REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

**UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE D'ORAN**

**« MOHAMED BOUDIAF »**

## FACULTE DE GENIE ELECTRIQUE

## DEPARTEMENT D'ELECTRONIQUE

# MEMOIRE

EN VUE DE L'OBTENTION DU DIPLOME DE

# MAGISTER

Spécialité : Electronique

Option : Automatique-Robotique-Productique

Présenté Par

**MEKAMCHA KHALID**

**Sujet:**

# DEVELOPPEMENT D'UN LOGICIEL DE G.P.A.O (GESTION DE PRODUCTION ASSISTEE PAR ORDINATEUR)

Devant le jury composé de:

| | | |
|---|---|---|
| **W. NOUIBET** | **MC (USTOMB)** | **Président** |
| **S. ZAKI** | **MC (UABT)** | **Encadreur** |
| **L. GHOMRI** | **MA (UABT)** | **Co-Encadreur** |
| **Z. AHMED-FOITIH** | **MC (USTOMB)** | **Examinateur** |
| **K. HACHEMI** | **MC (UOSénia)** | **Examinateur** |

# Remerciements

# Remerciements

Grâce soit rendue à Allah le clément et miséricordieux de nous avoir donné la force et le courage, et munis de patience pour accomplir ce travail.

Au terme de ce labeur et à l'occasion de cette évènement gratifiant, nous adressons nos sincères remerciements et exprimons toute notre gratitude à :

Mr SARI Zaki d'avoir accepté de nous encadrer pour ce projet, pour les conseils judicieux et pénétrants qu'il nous a prodigué tout au long de ce travail, ainsi que pour sa remarquable sympathie.

Melle GHOMRI Latifa notre co-encadreur qui a eu la complaisance de nous accueillir au centre de calcul, pour sa gentillesse, son aide précieuse et ses remarques subtiles.

Je remercie vivement Monsieur NOUIBET, pour m'avoir fait l'honneur d'accepter d'examiner mes travaux et de présider le jury de ma soutenance de thèse.

Je remercie aussi Les membres du jury et examinateurs, de nous avoir fait l'honneur d'examiner mon travail.

J'adresse un grand merci à BENSMAINE, BOUDGHENE STAMBOULI Yassine et KAZI TANI Chafik, sans qui cette thèse n'aurait jamais lieu, je les remercie tous pour Leur aide, leur disponibilité, leurs conseils, et leur soutien durant toute cette période, ils m'ont toujours redonné confiance et volonté. Qu'ils trouvent ici l'expression de ma sincère reconnaissance et de ma profonde gratitude.

Il me serait impossible de citer nommément toutes les personnes qui m'ont aidé, encouragé et soutenu afin que ce travail puisse voir le jour. Que toutes ces personnes trouvent ici l'expression de ma sincère reconnaissance.

**K.MEKAMCHA**

# *Dédicaces*

# Dédicaces

À celui qui m'a indique la bonne voie en me rappelant que la volonté fait toujours les grands hommes...
Que Dieu vous accorde sa sainte miséricorde et vous accueille en son vaste Paradis

**À mon père Boumediene**

À celle qui a attendu avec patience les fruits de sa bonne éducation,...
Pour son amour…
Pour tous ses sacrifices…
Pour tous l'enseignement qu'elle m'a transmis…
En témoignage de mon éternelle reconnaissance.

**À ma mère Khalida**

**À mon frère Farid et ses enfants Meriem et Layla**
**À ma sœur Doudja et ses enfants Lilia, Sami Boumediène et Zakaria**

En témoignage de mon amour et de ma profonde admiration.
Que Dieu vous protège et vous prête bonne santé et longue vie à vous é à vos adorables enfants

**À mes oncles Abderrahmane, Abdelaziz, Mourad et Noreddine**
**À mes tantes Nadéra, Chahida, et Noria**
**À tous mes cousins et cousines**

En témoignage de mon amour, de mon profond respect et de ma reconnaissance.

**À ma grand-mère Zoubiba**
**À mes cousins Hamid, Kheireddine et nassim**

**À mon oncle Moulay**
**À ma tante Fatiha**

Que Dieu vous accorde sa sainte miséricorde et vous accueille en son vaste Paradis

**À tous mes collègue de travail SDO, à mes amis**
**et à tous ceux qui me sont chers**

# *Sommaire*

# Introduction générale

Une entreprise ne peut assurer la satisfaction des ses salariés, de ses clients et de ses actionnaires privés ou publics qu'en développant la rentabilité de son activité, c'est-à-dire de sa production. L'augmentation constante des coûts de production rend indispensable la recherche de méthodes de gestion permettant aussi bien de maîtriser les coûts que de dominer des procédés sans cesse en évolution.

Gérer une production, c'est d'abord définir une politique de fabrication, analyser les fonctions et caractéristiques du produit, préparer techniquement la fabrication (méthodes). C'est ensuite assurer le plan de charge de la production, ordonnancer les tâches, prévoir et contrôler les coûts, maintenir le matériel. C'est enfin aménager le poste et les conditions de travail en fonction des aspirations des personnels[1].

La gestion de production a pour chacun une signification différente selon son domaine, ses pratiques de travail et la situation concurrentielle de son marché.

Pour optimiser entièrement la production d'une quelconque industrie ou entreprise, il est important de planifier avec un suivi en temps réel et une possibilité de re-planification. Il faut aussi pouvoir accéder aux données et statistiques – non seulement aux rapports quotidiens de production, mais aussi aux données pouvant donner lieu à des analyses plus approfondies, aux rapports mensuels, etc. Plusieurs fournisseurs proposent la couche supérieure de l'architecture informatique reliant tous les sous-systèmes entre eux et fournissant un aperçu complet, nécessaire pour gérer le processus de production dans sa totalité – c'est-à-dire offrant un suivi complet d'un bout à l'autre du processus, de la réservation publicitaire à la distribution.

La réduction des coûts et l'optimisation des processus sont devenus des thèmes familiers de l'industrie. Et si chaque maillon de la chaîne de production peut être efficace et prévisible, il est vraiment avantageux de pouvoir prévoir de manière

[1] Organisation et gestion de la production – Luc Boyer – Série Tables E.O.

automatique et en temps réel comment un processus affecte la production dans sa totalité. « Des coûts de production non planifiés surgissent généralement lorsqu'il y a des retards et c'est là que les choses peuvent devenir très chères. Il est facile d'optimiser la distribution lorsque toutes les étapes de production précédentes ont respecté leurs délais. Le problème est qu'elles ne le font jamais. C'est là qu'interviennent des logiciels de gestion de production, pour aider à construire une logique derrière la production et la distribution, de façon à minimiser les coûts générés par les retards. Et lorsque des problèmes apparaissent, le système doit fournir un support permettant de re-planifier la production et donc de minimiser les dégâts ».

Le premier chapitre va être consacré tout spécialement au domaine général de la G.P.A.O (Gestion de Production Assistée par Ordinateur), nous allons évoquer un bref historique de la gestion de production, nous verrons comment elle est organisée, planifiée, ordonnancée, etc. Nous ferons un petit passage sur l'enregistrement des données de productions, du contrôle de la production et des méthodes de la gestion de production.

Dans le deuxième chapitre nous parlerons des outils d'investigation utilisés dans le domaine de la gestion de Production, car pour mener à bien ces différentes tâches, les entreprises s'aident d'outils informatiques, de la gestion de la production assistée par ordinateur aux progiciels de gestion intégrés : PGI ou *ERP* en passant par la supervision.

La troisième partie quant à elle consiste à la réalisation d'un logiciel de G.P.A.O donc c'est la partie pratique de la thèse, nous verrons dans ce chapitre la partie modélisation du logiciel avec tous les différents moyens utilisés pour arriver à un modèle final de notre logiciel.

La dernière partie traite de la finalisation du logiciel et a pour fin de déterminer les différentes fonctions du logiciel avec un exemple simple de production.

# Chapitre 1

# LA G.P.A.O

# (Gestion de Production Assistée par Ordinateur)

## Introduction :

La **gestion de la production** est une Action ou manière de gérer, d'administrer de diriger d'organiser une entreprise au niveau de la production. Elle doit répondre aux questions: Qui, Fait Quoi, Quand (une notion de délais), Ou, Comment (organisation des postes, planning), Combien (notion de quantité) ? Ceci en vue de fabriquer des produits de qualité, dans les délais requis, au meilleur coût.

L'objectif est d'optimiser les processus de valeur ajoutée en améliorant de manière continue les flux allant des fournisseurs aux clients.

L'ensemble de ces activités doit être réalisé dans le respect des procédures établies (implicitement ou explicitement) par l'entreprise et tenir compte à la fois de la qualité de ses produits ou services, mais aussi de la sécurité de ses salariés ou de son environnement.

Le rôle et l'importance de la Gestion de la Production apparaissent au fur et à mesure de l'accroissement de la concurrence, elle devient de plus en plus importante dans l'organisation d'une entreprise car elle permet d'améliorer sa compétitivité.

## 1.1. Définitions et concepts de base :

La *production* est le processus conduisant à la création de produits par l'utilisation et la transformation de ressources. Les *opérations* sont les activités composant le processus de production.

Le terme « transformation » doit être entendu au sens large, puisqu'il recouvre la modification de l'apparence, des propriétés physico-chimiques, de l'emplacement (transport), etc.

Les « produits » peuvent être des biens (physiques) ou des services.

Les « ressources » consistent principalement en :

- Capital et équipements ;
- Main d'œuvre ;
- Matières (premières, produits semi-finis) ;

- Information.

Exemples : Brique, ciments, matériel de construction, architecte → maçons, électricien, plombier, techniques et appareils de construction → maison, batiment…

*La Gestion* : rationalisation volontaire et organisée de la production.

*1.1. La notion de nomenclature*

Liste hiérarchisée et quantifiée des articles entrant dans la composition d'un article parent. Composés - composants, nomenclature selon la diversité des matières premières utilisées et la diversité des produits finis proposés (IVAX).

*1.2. La notion de gamme*

Comment le produit sera élaboré à partir des matières premières. Enumération de la succession des opérations, les différents temps.

## 1.2. Organisation de la production[2] :

L'organisation de la production diffère selon les entreprises et leur environnement, la clientèle, les fournisseurs et les produits. Mais les objectifs étant similaires, il s'agit de produire en essayant d'approcher les 5 zéros.

Zéro stock
Zéro défaut
Zéro papier
Zéro panne
Zéro délai

*1.2.1. La gestion des flux :*

La **gestion des flux** consiste à gérer l'amélioration des activités manufacturières pour optimiser chacun de ses flux indépendamment.

Un flux, en gestion manufacturière, c'est l'ensemble des activités de production accomplies afin de transformer une (ou des) matière(s) première(s) pour devenir un ou plusieurs produits différents.

[2] http://fr.wikipedia.org/wiki/Gestion_de_la_production

Autrement dit, si une entreprise fabrique des produits différents mais que tous parcourent les mêmes étapes de production, il n'y a qu'un flux. Si par contre la moitié est fabriquée par une série d'activités et que l'autre moitié par une série différente d'activités, il y a deux flux et ainsi de suite.

Un flux peut aussi être appelé une chaîne de valeur (anglais : *value stream).*

*1.2.2. Gestion des stocks :*

Le stock représente l'ensemble des biens qui interviennent dans le cycle d'exploitation de l'entreprise soit pour être vendu en l'état, ou au terme d'un processus de production à venir, ou en cours, soit pour être consommé au premier usage.

Les stocks peuvent représenter la valeur commerciale à un moment donné des marchandises qu'une entreprise commercialise. Dans le milieu industriel, il existe plusieurs types de stock, chacun à un autre niveau de fabrication. Les principaux stocks sont :

*1.2.2.1 Le stock de marchandises :* les stocks des commerçants (revente à profit d'articles sans valeur ajoutée de transformation par l'entreprise).

*1.2.2.2 Le stock de matières premières :* il représente les articles qui ont été achetés auprès de fournisseurs en vue d'une transformation ultérieure.

*1.2.2.3 Le stock des produits en cours de fabrication (semi-finis) :* il représente les articles qui ne sont pas vendables. Ils doivent encore subir des transformations.

*1.2.2.4 Le stock des produits terminés (ou « produits finis ») :* il représente les articles que l'entreprise peut vendre après les avoir fabriquées.

Les modes de gestion des stocks peuvent se classer en trois grandes catégories :

- production sur stock, à partir d'un seuil, ou quantité minimum de réapprovisionnement,
- production juste à temps, type Kanban (mécanisme permettant d'asservir la production ou l'approvisionnement d'un composant à la consommation qui en est faite), en appel par l'aval,
- production à la demande, sur commande.

## 1.3. Planification[3] :

La planification permet de lancer la procédure de calcul des besoins, d'examiner les ordres de fabrication suggérés et éventuellement de les modifier. Nous pouvons alors effectuer un jalonnement à capacité infinie et un calcul des charges et en obtenir les résultats sur un graphique ou sur un tableau. Lorsque nous serons parvenus à un plan de production satisfaisant, nous pourrons transformer les ordres suggérés en ordres fermes.

La planification de la production ne peut intervenir que si les données techniques (nomenclatures, gammes) sont à jour, si nous avons entré des prévisions de vente et des commandes clients et si les paramètres de gestion des articles ont été spécifiés.

Les méthodes de planification principales sont :

- La méthode PERT : Project Evaluation and Review Technique, de planification de projet, initiée par l'US Navy dans les années 1950. C'est une méthode consistant à mettre en ordre sous forme de réseau plusieurs tâches qui grâce à leur dépendance et à leur chronologie concourent toutes à l'obtention d'un produit fini.

- le calcul MRP : calcul des approvisionnements en fonction des besoins prévisionnels en produits finis (J. Orlicky, 1975). En 1960, il représentait un système de planification qui détermine les besoins en composants à partir des demandes en produits finis et des approvisionnements existants ; en 1970, ils lui ont ajouté le calcul des charges de l'outil de production engendrées par les besoins en composants ; et en 1979 ils lui ont intégré un calcul des coûts de production et une planification des besoins intégrant la contrainte charge vs capacité de l'outil de production.

- la méthode OPT : planification des ordres de fabrication en priorité sur les outils de production à capacité limité (E. Goldratt, 1969).

[3] http://fr.wikipedia.org/wiki/Planification

## 1.4. Ordonnancement[4] :

L'ordonnancement se charge de lancer les ordres de fabrication ou d'achats (ou autre suivant le type de valeur ajouté par l'entreprise) auprès du service concerné, à la date planifiée.

La **théorie de l'ordonnancement** est une branche de la recherche opérationnelle qui s'intéresse au calcul de dates d'exécution optimales de tâches. Pour cela, il est très souvent nécessaire d'affecter en même temps les ressources nécessaires à l'exécution de ces tâches. Un problème d'ordonnancement peut être considéré comme un sous-problème de planification dans lequel il s'agit de décider de l'exécution opérationnelle des tâches planifiées.

Un problème d'ordonnancement consiste à organiser dans le temps la réalisation de tâches, compte tenu de contraintes temporelles (délais, contraintes d'enchaînement) et de contraintes portant sur la disponibilité des ressources requises.

En production (manufacturière, de biens, de service), on peut le présenter comme un problème où il faut réaliser le déclenchement et le contrôle de l'avancement d'un ensemble de commandes à travers les différents centres composant le système.

Un ordonnancement constitue une solution au problème d'ordonnancement. Il est défini par le planning d'exécution des tâches (« ordre » et « calendrier ») et d'allocation des ressources et vise à satisfaire un ou plusieurs objectifs. Un ordonnancement est très souvent représenté par un diagramme de Gantt (un outil utilisé (souvent en complément d'un réseau PERT) en ordonnancement et gestion de projet et permettant de visualiser dans le temps les diverses tâches composant un projet. Il permet de représenter graphiquement l'avancement du projet).

[4] http://fr.wikipedia.org/wiki/Ordonnancement

## 1.5. Contrôle de la production :

Le contrôle de la production se fait généralement sur le plan *qualité* et le plan *prix de revient.*

### *1.5.1. Life Cycle Cost :*

Le **Life Cycle Cost** (LCC), est un outil managérial qui permet de connaître le coût de production d'un produit pendant toute sa durée de vie depuis sa fabrication jusqu'au démontage.

### *1.5.2. Coût total de possession :*

Le **coût total de possession** (TCO : ***Total Cost of Ownership***) est un coût qui intègre tous les éléments constitutifs d'un produit manufacturé.

L'analyse TCO a été inventée par le Gartner Group en 1987. Le TCO doit inclure :

Le coût des matières premières ;

Le prix d'achat éventuel, les mensualités pour une location, les frais financiers ;

Les dépenses de mise en route (installation, formation...) ;

Les dépenses de fonctionnement (fonction de l'usage prévu) ;

Les dépenses d'entretien régulier ;

Les dépenses liées à la sécurité et à la qualité ;

Les dépenses d'arrêt de fonctionnement (dépollution, démontage...) ;

Les dépenses de retrait éventuel (reprise du matériel), ou la recette liée à la revente ;

Les dépenses liées aux **équipements et logiciels informatiques et aux conseils associés.**

Le calcul du TCO doit donc s'appuyer sur une solide Analyse du cycle de vie, ou une étude d'éco-conception multi-domaines.

L'éco-conception se base sur un double cahier des charges :

- Fonctionnel (il définit le besoin, les objectifs et liste les contraintes incontournables et contournables)

- Technique (sur la base d'une analyse qualitative et quantitative des impacts (sur un *cycle de vie*) et des substituts et alternatives disponibles dans les conditions techniques et économiques du moment, il définit le mode de production ou de réponse à une demande de service, en introduisant les améliorations environnementales possibles).

## 1.6. Méthodes et outils :

Les différentes méthodes, ou principes, de gestion de la production sont généralement formalisées sous forme d'outils facilitant leur mise en œuvre pratique.

### *1.6.1. Méthodes :*

Planification des besoins en composants ;

SMED (single minute exchange of die) ;

Le principe des 5S (rangement efficace du matériel) ;

La TPM, Maintenance Productive Totale ;

La théorie des contraintes (TOC) ;

Le Lean Management, l'entreprise « agile » (issu du Toyota Production System) ;

Le Kanban[5] : autorégulation de la production en fonction des quantités consommées (T. Ohno, 1959).

### *1.6.2. Outils :*

* Diagramme de Gantt et Réseau PERT pour la gestion de projet ;

* Diagramme de Pareto pour la gestion de la qualité ;

* Réseaux de Petri pour les enchaînements automatisés ;

* Fiches de production ou de manutention sous forme d'étiquettes Kanban.

Ces outils sont généralement informatisés :

* Progiciels ou logiciels de gestion de la production assistée par ordinateur ;

* Progiciels de gestion intégrés : PGI[6] ou *ERP* ;

* Progiciels de gestion de projet ;

[5] http://fr.wikipedia.org/wiki/Kanban

[6] http://fr.wikipedia.org/wiki/Progiciel_de_gestion_int%C3%A9gr%C3%A9

* Superviseurs servant à l'acquisition de données ou au pilotage.

La fonction production qui consiste à ***produire***, en ***temps voulu***, les ***quantités demandées*** par les clients dans des conditions de ***coût de revient*** et de ***qualité*** déterminés, envisage une organisation plus développée en terme de technologie et de qualité. En effet, la gestion de production exige d'appréhender des paramètres portant essentiellement sur l'estimation et le calcul des ressources humaines, matérielles et financières nécessaires à la production, des étapes du processus de production et des normes à respecter. De plus, les changements du marché et l'exigibilité de la demande ont réévalué la valeur de l'information en terme de sa diffusion au sein de l'entreprise et en terme de son rôle permettant de mettre en interaction l'ensemble des fonctions, d'où l'apparition de l' « informatisation » de la gestion de production.

## 1.7. La G.P.A.O : Gestion de Production Assistée par Ordinateur

### *1.7.1. Pourquoi la GPAO ?* [7]

Dans le cadre de la croissance et de la conquête des parts de marchés, les entreprises se voient dans l'obligation d'offrir aux clients le meilleur rapport qualité/prix d'une part, et de maximiser leurs marges par des gains de productivité d'autre part. Les gestionnaires de ces entreprises cherchent constamment à assurer le suivi régulier de la production dans toutes ses étapes, également le contrôle des procédés, des machines et de l'ensemble des ressources, cela implique nécessairement une maitrise efficace des flux d'informations. Effectivement, les progiciels de la GPAO permettent de répondre parfaitement à ces attentes notamment en optimisant la gestion des activités de production, et ce en assurant la gestion efficace de toutes les activités impliquées dans la réalisation des produits depuis l'approvisionnement des

[7] *« La Gestion de Production Assistée par Ordinateur « GPAO » par BISMILLAH Salwa, DABACH Imane, TANANE Malika et GRINAT Khadija »*

matières premières auprès des fournisseurs jusqu'à la livraison des produits aux clients.

### *1.7.2. Méthodes :*

Parler de compétitivité et de concurrence est une activité quotidienne au sein des entreprises. Si par le passé on pouvait se contenter de se baser sur des études de marché ou des rapports d'interventions commerciales pour cerner les attentes des clients et sa propre position par rapport aux concurrents; l'expansion et l'amélioration des moyens de communication, et la fameuse "mondialisation", sont aujourd'hui des facteurs à la fois de développement et de risque pour les sociétés.

L'informatisation de la Gestion de Production vise à maîtriser les flux des matières et les flux d'information se rapportant à l'objectif de l'entreprise, également à développer le système déjà retenu relatif aux types de production. La GPAO, qui traduit l'introduction de l'informatique dans la gestion de production, devient alors un outil indispensable pour maîtriser et régulariser toute la chaîne de production, du devis initial jusqu'à la livraison et la facturation du client[8].

L'accessibilité croissante des outils informatiques permet aujourd'hui à l'ensemble des acteurs des mondes industriels et entrepreneurial de disposer de moyens de contrôler et réguler les productions, en fonctions de paramètres réels (commandes fermes, reportings et tableaux de bords) ou prévisionnels (tendances, études de marché, projections, etc...).

[8] « *La Gestion de Production Assistée par Ordinateur « GPAO » par BISMILLAH Salwa, DABACH Imane, TANANE Malika et GRINAT Khadija »*

### *1.7.3. Origine de la GPAO[9] :*

#### *1.7.3.1. Rappel historique :*

La GPAO, pour Gestion de la Production Assistée par Ordinateur, est un concept de progiciel qui trouve son origine dans les années 1970 dans le secteur de l'industrie automobile.

A l'époque, la production automobile américaine, qui obéit au modèle fordiste, est une production dite "à la chaîne", qui repose sur le principe de la réduction des coûts de production par l'augmentation du volume et la réduction de modèles d'articles produits, les excédents de production devant être stockés.

Pour contrecarrer la domination Américaine, un ingénieur de Toyota, Taiichi Ohno, va définir un nouveau modèle de production, basé sur une régulation de la production en fonction des demandes (fabrication en flux tendus) et l'augmentation de la productivité par l'adoption de technologies de contrôle et de partage de l'information. Ce type de production implique une grande réactivité et adaptabilité des chaînes de production et par conséquent a été un des moteurs à l'origine de l'Automation, telle qu'on la connaît de nos jours.

Afin de réguler au mieux la production, des outils informatiques ont donc été déployés, à la fois pour contrôler les machines en charge de la fabrication mais également pour diminuer les délais entre les commandes et la réalisation des produits.

#### *1.7.3.2. L'informatique et la naissance de la GPAO[10] :*

La Gestion de Production Assistée par Ordinateur est une traduction de la tendance vers l'informatisation du système productif. Elle permet la mise en place d'un progiciel portant sur l'automatisation des différentes fonctions, et cela dans la

[9] http://fr.wikipedia.org/wiki/Gestion_de_la_production_assist%C3%A9e_par_ordinateur

[10] « *La Gestion de Production Assistée par Ordinateur « GPAO » par BISMILLAH Salwa, DABACH Imane, TANANE Malika et GRINAT Khadija »*

logique d'une vision d'ensemble comme le suppose le pilotage stratégique de l'entreprise.

La **GPAO** consiste à maîtriser les flux d'information et à traiter ceux issus du fonctionnement des services administratifs ou opérationnels au profit de l'organisation et de la gestion de la production. La **GPAO** assure à la fois le traitement de l'aspect organisationnel de la production et aussi l'introduction de l'ensemble des outils informatiques permettant d'assurer un pilotage optimisé du processus de production.

Les premières applications de la **GPAO** consistaient à traiter chaque activité de l'entreprise : production, comptabilité, gestion de la qualité en fonction de ses besoins propres en gestion de l'information. Par la suite, le procédé de l'informatisation «par étapes» des entreprises s'est traduit en premier lieu, par la mise en place de plusieurs systèmes (comptabilité, gestion commerciale, gestion de stocks) utilisant des bases de données indépendantes ce qui entraînait des erreurs, gaspillage de temps et manque de communication. Enfin, le développement d'un progiciel de **GPAO** utilisant une base de données commune était la forme développée de l'informatisation de la gestion de production de l'entreprise.

*1.7.3.3. Les besoins des Sociétés de biens :*

Les entreprises et industries qui fabriquent des produits sont soumises à des contraintes communes de gestion à la fois des matériaux, des matériels et des produits finis, contraintes regroupant les problèmes « productifs » d'approvisionnement, de stockage, de procédé de fabrication et d'expédition.

A ces problèmes liés directement à la nature même des produits réalisés, s'ajoutent les thèmes classiques de toute activité professionnelle: les négociations commerciales, le contrôle de la qualité et la facturation.

## 1.8. Enregistrement des données de production :

Un **Manufacturing Execution System** (**MES**) est un système informatique dont les objectifs sont d'abord de collecter en temps réel les données de production de tout ou partie d'une usine. Ces données collectées permettent ensuite de réaliser un certain nombre d'activités d'analyse : traçabilité (généalogie), contrôle de la qualité, suivi de production, ordonnancement, maintenance préventive et curative.

La **traçabilité** est l'information sur la chaîne de production et de distribution d'un produit. Ce sujet a une importance particulière pour les produits concernant la santé humaine, comme l'alimentation et les médicaments ou bien la sécurité (aéronautique, automobile, informatique par exemple). Elle est utilisée aussi dans le suivi du traitement des déchets, puisqu'un producteur de déchets en est responsable jusqu'à son élimination. Elle est aussi de plus en plus une préoccupation du consommateur-citoyen, qui veut par exemple être sûr que le produit ne fait pas intervenir d'éléments contraires à sa morale, comme par exemple le travail des enfants.

Le **contrôle de la qualité** est un aspect de la gestion de la qualité. La recherche systématique en matière d'amélioration de la qualité a été principalement menée par les industriels américains au cours du XX$^{e}$ siècle. Elle a connu une diffusion mondiale à l'occasion de l'aide à l'effort de guerre entrepris par les États-Unis à partir de 1946, avec une étape clé au Japon, notamment. Son intégration en Europe et en France, plus tardive, commencera durant les années 1990, sous l'impulsion des normes sur la qualité.

La **maintenance** vise à maintenir ou rétablir un bien dans un état spécifié afin que celui-ci soit en mesure d'assurer un service déterminé. La **maintenance** regroupe ainsi les actions de dépannage et de réparation, de réglage, de révision, de contrôle et de vérification des équipements matériels (machines, véhicules, objets manufacturés...) ou même immatériels (logiciels). Un service maintenance peut également être amené à participer à des études d'amélioration du process, et doit,

comme de nombreux services de l'entreprise, prendre en considération de nombreuses contraintes comme la qualité, la sécurité, l'environnement…
Souvent un MES se situe entre la partie automatisme de l'usine et les systèmes (ERP : ***Enterprise Resource Planning*** Un **Progiciel de Gestion Intégrée** (**PGI**)).

**Conclusion :**

Nous pouvons conclure ce chapitre par un petit récapitulatif concernant la GPAO, donc la GPAO, en tant que Gestion de la Production Assistée par Ordinateur, est un concept qui semble répondre aux attentes d'une entreprise, elle vient généralement sous forme d'un logiciel, qui est un programme modulaire de gestion de production permettant de gérer l'ensemble des activités, liées à la production, d'une entreprise industrielle :

Gestion des stocks et des achats ;

Gestion de commandes ;

Gestion des produits engendrés par ces commandes ;

Gestion des articles entrant dans la fabrication de ces produits et de leurs nomenclatures ;

Expédition des produits ;

Facturation.

La GPAO, en tant qu'outil de "Gestion de la Production", semble être l'exacte réponse aux exigences d'une entreprise en termes d'organisation interne et de prises de décisions.

Toutefois, nous sommes en droit de se demander comment un seul logiciel, définit en tant que somme d'applicatifs spécialisés, est à même de répondre à un éventail de besoins qui varie d'un secteur d'activité à un autre, d'une entreprise à une autre.

Nous allons, par conséquent, envisager l'étude de la GPAO sous l'angle d'un logiciel de gestion de l'approvisionnement et de la comptabilité, et voir comment un système centralisé aux concepts génériques peut se plier aux spécificités de chaque entreprise.

Pour répondre à l'ensemble de ces besoins, un certain nombre d'applicatifs plus ou moins spécifiques sont disponibles sur le marché, ou développés en interne si l'entreprise requiert des modalités d'utilisation spécifique (centralisation des données de plusieurs sites pour traitement groupé, besoin de restreindre l'accessibilité à certaines informations depuis l'extérieur...)

# Chapitre 2
# Les outils d'investigation dans le domaine de la G.P.A.O

## Introduction :

Se doter d'un logiciel de GPAO devient une nécessité pour de plus en plus d'entreprises afin d'améliorer et de développer leur compétitivité sur un marché fortement concurrentiel.

Le logiciel doit rester un outil "une mine d'informations" au service de l'homme et non pas le contraire, c'est pourquoi son choix doit être fait selon une démarche très rigoureuse et objective. Le chef de projet doit s'y prendre de la même façon que pour la mise en place d'un projet de gestion de production.

La GPAO recouvre deux dimensions : d'une part, une réflexion sur les méthodes d'organisation de la production et d'autre part, un ensemble d'outils informatiques qui permettent d'assurer un pilotage stratégique et optimisé du processus de production.

## 2.1. Informatiser l'organisation générale de l'entreprise :

### 2.1.1. Une solution : un progiciel pour toute l'entreprise

Dans une entreprise qui a commencé à s'informatiser, des outils informatiques hétérogènes coexistent à différents endroits souvent sans communiquer, et répondent à une seule fonction précise : la comptabilité, des tableaux du commercial avec des fichiers clients éparpillés, les informations techniques, les suivis des délais...

Un « progiciel » est une appellation issue de la combinaison « produit-logiciel », c'est un ensemble cohérent et indépendant de programmes, de supports de manipulation ou d'information et d'une documentation en vue de traiter des informations utiles à la production. Par ailleurs, trouver un progiciel adapté

exactement aux besoins peut s'avérer une tâche plus ou moins difficile du fait que chaque progiciel doit être spécifique à l'entreprise[11].

L'intérêt d'un outil de type Progiciel de Gestion Intégré est de centraliser toutes les informations nécessaires au bon fonctionnement de l'entreprise en informatisant les processus principaux :

- **Les processus de réalisation** : de la prospection commerciale, à l'offre de prix, la commande, au déclenchement des approvisionnements, au lancement en fabrication, à la livraison, jusqu'à la facturation.

- **Les processus supports** : la gestion des ressources humaines, la gestion des équipements, des achats, des stocks, la gestion de la relation client, le commerce électronique,

- **Les processus de pilotage** : le planning quotidien de gestion, les suivis à moyen terme (coûts, temps et délais), les tableaux de bord du chef d'entreprise pour le pilotage à long terme (marketing, financier...).

*2.1.1.1. Les avantages :*

- La saisie unique des informations partagée ;
- La recherche rapide et la disponibilité des informations ;
- L'optimisation de l'organisation et des processus ;
- La structuration d'un système d'information central et partagé ;
- L'homogénéisation des pratiques dans l'entreprise ;
- Le gain de temps par l'automatisation de la gestion ;
- La maîtrise des coûts et délais ;
- La possibilité d'anticiper et piloter avec de l'information fiable.
- …

[11] « *La Gestion de Production Assistée par Ordinateur « GPAO » par BISMILLAH Salwa, DABACH Imane, TANANE Malika et GRINAT Khadija »*

*2.1.1.2. Les inconvénients :*

- La difficulté de mise en œuvre et d'appropriation par les utilisateurs
- La nécessité de s'organiser et structurer les données techniques et les processus
- La nécessité de renforcer les compétences, notamment par de la formation
- Les besoins de maintenance
- Le coût du projet
- Un outil complexe bien souvent sous exploité
- Une dépendance à un éditeur unique

## 2.1.2 Les outils logiciels du marché

### 2.1.2.1. La typologie

Ces Pro logiciels sont composés de différents modules indépendants qui traitent une fonction, par exemple la production, le commercial ou la gestion financière, mais qui partagent une seule et unique base de données.

Désignés par Progiciel de Gestion Intégré (PGI ou en anglais, le sigle répandu, ERP pour Enterprise Ressource Planning), il est, selon le grand dictionnaire terminologique, un « logiciel qui permet de gérer l'ensemble des processus d'une entreprise, en intégrant l'ensemble des fonctions de cette dernière comme la gestion des ressources humaines, la gestion comptable et financière, l'aide à la décision, mais aussi la vente, la distribution, l'approvisionnement, le commerce électronique. » ; ces outils permettent de gérer au moins 3 fonctions complètes de l'entreprise, par exemple la production, la gestion de la relation client, et la comptabilité. Les principaux éditeurs d'ERP sont regroupés au sein d'associations :

BASDA pour *Business Application Software Developers Association*

BSA pour *Business Software Alliance.*

Pour les entreprises dont la préoccupation majeure est la production, les logiciels appelés Gestion de Production Assistée par Ordinateur (GPAO) sont spécialisés pour la gestion des données de la production (nomenclatures, articles,

gammes, plans de charge...) et intègrent les fonctions connexes qui interférent avec la production, notamment les stocks, le commercial, les coûts, les achats...

### 2.1.2.2. Un choix important de logiciels

L'offre de logiciel est très large. Selon le répertoire de progiciels du CXP[12], on peut comptabiliser environ 300 logiciels PGI et GPAO dont une centaine peut concerner des petites entreprises. Alors que les leaders du marché (SAP, ORACLE et Microsoft) ciblent essentiellement les groupes et PME importantes, les offres adaptées aux petites entreprises sont proposées par de nombreux éditeurs, avec en tendance, des évolutions au niveau de :

- L'ergonomie et la facilité d'utilisation,
- La spécialisation des applications par activité et métier,
- L'adaptation aux petites entreprises (c'est une cible clientèle recherchée car le marché des grandes entreprises arrive progressivement à saturation !),
- Des coûts plus accessibles pour les petites entreprises,
- L'arrivée de nouvelles solutions en logiciel libre...

### 2.1.2.3. Avantages

Optimisation des processus de gestion (flux économiques et financiers) ;

Cohérence et homogénéité des informations (un seul fichier articles, un seul fichier clients, etc.) ;

Intégrité et unicité du Système d'information ;

Partage du même système d'information facilitant la communication interne et externe ;

Minimisation des coûts : pas d'interface entre les modules, synchronisation des traitements, maintenance corrective simplifiée car assurée directement par l'éditeur et non plus par le service informatique de l'entreprise (celui-ci garde néanmoins sous sa

[12] http://www.cxp.fr/

responsabilité la maintenance évolutive : amélioration des fonctionnalités, évolution des règles de gestion, etc.) ;

Globalisation de la formation (même logique, même ergonomie) ;

Maîtrise des coûts et des délais de mise en œuvre et de déploiement ;

Ce dernier point est essentiel et la mise en œuvre d'un ERP/PGI dans une entreprise est fréquemment associée à une révision en profondeur de l'organisation des tâches et à une optimisation et standardisation des processus, en s'appuyant sur le « cadre normatif » de l'ERP/PGI.

Les ERP/PGI vont pouvoir gérer et prendre en charge :

Plusieurs entités ou organisations (filiales, etc.) ;

Plusieurs périodes (exercices comptables par exemple) ;

Plusieurs devises ;

Plusieurs langues pour les utilisateurs et les clients (cas des multinationales) ;

Plusieurs législations ;

Plusieurs plans de comptes ;

Plusieurs axes d'analyse en informatique décisionnelle.

#### 2.1.2.4. Inconvénients

Coût élevé (cependant, il existe des ERP/PGI qui sont des logiciels libres, les seuls Coûts étant alors la formation des utilisateurs et le service éventuellement assuré par le fournisseur du logiciel) ;

Périmètre fonctionnel souvent plus large que les besoins de l'organisation ou de L'entreprise (le progiciel est parfois sous-utilisé) ;

Lourdeur et rigidité de mise en œuvre ;

Difficultés d'appropriation par le personnel de l'entreprise ;

Nécessité d'une bonne connaissance des processus de l'entreprise (par exemple, une petite commande et une grosse commande nécessitent deux processus différents : il est important de savoir pourquoi, de savoir décrire les différences entre ces deux

processus de façon à bien les paramétrer et à adapter le fonctionnement standard de l'ERP/PGI aux besoins de l'entreprise) ;

Nécessité parfois d'adapter certains processus de l'organisation ou de l'entreprise au progiciel ;

Nécessité d'une maintenance continue.

#### 2.1.2.5. Impact

Les progiciels de gestion intégrés permettent à l'entreprise une meilleure maîtrise de ses activités de production. Le paradigme sur lequel ils se basent repose essentiellement sur une optimisation de l'utilisation des ressources, qu'elles soient humaines ou matérielles. Le PGI induit donc une orientation stratégique vers la réduction des coûts comme vecteur essentiel de la création de valeur donc de la croissance de l'entreprise. Ce modèle est critiqué depuis le début des années 1990 car il met l'entreprise (et éventuellement ses fournisseurs) au centre de l'attention, au détriment du client. Les principaux éditeurs de PGI se sont donc efforcés d'intégrer des fonctionnalités marketing afin d'évoluer vers le nouveau modèle de la gestion de la relation client.

#### 2.1.2.6. Évolution :

Les progiciels de gestion intégrés ont connu leur essor en profitant de l'évolution nécessaire des systèmes d'information pour le passage de l'an 2000 puis pour la mise en place de l'euro. En effet, il était séduisant de remplacer tous les logiciels de gestion de l'entreprise par un intégré offrant « l'état de l'art » plutôt que d'engager des corrections des programmes existants plus ou moins anciens.

Si cette démarche a parfois donné lieu à des démarrages dans l'urgence, l'enjeu de la mise en place d'un PGI aujourd'hui n'est plus de passer l'an 2000, mais d'optimiser la gestion des flux logistiques et financiers de l'entreprise. Dans ce contexte, les logiciels de gestion d'entreprises deviennent de plus en plus concurrentiels face à de grands PGI qui, certes, ont une vaste couverture fonctionnelle, mais qui, une fois sortis de

leur cœur de métier, traitent de manière plus complexe (voire moyenne) certaines fonctions.

D'autre part, l'intégration technique des traitements et des données arrive à un niveau de complexité à la limite du gérable (une correction pouvant avoir des impacts sur tout un ensemble de fonction, avec souvent des corrections pré-requises, co-requises et sous-requises). La plupart de ces progiciels ont désormais un module dédié à la gestion de ces corrections (le module qui gère les modules).

Un virage fonctionnel et technique est en passe d'être pris vers la distribution des fonctions en différentes applications, indépendantes techniquement et interfacées avec le noyau du PGI, le tout agencé autour d'un EAI. Cette architecture permettra d'intégrer des applications centrées sur une fonction ou un métier particulier comme la gestion des dépôts (IMS), la gestion des ateliers (MES), la gestion des laboratoires (LIMS), etc. Conciliant la profondeur métier avec l'intégration, tout en gardant l'indépendance de maintenance de chaque application.

Cette démarche est aujourd'hui embryonnaire, tant au niveau de la performance des architectures techniques que des méthodes de construction fonctionnelle de telles solutions. Mais l'évolution est engagée depuis quelques années dans la structure technique des progiciels et s'accélère.

### 2.1.2.7. Les logiciels libres :

Le développement d'offre en logiciel libre s'explique par le fait que la concurrence n'est plus sur le produit Pro logiciel, mais sur les services associés (paramétrage, formation, assistance...). Cependant, la gratuité d'une licence est loin de signifier un projet à coût réduit car les besoins en compétence sont bien plus importants. Les solutions de PGI libres sont récentes, peu nombreuses, peu spécialisées à un métier et difficilement adaptables, pour l'instant, à la très petite entreprise.

Un logiciel libre propose une licence particulière qui permet l'exécution du programme pour tous usages, la redistribution des copies, l'étude du programme, son amélioration et la publication de ses améliorations.

## 2.2 Typologie des ateliers de production :

Dans l'industrie, l'**atelier** est, dans une usine, l'espace consacré à la fabrication.

A l'origine l'**atelier** est un lieu où l'on travaille le bois. Il devient le lieu de création de l'artisanat et des beaux-arts. Il désigne également le groupe de personnes qui travaillent sous la direction d'un maître.

Au début de l'ère industrielle, les ateliers sont regroupés en manufactures puis en usines.

Les ateliers sont généralement restés, au sein des usines, des zones où est regroupé un même savoir-faire, la maîtrise d'un métier. Seules les petites entreprises, proches de l'artisanat, fabriquent ou réalisent leur production dans un atelier unique où sont implantés tous les moyens de fabrication, c'est-à-dire les postes de travail et les machines-outils.

En construction mécanique, les ateliers dédiés à un métier, sont parfois réimplantés en îlots de fabrication (préféré en français au mot « cellule »), regroupant des machines destinées à des usinages variés.

La **technologie de groupe** est une méthode consistant à regrouper des pièces par familles afin de tirer profit de leurs analogies.

La TGAO (Technologie de Groupe Assistée par Ordinateur) en est la version informatique. Elle est parfois partiellement intégrée à des progiciels de GPAO et de CAO.

Dresser une liste des ateliers que l'on peut rencontrer dans l'industrie équivaudrait à dresser une liste des métiers de toutes les industries. Cette liste correspond également aux différents procédés de fabrication. Il vaut mieux établir et consulter ces listes par domaine d'activité, de la création intellectuelle à l'usinage des métaux, en passant par le génie logiciel.

Il existe plusieurs types d'ateliers de production :

* L'atelier à postes de charge (machines ou postes de travail manuel) isolés, en anglais *jobshop*; la production y est discontinue; certains postes peuvent être regroupés en îlots.

* L'atelier a flux continu, en anglais *flowshop*, dont les postes sont mis en ligne (chaîne).

* L'atelier ou la cellule flexible, à production discontinue, dont les transferts entre postes sont automatisés.

* Les **ateliers flexibles** sont des ateliers rapidement reconfigurables en fonction des contraintes de production. Ils constituent une méthode alternative d'agencement des postes de travail quand la production en ligne, à la chaîne, devient trop complexe à organiser du fait de la diversification des produits.

L'industrie du logiciel a développé depuis les années 1970 un certain nombre d'outils informatiques permettant de mieux gérer la production sous ses divers aspects : Ordres de Fabrication (OF) - suivi des stocks - suivi des temps - gestion des coûts.

La GPAO est notamment caractérisée par un système de réapprovisionnement en produits et composants appelé calcul des besoins nets ou CBN.

Le CBN est basé sur le système MRP (Materials Requirements Plannings). II est le fruit des travaux de l'APICS (American Production and Inventory Control Society).

*MRP* : « Materials Requirements Planning », c'est un outil de gestion globale et de planification de la production de l'entreprise en flux poussé, assurant ainsi le calcul des besoins nets (besoin brut –stock) et permettant de maintenir des dates d'exigibilité correctes[13].

[13] « *La Gestion de Production Assistée par Ordinateur « GPAO » par BISMILLAH Salwa, DABACH Imane, TANANE Malika et GRINAT Khadija »*

Les fonctions de la GPAO sont communément incorporées, depuis les années 1990, aux Progiciels de gestion intégrés (ERP ou PGI) qui s'appliquent à toutes les fonctions de l'entreprise.

## 2.3. Modélisation des données :

La conception d'un système d'information n'est pas évidente car il faut réfléchir à l'ensemble de l'organisation que l'on doit mettre en place. La phase de conception nécessite des méthodes permettant de mettre en place un modèle sur lequel on va s'appuyer. La modélisation consiste à créer une représentation virtuelle d'une réalité de telle façon à faire ressortir les points auxquels on s'intéresse. Ce type de méthode est appelé *analyse*.

Dans la conception d'un système d'information, la *modélisation des données* est l'analyse et la conception de l'information contenue dans le système.

Il s'agit essentiellement d'identifier les entités logiques et les dépendances logiques entre ces entités. La modélisation des données est une représentation abstraite, dans le sens où les valeurs des données individuelles observées sont ignorées au profit de la structure, des relations, des noms et des formats des données pertinentes, même si une liste de valeurs valides est souvent enregistrée. Le modèle de données ne doit pas seulement définir la structure de données, mais aussi ce que les données veulent vraiment signifier (sémantique).

On peut modéliser un système de plusieurs manières, parmi elles l'approche cartésienne (années 70) qui s'appuie sur des méthodes d'analyse fonctionnelles et hiérarchiques ; l'approche systémique (années 80) où la modélisation des systèmes est abordée selon deux points de vues complémentaires : les données et les traitements, comme dans le cas de Merise ; l'approche objet (années 90) qui est une évolution de l'approche systémique vers une plus grande cohérence entre les objets et leurs dynamique comme la méthode OMT ; enfin on trouve des méthodes orientées objets à UML.

Dans notre cas nous allons nous intéressé seulement aux deux méthodes les plus utilisées : Merise et UML.

### 2.3.1. Merise

Depuis le début de la décennie 80, la méthode MERISE est largement utilisée dans les entreprises et dans les administrations. Les outils proposés par MERISE sont mis en œuvre par un nombre grandissant de concepteurs, d'organisateurs, d'analystes[14].

Dans la fameuse méthodologie "Merise", le processus de développement du modèle de données implique d'analyser les types de données qui auront un sens dans le système d'information, et les relations entre différentes données de ce système. Ainsi le modélisateur doit s'exprimer avec des représentations des modèles de données qui guident le processus de développement du logiciel. Dans les premières phases du projet de développement du logiciel, il faut faire ressortir l'étude d'un modèle conceptuel de données. Celui-ci peut être détaillé dans un modèle logique de données quelquefois appelé modèle organisationnel de données. Dans des phases ultérieures, ce modèle peut être traduit en un modèle physique de données.

Le but de cette méthode est d'arriver à concevoir un système d'information. La méthode MERISE est basée sur la séparation des données et des traitements à effectuer en plusieurs modèles conceptuels et physiques. La séparation des données et des traitements assure une longévité au modèle. En effet, l'agencement des données n'a pas à être souvent remanié, tandis que les traitements le sont plus fréquemment.

La méthode MERISE date de 1978-1979, et fait suite à une consultation nationale lancée en 1977 par le ministère de l'Industrie dans le but de choisir des sociétés de conseil en informatique afin de définir une méthode de conception de systèmes d'information. Les deux principales sociétés ayant mis au point cette méthode sont le CTI (Centre Technique d'Informatique) chargé de gérer le projet, et le CETE (Centre d'Etudes Techniques de l'Equipement) implanté à Aix-en-Provence.

[14] La méthode Merise Principes et outils – Hubert TARDIEU, Arnold ROCHFELD et René COLLETTI – éditions d'Organisation

### 2.3.2. Le Langage UML

**UML** (en anglais ***Unified Modeling Language***, « langage de modélisation unifié ») est un langage graphique de modélisation des données et des traitements. C'est une formalisation très aboutie et non-propriétaire de la modélisation objet utilisée en génie logiciel. Aujourd'hui, le standard industriel de modélisation objet est UML. Il se définit comme un langage de modélisation graphique et textuel destiné à comprendre et décrire des besoins, spécifier e documenter des systèmes, esquisser des architectures logicielles ; concevoir des solutions et communiquer des points de vue. Il unifie à la fois les notations et les concepts orientés objet[15].

UML est un moyen d'exprimer des modèles objet en faisant abstraction de leur implémentation, c'est-à-dire que le modèle fourni par UML est valable pour n'importe quel langage de programmation. UML est un langage qui s'appuie sur un méta-modèle, un modèle de plus haut niveau qui définit les éléments d'UML (les concepts utilisables) et leur sémantique (leur signification et leur mode d'utilisation). Le méta-modèle permet de se placer à un niveau d'abstraction supérieur car il est étudié pour être plus générique que le modèle qu'il permet de construire. Le méta-modèle d'UML en fait un langage formel possédant les caractéristiques suivantes:

- Un langage sans ambiguïtés
- Un langage universel pouvant servir de support pour tout langage orienté objet
- Un moyen de définir la structure d'un programme
- Une représentation visuelle permettant la communication entre acteurs d'un même projet
- Une notation graphique simple, compréhensible même par des non informaticiens

Le méta-modèle permet de donner des bases solides et rigoureuses à ce langage graphique, dont les représentations graphiques ne sont là que pour véhiculer des concepts de réalisation.

[15] Les cahiers du programmeur UML, modéliser un site e-commerce – Pascal Roques – édition EYROLLES

UML offre une manière élégante de représenter le système selon différentes vues complémentaires grâce aux diagrammes.

Lorsqu'une entreprise désire un logiciel, elle le réalise parfois en interne, mais le fait plus généralement réaliser par une société de services. Dans un cas comme dans l'autre il est nécessaire de définir l'ensemble des fonctionnalités que le logiciel doit posséder. Le demandeur du logiciel n'a parfois pas de compétences particulières en informatique et exprime donc ses souhaits sous forme d'un **CdCF** *(Cahier des Charges Fonctionnelle*s), c'est-à-dire un document décrivant sous forme textuelle l'ensemble des particularités que le logiciel doit posséder, les conditions qu'il doit remplir (système(s) d'exploitation visé(s)), les écueils à éviter, ainsi que les délais impartis, éventuellement des clauses sur le coût, les langages à utiliser, ...

Le CdCF est ainsi distribué à différentes sociétés de services (dans le cas d'une sous-traitance) sous forme d'un appel d'offre, auquel les sociétés vont répondre par un coût, un délai…

Lorsqu'une société obtient le marché et qu'elle décide (si elle a le choix) d'opter pour un langage orienté objet, il lui faut dans un premier temps créer un modèle (c'est là qu'intervient UML) afin:

- de présenter au client la façon de laquelle elle compte développer le logiciel,
- d'accorder tous les acteurs du projet (une application de grande envergure est généralement réalisée par modules développés par différentes équipes).

### 2.3.3. Comparaison entre Merise et UML :

| Merise | UML |
|---|---|
| **Points Communs** | |
| Entités (Objet) | Classes |
| Propriétés (Attributs) | Attributs |
| Occurrence | Objet, Instances |
| Cardinalité | Multiplicité |
| Sous-type | Sous-classe |
| Généralisation/Spécialisation | Généralisation/Spécialisation |
| Relation | Association |
| Acteurs | Acteurs |

| | |
|---|---|
| Flux | Messages |
| **Points Divergeant** | |
| Domaine informatique de gestion | Domaine informatique technique (temps réel) |
| **Concept Spécifiques** | |
| MLD | L'agrégation |
| MCT | Polymorphisme |
| MOT | Encapsulation |
| | DCO et DES |
| | Diagramme de déploiement |
| | Typage des langages objets |
| **Sur La Démarche** | |
| Découpage du processus de développement en étapes et phases de type « cascades ». L'étape suivante ne peut pas être commencée que si l'étape précédente est validée | Démarche itérative et incrémentale fondée sur la réalisation de prototypes successifs. |

## 2.4. Les Bases de Données :

Une base de données, usuellement abrégée en *BD* ou *BDD*, est un ensemble structuré et organisé permettant le stockage de grandes quantités d'informations afin d'en faciliter l'exploitation (ajout, mise à jour, recherche de données). Les deux mots clefs qui interviennent dans la définition d'une base de données sont: *structuration* (à l'aide du modèle de données) et *non répétition* (non redondance ou redondance minimale) des données[16].

Il existe plusieurs types de bases dont les bases de données orientées objet.

### 2.4.1. Oracle :

C'est un système de gestion de base de données relationnel (SGBDR) permettant d'assurer :

- La définition et la manipulation des données
- La cohérence des données
- La confidentialité des données
- L'intégrité des données
- La sauvegarde et la restauration des données

[16] http://nte-socio.univ-lyon2.fr/Marc_Grange/BDConception.htm

- ❖ La gestion des accès concurrents

*Architecture :*

Une base de données Oracle est constituée de plusieurs éléments :

- Des processus chargés en mémoire sur le serveur
- Des fichiers physiques stockés sur le serveur
- D'un espace mémoire sur le serveur appelé *SGA* (*System Global Area*).

### 2.4.2. MySQL :

#### 2.4.2.1. SQL :

SQL (**S**tructured **Q**uery **L**anguage) est le langage unique qui permet de décrire, manipuler, contrôler l'accès et interroger les bases de données relationnelles.

C'est un langage déclaratif, qui s'adresse à la fois aux utilisateurs "novices" et aux programmeurs confirmés. Il est régi par une norme (ANSI/ISO) qui assure la portabilité du langage sur différentes plates-formes aussi bien matérielles que logicielles. Une commande SQL écrite dans un environnement Windows sous ACCESS peut, souvent sans modification, être utilisée directement dans un environnement ORACLE sous Unix...[17]

#### 2.4.2.2. MySQL :

C'est un **gestionnaire de base de données libre**. Il est très utilisé dans les projets libres et dans le milieu industriel, il comprend un serveur de **bases de données relationnelles SQL** développé dans un souci de performances élevées et c'est un **multithread** (multiutilisateurs).

MySQL est un **logiciel libre** développé sous double licence en fonction de l'utilisation qui en est faite : dans un produit libre (open-source) ou dans un produit propriétaire, les bases de données sont accessibles en utilisant les **langages de programmation C, C++, C#, Delphi / Kylix, Eiffel, Java, Perl, PHP, Python, Ruby** et **Tcl.**

[17] http://nte-socio.univ-lyon2.fr/Marc_Grange/SQL.htm

SQL est un langage standardisé qui rend facile le stockage, la mise à jour et l'accès à l'information, Les principaux objectifs de **MySQL** sont la rapidité, la robustesse et la facilité d'utilisation, En Java, MySQL peut être utilisé de façon transparente avec le standard **JDO**.

La particularité de MySQL est de proposer plusieurs moteurs de stockage dans une même base de données[18].

#### 2.4.2.3. Différence entre SQL et MySQL :

MySQL c'est la base de donnée, SQL c'est le langage de programmation pour faire fonctionner cette base ou une autre (oracle par exemple)

### 2.4.3 Comparaison entre Oracle et MySQL :

| Oracle | MySQL |
|---|---|
| exploite des bases de données relationnelles depuis plus de 20 ans et certaines bases dépassent les 120 Téra octets. | exploite des bases de données relationnelles depuis moins de 2 ans et peu de bases exploitées de manière relationnelles dépassent quelque giga octets |
| est sans doute plus puissant, plus éprouvé, etc. Néanmoins, son dialecte SQL, qui a été à la pointe autrefois, est assez éloigné de la norme SQL (cela s'améliore) et surtout très pauvre. Oracle10 ne permet pas le regroupement externe, a une implémentation réduite du constructeur de ligne. | Les versions 4.1 et 5 de MySQL apportent tout cela, ainsi que des innovations particulièrement intéressantes, notamment une optimisation des regroupements. |
| est une grosse usine super puissante mais pas très innovante | est un petit labo de recherche, avec un côté un peu bricolo mais plein d'idées passionnantes ! |
| permet bien évidemment de faire beaucoup plus de choses, même s'il est effectivement en retard sur la norme SQL. | est très loin d'implémenter toutes les finesses du langage SQL et de plus il est très anti normatif ! |

### 2.4.4. PostgreSQL

PostgreSQL est un **système de gestion de base de données relationnelle** et objet (**SGBDRO**), c'est un outil **libre** disponible selon les termes d'une licence de type **BSD,** c'est un **logiciel libre** développé sous double licence en fonction de l'utilisation

[18] http://www.siteduzero.com/tutoriel-3-166641-les-transactions-avec-mysql-et-pdo.html

qui en est faite : dans un produit libre (open-source) ou dans un produit propriétaire, ce système est concurrent d'autres systèmes de gestion de base de données, qu'ils soient libres (comme **MySQL** et **FirebirdSQL**), ou propriétaires (comme **Oracle**, **Sybase**, **DB2** et **Microsoft SQL Server**).

Comme les projets libres **Apache** et Linux, PostgreSQL n'est pas contrôlé par une seule entreprise, mais est fondé sur une communauté mondiale de développeurs et d'entreprises, il peut stocker plus de types de données que les types traditionnels entiers, caractères, etc. L'utilisateur peut créer des types, des fonctions, utiliser l'héritage de type etc, il fonctionne sur toutes sortes d'**unix**. Depuis la version 8.0, PostgreSQL fonctionne également nativement sur Windows.

PostgreSQL possède de nombreuses caractéristiques :

- des interfaces graphiques pour gérer les tables ;
- des bibliothèques pour de nombreux langages (appelés *frontaux*) afin d'accéder aux enregistrements à partir de programmes écrits en :
  - **Java** (JDBC)
  - **langage C/C++**
  - ….

## 2.5. Les langages de Programmation :

### 2.5.1. Le Langage C++ :

Le C++ est la solution pour passer progressivement aux L.O.O. (langages orientés objet), sans avoir à réécrire l'existant, ainsi qu'en gardant la possibilité d'optimiser certaines parties du programme en restant *"proche de la machine"*. Son choix se fait pour l'une des raisons suivantes :

- Il sert à écrire des applications informatiques,
- Il est l'un des langages de programmation les plus utilisés et le plus innovant aujourd'hui
- C'est un langage compilé

- Il existe de très nombreux compilateurs : Visual C++ (de microsoft), C++ Builder (de Borland)…
- Avec C++Builder on peut développer des programmes Windows C++ avec plus de facilité et de rapidité

#### 2.5.1.1. Le C et le C++ :

Le langage C est un langage de programmation inventé par MM. Kernighan et Ritchie au début des années 70. Au début des années 90, Bjarne Stroustrup fait évoluer le langage vers le langage C++ en lui rajoutant notamment les notions orientées objet. Toutefois, bien que le C++ ait évolué à partir du C, et ait gardé un grand nombre de notions et de syntaxes de son «ancêtre», il s'agit de deux langages différents. Les modifications entre le C et le C++ sont nombreuses. La plus importante d'entre elles est l'introduction de la **Programmation Orientée Objet**, que l'on abrège couramment **POO**[19]**.**

#### 2.5.1.2. Avantages :

- Programmes plus faciles à écrire et à maintenir.
- Modification aisée, y compris des types de données.
- Modularité accrue (un objet bien défini resservira dans de nombreux programmes).

#### 2.5.1.3. Inconvénients :

- Programme résultant (exécutable) quelquefois un peu moins efficace (plus gros et moins rapide, mais aujourd'hui le prix de la vitesse et de la mémoire est inférieur au coût d'une optimisation de programme, on n'écrit plus grand chose en assembleur).

### 2.5.2. Le Langage Delphi :

Delphi est un environnement de programmation visuel orienté objet pour le développement rapide d'applications (RAD). En utilisant Delphi, nous pouvons créer de puissantes applications avec un minimum de programmation ; il fournit tous les

[19] http://www.siteduzero.com/tutoriel-3-11406-apprenez-a-programmer-en-c.html

outils nécessaires pour développer, tester et déployer des applications, notamment une importante bibliothèque de composants réutilisables, une suite d'outils de conception, des modèles d'applications et de fiches et des experts de programmation, ses principales caractéristiques sont :

- Possibilité de créer des applications Microsoft Windows 9.x et Windows XP très efficaces, avec un minimum de codage manuel,
- fournit tous les outils nécessaires pour développer, tester, déboguer et déployer des applications,
- Possède : - Une importante bibliothèque de composants réutilisables,
  - Un ensemble d'outils de conception,
  - Des modèles d'applications et de fiches,
  - Des experts de programmation.

**2.5.2.1. La programmation orientée objet**

a) Objectifs :

• Lier les données et les fonctions qui les manipulent afin d'éviter des accès aux données par des fonctions non autorisées.

• Obtenir une meilleure abstraction en cachant l'implémentation des techniques utilisées et en ne rendant visible que des points d'entrée. Ainsi, si l'implémentation change, le code utilisateur n'est pas affecté.

• Réutiliser l'existant dans un souci de productivité.

• Traiter les erreurs localement au niveau des objets sans que cela ne perturbe les autres parties du programme.

• Faciliter la maintenance.

b) Concepts :

Objet : En langage objet, tout est objet ! Un objet contient des données dites données membres ou attributs de l'objet et des procédures ou fonctions dites méthodes de l'objet.

Encapsulation : C'est le mécanisme consistant à lier les attributs et les méthodes au sein d'une même structure.

Héritage : L'héritage permet à un objet de récupérer les caractéristiques d'un autre objet (attributs et méthodes) et de lui ajouter de nouvelles caractéristiques.

Polymorphisme : Le polymorphisme permet d'attribuer à différents objets une méthode portant le même nom afin d'exprimer la même action, même si l'implémentation de la méthode diffère complètement.

**2.5.2.2. Connexion aux bases de données**

Delphi propose en standard des composants pour manipuler des bases de données et des tables relationnelles. Tous ces composants ont une base commune : le Borland Database Engine (BDE), un noyau stable, complet et puissant.

a) Définition :

Une base de données est une collection de données (ensemble de données) stockées sous forme de tables, où chaque table ( Table physique ou Fichier.DB ) contient des champs sous forme des colonnes où chaque colonne contient un ensemble de données (l'ensemble des données de chaque ligne de la table forme un enregistrement). On note que chaque champ da la table, possède un nom, et un type.[20]

b) Les étapes à suivre pour exploiter une BDD :

Pour exploiter une base de données à partir d'une application Delphi il faut suivre les étapes suivantes :

- Créer la première table de la base de données
- Enregistrer la table dans un dossier à votre choix.
- Refaire les démarches précédentes avec toutes les tables de la base de données et les sauvegarder dans le même dossier Tables
- Définir le chemin d'accès à la base de données (l'ensemble des tables)
- Créer un projet Delphi qui permet d'exploiter notre base de données en utilisant les composants nécessaires,

[20] http://islamonaa.com/vb/showthread.php?t=8607

- Création d'une base de données : Pour créer ou modifier une base de données, il faut passer par Module Base de Données.

c) Le Composant Table :

Le composant qui permet d'accéder à une table physique (Fichier.DB) de la base de données est le composant Table, nommée Table logique, de la classe TTable. Ce composant permet d'accéder aux données d'une Table Physique de la base de données en utilisant le moteur de bases de données Borland (BDE). Le composant Table donne un accès direct à chaque enregistrement et à chaque champ de la Table physique d'une base de données, comme il peut également travailler sur un sous-ensemble d'enregistrements d'une Table physique en utilisant des filtres.

d) DataBaseName :

C'est la propriété qui nous permet de spécifier le chemin d'accès (Alias) à une base de données

### 2.5.2.3. Paradox :

Paradox est un outil de développement très puissant mais facile à utiliser, il est construit autour du moteur de base de donnée de Borland (BDE , "Borland Database Engine"), il fonctionne parfaitement en réseau pour un petit groupe d'utilisateurs de ce fait, il est possible de conserver son interface utilisateur en Paradox tout en migrant ses données dans un SGBD ("Système de Gestion de Base de Données") de type client serveur plus robuste aux pannes et plus performant en cas de connexions très nombreuses. Paradox peut attaquer nativement ou via ODBC (interface d'accès standardisée au base de données) les principales bases de données existantes : Oracle, MS-SQLServeur, Interbase , MySQL... Cette ouverture de Paradox, exceptionnelle, en fait un excellent "couteau suisse" pour l'administrateur de bases de données confronté à des systèmes différents. Il sait tout faire ou presque : extraire des données, les

présenter, les manipuler, les convertir, les importer etc... Les temps de réponse sont eux aussi exceptionnels.[21]

### 2.5.3. Le Langage Java :

Java est un langage de programmation moderne développé par **Sun Microsystems**. Il ne faut surtout pas le confondre avec Javascript (petit langage de scripts utilisé sur les sites web), car Java n'a rien à voir. Une de ses plus grandes forces est son excellente portabilité : une fois votre programme créé, il fonctionnera automatiquement sous Windows, Mac, Linux, etc[22].

C'est un **langage objet** ressemblant au **langage C++**, ses caractéristiques sont :

- Simple
- Distribué
- Orienté objet
- Interprété
- Robuste
- Indépendant de l'architecture
- Portable

### 2.5.4. Comparaison entre Java et C++

- Java est très proche du langage C++ étant donné qu'il a quasiment la même syntaxe,
- Java est plus simple que le langage C++ bien qu'il s'en inspire
- Les caractéristiques critiques du langage C++ ont été supprimées. Cela comprend :

  - Les pointeurs
  - La surcharge d'opérateurs

[21] http://www.clairinfo.fr/fichiers/Tut_pdox_1.htm
[22] http://www.siteduzero.com/tutoriel-3-10601-programmation-en-java.html

- L'héritage multiple
- La libération de mémoire est transparente pour l'utilisateur (il n'est plus nécessaire de créer de destructeurs)
- Une meilleure gestion des erreurs
- Les chaînes et les tableaux sont désormais des objets faisant partie intégrante du langage

- Java est beaucoup moins rapide que le langage C++, il perd en rapidité ce qu'il gagne en portabilité...

**Conclusion :**

Le principe tiré de ces deux chapitres est celui du **système d'information** (SI) par le biais des caractéristiques suivantes :

- Est un ensemble de méthodes et moyens recueillant, contrôlant, mémorisant et distribuant les informations nécessaires à l'exercice de l'activité en tous points de l'organisation.

- Correspond à l'ensemble des objets gérés par des utilisateurs et des informaticiens pour décrire et fonctionner un certain nombre d'applications de gestion.

- Un langage de communication dans l'organisation, construit consciemment pour représenter d'une manière fiable et objective, rapidement et économiquement, certains aspects de ses activités passées ou à venir.

- Les phrases et les mots de ce langage sont des données et des messages dont le sens vient des règles de leur élaboration par des hommes ou par des machines.

Une méthode de conception de systèmes d'information est nécessaire pour formuler le problème informationnel et maîtriser la résolution pour construire un SI cohérent, pertinent, complet, fiable, flexible et adaptatif. Toute méthode met en œuvre 4 composants indissociables :

- Des modèles : Ensemble de concepts et règles pour utiliser ces derniers destinés soit à expliquer et représenter les phénomènes organisationnels, soit à expliquer et à représenter les éléments qui composent le SI et leurs relations.

- Un langage : Ensemble de constructions qui permettent de décrire formellement les images du SI élaborées aux différents stades du processus de conception, éventuellement en faisant appel à des méthodes.

- Une démarche : C'est un processus opératoire par lequel s'effectue le travail de modélisation, de description, d'évaluation et de réalisation du SI.

- Des outils : Ce sont les outils logiciels supportant la démarche (outils de documentation, d'évaluation, de simulation, d'aide à la conception ou à la réalisation).

C'est donc de cette manière là que nous allons procéder pour développer notre logiciel de GPAO et pour cela il n'y a pas mieux qu'un exemple démonstratif pour mieux comprendre la partie pratique de la thèse.

# Chapitre 3
# Partie modélisation du logiciel

## Introduction

Un ordinateur est une machine capable d'effectuer très rapidement des opérations arithmétiques ou logiques sur des données afin de fournir des résultats. Cependant, cette machine ne dispose d'aucune connaissance : pour obtenir qu'elle réalise les opérations souhaitées, il faut lui fournir des instructions qu'elle suivra ensuite à la lettre. En général, l'utilisateur dispose d'un «algorithme» qui décrit en détail comment on passe, au moyen d'un nombre fini d'opérations, des données aux résultats. L'algorithme (recette) est rédigé en langage humain, bien trop riche et complexe pour être compris par la machine. L'utilisateur devra tout d'abord transposer son algorithme en un jargon simpliste (un «langage de programmation ») pour obtenir un «programme» (on dit aussi «code» ou «code source») équivalent. Ce programme utilisateur sera traduit à son tour en une suite d'instructions primitives effectivement compréhensibles et exécutables par l'ordinateur. Cette traduction est réalisée par un programme spécialisé, le «compilateur». A chaque langage de programmation correspond un compilateur, qui porte le même nom. Le programme source est rangé dans un fichier situé sur le disque dur ou sur une disquette. Sauf instructions spéciales, le compilateur ne connaît que le contenu de ce fichier source. Le programme en langage machine (repéré par le suffixe .exe sous DOS et Windows) est lui aussi rangé dans un fichier (en général dans le même dossier que la source). Il peut alors être exécuté autant de fois que l'utilisateur le souhaite.

Le but du travail est le développement d'un logiciel qui possède un accès rapide à toutes les fonctions, une mise en œuvre rapide et une base de données compacte.

Notre logiciel que nous allons nommer « ProdIG » devra être doté d'une très grande richesse fonctionnelle notamment :

- Gestion des données techniques
- Gestion des stocks et des achats

- Gestion commerciale
- Gestion de la distribution
- …

## 3.1. Les étapes de conception du logiciel ProdIG:

Le recours à la modélisation est une pratique indispensable au développement logiciel, car un modèle sert à anticiper les résultats du codage. Le développement de logiciels de grande envergure passe par des méthodes d'analyse selon lesquelles la conception est réfléchie et effectuée. La méthode MERISE est la plus connue d'entre elles, elle permet de créer une structure de base de données en créant ainsi un modèle conceptuel de données (CDM : Conceptuel Data Model). Le MCD a pour but d'écrire de façon formelle les données qui seront utilisées par le système d'information. Il s'agit donc d'une représentation des données généralement sous forme de schéma, facilement compréhensible, permettant de décrire le système d'information à l'aide d'entités[23].

Nous avons la possibilité, à partir du schéma créé, de générer le PDM (Physical Data Model) ou modèle physique des données. Le modèle physique consiste à implémenter le modèle dans le SGBD, c'est-à-dire le traduire dans un langage de définition de données. Le langage généralement utilisé pour ce type d'opération est le SQL.

Nous pouvons par la suite exporter dans une multitude de formats la modélisation ainsi créée, ce mode permet de générer les instructions SQL de création des tables dans l'environnement souhaité.

[23] http://www.commentcamarche.net/contents/merise/mcd.php3

## 3.2. Modélisation de la gestion de production :

Notre projet 'Gestion de production' aura comme principe la création d'une base de données de gestion de la production d'une entreprise.

Le projet est une petite GPAO (Gestion de la Production assisté par Ordinateur). Nous modéliserons la base de données et l'exporterons sur une base de données courante pour pouvoir ensuite (en théorie) l'exploiter.

Le choix d'exportation sera choisi de façon toute logique, il s'agira de MySQL, base de données relationnelle gratuite très orientée 'Web' puisque fonctionnant nativement avec les langages de programmation PHP, ASPx, Perl et C#.

Le projet se doit d'être simple, rapide à effectuer et employant le maximum de fonctions du logiciel.

En premier lieu, le cahier des charges de l'application décrira les contraintes relatives à l'environnement GPAO et nous permettra de définir :

- les entités présentes dans la base de données
- les attributs pour chaque entité
- les liens entre chaque entité (relation)
- les cardinalités découlant des contraintes

## 3.3. Application de la méthode MERISE sur le système de production :

MERISE est une méthode de conception et de développement des systèmes d'information. Elle a été conçue pour couvrir les besoins tant des administrations que des entreprises[24].

L'intérêt de choisir Merise provient de ce que cette méthode s'enracine dans une conception systémique de l'organisation. Au lieu – comme dans l'approche traditionnelle de l'informatique de gestion - d'isoler des procédures pour en faire des applications et d'analyser ces procédures pour les automatiser, elle conçoit le système

[24] http://www.univ-ubs.fr/valoria/antoine/Enseignement/MSI_GL_UML/Merise.pdf

d'information comme un « objet artificiel » intermédiaire entre le système opérant (l'usine..), et le système de pilotage (la direction…), et comme un objet qui doit contenir une représentation de la dynamique de ces deux derniers systèmes[25].

La méthode merise constitue un point de départ évident pour la conception d'une application utilisant un SGBDR (Système de Gestion de Bases de Données Relationnelles). Au delà d'une simple méthodologie, Merise propose des contraintes (règles) a appliquer à son schéma de base de données afin de répondre à un certain niveau de qualité fonctionnelle, c'est ce qu'on appelle les formes normales.

### 3.3.1. Principe :

La méthode Merise consiste à concevoir un Modèle Conceptuel de Donnée (MCD), le transposer en Modèle Logique de Données Relationnelles (MLDR), puis à générer le Modèle Physique correspondant (MPD)

### 3.3.2. Modélisation du système de production :

Voilà comment se déroule une analyse dans le modèle merise d'un système de production :

1- Il faut tout d'abord transformer l'existant en modèle simple. Nous dégagerons les entités présentes dans le modèle à analyser, donc généralement dans une entreprise des clients passent une ou plusieurs commandes à un ou plusieurs fournisseurs. Une commande peut contenir un ou plusieurs articles.

2- A partir de cette phrase, nous pouvons dégager plusieurs entités :

| Client | Fournisseur | Commande | Article |
|---|---|---|---|
| | | | |

[25] La méthode Merise Principes et outils – Hubert TARDIEU, Arnold ROCHFELD et René COLLETTI – éditions d'Organisation

Une entité est la représentation d'un élément matériel ou immatériel ayant un rôle dans le système que l'on désire décrire.

Nous appellons **classe d'entité** un ensemble composé d'entités de même type, c'est-à-dire dont la définition est la même. Le classement des entités au sein d'une classe s'appelle *classification* (ou *abstraction*). Une entité est une *instanciation* de la classe. Chaque entité est composée de propriétés, données élémentaires permettant de la décrire.

3- Il faut ensuite pouvoir dégager les propriétés pour chaque entité, c'est-à-dire par quoi est caractérisée chaque entité :

- Un client est caractérisé par : son numéro de client, son nom, son adresse, son téléphone, son fax.
- Un fournisseur est caractérisé par : son numéro de fournisseur, son nom, son adresse, son téléphone, son fax.
- Une commande est caractérisé par : son numéro de commande, le nom du client, les achats et le solde.
- Un article est caractérisé par : son numéro d'article, sa désignation, sa disponibilité, son prix.
- Ce qui nous donne :

| Client |
| --- |
| Code_client |
| Nom |
| Adresse |
| Téléphone |
| Num_Compte |

| Fournisseur |
| --- |
| Code_fourn |
| Nom |
| Adresse |
| Téléphone |
| Fax |

| Commande |
| --- |
| Num_comman |
| Date_comman |
| Date_Livraison |
| Date_Recep |

| Produit |
| --- |
| Code_article |
| Désignation |
| Disponibilité |

4- Il faut ensuite identifier de façon unique chaque occurrence.

En effet, dans une entité (physiquement une table de données), il faut pouvoir distinguer les occurrences (chaque enregistrement) de façon unique : s'il existe deux clients dont le nom est 'Client01', le nom ne pourra pas être une 'clé' ou identifiant pertinent pour distinguer deux enregistrements (occurrences). On choisira donc un attribut qui identifie de nature unique l'occurrence, comme un code unique (code de client).

| Client |
| --- |
| Code_client |
| Nom |
| Adresse |
| Téléphone |
| Num_Compte |

| Fournisseur |
| --- |
| Code_fourn |
| Nom |
| Adresse |
| Téléphone |
| Fax |

| Commande |
| --- |
| Num_comman |
| Date_comman |
| Date_Livraison |
| Date_Recep |

| Produit |
| --- |
| Code_article |
| Désignation |
| Disponibilité |

**Figure 3-1 : entités de la Gestion de Production**

5- Etablir les relations entre les différentes entités et identifier les cardinalités.

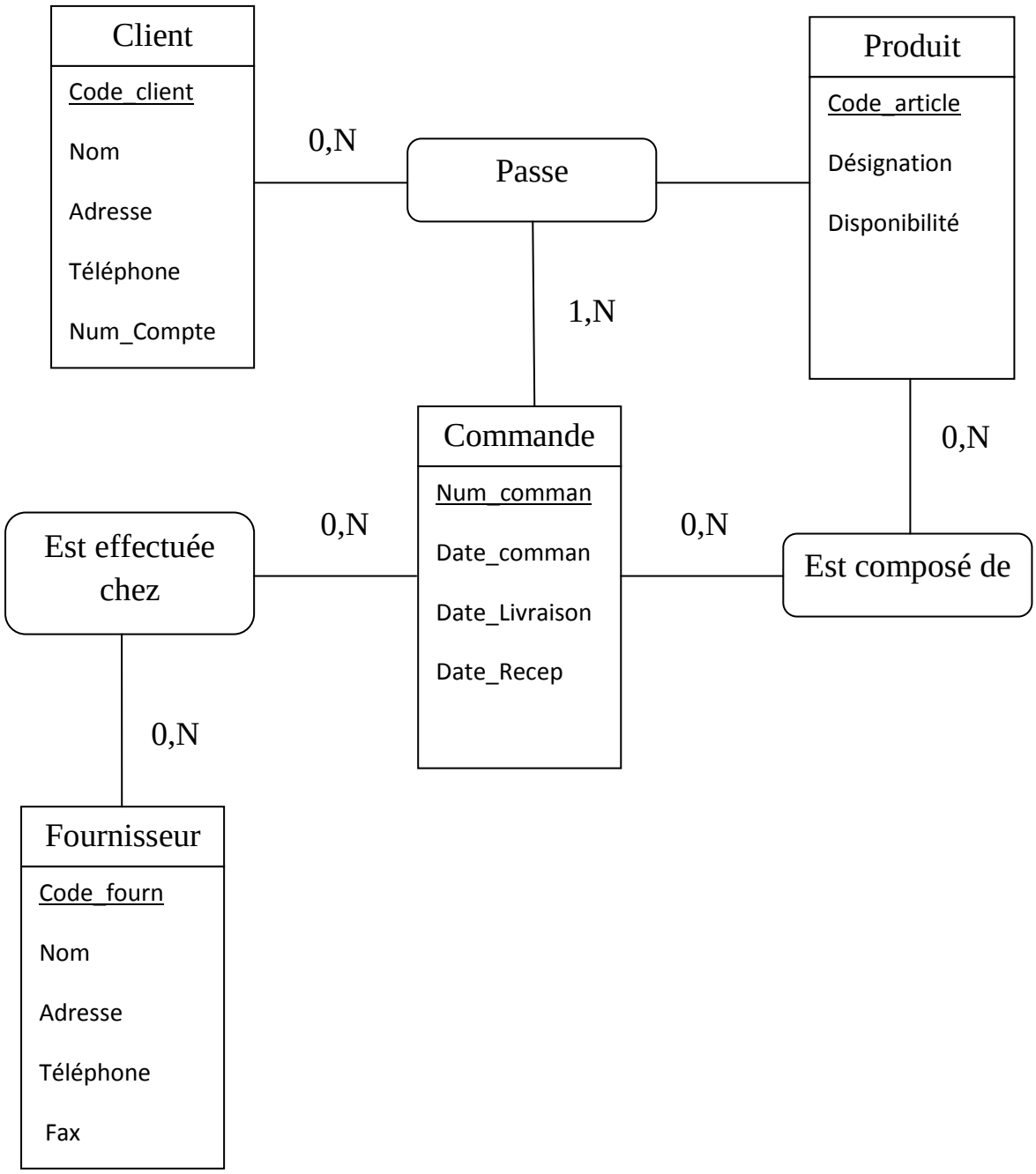



**Figure 3-2 : MCD Général de la Gestion de Production**

Les cardinalités sont dictées par ce que l'on nomme des contraintes. Les contraintes sont des règles qui lient les objets (entités) entre eux et dictent leurs relations. Ici, dans cet exemple, voici les contraintes qui s'établissent :

- Un client peut passer 0 ou une infinité de commande

- Une commande est passée par 1 ou une infinité de client

- Une commande comprend 0 ou plusieurs articles

- Un article compose 0 ou une infinité de commandes.

- Une commande est effectuée chez 0 ou une infinité de fournisseurs (0 car l'entreprise peut être son propre fournisseur : création de produits fini à partir de composant, commande interne…)

- Un fournisseur prépare 0 ou une infinité de commande.

*Remarque :* Nous considérons que nous pouvons passer une 'commande vide' ou 'commande à zéro' pour des besoins interne, c'est pour cela que nous pouvons avoir une commande comprenant 0 article (pas d'article). Le chiffre '0' dans d'une cardinalité exprime l'absence de contrainte forte entre les deux entités. '1' signifie qu'une entité ne peut exister indépendamment d'une autre, 'N' signifie 2 ou plus (une infinité).

Nous avons réalisé ici la modélisation d'un système existant. A partir de là, notre modèle devra être validé ou faire l'objet d'une étude approfondie si l'on désire ajouter des entités (éléments d'information). Bien entendu, on pourrait débattre sur ce modèle et son fonctionnement pouvant être différent, ce n'est qu'une solution parmi d'autres.

### 3.4. Programmation sous Delphi :

Pour la partie programmation nous avons opté pour le langage de programmation Delphi5 pour sa rapidité et simplicité de développement qui sont dues à une conception visuelle de l'application. Delphi propose un ensemble très complet de composants visuels prêts à l'emploi incluant la quasi-totalité des composants Windows (boutons, boîtes de dialogue, menus, barres d'outils…) ainsi que des experts permettant de créer facilement divers types d'applications et de librairies.

L'EDI (environnement de développement intégré) de Delphi est divisé en 3 parties :

- Fenêtre principale
- Inspecteur d'objet
- Espace de travail et éditeur de code

a) Fenêtre principale :


**Figure3-3 : Fenêtre principale de Delphi**

Nous y retrouvons les barres d'outils et la palette de composants. C'est à partir de cette fenêtre qu'on choisit et dépose les composants sur notre fiche dans l'espace de travail.

b) L'inspecteur objet :

L'inspecteur d'objet permet de:

- visualiser
- modifier
  - Les propriétés et événements des composants en mode création.
  - Les propriétés et évènements varient d'un composant à l'autre.
  - Les propriétés correspondent aux caractéristiques, aspects, comportements d'un objet.

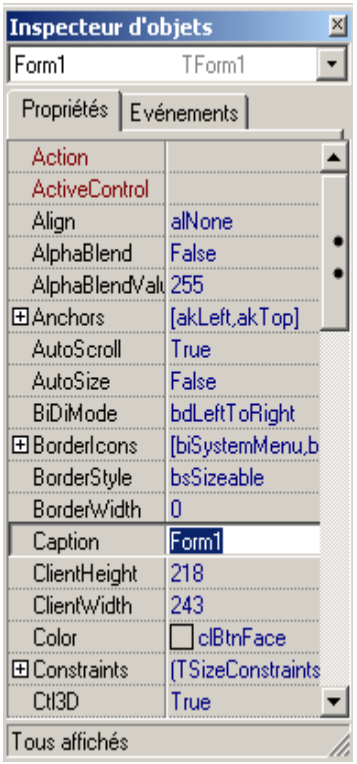



**Figure 3-4 : Inspecteur Objet**

c) L'Espace de travail et éditeur de code :

L'espace de travail sert à déposer les composants qui constituent notre programme. C'est dans cette fenêtre que nous créons la partie graphique de notre programme.

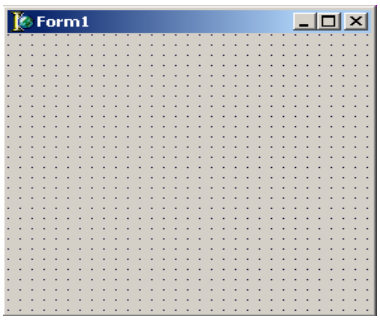



**Figure 3-5 : Espace de travail**

Nous écrivons tout notre code dans cet éditeur. C'est la fenêtre à laquelle le programmeur passera le plus clair de son temps.

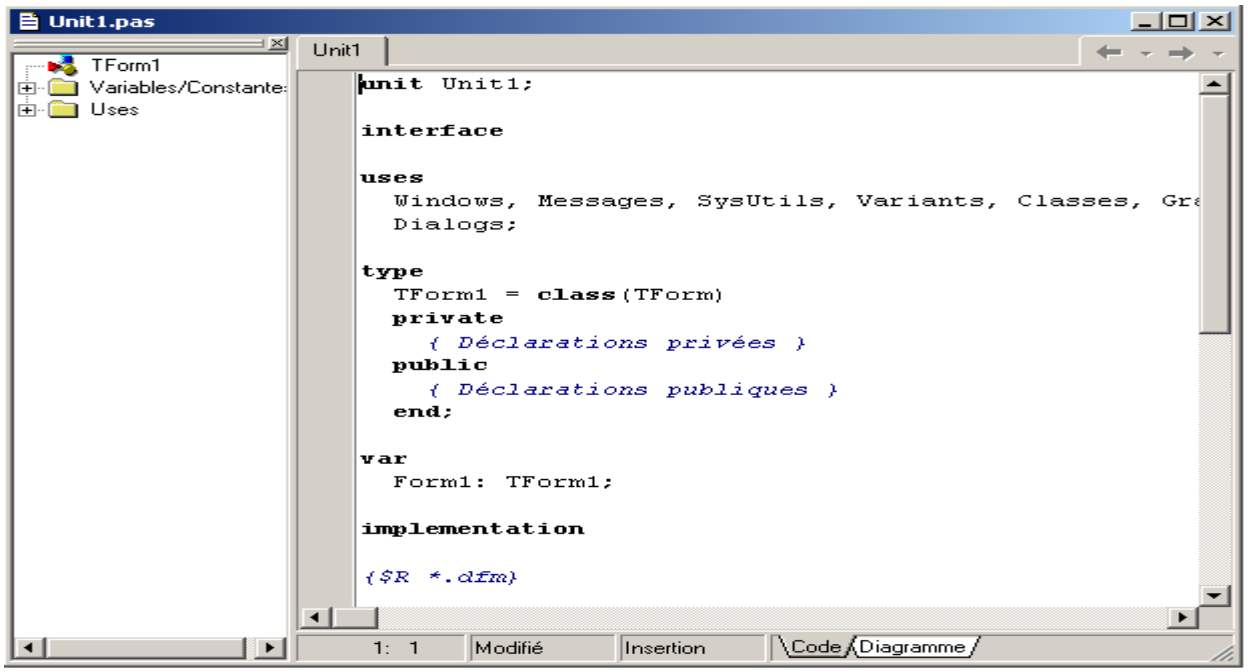



**Figure 3-6 : Editeur de code**

Delphi génère automatiquement le nom des composants en leur ajoutant un chiffre. L'EDI peut s'avérer complexe au novice, mais il deviendra rapidement indispensable. Il regroupe tout ce qu'un programmeur a besoin. Il n'est plus nécessaire de disposer de plusieurs programmes séparés pour faire ce dont on a besoin.

## 3.5. Programmation du logiciel :

*1ère étape :*

La 1ère étape concerne la création d'une base de données avec le ***Module de base de données***, et une application Delphi qui nous permet de gérer cette base de données.

- Pour créer une Base de données **Table** nous lançons le module base de données qui se trouve dans la liste de programme installés ou dans le menu *Outils* de la barre de menu Delphi.


- Dans le menu Fichier de Module base de donnés nous choisissons Nouveau puis Table Pour créer une nouvelle Table. Le composant qui permet d'accéder à une table physique (Fichier.DB) de la base de données est le composant Table. Ce composant permet d'accéder aux données d'une Table Physique de la base de données en utilisant le moteur de bases de données Borland (BDE). Le composant Table donne un accès direct à chaque enregistrement et à chaque champ de la Table physique d'une base de données, comme il peut également travailler sur un sous-ensemble d'enregistrements d'une Table physique en utilisant des filtres.


**Figure 3-7 : Module Base de données**

- Une petit fenêtre s'affiche pour choisir le type de table à créer (Paradox 7, Paradox 5.0 pour Windows, Paradox 4, Paradox 3.0, Visual dBASE, dBASE pour Windows ...etc), nous choisissons **Paradox7**.


Paradox est un outil de développement très puissant mais facile à utiliser, il est construit autour du moteur de base de donnée de Borland (BDE , "Borland Database Engine"), il fonctionne parfaitement en réseau pour un petit groupe d'utilisateurs de ce fait, il est possible de conserver son interface utilisateur en Paradox tout en migrant ses données dans un SGBD ("Système de Gestion de Base de Données") de type client serveur plus robuste aux pannes et plus performant en cas de connexions très nombreuses. Il sait tout faire ou presque : extraire des données, les présenter, les manipuler, les convertir, les importer etc... Les temps de réponse sont eux aussi exceptionnels.[26]

[26] http://www.clairinfo.fr/fichiers/Tut_pdox_1.htm

Les principales bases de données existantes : Oracle, MS-SQLServeur, Interbase, MySQL...

- Une fenêtre de saisie s'affiche pour éditer les champs de la table.

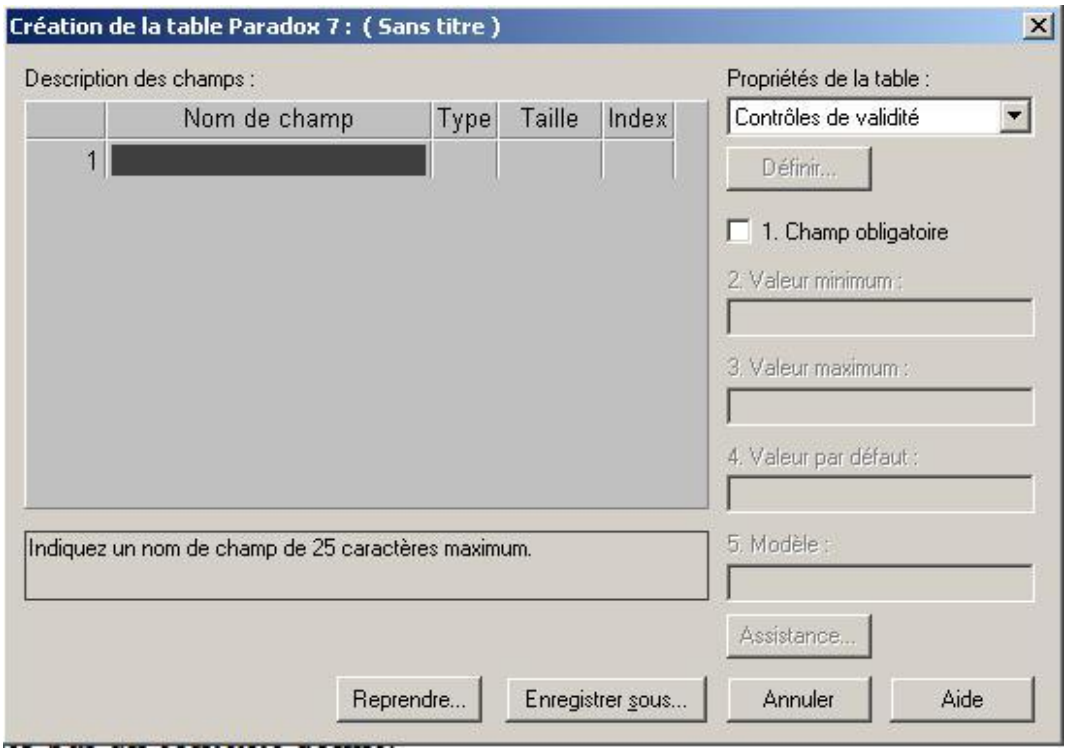
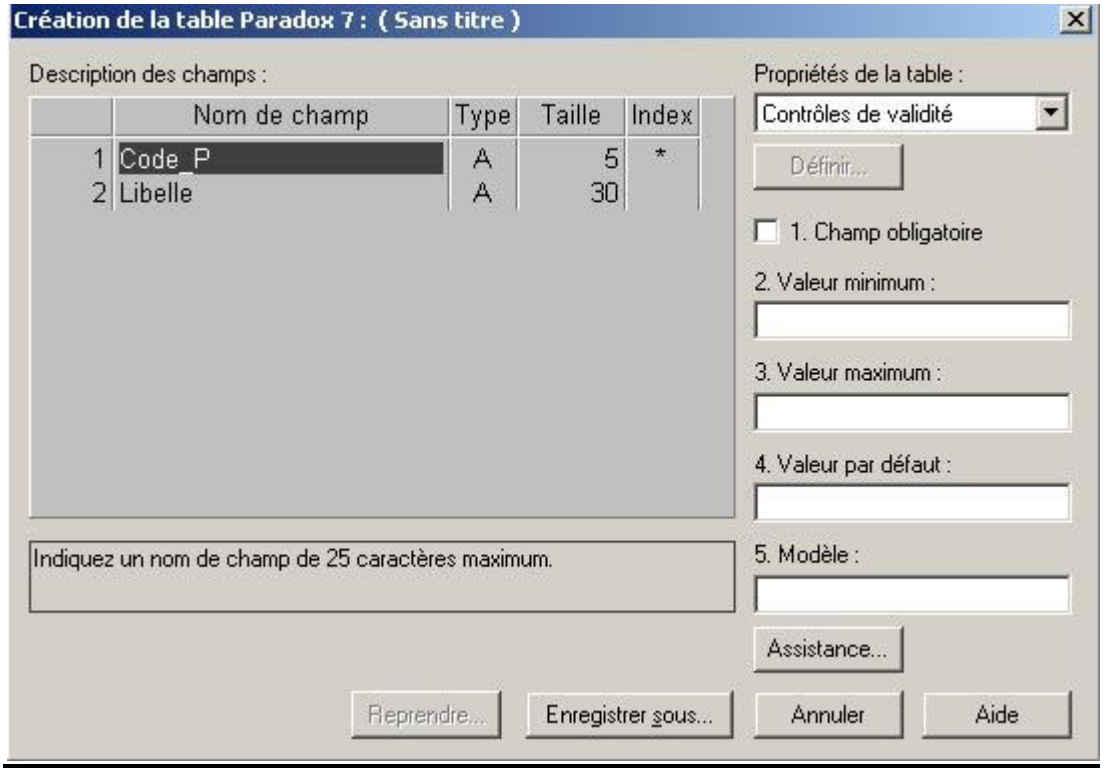



**Figure 3-8 : Création de table sous Paradox 7**

Pour l'entité Produit ; notre table doit avoir 2 Champs (Code_P,Libellé);

1. **Nom de champ:** Code_P , **Type:** A, **taille :**5 , **Index** : *
2. **Nom de champ:** Libellé , **Type:** A, **taille :**30 , **Index :**

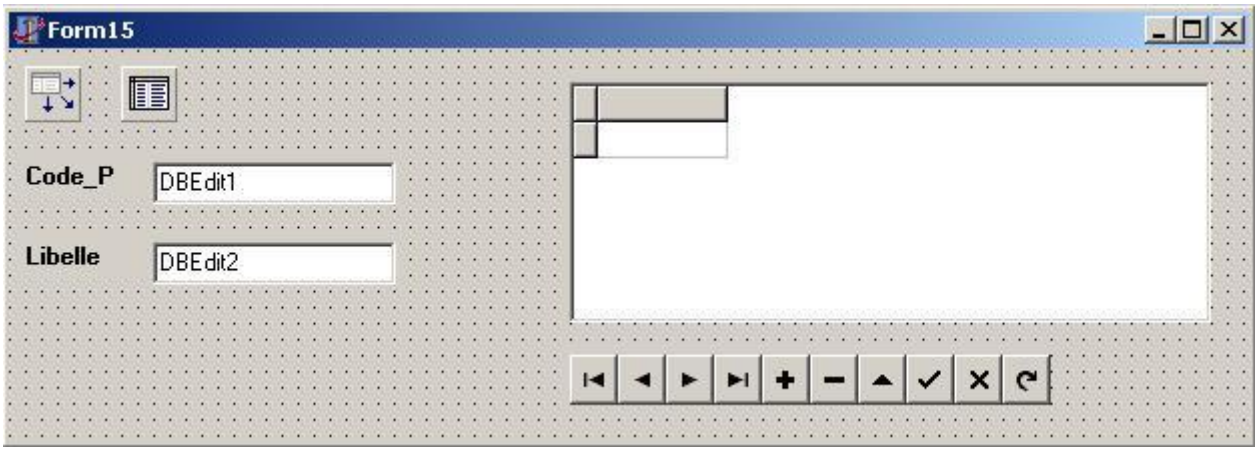
Nous remplissons le tableau pour avoir la forme suivante :




*2ème étape :*

- Après la création de toutes les tables avec le module base de données, nous allons maintenant créer une application Delphi Qui va exploiter les données de la table et pour ce faire on procède comme suit :

- Lancer Delphi5
- Dans le la barre de menu nous choisissons *Fichier\Nouveau\Application* Pour créer une nouvelle application.
- Nous placerons les composants suivants sur la fiche de conception: *une Table, une DataSource, une DBGrid, un DBNavigateur, deux Label, deux DBEdit*
- Nous renommons la Propriété *Caption* des *Label* l'un après l'autre : Code_P, Libellé et redimensionnez les DBEdit afin d'avoir la forme suivante :




- La table que nous avons créé auparavant, nommée Article.db est placée dans le dossier C:\Documents and Settings\Pablo\Mes documents\Logiciel GPAO\Tables

- Dans la propriété *DataSet* du composant *DataSource1* nous cliquons sur la petite flèche et choisissez *Table1*
- Dans la propriété *DatabaseName* du composant *Table1* nous insérons le chemin de la *table* a exploiter, dans notre exemple c'est ***C:\Documents and Settings\Pablo\Mes documents\Logiciel GPAO\Tables***.
- Dans la propriété *TableName* de la *Table1* nous cliquerons sur la petite flèche et choisissez ***Liste.db*** . (Dans la propriété *TableName* d'une table s'affiche toutes les tables existantes dans le dossier inséré dans la propriété *DatabaseName* pour choisir une d'elle.
- Nous allons maintenant ouvrir la *Table1* on mettant sa propriété *Active* a *True*.
- Nous metterons *DataSource1* dans la propriété DataSource du DBGrid1 et de DBNavigator1ainsi que dans tout les *DBEdit*.
- Nous cliquons sur la petite flèche qui se trouve dans la propriété DataField de DBEdit1 et choisissez Code_P, faisons la même chose pour *DBEdit2* et choisissons Libellé.
- Nous utiliserons les Boutons de ***DBNavigator*** pour *Ajouter, modifier, déplacer, ou supprimer* nos enregistrements.

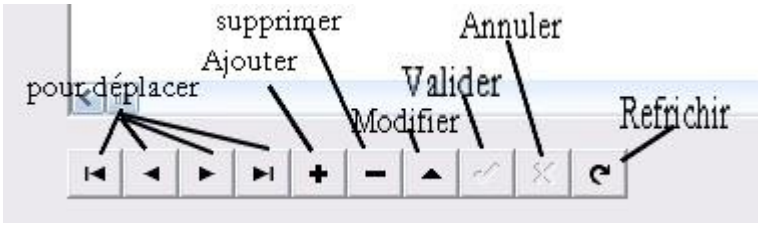



**Figure 3-9 : palette de navigation de la table**

*3ème étape :*

- Création des autres table en appliquant la même procédure que la 1ère et 2ème étape et pour simplifier le travail nous les regrouperons dans un DataModule ainsi nous aurons un accès facile aux différentes tables. Le nombre total des tables utilisées dans notre logiciel est de 19 tables dont 04 temporaires (pour plus de détail voir Annexe B):

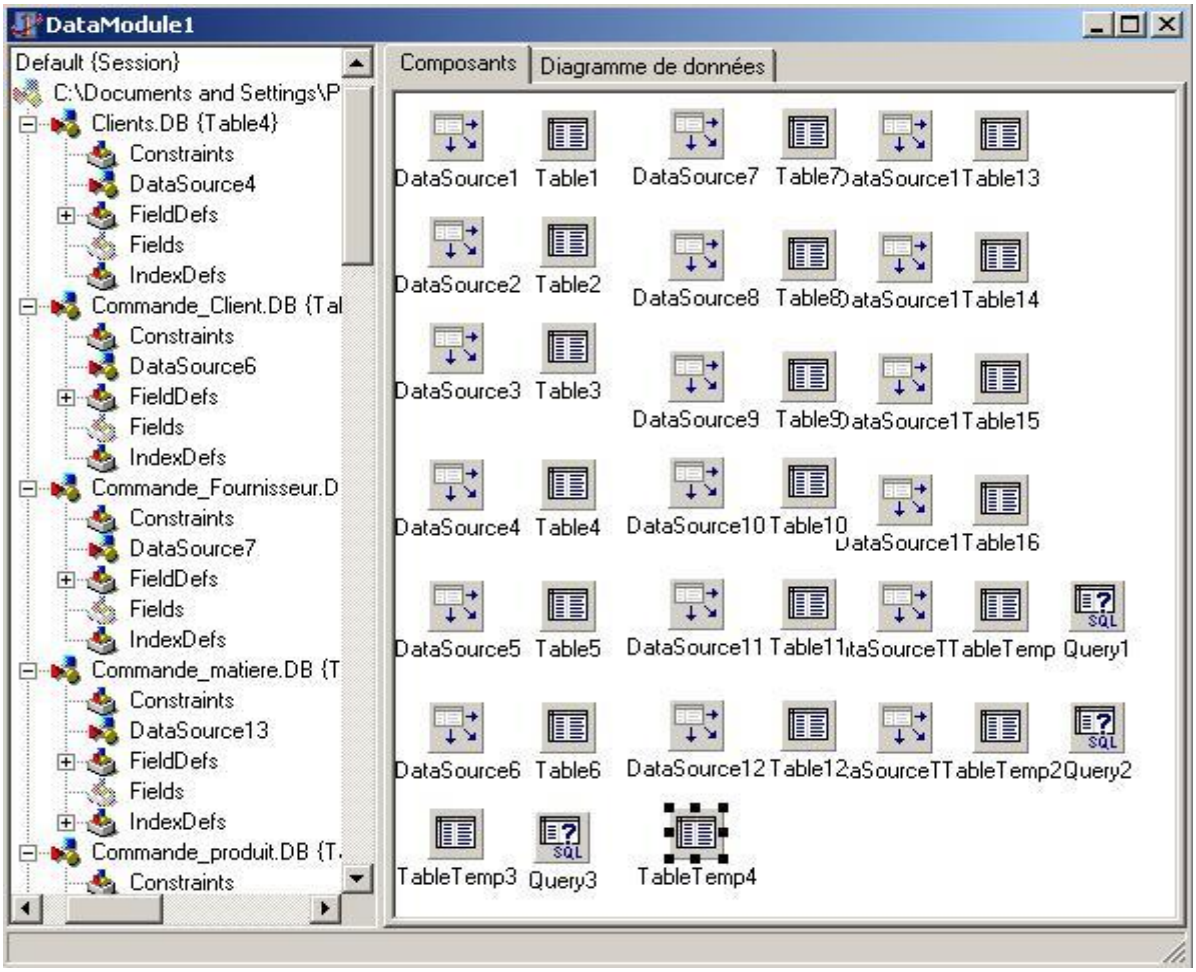



**Figure 3-10 : Module de base de données**

- La Table 1 représente l'entité Produit ;
- La Table 2 représente l'entité Produit Semi-Fini;
- La Table 3 représente l'entité Matière Première ;
- La Table 4 représente l'entité Client ;
- La Table 5 représente l'entité Fournisseur ;
- La Table 6 représente l'entité Commande Client ;
- La Table 7 représente l'entité Commande Fournisseur ;
- La Table 8 représente l'entité Ville ;
- La Table 9 représente l'entité Pays ;
- La Table 10 représente l'entité Poste de charge ;
- La Table 11 représente l'entité Mot de Passe ;
- La Table 12 représente l'entité Commande Produit ;
- La Table 13 représente l'entité Commande matière première ;
- La Table 14 représente l'entité Liste des Produits Semi-Finis ;
- La Table 15 représente l'entité Liste des matières Premières ;
- Les 04 tables restantes sont des tables temporaires : Par définition, une table temporaire (TEMPORARY TABLE) cesse d'exister à partir du moment où elle est supprimée à la fin de la session de connexion.

*Dernière étape :*

La dernière étape est la programmation de toutes les relations qui lient les différentes entités ou tables en utilisant le langage de programmation delphi5(pour plus de détail voir le code source sur Annexe A).

## Conclusion :

Le recours à la modélisation est depuis longtemps une pratique indispensable au développement logiciel, car un modèle sert à anticiper les résultats du codage ainsi la conception logicielle dépend d'une organisation et d'un contexte. Même si le concepteur se trouve le plus souvent face aux problèmes qu'il doit résoudre seul, il est nécessaire de maintenir une liaison avec l'extérieur. Une méthode de conception doit donc prendre en compte la répartition des tâches et permettre l'utilisation des entités externes par un contexte donné.

A travers ce chapitre nous somme parvenu à créer un MCD de notre logiciel à partir duquel nous avons procéder au développement de ProdIG et dont le résultat final est représenté dans le dernier chapitre à l'aide d'un exemple pratique conçu spécialement pour l'apprentissage des principaux fondements de la GPAO.

# *Chapitre 4*
# *Présentation de ProdIG*

## Introduction :

Pour mieux comprendre le principe de ProdIG nous allons étudier un exemple simple de gestion de production et qui peut être définit comme un exemple type. Cette partie, basée sur un exemple très simple, se propose de nous faire progressivement découvrir les fonctions de base du logiciel de GPAO. L'objectif est donc une présentation unifiée des grandes fonctions du logiciel, sans considérer explicitement à ce niveau les nombreux détails et paramètres de ces fonctions. La mission de cet exemple est donc d'illustrer la manière dont les principales fonctions d'une GPAO sont articulées entre elles.

Durant cet exemple, un modèle simplifié d'un système de production sera construit et saisi, et la gestion des flux sera assurée sur un certain horizon temps. L'énoncé se divise en sessions de travail qui constituent les différentes étapes de la mise au point et de l'exploitation d'une GPAO. Les premières sessions consistent essentiellement à saisir les données techniques décrivant l'usine considérée. Ensuite, ces données seront exploitées afin de piloter l'activité de production.

## 4.1. Informations générales sur le logiciel :

Ce chapitre dresse l'inventaire des fonctionnalités purement **GPAO** contenues dans le logiciel :

- Gestion des Données
- Gestion des Achats
- Gestion des Ventes
- Gestion des Besoins et Approvisionnements
- Gestion clients
- Gestion de production
- Gestion des fournisseurs
- Gestion du stock

Toutes les fonctionnalités du logiciel tendent vers l'objectif de la satisfaction individuelle de chacun des clients. ProdIG a été pensé comme un outil global de gestion au service de la relation client

## 4.2. Le problème de gestion de production considéré :

Nous prenons le cas d'une Usine qui fabrique des bibliothèques en bois, elle propose dans son catalogue deux modèles différents : une bibliothèque d'une largeur d'un mètre et une bibliothèque d'une largeur de deux mètres. Le problème consiste à planifier et organiser la production de ces bibliothèques pour les premiers temps. Pour ce faire, on dispose bien entendu de l'ensemble des informations nécessaires, qui seront saisies progressivement par l'utilisateur du logiciel.

### 4.2.1. Le produit considéré :

Une bibliothèque de ce type se compose de 3 panneaux extérieurs de soutien (deux petits sur les côtés et un grand à l'arrière), de 4 profilés permettant l'assemblage des éléments, de 3 étagères et de 12 taquets métalliques (4 par étagère).

### 4.2.2. L'usine considérée :

L'usine est essentiellement constituée de machines à bois permettant la réalisation des profilés et des renforts, de scies pour la découpe des panneaux et d'ateliers de montage.

### 4.2.3. Saisie des informations :

Dans un premiers temps seront saisies les informations qui décrivent l'ensemble des objets dont l'approvisionnement et la fabrication devront être gérés : les articles. Les articles correspondent d'une part aux différents composants (et composés) représentés au début de l'exercice et, d'autre part, aux matières utilisées pour fabriquer ces composants. Le résultat après saisie des informations sera donné comme suit :

| Code | Libellé | Unité | Magasin | Délai d'obtention |
|---|---|---|---|---|
| **Articles Fabriqués** | | | | |
| ARM100 | Armoire de 100 cm | UN | MAG | 03 |
| ARM200 | Armoire de 200 cm | UN | MAG | 03 |
| ETA100 | Etagère de 100 cm | UN | MAG | 03 |
| ETA200 | Etagère de 200 cm | UN | MAG | 03 |
| PANA100 | Panneau arrière 100 cm | UN | MAG | 03 |
| PANA200 | Panneau arrière 200 cm | UN | MAG | 03 |
| PANLAT | Panneau latéral | UN | MAG | 03 |
| PLET100 | Panneau d'étagère de 100 cm | UN | MAG | 03 |
| PLET200 | Panneau d'étagère de 200 cm | UN | MAG | 03 |
| PROFIL | Profilé | UN | MAG | 03 |
| **Articles Achetés** | | | | |
| BOIS002 | Bois 2mm (2m x 2m) | UN | MAG | 10 |
| BOIS010 | Bois 10mm (2m x 2m) | UN | MAG | 10 |
| LIN40 | Linteau bois (4m) | UN | MAG | 10 |
| TAQ000 | Taquet métallique | UN | MAG | 10 |

### 4.2.4. Informations commerciales pour les articles :

Les articles achetés sont approvisionnés via des fournisseurs qu'il faut préciser. Pour chaque fournisseur, il est nécessaire de décrire les articles qu'il livre et selon quelles conditions.

## 4.3. Gestion des nomenclatures :

Les articles saisis précédemment sont en fait reliés entre eux : les articles ARM100 et ARM200 sont fabriqués par assemblage des autres articles pour cela sont saisies les informations qui décrivent les liens existant entre les articles (composant, composé, sous-ensemble,...), sous la forme de nomenclatures de fabrication (schéma1).

## 4.4. Postes de charge et gammes de fabrication :

Les paramètres saisis sont les informations qui décrivent les moyens de production (scies, machines à bois,...), décrits à ce niveau sous la forme de postes de

charge, et les procédures de fabrication, saisies en tant que gammes de fabrication. Pour pouvoir créer les gammes de fabrication, il faut, bien entendu, avoir préalablement créé les postes de charge sur lesquels se déroulent les opérations.

### 4.4.1. Création des postes de charge :

Les postes de charge définissent et caractérisent les moyens de production que l'on veut gérer en termes de charge de travail.

### 4.4.2. Création des gammes :

Il est maintenant possible de définir les procédures de production, dénommé gammes de fabrication, après avoir entré le code et le libellé de la gamme, on introduit les caractéristiques des phases successives du processus. Pour chacune des phases, il faut entrer :

- Le numéro de phase,
- Le libellé,
- Le code du poste de charge,
- …..

### 4.4.3. Création des liaisons Articles-Gammes :

Après avoir créé toutes les gammes, il faut bien entendu spécifier pour chaque article quelle gamme est utilisée pour réaliser sa production.

## 4.5. Stockage et mouvement de stock :

Les données techniques principales ont été introduites. Avant de démarrer la fabrication en usine, on présente les mécanismes fondamentaux de saisie des mouvements d'articles entre les différents stocks. La maîtrise de ces mécanismes est bien entendu nécessaire pour la maîtrise d'un flux matière, associé à l'activité de production. Les stocks initiaux de l'entreprise sont définis préalablement par l’entreprise.

## 4.6. Entrée des commandes clients :

Nous allons maintenant introduire la demande qui doit être servie par l'usine. Nous allons entrer les commandes des clients puis planifier la production, estimer les charges et déterminer les commandes de matières premières à passer aux fournisseurs. Pour pouvoir saisir des commandes, il faut que les caractéristiques des clients aient été renseignées. Nous pouvons ensuite enregistrer des commandes pour les articles fabriqués.

## 4.7. Présentation de ProdIG :

### 4.7.1. Saisie des Produits :

La définition des produits dans notre logiciel permet de gérer tous types de produits (gamme, nomenclature, matière et consommations,...).

ProdIG gère aussi bien une fabrication à la commande qu'à la série.

Les produits sont gérés dans toutes leurs composantes :

- Variantes
- Gamme de fabrication
- Matière
- Nomenclature
- Caractéristiques générales
- Stock

Les références produit/client et produit/fournisseur sont gérées pour permettre de travailler directement à partir des codes, désignations et numéros de plan de l'interlocuteur.

Nous appelons la fenêtre Produits, par le menu Données et l'option Produits. Saisir alors dans le tableau ci-dessous les données relatives à Chaque produit.

Les produits à leur tour sont composés d'un ou plusieurs produits semi-finis et/ou de matières premières.

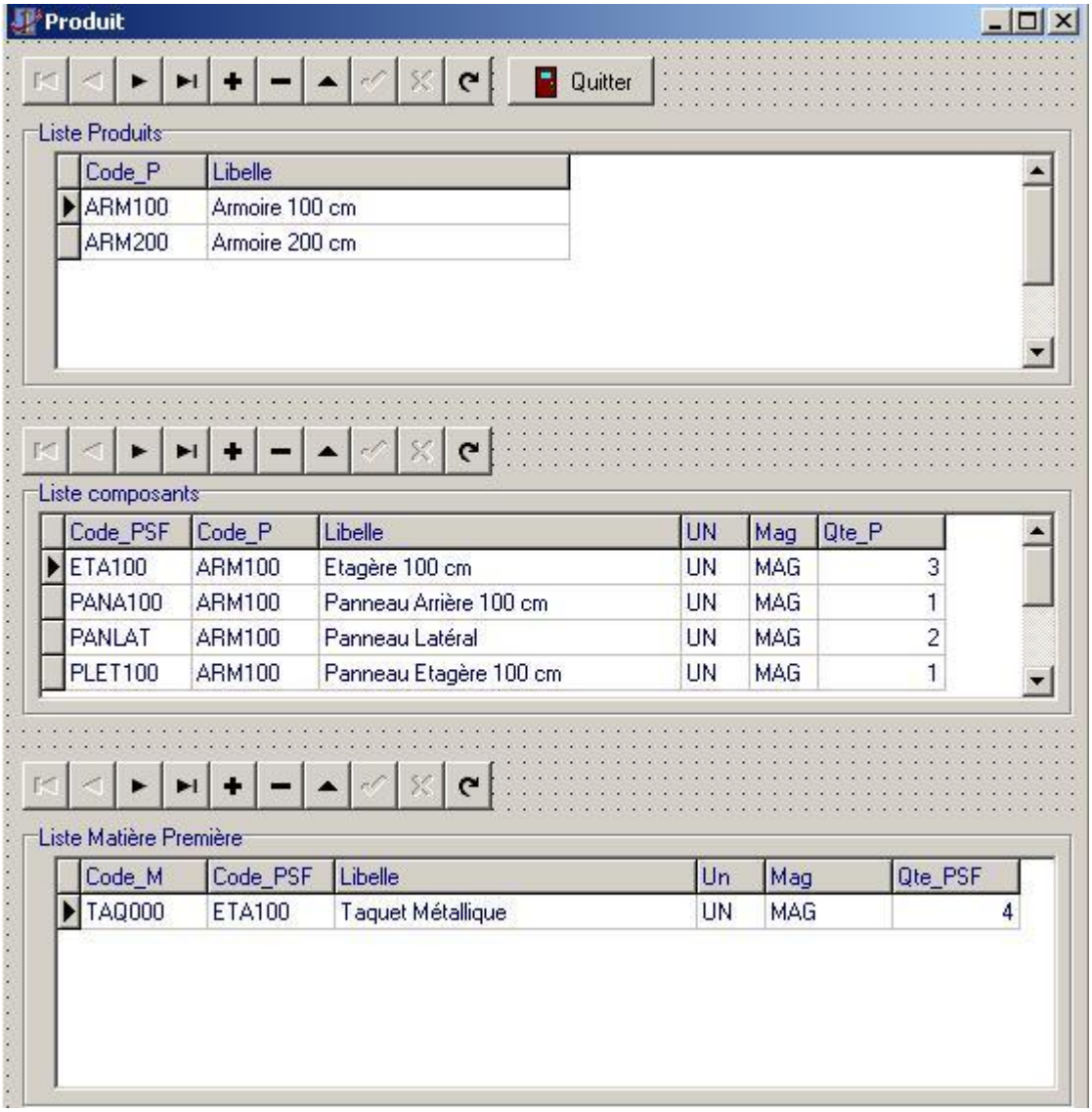



**Figure 4-1 : Fenêtre Produits**

Nous pouvons également introduire de nouvelles données concernant les produits semi-finis ou les matières premières en appelant respectivement la fenêtre par le menu données et l'option Produits Semi-finis ou l'option Matières Premières.

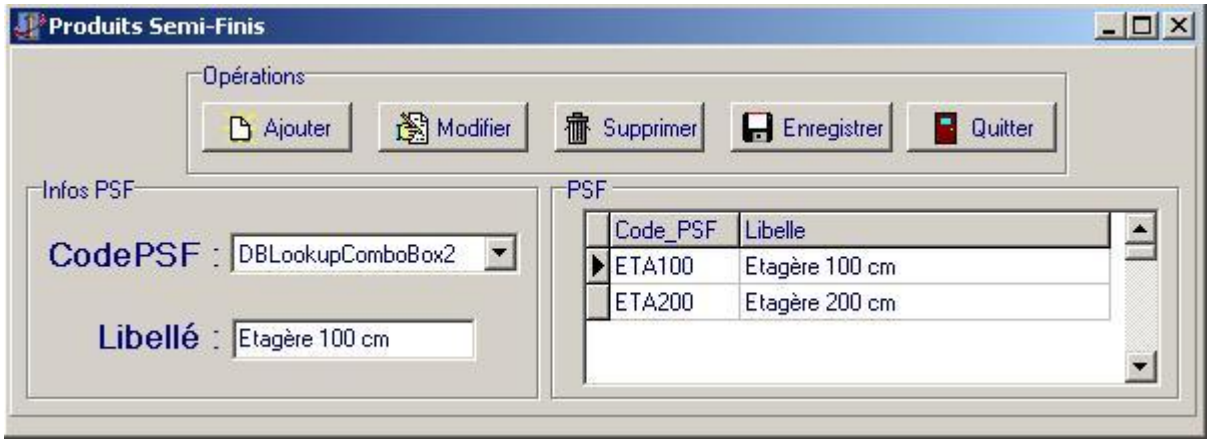



**Figure 4-2 : Fenêtre Produits Semi-finis**

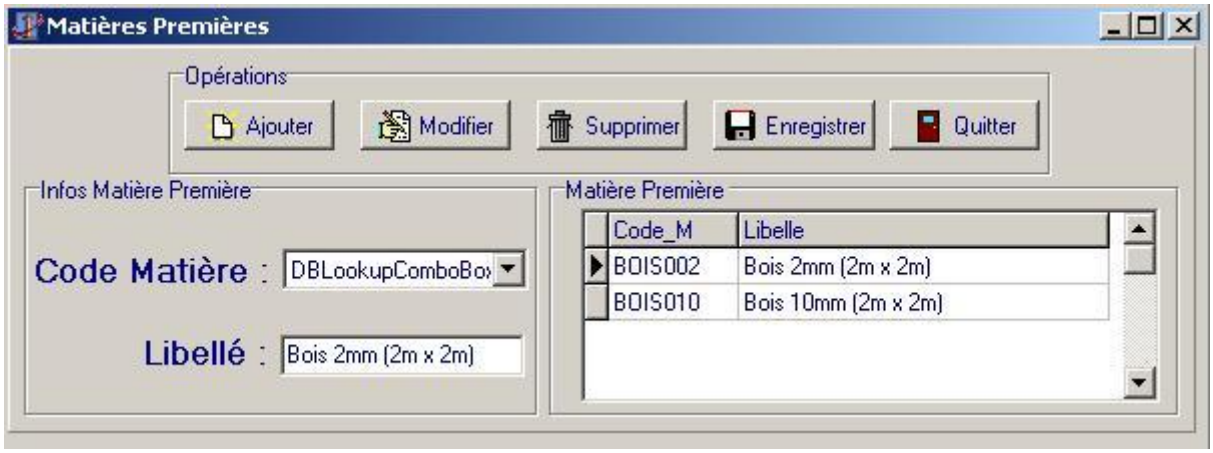



**Figure 4-3 : Fenêtre Matières Premières**

### 4.7.2. Stockage et mouvements de stock :

ProdIG permet un contrôle et un suivi du stock, soit par produit, soit globalement. Les saisies d'inventaires sont simples et rapides. A tout moment, une correction manuelle peut être apportée. Chaque mouvement est enregistré dans la base de données.

Nous allons définir les stocks initiaux de l'entreprise, ce qui constitue une des conditions préalables à la gestion des flux.

Le tableau suivant décrit les quantités des différents articles disponibles dans le magasin. Nous activons la fenêtre de saisie de ces informations, via le menu Stocks, option Mouvement de Stock. Ensuite, pour le magasin, nous entrons pour chaque article la quantité comptée telle qu'elle figure dans le tableau ci-dessous et ensuite nous validons par Enregistrer.

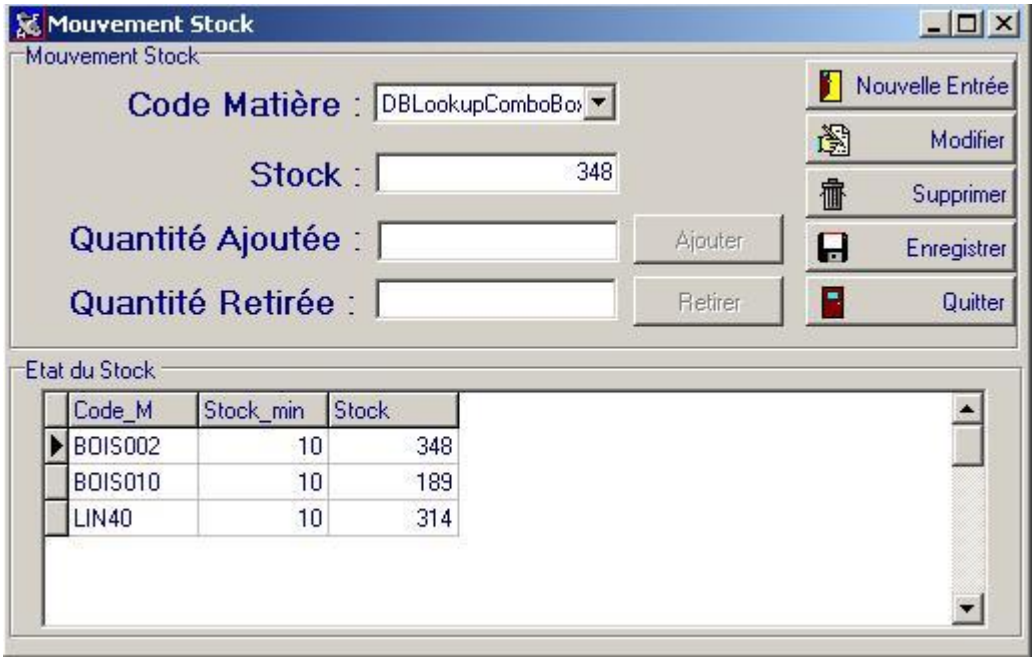



**Figure 4-4 : Fenêtre Mouvement Stock**

### 4.7.3. Entrée des commandes clients :

Nous allons maintenant introduire la demande qui doit être servie par l'usine. Nous allons entrer les commandes des clients puis planifier la production, estimer les charges et de déterminer les commandes de matières premières à passer aux fournisseurs.

Pour pouvoir saisir des commandes, il faut que les caractéristiques des clients aient été renseignées. Nous pouvons ensuite enregistrer des commandes pour les articles fabriqués.

*4.7.3.1. Saisie des données clients :*

ProdIG offre une véritable gestion de la relation clients, en intégrant une gestion des actions à réaliser. La fiche client comporte toutes les informations nécessaires à la gestion :

- Informations techniques
- Informations comptables
- Informations commerciales
- Adresses
- Téléphones
- etc...

L'organisation prédéfinie des données est complétée par l'utilisation de catégories et de caractéristiques libres, entièrement gérables par l'utilisateur. Par ce moyen, l'utilisateur organise librement ses données, de la façon la plus pertinente pour son organisation. Ses besoins de gestion pourront évoluer librement.

Nous activons la fenêtre des clients (menu Ventes, option Clients). Il suffit d'entrer le code du client et son nom ainsi que les renseignements nécessaires sur les zones appropriées.

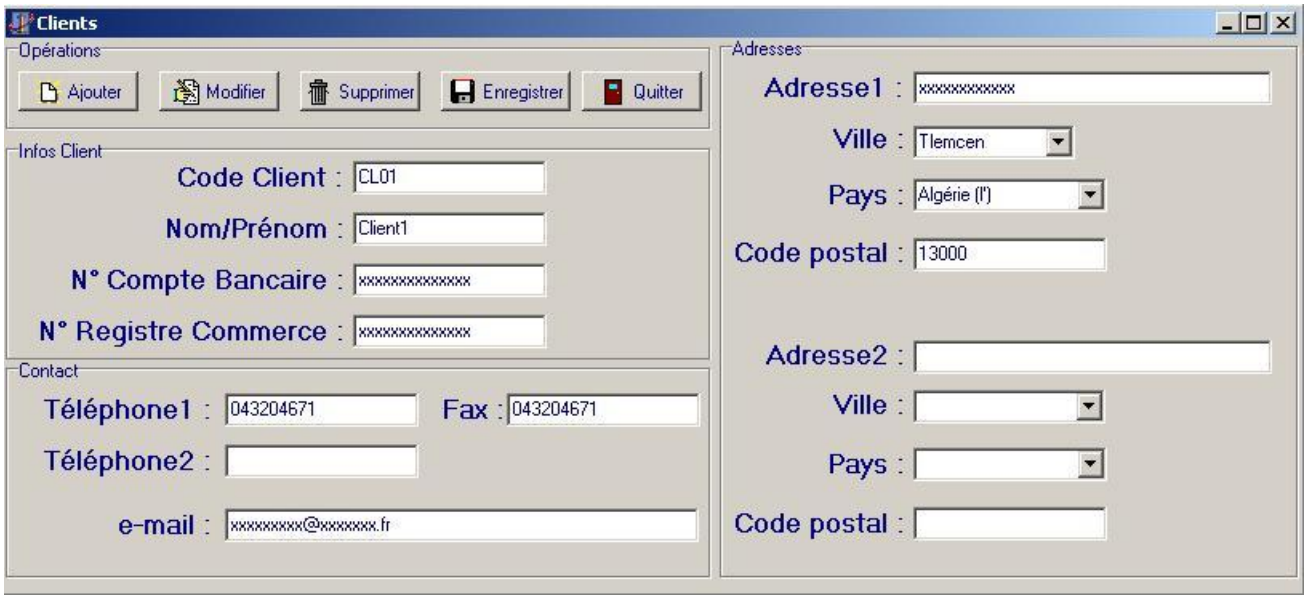



**Figure 4-5 : Fenêtre Clients**

*4.7.3.2. Saisie des commandes :*

Nous activons la fenêtre de gestion des commandes clients (menu Ventes, option Commandes clients) et saisissons les commandes décrites par les clients.

Pour entrer une nouvelle commande, nous cliquerons sur le bouton Ajouter ; le premier numéro libre est affecté. Nous entrons ensuite le code du client et la date de livraison demandée puis entrons le numéro de commande, le code de l'article et la quantité puis validerons en cliquant sur Lancer OF (Ordre de Fabrication), la quantité commandée sera automatiquement retirée du stock. Quand la commande sera effectuée nous cliquerons sur Enregistrer puis sur Quitter.

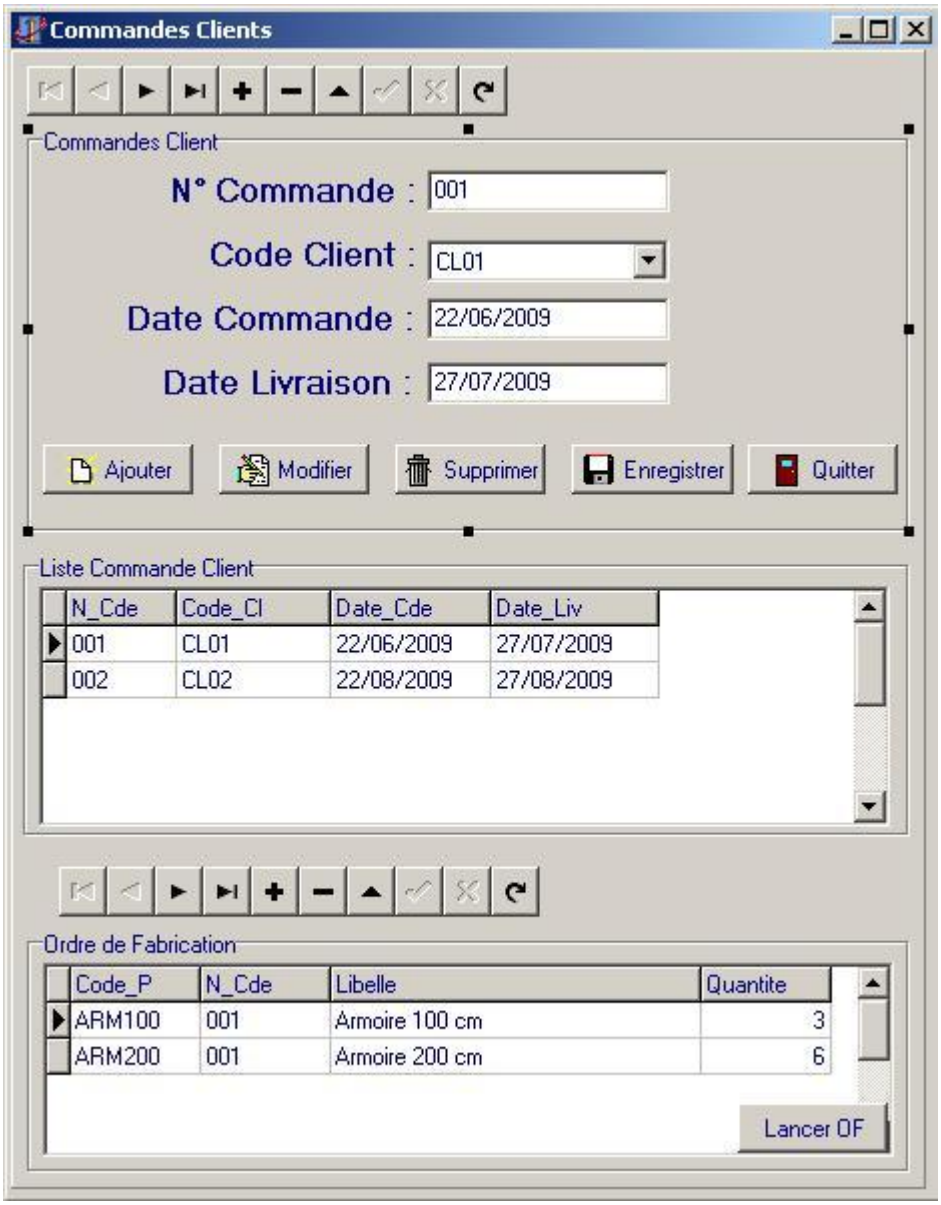



**Figure 4-6 : Fenêtre Commandes Clients**

A la réception de la commande client, nous transformons directement l'offre que le client a faite en commande. Nous modifions éventuellement la commande à enregistrer si des modifications sont advenues.

Après validation de la commande, nous imprimons directement l'Accusé de Réception de Commande. Le logiciel est destiné à simplifier les tâches quotidiennes de son utilisateur.

### 4.7.4. Entrée des commandes fournisseurs :

Nous souhaitons maintenant passer toutes les commandes pour les approvisionnements de matières premières.

Examinons les ordres de fabrication suggérés (menu Achats, option Ordres d'achat suggérés) et transmettons au service Achats qui doit les convertir en commandes aux fournisseurs.

Les ordres d'achat sont transformés en une commande ferme chez le fournisseur, via Achat, option Commande fournisseur. Nous cliquons sur Ajouter pour introduire cette nouvelle commande. Nous saisirons le Fournisseur approprié dans la zone Code fournisseur, la date de passation de la commande et la date de livraison officielle de cette dernière ainsi que la quantité nécessaire pour chaque matière première. Nous cliquerons alors sur Lancer OA (Ordre d'Achat), la quantité commandée sera ajoutée automatiquement dans le stock; quand la commande sera effectuée nous cliquons sur Enregistrer puis sur Quitter.


**Figure 4-7 : Fenêtre Commandes Fournisseurs**

La liste des fournisseurs doit être disponible auparavant via Achat, option Fournisseurs.

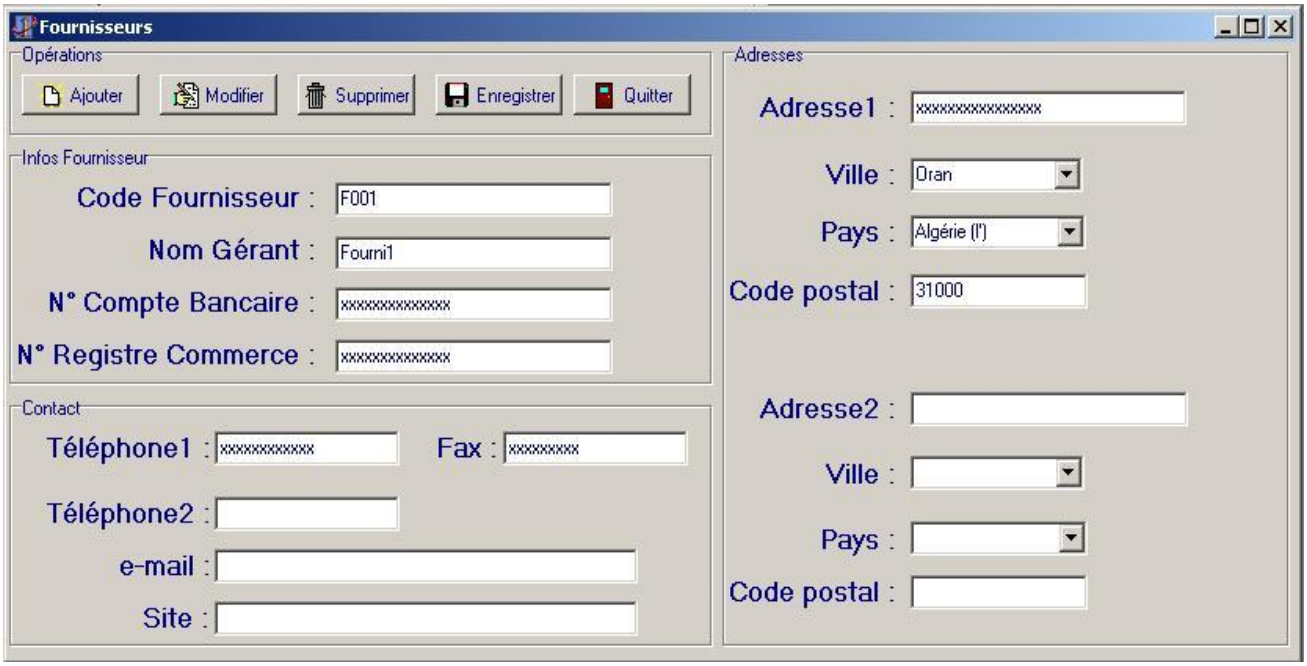



**Figure 4-8 : Fenêtre Fournisseurs**

## 4.8. Résultats et observations :

### *4.8.1. Les points forts :*

- De la prospection des clients jusqu'à la vente de produits finis, en passant par la gestion des stocks et des fournisseurs, par la production, ProdIG apporte une aide intelligente et efficace afin d'obtenir une gestion claire et optimale.

- ProdIG présent s'adapte à toutes les entreprises grâce à ses paramètres configurables ce qui nous permet de personnaliser complètement le logiciel, ainsi nous pouvons l'adapter totalement aux habitudes de travail ou de gestion de l'entreprise. La généralisation du multifenêtrage apporte une grande souplesse à l'utilisateur.

- Il nous permet de gérer un très grand nombre d'informations sur les clients, les fournisseurs, et ainsi que les personnes avec qui nous somme en contact.

- Il offre une meilleure réactivité pour répondre aux besoins Clients et une disponibilité immédiate des informations.

- Il calcul automatiquement le nombre de produit semi-fini et le nombre de matières premières nécessaire pour la fabrication du produit et les retires immédiatement du stock une fois l'ordre de fabrication est lancé.
- Il prévient l'utilisateur quand le stock de sécurité est atteint afin de pouvoir se réapprovisionner en lançant l'ordre d'achat de matières premières

### ***4.8.2. Les points faibles :***

- La gestion des temps et des opérations ainsi que la gestion du coût horaire et du temps unitaire par opération ne sont pas incluses dans ProdIG ce qui le rend moins autonome.
- ProdIG est incapable d'analyser les performances de notre outil de production ainsi nous ne pouvons pas connaître les causes d'arrêt de nos moyens de production et avoir en temps réel un état de notre système.
- Absence d'accès à distance à partir d'une simple connexion intranet ou internet.

### ***Conclusion :***

Tout comme un conducteur de véhicules automobiles ayant un tableau de bord lui permettant de réaliser un trajet en ayant des informations sur l'état de son véhicule, la mise en œuvre d'une GPAO permet de disposer d'un tableau de bord de l'activité de l'entreprise avec des informations en temps réel. La GPAO, (appelée à terme ERP : planification des ressources de l'entreprise), prend aujourd'hui en compte la gestion d'un ensemble de modules : commercial, devis, lancement, ordonnancement, suivi, facturation, interface avec comptabilité, analyse statistique de l'activité…

Un logiciel pour qu'il remplisse sa fonction doit absolument être simple d'utilisation. Nous n'oublions pas que l'entreprise n'est pas composée uniquement

d'informaticiens et n'a pas le temps pour s'adapter à un nouveau logiciel. C'est pourquoi que nous avons conçu une interface utilisateur claire, identique dans chaque traitement de l'information, et aussi respectueuse de la logique de l'activité de l'utilisateur ce qui facilite sa prise en main et le rend opérationnel immédiatement.

ProdIG n'est pas complet car il n'y a pas deux entreprises qui gèrent de la même façon. L'une voudra gérer ses temps de fabrications, bien connaître ses prix de revient, faire de la planification, faire du bilan d'affaires, mettre en place une démarche qualité, mesurer les tâches et les activités et gérer d'autres fonctions dans l'entreprise indispensable au bon fonctionnement de l'ensemble. C'est pourquoi notre logiciel en comparaison avec d'autres logiciels du marché présente un nombre important de failles car l'entreprise durant toute la durée de son activité ne trouvera pas tous les modules nécessaires à ses besoins qui peuvent évoluer dans le temps. Ceci est dû au manque d'expérience dans le domaine informatique (programmation, développement) ainsi à l'importance et à la taille du travail car la réalisation d'un projet de telle envergure nécessité une équipe de deux personne minimum, l'une spécialisée dans la recherche bibliographique et structurelle du projet l'autre dans la programmation et le développement du programme.

# Conclusion Générale

La gestion de production s'approprie donc divers mécanismes dont la GPAO n'est qu'un seul. Cette dernière remplie un ensemble de tâches, relatives à la gestion de production, par l'automatisation des fonctions de l'entreprise et l'optimisation des ressources existantes. Cependant, cette assistance par ordinateur de la production révèle des limites en matière de son utilisation et des contraintes à l'en contre de son bon fonctionnement. En effet, parmi les données de la base d'un progiciel de GPAO, figure la « gamme de fabrication » qui implique le calcul des charges lors de l'ordonnancement des ordres de fabrication. Ce calcul ne peut, par contre, permettre la connaissance des modes opératoires de chaque phase, chose nécessaire pour le pilotage d'une unité productive. En plus, la GPAO ne fournie qu'une approximation des « délais d'obtention ». Ces derniers étant présentés comme la somme des temps opératoires et des temps inter-opératoires, deviennent étroitement relatifs aux seuls temps opératoires dans la logique de la GPAO. Ces exemples sont parmi ceux qui traduisent les limites de l'utilisation de la GPAO ce qui renvoie automatiquement à l'importance de la complémentarité entre la GPAO et les autres astuces de le Gestion de production[27].

La conception du logiciel se résume donc à une activité intellectuelle s'effectuant dans le cadre d'un projet découpé en phases et visant à exprimer une solution, sous la forme d'un document normalisé.

Le but de cette thèse était tout d'abord la réalisation d'une étude bibliographique approfondis sur la gestion de production, ensuite le développement d'un logiciel capable d'assurer les fonctions de base de la gestion de production et permet d'effectuer toutes les opérations de gestion liées à la fabrication.

[27] Gestion de production : Connaissances fondamentales de P.HOMMEL

Il est à noter toutefois que le schéma type n'existe pas, chaque entreprise étant différente elle se doit de trouver la solution qui lui convient. Elle n'est pas obligée d'informatiser l'ensemble de son activité, bien souvent dans les industries graphiques, la première étape passe par la mise en place du devis : le paramétrage du logiciel même à ce niveau, amène l'entreprise à bien formaliser son processus.

Le développement d'Internet risque de modifier un certain nombre de paramètres, en particulier pour les petites entreprises qui auront la possibilité de se connecter sur un site de gestion de la production à distance. Cette connexion permettra la diminution des coûts globaux de mise en place et de ce fait, l'accès à la GPAO sera automatisé.

Pour couvrir l'ensemble des fonctions de notre logiciel et pour optimiser les performances de notre outil de production il faut prévoir l'insertion de nouveaux modules qui gèrent :

- Les prix de revient
- La planification, avec
  - calcul entièrement automatique (pas de positionnement manuel)
  - calcul a capacité finie (tenant compte de la capacité journalière, des horaires, congés, RTT de chacun)
  - simulation de fabrication d'un produit dans l'encours pour obtenir en quelques secondes une date prévisionnelle de sortie.
  - étendue maximum : 1 an
- La maintenance,
- Les interventions,
- La qualité
- …

En conclusion, les ERP sont devenus des outils incontournables, ils permettront aux entreprises d'améliorer leur fonctionnement interne mais ils ne doivent pas être uniquement compris et mis en œuvre comme des outils informatiques, ils doivent générer :

- de nouvelles méthodes de travail plus productives ;
- de nouvelles organisations plus réactives et efficaces ;
- de nouveaux profits pour les entreprises et leurs salariés.

**Listes des figures :**

***Bibliographie :***


- Organisation et gestion de la production – Luc Boyer – Série Tables E.O.
- « *La Gestion de Production Assistée par Ordinateur « GPAO » par BISMILLAH Salwa, DABACH Imane, TANANE Malika et GRINAT Khadija »*
- La méthode Merise Principes et outils – Hubert TARDIEU, Arnold ROCHFELD et René COLLETTI – éditions d'Organisation
- Les cahiers du programmeur UML, modéliser un site e-commerce – Pascal Roques – édition EYROLLES
- *La Gestion de Production Assistée par Ordinateur « GPAO » - BISMILLAH Salwa*
- « Méthodologie des systèmes d'information MERISE Cours du Cycle Probatoire» A. Lassus – A. Mundubeltz - B. Chaulet / CNAM ANGOULEME 2000-2001
- Gestion de production : Connaissances fondamentales de P.HOMMEL

***Internet :***

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

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

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

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

http://fr.wikipedia.org/wiki/Progiciel_de_gestion_int%C3%A9gr%C3%A9

www.developpez.com

www.commentcamarche.net

http://www.cxp.fr/

http://nte-socio.univ-lyon2.fr/Marc_Grange/BDConception.htm

http://nte-socio.univ-lyon2.fr/Marc_Grange/SQL.htm

http://www.siteduzero.com/tutoriel-3-166641-les-transactions-avec-mysql-et-pdo.html

http://www.siteduzero.com/tutoriel-3-11406-apprenez-a-programmer-en-c.html

http://www.siteduzero.com/tutoriel-3-10601-programmation-en-java.html

http://www.univ-ubs.fr/valoria/antoine/Enseignement/MSI_GL_UML/Merise.pdf

http://www.commentcamarche.net/contents/merise/mcd.php3

http://prolland.free.fr/works/methods/merise.txt

http://tcosnuau.free.fr/COURS/MERISE/MERISE.HTM

http://islamonaa.com/vb/showthread.php?t=8607

# *Annexe A*

## *Code Source*

*Table1:* Produit

```
unit Articles;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Db, DBTables, StdCtrls, ExtCtrls, Mask, DBCtrls, Grids, DBGrids, Buttons;
type
  TForm3 = class(TForm)
    DBNavigator1: TDBNavigator;
    GroupBox4: TGroupBox;
    DBGrid1: TDBGrid;
    DBNavigator2: TDBNavigator;
    DBNavigator3: TDBNavigator;
    GroupBox1: TGroupBox;
    GroupBox2: TGroupBox;
    DBGrid2: TDBGrid;
    DBGrid3: TDBGrid;
    BitBtn1: TBitBtn;
    procedure Button10Click(Sender: TObject);
    procedure CheckBox1Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure Button8Click(Sender: TObject);
    procedure Button9Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Edit2Change(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
  Form3: TForm3;
implementation

uses SarPro, datamodule ;
{$R *.DFM}
procedure TForm3.Button10Click(Sender: TObject);
begin
Form3.close;
end;
procedure TForm3.CheckBox1Click(Sender: TObject);
begin
end;
//if checkbox1.checked:=true then
```

```
procedure TForm3.Button7Click(Sender: TObject);
begin
DataModule1.table3.append;
end;
procedure TForm3.Button6Click(Sender: TObject);
begin
Form3.close;
end;
procedure TForm3.Button8Click(Sender: TObject);
begin
DataModule1.table1.delete;
end;
procedure TForm3.Button9Click(Sender: TObject);
begin
DataModule1.table1.post;
end;
procedure TForm3.Button2Click(Sender: TObject);
begin
DataModule1.Table2.post;
end;
procedure TForm3.Button3Click(Sender: TObject);
begin
DataModule1.Table1.delete;
end;
procedure TForm3.Button4Click(Sender: TObject);
begin
Form3.close;
end;
procedure TForm3.Edit2Change(Sender: TObject);
begin
//Edit2.text:=DBComboBox2.text;
end;
procedure TForm3.Button5Click(Sender: TObject);
begin
DataModule1.Table3.post
end;
procedure TForm3.Button1Click(Sender: TObject);
begin
Form3.close;
end;
procedure TForm3.BitBtn1Click(Sender: TObject);
begin
Form3.close;
end;
end.
```

*Table 2 :* Produit Semi-Fini

```
unit PrSF;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, DBCtrls, Mask, Grids, DBGrids, ExtCtrls, Buttons;
type
  TForm22 = class(TForm)
    Panel1: TPanel;
    GroupBox2: TGroupBox;
    DBGrid2: TDBGrid;
    GroupBox3: TGroupBox;
    Label2: TLabel;
    Label4: TLabel;
    DBEdit1: TDBEdit;
    DBLookupComboBox2: TDBLookupComboBox;
    GroupBox1: TGroupBox;
    BitBtn5: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn1: TBitBtn;
    BitBtn6: TBitBtn;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button11Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
  Form22: TForm22;
implementation
uses datamodule;
{$R *.DFM}
procedure TForm22.Button1Click(Sender: TObject);
begin
DataModule1.Table15.Append;
end;
procedure TForm22.Button2Click(Sender: TObject);
begin
DataModule1.Table15.Edit;
end;
```

```
procedure TForm22.Button3Click(Sender: TObject);
begin
DataModule1.Table15.Delete;
end;
procedure TForm22.Button4Click(Sender: TObject);
begin
DataModule1.Table15.Post;
end;
procedure TForm22.Button11Click(Sender: TObject);
begin
Form22.close;
end;
procedure TForm22.BitBtn6Click(Sender: TObject);
begin
DataModule1.Table15.Append;
end;
procedure TForm22.BitBtn1Click(Sender: TObject);
begin
DataModule1.Table15.Edit;
end;
procedure TForm22.BitBtn2Click(Sender: TObject);
begin
DataModule1.Table15.Delete;
end;
procedure TForm22.BitBtn4Click(Sender: TObject);
begin
DataModule1.Table15.Post;
end;
procedure TForm22.BitBtn5Click(Sender: TObject);
begin
Form22.close;
end;
end.
```

## *Table 3 :* Matière Première

```
unit MatP;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, DBCtrls, Mask, Grids, DBGrids, ExtCtrls, Buttons;
type
  TForm27 = class(TForm)
    Panel1: TPanel;
    GroupBox2: TGroupBox;
    DBGrid2: TDBGrid;
    GroupBox3: TGroupBox;
    Label2: TLabel;
    Label4: TLabel;
    DBEdit1: TDBEdit;
    DBLookupComboBox2: TDBLookupComboBox;
```

```
    GroupBox1: TGroupBox;
    BitBtn5: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn1: TBitBtn;
    BitBtn6: TBitBtn;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button11Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
  Form27: TForm27;
implementation
uses datamodule;
{$R *.DFM}
procedure TForm27.Button1Click(Sender: TObject);
begin
DataModule1.Table16.Append;
end;
procedure TForm27.Button2Click(Sender: TObject);
begin
DataModule1.Table16.Edit;
end;
procedure TForm27.Button3Click(Sender: TObject);
begin
DataModule1.Table16.Delete;
end;
procedure TForm27.Button4Click(Sender: TObject);
begin
DataModule1.Table16.Post;
end;
procedure TForm27.Button11Click(Sender: TObject);
begin
Form27.close;
end;
procedure TForm27.BitBtn6Click(Sender: TObject);
begin
DataModule1.Table16.Append;
end;
```

```
procedure TForm27.BitBtn1Click(Sender: TObject);
begin
DataModule1.Table16.Edit;
end;
procedure TForm27.BitBtn2Click(Sender: TObject);
begin
DataModule1.Table16.Delete;
end;
procedure TForm27.BitBtn4Click(Sender: TObject);
begin
DataModule1.Table16.Post;
end;
procedure TForm27.BitBtn5Click(Sender: TObject);
begin
Form27.close;
end;
end.
```

*Table 4 :* Client

```
unit Client;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, Db, DBTables, StdCtrls, Mask, DBCtrls, Grids, DBGrids, Buttons;
type
  TForm5 = class(TForm)
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    GroupBox2: TGroupBox;
    GroupBox3: TGroupBox;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label4: TLabel;
    Label7: TLabel;
    Label12: TLabel;
    Label13: TLabel;
    GroupBox5: TGroupBox;
    Label14: TLabel;
    Label15: TLabel;
    Label16: TLabel;
    Label17: TLabel;
    Label5: TLabel;
    DBEdit1: TDBEdit;
    DBEdit2: TDBEdit;
    DBEdit3: TDBEdit;
    DBEdit4: TDBEdit;
    DBEdit5: TDBEdit;
```

```
    DBEdit6: TDBEdit;
    DBEdit7: TDBEdit;
    DBEdit8: TDBEdit;
    DBEdit9: TDBEdit;
    DBEdit10: TDBEdit;
    DBEdit11: TDBEdit;
    DBComboBox1: TDBComboBox;
    DBComboBox2: TDBComboBox;
    DBComboBox3: TDBComboBox;
    DBComboBox4: TDBComboBox;
    Label3: TLabel;
    DBEdit12: TDBEdit;
    BitBtn5: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn1: TBitBtn;
    procedure Button11Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
  Form5: TForm5;
implementation
uses SarPro, datamodule;
{$R *.DFM}
procedure TForm5.Button11Click(Sender: TObject);
begin
Form5.close;
end;
procedure TForm5.Button1Click(Sender: TObject);
begin
DataModule1.Table4.append;
end;
procedure TForm5.Button2Click(Sender: TObject);
begin
DataModule1.Table4.edit;
end;
procedure TForm5.Button3Click(Sender: TObject);
```

```
begin
DataModule1.Table4.delete;
end;

procedure TForm5.Button4Click(Sender: TObject);
begin
DataModule1.Table4.post;
end;
procedure TForm5.BitBtn1Click(Sender: TObject);
begin
DataModule1.Table4.append;
end;
procedure TForm5.BitBtn2Click(Sender: TObject);
begin
DataModule1.Table4.edit;
end;
procedure TForm5.BitBtn3Click(Sender: TObject);
begin
DataModule1.Table4.delete;
end;
procedure TForm5.BitBtn4Click(Sender: TObject);
begin
DataModule1.Table4.post;
end;
procedure TForm5.BitBtn5Click(Sender: TObject);
begin
Form5.close;
end;
end.
```

## *Table 5 :* Fournisseur

```
unit Fournisseurs;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, Db, DBTables, StdCtrls, Mask, DBCtrls, Grids, DBGrids, Buttons;
type
  TForm4 = class(TForm)
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label2: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    GroupBox2: TGroupBox;
    GroupBox3: TGroupBox;
    Label8: TLabel;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label4: TLabel;
```

```
    Label7: TLabel;
    Label12: TLabel;
    Label13: TLabel;
    GroupBox5: TGroupBox;
    Label14: TLabel;
    Label15: TLabel;
    Label16: TLabel;
    Label17: TLabel;
    Label18: TLabel;
    DBEdit1: TDBEdit;
    DBEdit2: TDBEdit;
    DBEdit3: TDBEdit;
    DBEdit4: TDBEdit;
    DBEdit6: TDBEdit;
    DBEdit8: TDBEdit;
    DBEdit7: TDBEdit;
    DBEdit9: TDBEdit;
    DBEdit10: TDBEdit;
    DBEdit11: TDBEdit;
    DBEdit13: TDBEdit;
    DBEdit12: TDBEdit;
    DBEdit14: TDBEdit;
    DBComboBox3: TDBComboBox;
    DBComboBox4: TDBComboBox;
    DBComboBox2: TDBComboBox;
    DBComboBox1: TDBComboBox;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    procedure Button11Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
  Form4: TForm4;
implementation
uses SarPro, datamodule;
```

```
{$R *.DFM}
procedure TForm4.Button11Click(Sender: TObject);
begin
Form4.close;
end;
procedure TForm4.Button1Click(Sender: TObject);
begin
DataModule1.Table5.append;
end;
procedure TForm4.Button2Click(Sender: TObject);
begin
DataModule1.Table5.edit;
end;
procedure TForm4.Button3Click(Sender: TObject);
begin
DataModule1.Table5.delete;
end;
procedure TForm4.Button4Click(Sender: TObject);
begin
DataModule1.Table5.post;
end;
procedure TForm4.BitBtn1Click(Sender: TObject);
begin
DataModule1.Table5.append;
end;
procedure TForm4.BitBtn2Click(Sender: TObject);
begin
DataModule1.Table5.edit;
end;
procedure TForm4.BitBtn3Click(Sender: TObject);
begin
DataModule1.Table5.delete;
end;
procedure TForm4.BitBtn4Click(Sender: TObject);
begin
DataModule1.Table5.post;
end;
procedure TForm4.BitBtn5Click(Sender: TObject);
begin
Form4.close;
end;
end.
```

## *Table 6 :* Commande Client

```
unit Commandes;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Buttons, Db, DBTables, StdCtrls, Mask, DBCtrls, ExtCtrls, Grids, DBGrids;
type
```

```
  TForm6 = class(TForm)
    Panel1: TPanel;
    DBNavigator5: TDBNavigator;
    GroupBox7: TGroupBox;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    DBEdit7: TDBEdit;
    DBEdit8: TDBEdit;
    DBEdit9: TDBEdit;
    GroupBox8: TGroupBox;
    DBNavigator6: TDBNavigator;
    GroupBox9: TGroupBox;
    BitBtn2: TBitBtn;
    DBLookupComboBox1: TDBLookupComboBox;
    DBGrid1: TDBGrid;
    DBGrid2: TDBGrid;
    BitBtn5: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn6: TBitBtn;
    BitBtn1: TBitBtn;
    procedure Button2Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button8Click(Sender: TObject);
    procedure Button9Click(Sender: TObject);
    procedure Button10Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
  Form6: TForm6;
implementation
uses SarPro, datamodule, Unit16;
{$R *.DFM}
procedure TForm6.Button2Click(Sender: TObject);
```

```
begin
Form6.close;
end;
procedure TForm6.Button7Click(Sender: TObject);
begin
application.terminate;
end;
procedure TForm6.Button3Click(Sender: TObject);
begin
DataModule1.Table6.append;
end;
procedure TForm6.Button4Click(Sender: TObject);
begin
DataModule1.Table6.edit;
end;
procedure TForm6.Button5Click(Sender: TObject);
begin
DataModule1.Table6.delete;
end;
procedure TForm6.Button1Click(Sender: TObject);
begin
DataModule1.Table6.post;
end;
procedure TForm6.Button8Click(Sender: TObject);
begin
DataModule1.Table7.append;
end;
procedure TForm6.Button9Click(Sender: TObject);
begin
DataModule1.Table7.edit;
end;
procedure TForm6.Button10Click(Sender: TObject);
begin
DataModule1.Table7.delete;
end;
procedure TForm6.Button6Click(Sender: TObject);
begin
DataModule1.Table7.post;
end;
procedure TForm6.BitBtn2Click(Sender: TObject);
var ContrainteStock:boolean;
   ContrainteListe:string;
begin
 ContrainteStock:=true;
 ContrainteListe:='';
if  DataModule1.table6.FieldByName('Valider').value=false
 then
 begin
 datamodule1.Query1.Close;
  datamodule1.Query1.SQL.Clear;
```

```
  datamodule1.Query1.SQL.Add('delete from sf_temp ; ');
  datamodule1.Query1.ExecSQL;
  datamodule1.Query2.Close;
  datamodule1.Query2.SQL.Clear;
  datamodule1.Query2.SQL.Add('delete from sf2_temp ; ');
  datamodule1.Query2.ExecSQL;
  datamodule1.Query1.Close;
  datamodule1.Query1.SQL.Clear;
  datamodule1.Query1.SQL.Add('delete from mp_temp ; ');
  datamodule1.Query1.ExecSQL;
  datamodule1.Query2.Close;
  datamodule1.Query2.SQL.Clear;
  datamodule1.Query2.SQL.Add('delete from mp2_temp ; ');
  datamodule1.Query2.ExecSQL;
  datamodule1.Query1.Close;
  datamodule1.Query1.SQL.Clear;
  datamodule1.Query1.SQL.Add('insert into sf_temp (CODE_PSF,QTE_P) ');
  datamodule1.Query1.SQL.Add('select sf.code_psf, sf.qte_p*cp.quantite');
  datamodule1.Query1.SQL.Add('from produit_semi_fini sf , commande_produit cp, commande_client cc ');
  datamodule1.Query1.SQL.Add('where
cc.n_cde='''+datamodule1.Table12.FieldByName('N_Cde').AsString+'''');
  datamodule1.Query1.SQL.Add('and cp.n_cde=cc.n_cde');
  datamodule1.Query1.SQL.Add('and  cp.code_p=sf.code_p ');
  datamodule1.Query1.ExecSQL;
  datamodule1.Query2.Close;
  datamodule1.Query2.SQL.Clear;
  datamodule1.Query2.SQL.Add('insert into sf2_temp (CODE_PSF,QTE_P) ');
  datamodule1.Query2.SQL.Add('select enfant.code_psf, parent.qte_p*enfant.qte_p');
  datamodule1.Query2.SQL.Add('from sf_temp  parent , produit_semi_fini enfant  ');
  datamodule1.Query2.SQL.Add('where parent.code_psf=enfant.code_p ');
  datamodule1.Query2.ExecSQL;

  datamodule1.Query1.Close;
  datamodule1.Query1.SQL.Clear;
  datamodule1.Query1.SQL.Add('insert into sf_temp (CODE_PSF,QTE_P) ');
  datamodule1.Query1.SQL.Add('select sf2.code_psf, sf2.qte_p');
  datamodule1.Query1.SQL.Add('from sf2_temp sf2, sf_temp sf ');
  datamodule1.Query1.SQL.Add('where sf2.code_psf=sf.code_psf ');
  datamodule1.Query1.ExecSQL;
  datamodule1.Query2.Close;
  datamodule1.Query2.SQL.Clear;
  datamodule1.Query2.SQL.Add('delete from sf2_temp ; ');
  datamodule1.Query2.ExecSQL;
  datamodule1.Query2.Close;
  datamodule1.Query2.SQL.Clear;
  datamodule1.Query2.SQL.Add('insert into sf2_temp (CODE_PSF,QTE_P) ');
  datamodule1.Query2.SQL.Add('select sf.code_psf , sum(sf.qte_p)');
  datamodule1.Query2.SQL.Add('from sf_temp  sf  ');
  datamodule1.Query2.SQL.Add('group by sf.code_psf ');
  datamodule1.Query2.ExecSQL;
```

```
 datamodule1.Query1.Close;
 datamodule1.Query1.SQL.Clear;
 datamodule1.Query1.SQL.Add('insert into mp_temp (CODE_M,QTE) ');
 datamodule1.Query1.SQL.Add('select mp.code_m, sf2.qte_p*mp.qte_psf');
 datamodule1.Query1.SQL.Add('from sf2_temp sf2, matiere_premiere mp ');
 datamodule1.Query1.SQL.Add('where sf2.code_psf=mp.code_psf ');
 datamodule1.Query1.ExecSQL;
 datamodule1.Query2.Close;
 datamodule1.Query2.SQL.Clear;
 datamodule1.Query2.SQL.Add('insert into mp2_temp (CODE_M,QTE) ');
 datamodule1.Query2.SQL.Add('select mp.code_m , sum(mp.qte)');
 datamodule1.Query2.SQL.Add('from mp_temp  mp  ');
 datamodule1.Query2.SQL.Add('group by mp.code_m ');
 datamodule1.Query2.ExecSQL;
//*****************************************************
// vérifier si les quntités de la requête respectent
// le stock minimum, si c'est le cas "ContrainteStock"
// est à "true" sinon "ContrainteListe" va regrouper
// la liste des matières première qui ne respectent
// pas le stock minimum
//*****************************************************
 datamodule1.TableTemp4.open;
 datamodule1.TableTemp4.First;
 datamodule1.Table14.First;
 While (not datamodule1.Table14.Eof) Do
  BEGIN
   if datamodule1.TableTemp4.FindKey([datamodule1.Table14Code_m.value]) then
     begin
       if                    datamodule1.Table14.FieldByName('stock_min').Value                    >=
datamodule1.Table14.FieldByName('stock').Value-datamodule1.TableTemp4.FieldByName('qte').Value
        then
        begin
        ContrainteStock:=false;
        ContrainteListe:=ContrainteListe+' '+datamodule1.Table14.FieldByName('code_m').value;
        end;
    end;
   datamodule1.Table14.Next;
  end;
//*****************************************************
if ContrainteStock=true then
 begin
  datamodule1.Query1.Close;
  datamodule1.Query1.SQL.Clear;
  datamodule1.Query1.SQL.Add('update matiere ma ');
  datamodule1.Query1.SQL.Add('set ma.stock= ma.stock -');
  datamodule1.Query1.SQL.Add('(select  mp2.qte ');
  datamodule1.Query1.SQL.Add('from mp2_temp mp2');
  datamodule1.Query1.SQL.Add('where mp2.code_m = ma.code_m )');

 //datamodule1.Query1.Constrained:=true;
```

```
 //datamodule1.Table14.Constraints.BeginUpdate;
  datamodule1.Query1.ExecSQL;
  DataModule1.table6.Edit;
  DataModule1.table6.FieldByName('Valider').value:=true;
  datamodule1.Table14.Refresh;
  end
  else showmessage(ContrainteListe+' en dessous du stock minimum, veuillez réaprovisionner');
 end
 else
 begin
   showmessage('Cette commande a déja été validée, veuillez entrer une nouvelle commande')
 end;
end;
procedure TForm6.BitBtn6Click(Sender: TObject);
begin
DataModule1.Table6.append;
end;
procedure TForm6.BitBtn3Click(Sender: TObject);
begin
DataModule1.Table6.delete;
end;
procedure TForm6.BitBtn4Click(Sender: TObject);
begin
DataModule1.Table6.post;
end;
procedure TForm6.BitBtn5Click(Sender: TObject);
begin
Form6.close;
end;
procedure TForm6.BitBtn1Click(Sender: TObject);
begin
DataModule1.Table6.edit;
end;
end.
```

## *Table 7 :* Commande Fournisseur

```
unit CommandesF;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Buttons, Grids, DBGrids, DBCtrls, StdCtrls, Mask, ExtCtrls;
type
  TForm17 = class(TForm)
    Panel2: TPanel;
    DBNavigator1: TDBNavigator;
    GroupBox2: TGroupBox;
    Label4: TLabel;
    Label7: TLabel;
    Label2: TLabel;
    Label5: TLabel;
```

```
    DBEdit1: TDBEdit;
    DBEdit2: TDBEdit;
    DBEdit3: TDBEdit;
    GroupBox4: TGroupBox;
    DBNavigator3: TDBNavigator;
    GroupBox5: TGroupBox;
    BitBtn3: TBitBtn;
    DBGrid2: TDBGrid;
    DBLookupComboBox1: TDBLookupComboBox;
    DBGrid1: TDBGrid;
    BitBtn1: TBitBtn;
    BitBtn6: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    BitBtn2: TBitBtn;
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
  Form17: TForm17;
implementation
uses datamodule, BonCde;
{$R *.DFM}
procedure TForm17.Button2Click(Sender: TObject);
begin
Form17.close;
end;
procedure TForm17.Button3Click(Sender: TObject);
begin
DataModule1.Table7.Append;
end;
procedure TForm17.Button4Click(Sender: TObject);
begin
DataModule1.Table7.Edit;
end;
procedure TForm17.Button5Click(Sender: TObject);
begin
```

```
DataModule1.Table7.Delete;
end;
procedure TForm17.Button1Click(Sender: TObject);
begin
DataModule1.Table7.post;
end;
procedure TForm17.BitBtn3Click(Sender: TObject);
begin
if  DataModule1.table7.FieldByName('Valider').value=false
 then begin
 datamodule1.Query3.Close;
 datamodule1.Query3.SQL.Clear;
 datamodule1.Query3.SQL.Add('UPDATE matiere ma ');
 datamodule1.Query3.SQL.Add('SET ma.stock= ma.stock +');
 datamodule1.Query3.SQL.Add('(select  cm.quantite ');
 datamodule1.Query3.SQL.Add('from commande_matiere cm');
 datamodule1.Query3.SQL.Add('where cm.code_m = ma.code_m ');
 datamodule1.Query3.SQL.Add('and cm.n_cde ="'+datamodule1.Table7.fieldbyname('N_Cde').AsString +'")');
 datamodule1.Query3.SQL.Add('WHERE ma.code_m IN ');
 datamodule1.Query3.SQL.Add('(select cm.code_m');
 datamodule1.Query3.SQL.Add('from commande_matiere cm ');
 datamodule1.Query3.SQL.Add('where                                                                    cm.n_cde
="'+datamodule1.Table7.fieldbyname('N_Cde').AsString+'")');
 datamodule1.Query3.ExecSQL;
//datamodule1.Table14.Close;
//datamodule1.Table14.open;
//datamodule1.table14.Edit;
//datamodule1.table14.fieldbyname('Stock').asinteger:=datamodule1.table14.fieldbyname('Stock').asinteger
+ datamodule1.table13.fieldbyname('Quantite').asinteger;
// ne marche que pour la première case
 datamodule1.table7.edit;
 datamodule1.table7.fieldbyname('Valider').value:=true;
 datamodule1.Table14.Close;
 datamodule1.Table14.open;
 end
 else
 showmessage('veuillez entrer une nouvelle commande')  ;
end;
procedure TForm17.BitBtn6Click(Sender: TObject);
begin
DataModule1.Table7.Append;
end;
procedure TForm17.BitBtn1Click(Sender: TObject);
begin
DataModule1.Table7.Edit;
end;
procedure TForm17.BitBtn4Click(Sender: TObject);
begin
DataModule1.Table7.post;
end;
```

```
procedure TForm17.BitBtn5Click(Sender: TObject);
begin
Form17.close;
end;
procedure TForm17.BitBtn2Click(Sender: TObject);
begin
DataModule1.Table7.Delete;
end;
end.
```

*Table 8 :* Ville

Représente la liste de toutes les villes d'Algérie.

*Table 9 :* Pays

Représente la liste de tous les pays du monde.

*Table 10 :* Poste de charge

Représente les trois postes de charges à savoir le découpage, l'usinage et le montage.

*Table 11 :* Mot de Passe

```
unit GPAO;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Buttons, Db, DBTables;
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Entrer: TBitBtn;
    Button1: TButton;
    Label3: TLabel;
    Label2: TLabel;
    procedure Edit1Change(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure EntrerClick(Sender: TObject);
    procedure Edit1KeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
```

```
  Form1: TForm1;
   essai: integer;
implementation
uses SarPro, datamodule;
{$R *.DFM}
procedure TForm1.Edit1Change(Sender: TObject);
begin
if edit1.text<>'' then entrer.enabled:=true
              else entrer.enabled:=false;
end;
procedure TForm1.Button1Click(Sender: TObject);
begin
application.Terminate;
end;
procedure TForm1.EntrerClick(Sender: TObject);
begin
DataModule1.Table11.open;
if DataModule1.Table11.findkey([edit1.text]) then form2.showmodal
                        else showmessage('Mot de passe incorrect, veuillez le saisir de nouveau');
                        edit1.clear;
                      end;
procedure TForm1.Edit1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
if key =13 then begin
DataModule1.Table11.open;
if DataModule1.Table11.findkey([edit1.text]) then form2.showmodal
                        else showmessage('Mot de passe incorrect, veuillez le saisir de nouveau');
                        edit1.clear;
                      end;
end;
end.
```

## *Table 12 :* Commande Produit

```
unit Commandes;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Buttons, Db, DBTables, StdCtrls, Mask, DBCtrls, ExtCtrls, Grids, DBGrids;
type
  TForm6 = class(TForm)
    Panel1: TPanel;
    DBNavigator5: TDBNavigator;
    GroupBox7: TGroupBox;
    Label9: TLabel;
    Label10: TLabel;
    Label11: TLabel;
    Label12: TLabel;
    DBEdit7: TDBEdit;
    DBEdit8: TDBEdit;
```

```
    DBEdit9: TDBEdit;
    GroupBox8: TGroupBox;
    DBNavigator6: TDBNavigator;
    GroupBox9: TGroupBox;
    BitBtn2: TBitBtn;
    DBLookupComboBox1: TDBLookupComboBox;
    DBGrid1: TDBGrid;
    DBGrid2: TDBGrid;
    BitBtn5: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn3: TBitBtn;
    BitBtn6: TBitBtn;
    BitBtn1: TBitBtn;
    procedure Button2Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure Button8Click(Sender: TObject);
    procedure Button9Click(Sender: TObject);
    procedure Button10Click(Sender: TObject);
    procedure Button6Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
  Form6: TForm6;
implementation
uses SarPro, datamodule, Unit16;
{$R *.DFM}
procedure TForm6.Button2Click(Sender: TObject);
begin
Form6.close;
end;
procedure TForm6.Button7Click(Sender: TObject);
begin
application.terminate;
end;
procedure TForm6.Button3Click(Sender: TObject);
begin
DataModule1.Table6.append;
```

```
end;
procedure TForm6.Button4Click(Sender: TObject);
begin
DataModule1.Table6.edit;
end;
procedure TForm6.Button5Click(Sender: TObject);
begin
DataModule1.Table6.delete;
end;

procedure TForm6.Button1Click(Sender: TObject);
begin
DataModule1.Table6.post;
end;
procedure TForm6.Button8Click(Sender: TObject);
begin
DataModule1.Table7.append;
end;

procedure TForm6.Button9Click(Sender: TObject);
begin
DataModule1.Table7.edit;
end;
procedure TForm6.Button10Click(Sender: TObject);
begin
DataModule1.Table7.delete;
end;
procedure TForm6.Button6Click(Sender: TObject);
begin
DataModule1.Table7.post;
end;
procedure TForm6.BitBtn2Click(Sender: TObject);
var ContrainteStock:boolean;
   ContrainteListe:string;
begin
 ContrainteStock:=true;
 ContrainteListe:='';
if  DataModule1.table6.FieldByName('Valider').value=false
 then
 begin
 datamodule1.Query1.Close;
  datamodule1.Query1.SQL.Clear;
  datamodule1.Query1.SQL.Add('delete from sf_temp ; ');
  datamodule1.Query1.ExecSQL;
  datamodule1.Query2.Close;
  datamodule1.Query2.SQL.Clear;
  datamodule1.Query2.SQL.Add('delete from sf2_temp ; ');
  datamodule1.Query2.ExecSQL;
  datamodule1.Query1.Close;
  datamodule1.Query1.SQL.Clear;
```

```
  datamodule1.Query1.SQL.Add('delete from mp_temp ; ');
  datamodule1.Query1.ExecSQL;

  datamodule1.Query2.Close;
  datamodule1.Query2.SQL.Clear;
  datamodule1.Query2.SQL.Add('delete from mp2_temp ; ');
  datamodule1.Query2.ExecSQL;
  datamodule1.Query1.Close;
  datamodule1.Query1.SQL.Clear;
  datamodule1.Query1.SQL.Add('insert into sf_temp (CODE_PSF,QTE_P) ');
  datamodule1.Query1.SQL.Add('select sf.code_psf, sf.qte_p*cp.quantite');
  datamodule1.Query1.SQL.Add('from produit_semi_fini sf , commande_produit cp, commande_client cc ');
  datamodule1.Query1.SQL.Add('where
cc.n_cde='''+datamodule1.Table12.FieldByName('N_Cde').AsString+'''');
  datamodule1.Query1.SQL.Add('and cp.n_cde=cc.n_cde');
  datamodule1.Query1.SQL.Add('and  cp.code_p=sf.code_p ');
  datamodule1.Query1.ExecSQL;

  datamodule1.Query2.Close;
  datamodule1.Query2.SQL.Clear;
  datamodule1.Query2.SQL.Add('insert into sf2_temp (CODE_PSF,QTE_P) ');
  datamodule1.Query2.SQL.Add('select enfant.code_psf, parent.qte_p*enfant.qte_p');
  datamodule1.Query2.SQL.Add('from sf_temp  parent , produit_semi_fini enfant  ');
  datamodule1.Query2.SQL.Add('where parent.code_psf=enfant.code_p ');
  datamodule1.Query2.ExecSQL;

  datamodule1.Query1.Close;
  datamodule1.Query1.SQL.Clear;
  datamodule1.Query1.SQL.Add('insert into sf_temp (CODE_PSF,QTE_P) ');
  datamodule1.Query1.SQL.Add('select sf2.code_psf, sf2.qte_p');
  datamodule1.Query1.SQL.Add('from sf2_temp sf2, sf_temp sf ');
  datamodule1.Query1.SQL.Add('where sf2.code_psf=sf.code_psf ');
  datamodule1.Query1.ExecSQL;
  datamodule1.Query2.Close;
  datamodule1.Query2.SQL.Clear;
  datamodule1.Query2.SQL.Add('delete from sf2_temp ; ');
  datamodule1.Query2.ExecSQL;
  datamodule1.Query2.Close;
  datamodule1.Query2.SQL.Clear;
  datamodule1.Query2.SQL.Add('insert into sf2_temp (CODE_PSF,QTE_P) ');
  datamodule1.Query2.SQL.Add('select sf.code_psf , sum(sf.qte_p)');
  datamodule1.Query2.SQL.Add('from sf_temp  sf  ');
  datamodule1.Query2.SQL.Add('group by sf.code_psf ');
  datamodule1.Query2.ExecSQL;
  datamodule1.Query1.Close;
  datamodule1.Query1.SQL.Clear;
  datamodule1.Query1.SQL.Add('insert into mp_temp (CODE_M,QTE) ');
  datamodule1.Query1.SQL.Add('select mp.code_m, sf2.qte_p*mp.qte_psf');
  datamodule1.Query1.SQL.Add('from sf2_temp sf2, matiere_premiere mp ');
  datamodule1.Query1.SQL.Add('where sf2.code_psf=mp.code_psf ');
```

```
 datamodule1.Query1.ExecSQL;
 datamodule1.Query2.Close;
 datamodule1.Query2.SQL.Clear;
 datamodule1.Query2.SQL.Add('insert into mp2_temp (CODE_M,QTE) ');
 datamodule1.Query2.SQL.Add('select mp.code_m , sum(mp.qte)');
 datamodule1.Query2.SQL.Add('from mp_temp  mp  ');
 datamodule1.Query2.SQL.Add('group by mp.code_m ');
 datamodule1.Query2.ExecSQL;
//*****************************************************
// vérifier si les quntités de la requête respectent
// le stock minimum, si c'est le cas "ContrainteStock"
// est à "true" sinon "ContrainteListe" va regrouper
// la liste des matières première qui ne respectent
// pas le stock minimum
//*****************************************************
 datamodule1.TableTemp4.open;
 datamodule1.TableTemp4.First;
 datamodule1.Table14.First;
 While (not datamodule1.Table14.Eof) Do
  BEGIN
    if datamodule1.TableTemp4.FindKey([datamodule1.Table14Code_m.value]) then
      begin
        if datamodule1.Table14.FieldByName('stock_min').Value >=
datamodule1.Table14.FieldByName('stock').Value-datamodule1.TableTemp4.FieldByName('qte').Value
         then
         begin
         ContrainteStock:=false;
         ContrainteListe:=ContrainteListe+' '+datamodule1.Table14.FieldByName('code_m').value;
         end;
    end;
   datamodule1.Table14.Next;
  end;
//*****************************************************
if ContrainteStock=true then
 begin
  datamodule1.Query1.Close;
  datamodule1.Query1.SQL.Clear;
  datamodule1.Query1.SQL.Add('update matiere ma ');
  datamodule1.Query1.SQL.Add('set ma.stock= ma.stock -');
  datamodule1.Query1.SQL.Add('(select  mp2.qte ');
  datamodule1.Query1.SQL.Add('from mp2_temp mp2');
  datamodule1.Query1.SQL.Add('where mp2.code_m = ma.code_m )');

 //datamodule1.Query1.Constrained:=true;
 //datamodule1.Table14.Constraints.BeginUpdate;
  datamodule1.Query1.ExecSQL;
  DataModule1.table6.Edit;
  DataModule1.table6.FieldByName('Valider').value:=true;
  datamodule1.Table14.Refresh;
  end
```

```
  else showmessage(ContrainteListe+' en dessous du stock minimum, veuillez réaprovisionner');
 end
 else
 begin
   showmessage('Cette commande a déja été validée, veuillez entrer une nouvelle commande')
 end;
end;
procedure TForm6.BitBtn6Click(Sender: TObject);
begin
DataModule1.Table6.append;
end;
procedure TForm6.BitBtn3Click(Sender: TObject);
begin
DataModule1.Table6.delete;
end;

procedure TForm6.BitBtn4Click(Sender: TObject);
begin
DataModule1.Table6.post;
end;
procedure TForm6.BitBtn5Click(Sender: TObject);
begin
Form6.close;
end;
procedure TForm6.BitBtn1Click(Sender: TObject);
begin
DataModule1.Table6.edit;
end;
end.
```

## *Table 13 :* Commande matière première

```
unit CommandesF;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Buttons, Grids, DBGrids, DBCtrls, StdCtrls, Mask, ExtCtrls;
type
  TForm17 = class(TForm)
    Panel2: TPanel;
    DBNavigator1: TDBNavigator;
    GroupBox2: TGroupBox;
    Label4: TLabel;
    Label7: TLabel;
    Label2: TLabel;
    Label5: TLabel;
    DBEdit1: TDBEdit;
    DBEdit2: TDBEdit;
    DBEdit3: TDBEdit;
    GroupBox4: TGroupBox;
    DBNavigator3: TDBNavigator;
```

```
    GroupBox5: TGroupBox;
    BitBtn3: TBitBtn;
    DBGrid2: TDBGrid;
    DBLookupComboBox1: TDBLookupComboBox;
    DBGrid1: TDBGrid;
    BitBtn1: TBitBtn;
    BitBtn6: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    BitBtn2: TBitBtn;
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button4Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
  Form17: TForm17;
implementation
uses datamodule, BonCde;
{$R *.DFM}
procedure TForm17.Button2Click(Sender: TObject);
begin
Form17.close;
end;
procedure TForm17.Button3Click(Sender: TObject);
begin
DataModule1.Table7.Append;
end;
procedure TForm17.Button4Click(Sender: TObject);
begin
DataModule1.Table7.Edit;
end;
procedure TForm17.Button5Click(Sender: TObject);
begin
DataModule1.Table7.Delete;
end;
procedure TForm17.Button1Click(Sender: TObject);
begin
DataModule1.Table7.post;
```

```
end;
procedure TForm17.BitBtn3Click(Sender: TObject);
begin
if  DataModule1.table7.FieldByName('Valider').value=false
 then begin
 datamodule1.Query3.Close;
 datamodule1.Query3.SQL.Clear;
 datamodule1.Query3.SQL.Add('UPDATE matiere ma ');
 datamodule1.Query3.SQL.Add('SET ma.stock= ma.stock +');
 datamodule1.Query3.SQL.Add('(select  cm.quantite ');
 datamodule1.Query3.SQL.Add('from commande_matiere cm');
 datamodule1.Query3.SQL.Add('where cm.code_m = ma.code_m ');
 datamodule1.Query3.SQL.Add('and cm.n_cde ="'+datamodule1.Table7.fieldbyname('N_Cde').AsString +'")');
 datamodule1.Query3.SQL.Add('WHERE ma.code_m IN ');
 datamodule1.Query3.SQL.Add('(select cm.code_m');
 datamodule1.Query3.SQL.Add('from commande_matiere cm ');
 datamodule1.Query3.SQL.Add('where cm.n_cde
="'+datamodule1.Table7.fieldbyname('N_Cde').AsString+'")');
 datamodule1.Query3.ExecSQL;
//datamodule1.Table14.Close;
//datamodule1.Table14.open;
//datamodule1.table14.Edit;
//datamodule1.table14.fieldbyname('Stock').asinteger:=datamodule1.table14.fieldbyname('Stock').asinteger
+ datamodule1.table13.fieldbyname('Quantite').asinteger;
// ne marche que pour la première case
 datamodule1.table7.edit;
 datamodule1.table7.fieldbyname('Valider').value:=true;
 datamodule1.Table14.Close;
 datamodule1.Table14.open;
 end
 else
 showmessage('veuillez entrer une nouvelle commande')  ;
end;
procedure TForm17.BitBtn6Click(Sender: TObject);
begin
DataModule1.Table7.Append;
end;
procedure TForm17.BitBtn1Click(Sender: TObject);
begin
DataModule1.Table7.Edit;
end;
procedure TForm17.BitBtn4Click(Sender: TObject);
begin
DataModule1.Table7.post;
end;
procedure TForm17.BitBtn5Click(Sender: TObject);
begin
Form17.close;
end;
```

```
procedure TForm17.BitBtn2Click(Sender: TObject);
begin
DataModule1.Table7.Delete;
end;
end.
```

## *Table 14 :* Liste des Produits Semi-Finis

```
unit listPSF;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Grids, DBGrids, Buttons;
type
  TForm24 = class(TForm)
    DBGrid1: TDBGrid;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    procedure Button2Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
  Form24: TForm24;
implementation
uses datamodule, ImpresPSF;
{$R *.DFM}
procedure TForm24.Button2Click(Sender: TObject);
begin
Form24.close;
end;
procedure TForm24.Button1Click(Sender: TObject);
begin
Form23.quickrep1.preview;
end;
procedure TForm24.BitBtn1Click(Sender: TObject);
begin
Form23.quickrep1.preview;
end;
procedure TForm24.BitBtn2Click(Sender: TObject);
begin
Form24.close;
end;
end.
```

*Table 15 :* Liste des matières Premières

```
unit etatstock;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids, DBGrids, Db, DBTables, StdCtrls, Buttons;
type
  TForm13 = class(TForm)
    DBGrid1: TDBGrid;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    procedure Button2Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
  Form13: TForm13;
implementation
uses datamodule, ImpresEtaStock;
{$R *.DFM}
procedure TForm13.Button2Click(Sender: TObject);
begin
Form13.close;
end;
procedure TForm13.Button1Click(Sender: TObject);
begin
Form21.quickrep1.preview;
end;
procedure TForm13.BitBtn1Click(Sender: TObject);
begin
Form21.quickrep1.preview;
end;
procedure TForm13.BitBtn2Click(Sender: TObject);
begin
Form13.close;
end;
end.
```

Les tables temporaires se composent de deux types : Deux tables représentant les matières premières et les deux autres les produits semi-finis, elles sont conçus de la même manière que les tables des listes des produits semi-finis et des matières premières.

Nous ajouterons les codes sources suivants :

## *DATAMODULE :*

```
unit datamodule;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Db, DBTables;
type
  TDataModule1 = class(TDataModule)
    DataSource1: TDataSource;
    Table1: TTable;
    DataSource2: TDataSource;
    Table2: TTable;
    DataSource3: TDataSource;
    Table3: TTable;
    Table4: TTable;
    DataSource4: TDataSource;
    Table5: TTable;
    DataSource5: TDataSource;
    Table6: TTable;
    DataSource6: TDataSource;
    Table7: TTable;
    DataSource7: TDataSource;
    Table8: TTable;
    DataSource8: TDataSource;
    Table9: TTable;
    DataSource9: TDataSource;
    Table10: TTable;
    DataSource10: TDataSource;
    Table11: TTable;
    DataSource11: TDataSource;
    Table12: TTable;
    DataSource12: TDataSource;
    Table13: TTable;
    DataSource13: TDataSource;
    Table14: TTable;
    DataSource14: TDataSource;
    Table15: TTable;
    DataSource15: TDataSource;
    Table16: TTable;
    DataSource16: TDataSource;
    Query1: TQuery;
    TableTemp: TTable;
    DataSourceTemp: TDataSource;
    TableTemp2: TTable;
    DataSourceTemp2: TDataSource;
    Query2: TQuery;
    TableTemp3: TTable;
    TableTemp4: TTable;
```

```
    Query3: TQuery;
    Table14Code_M: TStringField;
    Table14Stock_min: TFloatField;
    Table14Stock: TFloatField;
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
  DataModule1: TDataModule1;
implementation
{$R *.DFM}
end.
```

## *Menu :*

```
unit ProdIG;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, Menus, ExtCtrls, Buttons, jpeg;
type
  TForm2 = class(TForm)
    MainMenu1: TMainMenu;
    Fichier1: TMenuItem;
    Quitter1: TMenuItem;
    donnes1: TMenuItem;
    Articles1: TMenuItem;
    ListeArticles1: TMenuItem;
    Achats1: TMenuItem;
    Fournisseurs1: TMenuItem;
    ListeFournisseurs1: TMenuItem;
    N3: TMenuItem;
    CommandeFournisseur1: TMenuItem;
    ListeCommandeFournisseur1: TMenuItem;
    Ventes1: TMenuItem;
    Clients1: TMenuItem;
    ListeClients1: TMenuItem;
    N6: TMenuItem;
    CommandesClients1: TMenuItem;
    ListeCommandesClients1: TMenuItem;
    Nomenclature1: TMenuItem;
    Stock1: TMenuItem;
    Entre1: TMenuItem;
    Inventaire1: TMenuItem;
    propos1: TMenuItem;
    Calculatrice1: TMenuItem;
    Calendrier1: TMenuItem;
    N7: TMenuItem;
    Timer1: TTimer;
```

```
proposdeSarPro2: TMenuItem;
Nomenclature2: TMenuItem;
Liste1: TMenuItem;
N2: TMenuItem;
MatirePremire1: TMenuItem;
ListeMatirePremire1: TMenuItem;
Panel1: TPanel;
Label2: TLabel;
Label4: TLabel;
Label5: TLabel;
Label6: TLabel;
Label8: TLabel;
Label9: TLabel;
Label10: TLabel;
Label12: TLabel;
Label13: TLabel;
Label19: TLabel;
Label20: TLabel;
Label21: TLabel;
Label22: TLabel;
Label23: TLabel;
Label24: TLabel;
Label25: TLabel;
Label26: TLabel;
Label27: TLabel;
Label28: TLabel;
Label29: TLabel;
Image1: TImage;
Label1: TLabel;
Label3: TLabel;
procedure Button1Click(Sender: TObject);
procedure Button2Click(Sender: TObject);
procedure Button3Click(Sender: TObject);
procedure Button4Click(Sender: TObject);
procedure Button5Click(Sender: TObject);
procedure Button6Click(Sender: TObject);
procedure Quitter1Click(Sender: TObject);
procedure Articles1Click(Sender: TObject);
procedure Fournisseurs1Click(Sender: TObject);
procedure Clients1Click(Sender: TObject);
procedure CommandesClients1Click(Sender: TObject);
procedure Entre1Click(Sender: TObject);
procedure Sortie1Click(Sender: TObject);
procedure CommandeFournisseur1Click(Sender: TObject);
procedure Nomenclature1Click(Sender: TObject);
procedure ListeArticles1Click(Sender: TObject);
procedure ListeFournisseurs1Click(Sender: TObject);
procedure ListeCommandeFournisseur1Click(Sender: TObject);
procedure ListeClients1Click(Sender: TObject);
procedure ListeCommandesClients1Click(Sender: TObject);
```

```
    procedure Inventaire1Click(Sender: TObject);
    procedure Calendrier1Click(Sender: TObject);
    procedure Timer1Timer(Sender: TObject);
    procedure Calculatrice1Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure Nomenclature2Click(Sender: TObject);
    procedure Liste1Click(Sender: TObject);
    procedure MatirePremire1Click(Sender: TObject);
    procedure ListeMatirePremire1Click(Sender: TObject);
    procedure dtailCommandes1Click(Sender: TObject);
    procedure proposdeSarPro2Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
  Form2: TForm2;
implementation
uses Articles, Fournisseurs, Client, Commandes, Stock, listearticles,
  listefournisseurs, listecommandefournisseur, listeclients,
  listecommandeclient, etatstock, calendrier, datamodule, CommandesF, PrSF,
  listPSF, MatP, ImpresMatP, listMatierP, DetailCde, BonCde;
{$R *.DFM}
procedure TForm2.Button1Click(Sender: TObject);
begin
form3.showmodal;
end;
procedure TForm2.Button2Click(Sender: TObject);
begin
Form4.showmodal;
end;
procedure TForm2.Button3Click(Sender: TObject);
begin
Form5.showmodal;
end;
procedure TForm2.Button4Click(Sender: TObject);
begin
Form6.showmodal;
end;
procedure TForm2.Button5Click(Sender: TObject);
begin
Form7.showmodal;
end;
procedure TForm2.Button6Click(Sender: TObject);
begin
application.terminate;
end;
procedure TForm2.Quitter1Click(Sender: TObject);
begin
```

```
application.terminate;
end;
procedure TForm2.Articles1Click(Sender: TObject);
begin
Form3.ShowModal;
end;
procedure TForm2.Fournisseurs1Click(Sender: TObject);
begin
Form4.showmodal;
end;
procedure TForm2.Clients1Click(Sender: TObject);
begin
Form5.showmodal;
end;
procedure TForm2.CommandesClients1Click(Sender: TObject);
begin
Form6.showmodal;
end;
procedure TForm2.Entre1Click(Sender: TObject);
begin
Form7.showmodal;
end;
procedure TForm2.Sortie1Click(Sender: TObject);
begin
Form7.showmodal;
end;
procedure TForm2.CommandeFournisseur1Click(Sender: TObject);
begin
Form17.showmodal;
end;
procedure TForm2.Nomenclature1Click(Sender: TObject);
begin
Form3.showmodal;
end;
procedure TForm2.ListeArticles1Click(Sender: TObject);
begin
form8.showmodal;
end;
procedure TForm2.ListeFournisseurs1Click(Sender: TObject);
begin
Form9.showmodal;
end;
procedure TForm2.ListeCommandeFournisseur1Click(Sender: TObject);
begin
Form11.showmodal;
end;
procedure TForm2.ListeClients1Click(Sender: TObject);
begin
Form10.showmodal;
end;
```

```
procedure TForm2.ListeCommandesClients1Click(Sender: TObject);
begin
Form12.showmodal;
end;
procedure TForm2.Inventaire1Click(Sender: TObject);
begin
Form13.showmodal;
end;
procedure TForm2.Calendrier1Click(Sender: TObject);
begin
Form14.showmodal;
end;
procedure TForm2.Timer1Timer(Sender: TObject);
begin
label10.Caption:=timetostr(time);
label13.Caption:=datetostr(date);
end;
procedure TForm2.Calculatrice1Click(Sender: TObject);
begin
winexec('calc.exe',0);
end;
procedure TForm2.Button7Click(Sender: TObject);
begin
Form17.ShowModal;
end;
procedure TForm2.Nomenclature2Click(Sender: TObject);
begin
Form22.ShowModal;
end;
procedure TForm2.Liste1Click(Sender: TObject);
begin
Form24.ShowModal;
end;
procedure TForm2.MatirePremire1Click(Sender: TObject);
begin
Form27.ShowModal;
end;
procedure TForm2.ListeMatirePremire1Click(Sender: TObject);
begin
Form25.ShowModal;
end;
procedure TForm2.dtailCommandes1Click(Sender: TObject);
begin
Form29.showmodal;
end;
procedure TForm2.proposdeSarPro2Click(Sender: TObject);
begin
form31.showmodal;
end;
end.
```

*Stock :*

```
unit Stock;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Db, DBTables, StdCtrls, Mask, DBCtrls, ExtCtrls, Grids, DBGrids, Buttons;
type
  TForm7 = class(TForm)
    GroupBox1: TGroupBox;
    Label1: TLabel;
    Label3: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    DBEdit2: TDBEdit;
    Edit1: TEdit;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    Edit4: TEdit;
    DBLookupComboBox1: TDBLookupComboBox;
    GroupBox4: TGroupBox;
    DBGrid2: TDBGrid;
    BitBtn3: TBitBtn;
    BitBtn4: TBitBtn;
    BitBtn5: TBitBtn;
    BitBtn6: TBitBtn;
    BitBtn7: TBitBtn;
    procedure Button10Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure Button5Click(Sender: TObject);
    procedure Button9Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button7Click(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure Edit4Change(Sender: TObject);
    procedure Edit1Change(Sender: TObject);
    procedure BitBtn3Click(Sender: TObject);
    procedure BitBtn4Click(Sender: TObject);
    procedure BitBtn5Click(Sender: TObject);
    procedure BitBtn6Click(Sender: TObject);
    procedure BitBtn7Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
  Form7: TForm7;
  N1:integer; // On déclare ici que N1 et N2 sont des variables de type integer (entier)
  N2:integer; // C'est à dire que N1 et N2 contiendront des données de type integer (entier)
```

```
  Resultat:integer; // même remarque que pour N1 et N2
  nouvstock: real;
implementation
uses SarPro, datamodule;
{$R *.DFM}
procedure TForm7.Button10Click(Sender: TObject);
begin
Form7.close;
end;
procedure TForm7.Button3Click(Sender: TObject);
begin
DataModule1.Table14.append;
end;
procedure TForm7.Button5Click(Sender: TObject);
begin
DataModule1.Table14.Append;
end;
procedure TForm7.Button9Click(Sender: TObject);
begin
DataModule1.Table14.post;
end;
procedure TForm7.Button2Click(Sender: TObject);
begin
DataModule1.Table14.post;
end;
procedure TForm7.Button7Click(Sender: TObject);
begin
datamodule1.table14.open;
datamodule1.table14.Edit;
datamodule1.table14.fieldbyname('Stock').asinteger:=datamodule1.table14.fieldbyname('Stock').asinteger +
(strtoint(edit4.text));
edit4.clear;
//datamodule1.table3.open;
//datamodule1.table3.Edit;
//if (strtoint(edit4.text)) > datamodule1.table3.fieldbyname('Stock').asinteger then begin
                                                  //     showmessage (Valeur exagérée, valeur incorrecte, veuillez
entrer une autre valeur');
                                                    //  edit4.Clear;
                                                    //  edit4.setfocus;
                                                    //  end
                                                  // else begin
                                                    //
datamodule1.table3.fieldbyname('Stock').asinteger:=datamodule1.table3.fieldbyname('Stock').asinteger      -
(strtoint(edit4.text));
                                                  //       if    datamodule1.table3.fieldbyname('Stock').asinteger <=
datamodule1.table3.fieldbyname('Stock_min').asinteger then showmessage ('stock min très bas);
                                                  //    edit4.clear;
                                                  //    end;

end;
```

```
procedure TForm7.BitBtn1Click(Sender: TObject);
begin
datamodule1.table14.open;
datamodule1.table14.Edit;
datamodule1.table14.fieldbyname('Stock').asinteger:=datamodule1.table14.fieldbyname('Stock').asinteger +
(strtoint(edit1.text));
edit1.clear;
end;
procedure TForm7.BitBtn2Click(Sender: TObject);
begin
datamodule1.table14.open;
datamodule1.table14.Edit;
if (strtoint(edit4.text)) > datamodule1.table14.fieldbyname('Stock').asinteger then begin
                                                    showmessage ('Impossible de retirer cette valeur, veuillez
entrer une autre');
                                                    edit4.Clear;
                                                    edit4.setfocus;
                                                    end
                                                else begin

datamodule1.table14.fieldbyname('Stock').asinteger:=datamodule1.table14.fieldbyname('Stock').asinteger -
(strtoint(edit4.text));
                                                    if      datamodule1.table14.fieldbyname('Stock').asinteger <=
datamodule1.table14.fieldbyname('Stock_min').asinteger then showmessage ('Alerte! Stock de Sécurité
atteint');
                                                    edit4.clear;
                                                    end;
end;

procedure TForm7.Edit4Change(Sender: TObject);
begin
if edit1.text<>'' then BitBtn1.enabled:=true
            else BitBtn1.enabled:=false;
end;
procedure TForm7.Edit1Change(Sender: TObject);
begin
if edit4.text<>'' then BitBtn2.enabled:=true
            else BitBtn2.enabled:=false;
end;
procedure TForm7.BitBtn3Click(Sender: TObject);
begin
DataModule1.Table14.append;
end;
procedure TForm7.BitBtn4Click(Sender: TObject);
begin
DataModule1.Table14.post;
end;
procedure TForm7.BitBtn5Click(Sender: TObject);
begin
Form7.close;
```

```
end;
procedure TForm7.BitBtn6Click(Sender: TObject);
begin
DataModule1.Table14.delete;
end;
procedure TForm7.BitBtn7Click(Sender: TObject);
begin
DataModule1.Table14.edit;
end;
end.
```

## *Calendrier:*

```
unit calendrier;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ComCtrls;
type
  TForm14 = class(TForm)
    MonthCalendar1: TMonthCalendar;
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
  Form14: TForm14;
implementation
{$R *.DFM}
end.
```

## *A Propos :*

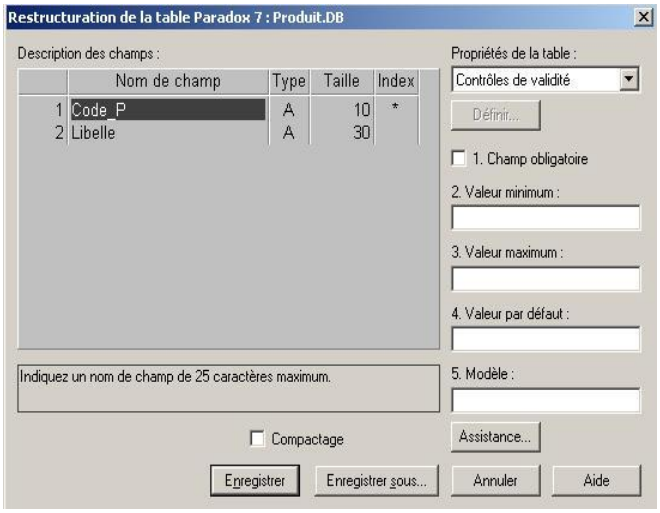
```
unit BonCde;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Qrctrls, QuickRpt, jpeg, ExtCtrls, StdCtrls, Buttons;
type
  TForm31 = class(TForm)
    Panel1: TPanel;
    Image1: TImage;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Label5: TLabel;
    Label6: TLabel;
    Label7: TLabel;
    Label8: TLabel;
```

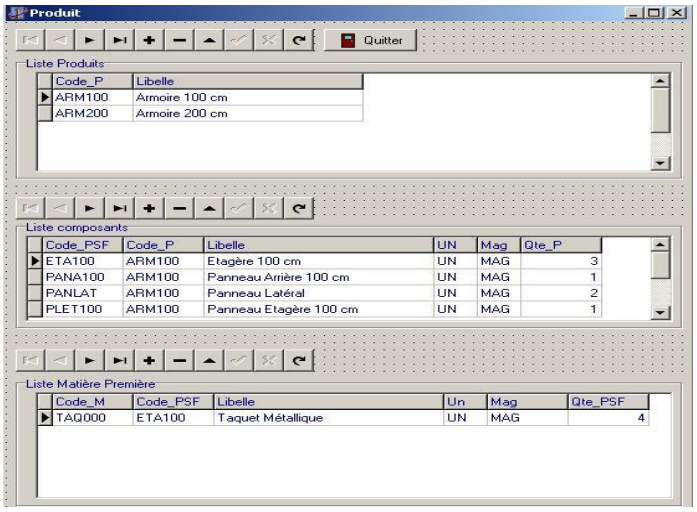
```
    BitBtn1: TBitBtn;
    procedure BitBtn1Click(Sender: TObject);
  private
    { Déclarations privées }
  public
    { Déclarations publiques }
  end;
var
  Form31: TForm31;
implementation
uses datamodule;
{$R *.DFM}
procedure TForm31.BitBtn1Click(Sender: TObject);
begin
form31.Close;
end;
end.
```

# *Annexe B*

## *Les Tables*

*Table1:* Produit




La table produit est définit par le code produit et son libellé, nous indiquerons pour chaque produit les quantités de produits semi-fini et de matières premières dont il est composé

*Table 2 :* Produit Semi-Fini (voir Table 1, tableau de produis semi-finis)

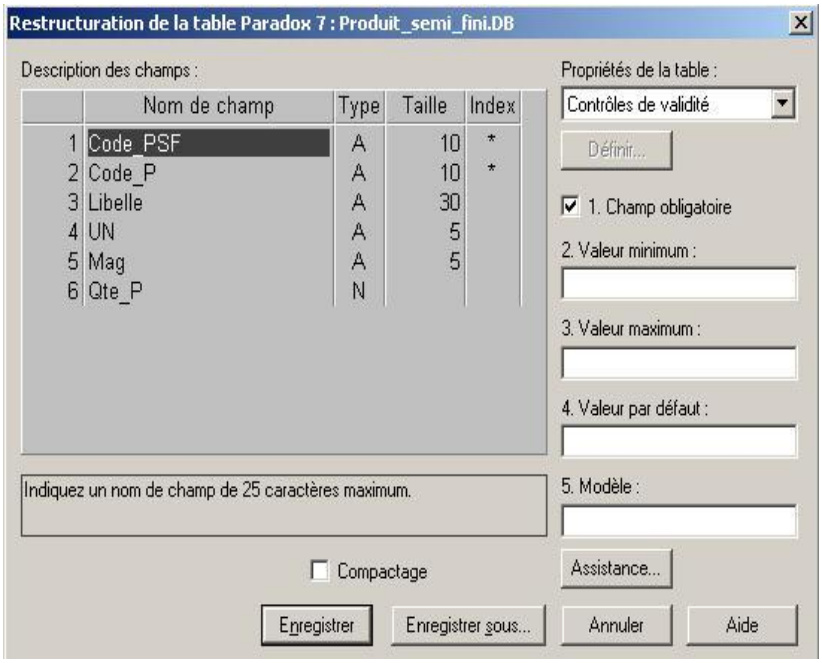


*Table 3 :* Matière Première (voir Table 1, tableau de matières premières)

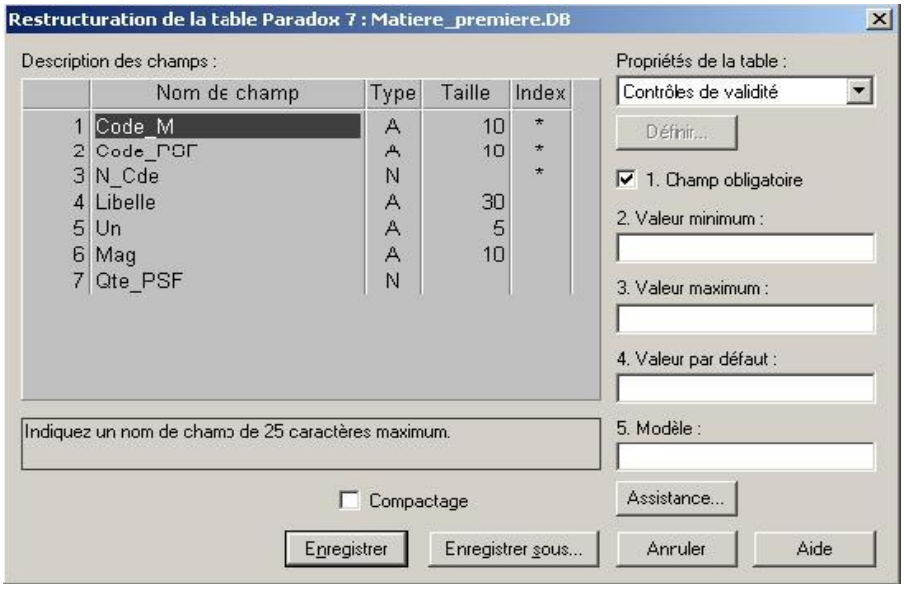
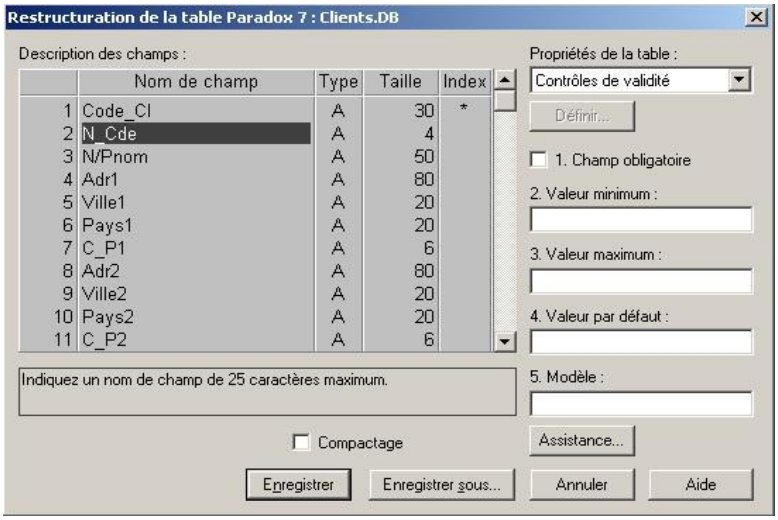



*Table 4 :* La table client contient diverses informations relatives au client.



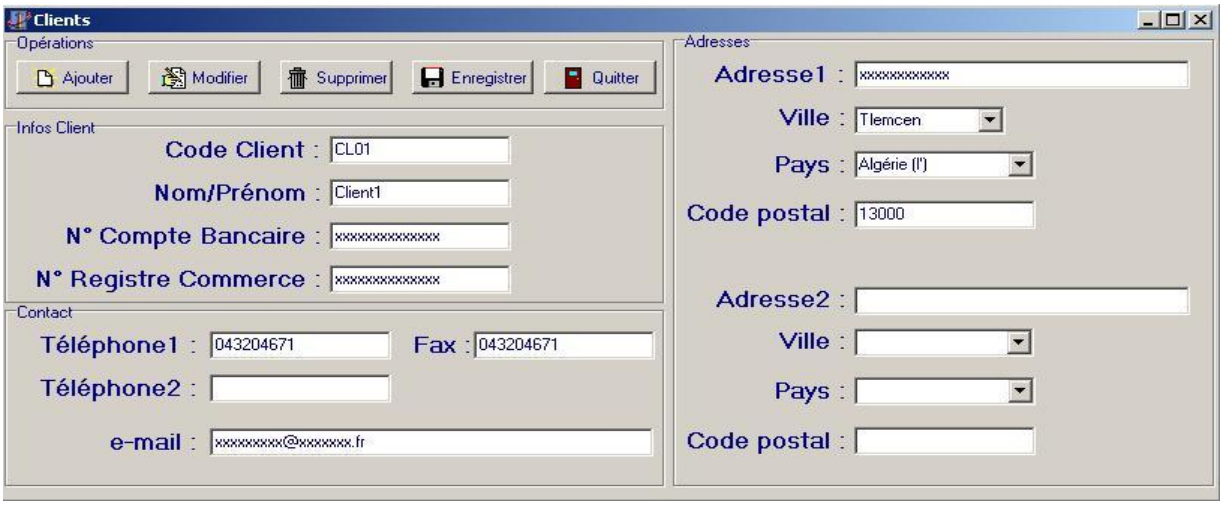
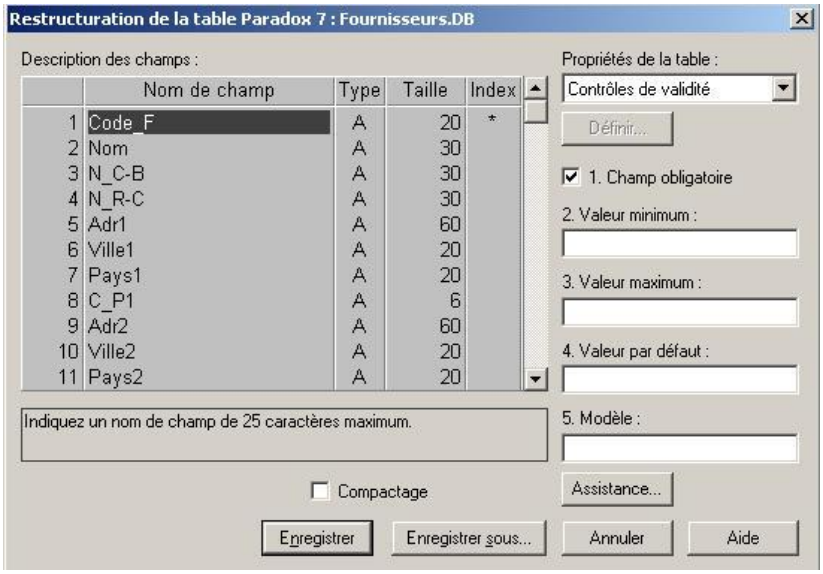



*Table 5 :* La table Fournisseur contient diverses informations relatives au fournisseurs.



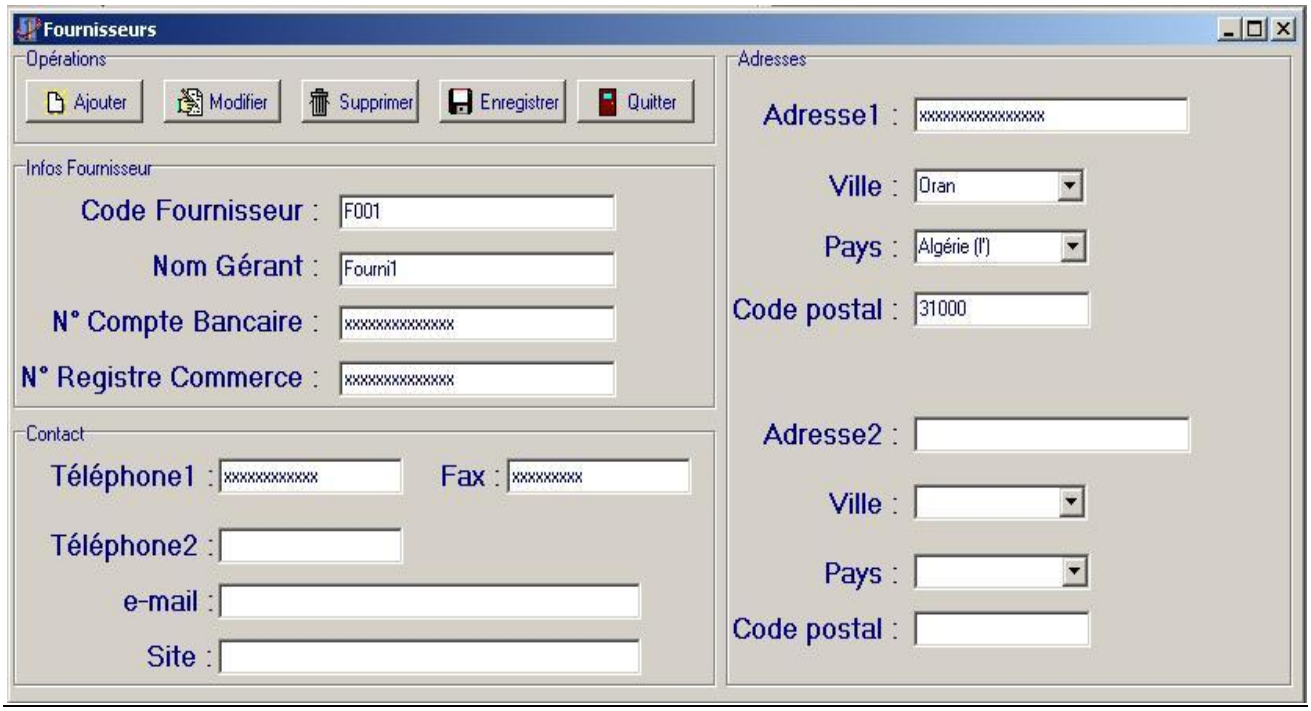
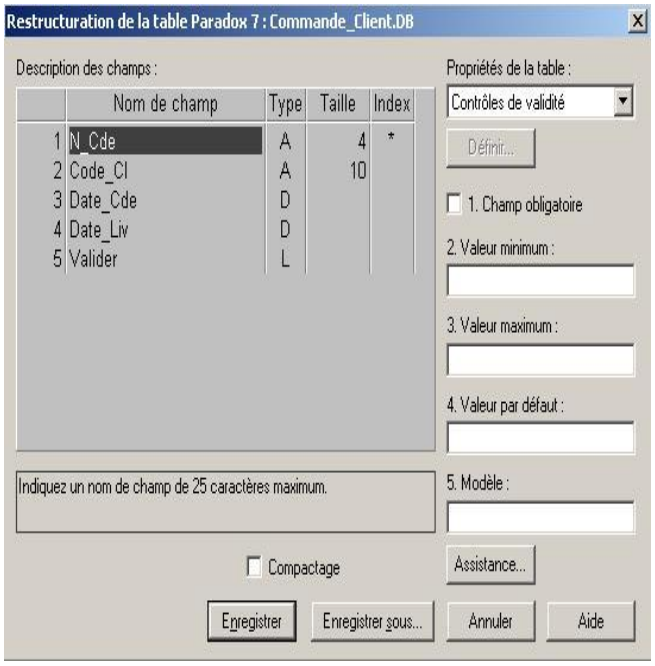
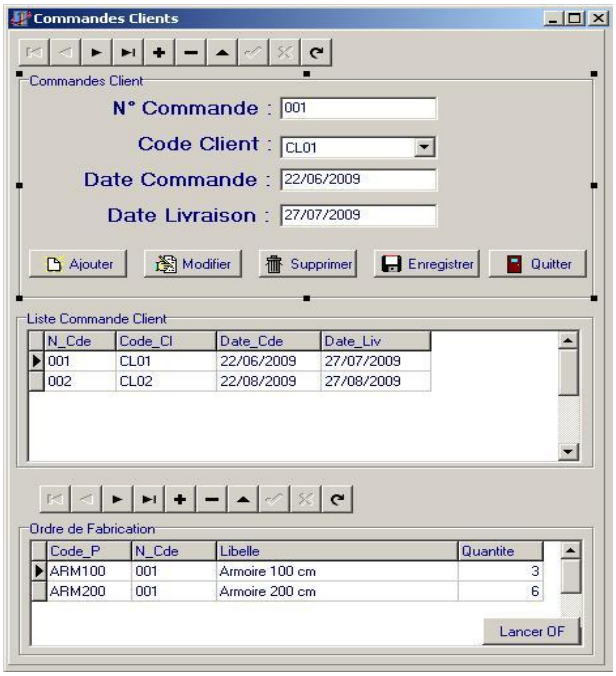



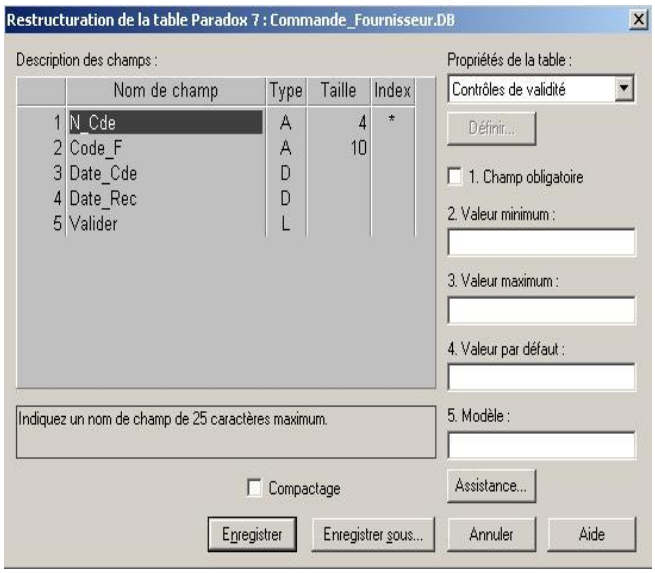
*Table 6 :* Commande Client

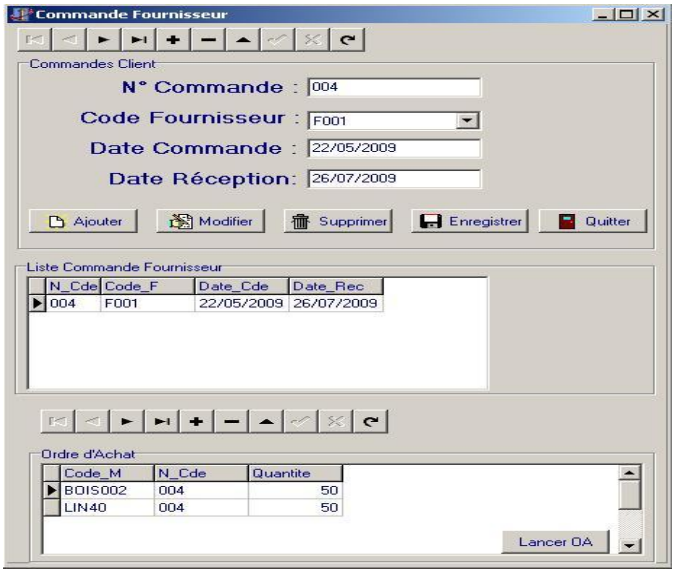



La table commande client concerne les commandes de produits effectuées par le client, chaque commande possède son propre identifiant ainsi une commande ne peut être validée qu'une seule fois et ceci grâce au champs « valider » ; en effet une fois l'ordre de fabrication lancé le champs « valider » passe de '0' logique à '1'.

*Table 7 :* Commande Fournisseur



La table commande fournisseur concerne les commandes de matières premières effectuées par l'entreprise, chaque commande possède son propre identifiant ainsi une commande ne peut être validée qu'une seule fois et ceci en suivant le même principe que la commande client.

*Table 8 :* Ville (contient le nom de toutes les villes de l'Algérie)

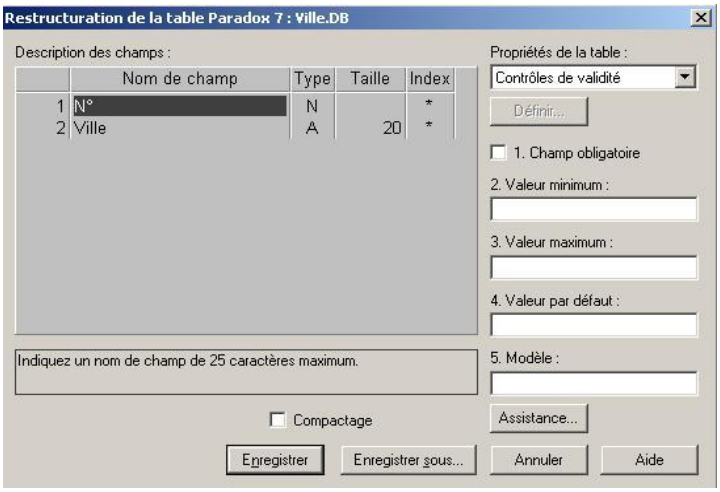



*Table 9 :* Pays (contient le nom de tous les pays du monde)

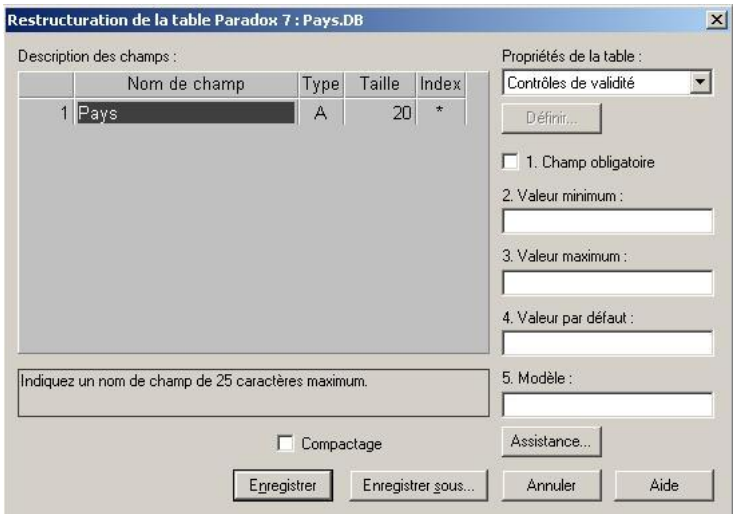


*Table 10 :* Poste de charge (chaque poste est définit par ses propres code et fonction)

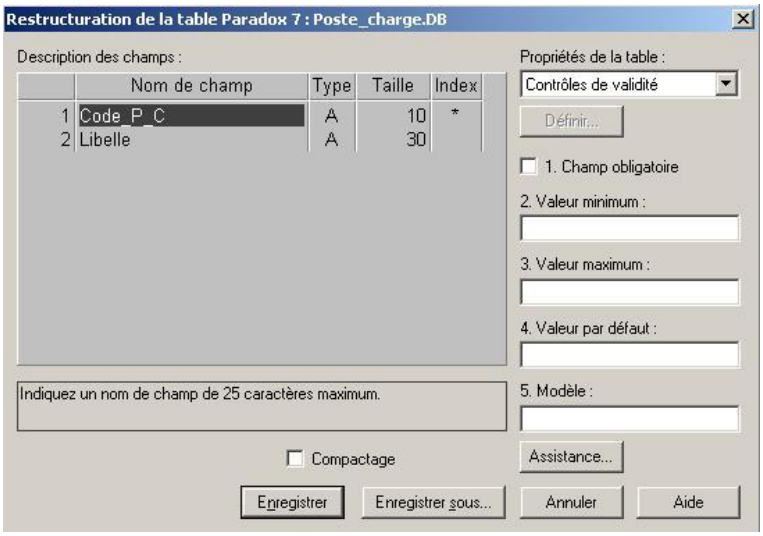



*Table 11 :* Mot de Passe (le mot de passe utilisé dans cette version est « gpao» ; bien sûr, nous pouvons le modifier à tout moment).

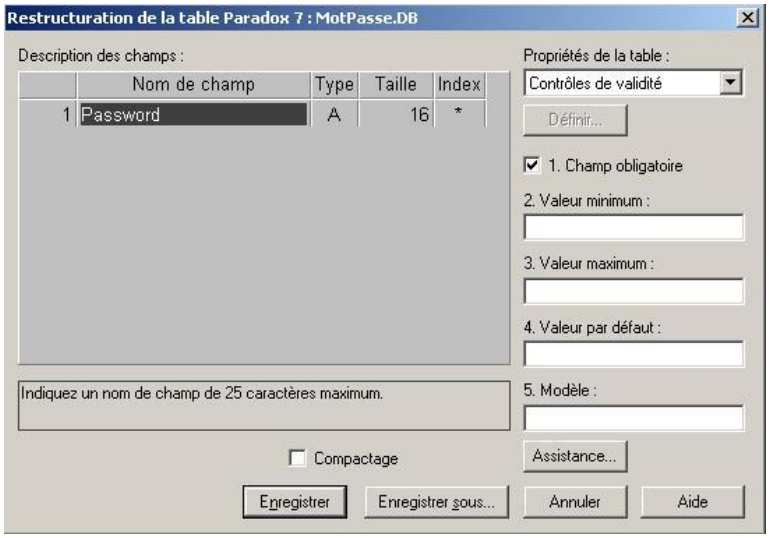


*Table 12 :* Commande Produit (voir Table 6, 2ème tableau)

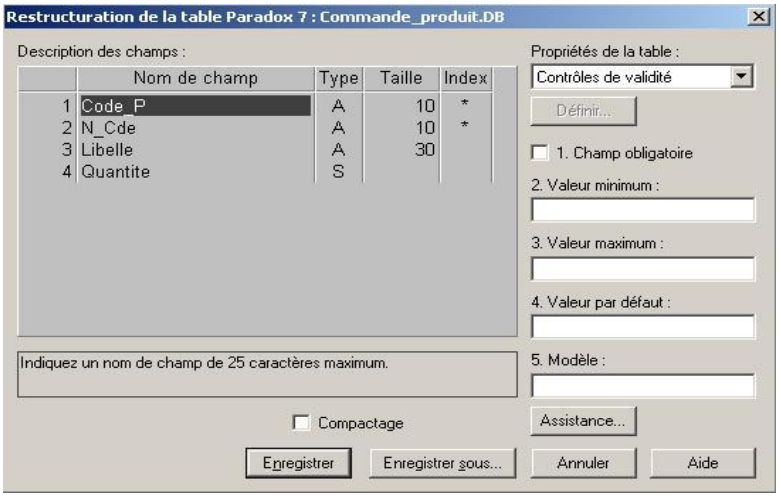



*Table 13 :* Commande matière première (voir Table 7, 2ème tableau)

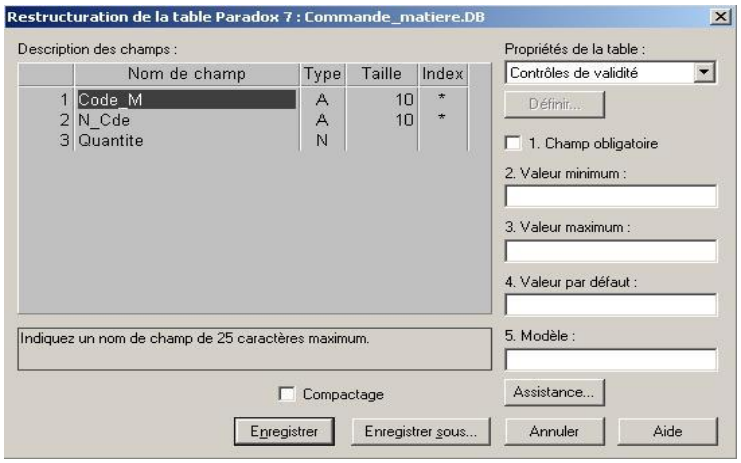
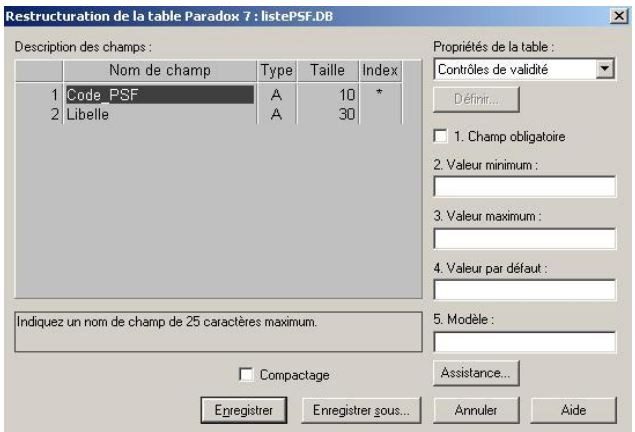



*Table 14 :* Liste des Produits Semi-Finis (Les produits semi-finis sont définis par leurs codes et leur libellé)



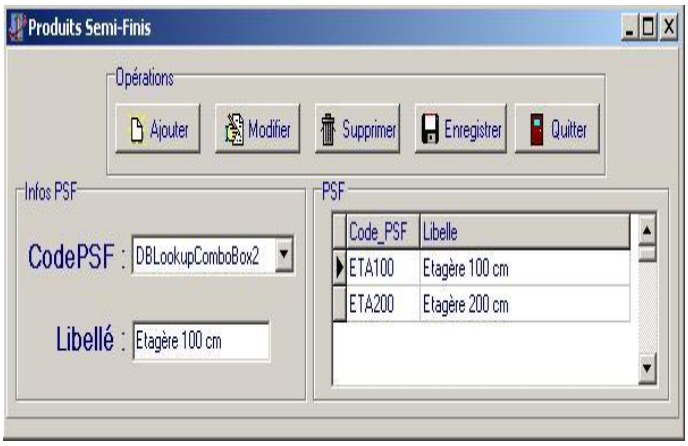
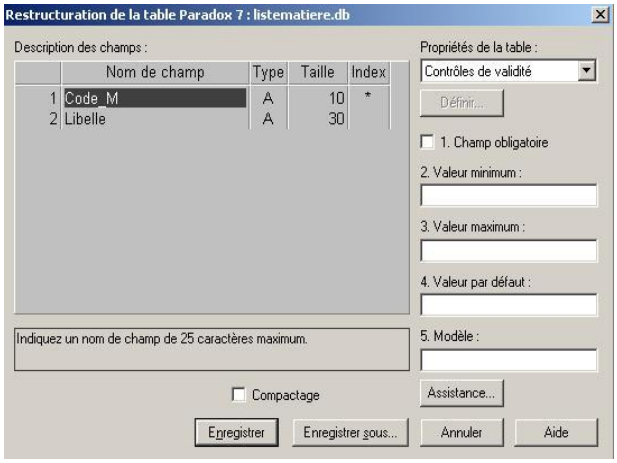
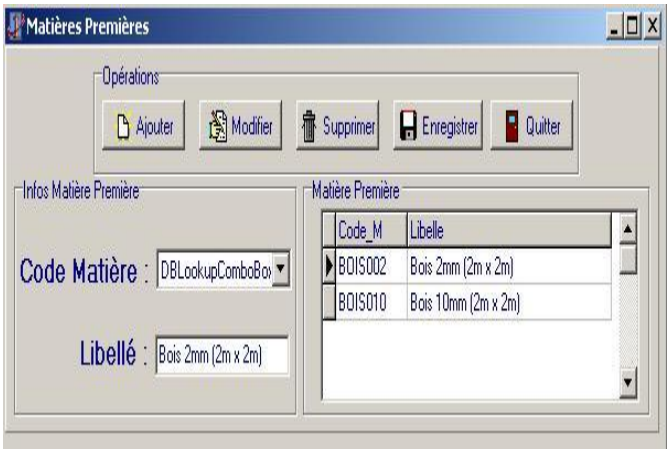



*Table 15 :* Liste des matières Premières (Les matières premières sont définies par leurs codes et leur libellé)




*Tables Temporaires :*

Les tables temporaires se composent de deux types : Deux tables représentant les matières premières et les deux autres les produits semi-finis, elles sont conçus de la même manière que les tables des listes des produits semi-finis et des matières premières.

**Résumé :**

Bien gérer la production d'une entreprise, c'est trouver une solution admissible par rapport à des objectifs fixés tout en réglant un ensemble de conflits de ressources.

La thèse développement d'un logiciel de gestion de production assistée par ordinateur vise à faire connaître d'une part les systèmes de production, leurs fonctions et les démarches de gestion que l'on peut y associer grâce à la réalisation d'une étude bibliographique approfondie sur les systèmes de production et leur gestion, d'autre part les outils informatiques que l'on peut mettre en œuvre pour l'aide à la décision d'où le développement d'un logiciel pouvant assurer les fonctions de base de la gestion de production.

**Mots – clés :** GPAO (Gestion de Production Assistée par Ordinateur) ; Conception ; Delphi

**خلاصة :**

حسن تسيير الإنتاج لمؤسسة ما هو إيجاد حل مقبول فيما يتعلق بالأهداف مع حسم مجموعة من نزاعات على الموارد.

الموضوع المطرح و هو برمجة تسيير الإنتاج، يسعى لتمكيننا من التعرف على كل من نظم الإنتاج، وظائفهم ونهج التسيير التي يمكن أن ترتبط إليها، و هذا يعود إلى الدراسة البيبليوغرافية للأنظمة الإنتاجية من جهة، و من جهة أخرى لأدوات الكمبيوتر التي يمكن توفيرها للمساعدة في اتخاذ قرار، هذا ما ينشئ تطوير برمجية بإمكانها توفير الوظائف أساسية لتسيير الإنتاج.

**الكلمات الرئيسية :** GPAO (إدارة الإنتاج بمساعدة الكمبيوتر) ؛ تصميم ؛ دلفي