N° d'ordre : 04/RS/ TCO Année Universitaire : 2009 /2010

UNIVERSITE D'ANTANANARIVO

ECOLE SUPERIEURE POLYTECHNIQUE

DEPARTEMENT TELECOMMUNICATION

MEMOIRE DE FIN D'ETUDES

en vue de l'obtention

du DIPLOME d'INGENIEUR

Spécialité : Télécommunication
Option : Réseau et Système (R.S)

par : **RALAIARY Andry Tanjona Harinosy**

CONCEPTION ET REALISATION D'UN LOGICIEL DE GESTION DE LOCATION DE VOITURE UTILISANT L'ARCHITECTURE CLIENT SERVEUR

Soutenu le 19 Septembre 2013 devant la Commission d'Examen composée de :

Président:

M. ANDRIAMIASY Zidora

Examinateurs:

M. RANDRIARIJAONA Lucien Elino

M. RASAMOELINA Jacques Nirina

M. RANDRIAMIHAJARISON Jimmy

Directeur de mémoire :

Mr. RATSIRANTO Albert

REMERCIEMENTS

Je ne saurais présenter ce mémoire sans rendre gloire à Dieu pour sa bonté et sa fidélité, de m'avoir donné la force, le courage et la santé durant l'élaboration de ce mémoire de fin d'études.

Ma vive gratitude s'exprime tout particulièrement à Monsieur ANDRIANARY Philippe, Professeur Titulaire, Directeur de l'École Supérieure Polytechnique d'Antananarivo, qui m'a donné l'opportunité de suivre mes études supérieures au sein de l'établissement.

Je tiens à témoigner ma reconnaissance et ma gratitude les plus sincères à Monsieur RATSIRANTO Albert, Ingénieur en chef de Télécommunication et enseignant au sein du Département Télécommunication, qui en tant que Directeur de ce mémoire, s'est toujours montré à l'écoute et très disponible tout au long de sa réalisation.

Ma sincère reconnaissance va à feu Monsieur RAZAKARIVONY Jules, Maître de Conférences, ancien Chef de Département Télécommunication, qui s'est toujours efforcé de trouver la meilleure voie pour nous, lors de notre formation.

Je tiens à remercier Monsieur RAKOTOMALALA Mamy Alain, Enseignant-Chercheur, Maître de conférences et Chef Département Télécommunication au sein de l'Ecole Supérieure Polytechnique d'Antananarivo de nous avoir acceptés et formés dans son département.

J'exprime également mes remerciements à Monsieur ANDRIAMIASY Zidora, Maître de Conférences, Enseignant-Chercheur à l'ESPA, qui nous a fait l'honneur de présider le Jury de ce mémoire.

J'exprime ma reconnaissance à tous les membres de jury qui ont voulu examiner ce travail :

- Monsieur RANDRIARIJAONA Lucien Elino, Assistant, Enseignant au sein du Département Télécommunication.
- Monsieur RASAMOELINA Jacques Nirina, Assistant d'enseignement et de recherche et enseignant au sein du Département Télécommunication.
- Monsieur RANDRIAMIHAJARISON Jimmy, enseignant-chercheur au sein du Département Télécommunication.

Mes vifs remerciements s’adressent également à tous les enseignants au sein du Département télécommunication ainsi qu’aux enseignants et au personnel de l’Ecole Supérieure Polytechnique d’Antananarivo qui ont assuré notre formation durant ces cinq années d’études.

Je n’oublierai pas ma famille pour leur soutien bienveillant et leurs encouragements, pour la réalisation de ce mémoire, comme en toute circonstance.

Et à tous ceux qui ont contribué de près ou de loin à l’élaboration de ce mémoire.

TABLE DES MATIERES

ABREVIATIONS

API	Application Programming Interface
ANSI	American National Standard Institute
DBA	Data Base Administrator
DDL	Data Definition Language
DML	Data Manipulation Language
DCL	Digital Command Language
EDI	Environnement de Développement Intégré
LGLV	Logiciel de Gestion de Location de Voiture
ISAM	Indexed Sequential Access Method
IDE	Integrated Development Environment
JAR	Java Archive
JDBC	Java Data Base Connectivity
JDK	Java Development Kit
JRE	Java Runtime Environment
MCD	Modèle Conceptuel de Données
OMT	Object Modeling Technique
ODBC	Open Data Base Connectivity
PC	Personal Computer
SUN	Stanford University Network
SQL	Structured Query Language
SGBD	Système de Gestion de Base de Données
UML	UnifiedModelingLanguage
URL	Uniform Resource Locator

VSAM Virtual Storage Access Method

INTRODUCTION

Face à la mondialisation et la globalisation, les entreprises veulent de plus en plus accroitre l'horizon de leur activité, réduire autant que possible leur coût d'exploitation, et aussi diminuer au maximum le temps d'exécution. Tous ces objectifs ne peuvent être atteints que si l'entreprise accompagne ces stratégies par une mise en place d'un système d'information en tant que services supports. Ainsi, les applications issues de cette démarche qui vont fournir ces services doivent-elles combiner le système d'information existant dans l'entreprise avec de nouvelles applications qui offrent ces services à des client et utilisateurs à n'importe quel moment. Aussi, ces services doivent-ils être : hautement disponible afin de répondre de manière fiable et efficiente aux différentes sollicitations des utilisateurs.

C'est cela qui m'a motivé à la conception et à la réalisation de la présente application intitulée LGLV «Logiciel de Gestion de Location de Voiture » utilisant l'architecture client-serveur, ce logiciel sert à mieux sécuriser, organiser et faciliter les accès aux données de l'entreprise. La réalisation de cette application nécessite diverses connaissances qui seront détaillées dans le présent mémoire.

Dans le premier chapitre nous allons voir ce que l'on entend par base de données, c'est à dire les objectifs, les avantages et les différents types de base de données. On va aussi parler des modèles conceptuels des données, puis un petit rappel sur l'algèbre relationnelle et pour terminer ce chapitre on va parler un peu du langage SQL « Structured Query Language ».

Le deuxième chapitre s'intitule « Accès aux bases de données à partir d'application JAVA ».Dans ce chapitre nous allons voir successivement :

- les outils nécessaires pour l'utilisation de JDBC (Java Data Base Connectivity) ;
- la présentation des classes de l'API JDBC ;
- la connexion et l'accès à une base de données ;
- l'obtention d'information sur la base de données;
- l'utilisation d'un objet PreparedStatement;
- l'utilisation des transactions
- et enfin l'amélioration des performances avec JDBC.

L'application « Gestion de location de voiture » utilise l'architecture client-serveur : il est primordial de parler d'Architecture client-serveur, et c'est ce que nous allons développer dans le troisième chapitre. Pour présenter l'architecture client-serveur on va aborder en premier lieu les différents modèles, en second lieu les principes généraux qui gouvernent l'architecture client-serveur et en dernier lieu les différents types d'architectures client-serveur.

Dans le dernier chapitre l'étude portera sur les étapes de la réalisation de l'application, Dans cette thématique, il convient d'abord de décrire les outils utilisés pendant la réalisation du logiciel, puis de déterminer des méthodologies de construction d'une application. Par ailleurs une explication de la méthode d'analyse MERISE® a été effectuée, suivie d'une modélisation de la partie statique et de la partie dynamique du système d'information du logiciel, permettant ainsi d'élaborer son MCD (Modèle Conceptuel de donnée). En dernier lieu, à la fin de ce chapitre figurera des exemples de fenêtres et quelques manipulations de l'application.

CHAPITRE 1
BASE DE DONNEES

1.1 Présentation générale

1.1.1 Introduction aux bases de données

Une base de données est un ensemble structuré d'informations enregistré avec le minimum de redondance. Elle doit être conçue pour permettre une consultation et une modification aisée de son contenu, si possible par plusieurs utilisateurs en même temps.

Le SGBD (Système de Gestions de Base de Données) est un concept informatique permettant d'insérer, de modifier et de rechercher efficacement des données spécifiques dans une grande masse d'informations. C'est aussi une interface facilitant le travail des utilisateurs en présentant l'information comme ils le souhaitent et donnant à chacun l'impression qu'il est seul à utiliser cette information.

Le SGBD est composé de trois couches successives :

- Le système de gestion de fichiers : Il gère le stockage physique de l'information. Il est dépendant du matériel utilisé (type de support, facteur de blocage, etc ...).
- Le SGBD interne : Il s'occupe du placement et de l'assemblage des données, gestion des liens et gestion des accès.
- Le SGBD externe : Il s'occupe de la présentation et de la manipulation des données au profil des concepteurs et des utilisateurs. Egalement il a pour attribution la gestion des langages de requêtes et des outils de présentation (états, formes, etc.).

1.1.2 Objectifs et avantages de l'utilisation des bases de données

Un système d'information peut être réalisé sans outil spécifique. On peut alors se demander quels sont les objectifs et avantages de l'approche SGBD par rapport aux systèmes de fichiers classiques. La réponse tient en neuf points fondamentaux :

- Indépendance physique : Les disques, la machine, les méthodes d'accès, les modes de placement, les méthodes de tris, et le codage des données ne sont pas apparents. Le SGBD

offre une structure canonique permettant la représentation des données réelles sans se soucier de l'aspect matériel.[2]

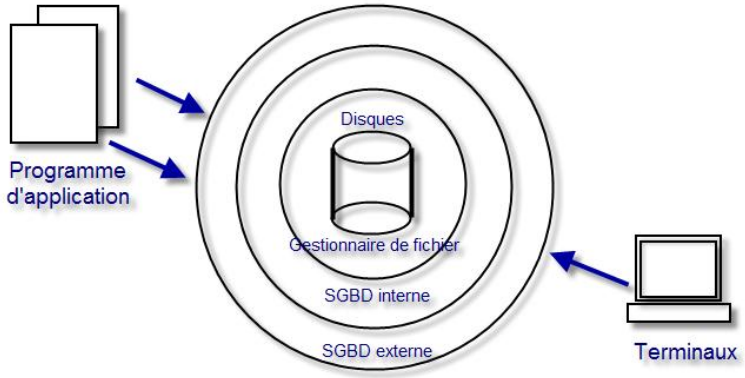

Figure 1.01 : *Le modèle à 3 couches*

- Indépendance logique : Chaque groupe de travail doit pouvoir se concentrer sur ce qui l'intéresse uniquement. Il doit pouvoir arranger les données comme il le souhaite même si d'autres utilisateurs ont une vue différente. L'administrateur doit pouvoir faire évoluer le système d'informations sans remettre en cause la vue de chaque groupe de travail.
- Manipulable par des non-informaticiens : Le SGBD doit permettre d'obtenir les données par des langages non procéduraux c'est-à-dire on doit pouvoir décrire ce que l'on souhaite sans décrire comment l'obtenir.
- Accès efficace aux données : Les accès disques sont lents relativement à l'accès à la mémoire centrale. Il faut donc offrir à l'utilisateur les meilleurs algorithmes de recherche de données qui sont physiquement contenues dans des disques.

Remarque: le système de gestion de fichiers y répond parfois pour des mono-fichiers (ISAM,VSAM etc...) mais dans le cas de relation multiples entre différents fichiers cela devient beaucoup plus complexe et dépend même parfois du contexte de la recherche effectuée.

- Administration centralisée des données : Le SGBD doit offrir aux administrateurs des données, des outils de vérification de cohérence des données, de restructuration éventuelle de la base, de sauvegarde ou de réplication. L'administration est centralisée et est réservée à un très petit groupe de personnes pour des raisons évidentes de sécurité.

- Non redondance des données : Le SGBD doit permettre d'éviter la duplication d'informations qui, outre la perte de place mémoire, demande des moyens humains importants ainsi que des ressource informatiques supplémentaires pour saisir et maintenir à jour plusieurs fois les mêmes données.
- Cohérence des données : Cette cohérence est obtenue par la vérification des contraintes d'intégrité. Cette dernière est une contrainte sur les données de la base, qui doit toujours être vérifiée pour assurer la cohérence de cette base. Les systèmes d'information sont souvent remplis de telles contraintes, le SGBD doit permettre une gestion automatique de ces contraintes d'intégrité sur les données. Par exemple :Un identifiant doit toujours être saisi, Le salaire doit être compris entre 4000F et 100000F, Le nombre de commandes du client doit être cohérent avec le nombre de commandes dans la base, L'emprunteur d'un livre doit être un abonné du club. Dans un SGBD les contraintes d'intégrité doivent pouvoir être exprimées et gérées dans la base et non pas dans les applications.
- Partage des données : Le SGBD doit permettre à plusieurs personnes (ou applications) d'accéder simultanément aux données tout en conservant l'intégrité de la base. Chacun doit avoir l'impression qu'il est seul à utiliser les données.
- Sécurité des données : Les données doivent être protégées des accès non autorisés ou mal intentionnés. Il doit exister des mécanismes permettant d'autoriser, contrôler et enlever des droits d'accès à certaines informations pour n'importe quel usager. Par exemple un chef de service pourra connaître les salaires des personnes qu'il dirige, mais pas de toute l'entreprise. Le système doit aussi être tolérant aux pannes: si une coupure de courant survient pendant l'exécution d'une opération sur la base, le SGBD doit être capable de revenir à un état dans lequel les données sont cohérentes.

Remarque : les neuf points ci-dessus, bien que caractérisant assez bien ce qu'est une base de données, ne sont que rarement réunis dans les SGBD actuels. C'est une vue idéale des SGBD. De plus, les SGBD sont interfacés à l'aide d'un langage unique : le SQL. Ce langage permet d'effectuer l'ensemble des opérations nécessaires sur la base de données, et permet aussi la gestion de transaction.

Une transaction est définie par quatre propriétés essentielles : ACID (Atomicité, Cohérence, Isolation, Durabilité).

Ces propriétés garantissent l'intégrité des données dans un environnement multiutilisateur :

- L'atomicité permet à la transaction d'avoir un comportement indivisible ; soit toutes les modifications sur les données dans la transaction sont effectives, soit aucune n'est réalisée. On comprend l'intérêt de ce concept dans l'exemple simple d'une transaction débitant un compte A et créditant un compte B : il est clair que la transaction est réussie si les deux opérations sont menées à leur termes.

- Cohérence : l'atomicité de la transaction garantit que la base de données passera d'un état cohérent à un autre état cohérent. La cohérence des données de la base est donc permanente.

- Isolation : l'isolation des transactions signifie que les modifications effectuées au cours d'une transaction ne sont visibles que par l'utilisateur qui effectue cette transaction. Au cours de la transaction, l'utilisateur pourra voir des modifications en cours qui rendent la base apparemment incohérente mais ces modifications ne sont pas visibles par les autres et ne le seront qu'à la fin de la transaction si celle-ci est correcte.

- Durabilité : la durabilité garantit la stabilité de l'effet d'une transaction dans le temps, même en cas de problème grave tel que la perte d'un disque dur.

1.1.3 Différents types de bases de données

Il existe différents types de base de données :

1.1.3.1 Les bases hiérarchiques :

Ce sont les premiers SGBD apparus (notamment avec IMS d'IBM). Elles font partie des bases navigationnelles constituées d'une gestion de pointeurs entre les enregistrements. Le schéma de la base doit être arborescent.

1.1.3.2 Les bases réseaux :

Sans doute les bases les plus rapides, elles ont très vite supplanté les bases hiérarchiques dans les années 70 (notamment avec IDS II d'IBM). Ce sont aussi des bases navigationnelles qui gèrent des pointeurs entre les enregistrements. Cette fois-ci le schéma de la base est beaucoup plus ouvert.

1.1.3.3 Les bases relationnelles :

Actuellement les bases le plus utilisées sont les bases relationnelles. Les données sont représentées en tables. Elles sont basées sur l'algèbre relationnelle et un langage déclaratif (en général le langage SQL).

1.1.3.4 Les bases déductives :

Les données sont aussi représentées en tables, le langage d'interrogation se base sur le calcul des prédicats et la logique du premier ordre.

1.1.3.5 Les bases objets :

Les données sont représentées en tant qu'instances de classes hiérarchisées. Chaque champ est un objet. De ce fait, chaque données est active et possède ses propres méthodes d'interrogation et d'affectation. [1][2]

1.1.4 Modèle conceptuel des données

Avant de s'attaquer à tout problème, il est toujours nécessaire de réfléchir profondément aux tenants et aboutissants de ce que l'on veut réaliser. La phase de conception nécessite souvent de nombreux choix qui auront parfois des répercussions importantes par la suite. La conception de bases de données ne fait pas exception à la règle. Les théoriciens de l'information ont donc proposé des méthodes permettant de structurer sa pensée et présenter de manière abstraite le travail que l'on souhaite réaliser. Ces méthodes ont donné naissance à une discipline« analyse » et à un métier « analyste ».

L'analyse est la discipline qui étudie et présente de manière abstraite le travail à effectuer. La phase d'analyse est très importante puisque c'est elle qui sera validée par les utilisateurs avant la mise en œuvre du système concret. Il existe de nombreuses méthodes d'analyse (AXIAL, OMT, Merise, UML, etc…). Merise sépare les données et les traitements à effectuer avec le système d'information en différents modèles conceptuels et physiques. Celui qui nous intéresse particulièrement ici est le MCD.

Le MCD (modèle conceptuel de données) est un modèle abstrait de la méthode Merise permettant de représenter l'information d'une manière compréhensible aux différents services de l'entreprise. Il permet une description statique du système d'informations à l'aide d'entités et d'associations.

Le travail de conception d'une base de données par l'administrateur commence juste après celui des analystes qui ont établi le MCD.

Voici quelques définitions propres au MCD :

1.1.4.1 La propriété

La propriété est une donnée élémentaire et indécomposable du système d'information. Par exemple une date de début de projet, la couleur d'une voiture, une note d'étudiant.

1.1.4.2 L'entité

L'entité est la représentation dans le système d'information d'un objet matériel ou immatériel ayant une existence propre et conforme aux choix de gestion de l'entreprise. L'entité est composée de propriétés. Par exemple une personne, une voiture, un client, un projet.

Figure 1.02 : *Représentation d'une entité*

1.1.4.3 L'association

L'association traduit dans le système d'information le fait qu'il existe un lien entre différentes entités.

Le nombre d'intervenants dans cette association caractérise sa dimension :

- réflexive sur une même entité ;
- binaire entre deux entités ;
- ternaire entre trois entités ;
- n-aire entre n entités.

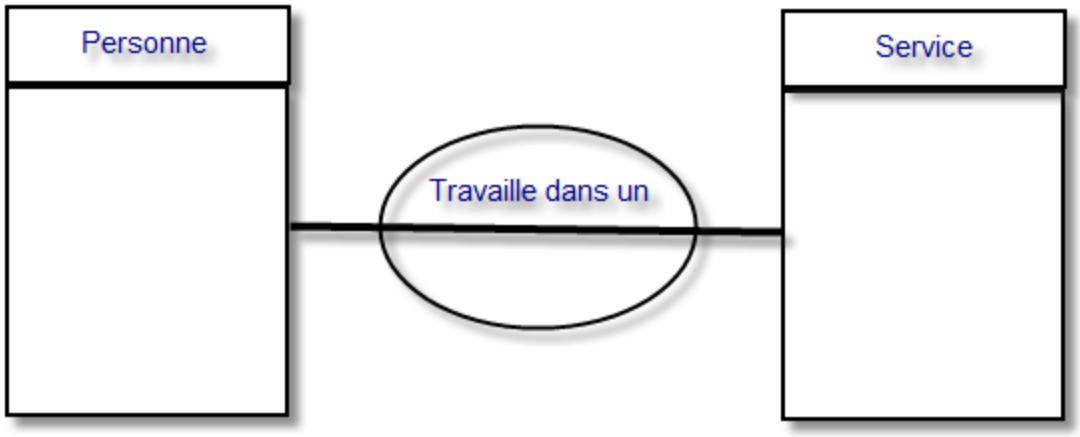

Figure 1.03 : *Représentation d'une association*

Des propriétés peuvent être attachées aux associations. Par exemple, un employé peut passer 25 % de son temps dans un service et 75 % de son temps dans un autre. L'association « travaille dans » qui relie une personne à un service portera la propriété « volume de temps passé »

Des propriétés peuvent être attachées aux associations.

1.1.4.4 Les cardinalités

Les cardinalités caractérisent le lien entre une entité et une association. La cardinalité d'une association est constituée d'une borne minimale et d'une borne maximale :

- minimale : nombre minimum de fois qu'une occurrence d'une entité participe aux occurrences de l'association, généralement 0 ou 1.
- maximale : nombre maximum de fois qu'une occurrence d'une entité participe aux occurrences de l'association, généralement 1 ou n.

Les cardinalités maximales sont nécessaires pour la création de la base de données, les cardinalités minimales sont nécessaires pour exprimer les contraintes d'intégrités.

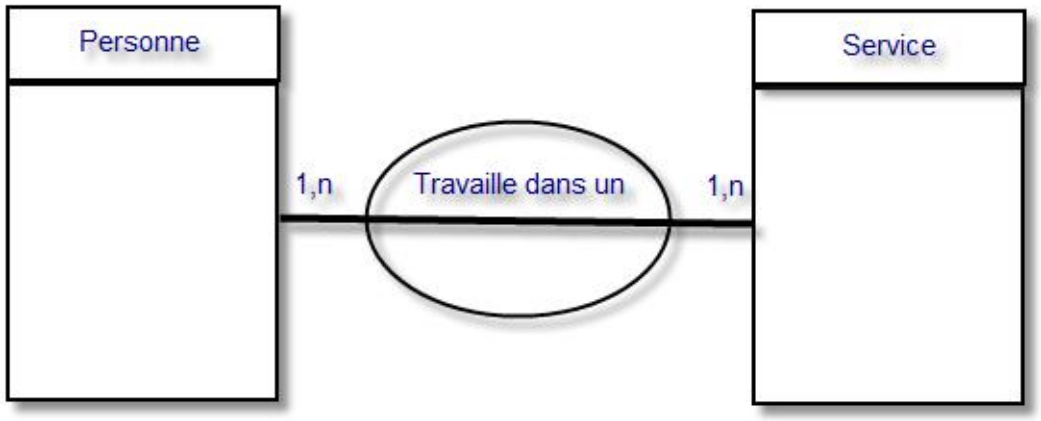

Figure 1.04 : *Représentation des cardinalités*

De ce schéma on en déduit que « une personne peut travailler dans plusieurs services ». On constate de plus que « dans chaque service il y a au moins 1 personne mais qu'il peut y en avoir plusieurs ». Enfin une mesure du « volume de travail » est stockée pour chaque personne travaillant dans un service donné. [1][2][5]

Remarque: il existe une notation "à l'américaine" dans laquelle on ne note que les cardinalités maximum.

Un lien hiérarchique est un lien 1:n en notation américaine.

Un lien maillé est un lien n:m en notation américaine.

1.1.4.5 Identifiant

L'identifiant d'une entité est constitué d'une ou plusieurs propriétés de l'entité telles qu'à chaque valeur de l'identifiant corresponde une et une seule occurrence de l'entité. L'identifiant d'une association est constitué de la réunion des identifiants des entités qui participent à l'association. L'identifiant est représenté en souligné dans le MCD.

1.2 L'algèbre relationnelle

1.2.1 Définition [9]

Algèbre relationnelle a été introduit par Codd en 1970 pour formaliser les opérations sur les ensembles. Ses principes reposent sur la création de nouvelles tables à partir des tables existantes, les nouvelles tables devenant des objets utilisables immédiatement. Les opérateurs de l'algèbre relationnelle permettant de créer les tables résultantes sont liés sur la théorie des ensembles.

Il existe deux familles d'opérations : les opérations ensemblistes et les opérations unaires.

D'une manière pratique les opérations les plus utilisées sont :

- Projection :

  La projection est l'ensemble des lignes de T obtenues en ne conservant que les colonnes i_1 ,i_2 ,i_3 ,…,i_k est noté$\pi_{i_1 ,i_2 ,i_3 ,\dots,i_k}$(T)

$$\pi_{i_1 ,i_2 ,i_3 ,\dots,i_k}(\mathrm{T}) = \{\mathrm{L}(i_1), \dots, \mathrm{L}i_k | \text{ L dans T}\} \quad (1.0.1)$$

  Soit (T) égale :

C1(nom)	C2(âge)	C3(adresse)	C4 (né à)
Bob	13	Lyon	Nice
Sam	7	Nice	Nice
Cathy	13	Brest	Brest
Julie	20	Lyon	Breast

Tableau 1.01: *Exemple d'une table T*

Exemple de Projection : π_{c_1,c_2} (T)

C1(nom)	C2(âge)
Bob	13
Sam	7
Cathy	13
Julie	20

Tableau 1.02: *Projection* $\pi_{c_1,c_2}(T)$

- Restriction :
  - Restriction par rapport à une constante : c'est l'ensemble des lignes L de T telles que L(i) $= a$ est noté σ_i=*a*(T).

$$\sigma_i = a(\mathrm{T}) = \{\mathrm{L} \mid \mathrm{L} \text{ dans T et L(i)} = a\} \quad (1.0.2)$$

C1(nom)	C2(âge)	C3(adresse)	C4 (né à)
Bob	13	Lyon	Nice
Julie	20	Lyon	Breast

Tableau 1.03: *Exemple restriction par rapport à une constante* $\sigma_{C3=}$ *« Lyon »(T)*

- Restriction inter-colonnes est l'ensemble des lignes *L* de *T* telles que *L(i)=L(j)* est noté σ_i*=j(T).*

$$\sigma_i = j(T) = \{\mathrm{L}\ \mathrm{L} \text{ dans T et } L(i) = L(j)\} \quad (1.0.3)$$

C1(nom)	C2(âge)	C3(adresse)	C4 (né à)
Sam	7	Nice	Nice
Cathy	13	Brest	Brest

Tableau 1.04: *Exemple de restriction inter-colonnes* σ_{C3}*=c4(T).*

- Jointure est l'ensemble des lignes L pouvant être obtenues par concaténation d'une ligne de *L1* de *T1* avec une *L2* de *T2* telle que $L_1(i) = L_2(j)$ est noté $T_1 \ltimes i=j\ T2$.

$$T_1 \ltimes i = j\ T2 = \{L_1.L_2 \mid L_1 \text{ dans } T_1 \text{ et } L_2 \text{dans } T_2 \text{ et } L_1(i) = L_2(j)\ \}(1.0.3)$$

C1(nom)	C2(adresse)
Bob	Lyon
Sam	Nice

Tableau 1.05: *Table Homme*

C1(nom)	C2(adresse)
Cathy	Brest
Julie	Lyon
Linda	Lyon

Tableau 1.06: *Table femme*

Quels sont les couples homme, femme d'une même ville ?

Homme$\ltimes_{c2=c2}$ $Femme$

C1(nom)	C2(adresse)	C3(nom)	C4(adresse)
Bob	Lyon	Julie	Lyon
Bob	Lyon	Linda	Lyon

Tableau 1.07: *Homme*$\ltimes_{c2=c2}$ $Femme$

1.2.2 Quelques remarques sur l'algèbre relationnelle

- L'algèbre relationnelle permet l'étude des opérateurs entre eux (commutativité, associativité, groupes d'opérateurs minimaux etc...).Cette étude permet de démontrer l'équivalence de certaines expressions et de construire des programmes d'optimisation qui transformeront toute demande en sa forme équivalente la plus efficace.

- D'une manière pratique les opérations les plus utilisées sont la projection, la restriction et la jointure naturelle. L'opération de jointure est en général très coûteuse. Elle est d'ailleurs proportionnelle au nombre de *tuples* du résultat et peut atteindre m * n *tuples* avec m et n les nombres de *tuples* des deux relations jointes.

- Avant d'avoir des requêtes efficaces en temps il est toujours préférable de faire des restrictions le plus tôt possible afin de manipuler des tables les plus réduites possibles.

- Les opérations de l'algèbre relationnelle respectent à certaines lois algébriques : commutativité, associativité etc... pour peu que l'ordre des colonnes dans les tables utilisées dans la base de données soit sans importance. C'est pourquoi les colonnes sont toujours nommées.

1.3 Le langage SQL

SQL est un langage de définition et de manipulation de bases de données relationnelles. Son nom est une abréviation de « Structured Query Language » (langage d'interrogation structuré).

SQL est un standard qui a été normalisé par l'organisme ANSI .Il existe aussi des « dialectes » propre à qui, pour chaque produit SQL réalisé, proposent des différences ou des compléments par rapport à la norme.

SQL contient un langage de définition de données, le DDL, permettant de créer, modifier ou supprimer les définitions des tables de la base par l'intermédiaire des ordres CREATE, DROP, et ALTER. Il contient aussi un langage de manipulation de données, le DML, par l'intermédiaire des ordres SELECT, INSERT, UPDATE, DELETE. Il contient enfin, le DCL un langage de gestion des protections d'accès aux tables en environnement multiutilisateurs par l'intermédiaire des ordres GRANT, REVOKE [1][2][7].

DDL	DML	DCL
ALTER CREATE COMMENT DESCRIBE DROP RENAME	DELETE INSERT SELECT UPDATE	GRANT REVOKE

Tableau 1.08: *Ordres SQL principaux*

Une requête SQL peut être utilisée de manière interactive ou incluse dans un programme d'application, quel que soit le langage.

Toutes les instructions SQL se terminent par un point-virgule. Un commentaire peut être introduit dans un ordre SQL par les signes /* et */ ou par le caractère % qui traite toute texte jusqu'à la fin de la ligne comme commentaire.

1.3.2 L'obtention des données

L'obtention des données se fait exclusivement par l'ordre SELECT. La syntaxe minimale de cet ordre est :

SELECT< liste des noms de colonnes> FROM< liste des noms de tables> ;

Elle permet aussi de faire de recherche c'est à dire de sélectionner des lignes spécifiques de la base de données en fonction de critères de recherche. Elle prend la forme suivante :

SELECT< liste des noms de colonnes> FROM< liste des noms de tables> WHERE nom_colonne=valeur ;

Mais nous verrons qu'elle peut être enrichie, de très nombreuses clauses permettant notamment d'exprimer les projections, les restrictions, les jointures, les tris.

Ordre des clauses du SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY

Tableau 1.09: *Ordres des clauses du SELECT*

1.3.3 La mise à jour d'informations

Pour le contenu d'une table relationnelle trois types de mise à jour sont nécessaires : l'ajout de nouveaux tuples, le changement de certain tuples et la suppression de certains tuples.

1.3.3.1 Insertion

INSERT permet d'ajouter des lignes dans une table. Dans sa forme la plus générale, SQL demande que les noms des colonnes soient explicitement cités. Les valeurs insérées sont alors en correspondance avec l'ordre dans lequel les colonnes sont citées.

Si l'ordre INSERT contient une clause VALUE alors une seule ligne est insérée dans la table ; si l'ordre INSERT contient une clause SELECT, alors plusieurs lignes peuvent être simultanément insérées dans la table. L'insertion de tuples peut donc se faire soit en extension par la clause VALUE soit en intension par un ordre SELECT imbriqué.

Une table citée dans le champ INTO de l'ordre INSERT ne peut pas être citée dans le champ FROM du sous-select de ce même INSERT. Il n'est donc pas possible d'insérer des éléments dans une table à partir d'une sous-sélection de cette même table.

1.3.3.2 Mise à jour

L'ordre UPDATE permet de modifier des lignes dans une table. L'expression caractérisant la modification à effectuer peut être une constante, une expression arithmétique ou le résultat d'un select imbriqué.

L'ordre UPDATE peut parfois poser des problèmes d'intégrité de la base. Par exemple, si une table possède comme clé unique un entier de 1 à n, que doit faire le SGBD lors d'une incrémentation de 1 de cette clé ?

Certain SGBD vérifient la cohérence de la base uniquement après chaque ordre SQL. C'est le cas d'Oracle ou DB2. Dans ce cas cette requête ne pose aucun problème. D'autres en revanche vérifient la cohérence de la base après chaque modification de ligne. C'est le cas d'Access. Dans ce cas, il risque d'y avoir un problème d'unicité de la clé durant l'exécution de la requête. Ce choix du mode de vérification de l'intégrité de la base n'est toujours pas tranché et il est possible de trouver selon les SGBD les deux modes d'exécution. D'une manière générale il n'est pas conseillé de faire des UPDATE sur des colonnes utilisées dans une clé primaire. [2][7]

1.3.3.3 Suppression

L'ordre DELETE permet de supprimer des lignes dans une table selon une qualification fixée. L'ordre DELETE FROM <table> ; permet de vider complètement une table. Néanmoins, dans ce cas, la table existe toujours bien qu'elle soit vide.

1.3.4 Définition des données : le DDL

Avant de parler des ordres de description de données DDL, précisions brièvement les types de données autorisés dans SQL.

1.3.4.1 Les types de données

Les types de données que l'on peut représenter dans un SGBD sont fortement dépendants de l'architecture de l'ordinateur sur lequel il tourne. Il existe donc de nombreuses variantes de types selon les SGBD. Néanmoins certains types se retrouvent dans la plupart des SGBD.

Voici les principaux types :

Les types alphanumériques :

CHAR(n) : longueur fixe de n caractères, n_max= 16383.

VARCHAR(n) : longueur variable, n représente le maximum.

Les types numériques :

NUMBER (n,d) : nombre de n chiffres dont d après la virgule ;

SMALLINT : mot signé de 16 bits (-32768 à 32767) ;

INTEGER : double mot signé de bits (-2E31 à 2 E31-1) ;

FLOAT : numérique flottant ;

Les types gestions de temps :

DATE : champ date (ex :31/12/2010)

TIME : champ heure (ex : 18 :45 :21.21) ;

TIMESTAMP : regroupe DATE et TIME ; [2][7]

1.3.4.2 La création de tables

Lorsque les données sont définies, on peut créer les tables. La commande CREATE TABLE permet de définir des colonnes, de leur associer un type de données et d'y ajouter des contraintes à vérifier. La syntaxe la plus simple est la suivante :

CREATE TABLE<nom-de-table>(<nom-de-colonne><type de données>,…)

1.3.4.3 Expression des contraintes d'intégrités

Le DDL doit offrir la possibilité d'exprimer ces contraintes dans la requête de création de table. Il est possible d'accoler différentes clauses gérant ces contraintes, bien qu'aucune ne soit obligatoire : il est notamment possible de nommer une contrainte de colonne :CONSTRAINT *nom* permet de nommer une contrainte.

CONSTRAINT *nom*	Permet de nommer *nom* une contrainte
DEFAULT	Précise une valeur par défaut
NOT NULL	Force la saisie de la colonne
UNIQUE	Vérifie que toutes les valeurs sont différentes
CHECK *condition*	Vérifie la condition *condition* précisée

Tableau 1.10: *Contrainte de colonne*

On peut nommer une contrainte de table notée CONSTRAINT *nom*, ainsi que de définir une clé comme PRIMARY KEY qui permet de spécifier que la colonne définit la clé primaire de la table. La clé primaire peut porter sur plusieurs colonnes. Il n'y a bien sûr qu'une clause de ce genre par table. Ainsi, les colonnes qui constituent la clé primaire ne peuvent plus être nulles et chaque clé doit être unique. Dans la majorité des SGBD un index est automatiquement créé sur cette clé. Et enfin l'intégrité référentielle nommé FOREIGN KEY (liste-coll) REFERENCES table(liste-col2) permet de spécifier que les colonnes liste-coll de la table en cours de définition référencent la clé primaire liste-col2 de la table étrangère spécifiée. La clé étrangère peut porter sur plusieurs colonnes. Il peut bien sûr y avoir plusieurs clés étrangères dans une même table. Les modifications automatiques à faire sur les clés étrangères en cas de changement de la clé primaire associée sont précisées par les clauses ON DELETE et ON UPDATE.

Une convention généralement admise consiste à nommer les contraintes d'intégrité référentielle par un nom préfixé par PK pour la clé primaire et par FK pour les clés étrangères.

CONTRAINT	Permet de nommer une contrainte
PRIMARY KEY	Déclare que la colonne est clé primaire
FOREIGN KEY	Déclare que la colonne est clé étrangère

Tableau 1.11: *Contrainte de table*

1.4 Conclusion

Pour obtenir une base de données fiable, il faut bien choisir le type de base de données approprié au besoin. Il faut aussi faire une analyse approfondie des travaux et enfin de maîtriser un langage de définition de données comme SQL.

CHAPITRE 2
L'ACCES AUX BASES DE DONNEES A PARTIR D'APPLICATIONS JAVA

2.1 Introduction

L'application LGLV est écrite avec le langage de programmation Java et attaque une base de données MySQL. C'est la raison de ce chapitre intitulé« Accès aux bases de données à partir d'application JAVA ».JDBC est l'acronyme de Java Data Base Connectivity et désigne une API définie par Sun pour permettre un accès aux bases de données avec Java.

Ce chapitre abordera successivement :

- les outils nécessaires pour utiliser JDBC ;
- les types de pilotes JDBC ;
- la présentation des classes de l'API JDBC ;
- la connexion et l'accès à une base de données ;
- l'obtention d'information sur la base de données ;
- l'utilisation d'un objet PreparedStatement ;
- l'utilisation des transactions
- et enfin l'amélioration des performances avec JDBC.

2.2 Les outils nécessaires pour utiliser JDBC

Pour pouvoir utiliser JDBC, il faut un pilote qui est spécifique à la base de données à laquelle on veut accéder. Avec le JDK, Sun fournit un pilote qui permet l'accès aux bases de données via ODBC. Ce pilote permet de réaliser l'indépendance de JDBC vis-à-vis des bases de données.

2.3 Type de pilote JDBC

Il existe quatre types de pilote JDBC :

- JDBC-ODBC bridge (type 1)

- Un driver écrit en java qui appelle l'API native de la base de données (Type 2)
- Un driver écrit en java utilisant un middleware (type 3)
- Un driver java utilisant le protocole natif de la base de données (type 4)

2.3.1 JDBC-ODBC bridge (Type 1)

Le pont JDBC-ODBC s'utilise avec ODBC et un pilote ODBC spécifique pour la base à accéder. Cette solution fonctionne très bien sous WINDOWS car il existe des pilotes ODBC pour la quasi-totalité des bases de données. Par contre cette solution « simple » pour le développement possède plusieurs inconvénients dont :

- La multiplication du nombre de couche rend complexe l'architecture et détériore un peu les performances.
- Lors du déploiement, ODBC et son pilote doivent être installé sur tous les postes où l'application va fonctionner.
- La partie native d'ODBC et son pilote rend l'application moins portable car elle est dépendante d'une plateforme

2.3.2 Un driver écrit en java qui appelle l'API native de la base de données (Type 2)

Ce type de driver convertit les ordres JDBC pour appeler directement les API de la base de données via un pilote natif sur le client. Ce type de driver nécessite aussi l'utilisation de code natif sur le client.

2.3.3 Un driver écrit en java utilisant un middleware (type 3)

Ce type de driver utilise un protocole réseau propriétaire spécifique à une base de données. Un serveur dédié reçoit les messages par ce protocole et dialogue directement avec la base de données. Ce type de driver peut être facilement utilisé par une applet mais dans ce cas le serveur intermédiaire doit obligatoirement être installé sur la machine contenant le serveur web.

2.3.4 Un driver java utilisant le protocole natif de la base de données (type 4)

Ce type de driver, écrit en java, appelle directement le SGBD par le réseau. Le driver est fourni par l'éditeur de la base de données.

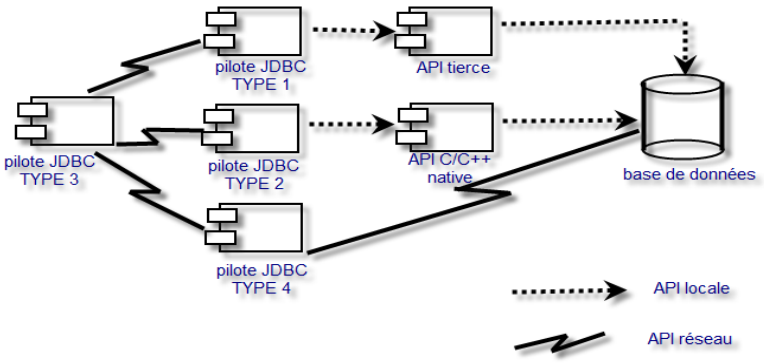

Figure 2.01 : *Les différentes sortes de pilotes JDBC*

2.4 La présentation des classes de l'API JDBC

Toutes les classes de JDBC sont dans le package java.sql donc il faut l'importer dans tous les programmes devant utiliser JDBC (import java.sql.* ;).Ce package est inclus dans JDK.

Il y a quatre classes importantes pour accéder aux données dont : *DriverManager*, *Connection*, *Statement* et *ResulSet.*

Classe	Rôle
DriverManager	Charge et configure le driver de la base de données
Connection	Réalise la connexion et l'authentification à la base de données
Statement	Contient la requête SQL et la transmet à la base de données.
ResulSet	Permet de parcourir les informations retournées par la base de données dans le cas d'une sélection de données

Tableau 2.01: *Les quatre Classes importantes pour accéder aux données et leurs rôles*

2.5 La connexion à une base de données

2.5.1 Le chargement du pilote

Pour se connecter à une base de données via ODBC, il faut tout d'abord charger le pilote JDBC-ODBC qui fait le lien entre les deux, Cette action peut être réalisée par :Class.forName (« sun.jdbc.odbc.JdbcOdbcDriver*) ;

Pour se connecter à une base en utilisant un driver spécifique, la documentation du driver fournit le nom de la classe à utiliser. Par exemple, si le nom de la classe est jdbc.DriverXXX, alors le chargement du driver se fera avec le code suivant : Class.forName("jdbc.DriverXXX"); (ex : Class.forName (« postgresql.Driver ») ;)

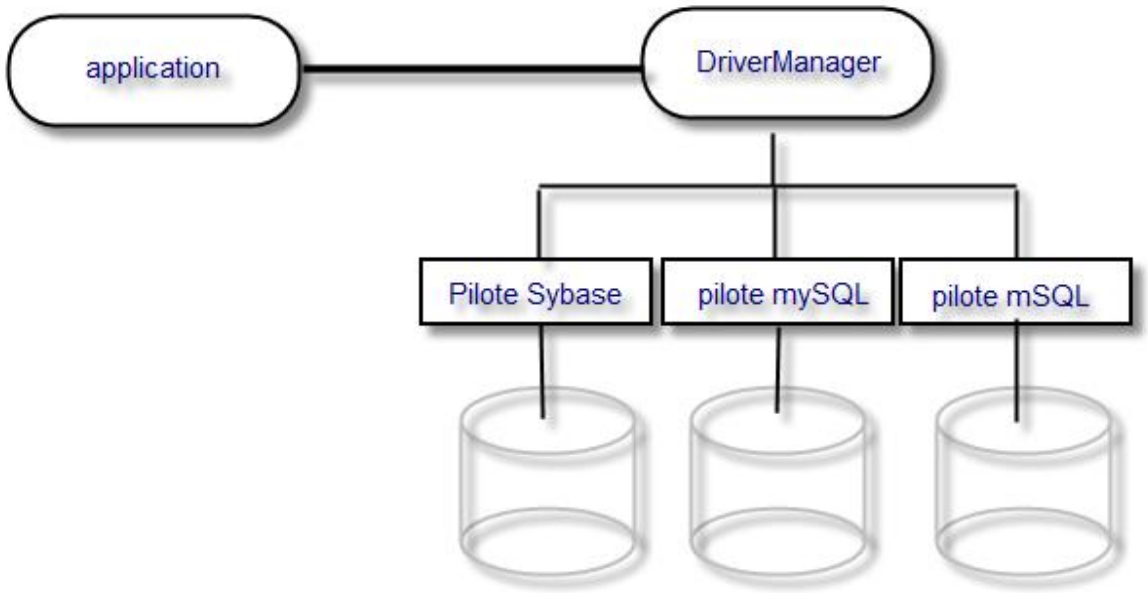

Figure 2.02 : *JDBC cache à l'application les spécificités d'implémentation de chaque base de données*

2.5.2 L'établissement de la connexion

Pour se connecter à une base de données, il faut instancier un objet de la classe *Connection* en lui précisant sous forme d'URL la base à accéder. Comme indique le tableau suivant

Exemple : Etablir une connexion sur la base testDB via ODBC
String DBurl = "jdbc:odbc:testDB"; con = DriverManager.getConnection(DBurl);

Tableau 2.02: *Exemple d'établissement d'une connexion sur la base testDB via ODBC*

La syntaxe URL peut varier d'un type de base de données à l'autre mais elle est toujours de la forme protocole :sous_protocole :nom, « jdbc » désigne le protocole et vaut toujours « jdbc ». « odbc » désigne le sous protocole qui définit le mécanisme de connexion pour un type de bases de données.

Le nom de la base de données doit être celui saisi dans le nom de la source sous ODBC ; La méthode *getConnection()* peut lever une exception de la classe java.sql.SQLException. Voici un exemple de code qui décrit la création d'une connexion avec le nom d'utilisateur et un mot de passe.

Connection con =DriverManager.getConnection (url, " nomUtilisateur", " motDePasse");

2.6 L'accès à la base de données

2.6.1 Les classes utiles pour obtenir des informations sur la base de données

Les objets qui peuvent être utilisés pour obtenir des informations sur la base de données sont indiqués dans le tableau ci-dessous. [1]

Classe	Rôle
DatabaseMetaData	Information à la base de données: nom des tables, index, version…
ResultSet	Résultat d'une requête et information sur une table. l'accès se fait par enregistrement par enregistrement
ResultSetMetaData	Information sur les colonnes (nom et type) d'un ResultSet

Tableau 2.03: *Les rôles des classes pour l'accès à la base de données*

2.6.2 L'exécution de requêtes SQL

Les requêtes d'interrogation SQL sont exécutées avec les méthodes d'un objet *Statement* que l'on obtient à partir d'un objet *Connection.*

Exemple:

ResultSetrésultats = null;
 	
String requete = "SELECT * FROM client";
 	
try {
 	
Statement stmt = con.createStatement();
 	
résultats = stmt.executeQuery(requete);
 	
} catch (SQLException e) {
//traitement de l'exception
}

Tableau 2.04: *Exemple de requête SQL avec traitement de l'exception*

Un objet de la classe *Statement* permet d'envoyer des requêtes SQL à la base. La création d'un objet *statement* s'effectue à partir d'une instance de la classe *Connection* comme suit :*Statementstmt = con.createStatement();*

Pour une requête de type interrogation(SELECT), la méthode à utiliser de la classe *Statement* est *executeQuery().* Pour des traitement de mise à jour, il faut utiliser la méthode *executeUpdate().* Lors de l'appel à la méthode d'exécution, il est nécessaire de lui fournir en paramètre la requête SQL sous forme de chaine.

Le résultat d'une requête d'interrogation est renvoyé dans un objet de la classe *ResulSet* par la méthode *executeQuery()* ;

Exemple :
ResultSetrs = stmt.executeQuery("SELECT * FROM employe");

Tableau 2.05: *Exemple de résultat d'une requête d'interrogation*

La méthode *executeUpdate()* retourne le nombre d'enregistrements qui ont été mis à jour

Exemple:

...
//insertion d'un enregistrement dans la table client
requete = "INSERT INTO client VALUES (3,'client 3','prenom 3')";
try {
Statement stmt = con.createStatement();
intnbMaj = stmt.executeUpdate(requete);
affiche("nb mise a jour = "+nbMaj);
} catch (SQLException e) {
e.printStackTrace();
}
...

Tableau 2.06: *Exemple d'insertion d'un enregistrement dans une table*

Lorsque la méthode *executeUpdate()* est utilisée pour exécuter un traitement de type DDL (Data Définition Langage : définition de données) comme la création d'un table, elle retourne 0. Si la

méthode retourne 0, cela signifie deux choses : le traitement de mise à jour n'a affecté aucun enregistrement ou le traitement concernait un traitement de type DDL.

Si l'on utilise *executeQuery()* pour exécuter une requête SQL ne contenant pas d'ordre SELECTalors une exception de type *SQLException* est levée.

Exemple:

```
...
requete = "INSERT INTO client VALUES (4,'client 4','prenom 4')";
try {
Statement stmt = con.createStatement();
ResultSetrésultats = stmt.executeQuery(requete);
} catch (SQLException e) {
e.printStackTrace();
}
```

Tableau 2.07: *Exemple d'utilisation SQLException*

2.6.3 La classe ResultSet

C'est une classe qui représente une abstraction d'une table qui se compose de plusieurs enregistrements constitués de colonnes qui contiennent les données.

Voici quelque méthode pour obtenir des données :

Méthode	Rôle
getInt(int)	Retourne le numéro de la colonne dont le numéro est passé en paramètre sous forme d'entier
getInt(String)	Retourne le numéro de la colonne dont le nom est passé en paramètre sous forme d'entier.
getFloat(int)	Retourne le contenu de la colonne dont le numéro est passé en paramètre sous forme de nombre flottant.
getDate(int)	Retourne le contenu de la colonne dont le numéro est passé en paramètre sous forme de date.
Next()	Se déplace sur le prochain enregistrement : retourne false si la fin est atteinte
Close()	Ferme le ResultSet
getMetaData()	Retourne un objet ResultSetMetaData associé au ResultSet

Tableau 2.08: *Méthode pour récupérer des données*

La méthode *getMetaData()* retourne un objet de la classe *ResultSetMetaData* qui permet d'obtenir des informations sur le résultat de la requête. Ainsi, le nombre de colonnes peut être obtenu grâce à la méthode *getColumnCount()* de cet objet.

Exemple:

ResultSetMetaDatarsmd;
rsmd = results.getMetaData();
 	
nbCols = rsmd.getColumnCount();

Tableau 2.09: *Exemple d'utilisation de la méthode getColumnCount*

2.7 L'obtention d'informations sur la base de données

2.7.1 La classe ResultSetMetaData

La méthode *getMetaData()* d'un objet *ResultSet* retourne un objet de type *ResultSetMetaData.* Cet objet permet de connaitre le nombre, le nom et le type des colonnes.

Méthode	Rôle
Int getColumnCount()	Retourne le nombre de colonnes du ResultSet
String getColumnName(int)	Retourne le nom de la colonne dont le numéro est donné
String getColumnLabel(int)	Retourne le libellé de la colonne donnée
BooleanisCurrency(int)	Retourne « true » si la colonne contient un nombre au format monétaire
BooleanisReadOnly(int)	Retourne « true » si la colonne est en lecture seule
BooleanisAutoIncrement(int)	Retourne « true » si la colonne est auto incrémentée
Int getColumntype(int)	Retourne le type de donnée SQL de la colonne

Tableau 2.10: *Méthode et rôle de La classe ResultSetMetaData*

2.7.2 La classe DatabaseMetaData

Un objet de la classe DatabaseMetaData permet d'obtenir des informations sur la base de données dans son ensemble : nom des tables, nom des colonnes dans une table, méthodes SQL supportées.

Méthode	Rôle
ResulSetgetcatalogs()	Retourne la liste du catalogue d'informations avec le pont JDBC-ODBC, on obtient la liste des bases de données enregistrées dans ODBC
ResultSetgetTables(catalog,schema,tableNames, columnNames)	Retourne une description de toutes les tables correspondant au TableNames donné et à toutes les colonnes correspondantes à columnNames.
ResultSetgetcolumns(catalog, shema, tableNames,columnNames)	Retourne une description de toutes les colonnes correspondantes au TableNames donné et à toutes les colonnes correspondantes à columnNames.
String getURL()	Retourne l'URL de la base à laquelle on est connecté
String getDriverName()	Retourne le nom du driver utilisé

Tableau 2.11: *Méthode et rôle de La classe DatabaseMetaData*

2.8 L'utilisation d'un objet PreparedStatement

L'interface *PreparedStatement* définit les méthodes pour un objet qui va encapsuler une requête précompilée. Ce type de requête est particulièrement adapté pour une exécution répétée d'une même requête avec des paramètres différents. Cette interface hérite de l'interface *Statement*. Lors de l'utilisation d'un objet de type *PreparedStatement,* la requête est envoyée au moteur de la base de données pour que celui-ci prépare d'abordson exécution

Un objet qui implémente l'interface *PreparedStatement* est obtenu en utilisant la méthode *preparedStatement()* d'un objet de type *Connection*. Cette méthode attend en paramètre une chaine de caractères contenant la requête SQL. Dans cette chaine, chaque paramètre est représenté par un caractère ?. [2][7]

Et pour exécuter la requête, l'interface *PreparedStatement* propose deux méthodes :

- *executeQuery()* : cette méthode permet d'exécuter une requête de type interrogation et renvoie un objet de type *ResultSet* qui contient les données issues de l'exécution de la requête.
- *executeUpdate()* : cette méthode permet d'exécuter une requête de type mise à jour et renvoie un entier qui contient le nombre d'occurrence impactées par la mise à jour.

2.9 L'utilisation des transactions

Une transaction permet de ne valider un ensemble de traitements sur la base de données que si tous les traitements de cet ensemble se sont tous effectués correctement. Par exemple une opération bancaire de transfert de fond d'un compte vers un autre oblige à la réalisation de l'opération de « débit » sur un compte et de l'opération de « crédit » sur l'autre compte. La réalisation d'une seule de ces opérations laisserait les données de la base dans un état inconsistant.

Une transaction est un mécanisme qui permet donc de s'assurer que toutes les opérations qui la compose seront réellement effectuées. Une transaction est gérée à partir de l'objet *Connection*. Par défaut, une connexion est en mode auto-commit. Dans ce mode, chaque opération est validée unitairement pour former la transaction. Pour pouvoir rassembler plusieurs traitements dans une transaction, il faut tout d'abord désactiver le mode auto-commit. La classe *Connection* possède la méthode *setAutoCommit()* qui attend un booléen qui précise le mode de fonctionnement. Par exemple *connection.setAutoCommit(false)* ;

Une fois le mode auto-commit désactivé, un appel à la méthode *commit()* de la classe *Connection* permet de valider la transaction courante. L'appel à cette méthode valide la transaction courante et crée implicitement une nouvelle transaction. Et si une anomalie intervient durant la transaction, il est possible de faire un retour en arrière pour revenir à la situation de la base de données au début de la transaction en appelant la méthode *rollback()* de la classe *Connection.*

2.10 Le traitement des erreurs JDBC

JDBC permet de connaitre les avertissements et les exceptions générées par la base de données lors de l'exécution de requête. La classe *SQLException* représente les erreurs émises par la base de données. Elle contient trois attributs qui permettent de préciser l'erreur :

2.10.1 Le message

Le *message* contient la description de l'erreur.

2.10.2 SQLState

SQLState c'est un code défini par les normes X/Open et SQL99

2.10.3 ErrorCode

ErrorCode est le code d'erreur du fournisseur du pilote

2.11 Amélioration des performances avec JDBC

Les opérations d'accès à la base de données sont généralement nombreuses et source des nombreux ralentissements dans une application, il est donc nécessaire de les prendre en compte dès le début d'un projet.

Voici quelques recommandations de base, regroupées par catégories, qui permettent d'améliorer les performances.

2.11.1 Le choix du pilote JDBC à utiliser

La qualité du pilote JDBC est importante notamment en termes de rapidité, type de pilote, version de JDBC supportée.

Le type du pilote influe grandement sur les performances :

- Pont JDBC/ODBC (type 1) : les pilotes de ce type sont à éviter car les différentes couches mises en œuvre (JDBC, pilote JDBC, ODBC, pilote ODBC, base de données) dégradent les performances.
- Utilisation d'une API native (type 2) : les pilotes de ce type ont généralement des performances moyennes.
- JDBC, pilote JDBC, middleware, DB (type 3) : les pilotes de types 3 communiquent avec un middleware généralement sur le serveur, ils sont généralement plus performant que ceux de type 1 et 2.

- JDBC, pilote JDBC, DB (type 4) : les pilotes de types 4 offre généralement les meilleures performances car ils sont écrits en Java et communiquent directement avec la base de données.

Il est donc préférable d'utiliser les pilotes de type 3 ou 4.

2.11.2 La mise en œuvre de best practices

Le terme « best practices » ou « bonne pratique » désigne, un ensemble de comportements qui font consensus et qui sont considérés comme indispensables par la plupart des professionnels en programmation. Plusieurs best practices sont communément mises en œuvre lors de l'utilisation du JDBC :

- Fermer les ressources inutilisées dès que possible (*Connection, Statement, ResultSet*)
- Limiter le nombre de données retournées par une requête SQL uniquement à celles utilisées.
- Toujours assurer un traitement des « warnings »et des « exceptions »de Java.

2.11.3 L'utilisation des connexions et des Statements

Il est préférable de maintenir une connexion ouverte et la réutiliser plutôt que créer une nouvelle connexion et la fermer à chaque opération sur la base de données. C'est ce que permettent les pools de connexions voir 2.11.4.

Si les accès sont en lecture seule, il est préférable d'utiliser la méthode *setReadOnly()* de l'objet *Connection* en lui passant le paramètre *true* pour permettre au pilote de faire des optimisations.

Il est possible de paramétrer la quantité de données reçues de la base de données en utilisant les méthodes *setMaxRows(),setMaxFieldSize()* et *SetFetchSize()* de l'interface *Statement*.

La méthode *nativeSQL()* de la classe *connection* permet d'obtenir la requête SQL native qui sera envoyée par le pilote à la base de données. [2]

2.11.4 L'utilisation d'un pool de connexions

La connexion vers une base de données est coûteuse en temps et en ressources. Le rôle d'un pool de connexion est de maintenir un certain nombre de connexions ouvertes à disposition de l'application dans un cache et de les proposer aux besoins.

Un pool peut être soit fourni par l'environnement d'exécution, soit être fourni par un tiers, soit être développé de toute pièce.

L'utilisation d'un pool de connexion est sûrement l'action la plus efficace pour des applications qui utilisent les accès à la base de données de façon importante.

Il est important de configurer correctement le pool de connexions utilisé, notamment la taille du pool, pour limiter la création et la destruction des connexions.

Un pool de connexions peut fonctionner selon deux modes principaux :

- Taille fixe : l'obtention d'une connexion, alors que toutes celles du pool sont en cours d'utilisation, implique l'attente de la libération d'une des connexions.
- Taille variable : le pool possède une taille minimale et maximale, avec une possibilité d'extension en cas de surcharge de travail.

2.11.5 La configuration et l'utilisation des ResultSets en fonction des besoins

Une bonne configuration et utilisation des objets de type *ResultSet* peuvent améliorer les performances. Il faut éviter d'utiliser la méthode *getObject()* mais utiliser la méthode getXXX() adaptée au type d'une donnée pour extraire sa valeur.

2.11.6 L'utilisation des PreparedStatement

Il est intéressant d'utiliser les *PreparedStatement* notamment pour les requêtes qui sont exécutées plusieurs fois avec les mêmes paramètres ou des paramètres différents (les valeurs des données fournies à la requête peuvent être paramétrées).

Une même requête exécutée avec des paramètres différents nécessite certains traitements identiques par la base de données : une partie de ces traitements sont réalisés une et une seule fois lors de la première utilisation d'un *PreparedStatement* par une connexion. Les appels suivants

avec la même connexion sont plus rapides puisque ces traitements ne sont pas refaits.

2.11.7 La maximisation des traitements effectués par la base de données

Par exemple pour obtenir un nombre d'occurrences, il est préférable d'effectuer une requête SQL contenant un *count*(*) plutôt que de parcourir un *ResultSet* avec un compteur incrémenté à chaque itération.

Il est possible d'utiliser les procédures stockées[ou stored procedure en anglais] pour les traitements lourds ou complexes sur la base de données plutôt que d'effectuer plusieurs appels à la base de données pour réaliser les mêmes traitements côté Java. Une procédure stockée est un ensemble d'instruction SQL précompilées, stockées dans une base de données et exécutées sur demande par le SGBD qui manipule la base de données. Les performances sont accrues car les traitements sont réalisés par la base de données ce qui évite notamment des échanges réseaux.

Attention ceci n'est vrai que pour des traitements complexes car une simple requête SQL s'exécutera plus rapidement que d'appeler une procédure stockée qui contient simplement la requête.

Il est préférable d'utiliser les marqueurs de paramètres dans les requêtes des objets de type *Statement* plutôt que de les passer en dur dans la requête.

2.11.8 Les optimisations sur la base de données

Les optimisations côté Java sont importantes mais il est aussi nécessaire de procéder à des optimisations côté base de données, généralement réalisées par un DBA dans des structures de taille moyenne ou importante.

Les quelques optimisations fournies ci-dessous sont assez généralistes : elles ne dispensent pas d'effectuer des optimisations spécifiques à la base de données utilisées.

- Il faut mettre en place les index utiles : l'ajout d'un index peut dramatiquement améliorer les performances mais trop d'index nuit car la base de données doit les maintenir à jour.
- Les bases des données fournissent des outils pour afficher le plan d'exécution d'une requête ou d'une procédure stockée pour faciliter leur optimisation (ajout d'index, modification des clauses de la requête,…).

- Si le pilote JDBC le permet, il peut être intéressant d'ajuster la taille des paquets échangés avec la base de données.
- Utiliser le type de données approprié aux données stockées en fonction des besoins par exemple : représenter une date avec un type DateTime (plus de sécurité dans l'utilisation de la donnée) ou varchar (traitement plus rapide).
- Il est préférable de stocker les chaînes de caractères en Unicode (encodage en UTF-8 par exemple) dans la base de données pour éviter les conversions. Ceci a cependant un impact important sur la taille de la base de données.[2]

2.11.9 L'utilisation d'un cache

L'utilisation d'un cache pour stocker les données peut éviter des accès à la base de données. Ceci est particulièrement adapté pour des données lues de façon répétitives ou dont les valeurs évoluent très peu ou pas du tout (données en lecture seule, données de références,…)

Mais il faut faire attention à la durée de vie des objets dans le cache afin d'éviter des problèmes de rafraichissement de données. Et aussi il ne faut pas mettre en cache les objets de types ResultSet : il faut le parcourir, stocker les données dans des objets du domaine et mettre ces objets dans la cache.

2.12 Conclusion

Pour accéder à une base de données à partir d'une application java il faut prendre en compte les améliorations des performances comme le choix de pilote, la mise en œuvre de best practices, l'utilisation d'un pool de connexions, la configuration et l'utilisation de la classe *ResultSet* et *PreparedStatement* en fonction des besoins, et les optimisations sur la base de données et enfin l'utilisation d'un cache.

CHAPITRE 3
ARCHITECTURE CLIENT-SERVEUR

3.1 Introduction

Ces dernière années ont vu une évolution majeure des systèmes d'information. Du passage d'une architecture centralisée à travers de grosses machines (Mainframe) vers une architecture distribuée basée sur l'utilisation de serveur et de postes clients grâce à l'utilisation des PC (Personal Computeur) et des réseaux.

Cette évolution est due essentiellement à la baisse des prix des outils informatiques et le développement des réseaux.

Le logiciel **LGLV** est une application utilisant l'architecture 2 tiers [voir 3.5.1]. Alors il est primordial de définir ce que l'on entend par Architecture client-serveur.

3.2 Définition

L'architecture client-serveur est un modèle de fonctionnement logiciel qui peut se réaliser sur tout type d'architecture matérielle (petites ou grosses machines), à partir du moment où ces architectures peuvent être interconnectées.

On parle de fonctionnement logiciel dans la mesure où cette architecture est basée sur l'utilisation de deux types d'application : un logiciel « serveur » et un logiciel « client » s'exécutant normalement sur deux machines différentes. L'élément important dans cette architecture est l'utilisation de mécanismes de communication entre les deux applications. Le dialogue entre les applications peut se résumer par ces deux phrases :

- le client demande un service au serveur.
- le serveur réalise ce service et renvoie le résultat au client

3.3 les principes généraux

Voici quelques principes d'un système Client-Serveur.

- Service :

Le serveur est fournisseur de services, et le client est le consommateur.

- Protocole :

La demande vient toujours du côté client, et le serveur attend les requêtes des clients.

- Partage des ressources :

Le serveur traite plusieurs clients en même temps et contrôle leurs accès aux ressources.

- Localisation :

Le logiciel client-serveur masque aux clients la localisation du serveur.

- Hétérogénéité.

Le logiciel client-serveur est indépendant des plate-formes matérielles et logicielles.

- Redimensionnement.

Il est possible d'ajouter et de retirer des stations clientes. Il est possible de faire évoluer les serveurs.

- Intégrité.

Les données du serveur sont gérées sur le serveur de façon centralisée. Les clients restent individuels et indépendants.

- Souplesse et adaptabilité.

On peut modifier le module serveur sans toucher au module client. La réciproque est vraie. Si une station est remplacée par un modèle plus récent, on modifie le module client (en améliorant l'interface, par exemple) sans modifier le module serveur.

Dans une architecture client-serveur, une application est constituée de trois grande parties ;

- L'interface utilisateur
- La logique des traitements
- La gestion des données

L'interface utilisateur est l'application qui se trouve en face du client.

La logique des traitements sont les requêtes.

La gestion des données est réalisée par le serveur de base de données qui s'occupe de la gestion complète des manipulations de données.

3.4 Les différents modèles de client-serveur

La différence entre les modèles client-serveur se situe essentiellement sur les taches assurées par le serveur.

3.4.1 Le client-serveur de donnée.

Dans ce cas, le serveur assure des tâches de gestion, stockage et de traitement de données. C'est le cas le plus connu de client-serveur et qui est utilisé par tous SGBD :

- La base de données avec tous ses outils est installée sur un poste serveur.
- Un logiciel d'accès est installé sur les postes client permettant d'accéder à la base de données du serveur.
- Tous les traitements sur les données sont effectués sur le serveur qui renvoie les informations demandées par le client.

3.4.2 Client-serveur de présentation

Dans ce cas la présentation des pages affichées par le client est intégralement prise en charge par le serveur. Cette organisation présente l'inconvénient de générer un fort trafic réseau.

3.4.3 Le client-serveur de traitement

Dans ce cas, le serveur effectue des traitements à la demande du client. Il peut s'agir de traitement particulier sur des données, de vérification de formulaires de saisie, de traitements d'alarmes …

Ces traitements peuvent être réalisés par des programmes installé sur des serveurs mais également intégrés dans des bases de données (triggers, procédures stockées), dans ce cas, les parties« donnée » et « traitement » sont intégrés.

3.5 Les différentes architectures

3.5.1 L'architecture 2-tiers

Dans une architecture 2-tiers, encore appelée client-serveur de données, le poste client se contente de déléguer la gestion des données à un service spécialisé. Le cas typique de cette architecture est une application de gestion fonctionnant sous Windows ou Linux et exploitant un SGBD centralisé.

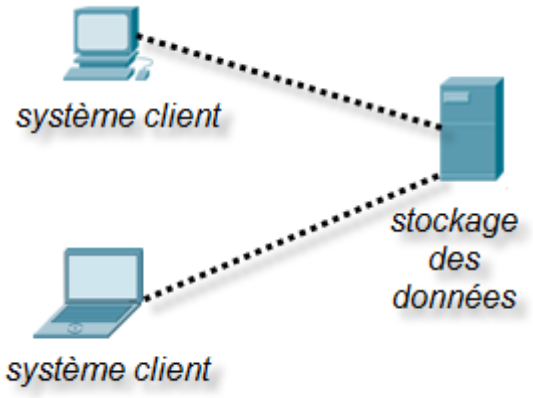

Figure 3.01 : *L'architecture client/serveur deux-tiers*

Ce type d'application permet de tirer parti de la puissance des ordinateurs déployés en réseau pour fournir à l'utilisateur une interface riche, tout en garantissant la cohérence des données, qui restent gérées de façon centralisée.

La gestion des données est prise en charge par un SGBD centralisé, s'exécutant le plus souvent sur un serveur dédié. Ce dernier est interrogé en utilisant un langage de requête qui, plus

souvent, est SQL. Le dialogue entre client et serveur se résume donc à l'envoi de requêtes et au retour des données correspondant aux requêtes.

Cet échange de messages transite à travers le réseau reliant le client et le serveur. Il met en œuvre des mécanismes relativement complexes qui sont, en général, pris en charge par un middleware.

L'expérience a démontré qu'il était coûteux et contraignant de vouloir faire porter l'ensemble des traitements applicatifs par le poste client. On en arrive aujourd'hui à ce que l'on appelle le client lourd, avec un certain nombre d'inconvénients :

- On ne peut pas soulager la charge du poste client, qui supporte la grande majorité des traitements applicatifs,
- Le poste client est fortement sollicité, il devient de plus en plus complexe et doit être mis à jour régulièrement pour répondre aux besoins des utilisateurs,
- Les applications se prêtent assez mal aux fortes montées en charge car il est difficile de modifier l'architecture initiale,
- La relation étroite qui existe entre le programme client et l'organisation de la partie serveur complique les évolutions de cette dernière.

Malgré tout, l'architecture 2-tiers présente de nombreux avantages qui lui permettent de présenter un bilan globalement positif :

- Elle permet l'utilisation d'une interface utilisateur riche,
- Elle a permis l'appropriation des applications par l'utilisateur,
- Elle a introduit la notion d'interopérabilité.

Les solutions 2-tiers ont tout de même leur place dans le développement d'applications. Les applications simple et qui ne demandent pas trop de maintenance sont de parfaites candidates à une application 2-tiers. Voici la liste des questions importantes qu'il faut se poser avant d'adopter une conception 2-tiers. Si la plupart des réponses à la question est « Oui », alors il est mieux d'utiliser l'architecture 2-tiers si non l'architecture 3-tiers est l'idéal.

- Pour votre application, sa date de mise sur le marché est-elle plus importante que son architecture ?
- Votre application utilise-t-elle une seule base de données ?

- Votre moteur de base de données se trouve-t-il sur un seul hôte ?
- Votre base de données va-t-elle conserver plus ou moins la même taille ?
- Votre base d'utilisateurs va-t-elle conserver plus ou moins la même taille ?
- Les contraintes sont-elle fixées sans pouvoir changer, ou très peu ?
- Vous attendez-vous à une maintenance minimale de votre application ?

Pour résoudre les limitations du client-serveur 2-tiers tout en conservant ses avantages, on a cherché une architecture plus évoluée, facilitant les forts déploiements à moindre coût. La réponse est apportée par les architectures distribuées.

3.5.2 L'architecture 3- tiers

Les limites de l'architecture 2-tiers proviennent en grande partie de la nature du client utilisé:

- le frontal est complexe et non standard (même s'il s'agit presque toujours d'un PC sous Windows),
- le middleware entre client et serveur n'est pas standard (dépend de la plate-forme, du SGBD …).

La solution résiderait donc dans l'utilisation d'un poste client simple communicant avec le serveur par le biais d'un protocole standard.

Dans ce but, l'architecture 3-tiers applique les principes suivants **:**

- Les données sont toujours gérées de façon centralisée,
- La présentation est toujours prise en charge par le poste client,
- La logique applicative est prise en charge par un serveur intermédiaire.

Cette architecture 3-tiers, également appelée client-serveur de deuxième génération ou client-serveur distribué sépare l'application en 3 niveaux de services distincts, conformes au principe précédent :

- Premier niveau : l'affichage et les traitements locaux (contrôles de saisie, mise en forme de données...) sont pris en charge par le poste client.
- Deuxième niveau : les traitements applicatifs globaux sont pris en charge par le service applicatif.

- Troisième niveau : les services de base de données sont pris en charge par un SGBD.

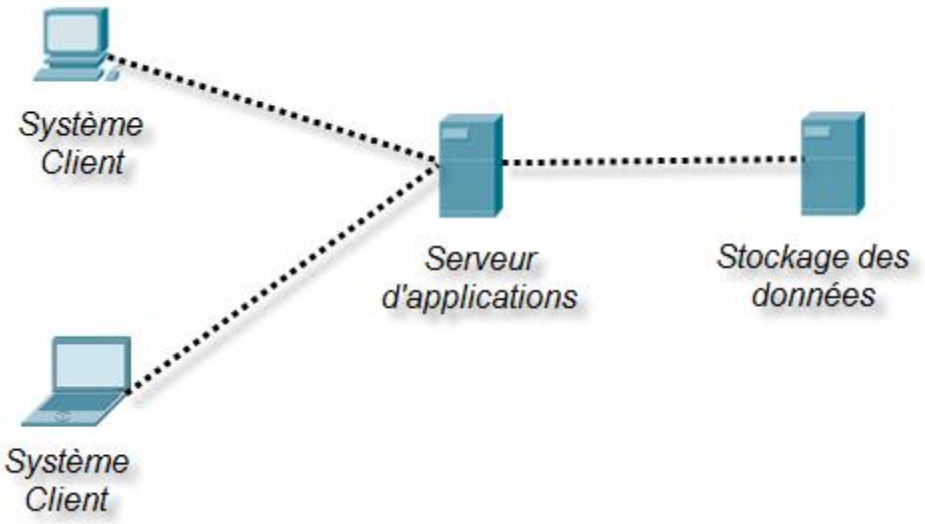

Figure 3.02 : *Une architecture 3-tiers*

Tous ces niveaux étant indépendants, ils peuvent être implantés sur des machines différentes, de ce fait :

- Le poste client ne supporte plus l'ensemble des traitements, il est moins sollicité et peut être moins évolué, donc moins coûteux.
- Les ressources présentes sur le réseau sont mieux exploitées, puisque les traitements applicatifs peuvent être partagés ou regroupés (le serveur d'application peut s'exécuter sur la même machine que le SGBD),
- La fiabilité et les performances de certains traitements se trouvent améliorées par leur centralisation.
- Il est relativement simple de faire face à une forte montée en charge, en renforçant le service applicatif.

Dans l'architecture 3- tiers, le poste client est communément appelé client léger ou en anglais« Thin Client », par opposition au client lourd des architectures 2-tiers. Il ne prend en charge que la présentation de l'application avec, éventuellement, une partie de logique applicative permettant une vérification immédiate de la saisie et la mise en forme des données.

Le serveur de traitement constitue la pierre angulaire de l'architecture et se trouve souvent fortement sollicité. Dans ce type d'architecture, il est difficile de répartir la charge entre client et serveur. On se retrouve confronté aux épineux problèmes de dimensionnement serveur et de gestion de la montée en charge rappelant l'époque des mainframes.

De plus, les solutions mises en œuvre sont relativement complexes à maintenir et la gestion des sessions est compliquée.

Les contraintes semblent inversées par rapport à celles rencontrées avec les architectures 2-tiers : le client est soulagé, mais le serveur est fortement sollicité.

Même si l'architecture 3-tiers présente certains avantages importants, elle n'est pas sans inconvénients. Le plus important est le niveau de complexité qu'elle ajoute à un système. Le système est composé d'un ensemble de composants distincts plus importants, ils sont donc plus complexes à gérer. Il est aussi difficile de trouver des ingénieurs logiciels ayant des compétences en programmation 3-tiers et notamment en ce qui concerne la gestion des transactions et sécurité.[12]

3.5.3 L'architecture n-tiers

L'architecture n-tiers a été pensée pour pallier aux limitations des architectures 3-tiers et concevoir des applications puissantes et simples à maintenir. Ce type d'architecture permet de distribuer plus librement la logique applicative, ce qui facilite la répartition de la charge entre tous les niveaux.

Cette évolution des architectures 3-tiers met en œuvre une approche objet pour offrir une plus grande souplesse d'implémentation et faciliter la réutilisation des développements.

Théoriquement, ce type d'architecture supprime tous les inconvénients des architectures précédentes :

- Elle permet l'utilisation d'interfaces utilisateurs riches,
- Elle sépare nettement tous les niveaux de l'application,
- Elle offre de grandes capacités d'extension,
- Elle facilite la gestion des sessions.

L'appellation ``n-tiers' pourrait faire penser que cette architecture met en œuvre un nombre indéterminé de niveaux de service, alors que ces derniers sont au maximum trois (les trois niveaux d'une application informatique). En fait, l'architecture n-tiers qualifie la distribution d'application entre de multiples services et non la multiplication des niveaux de service.

Cette distribution est facilitée par l'utilisation de composants ``métier', spécialisés et indépendants, introduits par les concepts orientés objets (langages de programmation et middleware). Elle permet de tirer pleinement partie de la notion de composants métiers réutilisables.

Ces composants rendent un service générique, si possible et clairement identifié. Ils sont capables de communiquer entre eux et peuvent donc coopérer en étant implantés sur des machines distinctes.

La distribution des services applicatifs facilite aussi l'intégration de traitements existants dans les nouvelles applications. On peut ainsi envisager de connecter un programme de prise de commande existant sur le site central de l'entreprise à une application distribuée en utilisant un middleware adapté.

Ces nouveaux concepts sont basés sur la programmation objet ainsi que sur des communications standards entre application. Ainsi est né le concept de Middleware objet.

3.6 Conclusion

Ces différentes architectures présentent chacune ses atouts et aussi faiblesses.

En bref, le choix de l'architecture à utiliser dépend de plusieurs paramètres comme les besoins de l'entreprise et aussi le budget de conception de l'architecture réseau.

CHAPITRE 4
REALISATION DE L'APPLICATION

4.1 Introduction

Dans les chapitres précédents les études nous ont permis de comprendre le fonctionnement des bases de données ainsi que l'accès à la base à partir d'application java et le fonctionnement des architectures client-serveur.

Dans ce chapitre on va aborder en premier lieu la présentation et les fonctionnements des logiciels utilisés lors de la conception du projet, en second lieu on abordera les services offerts par le logiciel de gestion de location de voiture utilisant l'architecture client-serveur.

4.2 Les outils utilisés pendant la conception du logiciel

Pendant la réalisation du logiciel, plusieurs outils ont été utilisés comme JDK 6 pour le langage de programmation JAVA, NetBeans 7.2 comme IDE, Win'Design pour la conception du MCD et MySQL pour le Système de Gestion de Base de Données(SGBD) et Photoshop pour le traitement d'image. Ces différents types d'outil vont être détaillés par la suite.

4.2.1 JDK 6

JDK est l'acronyme Java Development Kit et le chiffre 6 désigne le numéro de version de JDK. C'est un environnement de développement de Sun permettant de produire du code Java.

4.2.2 EDI NetBeans

L'EDI NetBeans est un environnement de développement, gratuit, se concentrant principalement sur la simplification de développement d'application JAVA. Il fournit un support pour tous les types d'applications Java, depuis le client riche jusqu'aux applications d'entreprises multicouches, en passant par les applications pour les mobiles supportant Java.

L'EDI est lui-même écrit en Java, ce qui permet au programmeur de le faire tourner sur n'importe quel système d'exploitation pour lequel un JDK Java 2 Standard Edition (version 1.4.2, 5.0 ou plus) est disponible. Des installateurs sont généralement fournis pour les systèmes Microsoft Windows, Solaris, Linux, MacOX et même OpenVMS.

La tâche principale de l'EDI est de rendre le cycle d'édition, compilation, débogage plus agréable en intégrant des outils pour ces activités, par exemple l'EDI:

- Identifie les erreurs d'encodage presque immédiatement et les indique dans l'éditeur de source
- aide à coder plus rapidement grâce aux fonctionnalités de remplissage automatique de code.
- fourni une aide de navigation visuelle, comme la fenêtre navigation et le « pliage de code », ainsi que de nombreux raccourcis clavier conçu spécialement pour les programmeurs Java.
- Affiche la documentation d'une classe pendant qu'on code dans l'éditeur de code.
- Affiche les erreurs de compilation qui apparait dans la fenêtre output avec des liens qui permet de se déplacer directement au bon endroit dans le code.
- Gere les noms des paquetages, et les références vers d'autres classes. Lorsqu'un utilisateur renomme ou déplace une classe, l'EDI identifie leurs places dans le code qui sont affectées par les changements et EDI génère les modifications appropriées à ces fichiers.

On peut également télécharger le profileur NetBeans pour ajouter au cycle traditionnel d'édition, de compilation, débogage, les tests de performance.

En plus de fournir un support pour l'encodage, l'EDI NetBeans est livré avec d'autres outils et bibliothèques, l'EDI intègre ces outils dans le « workflow » de l'EDI, mais on peut également les utiliser en ligne de commande.

Au déballage de l'EDI NetBeans, on peut trouver : Apache Ant, Tomcat, JUnit, Derby, GlassFish et les catalogues de solutions Java BluePrints. Mais on peut également télécharger d'autres outils.

4.2.3 Win'Design

Win'Design est un logiciel de conception de systèmes d'information. Il permet la génération des entités suivantes d'UML :

- Diagramme de classes
- Diagramme de cas d'utilisation
- Diagramme de séquence

- Diagramme de collaboration
- Diagramme d'activité
- Diagramme de déploiement et diagramme de composants

Dans notre cas Win'Design est utilisé à générer la requête SQL de la base de donnée après avoir entré le modèle conceptuel de donnée.

4.2.4 MySQL

MySQL est un Système de gestion de bases de données qui gère pour vous les fichiers constituant une base, prend en charge les fonctionnalités de protection et de sécurité et fournit un ensemble d'interfaces de programmation facilitant l'accès aux données.

MySQL consiste en un ensemble de programmes chargés de gérer une ou plusieurs bases de données, et fonctionnent selon une architecture client/serveur. Voir Figure 4.01

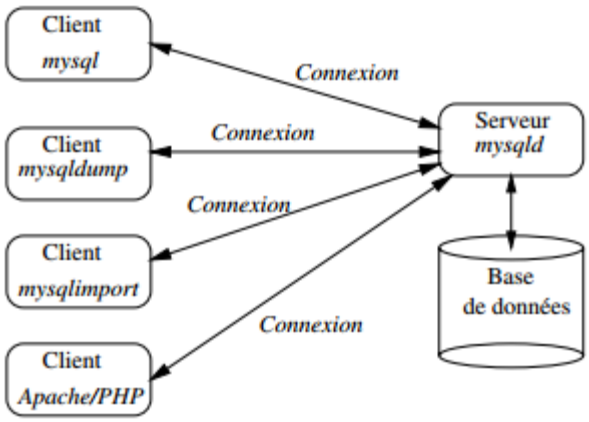

Figure 4.01 : *Serveur et client de MySQL*

Le serveur mysqld : le processus mysqld est le serveur de MySQL. Lui seul peut accéder aux fichiers stockant les données pour lire et écrire des informations.

MySQL fournit tout un ensemble de programmes, chargés de dialoguer avec mysqld par l'intermédiaire d'une connexion, pour accomplir un type de tâche particulier.[8]

4.2.5 Photoshop

Photoshop est un des produits de la Société Adobe destiné aux traitements d'image. A son début Photoshop n'était disponible que pour Macintosh mais Aujourd'hui il est principalement utilisé sur PC.

Dans notre cas, Photoshop servira au traitement de toutes les images de la présente application.

4.3 Méthodologie de construction d'une application :

Avant de réaliser une application, il convient de suivre des étapes bien précises. Le génie logiciel a été créé dans ce but. Il recommande de suivre les étapes suivantes :

- Expression des besoins
- Analyse
- Conception
- Implémentation
- Tests de vérification
- Validation
- Maintenance et évolution

4.3.1 Expression des besoins

C'est la description informelle des besoins exprimés par l'utilisateur.

- Besoins fonctionnels : ce sont les fonctionnalités attendues du système
- Besoins techniques
    - Moyen d'accès (local, distant,…)
    - Temps de réponse acceptable
    - Quantité d'informations à stocker

A la fin de cette étape, on devrait avoir un cahier des charges de l'application.

4.3.2 Analyse

En consultant le cahier des charges les services du système, ses contraintes et ses buts sont établis, puis nous les définissons pour qu'il soit compréhensible par les utilisateurs et les concepteurs.

A la fin de cette étape, la spécification de l'application est obtenue c'est-à-dire qu'on connait ce que l'application doit être et comment il peut être utilisé.

4.3.3 Conception

Elle consiste à apporter des solutions techniques aux descriptions définies lors de l'analyse :

On y définit les structures et les algorithmes. Cette phase sera validée lors des tests.

4.3.4 Implémentation

Cette étape servira à la matérialisation du logiciel en le réalisant sous la forme d'un ensemble de programmes ou d'unités de programmation.

4.3.5 Les tests de vérification

Ils permettent de réaliser des contrôles pour la qualité technique du système : il s'agit de relever les éventuels défauts de conception et de programmation (revue de code, tests des composants,…).

Il faut instaurer ces tests tout au long du cycle de développement et non à la fin pour éviter des reprises conséquentes du travail.

4.3.6 Validation

Tout au long de ces étapes, il doit y avoir des validations en collaboration avec l'utilisateur.

Une autre validation doit aussi être envisagée lors de l'achèvement du travail de développement, une fois que la qualité technique du système est démontrée. Cela permettra de garantir la logique et la complétude du système.

4.3.7 Maintenance et évolution

On considère deux sortes de maintenances

- Une maintenance corrective, qui consiste à traiter les « bugs ».
- Une maintenance évolutive, qui permet au système d'intégrer les nouveaux besoins ou des changements technologiques.

4.4 Modélisation de la partie statique du système d'information du logiciel

4.4.1 Règle de gestion

Les règles de gestion décrivent la nature des relations entre les entités d'un système d'information. L'ensemble de ces règles permet de définir un système correspondant à une problématique métier précisément adaptée aux besoins du client. Chaque règle correspond à une relation entre deux entités.

Ci-dessous un extrait de la règle de gestion pour notre application :

- Un client établi un contrat
- Un contrat affecte un véhicule
- Un véhicule subit une révision
- Un véhicule possède un modèle
- Chaque modèle a son tarif
- Chaque contrat a son tarif
- Un client loue un véhicule

4.4.2 Dictionnaire de données

Le dictionnaire de données est obtenu à partir d'une synthèse de la règle de gestion. Le logiciel Win'Design génère automatiquement les dictionnaires de données.

4.4.3 Les entités

L'entité est la représentation dans le système d'information d'un objet matériel ou immatériel ayant une existence propre et conforme aux choix de gestion de l'entreprise.

Dans le cas de ce logiciel, ci-dessous les différentes entités qui forment le MCD :

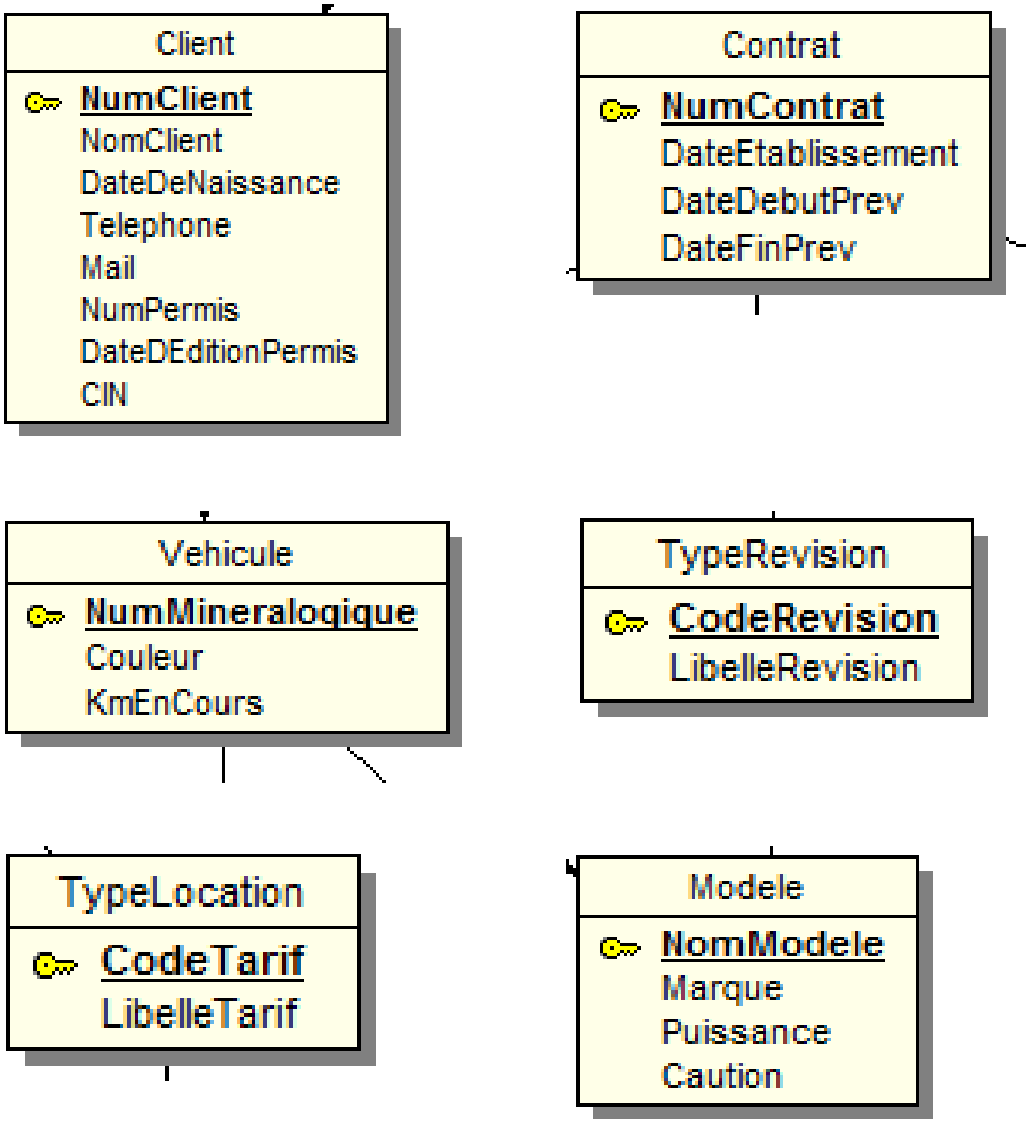

Figure 4.02 : *Les entités du logiciel*

4.4.4 Le modèle conceptuel de données

Le modèle conceptuel de données, représenté par la figure 4.02 est une représentation statique du système d'information de l'entreprise.

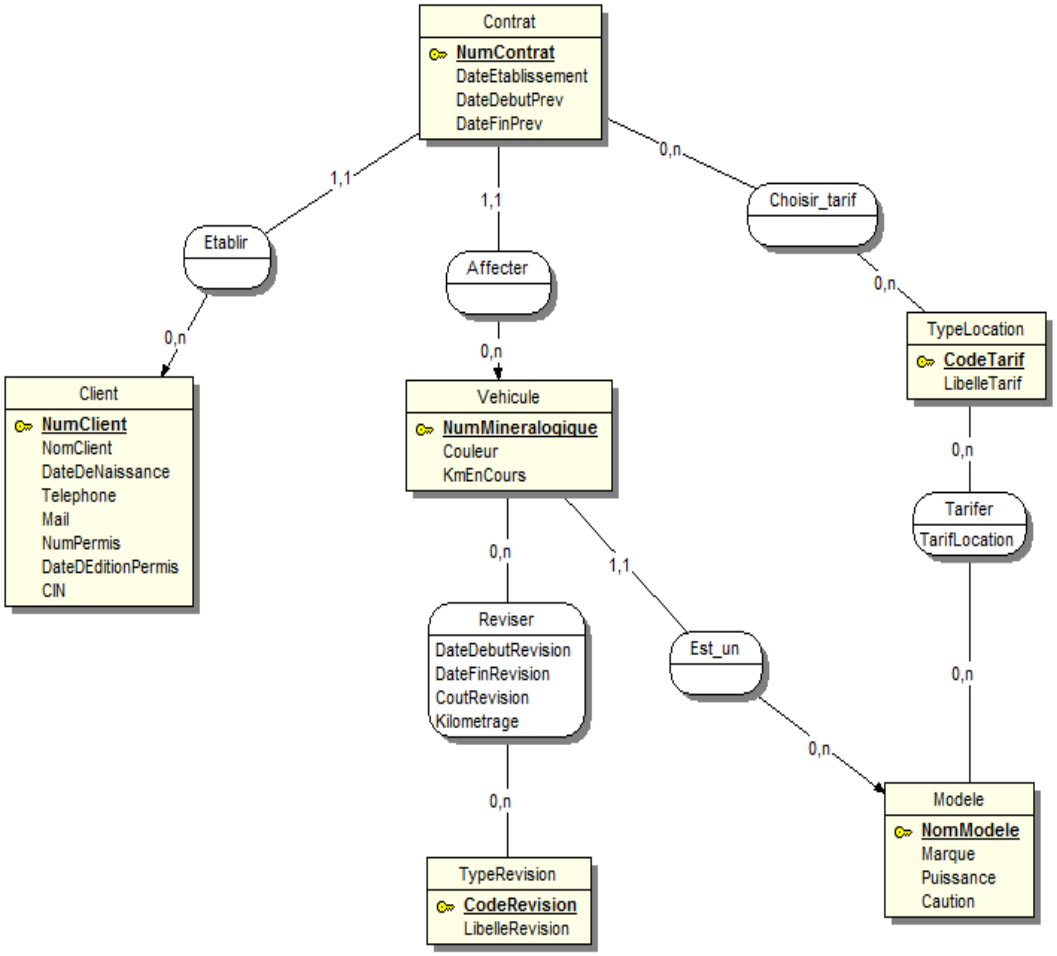

Figure 4.03 : *Modèle conceptuel de données du logiciel*

A partir de cette modèle, le logiciel Win'Design génère automatiquement le script pour la création de la base de données du logiciel.

4.5 Modélisation de la partie dynamique du système d'information du logiciel

4.5.1 Description du fonctionnement du logiciel

Le **LGLV** (logiciel de gestion de location de voiture) est une application de gestion travaillant en réseau et fonctionnant sous différents systèmes d'exploitation.

L'accès au logiciel requiert une Identification des utilisateurs avec l'usage de leur propre mot de passe. Le logiciel est très riche en fonctions, ces dernières sont regroupées dans un menu en haut de la fenêtre principale

4.5.2 Ordinogramme

La figure ci-dessous représente l'ordinogramme correspondant au système d'authentification de l'application LGLV.

T est le nombre de tentative

MDP est le mot de passe

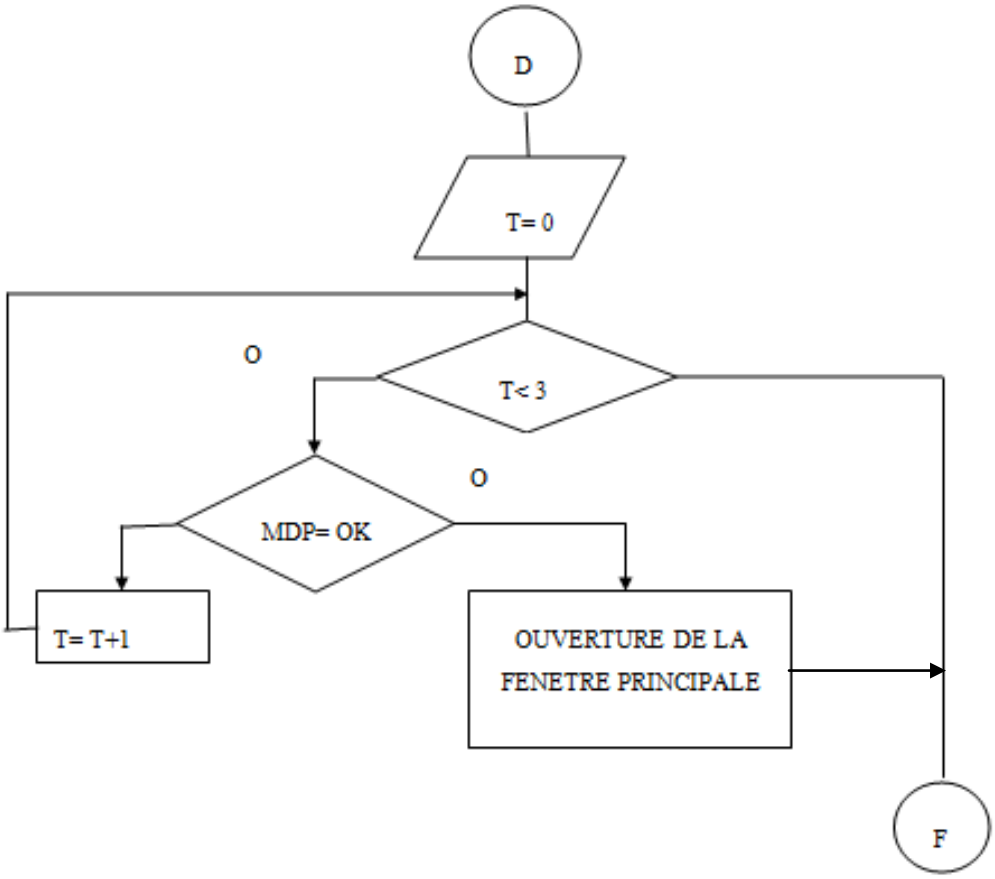

Figure 4.04 : *Ordinogramme de l'application*

4.5.3 Configuration réseau du logiciel

Voici un exemple de configuration réseau du LGLV

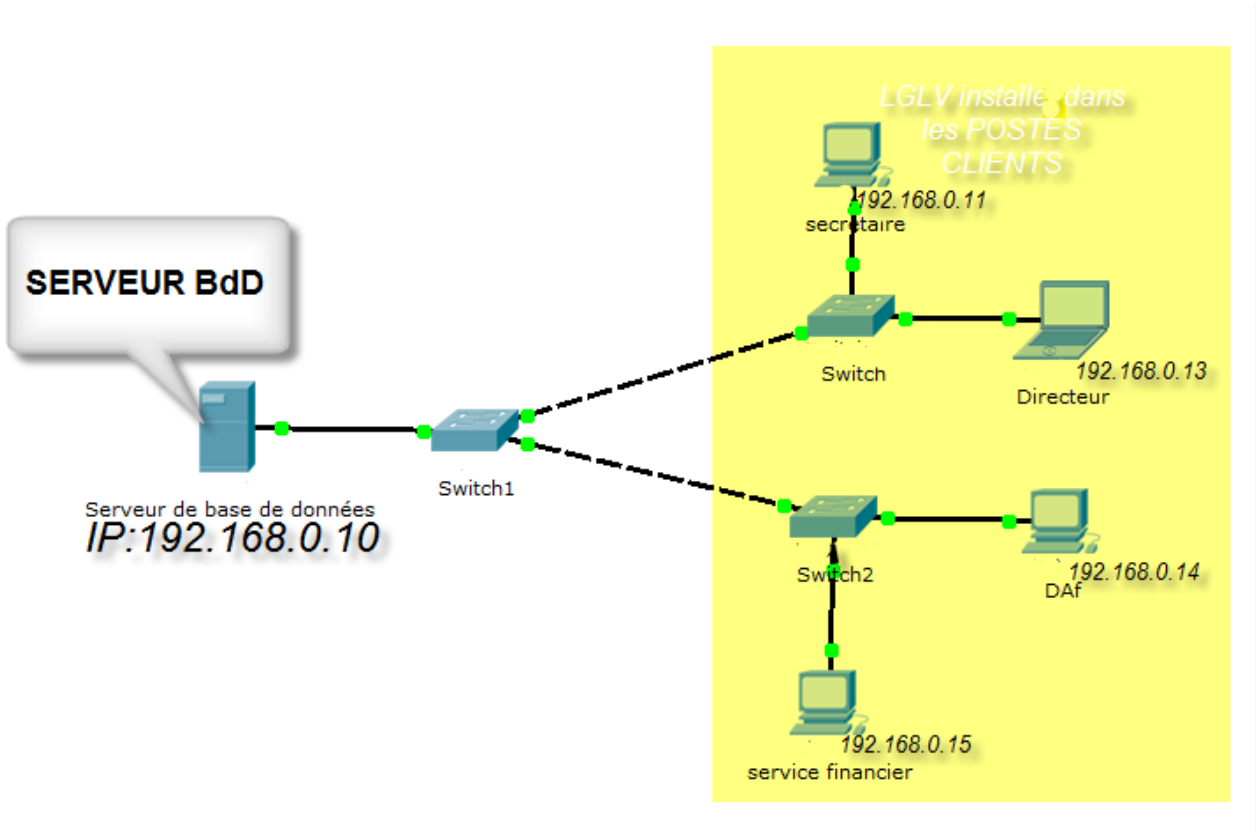

Figure 4.05 : *Configuration réseau du LGLV*

4.5.3.2 Description des équipements réseau utilisés

- Serveur de base de données :

  C'est dans cette machine qu'on a installé la base de données, ici on utilise WAMP serveur, l'adresse IP de la machine doit être fixe pour que les clients puissent envoyer leur demandes.

- Switch :

  Qui sert à raccorder les machines clientes au serveur.

- Machine Cliente :

  L'application LGLV est installée dans les machines clientes pour pouvoir accéder à la base de données installée au niveau du serveur. Il est préférable d'attribuer au serveur une adresse IP fixe pour assurer la sécurité de la base de données.

4.5.4 Les outils à mettre en place sur chaque Machine

4.5.4.1 Coté serveur

Il faut installer WampServeur dans la machine Serveur et lui importer la base de données contenant les données de notre application.

4.5.4.2 Coté clients

Pour pouvoir utiliser le LGLV, il faut installer la machine virtuelle de java JRE (Java Runtime Environment) et aussi l'application LGLV en format JAR (ou Java Archive) dans les machines clientes.

4.6 Exemple de fenêtre de l'application

4.6.1 La fenêtre d'Authentification

La fenêtre d'Authentification permet à chaque utilisateur de s'identifier et d'entrer son propre mot de passe.

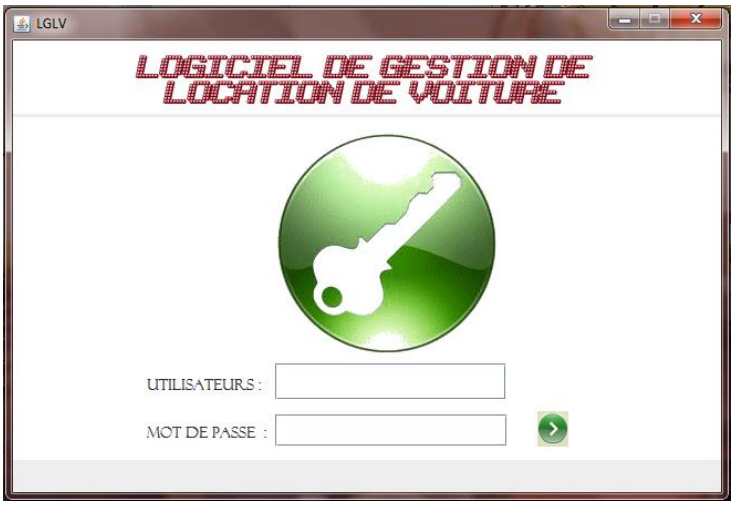

Figure 4.06 : *Fenêtre d'identification*

Apres validation de mot de passe l'utilisateur peut accéder à son compte et jouir des fonctions offertes par le logiciel.

Par contre si le nom d'utilisateur ou le mot de passe est incorrect le système affiche un message d'erreur (Figure 4.06). L'utilisateur a droit de faire 3 erreurs avant la fermeture de la fenêtre d'authentification.

Figure 4.07 : *Message d'erreur*

4.6.2 Fenêtre principale

Cette fenêtre se divise en 4 parties :

- La barre de menus qui contient les différentes options du logiciel comme : le tableau de bord, véhicule, Client, Financière, Option, et Aide
- La barre de logo, là où l'entreprise met son propre logo
- La barre de raccourci
- La Zone d'Affichage

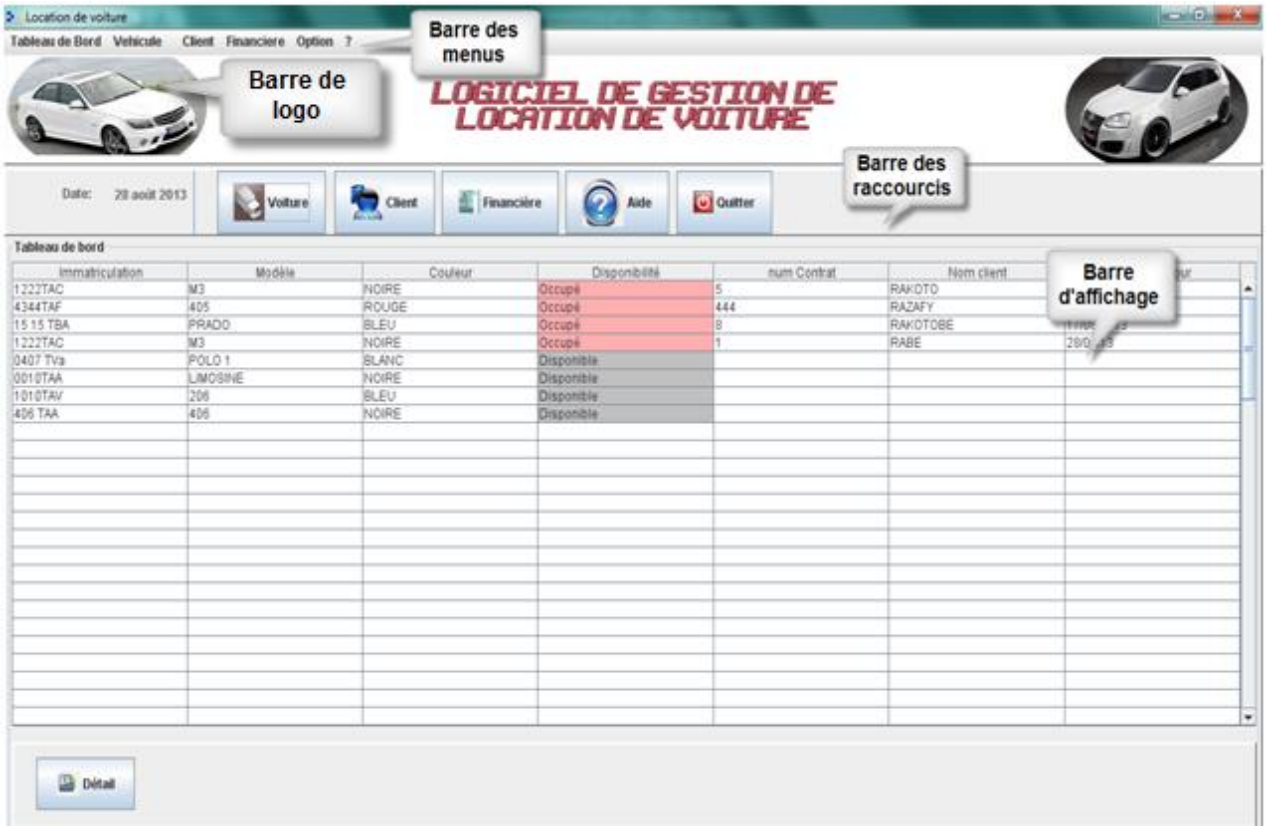

Figure 4.08 : *Fenêtre principale*

4.6.3 Quelques options du logiciel

Le logiciel de gestion de location de voiture offre plusieurs options comme l'affichage des voitures libres ou occupées avec le nom et le numéro de contrat du locataire et la date de retour prévue pour la voiture (figure 4.08).

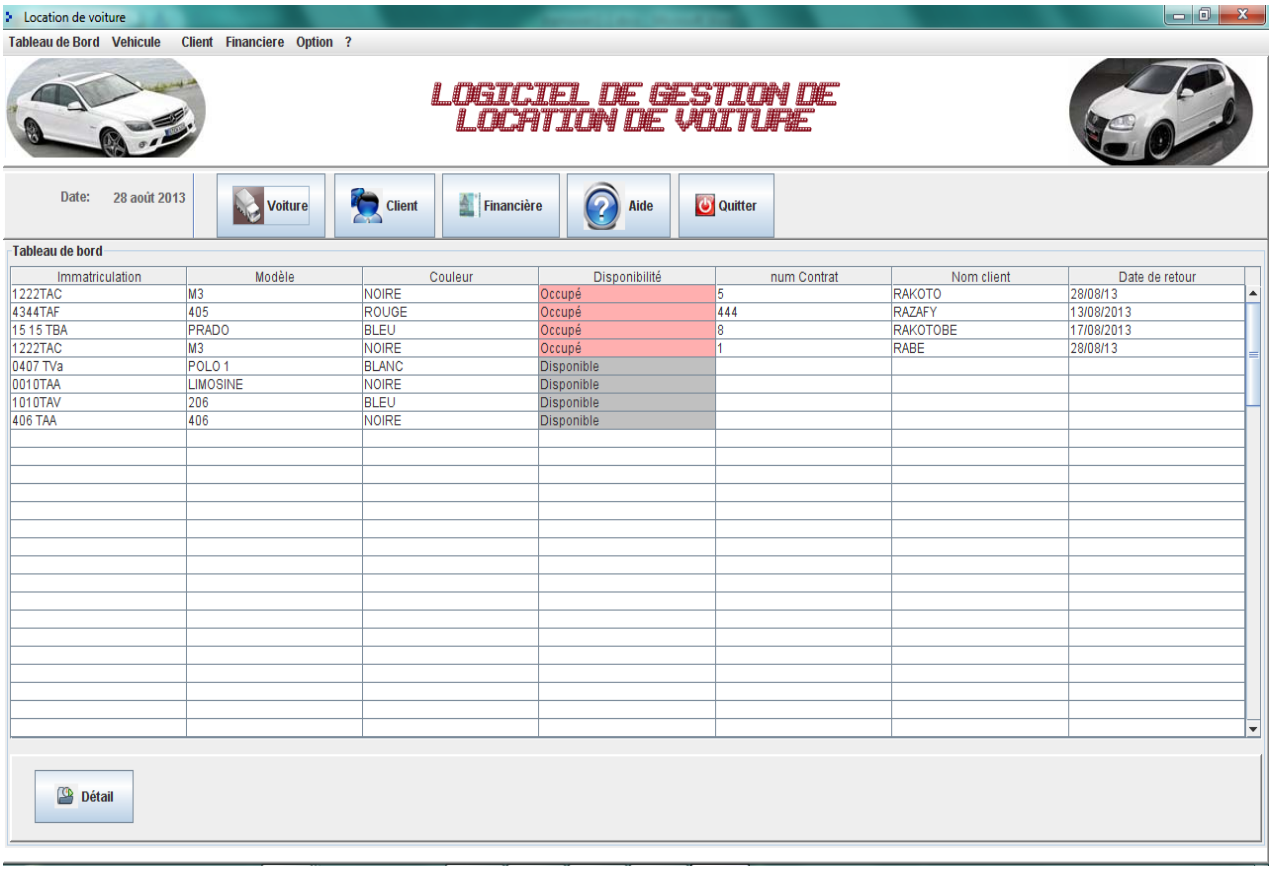

Figure 4.09 : *Affichage des listes des voitures occupées*

En cliquant sur le bouton Détail l'utilisateur peut voir les détails sur la voiture sélectionnée et aussi de préparer le contrat de location de la voiture sélectionnée en choisissant un tarif parmi les trois proposés.

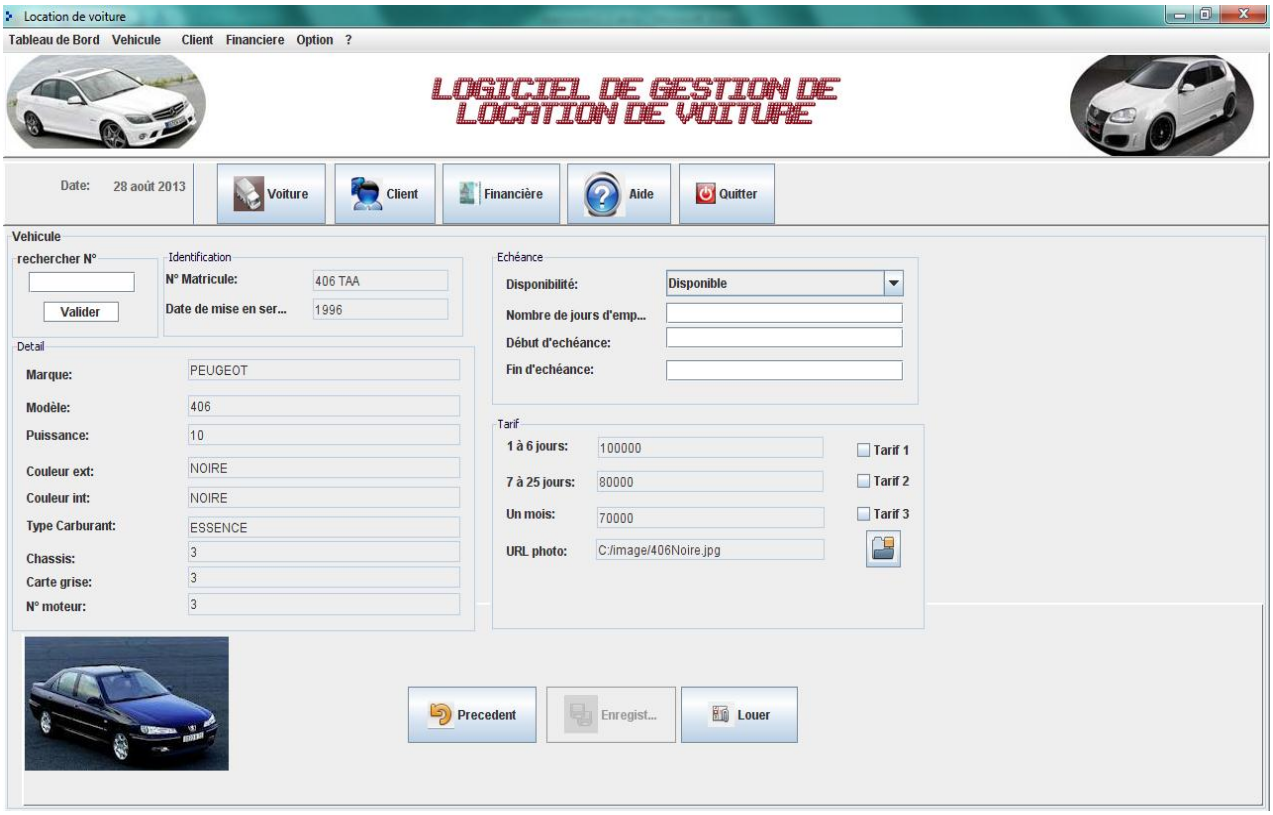

Figure 4.10 : *Détail de la voiture sélectionnée*

Apres avoir cliqué sur le bouton Louer, le logiciel demande d'entrer des informations utile concernant le client. Apres avoir enregistré ce dernier, l'utilisateur peut tirer un contrat en format PDF qui attend juste d'être signé par les deux entités.

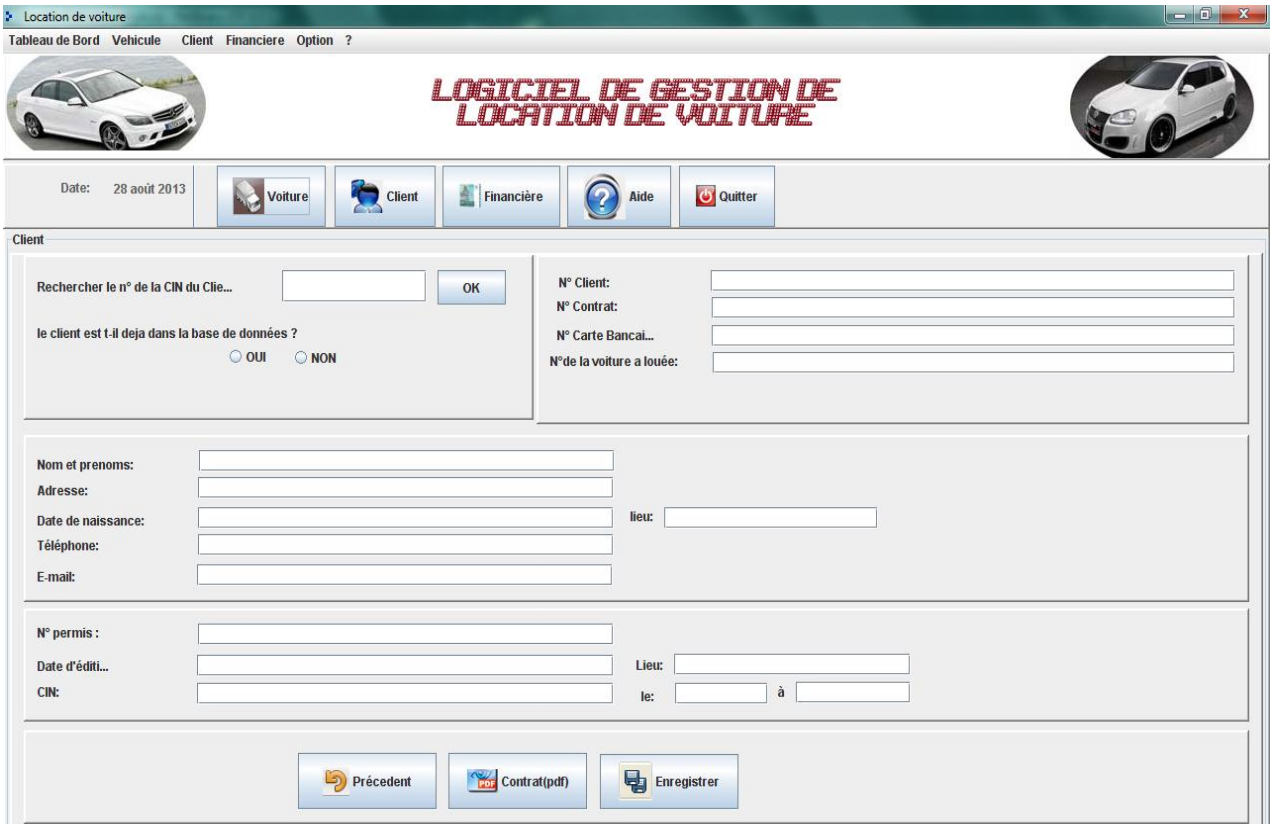

Figure 4.11 : *Information concernant le client*

Avec LGLV les utilisateurs peuvent ajouter de nouveau véhicule, et aussi modifier leur état dans le menu Véhicule. Ils peuvent aussi ajouter de nouveaux clients et regarder leur liste.

Dans le menu Financier il y a deux sous menus :

- Recette qui montre un tableau de toutes les recettes de l'entreprise.
- Dépense qui montre un autre tableau de toutes les dépenses de l'entreprise avec le type de révision des véhicules. Les utilisateurs peuvent aussi ajouter d'autres dépenses et d'exporter ce tableau en format EXCEL.

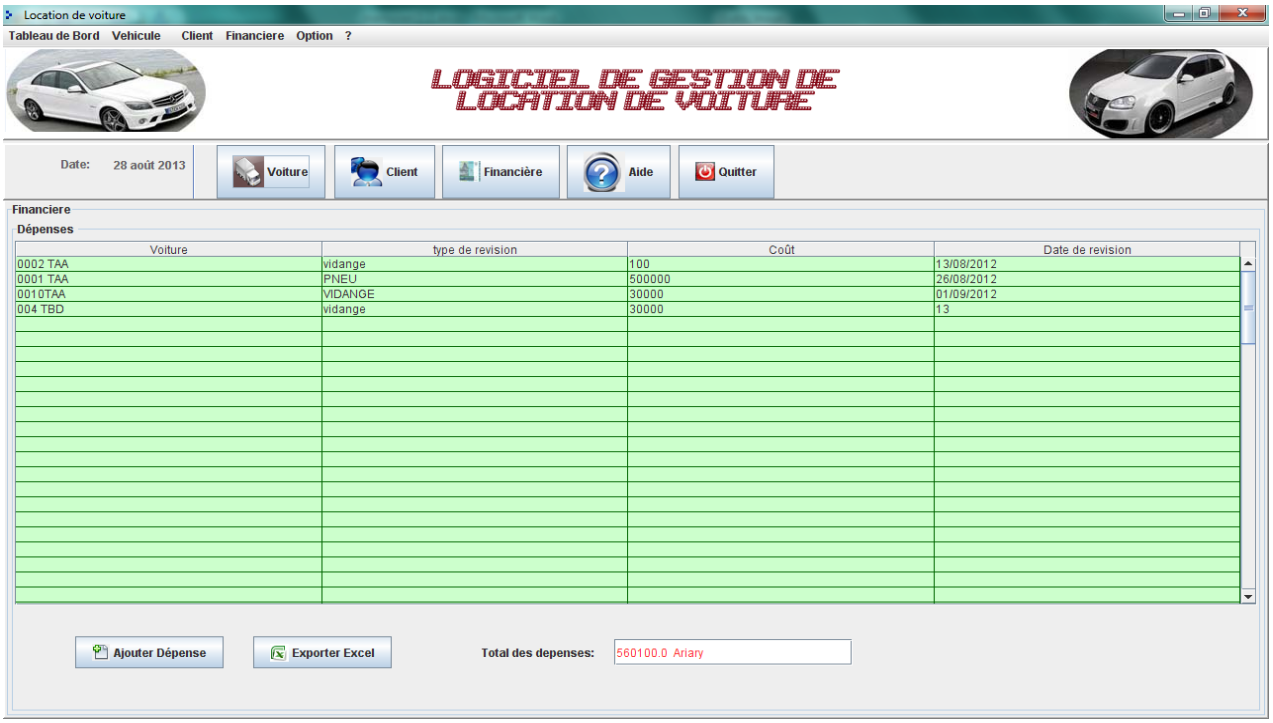

Figure 4.12 : *Affichage des dépenses*

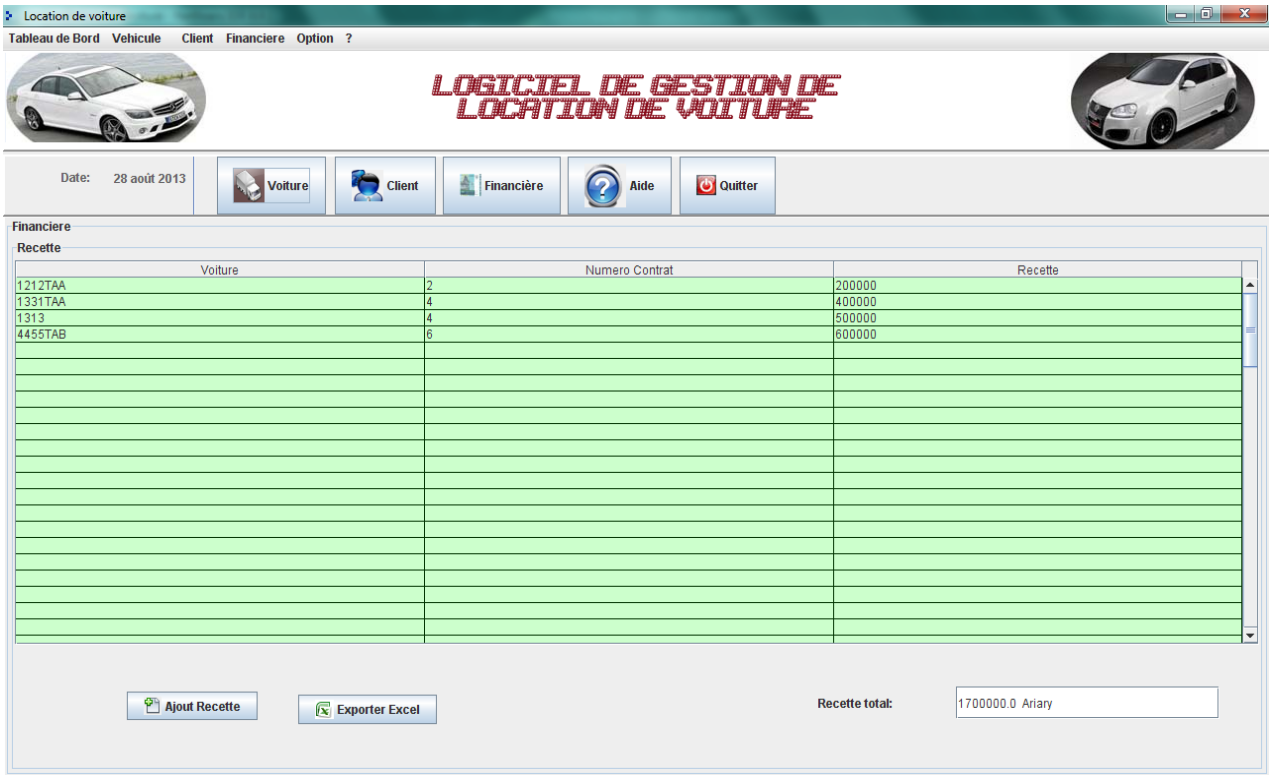

Figure 4.13 : *Affichage des recettes*

4.7 Conclusion

Avant d'entamer le développement d'application, une démarche méthodique doit être mise en place afin d'améliorer la production des applications. Plusieurs méthodes, qui suivent une démarche spécifique, ont été élaborées pour le développement d'application tel que Merise. Merise est une méthode de conception, de modélisation et de réalisation de projet informatique, elle sépare les données des traitements lors de ces différentes phases de développement du projet. Pour la partie réalisation le choix de l'architecture et outils de développement sont très important.

CONCLUSION GENERALE

L'objectif premier d'un système d'information quel qu'il soit est de permettre à plusieurs utilisateurs d'accéder aux même informations.

Le but de cette étude est la mise en place d'une architecture client-serveur. Cette architecture présente un coût moins élevé et un temps de réponse réseau beaucoup plus rapide par rapport à d'autres architectures client-serveur.

Le présent mémoire a porté sur l'étude de la gestion de location de voiture. Cette application permet au personnel d'une entreprise de location de voitures de sécuriser leurs données et aussi d'automatiser leurs tâches courantes. Par conséquent les procédures de location vont être plus efficaces et les pertes de données seront minimisées.

ANNEXE 1

CODE SOURCE DE LA FENETRE D'AUTHENTIFICATION

```
packagegestiondelocationvoiture;

importjava.awt.Dialog.ModalExclusionType;
importjava.awt.Image;
importjava.awt.Toolkit;
import java.net.URL;
importjavax.swing.JFrame;
importjava.sql.Statement;
importjava.sql.Connection;
importjava.sql.DriverManager;
importjava.sql.ResultSet;
importjava.sql.SQLException;
importjava.util.logging.Level;
importjava.util.logging.Logger;
importjavax.swing.JOptionPane;
importjavax.swing.JTable;
public class Authentification1 extends javax.swing.JFrame {
static Connection         connection;
static Statement          statement;
staticResultSetresultSet;
      String                 initiale;
String  i ;
int j=1;
    String serveur="jdbc:mysql://192.168.0.10/mlr1";
          String user= "root";
          String motdepassbase="";
   /** Creates new form Authentification */
public Authentification1() {
```

```
initComponents();

    }

@SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

        jLabel2 = new javax.swing.JLabel();
        jPanel1 = new javax.swing.JPanel();
jButtonEntrer = new javax.swing.JButton();
        jLabel3 = new javax.swing.JLabel();
jPasswordFieldMotDePasse = new javax.swing.JPasswordField();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
jTextFieldUtilisateur = new javax.swing.JTextField();
        jLabel6 = new javax.swing.JLabel();
        jPanel2 = new javax.swing.JPanel();
        jLabel1 = new javax.swing.JLabel();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setTitle("LGLV");
setLocationByPlatform(true);
setName("authentification"); // NOI18N
setResizable(false);

jLabel2.setBackground(new java.awt.Color(255, 255, 255));

jPanel1.setBackground(new java.awt.Color(255, 255, 255));
jPanel1.setForeground(new java.awt.Color(255, 255, 255));

jButtonEntrer.setBackground(new java.awt.Color(255, 255, 255));
```

```
jButtonEntrer.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/gestiondelocationvoiture/okbutton.png"))); //
NOI18N
jButtonEntrer.setBorder(null);
jButtonEntrer.setBorderPainted(false);
jButtonEntrer.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(java.awt.event.ActionEventevt) {
jButtonEntrerActionPerformed(evt);
        }
    });

jPasswordFieldMotDePasse.setToolTipText("mot de passe"); // NOI18N
jPasswordFieldMotDePasse.addKeyListener(new java.awt.event.KeyAdapter() {
public void keyPressed(java.awt.event.KeyEventevt) {
jPasswordFieldMotDePasseKeyPressed(evt);
        }
    });

jLabel4.setFont(new java.awt.Font("Felix Titling", 0, 14));
jLabel4.setText("UTILISATEURS :");

jLabel5.setFont(new java.awt.Font("Felix Titling", 0, 14));
jLabel5.setText("mot de passe  :");

jLabel6.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/gestiondelocationvoiture/clé.gif"))); // NOI18N

javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);
jPanel1.setLayout(jPanel1Layout);
jPanel1Layout.setHorizontalGroup(
jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(jPanel1Layout.createSequentialGroup()
```

```
            .addGap(121, 121, 121)
            .addComponent(jLabel5, javax.swing.GroupLayout.PREFERRED_SIZE, 108,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap(411, Short.MAX_VALUE))

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
               .addGap(474, 474, 474)
               .addComponent(jButtonEntrer, javax.swing.GroupLayout.PREFERRED_SIZE, 28,
javax.swing.GroupLayout.PREFERRED_SIZE)
               .addContainerGap(138, Short.MAX_VALUE)))

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
               .addGap(275, 275, 275)
               .addComponent(jLabel3)
               .addContainerGap(365, Short.MAX_VALUE)))

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
               .addGap(237, 237, 237)
               .addComponent(jPasswordFieldMotDePasse,
javax.swing.GroupLayout.PREFERRED_SIZE, 210,
javax.swing.GroupLayout.PREFERRED_SIZE)
               .addContainerGap(193, Short.MAX_VALUE)))

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
               .addGap(121, 121, 121)
               .addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE, 108,
javax.swing.GroupLayout.PREFERRED_SIZE)
               .addContainerGap(411, Short.MAX_VALUE)))
```

```
.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(237, 237, 237)
                .addComponent(jTextFieldUtilisateur,
javax.swing.GroupLayout.PREFERRED_SIZE, 209,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(194, Short.MAX_VALUE)))

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()
                .addContainerGap(236, Short.MAX_VALUE)
                .addComponent(jLabel6)
.addContainerGap(200, Short.MAX_VALUE)))
        );
jPanel1Layout.setVerticalGroup(
jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()
            .addContainerGap(271, Short.MAX_VALUE)
            .addComponent(jLabel5, javax.swing.GroupLayout.PREFERRED_SIZE, 26,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addContainerGap())
.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()
                .addContainerGap(264, Short.MAX_VALUE)
                .addComponent(jButtonEntrer)
.addContainerGap()))
.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
```

```
                .addContainerGap()
                .addComponent(jLabel3)
                .addContainerGap(297, Short.MAX_VALUE)))

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
jPanel1Layout.createSequentialGroup()
                .addContainerGap(268, Short.MAX_VALUE)
                .addComponent(jPasswordFieldMotDePasse,
javax.swing.GroupLayout.PREFERRED_SIZE, 29,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addContainerGap()))
.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(225, 225, 225)
                .addComponent(jLabel4, javax.swing.GroupLayout.PREFERRED_SIZE, 33,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(50, Short.MAX_VALUE)))

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addGap(222, 222, 222)
                .addComponent(jTextFieldUtilisateur,
javax.swing.GroupLayout.PREFERRED_SIZE, 33,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(53, Short.MAX_VALUE)))

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel6)
.addContainerGap(93, Short.MAX_VALUE)))
```

```
        );

jPanel2.setBackground(new java.awt.Color(255, 255, 255));

jLabel1.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/gestiondelocationvoiture/soratra.png"))); //
NOI18N

javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);
jPanel2.setLayout(jPanel2Layout);
jPanel2Layout.setHorizontalGroup(
jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 640, Short.MAX_VALUE)

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(jPanel2Layout.createSequentialGroup()
                    .addGap(98, 98, 98)
                    .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 506,
javax.swing.GroupLayout.PREFERRED_SIZE)
.addContainerGap(36, Short.MAX_VALUE)))
        );
jPanel2Layout.setVerticalGroup(
jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 79, Short.MAX_VALUE)

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(jPanel2Layout.createSequentialGroup()
                    .addComponent(jLabel1)
.addGap(0, 0, Short.MAX_VALUE)))
        );

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
```

```
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jLabel2, javax.swing.GroupLayout.DEFAULT_SIZE, 640,
Short.MAX_VALUE)
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addComponent(jPanel2, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
);
layout.setVerticalGroup(
layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
            .addGap(0, 382, Short.MAX_VALUE)
            .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE, 24,
javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(69, 69, 69)
                .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(29, Short.MAX_VALUE)))
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED_SIZE, 66,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(0, 340, Short.MAX_VALUE)))
    );
```

```
pack();
   }// </editor-fold>
public String etude(){
try
        {

try {
Class.forName("com.mysql.jdbc.Driver");
        } catch (ClassNotFoundException ex) {
//              Logger.getLogger(FenetreSuivi.class.getName()).log(Level.SEVERE, null, ex);
}
System.out.println("---------- aaaaaaaaaaaaaaaaaaa ----------");
//connection = DriverManager.getConnection("jdbc:mysql://192.168.0.10/mlr1","root","");
connection = DriverManager.getConnection(serveur,user,motdepassbase);
//                  connection =
DriverManager.getConnection("jdbc:mysql://localhost/mlr1","root","");
statement = connection.createStatement();
resultSet=statement.executeQuery("SELECT `itest` FROM `mlr1`.`ldap`WHERE `eyks` LIKE
'ratabe6'");
while (resultSet.next()){
initiale= resultSet.getString("itest");
//                  System.out.println("eeeeeee"+initiale);
}
connection.close();
statement.close() ;

        }
catch (SQLException ex){}

returninitiale;
}
```

```
private void jButtonEntrerActionPerformed(java.awt.event.ActionEventevt) {
etude();

      String utilisateur=jTextFieldUtilisateur.getText();


//       fd.setVisible(true);

System.out.println("eeeeeee"+initiale);


if (j<=3){
if(evt.getSource()==jButtonEntrer) {
          String choix=new String();

choix=(String) jPasswordFieldMotDePasse.getText();
System.out.println("choix="+choix);
i=initiale;
if (i.equals("fenosoa"))
       {
try
      {
try {
Class.forName("com.mysql.jdbc.Driver");
                  } catch (ClassNotFoundException ex) {
Logger.getLogger(Authentification1.class.getName()).log(Level.SEVERE, null, ex);
JOptionPane.showMessageDialog(this,"Serveuréteint"+j,"!!!!!!!!!!!",JOptionPane.OK_CANCEL_
OPTION);
                  }
connection = DriverManager.getConnection(serveur,user,motdepassbase);
//connection = DriverManager.getConnection("jdbc:mysql://localhost/Essai","root","");
statement = connection.createStatement();
```

```
System.out.println("---------- Insertion du Formulaire ----------");
System.out.println("Ouverture de la connexion à la base de données");

resultSet=statement.executeQuery("SELECT `motDePasse`FROM `authentification`WHERE
`Utilisateur` LIKE '"+utilisateur+"'");
while (resultSet.next()){
                                String MotDePasse= resultSet.getString("motDePasse");
System.out.println("itony mot de passse"+MotDePasse);

            if (choix.equals(""+MotDePasse))
            {
GestionLocationVoitureglv=new GestionLocationVoiture();

glv.Ouvrir();


GestionLocationVoiturefen= new GestionLocationVoiture();

fen.demarrer();


setVisible(false);

             }
else
             {
JOptionPane.showMessageDialog(this,"Votre mot de passe est incorrect nombre de tentative =
"+j+"/3","GLV-Message d erreur",JOptionPane.OK_CANCEL_OPTION);
intval=JOptionPane.OK_OPTION;
if(val==0){
                        j=j+1;
System.out.println(""+j);}
```

```
                } }}
catch (SQLException ex){
JOptionPane.showMessageDialog(this,"ffff "+j,"GLV- Message d
erreur",JOptionPane.OK_CANCEL_OPTION);
            }

}

else{
JOptionPane.showMessageDialog(this,"Entrer le code d'enregistrement!!!","GLV-Message d
erreur",JOptionPane.OK_CANCEL_OPTION);

}
   }
   }

else
{
setVisible(false);
 }
   }

public static Image sary(JFramej,String s){
Toolkit toolkit = Toolkit.getDefaultToolkit();
URL url = j.getClass().getResource(s);
Image image = toolkit.getImage(url);
return image;
}



private void jPasswordFieldMotDePasseKeyPressed(java.awt.event.KeyEventevt) {
```

```
if (evt.getKeyCode()==10){

etude();

      String utilisateur=jTextFieldUtilisateur.getText();

System.out.println("eeeeeee"+initiale);



if (j<=3){
         String choix=new String();

choix=(String) jPasswordFieldMotDePasse.getText();
System.out.println("choix="+choix);
         i=initiale;
if (i.equals("fenosoa"))
       {
try
      {
try {
Class.forName("com.mysql.jdbc.Driver");
connection = DriverManager.getConnection(serveur,user,motdepassbase);
                  } catch (ClassNotFoundException ex) {
Logger.getLogger(Authentification1.class.getName()).log(Level.SEVERE, null, ex);
JOptionPane.showMessageDialog(this,"Serveuréteint"+j,"!!!!!!!!!!!",JOptionPane.OK_CANCEL_
OPTION);
                  }

//connection = DriverManager.getConnection("jdbc:mysql://localhost/Essai","root","");
statement = connection.createStatement();
System.out.println("---------- Insertion du Formulaire ----------");
```

```
System.out.println("Ouverture de la connexion à la base de données");

resultSet=statement.executeQuery("SELECT `motDePasse`FROM `authentification`WHERE
`Utilisateur` LIKE '"+utilisateur+"'");
while (resultSet.next()){
                                String MotDePasse= resultSet.getString("motDePasse");
System.out.println("itony mot de passse"+MotDePasse);

          if (choix.equals(""+MotDePasse))
          {          GestionLocationVoiturefen= new GestionLocationVoiture();

fen.demarrer();
fen.Ouvrir();
setVisible(false);

           }
else
           {
JOptionPane.showMessageDialog(this,"Votre mot de passe est incorrect nombre de tentative =
"+j+"/3","GLV-Message d erreur",JOptionPane.OK_CANCEL_OPTION);
intval=JOptionPane.OK_OPTION;
if(val==0){
                          j=j+1;
System.out.println(""+j);}

            } }}
catch (SQLException ex){
JOptionPane.showMessageDialog(this,"ffff "+j,"GLV-Message d
erreur",JOptionPane.OK_CANCEL_OPTION);
          }

}
```

```
else{
JOptionPane.showMessageDialog(this,"Entrer le code d'enregistrement!!!","GLV-Message d
erreur",JOptionPane.OK_CANCEL_OPTION);
}
    }

else
{
setVisible(false);
 }

    }
    }
public static void main(String args[]) {
java.awt.EventQueue.invokeLater(new Runnable() {
public void run() {
new Authentification1().setVisible(true);
}
        });
    }

    // Variables declaration - do not modify
privatejavax.swing.JButtonjButtonEntrer;
privatejavax.swing.JLabel jLabel1;
privatejavax.swing.JLabel jLabel2;
privatejavax.swing.JLabel jLabel3;
privatejavax.swing.JLabel jLabel4;
privatejavax.swing.JLabel jLabel5;
privatejavax.swing.JLabel jLabel6;
privatejavax.swing.JPanel jPanel1;
privatejavax.swing.JPanel jPanel2;
```

```
privatejavax.swing.JPasswordFieldjPasswordFieldMotDePasse;
privatejavax.swing.JTextFieldjTextFieldUtilisateur;
    // End of variables declaration

}
```

ANNEXE 2

SCRIPT SQL POUR LA CREATION DE LA BASE DE DONNEES

```
DROP DATABASE IF EXISTS GLV;

CREATE DATABASE IF NOT EXISTS GLV;
USE GLV;
# ------------------------------------------------------------------------
----
#       TABLE : MODÈLE
# ------------------------------------------------------------------------
----

CREATE TABLE IF NOT EXISTS MODÈLE
(
   NOMMODÈLE CHAR(32) NOT NULL  ,
   MARQUE CHAR(32) NULL  ,
   PUISSANCE CHAR(32) NULL  ,
   CAUTION CHAR(32) NULL
, PRIMARY KEY (NOMMODÈLE)
 )
comment = "";

# ------------------------------------------------------------------------
----
#       TABLE : TYPELOCATION
# ------------------------------------------------------------------------
----

CREATE TABLE IF NOT EXISTS TYPELOCATION
 (
   CODETARIF CHAR(32) NOT NULL  ,
   LIBELLÉTARIF CHAR(32) NULL
   , PRIMARY KEY (CODETARIF)
 )
comment = "";

# ------------------------------------------------------------------------
----
#       TABLE : VEHICULE
# ------------------------------------------------------------------------
----

CREATE TABLE IF NOT EXISTS VEHICULE
(
   NOMINÉRALOGIE CHAR(32) NOT NULL  ,
   NOMMODÈLE CHAR(32) NOT NULL  ,
   COULEUR CHAR(32) NULL  ,
   KMENCOURS CHAR(32) NULL
   , PRIMARY KEY (NOMINÉRALOGIE)
 )
comment = "";
```

```
# ------------------------------------------------------------------------
----
#       INDEX DE LA TABLE VEHICULE
# ------------------------------------------------------------------------
----


CREATE  INDEX I_FK_VEHICULE_MODÈLE
     ON VEHICULE (NOMMODÈLE ASC);

# ------------------------------------------------------------------------
----
#       TABLE : CLIENT
# ------------------------------------------------------------------------
----

CREATE TABLE IF NOT EXISTS CLIENT
 (
   NOCLIENT CHAR(32) NULL  ,
   NOMCLIENT CHAR(32) NULL  ,
   ADRESSE CHAR(32) NULL  ,
   CODEPOSTAL CHAR(32) NULL  ,
   CIN CHAR(32) NOT NULL
   , PRIMARY KEY (CIN)
 )
comment = "";

# ------------------------------------------------------------------------
----
#       TABLE : CONTRAT
# ------------------------------------------------------------------------
----

CREATE TABLE IF NOT EXISTS CONTRAT
(
NOCONTRAT CHAR(32) NOT NULL  ,
   CIN CHAR(32) NOT NULL  ,
NOMINÉRALOGIE CHAR(32) NOT NULL  ,
   DATEETABLISSEMENT CHAR(32) NULL  ,
DATEDEBUTPREV CHAR(32) NULL  ,
   DATEFINPREV CHAR(32) NULL
, PRIMARY KEY (NOCONTRAT)
 )
comment = "";

# ------------------------------------------------------------------------
----
#       INDEX DE LA TABLE CONTRAT
# ------------------------------------------------------------------------
----


CREATE  INDEX I_FK_CONTRAT_CLIENT
ON CONTRAT (CIN ASC);
```

```
CREATE  INDEX I_FK_CONTRAT_VEHICULE
     ON CONTRAT (NOMINÉRALOGIE ASC);

# ------------------------------------------------------------------------
----
#       TABLE : MAINTENANCE
# ------------------------------------------------------------------------
----

CREATE TABLE IF NOT EXISTS MAINTENANCE
 (
   CODERÈVISION CHAR(32) NOT NULL  ,
   LIBELÉREVISION CHAR(32) NULL
   , PRIMARY KEY (CODERÈVISION)
 )
comment = "";

# ------------------------------------------------------------------------
----
#       TABLE : TARIFIER
# ------------------------------------------------------------------------
----

CREATE TABLE IF NOT EXISTS TARIFIER
(
NOMMODÈLE CHAR(32) NOT NULL  ,
   CODETARIF CHAR(32) NOT NULL  ,
TARIFLOCATION CHAR(32) NULL
   , PRIMARY KEY (NOMMODÈLE,CODETARIF)
 )
comment = "";

# ------------------------------------------------------------------------
----
#       INDEX DE LA TABLE TARIFIER
# ------------------------------------------------------------------------
----


CREATE  INDEX I_FK_TARIFIER_MODÈLE
     ON TARIFIER (NOMMODÈLE ASC);

CREATE  INDEX I_FK_TARIFIER_TYPELOCATION
     ON TARIFIER (CODETARIF ASC);

# ------------------------------------------------------------------------
----
#       TABLE : CHOISIR_TARIF
# ------------------------------------------------------------------------
----

CREATE TABLE IF NOT EXISTS CHOISIR_TARIF
 (
   CODETARIF CHAR(32) NOT NULL  ,
```

```
   NOCONTRAT CHAR(32) NOT NULL
   , PRIMARY KEY (CODETARIF,NOCONTRAT)
 )
comment = "";

# ------------------------------------------------------------------------
----
#       INDEX DE LA TABLE CHOISIR_TARIF
# ------------------------------------------------------------------------
----


CREATE  INDEX I_FK_CHOISIR_TARIF_TYPELOCATION
ON CHOISIR_TARIF (CODETARIF ASC);

CREATE  INDEX I_FK_CHOISIR_TARIF_CONTRAT
ON CHOISIR_TARIF (NOCONTRAT ASC);

# ------------------------------------------------------------------------
----
#       TABLE : REVISER
# ------------------------------------------------------------------------
----

CREATE TABLE IF NOT EXISTS REVISER
 (
   CODERÈVISION CHAR(32) NOT NULL  ,
   NOMINÉRALOGIE CHAR(32) NOT NULL  ,
   DATEDEBUTRÉVISION CHAR(32) NULL  ,
   DATEFINRÉVISION CHAR(32) NULL  ,
   COUTMAINTENANCE CHAR(32) NULL  ,
   KILOMETRAGE CHAR(32) NULL
   , PRIMARY KEY (CODERÈVISION,NOMINÉRALOGIE)
)
comment = "";

# ------------------------------------------------------------------------
----
#       INDEX DE LA TABLE REVISER
# ------------------------------------------------------------------------
----


CREATE  INDEX I_FK_REVISER_MAINTENANCE
     ON REVISER (CODERÈVISION ASC);

CREATE  INDEX I_FK_REVISER_VEHICULE
ON REVISER (NOMINÉRALOGIE ASC);


# ------------------------------------------------------------------------
----
#       CREATION DES REFERENCES DE TABLE
# ------------------------------------------------------------------------
----
```

```
ALTER TABLE VEHICULE
  ADD FOREIGN KEY FK_VEHICULE_MODÈLE (NOMMODÈLE)
      REFERENCES MODÈLE (NOMMODÈLE) ;


ALTER TABLE CONTRAT
  ADD FOREIGN KEY FK_CONTRAT_CLIENT (CIN)
      REFERENCES CLIENT (CIN) ;


ALTER TABLE CONTRAT
  ADD FOREIGN KEY FK_CONTRAT_VEHICULE (NOMINÉRALOGIE)
      REFERENCES VEHICULE (NOMINÉRALOGIE) ;


ALTER TABLE TARIFIER
  ADD FOREIGN KEY FK_TARIFIER_MODÈLE (NOMMODÈLE)
REFERENCES MODÈLE (NOMMODÈLE) ;


ALTER TABLE TARIFIER
ADD FOREIGN KEY FK_TARIFIER_TYPELOCATION (CODETARIF)
      REFERENCES TYPELOCATION (CODETARIF) ;


ALTER TABLE CHOISIR_TARIF
  ADD FOREIGN KEY FK_CHOISIR_TARIF_TYPELOCATION (CODETARIF)
      REFERENCES TYPELOCATION (CODETARIF) ;


ALTER TABLE CHOISIR_TARIF
  ADD FOREIGN KEY FK_CHOISIR_TARIF_CONTRAT (NOCONTRAT)
      REFERENCES CONTRAT (NOCONTRAT) ;


ALTER TABLE REVISER
  ADD FOREIGN KEY FK_REVISER_MAINTENANCE (CODERÈVISION)
      REFERENCES MAINTENANCE (CODERÈVISION) ;


ALTER TABLE REVISER
  ADD FOREIGN KEY FK_REVISER_VEHICULE (NOMINÉRALOGIE)
REFERENCES VEHICULE (NOMINÉRALOGIE) ;
```

BIBLIOGRAPHIES

[1] George R., « *JDBC et Java* », Guide de programmeur, 2è édition, 2001.

[2] Philippe M., « *Bases de Données* », U.F.R Université des Sciences et Technologies de Lille, Version 1.3, 1999.

[3] RANDRIARIJAONA L. E., «*Développement d'application d'entreprise* », Cours I5-TCO, Dép. TCO-E.S.P.A., A.U 2009-2010.

[4] Jean M.D., « *Développons en Java* », Version 1.40, aout 2010.

[5] RABEHERIMANANA L. , «*Système d'information* », Cours I5-TCO, Dép. TCO-E.S.P.A., A.U 2009-2010.

[6] Claude S., « *RESEAUX ET TELECOMS* »,2003.

[7] Richard G., « *Langage SQL* », Université de Nice Sophia-Antipolis, version 2.3, novembre 2000.

[8] Philippe R., « Pratique MySQL ET PHP »,4ème édition, 2009

[9] Céline R., « Eléments d'algèbre relationnelle », cours 2.

[10] http://www.CodeS-SourceS.com

[11] http://www.memoireenligne.com

PAGE DE RENSEIGNEMENT

Nom :RALAIARY

Prénoms : Andry Tanjona Harinosy

Adresse de l'auteur : lot B 93 bis Andrefantsena Sabotsy Namehana

Tel :+261 (0) 33 14 476 27

E-mail :ralaiaryandrytanjona@gmail.com

Titre du mémoire : « CONCEPTION ET REALISATION D'UN LOGICIEL DE GESTION DE LOCATION DE VOITURE UTILISANT L'ARCHITECTURE CLIENT SERVEUR»

Nombre de pages : 89

Nombres de figures : 21

Nombre de tableaux : 22

Mots clés : Application JAVA, architecture Client Serveur, logiciel de gestion de location de voiture.

Directeur de mémoire : Mr RATSIRANTO Albert

RESUME

La présente application porte sur l'élaboration d'un logiciel de location de voiture utilisant l'architecture client-serveur. Sa conception a nécessité plusieurs démarches. La première étape consiste à la présentation générale de la base de données ainsi que son accès à partir de l'application Java. Ceci, afin de pouvoir déterminer la base de données et le type de pilote le plus adapté à la réalisation de l'application. L'étape suivante présente les avantages et les inconvénients des différentes architectures Client-serveur, afin de définir l'architecture le mieux appropriée. La dernière étape est consacrée à la réalisation.

Cet ouvrage a permis de mieux comprendre les différentes étapes pour la réalisation d'une application Client-serveur, et sa mise en pratique au niveau de la gestion de location de voiture.

ABSTRACT

This application focuses on the development of software for car rental using the client-server architecture. Its design required several steps. The first step is the overview of the database and its access from the Java application. This is, in order to choose the database and the most suitable type of driver for making the application. The next step presents the advantages and disadvantages of different client server architectures to define the most appropriate architecture. The last step is devoted to the realization.

This work has led to a better understanding of the different steps for achieving a client application server and its implementation in the management of car rental.