Projet Final : Access

ANAIS, BERTRAND - - VIEUX - MELCHIOR
anais.bertrand---vieux--melchior@etu.univ-lyon1.fr

PRISCILLA, GOGUY
priscilla.goguy@etu.univ-lyon1.fr

MAHLÎ, REINETTE
mahli.reinette@etu.univ-lyon1.fr

13/10/2025

M1 Actuariat
ISFA

Table des matières

1 Préambule

1.1 Préambule

Toutes les ressources utilisées seront présentes en annexes et sur le ***repository*** github ci-dessus.
Nous souhaitons de prime abord créer avec *Microsoft Access* un outil indiquant des trajets possibles dans le métro de Paris en fonction d'une gare de départ et d'une gare d'arrivée.

Une frise chronologique nous accompagnera tout le long du projet. Elle marquera en vert notre avancée réelle et rouge, les ressources précédentes, utilisées pour construire et ou étoffer notre réponse.

2 Construction

2.1 1. Source des données

Les données

Nous disposions en premier lieu, d'une base de données formatée en fichier ***.csv***.
Elle présentait en ligne nos *enregistrements* et en colonnes nos variables respectives : *« ligne »*, *« temps »* et *« gare*[1] *»*.

En raison de la présence d'une en-tête dans nos données d'origine, nous dûmes importer ces dernières en connaissance de cause[2].

a. [1] Respectivement la ligne de départ, le temps en seconde pour parvenir à la station à partir du terminus de la ligne et le nom de la station.
b. [2] Notons par ailleurs l'utilisation des *«; »* comme séparateurs de champs.

L'importation

Il nous a fallu correctement définir le type de nos champs :
— *« le champ ligne »*, comme étant composé de textes courts[3].
— *« le champ temps »*, comme étant composé de variables quantitatives de type *double*[4].
— *« le champ gare »*, comme étant composé de textes courts.

c. [3] Certaines valeurs contenant des lettres ou des caractères spéciaux. Par exemple la valeur : *« 3bis »*, présente dans la ligne 167.
d. [4] Bien qu'elle ne contienne dans les faits que des entiers.

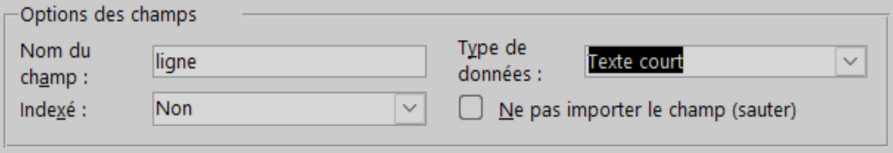


FIGURE 1 – Importation de la variable *« ligne* [1] *»*

ligne	temps	gare
1	0.00	Grande Arche de la Défense
1	56.00	Esplanade de la Défense
1	109.00	Pont de Neuilly

FIGURE 2 – Résultat de l'importation des variables

2.2 2. Table Choix

Pour la création de la *table « choix »*, deux solutions s'offraient à nous.
Nous pouvions d'un côté la créer à la main à l'aide d'outils natifs de *Microsoft Access* du module création, et de l'autre la confectionner à l'aide de *requêtes SQL*.

Dans le premier cas, il est toutefois nécessaire de définir le type de chaque champ dans le mode création, et de veiller à typer les deux champs en *« texte court »* afin de correctement stocker le nom des gares.

Requête *SQL*

Nous devons notre salut à une *ressource externe w3s*, dont l'aide s'est révélée précieuse concernant la création de table :

```
CREATE TABLE Choix(
    GareDépart TEXT,
    GareArrivée TEXT
);
```

et une *seconde* sur l'insertion de nouvelles mesures.

```
INSERT INTO Choix (GareDépart, GareArrivée)
VALUES ('Bastille', 'Nation');
```

Dans le cadre de la question, nous devions créer une ***« liste déroulante »*** pour faciliter la saisie des gares dans la *table*. Nous avons réalisé cela directement dans le *mode création* de la *table*, en utilisant l'assistant *Liste de choix*[5].

e. [5] Cet assistant permet de définir que les valeurs de la liste proviennent d'une autre table, en l'occurrence ici Lignes.

```
SELECT DISTINCT Lignes.gare
FROM Lignes
ORDER BY Lignes.gare;
```

Général | Liste de choix

Contrôle de l'affichage	Zone de liste déroulante
Origine source	Table/Requête
Contenu	SELECT DISTINCT ([Lignes].[gare]) FROM Lignes ORDER BY [gare];

FIGURE 3 – Le DISTINCT nous permet de supprimer les doublons

Lors de la mise en place initiale, nous avons constaté que la *requête* de sélection utilisée par l'assistant n'était pas exactement celle que nous souhaitions : certaines gares apparaissaient plusieurs fois dans la liste déroulante. Pour résoudre ce problème, nous avons modifié la requête en ajoutant un DISTINCT[6].

2.3 3. Requêtes Lignes de départ et Lignes d'arrivée

La *requête « LignesDépart »* (respectivement *« LignesArrivée »*) permet d'obtenir la liste des lignes passant par la gare de départ (respectivement d'arrivée).

Requête *« LignesDépart »*

```
1  SELECT Choix.[GareDépart], Lignes.[ligne], Lignes.[temps]
2  INTO LigneDépartT
3  FROM Lignes INNER JOIN Choix
4  ON Lignes.[gare] = Choix.[GareDépart] ;
```

Cette jointure garantit que seules les lignes correspondant à la gare de départ sélectionnée sont affichées.

NB :
Nous avons noté qu'il est important de relancer l'exécution de la requête à chaque modification de la gare de départ pour actualiser les résultats.

Nous procédons de façon sensiblement similaire pour la *requête « LignesArrivée »*.

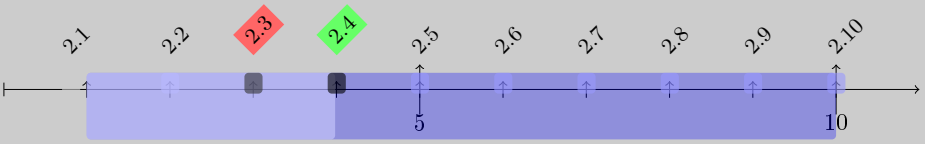
Requête *« LignesArrivée »*

```
1  SELECT Choix.[GareArrivée], Lignes.[ligne], Lignes.[temps]
2  INTO LigneArrivéeT
3  FROM Lignes INNER JOIN Choix
4  ON Lignes.[gare] = Choix.[GareArrivée] ;
```

6. [6] Afin que chaque gare n'apparaisse qu'une seule fois dans la liste déroulante.

2.4 4. Trajets directs

Notre doctrine a été toute autre pour cette 4^{e} partie.
Là où nous nous sommes (pour les précédentes implémentations) principalement concentrés sur des *requêtes sql*, pariant ainsi sur une plus grande efficacité de notre algèbre relationnel, nous avons ici profité des capacités natives de jointures *user friendly* du logiciel *Microsoft Access*.



Progression

Pour ce faire, nous nous sommes servis de *l'assistant de requête MA* afin de créer une **requête** simple basée sur nos deux requêtes précédentes, *« LignesDépart* [1] *»* et *« LignesArrivée* [2] *»*.

```
1  SELECT
2  LignesDépart.GareDépart,
3  LignesDépart.ligne AS L0,
4  LigneArrivée.GareArrivée,
5  Abs(LignesDépart.temps - LigneArrivée.temps) AS Temps
6  FROM LignesArrivée INNER JOIN LignesDépart
7  ON LignesArrivée.ligne = LignesDépart.ligne;
```

La **jointure** a été effectuée sur le champ *ligne*, ce qui permet de sélectionner uniquement les lignes communes aux deux gares. Chaque enregistrement renvoie le couple *(GareDépart, GareArrivée)* et la ligne correspondante, nommée *L0*. Pour calculer le temps de trajet entre les deux gares, nous avons utilisé la valeur absolue de la différence entre les temps d'accès depuis le terminus pour chaque gare.

Une autre approche aurait été d'utiliser une *instruction* IIF, afin d'ajuster la différence en multipliant par -1 si celle-ci était négative. Cette *méthode* permet de lister tous les trajets directs possibles et d'obtenir rapidement le temps de parcours pour chaque trajet.

Dans un second temps nous avons souhaité vérifier le cas où plusieurs lignes sont possibles (vérifiant que notre *requête* sélectionne bien tous les cas possibles).

GareDépart	L0	GareArrivée	Temps
Bastille	8	Concorde	1137
Bastille	1	Concorde	596

FIGURE 4 – Par exemple : *« Bastille - Concorde »*

2.5 5. Correspondances

Pour ce faire nous avons *instancié* 2 fois la table *« lignes »*, respectivement nommées : *« Lignes_1 »* (*« Lignes_2 »*).
La **requête** *« Corresp »* ne modifiant aucunement notre *dataset*, le couple *(Lignes_1, Lignes_2)* forme en conséquence un ensemble de deux copies distinctes de la même table.

La première copie (et respectivement la seconde), représentera la *« LigneAvant »* (et respectivement la *« LigneAprès »*).

Nous avons effectué une *jointure* entre nos deux *instances* de la table *« lignes »*, à l'aide des critères suivant :

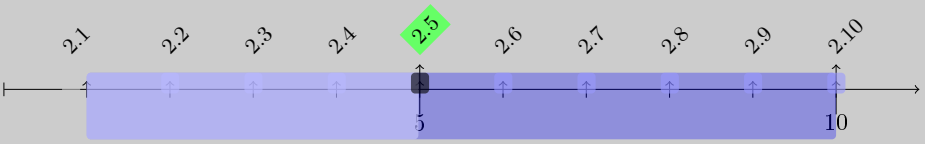
— La gare doit être commune aux deux lignes *(condition de JOINTURE)*.
— Les lignes doivent être distinctes[1].

Cette *jointure* permet de résumer dans une table unique toutes les informations nécessaires pour chaque correspondance.

Par ailleurs :

nous faisons correspondre à chaque *« correspondances »* un temps d'accès à la gare commune depuis les terminus de chaque ligne (par simple ajout dans SELECT), afin de connaître la durée nécessaire pour atteindre cette gare avant et après changement.

```
1   SELECT
2   Lignes_1.ligne AS LigneAvant,
3   Lignes_2.ligne AS LigneAprès,
4   Lignes_1.gare AS gare,
5
6   Lignes_1.temps AS TempsDepuisTerminusAvant,
7   Lignes_2.temps AS TempsDepuisTerminusApres
8   FROM Lignes AS Lignes_1 INNER JOIN Lignes AS Lignes_2
9   ON Lignes_1.gare = Lignes_2.gare
10  WHERE Lignes_1.ligne <> Lignes_2.ligne;
```



Progression

1. [1] Afin d'éviter qu'une ligne ne soit mise en correspondance avec elle-même (clause WHERE).

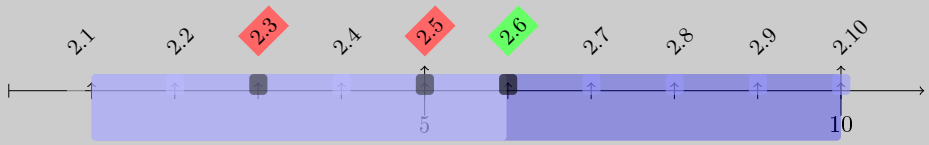
2.6 6. Trajets avec changements

Notre approche a été la suivante :
pour chaque gare de départ, nous commençons par identifier les lignes accessibles grâce à la *requête « LignesDépart* [1] *»*. Ensuite, nous utilisons la *requête « Corresp* [2] *»*, pour déterminer les gares de correspondance possibles (les gares où un changement de ligne peut s'effectuer).
Pour finir, nous vérifions que la gare d'arrivée se situe sur l'une des lignes listées par *« LignesArrivée* [3] *»*.

Notre *requête « Avec1Chgt »* repose sur deux jointures successives, lister ci-dessous :

— La première permet d'identifier la première partie du trajet et le changement[1].
— La seconde d'atteindre la gare finale[2].

Le champ *« temps »* correspond au temps total du parcours et est calculé selon le même principe que précédemment[3].



Progression

```
1       SELECT
2       LignesDépart.GareDépart,
3       LignesDépart.ligne AS L0,
4       Corresp.gare AS Chgt1,
5       Corresp.LigneAprès AS L1,
6       LignesArrivée.GareArrivée,
7
8       Abs(Corresp.TempsDepuisTerminusAvant - LignesDépart.temps) +
        Abs(Corresp.TempsDepuisTerminusApres - LignesArrivée.temps) AS
        temps
9       FROM
10      (LignesDépart INNER JOIN Corresp ON LignesDépart.ligne = Cor-
        resp.LigneAprès)
11      INNER JOIN LignesArrivée ON Corresp.LigneAprès = LigneArri-
        vée.ligne
12      WHERE
13      ((Corresp.gare <> LignesDépart.GareDépart) AND (Corresp.gare <>
        LigneArrivée.GareArrivée)) ;
```

1. [1] Placée entre *« LignesDépart »* et *« Corresp »*.
2. [2] Placée entre *« Corresp »* et *« LignesArrivée »*.
3. [3] Il s'agit de la valeur absolue de la différence entre les temps d'accès aux gares depuis les terminus, afin d'obtenir une durée positive pour chaque segment.

Nous tenons par ailleurs à attirer l'attention du lecteur sur la clause WHERE qui permet de s'assurer que les gares successives sont toutes distinctes (i.e. *« GareDépart »* différente de celle de correspondance et celle de correspondance différent de *« GareArrivée »*).

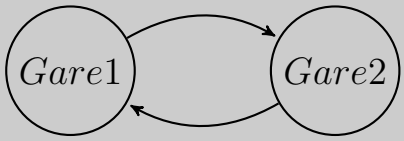
Concernant les trajets comportant deux changements, notre approche a été sensiblement similaire à la précédente :

« Partir d'une gare de départ, effectuer une première correspondance, puis une seconde, avant d'atteindre la gare d'arrivée. »

Pour ce faire, nous avons créé une copie de la *requête « Corresp »* (de la même façon que nous avions créé des copies de la table Lignes pour les correspondances précédentes).
Dans un premier temps, nous avions brièvement envisagé de réutiliser la *requête « Avec1Chgt »* pour déterminer le premier changement. Idée qui fut néanmoins rapidement écartée (dû au caractère trop restrictif[1] de la *requête*).

Il a également été nécessaire de filtrer correctement dans la clause WHERE : aucune gare successive ne doit être identique pour éviter des trajets incohérents.

Nous avons par ailleurs ajouté une condition supplémentaire stipulant que la deuxième gare de correspondance ne peut pas être la gare de départ, afin d'éviter des trajets du type :



De manière générale, tous les trajets qui repassaient par une gare déjà visitée ont été éliminés.
En revanche, nous n'avons pas exclu le fait de repasser par la même ligne, ce qui peut parfois permettre de raccourcir le trajet.

Ne sachant pas comment coder le *« valeur différente de »* en *MS ACCESS*, nous avons cherché de l'aide sur Internet et trouvé notre solution ici : *Table of operators - Microsoft Support*.

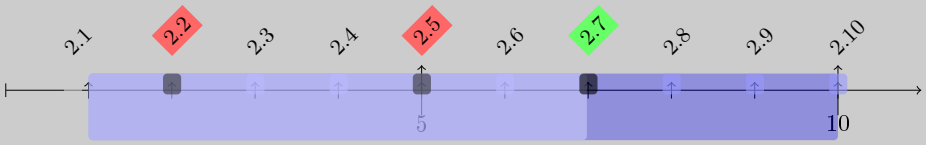
a. [1] Avec deux changements, il existe davantage de chemins possibles, et limiter le premier changement aux trajets déjà identifiés dans *« Avec1Chgt »* exclurait certaines solutions.

```
SELECT
LignesDépart.GareDépart,
LignesDépart.ligne AS L0,
Corresp.gare AS Chgt1,
Corresp.LigneAprès AS L1,

Corresp_1.gare AS Chgt2,
Corresp_1.LigneAprès AS L2,

LignesArrivée.GareArrivée,

Abs(Corresp.TempsDepuisTerminusAvant - LignesDépart.temps)
+ Abs(Corresp_1.TempsDepuisTerminusAvant - Corresp.TempsDepuisTerminusApres)
+ Abs(LignesArrivée.temps - Corresp_1.TempsDepuisTerminusApres)
AS temps

FROM ((LignesDépart
INNER JOIN Corresp ON LignesDépart.ligne = Corresp.LigneAvant)
INNER JOIN Corresp AS Corresp_1 ON Corresp.LigneAprès = Corresp_1.LigneAvant)
INNER JOIN LignesArrivée ON Corresp_1.LigneAprès = LignesArrivée.ligne

WHERE
((Corresp.gare <> LignesDépart.GareDépart)
AND
(Corresp_1.gare <> Corresp.gare)
AND
(LignesArrivée.GareArrivée <> Corresp_1.gare)
AND
(LignesArrivée.GareArrivée <> Corresp.gare))
;
```

2.7 7. Requêtes d'ajout, table des trajets

Nous allons procéder à la création de table :

```
CREATE TABLE Trajets(
    GareDépart TEXT,
    Chgt1 TEXT,
    Chgt2 TEXT,
    GareArrivée TEXT,
    Temps DOUBLE,
    L0 TEXT,
    L1 TEXT,
    L2 TEXT
);
```



Progression

L'aide utilisée lors de la *question 2* nous a permis de résoudre cette section rapidement, en nous rappelant comment lier correctement les colonnes de la table cible (à remplir) avec les éléments correspondants provenant des autres requêtes. Il était également important de veiller à ce que les résultats soient triés par ordre croissant de temps pour chaque requête d'ajout.

Cela a été réalisé grâce à l'instruction ORDER BY temps, qui, par défaut, trie les valeurs de manière croissante *(ASC)*. Rien de particulièrement complexe dans cette étape : il s'agissait simplement, en fonction du nombre de correspondances, de s'assurer de sélectionner les bons champs de la table Trajets et de les remplir avec les champs correspondants issus des requêtes créées précédemment.

```
1  SELECT
2  Avec0Chgt.GareDépart,
3  Avec0Chgt.GareArrivée,
4  Avec0Chgt.temps,
5  Avec0Chgt.L0
6
7  FROM Avec0Chgt
8  ORDER BY Avec0Chgt.temps;
```

```
1  INSERT INTO Trajets (GareDépart, Chgt1, Chgt2, GareArrivée,
   Temps, L0, L1)
2  SELECT
3  Avec1Chgt.GareDépart,
4  Avec1Chgt.Chgt1,
5  Avec1Chgt.GareArrivée,
6  Avec1Chgt.temps,
7  Avec1Chgt.L0,
8  Avec1Chgt.L1
9
10 FROM Avec1Chgt
11 ORDER BY Avec1Chgt.temps;
```

```
1   INSERT INTO Trajets (GareDépart, Chgt1, Chgt2, GareArrivée,
    Temps, L0, L1, L2)
2   SELECT
3   Avec2Chgt.GareDépart,
4   Avec2Chgt.Chgt1,
5   Avec2Chgt.Chgt2,
6   Avec2Chgt.GareArrivée,
7   Avec2Chgt.temps,
8   Avec2Chgt.L0,
9   Avec2Chgt.L1,
10  Avec2Chgt.L2
11
12  FROM Avec2Chgt
13  ORDER BY Avec2Chgt.temps;
```

Pour supprimer nous avons utilisé l'aide fournie ici notamment car nous ne savions pas supprimer tous les enregistrements mais uniquement plutôt certains enregistrements : *SQL DELETE Statement*.

```
1   DELETE FROM Trajets;
```

Cette requête nous permet de supprimer tous les enregistrements sans pour autant supprimer les entêtes et sans non plus supprimer la table.

2.8 8. Interface utilisateur

Pour cette question, nous avons créé un *formulaire Guide* à partir de la *table « Choix »*, ce qui permet d'afficher directement les deux champs qu'elle contient. Nous avons ensuite inséré, en *« mode Création »*, un bouton de commande que nous avons renommé afin qu'il reflète clairement sa fonction. La configuration des actions associées à ce bouton a été réalisée également en *« mode Création »*, via la fenêtre des propriétés et la gestion des événements déclenchés lors du clic. Nous avons ainsi défini une série d'actions : d'abord sauvegarder l'enregistrement en cours (mise à jour de la *table « Choix »*), puis exécuter la requête de suppression suivie des différentes requêtes d'ajout. L'ensemble a été paramétré dans une *macro*, en veillant à respecter l'ordre d'exécution approprié, comme l'illustre la capture d'écran ci-contre.

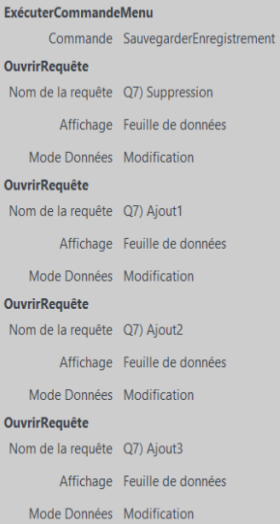
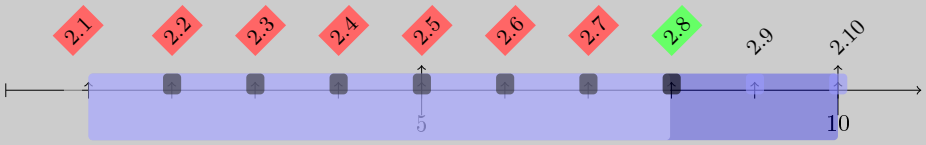


FIGURE 5 – Formulaire



Progression

2.9 9. Sous-formulaire

Pour cette question, nous avons préalablement créé un *formulaire* fondé sur la *table « Trajets »*, puis nous l'avons inséré en tant que *sous-formulaire* dans le *formulaire « Guide »* par simple *drag & drop* en *mode Création*. L'ergonomie s'en trouve grandement améliorée, l'ensemble des trajets possibles correspondant au choix de l'utilisateur étant directement affiché sur ce même *écran*.

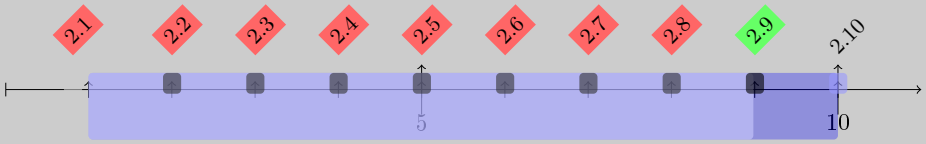
Nous avons également profité de cet instant pour ajouter un *bouton de navigation* afin de parcourir avec plus d'aisance les différents trajets listés dans ce *sous-formulaire*.

Le *bouton principal* a été complété.
Nous avons intégré une *action « Actualiser »* (présentée ci-contre), pour permettre au *sous-formulaire* de rafraichir son *gui*, suite à une sélection nouvelle.

Enfin, nous avons créé une *macro « AutoExec[10] »*, s'exécutant automatiquement au démarrage et ouvrant directement le *formulaire « Guide »*.

Cette configuration permet en théorie à l'utilisateur d'accéder immédiatement à l'interface de sélection et de visualisation des trajets dès l'ouverture de la base.

Cette première phase de travail d'ergonomie, en a entrainé une autre, que nous serons amenés à détailler dans les parties suivantes.



Progression

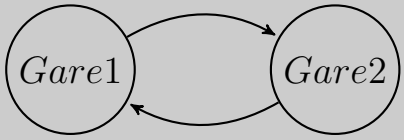
3 Compléments libres, Conclusions et Commentaires

3.1 10. Compléments libres

Il est à noter (pour clore cette partie) que plusieurs modifications ont étés apportés pour rendre notre modèle plus *user-friendly*.

1) Affinage de certaines requêtes

Tout d'abord, nous avons affiné certaines *requêtes*, notamment celle des trajets avec changements[1], afin de garantir une production cohérente de résultats.
i.c. pas de trajet de type :



2) Enrichissement du formulaire

Nous avons également enrichi le *formulaire* conçu lors de notre partie précédente[2] en y ajoutant un *bouton* supplémentaire permettant de naviguer plus facilement entre les différents trajets proposés.
Cette fonctionnalité rend l'utilisation du *sous-formulaire* plus intuitive et évite d'avoir l'affichage de tous les trajets possibles en un bloc.

3) Zone d'affichage textuelle du trajet a effectué dans le formulaire

Dans le cadre des améliorations apportées à l'interface utilisateur, nous avons ajouté sous le *sous-formulaire « Trajets*[3] *»* une zone d'affichage générant automatiquement une phrase décrivant le trajet sélectionné. Cette phrase se met à jour dynamiquement dès que l'utilisateur change de trajet dans le sous-formulaire. Pour commencer, nous avons ouvert le *sous-formulaire* contenant les trajets en *mode Création*. Nous y avons inséré une zone de texte sous le tableau des trajets, destinée à afficher une description lisible du parcours, sous forme de phrase complète.

Afin que le texte affiché s'adapte automatiquement au trajet en cours, nous avons renseigné dans la propriété Source contrôle de cette zone une expression conditionnelle utilisant IIf, permettant de gérer : les trajets sans changement, avec un changement et avec deux changements. Cette première documentation nous a été utile pour déterminer que l'on avait besoin d'une source de contrôle et du générateur d'expression : *Ajouter un contrôle Zone de texte à un formulaire ou un état - Support Microsoft*.

Cette partie là : *Utilisation du Générateur d'expressions - Support Microsoft* nous a aidé sur la syntaxe très précise : les points virgules dans le IIf, les crochets autour des noms de champs et ce type d'informations. Nous avons aussi utilisé une fonction de détection de valeur absente IsNull à l'aide de la documentation ici : *MS Access IsNull() Function*.

De plus, nous avons aussi eu besoin de l'opérateur de concaténation de chaînes de caractère : *l'esperluette* que nous avons trouvé ici : *MS Access Concat with &*.

Finalement, ces différents éléments nous ont permis de produire le code suivant dans source de contrôle :

```
1  =IIf(IsNull([Chgt1]);
2      "Prendre la ligne " & [L0] & " de " & [GareDépart] & " jusqu'à
   " & [GareArrivée] & ".";
3      IIf(IsNull([Chgt2]);
4          "Prendre la ligne " & [L0] & " depuis " & [GareDépart] &
   " jusqu'à " & [Chgt1] & ", puis la ligne " & [L1] & " jusqu'à " &
   [GareArrivée] & ".";
5          "Prendre la ligne " & [L0] & " depuis " & [GareDépart] & "
   jusqu'à " & [Chgt1] & ", puis la ligne " & [L1] & " jusqu'à " & [Chgt2]
   & ", puis la ligne " & [L2] & " jusqu'à " & [GareArrivée] & "."
6      )
7  )
```

Pour lire le code : premier IIf : si aucun changement n'est prévu, la formule génère simplement une phrase indiquant le trajet direct, sinon il y a au moins un changement. Un second IIf permet de vérifier si le deuxième champ de changement est vide. Si c'est le cas, la phrase correspond à un trajet avec un seul changement dans le cas contraire, la phrase décrit un trajet avec deux changements.

4) État graphique : temps de parcours minimal, maximal, moyen en fonction du nombre de changements (et requête associée)

Nous voulions produire un état graphique permettant de visualiser rapidement les temps de parcours selon le nombre de changements dans les trajets. Plus précisément pour chaque type de trajet (0, 1 ou 2 changements) : le temps minimal, le temps maximal, le temps moyen. Cette représentation permet de comparer visuellement la durée des trajets directs, avec un ou deux changements, et d'identifier facilement les écarts de temps.

Pour créer cet état, nous avions besoin d'une *requête* unique regroupant les trajets par nombre de changements et calculant, pour chaque groupe, le temps minimal, le temps maximal et le temps moyen.
L'idée était de préparer directement les données agrégées par type de parcours (0, 1 ou 2 changements), afin de pouvoir créer rapidement un graphique clair et synthétique dans l'état de restitution (d'où l'usage du GROUP BY).

Ensuite, pour regrouper les trajets selon le nombre de changements, nous avons créé un *champ* calculé appelé *« NbreChgt »* valant 0 si aucun changement c'est à dire *« siChgt1 »* est vide, 1 si Chgt1 est non vide et *« Chgt2 »* Vide et enfin 2 sinon. C'est ce que permet le Iif dans le SELECT. On agrège aussi selon cette donnée. Puis on utilise les fonctions sur les agrégats : MIN, MAX et MEAN.

```
1 SELECT IIF (IsNull([Chgt1]), 0 , IIF(IsNull([Chgt2]), 1, 2)) AS
  NbChgt,
2 MIN([Temps]) AS TempsMin,
3 MAX([Temps]) AS TempsMax,
4 Avg([Temps]) AS TempsMoyen
5 FROM Trajets
6 GROUP BY IIF(IsNull([Chgt1]), 0, IIF(IsNull([Chgt2]), 1, 2)) ;
```

Avec cette requête, la création de l'état se fait directement via l'outil Création d'état dans *Access* : et en sélectionnant un graphique puis comme source de données la *requête*. En abscisse nous avons mis le nombre de correspondances tandis qu'en ordonnée se trouve le minimum de temps nécessaire, le maximum, et enfin la moyenne.

5) Requête Avec3Chgt

Par la suite, nous avons aussi souhaité codé la *requête* permettant de faire 3 changements. Elle repose sur les mêmes bases que celle à 2 changements mais une jointure de plus est nécessaire. Enfin, c'est surtout la clause WHERE qui est embêtante à gérer : tous les cas doivent être envisagés pour ne pas repasser deux fois par une même gare. Voici le code final de notre *requête* :

```
1   SELECT
2
3   LignesDépart.GareDépart,
4   LignesDépart.ligne AS L0,
5   Corresp.gare AS Chgt1,
6   Corresp.LigneAprès AS L1,
7   Corresp_1.gare AS Chgt2,
8   Corresp_1.LigneAprès AS L2,
9   Corresp_2.gare AS Chgt3,
10  Corresp_2.LigneAprès AS L3,
11  LignesArrivée.GareArrivée,
12
13  ABS(Corresp.TempsDepuisTerminusAvant - LignesDépart.temps)
14  + ABS(Corresp_1.TempsDepuisTerminusAvant - Corresp.TempsDepuisTerminusApres)
15  + ABS(Corresp_2.TempsDepuisTerminusAvant - Corresp_1.TempsDepuisTerminusApres)
16  + ABS(LignesArrivée.temps - Corresp_2.TempsDepuisTerminusApres) AS temps
17
18  FROM (((LignesDépart
19  INNER JOIN Corresp ON LignesDépart.ligne = Corresp.LigneAvant)
20  INNER JOIN Corresp AS Corresp_1 ON Corresp.LigneAprès = Corresp_1.LigneAvant)
21  INNER JOIN Corresp AS Corresp_2 ON Corresp_1.LigneAprès = Corresp_2.LigneAvant)
22  INNER JOIN LignesArrivée ON Corresp_2.LigneAprès = LignesArrivée.ligne
23
24  WHERE (Corresp.gare <> LignesDépart.GareDépart)
25      AND (Corresp_1.gare <> Corresp.gare)
26      AND (Corresp_1.gare <> LignesDépart.GareDépart)
27      AND (Corresp_2.gare <> Corresp_1.gare)
28      AND (Corresp_2.gare <> Corresp.gare)
29      AND (Corresp_2.gare <> LignesDépart.GareDépart)
30      AND (LignesArrivée.GareArrivée <> Corresp_2.gare)
31      AND (LignesArrivée.GareArrivée <> Corresp_1.gare)
32      AND (LignesArrivée.GareArrivée <> Corresp.gare) ;
```

A l'issue de celle-ci, nous aurions pu (comme auparavant) créer une *requête* d'ajout et l'ajouter au tableau des trajets possibles etc.

6) État graphique : top 10 des trajets les plus courts (et requête associée)

Afin d'aider l'utilisateur dans son choix, nous avons souhaité mettre en place une visualisation claire des 10 trajets les plus courts. Pour ce faire, nous avons commencé par créer une requête dédiée, permettant de regrouper l'ensemble des informations nécessaires dans une seule table de travail. Nous avions besoin du temps de trajet, ainsi que du trajet lui même sous forme de chaîne de caractères. Cette requête sélectionne donc les trajets en les classant par temps croissant, et limite l'affichage aux dix durées les plus faibles. Elle inclut notamment un *champ textuel* décrivant le trajet, intégrant les gares de départ, de changement et d'arrivée, afin de disposer d'un libellé exploitable pour l'axe des ordonnées du graphique.

Nous n'allons pas réexpliquer exactement les détails des IIF afin de créer le libellé car le procédé est exactement le même que celui utilisé en source de contrôle, afin d'afficher une phrase décrivant le trajet dans notre formulaire Guide. En revanche, nous porterons une attention particulière au fait que nous ne voulions sélectionner que les 10 premiers trajets les plus courts. Pour cela une aide a été nécessaire : nous avons trouvé de l'aide ici : *How to select top 10 in Access query? - Stack Overflow* et ici pour mieux comprendre que sans la clause ORDER BY c'est un échantillon quelconque de 10 enregistrements qui aurait été produit : *Prédicats ALL, DISTINCT, DISTINCTROW, TOP - Support Microsoft*.

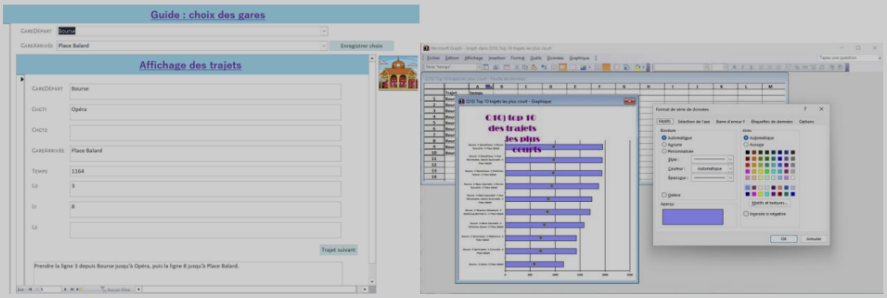
```
1   SELECT TOP 10
2
3   IFF(
4       IsNull(Chgt1),
5       GareDépart & " → " & GareArrivée,
6       IFF(
7           IsNull(Chgt2),
8           GareDépart & " → " & Chgt1 " → " & GareArrivée,
9           GareDépart & " → " & Chgt1 " → " & Chgt2 & " → " &
    GareArrivée
10      )
11  ) AS Trajet,
12
13  Trajet.temps
14  FROM Trajets
15  ORDER BY Trajets.temps;
```

Enfin, afin d'améliorer la lisibilité du graphique, plusieurs ajustements ont été réalisés. Les paramètres liés à l'orientation du texte, à la police, à la taille des caractères et aux couleurs ne pouvant pas être modifiés directement depuis les propriétés classiques, nous avons accédé aux propriétés avancées du graphique en double-cliquant sur celui-ci. Ces réglages ont permis d'optimiser l'affichage des libellés des trajets.

7) Amélioration des visuels

Par ailleurs, nous avons travaillé sur l'aspect visuel de l'interface des *formulaires* : amélioration des intitulés des fonctionnalités pour une meilleure lisibilité, harmonisation des couleurs et retouches esthétiques générales afin d'offrir une présentation plus claire et plus agréable (couleur de fond, police...). On a même ajouté une image de gare dans le formulaire Guide[1].

a. [1] Il suffit de la coller l'image depuis le presse papier (*Ctrl c + Ctrl v*).



Pour modifier l'aspect graphique des états, l'on a suivi le chemin suivant :

Clic droit $\longrightarrow$ *Objet Graphique* $\longrightarrow$ *Ouvrir* $\longrightarrow$ *Format Zone de traçage/Titre/...*

De même, pour les *états* nous avons aussi pris soin d'affiner l'aspect esthétique : couleurs, polices, orientation des libellés, e.t.c

Pour changer la couleur de fond des états, l'on a suivi le chemin suivant :

Clic Droit $\longrightarrow$ *Couleur d'arrière Plan/Remplissage*

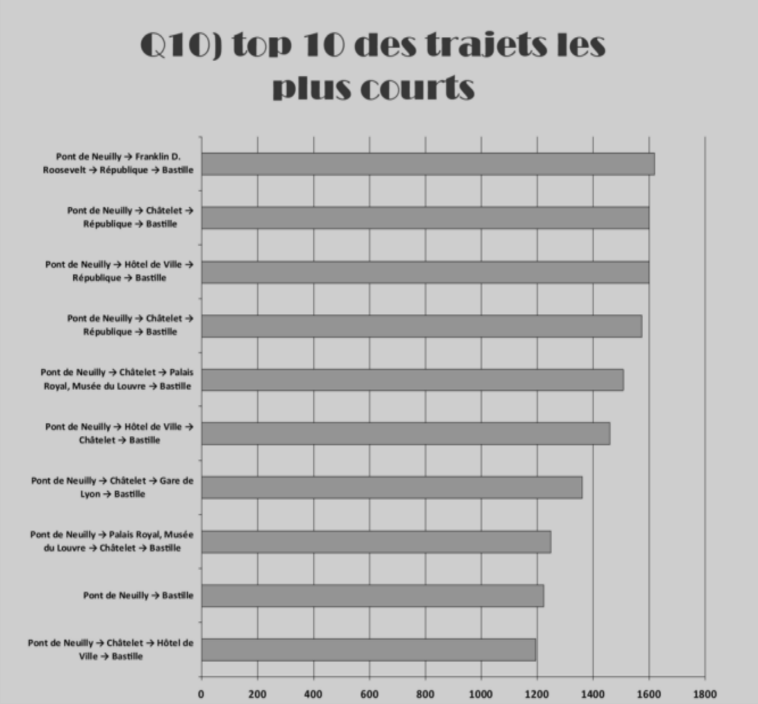
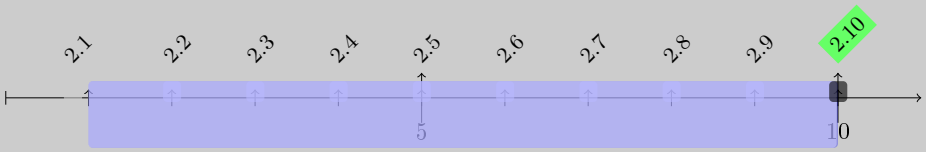


FIGURE 6 – *Exemple* : Les 10 trajets les plus courts entre **Pont de Neuilly** et **Bastille**



(FIN) Progression

3.2 Commentaires

Par soucis d'esthétisme, nous avons veillé à rendre nos *requêtes* aussi lisibles et aérées que possible.
Néanmoins, indépendamment des efforts qui ont été déployés, nous avons constaté que, même après enregistrement, la mise en forme visuelle est automatiquement réinitialisée lors de l'affichage du code *SQL*.

Le lecteur avisé, notera par ailleurs que nous ne nous sommes que très rarement (si ce n'est jamais) attarder sur la *complexité machine et ou temporelle* de notre code.
Et pour cause : l'architecture *SQL* nous est pour ainsi dire trop opaque pour nous permettre un tel niveau d'analyse. En outre la base de données était suffisamment légère pour que nous n'ayons pas été importunés par des questions de *volumétrie*.

FIN ! ! !

3.3 Annexes

Références

[0] : *Création de tables.*
[1] : *Insertions dans une table.*
[2] : *Opérateurs arithmétique.*
[3] : *Suppression d'enregistrements.*
[4] : *Ajouter un contrôle Zone de texte à un formulaire ou un état.*
[5] : *Utiliser la zone Générateur d'expressions.*
[6] : *Fonction IsNull.*
[7] : *MS Access Concat With &.*
[8] : *Prédicats ALL, DISTINCT, DISTINCTROW, TOP.*
[9] : *How to select top* 10 *in Access query ?*
[10] :

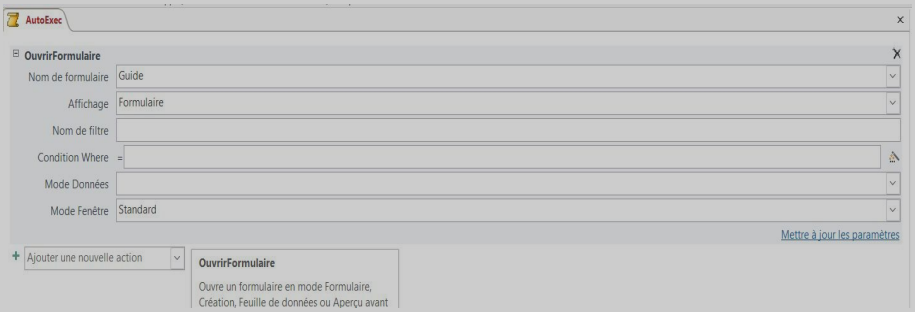


FIGURE 7 – Auto Exec

Contacts