Projet : Gestion de données à grande échelle

Yohan Gouzerh - Maxime Turlure

November 2018

Table des matières

1 Description du jeu de données

Notre jeu de données décrit l'ensemble des informations disponibles sur les stations de ski françaises : domaine skiable, hôtels, employés et commerce. Notre jeu contient 4 collections.

La première collection (Station) regroupe des informations de chaque station : nom, massif, prix du forfait, altitude et horaires du domaine. Les pistes sont caractérisées par : leur nom, le niveau de la piste (vert, bleu, rouge, noir), la longueur de la piste, l'altitude min et max de cette piste et enfin un booléen indiquant si la piste possède un éclairage ou non. Chaque station emploie des pisteurs. On dispose d'informations sur les remontées mécaniques : nom, type de remontée, date de mise en service et une référence vers l'employé qui en est responsable.

La seconde collection (Employé) est une collection classique qui regroupe des informations sur les employées : age, nom, prénom, sexe, adresse et date d'embauche.

La troisième collection (Hôtel) est aussi une collection classique sur les caractéristiques des hôtels : leur nom, la station dans laquelle il se trouve, l'adresse, le nombre d'étoiles et les diverses commodités (casier à ski, parking).

La dernière collection (Commerce) contient les informations des commerces de chaque station. On distingue 2 types de commerce : Les fromageries et les restaurants. Pour les fromageries, on a accès aux fromages vendus, pour les restaurants au type de cuisine.

2 Choix des index

Pour la collection Station, nous avons choisi le champ "massif" comme index afin de regrouper les stations par massif. Nous avons pris l'hypothèse que l'on comparera plus souvent des caractéristiques de station se trouvant dans le même massif.

Pour la collection Employé, nous avons choisi comme index le champ "embauche". Il parait fort probable que les requêtes qui seront faites sur les employés concerneront leur date d'embauche. Nous aurions pu choisir "nom" et "prénom". Mais les noms des employés étant tous différents choisir ces champs ne changerait rien aux performances.

Pour la collection Hôtel, nous avons choisi le champ "id_station" comme index. Encore une fois, l'hypothèse a été faite que les requêtes auront probablement tendance à filtrer les hôtels par station.

Enfin, pour la collection Commerce, nous avons choisi d'indexer par "type" puis par "id_station". Car nous pensons que les requêtes concerneront soi des restaurants soi des fromageries, mais rarement les deux à la fois. En second lieu, nous indexons les commerces par station, pour la même raison que les hôtels.

3 Requêtes

3.1 Requete 1

3.1.1 Objectif

Cette requête récupère les noms et le massif des stations possédant plus de 5 pistes noires, ainsi que l'altitude, le tout formaté sous la chaine : "'Massif' - 'Nom' ('altitudeMin'-'altitudeMax' m) : X pistes noires."

3.1.2 Explication

Elle déconstruit le tableau des pistes, groupe par station en comptant le nombre de pistes noires par station, vérifie que le nombre obtenu est supérieur ou égal à 5, puis concatène les différents éléments pour former la chaîne voulue.

3.1.3 Implantation

```
db.station.aggregate([
  {$unwind: "$pistes"},
  {$match : {"pistes.niveau": "noir"}},
  {$group:
    {_id:"$_id",
    "nom":{$first: "$nom"},
    "massif":{$first: "$massif"},
    "altitudeMin":{$first: "$altitudeMin"},
    "altitudeMax":{$first: "$altitudeMax"},
    "count": {$sum:1}}},
  {$match : {"count": {$gte: 5}}},
  {$project: {"station":
    {$concat:
      ["$massif", " - ", "$nom", " (", {$toString: "$altitudeMin"}, "-", {$toString: "$altitudeMax"}, "
      {$toString: "$count"} ," pistes noires."]
    }
  }}
], {"explain": true});
```

3.1.4 Index

Pas d'utilisation d'index.

3.1.5 Avantages et inconvénients

Requête qui fait une sélection dès le début les pistes noires, permettant de diminuer le nombre de tuples.

Le mécanisme de unwind/group est assez coûteux, scan à chaque fois.

3.2 Requete 2

3.2.1 Objectif

Pour chaque restaurant asiatique, récupérer la plus haute hauteur de piste à laquelle un client pourrait skier, en utilisant la fonction cursor.map()

3.2.2 Explication

Elle déconstruit le tableau de détails des restaurants, filtre suivant le type de cuisine, regroupe par restaurant. Puis fait un map, pour associer à chaque restaurant sa station avec la hauteur de sa piste la plus haute.

3.2.3 Implantation

```
db.commerce.aggregate([
  // Les détails sont en array donc on doit unwind
  {$unwind: "$detailsRestaurant"},
  // Commerces = Restaurant ou Commerce. On a des champs différents pour les deux, doit checker
  {$match: {
    "type": "Restaurant",
    "detailsRestaurant.typeCuisine": "Asiatique"
  }},
  {$group: {
    _id: "$_id",
    "id_station": {$first: "$id_station"},
    nom: {$first: "$nom"},
    cuisine: {$first: "$detailsRestaurant.typeCuisine"}
  }}
]).map(function(commerce){return {"nom": commerce.nom, "cuisine": commerce.cuisine, "station": db.stati
  {$match: {
    "_id": commerce.id_station
  }},
  // Fait le lien restaurant - station
  {$unwind: "$pistes"},
  // Récupère la hauteur de la piste la plus haute de la station
  {$group: {
    "_id": "$_id",
    "nom": {$first: "$nom"},
    "hauteur_piste_max": {
      $max: "$pistes.altitude.max"
    }
  }},
]).map(function(station){return {station}})
}})
```

3.2.4 Index

Utilise l'index sur le type. Permet de récupérer directement les restaurants, évite le scan complet de la base avec le listing des fromagers.

3.2.5 Avantages et inconvénients

L'utilisation de l'index pour la récupération des restaurants permet d'augmenter les performances.

Le mécanisme de unwind/group est assez coûteux. Et les sous-requêtes pour chaque restaurants sont énormément coûteuse, recalculant à chaque fois la hauteur maximale.

3.3 Requete 3

3.3.1 Objectif

Récupérer la moyenne des tailles des pistes par station, sans la fonction avg.

3.3.2 Explication

Elle déconstruit le tableau des pistes. Ensuite, elle regoupe ces pistes en insérant dans un tableau les longueurs de chaque piste. Puis elle réduit ce tableau à la taille du tableau et à la somme. Ensuite, elle effectue la division de la taille par la somme pour obtenir la moyenne.

3.3.3 Implantation

```
db.station.aggregate([
  // Récupère la liste des longeurs des pistes par station
  {$unwind: "$pistes"},
  {
    $group: {
      _id: "$_id",
      nom: {$first: "$nom"},
      pistes_longueurs: {
        $push: "$pistes.longueur"
      }
    }
  },
  // Réduit cette liste à la somme des longeurs et au nombre de valeurs
  {
    $project: {
      "nom": 1,
      "longueurs": {
        $reduce: {
          input: "$pistes_longueurs",
          initialValue: {somme: 0, nombre: 0},
          in: {
            somme: {
              $add: ["$$value.somme", {$toDouble: "$$this"}]
            },
            nombre: {
              $add: ["$$value.nombre", 1]
            }
          }
        }
      }
    }
  },
  // Effectue la division de la somme des longueurs au nombre de valeurs
  // pour avoir la moyenne
  {
    $project: {
      "nom": 1,
      "taille_moyenne": {
        $ceil: {
          $divide: ["$longueurs.somme", "$longueurs.nombre"]
        }
      }
    }
  }
]);
```

3.3.4 Index

Pas d'index.

3.3.5 Avantages et inconvénients

Le mécanisme de unwind/group est assez coûteux. Plus efficace avec la méthode avg.

3.4 Requete 4

3.4.1 Objectif

Récupérer les noms et les prénoms des employés qui s'occupent des remontées mécaniques et qui sont à la fois pisteurs.

3.4.2 Explication

Elle déconstruit le tableau des pisteurs et des remontées mécaniques. Ensuite elle vérifie qu'il y a bien des pisteurs et des employés aux remontées mécaniques. Puis compare les valeurs des deux champs. Si ces valeurs sont identiques, alors elle garde ce document et va récupérer les autres champs de l'employé.

3.4.3 Implantation

```
db.station.aggregate([
    // Explode the arrays
    {$unwind: "$pisteurs"},
    {$unwind: "$remonteesMecaniques"},
    // Verify that our keys exists
    {$match:{"pisteurs":{"$exists":true},"remonteesMecaniques.employe_id":{"$exists":true}}},
    // Compare the two fields and filter
    {$project: {
        "pisteurs":1,
        "Cmp": {"$cmp":["$pisteurs","$remonteesMecaniques.employe_id"]}
    }
    },
    // Check equals
    {$match:{"Cmp":0}},
    // Get the employer
    {$lookup: {
        from: "employe",
        localField: "pisteurs",
        foreignField: "_id",
        as: "employe",
    }},
    {$project: {"employe.prenom":1, "employe.nom":1}}
])
```

3.4.4 Index

Index sur employe.id_station utilisé ici pour la jointure, ce qui permet de gagner en performances par rapport à un "scan complet + selection".

3.4.5 Avantages et inconvénients

Lookup permet d'utiliser notre index.
Les deux premiers unwind génèrent beaucoup de résultats intermédiaires.

3.5 Requete 5

3.5.1 Objectif

Compter le nombre d'hotels avec des casiers à ski, le tout par étoile.

3.5.2 Explication

Cette requête utilise un algorithme de MapReducer. Le mapper va émettre le couple (étoiles, 1) pour chaque hotel. Le reducer va ensuite compter le nombre de fois par étoile que celle-ci a été émise.

3.5.3 Implantation

```
db.hotel.mapReduce(
  // Fonction Map
  function(){
    emit(this.etoiles, 1);
  },
  // Fonction Reduce
  function(key, values){
    return Array.sum(values);
  },
  {
    out : {inline : 1},
    // Travail sur les hotels avec des casiers à ski
    query: {"casierASki": true}
  }
)
```

3.5.4 Index

Pas d'index utilisé

3.5.5 Avantages et inconvénients

Algorithme très efficace pour un fort volume de donnée, permet de travailler ici avec un grand nombre d'hotels.