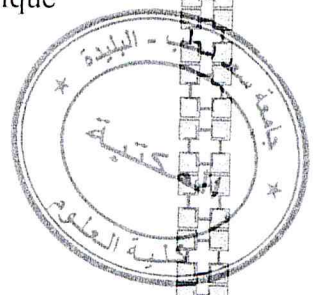
République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique



Université SAAD DAHLED, Blida
USDB.
Faculté des sciences.
Département informatique.

MEMOIRE

Présenté pour l'obtention du diplôme de Master

Spécialité:

Informatique

Option :

Ingénierie du logiciel

Thème :

Refactoring d'un logiciel de Data Warehouse au moyen des patrons de conception en temps réel

Présenté par :
HADJADJ Mohamed

Promoteur :
Dr. MAHIEDDINE Mohammed

Soutenue le:
Devant le jury composé de :

Présidant
Examinateur
Examinateur

Année universitaire 2011-2012.

MA-004-95-1

Dédicaces

Je dédie ce travail

A' Mes très chers parents et surtout à ma mère

Qu'elle a beaucoup sacrifie pour moi et qu'elle n'a jamais cessée

de m'encourager.

A' mes frères et à mes sœurs et toute ma grande famille.

A' mes amis et mes collègues

À mes amis de la section de

L'informatique

HADJADJ

Résumé

Le principal but de ce travail est de permettre l'analyse du logiciel de DW temps réel développé et la re-conception de ses côtés qui le nécessitent, à travers une bonne re-fabrication, pour son utilisation dans le domaine médical, actuellement appliqué à la prescription de médicaments dans les hôpitaux.

En basant sur un développement par patrons, avec un certain nombre d'autres techniques de Conception (re-conception, re-fabrication, maintenabilité, réutilisabilité, ...) on pourrait arriver à optimiser le logiciel sur le plan qualité du design, les possibilités qu'il pourrait offrir et sur le temps d'exécution a pour but de réduire le temps qu'il faudrait pour faire des décisions.

Mots Clés:

DW, schéma en étoile, base de données opérationnelle, table relationnelle, Refactoring, Meta Donnée, Staging Area, patron de conception, Observer/Observable, Singleton, table de faits, dimension, Client/Serveur, Java, PostgreSQL, MySQL.

Abréviations

DW : Data Warehouse (Entrepôt de données)

BI : Business Intelligence.

DM : Data Mining

DIM : Dimension

ETL: Extract, Transform and Load (ETC).

OLAP: On Line Analytical Process.

OLTP: On Line Transactional Process.

HOLAP: Hybrid On Line Analytical Process.

MOLAP: Multidimensional On Line Analytical Process.

ROLAP: Relational On Line Analytical Process.

PK: Primary Key.

SI: Systèmes d'Information.

SID: Systèmes d'Information Décisionnels.

SIAD : Systèmes d'Information d'Aide à la Décision

SGBD : Système de Gestion de Base de Données.

SQL : Structured Query Language.

UML: Unified Modeling Language.

Table des matières

Introduction Générale

Les DW constituent de nos jours l'infrastructure de base des systèmes d'aide à la décision. Ils permettent de garder des grands volumes d'informations historiques permettant ainsi de découvrir des tendances et des informations similaires qui sont indispensables aux organisateurs.

Dans ce travail nous allons des techniques de Conception (re-conception, re-fabrication, maintenabilité, réutilisabilité, ...) pour la refactoring d'un logiciel de DW en temps réel, depuis les bases opérationnelles jusqu'aux phases Quering et Reporting sur le plan qualité du design, et sur le temps d'exécution.

Parmi les techniques de refactoring ont utilisant les patrons de conception qu'ils sont de plus en plus utilisés dans la conception de logiciels réutilisables, et on a donc étudié leur utilisation dans le domaine des entrepôts de données.

Le but de ce travail est la réalisation d'un D W temps réel, à base des techniques de refactoring en utilisant le schéma en étoile qui lui devrait être temps réel, pour son utilisation dans le domaine médical, actuellement appliqué à la prescription de médicaments dans les hôpitaux.

Le développement de DW pour le domaine médical vise à ce qu'il soit utilisé comme une structure dans l'assistance à la prise de décision pour le contrôle de l'approvisionnement en médicaments dans les hôpitaux. On projette de l'étendre à l'avenir pour le domaine du diagnostic des patients.

Problématique

La création d'un logiciel de A à Z prend beaucoup de temps sur la réalisation lui-même et sur la validation de ce projet (2ans pour les logiciels de DW) [Dou05] et les couts de réalisation sont élevés.

Pour cela les développeurs travailler sur des logiciels qui sont open source avec des améliorations (refactoring) dans le but de construire des logiciels de bon qualité qui marche avec les besoins de l'utilisateurs et pour minimiser les couts et le temps de réalisations.

Dans cette situation nous détectons les problèmes suivants :

- Comment construire un logiciel de DW en temps réel à partir des besoins des utilisateurs ?
- Quelle architecture logicielle retenir ?
- Quelle sont les techniques utiliser, et pour quels besoins ?
- Comment organiser les grands volumes de données ?
- Quelle démarche adopter ?

Objectifs

L'objectif de ce thème est d'améliorer un logiciel de DW en temps réel au moyen de patrons de conception pour construire un logicielle dans un cycle de vie long Ont utilisé les technique de refactoring. Ainsi, les principaux objectifs assignés au projet sont :

- La réduction de la durée globale de fonctionnalité de system.
- Offrir aux décideurs et aux analystes la possibilité de faire des analyses appropriées.
- Ajouter d'autre fonctionnalité.
- Faciliter le suivi des patients.

Organisation du mémoire

Afin de présenter notre travail, le présent mémoire est organisé en trois chapitres et se Présente comme suit :

Après une **introduction générale** dans laquelle nous présentons notre projet, ainsi que la problématique et les objectifs visés.

Le premier chapitre présente les notions théoriques de base sur les DW, les concepts principaux d'OLAP, les principes de la modélisation multidimensionnelle ainsi l'architecture d'un DW et ses composants :

Les différents modèles multidimensionnels pour la construction d'un système décisionnel, ainsi que l'ensemble des opérations pour la manipulation des données multidimensionnelles.

Le deuxième chapitre, présente un aperçu sur la conception par patrons, ainsi que les patrons que nous avons utilisé dans notre conception et enfin la notion de la communication client-serveur.

Le troisième chapitre présente l'aspect fonctionnel, statique et dynamique de notre système, ceci à travers un ensemble de diagrammes de cas d'utilisation, diagramme de classes, et diagrammes de séquence.

Le quatrième chapitre décrie l'architecture globale de la solution, en présentant les différents outils utilisés, et nous donnons quelques images de l'application.

Finalement, une **conclusion générale** dans laquelle nous identifiant ses limites et les améliorations possibles, ainsi que les travaux futurs.

CHAPITRE 1

Généralités sur les entrepôts de données

1. Introduction

Les DW constituent de nos jours l'infrastructure de bases des systèmes d'aide à la décision. Ils permettent de garder des grands volumes d'informations historiques, permettant ainsi de découvrir des tendances et des informations similaires qui sont indispensables aux organisateurs.

Dans ce chapitre, nous décrivons les principaux concepts des entrepôts de données (DW) et de l'analyse multidimensionnelle et présentons les caractéristiques principales de leur modélisation.

2. Les Systèmes Décisionnel

2.1. Définition

Parmi les plus importantes définitions des systèmes décisionnels « B.I. » on trouve :

« Le Décisionnel est le processus visant à transformer les données en informations et, par l'intermédiaire d'interrogations successives, transformer ces informations en connaissances. » [Dre01].

« Le SID est un système permettant aux décideurs de l'entreprise de disposer d'informations pertinentes et d'outils d'analyse puissants pour aider à prendre les bonnes décisions au bon moment » [Dev02].

2.2. Système décisionnel VS système transactionnel

C'est à partir des années 90 que les entreprises ont compris que les données sont non seulement utiles dans le cadre d'utilisation opérationnelle. Mais qu'on peut leur trouver une utilisation stratégique. Alors ces données constituent la matière première des SID.

Le tableau suivant montre la différence entre les systèmes transactionnels et les systèmes décisionnels du point de vue de leur usage et des données qu'ils utilisent [Bou10] :

	Systèmes transactionnels	Systèmes décisionnels
Données	Orientés applications	Orientés sujets
	Situation instantanée (à jour)	Situation historique
	Données détaillés et codées non redondantes	Informations agrégées cohérentes souvent avec redondance
	Données changeant constamment (dynamique)	Informations stables et synchronisées dans le temps (statique)
	Pas de référentiel commun	Un référentiel unique
Usage	Assurent l'activité au quotidien	Permettent l'analyse et la prise de décision
	Pour les opérationnels	Pour les décideurs et les analystes
	Mises à jour et requêtes simples	Lecteur seulement et requêtes complexes
	Temps de réponse immédiat	Temps de réponse moins rapide
	Faible volume a chaque transaction	Large volume manipulé
	Conçus pour la mise à jour	Conçus pour l'extraction

Tableau I.1: *Le transactionnel contre le décisionnel.*

3. Le data warehouse (DW)

3.1. Définition

Un certain nombre de définitions ont été proposées pour les entrepôts de données « DW » est défini de plusieurs façons comme suit :

- « Le DW est une collection de données orientées sujet, intégrées, non volatiles et évolutives dans le temps, organisées pour le support d'un processus d'aide à la décision. » [Inm02].

« Une collections de données intégrées, variable dans le temps, non volatiles utilisées pour la prise de décision dans une organisation » [Cha79].

Nous détaillons ces caractéristiques :

➢ **Orientées sujets**

« DW est organisé autour des sujets majeurs de l'entreprise, contrairement aux données des systèmes opérationnels » [Nak98].

Les données des DW sont organisées par sujet plutôt que par application. Par exemple, une chaîne de magasins d'alimentation organise les données de son entrepôt par rapport aux ventes qui ont été réalisées par produit et par magasin au cours d'un certain temps.

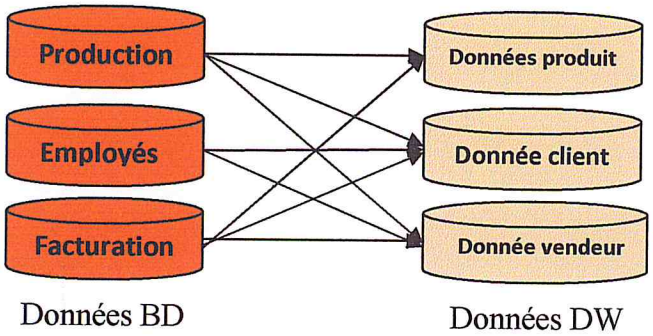


Figure I.1 : *Données thématiques*

➢ **Intégrées**

Les données alimentant le DW proviennent de multiples sources hétérogènes. Les données des systèmes de production doivent être converties, reformatées et nettoyées, de façon à avoir une seule vision globale dans l'entrepôt de données.

L'intégration des données consiste à contenir leurs hétérogénéités pour donner au contenu de l'entrepôt de données une présentation homogène et pour garantir sa qualité avant d'être intégrées dans l'entrepôt de données, les données doivent donc être mise en forme et unifiées afin d'en avoir un état cohérent.

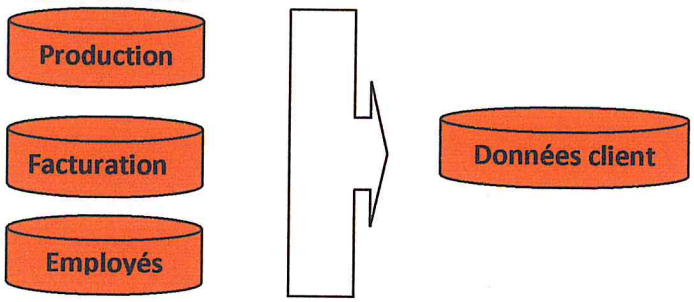


Figure I.2 : *Données intégrées*

➢ **Non volatiles**

A la différence des données opérationnelles, celles de l'entrepôt « DW » sont permanentes et ne peuvent pas être modifiées. Le rafraîchissement de DW consiste à ajouter de nouvelles données, sans modifier ou perdre celles qui existent.

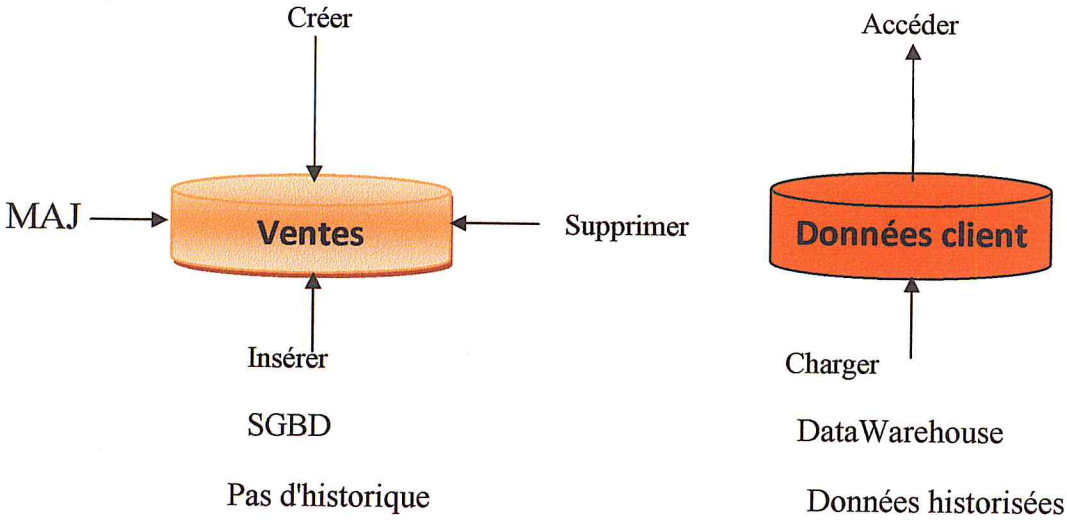


Figure I.3 : *Données non volatiles.*

➢ **Historisées**

En générale, dans un système de production, la mise à jour des données se fait lors de chaque nouvelle transaction. Une mise à jour annule et remplace l'ancienne valeur.

Dans un DW la donnée ne doit jamais être mise à jour. Un référentiel temps doit être associé à la donnée afin qu'on soit capables d'identifier une valeur particulière dans le temps.

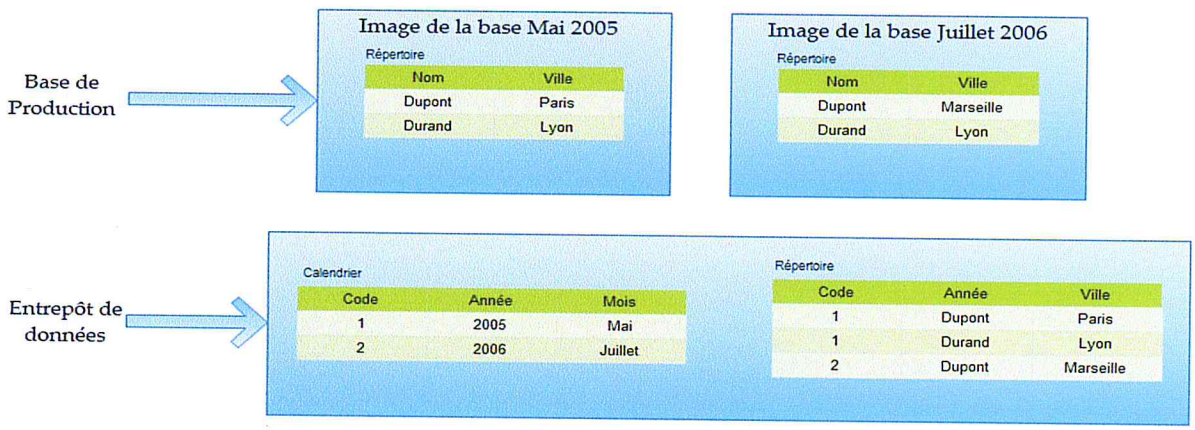


Figure I.4 : *Données historisées*

➢ **Organisées pour le support d'un processus d'aide à la décision**

Les données du DW sont organisées de manière à permettre l'exécution des processus d'aide à la décision (Reporting, DM,…).

3.2. Objectif d'un DW

- Intégrer des sources de données, donc résoudre le problème d'hétérogénéité des différentes sources.
- Supporter un processus d'analyse en ligne, exploiter l'entrepôt de données.
- Construire de l'information utile pour l'aide à la prise de décision.

3.3. DW temps réel (DWTR)

Les DWTRs sont caractérisés par l'envoi de données en continu, de manière asynchrone et multipoints offrant à l'utilisateur l'information la plus à jour possible. Presque immédiatement après que les données d'origine soient écrites, ces données vont se déplacer de la source d'origine directement vers l'entrepôt de données(DW) [Sco10].

Les principaux avantages des DWTR sont :

- Une prise de décision actualisée
- L'accélération de la décision
- L'alimentation temps réel des tables de DW

3.4. Les éléments d'un DW

3.4.1. Le système source

« Système opérationnel d'enregistrement, dont la fonction consiste à capturer les transactions liées à l'activité de l'entreprise » [Kim97].

Il peut être représenté par les systèmes de productions ou bien de données externes. La principale priorité du système source est le temps de disponibilité.

3.4.2. La zone de préparation des données (ETL)

« Ensemble de processus qui nettoient, transforment, combinent, archivent, suppriment les doublons, c'est-à-dire prépare les données sources en vue de leur intégration puis de leur exploitation au sein de DW» [Kim97].

Cette zone comprend la préparation englobe tout ce qu'il y a entre les bases de production (system source) et la présentation des données (DW). Elle est constituée d'un ensemble de processus appelé ETL, « Extract, transform and Load », les données sont extraites et stockées pour subir les transformations nécessaires avant leur chargement.

3.4.2.1. L'extraction des données

« L'extraction est la première étape du processus d'apport de données à l'entrepôt de Données(DW). Extraire, cela veut dire lire et interpréter les données sources et les copier dans la zone de préparation en vue de manipulations ultérieures. » [Kim 05].

L'extraction des données consiste à collecter les données utiles dans le système de production pour rafraîchir la base décisionnelle, il est nécessaire ensuite de planifier ces extractions afin d'éviter les saturations du système de production.

3.4.2.2. La transformation des données

La transformation est la seconde phase du processus. Cette étape, assure en réalité plusieurs tâches qui garantissent la fiabilité des données et leurs qualités. Ces tâches sont :

- Consolidation des données.
- Correction des données et élimination de toute ambiguïté.
- Elimination des données redondantes.
- Compléter et renseigner les valeurs manquantes.

3.4.2.3. Le chargement des données

C'est la dernière phase de l'alimentation d'un entrepôt de données, le chargement est une étape indispensable. Elles sont mises à disposition des outils d'analyse DM, L'analyse multidimensionnelle OLAP, …etc.

3.4.2.4. Les approches ETL

Il existe plusieurs approches de la mise en œuvre ETL. La plus utilisée c'est celle de Kimball qui est un ETL avec Base de données temporaire « Staging Area»

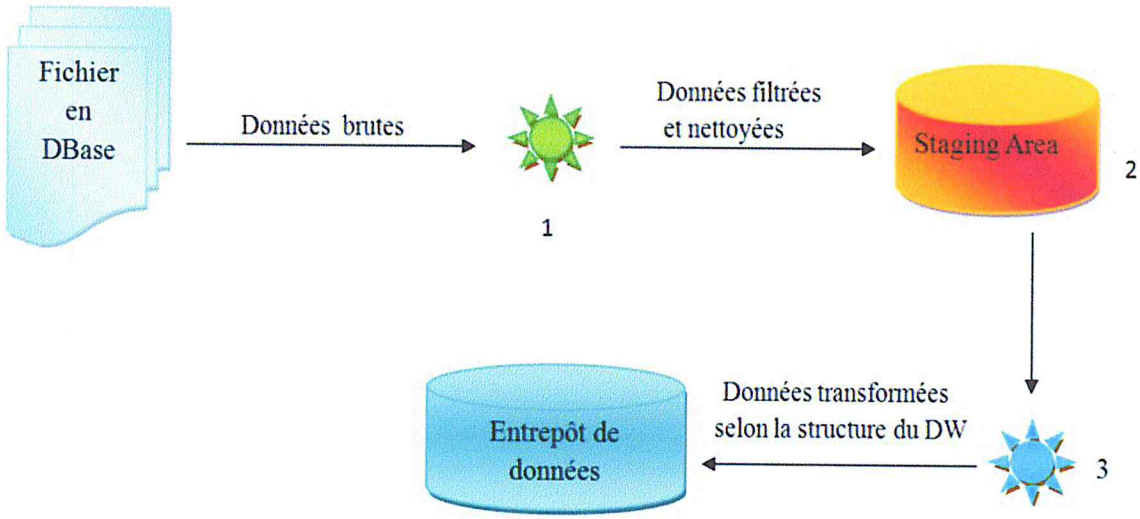


Figure I.5 : *Schéma du processus ETL*

1 : Cette opération représente la phase d'extraction, filtrage et nettoyage des données, mais avec le même niveau de granularité qui existe dans la base opérationnelle.

2 : La Staging Area (zone de préparation) est une base intermédiaire qui reçoit des données filtrées et nettoyées, pour lancer la deuxième phase : transformation des données (selon le format adapté par la structure de l'entrepôt de données). Cette zone de données prend en considération la minimisation de l'intervention des administrateurs du système décisionnel, et de ne pas prendre des données en double dans le cas où le chargement s'arrête sans qu'il soit fini (coupure de courant électrique, panne de la machine …).

3 : Alimentation de base multidimensionnelle (tables de faits et dimensions) et Agrégation de données.

3.4.3. Le magasins de données

« Le magasin de données ou Data Mart est défini comme un sous-ensemble logique d'un DW » [Kim97].

Ce sont des extraits de DW orientés métier et activités, contenant un volume moindre de données, permettant alors des analyses plus rapides.

3.4.4. Les métadonnées

Les métadonnées constituent un dictionnaire et une véritable aide en ligne permettant de connaitre l'information contenue dans l'entrepôt de données. Elles sont idéalement intégrées dans un référentiel.

Les principales informations sont destinées :

- A l'utilisateur (sémantique, localisation).
- Aux équipes responsables des processus de transformation des données du système de production vers l'entrepôt de données (localisation dans les systèmes de production, description des règles, processus de transformation).
- Aux équipes responsables des processus de création des données agrégées à partir des données détaillées.
- Aux équipes d'administration de la base de données (structure de la base implémentant l'entrepôt de données).
- Aux équipes de production (procédures de changement, historique de mise à jour,…).

3.5. Architecture d'un DW

Après avoir exposé et défini chacun des éléments constituant l'environnement d'un DW, il serait intéressant de connaitre le positionnement de ces éléments dans une architecture globale d'un DW [Bou10] :

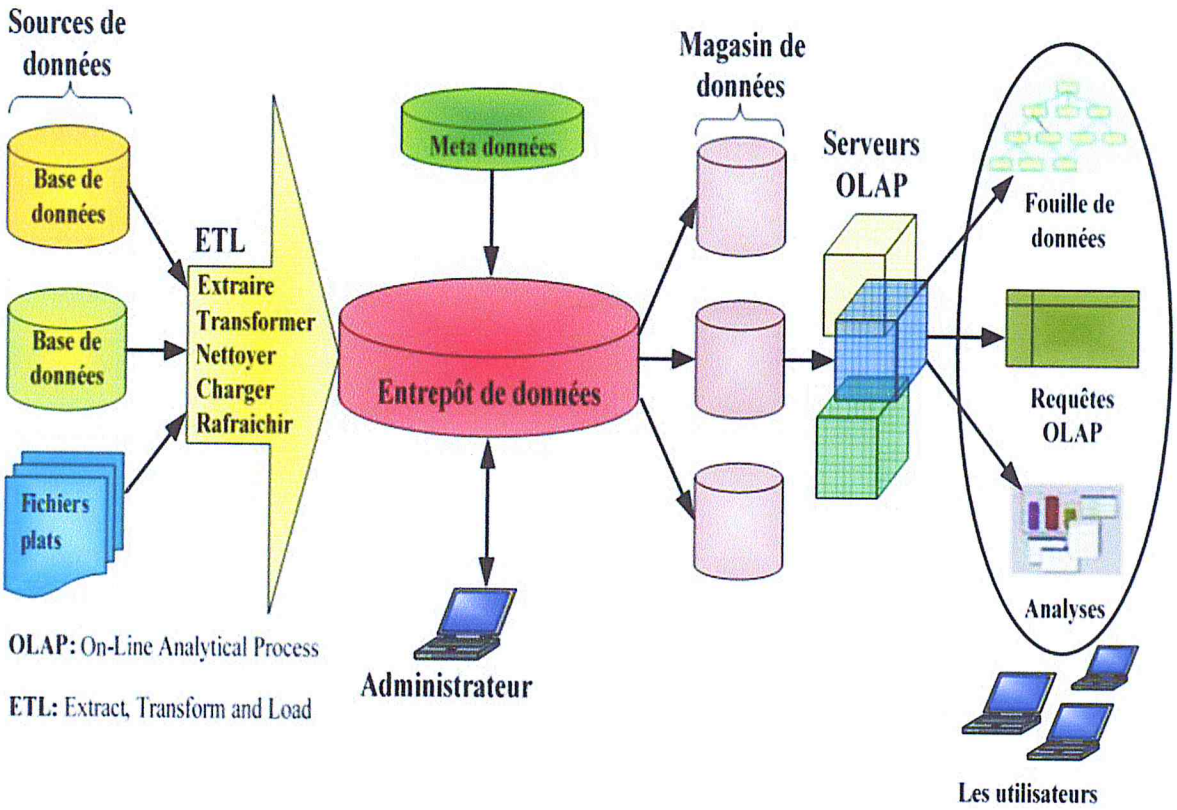


Figure I.6 : *Architecture détaillée du DW.*

3.6. Mise en œuvre du DW

La mise en œuvre du DW comporte le :

3.6.1. Reporting

Destiné essentiellement à la production de rapports et de tableaux de bord, « il est la présentation périodique de rapports sur les activités et résultats d'une organisation, d'une unité de travail ou du responsable d'une fonction, destinée à en informer ceux chargés de les superviser en interne ou en externe, ou tout simplement concernés par ces activités ou résultants » [site1].

3.6.2. Analyse dimensionnelle des données

L'analyse dimensionnelle est sans doute celle qui exploite et fait ressortir au mieux les capacités de l'entrepôt de données. Le but par l'analyse dimensionnelle est d'offrir aux utilisateurs la possibilité d'analyser les données selon différents critères afin de confirmer une tendance ou suivre les performances de l'entreprise.

3.6.2.1. Tableaux de bord

Les tableaux de bord sont un outil de pilotage qui donne une vision sur l'évolution d'un processus, afin de permettre aux responsables de mettre en place des actions correctives.
« Le tableau de bord est un ensemble d'indicateurs peu nombreux conçus pour permettre aux gestionnaires de prendre connaissance de l'état et de l'évolution des systèmes qu'ils pilotent et d'identifier les tendances qui les influenceront sur un horizon cohérent avec la nature de leurs fonctions » [Bou03].

3.6.2.2. DM

Au sens littéral du terme, le DM signifie le forage de données. Le but de ce forage est d'extraire de la matière brute qui, dans notre cas, représente de nouvelles connaissances. L'idée de départ veut qu'il existe dans toute entreprise des connaissances utiles, cachées sous des gisements de données.

Le DM permet donc, grâce à un certain nombre de techniques, de découvrir ces connaissances en faisant apparaître des corrélations entre ces données.

3.7. Modélisation des données d'un DW

« Au niveau de l'entrepôt, pour pouvoir exploiter facilement les données, le concepteur doit réaliser une classification par sujet fonctionnel plutôt que par application, donc on peut dire

qu'un entrepôt de donnée regroupe un ensemble de sujets principaux de l'organisation » [Dou05].

3.7.1. Concepts de modélisation

3.7.1.1. Fait

Un fait représente un sujet d'analyse. Il est constitué de plusieurs mesures relatives au sujet traité, ces mesures sont numériques et généralement valorisées de façon continue.

Dans la plupart des modèles multidimensionnels, les faits sont implicitement représentés par la combinaison des valeurs des dimensions [Mar06].

3.7.1.2. Mesure

Les mesures sont les éléments mesurables de l'activité. Ces éléments sont des indicateurs appréciables, comme par exemple **délai, quantité, taux, coût,** ou une **note**.

Pour chaque mesure souhaitée il faut se demander quels sont les éléments dont on souhaite mesurer la magnitude (prix, taux, intensité, température, …etc.). Quand il y a une demande de requête d'une mesure, le serveur calcule les valeurs de tous les membres spécifiés dans la requête [Mar06].

3.7.1.3. Table de Faits

La table centralisée dans un schéma en étoile est appelée table de faits.

Les tables de faits sont généralement longues et maigres en données.

Une table de fait a généralement deux types de colonnes: ceux qui contiennent des faits et ceux qui sont clés étrangères aux tables de dimension. La clé primaire d'une table de faits est généralement un composite clé qui est constitué de l'ensemble de ses clés étrangères [Dou05].

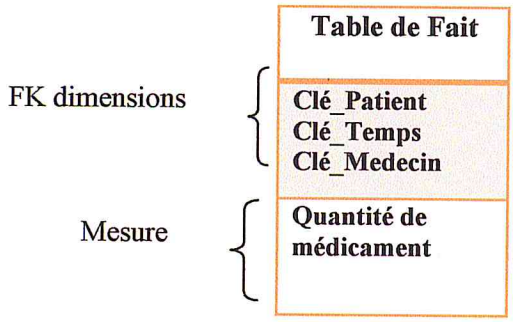


Figure I.7 : *Table de fait*

3.7.1.4. Table de dimension

Les tables de dimension sont les tables qui raccompagnent une table de faits. une dimension est constituée de nombreuses colonnes qui décrivent une ligne. C'est grâce à cette table que l'entrepôt de données est compréhensible et utilisable; elles permettent des analyses en tranches et en dés.

« Une table de dimension établit l'interface homme / entrepôt, elle comporte une clé primaire » [Kim02].

3.7.2. Comparatif entre les tables de faits et les tables de dimensions

Le tableau suivant récapitule les différences au niveau des données de ces tables [Bou10] :

	Tables de faits	Tables de dimensions
Structure	Peu de colonnes beaucoup de lignes	Peu de lignes beaucoup de colonnes
Données	Mesurable, généralement numérique	Descriptives généralement textuelles
Référentiel	Plusieurs clés étrangères	Une clé primaire
Valeur	Prend de nombreuses valeurs	Plus ou moins constantes
Manipulation	Participe à des calculs	Participe à des contraintes
Signification	Valeurs de mesure	Descriptive
Rôle	Assure les relations entre les dimensions	Assure l'interface homme/entrepôt de données

Tableau I.2 : *Tableau comparatif entre les tables de faits et les tables de dimensions.*

3.7.3. Schémas de modélisation multidimensionnelle

3.7.3.1. Schéma en étoile

Ce modèle se présente comme une étoile dont le centre n'est autre que la table des faits et les branches sont les tables de dimension. La force de ce type de modélisation est sa lisibilité et sa performance. Elle est caractérisée généralement par une table de faits très large qui contient l'information primaire dans l'entrepôt de données, et un certain nombre de tables de dimension beaucoup plus petites (ou tables de consultation), dont chacune contient des informations sur les entrées pour un attribut particulier dans la table de faits.

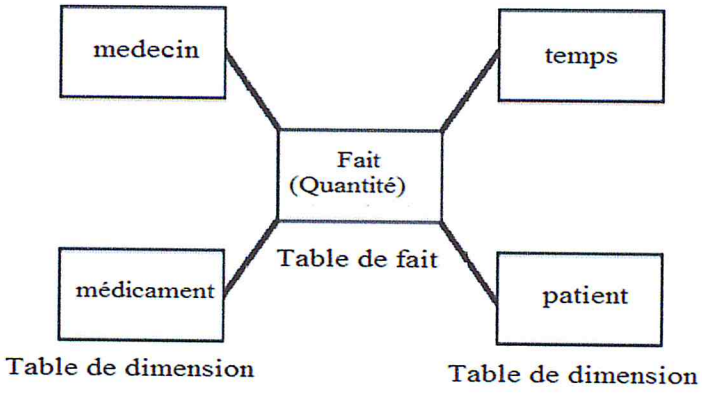


Figure I.8 : *Schéma en étoile*

3.7.3.2. Le schéma en flocon de neige

La modélisation en flocon est une modélisation en étoile pour laquelle on éclate les tables de dimensions en sous tables selon la hiérarchie de cette dimension.

L'avantage de cette modélisation est de formaliser une hiérarchie au sein d'une dimension.

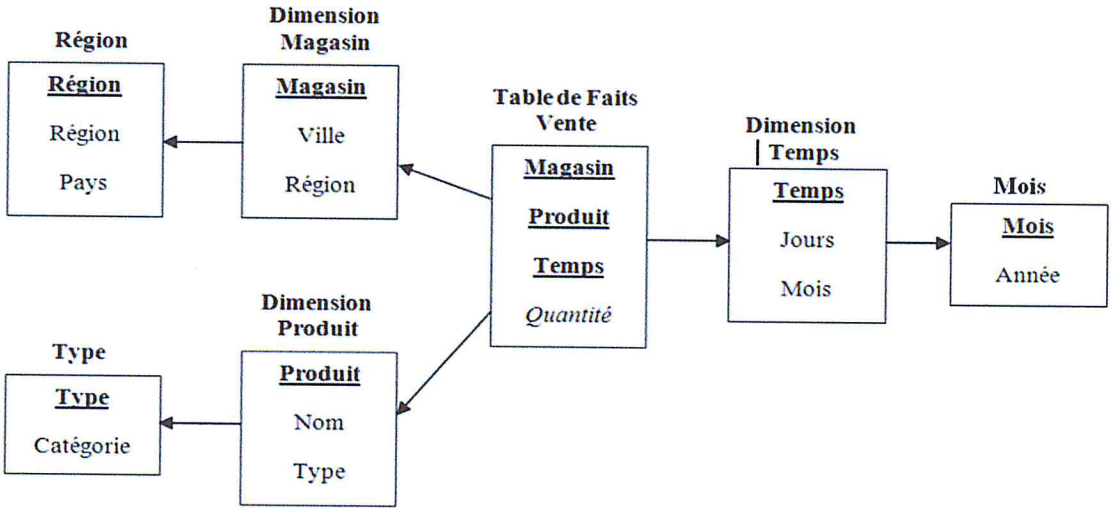


Figure I.9 : *Schéma en Flocon.*

3.7.3.3. Schéma en Constellation

Il s'agit de fusionner plusieurs modèles en étoile qui utilisent des dimensions communes. Un modèle en constellation comprend donc plusieurs faits et de dimensions communes ou non.

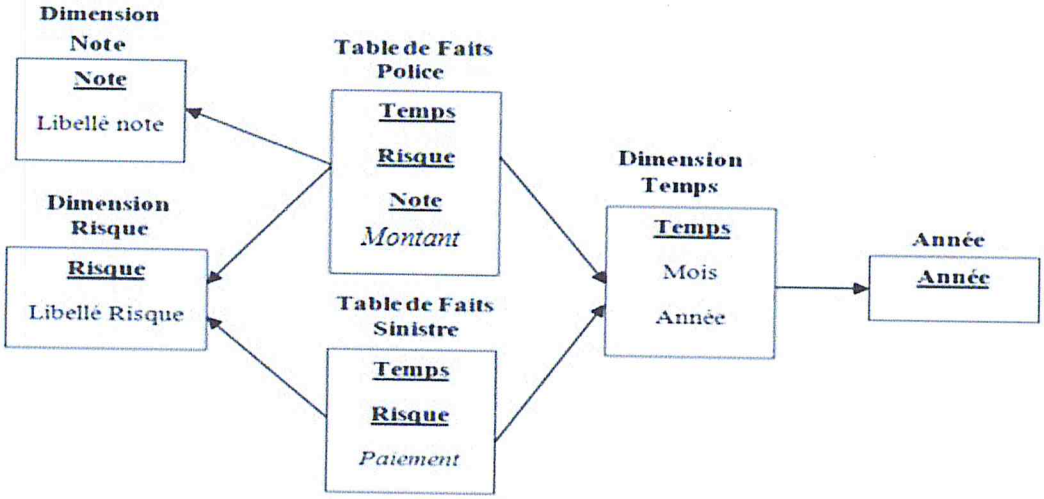


Figure I.10 : Schéma *en constellation.*

3.7.4. OLAP

Le terme OLAP (On-Line Analytical Processing) désigne une classe de technologies conçue pour l'accès aux données et pour une analyse instantanée de ces dernières, dans le but de répondre aux besoins de Reporting et d'analyse.

R. Kimball définit le concept « OLAP » *comme* «l'Activité globale de requêtage et de présentation de données textuelles et numériques contenues dans l'entrepôt de données; Style d'interrogation spécifiquement dimensionnel » [Kim05].

3.7.4.1. Opérations OLAP liées à la structure

Ce sont les opérations qui agissent sur la structure multidimensionnelle, qui visent à changer le point de vue des données.

- **Extraction d'un bloc de données (Dice)**

Elle extrait un « sous-cube » du cube principal. Donc elle travaille que sous un sous-cube. L'opération est illustrée dans la figure suivante [Bou10] :

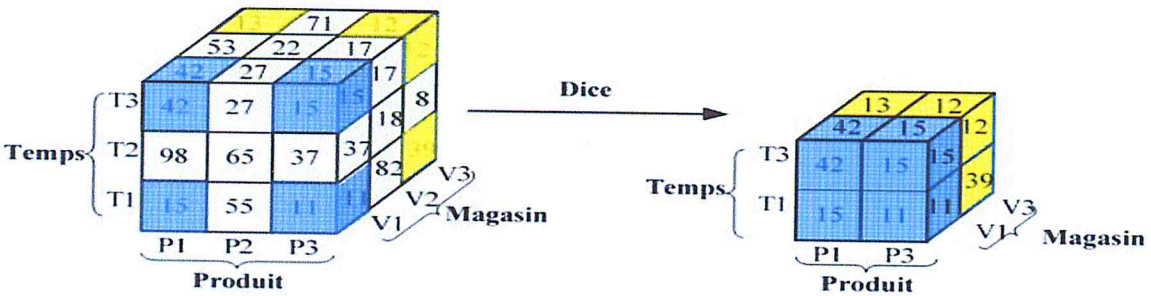


Figure I.11: *Dicing.*

➢ **Pivot (Rotate)**

Cela consiste à faire effectuer à un cube une rotation autour d'un de ces axes passant par le centre de deux faces opposées, de manière à présenter un ensemble différent. Un exemple explicatif est représenté dans la figure suivante [Bou10] :

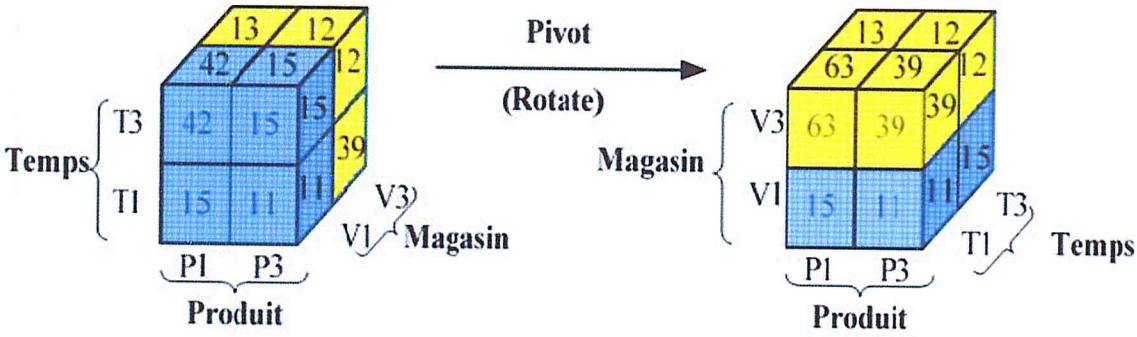


Figure I.12 : Opération *Rotate.*

➢ **Permutation (Switch)**

Consiste à inter-changer la position des membres d'une dimension. Cette opération est illustrée dans la figure suivante [Bou10]:

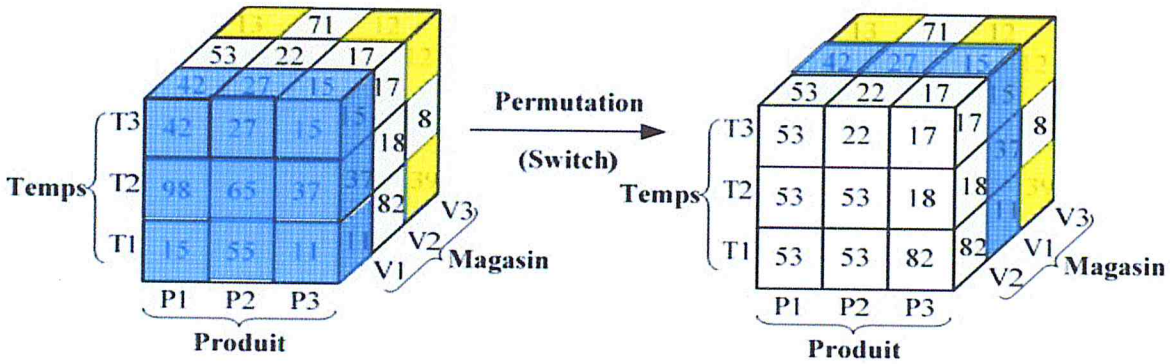


Figure I.13 : *Opération Switch.*

➢ **Extraction d'une Tranche du cube (Slice) :**

Cela consiste à sélectionner une dimension, il s'agit de couper une tranche du cube afin d'observer les données de la dimension. Cette opération est illustrée dans la figure suivante [Bou10]:

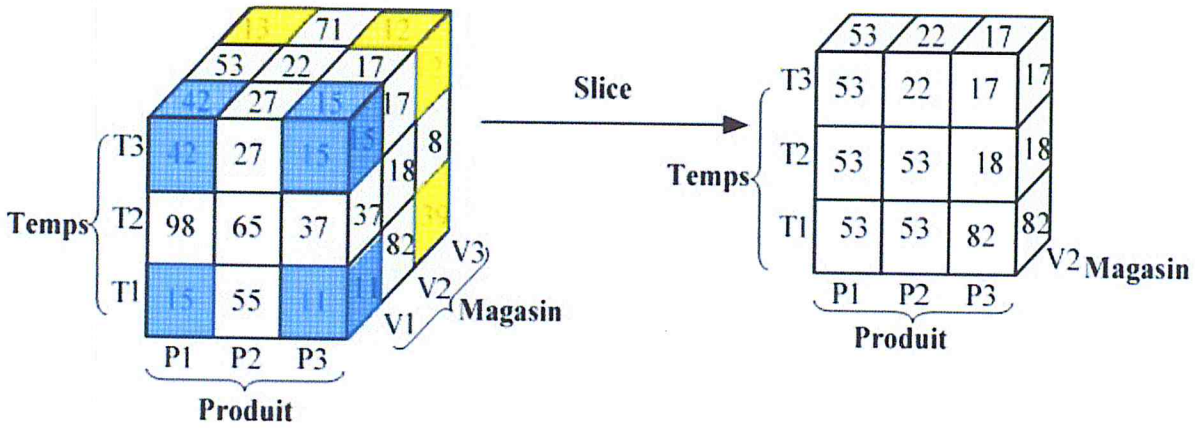


Figure I.14 : *Opération Slice.*

3.7.4.2. Les opérations OLAP liées à la granularité

Les opérations agissant sur la granularité d'observation des données, caractérisent la hiérarchie de navigation entre les différents niveaux. Elles correspondent aux opérations suivantes [Bou10]:

- **Forage vers le haut (Drill-up ou Roll-up ou Scale-up)**

Cela consiste à représenter les données du cube à un niveau de granularité supérieur conformément à la hiérarchie définie sur la dimension. Une fonction d'agrégation est utilisée (somme, moyenne, etc.) en paramètre à l'opération, indique comment sont calculés les valeurs du niveau supérieur à partir de celles du niveau inférieur.

- **Forage vers le bas (Drill-down ou Roll-down ou Scale-down)**

Elle fait l'inverse de l'opération précédente (Roll-up), elle consiste à représenter les données du cube à un niveau de granularité de niveau inférieur, donc sous une forme plus détaillée (descendre dans la hiérarchie d'une dimension). Cette opération est illustrée dans la figure suivante [Bou10] :

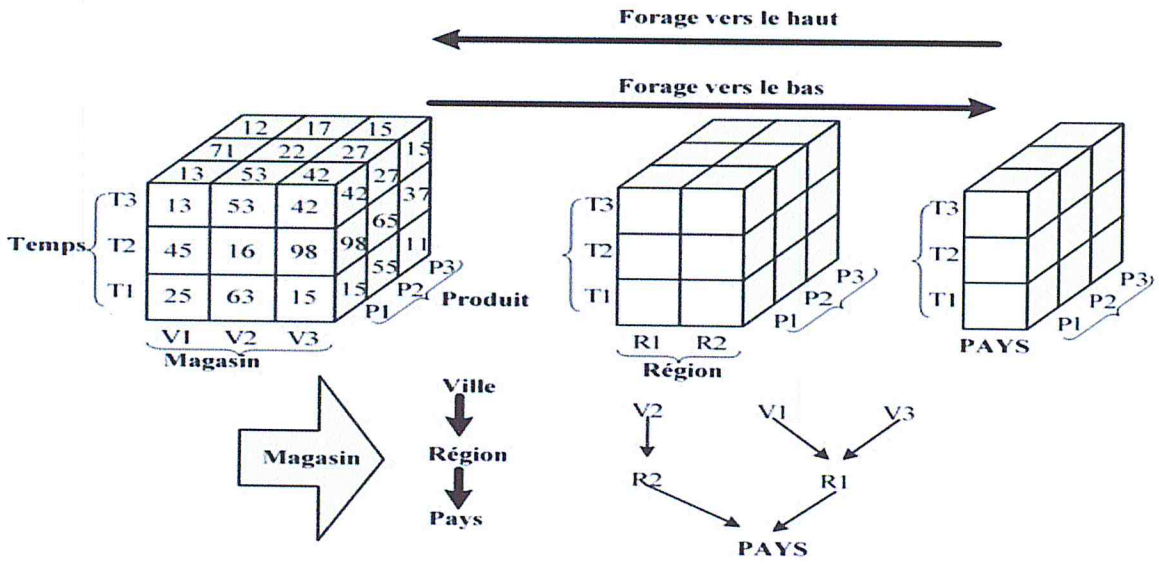


Figure I.15 : *Application du forage vers le bas et vers le haut sur le cube "Ventes".*

4. Conclusion

Dans ce chapitre, nous avons évoqué les principaux concepts liées à la conception et à l'exploitation des entrepôts de données, où nous avons vu les caractéristiques des données de l'entrepôt et l'architecture associée. Nous avons vu, également, les différents Schémas de modélisation multidimensionnelle et les opérations de manipulation.

CHAPITRE 2

Les Techniques de refactoring

1. Introduction

Dans ce chapitre, nous décrivons les principaux concepts sur le refactoring, la conception par patrons ainsi que les patrons que nous avons utilisés dans notre conception.

Ce chapitre examinera également la communication client-serveur du fait que nous allons utiliser dans le refactoring de notre DWTR.

2. Le REFACTORING

2.1. Définition

Le refactoring est une activité d'ingénierie logicielle consistant à modifier le code source d'une application de manière à améliorer sa qualité sans altérer son comportement vis-à-vis des utilisateurs. [Fow00]

Le refactoring est une démarche qui vise à pallier activement les problèmes de l'évolution logicielle, notamment celui de l'érosion. Il s'agit donc d'une activité au long court, devant être dotée d'une feuille de route continuellement mise à jour au gré des changements [Jea05].

Elle doit être envisagée comme un processus continu plutôt que comme un chantier devant être mené ponctuellement. Il peut être effectué en parallèle de la maintenance dès lors qu'il est compatible avec ses contraintes de réactivité. À l'instar de la correction d'erreur, plus un refactoring est effectué tôt moins il coûte cher.

2.2. Cas où le refactoring pourrait être appliquée

Les cas où le refactoring pourrait être appliquée, sont:

1. L'extraction d'un composant réutilisable. Par exemple, un besoin d'un nouveau produit supportant les nouvelles approches du processus, mais avec une interface utilisateur qui est compatible avec l'ancien système.
L'extraction l'interface utilisateur composant requiert d'abord la refactorisation les logiciels existants.

2. L'amélioration de la cohérence entre les composantes. Par exemple, dans le cas où deux composantes d'un système logiciel sont implémentées par les membres différents du projet. Bien que ces composants aient été d'abord pensés individuellement, on pourrait ensuite découvrir qu'ils partagent une abstraction commune.
Pour rendre le désign du système plus facile à comprendre, et de réduire les les coûts de maintenance, il est souhaitable de re-fabriquer (refactoring) le système pour l'amener à exprimer les éléments communs entre les deux composantes.

3. Le soutien de la conception itérative d'un framework orienté Object.

4. Les bons frameworks sont généralement le résultat de beaucoup de design itératif et donc de beaucoup de travail, impliquant des changements structurels.

2.3. Le but de refactoring

Nous utilisons le refactoring dans les buts :

- Obtenir une conception réutilisable.
- Améliorer la lisibilité du code, pour pouvoir le changer ou le modifier assez facilement.
- Prévoir des évolutions futures, de point de vue conception ou programmation (technique de programmation comme introduction d'une interface/classe abstraite)
- Restructuration/ ré-organisation du code pour mieux le comprendre, mieux le faire l'évoluer (rajouter d'autre fonctionnalités), mieux trouver les erreurs.

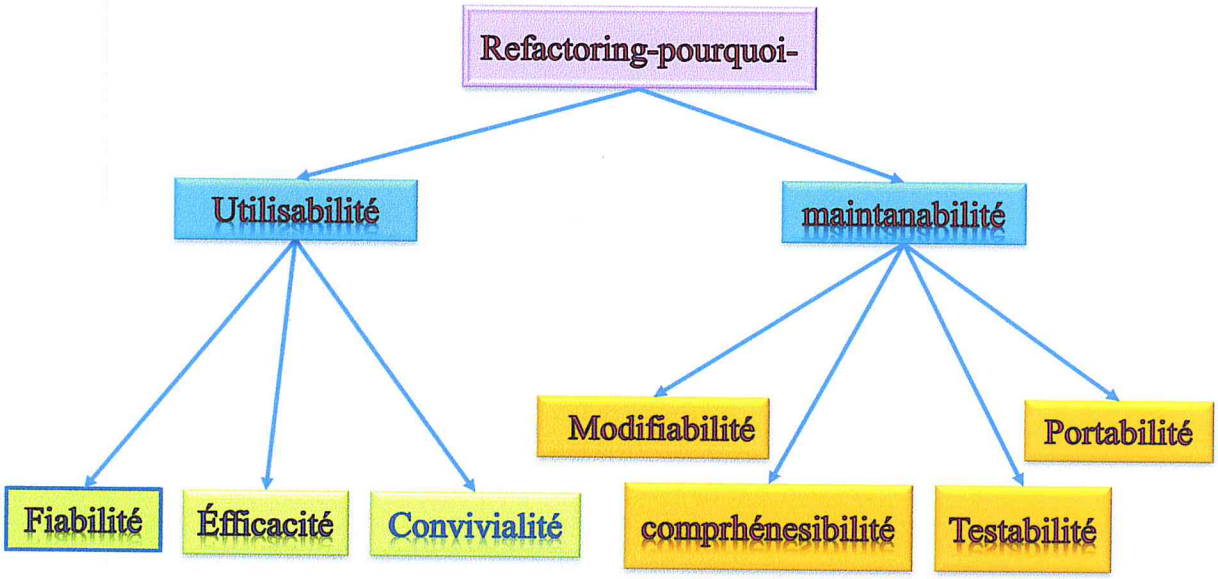


Figure II.1 : *but de refactoring.*

2.4. Refactoring orienté objet

Une liste de refactoring orienté objet a été établie [Jea05] :

1. la définition d'une super-classes abstraite d'une ou de plusieurs classes
2. la spécialisation d'une classe par la définition de sous-classes, et l'utilisation de sous classe en vue d'éliminer les tests conditionnels
3. la modification la façon dont l'ensemble ou la pièce d'association entre les classes est
6. le remplacement d'un segment de code par un appel de méthode
7. le changement des noms de classes, variables et méthodes
8. le remplacement de l'accès sans restriction à des variables avec un membre de plus d'interface abstraite

2.5. Exemples de Refactoring

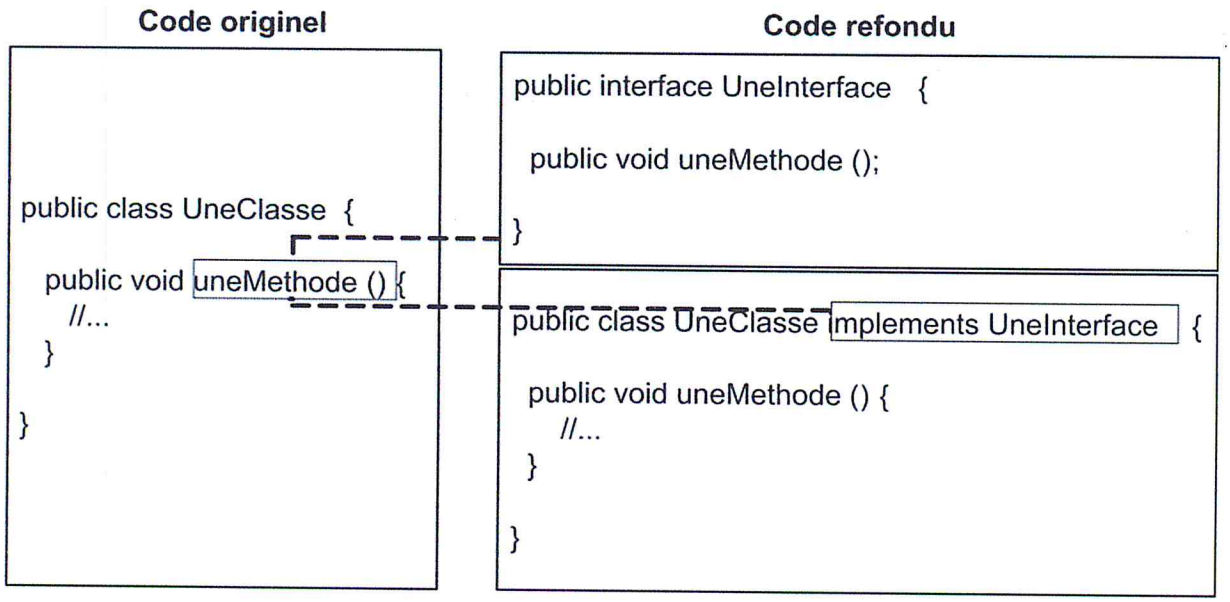


Figure II.2 : *Extraction Interface.*

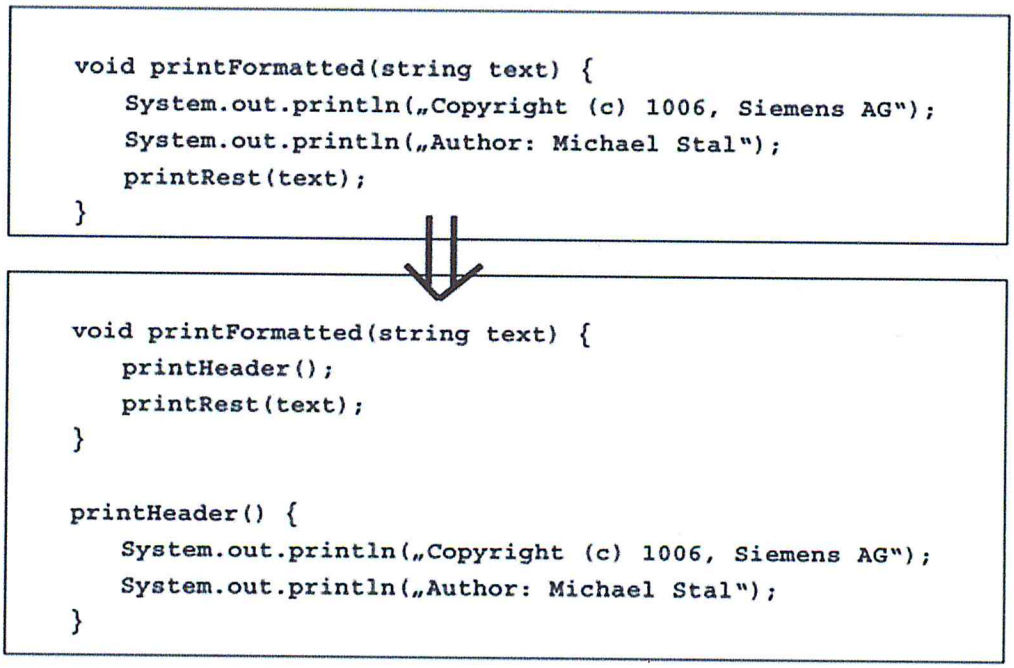


Figure II.3 : *Extraction méthode.*

```
double getSpeed() {
    switch(_type) {
       case EUROPEAN: return getBaseSpeed();
       case AFRICAN: return getBAseSpeed() - numberOfCoconuts;
       ...
    }
}
```

```
class Bird {
    public double getSpeed() ...
}
class European : Bird {
    public double getSpeed() {
       return getBaseSpeed():
    }
}
```

Figure II.4 : *Remplacé Condition Par Polymorphisme.*

```
int withdraw(int amount) {
      if (amount > _balance)
                return -1;
      else {
                _balance -= amount;
                return 0;
      }
   }
```

```
void withdraw(int amount) throws BalanceException {
      if (amount > _balance) throw new BalanceException();
      _balance -= amount;
}
```

Figure II.5 : *Remplacé erreur de code Par Exception.*

3. PATRONS DE CONCEPTION

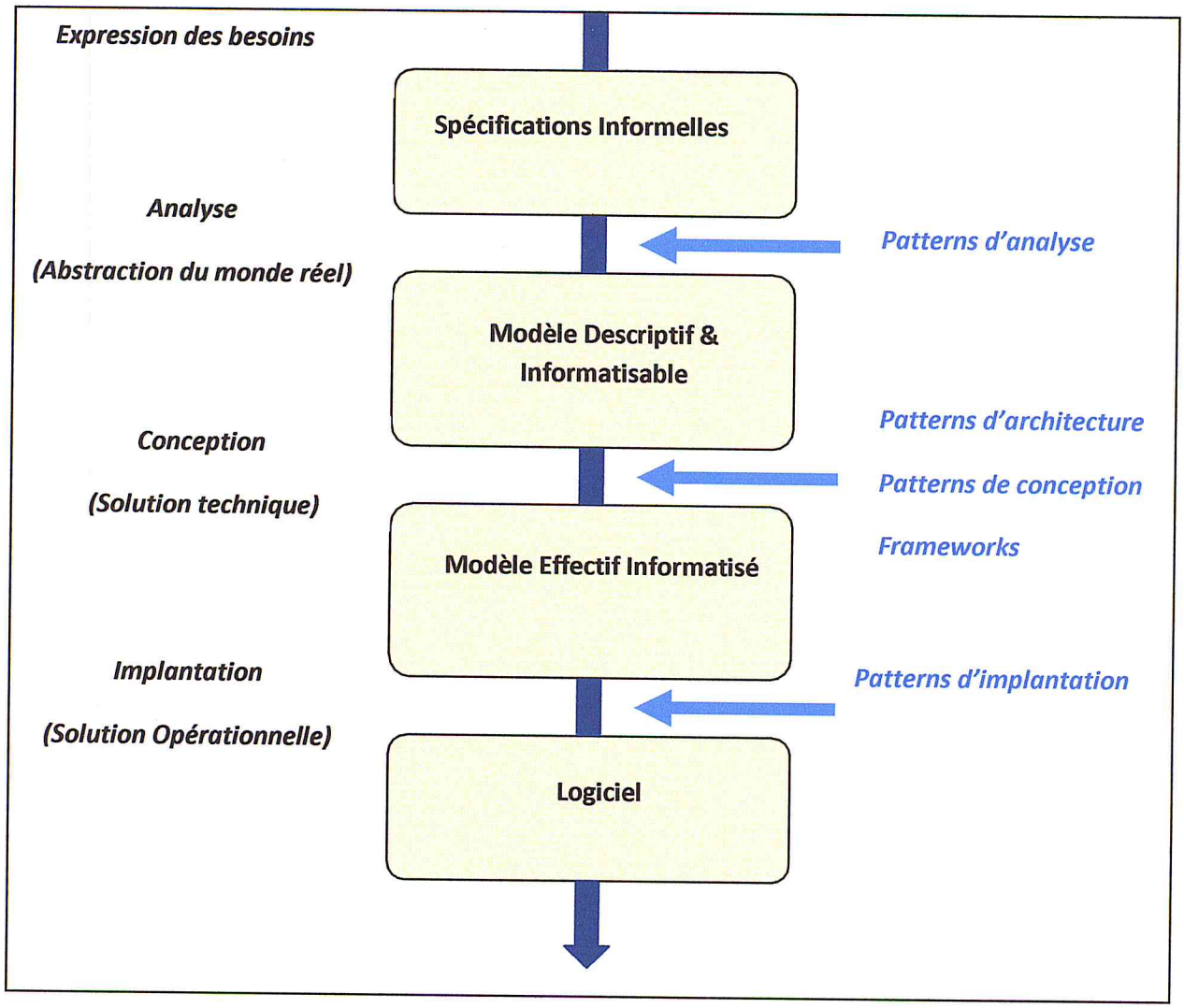
3.1. Définition

En prenant un dictionnaire comme référence, le mot « *patron* » tel qu'il est traduit de l'anglais : « *pattern* », veut dire modèle d'un ouvrage, d'un objet … etc.

Les patterns sont des solutions éprouvées à des problèmes spécifiques et récurrents « Un patron décrit un problème devant être résolu, une solution, et le contexte dans lequel cette solution est considérée. Il nomme une technique et décrit ses coûts et ses avantages. Il permet à une équipe d'utiliser un vocabulaire commun pour décrire leurs modèles » [Joh97]

On distingue : [Boi04], [Lon03].

- **les *patterns d'analyse :*** Ils ont pour but de guider les étapes d'analyse d'un cycle de vie d'un logiciel. Ils permettent d'identifier les problèmes récurrents dans l'expression des besoins des applications et de transformer ces expressions en des modèles réutilisables.
- **les *patterns architecturaux :*** Ce sont des schémas d'organisation structurelle fondamentaux pour les logiciels.
- **les *patterns de conception* (*design patterns*) :** Ils sont reconnus comme des bonnes techniques du génie logiciel à objets. Cette technique améliore le cycle de vie du logiciel en facilitant la conception, la documentation, la maintenance .Ils sont considérés comme des micro-architectures qui visent à réduire la complexité, à promouvoir la réutilisation et à fournir un vocabulaire commun aux concepteurs.
- **les *frameworks (cadres d'applications) :*** Ce sont des sous systèmes prêt à être instancier avec des possibilités bien définies d'adaptation et d'extension.
- **les *idiomes* (*patterns d'implantation*) :** Ce sont des patterns de bas niveau dans un langage de programmation donné. Ils sont destinés à montrer comment réaliser, dans un langage donné, un trait absent de ce langage. Par exemple comment réaliser l'héritage multiple en Java, comment représenter des évolutions multiples d'objets...



FigureII.6 : *les patrons.* [Mar99]

3.2. LES DIFFÉRENTES APPROCHES D'APPLICATION DES PATRONS

L'application des patrons peut se faire selon différentes approches. Flofijin et al. [Flo97] en distinguent trois :

◈ ***Approche descendante*** **:** l'utilisateur choisit un patron, l'outil utilisé génère un ensemble d'éléments de conception (classes, méthodes, associations, etc.) composant le patron.

◈ ***Approche ascendante*** **:** contrairement à l'approche précédente où de nouveaux éléments sont créés, dans cette approche, un ensemble d'éléments, reflétant la structure d'un patron particulier, est associé à une instance de ce patron.

◈ ***Approche mixte*** **:** dans ce cas, on complète un ensemble d'éléments reflétant partiellement la structure d'un patron avec d'autres éléments et associations de façon à ce que le nouvel ensemble reflète toute la structure du patron.

3.3. Avantages

L'utilisation des patrons de conception offre plusieurs avantages, comme :

- Ils sont réutilisables et permettent de mettre en avant les bonnes pratiques
- On gagne en rapidité de conception (on diminue également les coûts) et en qualité,
- Ils sont largement documentés et connus d'un grand nombre de développeurs,
- Ils facilitent la communication entre les développeurs,
- Ils répondent à un problème de conception grâce à une solution éprouvée et validée par des experts.

3.4. Le pattern de conception *Observer*

3.4.1. Principe

Le patron Observer définit une dépendance un à plusieurs entre objets, lorsque un objet change d'état tous les objets qui en dépendent sont informés et sont automatiquement mis à jour. [SR02]

La structure du patron Observer est donnée par le diagramme de classes suivant :

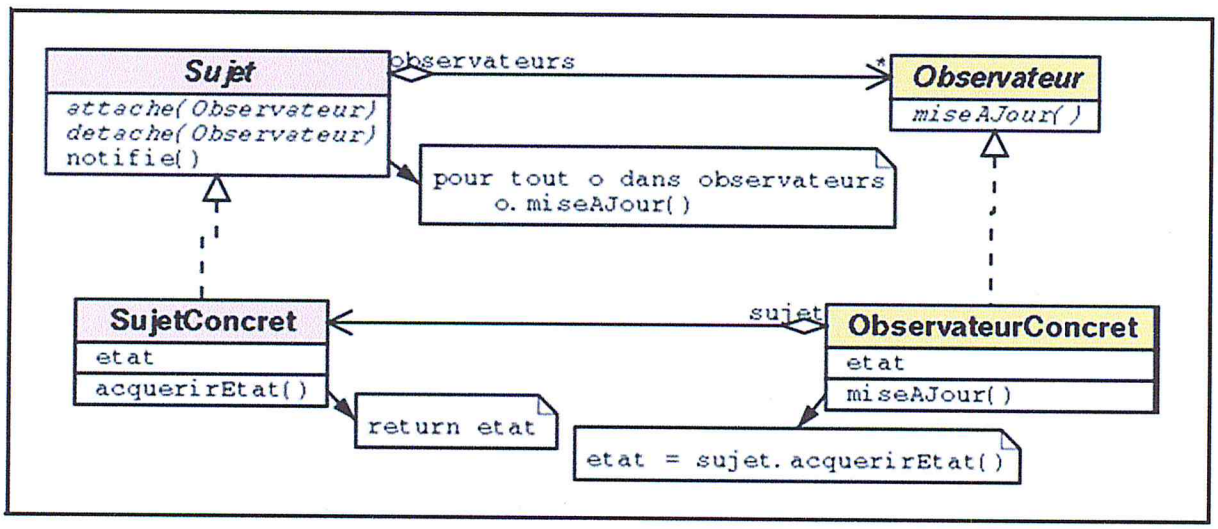


Figure II.7 : *Diagramme de classe (UML) exprimant la structure du pattern Observer.*

- **Subject(Sujet)**

  Fournit une interface pour attacher et détacher les objets observers.

- **Observer(Observateur)**

  Définit une interface pour la fonction update () qui doit être exécutée après la notification

- **ConcreteSubject(SujetConcret)**

  Représente l'objet observé.

  Sauvegarde l'état intéressé par l'objet de **ConcreteObserver**.

  Envoie une notification à ses Observers lorsqu'il change d'état.

- **ConcreteObserver(ObservateurConcret)**

  Sauvegarde l'état de l'objet observé.

  Implémente la fonction update () de l'interface **Observer** pour récupérer le nouveau état de l'objet observé.

3.4.2. Observer en java

L'API java offre deux packages implémentant le patron observer :

- **Java.util.Observer :** c'est une interface qui comporte la fonction **update(),** cette fonction est exécutée quand un observer est notifié d'un changement.
- **Java.util.Observable :** comporte la classe *Observable* qui sert de classe mère pour toute classe qui veut avoir les fonctionnalités du model.

Elle comporte les fonctions et procédures suivantes :

- public void addObserver(Observer obs) : ajoute un observer a la liste interne d'observers.
- public void deleteObserver(Observer obs) : supprime un observer de la liste.
- public void deleteObservers() : vide la liste.
- public int countObservers() : retourne le nombre d'observers présents dans la liste.
- protected void setChanged() : modifie l'"indicateur interne de changement pour qu'il indique que l'objet a changé.
- protected void clearChanged() : efface les indicateurs internes de changement.
- public boolean hasChanged() : retourne « true » si cet observable a changé.

- public void notifyObservers() : avertir les observers d"un changement.
- public void notifyObservers(Object obj) : avertir les observers d'un changement.

3.5. Le patron de conception *Singleton*

3.5.1. Principe

Le patron Singleton permet de s'assurer qu'une classe n'a qu'une seule instance, tous les objets qui utilisent une instance de cette classe, utilisent la même instance [Che 07]. La structure du patron singleton est donnée par le diagramme de classe suivant :

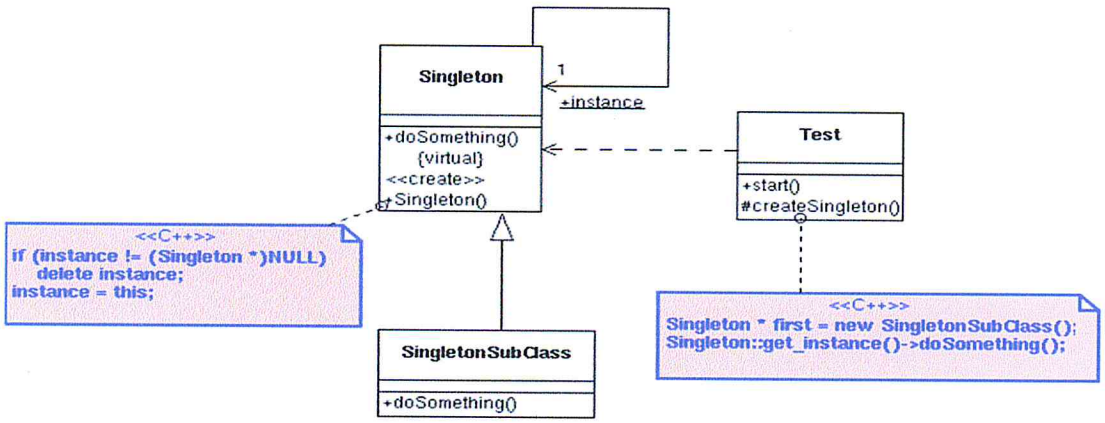


Figure II.8 : *Diagramme de classe (UML) exprimant la structure du pattern Singleton.*

3.5.2. *Singleton* en java

La classe doit comporter un attribut statique, généralement appelé *instance*, destiné à recevoir la référence de l'instance unique pour l'ensemble du logiciel, et une méthode, généralement appelée *getInstance*, renvoyant la valeur d'instance. Si *instance* est vide, *getInstance* crée une nouvelle instance de la classe en la stockant dans l'attribut instance et la renvoie à l'appelant.

Le code ci-dessous montre un exemple d'implémentation d'un singleton en Java :

```
public class MySingleton {
//...
static MySingleton instance = null;
protected MySingleton() {
//...
}
```

```
public static MySingleton getInstance() {
if (instance==null) {
instance = new MySingleton();
}
return instance;
 }
//
}
```

4. Système et architecture Client-serveur

4.1. Définition

L'architecture Client-serveur est un environnement de communication dans lequel des ***machines clientes***(des machines faisant partie du réseau) contactent un ***serveur***, une machine généralement puissante en termes de capacités d'entrée-sortie, qui leur fournit des services. Ces services sont des programmes fournissant des données telles que l'heure, des fichiers, une connexion, etc. Les services sont exploités par des programmes, appelés ***programmes clients***, s'exécutant sur les machines clientes

4.2. Le Client

En informatique, le mot ***client*** est utilisé dans le contexte du modèle ***client-serveur***. Il peut désigner une machine cliente ou un programme qui accède à un service sur une machine distante. Mais il peut aussi désigner l'utilisateur derrière cette machine ou le logiciel qui réalise cet accès.

Dans un réseau informatique un ***client*** est le logiciel qui envoie des actions relatives à la formulation des demandes, ainsi que pour la personne qui opère les demandes.

4.3. Le serveur

En informatique, par analogie, on appelle un ***serveur*** une machine ou un programme qui offre un service à un ***client***, par exemple : un serveur Web, un serveur de fichiers, etc.

Service: Toute ressource (donnée, fichier de ressources (de données traitées ou emmagasinées, de contrôle, d'informations système (temps CPU)), etc.

4.4. Fonctionnement d'un Système Client-serveur

Un système ***client/serveur*** fonctionne selon le schéma suivant :

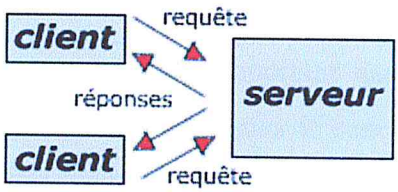


Figure II.9 : *La communication client/serveur.*

- ***Le client*** émet une requête vers le serveur grâce à son adresse IP et le port, qui désigne un service particulier du serveur.
- ***Le serveur*** reçoit la demande et répond à l'aide de l'adresse de la machine cliente et son port.

4.5. Architecture Client-serveur

L'architecture ***client/serveur*** désigne un mode de communication entre plusieurs ordinateurs d'un réseau pour une demande de service localisé au niveau d'un serveur de la part d'un ou de plusieurs clients.

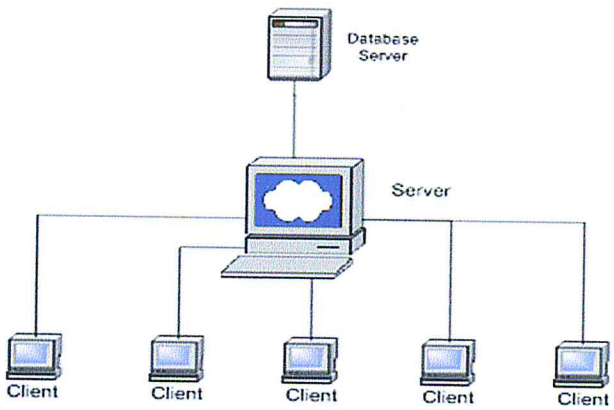


Figure II. 10 : *L'architecture client/serveur.*

CHAPITRE 3

Conception et implémentation

1. Introduction

Dans cette partie nous allons aborder notre travail qui concerne l'utilisation des techniques du génie logiciel dans l'optimisation d'un logiciel de DW.

Les techniques d'optimisation utilisées sont basées sur des techniques de l'orientée objets, des patrons de conception, et de la re-fabrication (refactoring).

Pour concevoir notre travail on va utiliser UML comme un langage de modélisation objet par ce qu'il riche par le nombre de diagrammes qu'il offre. Mais nous on s'est contenté dans notre travail de trois diagrammes seulement (le diagramme de classe, de séquence, et de cas d'utilisation).

2. Schéma architectural de notre travail

L'architecture du DW après avoir effectué des transformations sujet de notre étude est scindée en deux parties et mettant en jeu les composants essentiels suivants:

- **coté source de données**

  Dans ce côté, on trouve les parties suivantes :

  - Les sources de données sont sous les SGBDs MySQL et PostgreSQL.
  - l'outil ETL, pour l'extraction et la transformation des données sources.
  - Un serveur du coté sources de données, pour l'acheminement des données extraites par l'outil ETL sur réseau.
  - Une interface utilisateur graphique du côté source de données pour consulter et faire des mises à jour (MAJ) sur différents tables.

- **coté DW**

Le coté DW qui va comporter les parties suivantes :

- Un Client, pour la réception des données envoyées par le serveur sur réseau.
- Un organisateur de données, pour l'organisation ETL (rajout d'une valeur temporelle par exemple).
- Un intégrateur dans la table de fait.
- DW, pour centralisé des informations qui concerne la table de fait et les dimensions.
- Méta donnée, pour présenter une vision globale sur les données aux administrateurs et aux utilisateurs.
- Quering, pour faire des requêtes sur la table de fait et les dimensions.
- Les outils d'analyse et d'interrogation (Reporting), pour faire les histogrammes et les diagrammes (diagramme en camembert,…).
- Une interface utilisateur du côté DW, pour construire facilement les requêtes(Quering), et la visualisation des résultats (Reporting) pour les décideurs (directeurs, chef d'entreprises, …).

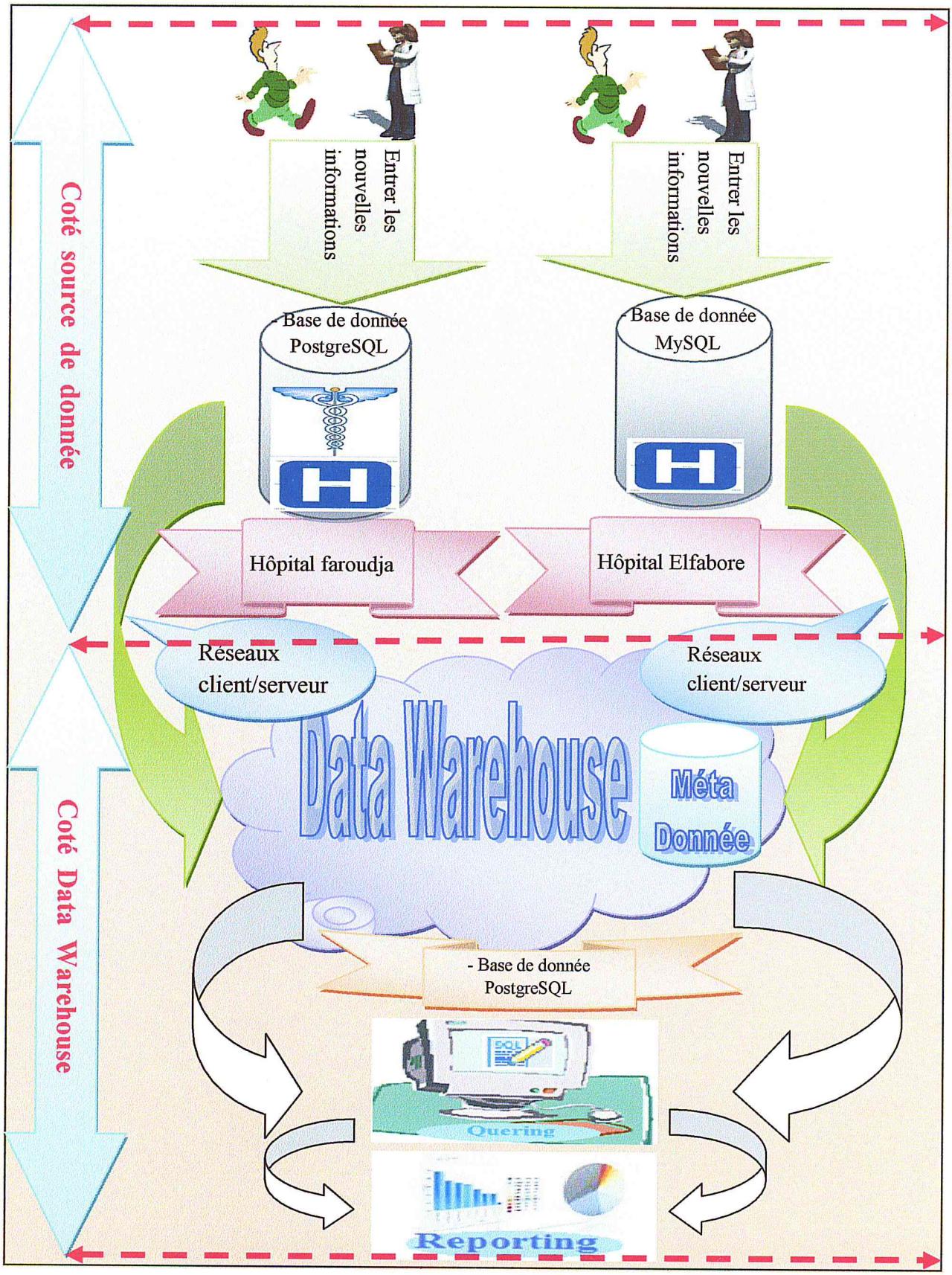


Figure III.1 : *Schéma générale.*

3. Coté source de données

3.1. Les sources de données

La première étape consiste à identifier les sources de données qui sont nécessaires et disponibles pour tous les faits et toutes les dimensions de l'entrepôt de données. Nous devons également connaître les caractéristiques de la source de données, telles que son type de fichier, structure de l'enregistrement, et l'accessibilité.
Dans notre travail nous travaillons sur deux sources de données MySQL et PostgreSQL, ce sont deux bases de données relationnelles, nous avons créé les tables SQL dans les deux bases de données.

3.1.1. Modèle du domaine (classe d'analyse)

Le modèle du domaine après analyse du domaine sert à faire ressortir des objets liés au domaine, et explique le déroulement de ce dernier par rapport à la réalité donc nous avons représenté le domaine par un diagramme de classe. Le diagramme de classe exprime de manière générale la structure statique de la base de données. Il consiste à définir les classes d'objets et les relations entre elles en se basant sur la connaissance générale du domaine hospitalier considéré et sur les règles de gestion [Dra10].

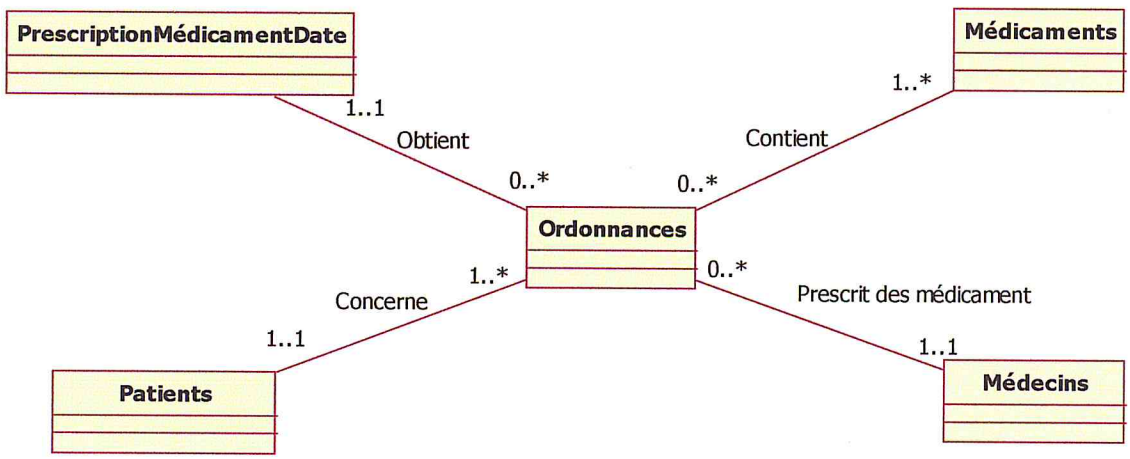


Figure III.2 : *Diagramme de class de modèle d'analyse.* [Dra10].

3.1.2. Dictionnaire des objets du domaine

Classe	Attributs	Code attributs	Méthodes
Patients	code patient Nom patient Prénom patient Date naissance patient Sexe patient Ville patient Wilaya patient Téléphone patient Adresse patient Profession patient Groupe sanguin patient Poids patient Taille patient Hôpital patient	**CodePt : *Int*** Nom : *String* Prenom : *String* DateNaiss : *Date* Sexe : *Char* Ville: *String* Wilaya: *String* Telephone : *String* Adresse : *String* Profession: *String* GroupSang : *String* Poids :*Int* Taille : *Int* Hopital: *String*	AjouterPatient() SupprimerPatient() ModifierPatient()
Médecins	Matricule médecin Nom médecin Prénom médecin Date naissance Adresse médecin Téléphone médecin Sexe médecin Nationalité médecin Grade médecin Spécialité médecin Service médecin Email médecin Hôpital médecin	**MatMed : *Int*** Nom: *String* Prenom : *String* DateNaiss: *Date* Adresse : *String* Telephone : *String* Sexe : *String* Nation : *String* Grade : *String* Spécialité : *String* Service : *String* Email : *String* Hôpital : *String*	AjouterMedecin() ModifierMedecin( ) SupprimerMedeci n()
Ordonnance	Numéro ordonnance	**NumOrd : *Int***	AjouterOrd() SupprimerOrd()
PrescriptionMédicamentDate	Code de la date La date Le jour Le nom de mois Le numéro de mois La quarter de mois L'année	**Datesk : *Int*** Date : *Date* Jour : *Int* MoisNom : *String* Mois : *Int* Quarter : *Int* Année : *Int*	
Médicaments	Code médicament Libellé médicament Nom médicament Date fabrication Date Périmassions Prix médicament Etat médicament Service médicament Fabricant médicament	**CodeMedi : *String*** Libel : *String* Nom : *String* DateFab : *Date* DatePér : *Date* PrixMedi : *Int* EtatMedi : *Int* ServiceMedi : *String* Fabricant : *String*	AjouterMedi() ModifierMedi() SupprimerMedi()

	Distribution	Distribut : *String*	LireMedi()

Tableau III.1 : *dictionnaire des objets du domaine.*

3.1.3. Modèle de conception

Le nombre de médicament prescris n'est pas fixé, il peut être de un à plusieurs médicament au niveau de table, il devrait normalement se trouver dans l'ordonnance. Il se trouve que le nombre de médicaments dans l'ordonnance est variable d'une ordonnance à une autre, et on ne sait pas définir une table avec un nombre de colonne variable.

Donc on rajoute une classe (un objet artificiel) ***MédicamentsPrescris*** qui va faire le lien entre l'ordonnance et le médicament qui à étais inscrit on l'appelle ***MédicamentsPrescris***. Cette solution est intéressent parce qu'elle nous permet de déplacer le problème du lien de l'Ordonnance avec le Médicament à MédicamentsPrescris avec l'ordonnance [Dra10]..

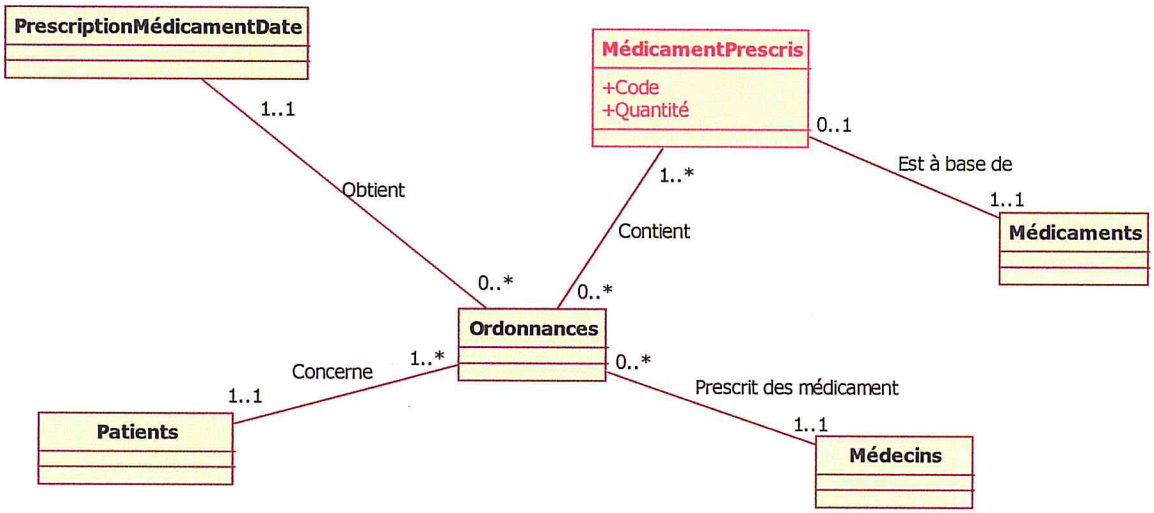


Figure III.3: *Diagramme de classe de modèle de conception.* [Dra10].

3.1.4. Le modèle de données relationnel

- **Médicaments** (*CodeMedi*, Libel, Nom, DateFab, DatePér, PrixMedi, EtatMedi, ServiceMedi, Fabricant, Distribut).
- **Patients** (*CodePt*, Nom, Prenom, DateNaiss, Sexe, Ville, Willaya, Telephone, Adresse, Profession, GroupSang, Poids, Taille, Hopital).

- **Médecins** *(MatMed*, Nom, Prenom, DateNaiss, Adresse, Telephone, Sexe, Nation, Grade, Spécialité, Service, Email, Hôpital).
- **PrescriptionMédicamentDate** *(DateSk*, Date, Jour, MoisNom, Mois, Quarter, Année)
- **Ordonnance** (*NumOrd*, *codePt#*, *MatMed#*, *DateSk#)*
- **MédicamentPriscris** (*Code*, *CodeMedi#*, *NumOrd#*, Quantité)

___ représente la clé primaire de la table.

___# représente la clé étrangère de la table.

Les principales informations de la source de données et de l'entrepôt de données (DW) sont présentées au tableau suivant qui s'appelée la carte source de données, car elle carte englobe toutes les données de la source à la cible.

Source	Type	Les noms des tables	Les tables dans l'entrepôt de données
La base de données patiente	**MySQL**	Ordonnances Médecins MédicamentPrescris Médicaments PrescriptionMédicamentDate Patients	OrdonnanceDim MédecinDim MédicamentPrescriDim MédicamentDim PrescriptionMédicamentDateDim PatientDim PrescriptionMédicamentFactTable
La base de données patients 1	**PostgreSQL**	Ordonnances Médecins MédicamentPrescris Médicaments PrescriptionMédicamentDate Patients	OrdonnanceDim MédecinDim MédicamentPrescriDim MédicamentDim PrescriptionMédicamentDateDim PatientDim PrescriptionMédicamentFactTable

Tableau III.2: *La carte de la source de données.*

3.2. L'outil ETL

Cette partie, porte sur le processus ETL et a pour rôle d'extraire (Extract) les données dont nous avons besoin pour le DW, à partir des bases de donnée sources, de les transformer (filtrage, tri, homogénéisation, nettoyage…). Ensuite de les charger (Load).

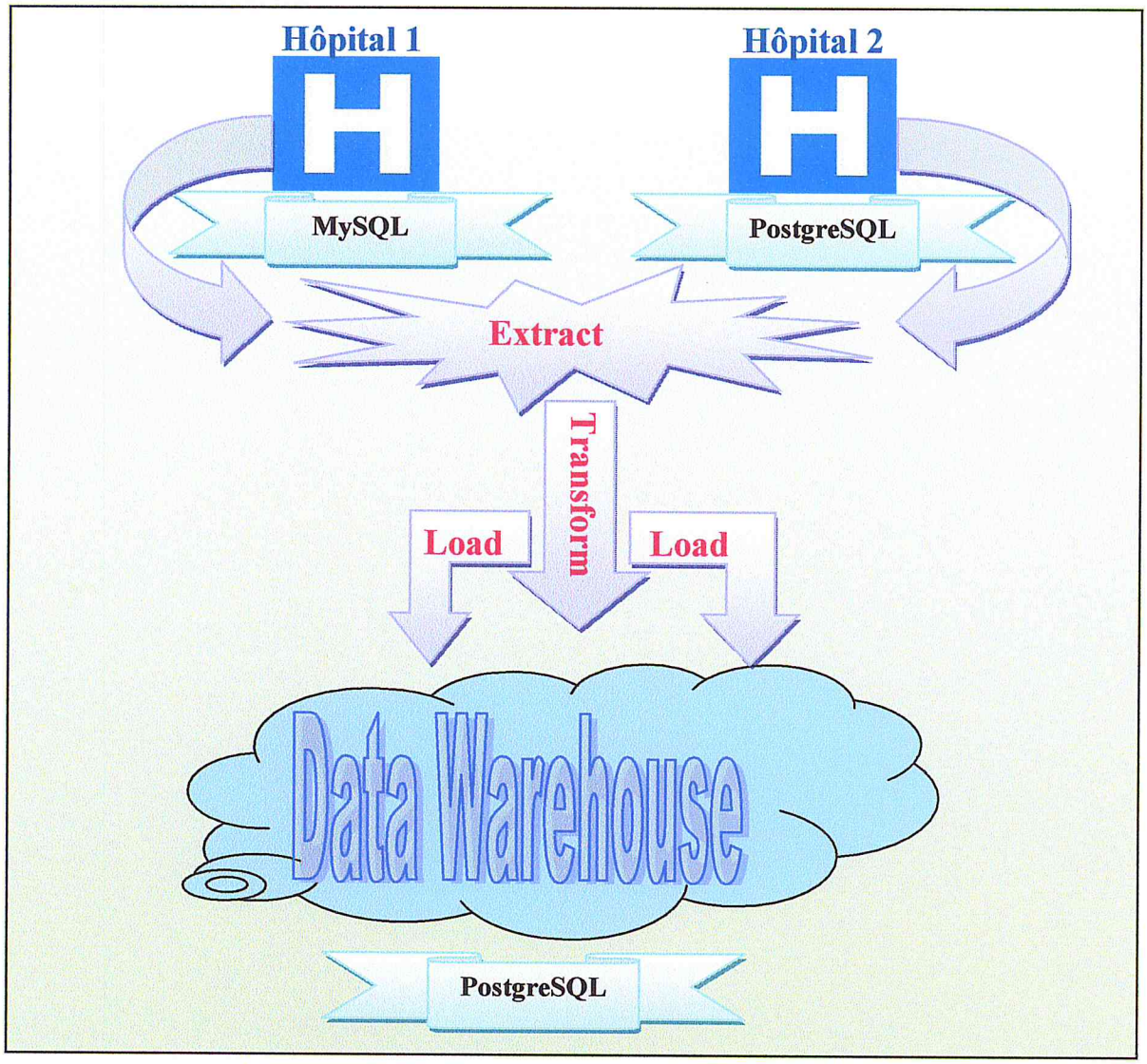


Figure III.4: *ETL*

3.3. Un serveur de bases de données

Un serveur de bases de données a pour but d'offrir une réponse rapide aux requêtes (Indépendamment de la taille de la base de données et la complexité).

Ceci peut être réalisé grâce à l'utilisation d'une architecture *Client/Serveur*. Pour réaliser cette technique nous avons utilisé le protocole TCP parce que elle garantit la réception de données dans l'ordre d'envoie et nous avons deux serveurs un pour l'hôpital de Faroudja et l'autre pour l'hôpital Elfabore.

Nous avons codé les données des sources dans le serveur de la base de données (dans chaque base de donnée MySQL et PostgreSQL) qui sont au format des enregistrements de tableau à des chaines de caractère le début de chaque chaine c'est le nom de la table correspondant et envoyer cette chaine au serveur client mais dans notre cas le serveur c'est lui

qui envoyer les données automatiquement, ce n'est pas le client qui demande au serveur d'envoyer les données, donc à chaque fois où il y a de nouvelle donnée notre serveur envoyer directement la donnée aux différents clients dont le rôle est justement de reconnaître les données qui leurs sont prédestinées pour les récupérer et les traiter chacun à sa manière.

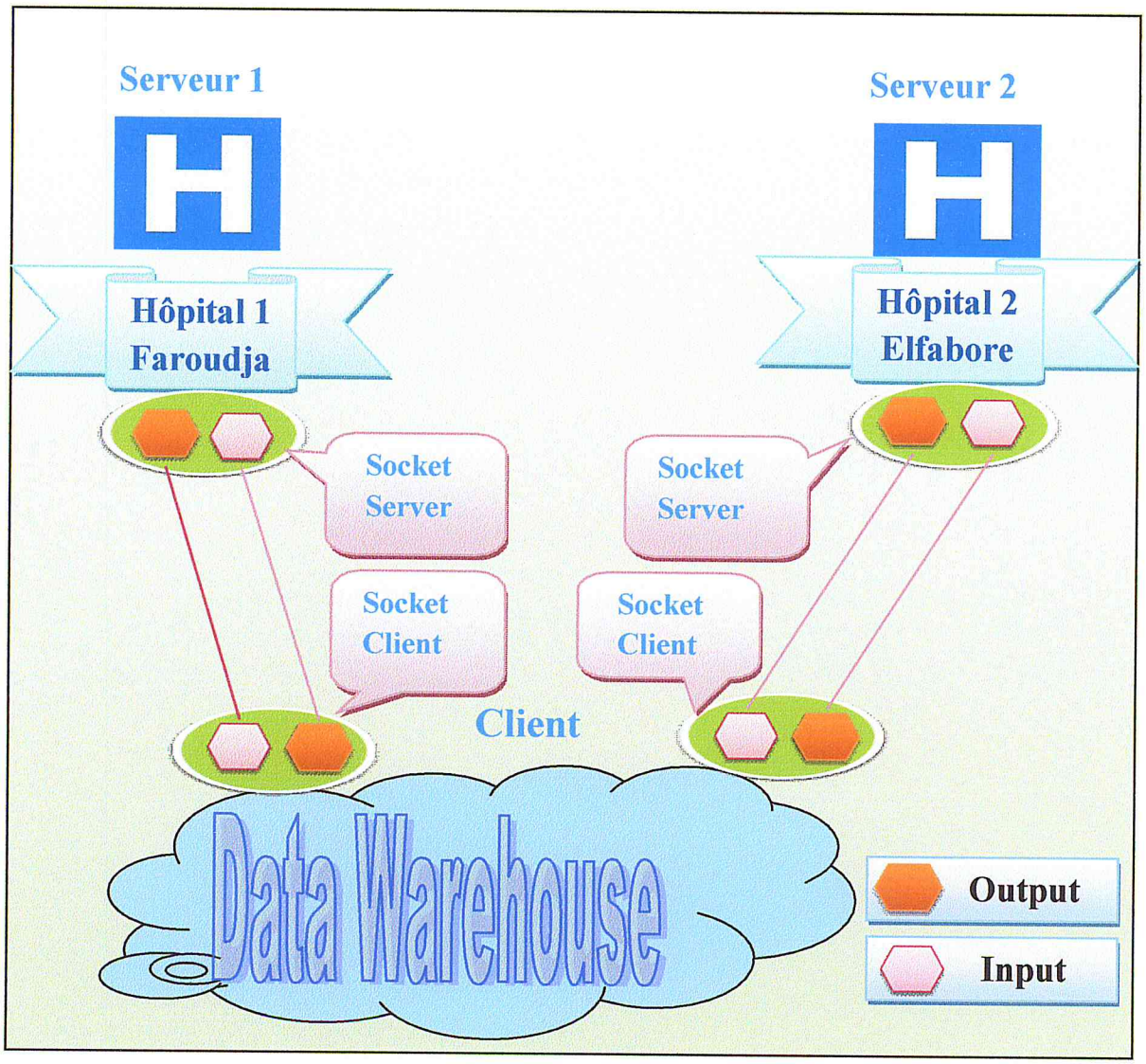


Figure III.5: *Le mode Client/Serveur.*

3.4. Capture des besoins

L'objectif est de déterminer et de modéliser ce que doit faire le système. Pour cela, UML nous donne cette possibilité grâce au diagramme de cas d'utilisation.

3.4.1. Diagramme de cas d'utilisation

Il permet de facilement exprimer graphiquement les différentes fonctionnalités du système et l'interaction des acteurs avec le système [Gab08].

Les cas d'utilisation constituent un moyen de recueillir et décrire les besoins des acteurs du système. Ils peuvent être aussi utilisés ensuite comme moyen d'organisation du développement du logiciel, notamment pour la structuration et le déroulement des tests du logiciel [Gab08].

3.4.2. Identification des cas d'utilisation

Le tableau suivant liste les cas d'utilisation :

Cas d'utilisation	Acteur	Résumé
Authentification	Utilisateur	S'authentifier pour accéder au système, selon le droit d'accès
Consultation générale	Utilisateur	Consulter Ordonnances, Médecins, MédicamentPrescris, Médicaments, PrescriptionMédicamentDate, Patients
Mise à jour	Utilisateur	Mettre à jour Ordonnances, Médecins, MédicamentPrescris, Médicaments, PrescriptionMédicamentDate, Patients

Tableau III.3 : *Liste des cas d'utilisation.*

- **Diagramme de cas d'utilisation global**

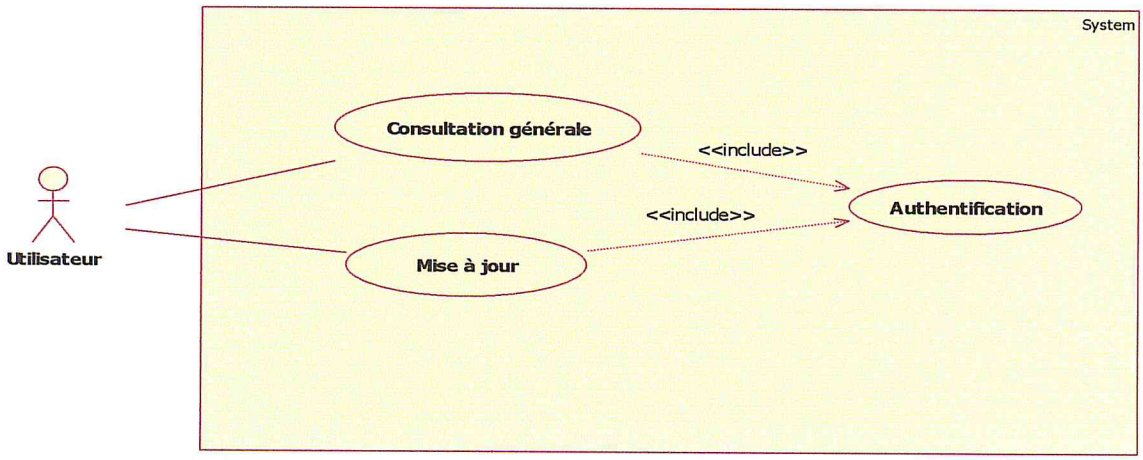


Figure III.6 : *Diagramme de cas d'utilisation global.*

Cas d'utilisation «Authentification» :

Avant refactoring

Il n'existait pas d (authentification, tout utilisateur pouvait lancer le programme directement par la classe *Test3333*

Après refactoring

Titre : Authentification

But : Accéder au menu principal (IHM).

Résumé : l'utilisateur du système doit entrer son couple (identifiant/mot de passe) pour accéder au menu principal (IHM) et bénéficier des fonctionnalités de system.

Acteurs : utilisateur.

Pré conditions :

- Introduit son identifiant et son mot de passe

Enchaînements :

1) L'utilisateur accède au système.
2) Il introduit son identifiant et son mot de passe dans la zone d'identification

Post conditions :

- L'authentification est établie

Figure III.7: *Description du cas d'utilisation « Authentification ».*

Cas d'utilisation « Consultation générale »

Avant refactoring

Il n'existait pas, l'utilisateur consultait les tables de la base de données directement à partir de MySQL et de PostgreSQL

Après refactoring

Titre : Consultation générale

But : Consultation.

Résumé : l'utilisateur du système peut consulter Ordonnances, Médecins, MédicamentPrescris, Médicaments, PrescriptionMédicamentDate, Patients

Acteur : utilisateur.

Pré conditions :

- L'utilisateur doit être authentifié.

Enchaînements:

1. Le système affiche la liste des informations qui peuvent être consultés.
2. L'utilisateur sélectionne une information.
3. Le système affiche le résultat.

Post condition :

- Aucune

Figure III.8 : *Description du cas d'utilisation « Consultation générale ».*

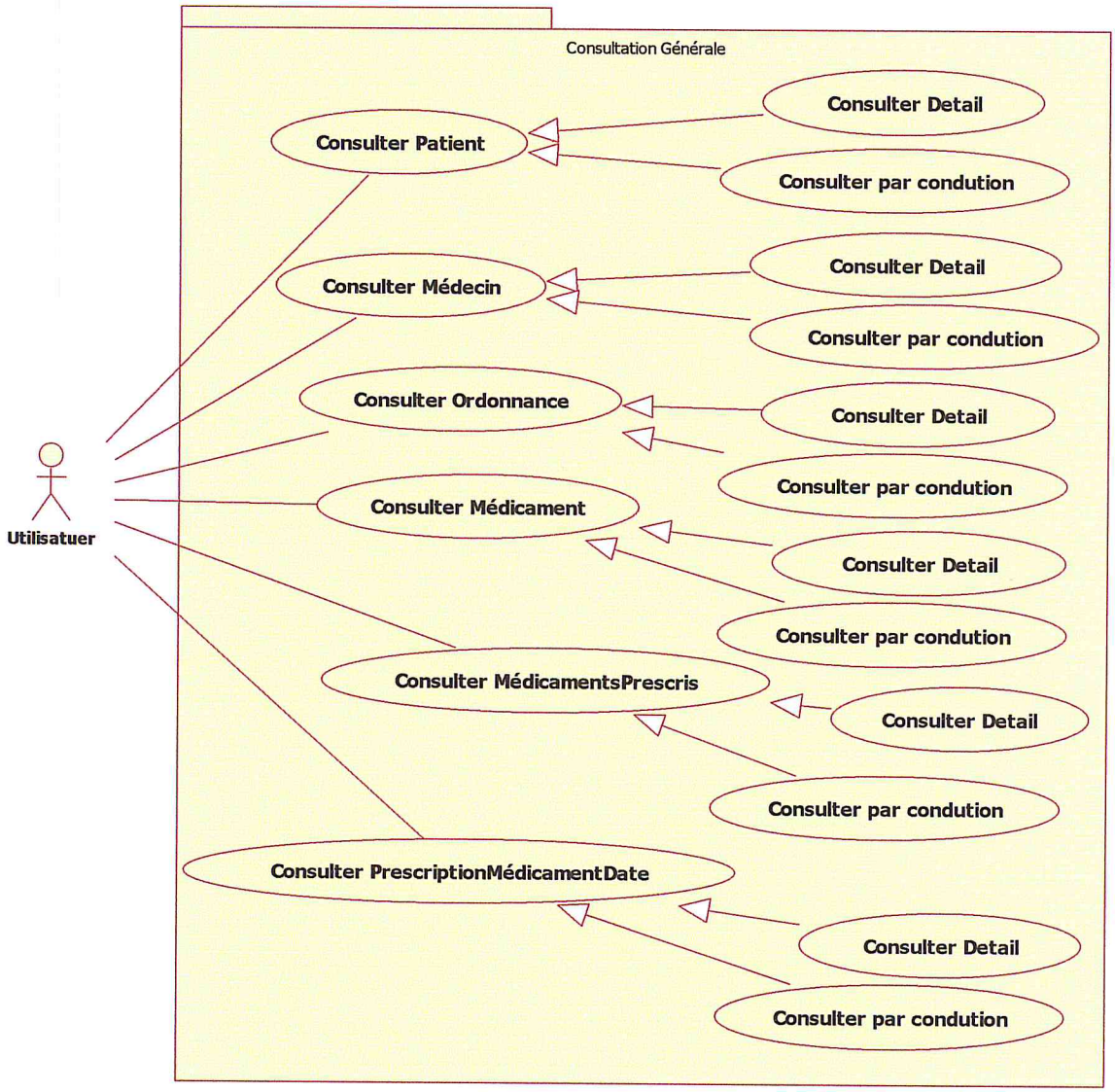


Figure III.9 : *Diagramme de cas d'utilisation « consultation générale ».*

- **Cas d'utilisation « MISE À JOUR »**

Avant le refactoring

L'utilisateur pouvait ajouter dans les tables directement à partir du code dans la méthode *main* par la méthode[Dra10]

insertField(table,[] name Field, [] valueField) de la classe DB

Après refactoring

Titre : Mise à jour.

But : la Mise à jour.

Résumé : l'utilisateur peut modifier, supprimer ou créer Ordonnances, Médecins, MédicamentPrescris, Médicaments, PrescriptionMédicamentDate, Patients

Acteur : utilisateur

Pré conditions :

- L'utilisateur doit être authentifié.

Enchaînements

1. L'utilisateur sélectionne une opération (ajout, modification, ou suppression).
2. Le système affiche le formulaire.
1. Le système vérifie ensuite valide les informations entrées.
2. Le système affiche la nouvelle liste.

Post condition :

- Aucune

Figure III.10 : *Description du cas d'utilisation « Mise à jour ».*

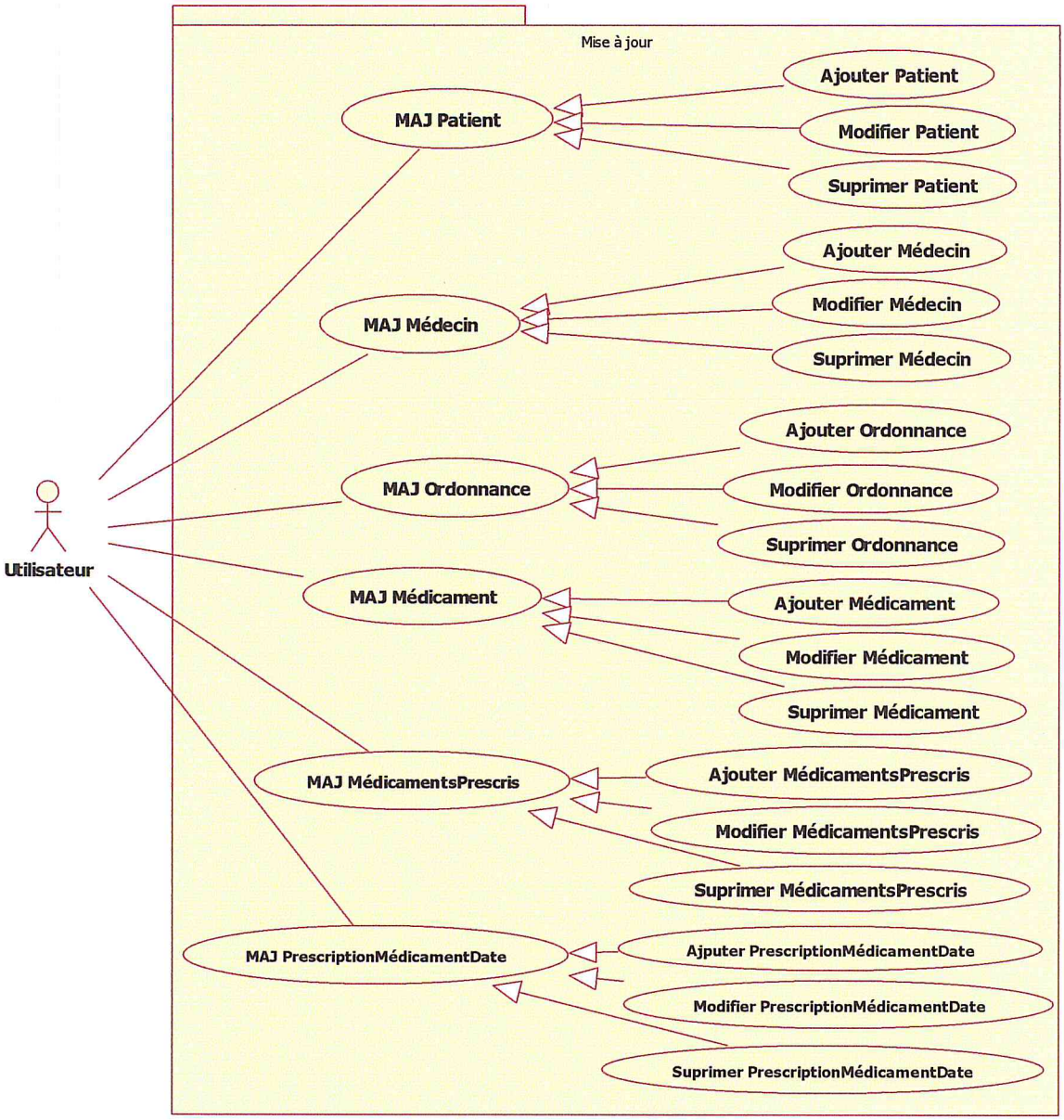


Figure III.11: *Diagramme de cas d'utilisation « Mise à jour ».*

3.5. La création et la MAJ des tables dans la base de données opérationnel

Avant refactoring[Dra10]

- **La classe *DB* :**

Cette classe joue un rôle très important dans la création et la MAJ des tables. On pourrait dire que c'est la classe générale par laquelle passe tous les traitements concernant les bases de données.

Elle est conçue comme un patron *Singleton*, pour la simple raison, qu'on n'a prévu qu'une seul générateur des tables pour un même environnement.

Elle est à la base d'une caractéristique importante du domaine de l'entrepôt de donnée, qui est la création, la suppression et la mise à jour. Donc la classe *DB* est à la base de l'insertion la suppression et la mise à jour de toutes les tables de serveur MySQL (L'hopital1) ou PostgreSQL (Hopital2).

On peut dire donc, que pour tous les *Tables* de MySQL ou PostgreSQL, il ne doit y avoir qu'un seul objet qui est utilisé pour insérer ou supprimer ou faire un mis à jours. La classe *DB* est conçue aussi au moyen du patron *Observer*. Elle est Observable de la classe *SocketServer* pour lui transmettre les changements effectués instantanément d'où le coté temps réel du DW.

- **La classe *AConnection***

C'est une classe *Abstraite* qui à tous les méthodes nécessaire pour établir une connexion avec la base de donnée.

La classe *AConnection* définit donc les principales méthodes de connexion.

- **La classe *DBTable***

Cette classe est une classe *abstraite*, elle a extrait de la super classe *Hashtable*.

A part quelques modifications mineures, a été implémentée par belgasmia [Bel09].

- **La classe Médicaments**

C'est la classe qui contient tous les champs (les informations) de la table *médicaments* de la base de données.

- **La classe Ordonnances**

C'est la classe qui contient tous les champs (les informations) de la table *ordonnances* de la base de données, elle est présentée comme la classe *Médicaments*.

- **La classe *Patients* :**

C'est la classe qui contient tous les champs (les informations) de la table *patients* de la base de données, elle est représentée comme la classe *Médicaments*.

- **La classe *Médecins* :**

C'est la classe qui contient tous les champs (les informations) de la table *médecins* de la base de données, elle est représentée comme la classe *Médicaments*.

- **La classe *MédicamentsPriscris* :**

C'est la classe qui contient tous les champs (les informations) de la table *médicamentsPriscris* de la base de données, elle est représentée comme la classe *Médicaments*.

- **La classe *PréscriptionMédicamentDate*:**

C'est la classe qui contient tous les champs (les informations) de la table *prescriptionMédicamentDate* de la basse de donnée (représente la date).

- **La classe *SocketServer*:**

C'est un Observer de la classe *DB*, elle a pour but envoyer les données aux Socket client (serveur de client).

- **La classe *TestDrop* :**

Cette classe est utilisée pour supprimer les tables qui sont dans la base de données.Nous avons ordonné les tables dans une sorte qu'on peut les supprimer, parce que il y a des tables qui dépend des autres tables (la clé primaire d'une table1 c'est une clé étrangère une autre table2).

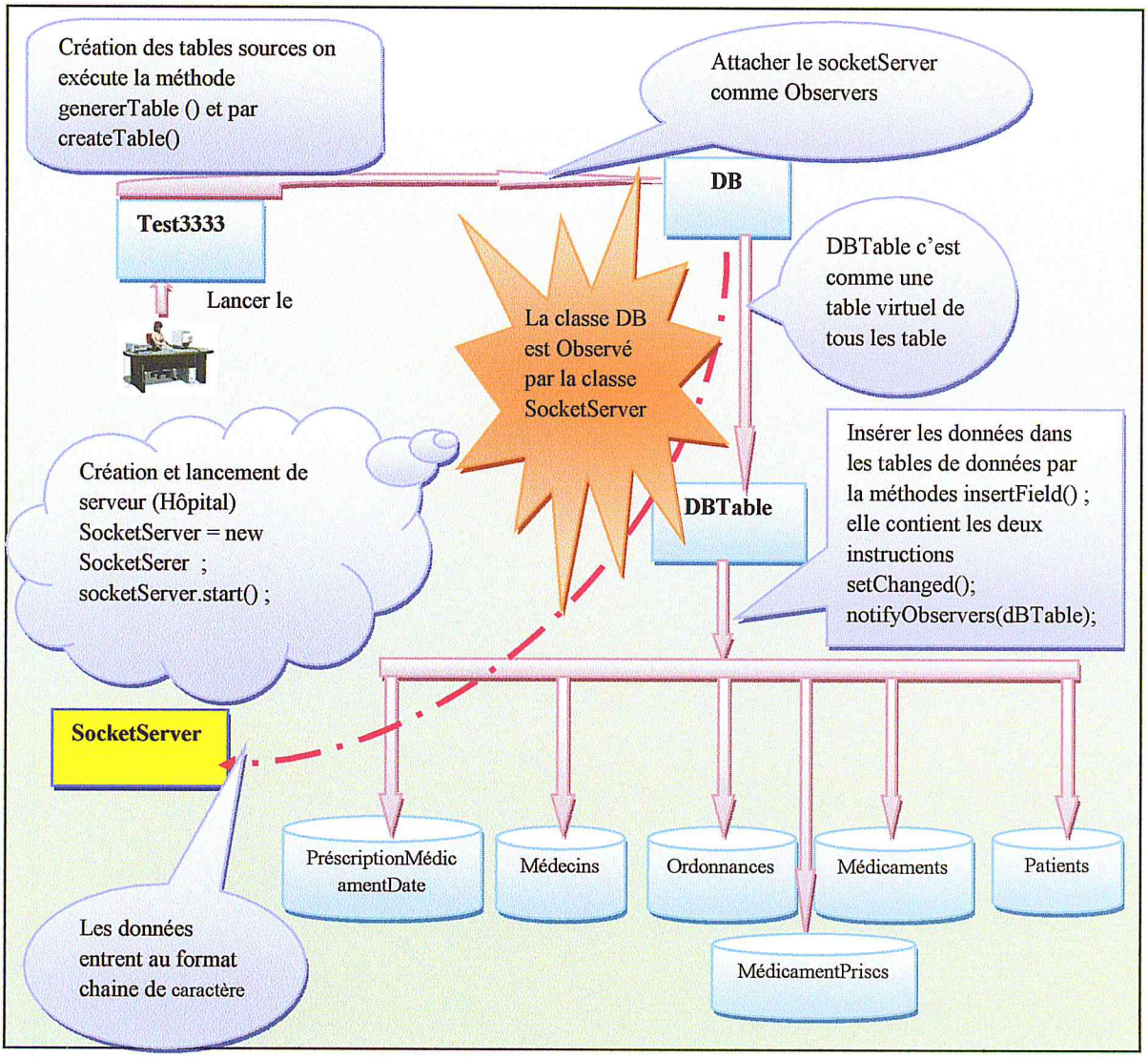


Figure III.12: *Le cheminement des données avant refactoring.*[Dra10]

Après refactoring

On va ajouter les classes suivantes :

- **La classe *IHH* :**

  C'est un *Observer* de la classe *Authentification*, elle contient le menu principal (consultation, mise à jour) du serveur (hôpital).

- **La classe *MAJ* :**

  La classe *MAJ* a pour but de remplir les champs (les informations) des tables de la base de données du serveur (hôpital).

- **La classe *Authentification* :**

  C'est la classe qui contient va lancer la méthode *main ()* après authentification, ici on va remplir le formulaire d'authentification.

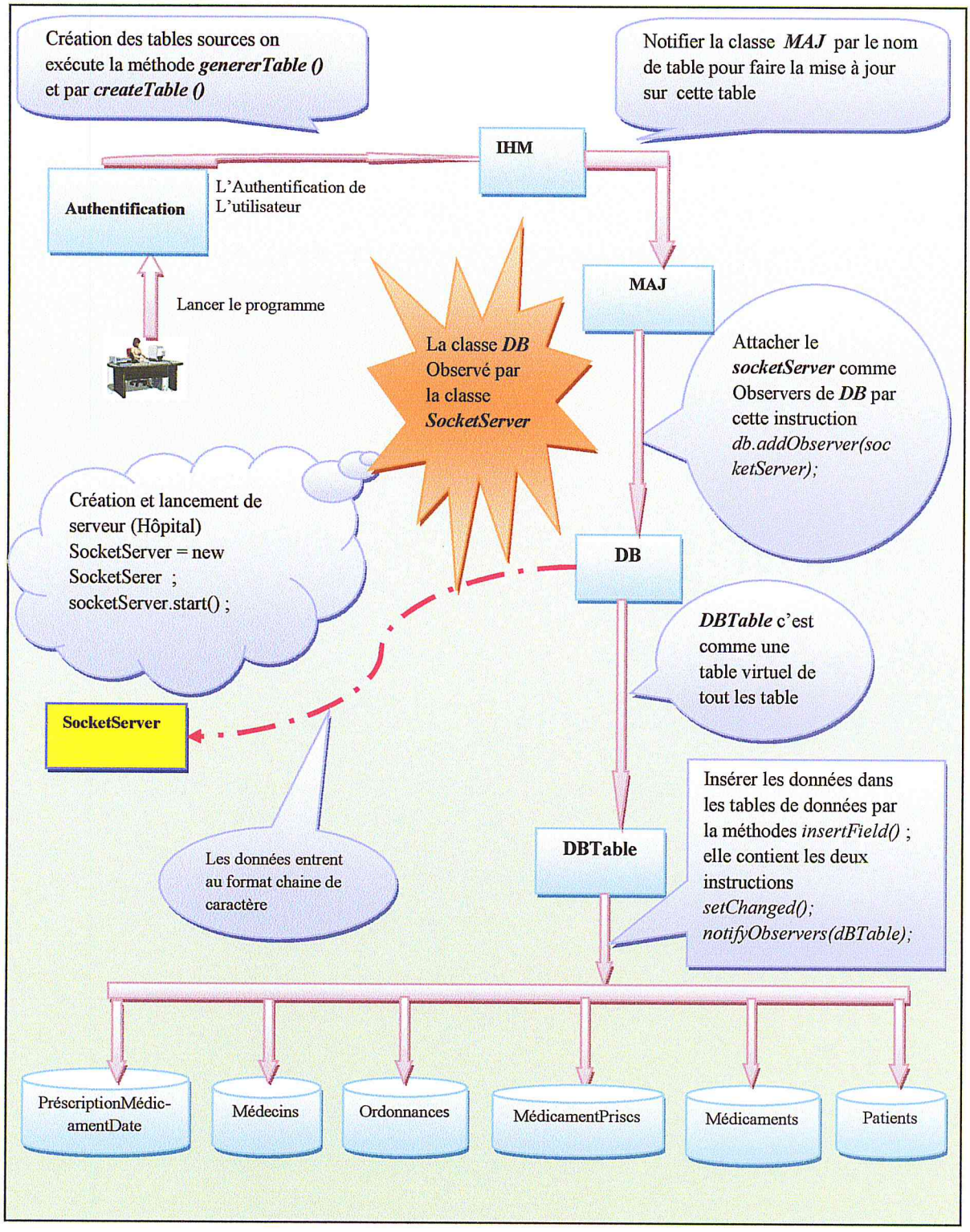


Figure III.13: *Le cheminement des données après refactoring.*

3.6. Diagrammes de séquences

L'objectif du diagramme de séquence est de représenter les interactions entre les objets d (un système en indiquant la chronologie des échanges [Gab08].

- **« Authentification »**

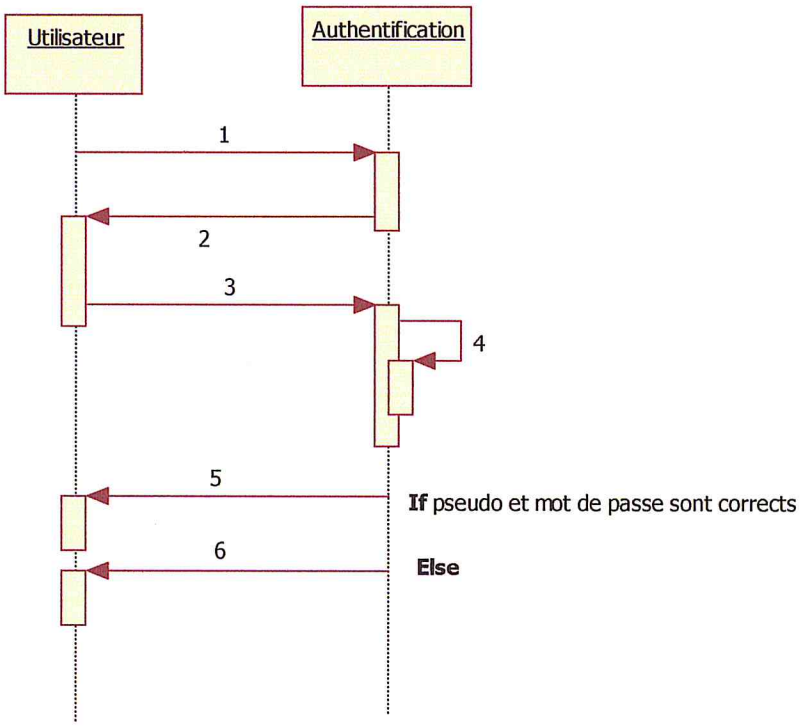


Figure III.14 : *Diagramme de séquence « Authentification ».*

1- lancer le programme.

2- Afficher le formulaire d'authentification.

3- Saisir les champs.

4- Vérifier.

5- autoriser la Connexion.

6- afficher un Message d'erreur.

- **« MAJ »**

Avant refactoring

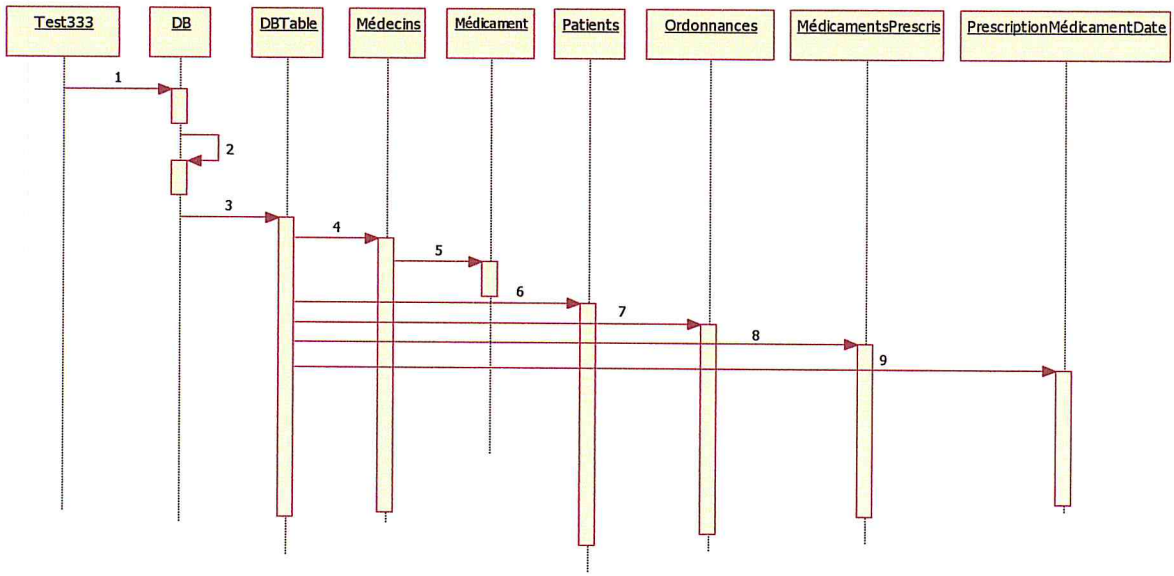


Figure III.15 : *Diagramme de séquence « MAJ »avant refactoring*[Dra10].

1 – Lancer le programme de serveur d'hôpital.

2 – La classe *DB* est un Observable quand la classe *DB* reçu un signe elle change l'état.

3 – La classe *DB* doit notifier la classe *DBTable* qui représente la table virtuel des tables physique.

4 – la classe *DBTable* informe la classe *Médecins* pour crée et remplir la table *médecins.*

5 – la classe *DBTable* informe la classe *Médicaments* pour crée et remplir la table *médicaments.*

6 – la classe *DBTable* informe la classe *Patients* pour crée et remplir la table *patients.*

7 – la classe *DBTable* informe la classe Ordonnances pour crée et remplir la table *ordonnances.*

8 – la classe *DBTable* informe la classe MédicamentPriscris pour crée et remplir la table *médicamentpriscris.*

9 - la classe *DBTable* informe la classe PrescriptionMédicamentDate pour crée et remplir la table *prescriptionMédicamentDate.*

Après refactoring

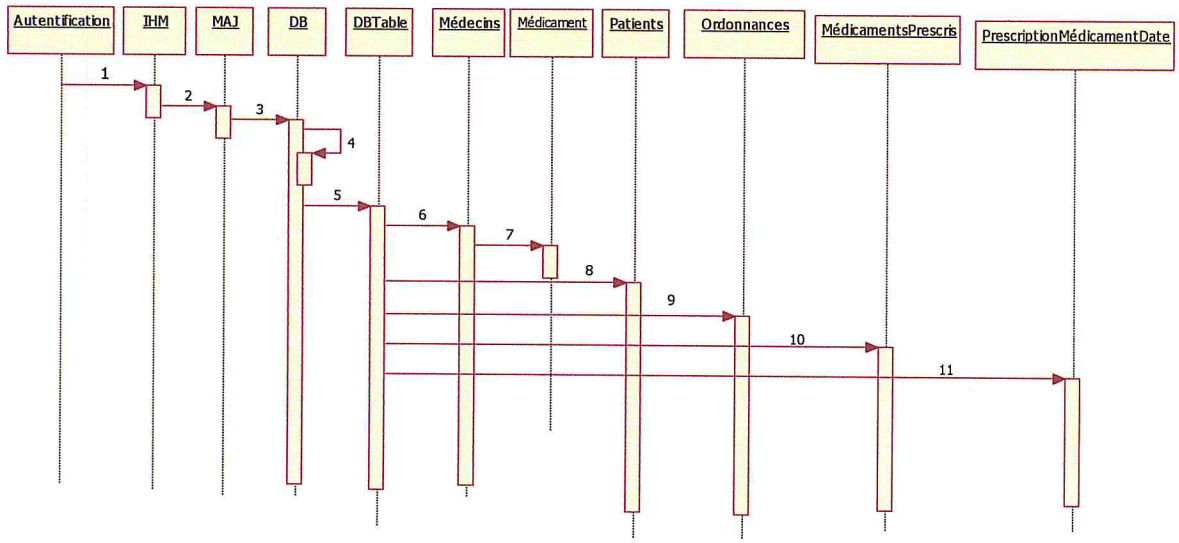


Figure III.16 : *Diagramme de séquence « MAJ »après refactoring.*

1 – Quand l'utilisateur entre un nom et mot de passe corrects la classe *authentification* notifie la classe *IHM* qui contient le menu principale.

2- la classe *IHM* va notifier la classe *MAJ* par le nom de table pour faire la mise à jour.

3-la classe *MAJ* notifie la classe *DB* qui contient les méthodes de mise à jour.

4 – La classe DB est un Observable quand la classe *DB* reçoit un signe elle change l'état.

5 – La classe DB doit notifier la classe *DBTable* qui représente la table virtuelle des tables physique.

6– la classe *DBTable* informe la classe *Médecins* pour crée et remplie la table *médecins*.

7 – la classe *DBTable* informe la classe *Médicaments* pour crée et remplie la table *médicaments.*

8 – la classe *DBTable* informela classe *Patients* pour créer et remplir la table *patients.*

9– la classe *DBTable* informe la classe *Ordonnances* pour créer et remplir la table *ordonnances.*

10 – la classe *DBTable* informe la classe *MédicamentPriscris* pour créer et remplir la table *médicamentpriscris.*

11 - la classe *DBTable* informe la classe *PrescriptionMédicamentDate* pour créer et remplir la table *prescriptionMédicamentDate*.

- **« suppression de la base de données »**

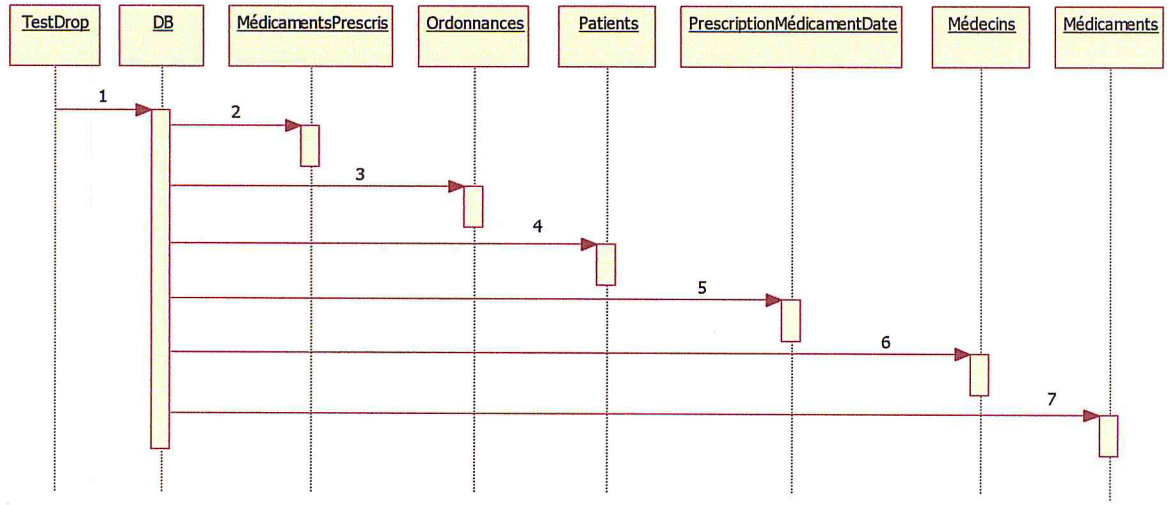


Figure III.17 : *Diagramme de séquence « suppression de la base de données ».*

1 – Nous avons lancé le système par la classe *TestDrop* et envoyé un signe à la classe *DB*.

2 – la classe *DB* va utiliser la méthode.

public voiddropTable(String tableName) throwsSQLException { } : pour supprimer la table *MédicamentPriscris.*

3 - la classe *DB* va utiliser la méthode :

public void dropTable(String tableName) throws SQLException { } pour supprimer la table Ordonnances.

4 – la classe *DB* va utiliser la méthode

public void dropTable(String tableName) throwsSQLException { } pour supprimer la table *Patients.*

5 – la classe *DB* va utiliser la méthode

public void dropTable(String tableName) throws SQLException { } pour supprimer la table *PrescriptionMédicamentDate.*

6 – la classe *DB* va utiliser la méthode

public void dropTable(String tableName) throwsSQLException { } pour supprimer la table *Médecins.*

7 - la classe *DB* va utiliser la méthode

public void dropTable(String tableName) throws SQLException { } pour supprimer la table *Médicaments.*

4. Coté DW

4.1 La modélisation de la partie client

Dans le coté client nous avons un client qui reçoit les données à partir de deux serveurs d'hôpitaux, donc quand la chaine de caractère arrive au client, ce dernier la décode pour voir de quel hôpital il s'agit, et elle provient et de quelle table. On doit mettre en œuvre un client multi-mode utilisateur / serveur.

La classe *SocketClient* :

La classe *SocketClient* est utilisée côté client : elle tente de communiquer avec le serveur. C'est un objet du type Socket qui prend en charge la transmission des données.

Cette classe représente la partie client du socket. Un objet de cette classe est associé à un port sur lequel il va établir une connexion à un serveur. Généralement, à l'envoie d'une demande de connexion, un thread est lancé pour assurer le dialogue avec le serveur. Il ne peut traiter qu'une connexion en même temps. Pour pouvoir traiter plusieurs connexions simultanément, il faut créer un nouveau thread contenant les traitements à réaliser sur le socket pour notre cas nous avons lancé deux threads parce qu'on a deux hôpitaux.

SocketServer (de serveur de l'hopital).

- Crée la Socket :

  Socket server = new Socket(host,port);

- Récupère les flux d'entrée et de sortie :

  InputStream in = server.getInputStream();

  OutputStream out = server.getOutputStream();

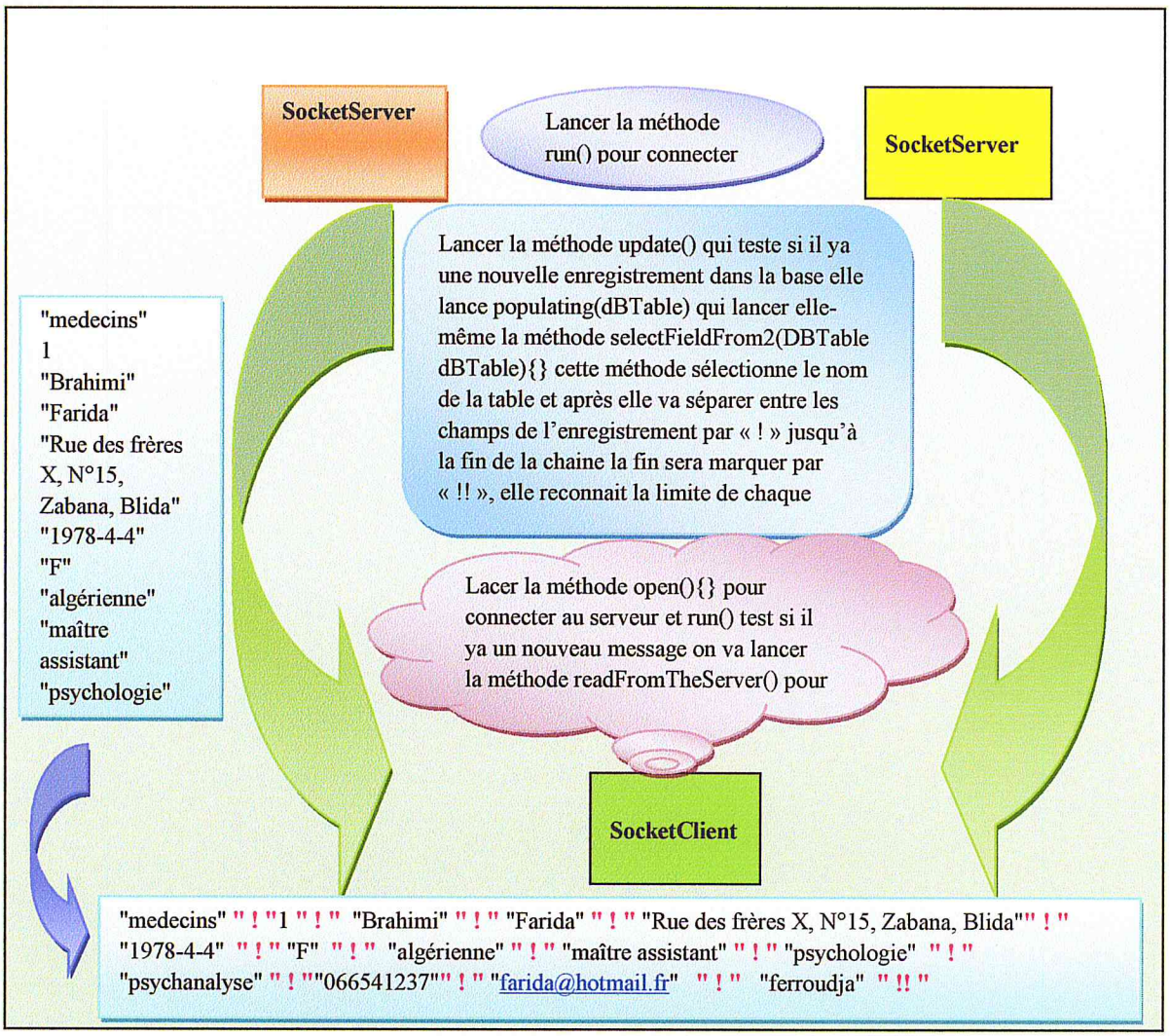


Figure III.18: *Le transfert des données entre le client et les deux serveurs.*

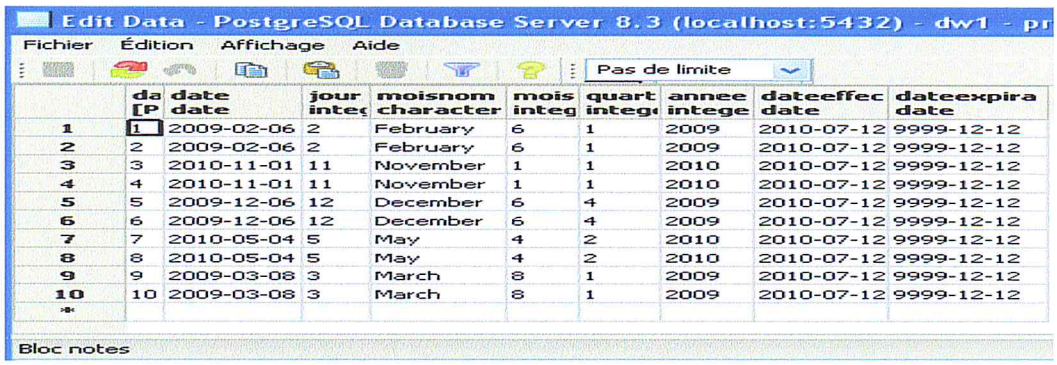
4.2. Remplissage de la dimension Date

La dimension date est « *la seule dimension qui figure systématiquement dans tout entrepôt de données, car en pratique tout entrepôt de données est une série temporelle.* » [Kim02].

Tout d'abord, une dimension de date contient les temps, et les temps sont d'une importance capitale car l'une des principales fonctions d'un entrepôt de données est de stocker les données historiques. Par conséquent, les données dans un entrepôt de données a toujours un aspect du temps. Un autre aspect unique d'une dimension de date, c'est que nous avons la possibilité de générer les dates pour remplir la dimension date de l'intérieur de l'entrepôt de données. Dans l'entrepôt de données, nous commençons par créer une table date_dim qui a une colonne

de date. Les valeurs dans ce tableau seront alors devenues une référence pour d'autres tableaux avec des valeurs de date. Dans notre entrepôt de données, la table **PrescriptionMédicamentDateDim** (la table date_dim) concerne la table **PrescriptionMédicamentFactTable** par la clé de substitution codedate.

En plus de la date elle-même, la dimension date a d'autres données, telles que le nom du mois et du trimestre. [Dra10]



	da [P	date date	jour inte	moisnom character	mois integ	quart integ	annee intege	dateeffec date	dateexpira date
1	1	2009-02-06	2	February	6	1	2009	2010-07-12	9999-12-12
2	2	2009-02-06	2	February	6	1	2009	2010-07-12	9999-12-12
3	3	2010-11-01	11	November	1	1	2010	2010-07-12	9999-12-12
4	4	2010-11-01	11	November	1	1	2010	2010-07-12	9999-12-12
5	5	2009-12-06	12	December	6	4	2009	2010-07-12	9999-12-12
6	6	2009-12-06	12	December	6	4	2009	2010-07-12	9999-12-12
7	7	2010-05-04	5	May	4	2	2010	2010-07-12	9999-12-12
8	8	2010-05-04	5	May	4	2	2010	2010-07-12	9999-12-12
9	9	2009-03-08	3	March	8	1	2009	2010-07-12	9999-12-12
10	10	2009-03-08	3	March	8	1	2009	2010-07-12	9999-12-12

Figure III.19: *La dimension PrescriptionMédicamentDateDim.*

4.3. Un organisateur de données [Dra10]

Pour organiser les données nous avons créé une méthode qui rajouter à chaque enregistrement qui vient de la source de donnée deux champs de la **dateEffective** et **dateExpiration** pour les médicaments et la dimension date et dateEntrer et dateSortie pour les patients et les médecins et pour remplir les tables de dimensions nous avons utilisé les patrons de conception Observer qui envoyer chaque information reviens au client à tous les tables de dimension et une et une seul prendre cette information et tout ça c'est par la vérification de début de la chaine si le début de la chaine envoyer correspond à ce que la dimension cherche donc cette dimension va prendre cette information.

4.4. Modèle physique de l'entrepôt

C'est le modèle en étoile qui a été choisi pour la conception de l'entrepôt de données.

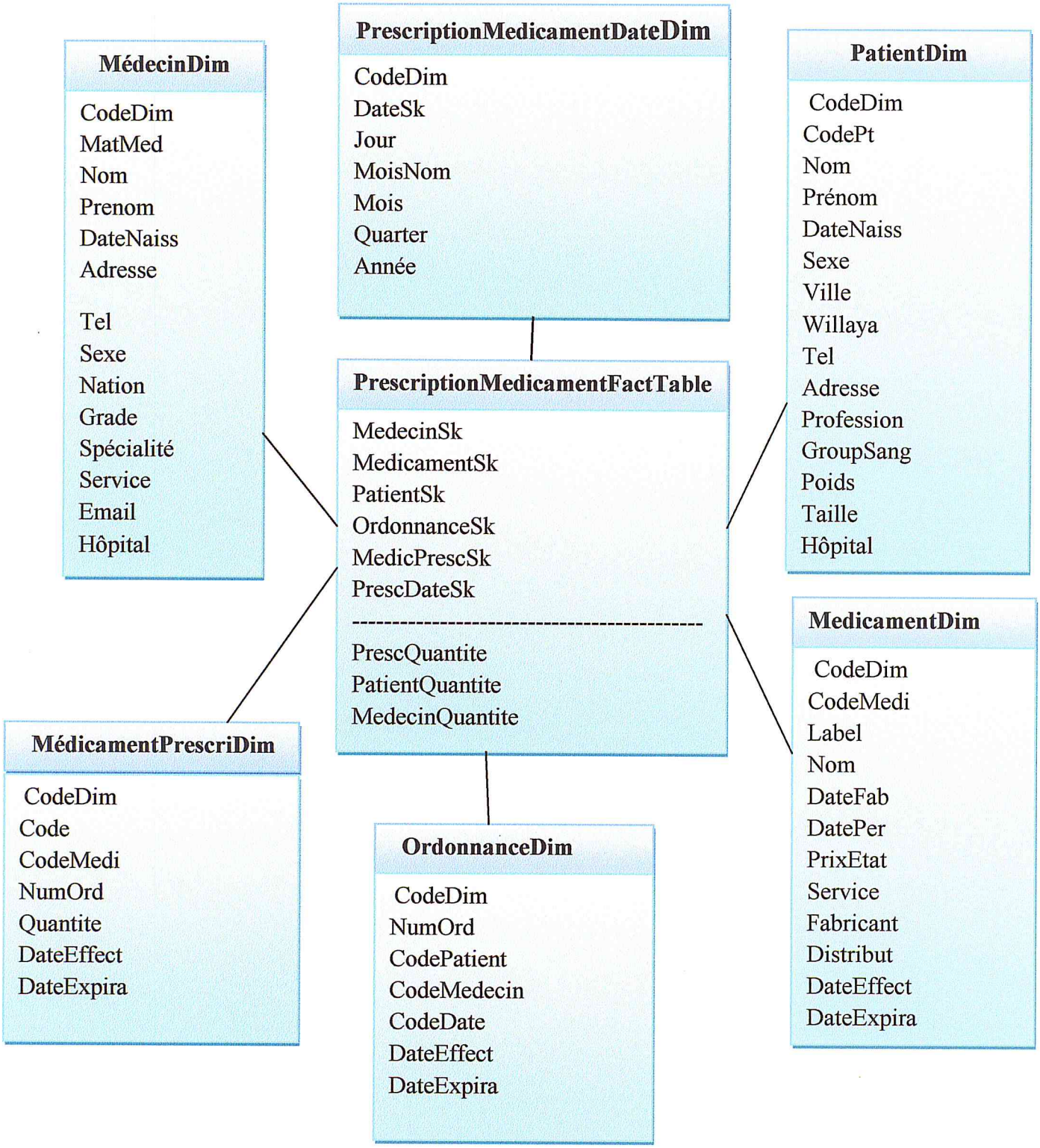


Figure III.20: *Schéma en étoile* [Dra10]

4.5. La création et le chargement de l'entrepôt de données

4.5.1. Avant Refactoring[Dra10]

- **La classe *AConnection* :**

C'est une interface pour faire une connexion à une base de données pour opérer des requêtes sur une table relationnelle, comme par exemple :

- *selectField(String table, String whereClause) throws SQLException:* Selection de champs.
- *public abstract booleaninsertField(String table,StringwhereClause) throws SQLException:*Insertion de champs.
- *public abstract void updateField(String table,StringwhereClause) throws SQLException :* Mise à jour de champs.
- *abstract void deleteField(String table,StringwhereClause) throws SQLException:* Suppression de champs.

- **La classe *DB* :**

Cette classe permettant d'accéder à une base de données sous MySQL ou sous PostgreSQL, qui permet d'opérer des requêtes sur des tables d'une base de données MySQL ou PostreSQL

- **La classe DB*Table* :**

Cette classe abstraite, a été implémentée par belgasmia[Bel09].

Elle modélise une table d'une base de données relationnelle, implémentée comme une *Hashtable*.

- **La classe *Dimension* :**

C'est une classe qui hérite de *DBTable* et qui est un Observer (c'est une *BDTable*). Elle est Observable de la table de faits (*FactTable*) et Observer de HTTPServer (*SocketClient)* dans cette classe nous avons rajouté la date quand on a une nouvelle information cette classe prends cette enregistrement qui est de format d'une chaine de caractère séparer par « ! » on va la parser par la méthode :

public void parse(String stringField) throws SQLException{}.

- **La classe *Dimension2* :**

C'est une classe abstraite intermédiaire et elle est une Dimension Observable de la table de fait (*FactTable*).

- **La classe *MédecinDim :***

La classe *MédecinDim* est une Dimension.

Nous avons rajouté deux champs par rapport à la classe *Médecins* dans le serveur qui sont la *date d'entrée* et la *date de sortie*.

- **La classe *MédicamentDim* :**

La classe MédicamentDim est une Dimension.

Nous avons rajouté deux champs par rapport de la classe *Médicaments* dans le serveur

Date effectif et *Date de périmassions*.

- **La classe *PatientDim :***

La classe *PatientDim* est une Dimension.

Nous avons rajouté deux champs supplémentaires par rapport la classe *Médicaments* dans le serveur : *Date d'entrée* et *Date de sortie*.

- **La classe *OrdonnanceDim* :**

La classe *OrdonnanceDim* est une Dimension.

Nous avons rajouté deux champs par rapport de la classe *Médicaments* dans le serveur *Date effectif* et *Date périmassions*.

- **La classe *MédicamentPrescriDim* :**

La classe *MédicamentPrescriDim* est une Dimension.

Nous avons rajouté deux champs par rapport de la classe *MédicamentPresciDim* dans le serveur *Date effectif* et *Date périmassions*.

- **La classe *PrescriptionMédicamentDateDim* :**

La classe *PrescriptionMédicamentDateDim* est une Dimension.

Nous avons rajouté deux champs par rapport de la classe *PrescriptionMédicamentDate* dans le serveur *Date effectif* et *Date périmassions*.

- **La classe *FactTable* :**

La classe *FactTable* est une *DBTable*, et elle est observée par la classe *Dimension*.

- **La classe *PrescriptionMédicamentFactTable* :**

C'est la classe ou nous avons géré les requêtes SQL, donc elle est responsable de la gestion des requêtes, elle est observée par la classe *Dimension2*.

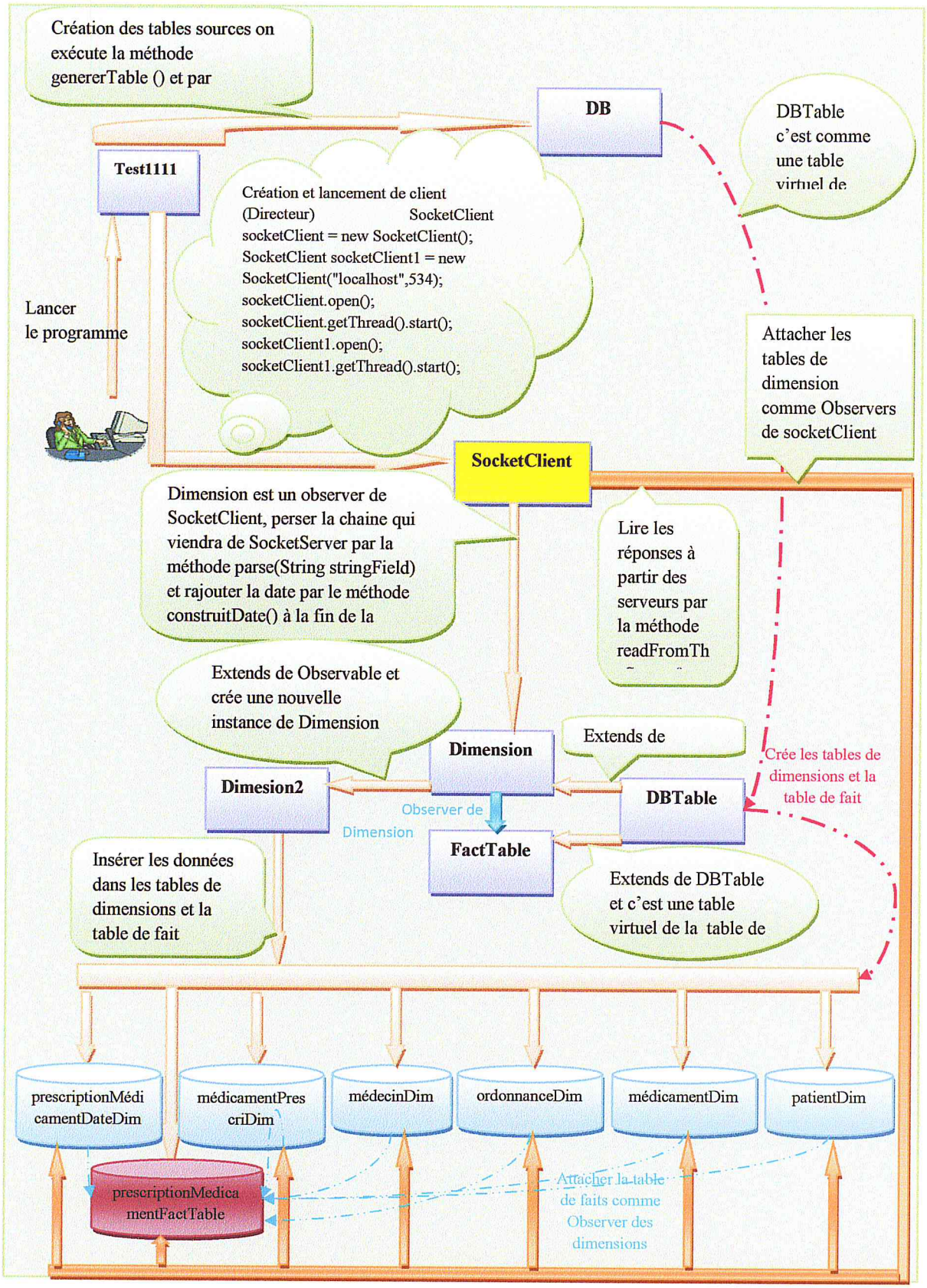


Figure III.21: *Remplissage de l'entrepôt de données avant refactoring*[Dra10]

4.5.2. Après Refactoring

- **La classe *StagingArea***

C'est une interface de file (queue) indique que vous pouvez ajouter des éléments à la fin d'une queue, supprimer des éléments au début d'une queue, et déterminer le nombre d'éléments contenus dans une queue. Les queues sont utilisées si vous devez enregistrer des objets et les récupérer selon la règle *"premier entré, premier sorti"*

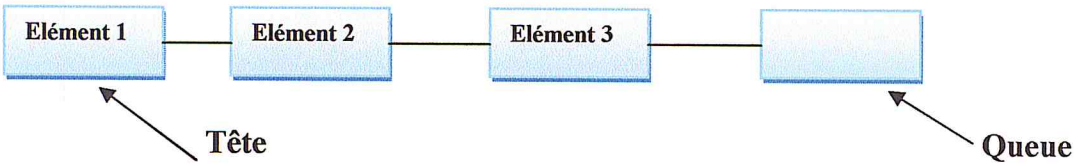


Figure III.22 : *une queue.*

Il existe deux implémentations courantes des queues. La première se sert d'un tableau circulaire et la seconde, d'une liste chaînée.

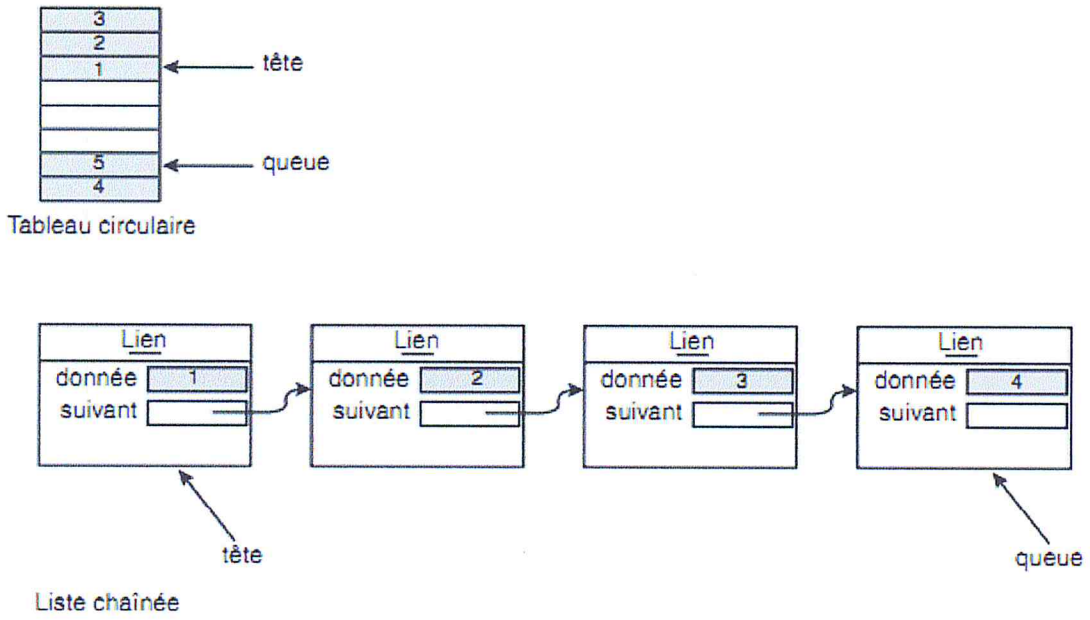


Figure III. 23 : *l'implémentation d'une queue*

Pour notre travail on va utiliser liste chainée

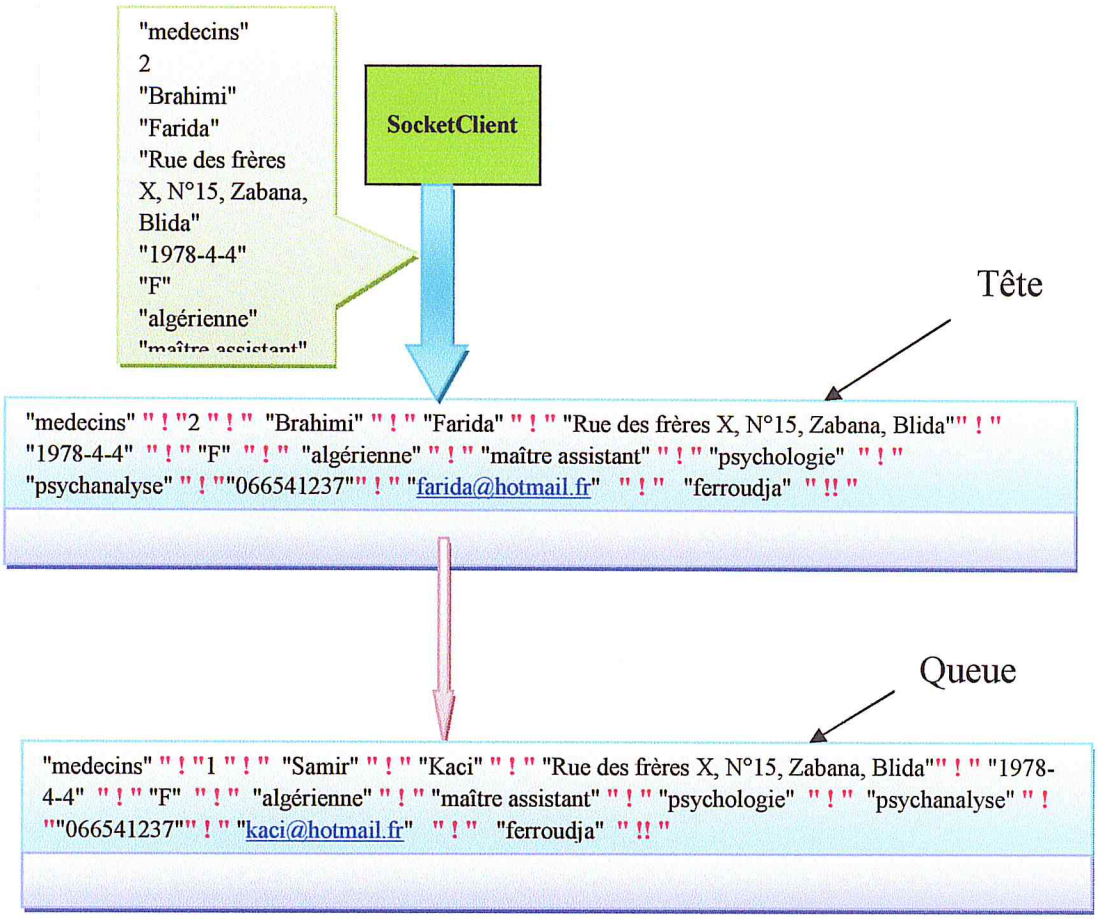


Figure III. 24 : *Utilisation de liste chainée.*

Remarque

Les classes restent les même sauf :

Classe *SocketClient* qui devient observable de *StagingArea*, et la classe *StagingArea* devient observable des *dimensions*.

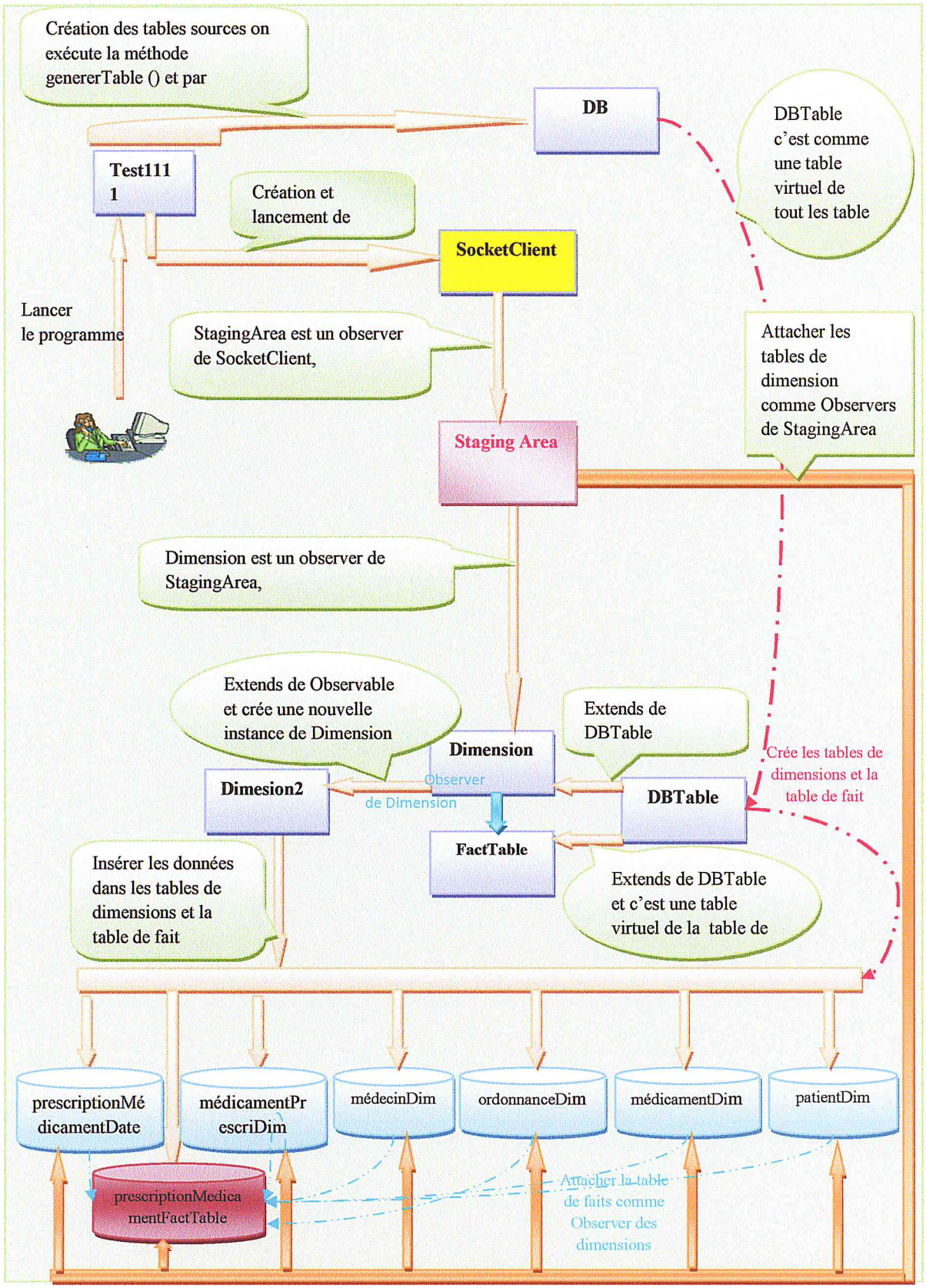


Figure III.25: *Remplissage de l'entrepôt de données après refactoring*

4.6. Diagramme de classe

4.6.1. Avant Refactoring

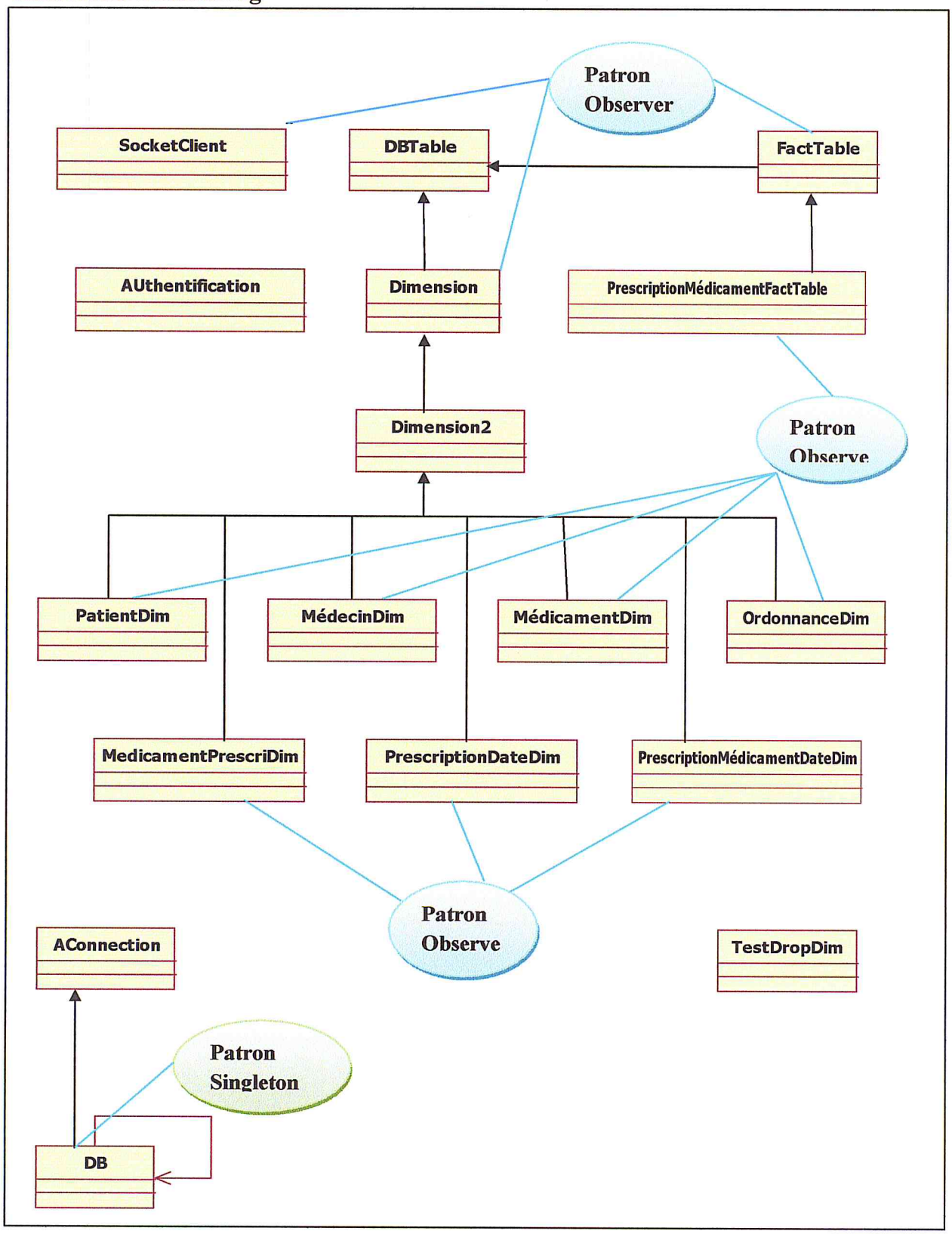


Figure III.26:*Diagramme de classe de serveur Client Avant Refactoring.* [Dra10]

4.6.2. Après Refactoring

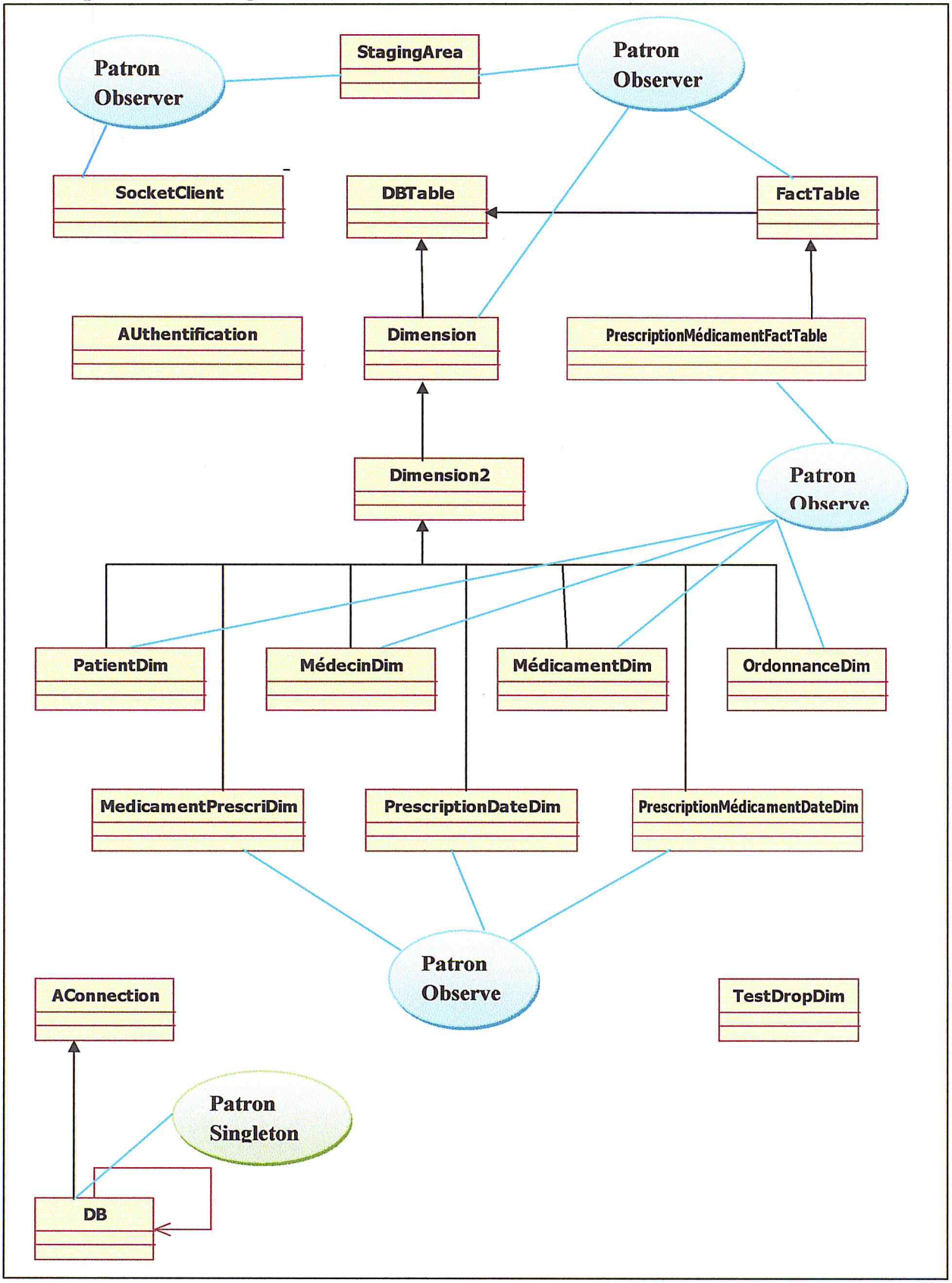


Figure III.27:*Diagramme de classe de serveur Client Après Refactoring.*

4.7. Diagramme de séquence

- ***« La création des tables des dimensions et de la table de fait »***

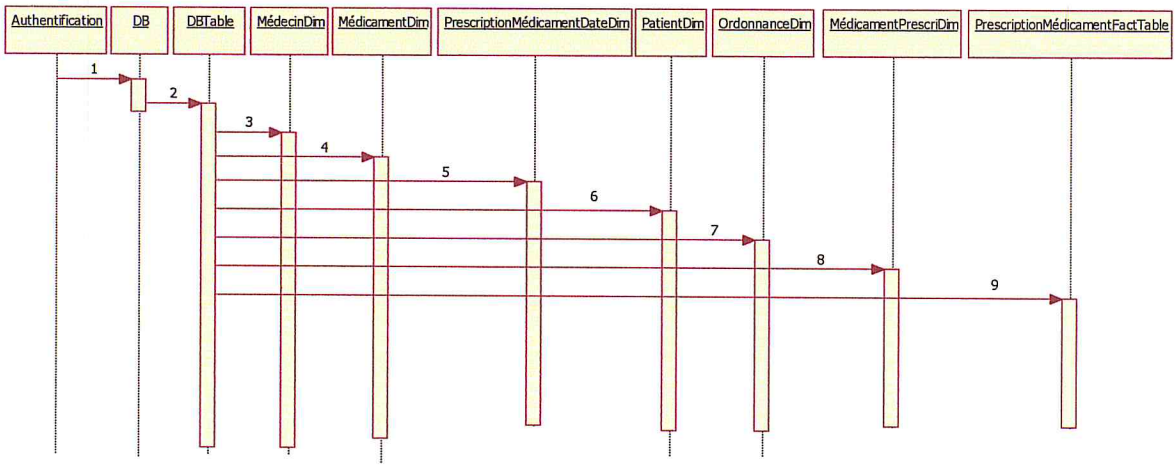


Figure III.28: *Diagramme de séquence de la création des tables des dimensions et de la table de fait*

1 – Lancer le programme du client (l'administrateur), la classe *Authentification* envoie un signe à la classe *DB* si son couple (identifiant/mot de passe) est correct.

2 – on va appeler les méthodes quand nous avons besoin de la classe *DB* parce que c'est la classe qui contient presque toutes les méthodes et ici nous avons utilisé par exemple la méthode :

public void createTable(String nomTable,Stringcolonne){ }.

Cette méthode crée les tables et la classe *DBTable* est la table virtuelle de toutes les tables donc cette classe utilise les méthodes de la classe *DB*.

3 – La classe *DBTable* envoie un signal à la classe *MédecinDim* pour créer la table *médecinDim.*

4 – La classe *DBTable* envoie un signal à la classe *MédicamentDim* pour créer la table *médiamentDim.*

5 – La classe *DBTable* envoie un signal à la classe *PrescriptionMédicamentDateDim* pour crée la table *prescriptionMédicamentDateDim.*

6 – La classe DBTable envoie un signal à la classe PatientDim pour créer la table patientDim..

7 – La classe *DBTable* envoie un signal à la classe *OrdonnanceDim* pour créer la table *ordonanceDim.*

8 – La classe *DBTable* envoie un signal à la classe *MédicamentPrescriDim* pour créer la table *médicamentPrescriDim.*

9 - La classe *DBTable* envoie un signal à la classe *PrescriptionMédicamentFactTable* pour créer la table *prescriptionMédicamentFactTable.*

- « ***La suppression de les tables de dimension Et la table de fait»***

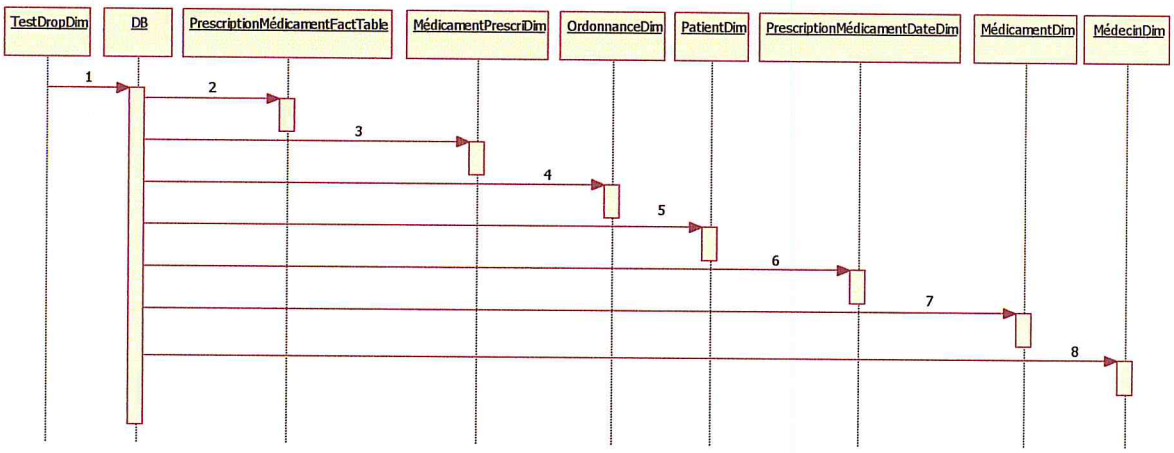


Figure III.29: *Diagramme de séquence de la suppression de les tables de dimension Et la table de fait.*

1 – Nous avons lancé le système par la classe TestDropDim qui contient le main.

2 - la classe *DB* va utiliser la méthode

public void dropTable(String tableName) throws SQLException { } pour supprimer la table de dimension *PrescriptionMédicamentFactTable* c'est la première table à supprimer parce qu' elle contient les clés primaire des tables de dimensions comme des clés étrangère.

3 - la classe *DB* va utiliser la méthode

public void dropTable(String tableName) throws SQLException { } pour supprimer la table de dimension *MédicamentPrescriDim.*

4 - la classe *DB* va utiliser la méthode

public void dropTable(String tableName) throws SQLException { } pour supprimer la table de dimension *OrdonnanceDim.*

public void dropTable(String tableName) throws SQLException { } pour supprimer la table de dimension *OrdonnanceDim.*

5 - la classe *DB* va utiliser la méthode

public void dropTable(String tableName) throws SQLException { } pour supprimer la table de dimension *PatientDim.*

6 - la classe *DB* va utiliser la méthode

public void dropTable(String tableName) throws SQLException { } pour supprimer la table de dimension *PrescriptionMédicamentDateDim.*

7 - la classe *DB* va utiliser la méthode

public void dropTable(String tableName) throws SQLException { } pour supprimer la table de dimension *MédicamentDim.*

8 - la classe *DB* va utiliser la méthode

public void dropTable(String tableName) throws SQLException { } pour supprimer la table de dimension *MédecinDim.*

4.8. Modélisation par patron

Dans cette partie on va montrer la source de donnée qui est l'hôpital vers l'administration de la prise de décision notre modélisation repose sur la modélisation par patrons Observer/Observable et Singleton.

4.8.1. Avant Refactoring

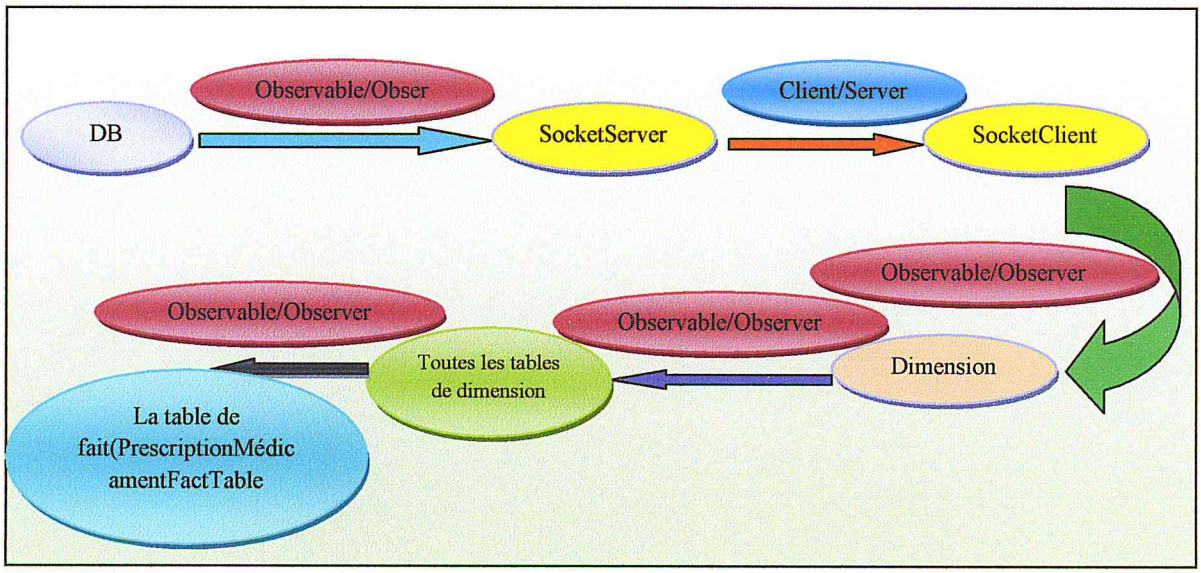


Figure III.30: *La modélisation par patron entre les classes avant refactoring[Dra10]*

4.8.2. Après Refactoring

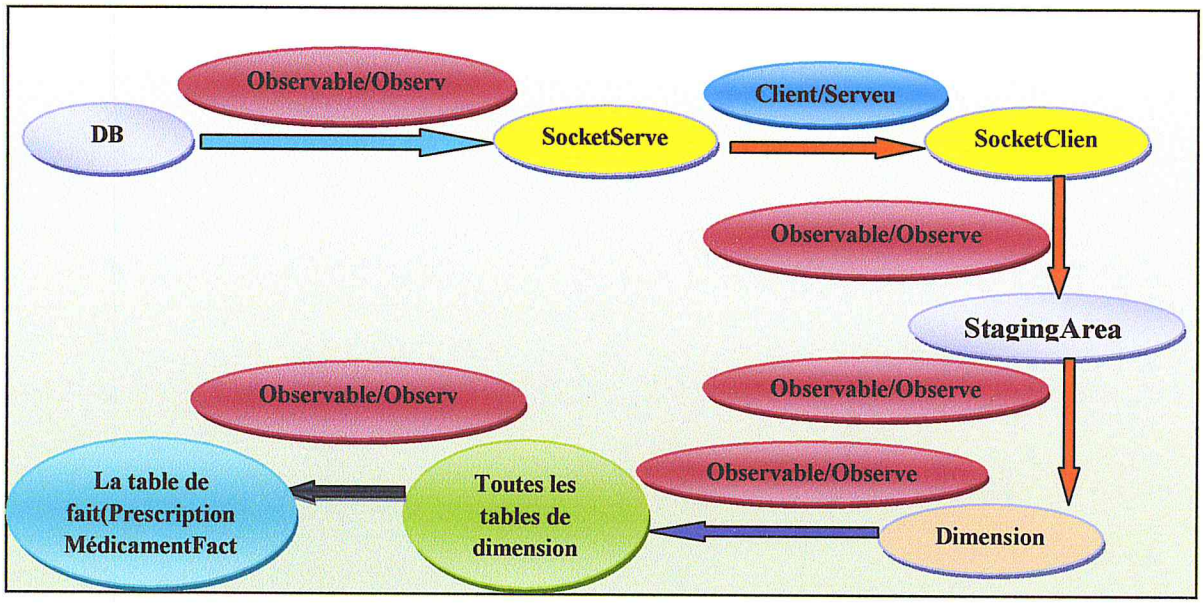


Figure III.31: *La modélisation par patron entre les classes après refactoring.*

4.9. Un intégrateur de données dans la table de fait

Pour intégrer les données dans la table de fait nous avons utilisé le patron de conception Observer si il y a une nouvelle information dans le table de dimension, il faut envoyer cette information à la table de fait (prendre sont clé primaire et écrire ce clé comme une clé étrangère dans la table de fait) et nous utilisons les requêtes SQL, utilisent les requête pour charger les clé étrangère de la table de fait, et attend jusqu'à que il y ait au moins une information dans chaque dimension pour que nous puissions prendre les décisions, c'est-à-dire opérer des quering et faire des viewing et reportings.

4.10. Meta Donnée

4.10.1. Avant Refactoring

Il n'existe pas.

4.10.2. Après Refactoring

Le JDBC est capable des informations sur la structure d'une base de données et de ses tableaux. Par exemple, vous pouvez obtenir une liste des tableaux d'une base de données particulière ou le nom de chaque colonne et le type de données associées de chaque tableau.

Ces informations de structure sont extrêmement importantes pour les programmeurs qui doivent écrire des outils pour travailler sur n'importe quelle base de données.

Pour obtenir des informations sur une base de données, il faut demander un objet de type *DatabaseMetaData* à la connexion de la base de données.

DatabaseMetaData meta = conn.getMetaData();

Vous êtes maintenant prêt à obtenir certaines métadonnées. Par exemple, l'appel

ResultSet mrs = meta.getTables(null, null, null, new String[] { "TABLE" })

Renvoie un ensemble de résultats contenant des informations sur tous les tableaux d'une base de données.

Chaque ligne de l'ensemble de résultats contient des informations sur un tableau de la base de données. Seule la troisième colonne nous intéresse. Les noms de tableaux se trouvent donc dans *rs.getString(3).*

Voici le code permettant de remplir le menu déroulant :

```
while (mrs.next())
tableNames.addItem(mrs.getString(3));
rs.close();
```

La classe *DatabaseMetaData* propose des informations sur la base de données. Une deuxième classe de métadonnées, *ResultSetMetaData*, rapporte des informations sur un ensemble de résultats. Lorsque vous obtenez un ensemble de résultats à partir d'une requête, vous pouvez demander le nombre de colonnes total, ainsi que le nom, le type et le nombre d'éléments de chaque colonne.

```
ResultSet mrs = stat.executeQuery("SELECT * FROM " + tableName);
ResultSetMetaData meta = mrs.getMetaData();
for (int i = 1; i <= meta.getColumnCount(); i++){
String columnName = meta.getColumnLabel(i);
int columnWidth = meta.getColumnDisplaySize(i);
... ... ... ... ... ... ... .. }
```

4.11. Quering

4.11.1. Avant refactoring

Il y'avait un nombre limité de Quering , et qui sont prédifinies et préétablies par le code c -à-d l'utilisateur va choisir un requête parmi les 4 requêtes.

Les requêtes existantes sont [Dra10]:

Requête 1:*Medecins-patients pour médicaments*

SELECT c.nom, f.nom

FROM presmedftab7 a , medecinsdim7 c , patientsdim7 f

WHERE a.medicamentsk =c.code AND a.patientsk =f.code AND c.hopital =f.hopital *GROUP BY* c.nom, c.hopital, f.hopital, f.nom

Requete2: *All from FactTable*

SELECT *

FROM presmedftab7

Requete3:*Medecins-Patients Jour-Mois-Année de presc pour Médicament doliphrane*

SELECT c.nom, c.prenom, b.nom, d.quantite, g.jour, g.moisnom, g.annee

FROM presmedftab7 a , medecinsdim7 b , patientsdim7 c , medicamentsprescridim7 d , ordonnancesdim7 e , medicamentsdim7 f , prescmeddatesdim7 g

WHERE c.code =e.codePatient *AND* e.code =d.codeOrd *AND* g.datesk =e.codeDate *AND* f.code =d.codeMedi *AND* b.code =e.codeMedecin

GROUP BY c.nom, c.prenom, c.hopital, b.nom, b.hopital, d.quantite, g.jour, g.moisnom, g.annee, f.nom

HAVING f.nom = 'doliprane' *AND* b.hopital =c.hopital

Requete4 : *Patients pour groupe sanguin A+*

SELECT c.nom, c.prenom, c.groupSang, c.hopital, *COUNT*(a.patientquantite)

FROM presmedftab7 a , patientsdim7 c

WHERE c.code =a.patientsk

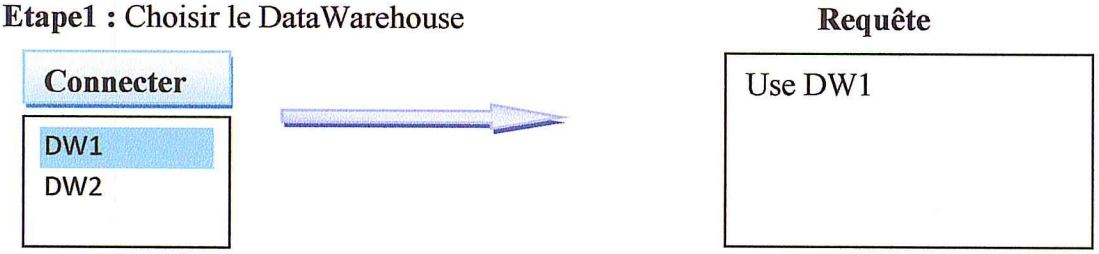
GROUP BY c.nom, c.prenom, c.hopital, c.groupSang

HAVING c.groupSang = 'A+' *AND* c.hopital = 'ferroudja' *AND COUNT*(a.patientquantite) >

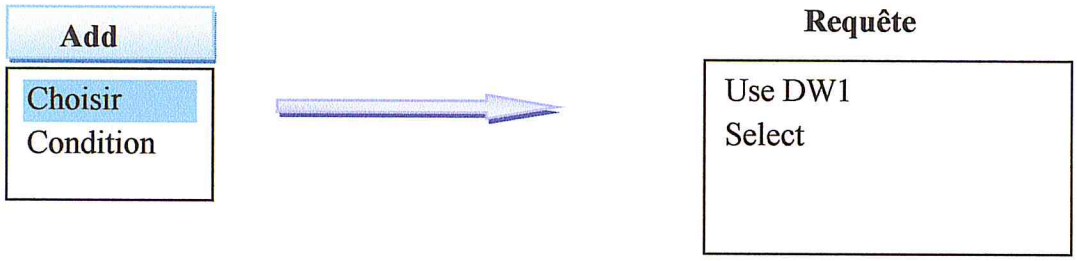
4.11.2. Après refactoring

Plus général que le précédent parce que l'utilisateur peut construire plusieurs sortes ded requêtes en se faisant aider par une interface utilisateur.

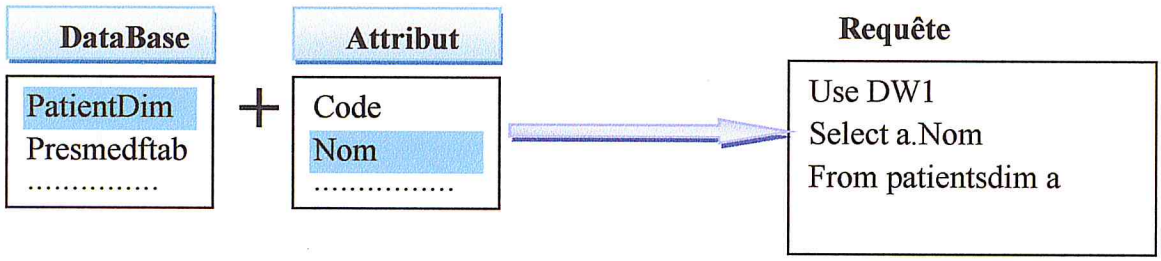
Etape1 : Choisir le DataWarehouse



Etape 2 : Choisir les résultats de sortie



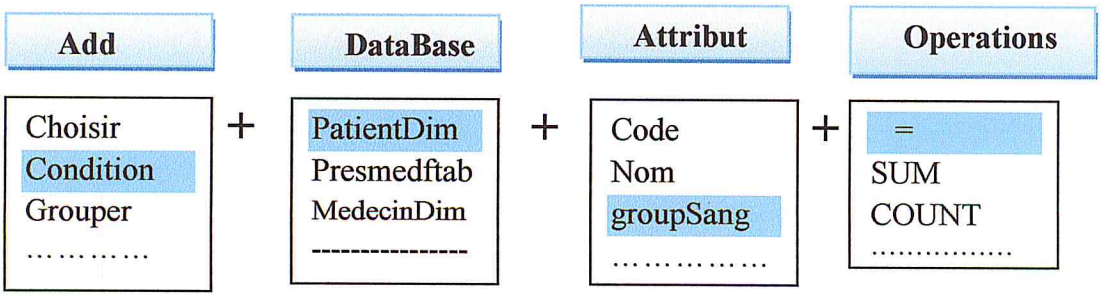
Etape3 : Entrer le nom de la table et les attributs



Remarque

On peut sélectionner plusieurs attributs pour plusieurs tables

Etape4 : Entrer les Conditions



Requête

```
Use DW1
Select a.Nom
From patientsdim a
Where a.groupSang = 'A+'
```

Remarque

On peut ajouter plusieurs condition sur les attributs, et on peut ajouter (GROUP BY , *HAVING, ...).*

4.12. Reporting

4.12.1. Avant refactoring

Quand l'utilisateur choisir une requête le résultat sera affiché sous forme d'une table [Dra10]

nom	nom
Dekhlifi	Doudaoui
Drairi	Dziri

Figure III.32: *reporting avant refactoring*

4.12.2. Après refactoring

Après construire la requête le résultat sera affiché sur trois forme (Table, histogramme, diagramme en camembert).

nom	nom
Dekhlifi	Doudaoui
Drairi	Dziri

Figure III.33: *reporting après refactoring.*

CHAPITRE 4

Réalisation et Tests

1. Introduction

Notre application porte sur la refactoring d'un DW pour le domaine médical, et nous avons essayé de mettre à la disposition du corps médical un outil de travail qui leur faciliterait la gestion des médicaments dans l'hôpital, pour nous permettons de faciliter la gestion des données (consultation et stockage,…) qui se trouvent réparties dans les différent hôpitaux.

L'objectif consiste à gerer les médicaments dans les hopitaux et le contrôle de l'approvisionnement de médicaments dans les hôpitaux.

2. Outils utilisés

2.1. Eclipse

Eclipse est un environnement de développement d'une façon orientée objet et vous permet d'avoir une application bien structurée, modulable, maintenable beaucoup plus facilement et efficace. Elle est principalement écrite en Java (à l'aide de la bibliothèque graphique SWT, …), et ce langage, grâce à des bibliothèques spécifiques, est également utilisé pour écrire des extensions [Oli06].

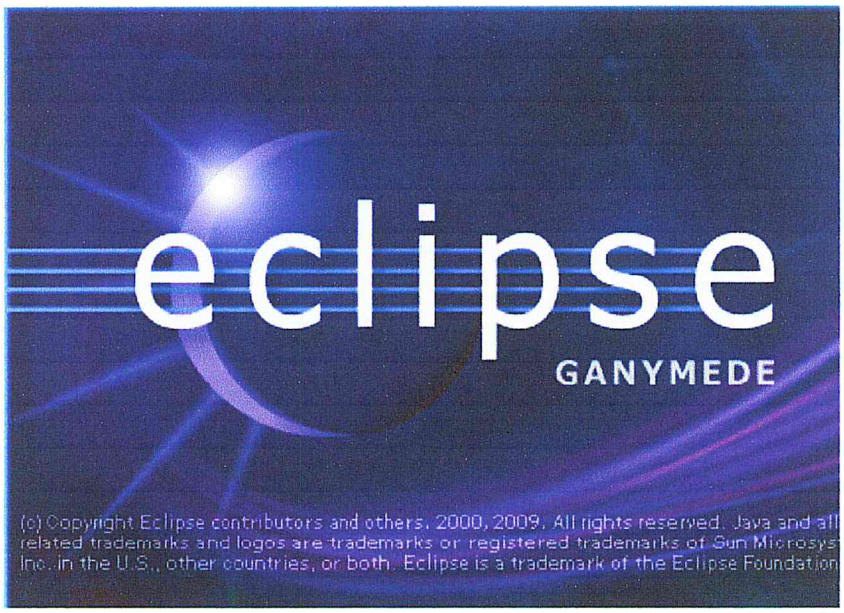


Figure IV.1: *Eclipse*

2.2.MySQL

MySQL est un système open source de base de données qui utilise Structured Query Langage (SQL) pour fonctionner. Elle fonctionne sur plus de 20 plateformes incluant (Linux, Windows, Mac OS, ….) Vous offrant une grande flexibilité.

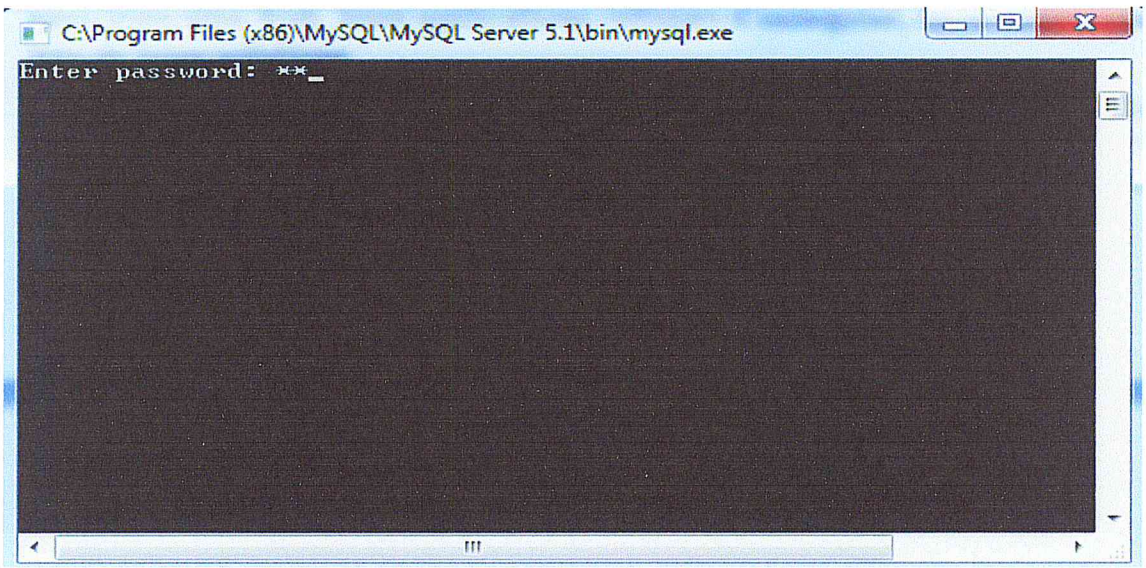


Figure IV.2 : *MySQL.*

2.3. PostgreSQL

PostgreSQL est un serveur de base de données transactionnel très puissant supportant le langage SQL, est développés selon le mode open source. En terme de taille, les limitations techniques n'existent quasiment pas puisqu'une simple table peut contenir jusqu'à 64 téra-octets de données.

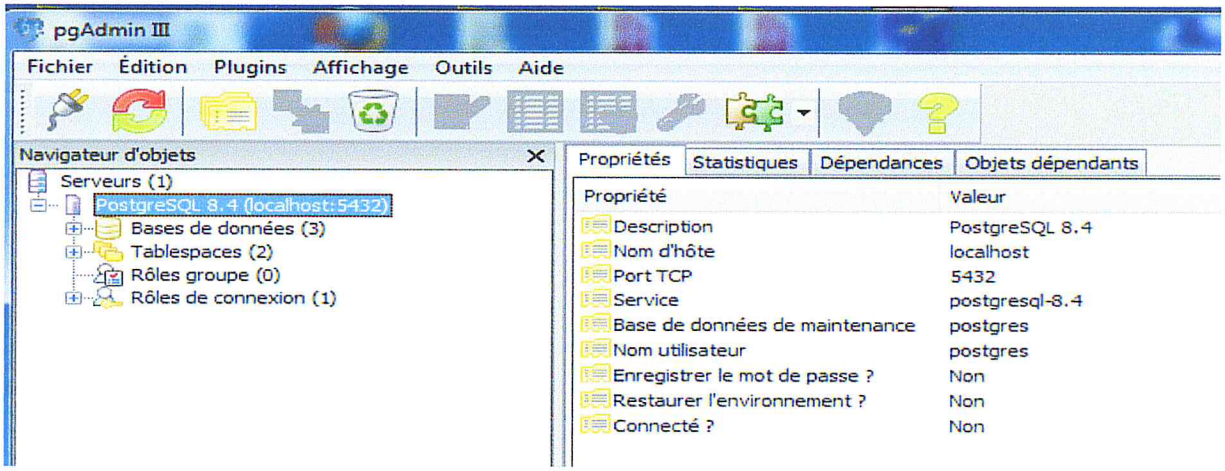


Figure IV.3: *PostgreSQL.*

3. Les tables de notre projet

3.1. Les tables de la base de données patients1 sous MySQL

Dans cette partie nous avons présenté le contenue des tables de MySQL dans la base de données opérationnel :

- Le continue de la table medecin sous MySql de la base de donnée relationel :

MySQL Command Line Client

mysql> select * from medecins;

code	nom	prenom	adresse	dateNais	sexe	nation	grade	specialite	service	telephone	email
1	Brahimi1	Farida	Rue des frÞres X, N°15, Zabana, Blida	1978-04-04	F	algúrienne	ma¯tre assistant	psychologie	psychanalyse	066541237	farida@hotmail.fr
2	Drai1	Khadidja	Rue des enseignants, N°22, Blida	1980-06-13	F	algúrienne	ma¯tre assistant	gen├®raliste	mùdecine gúnúrale	0775121452	dkhadidja@yahoo.fr
3	Dekhli1	Abdelkader	Citú des plantes, N°16, Blida	1971-03-08	M	algúrienne	professeur	chirurgie	pneumologie	0795231274	dabdelkader@yahoo.fr
4	Ghúraba1	Madjid	Rue de Blida, N°123, Mouza´a, Blida	1970-11-23	M	algúrienne	ma¯tre assistant	chirurgie	pneumologie	0665414547	gmadjid@yahoo.fr
5	Kaino1	Mira	Citú des roses, N°45, Blida	1975-03-07	M	algúrienne	ma¯tre assistant	chirurgie	cancúrologie	0556931275	kmira@yahoo.fr

5 rows in set (0.00 sec)

Figure IV.4 : *La table médecins sur MySQL*

- La table Médicaments sous MySQL de la base de données opérationnel

MySQL Command Line Client

mysql> select * from medicaments;

code	label	nom	dateFab	datePer	prix	etat	service	fabricant	distribut
1	MA45	doliprane	2009-12-12	2012-12-31	150	1	urgences	Saidal	SPA
2	CSD2	nom2	2009-12-12	2012-12-31	450.41	0	urgences	Saidal	SPA
3	HJ	nom3	2009-12-12	2012-12-31	850.2	2	urgences	Saidal	SPA
4	POL45	nom4	2009-12-12	2012-12-31	230	2	urgences	Saidal	SPA
5	RFDS8	nom5	2009-12-12	2012-12-31	650.23	1	urgences	Saidal	SPA

5 rows in set (0.01 sec)

Figure IV.5: *La table médicaments sur MySQL.*

- La table Patients sous MySQL de la base de données opérationnel

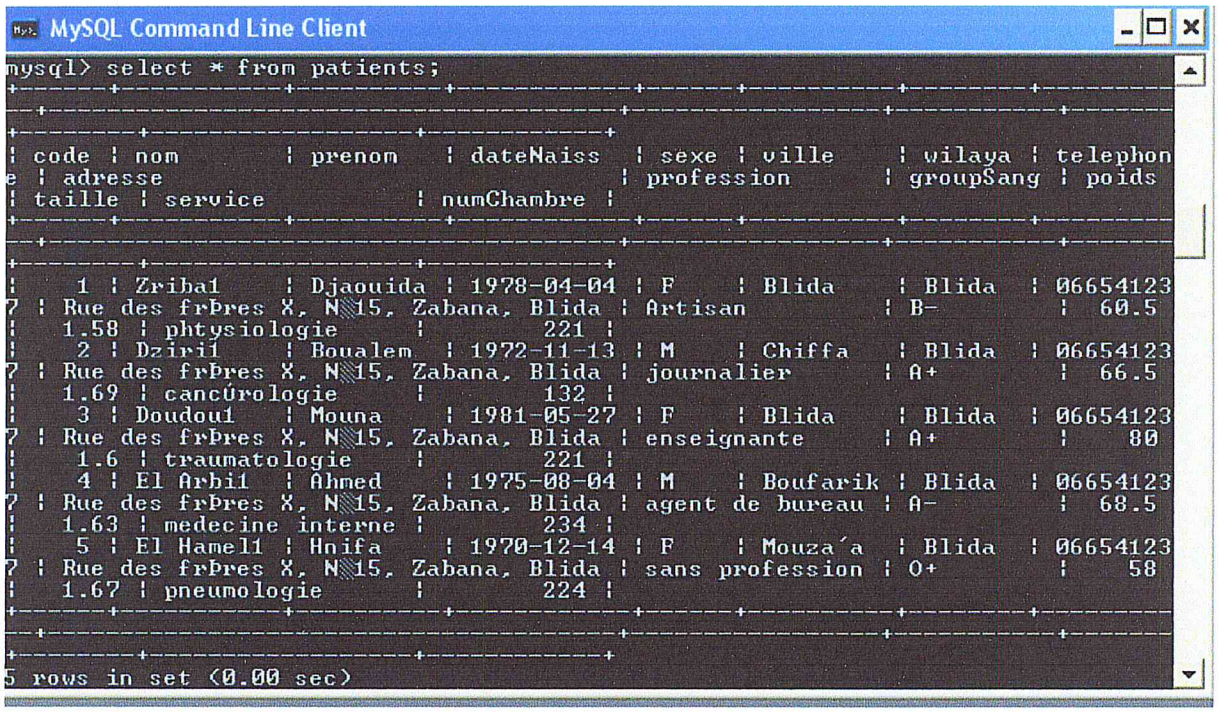


Figure IV.6 : *La table Patients sur MySQL*

- La table Ordonnances sous MySQL de la base de données opérationnel

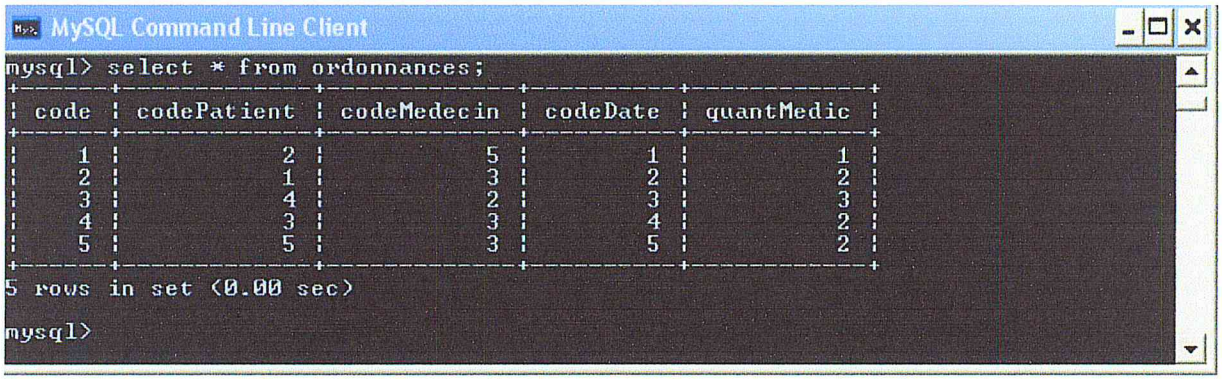
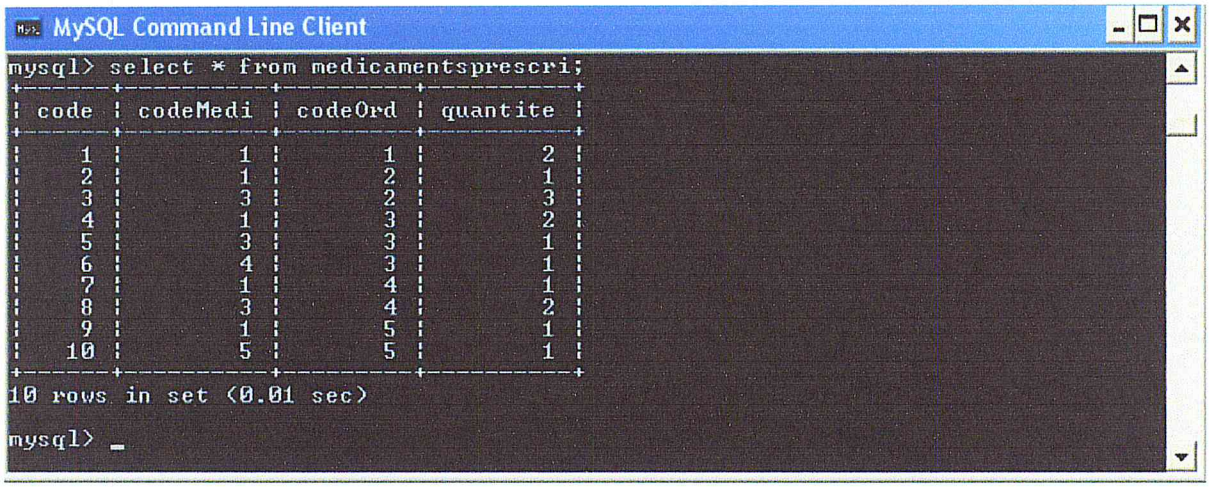


Figure IV.7: *La table Ordonnances sur MySQL*

- La table Médicamentsprescri sous MySQL de la base de données opérationnel



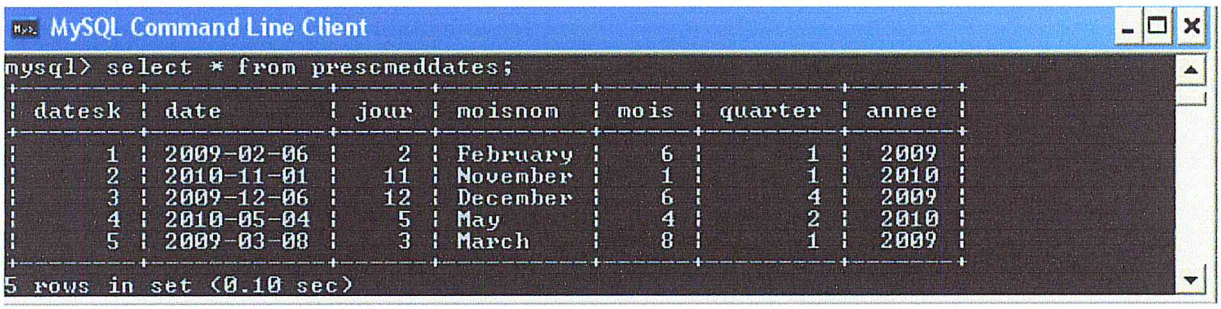
MySQL Command Line Client

```
mysql> select * from medicamentsprescri;
+------+----------+---------+----------+
| code | codeMedi | codeOrd | quantite |
+------+----------+---------+----------+
|    1 |        1 |       1 |        2 |
|    2 |        1 |       2 |        1 |
|    3 |        3 |       2 |        3 |
|    4 |        1 |       3 |        2 |
|    5 |        3 |       3 |        1 |
|    6 |        4 |       3 |        1 |
|    7 |        1 |       4 |        1 |
|    8 |        3 |       4 |        2 |
|    9 |        1 |       5 |        1 |
|   10 |        5 |       5 |        1 |
+------+----------+---------+----------+
10 rows in set (0.01 sec)

mysql>
```

Figure IV.8 : *La table medicamentsprescris sur MySQL.*

La table PrescriptionMédicamentDate sous MySQL de la base de données opérationnel



MySQL Command Line Client

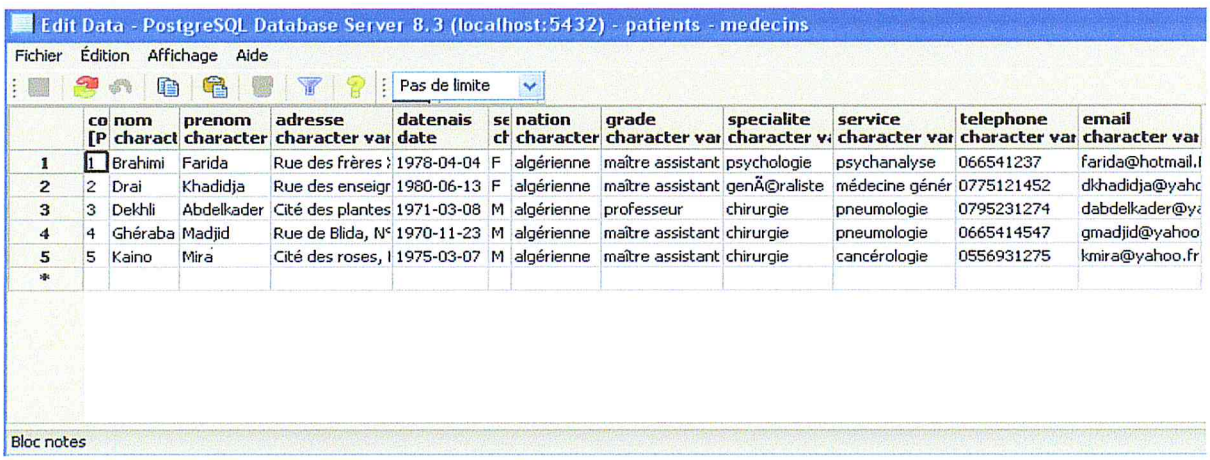
```
mysql> select * from prescmeddates;
+---------+------------+------+-----------+------+---------+--------+
| datesk  | date       | jour | moisnom   | mois | quarter | annee  |
+---------+------------+------+-----------+------+---------+--------+
|       1 | 2009-02-06 |    2 | February  |    6 |       1 |   2009 |
|       2 | 2010-11-01 |   11 | November  |    1 |       1 |   2010 |
|       3 | 2009-12-06 |   12 | December  |    6 |       4 |   2009 |
|       4 | 2010-05-04 |    5 | May       |    4 |       2 |   2010 |
|       5 | 2009-03-08 |    3 | March     |    8 |       1 |   2009 |
+---------+------------+------+-----------+------+---------+--------+
5 rows in set (0.10 sec)
```

Figure IV.9 : *La table PrescriptionMédicamentDate sur MySQL.*

3.2. Les tables de la base de données patients sous PostgreSQL

Dans cette partie nous avons présenté le contenue des tables de PostgreSQL dans la base de données opérationnel :

La table Médecins sous PostgreSQL de la base de données opérationnel



Edit Data - PostgreSQL Database Server 8.3 (localhost:5432) - patients - medecins

Fichier Édition Affichage Aide

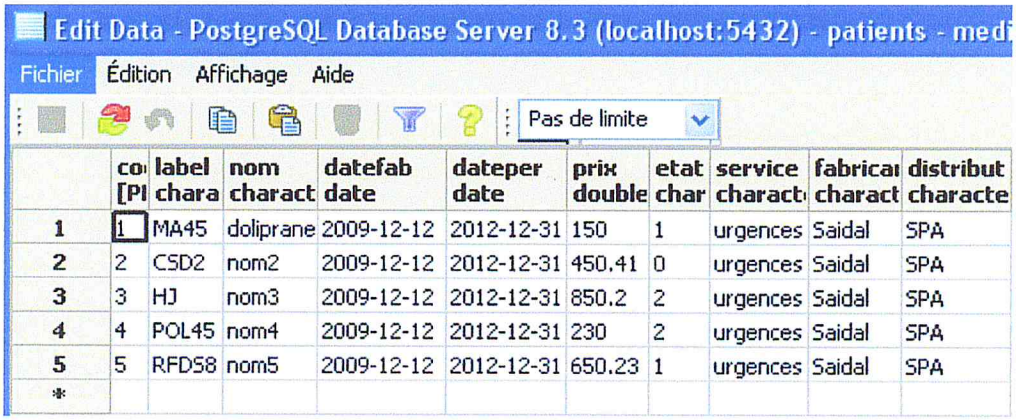
Pas de limite

	co [P	nom charact	prenom character	adresse character var	datenais date	se ch	nation character	grade character var	specialite character v	service character var	telephone character var	email character var
1	1	Brahimi	Farida	Rue des frères ;	1978-04-04	F	algérienne	maître assistant	psychologie	psychanalyse	066541237	farida@hotmail.l
2	2	Drai	Khadidja	Rue des enseigr	1980-06-13	F	algérienne	maître assistant	genÃ©raliste	médecine génér	0775121452	dkhadidja@yahc
3	3	Dekhli	Abdelkader	Cité des plantes	1971-03-08	M	algérienne	professeur	chirurgie	pneumologie	0795231274	dabdelkader@y
4	4	Ghéraba	Madjid	Rue de Blida, N°	1970-11-23	M	algérienne	maître assistant	chirurgie	pneumologie	0665414547	gmadjid@yahoo
5	5	Kaino	Mira	Cité des roses, I	1975-03-07	M	algérienne	maître assistant	chirurgie	cancérologie	0556931275	kmira@yahoo.fr
*												

Bloc notes

Figure IV.10: *La table Médecins sur PostgreSQL.*

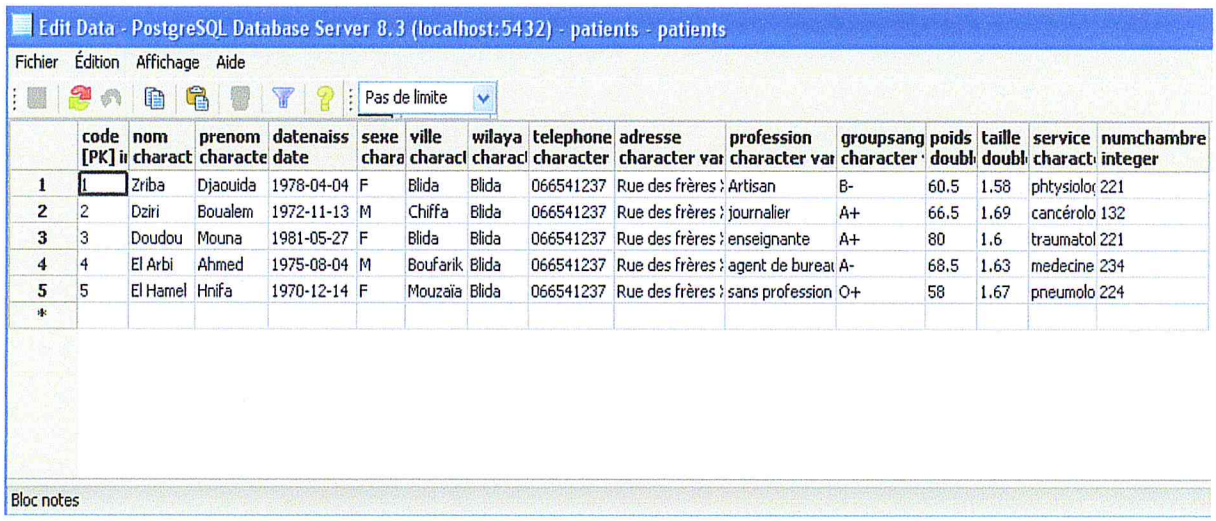
- La table Médicaments sous PostgreSQL de la base de données opérationnel



	co [PI	label chara	nom charact	datefab date	dateper date	prix double	etat char	service charact	fabricat charact	distribut characte
1	1	MA45	doliprane	2009-12-12	2012-12-31	150	1	urgences	Saidal	SPA
2	2	CSD2	nom2	2009-12-12	2012-12-31	450.41	0	urgences	Saidal	SPA
3	3	HJ	nom3	2009-12-12	2012-12-31	850.2	2	urgences	Saidal	SPA
4	4	POL45	nom4	2009-12-12	2012-12-31	230	2	urgences	Saidal	SPA
5	5	RFDS8	nom5	2009-12-12	2012-12-31	650.23	1	urgences	Saidal	SPA
*										

Figure IV.11 : *La table Médicaments sur PostgreSQL*

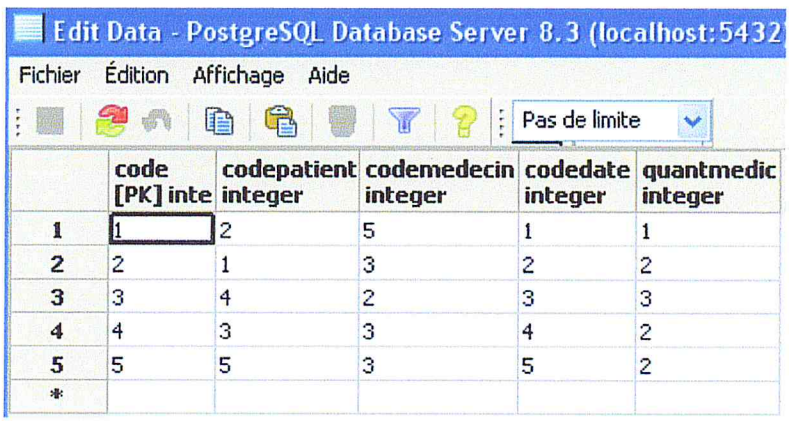
- La table Patients sous PostgreSQL de la base de données opérationnel



	code [PK] i	nom charact	prenom characte	datenaiss date	sexe chara	ville charact	wilaya charact	telephone character	adresse character var	profession character var	groupsang character	poids doubl	taille doubl	service charact	numchambre integer
1	1	Zriba	Djaouida	1978-04-04	F	Blida	Blida	066541237	Rue des frères	Artisan	B-	60.5	1.58	phtysiolo	221
2	2	Dziri	Boualem	1972-11-13	M	Chiffa	Blida	066541237	Rue des frères	journalier	A+	66.5	1.69	cancérolo	132
3	3	Doudou	Mouna	1981-05-27	F	Blida	Blida	066541237	Rue des frères	enseignante	A+	80	1.6	traumatol	221
4	4	El Arbi	Ahmed	1975-08-04	M	Boufarik	Blida	066541237	Rue des frères	agent de bureau	A-	68.5	1.63	medecine	234
5	5	El Hamel	Hnifa	1970-12-14	F	Mouzaïa	Blida	066541237	Rue des frères	sans profession	O+	58	1.67	pneumolo	224
*															

Figure IV.12 : *La table Patients sur PostgreSQL*

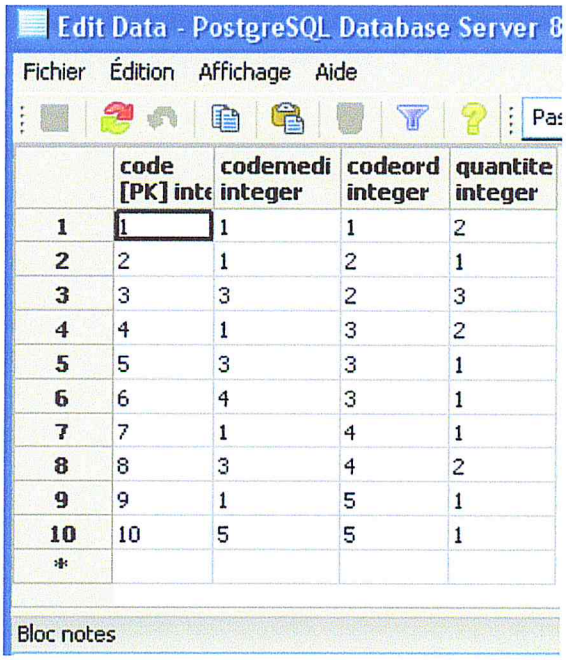
- La table Médecins sous PostgreSQL de la base de données opérationnel



	code [PK] inte	codepatient integer	codemedecin integer	codedate integer	quantmedic integer
1	1	2	5	1	1
2	2	1	3	2	2
3	3	4	2	3	3
4	4	3	3	4	2
5	5	5	3	5	2
*					

Figure IV.13: *La table MédicamentPriscris sur PostgreSQL.*

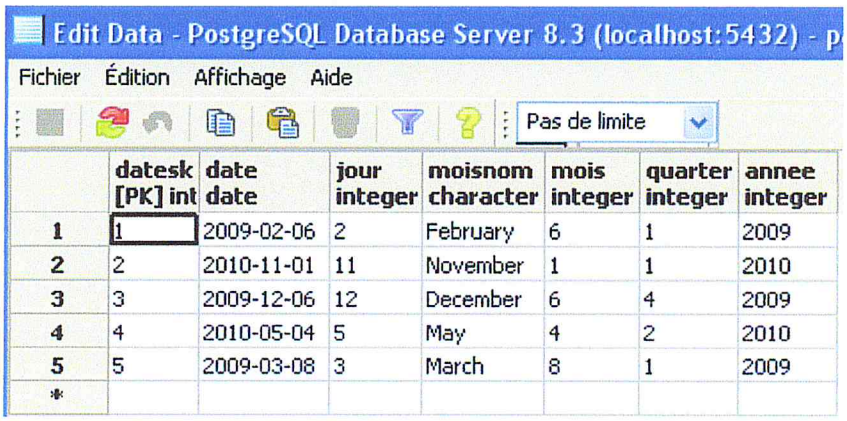
- La table Ordonnances sous PostgreSQL de la base de données opérationnel



	code [PK] inte	codemedi integer	codeord integer	quantite integer
1	1	1	1	2
2	2	1	2	1
3	3	3	2	3
4	4	1	3	2
5	5	3	3	1
6	6	4	3	1
7	7	1	4	1
8	8	3	4	2
9	9	1	5	1
10	10	5	5	1
*				

Figure IV.14 : *La table Ordonnance sur PostgreSQL*

- La table PrescriptionMédicamentDate sous PostgreSQL de la base de données opérationnel



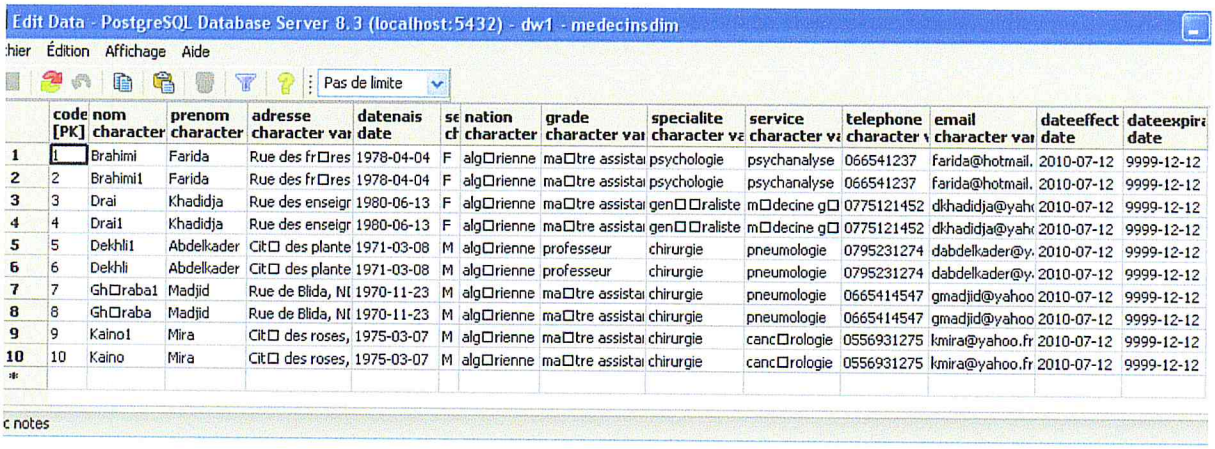
	datesk [PK] int	date date	jour integer	moisnom character	mois integer	quarter integer	annee integer
1	1	2009-02-06	2	February	6	1	2009
2	2	2010-11-01	11	November	1	1	2010
3	3	2009-12-06	12	December	6	4	2009
4	4	2010-05-04	5	May	4	2	2010
5	5	2009-03-08	3	March	8	1	2009
*							

Figure IV.15 : *La table PrescriptionMédicamentDate sur PostgreSQL0.*

3.3. Les tables de dimension de DW sous PostgreSQL

Dans cette partie nous avons présenté le contenue des tables de dimension de PostgreSQL dans l'entrepôt de données :

- La table MédecinDim sous PostgreSQL de l'entrepôt de données



Edit Data - PostgreSQL Database Server 8.3 (localhost:5432) - dw1 - medecinsdim

hier Édition Affichage Aide

Pas de limite

	code [PK]	nom character	prenom character	adresse character var	datenais date	se ch	nation character	grade character var	specialite character va	service character va	telephone character	email character var	dateeffect date	dateexpira date
1	1	Brahimi	Farida	Rue des fr□res	1978-04-04	F	alg□rienne	ma□tre assistai	psychologie	psychanalyse	066541237	farida@hotmail.	2010-07-12	9999-12-12
2	2	Brahimi1	Farida	Rue des fr□res	1978-04-04	F	alg□rienne	ma□tre assistai	psychologie	psychanalyse	066541237	farida@hotmail.	2010-07-12	9999-12-12
3	3	Drai	Khadidja	Rue des enseigr	1980-06-13	F	alg□rienne	ma□tre assistai	gen□□raliste	m□decine g□	0775121452	dkhadidja@yah	2010-07-12	9999-12-12
4	4	Drai1	Khadidja	Rue des enseigr	1980-06-13	F	alg□rienne	ma□tre assistai	gen□□raliste	m□decine g□	0775121452	dkhadidja@yah	2010-07-12	9999-12-12
5	5	Dekhli1	Abdelkader	Cit□ des plante	1971-03-08	M	alg□rienne	professeur	chirurgie	pneumologie	0795231274	dabdelkader@y.	2010-07-12	9999-12-12
6	6	Dekhli	Abdelkader	Cit□ des plante	1971-03-08	M	alg□rienne	professeur	chirurgie	pneumologie	0795231274	dabdelkader@y.	2010-07-12	9999-12-12
7	7	Gh□raba1	Madjid	Rue de Blida, NI	1970-11-23	M	alg□rienne	ma□tre assistai	chirurgie	pneumologie	0665414547	gmadjid@yahoo	2010-07-12	9999-12-12
8	8	Gh□raba	Madjid	Rue de Blida, NI	1970-11-23	M	alg□rienne	ma□tre assistai	chirurgie	pneumologie	0665414547	gmadjid@yahoo	2010-07-12	9999-12-12
9	9	Kaino1	Mira	Cit□ des roses,	1975-03-07	M	alg□rienne	ma□tre assistai	chirurgie	canc□rologie	0556931275	kmira@yahoo.fr	2010-07-12	9999-12-12
10	10	Kaino	Mira	Cit□ des roses,	1975-03-07	M	alg□rienne	ma□tre assistai	chirurgie	canc□rologie	0556931275	kmira@yahoo.fr	2010-07-12	9999-12-12

c notes

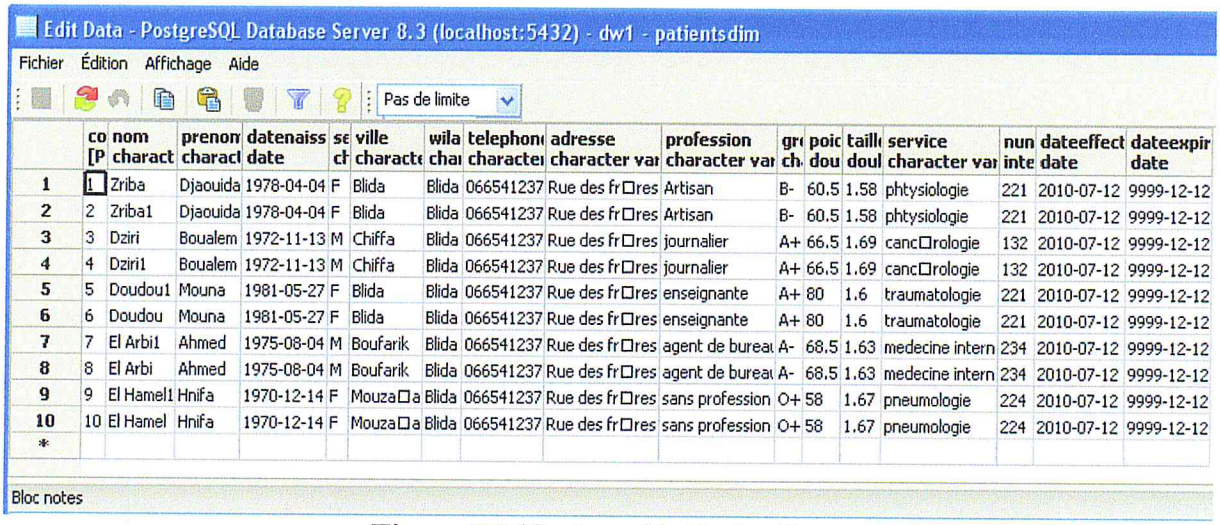
Figure IV.16 : *La table medecinDim*

- La table MédicamentDim sous PostgreSQL de l'entrepôt de données

	codedim [PK] integer	code integer	label character var	nom character var	datefab date	dateper date	prix double precis	etat character var	service character var	fabricant character var	distribut character var	dateeffect date	dateexpira date
1	1	1	MA45	dolprane	2009-12-12	2012-12-31	150	1	urgences	Saidal	SPA	2010-11-12	9999-12-12
2	2	2	CSD2	amoxicilin	2009-12-12	2012-12-31	450.41	0	urgences	Saidal	SPA	2010-11-12	9999-12-12
3	3	3	HJ	tegritol	2009-12-12	2012-12-31	850.2	2	urgences	Saidal	SPA	2010-11-12	9999-12-12
4	4	4	POL45	asp□gic	2009-12-12	2012-12-31	230	2	urgences	Saidal	SPA	2010-11-12	9999-12-12
5	5	5	RFDS8	lecothyrox	2009-12-12	2012-12-31	650.23	1	urgences	Saidal	SPA	2010-11-12	9999-12-12
6	6	6	M5	interjenon	2009-12-12	2012-12-31	150	1	urgences	Saidal	SPA	2010-11-12	9999-12-12
7	7	7	AMO01	sommazuie	2009-12-12	2012-12-31	450.41	0	urgences	Saidal	SPA	2010-11-12	9999-12-12
8	8	8	DX11	scipril plus	2009-12-12	2012-12-31	850.2	2	urgences	Saidal	SPA	2010-11-12	9999-12-12
9	9	9	OS	xyprexa	2009-12-12	2012-12-31	230	2	urgences	Saidal	SPA	2010-11-12	9999-12-12
10	10	10	RF8	donjeual	2009-12-12	2012-12-31	650.23	1	urgences	Saidal	SPA	2010-11-12	9999-12-12

Figure IV.17: *La table medicamentDim.*

- La table PatientDim sous PostgreSQL de l'entrepôt de données



Edit Data - PostgreSQL Database Server 8.3 (localhost:5432) - dw1 - patientsdim

Fichier Édition Affichage Aide

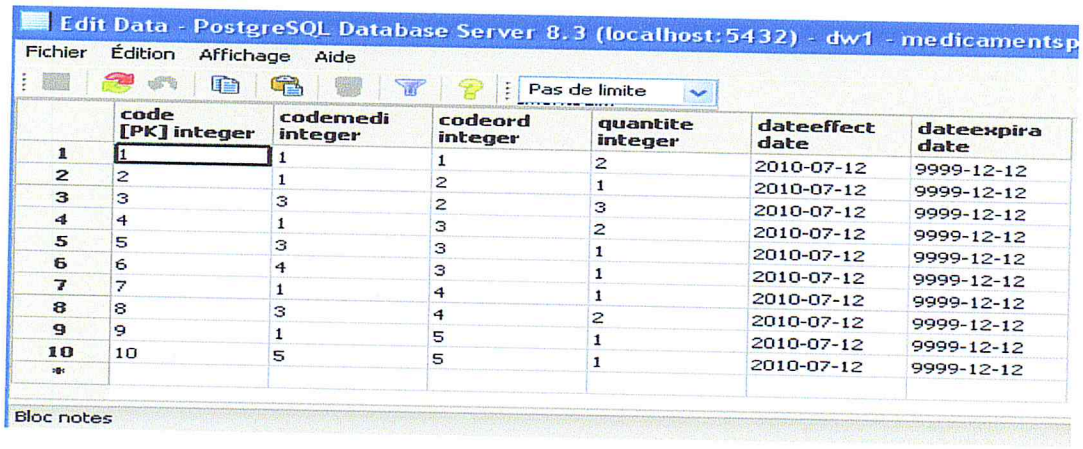
Pas de limite

	co [P	nom charact	prenom charact	datenaiss date	se ch	ville characte	wila cha	telephon characte	adresse character var	profession character var	gr ch	poic dou	taill doul	service character var	nun inte	dateeffect date	dateexpir date
1	1	Zriba	Djaouida	1978-04-04	F	Blida	Blida	066541237	Rue des fr□res	Artisan	B-	60.5	1.58	phtysiologie	221	2010-07-12	9999-12-12
2	2	Zriba1	Djaouida	1978-04-04	F	Blida	Blida	066541237	Rue des fr□res	Artisan	B-	60.5	1.58	phtysiologie	221	2010-07-12	9999-12-12
3	3	Dziri	Boualem	1972-11-13	M	Chiffa	Blida	066541237	Rue des fr□res	journalier	A+	66.5	1.69	canc□rologie	132	2010-07-12	9999-12-12
4	4	Dziri1	Boualem	1972-11-13	M	Chiffa	Blida	066541237	Rue des fr□res	journalier	A+	66.5	1.69	canc□rologie	132	2010-07-12	9999-12-12
5	5	Doudou1	Mouna	1981-05-27	F	Blida	Blida	066541237	Rue des fr□res	enseignante	A+	80	1.6	traumatologie	221	2010-07-12	9999-12-12
6	6	Doudou	Mouna	1981-05-27	F	Blida	Blida	066541237	Rue des fr□res	enseignante	A+	80	1.6	traumatologie	221	2010-07-12	9999-12-12
7	7	El Arbi1	Ahmed	1975-08-04	M	Boufarik	Blida	066541237	Rue des fr□res	agent de bureau	A-	68.5	1.63	medecine intern	234	2010-07-12	9999-12-12
8	8	El Arbi	Ahmed	1975-08-04	M	Boufarik	Blida	066541237	Rue des fr□res	agent de bureau	A-	68.5	1.63	medecine intern	234	2010-07-12	9999-12-12
9	9	El Hamel1	Hnifa	1970-12-14	F	Mouza□a	Blida	066541237	Rue des fr□res	sans profession	O+	58	1.67	pneumologie	224	2010-07-12	9999-12-12
10	10	El Hamel	Hnifa	1970-12-14	F	Mouza□a	Blida	066541237	Rue des fr□res	sans profession	O+	58	1.67	pneumologie	224	2010-07-12	9999-12-12

Bloc notes

Figure IV.18 : *La table PatientDim.*

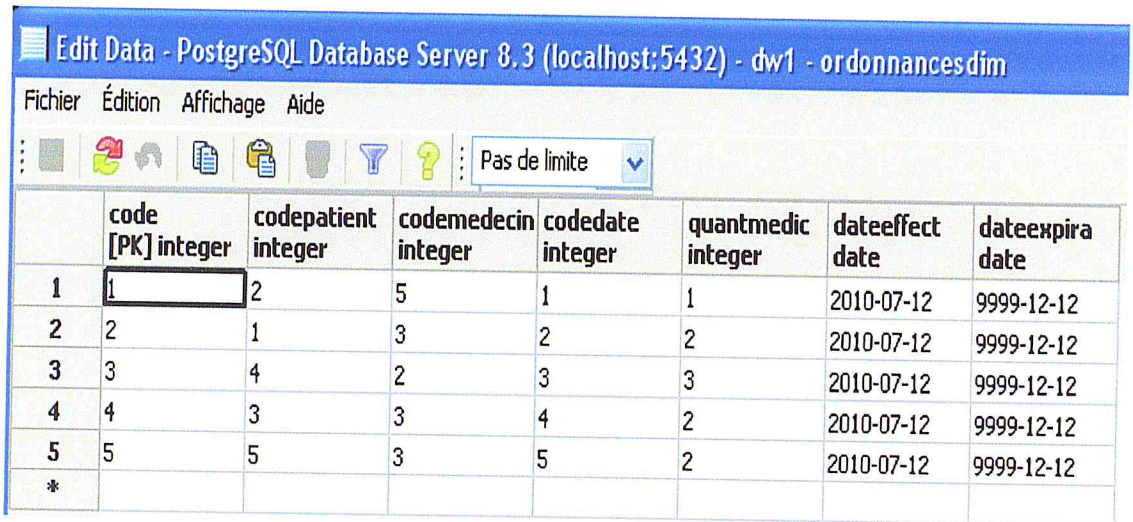
- La table MédicamentPrescriDim sous PostgreSQL de l'entrepôt de données



	code [PK] integer	codemedi integer	codeord integer	quantite integer	dateeffect date	dateexpira date
1	1	1	1	2	2010-07-12	9999-12-12
2	2	1	2	1	2010-07-12	9999-12-12
3	3	3	2	3	2010-07-12	9999-12-12
4	4	1	3	2	2010-07-12	9999-12-12
5	5	3	3	1	2010-07-12	9999-12-12
6	6	4	3	1	2010-07-12	9999-12-12
7	7	1	4	1	2010-07-12	9999-12-12
8	8	3	4	2	2010-07-12	9999-12-12
9	9	1	5	1	2010-07-12	9999-12-12
10	10	5	5	1	2010-07-12	9999-12-12
*						

Figure IV.19: *La table MédicamentPrescriDim.*

- La table OrdonnancesDim sous PostgreSQL de l'entrepôt de données



	code [PK] integer	codepatient integer	codemedecin integer	codedate integer	quantmedic integer	dateeffect date	dateexpira date
1	1	2	5	1	1	2010-07-12	9999-12-12
2	2	1	3	2	2	2010-07-12	9999-12-12
3	3	4	2	3	3	2010-07-12	9999-12-12
4	4	3	3	4	2	2010-07-12	9999-12-12
5	5	5	3	5	2	2010-07-12	9999-12-12
*							

Figure IV.20 : *La table OrdonnancesDim.*

- La table PrescriptionMédicamentDateDim sous PostgreSQL de l'entrepôt de données

Edit Data - PostgreSQL Database Server 8.3 (localhost:5432) - dw1 - pr

Fichier Édition Affichage Aide

Pas de limite

	da [P	date date	jour integ	moisnom character	mois integ	quart integ	annee intege	dateeffec date	dateexpira date
1	1	2009-02-06	2	February	6	1	2009	2010-07-12	9999-12-12
2	2	2009-02-06	2	February	6	1	2009	2010-07-12	9999-12-12
3	3	2010-11-01	11	November	1	1	2010	2010-07-12	9999-12-12
4	4	2010-11-01	11	November	1	1	2010	2010-07-12	9999-12-12
5	5	2009-12-06	12	December	6	4	2009	2010-07-12	9999-12-12
6	6	2009-12-06	12	December	6	4	2009	2010-07-12	9999-12-12
7	7	2010-05-04	5	May	4	2	2010	2010-07-12	9999-12-12
8	8	2010-05-04	5	May	4	2	2010	2010-07-12	9999-12-12
9	9	2009-03-08	3	March	8	1	2009	2010-07-12	9999-12-12
10	10	2009-03-08	3	March	8	1	2009	2010-07-12	9999-12-12
*									

Bloc notes

Figure IV.21 : *La table PrescriptionMédicamentDateDim.*

4. les Tests

4.1. Avant refactoring

- Quand on lance le client et les deux serveur nous voyons que les serveurs envoient les informations et le client reçoit ces informations [Dra10].

Figure IV.22: *Les deux serveurs et le client.*

- Quand le client recevoir au moins une information dans chaque dimension l'utilisateur (le directeur) pourrait choisir quelle question veule connaitre.

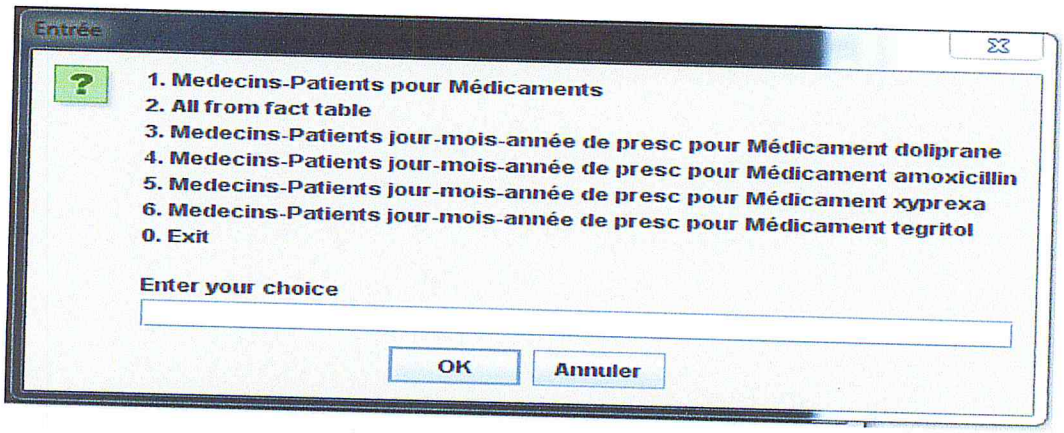
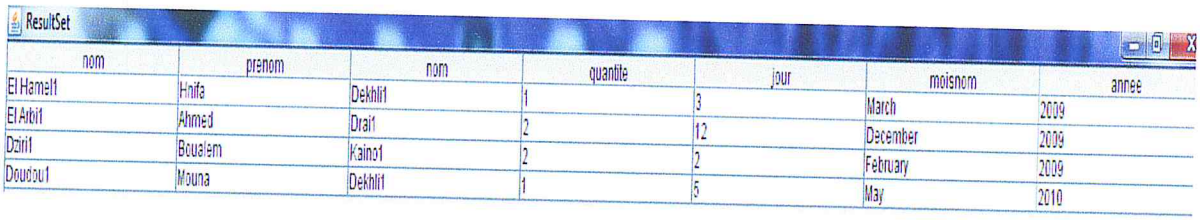


Figure IV.23 : *Quering.*[Dra10]

- Quand le directeur choisi une question le résultat sera affiché sur l'écran au format d'un tableau.



nom	prenom	nom	quantite	jour	moisnom	annee
El Hamel1	Hnifa	Dekhlif	1	3	March	2009
El Arbi1	Ahmed	Drai1	2	12	December	2009
Dziri1	Boualem	Kaino1	2	2	February	2009
Doudou1	Mouna	Dekhlif	1	5	May	2010

Figure III.24: *Reporting*[Dra10]

4.2. Après refactoring

- Quand on lance le client et les deux serveur, les trois formulaires suivant s'affichent :

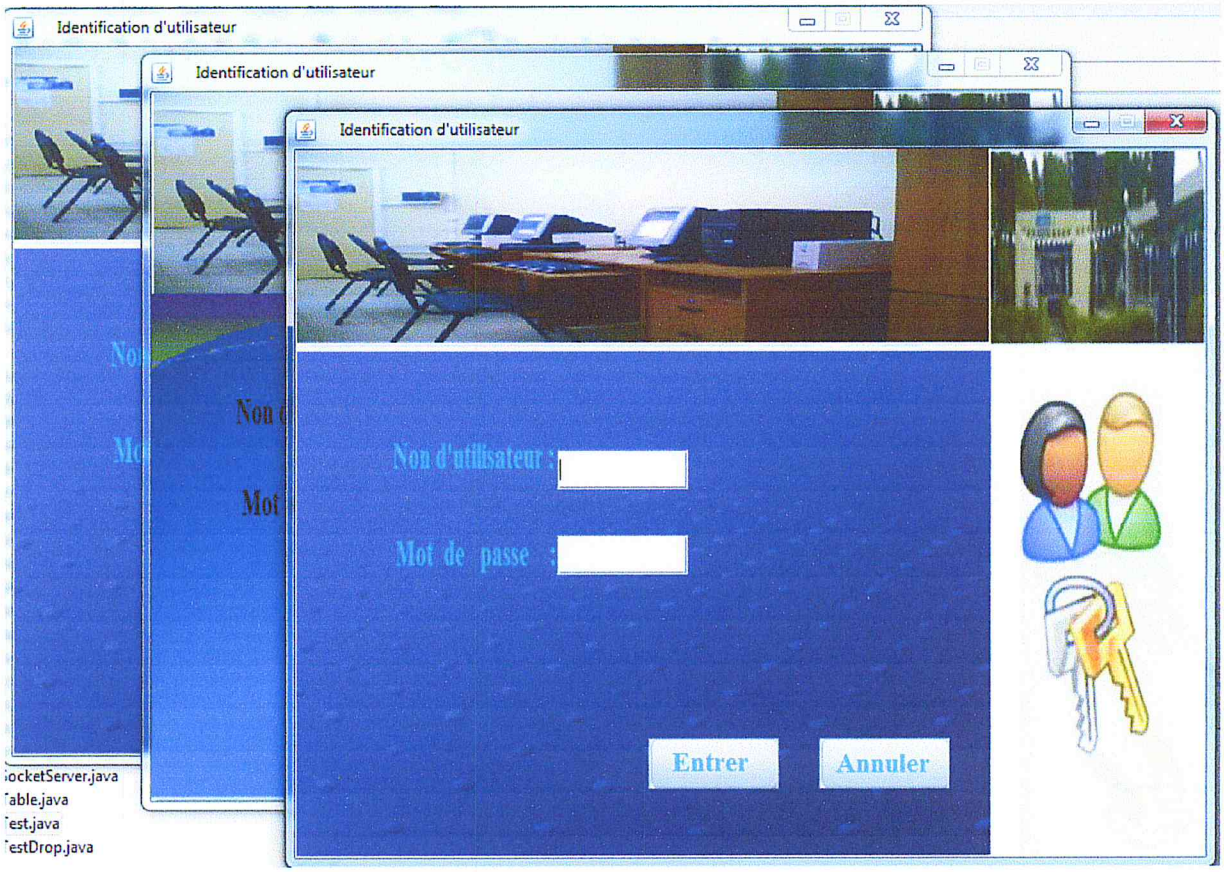


Figure IV.25:*Authentifications.*

- Si le « Nom d'utilisateur » et « Mot de passe » sont vérifiés pour (hôpital Farroudja, lopitale Alfabort, dw) les formulaires suivant s'affichent :

Figure IV.26: *Page principale pour les deux hôpitaux.*

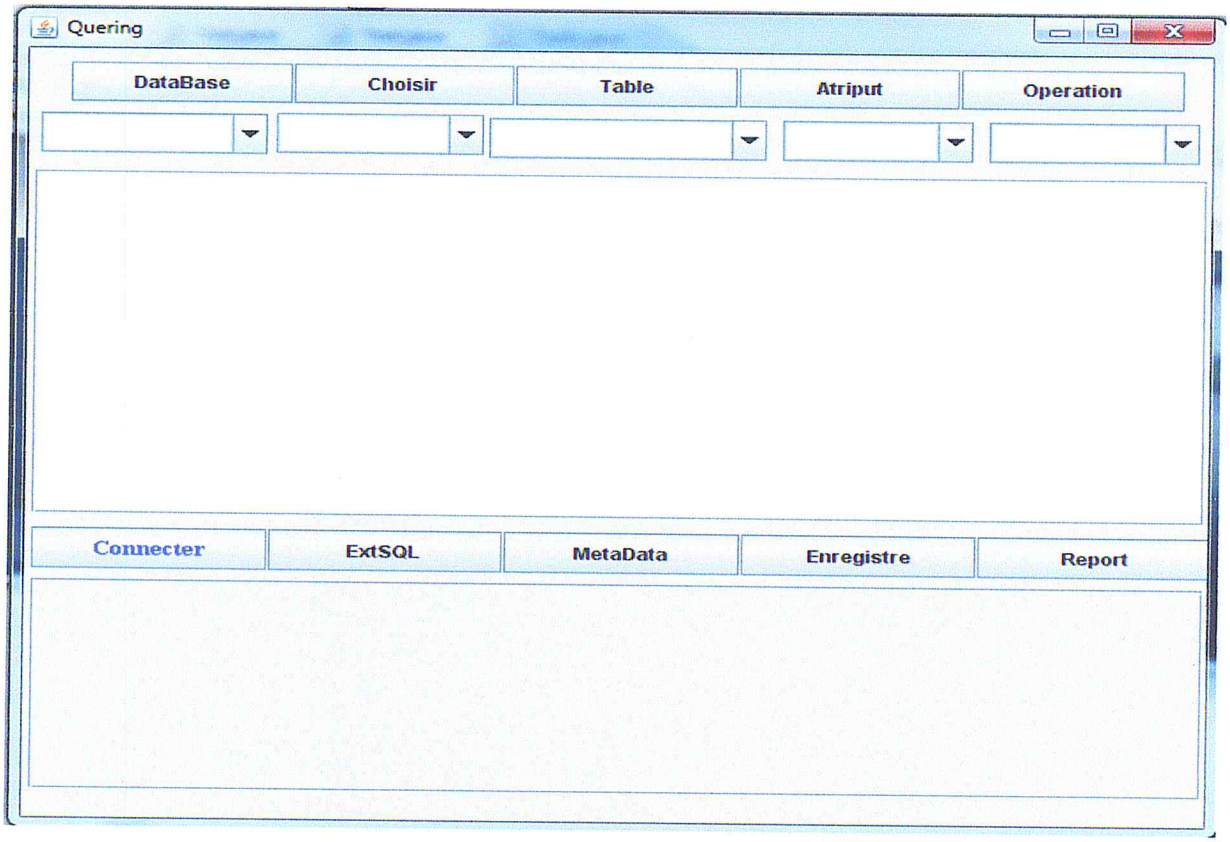


Figure IV.27:*Page principale de DW.*

- Quand l'utilisateur d'un serveur(hopital Alfabort par exemple) choisir de consulter une table (patient par exemple) les formilaires suivant s'afiches :

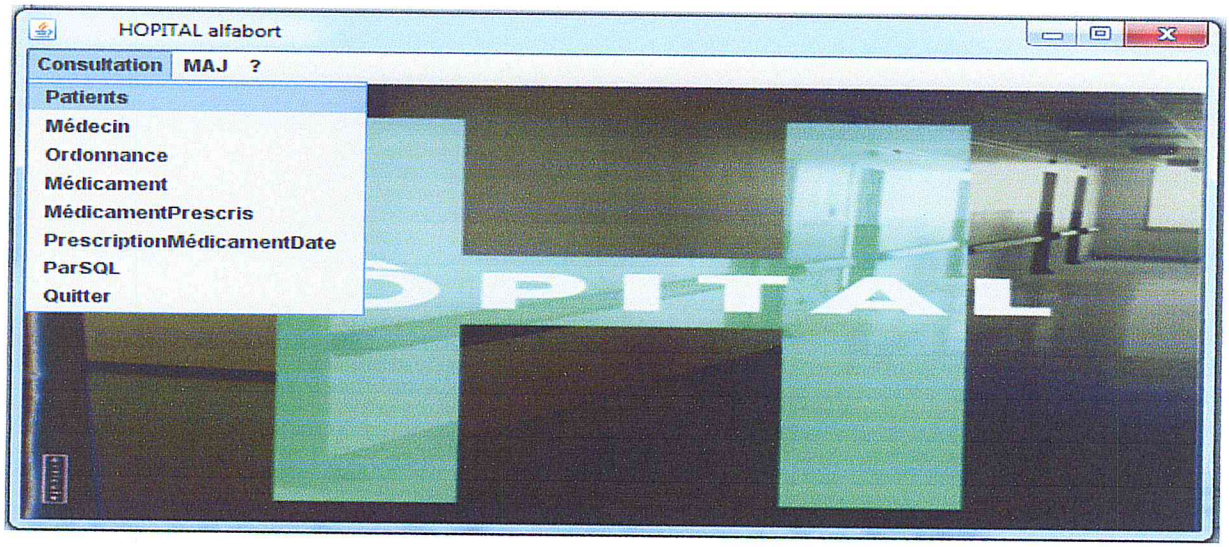


Figure IV.28: *Page Consultation.*

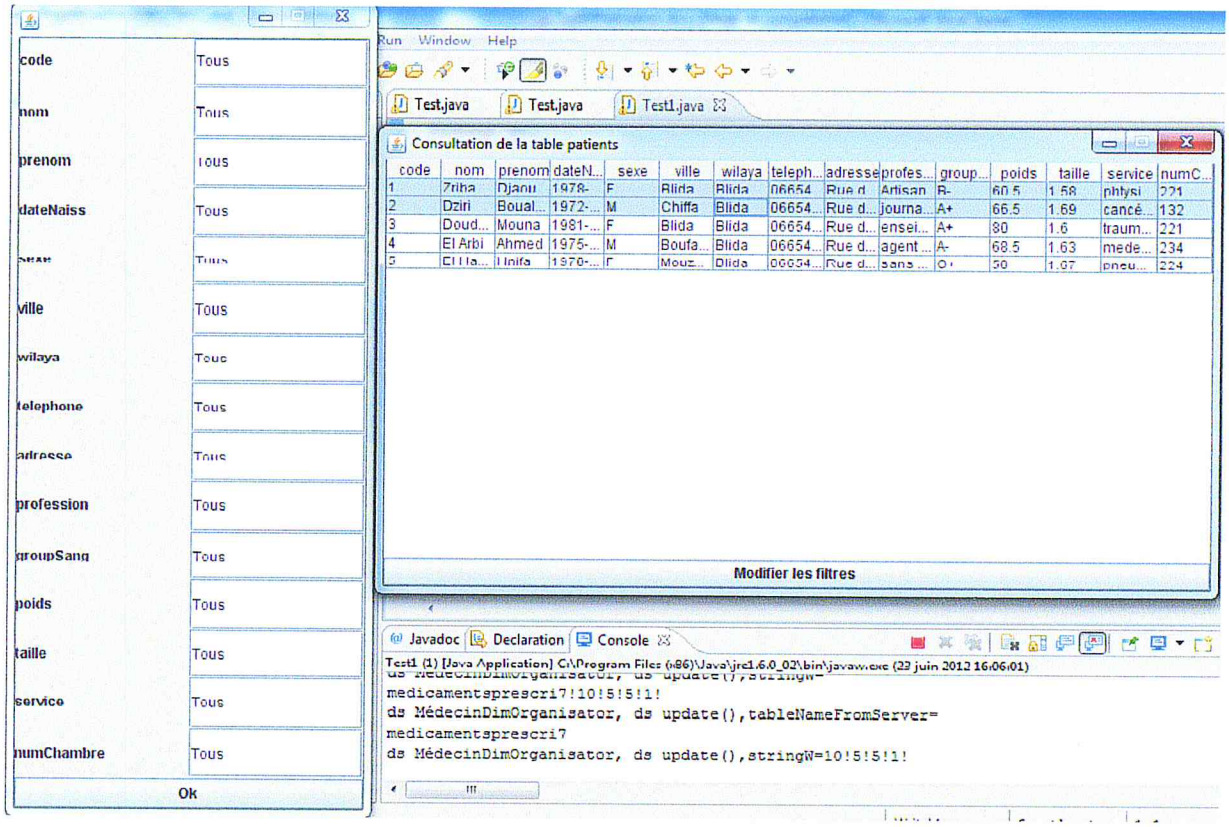


Figure IV.29: *Consulter les patients*

- Quand l'utilisateur d'un serveur(hopital par exemple) choisir d'ajouter dans une table (patient par exemple) le formilaire suivant s'afiche :

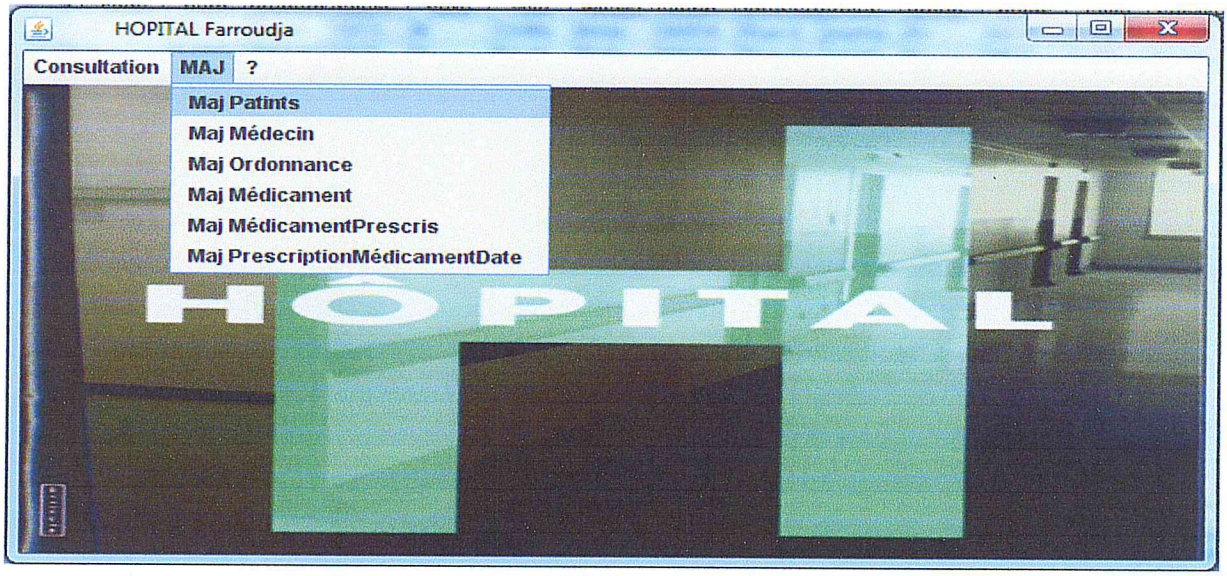


Figure IV.30:*MAJ*

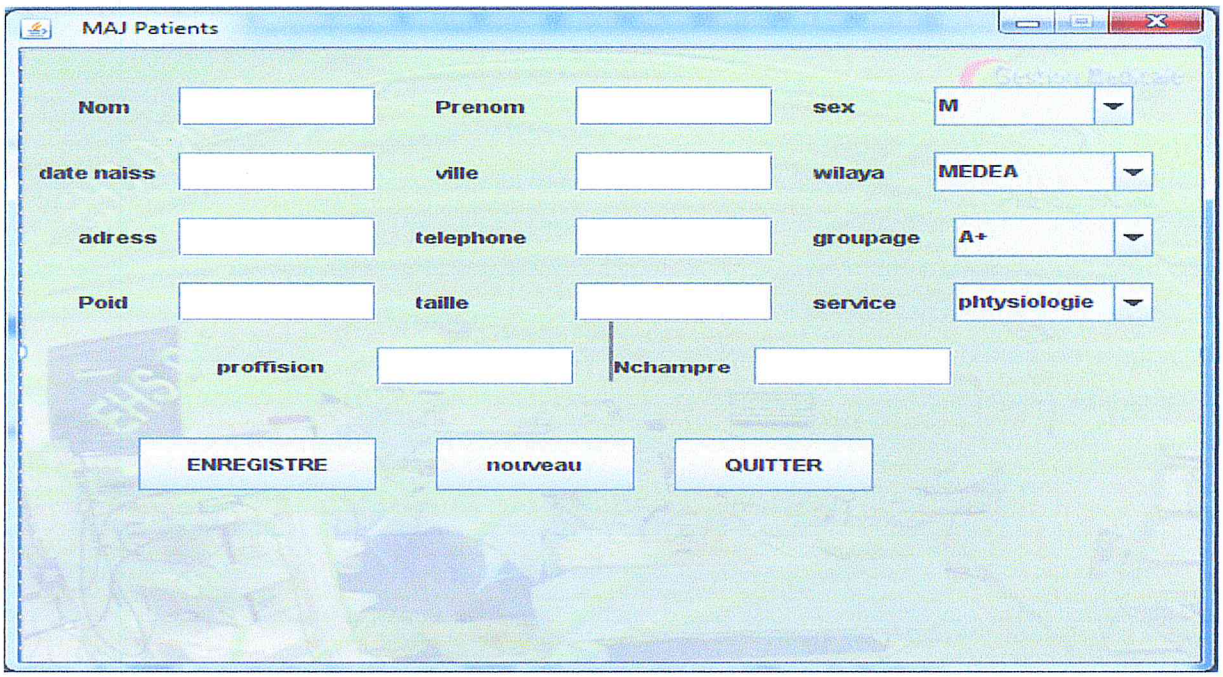


Figure IV.31: *MAJ Patients.*

- Quand l'utilisateur de DW cliquer sur « MetaData » le formilaire suivant s'afiches :

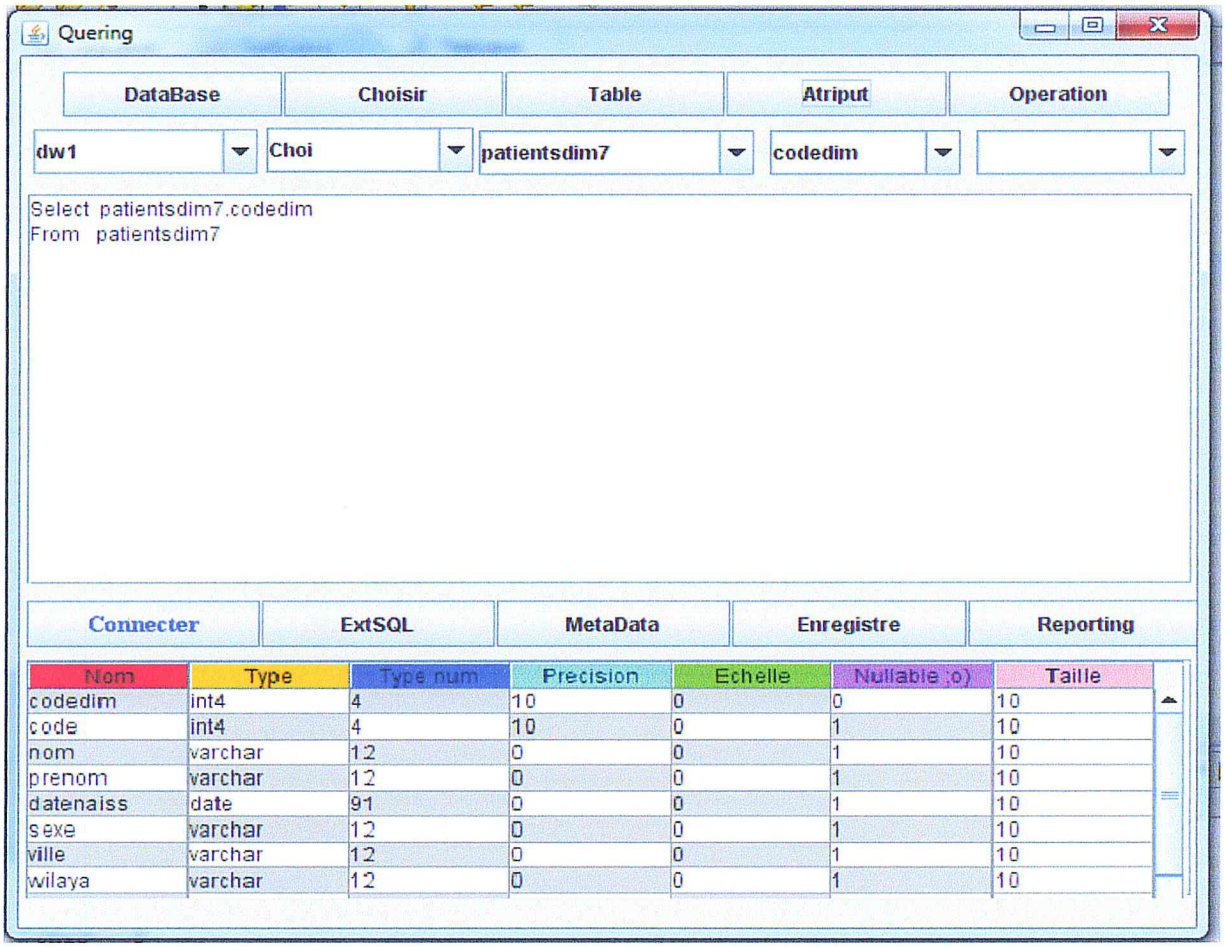


Figure IV.32: *MetaData.*

- Quand l'utilisateur de DW construire la requete SQL(directement ou utiliser le button pour faciliter de construire la requete) et cliquer sur « ExtSQL » le formilaire suivant s'afiches :

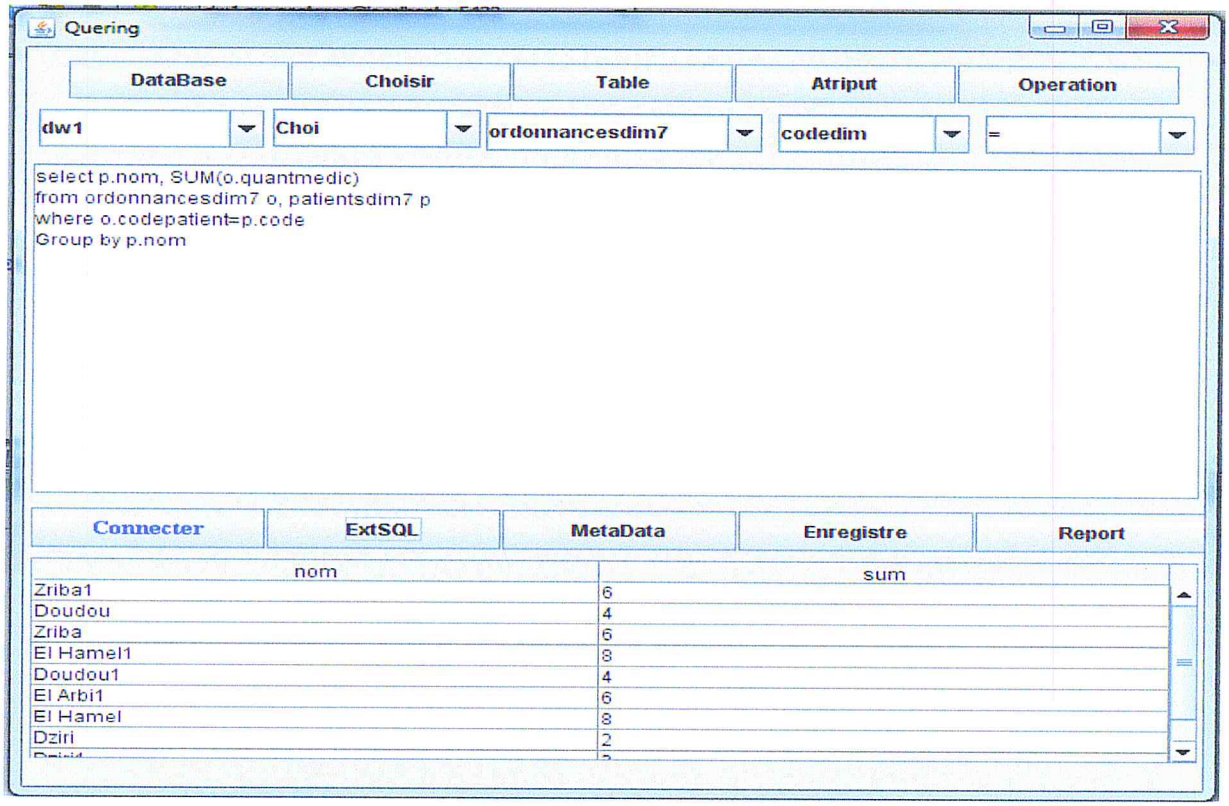


Figure IV.33: *Quering.*

- Quand l'utilisateur de DW cliquer sur « Report » les formilaires suivant s'afichent :

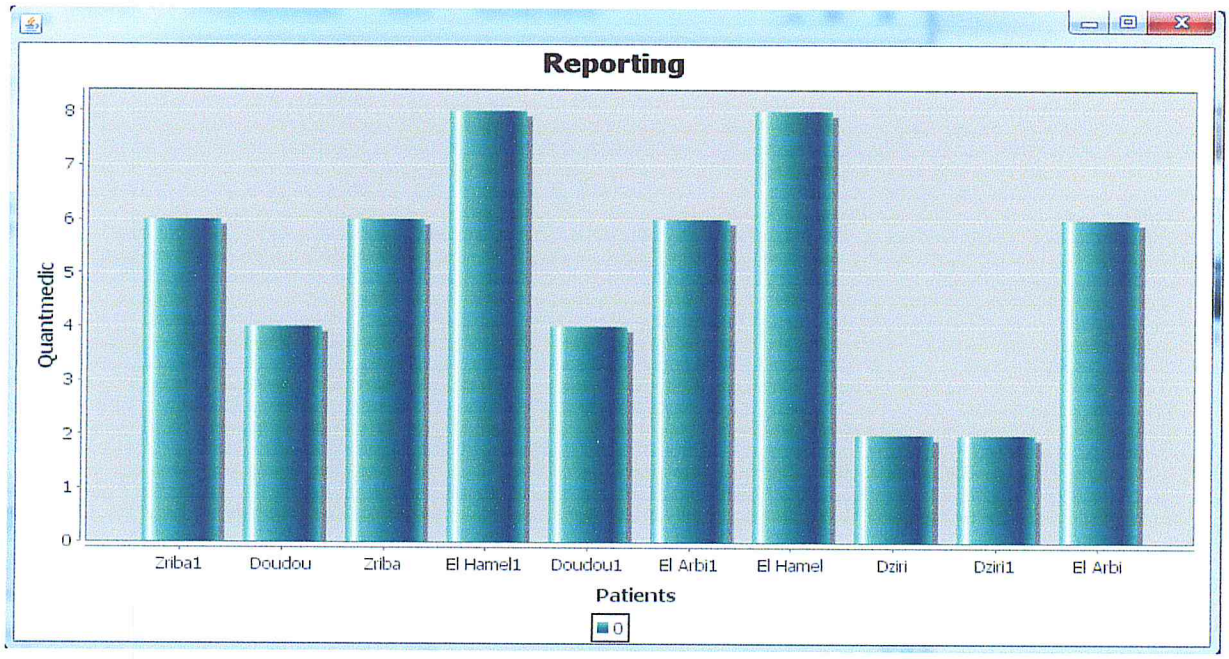


Figure IV.34:*Reporting1.*

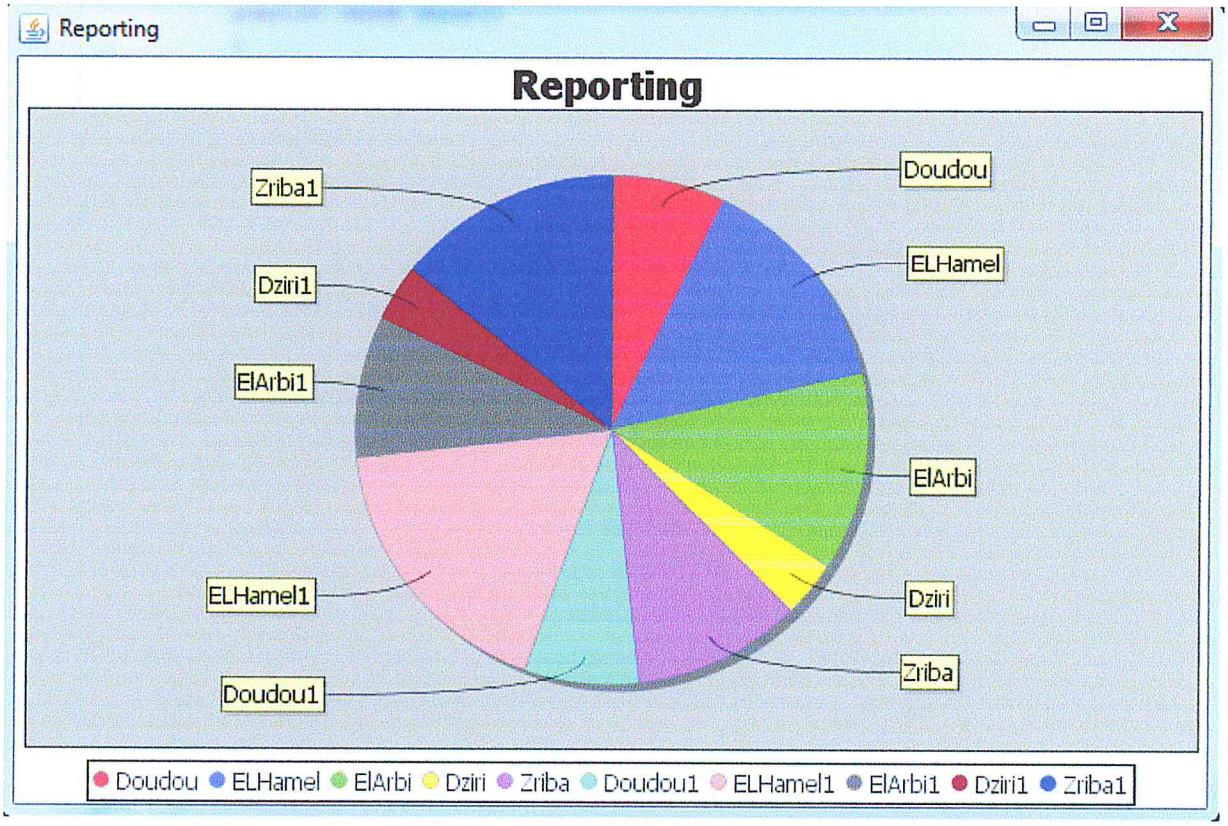


Figure IV.34:*Reporting2*

5. Conclusion

Dans ce chapitre nous avons abordé le déploiement de notre système. Nous avons Commencé par décrit les outils de développement, les diffèrent source de donnée utiliser, et nous avons fini par une présentation de différent prototype réalisé.

Conclusion Générale

Toutes les entreprises disposent de données. Elles proviennent soit de leurs systèmes opérationnels, soit de l'extérieur. Ces entreprises doivent posséder des informations nécessaires pour prendre de bonnes décisions. Pour cela, il est fondamental de mettre en place une informatique décisionnelle pour obtenir une meilleure compréhension des informations disponibles. Le but espéré par ces systèmes est de perfectionner le processus de prise de décision.

L'informatique décisionnelle apporte des nouvelles solutions pour la modélisation, l'interrogation et la visualisation de données dans un objectif d'aide à la décision.

Le travail présenté dans ce mémoire concerne la conception et la refactoring d'un DW en La finalité de ce travail est donc de mettre à la disposition du corps médical un outil de travail qui leur faciliterait la gestion des médicaments dans l'hôpital. L'approche DW a beaucoup de problème d'intégration et de modélisation.

Le principal but de ce travail est de permettre l'analyse du logiciel de DW temps réel développé et la re-conception de ses côtés qui le nécessitent, à travers une bonne re-fabrication, pour permettre sa maintenabilité.

Le modèle que nous avons utilisé gère le processus complet depuis les bases de données opérationnelles, en passant par une architecture client-serveur, ensuite un outil ETL, jusqu'à le DW, munis d'un module de Quering et un de Reporting simple, et tout cela en adoptant une architecture à base de patrons de conception (Observer/Observable et singleton).

Les futures tendances pourraient aller vers la généralisation de ce modèle afin d'inclure d'autres motifs. En outre, d'autres caractéristiques pourraient être rajoutées pour généraliser notre modèle pour tous les entrepôts temps réel à base de patrons, et l'appliquer pour le domaine médical et les autres besoins des hôpitaux.

Conclusion Générale

La finalité de ce travail serait de mettre, dans le futur, à la disposition du corps médical un outil de travail qui lui faciliterait la gestion des dossiers médicaux des malades en y incluant des données multimédias telles que les diagnostics de type texte classique ainsi que les images médicales qui peuvent être d'origine radiologique, IRM ou scanner. Les médecins traitants ont accès à toutes les informations concernant leur patient, sans se soucier du site où se trouvait initialement l'enregistrement physique des données.

Parmi les perspectives qui restent à explorer, nous pouvons citer en particulier la liste de développement future suivante :

- Intégrer les données provenant de système XML et de fichiers plats car parmi les avantages d'un modèle dimensionnel réside dans le fait qu'il est extensible pour accueillir des données et des besoins d'analyse non prévues au départ.
- Développement d'un outil DM pour la bonne exploitation des données du DW.
- De revoir la granularité de l'information à transférer.
- Le programme sera élargi pour le diagnostic des patients.
- L'incorporation d'un outil OLAP.

Bibliographie

[Bou10] Bouzidiel. H

Conception et implémentation d'un entrepôt de données et application des techniques du Data Mining pour l'analyse des échecs/succès des étudiants de l'ESI

Mémoire d'Ingénieur d'Etat en Informatique

ESI

2010

[Bou03] Bouquin. H

Le contrôle de gestion

P.U.F

2003.

[Boi04] BOISSIER. O

Analyse, Conception Objet (Design Pattern Introduction)

SMA/G2I/ENS Mines Saint-Etienne,

Avril 2004

[Bel09] Belgasmia. A

Etude et réalisation d'un systeme d'archivage

Mémoire d'Ingéniorat en Informatique

Université de saad dahlab blida,

2009.

[Sco10] Scott.P

Getting real-an introductionto real time data warehousing

www.rittmanmead.com

[Cha79] Chaudhuri S

An Overview of Data warehousing and OLAP Technology

1979

[Che07] Chevaillier. P, Cyrille. B

Techniques avancées de programmation Objet

Laboratoire d'Informatique des Systèmes Complexes(LISyC)

Ecole Nationale d'Ingénieurs de Brest (ENIB)

2007.

[Dev02] A. Devisy

Livre blanc e-Business Intelligence, l'effet internet sur le décisionnel

2002

[Dou05] Doucet. A and Gangarski. S

Entrepôts de données pour l'aide à la décision médicale

THÈSE pour obtenir le grade de Docteur de l'Université Joseph Fourier

27 Juin 2005.

[Dre01] Dresner. H

BI: Making the Data Make Sens

Gartner Group

2001.

[Dra10] Drai.A

Etude, modélisation et réalisation d'un entrepôt de données au moyen de la technologie orientée objets et des patrons de conception

Mémoire d'Ingénieur d'Etat en Informatique

USDB

2010

[Fow00] Fowler. M

Refactoring. Improving the Design of Existing Code

Addison-Wesley

2000.

[Flo97] Florijn. G, Meijers M. , et Van Winsen .P

Tool support for object-oriented patterns »

In Lecture Notes in Computer Science.

ECOOP97, Finland.

1997.

[Gab08] GABAY. J

UML 2 analyse et conception.

Paris : Dunod,

2008.

ISBN 978-2-10-053567-5.

[Lon03] Lonchamp. J

Génie Logiciel Quatrième partie:les techniques de conception

CNAM - CRA Nancy,

2003

[Inm02] Inmon. H

Building the Data Warehouse Third Edition

Wiley Computer Publishing

2002.

[Jea05] Jean-Philippe.

Refactoring des applications Java/J2EE

ÉDITIONS EYROLLES 61, bd Saint-Germain 75240 Paris

CEDEX 05

ISBN : 2-212-11577-6

[Joh97] Johnson. R.E

Frameworks = (components + patterns)",

Communications of the ACM, vol.40 , No.10, pp. 39-42,

Octobre 1997

[Kim97] Kimball. R

Entrepôt de données

International Thomson Publishing France

1997.

[Kim05] Kimball. R. et Caserta. J.

The Data warehouse ETL Toolkit

Wiley Publisshing,

INC

2005

[Kim02] Kimball R. et Ross M

Entrepôts de Données : Guide Pratique de Modélisation Dimensionnelle 2ème édition

2002.

[Mar99] Martel. F

DOCUMENTATION STRUCTURÉE DU DESIGN DES CADRES D'APPLICATION

Mémoire présenté à la Faculté des études supérieures de l'Université de Montréal.

Octobre 1999.

[Mar06] Marhoumi.F

Entrepôts de données XML : Développement d'un outil Extraction Transformation Load

Université Libre de Bruxelles Faculté des Sciences Appliquées,

20006

[Nak98] Nakache.D

Data warehouse et Data mining

Conservatoire Nationale des Arts et Métiers de Lille

Version 1.1

15 juin 2008.

[Oli06] Olivier.S

Introduction à la programmation orientée objets en JAVA

Edition2005-2006

[SR02] SD. Roger Whitney

Observer

San Diego State University, CS 635 Advanced Object-Oriented Design & Programming, Spring Semester,

2002.

[Site 1]

http://fr.wikipedia.org/wiki/Reporting.