TROIS OPERATIONS DE MISE A JOUR D'UNE TABLE AVEC CA-CLIPPER

Sindano W.K. et Hamuli K.
Université Catholique de Bukavu, B.P. 285 - Bukavu, RDC.

Résumé

Une table est un ensemble de couples (clé, informations) où chaque information est accessible par l'intermédiaire de la clé. Les opérations courantes sur une table sont : la création, la modification et la suppression d'une clé ainsi que les informations liées à cette clé. Ces trois opérations classiques servent à la mise à jour de l'ensemble des valeurs d'une table.

CA-Clipper est un compilateur écrit en C, qui est un gestionnaire de bases de données relationnelles où l'ensemble de données est perçu sous forme de tables qui sont utilisées dans de nombreux domaines de la gestion, par exemple la gestion de stock, la comptabilité, paie, etc. Pour l'illustration de ces trois opérations, nous avons considéré le modèle de stock sur les entités famille, sous-famille et article. En outre, ces trois procédures principales : créer_sfam(), modifier_sfam() et supprimer_sfam(), nous avons développé de fonctions complémentaires utiles : consulter_sfam(), recherche(), etc.

Abstract

A table is a set of couples(key, informations) in which information is reached by mean of the key. The current operations to update informations in a table are : creation, modification, and deletion of a key and all informations liked to it.

CA-Clipper is a compiler written in C language; it is a relationnal database manager in which the set of data appears as a table and it is used in many fields of management that is for example the stock management. In the next lines we offer to CA-Clipper developers the three table updating operations, most important procedures of basic module in programming of management softwares like accountancy, pay, stock, specificatio, invoicing, etc. The illustration we took for these three operations is a stock model during implementation of a subfamily of article, in form of three subroutines : creer_sfam(), modifier_sfam() and supprimer_sfam(). There are also other important functions called by the three procedures : consulter_sfam(), recherche(), etc.

1. Introduction

• Le CA-Clipper (marque déposée de Computer Associates International, Inc.) est avant tout un langage destiné aux développeurs, il repousse fort loin les limites des applications de BASE par ses nombreuses améliorations notamment [1] : un interfaçage avec le langage C et l'assembleur, les réseaux, l'utilisation avec des systèmes d'exploitation multiutilisateurs compatibles DOS (exemple Novell), l'approche de la programmation orientée objet. Tous ces points permettent aux développeurs de produire des logiciels performants en mettant l'accent sur l'importance du découpage d'une application en plusieurs unités logiques plus petites ou modules et sur la définition de relations définissant les liens entre ces diverses unités. Cette programmation est désignée sous le nom de programmation modulaire. Aujourd'hui les travaux sont entrepris pour donner une conception unique standardisée de la notion de modularité, bien qu'il n'y ait pas à l'heure actuelle le langage susceptible de satisfaire tout le monde. Les avantages apportés par cette notion peuvent s'énoncer par les principes suivants : principe de simplicité, de transparence, de lisibilité et d'efficacité(2).

L'enregistrement sous forme d'une information structurée d'un événement d'activité s'effectue par l'écriture sur un support quelconque des valeurs de ses caractéristiques(3). Chaque fois qu'un événement d'activité concernant un élément est pris en compte dans le système d'information, il est nécessaire de créer (ajouter, insérer), modifier ou supprimer la valeur d'une ou plusieurs caractéristiques de situation de cet élément. Cette transformation (modification) de valeur est appelée mise à jour.

Dans cette optique nous proposons aux développeurs de CA-Clippeer les trois opérations de mise à jour(4) des tables qui font partie de procédures du module de base pour la programmation de logiciels de gestion (comptabilité, stock, facturation, devis, paie, etc.)(5). Les fonctions types représentant ces trois opérations sont illustrées sur un modèle de stock lors d'implémentation d'une de sous-famille d'articles: Créer_sfam(), modifier_sfam() et supprimer_sfam(). A ces trois procédures, il est conseillé d'enjoindre une quatrième procédure consulter_sfam() qui sont précédées par d'autres fonctions utilitaires comme initialise(), recherche(), masque(), etc ...

2. TABLES ET INDEXES

2.1. Codification

Une table appartenant à une base de données relationnelles doit avoir un champ clé qui facilite la recherche de données, généralement cette clé est codifiée de manière à avoir de préférence une longueur assez courte (moins de dix caractères significatifs). Les méthodes de codifications sont nombreuses(6), nous utilisons la méthode analytique (facile à retenir), elle consiste à définir les articles par une série de caractéristiques. On procède le plus souvent par famille ou série :

Famille	Sous-famille	Article	Caractéristique
AL	BO	JW	008
AL	BO	JW	072
AL	BO	JW	100
AL	BO	Wh	072

Le code du second article AL-BO-JW-072 désigné par :

AL : famille Alimentation
BO : sous-famille Boisson
JW : article Johny Walker
072 : caractéristique bouteille de 72 cl.

Malgré la diversité de codifications, un bon code doit obéir à trois règles :

* désigner sans ambiguïté chaque article ;
* être stable ;
* être pratique.

2.2. Tables

Pour la gestion de différents articles, nous avons besoin de quatre tables (relations) telle que suggéré par la codification : tables pour les familles, sous-familles, articles et caractéristiques. Les quatre tables seront constituées de champs suivants :

Famille = {code_fam, désignation}
Sousfam = {code_sfa, désignation}
Article = {code_art, désignation}
Caracter ={ code_car, désignation, unité, condit, qte_crit}

La création et l'indexation de ces tables peut se faire par un module écrit en CA-Clipper de manière très simple(7), il sera exécuté à chaque installation du logiciel :

```
&& Longueurs de différents codes
LCdFa : = 1
LCdSf : = 3
LCdAr : = 6
LCdCa : = 9

&& Création de la table famille et son index
structure := {{"code_fam", "C", LCdFa,0}, { "designation", "C",25,0}}
DBCREATE("famille",structure)
USE famille
INDEX ON code_fam TO famille
CLOSE DATA

&& Création de la table sous-famille et son index
structure:= {{"code_sfa", "C",LCdSf,0}, {"designation", "C",25,0}}
DBCREATE("sfamille", structure)
USE sfamille
INDEX ON code_sfa TO sfamille
CLOSE DATA

&& Création de la table article et son index
structure:= {{ "code_art","C",LCdAr,0}, {"designation","C"25,0}}
DBCREATE("article", structure)
USE article
INDEX ON code_art TO article
CLOSE DATA
```

```
&& Création de la table caractéristique et son index
structure := {{ “code_car”,”C”,LCdCa,0},
{“designation”,”C”,25,0},;
{“condit”,”C”,25,0}, {“unite”,”C”,3,0},
{“qte_crit”,”N”,3,0}}
DBCREATE(“caracter”,structure)
USE caracter
INDEX ON code_car TO caracter
CLOSE DATA
```

3. Algorithmes

Les algorithmes proposés peuvent être traduits dans différents langages de programmation structurés classique comme l’Algol, Le Pascal, le C, etc ... (8),(9).

3.1. Procédure créer sous famille

```
Procédure Créer_sfam()
Initialiser_variables()
Masque_saisie()
TANT QUE choix ≠terminer
saisir la clé
Recherche(clé, table)
SI clé trouvée
afficher information : famille, sous famille message d’erreur
SI NON
saisir les informations sous famille
                FIN SI
                Activer_choix()
                SI choix == sauver
                        Ecrire_enregistrement()
                FIN SI
                FIN BOUCLE
        Fin procédure
```

3.2. Procédure modifier sous famille

```
Procédure Modifier_sfam()
        Initialiser_variables()
        Masque_saisie()
        TANT QUE choix ≠ terminer
                saisir la clé
                Recherche(clé, table)
                SI clé trouvé
                        afficher information : famille, sous famille
                        modifier les informations sous famille
                SI NON
                        messages d’erreur
                FIN SI
                Activer_choix()
                SI choix == sauver
                        Ecrire_enregistrement()
                FIN SI
```

```
            FIN BOUCLE
        Fin procédure
```

3.3. Procédure consulter sous famille

```
    Procédure Consulter_sfam()
        Initialiser_variables()
        Masque_saisie()
        TANT QUE choix ≠terminer
            saisir la clé
            Recherche(clé, table)
            SI clé trouvée
                afficher information : famille, sous famille
            SI NON
                messages d'erreur
            FIN SI
            Activer_choix()
        FIN BOUCLE
    Fin procédure.
```

3.4. Procédure supprimer sous famille

```
    Procédure Supprimer_sfam()
        Initialiser_variables()
        Masque_saisie()
        TANT QUE choix ≠terminer
            saisir la clé
            Recherche(clé, table)
            SI clé trouvée
                afficher information : famille, sous famille
            SI NON
                messages d'erreur
            FIN SI
            Activer_choix()
            SI choix== supprimer
                Ecrire_enregistrement()
            FIN SI
        FIN BOUCLE
    Fin procédure
```

4. BIBLIOTHEQUE DE MISE A JOUR EN CA-Clipper

L'écriture de différents modules respecte les conventions suivantes : les mots clés du langage CA-Clipper sont présentés en italique gras majuscule, les variables mémoires contiennent un premier caractère minuscule qui représente le type de la variable (c pour alphanumérique, n pour numérique et l pour logique) suivi.

L'interface utilisateur est l'élément de référence qui permet à l'utilisateur de juger la qualité d'un système(10). L'interface interactive est utilisé dans les différentes procédures principales en combinant la technique de formulaires et de menus.

4.1. Procédure créer sous famille

****

```
    * La procédure Creer_sfam() : - Elle crée une sous famille après avoir vérifié que
la famille
    *                                existe et que la clé de la sous famille à créer
n'existe pas.
    *                                 - Elle fait appel aux fonctions et procédures
suivantes:
    *                                       Masque(), Init-variable(), Recherche(),
Ecrire_Enreg(),
    *                                                                       Mssg().
*************************************************************************/

            PROCEDURE Creer_sfam ()
            LOCAL GetList:= {} , nA
            PUBLIC nNbr_car, cCode_sfam, cDesi_sfam, cDesi_fam, nPx, nPy
            CLS
            SETCURSOR(SC_NORMAL)
            Masque(2,"Créer une sous famille")
            Init_Variable(2)
            DO WHILE.T.
                @nPx+2,nPy GET cCode_sfam PICT Format(1,ncar,0) VALID
                Recherche(cCode_sfam,3,.F.)
                @nPx+6,nPy GET cDesi_sfam PICT "@!"
                READ

                IF(LASTKEY()==K_ESC.OR.LEN(ALLTRIM(cCode_sfam))
!=nNbr_car)
                EXIT
                ENDIF
                    nA:=Mssg(1)
                    DO CASE
                        CASE nA==1
                                Ecrire_Enreg(4)
                                Init_Variable(2)
                                @nPx+4,nPy SAY cDesi_fam
                                @nPx+6,nPy SAY cDesi_fam
                         CASE nA==2
                        CASE nA==3
                            Ecrire_Enreg(4)
                            EXIT
                        OTHERWISE
                            EXIT
                    ENDCASE
            ENDDO
            SETCURSOR(SC_NONE)
            CLS
            RETURN
```

4.2. Procédure modifier sous famille

```
    /*************************************************************************
    * La procédure Modifier_sfam() : - Elle modifie une sous famille après avoir
vérifié que la clé
    *                                de la sous famille à modifier existe.
```

```
    *                                   - Elle fait appel aux fonctions et procédures
suivantes :
    *                                       Masque(), Init_variable(), Recherche(),
Ecrire_Enreg(),
    *                                  Mssg().
     ************************************************************************/
            PROCEDURE Modifier_sfam ()
            LOCAL GetList := {} ,nA
            PUBLIC nNbr_car, cCode_sfam, cDesi_sfam, nPx, nPy

            CLS
            SETCURSOR(SC_NORMAL)
            Masque(2,"Modifier une sous famille")
            Init_Variable(2)
            DO WHILE.T.
               @nPx+2,nPy GET cCode_sfam PICT Format(1,ncar,0) VALID
                     Recherche(cCode_sfam, 4,.T.)
              @nPx+6,nPy GETcDesi_sfam PICT"@!"
             READ

             IF (LASTKEY()==K_ESC.OR.LEN(ALLTRIM(cCode_sfam))
                              !=nNbr_car)
                        EXIT
             ENDIF
            nA := Mssg(1)
            DO CASE
                     CASE nA == 1
                        Ecrire_Enreg(5)
                        Init_Variable (2)
                        @nPx+4,nPy SAY cDesi_fam
                        @nPx+6,nPy SAY cDesi_fam
                      CASE nA == 2
                      CASE nA == 3
                        Ecrire_Enreg(5)
                        EXIT
               OTHERWISE
                        EXIT
                  ENDCASE
            ENDDO
            SETCURSOR(SC_NONE)
            CLS
            RETURN
```

4.3. Procédure consulter sous famille

```
/*********************************************************************
* La procédure Consulter_sfam(): - Elle consulte une sous famille après avoir vérifié que la
*                                  clé de la sous famille à consulter existe.
*                                  - Elle fait appel aux fonctions et procédures suivantes :
*                                  Masque(), Init_variable(), Recherche(), Mssg().
*********************************************************************/
PROCEDURE Consulter_sfam()
LOCAL GetList:= {}, nA
PUBLIC nNbr_car, cCode_sfam, cDesi_sfam, cDesi_fam, nPx, nPy

CLS
SETCURSOR(SC_NORMAL)
Masque(2, “Consulter une sous famille”)
Init_Variable(2)
DO WHILE.T.
    @nPx+2,nPy GET cCode_sfam PICT Format(1,ncar,0) VALID Recherche(cCode_sfam, 4,.T.)
    READ

    nA := Mssg(1)
    DO CASE
        CASE nA==1
            Ecrire_Enreg(2)
            Init_Variable(1)
            @nPx+4,nPy SAY cDesi_fam
            @nPx+6,nPy SAY cDesi_sfam
        CASE nA==2
        CASE nA==3
            Ecrire_Enreg(2)
            EXIT
        OTHERWISE
            EXIT
    ENDCASE
ENDDO
SETCURSOR(SC_NONE)
CLS
RETURN
```

4.4. Procédure supprimer sous famille

```
/*********************************************************************
* La procédure supprimer_sfam(): - Elle supprime une sous famille après avoir vérifié que la
*                                  clé de la sous famille à supprimer existe.
*                                  - Elle fait appel aux fonctions et procédures suivantes :
*                                  Masque(), Init_variable(), Recherche(), Ecrire_Enreg(),
*                                  Mssg().
*********************************************************************/
```

```
PROCEDURE Supprimer_sfam()
LOCAL GetList :=  , nA
PUBLIC nNbr_car, cCode_sfam, cDesi_fam, cDesi_sfam, nPx, nPy

CLS
SETCURSOR(SC_NORMAL)
Masque(1, “Supprimer une sous famille”)
Init_Variable(1)
DO WHILE.T.
        @nPx+2,nPy GET cCode_sfam PICT Format(1,ncar,0) VALID
                    Recherche(cCode_sfam, 2,.T.)
        READ

        IF(LASTKEY()==K_ESC.OR.LEN(ALLTRIM(cCode_sfam))
        !=nNbr_car)
            EXIT
        ENDIF
        nA:=Mssg(1)
        DO CASE
             CASE nA==1
                  b:=Mssg(2)
                   IFb==2
                       Ecrire_Enreg(6)
                        Init_Variable(2)
                        @nPx+4,nPy SAY cDesi_fam
                        @nPx+6,nPy SAY cDesi_sfam
                    ENDIF
                CASE nA == 2
                CASE nA == 3
                      b:=Mssg(2)
                      IFb== 2
                         Ecrire_Enreg(6)
                      ENDIF
                      EXIT
                OTHERWISE
                       EXIT
        ENDCASE
ENDDO
SETCURSOR(SC_NONE)
CLS
RETURN
```

4.5 Fonctions et procédures utilitaires

4.5.1. Fonction Recherche

```
 /*************************************************************************
```

* La fonction Recherche retourne res_rech une valeur logique Vrai ou Faux si respectivement
* le résultat de la recherche correspond ou non au résultat attendu.
* Les paramètres utilisés :
* val_clef : valeur de la clé de recherche
* num_ovrt : numéro d’ordre d’ouverture de la table de recherche

```
*                   res_att                 : résultat attendu de la recherche
* Les fonctions appelées :
*                   Ouvrir_Table, Trouver_Enreg(), Afficher_rubr(), Mssg().
 ********************************************************************/

            FUNCTION Recherche(val_clef, num_ovrt, res_att)
            LOCAL res, res_rech, trvr

                    res_rech:=.F.
                    IF Ouvrir_Table(num_ovrt)
                        trvr:=Trouver_Enreg(val_clef)
            IF trvr==.T.
                res:=Afficher_rubr(val_clef, num_ovrt, res_att, 1)
            ELSE
                res:=Afficher_rubr(val_clef, num_ovrt, res_att, 0)
            ENDIF
            IF res_att==res
                res_rech:=.T.
      ELSE
          Mssg(0)
      ENDIF
    ENDIF
  RETURN res_rech
```

4.5.2. Fonction Ouvrir table

```
/*************************************************************************
* La fonction Ouvrir_Table retourne res_ouvrir une valeur logique Vrai si la table est
*                              ouverte et si non elle retourne la valeur Faux.
*  Le paramètre utilisé :
*                  num_ovrt   : numéro d'ordre d'ouverture de la table de recherche
*************************************************************************/
        FUNCTION Ouvrir_Table(num_ovrt)
        LOCAL res_ouvrir
                res_ouvrir:=.F.
                DO CASE
                     CASE num_ovrt==1
                          USE famille INDEX cCode_fm
                           IF USED()
                              res_ouvrir:=.T.
                           ENDIF
                    CASE num_ovrt==2
                          USE famille INDEX cCode_fm
                          IF USED()
                              res_ouvrir:=.T.
                          ENDIF
                    CASE num_ovrt==3
                          USE sfamille INDEX cCode_sfm
                           IF USED()
                              res_ouvrir:=.T.
                           ENDIF
                     CASE num_ovrt==4
                          USE sfamille INDEX cCode_sm
```

```
                    IF USED()
                        res_ouvrir:=.T.
                    ENDIF
                OTHERWISE
                    Mssg(1)
            ENDCASE
        RETURN res_ouvrir
```

4.5.3. Fonction Trouver enregistrement

```
/*************************************************************************
* La fonction Trouver_Enreg retourne res_trvr une valeur logique Vrai si la clé
* recherchée est trouvée si non elle retourne la valeur Faux.
* Le paramètre utilisé :
*                va_clef : valeur de la clé de recherche
*************************************************************************/
    FUNCTION Trouver_Enreg(val_clef)
    LOCAL tRes_trvr, type_chmp, tRech_auto
        tRes_trvr:=.F.
        tRech_auto:=.F.
        type_chmp:=TYPE(val_clef)
        IF type_chmp=='C'
            IF LEN(ALLTRIM(val_clef)) !=0
                tRech_auto:=.T.
            ENDIF
        ENDIF
        IF tRech_auto
            SEEK val_clef
        IF FOUND()
            tRes_trvr :=.T.
        ENDIF
      ENDIF
    RETURN tRes_trvr
```

4.5.4. Fonction Afficher rubriques

```
/*************************************************************************
* La fonction Afficher_rubr retourne res_affiche une valeur logique Vrai si la
recherche
* donne le résultat attendu, elle retourne la valeur logique Faux dans le cas contraire.
* Le paramètre utilisé :
*                val_clef     : valeur de la clé de recherche
*                num_ovrt     : numéro d'ordre d'ouverture de la table de recherche
*                res_att      : résultat attendu de la recherche
*                mode         : affichage de tous les éléments si mode = 1, si non une
*                               partie trouvée
* Les fonctions appelées :
*                 Ouvrir_Table, Trouver_Enreg(), Mssg().
*************************************************************************/
    FUNCTION Afficher_rubr(val_clef, num_ovrt, tRes_att, mode)
    FIELD code
    LOCAL tRes_affiche
    PUBLIC nPx, nPy, cDesi_fam, cDesi_sfam, LCdFam
        tRes_affiche:= !tRes_att
```

```
IF mode == 1
    tRes_affiche:=.T.
     DO CASE
         CASE num_ovrt == 1
         CASE num_ovrt == 2
            CASE num_ovrt == 3
                cDesi_sfam := designation
                 USE famille INDEX cCode_fam
                 SEEK SUBSTR(val_clef, 1, LCdFam)
                 cDesi_sfam := designation
                 @nPx+4, nPy SAY cDesi_fam
                 @nPx+6, nPy SAY cDesi_sfam
           CASE num_ovrt == 4
           OTHERWISE
                Mssg(1)
          ENCASE
         ELSE
             DO CASE
                 CASE num_ovrt == 1
                      tRes_affiche: = tRes_att
              CASE num_ovrt == 3
                     cDesi_sfam : = designation
                     IF Ouvrir_Table (1)
                         trvr      :=      Trouver_Enreg(SUBSTR(va_clef,      1,
                                 LCdFam))
                          IF trvr
                             cDesi_fam : =designat
                             @nPx+4, nPy SAY cDesi_fam
                             tRes_affiche :=tRes_att
                     ENDIF
                           ENDIF
                        CASE num_ovrt == 4
                         OTHERWISE
                             Mssg(1)
                     ENDCASE
                     ENDIF
                      CLOSE DATA
               RETURN tRes_affiche
```

4.5.5. Fonction Message

```
/*********************************************************************
* La fonction Message retourne num_mssg le numéro du message choisi selon la
valeur
* du paramètre cas.
*
* Le paramètre utilisé :
*                  cas                  : le numéro d'un groupe de choix de messages
*                  num_msg  : numéro du message
*                  scolor       : nouvelle couleur du message (clignotement)
*                  ecran        : conservation de l'écran avant message
* Les fonctions appelées :
*                   Cadre(), Mssg().
```

```
**********************************************************************
        FUNCTION Message(cas)
        PRIVATE ecran, num_msg, scolor

                DO CASE
                     CASE cas == 0
                          ecran := SAVESCREEN(0,0,24,79)
                          Cadre(0,58,2,79,"",1)
                          scolor :=SETCOLOR("GR+*/B+")        && Clignotement
                          @()1,59 SAY "Données Incorrectes"
                          SETCURSOR(SC_NONE)
                          INKEY(60)
                          SETCOLOR(scolor)
                          SETCURSOR(SC_ NORMAL)
                          RESTSCREEN(0, 0, 24, 79, ecran)
                          num_msg = 0
                       CASE cas == 1
                            ecran := SAVESCREEN(0,0,24,79)
                            Cadre(19,64,24,7,"",2)
                            @20,65 PROMPT ' Continuer '
                            @21,65 PROMPT ' Corriger '
                              @22,65 PROMPT ' Terminer '
                            @23,65 PROMPT ' Abandonner  '
                            MENU TO num_msg
                            RESTSCREEN(0, 0, 24, 79, ecran)
                       CASE cas == 2
                            ecran :=SAVESCREEN(0,0,24,79)
                            Cadre(19,49,23,79,"",2)
                            @20,51 SAY ' Confirmer la SUPPRESSION '
                            @22,59 PROMPT ' NON '
                            @22,66 PROMPT ' OUI '
                            MENU TO num_msg
                            RESTSCREEN(0, 0, 24,79, ecran)
                       CASE cas == 3
                       OTHERWISE
                             Mssg(1)
                      ENDCASE
                RETURN num_msg
```

4.5.6. Procédure Initialiser variables

```
/**********************************************************************
* La procédure InitVar initialise les différentes variables publiques suivant le
paramètre choix.
**********************************************************************/
        PROCEDURE InitVar(choix)
        PUBLIC nb_car, cCode_fam, vcDesi_fam, cCode_sfam, vcDesi_sfam
                DO CASE
                    CASE choix == 1
                         nNbr_car : = 2
```

```
                    cCode_fam : = SPACE(ncar)
                    vcDesi_fam : = SAPCE(25)
                CASE choix == 2
                    nNbr_car : = 5
                    cCode_sfam : =SPACE(ncar)
                    vcDesi_sfam : =SAPCE(25)
                OTHERWISE
               ENDCASE
    RETURN
```

4.5.7. Procédure Masque de saisie

```
/*********************************************************************
* La procédure Masque affiche les différents écrans de saisie et leurs titres.
* Le paramètre utilisé :
*                num                : le numéro du masque
*               titre          : le titre du masque
*               nPx, nPy       : les coordonnées d'affichage
*               ecran          : conservation de l'écran avant message
* La fonction appelée :
*                Cadre().
*********************************************************************/
    PROCEDURE Masque(num,titre)
    PUBLIC nPx, nPy

        DO CASE
            CASE num = 1
                 nPx : = 6
                nPy : = 2
                Cadre(nPx,nPy,nPx+12, nPy+73,titre,5)
                @nPx+02,nPy+01 SAY "          CODE FAMILLE : "
                @nPx+04,nPy+01 SAY "DESIGNATION FAMILLE:"
                nPy += 20
            CASE num = 2
                 nPx:=6
                nPy:=2
               Cadre(nPx,nPy,nPx+14,nPy+73,titre,5)
               @nPx+02,nPy+01 SAY "          CODE SOUS FAMILLE: "
               @nPx+04,nPy+01 SAY "       DESIGNATION FAMILLE: "
               @nPx+04,nPy+01 SAY "DESIGNATION SOUS FAMILLE: "
               nPy += 20
            CASE num = 3
            OTHERWISE
         ENDCASE
    RETURN

    FUNCTION Cadre(x1,y1,x2,y2titre,type)

        IF (x1<0.OR. x1>24)
            x1 :=0
        ENDIF
        IF (x2<0.OR. x2 >24)
           x1 :=24
```

```
            ENDIF
            IF (y1<0.OR.y1 > 79)
                y1 := 0
            ENDIF
            IF (y2<0.OR.y2 > 79)
                y2 :=79
            ENDIF

            DO CASE
                CASE type == 1
                    @x1,y1,x2,y2 BOX B_SINGLE
                CASE type == 2
                    @x1,y1,x2,y2 BOX B_DOUBLE
                CASE type == 3
                    @x1,y1,x2,y2 BOX B_SINGLE_DOUBLE
                CASE type == 4
                    @x1,y1,x2,y2 BOX B_DOUBLE_SINGLE
                CASE type == 5
                    @x1,y1,x2,y2 BOX B_FULL
                OTHERWISE
            ENDCASE
            IF LEN(ALLTRIM(titre)) != 0
                    Centre(x1,y1,y2,titre)
            ENDIF
        RETURN NIL
*
        FUNCTION Centre(x1,y1,y2,titre)
        LOCAL nPosition
                nPosition:=(y1 + y2 - LEN(titre) + 1)/2
                IF nPosition < 2
                    nPosition :=y1 + 1
                ENDIF
                @x1, nPosition SAY titre
        RETURN NIL
```

4.5.8. Procédure Ecrire Enregistrement

```
/*************************************************************************
* La procédure Ecrire_Enreg écrit dans la table choisie (création, modification et
*                                                    suppression d'un enregistrement).
* Le paramètre utilisé :
*                  chois        : le numéro d'ouverture d'une table
*************************************************************************/

        PROCEDURE Ecrire_Enreg(choix)
        FIELD code_fam, code_sfam, designation
        PUBLIC cCode_fam, cCode_sfam, cDesignat

            DO CASE
                CASE choix == 1          && 1,2, et 3 réservés pour la famille
                CASE choix == 2
                CASE choix == 3
                CASE choix == 4
```

```
                USE sfamille INDEX code_fm
                APPEND BLANK
                REPLACE code_sfam WITH cCode_sfam
                REPLACE designation WITH cDesignat
            CASE choix == 5
                USE sfamille INDEX code_fm
                SEEK cCode_sfam
                REPLACE designation WITH cDesignat
            CASE choix == 6
                  USE sfamille INDEX code_fm
                  SEEK cCode_sfam
                DELETE
                PACK
            OTHERWISE
        ENDCASE
RETURN
```

5. DISCUSSION

Les applications classiques d'une organisation telles que la paie, la gestion du personnel, la comptabilité, la gestion des commandes (stock) sont décomposables ; le niveau de décomposition le plus bas correspond au découpage en opérations. Ainsi, les trois opérations développées sont indispensables pour ces différentes activités d'une organisation : pour la paie, les entités à subir les opérations sont le service, agent, les différents codes, etc ; pour la comptabilité, le compte, le compte principal, le sous compte ou encore le code budgétaire.

Les fonctions opérations complémentaires comme Recherche(), Masque(), etc sont appelées par les trois fonctions avec des arguments différents, elles sont conçues d'une manière ergonomique, modulaire et claire, trois qualités importantes pour la maintenance aisée d'un logiciel [10].

Lors de l'application d'une opération modification, si elle concerne la valeur de la clé, il faut la traiter comme une suppression suivie d'une création. La visualisation peut se faire de deux manières : avec la fonction BROWSE() qui présente tous les enregistrements de la table sous forme de lignes ou avec une fonction utilisateur ayant un masque comme celui de la modification qui présente un seul enregistrement à la fois contrairement à BROWSE().

6. CONCLUSION

La bibliothèque fournie est la première pierre pour la mise en oeuvre des logiciels faciles à maintenir avec un bon découpage de l'application par la modularité de fonctions et la clarté de structures. L'approche de la programmation orientée objet permet l'emploi aisé de cette bibliothèque de mise à jour pour les divers logiciels de gestion. Mais beaucoup de travail reste à faire. Les efforts doivent aller dans le sens de la perfection en gardant en esprit la construction des fonctions pour réaliser les différentes règles de gestion liées à chaque application ou fonction.

REFERENCES

[1] Rick Spence, Clipper programming guide, Microtrend Books, 1992.

[2] J. Yahouedeou, Techniques de programmation en C, Sybex, Paris, 1989.
[3] Galacsi, Sciences et Pratiques de l'informatique, Bordas, Paris, 1986.
[4] C. Delobel, Bases de données et systèmes relationnels, Bordas, Paris, 1982.
[5] R. Reix, Informatique appliquée à la comptabilité et à la gestion, Foucher, Paris, 1980.
[6] R. Heude, Comment mettre en place une gestion informatique des stocks, L'Usine, Paris, 1982.
[7] Computer Associates, CA-Clipper programming for Dos, Computer Associates, 1992.
[8] J. Courtin, Initiation à l'algorithmique et aux structures de données, Bordas, Paris, 1989.
[9] Knuth, Fundamental algorithms, Addison Wesley, 1973.
[10] I. Sommerville, Le génie logiciel et ses applications, Interditions, 1988.