Please indicate below the tool you have analyzed for Homework 3, and past the code you have realized.
Your code must include all queries and indexes (if any), and possibly the script you have used to populate the database (in case you used a tool for this, e.g., compass for MongDB, please specify). If you have interacted with the database system through an external programming language, e.g., Python, insert your functions/program.

==========

GROUP 19: Gianluca Frezza (1909722), Ludovica Mazza (1917778)

==========

INTRODUCTION

We decided to use **MongoDB** for our homework assignment: since it's able to handle large amounts of data and can provide good capabilities for aggregation we thought it would be the best fit for our dataset.
In particular, we used MongoDB Compass.

Before entering into the queries, it is essential to provide a brief introduction on the changes made to the original dataset.

We initially had 11 original tables, but we found redundant information because some tables contained only foreign keys. Consequently, we chose to merge specific tables, writing the appropriate queries in postgres, to reduce the number of final collections to be created on MongoDB.

Specifically:

We created an "**event_sport**" table by merging the "**event**" and "**sport**" tables to include all relevant information related to various sports and events.

- **event_sport**

```
select e.id as event_id, e.sport_id, e.event_name, s.sport_name
from olympics.event e
inner join olympics.sport s on s.id = e.sport_id
```

We created a "**games_city**" table, merging the "**games**," "**games_city**," and "**city**" tables, to get the information related to the various Olympic games and the corresponding host cities.

- **games_city**

```
select
gc.games_id,gc.city_id,g.games_year,g.games_name,g.season,c.city_n
ame, c.region
from olympics.games g
inner join olympics.games_city gc on g.id = gc.games_id
inner join olympics.city c on c.id = gc.city_id
```

We created a table "**person_region**," by merging the tables "**person**," "**person_region**," and "**noc_region**," to include all details about an individual and his or her country of origin. In addition, we incorporated the "**age**" field from the "**games_competitors**" table into this "**person_region**" collection.

- **person_region**

```
select distinct nr.noc,nr.region_name,pr.person_id, pr.region_id,
p.full_name, p.gender, gc.age
from olympics.noc_region nr
inner join olympics.person_region pr on pr.region_id = nr.id
inner join olympics.person p on p.id = pr.person_id
inner join olympics.games_competitor gc on p.id = gc.person_id
```

Finally, we created a single table "**person_games_event_medal**," which includes the identifiers from the previous three collections. This to facilitate the join between these collections and also included the medal information, such as the "**medal_id**" and "**medal_name**" fields. As a result, the "**medal**", "**games_competitor**" and "**competitor_event**" tables could also be eliminated.

- **person_games_event_medal**

```
select ce.event_id, ce.medal_id, gc.games_id, gc.person_id
from olympics.competitor_event ce
inner join  olympics.games_competitor gc on ce.competitor_id =
gc.id
inner join olympics.medal m on m.id = ce.medal_id
```

So we are left with 4 tables to import as new collections in MongoDB : "**event_sport**", "**games_city**", "**person_region**", "**person_games_event_medal**".

Now, since most of our queries would have required a join between all four of these collections to obtain the desired information, in order to further reduce computational time, we decided to immediately create a fifth collection, called "four_join" containing the information resulting from the join on all four collections.
In this way, for all queries that needed all four collections, we had the aggregation pipeline start directly from the four_join table.

This is the code we used to create the four_join collection.

```
db.person_region.aggregate([
  {
    $lookup: {
      from: "person_games_event_medal",
      localField: "person_id",
      foreignField: "person_id",
      as: "person_medals"
    }
  },{ $match : { "person_medals" : { $ne : []}}},
  {
    $lookup: {
```

```
      from: "games_city",
      localField: "person_medals.games_id",
      foreignField: "games_id",
      as: "game"
    }
  },{ $match : { "game" : { $ne : []}}},
  {
    $lookup: {
      from: "event_sport",
      localField: "person_medals.event_id",
      foreignField: "event_id",
      as: "event"
    },
  },
{
    $match : { "event" : { $ne : []}}},

    {$out:”four_join”}])
```

Q1) This query returns all countries that have never participated in the sport 'swimming'.

Step 1: combine data from the "person_region," "person_games_event_medal," and "event_sport" collections to find countries that have participated in swimming events. Then group the results by the "region_id" field and export the output to a new collection called "swimming."

```
db.person_region.aggregate([
  {
    $lookup:
    /* lookup between person_region and person_games_event_medal
*/
      {
        from: "person_games_event_medal",
        localField: "person_id",
        foreignField: "person_id",
        as: "medal",
      },
  },
  {
    $lookup:
      /* lookup between medal and event_sport */
      {
        from: "event_sport",
        localField: "medal.event_id",
        foreignField: "event_id",
        as: "event",
      },
  },
  {
    $match:
      /* delete the documents which are not matched → since the
lookup is the equivalent of a left outer join, using $match we are
able to exclude all elements outside the intersection of the
tables, so we have the equivalent of an inner join, which is what
we need */
      {
        event: {
          $ne: [],
        },
      },
  },
  {
    $match:
      /* select the countries which have participated in swimming
*/
      {
        "event.sport_name": "Swimming",
      },
  },
  {
    $group:
      /* group by region_id */
      {
        _id: "$region_id",
```

```
      },
  },
  {
    $project: {
      /* Project the result, excluding _id and renaming _id to
region_id */
      _id: 0,
      region_id: "$_id",
    },
  },
  { /* export the results of the aggregation */
    $out: "swimming",
  }])
```

Step 2: This query identifies countries that have never participated in swimming competitions by performing a lookup between the "person_region" collection and the "swimming" collection. It groups the resulting documents by "region_name" and returns the unique region names of those countries.

```
db.person_region.aggregate([
  {
    $lookup:
      /* lookup between person_region and swimming */
      {
        from: "swimming",
        localField: "region_id",
        foreignField: "region_id",
        as: "matched_docs",
      },
  },
  {
    $match:
      /* we only consider non-matched documents → countries that
have never participated in swimming competitions */
      {
        matched_docs: {
          $size: 0,
        },
      },
  },
  {
    $group:
      /* group by region_name */
      {
        _id: "$region_name",
      },
  },
  {
    $project: {

     /* Project the result, excluding _id and renaming _id to
region_name */

      _id: 0,
```

```
    region_name: "$_id",
  },
}])
```

Q2) This query returns the person who won the most medals, how many medals he won, in which year and how old he was.
It identifies the person with the highest total number of medals from the "person_games_event_medal" collection. It retrieves additional information such as the person's full name, age, and the details of the games they participated in (year and season). The query performs lookups with the "person_region" and "games_city" collections to retrieve the corresponding data based on the IDs.

```
db.person_games_event_medal.aggregate([
  {
    $group: {

     /* Groups the documents from the "person_games_event_medal"
collection by the "person_id" field.
Calculates the total number of medals for each person using the
$sum operator.
Retrieves the "age" and "games_id" fields using the $first
operator. */

      _id: "$person_id",
      totalMedals: {
        $sum: 1,
      },
      age: {
        $first: "$personData.age",
      },
      games_id: {
        $first: "$games_id",
      },
    },
  },
  {
    $sort: {

/* Sort the documents in descending order based on the
"totalMedals" field. */

      totalMedals: -1,
    },
  },
  {
    $limit:

/* Limits the number of documents in the pipeline to 1 (only the
document with the highest total medals will be considered). */

 1,

},
  {
    $lookup: {

 /* lookup between the result of the previous stages and the
"person_region" collection.*/
```

```
      from: "person_region",
      localField: "_id",
      foreignField: "person_id",
      as: "personData",
    },
  },
  {
    $unwind: {

/* Deconstructs the "personData" array field created by the
previous $lookup stage, creating a new document for each element
in the array. */

      path: "$personData",
    },
  },
  {
    $lookup: {

/* lookup between the result of the previous stages and the
"games_city" collection. */

      from: "games_city",
      localField: "games_id",
      foreignField: "games_id",
      as: "gamesData",
    },
  },
  {
    $unwind: {

/* Deconstructs the "gamesData" array field created by the
previous $lookup stage, creating a new document for each element
in the array. */

      path: "$gamesData",
    },
  },
  {
    $project: {

/* Modifies the output documents, including only the specified
fields.Excludes the default "_id" field and renames some fields
for clarity. */

      _id: 0,
      full_name: "$personData.full_name",
      totalMedals: 1,
      age: "$personData.age",
      games_year: "$gamesData.games_year",
      season: "$gamesData.season",
    },
  },
  {
    $limit: /*Limits the number of documents in the pipeline to
1*/
```

```
1,}])
```

Q3) This query returns in the last 3 Olympics (which are the ones we remember) who took part in 'Women's 100-meter butterfly swimming' and 'Athletics Men's discus throwing' (we considered swimming and athletics because they were the sports we played competitively) and what kind of medals they won.

```
db.four_join.aggregate([

  /* unwind to flatten the arrays and create separate documents
for each element. */

  {
    $unwind: "$person_medals",
  },
  {
    $unwind: "$game",
  },
  {
    $unwind: "$event",
  },
  {
    $match:

      /* filter the documents based on specific conditions.
It selects documents where the "games_year" matches any of the
years 2016, 2012, or 2008, where the "event_name" matches either
"Swimming Women's 100 metres Butterfly" or "Athletics Men's Discus
Throw" and excludes documents where the "medal_id" field in
"person_medals" is equal to 4 (because we want the athletes
considered to have won at least one medal).
 */

      {
        "game.games_year": {
          $in: [2016, 2012, 2008],
        },
        "event.event_name": {
          $in: [
            "Swimming Women's 100 metres Butterfly",
            "Athletics Men's Discus Throw",
          ],
        },
        "person_medals.medal_id": {
          $ne: 4,
        },
      },
  },
  {
    $sort:

      /* sort data based on games_year field in ascending order.
*/

      {
        "game.games_year": 1,
      },
```

```
},
{
  $project: {
    _id: 0,
    full_name: "$full_name",
    games_name: "$game.games_name",
    event_name: "$event.sport_name",
    medal_name: "$person_medals.medal_name",
  },
}])
```

Q4) This query returns in which year and in which city did Italians take the most medals and how many medals did they take in total and for each type.

```
db.four_join.aggregate([
  {
    $match: {

/* filter documents where the "noc" field (National Olympic
Committee) is equal to "ITA" (representing Italy). */

      noc: "ITA",
    },
  },
  {
    $unwind:

/* unwinds the "game" array, creating separate documents for each
element. */

"$game",

  },

  {
    $project: {

/* modify the output documents, in particular:
   -  exclude the default _id field.
   -  include the "city_name" field from the "game" array as
      "city".
   -  calculate the "totalMedals" field using the $size operator to
      count the number of elements in the "person_medals" array
      that satisfy certain conditions specified with $filter. */


      _id: 0,
      city: "$game.city_name",
      totalMedals: {
        $size: {
          $filter: {
            input: "$person_medals",
            cond: {
              $and: [
                {
                  $ne: ["$$this.medal_id", 4],
                },
                {
                  $eq: [
                    "$$this.games_id",
                    "$game.games_id",
                  ],
                },
```

```
              ],
            },
          },
        },
      },
    },
  },
  {
    $group: {

/* groups the documents by the "city" field and calculates the sum
of "totalMedals" within each group. */

      _id: "$city",
      totalMedals: {
        $sum: "$totalMedals",
      },
    },
  },
  {
    $sort: {

/*  sorts the documents in descending order based on the
"totalMedals" field. */

      totalMedals: -1,
    },
  },
  {
    $limit:

/*  limits the result to only the document with the highest
"totalMedals". */

1

}])
```

Q5) This query returns which was the strongest country in 'Tug-of-War' and in 'Athletics Men's Stone Throw' and the total number of medals that country won in that competition.

```
db.four_join.aggregate([
    {
    $match: {

/*  filters documents based on the condition that the "event_name"
field is either "Athletics Men's Stone Throw" or "Tug-Of-War Men's
Tug-Of-War".

      $or: [
        {
          "event.event_name": {
            $eq: "Athletics Men's Stone Throw",
          },
        },
        {
          "event.event_name": {
            $eq: "Tug-Of-War Men's Tug-Of-War",
          },
        },
      ],
    },
  },
  {
    $project: {

/* modify the output documents in the following way:
   -  exclude the default _id field.
   -  include the "region_name" field as is.
   -  calculate the "discus_count" field by counting the number of
      elements in the "person_medals" array that have an "event_id"
      matching the "Athletics Men's Discus Throw" event.
   -  calculate the "tug_of_war_count" field by counting the number
      of elements in the "person_medals" array that have an
      "event_id" matching the "Tug of War" event. */

      _id: 0,
      region_name: 1,
      discus_count: {
        $size: {
      /* the size of the array which respect the
          following conditions*/
          $filter: {
            input: "$person_medals",
            cond: {
              $eq: [
```

```
                "$$this.event_id",
                {
                  $arrayElemAt: [
                    "$event.event_id",
                    {
                      $indexOfArray: [
                        "$event.event_name",
                        "Athletics Men's Discus Throw",
                      ],
                    },
                  ],
                },
              ],
            },
          },
        },
      },
      tug_of_war_count: {
        $size: {
          $filter: {
            input: "$person_medals",
            cond: {
              $eq: [
                "$$this.event_id",
                {
                  $arrayElemAt: [
                    "$event.event_id",
                    {
                      $indexOfArray: [
                        "$event.event_name",
                        "Tug of War",
                      ],
                    },
                  ],
                },
              ],
            },
          },
        },
      },
    },
  },
  {
    $group: {

      /* group the documents by the "region_name" field and
calculate the sum of "discus_count" and "tug_of_war_count" within
each group. */
```

```
      _id: "$region_name",
      total_discus_count: {
        $sum: "$discus_count",
      },
      total_tug_count: {
        $sum: "$tug_of_war_count",
      },
    },
  },
]
)
```

Q6) This query returns the 5 countries that won the most medals at the Olympic games (summer and winter), how many medals they won in total and how many in the summer and in the winter.

```
db.four_join.aggregate([
  {
    $unwind:

/*  unwind to deconstruct the "game" array. */

"$game",

  },
  {
    $group: {

/*  group the documents by the "region_name" field and calculates
the following fields within each group:
   -  "totalMedals": Counts the total number of medals within each
      group.
   -  "summerMedals": Counts the number of medals won in the Summer
      season by checking the "season" field in the "game" array.
   -  "winterMedals": Counts the number of medals won in the Winter
      season by checking the "season" field in the "game" array. */

      _id: "$region_name",
      totalMedals: {
        $sum: 1,
      },
      summerMedals: {
        $sum: {
          $cond: [
            {
              $eq: ["$game.season", "Summer"],
            },
            1,
            0,
          ],
        },
      },
      winterMedals: {
        $sum: {
          $cond: [
            {
              $eq: ["$game.season", "Winter"],
            },
            1,
            0,
          ],
        },
      },
    },
```

```
  },
  {
    $sort: {

/* sorts the documents based on the "totalMedals" field in
descending order. */

      totalMedals: -1,

    },
  },
  {
    $limit:

/* limits the output to 5 documents returning only the top 5
regions with the highest medal counts. */

5,

  },
]
)
```

Q7) This query returns all the athletes who have won at least 3 medals, who have participated in at least 5 different events in their life and who have participated in at least 3 different Olympic Games.

```
db.four_join.aggregate([
  {
    $match:

      /* filter documents where the "medal_id" in the
"person_medals" array is not equal to 4, because we want the
athletes considered to have won at least one medal */

      {
        "person_medals.medal_id": {
          $ne: 4,
        },
      },
  },
  {
    $group:

      /*  group the documents based on "person_id", "full_name",
"age", and "region_name" and within each group, calculate the
following fields:
   - "num_medals": Counts the number of elements in the
     "person_medals" array.
   - "num_games": Counts the number of elements in the "game"
     array.
   - "num_events": Counts the number of elements in the "event"
     array. */

      {
        _id: {
          person_id: "$person_id",
          full_name: "$full_name",
          age: "$age",
          region_name: "$region_name",
        },
        num_medals: {
          $sum: {
            $size: "$person_medals",
          },
        },
        num_games: {
          $sum: {
            $size: "$game",
          },
        },
```

```
        num_events: {
          $sum: {
            $size: "$event",
          },
        },
      },
  },
  {
    $match: {

/* filter the documents based on the following conditions:
   -  "num_games" must be greater than or equal to 3.
   -  "num_events" must be greater than or equal to 5.
   -  "num_medals" must be greater than or equal to 3. */

      num_games: {
        $gte: 3,
      },
      num_events: {
        $gte: 5,
      },
      num_medals: {
        $gte: 3,
      },
    },
  },
  {
    $project: {

/*  shape the output by selecting and renaming the desired fields.
The result will include documents with the following fields:
"full_name", "num_medals", "num_games", and "num_events". */

      _id: 0,
      full_name: "$_id.full_name",
      num_medals: "$num_medals",
      num_games: "$num_games",
      num_events: "$num_events",
    },
  }])
```

Q8) This query returns the country and the total number of gold medals won by athletes representing the same region where the Olympic Games were held.

```
db.person_region.aggregate([
  {
    $graphLookup: {

/*  graph lookup on the "person_games_event_medal" collection,
starting with the "person_id" field of the current document. It
follows the connections between documents based on the "person_id"
field and stores the matching documents in the "medals" array. It
only considers documents where the "medal_id" is 1.
In this case we use a graph lookup because it supports filtering
during the traversal process. In this query, it uses the
"restrictSearchWithMatch" option to filter the
"person_games_event_medal" documents based on the "medal_id"
field. This ensures that only documents with a specific medal type
(medal_id = 1) are considered in the lookup. */

      from: "person_games_event_medal",
      startWith: "$person_id",
      connectFromField: "person_id",
      connectToField: "person_id",
      as: "medals",
      restrictSearchWithMatch: {
        medal_id: 1,
      },
    },
  },
  {
    $match: {

/*  filter documents where the "medals" array is not empty to get
an "inner join" */

      medals: {
        $ne: [],
      },
    },
  },
  {
    $lookup: {
      from: "games_city",
      localField: "medals.games_id",
      foreignField: "games_id",
      as: "games",
    },
  },
```

```
  {
    $match: {
      games: {
        $ne: [],
      },
    },
  },
  {
    $match: {

/* filter documents where the "region_name" is present in the
"region" field of the "games" array. */

      $expr: {
        $in: ["$region_name", "$games.region"],
      },
    },
  },
  {
    $group: {

/*  group the documents by "region_name" and calculate the total
number of documents in each group, storing the result in the
"total_medals" field. */

      _id: {
        region: "$region_name",
      },
      total_medals: {
        $sum: 1,
      },
    },
  },
  {
    $sort: {

/* sorts the documents in a descending order */
      total_medals: -1,
    },
  },
  {
    $project: {
      region_name: "$_id.region_name",
      total_medals: 1,
    },
  },
]
)
```

Q9) This query returns the youngest and oldest person, and how old were both, to have won a medal in the Olympic games, in which season they participated and the sport in which they won.

```
db.four_join.aggregate([
  {
    $match: {

/*  filters the documents to only include those where the "age"
field exists and is of type "number". */

      age: {
        $exists: true,
        $type: "number",
      },
    },
  },
  {
    $facet: {

/* Retrieve youngest and oldest athletes:

   -  Youngest athletes:
Sort the documents in ascending order based on the "age" field.
Limit the result to only the first document (youngest athlete).
Project and reshape the selected fields, including "_id",
"full_name", "gender", "age", "season" (using the first element of
the "game.season" array), and "sport_name" (using the first
element of the "event.sport_name" array).

   -  Oldest athletes:
Sort the documents in descending order based on the "age" field.
Limit the result to only the first document (oldest athlete).
Project and reshape the selected fields, similar to the youngest
athletes.

      youngest: [
        {
          $sort: {
            age: 1,
          },
        },
        {
          $limit: 1,
        },
        {
          $project: {
            _id: 1,
            full_name: 1,
            gender: 1,
            age: 1,
            season: {
```

```
              $arrayElemAt: ["$game.season", 0],
            },
            sport_name: {
              $arrayElemAt: [
                "$event.sport_name",
                0,
              ],
            },
          },
        },
      ],
      oldest: [
        {
          $sort: {
            age: -1,
          },
        },
        {
          $limit: 1,
        },
        {
          $project: {
            _id: 1,
            full_name: 1,
            gender: 1,
            age: 1,
            season: {
              $arrayElemAt: ["$game.season", 0],
            },
            sport_name: {
              $arrayElemAt: [
                "$event.sport_name",
                0,
              ],
            },
          },
        },
      ],
    },
  },
  {
    $project: {

/* Concatenates the arrays of youngest and oldest athletes into a
single array named "athletes". */

      athletes: {
        $concatArrays: ["$youngest", "$oldest"],
      },
    },
  },
```

```
  {
    $unwind:

/* Unwind the "athletes" array, creating a separate document for
each element in the array. */

"$athletes",

  },
  {
    $replaceRoot: {

/* Replaces the root document with the "athletes" document,
effectively promoting it to the top-level. */

      newRoot: "$athletes",

    },
  },
]
)
```

Q10) This query returns how many women in total participated in the Olympic Games before and after 1968

```
db.four_join.aggregate ([
  {
    $match: {

/* filter the documents to only include those with a gender field
equal to "F" (female). */

      gender: "F",
    },
  },
  {
    $match: {

/*  filter the documents to only include those where the "game"
array is not empty. */

      game: {
        $ne: [],

      },
    },
  },
  {
    $group: {

/* group the documents based on the game year of the first element
in the "game.games_year" array.
Using the $cond operator, it checks if the game year is less than
1968: if the condition is true, it assigns the value "Before 1968"
to the "_id" field, otherwise "After 1968".

Then, through the $group, we accumulate the unique full names of
participants in the "participants" field using the $addToSet
operator. */

      _id: {
        $cond: {
          if: {
            $lt: [
              {
                $first: {
                  $slice: ["$game.games_year", 1],
                },
              },
              1968,
```

```
            ],
          },
          then: "Before 1968",
          else: "After 1968",
        },
      },
      participants: {
        $addToSet: "$full_name",
      },
    },
  },
  {
    $project: {

/*  Reshape the output: It retains the "_id" field, which
represents the categorized group (either "Before 1968" or "After
1968").
Then it calculates the count of participants in each group by
taking the size of the "participants" array using the $size
operator.
The result is stored in the "count" field.*/

      _id: 1,
      count: {
        $size: "$participants",
      },
    },
  },
])
```