Vk

Geste de savoir

Les sources de données Excel avec VBA

lundi 26 août 2024

Table des matières

Introduction

Il arrive de devoir lire des données structurées que ce soit depuis des fichiers textes, des bases de données ou encore d'autres classeurs.

Nous pourrions alors concevoir un programme qui établirait une connexion vers cette source, lirait les données, effectuerait potentiellement des modifications (transformation, ajout de nouvelles colonnes, …), écrirait le résultat dans une feuille, puis fermerait la connexion. Cela fonctionnerait… mais il y a plus simple !

En effet, Excel met à disposition ce qui est appelé "source de données", nous permettant de faire la même chose en quelques clics (j'exagère à peine… 🍊).

Ce billet va donc présenter les sources de données Excel et leur usage en VBA.

1. Création

Dans les versions les plus récentes d'Excel, nous pouvons ajouter des sources de données à partir d'une multitude de types différents (texte, classeur, web, ODBC, base de données SQL Server, …).

Pour ce billet, nous nous baserons sur un fichier texte assez basique qui contient une ligne d'entête et des lignes de données :

Fichier coordonnees_points.txt

```
x;y
1;20
15;5
3;2
```

1.1. Création depuis Excel

Pour ajouter une source de données depuis Excel, il faut se rendre dans l'onglet "Données".

Comme nous voulons importer un fichier texte, nous choisissons le type adéquate depuis le groupe "Récupérer et transformer des données" du ruban :

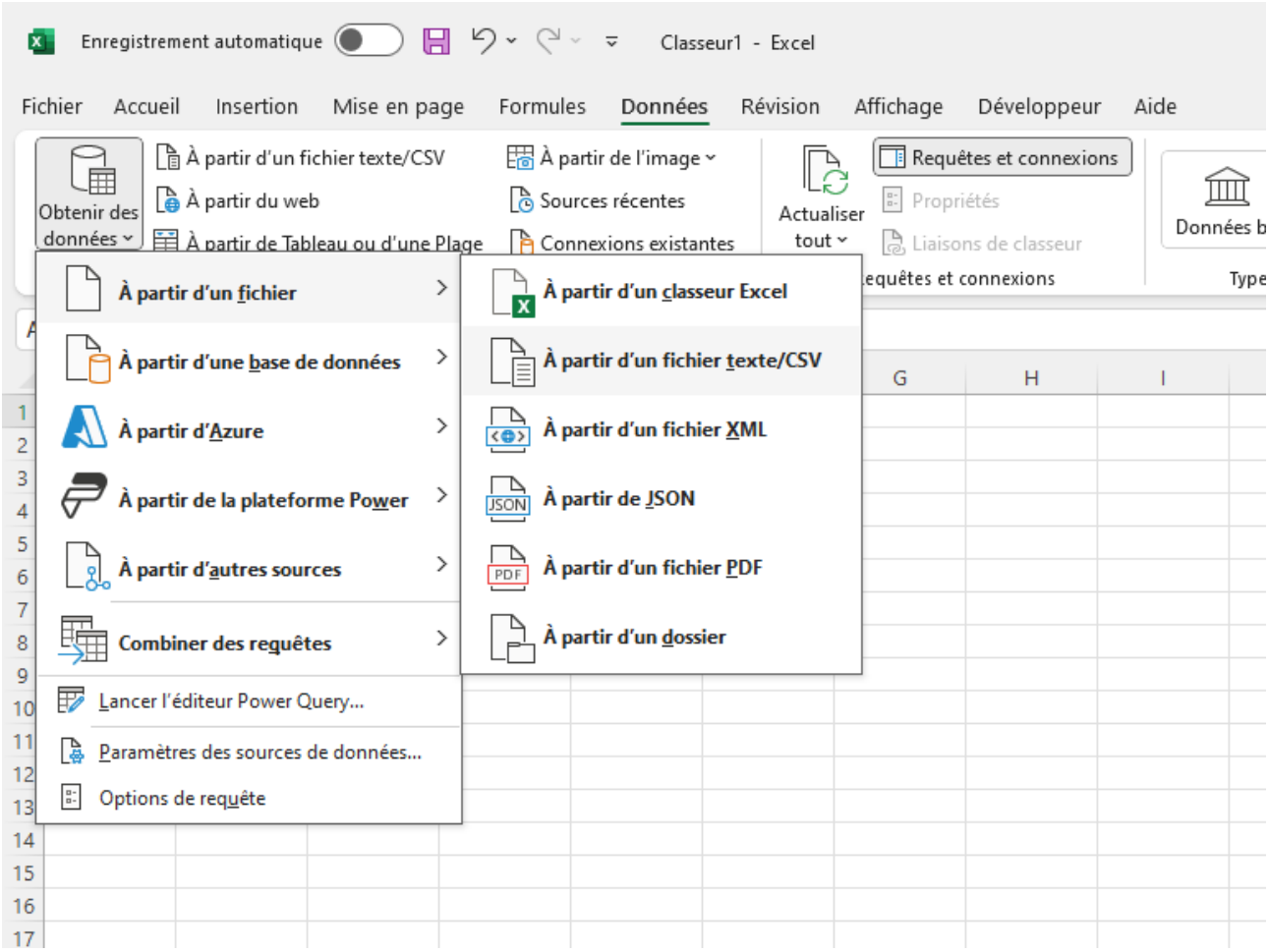

FIGURE 1.1. – Obtenir des données à partir d'un fichier texte/CSV.

Un explorateur de fichiers s'ouvre alors nous permettant de sélectionner notre fichier.

Une fois ce dernier chargé, une fenêtre d'aperçu apparaît nous permettant, entres autres, de choisir un délimiteur si celui-ci n'est pas correctement détecté ainsi que d'accéder à la transformation des données.

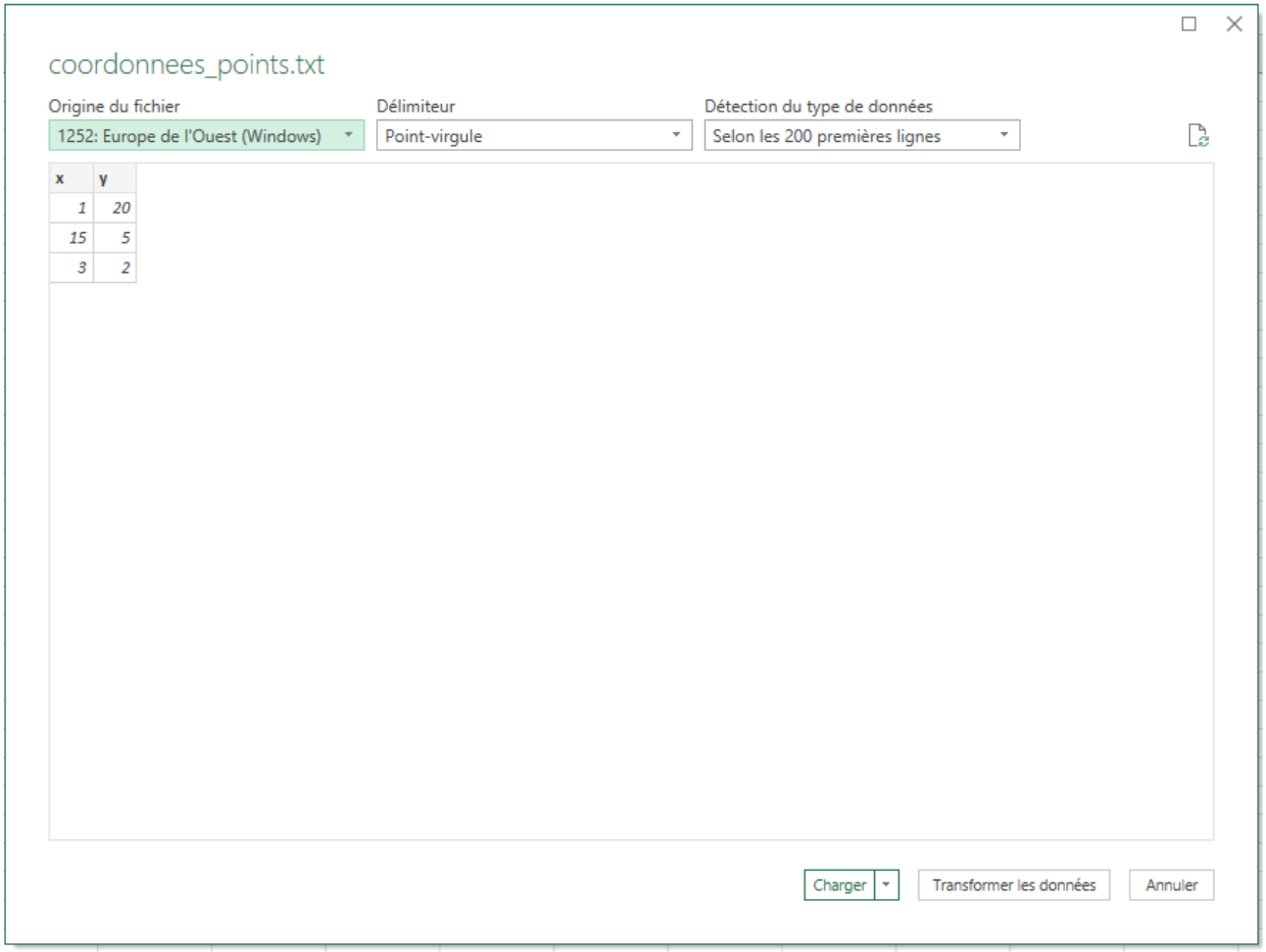

FIGURE 1.2. – Aperçu obtention des données.

La transformation des données peut être nécessaire pour éviter une mauvaise lecture de celles-ci. En effet, il faut par exemple parfois forcer une donnée numérique à être interprétée comme du texte sans quoi il y aura une perte : `00132` deviendra `132` ou encore `1234665465454892E203` deviendra `1,23466546545489E+218`. Autant le premier cas se rattrape assez facilement avec Excel, autant il y une perte définitive avec le second : ici le `2` avant le `E` est perdu.

Pour notre exemple, nous pouvons nous contenter de charger les données via le bouton approprié.

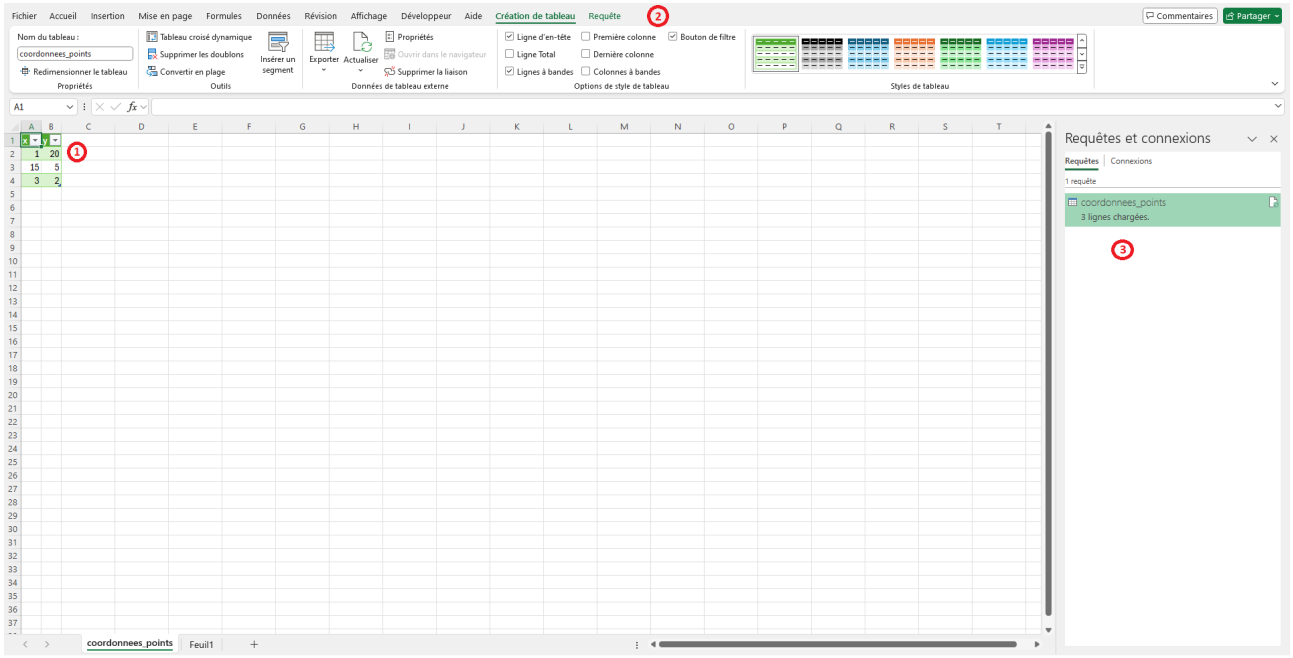

FIGURE 1.3. – État une fois les données chargées.

Les données sont chargées et nous pouvons remarquer qu'une liste de données a été créée dans un nouvel onglet (n°1 dans l'image ci-dessus). Tous les deux portent le nom du fichier choisi. Comme nous sommes placés dans ce tableau, nous voyons également deux onglets spécifiques au tableau et à la requête (n°2 ci-dessus). Enfin, tout à droite se trouve un panel "Requêtes et connexions" (n°3 dans l'image ci-dessus) qui contient notre nouvelle requête. La requête créée porte elle aussi le nom du fichier.

1.2. Création en VBA

Je vous avoue que je n'ajoute pas souvent des sources de données en VBA, parce que c'est plus simple et plus rapide depuis Excel, et que ce n'est pas une opération que j'ai déjà eue à devoir faire en masse.

Toutefois, nous pouvons imaginer devoir le faire dynamiquement depuis un programme alors voici ce qui dit l'enregistreur de macros pour la même chose :

```
1  Option Explicit
2
3  Sub Macro3()
4  '
5  ' Macro3 Macro
6  '
7  '
8      ActiveWorkbook.Queries.Add Name:="coordonnees_points",
           Formula:= _
```

```
        "let" & Chr(13) & "" & Chr(10) & _
            "    Source = Csv.Document(File.Contents(""C:\Users\Utilisateur\Des
             & Chr(13) & "" & Chr(10) &
            _
            "    #""En-têtes promus"" = Table.PromoteHeaders(Source, [PromoteA
             & Chr(13) & "" & Chr(10) &
            _
            "    #""Type modifié"" = Table.TransformColumnTypes(#""En-têtes pro
             &
            _
        " Int64.Type}})" & Chr(13) & "" & Chr(10) & "in" & Chr(13)
            & "" & Chr(10) & "    #""Type modifié"""
    ActiveWorkbook.Worksheets.Add
    With ActiveSheet.ListObjects.Add(SourceType:=0, Source:= _
        _
            "OLEDB;Provider=Microsoft.Mashup.OleDb.1;Data Source=$Workbook$;Loc
            _
     , Destination:=Range("$A$1")).QueryTable
        .CommandType = xlCmdSql
        .CommandText = Array("SELECT * FROM [coordonnees_points]")
        .RowNumbers = False
        .FillAdjacentFormulas = False
        .PreserveFormatting = True
        .RefreshOnFileOpen = False
        .BackgroundQuery = True
        .RefreshStyle = xlInsertDeleteCells
        .SavePassword = False
        .SaveData = True
        .AdjustColumnWidth = True
        .RefreshPeriod = 0
        .PreserveColumnInfo = True
        .ListObject.DisplayName = "coordonnees_points"
        .Refresh BackgroundQuery:=False
    End With
    Range("G5").Select
End Sub
```

i Recourir à l'enregistreur de macros est un bon moyen de transcrire rapidement des opérations en VBA afin d'en apprendre davantage sur la syntaxe, même si ses résultats ne sont pas toujours optimaux.

En analysant le code ci-dessus, nous pouvons voir qu'un objet de type requête (`Queries`) est ajouté puis qu'une liste de données est créée (`ListObject`). Cette dernière est un peu particulière puisque sa propriété `QueryTable` (représentant une table de requête) est définie et lit la requête.

C'est donc vers les tables de requête qu'il faudrait se pencher pour faire cette opération en VBA. À noter qu'une table de requête n'est pas forcément à écrire avec une liste de données

(l'objet `Range` possède aussi cette propriété) et qu'il n'y a pas toujours besoin d'un objet requête (`Queries`) non plus.

Si nous souhaitons recharger les données (comme nous le verrons un peu plus tard), les modifications manuelles apportées dans la liste de données seront écrasées.

Au cours de cette section, nous avons étudié les moyens d'ajouter une source de données.

2. Modifications via Power Query

Comme indiqué en introduction, nous pouvons ajouter une étape de modifications (transformation, ajout de colonnes, …) pour les données lues.

Cela se fait via l'éditeur Power Query. Le but de ce billet n'est pas de vous présenter ce dernier, mais de voir la manière dont celui-ci s'insère dans l'utilisation des sources de données.

Pour modifier les données lues, il faut modifier la requête. Cela se fait soit depuis l'onglet, soit depuis le panel de droite, comme montré ci-dessous :

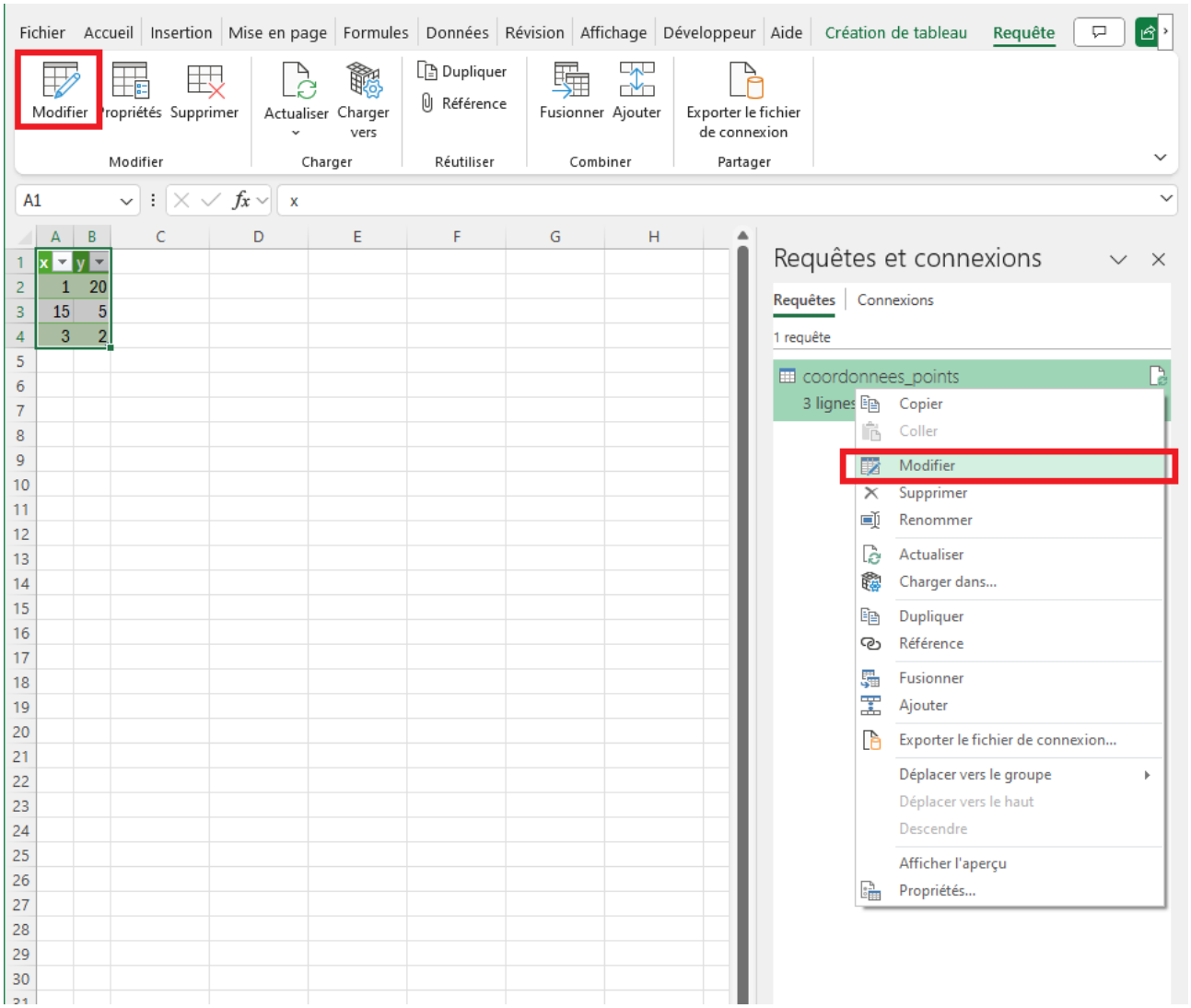

FIGURE 2.4. – Modifier la requête.

L'éditeur Power Query s'ouvre. Pour cet exemple, nous cliquons sur "Colonne personnalisée" de l'onglet "Ajouter une colonne". Ensuite, nous saisissons "Coordonnées" pour le nom de colonne et `"(" & Number.ToText([x]) & ", " & Number.ToText([y]) & ")"` pour la formule et validons.

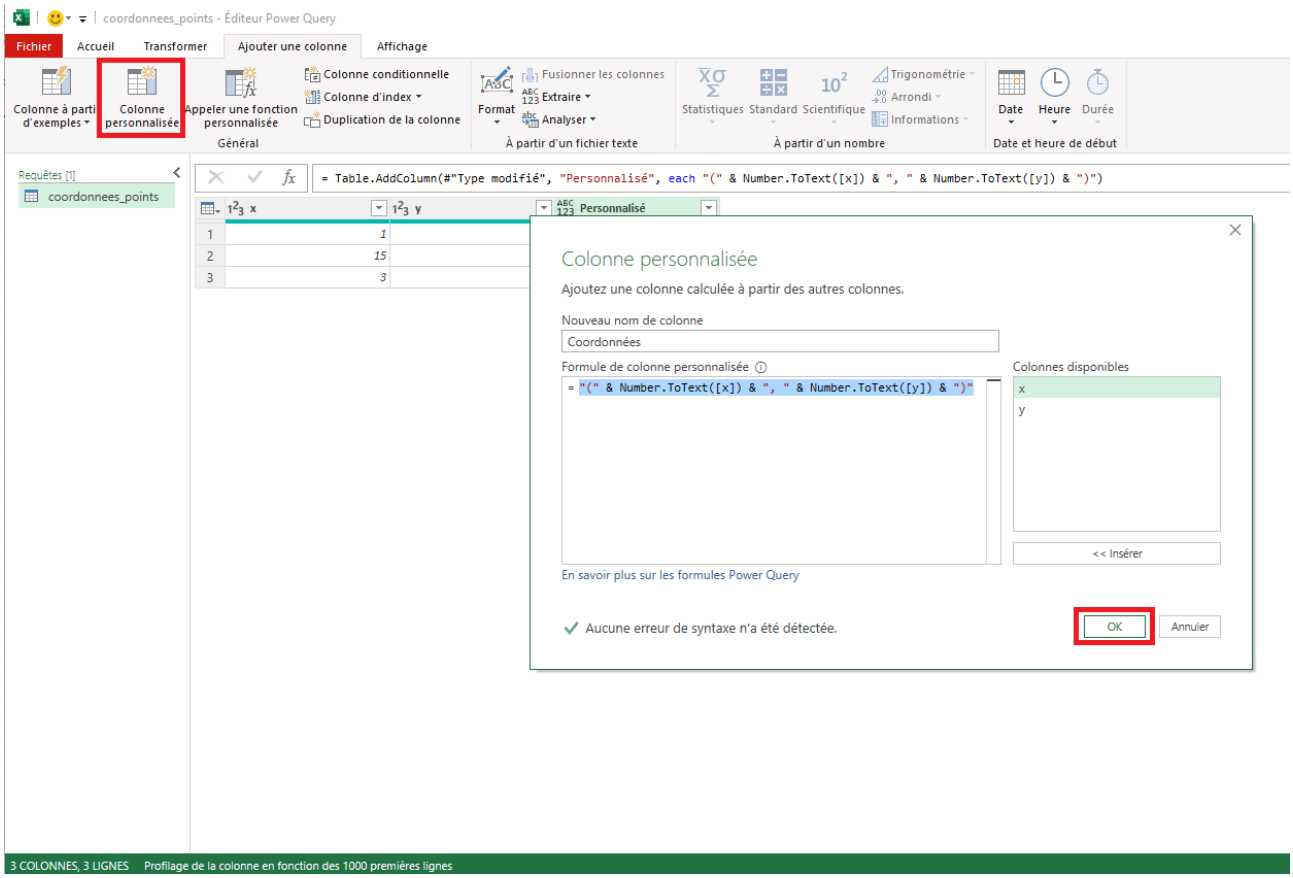

FIGURE 2.5. – Modification des données à travers l'ajout d'une colonne via Power Query.

Il ne nous reste maintenant plus qu'à fermer Power Query et charger le résultat obtenu :

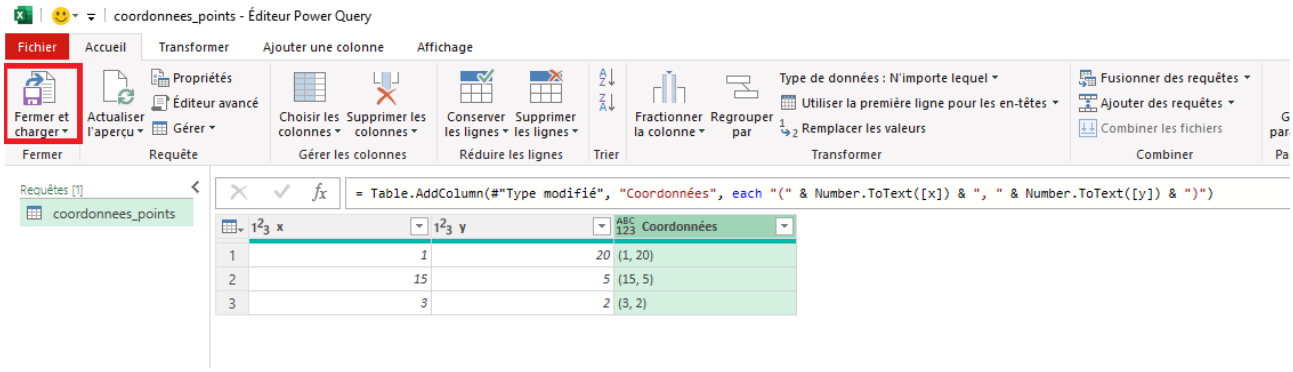

FIGURE 2.6. – Fermer Power Query et charger le résultat.

Nous constatons que nos modifications ont bien été repercutées dans notre `ListObject`.

Cette section a montré comment Power Query était lié aux sources de données.

3. Actualisation

Les données d'origine sont susceptibles d'évoluer (ajout, suppression, modification).

En conséquence, il nous faudrait récupérer la nouvelle version de celles-ci. Pas de problème puisqu'une fois la connexion établie, nous n'avons qu'à l'actualiser !

Ce qui fait foi dans la liaison vers un fichier est le chemin vers celui-ci. Si ce fichier est écrasé, il faut juste veiller à ce que le nom soit bien le même pour que la connexion puisse continuer à fonctionner. Il est ainsi possible de remplacer un fichier source par un autre en gardant la même connexion.

Pour cette section, nous commençons par modifier le contenu de notre fichier texte ainsi :

Fichier coordonnees_points.txt

```
x;y
14;5
3;2
4;3
```

3.1. Actualisation

Lors de l'actualisation, il se peut qu'il manque des informations de connexion (en particulier pour les destinataires du classeur). Une fenêtre s'ouvre alors pour demander les informations manquantes qui seront ensuite stockées. Si `DisplayAlert` est à `False`, elle ne s'affichera pas et la macro terminera par une erreur. Dans ce cas, il faut soit laisser la fenêtre des alertes, soit lancer l'actualisation depuis Excel pour obtenir cette fenêtre.

3.1.1. En VBA

Cette fois-ci, je présente l'approche VBA en premier, car c'est tout l'intérêt pour nos macros.

Il y a plusieurs moyens de procéder :

```
' Actualisation à partir de l'objet WorkbookConnection
ActiveWorkbook.Connections("Requête - coordonnees_points").Refresh

' Actualisation à partir de l'objet table de requête
Sheets("coordonnees_points").ListObjects("coordonnees_points").Que
    ryTable.Refresh

' Actualisation pour tout le classeur
ActiveWorkbook.RefreshAll
```

3.1.2. Depuis Excel

Cela correspond aux éléments suivant dans l'interface :

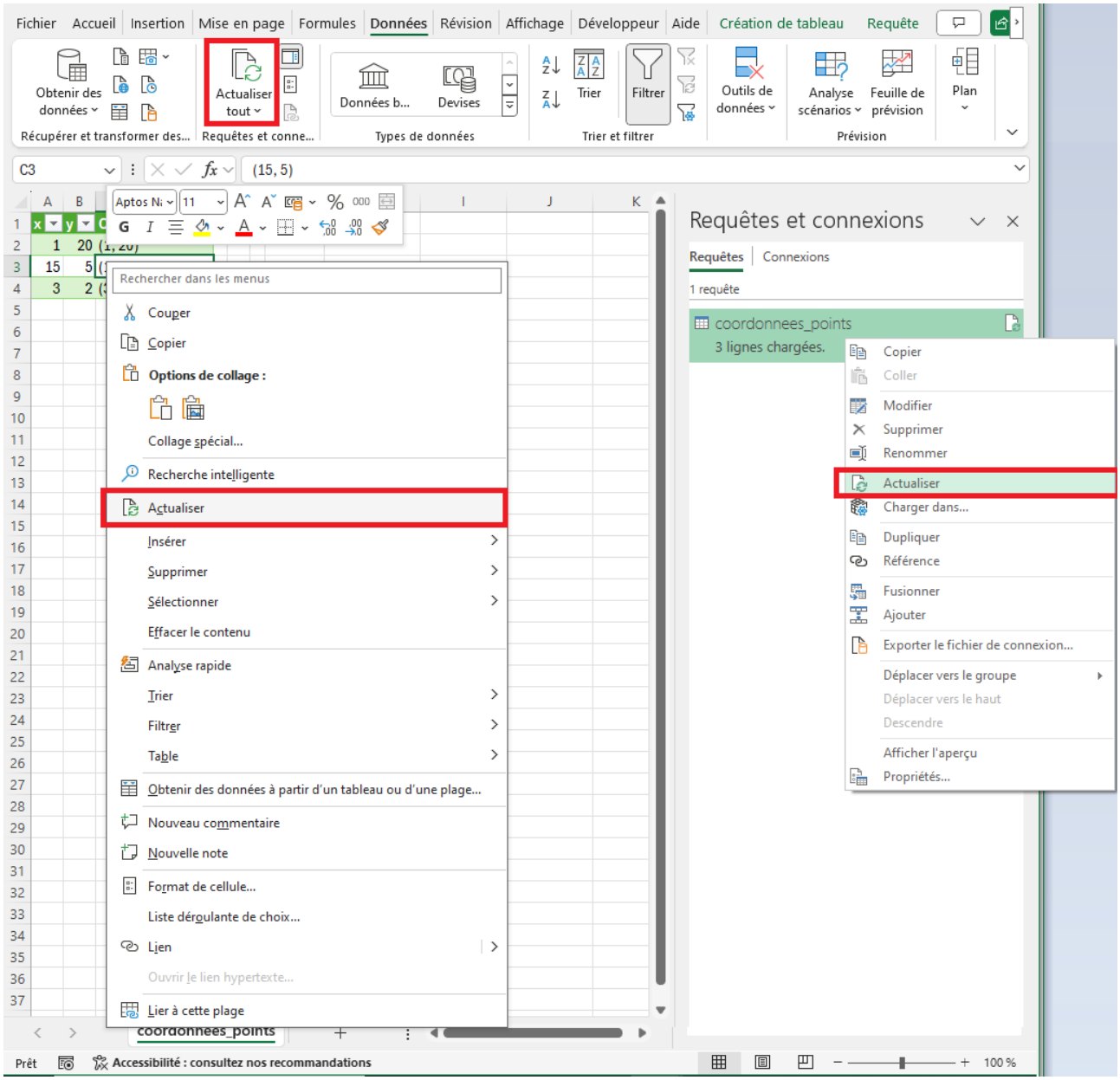

FIGURE 3.7. – Actualiser source de données.

Nous pouvons observer qu'en utilisant une de ces options, les données sont bien mises à jour.

3.2. Mode arrière-plan

Il est possible de définir le caractère bloquant ou non (respectivement synchrone et asynchrone) de l'actualisation soit via l'interface soit via un paramètre facultatif `BackgroundQuery` de `Refresh`.

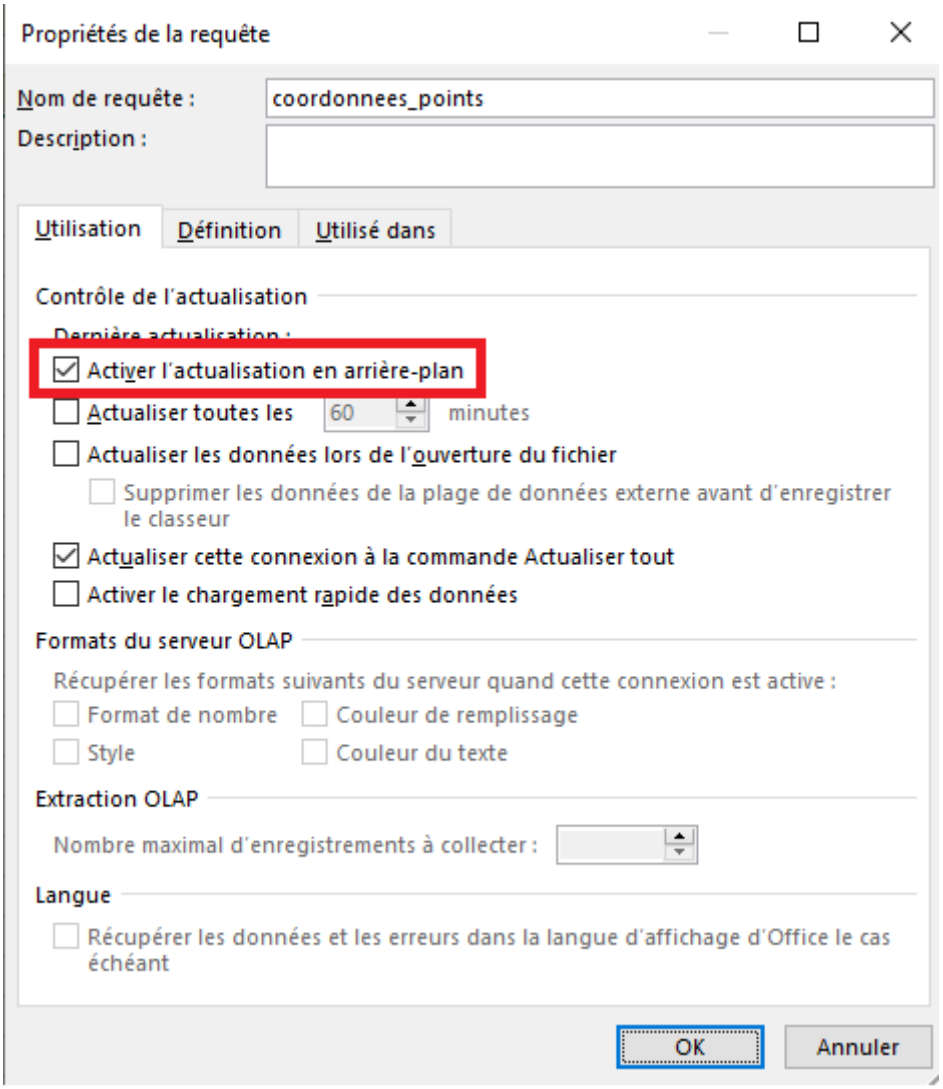

FIGURE 3.8. – Propriété d'arrière-plan depuis les paramètres de la requête.

Il peut être nécessaire de passer en synchrone pour attendre que les données soient bien chargées avant de faire d'autres opérations, en décochant cette case.

3.3. Événements

Les objets `QueryTable` possèdent deux événements possibles pour potentiellement annuler l'actualisation puis vérifier qu'elle s'est bien déroulée :

```
1 QueryTable_BeforeRefresh(Cancel As Boolean) ' Avant actualisation
2 QueryTable_AfterRefresh(Success As Boolean) ' Après actualisation
```

Pour les utiliser, il faut créer un module de classe et déclarer un objet `QueryTable` avec événements comme ceci :

```
Option Explicit

Private WithEvents mQTEvenement As QueryTable
```

Cela nous permet d'ajouter les procédures d'événement au moyen des zones d'objet et de procédures/événements :

```
Private Sub mQTEvenement_BeforeRefresh(Cancel As Boolean)
    MsgBox ("Avant !")
End Sub

Private Sub mQTEvenement_AfterRefresh(ByVal Success As Boolean)
    MsgBox ("Après !")
End Sub
```

Ensuite, il nous faut ajouter le liant entre la table de requête existante et la propriété en tant que variable membre :

```
Public Property Set qtEvenement(ByRef oQT As QueryTable)
    Set mQTEvenement = oQT
End Property
```

Pour résumer, cela donne pour cette classe :

Module de classe clsQTEvenement

```
Option Explicit

Private WithEvents mQTEvenement As QueryTable

Private Sub mQTEvenement_BeforeRefresh(Cancel As Boolean)
    MsgBox ("Avant !")
End Sub

Private Sub mQTEvenement_AfterRefresh(ByVal Success As Boolean)
    MsgBox ("Après !")
End Sub

Public Property Set qtEvenement(ByRef oQT As QueryTable)
    Set mQTEvenement = oQT
End Property
```

N'oublions pas l'instanciation de celle-ci et la liaison que nous ferons dans l'objet classeur pour cet exemple :

ThisWorkbook

```
Option Explicit

Dim oQTEvenement As clsQTEvenement

Private Sub Workbook_Open()
    Set oQTEvenement = New clsQTEvenement
    Set oQTEvenement.qtEvenement = Worksheets("coordonnees_points"
        ).ListObjects("coordonnees_points").QueryTable
End Sub
```

En relançant le classeur et en actualisant, nous nous rendons compte que ça fonctionne bien.

À travers cette section, nous avons vu l'actualisation des données.

Conclusion

Le long de ce billet, nous avons appris le fonctionnement des sources de données d'Excel (ajout, modifications via l'éditeur Power Query et actualisation). C'est beaucoup plus simple que de tout programmer soi-même.

Pour accéder aux données résultantes en VBA, il ne nous reste alors plus qu'à lire le contenu de la liste de données comme vu dans le précédent billet.

Lorsque le classeur est partagé, il faut veiller à ce que ces sources soient accessibles par le destinataire. Sinon, il ne pourra pas actualiser les données. Si nécessaire, il faut également penser à définir des autorisations (bouton "Obtenir des données" puis "Paramètres des sources de données…" depuis l'onglet "Données").

À bientôt !

Quelques ressources :

— La documentation concernant l'objet WorkbookConnection
— La documentation concernant l'objet QueryTable
— La documentation concernant l'utilisation d'événements avec l'objet QueryTable

Merci à @kayou pour le retour.