Rapport

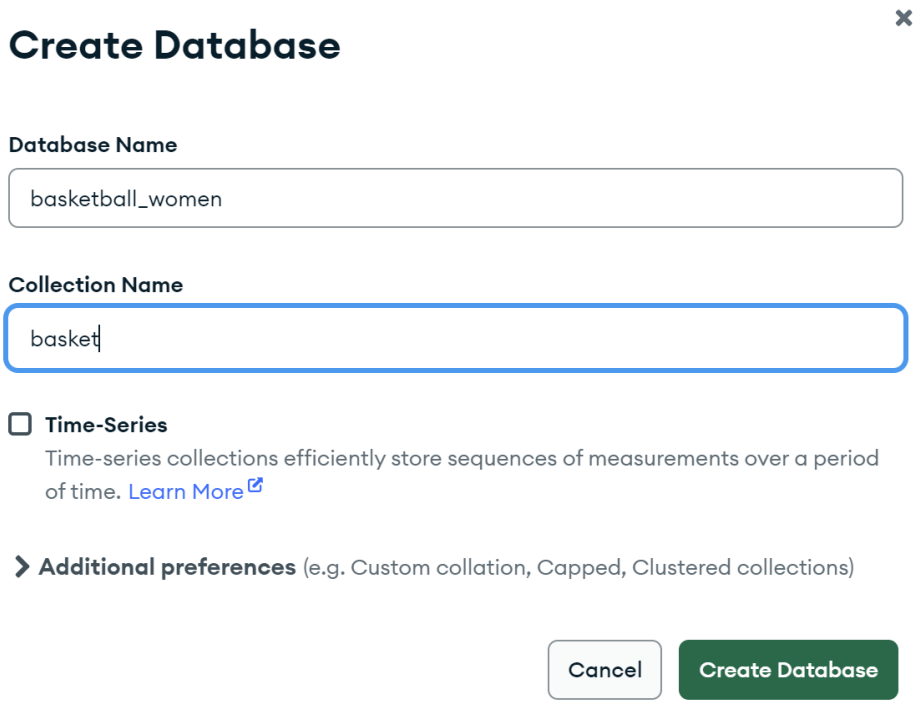
MongoDB

Table des matières

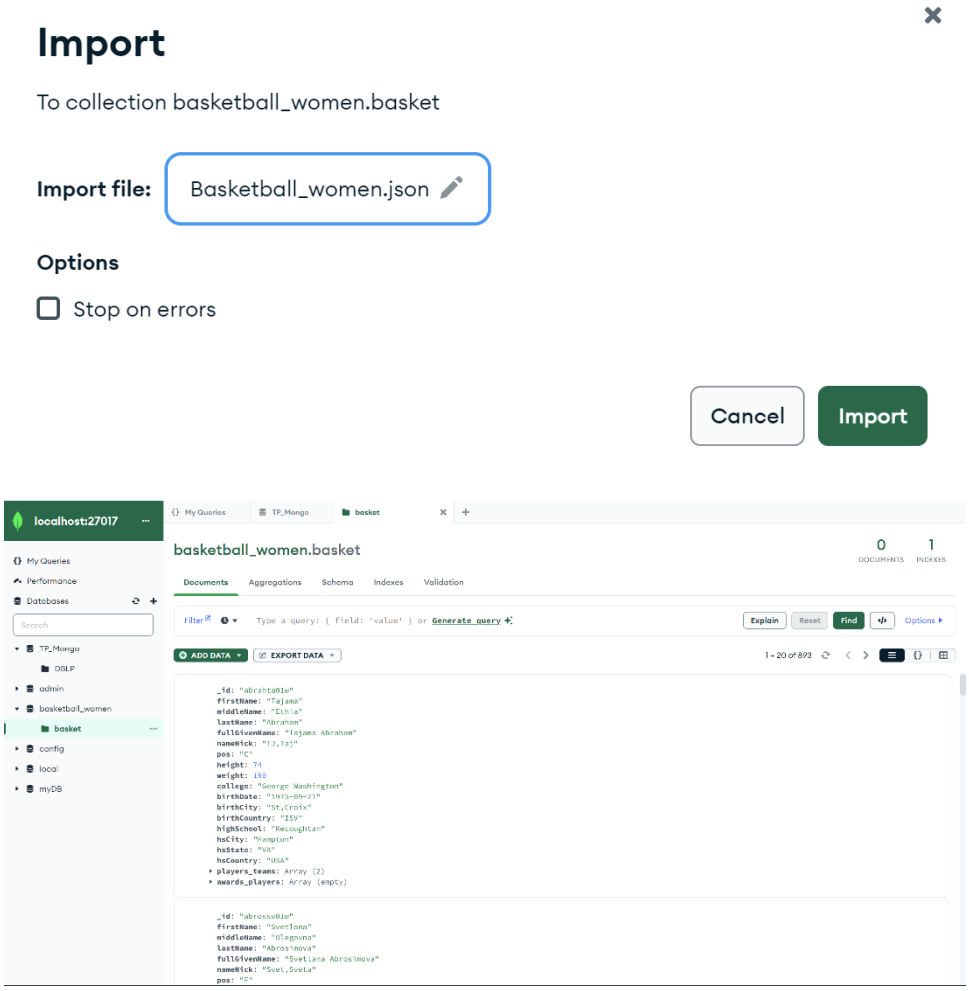
1. Data import

We chose the *basketball_women.json* dataset, its difficulty is 2 (complex dataset) so we need to do 6 simple queries, 2 complex queries, and 1 hard query.

First, we create a database basketball_women and a collection named “basket”:



Then we import the dataset:



We have 893 elements.

2. Simple queries (6 queries)

First of all, we don't forget de select the database we want to use (here, basketball_women):

```
> use basketball_women
< switched to db basketball_women
```

1) Our first query prints the full table of players.

To make this query, we just use db.basket.find({}), the brackets are empty because we don't need any conditions and we don't need to select any specific attribute, we want everything. The find() function is just an equivalent of the normal SELECT in SQL language.

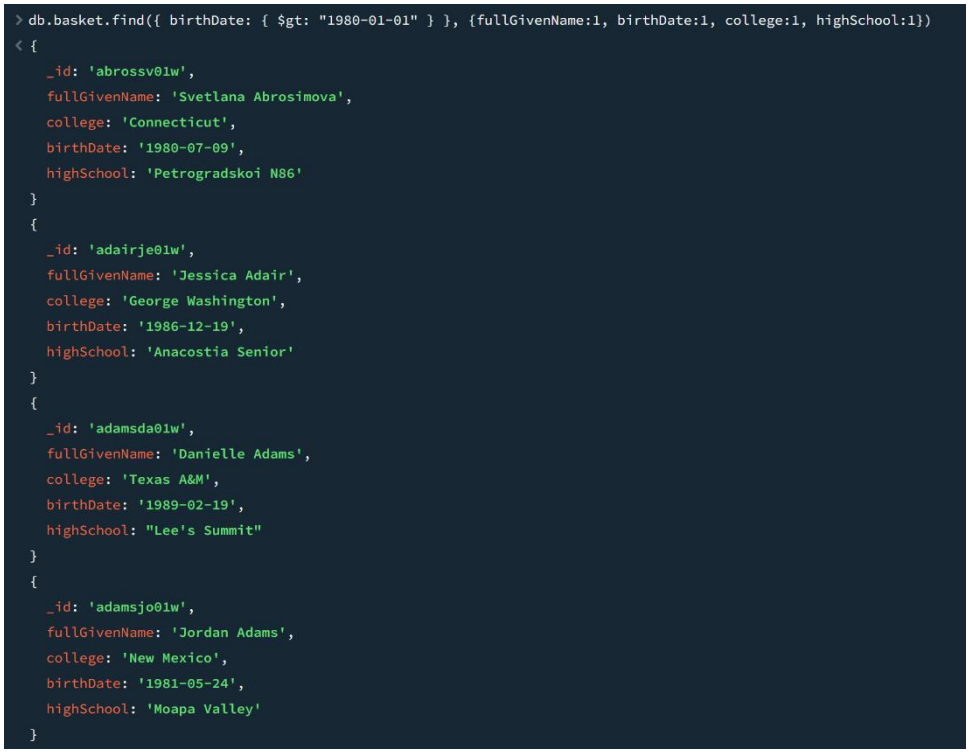
2) In the second query we search all the players who received an award in their career:

db.basket.find({awards_players:{$exists:true}}, {_id:0,firstName:1, lastName:1});

- The first parameter is for the condition:
  - We need players whose array awards_players are not empty, so we wrote the variable $exists: true. If we want players who never won an award, we replace the true value by false.
- The second parameter is for the attribute that we want the display:
  - _id:0 means that we didn't want to select it because it's always selected by default even if we don't write it inside the query and it's because it's a parimary key in our json dataset. That's why we put a 0 value to not display it instead of a 1.
  - firstName:1, lastName:1 is for displaying this to attributes firstName and lastName, so we wrote the value 1 in order to select them.

```
> db.basket.find({awards_players:{$exists:true}}, {_id:0,firstName:1, lastName:1});
< {
    firstName: 'Tajama',
    lastName: 'Abraham'
  }
  {
    firstName: 'Svetlana',
    lastName: 'Abrosimova'
  }
  {
    firstName: 'Jessica',
    lastName: 'Adair'
  }
  {
    firstName: 'Danielle',
    lastName: 'Adams'
  }
  {
    firstName: 'Jordan',
    lastName: 'Adams'
  }
  {
    firstName: 'Michael',
    lastName: 'Adams'
  }
  {
    firstName: 'Richie',
    lastName: 'Adubato'
```

3) In the third query, we want all the players who were born after 1980 and in which college or high school they played.



```
> db.basket.find({ birthDate: { $gt: "1980-01-01" } }, {fullGivenName:1, birthDate:1, college:1, highSchool:1})
< {
    _id: 'abrossv01w',
    fullGivenName: 'Svetlana Abrosimova',
    college: 'Connecticut',
    birthDate: '1980-07-09',
    highSchool: 'Petrogradskoi N86'
  }
  {
    _id: 'adairje01w',
    fullGivenName: 'Jessica Adair',
    college: 'George Washington',
    birthDate: '1986-12-19',
    highSchool: 'Anacostia Senior'
  }
  {
    _id: 'adamsda01w',
    fullGivenName: 'Danielle Adams',
    college: 'Texas A&M',
    birthDate: '1989-02-19',
    highSchool: "Lee's Summit"
  }
  {
    _id: 'adamsjo01w',
    fullGivenName: 'Jordan Adams',
    college: 'New Mexico',
    birthDate: '1981-05-24',
    highSchool: 'Moapa Valley'
  }
```

We can see that a lot of players come from USA's high school or college.

db.basket.find({ birthDate: {$gt: "1980-01-01"} }, {fullGivenName:1, birthDate:1, college:1, highSchool:1});

- The first parameter is to get all the players who were born after 1980, "gt" means "greater than".
- The second parameter's aim is to display these 4 attributes: The name of the player, her birth date, her high school and college where she played before. However, as we said previously, if we didn't specify that we don't need the id, it will automatically display it!

4) In the fourth query, we search for the players that did their first season in 2010.

```
basketball_women> db.basket.find(
                    { "players_teams.0.year": 2010 },
                    { _id: 0, firstName: 1, "players_teams.year": 1 })
```

db.basket.find({ "players_teams.0.year": 2010 },{ _id: 0, firstName: 1, "players_teams.year": 1 })

- The first parameter is to get all the players where the year of the first element of the array players_teams is equal to 2010. This allows us to select only the players that did their first season in 2010.
- The second parameter is used to display only the first name and the year that he played. The other information are useless for us and we precise _id:0 to not display the id.

5) In the fifth query, we search the players who played the position G which means “Point guard” (meneuse)

db.basket.find({"fullGivenName": { $exists: true }, "pos": { $regex: /G/i }}, { _id: 0, fullGivenName: 1, pos: 1 })

- {"fullGivenName": { $exists: true }, "pos": { $regex: /G/i }} :
  - "fullGivenName": { $exists: true } : The first parameter means we want players which have a fillGivenName different of null value.
  - "pos": { $regex: /G/i } : The second parameter is a regex which means we want to retrieve all the players who have a “G” in their “pos” attribute (it’s the meaning of the “/”). Then the “i” stands for the pattern should match both uppercase and lowercase versions of the letters.

```
>_MONGOSH
> db.basket.find(
    {
      "fullGivenName": { $exists: true },
      "pos": { $regex: /G/i }
    },
    { _id: 0, fullGivenName: 1, pos: 1 }
  )
< {
    fullGivenName: 'Elisa Aguilar',
    pos: 'G'
  }
  {
    fullGivenName: 'Matee Ajavon',
    pos: 'G'
  }
  {
    fullGivenName: 'Marcie Alberts',
    pos: 'G'
  }
  {
    fullGivenName: 'Markita Aldridge',
    pos: 'G'
  }
  {
    fullGivenName: 'Erin Alexander',
    pos: 'G'
  }
  {
    fullGivenName: 'Tawona Alhaleem',
    pos: 'F-G'
  }
```

We noticed some interesting things. Indeed, some players have “F-G” or “G-F” (it’s the same thing), F-G means “Forward-Guard” and that’s to say these players can play 2 different positions, “Point guard” or “Power Forward” (ailier).

6) In the last query we search the total number of different college which are present in our dataset:

db.basket.distinct("college").filter(college => college !== "none").length

- distinct("college"): means we retrieve the distinct name of college, and we don’t want to count them twice or much.
- filter(college => college !== "none"): means after retrieving the distinct college, there is one “none” value for the players who didn’t play in a college so we filter it and remove the “none” value.
- Length: is applied to the filtered array to determine the count of distinct colleges.

```
> db.basket.distinct("college").filter(college => college !== "none").length
< 160
```

We can conclude for the 893 players in our dataset, there are “only” 160 different college.

3. Complex queries (2 queries)

1) Let's display the player who played for the most of teams in their whole careers sorting by descending order:

```
> db.basket.aggregate([
    {
      $unwind: "$players_teams"
    },
    {
      $group: {
        _id: "$_id",
        playerName: { $first: "$fullGivenName" },
        distinctTeams: { $addToSet: "$players_teams.tmID" }
      }
    },
    {
      $project: {
        _id: 0,
        playerName: 1,
        numDistinctTeams: { $size: "$distinctTeams" }
      }
    },
    {
      $sort: { numDistinctTeams: -1 }
    }
  ])
```

db.basket.aggregate([
{
$unwind: "$players_teams"
},
{
$group: {
_id: "$_id",
playerName: { $first: "$fullGivenName" },
distinctTeams: { $addToSet: "$players_teams.tmID" }
}
},
{
$project: {
_id: 0,
playerName: 1,
numDistinctTeams: { $size: "$distinctTeams" }
}

```
  },
  {
    $sort: { numDistinctTeams: -1 }
  }
])
```

Let’s explain what the query did:

- $unwind: "$players_teams":

Deconstructs the "players_teams" array, creating a separate document for each team.

- $group: { _id: "$_id", playerName: { $first: "$fullGivenName" },distinctTeams: { $addToSet:"$players_teams.tmID" }}:

Groups the documents by player ID, keeps the first instance of the player name, and creates an array of distinct teams they have played for.

  - _id: "$_id": Grouping documents based on their unique _id field.
  - playerName: { $first: "$fullGivenName" }: Selecting the first encountered value of "fullGivenName" as the player's name within each group.
  - distinctTeams: { $addToSet: "$players_teams.tmID" }: Creating an array of unique team IDs that the player has played for within each group using $addToSet.

- $project: {_id: 0,playerName: 1,numDistinctTeams: { $size: "$distinctTeams" }}:

Projects only the player name and the number of distinct teams.

- $sort: { numDistinctTeams: -1 }:

Sorts the result in descending order based on the number of distinct teams.

```
{
  playerName: 'Taj McWilliams-Franklin',
  numDistinctTeams: 9
}
{
  playerName: 'Stacey Lovelace',
  numDistinctTeams: 8
}
{
  playerName: 'Kristen Rasmussen',
  numDistinctTeams: 8
}
{
  playerName: 'Sheri Sam',
  numDistinctTeams: 8
}
{
  playerName: 'Tamara Moore',
  numDistinctTeams: 7
}
{
  playerName: 'Shannon Johnson',
  numDistinctTeams: 7
}
{
  playerName: 'Adrienne Goodson',
  numDistinctTeams: 7
}
{
  playerName: 'Kelly Miller',
  numDistinctTeams: 7
}
```


Overall, we can conclude that the player who played in most of the teams in their career is Taj McWilliams-Franklin who played in 9 different teams followed by Stacey Lovelace, Kristen Rasmussen and Sheri Sam who played in 8 different teams!

2) In the second query, we will find the players that that played together with the same last name:

```
db.basket.aggregate([
  {$unwind:"$players_teams"},
  {$group:{
    _id:{lastName:"$lastName", year:"$players_teams.year", tmID:"$players_teams.tmID"},
    players:{$push:{_id:"$_id", name:"$fullGivenName"}},
    count:{$count:{}}} },
  {$match:{count:{$gt:1}}},
  {$sort:{count:-1, "_id.year":1}},
  {$project:{_id:0, lastName: "$_id.lastName", year:"$_id.year", tmID:"$_id.tmID", players:"$players", count:"$count"}}
])
```

db.basket.aggregate([
 {
  $unwind: "$players_teams"

```
},
{
 $group:
 {
  _id: { lastName: "$lastName", year: "$players_teams.year", tmID: "$players_teams.tmID" },
  players: { $push: {_id: "$_id", name: "$fullGivenName"} },
  count: {$count:{}}
 }
},
{
 $match: {count:{$gt:1}}
},
{
 $sort: {count:-1, "_id.year":1}
},
{
 $project:
 {
  _id:0,
  lastName: "$_id.lastName",
  year:"$_id.year",
  tmID:"$_id.tmID",
  players:"$players",
  count:"$count"
 }
}
])
```

Let's explain what the query did:

- $unwind: "$players_teams":

Deconstructs the "players_teams" array, creating a separate document for each team.

- $group:

```
{
 _id: { lastName: "$lastName", year: "$players_teams.year", tmID: "$players_teams.tmID" },
 players: { $push: {_id: "$_id", name: "$fullGivenName"} },
 count: {$count:{}}
}
```

  - _id: { lastName: "$lastName", year: "$players_teams.year", tmID: "$players_teams.tmID" }

    Groups the documents by player lastName, and year and teamId of the team they played in.
  - players: { $push: {_id: "$_id", name: "$fullGivenName"} }

    Creates an array document with _id and full name of all players grouped as defined previously. Why doing a document array instead of a string of players name array? Because there could be 2 players with the same first name, so we add _id to differentiate.

- count: {$count:{}}

Count these number of players grouped

- $match:{count:{$gt:1}}

Select only players group with more than one person

- $sort:{count:-1, "_id.year":1}

Sort the group by this number of player (descendent order) and year (ascendent order)

- $project: {_id:0, lastName: "$_id.lastName", year:"$_id.year", tmID:"$_id.tmID", players:"$players", count:"$count" }

Project the selected documents as follows, to have a better view of elements. {lastName, year, tmID, players[], count} insteed of {_id:{lastName, year}, tmID, players[], count}


```
{
  lastName: 'Barnes',
  year: 1998,
  tmID: 'SAC',
  players: [
    {
      _id: 'barnead01w',
      name: 'Adia Barnes'
    },
    {
      _id: 'barnequ01w',
      name: 'Quacy Barnes'
    }
  ],
  count: 2
}
{
  lastName: 'Jackson',
  year: 1998,
  tmID: 'WAS',
  players: [
    {
      _id: 'jacksan01w',
      name: 'Angela Jackson'
    },
    {
      _id: 'jacksta01w',
      name: 'Tammy Jackson'
    }
  ],
  count: 2
}
{
  lastName: 'Johnson',
```

4. Hard query (1 query)

For the hard query, let's see which players have the best ratio of points per minutes played. Moreover, we decided to select only the players that played at least 100 minutes. By doing this, this prevents having players who have a very good ratio having only played 10 or 15 minutes.

```
basketball_women> db.basket.aggregate([
                    { $unwind: "$players_teams" },
                    { $group: {
                        _id: "$_id",
                        firstName: { $first: "$firstName" },
                        lastName: { $first: "$lastName" },
                        pos: { $first: "$pos" },
                        totalPoints: { $sum: "$players_teams.points" },
                        totalMinutes: { $sum: "$players_teams.minutes" },
                        totalGamesPlayed: { $sum: "$players_teams.games" },
                        totalTime: { $sum: "$players_teams.minutes" }
                    }},
                    { $match: { totalMinutes: { $gt: 100 } }},
                    { $project: {
                        _id: 0,
                        firstName: 1,
                        lastName: 1,
                        pos: 1,
                        totalGamesPlayed: 1,
                        totalTime: 1,
                        averagePointsPerMinute: { $divide: ["$totalPoints", "$totalMinutes"] }
                    }},
                    { $sort: { averagePointsPerMinute: -1 }}
                  ])
```

```
db.basket.aggregate([
 { $unwind: "$players_teams" },
 { $group: {
    _id: "$_id",
    firstName: { $first: "$firstName" },
    lastName: { $first: "$lastName" },
    pos: { $first: "$pos" },
    totalPoints: { $sum: "$players_teams.points" },
    totalMinutes: { $sum: "$players_teams.minutes" },
    totalGamesPlayed: { $sum: "$players_teams.games" },
    totalTime: { $sum: "$players_teams.minutes" }
 }},
 { $match: { totalMinutes: { $gt: 100 } }},
 { $project: {
    _id: 0,
    firstName: 1,
    lastName: 1,
    pos: 1,
    totalGamesPlayed: 1,
```

```
    totalTime: 1,
   averagePointsPerMinute: { $divide: ["$totalPoints", "$totalMinutes"] }
  }},
  { $sort: { averagePointsPerMinute: -1 }}
])
```

Let's explain what the query did:

- $unwind: "$players_teams"

Deconstructs the "players_teams" array, creating a separate document for each team.

- $group: { _id: "$_id", firstName: { $first: "$firstName" },lastName: { $first: "$lastName" }, pos: { $first: "$pos" },totalPoints: { $sum: "$players_teams.points" }, totalMinutes: { $sum: "$players_teams.minutes" }, totalGamesPlayed: { $sum: "$players_teams.games" },totalTime: { $sum: "$players_teams.minutes" }

Groups the documents by id, keeps the first instance of first Name, last Name and position. We also create a totalPoints field which is the sum of the player points, a totalMinutes filed which is the sum of the player minutes, a TotalGamesPlayed field which is the sum of the player games and a totalTime field which is the sum of the time he has played.

- $match: { totalMinutes: { $gt: 100 }

This selects only the players that have played for at least 100 minutes. We used the $gt that means greater than and we filter on every totalMinutes that are greater than 100 minutes.

- $project: {_id: 0,firstName: 1,lastName: 1,pos: 1,totalGamesPlayed: 1,totalTime: 1, averagePointsPerMinute: { $divide: ["$totalPoints", "$totalMinutes"] }

Projects the last name, first name, totalGamesPlayed, totalTime, and also a averagePointsPerMinute that is the division of the number of points and the number of minutes. The _id : 0 is used to say that we don't want to project the _id.

- $sort: { averagePointsPerMinute: -1 }}

Sort the result by decreasing averagePointsPerMinute to see the players that has the best ratio.

```
< {
    firstName: 'Maya',
    lastName: 'Moore',
    pos: 'F',
    totalGamesPlayed: 68,
    totalTime: 952,
    averagePointsPerMinute: 1.0577731092436975
  }
  {
    firstName: 'Tina',
    lastName: 'Charles',
    pos: 'C',
    totalGamesPlayed: 101,
    totalTime: 2191,
    averagePointsPerMinute: 0.7859424920127795
  }
  {
    firstName: 'Kristi',
    lastName: 'Toliver',
    pos: 'G',
    totalGamesPlayed: 127,
    totalTime: 1844,
    averagePointsPerMinute: 0.7852494577006508
  }
```


Overall, we can conclude by saying that the player with the best ratio is Maya more with an average of 1.05 points per minute. This score is impressive, especially since it's an average over 68 matches. We decided to also project the position because we could expect to see one majority position like shooting guard that are expected to make 3 pointers, as we can expect the center to make the highest number of blocks. But finally, we can see a lot of diversity regarding the position for the average points per minute.