Q1) Pour importer les données dans Access, nous avions à disposition un fichier externe au format CSV, nous avons donc été dans la rubrique d'importation externe. Nous avons remarqué que la première ligne du fichier contient les en-têtes des colonnes donc nous devons cocher une case en faveur de la présence d'en-tête lors de l'importation. Les données sont séparées par des points-virgules.
Lors de l'importation, il est important de définir correctement les types de champs : le champ ligne a été défini comme texte court (cf. capture d'écran ci-dessous), car certaines valeurs contiennent des lettres ou des caractères spéciaux (par exemple 7bis) ; le champ temps a été défini comme réel double pour représenter les durées de parcours ; enfin, le champ gare a été défini comme texte court pour stocker les noms des gares.

ligne	temps	gare
1	0.00	Grande Arche de la Défense
1	56.00	Esplanade de la Défense
1	109.00	Pont de Neuilly
1	166.00	Les Sablons
1	223.00	Porte Maillot
1	293.00	Argentine
1	378.00	Charles de Gaulle, Étoile
1	469.00	George V
1	559.00	Franklin D. Roosevelt
1	652.00	Champs Élysées, Clémenceau
1	735.00	Concorde
1	821.00	Tuileries
1	879.00	Palais Royal, Musée du Louvre
1	931.00	Louvre, Rivoli

(Si on fait du Latex je pense que c'est bien de laisser ça en capture d'écran)

Q2) Pour la création des tables, nous avions deux possibilités. Dans un premier temps, nous avons choisi de créer la table à la main. Dans ce cas, il est nécessaire de définir le type de chaque champ dans le mode création, et nous avons veillé à typer les deux champs en texte court afin de stocker correctement les noms de gares. Nous avons ensuite entrer l'enregistrement à la main
Par la suite, nous avons également exploré une autre approche : créer la table directement à l'aide d'une requête SQL, ce qui nous a permis d'automatiser la création des champs avec les types appropriés, en nous appuyant sur des informations trouvées sur Internet : ici pour la création de

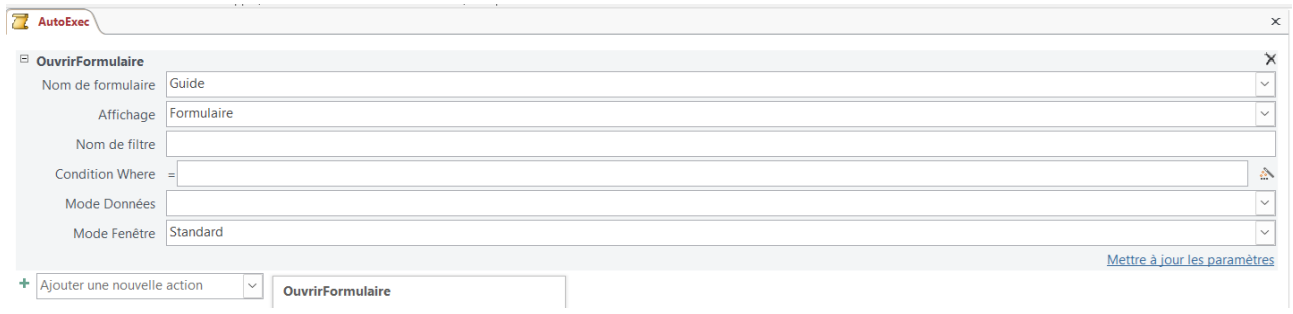


table : SQL CREATE TABLE Statement et ici pour ajouter un enregistrement : SQL INSERT INTO Statement

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

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

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 (cf.capture écran), afin que chaque gare n'apparaisse qu'une seule fois dans la liste déroulante.

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];

(Idem si on fait du Latex je pense que c'est bien de laisser ça en capture d'écran pour bien expliquer où ça se situe et ce qu'on a modifié)

Q3) La requête LignesDépart permet d'obtenir la liste des lignes passant par la gare de départ. Pour cela, nous avons utilisé l'assistant de requête afin de créer une jointure entre la table Choix et la table Lignes, en associant le champ gare de la table Lignes avec le champ GareDépart de la table Choix. Cette jointure garantit que seules les lignes correspondant à la gare de départ sélectionnée sont affichées. 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.

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

De manière similaire, la requête LignesArrivée liste les lignes passant par la gare d'arrivée. La méthode est identique : on crée une jointure entre Gare et GareArrivée et on relance la requête après tout changement de gare d'arrivée pour obtenir les lignes correspondantes.

Chaque requête renvoie trois colonnes : la gare concernée (GareDépart ou GareArrivée), le nom de la ligne et le temps d'accès depuis le terminus.

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

Q4) Pour cela, nous avons utilisé l'assistant de requête afin de créer une requête simple basée sur nos deux requêtes précédentes, LignesDépart et LignesArrivée. 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.

```
SELECT
LignesDépart.GareDépart,
LignesDépart.ligne AS L0,
LignesArrivée.GareArrivée,
Abs(LignesDépart.temps - LignesArrivée.temps) AS Temps

FROM LignesArrivée INNER JOIN LignesDépart
ON LignesArrivée.ligne = LignesDépart.ligne;
```

Nous avons aussi voulu vérifier dans le cas où plusieurs lignes sont possibles le résultat (vérifiant que notre requête sélectionne bien tous les cas possible) par exemple pour le cas : Bastille - Concorde :

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

Q5) Pour cela, nous avons utilisé deux instances de la table Lignes, nommées Lignes_1 et Lignes_2, qui sont simplement deux copies de la même table. Nous avons effectué une jointure entre ces deux instances, avec comme critères :

- la gare doit être commune aux deux lignes (condition de JOINTURE)
- les lignes doivent être distinctes, afin d'éviter qu'une ligne ne soit mise en correspondance avec elle-même (clause WHERE)

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

Enfin, pour chaque correspondance, nous avons également ajouté les temps d'accès à la gare commune depuis les terminus de chaque ligne (par simple ajout dans le SELECT), afin de connaître la durée nécessaire pour atteindre cette gare avant et après le changement.

```
SELECT

Lignes_1.ligne AS LigneAvant,
Lignes_2.ligne AS LigneAprès,
Lignes_1.gare AS gare,

Lignes_1.temps AS TempsDepuisTerminusAvant,
Lignes_2.temps AS TempsDepuisTerminusApres

FROM Lignes AS Lignes_1 INNER JOIN  Lignes AS Lignes_2

ON  Lignes_1.gare = Lignes_2.gare

WHERE Lignes_1.ligne <> Lignes_2.ligne;
```

Q6) Notre approche a été la suivante : pour chaque gare de départ, nous commençons par identifier les lignes accessibles grâce à la requête LignesDépart. Ensuite, nous utilisons la requête Corresp pour déterminer les gares de correspondance possibles, c'est-à-dire les gares où un changement de ligne peut s'effectuer. Enfin, nous vérifions que la gare d'arrivée se situe sur l'une des lignes listées par LignesArrivée.
Ainsi, la requête repose sur deux jointures successives : la première entre LignesDépart et Corresp pour identifier la première partie du trajet et le changement, la deuxième entre Corresp et LignesArrivée pour atteindre la gare finale.

Le champ temps correspond au temps total du parcours et est calculé selon le même principe que précédemment : 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.

Il fallait ici prêter attention à la clause WHERE pour s'assurer que les gares successives sont toutes distinctes i.e. GareDépart différente de celle de correspondance et celle de correspondance différent de GareArrivée.

```
SELECT

LignesDépart.GareDépart,
LignesDépart.ligne AS L0,

Corresp.gare AS Chgt1,

Corresp.LigneAprès AS L1,
LignesArrivée.GareArrivée,

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


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


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

Pour les trajets comportant deux changements, nous avons suivi une logique similaire à celle des trajets avec un seul changement : partir d'une gare de départ, effectuer une première correspondance, puis une deuxième, avant d'atteindre la gare d'arrivée. Pour gérer cette deuxième

correspondance, nous avons créé une copie de la requête Corresp, de la même manière que nous avions créé des copies de la table Lignes pour les correspondances précédentes.

Dans un premier temps, nous avions envisagé de réutiliser la requête Avec1Chgt pour déterminer le premier changement, mais nous avons rapidement constaté que cela serait trop restrictif. En effet, avec deux changements, il existe davantage de chemins possibles, et limiter le premier changement aux trajets déjà identifiés dans Avec1Chgt exclurait certaines solutions.

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 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 Gare1 → Gare2 → Gare1, c'est-à-dire revenir au point de départ après un premier changement. 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

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

Q7) Création de table comme auparavant avec la même aide :

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

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.

```
INSERT INTO Trajets ( GareDépart, GareArrivée, Temps, L0 )

SELECT
Avec0Chgt.GareDépart,
Avec0Chgt.GareArrivée,
Avec0Chgt.temps,
Avec0Chgt.L0

FROM Avec0Chgt

ORDER BY Avec0Chgt.temps;
```

```
INSERT INTO Trajets (GareDépart, Chgt1, GareArrivée, Temps, L0, L1)

SELECT
Avec1Chgt.GareDépart,
Avec1Chgt.Chgt1,
Avec1Chgt.GareArrivée,
Avec1Chgt.temps,
Avec1Chgt.L0,
Avec1Chgt.L1

FROM Avec1Chgt

ORDER BY Avec1Chgt.temps
;
```

Pour

```
INSERT INTO Trajets (GareDépart, Chgt1, Chgt2, GareArrivée, Temps, L0, L1, L2)

SELECT
Avec2Chgt.GareDépart,
Avec2Chgt.Chgt1,
Avec2Chgt.Chgt2,
Avec2Chgt.GareArrivée,
Avec2Chgt.temps,
Avec2Chgt.L0,
Avec2Chgt.L1,
Avec2Chgt.L2

FROM Avec2Chgt

ORDER BY Avec2Chgt.temps
;
```

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.

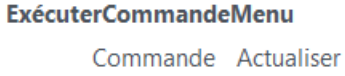
```
DELETE FROM Trajets;
```

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

Q8) – 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.

Q9) Pour cette question, nous avons d'abord créé un formulaire fondé sur la table Trajets, puis nous l'avons inséré en tant que sous-formulaire dans le formulaire Guide par un simple glisser-déposer en mode Création. Cela permet d'afficher directement, dans le même écran, l'ensemble des trajets possibles correspondant au choix de l'utilisateur.

Nous avons également ajouté un bouton de navigation afin de parcourir facilement les différents trajets listés dans ce sous-formulaire. Par ailleurs, le bouton principal du formulaire Guide a été complété :



nous avons intégré une action Actualiser (cf ci-contre) pour que le sous-formulaire se mette automatiquement à jour après le choix de l'utilisateur.

Enfin, nous avons créé une macro AutoExec qui s'exécute automatiquement au démarrage et ouvre directement le formulaire Guide. Cette configuration permet à l'utilisateur d'accéder immédiatement à l'interface de sélection et de visualisation des trajets dès l'ouverture de la base. (cf ci-dessous)

Q10) Pour cette dernière question, nous avons apporté plusieurs améliorations destinées à améliorer la qualité générale de l'interface et l'expérience utilisateur.

1) Affinage de certaines requêtes

Tout d'abord, nous avons affiné certaines requêtes, notamment celle développée en Q6, afin de garantir une production cohérente des résultats (pas de trajet de type 1→2→1 comme expliqué auparavant).

2) Enrichissement du formulaire

Nous avons également enrichi le formulaire conçu en Q9 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 (Q10), nous avons ajouté sous le sous-formulaire Trajets 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 plutôt été 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 :

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

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) Etat 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 Chg1 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.

```
SELECT IIf(IsNull([Chgt1]), 0, IIf(IsNull([Chgt2]), 1, 2)) AS NbChgt,
Min([Temps]) AS TempsMin,
Max([Temps]) AS TempsMax,
Avg([Temps]) AS TempsMoyen
FROM Trajets
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 nécessaires. 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 :

```
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,
Corresp_2.gare AS Chgt3,
Corresp_2.LigneAprès AS L3,
LignesArrivée.GareArrivée,

Abs(Corresp.TempsDepuisTerminusAvant - LignesDépart.temps)
 + Abs(Corresp_1.TempsDepuisTerminusAvant - Corresp.TempsDepuisTerminusApres)
 + Abs(Corresp_2.TempsDepuisTerminusAvant - Corresp_1.TempsDepuisTerminusApres)
 + Abs(LignesArrivée.temps - Corresp_2.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 Corresp AS Corresp_2 ON Corresp_1.LigneAprès = Corresp_2.LigneAvant)
INNER JOIN LignesArrivée ON Corresp_2.LigneAprès = LignesArrivée.ligne


WHERE (Corresp.gare <> LignesDépart.GareDépart)
   AND (Corresp_1.gare <> Corresp.gare)
   AND (Corresp_1.gare <> LignesDépart.GareDépart)
   AND (Corresp_2.gare <> Corresp_1.gare)
   AND (Corresp_2.gare <> Corresp.gare)
   AND (Corresp_2.gare <> LignesDépart.GareDépart)
   AND (LignesArrivée.GareArrivée <> Corresp_2.gare)
   AND (LignesArrivée.GareArrivée <> Corresp_1.gare)
   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) Etat 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 cela, 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 rééxpliqué 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. (Cf, requête ci dessous).

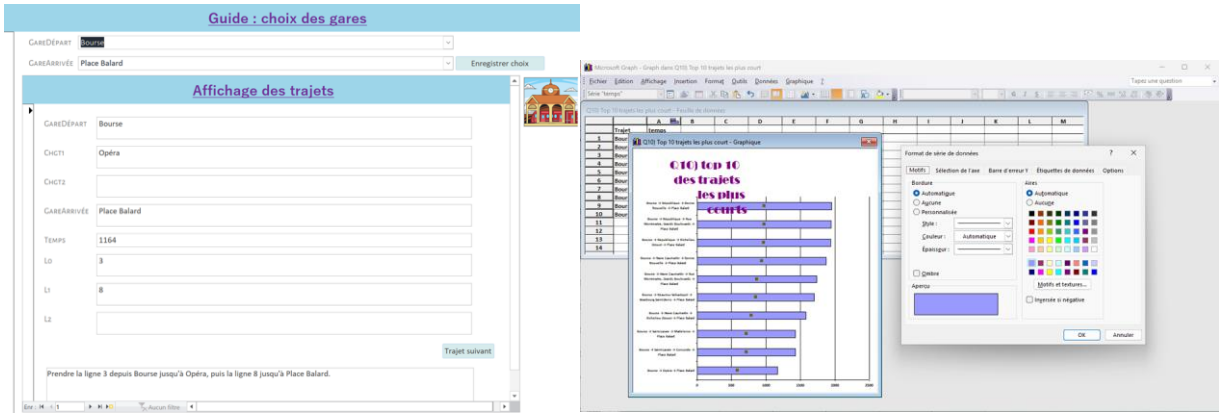
afin

```
SELECT TOP 10

IIf(
      IsNull(Chgt1),
      GareDépart & " → " &  GareArrivée,
      IIf(
         IsNull(Chgt2),
         GareDépart & " → " & Chgt1 & " → " & GareArrivée,
         GareDépart & " → " & Chgt1 & " → " & Chgt2 & " → " & GareArrivée
      )
   ) AS Trajet,

Trajets.temps

FROM Trajets
ORDER BY Trajets.temps;
```

Enfin, 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

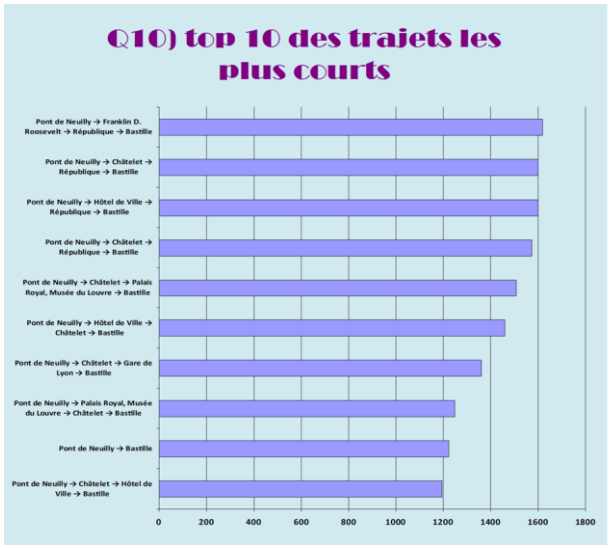
Par ailleurs, nous avons travaillé sur l'aspect visuel de l'interface des formulaires : amélioration des intitulés des fonctionnalités pour une meilleure lisibilité, harmonisation des couleurs et retouches esthétiques générales afin d'offrir une présentation plus claire et plus agréable (couleur de fond,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



De même, pour les états nous avons aussi pris soin d'affiner l'aspect esthétique : couleurs, polices et orientation des libellés.

Pour modifier l'aspect graphique des états, on doit faire : Clic droit->Objet Graphique -> Ouvrir -> Format Zone de traçage/Titre/…. Pour changer la couleur de fond des états, on fait Clic Droit -> Couleur d'arrière Plan/Remplissage.

Pour l'exemple entre Pont de Neuilly et Bastille les 10 trajets les plus courts sont :



Enfin, nous avons veillé à rendre nos requêtes aussi lisibles et aérées que possible, comme cela est visible dans le rendu présenté. Néanmoins, indépendamment de cette volonté, nous avons constaté que, même après enregistrement, la mise en forme visuelle est automatiquement réinitialisée lors de l'affichage du code SQL.