Utilisation avancée de MapInfo sans MapBasic

Partie II

Gestion des colonnes d'une table MapInfo

Le quatuor "Add Column – Alter Table – Select – Update"

Jacques Paris
professeur honoraire, principal Paris PC Consult Enr.
jacques@paris-pc-gis.com

version initiale 11/11/2002 – Jacques Paris

document en format US Letter

Table des matières

Gestion des colonnes d'une table MapInfo :

Le quatuor "Add Column – Alter Table – Select– Update"

La gestion des colonnes de tables MapInfo peut être faite à partir du menu ou grâce à certains énoncés MapBasic en nombre restreint. Comme l'utilisation de la fenêtre MapBasic offre souvent des possibilités plus opérationnelles que les menus, les énoncés MapBasic seront seuls utilisés dans ce document, les utilisateurs voulant employer les menus pouvant faire une simple adaptation des présentations

Par gestion, on comprend les opérations sur une table concernant le nombre de colonnes (ajout, retrait), leurs caractéristiques (type, nom, ordre) et leur contenu (valeurs des cellules) Nous allons d'abord examiner les commandes principales et quelques connexes utiles dans la réalisation de certains scénarios, voir les différents types d'opérations et comment choisir quand certaines alternatives sont possibles, et finalement présenter divers exemples pratiques.

Conventions d'écriture

SELECT	(majuscules) mots clefs requis
<nom_table>	à remplacer (y compris les <>) par un « mot »
"nouvelle_spéc"	à remplacer mais les " " doivent être conservés
{ aaa | bbb }	un des termes est requis
[aaa]	terme optionnel (les crochets sont omis)
[aaa | bbb | ccc]	un choix à faire entre plusieurs termes optionnels exclusifs

1 – Les commandes principales

1 – 1 Add Column

Objectif : Ajouter une colonne temporaire à une table ouverte, ou mettre à jour une colonne existante avec des données contenues dans une autre table.

Syntaxe1 : Entrée directe des valeurs des enregistrements dans une colonne temporaire

```
ADD COLUMN  <table_destination> (<nom_colonne> [<type_colonne>] )
        [ VALUES <const> [,<const> ... ]  ]
```

Syntaxe2 : Valeurs pour une colonne temporaire ou déjà existante calculées à partir d'une autre table

```
ADD COLUMN  < table_destination > (<nom_colonne> [<type_colonne>] )
        FROM <table_origine> SET TO <expression>
        [ WHERE { <col_destination>=<col_origine> |
                WITHIN | CONTAINS | INTERSECTS } ]
        [ DYNAMIC ]
```

< table_destination >	nom d'une table ouverte à modifier
<nom_colonne>	nom de la colonne à ajouter (jusqu'à 31 caractères; lettres, chiffres et _; ne peut pas commencer par un chiffre) ou à mettre à jour (déjà existante)
<type_colonne>	type de données de la colonne à ajouter Char(largeur) Float Integer SmallInt Decimal (largeur, chiffres après virgule) Date Logical Si pas spécifié, est pris comme Float. Doit correspondre au type de la colonne origine ou à la nature des résultats de l'<expression> Omis si la colonne existe déjà.
<table_origine>	nom d'une table ouverte qui fournira les données
<expression>	expression utilisée pour calculer les valeurs à mettre dans la nouvelle colonne. En général, fait appel à la <table_origine>. Peut comprendre des fonctions d'agrégation.
<col_destination>	colonne dans < table_ destination > (pas de mention de la table)
<col_origine>	colonne dans <table_origine> (id.)

Détails et commentaires :

1 - Permet de créer une seule colonne à la fois, mais plusieurs colonnes temporaires peuvent être ajoutées à la même table.

2 - Si le nombre de valeurs spécifiées est inférieur au nombre d'enregistrements, la valeur des autres cas reste celle d'initialisation de la colonne (voir Alter Table (Add ...)).

3 – La jointure entre les deux tables peut être faite de différentes façons

- en omettant « where ». MapInfo choisit la méthode à employer.

- en spécifiant la colonne dans chaque table contenant des valeurs à faire correspondre « where colX.destination=colY.origine »
  Les valeurs dans la colonne destination doivent être uniques, mais ce n'est pas le cas dans la colonne source. En effet, la multiplicité d'une valeur donnée dans la colonne origine est une des bases des opérations d'agrégation.

- en utilisant des opérateurs géographiques spécifiques si les deux tables contiennent des objets: « where contains » si l'objet de la table origine contient un objet de la table destination, « where within » si l'objet de la table origine est à l'intérieur d'un objet de la table destination, « where intersects » si l'objet de la table origine intercepte un objet de la table destination. Noter le sens de la condition, toujours l'objet de la table source en premier, celui de la table destination en deuxième.

4 - Le mot clé DYNAMIC ne peut être employé que si les données sont tirées d'une autre source et n'est reconnu par MI que si la colonne en question est une nouvelle temporaire; il ne s'applique donc pas lors d'une mise à jour d'une colonne existante.

DYNAMIC permet de maintenir un lien entre les deux tables : les valeurs de la colonne temporaire seront mises à jour automatiquement si celles de la table origine sont modifiées. Ce lien est brisé dans certaines circonstances : si la table origine est fermée, ou dans le cas de jointure géographique, si l'une des cartes est fermée. Dans ces cas, les valeurs de la colonne temporaire sont figées.

Cette commande est la seule moyen pour avoir une mise à jour automatique d'une table par une autre avec MapInfo. Les colonnes permanentes ne le permettent pas. Pour pouvoir conserver ces liens qui disparaissent à la fermeture de la table destination, il faut créer un document WOR qui à sa ré-ouverture régénérera la colonne temporaire et ses liens de mise à jour automatique.

Si <expression> utilise des colonnes des deux tables, les résultats sont mis à jour dès qu'il y a un changement dans l'une ou l'autre table.

5 – Les règles de construction des expressions sont détaillées dans le document SQL déjà mentionné. Dans ce cas, elles peuvent faire appel à des colonnes ou à des fonctions impliquant des colonnes appartenant aux deux tables. Il suffit d'employer le format <nom_de_table>.<nom_de_colonne> pour les distinguer.

6 - Si une jointure existe entre les deux tables, plusieurs fonctions agrégatives peuvent être utilisées dans <expression>. Dans la table suivante, par « enregistrements de la table origine », il faut comprendre les cas de la table origine qui correspondent à un enregistrement de la table destination.

AVG (*col*)	Moyenne des valeurs des enregistrements de la table origine
COUNT (*)	Nombre d'enregistrements de la table origine
MAX (*col*)	Plus grande valeur dans la colonne *col* des enregistrements de la table origine
MIN (*col*)	Plus petite valeur dans la colonne *col* des enregistrements de la table origine
SUM (*col*)	Somme des valeurs dans la colonne *col* des enregistrements de la table origine
WTAVG (*col, wt_col*)	Moyenne pondérée des valeurs dans la colonne *col* des enregistrements de la table origine, par celles dans la colonne *wt_col*
PROPORTION AVG (*col*)	Moyenne calculée sur la base de la part d'un objet dans un autre
PROPORTION SUM	Somme calculée sur la base de la part d'un objet dans un autre

(*col*)

PROPORTION WTAVG (*col, wt_col*)	Moyenne pondérée calculée sur la base de la part d'un objet dans un autre

Les trois dernières fonctions font appel à une opération géographique consistant à trouver le pourcentage d'un objet de la table origine se trouvant à l'intérieur d'un objet de la table destination. Cela implique des restrictions sur le type des objets dans les tables; ceux de la table destination doivent être des régions et les calculs de proportionnalité ne sont vrais que si la table origine ne contient que des régions, car MapInfo traite les objets d'autres types comme complètement à l'intérieur ou à l'extérieur suivant que leur centroïde est ou non à l'intérieur de l'objet de la table destination.

7 – Si la table à laquelle une colonne est ajoutée est affichée dans une fenêtre-tableau, les résultats de l'ajout n'y sont pas visibles immédiatement. Une nouvelle fenêtre-tableau doit être ouverte.

1 – 2 Alter Table

N.B. ne peut pas être utilisé sur des tables liées.

Objectif : Modifier la structure d'une table ouverte.

Syntaxe :

```
ALTER TABLE <nom_de table> (
        [ADD <nom_colonne> <type_colonne> [,...] ]
        [MODIFY <nom_colonne> <type_colonne> [,...] ]
        [DROP <nom_colonne> [,...] ]
        [RENAME <nom_col_ancien> <nom_col_nouveau> [,...] ]
        [ORDER <nom_colonne>, <nom_colonne> [,...] ]
        )
        [INTERACTIVE]
```

<nom_de_table>	nom d'une table ouverte
<nom_colonne>	nom d'une colonne (jusqu'à 31 caractères; lettres, chiffres et _; ne peut pas commencer par un chiffre)
<type_colonne>	type de données de cette colonne Char(largeur) Float Integer SmallInt Decimal(largeur, chiffres après virgule) Date Logical
<nom_col_ancien>	nom de la colonne à renommer
<nom_col_nouveau>	nouveau nom pour la colonne

Détails et commentaires :

1 - MODIFY, DROP, RENAME et ORDER ne s'appliquent pas à des colonnes temporaires.

2 – On ne peut pas supprimer toutes les colonnes d'une table existante; il y aurait alors refus de la fenêtre MapBasic avec une « erreur d'origine inconnue ». Le message correspondant avec la commande du menu est plus explicite. Une table MI n'existe en effet que si au moins une colonne est définie, même si elle est vide (c'est une condition essentielle pour la création d'une nouvelle table)

S'il faut supprimer toutes les colonnes existantes, il faut d'abord en créer au moins une nouvelle qui restera dans la table. Création et suppression peuvent se faire dans la même commande si leur ordre est bien respecté.

1 – 3 Select

Objectif : Sélectionner des colonnes et des lignes d'une ou plusieurs tables dans une table temporaire. Permettre aussi des fonctions de tri et d'agrégation..

Syntaxe :
```
SELECT <liste_expressions> FROM <nom_table> [, ...]
    [ WHERE <groupe_conditions> ]
    [ INTO <table_résultats> [ NOSELECT ]  ]
    [ GROUP BY <liste_colonnes> ]
    [ ORDER BY <liste_colonnes> ]
```

Cette commande est présentée en détail dans un document qui lui est consacré (Utilisation avancée de MapInfo sans MapBasic - Partie I –SQL dans sa fenêtre ou via la fenêtre Mapbasic)

Détails et commentaires (dans la perspective de ce document) :

+1 - Les résultats d'une sélection ne sont visibles que dans une nouvelle fenêtre-carte à ouvrir avec le nom « selection » ou avec <table-résultats> si un nom est spécifié dans l'option INTO. Si les enregistrements sélectionnés sont identifiés visuellement dans une carte ou dans un tableau, le contenu tabulaire de la sélection n'est visible qu'en ouvrant une nouvelle fenêtre-données.

+2 - Les résultats d'une sélection sont temporaires; la table « selection » est fermée si une des tables dont elle est tirée est fermée. Une sélection peut toujours être enregistrée sous un nom quelconque, du moment qu'il est différent de ceux des tables ouvertes.

+3 - L'énoncé d'une sélection peut être enregistré en sauvegardant un document WOR. Il faut pour cela que dans la commande Menu Options | Préférences | Startup l'option « enregistrer requêtes dans WOR » soit choisie.

1 – 4 Update

Objectif : Modifier un ou plusieurs enregistrements d'une table ouverte

Syntaxe 1 : *Table entière*

```
UPDATE <nom_de_table> SET <nom_colonne> = <expression> [, ... ]
```

Syntaxe 2 : *Enregistrement spécifique*

```
UPDATE <nom_de_table> SET <nom_colonne> = <expression> [, ... ]
    WHERE ROWID = <#enregistrement>
```

<nom_de_table>	nom d'une table ouverte
<nom_colonne>	nom d'une colonne
<expression>	une constante ou le résultat de l'expression à assigner à une colonne
<#enregistrement>	le numéro d'un enregistrement spécifique

Détails et commentaires :

1 – UPDATE est le seul énoncé qui permette de travailler sur le contenu de la colonne OBJ mais cela n'est possible que par le biais de la fenêtre MapBasic, la commande *Table* | *Mettre à jour d'une colonne* n'offrant pas *obj* parmi les colonnes disponibles pour cette opération.

2 – La mise à jour est faite sur la table en entier ou sur le seul enregistrement spécifié dans <#enregistrement>. Il est alors indispensable de connaître le ROWID de l'enregistrement voulu. *Rowid* est une des deux colonnes « cachées » d'une table; elle est inaltérable de l'extérieur (lecture seulement) alors que l'autre, *obj,* peut être modifiée.

Une mise à jour partielle est possible si les enregistrements désirés peuvent être « sélectionnés »; l'opération est alors faite dans un deuxième temps en utilisant comme nom de table la « sélection ».

3 – La mise à jour peut se faire sur plusieurs colonnes à la fois. Rien n'est précisé des conséquences possibles de l'utilisation d'une colonne mise à jour dans la mise à jour d'une autre colonne Mais il semble que la première mise à jour ne soit pas « disponible » pour la seconde. Ainsi

update tablex set col2=col1/3, col3=col2+5

met à jour col2 et pas col3. Il faut alors utiliser des commandes « progressives ».

> À noter que ce même genre de limitation s'applique aussi à SELECT ... GROUP BY...; ainsi il est impossible de calculer un taux tout en faisant l'agrégation comme dans sum(var1/sum(var2))

4 – Si la table est affichée dans une fenêtre-tableau, les résultats d'un UPDATE sont visibles immédiatement.

2 – Les commandes annexes

2 – 1 Browse

Objectif : Ouvrir une nouvelle fenêtre- données

Syntaxe :
```
BROWSE { * | <liste_d'expressions> }  FROM <nom_de_table>
        [ POSITION ( <x>, <y> ) [ UNITS "unité" ] ]
        [ WIDTH  <largeur> [ UNITS "unité" ] ]
        [ HEIGHT <hauteur> [ UNITS "unité" ] ]
        [ ROW <n> ]  [ COLUMN <m.> ]
        [ MIN | MAX ]
```

<liste_d'expressions>	expressions-colonne séparées par des virgules. Ces expressions peuvent être des noms de colonne ou des façons de calculer à partir d'une ou plusieurs colonnes avec possibilité d'alias.
<x>, <y>	coordonnées du coin supérieur gauche de la fenêtre
<largeur>	largeur de la fenêtre
<hauteur>	hauteur de la fenêtre
"unité"	x, y, largeur et hauteur sont exprimées dans l'une des unités-papier standard : cm, in mm, pt, pica (par défaut 'pouce')
<n>	indice de la ligne à afficher dans la première rangée du tableau
<m.>	indice de la colonne à afficher sut la gauche du tableau
option MIN | MAX	la fenêtre sera affichée soit comme icône soit en plein écran. Si elle est aussi définie en position et taille, ces valeurs seront utilisées quand la fenêtre sera « restaurée ».

Détails et commentaires :

1 – La commande du menu Window | Données est l'équivalent de « Browse * from <table> ». Pour changer l'ordre des colonnes, les choisir, redéfinir des expressions ou donner des alias, il faut utiliser le menu « Browse | Choisir champs... » (et dans ce cas aucune « contrepartie » n'est affichée dans la fenêtre MapBasic) ou écrire la commande complète dans la fenêtre MB.

2 – Exemple de liste d'expressions comprenant des alias. La table « world » est ouverte:

```
Browse country "Pays", population, population / area(obj, "sq km") "Densité"
```

3 – La "mise-en-forme" d'une fenêtre- données est perdue dès qu'elle est fermée, si elle n'est pas inscrite dans la fenêtre MapBasic, mais.il est possible de sauvegarder l'information nécessaire pour l'ouvrir à nouveau dans les mêmes conditions en l'extrayant d'un document WOR réalisé après son ouverture.

L'extrait suivant correspond à la commande :

```
Browse  afield,bid+z "somme" from polyclock row 3 width 2
```

```
Browse afield, BID+z "somme"  From polyclock
  Position (0.0520833,0.0520833) Units "in"
  Width 2 Units "in" Height 2.95833 Units "in"
  Row 3 Column 0
```

4 – L'option de présentation du menu « Browse | Options | Show grid lines » est accessible de la fenêtre MapBasic avec aussi le positionnement du tableau dans la fenêtre grâce à :

```
SET BROWSE [ WINDOW  <#fenêtre> ]
```

[GRID { ON | OFF] [ROW <n> } [COLUMN <m>]

Si l'option WINDOW est absente, l'opération s'appliquera à la fenêtre-données la plus récente.

2 – 2 Commit Table

Objectif : Sauvegarder les changements faits à une table, ou faire une copie d'une table

Syntaxe 1 : *Sauvegarde des changements (= Fichier | Enregistrer table)*

```
COMMIT TABLE <nom_de_table>
```

Syntaxe 2 : *Copie d'une table ouverte (= Fichier | Enregistrer table **sous**)*

```
COMMIT TABLE <nom_de_table>
     [ AS "nouvelle_spéc"
          [ TYPE { NATIVE |
                    DBF [CHARSET "page_caractères" ]  |
                    ACCESS DATABASE "db_spéc"
                              [ VERSION <db version> ]
                              TABLE "nom_nouvelle_table"
                              [ PASSWORD <clé> ] [CHARSET "page_caractères" ]  |
                    QUERY
                   } ]
          [ COORDSYS <coordsys spécification> ]
          [ VERSION <version id> ] ]
       [ { INTERACTIVE | AUTOMATIC
          { NOCOLLISION | APPLYUPDATES | DISCARD UPDATES } } ]
```

<nom_de_table>	nom d'une table ouverte
"nouvelle_spéc"	nom de la table sauvegardée. Peut contenir la spécification complète du répertoire.
"page_caractères"	une des polices de caractères de Windows à utiliser dans l'interprétation des codes de caractères. Souvent "WindowsLatin1"
"db_spéc"	nom d'une banque Access. Si elle n'existe pas, Mi créera une nouvelle MBD
<db version>	3.0 (Access 95/97) ou 4.0 (Access 2000). Par défaut, 4.0
"nom_nouvelle_table"	nom de la nouvelle table comme elle apparaîtra dans Access.
<clé>	mot de passe pour la base de données. Doit être spécifié si la sécurité de la base est activée.
<coordsys spécification>	spécification du système de projection pour la carte.
<version id>	100, 300 à compter de la version MI3.0, 410 si table Access.

Détails et commentaires :

1 – La copie sauvegardée n'est pas une copie conforme dans ce sens que s'il y avait des enregistrements annulés, la table originale n'ayant pas été compactée, ils ne se retrouvent pas dans la copie.

2 – L'option [INTERACTIVE...] n'est utile que dans la situation où plusieurs personnes éditent la même table. Les mots clés précisant l'alternative AUTOMATIC ... permettent de spécifier comment MI devrait résoudre de conflits éventuels. Cette option n'a donc aucune importance avec les tables conventionnelles.

3 – Un COMMIT ... AS est un excellent moyen de changer définitivement la projection d'une table en spécifiant une différente CoordSys.

4 – MI refusera d'enregistrer une table avec le même nom que celui de la table ouverte servant de base.

2 – 3 Objects Split

Objectif : Découper les objets de la cible avec ceux qui sont sélectionnés

Syntaxe :
OBJECTS SPLIT INTO TARGET
[DATA <colonne> = <expression> [, <colonne> = <expression>]]

<colonne> nom d'une colonne

<expression> façon d'attribuer une valeur pour les nouveaux objets

colonneA=colonneA
les nouveaux objets conservent la valeur de cette colonne.

colonneB="split1"
les nouveaux objets reçoivent tous la valeur "split1". Le type de donnée doit être compatible avec celui de la colonne.

colonneC=PROPORTION(colonneC)
(colonnes numériques seulement) les nouveaux objets reçoivent une part de la valeur de l'original proportionnelle à leur taille comparée à celle de l'objet original.

Détails et commentaires :

1 – Les objets à découper peuvent être des objets fermés (régions, rectangles,...) ou des objets linéaires (lignes, polylignes, arcs). Les objets « coupeurs » doivent être des objets fermés.

2 – Les objets à découper sont d'abord transformés en cible (SET TARGET ON) puis les « coupeurs » sont sélectionnés. La commande OBJECTS SPLIT peut alors être exécutée.

3 – Un objet à découper est transformé en autant d'objets qu'il y a d'objets coupeurs qui l'intersectent, plus éventuellement un objet supplémentaire formé des parties non découpées de l'objet original. Les nouveaux objets sont placés à la fin de la table et les objets de la cible sont éliminés; si l'un n'est pas découpé, il est aussi éliminé et remplacé par l'équivalent avec les autres nouveaux.

3 – L'option DATA permet d'assigner des valeurs aux nouveaux objets. Seules les colonnes spécifiées recevront des valeurs, les autres auront un blanc (chaînes) ou un zéro (numériques et assimilées). Une façon rapide de construire cette liste est de commencer avec

COL1=COL1, COL2=COL2, ... , COLn=COLn

n étant le nombre de colonnes dans la table, puis de l'ajuster en fonction des besoins.

2 – 4 Open Table

Objectif : Ouvrir une table MI pour lecture et écriture

Syntaxe : OPEN TABLE “table_réf” [AS <nom_table>] [HIDE] [READONLY]
[INTERACTIVE] [PASSWORD <pwd>] [NOINDEX]
[VIEW AUTOMATIC] [DENYWRITE]

“table_réf”	référence complète de la table à ouvrir. Ex. : ”C:/data/US/states” L’extension ”.TAB” n’est pas requise.
<nom_table>	le nom sous lequel MI reconnaîtra cette table. Par défaut, MI donne le nom du fichier. Dans l’exemple précédent, ce serait « states »
<pwd>	mot de passe au niveau base de données si la sécurité est active. Seulement pour des tables Access

Options :

HIDE	ne paraît pas dans les listes de tables ouvertes, mais reste accessible via MapBasic (ex. : MAP FROM...)
READONLY	ne pourra pas être éditée par l’utilisateur. N’est pas changeable par MapBasic
INTERACTIVE	si la table ne peut pas être trouvée à l’endroit indiqué, MI ouvre un dialogue pour la localiser. Si cette option est absente et que la table n’est pas trouvée, il y aura une erreur.
PASSWORD	Seulement pour une table Access (voir <pwd> plus haut)
NOINDEX	Seulement pour une table Access : l’index MI ne sera pas reconstruit à l’ouverture de la table
VIEW AUTOMATIC	Quand des objets “hotlink” existent, permet le lancement de ce qui est associé à ces objets (table, wor, application) dans le contexte propre s’il existe (ex. ajout à une fenêtre carte) ou d’un nouveau contexte (ex. nouvelle fenêtre carte)
DENYWRITE	Seulement pour une table Access : l’accès en écriture de cette table est interdit.

Détails et commentaires :

1 – La table à ouvrir doit être une table MapInfo, c’est-à-dire une table définie par un fichier .TAB associé à différents fichiers nécessaires pour définir : une table MI originale (avec les fichiers .MAP, .DAT, .IN, possiblement .IND,..), une table sans données graphiques (avec un fichier .TXT, .XLS, .DBF...) ou une table image (avec un fichier « raster » de format acceptable). La création d’une table se fait par CREATE TABLE pour une table MI originale, avec REGISTER TABLE pour les autres.

2 – L'erreur qu'entraîne l'absence de la table à l'endroit désigné lorsque INTERACTIVE est omis est très fréquente avec les documents WOR. Cela se produit surtout quand les fichiers ont été déplacés depuis la création du document. Avant la version 6.5 MI n'ajoutait pas cette option dans un énoncé OPEN TABLE si la table se trouvait dans le même répertoire que celui où allait être sauvegardé le document; ceci multipliait les chances d'avoir des WOR inutilisables tant que cette option n'y était pas rajoutée.

2 – 5 Set Target

Objectif : Définir ou supprimer une cible faites d'objets

Syntaxe : SET TARGET { ON | OFF }

Détails et commentaires :

1 – L'existence d'une cible est nécessaire pour certaines opérations de modification d'objets. Les objets qui constituent une cible sont ceux sur lesquels l'opération va porter. Il faut les sélectionner et établir la cible (SET TARGET ON) ensuite au besoin sélectionner d'autres objets qui « agiront » sur ceux de la cible, et lancer l'opération.

2 – La cible peut être « vidée » par SET TARGET OFF pour pouvoir la redéfinir; certaines opérations le font automatiquement (OBJECTS SPLIT INO TARGET par exemple)

3 – Les commandes requérant une cible sont OBJECTS COMBINE [INTO TARGET] , OBJECTS ERASE INTO TARGET, OBJECTS INTERSECT INTO TARGET, OBJECTSOVERLAY INTO TARGET, et OBJECTS SPLIT INTO TARGET. Pour la première seulement la cible est une option.

3 – Les opérations principales

3 – 1 Création d'une colonne

3 – 1 – 2 Colonne permanente

La création d'une nouvelle colonne dans une table existante se fait par la commande **Alter Table**; la nouvelle colonne ainsi créée est ajoutée à la structure existante avec le type et le nom spécifiés; les valeurs de cette colonne sont initialisées selon le type (0 pour tous les types numériques avec les positions décimales pour Decimal, F (=Faux, =0) pour Logical, "" pour Caractères et Date).

Une création de colonne est instantanée mais entraîne la fermeture des représentations de la table : comme couche dans une fenêtre-carte, (si elle est la seule couche, la fenêtre est fermée), ou comme tableau (la fenêtre tableau est fermée), mais la table en elle-même n'est pas fermée et peut donc être replacée dans ses représentations sans avoir besoin de l'ouvrir. De plus, les changements sont définitifs, il n'y a pas besoin de sauvegarder la table.

Exemple :

Alter table lollipop (add Sucre Float, Calories SmallInt)

Rajoute à la table Lollipop les colonnes "Sucre" (type Float) et "Calories" (type Smallint) avec des valeurs 0 dans les deux.

3 – 1 – 2 Colonne temporaire

Une colonne temporaire peut être créée avec **Add Column.** On peut lui assigner des valeurs directement ou en établissant une jointure avec une autre table.

Exemple :

Add column lollipop (poids) values 23,21,13,24,1,2,45

Rajoute une colonne temporaire 'poids' (type Float par défaut) à la table Lollipop qui contiendra dans ses sept premières rangées les valeurs indiquées.

Exemple :

Add column lollipop (commandes) from orders set to sum(montant) where marque=type

Rajoute une colonne temporaire 'commandes' à la table Lollipop qui contiendra dans chaque rangée la somme des valeurs de la colonne 'montant' des lignes de la table 'orders' pour lesquels le contenu de la colonne 'type' est égal à celui de la colonne 'marque dans la table Lollipop.

3 – 1 – 3 Récupération d'une colonne temporaire

Quand une ou plusieurs colonnes temporaires ont été créées, on peut récupérer les conditions de leur création en sauvegardant un document WOR. Pour chaque colonne temporaire, on pourra y trouver la ligne clé reproduction fidèle de la commande ADD COLUMN. On peut alors relancer le WOR, ou copier/coller dans la fenêtre MapBasic la première ligne ou les deux, à condition que les 2 tables soient ouvertes au préalable..

```
Add Column "polyclock_clk" (new3) From polyclock Set To z+z Where COL1 = COL1  Dynamic
Browse * From polyclock_clk
```

Extrait d'un WOR qui pourrait être réutilisé directement dans la fenêtre MapBasic quand les 2 tables sont fermées. Noter que la jointure par colonne est faite par des COLn plutôt que par des noms spécifiques comme ils avaient été donnés dans la commande.

3 – 1 – 4 Transformation d'une colonne temporaire en permanente

Quand une colonne temporaire a été créée, elle n'existe que le temps que la table reste ouverte. Quand on ferme la table, la colonne disparaît sans laisser de trace (elle n'est pas considérée comme une modification à la table) et même une sauvegarde simple ne permet pas de la conserver. Pour pouvoir le faire, il faut utiliser un « Commit table ... as ... » , c'est à dire faire une copie sous un nom différent de la table d'origine mais incluant la nouvelle colonne.

3 – 2 Élimination d'une colonne

La suppression de colonnes dans une table existante se fait avec la commande ALTER TABLE (DROP...). Pour supprimer une colonne temporaire, il faut fermer la table et la réouvrir; on ne peut pas y faire de référence directe car elle n'est pas enregistrée dans la structure de la table..

3 – 3 Remplissage ou mise à jour d'une colonne permanente

Nous avons deux situations légèrement différentes. Dans la première, une colonne d'une table ouverte (destination) doit être mise à jour avec le contenu de la colonne d'une autre table (source). Dans l'autre, les valeurs de la colonne sont mises à jour en fonctions de règles internes seulement.

Dans le premier cas, on utilise la commande du menu « **Table | Mettre à jour colonne** », mais dès que l'on spécifie une table source, MI identifie la colonne à mettre à jour comme une nouvelle colonne temporaire. Il faut dans ce cas définir la table source et la façon de calculer les valeurs (simple colonne ou expression) et ensuite seulement choisir la colonne à mettre à jour. Cette commande se traduit en fait par un ADD COLUMN où la 'nouvelle' colonne est une déjà existante (le type de données est alors omis). Seulement les lignes correspondant aux conditions de jointure sont remplies

La **commande UPDATE** ne prend les données nécessaires que de façon interne (colonne dérivée par une expression sur colonnes existantes) ou de valeurs reçues directement. Toute modification aux objets même doit passer par un UPDATE, la colonne obj n'étant pas accessible par la commande du menu (ce qui n'est pas surprenant puisqu'elle se traduit par un ADD COLUMN).

Il ne faut pas oublier que **ADD COLUMN** permet de mettre à jour une colonne existante en faisant une jointure (where) sur la table même alors que UPDATE ne le permet pas, mais cette possibilité n'existe que via la fenêtre MB.

Il est possible de **fragmenter la mise à jour**, c'est-à-dire d'appliquer des « règles de remplissage » qui ne s'appliquent qu'à des parties distinctes de la table. Ceci implique que les enregistrements soumis à une règle sont d'abord sélectionnés et que la mise à jour est appliquée à la sélection et non à la table entière. Cette opération demandant du temps doit être répétée pour chacune des règles. C'est la procédure que l'on suit généralement pour la re-classification de données.

3 – 4 Fusion des colonnes de deux tables

Fusionner deux tables en une, c'est-à-dire « coller » leurs colonnes ensemble, s'effectue d'habitude avec un « Select ». Certaines conditions sont imposées sur cette opération dont il faut être conscient avant de définir ses paramètres. Tout d'abord, le résultat de cette commande est une nouvelle table temporaire qu'il faut enregistrer pour pouvoir la conserver et cela sous un nom différent de ceux des tables originales (comme elles sont ouvertes pour que la sélection se fasse, elles ne peuvent être modifiées; il faudrait les fermer ce qui éliminerait la table-résultat.)

Ensuite, la spécification des colonnes peut être contraignante. La façon la plus simple est naturellement d'utiliser * (toutes les colonnes des deux tables), mais dès que l'on désire une sélection de colonnes d'une des tables, il faut spécifier toutes les colonnes, même si on veut toutes celles d'une table. Il est souvent plus facile et moins risqué de fusionner toutes les colonnes et de supprimer par la suite celles qui ne sont pas nécessaires.

Qui dit fusion de colonnes, dit aussi jointure des cas. Quand plus d'une table est spécifiée dans un Select, la condition de jointure des cas (Where) est requise. Elle peut se faire par le contenu d'une colonne et le plus simple est quand les deux tables ont une colonne dont le contenu est un identificateur unique permettant une jointure un à un (tableA.col1=tableB.col3). Dans ce cas, la table-résultat va **contenir seulement les cas qui sont communs aux deux tables.**

La condition de jointure peut aussi être une de géographie (tableA.obj contains tableB.obj).

De plus dans les deux cas, il peut y avoir des relations « un à plusieurs » (plusieurs cas de B correspondent à un de A) et on peut utiliser des fonctions d'agglomération pour générer de nouvelles colonnes, mais ce sujet fait plutôt partie de la Partie I. Dans la situation symétrique de relations « plusieurs à un », tous les cas de tableA correspondant au cas dans tableB reçoivent les valeurs de ce cas dans tableB.

3 – 5 Changement de l'ordre des colonnes.

La syntaxe de « ALTER TABLE (ORDER <nom_col_x> <nom_col_w>, ...) » est à utiliser pour simplement changer l'ordre des colonnes spécifiées; les autres colonnes restent dans leur ordre original à la suite de celles dans la liste de cette commande.

3 – 6 Changement de nom de colonnes.

La syntaxe de « ALTER TABLE (RENAME <nom_col_ancien> <nom_col_nouveau>, ...) » est à utiliser pour simplement renommer une ou plusieurs colonnes.

3 – 7 Opérations multiples en une seule commande.

On peut utiliser Select pour combiner plusieurs opérations dans une seule commande. En effet, on peut spécifier dans la liste des colonnes celles que l'on désire avoir, leur ordre, leur nom (en les faisant suivre par un alias entre " ") et même générer de nouvelles colonnes dérivées. Bref, on peut ainsi créer une nouvelle table (partie tabulaire seulement) qui ne ressemblerait pas visuellement beaucoup à l'originale, excepté qu'elle contiendrait les mêmes données ou des données dérivées, mais pas à proprement parler de nouvelles données.

Rien n'empêche de réaliser ces opérations multiples à l'occasion d'une jointure avec une autre table. Mais plus on complexifie la commande, plus grands sont les risques d'erreurs d'écriture. Cependant, dans le cas d'opérations répétitives, la mise au point d'une telle commande peut valoir le coup si on en a prévu sa réutilisation de façon intelligente.

4 – Choix stratégiques

L'utilisateur devra faire certains choix avant de se lancer dans de telles opérations. Certains de ces choix sont liés aux commandes mêmes, d'autres concernent l'ensemble d'une procédure.

4 – 1 Permanence de l'effet de la commande

Un paramètre du choix est l'effet de la « permanence » de la commande c'est-à-dire ce qui est automatiquement sauvegardé, ce qui peut l'être directement et ce qui ne peut l'être qu'en faisant une copie de la table « modifiée ».

Tout changement à la structure même d'une table est fait de façon permanente. Ainsi « ALTER TABLE tableX (ADD colABC float) » crée une colonne de zéros et aucun enregistrement de la table n'est nécessaire par la suite pour conserver ce type de changement.

Si ce type d'opération est suivi d'une mise à jour de la colonne avec un « UPDATE... » ou d'un « ADD COLUMN... » sur une colonne existante, alors un enregistrement sera nécessaire pour sauvegarder les nouvelles valeurs dans la table.

Par contre, toute création temporaire devra être suivie d'un enregistrement du résultat avec un nom de table différent (colonne temporaire dans une table existante avec « ADD COLUMN ... » ou table temporaire avec « SELECT... »).

ATTENTION : le résultat d'une opération « temporaire » peut être détruit par une opération « permanente »; ainsi une colonne temporaire ajoutée par « ADD COLUMN...» disparaîtra si la structure de la table est modifiée avec un « ALTER TABLE...» et la table-résultat d'un « SELECT... » sera fermée dans les mêmes circonstances.

4 – 2 Effets de la commande sur les cas présents dans les résultats.

Une opération, outre ses effets sur la structure de la table originale, peut aussi avoir un impact sur la constitution de la liste des cas ou sur les cas pour lesquels les données ont été modifiées ou ajoutées.

« ALTER TABLE tableX (ADD colABC float) » initialise une nouvelle colonne, Tous les cas sont donc traités. Il en va de même pour toute modification de structure et pour toute mise à jour de colonne ne faisant pas appel à une seconde table.

C'est quand il y a jointure que les effets sont sensibles. S'il s'agit d'une mise à jour d'une colonne ou de l'ajout d'une colonne existante dans la table de base, de vraies données (différentes des valeurs par défaut assignées lors de la création d'une colonne) ne seront affectées qu'aux seuls cas présents dans la table de base et dans la table d'ajout. La liste des cas restant dans la table de base n'est pas altérée; aucun cas n'est supprimé ni ajouté. Il est alors difficile de faire la

différence entre un « vrai » zéro provenant de la table d'ajout et un zéro par défaut parce qu'il n'y pas de correspondance pour ce cas entre les deux tables.

Par contre, la table-résultat d'un « Select... » ne comprend que les cas que l'on retrouve dans les deux tables de bas et d'ajout.

4 – 3 Conservation de résultats intermédiaires.

Certaines procédures peuvent parfois être réalisées par une ou quelques commandes complexes. Il est important de savoir si certains résultats intermédiaires ne devraient pas être alors enregistrés, ce qui pourrait privilégier l'usage de commandes plus simples mais fournissant des résultats intéressants pour plusieurs raisons :

- ils pourraient être réutilisables dans d'autres procédures
- ils permettraient de valider la procédure même au fur et à mesure

De plus, la formulation et l'écriture de commandes complexes est souvent la source d'erreurs de syntaxe (mauvaise conception) ou d'orthographe (mauvaise traduction).

5 – Exemples

L'objectif de cette section n'est pas de donner des exemples de mise en œuvre des différentes commandes prises individuellement mais plutôt de déboucher sur des procédures mettant en jeu plusieurs commandes pour arriver aux résultats recherchés.

Certains de ces exemples débouchent sur une double solution pour montrer qu'il y a souvent le choix entre deux procédures de nature assez différente pour arriver au même résultat, et comment on peut élaborer une solution plus « sophistiquée » avec l'expérience de plus simples.

Plusieurs contributeurs s'y reconnaîtront au passage et je les remercie de leurs apports.

5 – 1 Exemple Un

Problème: Pour chaque bloc d'une table, calculer la valeur moyenne d'un champ pour les blocs qui lui sont adjacents (y compris le bloc lui-même) et enregistrer cette valeur.

La table originale contient une colonne avec identificateurs uniques des blocs, Table_ori.ID.

Procédure :

1 – Faire une copie de Table_ori qui doit être déjà ouverte comme Table_cop

```
commit table <Table_ori> as "Table_Cop"
open table "Table_Cop" as <Table_Cop >
```

2 – Exécuter la requête suivante

```
select table_ori.id, table_ori.<col_num> from table_ori,table_cop
      where table_ori.obj intersects table_cop.obj into sele
```

La table-résultat contient des enregistrements multiples pour chaque bloc : tous ceux qui sont adjacents au bloc et le bloc lui-même, et deux seules colonnes, l'identificateur du bloc et les valeurs des blocs adjacents,

3 – Faire une requête sur la table-résultat « Sele ».

```
select id, avg(col_num) from sele group by id into moyenne
```

4 – Si la colonne « moyenne » doit être placée dans Table_Ori, il faut alors procéder à la fusion de cette colonne dans la table.

Variante 1 : id mais la moyenne est calculée sur les adjacents excluant le bloc lui-même

Procédure 1 :

Il suffit de modifier la commande de l'étape 2 pour rajouter une condition d'exclusion quand les deux ID sont identiques.

```
select table_ori.id, table_ori.<col_num> from table_ori,table_cop
      where table_ori.obj intersects table_cop.obj and
```

```
table_ori.id <> table_cop.id into sele
```

Variante 2: Pour chaque bloc d'une table, calculer un taux moyen sur les blocs qui lui sont adjacents (y compris le bloc lui-même) et enregistrer cette valeur.

Procédure 2: (Éric Mauvière)

<var1>	variable « numérateur » du taux (ex. les plus de 64 ans)
<var2>	variable « dénominateur » du taux (ex. la population totale)
données	le nom de la table

1 – On va rajouter une colonnes qui contiendra les valeurs combinées du numérateur et du dénominateur (numérateur * 10^9 + dénominateur) ce qui permet d'en faire les sommes partielles dans une seule opération. Si une somme partielle du dénominateur excédait 10^9, il faudrait ajuster ce paramètre à la hausse.

```
add column données (varc) from données set to
      sum((10^9)*var1+var2) where intersects
```

2 – Puis une deuxième colonne pour le taux lissé sur les blocs adjacents calculé en décomposant la variable temporaire dans ses parties constituantes.

```
add column données (varliss) from données set to
      100*round(varc/10^9,1)/(varc -10^9*round(varc/10^9,1))
```

On peut remarquer que "ADD COLUMN... » permet de faire la jointure d'une table sur elle-même ce qui évite dans ce cas d'avoir à en faire une copie.

5 – 2 Exemple Deux

Problème: Il s'agit de mettre à jour une colonne existante d'une table avec des valeurs « traduites » de celles d'une autre colonne.

C'est le problème général d'avoir à compléter une table en ajoutant une expression littérale correspondant à des codes numériques.

Procédure 1

Cette une approche itérative doit être répétée pour chaque valeur du code. On choisit les cas pour une valeur donnée du code et on fait une mise à jour de la colonne alpha pour la sélection obtenue

```
Select * from TableX where col_code=<valeur_i> into sele
Update sele set col_alpha=<chaîne_i>
```

Façon très fondamentale de résoudre un tel problème, elle peut devenir fastidieuse et risquée à cause du nombre de corrections à faire puisque les valeurs du code et du texte correspondant doivent être rentrées par chaque valeur du code.

Procédure 2

Cette procédure est basée sur le principe de « tables de référence » (lookup tables). Nous allons mettre dans une table sans objets graphiques et ayant 2 colonnes seulement les valeurs qu'il nous faut (<col_code> et <code_alpha>). Cela peut se faire tout simplement avec Excel puis en ouvrant le fichier xls comme une table MI (appelons cette table Tab_Ref). La table de base contient comme il se faut la colonne de code <code>

```
Add column TableX (alphatexte) from Tab_Ref
      set to col_alpha where code=col_code
```

Une autre façon de créer Tab_Ref est par un « Select » sur la table de base

```
Select code, alphatexte from TableX group by code
      Order by code into sele
```

La colonne alphatexte est comprise dans la liste même si elle est vide pour ne pas avoir à l'ajouter ensuite. Il faut alors sauvegarder « selec » sous un nom choisi, la réouvrir, remplir la colonne col_alpha, enregistrer la table pour l'utiliser dans le « Add Column... »

```
Commit table sele as "Tab_Ref"
Open table "Tab_Ref" as Tab_Ref
Browse * from Tab_Ref
      Entrée des données ....
Commit table Tab_Ref
```

La façon de générer la table avec un « Select... » peut paraître plus rapide mais il faut être bien sûr que toutes les valeurs du code numérique soient bien présentes dans TableX. Si on recherche une façon de reproduire le traitement d'autres tables semblables, c'est-à-dire de réutiliser Tab_Ref, la façon Excel permet de faire des listes complètes et dans le bon ordre du premier coup alors que le rajout ou la suppression de cas (=codes) dans la table MI sont plus complexes.

5 – 3 Exemple Trois

Problème : Disposant d'une table de régions (mailles) et d'une de routes (routes) avec la quantité de pollution émise par chaque tronçon, il faut calculer pour chaque région la quantité de polluant qui s'y trouve émise. On fait l'hypothèse que l'émission de polluant est uniforme le long d'un tronçon.

Procédure 1 :

Nous allons produire des résultats intermédiaires avant d'en faire la synthèse.

1 – Il s'agit d'abord de découper les tronçons de routes pour les limiter aux régions et réajuster les valeurs numériques en proportion avec les nouvelles longueurs des tronçons.

```
select * from ROUTES
set target on
select * from MAILLES
```

On peut se servir de la commande du menu « Objects | Split » ou dans la fenêtre MapBasic

```
Objects Split Into Target Data <col_num> = proportion (<col_num>)
```

<col_num> est la colonne de ROUTES contenant les quantités à synthétiser.

Si l'on désire conserver toutes les autres variables qui pourraient être dans l'original de ROUTES, il faudrait en spécifier la liste dans cette commande par des colX=colX pour celles passées directement.

Si l'on désire conserver le résultat de ce découpage, il suffit d'enregistrer la table ROUTES préférablement sous un autre nom si une copie de l'original n'existe pas encore.

2 – Il faut ensuite calculer les totaux. Si une colonne numérique (<Total>) n'existe pas dans MAILLES pour les recevoir, on peut la créer avec « ALTER TABLE ... (ADD...) » et la mettre à jour avec un « ADD COLUMN... », ou créer une colonne temporaire avec « ADD COLUMN... » et sauvegarder les résultats sous un autre nom de table. Je préfère la première façon qui ne multiple les tables avec des noms différents mais pratiquement le même contenu.

```
Add column MAILLES from ROUTES set to Total=sum(<col_num>)
      where within
```

Cette méthode produit comme résultats intermédiaires, la carte des ROUTES re-tronçonnées aux limites des régions avec les valeurs réajustées; les totaux sont dans la table MAILLE.

Procédure 2 : (Denis Jouin)

Cette procédure produit avec une seule commande les totaux demandés. Elle exige qu'il y ait dans MAILLES une colonne d'identificateur unique (Mailles.ID)

```
Select MAILLES.ID, sum(<col_num> * objectlen(overlap(ROUTES.obj,
      MAILLES.obj, "km") / objectlen(ROUTES.obj, "km"))
      from MAILLES, ROUTES where MAILLES.obj intersects
      ROUTES.obj group by MAILLES.ID into SELE
```

SELE contient dans ses 2 colonnes l'identificateur de MAILLES et le total correspondant. Si on veut ces totaux dans la table MAILLES, il faudra transférer ces valeurs dans une nouvelle colonne de MAILLES.

Une seule opération est donc nécessaire et il n'y a que la table-résultat qui est produite. Suivant les circonstances cette procédure peut être la plus rapide.

5 – 4 Exemple Quatre (inspiré par Éric Mauvière)

Problème : Déterminer la commune la plus peuplée d'un canton

L'exemple commune/canton peut être étendu à tout couple de découpage régional « emboîté »

Procédure :

<canton> code numérique de canton,
<comm> code numérique de commune,
<popul> la population des communes décrites dans la carte « FOND »

1 – On va chercher le maximum sur une variable composite faite de la population et du code commune. Si celui-ci ne dépasse pas 10000 par exemple, cette variable sera « population * 10000 + code_commune ».

```
Select <canton>, max(<popul>*10000+<comm>) "Interm"
      from FOND group by <canton> into SELE
```

Le maximum entre 2 valeurs de <popul> ne sera pas affecté par la portion rajoutée du code de commune. Le seul impact possible serai s'il y avait égalité de population; la commune qui sera retenue serait alors celle avec le plus « gros » code.

2 – Il faut ensuite décomposer la colonne Interm pour retrouver le code de la commune et sa population

```
Select <canton>, interm-int(interm/10000)*10000 "Compole",
      int(interm/10000) "Population" from SELE into POLES
```

La table POLES (temporaire, à enregistrer si nécessaire) contient la réponse au problème : le code du <canton>, celui de la commune dans « Compole » et sa population correspondante.

5 – 5 Exemple Cinq (Éric Mauvière)

Problème : Ajouter une colonne contenant le total cumulé d'une autre colonne.

Procédure :

La table "Données" contient au moins une colonne numérique <col_num> sur laquelle le cumul se fera et a déjà été triée sur cette colonne.

1 - On numérote les lignes de la table source et on crée un champ test=1, puis on fait une copie de la table sous le nom « Copie ».

```
Alter Table données ( add Idrow Integer, _test smallint )
Update données Set Idrow = rowid, test=1
Commit Table données
Commit Table données As "c:\temp\copie.tab"
Open Table "c:\temp\copie.tab"
```

2 – Pour construire notre commande, il faut se rappeler que la jointure cartésienne de la table source et de sa copie est tolérée par Mapinfo dès lors qu'une condition de jointure géographique ou qu'une égalité entre deux champs lui est fournie (rôle de la variable "test"); la colonne cachée RowID ne peut pas être utilisée directement pour faire une jointure d'où la colonne Idrow.

```
Select données.idrow, données.<col_num>, sum(copie.<col_num>)
"Cumul" from données, copie where données.test= copie.test and
données.idrow>= copie.idrow group by 1 into SommeCumul
```