Utilisation avancée de MapInfo sans MapBasic

Partie I

SQL dans sa fenêtre ou via la fenêtre MapBasic

Sébastien RODDIER
Géologue Spécialiste Sites et Sols Pollués & SIG.
sebastien.roddier@caramail.com

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

version initiale 3/10/2002 - Sébastien Roddier– (“Utilisation avancée de MapInfo via la fenêtre MapBasic”)

version révisée avec nouveau titre 23/10/2002 – Jacques Paris

document en format US Letter

Table des matières

La commande « Select » dans MapInfo 1

Généralités 2

1 – Syntaxe générale 2
- Condition avec opérateur de comparaison 3
- Condition avec opérateur de « ressemblance » 3
- Condition avec opérateur géographique 4
- Groupe de conditions 5
- Option INTO 6
  - Option NOSELECT 6
- Option GROUP BY 6
- Option ORDER BY 7

2 – Principes de construction des expressions 7
- Ordre de priorité des opérateurs 8
- Fonctions utiles dans les expressions 8

3 – Principes d'utilisation d'une sous-sélection 10
4 – Sélection par caractéristique graphique avec différents types d'objets 11
5 – Système de coordonnées lors d'une sélection 13
6 – Jointure de plus de 2 tables dans le même "Select". 13

Exemples de mise en œuvre 14

1 – Sélections attributaires 14
- 1 – 1 Sur une table 14
- 1 – 2 Sur plusieurs tables 15
- 1 – 3 Fusion de 2 tables par sélection sur une colonne « commune » 15
- 1 – 4 Sélection des enregistrements d'une table ne se trouvant pas dans une autre 16
- 1 – 5 Sélectionner les enregistrements qui n'ont pas d'objets 16

2 – Requêtes géographiques 16
- 2 – 1 Sélection d'objets en intersection spatiale totale ou partielle. 16
- 2 – 2 Sélection d'objets en intersection spatiale totale 16
- 2 – 3 Requêtes géographiques multiples 17
- 2 – 4 Requêtes basées sur l'adjacence des objets 18
- 2 – 5 Requêtes basées sur la proximité 19
- 2 – 6 Requêtes basées sur les distances entre objets 19
- 2 – 7 Requêtes basée sur la position relative des objets. 19
- 2 – 8 Des points et des lignes 20
- 2 – 9 Sélection des objets qui ne se trouvent pas dans une zone donnée 21
- 2 – 10 Construction d'une matrice d'adjacence pour une table de régions 21

3 – Requêtes utilisant des caractéristiques géométriques 22
- 3 – 1 Requêtes retournant des informations surfaciques sur les objets. 22
- 3 – 2 Requêtes retournant des informations linéaires sur les objets. 23

4 – Requêtes utilisant des caractéristiques graphiques 24
- 4 – 1 Sélections basées sur les attributs graphiques des objets 24
- 4 – 2 Requêtes sur les types d'objets 25
- 4 – 3 Requêtes retournant des informations sur la géographie des objets 26

5 – Travailler avec des objets et données textes 27
- 5 – 1 Sélection d'objets textes en fonction de certaines de leurs caractéristiques 27
- 5 – 2 Concanétation / dé-concanétation de chaîne de caractères 27
- 5 – 3 Sélection des cas dont un champ texte contient une chaîne donnée 28
- 5 – 4 Extraction du dernier mot d'une phrase 28

6 – Groupement des résultats par colonnes. 29
7 – Triage des résultats par ordre croissant ou décroissant 30
8 – Exemples hors catégories 30
- 8 – 1 Assignation de la plus grande fréquence 30

La commande « Select » dans MapInfo

Nous allons tout d'abord expliquer les éléments principaux de la syntaxe de cette commande, et fournir certaines informations élargissant le domaine d'application de cette commande. Puis nous verrons comment mettre en œuvre ces principes dans toute une série d'exemples, tirés d'un peu partout et souvent adaptés à nos objectifs. Mais tout d'abord, quelques remarques préliminaires.

Le résultat d'une sélection en MapInfo a toujours la forme d'une table contenant les enregistrements qui ont satisfait aux conditions imposées et possédant les attributs (colonnes) tels que spécifiés. Nous identifierons cette table comme « résultat_SQL » ; elle est nommée initialement par MI « selection » et renommée « QueryN » si une utilisation en est faite (avec N de 1 à ... durant une même session) mais elle peut recevoir au moment de sa création un nom choisi par l'utilisateur.

Une commande SQL peut être composée et soumise directement dans la fenêtre MapBasic, ou le plus souvent aussi en entrant les divers éléments dans les « boîtes » correspondant de la fenêtre SQL. Nous utiliserons ici la forme « écrite » de la fenêtre MB, laissant à l'utilisateur l'adaptation à la fenêtre SQL; nous signalerons les cas où le recours à la fenêtre MB est la seule possibilité.

Nous devons cependant signaler une différence entre les deux techniques d'entrée des données: elle concerne la visualisation des résultats. L'équivalent de cocher « visualiser les résultats » dans la fenêtre SQL est obtenu par une seconde commande dans la fenêtre MapBasic :

Browse * from <résultat_SQL>

<résultat_SQL> selection ou nom attribué par l'utilisateur avec INTO.

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

Généralités

1 – Syntaxe générale

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

<liste_expressions>

liste des colonnes qui seront présentes dans la sélection, dans l'ordre de cette liste et contenant les résultats des expressions spécifiées.

Si une seule table est spécifiée, le nom de ses colonnes peut être utilisé directement dans les expressions. S'il y a plus d'une table, les références aux colonnes doivent inclure le nom de la table (table_ A.colonne_1, table_B.colonne_6 ...)

Si toutes les colonnes de la table spécifiée sont conservées dans le même ordre et sans changement, on peut remplacer liste_expressions par *****. Comme l'astérisque ne peut pas être utilisé en conjonction avec une autre expression, si plusieurs tables sont spécifiées et l'astérisque utilisé, la sélection contiendra toutes les colonnes de toutes les tables.

L'expression spécifiant le contenu d'une colonne peut être le nom d'une colonne originale ou une formule impliquant une ou plusieurs colonnes originales (voir plus bas « Principes de construction des expressions »). Ex. :

```
Select country, Round(population,1000000), from world
```

Cet énoncé produit une table de 2 colonnes, le nom du pays et sa population en millions.

Une expression peut aussi donner un nom (alias) à la colonne qu'elle crée; le nom entre " " doit suivre alors directement la spécification du contenu. Ex. :

```
Select country “Pays”, Round(population,1000000) “Millions” from world
```

Les deux colonnes ont alors les noms de Pays et Millions (plutôt que country et population de l'exemple précédent).

<nom_table>

nom d'une table ouverte.

Si les expressions font appel à plusieurs tables, celles-ci doivent toutes être spécifiées ici, en les séparant par des virgules.

<groupe_conditions>

un cas est sélectionné s'il remplit les conditions spécifiées dans un groupe de conditions (un groupe peut être réduit à une simple condition)

Une condition doit pouvoir être évaluée comme vraie ou fausse. Elles a deux formes possibles :

- un terme unique:
  « obj » est vrai pour tous les enregistrements pour lesquels un objet existe,
  « col_logic » est vrai si le contenu de cette colonne de type LOGICAL est vrai

- une condition faisant appel à deux termes reliés par un opérateur :
  « col1 > col2 » est vrai pour tous les enregistrements pour lesquels le contenu de la colonne 1 est plus grand que celui de la colonne 2

  Il y a deux types d'opérateurs utilisés dans ces conditions ; les opérateurs de comparaison, et les opérateurs géographiques.

conditions avec opérateur de comparaison

<terme calculé> { < | <= | = | >= | > | <> } <terme de référence>

<terme calculé>

peut être :

- une simple colonne. Si plusieurs tables sont spécifiées, il faut utiliser la forme tableX.colN.
- une expression mettant en jeu une ou plusieurs colonnes avec ou sans appel à des fonctions MapBasic

<terme de référence>

peut être :

- une valeur fixe (12, ou « oui »)
- une simple colonne. Si plusieurs tables sont spécifiées, il faut utiliser la forme tableY.colM.
- une expression mettant en jeu une ou plusieurs colonnes avec ou sans appel à des fonctions MapBasic

Une condition peut être « niée » en utilisant l'opérateur NOT

NOT var1 > 2   équivaut à   var1<= 2

Les variables **OBJ** et **ROWID** (colonnes non visibles) peuvent être utilisées dans toute condition.

conditions avec opérateur de « ressemblance »

<variable_texte> LIKE "chaîne_de caractères"

Ne peut s'appliquer qu'à du texte. La chaîne peut contenir des caractères spéciaux qui permet d'étendre les combinaisons spécifiées par les caractères données :

% (pourcentage)	représente zéro ou un nombre indéterminé de caractères %fle% est vrai pour tout mot contenant fle
_ (sous ligné)	représente un seul caractère __fle_ est vrai pour tout mot de 6 caractères ayant fle en position 3 à 5

conditions avec opérateur géographique

<objetA> <opérateur_géographique> <objetB>

une seule table est spécifiée

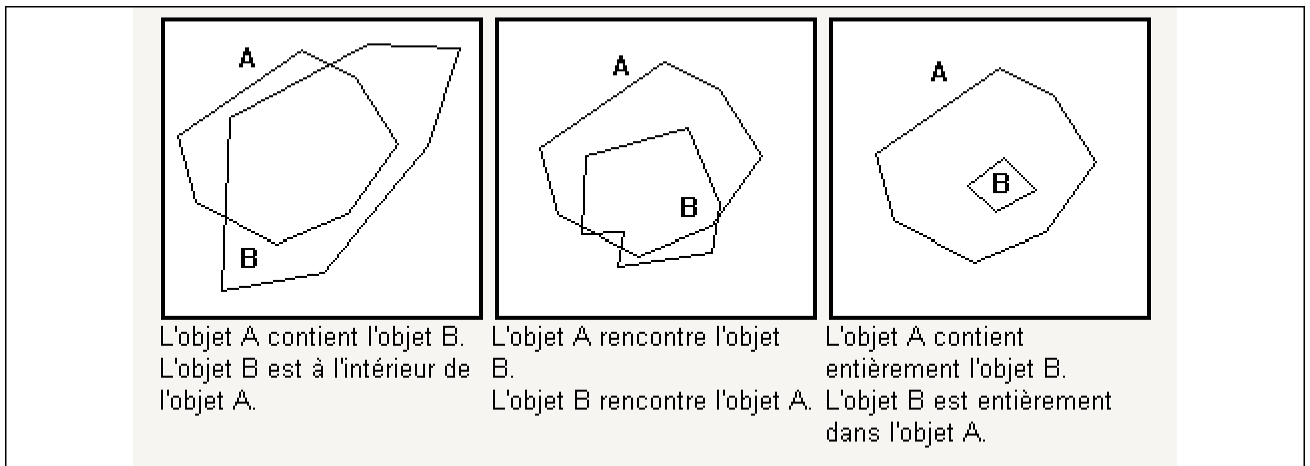
<objetA>	=	obj
<objetB>	=	une variable objet, définie au préalable dans la fenêtre MapBasic

au moins deux tables sont spécifiées

<objetA>	=	tableA.obj
<objetB>	=	tableB.obj

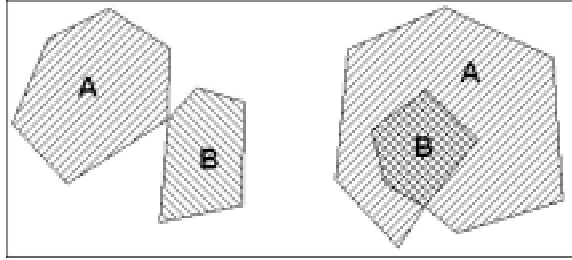
Il y a 7 opérateurs géographiques

objetA **Contains** objetB	objetA contient le centroïde de objetB
objetA **Contains Part** objetB	objetA contient en partie objetB
objetA **Contains Entire** objetB	objetA contient entièrement objetB
objetA **Within** objetB	le centroïde de objetA est dans objetB
objetA **Partly Within** objetB	une partie de objetA est dans objetB
objetA **Entirely Within** objetB	objetA est entièrement dans objetB
objetA **Intersects** objetB	les 2 objets se rencontrent



Les diagrammes ci-dessus peuvent prêter à confusion. Les explications du premier à gauche ne sont vraies que si les 2 centroïdes sont situés dans la partie commune aux 2 objets. Comme il est possible que les centroïdes ne soient pas là où on peut les attendre, un tel diagramme ne montrant pas la position des centroïdes ne peut pas servir de base sûre à une telle estimation.

Il faut aussi remarquer que la « rencontre » (**intersects**, diagramme du milieu) peut être « détectée » si les 2 objets n'ont qu'un seul point en commun comme dans le diagramme suivant où l'objet B est soit à l'extérieur soit à l'intérieur de A, et ceci s'applique aussi à **contains part** et **partly within**:



Toute une série d'équivalences (« vrai » pour toutes), d'exclusions (« vrai » pour une, « faux » pour l'autre) ou d'existences simultanées (l'état de l'une ne permet pas de conclure sur l'état de l'autre) entre expressions peut être imaginée. Si

A contains entire B == B entirely within A

ces 2 expressions « excluent » les suivantes

A partly within B == B partly within A == A contains part B ==
B contains part A == A intersects B == B intersects A

Mais

A contains B n'implique pas que A intersects B

ni que

B contains A (qui n'implique pas que B intersects A)

groupe de conditions

Les conditions impliquant opérateurs de comparaison ou opérateurs géographiques peuvent être assemblées en un groupe en utilisant les opérateurs **AND** (les 2 conditions doivent être vraies pour que le cas soit choisi) et **OR** (une des deux conditions doit être vraie pour que le cas soit choisi). L'utilisation des parenthèses est souvent nécessaire pour contrôler exactement ces assemblages de conditions.

(var1>4 AND var2) OR area(obj, "sq km")>100

sélectionne les cas pour lesquels var1 est plus grand que 4 et var2 (logique) est « vrai », et les cas pour lesquels l'objet correspondant a une superficie supérieure à 100 km carrés (sans dédoublement des cas où les 2 conditions sont vraies)

Simplification de groupes de conditions avec opérateurs de comparaisons

Le groupe de conditions   var1 >= 12 AND var1 <= 23
peut être remplacé par   var1 **BETWEEN** 12 **AND** 23

Noter que les valeurs extrêmes sont retenues dans la nouvelle expression.

Le groupe de conditions   var1 = 1 OR var1 = 4 OR var1 = 6
peut être remplacé par   var1 = **ANY** (1,4,6)
ou   var1 **IN** (1,4,6)

Noter que ANY() est une fonction alors que IN est un mot clé.

Option INTO

INTO <table_résultats> [NOSELECT]

<table_résultats>

nom à attribuer à la table <résultats_SQL>

Si ayant fait une sélection, un quelconque usage est fait de son résultat, par défaut MI conservera temporairement (pour la durée de la session, à moins qu'elle ne soit fermée spécifiquement) le résultat de la sélection avec le nom de table QueryN (N commence à 1 avec une nouvelle session et augmente à chaque sélection qui est « utilisée »).

Pour éviter l'augmentation continue de N (et le risque d'en atteindre la valeur maximum acceptable par MI) ainsi que d'accumuler les tables correspondantes pouvant occuper un espace utile précieux, il est recommandé de donner à la sélection un nom particulier, même si ce nom n'avait pas d'usage particulier par la suite.

Option NOSELECT

Elle permet de ne pas annuler une sélection déjà existante tout en faisant cette nouvelle sélection; sans NOSELCT, la nouvelle sélection remplacerait la précédente. Cette option n'est disponible que dans le cadre de l'option INTO.

Elle offre aussi l'avantage de ne pas nécessiter de rafraîchissement de l'écran (fenêtre carte et/ou tableau) avec la nouvelle sélection d'où un gain de temps appréciable dans certaines conditions.

Option GROUP BY

GROUP BY <liste_colonnes>

Quand cette option est utilisée, la table <résultats_SQL> est différente en structure des tables ordinaires. En effet elle ne contient pas d'objets, et elle compte un nombre de cas (lignes) égal au nombre de valeurs différentes trouvées dans la colonne qui sert à faire le groupement (ou au nombre de combinaisons de valeurs trouvées dans les colonnes de groupement s'il y en a plus qu'une) et les colonnes telles que définies dans <liste_expressions> avec certaines différences comparées à une requête sans GROUP BY.

<liste_expressions> pour cette option

Cette liste doit contenir

- la ou les colonnes servant au groupement sous une forme simple ou utilisée comme argument de certaines fonctions,
  select county, state ...
  donne une table de tous les comtés distingués par état
  select month(colonne_date) ...
  donne une table par mois présent dans colonne_date

- des colonnes pour recueillir les résultats des agrégations.
  - Il peut y avoir une colonne « **Count(*)** » donnant le nombre de cas dans chaque catégorie de la table
  - une ou plusieurs colonnes définies comme suit :

Avg(colonne) Valeur moyenne pour la colonne spécifiée par groupe.
Max(colonne) Valeur maximum pour la colonne spécifiée par groupe.

Min(colonne) Valeur minimum pour la colonne spécifiée par groupe.
Sum(colonne) Somme des valeurs pour la colonne spécifiée par groupe.
WtAvg(colonne, colonne_poids)
Moyenne pondérée par groupe pour la colonne spécifiée pondérée par la colonne_poids

<liste_colonnes>

Si les colonnes sont utilisées sous une forme simple, c'est la liste de noms de colonnes utilisées pour le regroupement, séparées par des virgules si plus d'une, tout comme dans <liste_expressions>.

Si les colonnes de regroupement sont dérivées (obtenues par une fonction par exemple), il faut utiliser pour cette liste les indices des colonnes de la <liste_expressions> servant au regroupement sous la forme « col1, col2 » ou simplement « 1, 2 ». Noter qu'on peut toujours utiliser cette solution même pour les colonnes listées sous forme simple.

Option ORDER BY

ORDER BY <liste_colonnes>

Cette option permet de trier la table <résultats_SQL> en fonction des valeurs contenues dans la ou les colonnes spécifiées pour le tri.

<liste_colonnes>

Même façon de construire la liste que pour ORDER BY.

Le tri se fait par défaut en ordre ascendant ou croissant. Si une colonne doit être triée dans l'autre manière, il faut ajouter un DESC après son nom. Ainsi

Select * cities ORDER BY State, population DESC

produit une table <résultats_SQL> des états en ordre alphabétique normal, et à l'intérieur de chaque état des villes en ordre décroissant de leurs populations

2 – Principes de construction des expressions

Une expression contient les règles de calcul d'une nouvelle colonne (=variable) à partir d'une ou plusieurs colonnes originales. Elle peut être aussi simple que le nom de la colonne originale. Elle peut impliquer une ou plusieurs colonnes dont l'assemblage est fait à l'aide d'opérateurs et/ou de fonctions, le tout structuré par l'utilisation de parenthèses pour introduire une priorité à respecter dans les calculs .

Les opérateurs disponibles sont les suivants :

+	somme	a + b
-	différence	a - b
	négation	-a
*	multiplication	a * b
/	division	a / b
\	division en nombres entiers	a \ b

	(partie décimale oubliée)	
Mod	reste d'une division en nombres entiers (a = (a \ b) * b + a Mod b)	a Mod b
^	élevé à une puissance	a ^ b
&	concanétation de chaînes de caractères (le + peut être aussi utilisé en MB)	e & f

- Ordre de priorité des opérateurs

Lorsque MapInfo évalue des expressions, il doit savoir quels éléments calculer en premier, autrement dit quelle est leur priorité. Par convention, les opérateurs ont des niveaux de priorité différents. Ceux dont la priorité est la plus élevée sont calculés en premier. Le tableau qui suit indique l'ordre de priorité des divers opérateurs de MapInfo. Les opérateurs ayant la même priorité sont évalués selon leur place dans l'expression de la gauche vers la droite.

Priorité la plus haute	()	parenthèses
	^	puissance
	-	négation
	* / \ Mod	multiplication, division
Priorité la plus basse	+ - &	addition, soustraction

- Fonctions utiles dans les expressions

Fonctions trigonométriques

Cos(num)	Cosinus d'un nombre num est exprimé en radians.
Sin(num)	Sinus d'un nombre num est exprimé en radians.
Tan(num)	Tangente d'un nombre num est exprimé en radians.
Acos(num)	Angle en radians dont le cosinus est num
Asin(num)	Angle en radians dont le sinus est num
Atan(num)	Angle en radians dont la tangente est num

Fonctions de troncation numérique

Fix(num)	Partie entière d'un nombre
Int(num)	Entier le plus proche <= num
Round(num1, num2)	Nombre arrondi de num1 à la valeur la plus proche de num2 (par ex., si num2 =10, num1 est arrondi à la dizaine la plus proche).

Autres fonctions mathématiques

Abs(num)	Valeur absolue d'un nombre.
Exp(num)	e à la puissance (num)
Log(num)	Logarithme naturel (base de e) de num (doit être >0). Pour obtenir le logarithme dans la base 10, calculer: log(num) / log(10)
Maximum(num1 , num2)	Le plus grand de deux nombres.
Minimum(num1 , num2)	Le plus petit de deux nombres.
Rnd(1)	Nombre au hasard
Sgn(num)	–1, 0 ou 1 selon le signe de l'expression calculée
Sqr(num)	Racine carrée de num (doit être >0).

Fonctions pour chaînes de caractères

Fonction	Description
Asc(chaîne)	Code du premier caractère de (chaîne)
Chr$(num)	Caractère qui correspond au code caractère (Chr$(65) >> "A").
DeformatNumber$(ch)	Chaîne ne comportant pas de séparateur de milliers
Format$(num , “modèle“)	Chaîne représentant un nombre mis en forme selon le modèle donné. Par exemple, Format$(12345.678, "$,#.##") retourne "$12,345.68".
FormatNumber$(num)	Chaîne représentant un nombre mis en forme selon les spécifications du système d’opérations local. Cette fonction plus simple à utiliser que Format$ offre un contrôle moindre sur le format
InStr(num , ch1 , ch2)	Position dans la chaîne ch1 à partir de la position num, de l’occurrence de la chaîne ch2 (zéro si cette chaîne ch2 est introuvable). Pour commencer la recherche au début de la chaîne, attribuez à num la valeur un (1).
LCase$(ch)	Chaîne ch en caractères minuscules.
Left$(ch , num)	Les num premiers caractères de la chaîne ch.
Len(ch)	Nombre de caractères d'une chaîne, y compris les espaces du début, entre les mots et de la fin.
LTrim$(ch)	Chaîne sans les espaces éventuels au début de ch
Mid$(ch, num1, num2)	Chaîne des num2 caractères de la chaîne ch à partir de la position num1.
Proper$(ch)	Chaîne avec la première lettre de chaque mot en majuscules.
Right$(ch , num)	Chaîne des num derniers caractères de la chaîne ch.
RTrim$(ch)	Chaîne sans les espaces éventuels à la fin de ch
Space$(num)	Chaîne de num espaces
Str$(expr)	Chaîne représentant un « nombre », le type d’un objet (Line, Text...) ou une date éctite selon les spécifications du système d’opérations local
String$(num, “expr”)	Chaîne formée de num répétitions de l’expression “expr”
UCase$(ch)	Chaîne ch en caractères majuscules.
Val(ch)	Valeur numérique de la chaîne ch. Par exemple, Val("18 rue X") retourne le nombre 18.

Fonctions géographiques

Fonction	Description
Area(obj , “unité“)	Superficie d'un objet. Unité de superficie, telle que "sq km" ou "hectare".
AreaOverlap(obj1, obj2)	Surface commune à deux objets fermés, en unités courantes.
CentroidX(obj)	Abscisse (coordonnée x) du centroïde d'un objet.
CentroidY(obj)	Ordonnée (coordonnée y) du centroïde d'un objet.
Distance(num_x1, num_y1, num_x2, num_y2, “unité“)	Distance entre deux points. Coordonnées x et y du point de départ puis celles du point d'arrivée. Unité de distance, telle que "mi" ou "km".
ObjectLen(obj , “unité“)	Longueur de l'objet. Unité de distance, telle que "mi" ou "km". Seuls les objets lignes, polylignes et arcs de cercle ont une longueur.
Perimeter(obj , “unité“)	Périmètre d'un objet. Unité de distance, telle que "mi" ou "km". Seuls les objets polygones, ellipses et rectangles ont un périmètre
ProportionOverlap(obj1, obj2)	Proportion de l’objet obj1 recouvert par l’objet obj2.

A partir de la version 5.5, MapInfo a introduit de nouvelles façons de calculer certaines valeurs Ainsi Area(), Distance(), ObjectLen() et Perimeter() existent aussi en deux variantes : CartesianArea(), CartesianDistance(), CartesianObjectLen() CartesianPerimeter() and SphericalArea(),SphericaDistance(),SphericaObjectLen(), SphericaPerimeter(). Les algorithmes de calculs tiennent compte du système de coordonnées de la table. Les « cartesian » appliqués à des tables non projetées (en degrés) retournent la valeur –1. Les « spherical » appliqués à des tables « non-earth » (ne pouvant pas être converties en coordonnées sphériques) retournent la valeur –1.

Les fonctions originales utilisent un algorithme « spherical » sauf si la table est en « non-earth ».

Fonctions concernant les dates

Une variable dimensionnée « Date » est emmagasinée dans MI sous la forme AAAAMMDD. Une telle variable peut être utilisée dans des calculs très limités :

date + ou – constante numérique
date1 – date2

(Date) est une valeur représentant une date dans le format AAAAMMDD
(chaîne_date) est une date dans le format du système d'opérations local sous la forme d'une chaîne de caractères.

CurDate()	Date courante dans le format AAAAMMDD
Day(date)	Quantième du mois (1 - 31)
FormatDate$(date)	Date mis en forme selon les spécifications du système d'opérations local.
Month(date)	Mois (1 - 12).
StringToDate(chaîne_date)	Date dans le format AAAAMMDD
Weekday(date)	Jour de la semaine (1 - 7). 1 représente le dimanche
Year(date)	Année (par exemple, 1994).

Fonctions retournant des objets

Les fonctions qui retournent des objets ne peuvent pas être utilisées dans un « Select », la liste des expressions ne pouvant contenir spécifiquement la colonne OBJ. Ne pouvant utiliser SQL pour créer des objets par transformation, il faut passer par UPDATE qu l'on applique éventuellement sur une sélection préalable.

3 – Principes d'utilisation d'une sous-sélection

Le « Select » permet d'utiliser ce que MI appelle une sous-sélection, qui devrait plutôt être appelée une pré-sélection, dans des conditions bien définies. D'abord elle ne peut être qu'un des termes d'une comparaison; ensuite la comparaison doit tenir compte de la nature des résultats de cette sous-sélection; finalement il ne peut y en avoir qu'une par « Select »

Je préfèrerais le terme pré-sélection parce que quand elle est présente dans un « Select », Mi commence par la calculer, c'est à dire prépare une table contenant les résultats, et se sert ensuite de cette table comme un terme constant dans la comparaison correspondante. Une conséquence de cette façon de fonctionner est qu'il importe peu techniquement que la sous-sélection soit dans le premier ou le deuxième terme de la comparaison puisqu'elle est toujours calculée en premier.

Voici un exemple tout simple : pour connaître les départements d'une table (dep) qui n'ont pas d'étangs (définis dans une table etang):

select * from dep where not obj contains any (select obj from etang)

La sous-sélection (select obj from etang) est **contenue entre parenthèses** et entre comme argument de la fonction ANY() dans **la définition du second terme d'une comparaison** « niée » (précédée de l'opérateur NOT) et utilisant un **opérateur** géographique « contains ».

Une sous-sélection extrait une table temporaire (invisible) avec **une seule variable** qui doit être en rapport avec le premier terme de la comparaison; cette variable peut être

- une des colonnes de la table spécifiée ou « obj »
- une expression basée sur une ou plusieurs colonnes
- une valeur « agrégée » (il n'est pas nécessaire d'utiliser GROUP BY dans ce contexte)

```
(select col2 from tableC)
(select area(obj,"sq,km") from regions)
(select avg(population) from depart)
```

Toute combinaison est possible; ainsi, pour sélectionner toutes les régions dont la surface est supérieure à la moyenne (non connue), une possibilité est la suivante

```
select * from tab_1 where area(obj,"sq,km") > (select avg(area(obj,"sq,km")) from tab_1)
```

Une sous-sélection peut contenir des conditions précédées d'un « WHERE ». Le résultat d'une sous-sélection peut aussi être dirigé vers une nouvelle table visible avec un « INTO », mais cela est probablement pas bien efficace et n'est pas recommandé.

L'opérateur doit pouvoir accepter les résultats de la sous-sélection. Si le résultat est une valeur unique (comme avec la valeur agrégée par la fonction AVG()) une comparaison directe peut être faite (<, <=, ...). Si le résultat contient plusieurs valeurs, on doit utiliser alors des opérateurs plus complexes comme

= ANY (sous-sél,)	pour trouver les cas d'une table ayant la même clé dans une autre table
<> ALL (sous-sél.)	pour trouver les cas d'une table absent dans l'autre sur la base d'une clé

4 – Sélection par caractéristique graphique avec différents types d'objets

Les caractéristiques graphiques sont celles que l'on peut obtenir avec certaines fonctions MapBasic comme ObjectInfo(), ObjectGeography()... . Elles ont trait à des données géométriques (coordonnées de nœuds ...), de style de présentation (Pen, Brush...) même de contenu (le texte d'un objet texte).

Problème : certaines caractéristiques graphiques n'existent pas pour tous les types d'objets.

Ainsi

```
select * from polys where str$(objectinfo(obj,3))<>"Brush (2, 0, 16777215)" into sel
```

entraîne une erreur « objectinfo argument 2 out off range » si la table contient d'autres objets que des régions et des textes (la valeur 3 du deuxième argument n'est reconnue que pour les régions comme brosse et pour les textes comme chaîne de caractères).

On pourrait imaginer rajouter une condition quant au type de l'objet comme dans

```
select * from polys where str$(objectinfo(obj,3))<>"Brush (2, 0, 16777215)"
and int(objectinfo(obj,1))=7 into sel
```

La même erreur apparaîtrait car dans MI les deux comparaisons du groupe sont évaluées pour chaque cas indépendamment et cela quel que soit leur ordre.

Une première solution consiste à procéder par étape, une première sélection fait sortir les objets du type voulu, une deuxième faite sur les résultats de la première ceux avec la caractéristique recherchée.

```
select * from polys int(objectinfo(obj,1))=7 into sel1
select * from sel1 where str$(objectinfo(obj,3))<>"Brush (2, 0, 16777215)" into sel2
```

Une deuxième solution fait appel à une sous-sélection et n'a donc pas besoin d'une table intermédiaire.

```
select * from polys where
col1 in (select col1 from polys where str$(objectinfo(obj,1))="7"))
and str$(objectinfo(obj,3))<>"Brush (2, 0, 16777215)" into sel
```

Il faut utiliser col1 (qui existe toujours dans une table MI) et non obj parce que si on utilise obj, on obtient une erreur du type « Data mismatch »; il faut se rappeler que MI ne peut pas faire de comparaisons entre objets, seulement sur des valeurs attributaires et l'opérateur « in » implique qu'il y a comparaison. La colonne spécifiée dans l'expression de sous-sélection n'est ici qu'un prétexte pour permettre la construction de la sous-table sur la base du « where ». La table "Lines" contient des polylignes, la table "Areas" des régions. Il faut trouver les polylignes dont le point de départ est dans une des régions.

```
SELECT * FROM Lines WHERE
    CreatePoint(ObjectNodeX(obj,1,1),ObjectNodeY(obj,1,1))
    WITHIN ANY (SELECT obj FROM Areas)
```

Pour trouver les polylignes finissant dans une des régions, il faut faire l'hypothèse que ce sera la fin de la première section si la polyligne a plusieurs sections.

```
SELECT * FROM Lines WHERE
    CreatePoint(ObjectNodeX(obj,1,objectinfo(obj,22)),
    ObjectNodeY(obj,1,objectinfo(obj,22)))
    WITHIN ANY (SELECT obj FROM Areas)
```

Il n'est pas possible de travailler avec des "fins de polylignes" ayant plusieurs sections car il faudrait alors « encastrer » un autre niveau de fonction et Mapinfo ne l'accepte pas, comme dans l'extrait suivant

```
ObjectNodeX(obj,objectinfo(obj,21),
objectinfo(obj,objectinfo(obj,21)+1))
```

objectinfo() dans le deuxième argument est acceptable mais pas celui qui est enchâssé dans le troisième.

5 – Système de coordonnées lors d’une sélection

Par défaut, MapBasic/MapInfo utilise le système de coordonnées non-projetées en degrés décimaux de longitude et latitude; ce système peut être modifié par l’utilisateur (ou par des applications lancées par lui) mais il n’y a pas de fonction permettant de savoir exactement quel est le système courant..

Tout appel à une fonction produisant des coordonnées retournera les coordonnées dans le système courant, toute constants passée pour des calculs sera lue dans ce même système. Pour être sûr que le bon système est « actif », il suffit de faire exécuter dans la fenêtre MapBasic, la commande suivante :

```
Set coordsys <spécification du système de coordonnées>
```

Si on veut utiliser le système tel que défini dans une table ouverte en particulier, la commande se simplifie en

```
Set coordsys table <nom_de_la_table>
```

6 – Jointure de plus de 2 tables dans le même “Select”.

Il faut pour cela dans le « Where » établir d ‘abord une relation entre 2 tables, puis une relation entre la deuxième et la troisième. Il est fortement recommandé de lister les tables dans l’ordre avec lequel les relations sont établies.

```
Select * from  pointtable, regiontable, pointable2
        where pointtable.obj within regiontable.obj
        and regiontable.obj contains pointtable2.obj
```

Exemples de mise en œuvre

Remarques importantes :

Nous employons les termes Requêtes et Sélections pour différencier deux finalités différentes mais complémentaires dans certains cas de l'utilisation de « Select ». Le résultat d'une « sélection » est une table ne contenant que les cas qui ont été retenus par le <groupe_conditions> utilisé. Une requête produit une table dont les colonnes sont différentes de celles dans la(es) table(s) originale(s) , soit parce qu'elles ne s'y retrouvent pas toutes (quand * n'est pas utilisée), soit parce que certaines ont été ajoutées selon la <liste_expressions> (colonnes dérivées). Il est bien entendu que les deux finalités peuvent exister dans une même commande.

Dans tous les cas, la nouvelle table devra être sauvegardée, qu'elle ait reçu un nom (avec l'option INTO) ou non. Cette possibilité qui existe toujours n'est pas mentionnée dans les exemples qui suivent pour ne pas encombrer le texte.

Dès qu'une colonne dérivée est définie dans liste_expressions, toutes les autres colonnes que l'on veut conserver dans la table dérivée doivent être explicitement présentes dans la liste car l'utilisation de * (= toutes les colonnes) est incompatible avec la spécification d'une seule, dérivée ou non. S'il s'agit donc de rajouter une colonne dérivée à une table, le plus efficace serait alors d'ajouter une colonne et de la mettre à jour avec « Update ».

1 – Sélections attributaires

1 – 1 Sur une table

Objectif : sélectionner des données/objets en fonction de la valeur de un ou plusieurs attributs provenant d'une seule table.

Syntaxe :

```
SELECT {liste_expressions|*} FROM <nom_table>
        WHERE <groupe_conditions>
```

Exemple1 :

```
select * from lithologie where Code = 1
select col1, col3, col4 from lithologie
        where Code <> 1 and (Rowid between 25 and 50)
```

Exemple2

```
select * from villes where (pop_91 – pop_81)/pop_81>1.2
```

Sélectionne les villes dont la population a augmenté de plus de 20% entre 1981 et 1991

Exemple3 :

```
select Nom, pop_91 – pop_81 ”Evolution 1981-91” from villes
```

Retourne une table contenant le nom des villes et la différence entre les populations de 1981 et 1991.

Objectif : sélectionner des données/objets en fonction d'un attribut-caractères

Syntaxe : SELECT {liste_expressions|*} FROM <nom_table>
WHERE <colonne_caractères> LIKE <référence>

Exemple1 : select * from MaTable where Nom_Rue LIKE "%Avenue%"

Trouve toutes les rues dont le nom contient Avenue

Exemple2 : select * from MaTable where Nom_Rue LIKE "Avenue%"

Trouve toutes les rues dont le nom commence par Avenue

Exemple3 : select * from MaTable where Nom_Rue LIKE "%Avenue"

Trouve toutes les rues dont le nom finit par Avenue

Noter que LIKE est insensible à la casse.

1 – 2 Sur plusieurs tables

Syntaxe : SELECT {liste_expressions|*} FROM <Table_1>, <Table_2> WHERE
< table_1.ChampX > {opérateur} < table_2.ChampY >

Exemple1 : Select lithologie.Code, Lithologie_code.Libellé_ from lithologie, Lithologie_code where lithologie.Code = Lithologie_code.Code

Retourne une table contenant le code lithologique (pris dans la table lithologie) et le libellé de la lithologie (pris dans la table lithologie_code). Ces deux tables ont pour champ commun « Code ».

Exemple2 : Select lithologie.Code, Lithologie_code.Libellé_ from lithologie, Lithologie_code where lithologie.Code = Lithologie_code.Code and Lithologie_code.Code = 1

Retourne une table contenant le code lithologique (pris dans la table lithologie) et le libelle de la lithologie (pris dans la table lithologie_code) incluant les cas ayant pour valeur de « Code » 1.

Exemple3 : Select * from lithologie where not Code in (select Code from Lithologie_code)

Permet de sélectionner les cas de la table « Lithologie » qui n'ont pas de correspondance de l'attribut Code dans la table « Lithologie_code ». Très utile pour déterminer si tous les codes d'une table ont été traduits en clair dans une table de correspondance.

1 – 3 Fusion de 2 tables par sélection sur une colonne « commune »

Si deux tables ont une colonne contenant le même identificateur unique (pas de répétitions dans l'une ni dans l'autre), on peut fusionner ces deux tables par un select

Syntaxe : SELECT * FROM <table1>,<table2>
WHERE <table1.col_ID> = <table2.col_ID>

Exemple : select * from villes_1990, villes_2000 where villes_1900.code = villes_2000.code into sel_villes

La table sel_villes devra être sauvegardée sous ce nom ou un autre. Elle contient toutes les colonnes des deux tables (y compris celles ayant servi à faire la jointure) et tous les cas qui se retrouvent dans les 2 tables, à l'exclusion donc des cas présents dans une seule table.

1 – 4 Sélection des enregistrements d'une table pas dans une autre

Deux tables contiennent au moins un champ « commun » pouvant avoir un nom différent dans chaque table, ici Champ_RefA dans une table, Champ_RefB dans l'autre. Il s'agit de trouver les enregistrements de TableA qui ne sont pas dans TableB.

Syntaxe : SELECT * FROM <TableA> WHERE NOT <Champ_RefA> IN (SELECT <Champ_RefB> FROM TableB)

1 – 5 Sélectionner les enregistrements qui n'ont pas d'objets

Syntaxe : SELECT * FROM <Table> WHERE NOT OBJ

2 – Requêtes géographiques

Objectifs : Sélectionner des cas en fonction des relations spatiales qui les unissent.

2 – 1 Sélection d'objets en intersection spatiale totale ou partielle.

Syntaxe : SELECT {liste_expressions|*} FROM <Table1>, <Table2> WHERE <Table1.obj> INTERSECTS <Table2.obj>

Exemple : Select * from Mailles, lithologie where Mailles.Obj Intersects lithologie.Obj

Tous les objets de la table « mailles » en intersection avec des objets de la table « Lithologie » sont sélectionnés.

2 – 2 Sélection d'objets en intersection spatiale totale

Syntaxe : SELECT {liste_expressions|*} FROM <Table1, Table2> WHERE <Table1.obj> ENTIRELY WITHIN <Table2.obj>

Exemple : Select * from Mailles, lithologie where Mailles.Obj Entirely Within lithologie.Obj

Tous les objets de la table « mailles » entièrement contenus dans des objets de la table « Lithologie » sont sélectionnés.

2 – 3 Requêtes géographiques multiples

<u>Objectifs</u> : Sélectionner des cas en fonction des relations spatiales qui les unissent en spécifiant des sous-sélections sur la valeur de un ou plusieurs attributs.

<u>Syntaxe</u> : SELECT {liste_expressions|*} FROM <Table1, Table2>
WHERE <Table1.obj> INTERSECTS <Table2.obj> AND <condition>

<u>Exemple1</u>: Select * from lithologie, hydro where lithologie.Obj intersects hydro.Obj and hydro.TOPONYMIE = "LE TAGNONE"

Tous les objets de la table « Lithologie » en intersection avec la rivière « Le Tagnone » sont sélectionnés.

Autre possibilité d'arriver au même résultat :

<u>Exemple2</u>: Select * from lithologie where obj intersects any
(select obj from hydro where hydro.TOPONYMIE = "LE TAGNONE")

Dans l'exemple 1, la table résultats_SQL contient les attributs des deux tables alors que dans le second exemple elle ne contient que les attributs de la table « Lithologie ».

<u>Exemple3</u> : Select * from Mailles, lithologie, hydro where Mailles.obj intersects lithologie.obj and lithologie.obj Intersects hydro.obj and hydro.TOPONYMIE = "LE TAGNONE"

Les objets de la table « Mailles » qui se trouvent en intersection même partielle avec les objets de la table « Lithologie » eux mêmes en intersection avec la rivière « Le Tagnone » sont sélectionnés.

Dans cet 'exemple, les cellules de « Mailles » sont sélectionnées autant de fois qu'elles se trouvent en intersection d'une part avec les objets de la table « Lithologie » d'autre part avec les objets de la table « Hydro »

<u>Exemple 4</u> : Select * from Mailles, lithologie, hydro where Mailles.obj intersects lithologie.obj and lithologie.obj Intersects hydro.obj and hydro.TOPONYMIE = "LE TAGNONE" group by Mailles.z

Pour s'affranchir de la contrainte de l'exemple 3, on demande de grouper la sélection selon un identifiant de la table « Mailles » (champ : Mailles.z »). Le logiciel ne conserve alors que les occurrences uniques de cet identifiant par contre il perd la relation avec les objets graphique. La table résultante est uniquement attributaire.

2 – 4 Requêtes basées sur l'adjacence des objets

<u>Objectifs</u> : Ce type de requête permet de trouver pour un objet donné l'ensemble des objets qui lui sont contigus.

<u>Sur l'ensemble d'une table</u>

Pour se faire, nous allons créer une copie de la table d'origine pour comparer la position des objets de la table d'origine par rapport aux objets de la copie de la table d'origine. La colonne ID doit contenir des identificateurs uniques.

<u>Syntaxe</u> : SELECT {liste_expressions|*} FROM <Table1>, <Copie de Table1>
WHERE <Table1.obj> INTERSECTS <Copie de Table1.obj> AND
<Table1.ID> <> <Copie de Table1.ID>

<u>Exemple</u> : Select Mailles.z, Mailles_bis.z from Mailles, Mailles_bis where Mailles.obj Intersects Mailles_bis.obj and Mailles.z <> Mailles_bis.z.

Le résultat est une table à deux colonnes où pour chaque identificateur de la table d'origine (Mailles.z), on a la liste des identificateurs des objets adjacents (Mailles_bis.z).

<u>Pour un objet donné</u>

<u>Syntaxe1</u> : SELECT {liste_expressions|*} FROM <Table>
WHERE OBJ INTERSECTS <variable objet> AND
AREAOVERLAP(OBJ, <variable objet>) = 0

La variable objet est définie grâce à la fenêtre MapBasic en sélectionnant d'abord la région voulue.

<u>Exemple</u> : select * from ma_table where obj intersects reg and areaoverlap(obj, reg)=0

Avec au préalable dans le fenêtre Mapbasic
dim reg as object
reg = selection.obj

La sélection contient toutes les régions adjacentes par au moins un point à celle choisie et exclue la région de référence.

<u>Syntaxe2</u> : SELECT {liste_expressions|*} FROM <Table>
WHERE OBJ INTERSECTS
ANY(SELECT OBJ FROM <Table> WHERE <cas_ID>)

<cas_ID> est une condition permettant d'identifier un cas spécifique, comme le nom d'une colonne avec identificateur unique « = » une valeur donnée

<u>Exemple</u> : Select * from matable where obj intersects any (select obj from matable where matable.CODE_DEPART = 46)

La sélection contient tous les départements limitrophes du 46 y compris le 46.

Cette syntaxe exige de connaître la valeur de l'identificateur unique. Elle pourrait être complétée par une condition d'exclusion de la région de référence telle que

and matable.CODE_DEPART <> 46

2 – 5 Requêtes basées sur la proximité

Objectifs : Trouver des objets se trouvant à une distance donnée d'un objet sélectionné.

Syntaxe : SELECT {liste_expressions|*} FROM <Table> WHERE OBJ WITHIN ANY (SELECT BUFFER (OBJ , <lissage du buffer>, <distance de sélection> , "unité de distance") FROM SELECTION)

Exemple : Select * from Forages where obj within any (select buffer (obj , 12 , 10 , "km") from selection) into sel

Dans l'exemple ci dessus tous les points de la table « Forages » se trouvant à moins de 10 km de l'objet sélectionné sont identifiés.

2 – 6 Requêtes basées sur les distances entre objets

Objectifs : Trouver les distances qui séparent les objets d'une table par rapport à un point sélectionné. Le champ ID de la table contient des identificateurs uniques.

Syntaxe : SELECT <Table.ID>, DISTANCE (<coordx de l'objet sélectionné>, <coordy de l'objet sélectionné>, CENTROIDX(obj), CENTROIDY(obj), "km") ["alias"] FROM <Table> [ORDER BY [Distance | alias]]

Exemple1 : Select Forages.INAT, distance (1166592.800, 1708088.100, centroidx(obj), centroidY(obj), "km") "Distance" from Forages order by DISTANCE

Dans l'exemple ci dessus nous calculons les distances qui séparent l'objet de coordonnées (1166592.800, 1708088.100) des centroïdes des objets de la table forage et nous trions les résultats par distance croissante.

Exemple2 : Select Forages.INAT, min(distance (1166592.800, 1708088.100, centroidx(obj), centroidY(obj), "km")) "Distance Min" from Forages where Forages.INAT <> "11147X0034"

Dans l'exemple ci dessus nous sélectionnons l'objet se trouvant le plus près de l'objet de coordonnées (1166592.800, 1708088.100) en excluant le forage spécifié..

2 – 7 Requêtes basée sur la position relative des objets.

Objectifs : Trouver des objets se trouvant dans une certaine position par rapport à un autre objet.

Syntaxe : SELECT {liste_expressions|*} FROM <Table>
WHERE <groupe_expressions_position_relative>

<groupe_expressions_position_relative>

Mêmes règles générales que pour <groupe_expressions > mais une expression est de la forme

<mesure de position de OBJ> { < | <= | = | >= | > | <> } <mesure de réf'érence>

La mesure de position est fournie par une des fonctions disponibles comme CentroidX() ou CentroidY(). Il faut s'assurer avant que **la mesure de référence soit exprimée avec les unités dont se sert MapBasic à ce moment là**.

Exemple1 : Select INAT from Forages where CentroidY(obj) > 1708088.100 and INAT <> "11147X0034"

Dans l'exemple ci dessus tous les objets de la table forages situés au nord du point de coordonnées (1166592.800, 1708088.100) et dont le numéro d'identificateur est différent de "11147X0034" sont sélectionnés.

En combinant les expressions logiques sur CENTROIDX(OBJ) et sur CENTROIDY(OBJ) on peut arriver à sélectionner les objets dans n'importe qu'elle direction.

2 – 8 Des points et des lignes

Supposons qu'une table contient des polylignes (un réseau de quelque chose) et une autre des points représentant les intersections/jonctions de polylignes et leurs extrémités isolées. Comment sélectionner les polylignes passant par un point donné, ou les points reliés à une polyligne ?

Dans les 2 cas nous commençons par « charger » l'objet donné grâce à la fenêtre MapBasic

```
dim o as object
o = selection.obj
```

Si un point est donné

Syntaxe :
```
SELECT * FROM <table polylignes>
        WHERE OBJ INTERSECTS BUFFER (<objet>, 6, <largeur>, "unité")
```

<largeur> le rayon du tampon. Il suffit d'être certain que le tampon est réellement créé. La largeur est exprimée en **"unité"** de distance

Exemple : select * from aqueducs where obj intersects buffer(o, 6,1 , "m")

Le nombre de 6 points pour former un cercle (ou le demi-cercle autour des extrémités) est bien suffisant et accélère le travail. Dans cette carte en projection métrique, un mètre fera l'affaire. En cas de doute, on peut jouer avec ce paramètre.

La sélection contient les polylignes passant par le point donné.

Si une polyligne est donnée

Syntaxe :
```
SELECT * FROM <table points>
        WHERE OBJ INTERSECTS BUFFER (<objet>, 6, <largeur>, "unité")
```

<largeur> le rayon du tampon. Il suffit d'être certain que le tampon est réellement créé. La largeur est exprimée en **"unité"** de distance

Exemple : select * from extrémités where obj intersects buffer(o, 6,1 , "m")

Le nombre de 6 points pour former un cercle (ou le demi-cercle autour des extrémités) est bien suffisant et accélère le travail. Dans cette carte en projection métrique, un mètre fera l'affaire. En cas de doute, on peut jouer avec ce paramètre.

La sélection contient les points aux extrémités et le long de la polyligne donnée.

2 – 9 Sélection des objets qui ne se trouvent pas dans une zone donnée

Une table contient les objets, l'autre une région donnée.

Régions.

Syntaxe : SELECT * FROM <table_régions> WHERE NOT OBJ INTERSECTS ANY(SELECT OBJ FROM <Table_zone>)

Points.

Syntaxe : SELECT * FROM <table_points> WHERE NOT OBJ WITHIN ANY(SELECT OBJ FROM <Table_zone>)

2 – 10 Construction d'une matrice d'adjacence pour une table de régions

Ce problème ne peut être résolu que par une procédure itérative, ce qui n'est pas l'idéal quand il n'y a que la fenêtre MapBasic de disponible. Il implique aussi l'existence d'une colonne (logical) par région ce qui limite le nombre de régions à 250. Mais c'est une bonne occasion de montrer comment on peut réaliser une boucle de la fenêtre MB.

La table Régions ne contient que des régions et a été compactée auparavant et contient autant de colonnes Logical qu'il y a de régions.

Entrez dans la fenêtre MB et exécutez les lignes suivantes (les numéros ne sont que comme référence pour les explications ; ils ne font pas partie des commandes)

```
1	dim i as smallint
2	dim colref as alias
3	dim adjobj as object
4	i=1
5	colref="col"+i
```

Entrez les lignes suivantes sans les exécuter. Puis sélectionnez les toutes et exécutez. En cliquant sur la barre de la fenêtre MB, réactivez ce bloc et exécutez ; répétez jusqu'à la fin

```
6	fetch rec i from regions
7	adjobj=polyson.obj
8	select * from regions where rowid <> i and obj intersects adjobj into adjzons noselect
9	update adjzons set colref=1
10	i=i+1
11	colref="col"+i
```

i (1, 4) est le compteur d'enregistrements et adjobj (3) l'objet permettant de faire les intersections. Colref est un alias (2, 5) qui nous permettra de choisir la bonne colonne par sa position de 1 jusqu'au nombre de régions quel que soit l'entête qu'elle ait dans la table.

(6) On se positionne d'abord sur le bon enregistrement (N.B. cette technique exige qu'il n'y ait pas d'enregistrements sans objet graphique, ni « vide ») pour saisir l'objet (7).

Le Select (8) retient tous les objets qui touchent à l'objet choisi à l'exclusion de lui-même (N.B. « intersects » détecte tout objet « partageant » l'espace de l'objet de référence même si ce n'est que par un seul point en commun. Le résultat de la sélection est mis dans la table temporaire adjzons et l'option NOSELECT est utilisée car nous ne voulons pas nécessairement voir ces sélections et elle no fera ainsi gagner du temps.

La table temporaire est mis à jour pour la colonne spécifiée par l'alias (9)

Le compteur et l'alias sont « augmentés » (10, 11) et la boucle peut reprendre.

3 – Requêtes utilisant des caractéristiques géométriques

3 – 1 Requêtes retournant des informations surfaciques sur les objets.

Sur une table

<u>Objectifs</u> *: Sélectionner des objets en fonction de leurs caractéristiques surfaciques.*

<u>Syntaxe</u> : SELECT <liste_de_variables> , AREA(OBJ, "unité de surface") ["alias"]
FROM <Table>

<u>Exemple1</u> : Select ID_#, area(obj, "sq km") "surface" from lithologie

Retourne l'identificateur et la surface de chaque objet de la table « Lithologie »

<u>Exemple2</u> : Select Code, sum(area(obj, "sq km")) "surface" from lithologie
group by Code order by Code

Retourne la surface totale des objets par occurrence de l'attribut « Code » trié

Sur plusieurs tables

<u>Objectifs</u> *: Trouver les surfaces d'intersection d'objets se trouvant dans des tables différentes.*

<u>Syntaxe1</u> : SELECT <Table1.ID>, <Table2.ID>,
AREAOVERLAP (Table1.OBJ, Table2.obj) ["alias"]
FROM <Table1> ,<Table2> WHERE Table1.OBJ INTERSECTS Table2.Obj

<u>Syntaxe2</u> : SELECT <Table1.ID>, <Table2.ID>,
PROPORTIONVERLAP (Table1.OBJ, Table2.obj) ["alias"]
FROM <Table1> ,<Table2> WHERE Table1.OBJ INTERSECTS Table2.Obj

Exemple1 : Select lithologie.Code, Mailles.z, areaoverlap(lithologie.obj, Mailles.obj) "Surface d'intersection" from lithologie, Mailles where lithologie.Obj Intersects Mailles.Obj

Retourne la surface d'intersection des objets de la table « Lithologie » avec les objets de la table « Mailles ».

Exemple2 : Select lithologie.Code, Mailles.z, proportionoverlap(lithologie.obj, Mailles.obj) from lithologie, Mailles where lithologie.Obj Intersects Mailles.Obj

Même chose mais pour le pourcentage d'intersection (syntaxe 2)

Exemple3 : Select lithologie.Code, Mailles.z, ((ProportionOverlap(lithologie.obj, ailles.obj))*100)/ area(Mailles.obj, "sq km") "Pourcentage" from lithologie, Mailles where lithologie.Obj Intersects Mailles.Obj order by Pourcentage into Selection

Retourne le pourcentage d'intersection entre chaque objet de la table « Lithologie » et les objets de la table « Mailles »

Exemple4 : Select lithologie.Code, Mailles.z, round (((ProportionOverlap(lithologie.obj, Mailles.obj))*100)/area(Mailles.obj, "sq km"), 0.01) "Pourcentage" from lithologie, Mailles where lithologie.Obj Intersects Mailles.Obj order by Pourcentage asc into Selection

Retourne la même chose que précédemment mais avec des résultats arrondis.

Exemple 5 : Select lithologie.Code, Mailles.z from lithologie, Mailles where lithologie.Obj Intersects Mailles.Obj and (proportionoverlap(lithologie.obj, Mailles.obj)) = 1

Sélectionne tous les objets de la table « Lithologie » entièrement compris (proportion de recouvrement = 1) dans un objet de la table « Mailles ».

3 – 2 Requêtes retournant des informations linéaires sur les objets.

Objectifs : Trouver la longueur des objets linéaires

Syntaxe : SELECT <liste_de_variables>, OBJECTLEN(OBJ, "Unité de distance")
FROM <Table>

Exemple1 : Select TOPONYMIE, ObjectLen(obj, "km") from hydro

Retourne la longueur en km de chaque objet de la table Hydro

Exemple2 : Select TOPONYMIE, sum(ObjectLen(obj, "km")) from hydro group by TOPONYMIE

Retourne la longueur cumulée des cours d'eau portant le même nom.

Objectifs : Trouver le périmètre des objets fermés

Syntaxe : SELECT <liste_de_variables>, PERIMETER(OBJ, "Unité de distance")
FROM <Table>

Exemple1 : Select TOPONYMIE, ObjectLen(obj, "km") from Lacs

Retourne le périmètre en km de chaque objet de la table Lacs

Exemple2 : Select TOPONYMIE, sum(Perimeter(obj, "km")) from Lacs group by TOPONYMIE

Retourne le périmètre cumulé des lacs portant le même nom.

4 – Requêtes utilisant des caractéristiques graphiques

4 – 1 Sélections basées sur les attributs graphiques des objets

Objectifs : sélectionner les objets en fonction de leurs attributs graphiques (couleur par ex)

Syntaxe : SELECT {liste_expressions|*} FROM <Table> WHERE
STYLEATTR(OBJECTINFO(OBJ, <Argument 1>), <Argument 2>) <OPÉR><RÉF>

<Argument 1> correspond aux arguments de la fonction Objectinfo() :

Dans les fenêtres MapBasic ou SQL, utiliser le code numérique :

OBJ_INFO_PEN	2	retourne le n° de style de ligne
OBJ_INFO_SYMBOL	2	retourne le n° de symbole
OBJ_INFO_TEXTFONT	2	retourne la police courante
OBJ_INFO_BRUSH	3	retourne le n° de trame courante

<Argument 2> correspond aux arguments de la fonction Styleattr() . Noter que tous les codes n'existent pas pour tous les styles.

Dans les fenêtres MapBasic ou SQL, utiliser le code numérique :

PEN_WIDTH	1	retourne l'épaisseur d'une ligne
PEN_PATTERN	2	retourne le style de ligne
PEN_COLOR	4	retourne la couleur de la ligne
BRUSH_PATTERN	1	retourne le style de trame
BRUSH_FORECOLOR	2	retourne la couleur d'avant plan
BRUSH_BACKCOLOR	3	retourne la couleur d'arrière plan
FONT_NAME	1	retourne le nom de la police
FONT_STYLE	2	retourne le style de police (1 = gras, etc…)
FONT_POINTSIZE	3	retourne la taille de la police
FONT_FORECOLOR	4	retourne la couleur d'avant plan de la police
FONT_BACKCOLOR	5	retourne la couleur d'arrière plan de la police
SYMBOL_CODE	1	retourne le code du symbole utilisé
SYMBOL_COLOR	2	retourne la couleur du symbole
SYMBOL_POINTSIZE	3	retourne la taille des symboles ponctuels

SYMBOL_ANGLE 4 retourne l'angle du symbole
SYMBOL_FONT_NAME 5 retourne le nom du symbole
SYMBOL_FONT_STYLE 6 retourne le style du symbole
SYMBOL_KIND 7 retourne le type de symbole (1=vecteur, 2=police, 3=personnalisé)
SYMBOL_CUSTOM_NAME 8 retourne le nom du symbole personnalisé
SYMBOL_CUSTOM_STYLE 9 retourne le style du symbole personnalisé

<OPÉR> < , <= , = , >= , > , <> , LIKE

<RÉF> valeur de référence appropriée. Attention à l'orthographe, surtout aux espaces

Exemple : Select * from forage where StyleAttr(ObjectInfo(obj, 2), 2) = 16711680

*16711680 est le code interne MapInfo pour la couleur rouge traduite à partir du code RGB en suivant la formule suivante : (red * 65536) + (green * 256) + blue*

4 – 2 Requêtes sur les types d'objets

Objectifs : sélectionner des objets en fonction de leur type (points, polygones , polylignes, etc…)

Syntaxe1: SELECT * FROM <Table> WHERE STR$(OBJ) = "type_en_caractères"

Syntaxe2: SELECT * FROM <Table> WHERE STR$(Ojectinfo(obj,1) = "type_ en_numér."

Syntaxe3: SELECT * FROM <Table> WHERE INT(Ojectinfo(obj,1) = type_ en_numérique

types d'objet (numérique et caractères)

1 Arc	2 Ellipse	3 Line
4 Polyline	5 Point	7 Region
8 Rectangle	9 Rounded Rectangle	10 Text

Exemple1 : Select * from lithologie where str$(obj) = "line"
Exemple1a : Select * from lithologie where str$(Ojectinfo(obj,1)) = "4"
Exemple1b : Select * from lithologie where int(Ojectinfo(obj,1)) = 4

Les exemples ci dessus, tous équivalents, permettent de sélectionner tous les objets de type ligne. Très intéressant quand on fait de la correction sur les couvertures ou pour utiliser certaines fonctions qui ne marchent que sur des types donnés. (voir application dans la section présentant les principes de la sous-sélection)

Exemple2 : Select count(*), str$(obj) "Type" from lithologie group by Type

Cet exemple permet de compter le nombre d'objets différents par type d'objet présent

4 – 3 Requêtes retournant des informations sur la géographie des objets

<u>Objectifs</u> : Retourne les caractéristiques géographiques des objets d'une couverture

<u>Syntaxe</u> : SELECT OBJECTGEOGRAPHY (OBJ, <argument>) FROM <Table>.

<Argument> un argument de la fonction Objectgeography() :

Dans les fenêtres MapBasic ou SQL, utiliser le code numérique :

OBJ_GEO_MINX	1 coordonnées xmin du rectangle englobant
OBJ_GEO_LINEBEGX	1 coordonnées x du nœud de départ
OBJ_GEO_POINTX	1 coordonnées x du point
OBJ_GEO_MINY	2 coordonnées ymin du rectangle englobant
OBJ_GEO_LINEBEGY	2 coordonnées y du nœud de départ
OBJ_GEO_POINTY	2 coordonnées y du point
OBJ_GEO_MAXX	3 coordonnées xmax du rectangle englobant
OBJ_GEO_LINEENDX	3 coordonnées y du nœud de fin
OBJ_GEO_MAXY	4 coordonnées ymax du rectangle englobant
OBJ_GEO_LINEENDY	4 coordonnées y du nœud de fin
OBJ_GEO_ARCBEGANGLE	5 angle de départ d'un arc de cercle
OBJ_GEO_TEXTLINEX	5 coordonnées xmin d'un label de type texte
OBJ_GEO_ROUNDRADIUS	5 diamètre du cercle d'un rectangle arrondi
OBJ_GEO_ARCENDANGLE	6 angle de fin dun arc de cercle
OBJ_GEO_TEXTLINEY	6 coordonnées ymin dun label de type texte

<u>Syntaxe</u> : SELECT OBJECTINFO (OBJ, <argument>) FROM <Table>.

<Argument> un argument de la fonction ObjectInfo() :

Dans les fenêtres MapBasic ou SQL, utiliser le code numérique :

OBJ_INFO_NPNTS	20 retourne le nombre de nœuds d'un objet
OBJ_INFO_SMOOTH	4 retourne vrai si une polyligne est lissée
OBJ_INFO_NPOLYGONS	21 retourne le nombre de polygones d'une région ou de sections d'une polyligne
OBJ_INFO_NPOLYGONS +*N*	21 + N retourne le nombre de nœuds du Nième polygone d'une région, ou section d'une polyligne

<u>Exemple1</u> : Select objectgeography(obj, 1) "xmin", objectgeography(obj, 2) "ymin", objectgeography(obj, 3) "xmax", objectgeography(obj, 4) "ymax" from Lithologie

Permet d'afficher les coordonnées du rectangle englobant de chacun des objets de la table lithologie

<u>Exemple2</u> : Select objectinfo(obj, 20) "Nbr de points", objectinfo(obj, 21) "Nbr de multipolygones" from Lithologie

Permet d'afficher le nombre de nœuds de chaque objet ainsi que le nombre de polygones associé à chaque objet.

5 – Travailler avec des objets et données textes

5 – 1 Sélection d'objets textes en fonction de certaines de leurs caractéristiques

On peut utiliser les fonctions ObjectInfo() et ObjectGeography() pour atteindre d'autres caractéristiques que les éléments de style (police, couleurs, ...)

Fonction ObjectInfo(OBJ,<argument>)

<argument> (code numérique seulement)

OBJ_INFO_TEXTSTRING	3 retourne une chaîne de caractères
OBJ_INFO_TEXTSPACING	4 retourne l'espacement du texte (1, 1.5 ou 2)
OBJ_INFO_TEXTJUSTIFY	5 retourne le code de justification du texte (0 = G, 1 = C, 2 = D)
OBJ_INFO_TEXTARROW	6 retourne le code d'existence et de type de flèche associée à un texte : 0 pas de ligne, 1 ligne simple, 2 flèche)

Fonction ObjectGeography(OBJ,<argument>)

<argument> (code numérique seulement)

OBJ_GEO_TEXTANGLE	7 angle du texte

Trouver les enregistrements dont le texte contient une certaine chaîne.

Syntaxe : SELECT * FROM <table> WHERE OBJECTINFO(OBJ,3) LIKE "référence"

Exemple1 : select * from matable where objectinfo(obj,3) like "%puits%"

Trouve tous les objets textes dont la chaîne contient PUITS dans n'importe quelle position

Trouver les enregistrements dont le texte n'est pas horizontal.

Syntaxe : SELECT * FROM <table> WHERE OBJECTGEOGRAPHY(OBJ,7) <> 0

Exemple2 : select * from matable where objectgeography(obj,7) <> 0

Trouve tous les objets textes dont le texte fait un angle avec l'horizontale.

5 – 2 Concanétation / dé-concanétation de chaîne de caractères

Une table de renseignements sur les forages contient les variables Prénom (pour le prénom du propriétaire) et Nom (pour le son nom de famille). Si vous souhaitez que la table des résultats indique le nom complet de chaque propriétaire, vous pouvez rajouter une colonne dérivée.

Exemple 1 : Select Prénom + " " + Nom "Propriétaire" from Forages

Retourne le nom complet du propriétaire par concaténation des deux champs.

Il peut être nécessaire d'extraire des caractères particuliers dune chaîne de caractères. Ainsi, par exemple, les forages géologiques sont répertoriés sous le code BSS. Ce code (ex. 00041X0032) est composé de la manière suivante : les 5 premiers chiffres font référence à la carte géologique au 50.000 où se trouve le forage ; la lettre est un code de reconnaissance interne; les derniers chiffres correspondent au numéro d'enregistrement du forage à la BSS.

Si on veut que la table des résultats indique seulement le numéro de la carte au 50.000, on peut inclure une colonne dérivée dans le champ <liste_expressions>.

<u>Exemple 2</u> :   Select left$(BSS,5) « Numero carte 50000 » from forages

> *Retourne dans la colonne Numero carte 50000 les 5 caractères de gauche de la chaîne de caractères du champ « BSS ».*

5 – 3 Sélection des cas dont un champ texte contient une chaîne donnée

<u>Syntaxe :</u>   SELECT * FROM <table> WHERE INSTR(1,<col_texte>,"chaîne_donnée ") > 0

<u>Exemple :</u>   select * from ma_table where instr(1,nom_rue, " ")>0

> *Sélectionne tous les cas de ma_table où il y a au moins un espace dans la colonne nom_rue.*
>
> *À comparer avec*
> Select * from ma_table where nom_rue like "% %"

5 – 4 Extraction du dernier mot dune phrase

Objectif :   obtenir une colonne contenant le dernier mot dune phrase (ex. dune adresse)

Ceci pourrait se faire par un UPDATE de la colonne appropriée avec

```
Right$(Col_Texte,Len(Col_Texte)-InStr(InStr(1,Col_Text," ")+1,Col_Text," "))
```

seulement sil y avait toujours 2 noms dans la colonne texte. Comme il faut s'attendre à en avoir un nombre indéfini à chaque cas, on peut utiliser la procédure suivante qu fait appel à un SELECTt dans une procédure itérative de réduction de la chaîne :

1 – ajouter 2 colonnes (Long Smallint, Droite Char(aussi large que l'originale))

2 – mettre à jour ces colonnes

```
update <table>  set droite = <col_texte>
update <table>  set long = len(droite)
```

3 –

```
select * from <table> where instr(1, droite, " ")> 0 into SEL
```

4 –

```
update SEL set droite=right$( droite, long - instr(1, droite, " "))
update SEL set long = len(droite)
select * from SEL where instr(1, droite, " ")> 0 into SEL
```

> *Répéter 4 en bloc jusqu'à ce qu'il ny ait plus d'objets dans sel. Utiliser naturellement la fenêtre MapBasic pour au moins 4 -.*

6 – Groupement des résultats par colonnes.

Supposons que vous ayez une table sur les commandes clients. Chaque ligne de la table représente une commande. Une colonne de la table contient le nom du vendeur (VENDEUR) ayant réalisé la commande, une autre contient le nom du client (CLIENT) tandis qu'une troisième contient le montant de la commande (AMOUNT).

Pour chaque vendeur, supposons que vous vouliez savoir :

- *le nombre de commande prises,*
- *le montant moyen de ses commandes,*
- *la valeur totale des commandes prises*

Syntaxe : SELECT <liste_expressions> FROM <Table> [WHERE <groupe_conditions>]
GROUP BY <liste_colonnes>

Exemple 1 : Select Vendeur, count(*), Avg(AMOUNT), Sum(AMOUNT) from Commandes
Group By Vendeur

Remarquez le champ Grouper par colonnes et les trois opérateurs d'agrégation dans la zone Colonnes. MapInfo procède comme suit :

1. Il recherche toutes les lignes concernant un vendeur donné.
2. Il compte le nombre de lignes : Count(*)
3. Il calcule la valeur totale des commandes du vendeur : Sum(AMOUNT).
4. Il calcule la valeur moyenne des commandes du vendeur : Avg(AMOUNT)

MapInfo fait de même pour chaque vendeur. Il en résulte une table n'ayant qu'une ligne pour chaque vendeur. Les opérateurs d'agrégation (Count, Avg et Sum) calculent le sous-total des valeurs de toutes les lignes ayant une même valeur par Vendeur.

Exemple 2 : Select Client, Count(*), Avg(AMOUNT), Sum(AMOUNT) from Commandes
Group by Client

Il sagit à peu près de la même requête que précédemment, sauf que nous souhaitons maintenant effectuer un regroupement par client et non plus par vendeur. La sélection SQL effectuera les mêmes calculs que ci-dessus pour chaque client.

Exemple 3 : Select Vendeur, Client, count(*), average(AMOUNT), sum(AMOUNT) from
Commandes group by Vendeur, Client

Deux noms de colonnes ont été indiqués dans la zone Colonnes. Dans ce cas, MapInfo groupera les lignes d'abord par vendeur puis par client. La table de résultat de cette requête contiendra une ligne pour chaque combinaison client/vendeur. Si un client a passé des commandes par l'intermédiaire de plusieurs vendeurs, une ligne résumera les commandes passées avec chaque vendeur. Les lignes seront d'abord groupées par vendeur, puis pour chacun deux, par client.

7 – Triage des résultats par ordre croissant ou décroissant

Par défaut, MapInfo trie une table par ordre croissant. Si vous effectuez le tri par ordre alphabétique, cela signifie que les A apparaîtront avant les B, etc. ... Si vous triez par valeurs numériques, l'ordre croissant signifie que les petits nombres apparaîtront avant les grands nombres.

Si vous voulez trier par ordre décroissant, cela signifie que les grands nombres apparaissent avant les petits, il vous faut placer le mot DECR à la suite du nom de la colonne dans liste_colonnes du « Group by ».

Syntaxe : SELECT <liste_expressions> FROM <Table> WHERE <groupe_conditions>
GROUP BY <liste_colonnes> ORDER BY <liste_colonnes> [DESC]

Exemple 1 : Select * from World order by Population desc

entraîne le tri de la table à partir de la colonne Population, en ordre décroissant.

Exemple 2 : Select * from forage order by INAT , BSS Ddesc

Retourne une table où les enregistrements sont triés par numéro INAT croissant et par BSS décroissant.

Lorsque MapInfo effectue un tri sur plusieurs niveaux, chaque niveau du tri dispose de ses propres paramètres croissant/décroissant.

8 – Exemples hors catégories

8 – 1 Assignation de la plus grande fréquence

Une table de régions (Polys), une de points (Points) contenant en particulier une colonne (Name) de noms (plusieurs points peuvent avoir le même nom). On veut assigner à une nouvelle colonne de Polys le nom le plus fréquent dans chaque région.

On procède dabord à une double sélection :

```
SELECT Polys.ID, Points.Name, Count(*) "Cnt" INTO CountTab
    FROM Polys, Points
    WHERE Polys.Obj Contains Points.Obj
    GROUP BY Polys.ID, Points.Name
```

Produit une table temporaire “CountTab” ayant les colonnes ID, Name, Cnt et une rangée pour chaque combinaison existante de ID de Polys et Name de Points

```
SELECT ID, Name "Dominant", Max(Cnt) "DomCount"
    INTO DomTab
    FROM CountTab
    GROUP BY ID
```

Produit une table temporaire "DomCount" ayant les colonnes ID, Dominant, DomCount et une rangée pour chaque ID

La mise à jour de la table Polys doit se faire par l'intermédiaire de « Update » (On suppose que la colonne PolyName qui recevra le nom dominant existe déjà)

```
UPDATE Polys SET Polys.PolyName = DomTab.Dominant
    WHERE Polys.ID = DomTab.ID
```