Compte rendu TP 2 : MongoDB

Antonin Durand

William Zehren

INF3

Table des matières

[Demande 1 : 3](#_Toc150466302)

[Demande 2 : 5](#_Toc150466303)

[Demande 3 : 6](#_Toc150466304)

# Demande 1 :

*Proposer une organisation des données du premier TP sous la forme d’une base de données MongoDB. L’organisation vise à permettre l’évaluation efficace des requêtes données dans le premier TP. Il faut définir :*

1. *Les collections*
2. *Pour chaque collection, expliquer la structure des documents (Objet JSON). Donner un exemple*

Avant de commencer à réfléchir à la structure de notre base de données MongoDB, nous avons étudié les tables SQL que nous avions à notre disposition.

Nous avons donc interprété la fonction de chaque table dans la base de données de Ile De France Mobilités :

**small\_arrets\_lignes** 🡪 Chaque tuple correspond à un arrêt/station, avec les informations de ce dernier (longitude, latitude, ville, code postal)

**small\_routes** 🡪 Chaque tuple correspond à une ligne avec son nom

**small\_trips** 🡪 Chaque tuple correspond à un train dans une ligne (C, H, ...) mais à des horaires différents

**small\_stop\_times** 🡪 Chaque tuple correspond à un arrêt pour un train donné à un horaire donné. On a aussi le numéro de l'arrêt dans le trajet du train (1 pour le départ, 20 pour le terminus par exemple)

Nous avons également interprété l’utilité des champs les plus importants retrouvés dans chaque table :

**route\_id** 🡪 id d’une ligne (B, C, …)

**trip\_id** 🡪 id du train, correspond à un des trains en fonction de son heure de départ

**trip\_headsign** 🡪 correspond à l’identifiant d’un train (défini par IDFM)

**stop\_id** 🡪 id d’un arrêt

**stop\_sequence** 🡪 ordre d’arrivée d’un train aux arrets de son trajet

Après réalisé cette étude sur les tables de la base de données d’origine, nous avons pu imaginer une structure pour implémenter les données dans une base de données MongoDB.

Nous avons donc choisi de définir deux collections pour former notre base de données MongoDB :

**CollectionTrips** :

{

"route\_id" = "IDFM:C01727",

"trip\_id" = "IDFM:TN:SNCF:28bf25f5-5248-48be-96b0-c62b689c4543",

"trip\_headsign" : "MONA",

"listeArrets" : [

{

"stop\_id" : "IDFM:monomodalStopPlace:44411",

"stop\_name" : "Pontoise",

"stop\_lon" : "2.0957859968423524",

"stop\_lat" : "49.04673337596684",

"OperatorName" : "SNCF",

"Nom\_commune" : "Pontoise",

"Code\_insee" : "95500",

"arrival\_time" : "12:58:50",

"departure\_time" : "12:58:50",

},

{

"stop\_id" : "IDFM:monomodalStopPlace:47599",

"stop\_name" : "Saint-Ouen-l'Aumône 2.105230857870794",

"stop\_lon" : "2.105230857870794",

"stop\_lat" : "49.04530690087292",

"OperatorName" : "SNCF",

"Nom\_commune" : "Saint-Ouen-l'Aumône",

"Code\_insee" : "95572",

"arrival\_time" : "13:00:20",

"departure\_time" : "13:01:00",

}

]

}

Cette collection servira à déterminer le trajet emprunté par un train dans le bon ordre d’arrêt, cela en fonction de sa ligne (C, B, …), de son arrêt de départ et d’arrivée et de son horaire de départ/arrivée.

On stocke également les diverses informations liées à chaque arrêt.

**CollectionRoutes:**

{

"route\_id" : "IDFM:C01727",

"route\_short\_name" : "C",

"route\_long\_name" "C":

}

Cette collection servira à stocker les identifiants des différentes lignes pour les réutiliser dans notre CollectionTrips.

# Demande 2 :

*Créer des scripts pour restructurer les données du TP2 conformément à la structure de document (Objet JSON) choisie. Les scripts peuvent être définis dans un langage choisi librement.*

# Demande 3 :

*Exprimer les requêtes données dans le premier TP sous la forme des requêtes MongoDB.*

**Requête 1: Étant donnée une ligne X (par exemple, le RER B), quelles sont ses stations?**

db.CollectionTrips.distinct("listeArrets.stop\_name", { "route\_id": "IDFM:C01727" })

**Requête 2: Quelles sont toutes les stations situées au sud (géographiquement) d'une station X donnée?**

var stationName = "Massy-Palaiseau";

var latitudeX = db.CollectionTrips.findOne({

"listeArrets.stop\_name": stationName

}, {

"listeArrets.stop\_lat": 1,

"\_id": 0

}).listeArrets[0].stop\_lat;

db.CollectionTrips.find({

"listeArrets.stop\_lat": { $lt: latitudeX }

}, {

"listeArrets.stop\_name": 1,

"\_id": 0

})

**Requête 3: Quelles sont toutes les stations au sud d'une station X donnée et qui sont desservies par le même train?**

var stationName = "Pontoise";

var trainId = "IDFM:TN:SNCF:28bf25f5-5248-48be-96b0-c62b689c4543"; // Remplacer par l’id du train

var latitudeX = db.CollectionTrips.findOne({

"listeArrets.stop\_name": stationName,

"trip\_id": trainId

}, {

"listeArrets.stop\_lat": 1,

"\_id": 0

}).listeArrets[0].stop\_lat;

db.CollectionTrips.find({

"trip\_id": trainId,

"listeArrets.stop\_lat": { $lt: latitudeX }

}, {

"listeArrets.stop\_name": 1,

"\_id": 0

})

**Requête 4: Étant donné deux stations X et Y, y a-t-il une connexion directe de X à Y (c'est une question non triviale, prenez par exemple les stations Robinson et Massy-Palaiseau)?**

var stationX = "Robinson";

var stationY = "Massy-Palaiseau";

var routeId = "B";

// Trouver les trajets qui desservent la station X

var tripsForStationX = db.CollectionTrips.find({

"route\_id": routeId,

"listeArrets.stop\_name": stationX

}, {

"trip\_id": 1,

"\_id": 0

}).toArray().map(item => item.trip\_id);

// Vérifier si la station Y est également desservie par les mêmes trajets et qu'elle est située après la station X

var connectionExists = db.CollectionTrips.findOne({

"route\_id": routeId,

"trip\_id": { $in: tripsForStationX },

"listeArrets.stop\_name": stationY,

"listeArrets.stop\_sequence": { $gt: db.CollectionTrips.findOne({

"route\_id": routeId,

"trip\_id": { $in: tripsForStationX },

"listeArrets.stop\_name": stationX

}).listeArrets[0].stop\_sequence }

}, {

"\_id": 0

});

print(connectionExists ? "Connexion directe trouvée." : "Pas de connexion directe trouvée.");

**Requête 5: Étant donné deux stations X et Y de la même ligne, combien de stations les séparent? (Attention: requête imprécise)**

var stationX = "Robinson";

var stationY = "Massy-Palaiseau";

var routeId = "B";

// Trouver les trajets qui desservent les stations X et Y

var tripsForStations = db.CollectionTrips.find({

"route\_id": routeId,

"listeArrets.stop\_name": { $in: [stationX, stationY] }

}, {

"trip\_id": 1,

"listeArrets": 1,

"\_id": 0

}).toArray();

// Extraire les informations sur les arrêts des stations X et Y

var stopsInfo = {};

tripsForStations.forEach(function (trip) {

trip.listeArrets.forEach(function (stop) {

stopsInfo[stop.stop\_name] = stop;

});

});

// Trouver l'index de chaque station dans la liste des arrêts

var indexX = stopsInfo[stationX].stop\_sequence;

var indexY = stopsInfo[stationY].stop\_sequence;

// Calculer le nombre de stations entre X et Y

var stationsSeparation = Math.abs(indexX - indexY);

print("Le nombre de stations entre " + stationX + " et " + stationY + " est : " + stationsSeparation);

**Requête 6: Étant donné deux stations X et Y pour lesquelles il existe une connexion directe, quelle est la durée minimale d'un trajet de X à Y?**

var stationX = "Robinson";

var stationY = "Massy-Palaiseau";

var routeId = "B";

// Trouver les trajets qui desservent les stations X et Y

var tripsForStations = db.CollectionTrips.find({

"route\_id": routeId,

"listeArrets.stop\_name": { $in: [stationX, stationY] }

}, {

"trip\_id": 1,

"listeArrets": 1,

"\_id": 0

}).toArray();

// Extraire les informations sur les arrêts des stations X et Y

var stopsInfo = {};

tripsForStations.forEach(function (trip) {

trip.listeArrets.forEach(function (stop) {

stopsInfo[stop.stop\_name] = stop;

});

});

// Trouver tous les trajets directs entre X et Y

var directConnections = [];

tripsForStations.forEach(function (trip) {

var indexX = trip.listeArrets.findIndex(stop => stop.stop\_name === stationX);

var indexY = trip.listeArrets.findIndex(stop => stop.stop\_name === stationY);

if (indexX !== -1 && indexY !== -1 && indexX < indexY) {

directConnections.push({

"trip\_id": trip.trip\_id,

"departure\_time\_X": trip.listeArrets[indexX].departure\_time,

"arrival\_time\_Y": trip.listeArrets[indexY].arrival\_time

});

}

});

// Trouver la durée minimale d'un trajet entre X et Y

var minDuration = Number.MAX\_SAFE\_INTEGER;

var bestConnection = null;

directConnections.forEach(function (connection) {

var departureTimeX = new Date("1970-01-01T" + connection.departure\_time\_X + "Z");

var arrivalTimeY = new Date("1970-01-01T" + connection.arrival\_time\_Y + "Z");

var duration = arrivalTimeY - departureTimeX;

if (duration < minDuration) {

minDuration = duration;

bestConnection = connection;

}

});

print("La durée minimale d'un trajet entre " + stationX + " et " + stationY + " est : " + (minDuration / (1000 \* 60)) + " minutes");

print("La meilleure connexion directe est avec le train ID : " + bestConnection.trip\_id);

**Requête 7: Étant donné deux stations X et Y pour lesquelles il n'existe pas de connexion directe, y a-t-il une connexion avec un seul changement entre les deux stations?**

var stationX = "Robinson";

var stationY = "Massy-Palaiseau";

var routeId = "B";

// Trouver tous les trajets qui desservent la station X

var tripsForStationX = db.CollectionTrips.find({

"route\_id": routeId,

"listeArrets.stop\_name": stationX

}, {

"trip\_id": 1,

"listeArrets": 1,

"\_id": 0

}).toArray();

// Trouver tous les trajets qui desservent la station Y

var tripsForStationY = db.CollectionTrips.find({

"route\_id": routeId,

"listeArrets.stop\_name": stationY

}, {

"trip\_id": 1,

"listeArrets": 1,

"\_id": 0

}).toArray();

// Trouver les connexions avec un seul changement

var connectionsWithOneChange = [];

tripsForStationX.forEach(function (tripX) {

tripsForStationY.forEach(function (tripY) {

// Trouver l'intersection des arrêts des trajets X et Y

var commonStops = tripX.listeArrets.filter(stopX =>

tripY.listeArrets.some(stopY => stopY.stop\_name === stopX.stop\_name)

);

// S'il y a une intersection, il peut y avoir une connexion avec un seul changement

if (commonStops.length > 0) {

commonStops.forEach(function (commonStop) {

connectionsWithOneChange.push({

"trip\_id\_X": tripX.trip\_id,

"departure\_time\_X": commonStop.departure\_time,

"trip\_id\_Y": tripY.trip\_id,

"arrival\_time\_Y": commonStop.arrival\_time

});

});

}

});

});

// Afficher les connexions avec un seul changement

if (connectionsWithOneChange.length > 0) {

print("Connexions avec un seul changement entre " + stationX + " et " + stationY + ":");

connectionsWithOneChange.forEach(function (connection) {

print(" - Prendre le train ID " + connection.trip\_id\_X + " à " + stationX +

" à " + connection.departure\_time\_X + ", puis prendre le train ID " +

connection.trip\_id\_Y + " à " + connection.arrival\_time\_Y);

});

} else {

print("Pas de connexion avec un seul changement entre " + stationX + " et " + stationY + ".");

}

**Requête 8: Combien de changements de ligne sont nécessaires pour aller de la station X à la station Y?**

var stationX = "Robinson";

var stationY = "Massy-Palaiseau";

// Trouver tous les trajets qui desservent la station X

var tripsForStationX = db.CollectionTrips.find({

"listeArrets.stop\_name": stationX

}, {

"trip\_id": 1,

"route\_id": 1,

"\_id": 0

}).toArray();

// Trouver tous les trajets qui desservent la station Y

var tripsForStationY = db.CollectionTrips.find({

"listeArrets.stop\_name": stationY

}, {

"trip\_id": 1,

"route\_id": 1,

"\_id": 0

}).toArray();

// Trouver les identifiants uniques de lignes pour les trajets partant de X

var uniqueRoutesForStationX = new Set(tripsForStationX.map(trip => trip.route\_id));

// Trouver les identifiants uniques de lignes pour les trajets arrivant à Y

var uniqueRoutesForStationY = new Set(tripsForStationY.map(trip => trip.route\_id));

// Trouver les lignes communes entre X et Y

var commonRoutes = Array.from(new Set([...uniqueRoutesForStationX].filter(routeX => uniqueRoutesForStationY.has(routeX))));

// Afficher le nombre de changements de ligne nécessaires

var numberOfChanges = commonRoutes.length > 0 ? 0 : 1; // Si des lignes communes existent, aucun changement n'est nécessaire

print("Le nombre de changements de ligne nécessaires pour aller de " + stationX + " à " + stationY + " est : " + numberOfChanges);

**Requête 9: Quel est le nombre maximal de changements pour un trajet entre deux stations? (pire cas)**

var stationX = "Robinson";

var stationY = "Massy-Palaiseau";

var routeId = "B";

// Fonction récursive pour trouver les connexions avec un nombre maximal de changements

function findConnections(currentStation, currentChanges, maxChanges, visitedStations, currentPath) {

if (currentStation === stationY && currentChanges <= maxChanges) {

print("Trajet trouvé avec " + currentChanges + " changements : " + currentPath.join(" -> "));

return;

}

if (currentChanges > maxChanges) {

return;

}

visitedStations.add(currentStation);

db.CollectionTrips.find({

"route\_id": routeId,

"listeArrets.stop\_name": currentStation

}, {

"trip\_id": 1,

"listeArrets": 1,

"\_id": 0

}).forEach(function (trip) {

trip.listeArrets.forEach(function (stop) {

if (!visitedStations.has(stop.stop\_name)) {

var newPath = currentPath.slice();

newPath.push(stop.stop\_name);

findConnections(stop.stop\_name, currentChanges + 1, maxChanges, new Set(visitedStations), newPath);

}

});

});

}

// Appel initial à la fonction

findConnections(stationX, 0, 10, new Set(), [stationX]);

**Requête 10: Quelles sont les stations X et Y pour lesquelles le trajet le plus simple nécessite un nombre maximal de changements?**

var maxChanges = 10; // Nombre maximal de changements autorisés

var allStations = db.CollectionTrips.distinct("listeArrets.stop\_name");

var pairsWithMaxChanges = [];

// Parcourir toutes les paires de stations

for (var i = 0; i < allStations.length - 1; i++) {

for (var j = i + 1; j < allStations.length; j++) {

var stationX = allStations[i];

var stationY = allStations[j];

// Fonction récursive pour trouver les connexions avec un nombre maximal de changements

function findConnections(currentStation, currentChanges, visitedStations) {

if (currentStation === stationY && currentChanges <= maxChanges) {

return true;

}

if (currentChanges > maxChanges) {

return false;

}

visitedStations.add(currentStation);

var result = false;

db.CollectionTrips.find({

"route\_id": routeId,

"listeArrets.stop\_name": currentStation

}, {

"trip\_id": 1,

"listeArrets": 1,

"\_id": 0

}).forEach(function (trip) {

trip.listeArrets.forEach(function (stop) {

if (!visitedStations.has(stop.stop\_name)) {

result = result || findConnections(stop.stop\_name, currentChanges + 1, new Set(visitedStations));

}

});

});

return result;

}

// Teste si la paire de stations nécessite le nombre maximal de changements

if (findConnections(stationX, 0, maxChanges, new Set())) {

pairsWithMaxChanges.push({ "stationX": stationX, "stationY": stationY });

}

}

}

// Afficher les paires de stations nécessitant le nombre maximal de changements

print("Paires de stations nécessitant le nombre maximal de changements :");

pairsWithMaxChanges.forEach(function (pair) {

print(pair.stationX + " -> " + pair.stationY);

});