(film) projection inverse film :: projeteDans;
float numProjection() ;/* function calculant le nombre des projections
dans un théatre*/
```

[2] Cet exercice est tiré d'un exemple cité dans (Ramakrishnan 2003)

```
};
```

Répondez aux questions suivantes:

a) Définissez la classe projectionSpeciale en héritant la classe film tout en dotant la nouvelle classe par les attributs maxParticipants et beneficeCharite pour désigner respectivement le nombre maximum de participants et le bénéfice de charité collecté.

b) Ecrivez la requête OQL qui retourne les paires de films et de théâtres de sorte que le film soit projeté au théâtre et que le théâtre montre plus d'un film

c) Ecrivez la requête OQL qui retourne pour chaque prix de ticket le nombre de films montrés.

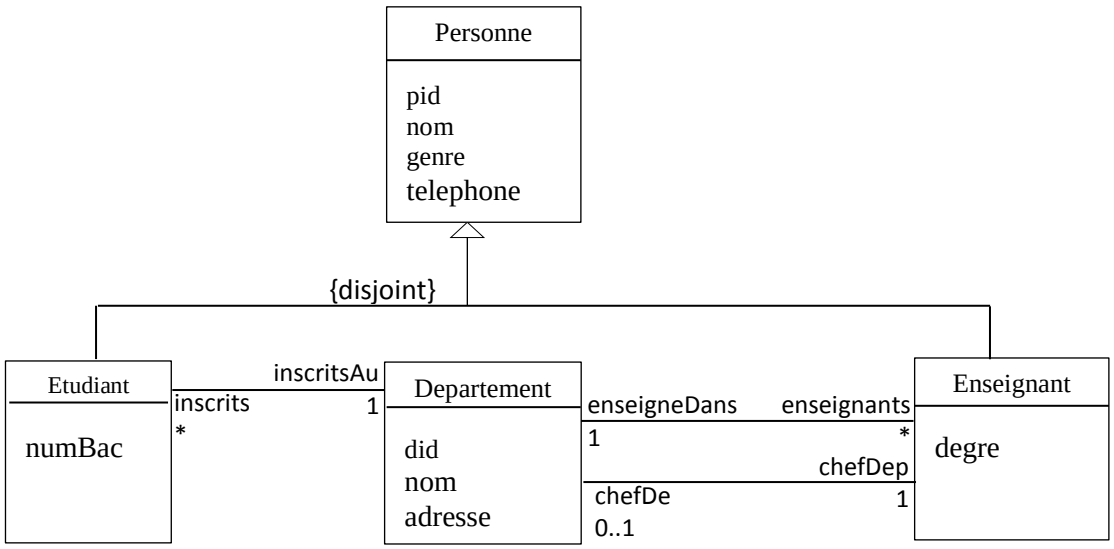
d) Ecrivez la requête OQL qui retourne les dix premiers théâtres ayant les prix de ticket les moins chers.

e) Expliquez brièvement que fait la requête suivante :

```
select bas, haut, num:avg(select p.t.numProjection() from partition p)
from theatre t
group by bas: t.PrixTicket < 100, haut:t.prixTicket>=100
```

Exercice 3:

Soit le schéma suivant:



Répondez aux questions suivantes :

a) Ecrivez les requêtes SQL qui définissent les classes personne, etudiant, enseignant et departement.

b) Ecrivez une requête SQL qui retourne le nombre des étudiants inscrits au département ainsi que le nombre des enseignants qui enseignent dans le même département en incluant le nom de ce dernier.

c) Ecrivez une requête SQL qui retourne les noms des départements et leur chef de département selon l'ordre croissant du nombre des enseignants de chaque département.

3.7. Références du chapitre

[Dietrich 2011] S.W. Dietrich and S.D. Urban, Fundamentals of Object Databases: Object-Oriented and Object-Relational Design, ISBN: 9781608454761, DOI 10.2200/S00315ED1V01Y201012DTM012, Morgan & Claypool, (2011).

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

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

CHAPITRE 4

Base de Données Distribuées

L'idée principale derrière la technologie des Bases de Données Distribuées (BDDs), ou qualifiées aussi par Bases de Données Réparties, est de combiner les deux technologies : les Bases de Données et les réseaux informatiques. Même si cette combinaison semble contradictoire du fait que les BDs favorisent la centralisation de traitement des données alors que les réseaux informatiques favorisent la décentralisation de traitement des données mais réellement le but dans les BDDs est de réaliser une intégration des données et non une centralisation.

4.1. Motivation

Après avoir vu leurs données augmentées phénoménalement et devenues très dispersées, les organisations ont émis le besoin à :

- des serveurs de BDs qui offrent de bonne performance en temps d'accès,
- des techniques de partitionnement de données,
- des systèmes d'accès parallèle aux données,

Dans ce contexte, les Bases de Données Distribuées (BDDs) se sont imposées comme une solution idéale aux problèmes rencontrés dans ces organisations.

4.2. Définition d'une BDD

Une Base de Données Distribuées est un ensemble de plusieurs bases de données liées logiquement et distribuées sur un réseau informatique (Özsu 2011).

D'après la définition, on comprend que les données dans une BDD peuvent être hétérogènes et localisées sur différents sites mais elles apparaissent à un utilisateur final comme une base unique, homogène et intégrée.

4.3. Définition d'un SGBDD

Un Système de Bases de Données Distribuées (SGBDD) est un système qui gère des collections de BDs logiquement reliées, distribuées sur un réseau, en fournissant un mécanisme d'accès qui rend la distribution transparente aux utilisateurs (Özsu 2011).

La transparence fait référence au fait qu'un système transparent cache les détails de la mise en œuvre aux utilisateurs ce qui favorise le développement d'applications complexes.

4.4. Avantages des SGBDD

De nombreux avantages des SGBDDs sont cités dans la littérature (Ramakrishnan 2003)(Özsu 2011). Nous citons dans ce qui suit l'essentiel de ces avantages.

4.4.1. Indépendance des données

L'indépendance des données est logique en assurant l'immunité des applications utilisateurs aux modifications de la structure logique de la BD. Elle est aussi physique en cachant les structures de stockage aux applications de l'utilisateur.

4.4.2. Transparence de la distribution

Dans les BDDs, l'utilisateur ne fait pas la différence entre les applications qui tournent sur des BDs centralisées et celles qui tournent sur des BDDs que ce soit par rapport aux services fournis ou aux données interrogées

4.4.3. Transparence de la fragmentation

Les objets des BDDs étant fragmentés, le système assure une stratégie de traitement de requête basée sur les sous-relations (fragments) plutôt que sur les relations entières même si la requête est spécifiée sur ces dernières.

4.4.4. Fiabilité garantie par les transactions distribuées

En garantissant une prise en charge des transactions distribuées, les SGBDDs assurent aux applications utilisateurs d'accéder à une seule image logique de la BD sans se préoccuper de la coordination de leurs accès à des BDs locales individuelles ou de la possibilité de panne de site ou de liaison de communication lors de l'exécution de leurs transactions.

4.4.5. Performance améliorée

La performance des SGBDDs est assurée par le fait que d'un côté ces derniers fragmentent conceptuellement les DBs en stockant les données à

proximité de leurs points d'utilisation ce qui diminue considérablement les conflits pour les services CPUs et les entrées/sorties. De l'autre côté, le parallélisme inhérent aux SGBDDs peut être exploité pour assurer un parallélisme inter-requêtes (exécution de plusieurs requêtes simultanément) et intra-requêtes (division d'une requête en plusieurs sous-requêtes et l'exécution de chacune d'elles sur un site différent).

4.5. Architectures des BDDs

Trois modèles d'architectures sont les plus utilisés pour la séparation des fonctionnalités des SGBDs, à savoir : le modèle Clien/Server, le modèle Peer to Peer et le modèle Multi-bases.

4.5.1. Modèle Client/Server

Le modèle Client/Server est un modèle distribué est constitué d'un système de haute performance, le serveur, et plusieurs systèmes de basses performances et les clients. Le serveur est l'unité centrale d'enregistrement ainsi que le seul fournisseur de contenu et de service. Un client demande uniquement du contenu ou l'exécution de service, sans partager aucune de ses propres ressources (Schollmeier 2002).

L'idée principale dans l'architecture Client/Server réside dans la séparation entre les fonctionnalités. Les clients sont responsables de l'interface utilisateur et le serveur gère les données et exécute les transactions. Le courrier électronique (Email), l'impression en réseau et le World Wide Web sont de bons exemples du modèle Client/server (Sun 2009).

4.5.2. Modèle Peer to Peer (P2P)

Contrairement au modèle Client/Server où un processus client envoie une seule requête au processus serveur, dans le modèle Peer to Peer (P2P) il n'est pas nécessaire d'avoir une coordination centralisée du fait que les nœuds participants sont à la fois fournisseurs et consommateurs de ressources (Schollmeier 2002).

Effectivement, dans le modèle P2P un processus client a la capacité de diviser une requête en sous-requêtes pour qu'elles soient exécutées sur différents sites et par la suite rassembler les réponses à ces sous-requêtes. De même, lorsqu'un processus serveur reçoit une requête qui nécessite des données situées sur d'autres serveurs, il décompose la requête en sous-requêtes pour qu'elles soient exécutées sur d'autres serveurs et rassemble les réponses pour répondre à la requête originale.

4.5.3. Modèle Multi-BDs

Les systèmes Multi-BDs font référence au cas où les SGBDs individuels (distribués dans notre cas) sont totalement autonomes et il n'existe aucune coopération entre eux.

Les systèmes multi-BDs sont caractérisés par le fait que leur schéma conceptuel global définit une vue conceptuel sur seulement une collection de quelques BDs locales que chaque SGBD local estime la nécessité de les partager. Alors qu'un schéma conceptuel global dans un SGBDD définit une vue conceptuel sur la BD entière. Une autre caractéristique très importante qui définit les systèmes multi-BDs est que leur schéma conceptuel global est conçu à partir des schémas conceptuels locaux vers le schéma global c-à-d sa conception suit une démarche ascendante « buttom-up ».

4.6. Conception d'une BDD

Deux approches sont adoptées dans la conception des BDDs à savoir : la conception descendante « Top down design » et la conception ascendante « Bottom up design ».

4.6.1. Conception descendante

La conception descendante est adaptée aux SGBDs homogènes et intégrés dans laquelle un schéma conceptuel global est établit et, par la suite, distribué en schémas conceptuels locaux correspondant aux différents sites. La distribution se fait en deux étapes : (i) La fragmentation et (ii) L'allocation des fragments sur les sites (Figure 14).

La démarche de la conception descendante d'une BDD telle qu'elle est schématisée dans la figure 14 peut être décrite comme suit :

- Concevoir un schéma conceptuel global de la BDD.
- Etablir un schéma de fragmentation décrivant comment la BD est partitionnée en fragments. L'intersection de ces derniers doit être vide et leur réunion doit reconstituer la BD.
- Etablir un schéma d'allocation décrivant comment les différents fragments sont distribués sur les différents sites.
- Etablir un schéma local pour chaque site couvrant ses fragments.
- Etablir un schéma interne décrivant l'implémentation des données des fragments sur les supports physiques de stockage.

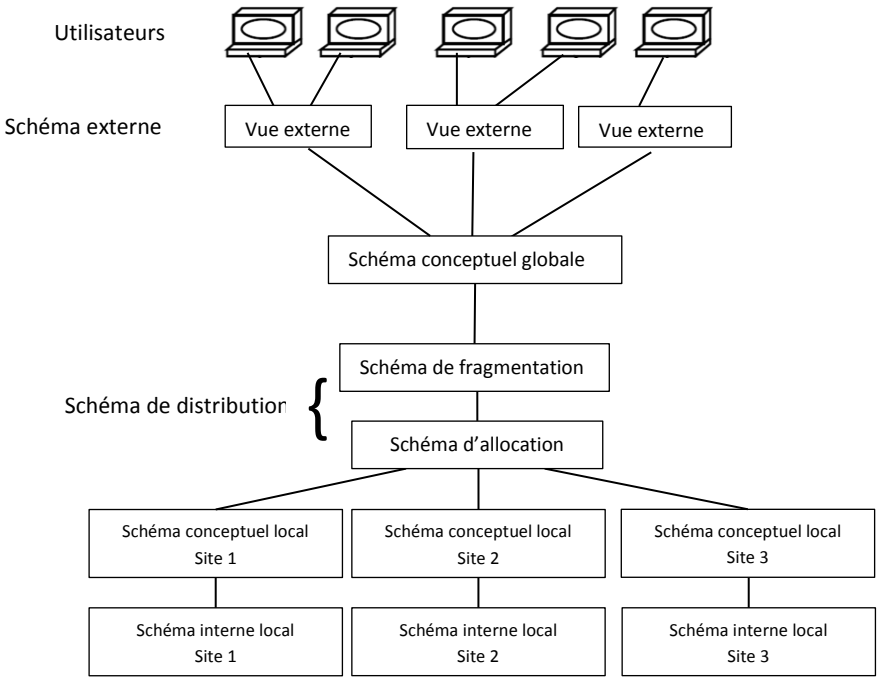


Figure 14: Démarche de la conception descendante d'une BDD

4.6.1.1. Fragmentation

La fragmentation consiste à la décomposition d'une BD en un ensemble de sous BDs. En divisant une relation en plusieurs fragments, la fragmentation permet l'exécution simultanée de plusieurs transactions et l'exécution parallèle d'une requête en la divisant en un ensemble de sous-requêtes. Une fragmentation optimale est celle qui minimise le temps d'exécution des applications utilisateurs qui s'exécutent sur ces fragments.

La fragmentation peut être de type horizontal comme elle peut être de type vertical. Il est possible d'obtenir une fragmentation hybride en imbriquant les deux types.

Afin d'éviter un changement sémantique de la BD, la fragmentation doit respecter trois règles importantes (Desfontain 2000):

(i) **La complétude :** assure que les données d'une relation soient relatées dans les fragments sans aucune perte. Les données représentent les tuples d'une table dans le cas d'une fragmentation horizontale alors que dans le cas d'une fragmentation verticale, elles représentent les attributs d'une table.

(ii) **La reconstruction :** assure que toute relation décomposée en fragments peut être reconstruite à l'aide d'une opération à partir de ses fragments.

(iii) **La disjonction :** assure que toute donnée qui appartienne à un fragment de la relation n'appartienne à aucun autre fragment. Dans le cas d'une fragmentation horizontale, les fragments horizontaux

sont disjoints. Dans le cas d'une fragmentation verticale, les attributs non primaires sont disjoints à l'exception de la clé primaire qui doit être répétée dans les fragments pour assurer la reconstruction.

4.6.1.1.1. Fragmentation horizontale

La fragmentation horizontale consiste à diviser une table en ensembles de tuples. Pour cela il est nécessaire de déterminer les prédicats (les conditions de sélection) les plus important(e)s dans les requêtes des utilisateurs. La règle 80/20 (Wiederhold 1982) peut être utilisée comme guide dans la détermination de ces prédicats.

ETUD

ID	NOM	AGE	GENRE
1	S.Boukathem	18	F
2	A.Mabrouk	21	M
3	M.Safir	20	M
4	H.Boughadou	21	F
5	M.Bouhadi	22	M

Tableau 3: Table des étudiants

Soit une relation $R(A_1, A_2, \ldots, A_n)$, où A_i est un attribut défini sur un domaine D_i, un prédicat simple p_j défini sur R a la forme suivante :

$p_j : A_i \; \theta \; Value$

où $\theta \in \{=, <, \neq, \leq, >, \geq\}$ et $Value \in D_i$

AGE ≤ 200000 est un exemple de prédicat simple défini sur la relation ETUD (Tableau 4**)**.

Un ensemble de prédicats simples P_{ij} est noté par Pr_i

La combinaison avec conjonction de plusieurs prédicats simples est appelée *minterm*.

Soit $Pr_i = \{P_{i1}, P_{i2}, \ldots, P_{im}\}$ un ensemble de prédicats simples sur la relation R_i, le *minterm* $M_i = \{m_{i1}, m_{i2}, \ldots, m_{iz}\}$ est défini comme suit :

$$M_i = \left\{ m_{ij} \backslash m_{ij} = \bigwedge_{p_{ij} \in Pr_i} P_{ik}^* \right\}, 1 \leq k \leq m, \; 1 \leq j \leq z$$

Où $p_{ik}^* = p_{ik} \quad ou \quad p_{ik}^* = \neg p_{ik}$ c'est-à-dire, chaque prédicat peut être décrit dans sa forme positive ou dans sa forme négative.

Deux types de fragmentation horizontale sont définis : fragmentation horizontale primaire et fragmentation horizontale dérivée.

a) Fragmentation horizontale primaire

La fragmentation horizontale primaire est définie par une opération de sélection sur une relation. La reconstruction de la relation fragmentée est reconstruite par une opération d'union.

Soit une relation *R*, ses fragments horizontaux sont donnés par la formule suivante : $R_i = \sigma_{F_i}(R),\ 1 \leq i \leq w$

Où F_i peut être un prédicat simple comme elle peut être un *minterm*.

On peut par exemple, fragmenter la relation ETUD en deux fragments horizontaux comme suit $ETUD_1$ et $ETUD_2$ (Figure 15):

$ETUD_1 = \sigma_{AGE \leq 20}$ (ETUD)

$ETUD_2 = \sigma_{AGE > 20}$ (ETUD)

$ETUD_1$

ID	NOM	AGE	GENRE
1	S.Boukathem	18	F
3	M.Safir	20	M

$ETUD_2$

ID	NOM	AGE	GENRE
2	A.Mabrouk	21	M
4	H.Boughadou	21	F
5	M.Bouhadi	22	M

Figure 15: Fragmentation horizontale primaire de la relation ETUD

b) Fragmentation horizontale dérivée

La fragmentation horizontale dérivée ou appelée aussi par voisinage est définie sur une relation fils d'un lien selon une opération de sélection sur la relation mère du lien.

Soit *R* et *S*, respectivement la relation mère et la relation fils d'un lien.

Les fragments drivés de la relation *R* sont donnés par la formule suivante :

$R_i = R\ xS_i, 1 \leq i \leq w$

Où *w* est le nombre maximum des fragments qui peuvent être définis sur *R* et $S_i = \sigma_{F_i}(S)$ où F_i sont les prédicats de la fragmentation horizontale primaire de la relation *S*.

Exemple : Soit la relation MOY définie dans le tableau 5:

MOY

ID	MOYENNE
1	14
2	9
3	11
4	8
5	15

Tableau 4: Table des moyennes des étudiants

Les fragments $ETUD_1$, ETUD2 de la relation ETUD peuvent être définis comme suit :

$ETUD_1 = ETUD \times MOY_1$

$ETUD_2 = ETUD \times MOY_2$

Où

$MOY_1 = \sigma_{MOY \geq 10}(MOY)$

$MOY_2 = \sigma_{MOY < 10}(MOY)$

Le résultat de la fragmentation est décrit dans la Figure 16 :

$ETUD_1$

ID	NOM	AGE	GENRE
1	S.Boukathem	18	F
3	M.Safir	20	M
5	M.Bouhadi	22	M

$ETUD_2$

ID	NOM	AGE	GENRE
2	A.Mabrouk	21	M
4	H.Boughadou	21	F

Figure 16 : Fragmentation horizontale dérivée de la relation ETUD

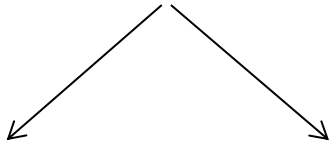
4.6.1.1.2. Fragmentation verticale

La fragmentation verticale utilise la projection pour produire des fragments R_1, R_2, …, R_n où chacun d'eux contient un sous-ensemble d'attributs de la relation *R* ainsi que sa clé primaire (Figure 17). La relation fragmentée *R* est reconstruite par opération de jointure.

Il est évident qu'à cause du nombre important des combinaisons possibles que la fragmentation verticale est plus compliquée que la fragmentation horizontale.

ETUD

ID	NOM	AGE	GENRE
1	S.Boukathem	18	F
2	A.Mabrouk	21	M
3	M.Safir	20	M
4	H.Boughadou	21	F
5	M.Bouhadi	22	M



$ETUD_1$

ID	NOM
1	S.Boukathem
2	A.Mabrouk
3	M.Safir
4	H.Boughadou
5	M.Bouhadi

$ETUD_2$

ID	AGE	GENRE
1	18	F
2	21	M
3	20	M
4	21	F
5	22	M

Figure 17: Fragmentation verticale de la relation ETUD

4.6.1.1.3. Fragmentation hybride

Dans les cas où une fragmentation horizontale ou verticale ne soit pas satisfaisante par rapport aux besoins des applications utilisateurs, une fragmentation hybride est appliquée. Appelée aussi mixte ou imbriquée, la fragmentation hybride consiste à effectuer une fragmentation horizontale suivie par une fragmentation verticale ou vice versa (Figure 18).

Par conséquent, une fragmentation hybride est appliquée sur une relation *R* en combinant des sélections et des projections. Tandis que la reconstruction de la relation *R* est réalisée en combinant des unions et des jointures.

La figure 18 montre comment la relation ETUD est fragmentée horizontalement en deux fragments $ETUD_1$ et $ETUD_2$ où une sélection est appliquée en utilisant respectivement les prédicats AGE ≤ 20 et AGE > 20. Ensuite, le fragment $ETUD_1$ est fragmenté verticalement en deux fragments $ETUD_{11}$ et $ETUD_{12}$. De même, le fragment $ETUD_2$ est fragmenté en deux fragments $ETUD_{21}$, $ETUD_{22}$.

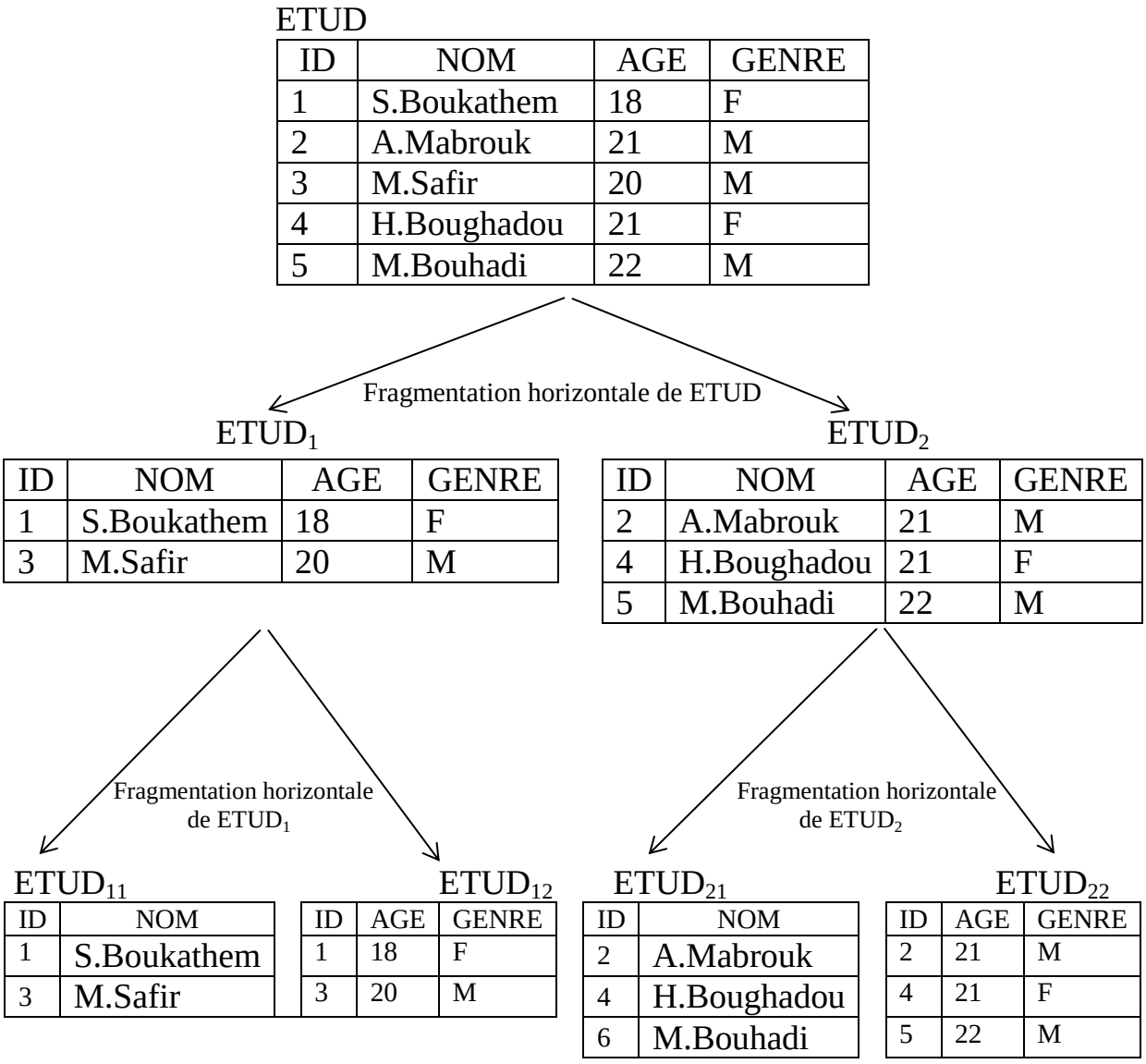


Figure 18: Fragmentation hybride de la relation ETUD

4.6.1.2. Allocation

Une fois la fragmentation est achevée, l'allocation s'impose par le fait qu'il faut affecter les fragments de données aux différents sites. L'affectation des fragments est faite en fonction de l'origine prévue des requêtes qui ont servi à la fragmentation. Par conséquent, le fait de placer les requêtes dans les sites où elles sont le plus utilisées peut constituer un moyen efficace d'optimisation dans le sens où cette manière de placement minimise le transfert de données entre les sites.

Ces recommandations doivent être respectées dans la définition du schéma d'allocation en donnant la priorité respectivement aux origines des requêtes de sélection et de mise à jour des données.

4.6.2. Conception ascendante

La conception ascendante « buttom-up » est adaptée aux systèmes de multi-DBs où l'objectif d'intégrer des bases de données existantes en une seule base de données est visé. En d'autres termes, la conception ascendante vise à intégrer les schémas conceptuels locaux existants en un seul schéma conceptuel global.

L'intégration peut être physique ou logique (Jhingran 2002). Quand elle est physique on parle des systèmes Data Warehouses où la base de données intégrée est matérialisée. Dans le cas logique, on parle des systèmes Middleware où le schéma conceptuel global reste entièrement virtuel et n'est pas matérialisé.

4.7. Réplication

La réplication consiste à stocker une relation ou un fragment de relation en plusieurs copies. Par exemple, si une relation R est fragmentée en R1, R2 et R3. Le fragment R1 peut avoir une seule copie, le fragment R2 peut être répliqué sur deux autres sites et le fragment R3 peut être répliqué sur tous les sites (Özsu 2011).

Une BD peut être entièrement répliquée sur chaque site comme elle peut être partiellement répliquée où les fragments sont distribués sur des sites de manière à ce que des copies d'un fragment soient stockées sur plusieurs sites.

4.7.1. Objectifs de la réplication

Beaucoup de problèmes liés à la distribution des données dans une BDD trouvent leur solution dans la réplication, c'est pour cela que les BDDs sont généralement répliquées.

Effectivement, par la réplication il est voulu atteindre les objectifs suivants :

- **Résistance aux pannes :** puisque les données sont copiées sur plusieurs sites, elles peuvent être accessibles lorsque certains sites sont en panne.
- **Allègement du trafic réseau :** la multiplication des copies de données permet de les rapprocher de leur point d'accès ce qui améliore le temps de réponse.
- **Evolutivité :** La réplication permet de soutenir le développement des systèmes en termes de nombre de sites en garantissant des temps de réponse acceptables.

4.7.2. Types de réplication

Le choix de l'endroit où les mises à jour sont d'abord effectuées détermine le type de la réplication.

- **Réplication centralisée :** La réplication est centralisée lorsque les mises à jour sont d'abord effectuées dans un site maître appelé primaire.

  La réplication centralisée à son tour peut être **(i) synchrone** lorsque les mises à jour sont propagées vers les sites en temps réel, comme elle peut être **(ii) asynchrone** lorsque les mises à jour sont propagées en différé.

- **Réplication distribuée :** Contrairement à la réplication centralisée, une réplication est dite distribuée lorsque les mises à jour sont autorisées sur toute réplication. De même, une réplication distribuée peut être synchrone comme elle peut être asynchrone.

4.8. Traitement et optimisation des requêtes

En plus du problème d'optimisation des requêtes qu'est un problème complexe dans les BDs centralisés, d'autres facteurs de complication s'ajoutent dans les BDDs notamment quand-il s'agit d'exécuter des requêtes sur des relations fragmentées et/ou distribuées induisant des frais de communication et des temps de réponse élevés.

4.8.1. Complexité et mesures

Dans les BDs centralisées, la complexité d'une requête est déterminée seulement par les facteurs **(i) d'Entrées/ Sorties sur les disques** qui représente le coût d'accès aux données et **(ii) le coût CPU** qui représente le coût de traitement des données pour l'exécution des opérations algébriques (jointures, sélections,…). Alors que dans le contexte distribué, il faut ajouter à ceux-là le facteur de **(iii) communication sur le réseau** qui représente le temps nécessaire pour le transfert de données entre les sites impliqués dans l'exécution de la requête.

Effectivement, dans le contexte distribué, une requête est transformée du niveau BDD (calcul relationnel) à un niveau BDs locales (algèbre relationnelle). Par conséquent, une requête peut être transformée de plusieurs façons mais qui donnent le même résultat à l'exécution. Cela mène à considérer un aspect très important dans le processus de traitement d'une requête. Il s'agit de l'optimisation de la requête qui consiste à retenir la meilleure transformation qui minimise la consommation des ressources dans l'exécution de requête.

A cet effet, deux mesures sont définies pour la consommation des ressources :

- **Coût total :** est la somme de tous les temps engagés dans le traitement de la requête sur différents sites et dans la communication intersites (Le coût est généralement exprimé en termes d'unités de temps).
- **Temps de réponse :** est le temps écoulé pour exécuter la requête qui peut être inférieur du coût total puisque les opérations constituant la requête peuvent être exécutées en parallèles sur différents sites.

4.8.2. Décomposition des requêtes et localisation des données

Dans le processus de traitement d'une requête, l'optimisation de cette dernière est précédée par la transformation de la requête de calcul spécifiée par des relations distribuées (i.e. des relations globales) en une requête algébrique définie sur des fragments de relation. Cette transformation est obtenue par l'application successive de la décomposition de la requête et la localisation des données (Özsu 2011).

- **Décomposition d'une requête :** la décomposition de requête est la même dans les BDs centralisées et distribuées. La décomposition d'une requête doit être constituée des phases ci-dessous mentionnées dont leur application aboutit à une requête sémantiquement correcte et sans redondance.
  - **(i) Normalisation :** Les prédicats de la clause where sont écrits sous la forme normale conjonctive
  - **(ii) Analyse :** L'analyse permet de rejeter la requête qu'il serait impossible de traiter (rejet pour mouvais typage des prédicats, ex. nom > 1).
  - (iii) **Elimination des redondances :** Les redondances sont éliminées par l'application des règles d'idempotence (ex. $p \wedge p \Leftrightarrow p$, $p \vee p \Leftrightarrow p$, $p_1 \wedge (p_1 \vee p_2) \Leftrightarrow p_1$, $p_1 \vee (p_1 \wedge p_2) \Leftrightarrow p_1$).
  - **(iv) Réécriture :** La décomposition s'achève par la réécriture de la requête sous forme d'algèbre relationnelle.

- **Localisation des données distribuées:** Le rôle de cette opération est de prendre en compte la distribution des données en translatant la requête algébrique des relations globales en requête algébrique exprimée par les fragments physiques en utilisant les informations enregistrées dans le schéma de fragmentation.

4.8.3. Optimisation des requêtes

La requête résultante de la décomposition et la localisation peut être exécutée systématiquement en ajoutant les primitives de la communication. Il est même possible d'obtenir plusieurs plans d'exécution équivalents de la requête en modifiant l'ordre des opérations qui la constituent. A ce stade, l'optimisation de la requête intervienne en cherchant l'ordre optimal d'opérations de la requête. Le choix du plan optimal est dépendant du résultat de la comparaison entre les coûts d'exécution (coût E/S + coût CPU + coût de communication) de tous les plans d'exécution possibles de la requête.

Dans ce qui suit, nous présentons quatre approches d'optimisation de requêtes à savoir l'approche dynamique, statique, basée semi-jointure et hybride (Özsu 2011).

(i) **Approche dynamique :** (ex. l'algorithme INGERS). L'idée est de minimiser la combinaison entre le temps de communication et le temps de réponse en ignorant le coût de transmission des données au site de résultats.
L'algorithme prend en compte la topologie du réseau. Il est alimenté en entrée par les tuples du calcul relationnel exprimés sous la forme normale conjonctive ainsi que les informations du schéma (type de réseau, l'emplacement et la taille de chaque fragment). L'algorithme est exécuté sur le site maître où la requête est initialisée.

(ii) Approche statique : (ex. l'algorithme R*). Se base sur la sélection du meilleur plan d'exécution de la requête. Le site maître où la requête est initialisée prend toutes les décisions intersites telles que la sélection des sites d'exécution, les fragments ainsi que les méthodes de transfert des données. Alors que les autres sites impliqués dans l'exécution de la requête prennent le reste des décisions locales telles que celles qui concernent l'ordre des jointures locales et la génération des plans d'accès locaux. La fonction objective inclue le coût des traitements locaux et les coûts de transmission.

(iii) Approche basée semi-jointure : (ex. l'algorithme SDD-1). Se base sur la semi-jointure, qu'est en fait une double-jointures, dans le but de minimiser le coût de la communication tout en ignorant le coût de transfert au site final.

(iv) **Approche hybride :** L'approche hybride est proposée dans le but de trouver une solution au problème d'estimation des coûts et de

comparaison des plans d'exécution au moment de la compilation ainsi que le problème de non disponibilité des sites au moment de l'exécution. Les algorithmes de cette approche procèdent généralement en deux étapes

- Au moment de la compilation, génèrent un plan statique qui spécifie l'ordre des opérations et les méthodes d'accès sans considérer l'emplacement de stockage des relations.
- Au démarrage, génèrent un plan d'exécution en effectuant une sélection de site et de copie et en attribuant les opérations aux sites.

4.9. Gestion des transactions distribuées

Une transaction est une séquence d'opérations atomiques de lecture et d'écriture sur une base de données. Une transaction peut être considérée comme un programme de requêtes d'accès à la base de données délimitée par **Begin_transaction** et **end**. Dans ce cas, une transaction peut être normalement terminée par un *commit*, comme elle peut être interrompue par un *abort*. Quand une transaction est annulée et l'état de la base de données précédent son exécution est restitué on dit que la transaction est interrompue par un *rollback*.

4.9.1. Propriétés ACID

Afin de garantir la cohérence et la fiabilité à une transaction, cette dernière doit respecter les exigences recommandées par les propriétés suivantes : **A**tomicité, **C**ohérence, **I**solation et **D**urabilité (ACID).

- **Atomicité :** Une transaction doit être traitée comme une seule unité d'opération. Par conséquent, les opérations de la transaction sont soit toutes exécutées ou aucune d'entre elles.
- **Cohérence :** Une transaction doit ramener une base de données d'un état cohérent à un autre cohérent. La cohérence est bien sûre vérifiée par les règles d'intégrité.
- **Isolation :** Une transaction doit voir une base de données en un état cohérent et ne doit révéler ses résultats à une autre transaction concurrente qu'après sa validation.
- **Durabilité :** Quand une transaction est validée, ses résultats doivent être permanents et ne peuvent être effacés de la base de données.

4.9.2. Contrôle de concurrence

Le contrôle de concurrence regroupe les méthodes qui garantissent un comportement correct des transactions notamment leur isolation. L'objectif

principal du contrôle de concurrence est donc d'assurer la cohérence de la base de données dans un environnement distribué à multiutilisateurs où plusieurs utilisateurs peuvent accéder simultanément à la base de données. Cela est assuré par ce qu'on appelle la sérialisabilité qui consiste à contrôler l'exécution simultanée des transactions de manière à générer le même résultat qu'une exécution séquentielle.

On peut distinguer deux approches dans la gestion de l'isolation des transactions : approche d'isolation multiversion et approche à verrouillage.

4.9.2.1. Approche d'isolation multiversion

Cette approche consiste à utiliser plusieurs versions de données pour fournir des lectures non bloquantes. En d'autres termes, chaque fois qu'une transaction écrit un élément de données, elle crée une nouvelle version de cet élément au lieu de l'écraser (Liu 2009).

Implémenté par Berenson et al, l'algorithme d'isolation snapshot est un exemple d'isolation multiversion. Snapshot consiste à ce que lors de l'exécution d'une transaction T, qui à chaque lecture d'un élément de donnée x, elle ne voit pas nécessairement la dernière valeur écrite à T, mais elle voit la version de x qui a été produite par la dernière transaction qui l'a validé parmi les transactions qui se sont engagées avant le début de la transaction T et qui ont également modifié x (Berenson 1995). Snapshot est implémenté par Oracle RDBMS, PostgreSQL, SQL Server 2005 et Oracle Berkeley DB (Cahill 2008).

4.9.2.2. Approche à verrouillage

L'approche à verrouillage est la technique la plus utilisée pour sérialiser les transactions. Elle est basée sur l'utilisation des verrous en lecture et écriture qui garantissent qu'un élément de données soit accessible par une seule opération à la fois.

Les algorithmes de confirmation en deux phases « two-phase locking » (2PL) sont un exemple de l'approche à verrouillage. Les 2PLs exigent qu'aucune transaction ne doive demander un verrou après avoir libérer l'un de ses verrous. Le protocole 2PL exploite deux types de sites : (i) un site coordinateur où la transaction est initialisée, (ii) des sites participants où les éléments de données sont stockés et sur lesquels les opérations sont exécutées.

Toutefois, le protocole 2PL peut être centralisé (C2PL) lorsque le site coordinateur détient seul la gestion des verrous, comme il peut être distribué (D2PL) lorsque la gestion des verrous est déléguée à tous les sites.

Les algorithmes 2PLs exécutent les transactions en deux phases : (i) phase de croissance où chaque transaction obtienne des verrous et accède aux éléments de données, (ii) phase de validation au cours de laquelle la transaction libère des verrous.

4.10. En résumé

Suite à l'augmentation phénoménale des données et leur dispersion dans les différents systèmes dans les organisations, le besoin à une gestion de ces données distribuées s'est manifesté. La définition de l'essentiel des concepts sur lesquels sont basées les BDDs tels que la définition des BDDs et des SGBDDs ont été détaillée et discutées. Les avantages de la gestion des données distribuées ont été dressés. Après avoir expliqué les architectures des principaux modèles des BDDs, la conception d'une BDD basée sur l'approche descendante et celle ascendante a été détaillée et discutée tout en focalisant sur les principaux concepts de la conception descendante à savoir la fragmentation et l'allocation des données. Finalement, les concepts de la réplication ainsi que le traitement et l'optimisation des requêtes ont été discutés.

4.11. Exercices

Exercice 1 :

Reconsidérons la relation ETUD du tableau 4, et soit les deux prédicats p_1 et p_2 tels que : p_1 : AGE <20 et p_2 : AGE>20.

a) Faites une fragmentation horizontale de la relation selon $\{p_1, p_2\}$.

b) Expliquez pourquoi le résultat de la fragmentation ($ETUD_1$, $ETUD_2$) ne remplit pas les règles de la fragmentation correcte.

c) Modifiez donc les prédicats p_1 et p_2 pour qu'ils puissent partitionner la relation ETUD en respectant les règles de la fragmentation correcte. Montrez que le résultat obtenu après modification respecte les propriétés de complétude, de reconstruction et de disjonction.

Exercice 2 : [3]

Une entreprise possède plusieurs entrepôts répartis géographiquement, stockant et vendant des produits. Considérez le schéma de base de données partiel suivant:

ARTICLE (ID, NomArticle, Prix,…)
STOCK (ID, entrepot, quantite,…)
CLIENT (ID, NomClient, Adresse, Credit,…)
COMMANDE_CLIENT (ID, entrepôt, solde,…)
COMMANDE (ID, Entrepot, IDClient, Date)
COMMANDE_EN_LIGNE (ID, IDArticle, Montant,…)

La base de données contient les relations avec des informations sur les produits (ARTICLE contient les informations générales sur les produits, STOCK contient, pour chaque produit et pour chaque entrepôt, le nombre de pièces actuellement en stock). De plus, la base de données stocke des informations sur les clients, par exemple, des informations générales sur les clients sont stockées dans la table CLIENT. Les principales activités concernant les clients sont la commande de produits, le paiement des factures et les demandes d'informations générales. Il existe plusieurs tables pour enregistrer les commandes d'un client. Chaque commande est regroupée dans les tables COMMANDE et COMMANDE_EN_LIGNE. Pour chaque commande/achat, une entrée existe dans la table de commande, ayant un ID, indiquant l'identifiant du client, l'entrepôt auquel la commande a été soumise, la date de la commande, etc. Un client peut avoir plusieurs commandes en attente à un entrepôt. Au sein de chaque commande, plusieurs produits peuvent être commandés. COMMANDE_EN_LIGNE contient une entrée pour chaque produit de la commande, qui peut inclure un ou plusieurs produits. COMMANDE_CLIENT est une table récapitulative qui liste, pour chaque client et pour chaque entrepôt, la somme de toutes les commandes existantes.

d) L'entreprise possède un service à la clientèle composé de plusieurs employés qui reçoivent les commandes et les paiements des clients, interrogent les données des clients locaux pour rédiger des factures ou enregistrer les chèques de paie, etc. En outre, ils répondent à tout type de demandes que les clients pourraient avoir. Par exemple, la commande de produits modifie (met à jour/insère) les tables

[3] Cet exercice est cité dans (Ramakrishnan 2003)

COMMANDE_CLIENT, COMMANDE, COMMANDE_EN_LIGNE et STOCK. Pour être flexible, chaque employé doit pouvoir travailler avec n'importe lequel des clients. La charge de travail est estimée à 80% de requêtes et 20% de mises à jour.

La charge de travail étant orientée requêtes, la direction a décidé de construire un cluster de PCs équipés chacun de sa propre base de données pour accélérer les requêtes grâce à un accès local rapide. Comment répliqueriez-vous les données à cette fin? Quel type de réplication utilisez-vous pour maintenir la cohérence des données?

e) La direction de la société doit décider chaque trimestre fiscal de ses offres de produits et de ses stratégies de vente. À cette fin, elle doit constamment observer et analyser les ventes des différents produits dans les différents entrepôts ainsi que le comportement des consommateurs. Comment répliqueriez-vous les données à cette fin? Quel type de réplique utiliseriez-vous pour maintenir la cohérence des données

4.12. Références du chapitre

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

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

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

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

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

[Özsu 2011] M.T. Özsu, P.Valduriez, Principles of Distributed Database Systems, 3ème Edition, ISBN 10.1007/978-1-4419-8833-8_5, Springer science+Business Media, LLC (2011).

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

[Schollmeier 2002] R. Schollmeier, A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications, Proceedings of the First International Conference on Peer-to-Peer Computing, IEEE (2002).

[Sun 2009] Distributed Application Architecture, Sun Microsystem, http://java.sun.com/developer/Books/jdbc/ch07.pdf, visualisé le 03/12/2019.

[Wiederhold 1982] G. Wiederhold, Database Design, McGraw-Hill, 2ème edition, (1982).

CHAPITRE 5

Data Warehouse

Depuis l'avènement des premières versions des bases de données, ces dernières ont été centrées sur le traitement opérationnel, généralement transactionnel, répondant aux besoins d'une certaine communauté d'utilisateurs. Pendant ces dernières années, d'autres besoins se sont fait sentir auprès d'une autre communauté d'utilisateurs, généralement des décideurs dans leurs organisations, qui traitent avec l'information et l'analyse. Le traitement analytique examine de grande quantité de données pour détecter les tendances au lieu de regarder un ou deux enregistrements de données comme c'est le cas dans le traitement opérationnel. Dans ce contexte, la notion de Data Warehouse (DW) a émergé dans le sens de satisfaire aux besoins des décideurs dans le processus décisionnel. Dans un DW, toutes les informations, qu'elles que soient et où qu'elles soient, doivent être considérées et exploitées du moment qu'elles soient en relation avec l'organisation en question. C'est pour cela que, tous les modèles de BD (BDR, BDOO, BDRO, BDD) qu'on a déjà vus dans les chapitres précédents peuvent être utilisés à tous les niveaux d'un DW.

5.1. Motivation et objectifs des DWs

Beaucoup de besoins relatives à la gestion de l'information se font sentir auprès des acteurs des organisations (entreprises) et pour lesquelles cherchent des solutions. Parmi ces besoins on peut citer les suivants :

- Ils collectent beaucoup de données auxquelles ils ne peuvent pas y accéder.
- Ils ont besoin de voir ce qu'est important.
- Ils perdent beaucoup de temps à savoir qui a les bons chiffres au lieu de concentrer les efforts sur la prise de décision.
- Ils veulent utiliser des informations dans la prise de décision.

Ces besoins mènent tous à exiger des objectifs pour les DWs dont nous citons les plus importants d'entre eux (Kimball 2005) (Kimball 2013):

- **Accessibilité des informations :**
  Le contenu du DW doit être compréhensible et la navigation dans son contenu doit être facile et rapide.

- **Cohérence des informations :**
  Les informations étant rassemblées à partir de différentes sources, elles doivent être nettoyées et cohérentes dans le sens où si deux unités de mesure ont le même nom elles doivent alors signifier la même chose et si à l'inverse deux unités de mesure n'ont pas la même signification elles doivent porter des noms différents.

- **Adaptation au changement :**
  Un DW doit être capable d'assumer un changement continu sans pour autant permettre un bouleversement des données ou des technologies existantes.

- **Présentation des informations en temps opportun :**
  Un DW doit être capable de rendre les informations exploitables en un temps raisonnable pour aider les utilisateurs dans la prise de décision.

- **Sécurité des informations :**
  Un DW est souvent amené à contenir des informations confidentielles de l'organisation ce qui lui inflige la responsabilité, non seulement, de sécuriser les données mais aussi d'offrir aux gestionnaires d'avoir une visibilité sur leurs utilisations.

- **Fiabilité des informations :**
  Afin d'assumer le rôle d'un système d'aide à la décision, un DW doit avoir les bonnes données pour une meilleure décision.

5.2. Définition d'un Data Warehouse

Le Data Warehouse est une collection de données orientées sujet, intégrées, non volatiles, historisées, variant dans le temps pour le support d'un processus d'aide à la décision (Inmon 2005).

Nous expliquons dans ce qui suit ces caractéristiques (Vaisman 2014).

Orientées sujets : cela veut dire que le DW se concentre sur les besoins analytiques des différents secteurs d'une organisation.

Contrairement aux systèmes opérationnels qui sont orientés programmes, les Data Warehouses sont orientés sujets. Pour une compagnie d'assurance par exemple, les applications peuvent concerner le traitement de l'automobile, de la vie, de la santé et les accidents alors que les principaux sujets peuvent être les clients, les polices d'assurance, les primes et les réclamations.

Intégrées : signifie relier les données obtenues à partir de plusieurs systèmes opérationnels en résolvant les problèmes dus aux différences de définition et de contenu des données telles que les différences de format et de codification des données, les synonymes (champs avec des noms différents mais les mêmes données), homonymes (champs avec le même nom mais différentes significations), la multiplicité des occurrences.

Non volatiles : les données ne sont ni modifiables ni supprimables ce qui leurs garantit une durée de vie beaucoup plus longue par rapport aux données au niveau opérationnel. Le seul changement provient du chargement de nouvelles données

Variant dans le temps : signifie que plusieurs valeurs sont enregistrées pour la même information ainsi que la date à laquelle les modifications ont été apportées à ces valeurs. Cela veut dire qu'un DW montre l'évolution à travers le temps et non seulement les données récentes.

5.3. L'approche Data Warehouse

L'approche du DW repose sur l'idée d'aller de la donnée à l'information. Répondre par exemple à la requête « Quelle impact pourra avoir la situation familiale d'un employé sur son travail supplémentaire? » va nous amener à chercher toutes les données relatives à la situation familiale pendant une période donnée ainsi que celle relatives au travail réalisé par cet employé pendant la même période dans les applications du système et d'essayer de réunir ces

données qui peuvent être dans différents modèles de BDs (BDR, BDOO, BDRO, BDD). Il faut ensuite intégrer ces données (c'est à dire, les nettoyer, les transformer, les combiner, en supprimer les doublons, les purger, les standardiser, mettre leurs dimensions en conformité) afin de générer des informations qui peuvent constituer une réponse à la requête.

Cependant, l'intégration n'est pas le seul obstacle qui complique le processus de la réponse à la requête. La non-historisation des données au niveau des applications du système constitue un autre obstacle à la satisfaction de la requête.

Afin de pouvoir répondre à une telle requête, il est nécessaire d'ajouter un autre type de données qu'est celui des données dérivées. Cela nous mène à distinguer deux types de données : primitives et dérivées.

Les données primitives représentent les données brutes générées au niveau opérationnel alors que les données dérivées représentent les informations générées par intégration au niveau décisionnel. Le tableau 6 montre quelques différences entre les données primitives et les données dérivées.

Données primitives	Données dérivées
- Orientées application - Détaillées - Destinées à la communauté des exécuteurs - Mises à jour - Accessible à une unité à la fois - Sans redondance - Avec structure statiques et contenu variable - Haute probabilité d'accès - Actuelles	- Orientées sujet - Résumées ou calculées - Destinées à la communauté des décideurs - N'est pas mises à jour - Accessible à un ensemble à la fois - Avec redondance - Avec structure flexible - Faible probabilité d'accès - Historisées

Tableau 5 : Différences entre données primitives et données dérivées (Inmon 2005)

Les données primitives et les données dérivées sont tellement différentes (Tableau 6) qu'elles ne peuvent résider dans la même base de données. C'est la raison pour laquelle les données dérivées en plus des données intégrées et historisées sont déposées dans des bases de données séparées des données primitives.

La différence entre les données primitives et les données dérivées mène à considérer des niveaux de base pour les données allant des sources de données

jusqu'à l'exploitation des données. La figure 19 montre comment les données sont reprises à partir de sources de données et ensuite préparées pour être intégré dans un Data Warehouse. Ce dernier, qui couvre toutes les données, peut être divisé en plusieurs sous-ensembles appelés Datamarts et finalement peut exploiter les données par différents outils tels qu'un système de requêtes.

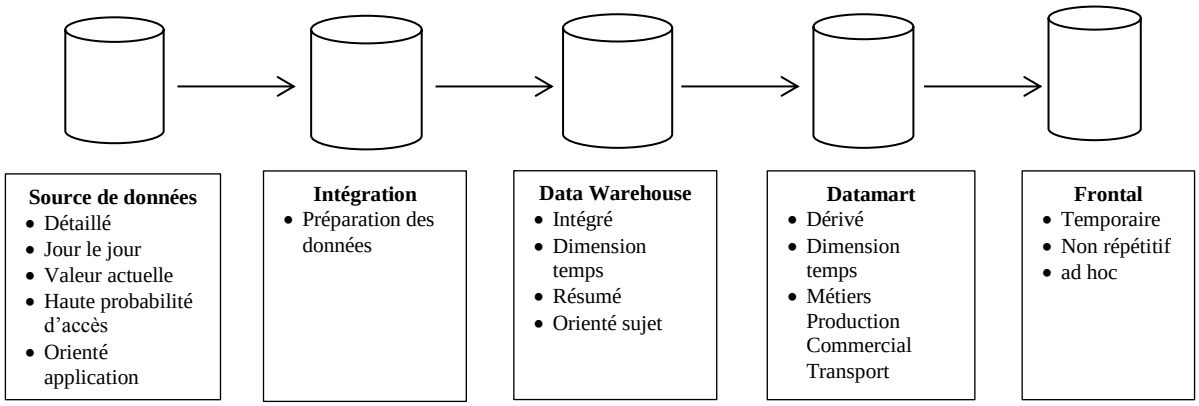


Figure 19 : Niveaux des données d'un DW

La figure 20 montre un exemple sur les niveaux de données d'un employé dans un DW. Le niveau opérationnel contient les données actuelles de l'employé Mohamed Arbaoui dans plusieurs sources. Dans le niveau d'intégration, une préparation des données est réalisée. On peut remarquer par exemple que le prénom de l'employé est écrit en différentes façons (Muhamed, Mohamed et Mohammed) et que l'écriture « Mohamed » est adoptée pour l'intégration. La même chose est faite pour la situation familiale et le nombre d'enfants où respectivement les écritures « SF » et « 3 » sont adoptées. Dans le niveau Data Warehouse, on trouve plusieurs enregistrements sur Mohamed Arbaoui qui montre historiquement sa situation familiale (mariage et nombre d'enfants) sans que les données chevauchent. Si par exemple, la situation familiale de Mohamed Arbaoui change alors un nouvel enregistrement sera créé dans ce niveau. Dans le niveau Datamart (appelé aussi niveau départemental, ou niveau OLAP, ou encore niveau multidimensionnel) on trouve les données relatives à une fonctionnalité spécifique du système telle que les ressources humaines comme le cas de notre exemple où les données sont résumées par mois. Dans le niveau final, en l'occurrence le niveau frontal, les données sont temporaires répondant à une requête qui décèle les tendances sur un aspect donné. Dans notre exemple il est question de mesurer l'impact de la situation familiale sur le travail supplémentaire en heures réalisé par un employé.

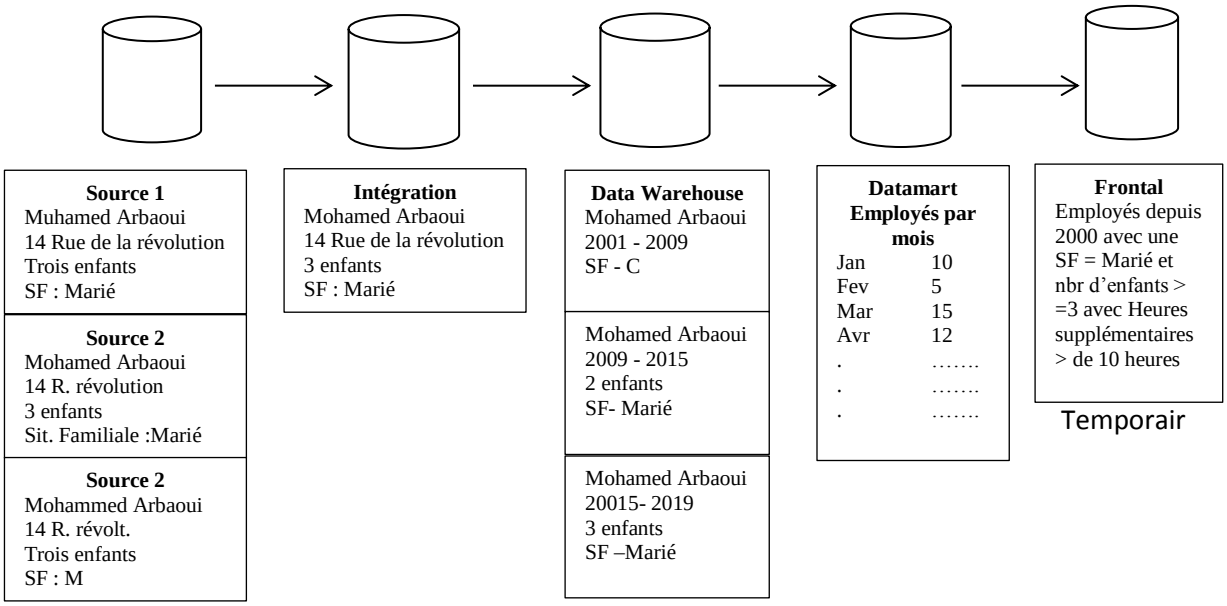


Figure 20 : Les niveaux de données pour un exemple simple : un employé

5.4. Architecture des DWs

Nous avons vu dans la section précédente 5.3 que l'approche DW est basée sur des concepts de base en l'occurrence : données primitives, données orientées sujets, historisées et variant dans le temps, intégration des données, données dérivées ainsi que la séparation entre ces derniers et celles dites primitives.

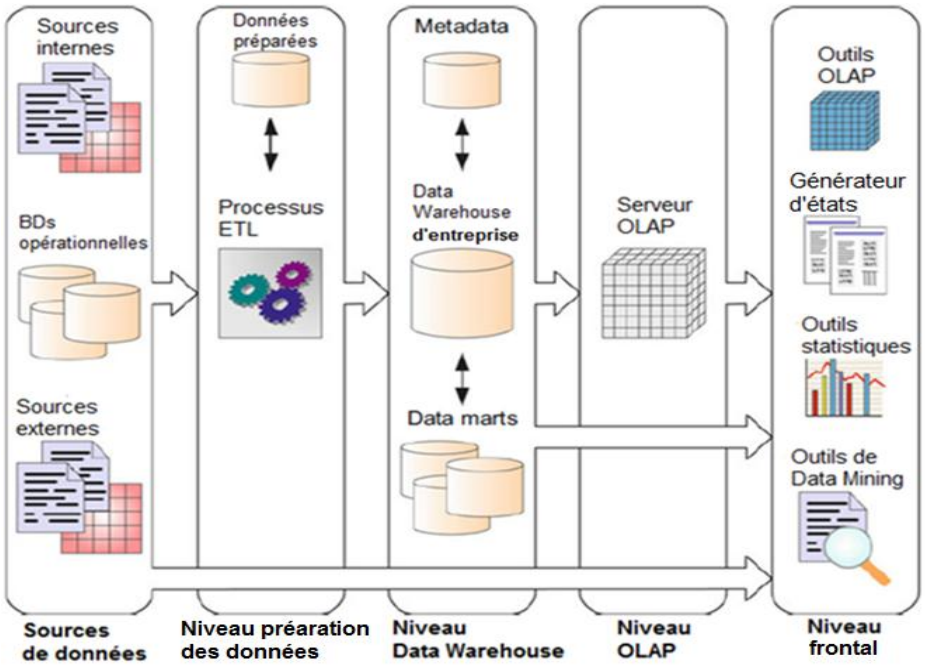


Figure 21: Architecture typique d'un DW (Vaisman 2014)

Sur la base de ces concepts, une architecture globale d'un DW est présentée dans la figure 21. L'architecture est constituée de plusieurs niveaux : le niveau préparation des données, le niveau data Warehouse, le niveau OLAP et finalement le niveau frontal.

5.4.1. Le niveau préparation des données

A ce niveau, les données sont extraites à partir des BDs opérationnelles qui peuvent être des BDRs, BDOOs, BDROs ou BDDs et d'autres sources de données (internes ou externes de l'organisation). Ces données sont ensuite intégrées dans le data Warehouse. La préparation des données est réalisée par les outils (ETL) : Extraction, Transformation et Loading (chargement). Une base de données intermédiaire, généralement relationnelle (BDR), est dédiée à l'exécution de l'extraction et la transformation des données avant leur chargement dans le Data Warehouse. De telle base de données est appelée opérationnel.

a) Extraction

Les données sont extraites à partir de multiple sources de données hétérogènes, internes ou externes à l'organisation. Ces sources peuvent être opérationnelles ou des fichiers avec différentes formats.

b) Transformation

Après l'extraction des données, plusieurs transformations peuvent être appliquée telles que le nettoyage (conversion en format standardisé et suppression des erreurs et des incohérences dans les données telles que les erreurs d'orthographe et les conflits de domaine), combinaison des données provenant de plusieurs sources et suppression des doublons.

c) Loading (chargement)

Les données transformées sont alors chargées dans le DW. Cela inclus également la propagation des mises à jour des sources de données vers le DW à une fréquence qui peut être, selon la politique de l'organisation, mensuelle, plusieurs fois par jour ou presque en temps réel.

5.4.2. Le niveau Data Warehouse

A ce niveau, il y a trois composantes importantes à savoir : un Data Warehouse d'entreprise, un ou plusieurs Datamarts et une Metadata.

Data Warehouse d'entreprise: il est centralisé et couvre toute l'organisation

Datamart: par contre, le Datamart est spécifique à une fonction ou un département dans l'organisation. Le Datamart est chargé par des données (déjà nettoyées et intégrées) du Data Warehouse d'entreprise.

Metadata: les métadonnées sont définies comme des « données sur les données » ou « les données relatives aux données » et peuvent être classées en deux types :

(i) métadonnées de métier qui décrivent la sémantique des données, les règles organisationnelles et les contraintes relatives à ces données.

(ii) métadonnées techniques qui décrivent la manière avec laquelle les données sont structurées et stockées dans le système ainsi que les applications qui les manipulent.

5.4.3. Le niveau OLAP

Ce niveau est composé d'un serveur OLAP fournissant aux utilisateurs professionnels des données multidimensionnelles à partir du Data Warehouse ou des Datamarts.

5.4.4. Le niveau frontal

Comme montré dans la figure 21, afin de permettre aux clients d'exploiter les données du DW, le niveau interface leurs fournit les outils suivants :

- **Outils OLAP :** facilitent la formulation des requêtes ad hoc complexes impliquant de grandes quantités de données.
- **Générateurs d'états :** produisent des rapports en utilisant des requêtes prédéfinies.
- **Outils statistiques :** utilisent des méthodes statistiques dans l'analyse et la visualisation des données.
- **Outils Data Mining :** permettent d'analyser les données dans le but d'établir un modèle ou déceler les tendances afin d'aider à la prédiction.

5.5. Data Warehouse vs Datamart

Un Datamart peut être considéré comme un sous-ensemble d'un Data Warehouse. En effet, un DW couvre entièrement une organisation et contient

des données sur différents sujets alors qu'un Datamart contient des données d'un seul sujet. Un DW est chargé par des données qui proviennent de différentes sources de systèmes opérationnels alors qu'un Datamart est chargé par des données du DW lui-même (Jensen 2010).

Concernant la relation entre les DWs et les Datamart, certains considèrent qu'un DW est une collection de Datamarts ce qui veut dire qu'un DW est construit d'une manière ascendante « Buttom-up » à partir de petits Datamarts alors que le célèbre Inmon proclame l'inverse en considérant que la construction d'un DW est plutôt descendante « Top-down » où les Datamarts sont dérivés à partir des DWs (Inmon 1998).

5.6. Modèle multidimensionnel

Le modèle multidimensionnel est fondamental pour plusieurs systèmes d'aide à la décision tels que le Data Warehouse car le modèle multidimensionnel est bien adapté aux requêtes telles que celles citées par (Golfarelli 2009):

- quel est le montant total des recettes enregistrées l'an dernier par état et par catégorie de produit ?
- quelles commandes maximisent les recettes ?

Il est évident que le fait d'utiliser les langages traditionnels tels que SQL pour exprimer ces types de requêtes est une tâche difficile. Il est aussi évident que l'exécution de telles requêtes sur des BDs opérationnels prendra un temps de réponse inacceptable.

Effectivement, les SGBDs opérationnels ou OLTP ne peuvent assurer une analyse de données basée sur l'exécution de requêtes complexes nécessitant la jointure de beaucoup de tables et l'agrégation de larges volumes de données incluant aussi leurs historisation.

Ces besoins nécessitent ce qu'on appelle les systèmes OLAP qui sont orientés vers l'analyse de données dans le but d'aide à la décision. Les systèmes OLAP se concentrent sur les requêtes analytiques qui exigent l'agrégation, beaucoup de temps, la lecture transversale des enregistrements des tables et de nouvelles techniques d'indexation. Il est évident que le besoin à des solutions qui peuvent supporter ces techniques est plus que pressant. Le Data Warehouse est défini justement dans le but de répondre à ce besoin.

Data Warehouse et OLAP se basent sur le modèle multidimensionnel qui visualise les données dans un espace de n-dimensions appelé les données cube. Ces dernières sont définies par les faits et les dimensions. Dans ce qui suit nous définissons les concepts fondamentaux du modèle multidimensionnel à savoir : les cubes, les dimensions, les faits et les mesures (Jensen 2010) et nous abordons à la fin les principales opérations OLAP.

5.7.1. Cubes

Un cube est une structure de données multidimensionnelles définie dans le but de capturer et analyser des données. Un cube généralise la feuille de calcul d'un tabulaire avec une dimension supérieure à deux. Un cube peut avoir une dimension supérieure à trois contrairement à ce que son nom peut signifier. Pour cette raison le mot hypercube est parfois utilisé à la place de cube. Une collection de cubes constitue une base de données multidimensionnelle ou un Data Warehouse multidimensionnel.

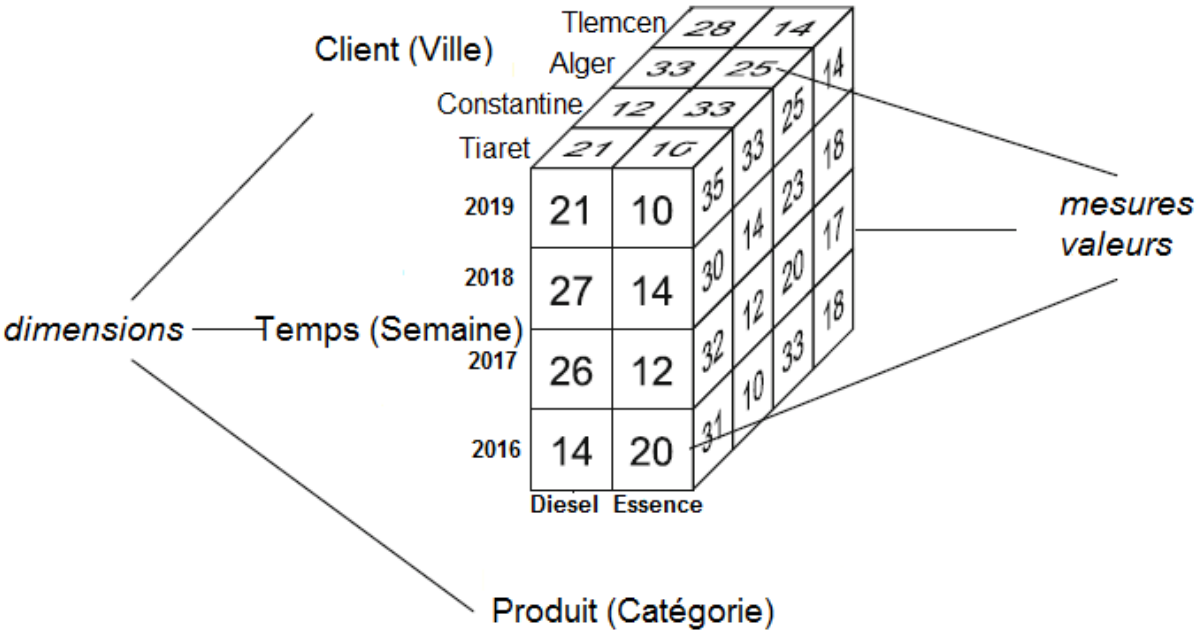


Figure 22: Un cube en trois dimensions pour les données de vente avec les dimensions : *Produit*, *Temps* et *Client*, et une quantité de mesure

Un cube est composé de cellules où chacune d'elles représente une intersection de dimensions. Une cellule quand elle est non vide est appelée un fait et quand elle est vide cela signifie qu'il n'y a aucune information à enregistrer pour les valeurs de dimensions données. Dans la figure 22, il y a une seule mesure à savoir le nombre de ventes pour chaque fait qui représente une combinaison d'une année, de type d'énergie de voiture, et de ville. Par exemple,

le total de ventes dans la ville de 'Tiaret' en 2019 est 31 en additionnant les deux nombres 21 et 10.

5.7.2. Dimensions

Les dimensions sont utilisées pour la sélection des données et leur l'agrégation. Le niveau de la dimension est la granularité (le niveau du détail) à laquelle (auquel) les mesures représentent les dimensions. Dans l'exemple de la figure 22, les ventes sont agrégées aux niveaux *Catégorie*, *Semaine* et *Ville* respectivement pour les dimensions *Produit*, *Temps* et *Client*. Les instances d'une dimension sont appelées membres. Par exemple, *Diesel* et *Essence* sont des membres de la dimension *Produit* au niveau *Catégorie*. Les dimensions ont aussi des attributs associés qui les décrits. Par exemple, la dimension *Produit* doit avoir les attributs *NumProduit* et *Prix* qui ne sont pas visible sur la figure 22.

Les dimensions sont organisées hiérarchiquement en niveaux de détail. Par exemple, pour la dimension *Produit*, la granularité la plus faible est *Produit*, qui est agrégée en *Catégorie* (Figure 23).

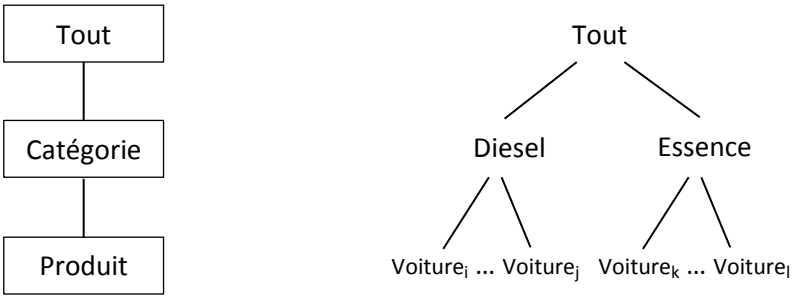


Figure 23: Hiérarchie de la dimension *Client*

La hiérarchie de la dimension est définie dans un Metadata au niveau du cube ou au niveau du modèle multidimensionnel, si la dimension peut être partagée.

5.7.3. Faits

Les faits sont les objets des sujets à être analysés dans le but de comprendre leurs comportements tels que les ventes dans l'exemple de la figure 22. Les faits sont définis par leur combinaison de valeurs des dimensions. Donc, les faits existent dans les cellules non vides. Dans la figure 22, chaque nombre affiché dans la cellule représente donc le nombre de voitures vendues par catégorie, par année et par ville du client.

5.7.4. Mesures

Chaque mesure réalise une fonction d'agrégation qui combine plusieurs valeurs de mesures en une seule à un certain détail. Ainsi, une mesure a deux composants : une propriété numérique et une formule telle qu'une simple sommation (SUM). Dans la figure 22, la mesure *Client* fait une agrégation au niveau de la ville. Si on souhaite avoir une mesure au niveau du pays on doit faire une agrégation de toutes les ventes au niveau du pays en utilisant par exemple l'opération de sommation (SUM).

Cependant, il est indispensable de respecter certaines conditions qui régissent l'agrégation afin d'obtenir des résultats cohérents. Dans (Vaisman 2014), les conditions suivantes sont mentionnées:

- **Disjonction des Instances :** Dans un niveau donné il doit y avoir des ensembles d'instances disjoints. Par exemple, dans la figure 23, un produit ne doit pas figurer dans deux catégories différentes sinon la vente de ce produit sera calculée deux fois.
- **Complétude :** Toutes les instances doivent être liées dans la hiérarchie. Par exemple, dans la figure 23, la hiérarchie du *Produit* doit contenir toutes les voitures sinon le résultat d'agrégation sera incorrect.
- **Exactitude :** cette condition réfère à l'utilisation correcte de la fonction d'agrégation du fait que les mesures peuvent être de types différents. Cela mène à distinguer trois types de mesures: **(i) Mesures additives** qui peuvent être résumées de manière significatives en utilisant l'addition. La mesure de quantité dans le cube (Figure 22) est un exemple de mesure additive, **(ii) Mesures semi-additives** qui peuvent être résumées en utilisant l'addition pour quelques dimensions mais pas pour toutes les dimensions telles que l'exemple des quantités d'inventaire qui ne peuvent être agrégées pour deux périodes différentes, **(iii) Mesures non-additives** qui ne peuvent être résumées en utilisant l'addition pour aucune dimension telles que l'exemple du prix.

5.7.5. Opérations OLAP

Le modèle multidimensionnel permet de visualiser les données à partir de plusieurs points de vue et plusieurs niveaux de détail en exploitant les dimensions et leurs hiérarchies ce qui offre un environnement interactif pour l'analyse des données. Nous détaillons dans ce qui suit deux importantes opérations OLAP à savoir Roll-up et Drill-up et résumons le reste dans le tableau 7.

L'opération Roll-up agrège les mesures le long d'une hiérarchie. Par exemple dans la figure 22, si on souhaite calculer la dimension *Client* au niveau du *Pays* on doit appliquer une opération Roll-up puisque le niveau *Pays* est plus élevé que le niveau *Ville*.

Par contre, L'opération Drill-down (l'opposée de l'opération Roll-up) désagrège les mesures le long d'une hiérarchie pour obtenir une granularité plus fine. Par exemple dans la figure 22, si on souhaite calculer la dimension *Temps* au niveau du Jour on doit appliquer donc une opération Drill-up puisque le niveau *Jour* est inférieur au niveau *Semaine*.

Opération	Objectif
Add mesure	Ajoute une nouvelle mesure à un cube calculée à partir d'autres mesures ou dimension
Opérations d'agrégation	Agrègent les cellules d'un cube
Dice	Conserve les cellules d'un cube qui satisfait une condition booléenne sur les niveaux de dimension, attributs ou mesures
Différence	Supprime les cellules d'un cube se trouvant dans un autre cube. Les deux cubes doivent avoir le même schéma
Drill-across	Fusionne deux cubes qui ont le même schéma et les mêmes instances à l'aide d'une jointure.
Drill-down	Désagrège des mesures le long d'une hiérarchie pour obtenir une granularité plus fine. Elle est l'opposée de Roll-up
Drill-through	Affiche les données des systèmes opérationnels à partir desquels le cube a été dérivé.
Drop mesure	Supprime des mesures d'un cube.
Pivot	Applique une rotation des axes d'un cube pour fournir une présentation différente de ses données.
Recursive roll-up	Applique récursivement une Roll-up le long d'une hiérarchie jusqu'à atteindre le niveau supérieur.
Rename	Renomme un ou plusieurs éléments de schéma d'un cube.
Roll-up	Agrège des mesures le long d'une hiérarchie. Elle est l'opposée de Drill-up
Roll-up*	Notation abrégée pour une séquence d'opérations de Roll-up.
Slice	Supprime une dimension d'un cube en fixant une valeur unique à un niveau de la dimension.
Sort	Ordonne les membres d'une dimension selon une expression.
Union	Combine les cellules de deux cubes qui ont le même schéma mais leurs membres sont disjoints.

Tableau 6: Résumé des opérations OLAP (Vaisman 2014)

5.8. ROLAP, MOLAP et HOLAP

Nous avons vu précédemment que l'analyse interactive des données dans un environnement DW est appelée On-line Analytical Processing (OLAP). Lorsque, les données d'un modèle dimensionnel sont stockées dans une base de données relationnelle, l'analyse est appelée ROLAP pour Relational On-Line Analytical Processing. Les systèmes ROLAP étendent le SQL pour prendre en charge le modèle dimensionnel et les fonctions avancées d'OLAP. Le fait de stocker les données dans des BDRs, cela a beaucoup d'avantages puisque les BDRs sont bien standardisées et offre une large capacité de stockage.

Lorsque les cubes de données d'un DW sont stockées dans des tableaux multidimensionnels, l'analyse de ces données est appelée MOLAP pour Multidimensional On-Line Analytical Processing.

Le principal avantage des MOLAPs par rapport aux ROLAPs est que les opérations multidimensionnelles peuvent être réalisées d'une façon facile et naturelle avec les MOLAPs sans aucune nécessité à de complexes opérations de jointure (Golfarelli 2009).

Quant aux systèmes HOLAP (Hybrid On-Line Annalytical Processing), ils regroupent les avantages des technologies ROLAP et MOLAP en stockant de grandes volumes de données détaillées dans des serveurs ROLAP et les données agrégées dans des serveurs MOLAP (Liu 2009).

5.9. En résumé

Le Data Warehouse a émergé principalement suite à la nécessité des systèmes d'aide à la décision qui s'est manifestée auprès des décideurs des organisations. L'essentielle des objectifs d'un Data Warehouse a été dressé et une définition reflétant l'aspect décisionnel a été donnée. Une architecture en niveaux a été détaillée tout en réservant une attention spéciale au concept très important sur lequel sont basés les Data Warehouse à savoir le modèle dimensionnel.

5.10. Exercices

Exercice 1 :

En reconsidérant l'exemple de la figure 22, répondez aux requêtes suivantes :

a) Faite une opération de Roll-up au niveau pays de la dimension Client.

b) Faite une opération de Drill-down au niveau mois de la dimension Temps.

Exercice 2 : [4]

Un entrepôt de données d'un opérateur téléphonique comprend cinq dimensions: client appelant, client appelé, heure, type d'appel et programme d'appel et trois mesures: nombre d'appels, durée et montant.

Définissez les opérations OLAP à effectuer pour répondre aux requêtes suivantes. Proposez les hiérarchies de dimensions en cas de besoin.

a) Montant total collecté par chaque programme d'appel en 2019.

b) Durée totale des appels passés par les clients Algérois en 2019.

c) Nombre total d'appels en week-end passés par des clients de de la ville Tiaret à des clients de la ville d'Alger en 2019.

d) Durée totale des appels internationaux lancés par des clients en Algérie en 2019.

e) Montant total perçu auprès des clients Algérois inscrits au programme d'entreprise en 2019.

Exercice 3 :

On suppose que la compagnie aérienne AIR ALGERIE a construit un Data Warehouse contenant des informations sur les vols de ses avions. Ce dernier comprend six dimensions, à savoir l'aéroport de départ, l'aéroport d'arrivée, le vol, l'avion, l'heure d'arrivée et l'heure de départ, et trois mesures, à savoir le nombre de passagers, la durée et la distance en kilomètres.

Définissez les opérations OLAP à effectuer ainsi que les hiérarchies de dimensions nécessaires en répondant aux requêtes suivantes.

[4] Cet exercice est adapté à partir d'un exercice proposé dans (Vaisman 2014)

a) Nombre total de kilomètres parcourus par les vols d'Air Algérie en 2019.

b) Nombre total de voyages au départ ou à destination de Bruxelles en 2019.

c) Nombre de passagers des vols Alger-Bruxelles ou Bruxelles-Alger durant 2019.

d) La durée totale des vols Alger-Bruxelles ou Bruxelles-Alger durant 2019.

5.11. Références du chapitre

[Golfarelli 2009] M. Golfarelli, S. Rezi, Data Warehouse design, Modern principles and methodologies, McGraw-Hill companies, (2009).

[Inmon 1998] W. H. Inmon, Data Mart Does Not Equal DataWarehouse, DMReview, (1998).

[Inmon 2005] W.H. Inmon, Building the Data Warehouse, 4ème Edition, Wiley Publishing, Inc, ISBN-13: 978-0-7645-9944-6, (2005).

[Jensen 2010] C.S. Jensen, T.B. Pedersen, C. Thomsen, Multidimensional Databases and Data Warehousing, Morgan & Claypool Publishers, DOI 10.2200/S00299ED1V01Y201009DTM009, (2010).

[Kimball 2005] R. Kimball, L. Reeves, M. Ross, W. Thornthwaite, Le data warehouse: Guide de conduite de projet, Edition Eyrolles, ISBN:2-212-11600-4, (2005).

[Kimball 2013] R. Kimball, M. Ross, The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, 3ème Edition, John Wiley & Sons, Inc, (2013).

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

[Vaisman 2014] A. Vaisman and E. Zimanyi, Data Warehouse Systems, Data-Centric Systems and Applications, DOI:10.1007/978-3-642-54655-6 3, Springer-Verlag Berlin Heidelberg, (2014).

Conclusion

Ce polycopié a été axé sur la notion de base de données où il a été question de définir cette dernière et suivre son évolution. Nous avons évoqué la notion de SI pour un but pédagogique afin de montrer comment les BDs ont été exploitées dans un premier temps pour le stockage et le partage des données et par la suite pour le traitement analytique des données. Cela a été expliqué en discutant les sous-systèmes constituant un SI en l'occurrence : le sous-système opérationnel, système d'information et décisionnel (chapitre 1).

Quant à l'évolution des BDs, nous l'avons expliqué à travers la discussion des modèles de BDs. Effectivement, nous avons vu comment les limites constatées avec les systèmes de gestion des fichiers ont poussé vers l'émergence du modèle relationnel (chapitre 2). Par la suite, l'avènement du concept de l'orienté objet a donné naissance au modèle orienté objet des BDs. Cependant, la complexité de ce dernier à laisser une brèche au modèle relationnel pour s'étendre en modèle relationnel-objet (chapitre 3). Entre temps, le développement rapide des réseaux a imposé la définition du modèle distribué (chapitre 4). Finalement, le besoin à des systèmes d'aide à la décision présenté auprès des décideurs des organisations a montré la nécessité du traitement analytique des données ce qui a donné naissance au modèle Data Warehouse (chapitre 5).