Vk

Zeste de savoir

Les listes de données Excel en VBA

10 juin 2023

Table des matières

Introduction

Les listes de données, aussi connues sous le nom de tableaux, sont une fonctionnalité précieuse de Microsoft Excel.

Elles permettent de structurer les informations plus facilement (report automatique des formats et des formules pour les nouvelles lignes, personnalisation de l'apparence poussée, …) ainsi que de naviguer dans celles-ci plus rapidement (tri, filtres, totaux, formules, …).

Ce qui est utilisable en Excel l'est généralement en VBA, et nous pouvons donc tirer parti de ces possibilités dans nos macros. De plus, les `ListObjects` sont plus simples à manier que des plages dynamiques normales, car nous n'avons pas à déterminer leur taille ou l'emplacement de la ligne à insérer par exemple.

À travers ce billet nous allons voir comment se composent ces listes de données et comment les utiliser en VBA.

C'est parti!

1. Création, accès et structure

Pour commencer, nous allons créer un tableau, y accéder en VBA et voir comment il se compose.

1.1. Création

Nous partons de la plage normale suivante:

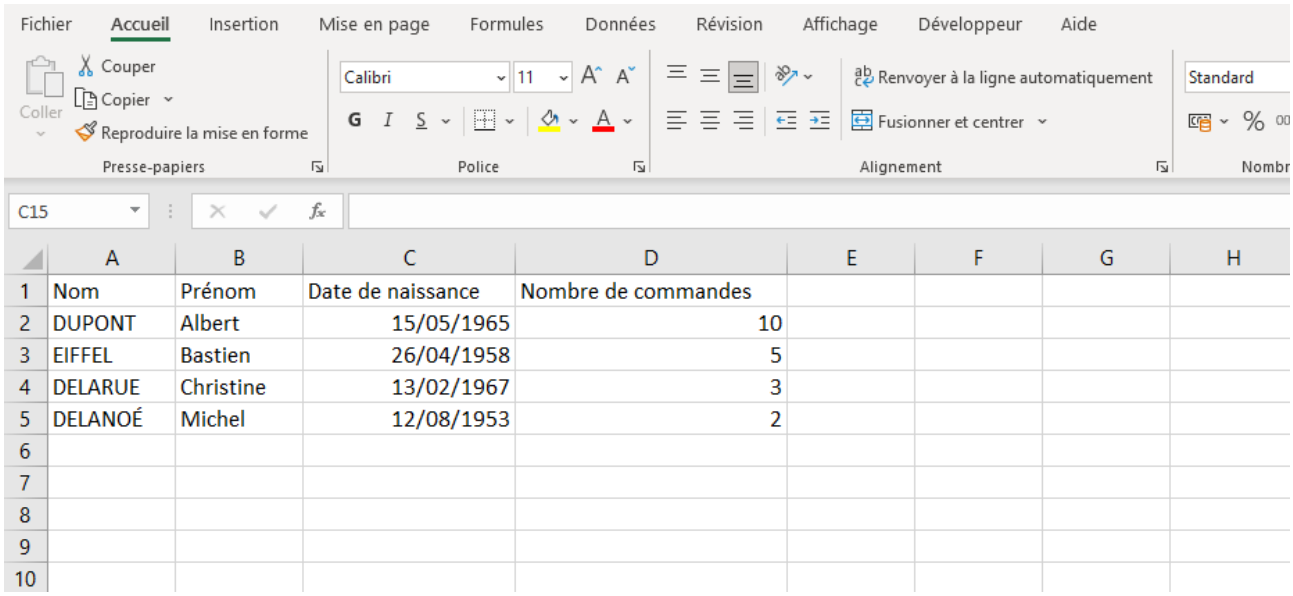


FIGURE 1.1. – Plage normale de départ

Pour créer notre liste de données, nous sélectionnons cette plage puis cliquons sur l'option "Tableau" du ruban dans le menu "Insertion". Une fenêtre de confirmation s'ouvre alors dans laquelle nous cochons avoir des en-têtes et validons.

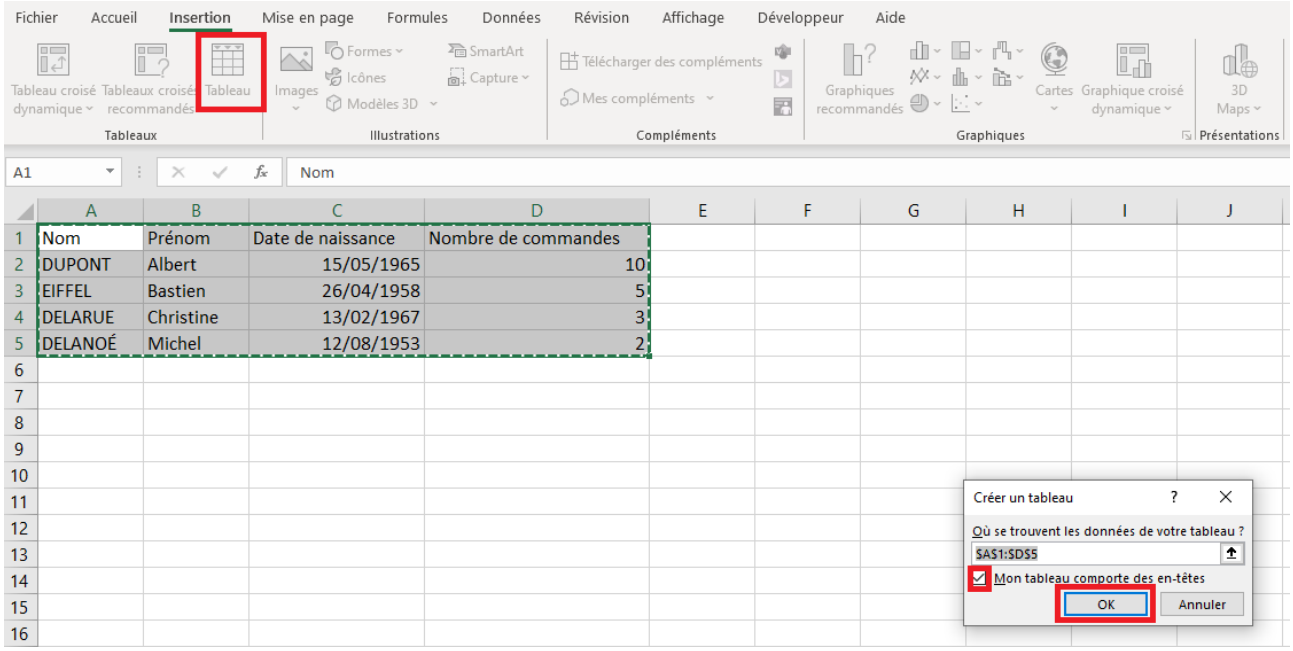


FIGURE 1.2. – Insertion d'une liste de données

Nous achevons cette étape en donnant un nom parlant et en ajoutant la ligne des totaux avec la coche, comme illustré ci-dessous:

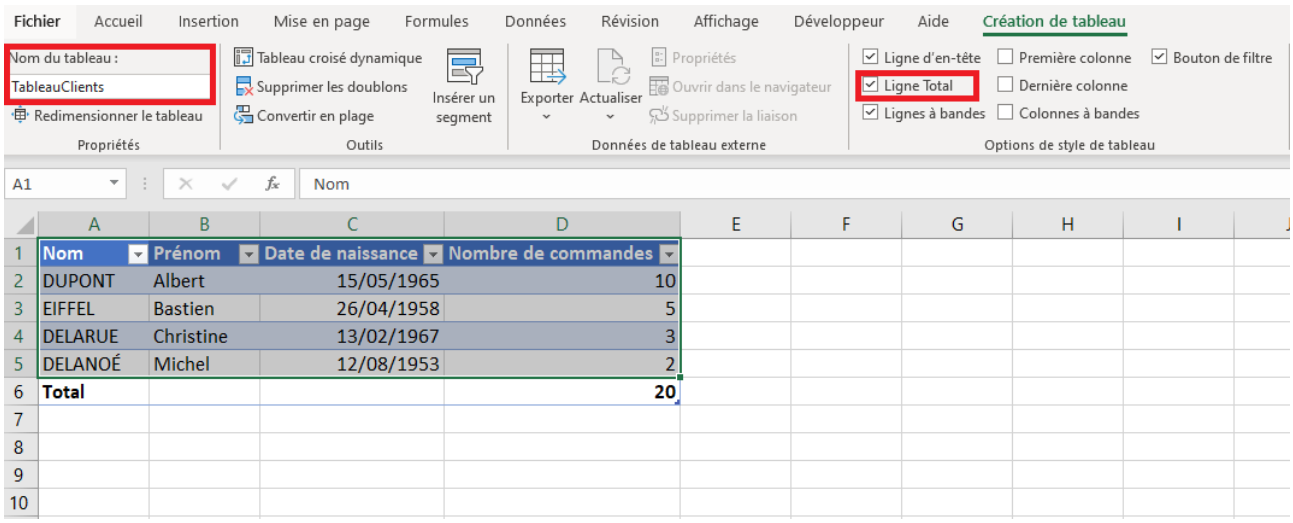


FIGURE 1.3. – Paramétrage du tableau

Maintenant que notre tableau est créé, voyons comment il se compose.

1.2. Accès

Mais avant cela, nous allons y accéder en VBA. Pour ce faire, les objets feuilles possède une propriété `ListObjects` qui est une collection de tableaux.

```vba
Option Explicit

Public Const FEUILLE_CLIENTS_NOM As String = "Feuil1"
Public Const LO_CLIENTS_NOM As String = "TableauClients"

Public Sub Exemple_Acces_ListObject()
    Dim loClients As ListObject
    Set loClients =
        Sheets(FEUILLE_CLIENTS_NOM).ListObjects(LO_CLIENTS_NOM)

    Debug.Print (loClients.Name) ' TableauClients
End Sub
```

Le nom du tableau s'affiche bien lorsque nous exécutons ce code.

1.3. Structure

Comprendre comment un tableau est construit nous permet de savoir quelles propriétés utiliser pour faire les opérations voulues (ajouter une ligne dans le corps, masquer la ligne des totaux, …). Il y a deux façons de considérer la structure d'un tableau.

1.3.1. En-tête, Corps et Total

Premièrement, une liste de données est faite de trois blocs principaux: la ligne d'en-têtes, le corps et la ligne des totaux.

Ces blocs correspondent respectivement aux propriétés `HeaderRowRange`, `DataBodyRange` et `TotalsRowRange` comme montré ci-dessous:

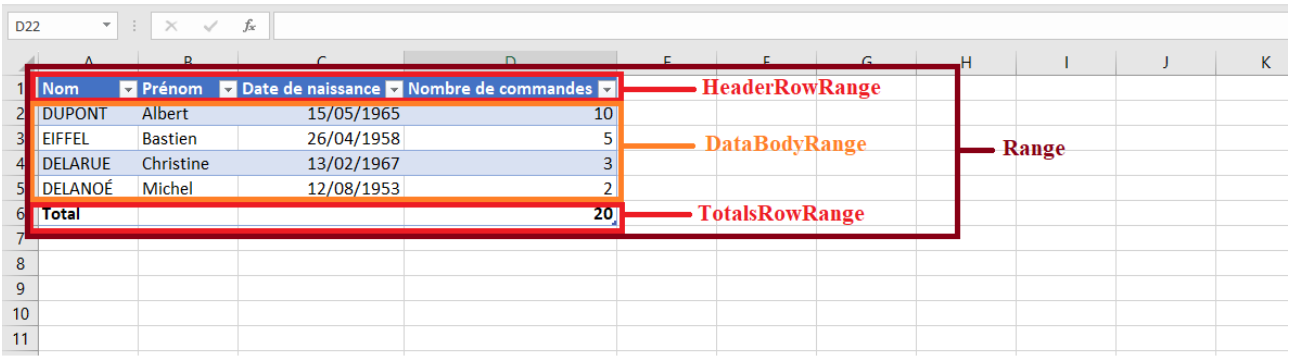


FIGURE 1.4. – Différents blocs constituants le tableau

L'ensemble fait partie de la plage globale accessible via la propriété `Range`.

```vba
Public Sub Exemple_Blocs_ListObject()
    Dim loClients As ListObject
    Set loClients =
        Sheets(FEUILLE_CLIENTS_NOM).ListObjects(LO_CLIENTS_NOM)

    Debug.Print (loClients.Range.Address) ' $A$1:$D$6
    Debug.Print (loClients.HeaderRowRange.Address) ' $A$1:$D$1
    Debug.Print (loClients.DataBodyRange.Address) ' $A$2:$D$5
    Debug.Print (loClients.TotalsRowRange.Address) ' $A$6:$D$6
End Sub
```

1.3.2. Lignes et colonnes

Deuxièmement, une liste de données est un ensemble de lignes et de colonnes.

Ces lignes correspondent à la propriété `ListRows` qui est une collection de `ListRow`. Comme vous pouvez le voir avec l'image ci-dessous, seules les lignes du corps du tableau font partie de la collection `ListRows`.

Ces colonnes quant à elles correspondent à la propriété `ListColumns` qui est une collection de `ListColumn`. Comme vous pouvez le voir avec l'image ci-dessous, la colonne entière est prise en compte.

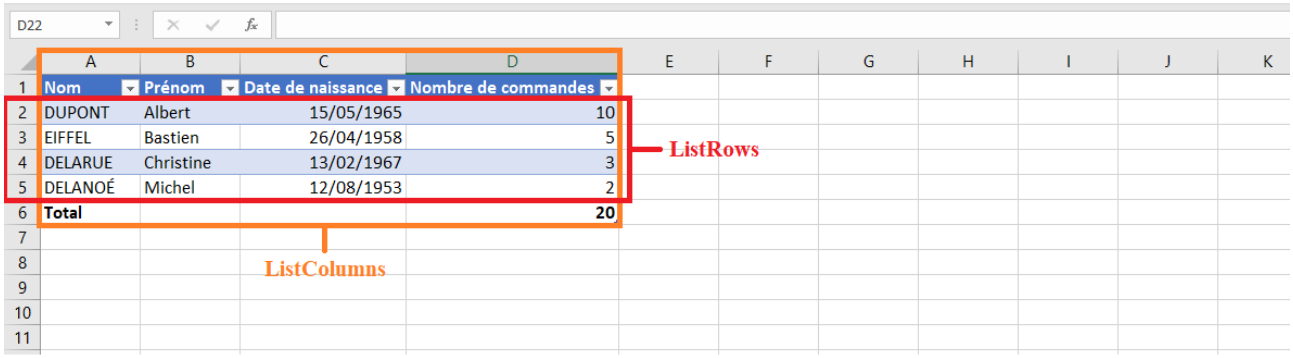


FIGURE 1.5. – Les lignes et les colonnes du tableau

Le code suivant nous confirme cela:

```vba
Public Sub Exemple_Lignes_et_colonnes_ListObject()
    Dim loClients As ListObject
    Set loClients =
        Sheets(FEUILLE_CLIENTS_NOM).ListObjects(LO_CLIENTS_NOM)

    Dim lcColonneClient As ListColumn
    For Each lcColonneClient In loClients.ListColumns
        Debug.Print (lcColonneClient.Range.Address)
        ' $A$1:$A$6
        ' $B$1:$B$6
        ' $C$1:$C$6
        ' $D$1:$D$6
    Next lcColonneClient

    Dim lrLigneClient As ListRow
    For Each lrLigneClient In loClients.ListRows
        Debug.Print (lrLigneClient.Range.Address)
        ' $A$2:$D$2
        ' $A$3:$D$3
        ' $A$4:$D$4
        ' $A$5:$D$5
    Next lrLigneClient
End Sub
```

1.3.2.1. Colonnes Même si un objet `ListColumn` englobe toute la colonne, nous pouvons obtenir les parties qui nous intéressent avec les propriétés adéquates: `Name` pour la valeur de l'en-tête, `DataBodyRange` pour la plage du corps, et `Total` pour la valeur de la ligne de totale:

```vba
Public Sub Exemple_Colonne_ListObject()
    Dim loClients As ListObject
    Set loClients =
        Sheets(FEUILLE_CLIENTS_NOM).ListObjects(LO_CLIENTS_NOM)

    With loClients.ListColumns("Nombre de commandes")
        Debug.Print (.Name) ' Nombre de commandes
        Debug.Print (.DataBodyRange.Address) ' $D$2:$D$5
        Debug.Print (.Total) ' 20
    End With
End Sub
```

Pendant cette section, nous avons en appris plus sur la structure des liste de données.

2. Utilisation

Notre liste de données étant définie, il est temps d'apprendre à s'en servir.

Certaines opérations sont accessibles de manière haut niveau, c'est-à-dire que l'objet `ListObject` expose des propriétés ou des méthodes prêtes à l'usage pour faire directement ce que l'on souhaite.

D'autres en revanche nous demandent un peu plus de réflexion et de passer par des propriétés intermédiaires (comme celles étudiées dans la section précédente) voire de réaliser des opérations supplémentaires (tester que le corps du tableau n'est pas vide par exemple). Pour ces dernières, il sera préférable d'élaborer ses propres procédures hauts niveaux (procédure pour vider un objet liste de données par exemple) à l'avenir.

2.1. Lire, écrire et supprimer des données

Le but d'avoir une liste de données est de pouvoir manipuler des données.

Nous allons nous concentrer sur le corps du tableau, car c'est lui qui contient l'essentiel.

2.1.1. Lecture des données

Bien souvent, nous voudrons lire des informations du `ListObject`.

2.1.1.1. Lecture d'une ligne Si nous voulons lire une ligne en particulier, nous savons qu'il existe la collection `ListRows` vue plus tôt et c'est grâce à celle-ci que nous allons faire cela, en lui spécifiant un index.

```vba
Public Sub Exemple_Lire_Ligne()
    Dim loClients As ListObject
    Set loClients =
        Sheets(FEUILLE_CLIENTS_NOM).ListObjects(LO_CLIENTS_NOM)

    Dim lIndexDerniereLigne As Long
    lIndexDerniereLigne = loClients.ListRows.Count ' 4

    If lIndexDerniereLigne > 0 Then
        ' Lecture dernière ligne
        Dim vContenuLigne As Variant
        vContenuLigne =
            loClients.ListRows(lIndexDerniereLigne).Range.Value

        Debug.Print (vContenuLigne(1, 1) & " " & vContenuLigne(1,
            2)) ' DELANOÉ Michel
    Else
        Debug.Print ("Aucune Ligne")
    End If
End Sub
```

2.1.1.2. Lecture d'une colonne Plus rarement, nous voudrons accéder aux données d'une colonne. Comme vu précédemment, nous utiliserons la collection `ListColumns` pour cela, avec un index ou un indice.

```vba
Public Sub Exemple_Lire_Colonne()
    Dim loClients As ListObject
    Set loClients =
        Sheets(FEUILLE_CLIENTS_NOM).ListObjects(LO_CLIENTS_NOM)

    Dim lTotalCommandes As Long

    If loClients.ListRows.Count Then
        lTotalCommandes = WorksheetFunction.Sum(loClients.ListColu
            mns("Nombre de commandes").DataBodyRange)
    End If

    Debug.Print ("Total de commandes : " & lTotalCommandes) '
        Total de commandes : 20
End Sub
```

2.1.1.3. Lecture de l'intégralité du corps Si maintenant je vous demande de lire l'intégralité du tableau, vous choisiriez sans doute de parcourir celui-ci ligne par ligne ou colonne par colonne et ça fonctionnerait très bien. Cependant, il existe une autre solution possible et qui est plus optimisée: celle de lire l'intégralité du tableau d'un coup!

Pour cela, nous allons utiliser la propriété `DataBodyRange`.

```
Public Sub Exemple_LireTout_ListObject()
    Dim loClients As ListObject
    Set loClients =
        Sheets(FEUILLE_CLIENTS_NOM).ListObjects(LO_CLIENTS_NOM)

    If loClients.DataBodyRange Is Nothing Then
        Debug.Print ("Aucun contenu !")
        Exit Sub
    End If

    Dim vContenuTableau As Variant
    vContenuTableau = loClients.DataBodyRange.Value

    Dim lIndexLigne As Long
    For lIndexLigne = LBound(vContenuTableau, 1) To
        UBound(vContenuTableau, 1)
        Debug.Print (vContenuTableau(lIndexLigne, 1) & " " &
            vContenuTableau(lIndexLigne, 2))
        ' DUPONT Albert
        ' EIFFEL Bastien
        ' DELARUE Christine
        ' DELANOÉ Michel
    Next lIndexLigne
End Sub
```

2.1.2. Écriture des données

Ecrire des données, c'est soit en modifier, soit en ajouter.

2.1.2.1. Modification des données

Pour modifier des données en particulier, nous pouvons accéder aux différentes cellules et appliquer nos changements.

Si les modifications sont nombreuses, il peut être plus optimisé de charger le contenu dans un tableau en mémoire pour le modifier et ensuite l'écrire d'un coup à la place de l'ancien.

2.1.2.2. Ajout d'une ligne

Pour ajouter une ligne, nous allons ajouter un élément `ListRow` à la propriété `ListRows` à travers la méthode `Add`. Cette méthode peut prendre l'index de la position où insérer la ligne à créer, ce qui décale les autres lignes automatiquement.

```
Public Sub Exemple_AjouterLigne_ListObject()
    Dim loClients As ListObject
```

```
    Set loClients =
        Sheets(FEUILLE_CLIENTS_NOM).ListObjects(LO_CLIENTS_NOM)

    Dim vContenuLigne As Variant
    vContenuLigne = Array("ANDERSON", "Simon",
        CDate("25/08/1985"), 4)

    Dim lrLigne As ListRow
    Set lrLigne = loClients.ListRows.Add
    lrLigne.Range = vContenuLigne
End Sub
```

2.1.2.3. Ajout d'une colonne Pour ajouter une colonne, nous utiliserons ici aussi la méthode `Add` cette fois ci-appliquée à la collection `ListColumns`.

```
Public Sub Exemple_AjouterColonne_ListObject()
    Dim loClients As ListObject
    Set loClients =
        Sheets(FEUILLE_CLIENTS_NOM).ListObjects(LO_CLIENTS_NOM)

    Dim lcColonne As ListColumn
    Set lcColonne = loClients.ListColumns.Add(3)

    lcColonne.Name = "Nom complet"
    lcColonne.DataBodyRange.FormulaR1C1 =
        "=TEXTJOIN("" "", TRUE, [@Nom], [@[Prénom]])"
End Sub
```

C2 =JOINDRE.TEXTE(" "; VRAI;[@Nom]; [@Prénom])

	A	B	C	D	E
1	Nom	Prénom	Nom complet	Date de naissance	Nombre d
2	DUPONT	Albert	DUPONT Albert	15/05/1965	10
3	EIFFEL	Bastien	EIFFEL Bastien	26/04/1958	5
4	DELARUE	Christine	DELARUE Christine	13/02/1967	3
5	DELANOÉ	Michel	DELANOÉ Michel	12/08/1953	2
6	ANDERSON	Simon	ANDERSON Simon	25/08/1985	4
7	Total				24

FIGURE 2.6. – Ajout d'une colonne

2.1.2.4. Ajout de plusieurs lignes Là encore, dans un but d'optimisation, il est souhaitable d'écrire les plus grandes plages de données possibles en une fois.

```
Public Sub Exemple_AjouterLignes_ListObject()
    Dim loClients As ListObject
    Set loClients =
        Sheets(FEUILLE_CLIENTS_NOM).ListObjects(LO_CLIENTS_NOM)

    Dim vContenuLignes(2, 4) As Variant
    vContenuLignes(0, 0) = "ANDERSON"
    vContenuLignes(0, 1) = "Simon"
    vContenuLignes(0, 2) = CDate("25/08/1985")
    vContenuLignes(0, 3) = 4
    vContenuLignes(1, 0) = "ZACHARIE"
    vContenuLignes(1, 1) = "Didier"
    vContenuLignes(1, 2) = CDate("25/08/1972")
    vContenuLignes(1, 3) = 0

    loClients.ListRows.Add.Range.Resize(UBound(vContenuLignes, 1),
        UBound(vContenuLignes, 2)).Value = vContenuLignes
End Sub
```

2.1.3. Suppression des données

Il faut parfois supprimer les données pour réinitialiser le tableau à un état de départ ou encore pour purger certaines données.

Par exemple, nous pouvons imaginer un tableau de commandes à préparer qui serait rempli chaque jour suite à une macro. Avant de le remplir, il faudrait alors supprimer les données de la veille.

De plus, nous pouvons aussi imaginer une macro qui supprimerait les clients inactifs de notre tableau en parcourant chaque ligne pour cela.

2.1.3.1. Suppression d'une ligne ou d'une colonne La méthode `Delete` appliquée à une ligne ou une colonne supprime celle-ci. Il faut alors indiquer l'indice ou la clef de l'élément à supprimer.

```
loClients.ListRows(loClients.ListRows.Count).Delete ' Suppression
    de la dernière ligne
loClients.ListColumns("Nom complet").Delete ' Suppression de la
    colonne 'Nom complet'
```

À noter que seul le contenu se trouvant à l'intérieur du tableau est supprimé. En reprenant notre code, si nous avions du contenu à côté de la dernière ligne ou superposé à la colonne 'Nom complet', il n'aurait pas été supprimé.

2.1.3.2. Suppression du contenu du corps Pour supprimer l'intégralité du corps du tableau, nous utiliserons la méthode `Delete` là-encore, cette-fois appliquée au `DataBodyRange`.

```
1 If Not loClients.DataBodyRange Is Nothing Then
2     loClients.DataBodyRange.Delete
3 End If
```

2.2. Filtrer et trier

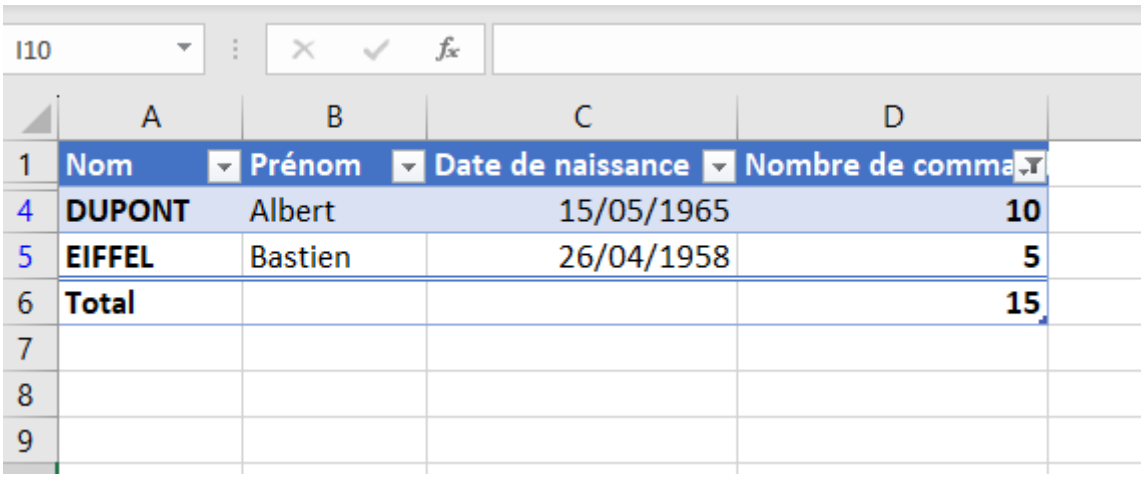
Microsoft Excel permet de filtrer, mais aussi de trier les données.

Pour le faire en VBA, j'utilise en général l'enregistreur de macro afin de récupérer la syntaxe.

2.2.1. Ajout et suppression filtres

2.2.1.1. Ajout filtre Ajouter un filtre se fait via la méthode `AutoFilter` d'une plage.

```
1 Public Sub Exemple_AjoutFiltre_ListObject()
2     Dim loClients As ListObject
3     Set loClients =
          Sheets(FEUILLE_CLIENTS_NOM).ListObjects(LO_CLIENTS_NOM)
4
5     loClients.Range.AutoFilter Field:=4, Criteria1:=">=5",
          Operator:=xlAnd
6 End Sub
```



	A	B	C	D
1	Nom	Prénom	Date de naissance	Nombre de comma
4	DUPONT	Albert	15/05/1965	10
5	EIFFEL	Bastien	26/04/1958	5
6	Total			15
7				
8				
9				

FIGURE 2.7. – Exemple tableau filtré

2.2.1.2. Suppression filtres La méthode `ShowAllData` appliquée à la propriété `Autofilter` enlève les filtres.

```
Public Sub Exemple_SuppressionFiltres_ListObject()
    Dim loClients As ListObject
    Set loClients =
        Sheets(FEUILLE_CLIENTS_NOM).ListObjects(LO_CLIENTS_NOM)

    loClients.AutoFilter.ShowAllData
End Sub
```

2.2.1.3. Ajout tri Nous pouvons ajouter un critère de tri `SortFields` de cette manière:

```
Public Sub Exemple_AjoutTri_ListObject()
    Dim loClients As ListObject
    Set loClients =
        Sheets(FEUILLE_CLIENTS_NOM).ListObjects(LO_CLIENTS_NOM)

    loClients.Sort.SortFields.Add2 Key:=Range( _
        "TableauClients[[#Headers],[#Data],[Nombre de commandes]]"
            ), SortOn:=
            _
        xlSortOnValues, Order:=xlDescending,
            DataOption:=xlSortNormal
End Sub
```

2.2.1.4. Suppression tris La méthode `Clear` sur les critères de tri `SortFields` les supprime.

```vba
Public Sub Exemple_SuppressionTri_ListObject()
    Dim loClients As ListObject
    Set loClients =
        Sheets(FEUILLE_CLIENTS_NOM).ListObjects(LO_CLIENTS_NOM)

    loClients.Sort.SortFields.Clear
End Sub
```

Nos filtres n'impactent pas notre parcours des données. En utilisant `Exemple_LireTout_ListObject`, nous pouvons remarquer que toutes les données sont lues, mais de façon triées. En fait, cela est dû que les lignes à ne pas afficher sont masquées. Pour récupérer uniquement les lignes visibles, nous pouvons utiliser `SpecialCells` : `loClients.DataBodyRange.SpecialCells(xlCellTypeVisible).Value`.

2.3. Mettre en forme

Plusieurs propriétés sont à la disposition du développeur pour interagir avec la mise en forme du tableau.

2.3.1. Style du du tableau

La propriété `TableStyle` nous permet de lire ou modifier le style du tableau.

```vba
Public Sub Exemple_Style_ListObject()
    Dim loClients As ListObject
    Set loClients =
        Sheets(FEUILLE_CLIENTS_NOM).ListObjects(LO_CLIENTS_NOM)

    With loClients
        Debug.Print (.TableStyle) ' TableStyleMedium2
        .TableStyle = "TableStyleMedium3"
    End With
End Sub
```

2.3.2. Mise en avant colonnes

Il est possible de mettre en avant visuellement la première et la dernière colonne. Cela peut être utile pour distinguer visuellement la donnée du reste (identifiant, total, …)

2.3.2.1. Première colonne La propriété de type booléenne `ShowTableStyleFirstColumn` nous permet de lire ou modifier la mise en avant de la première colonne.

```
1 loClients.ShowTableStyleFirstColumn = True
```

2.3.2.2. Dernière colonne La propriété de type booléenne `ShowTableStyleLastColumn` nous permet de lire ou modifier la mise en avant de la dernière colonne.

```
1 loClients.ShowTableStyleLastColumn = True
```

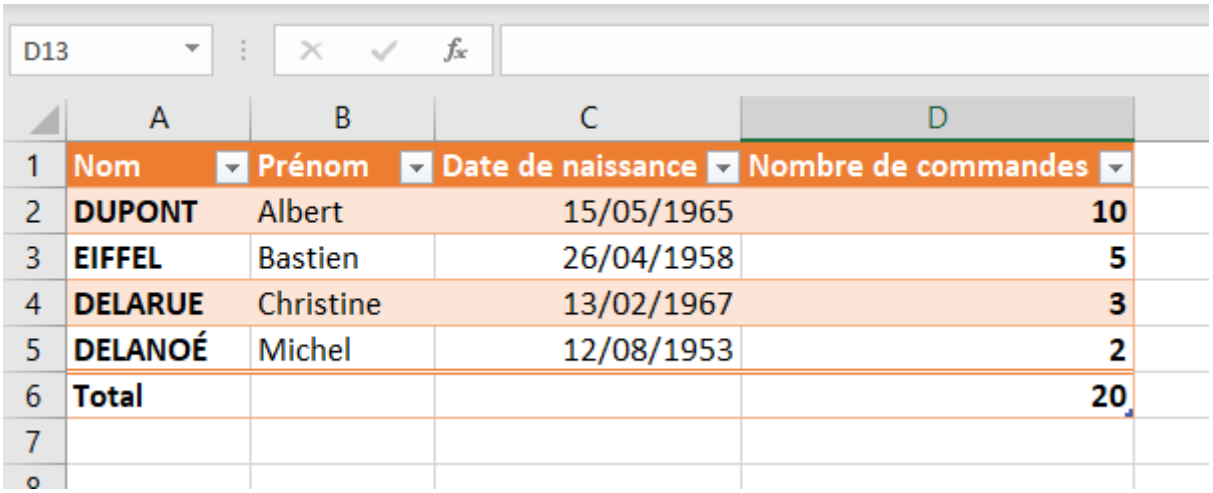


FIGURE 2.8. – Mise en avant première et dernière colonne

2.3.3. Alternance mise en forme

Par défaut, l'alternance de la mise en forme se fait par ligne du corps du tableau, mais nous pouvons aussi changer cela. Le but est d'avoir une distinction plus nette des données.

2.3.3.1. Lignes à bandes La propriété de type booléenne `ShowTableStyleRowStripes` nous permet de lire ou modifier l'alternance de mise en forme au niveau des lignes du corps du tableau.

```
1 loClients.ShowTableStyleRowStripes = False
```

2.3.3.2. Colonnes à bandes La propriété de type booléenne `ShowTableStyleColumnStripes` nous permet de lire ou modifier l'alternance de mise en forme au niveau des colonnes du corps du tableau.

```
1 loClients.ShowTableStyleColumnStripes = True
```

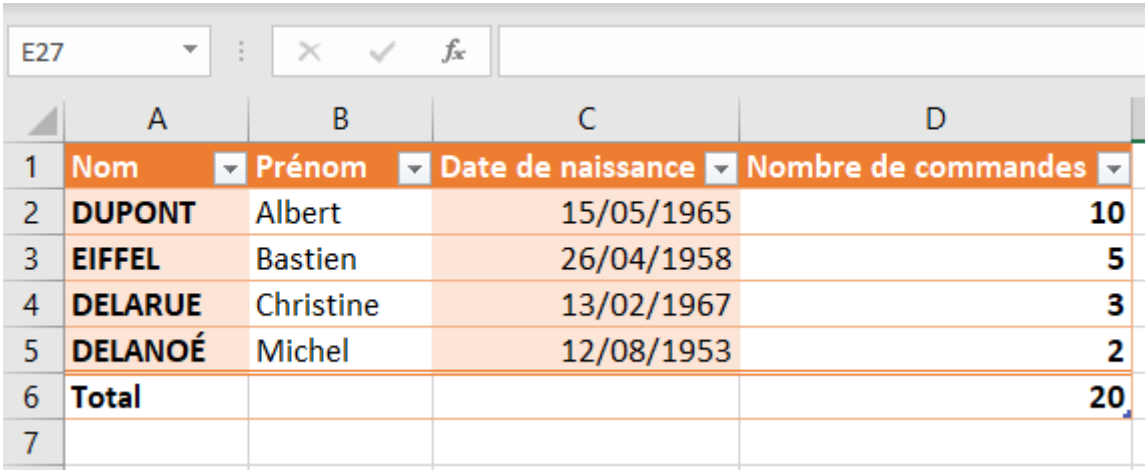


FIGURE 2.9. – Alternance bandes

Remarquons qu'il est possible de combiner les deux, l'un n'excluant pas l'autre. Toutefois, nous perdons en ergonomie et ça n'a guère de sens de le faire.

2.4. Masquer ou Afficher

Nous pouvons masquer ou afficher certaines parties de la liste de données en VBA.

2.4.1. Ligne d'entête

La propriété de type booléenne `ShowHeaders` nous permet de lire ou modifier la visibilité de la ligne des en-têtes.

```
1 loClients.ShowHeaders = Not loClients.ShowHeaders ' Toggle
      visibilité ligne d'en-tête
```

2.4.2. Ligne Total

La propriété de type booléenne `ShowTotals` nous permet de lire ou modifier la visibilité de la ligne des totaux.

```
1 loClients.ShowTotals = Not loClients.ShowTotals ' Toggle
      visibilité ligne Total
```

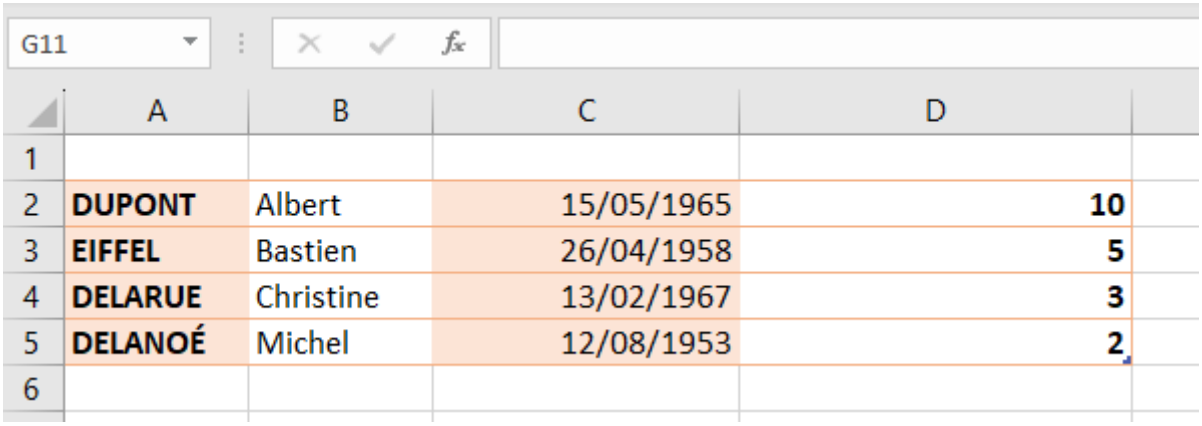


FIGURE 2.10. – Ligne d'en-tête et Total masquées

Comme nous l'avons vu, certaines opérations ne sont pas disponibles de façon haut niveau ce qui peut nous encourager à écrire nos propres procédures ou fonctions génériques. Par exemple, nous pourrions écrire une procédure supprimant le contenu d'un tableau qui ferait appel à une fonction afin de tester si le contenu de ce tableau est déjà vide.

Au fil de cette section, nous avons vu différentes façons d'utiliser des `ListObjects` dans nos macros.

Conclusion

C'est déjà la fin de ce billet.

Au cours de celui-ci, nous avons vu comment se présentaient les listes de données et comment les utiliser en VBA.

Si vous souhaitez faire des opérations non présentées dans ce billet, n'oubliez pas que la documentation et l'enregistreur de macros sont vos alliés (enfin méfiez-vous quand même de ce dernier… 😈)!

Dans le cas d'un programme avec beaucoup de données devant persister, les tableaux ne doivent pas se substituer à une base de données telles que Microsoft Access par exemple.

À bientôt!

Quelques ressources:

— La documentation de l'objet ListObject
— Ce tutoriel sur Excel et VBA