# Aneks A – Kodi dhe ekzekutimet teknike

## Përmbajtja

# 1. NoSQL Data

## 1.1. Query 1 - Shtetet jo-NATO me dalje në det dhe me më shumë se 10 lumenj

```
db.geo_river.aggregate([
 // 1. Join with geo_sea collection to check if country has sea access
{
  $lookup: {
    from: "geo_sea",
    localField: "Country",
    foreignField: "Country",
    as: "sea_access"
  }
},
{
  $match: {
    sea_access: { $ne: [] } // Keep only countries with sea access
  }
},
// 2. Join with ismember collection to exclude countries that are NATO members
{
  $lookup: {
    from: "ismember",
    let: { code: "$Country" },
    pipeline: [
      {
        $match: {
          $expr: {
            $and: [
              { $eq: ["$Country", "$$code"] },
              { $eq: ["$Organization", "NATO"] },
              { $eq: ["$Type", "member"] }
            ]
          }
        }
      }
    ],
    as: "nato_status"
  }
},
{
```

```
  $match: {
    nato_status: { $eq: [] } // Keep only countries that are NOT NATO members
  }
},
// 3. Group by country and collect river names
{
  $group: {
    _id: "$Country",
    rivers: { $addToSet: "$River" }
  }
},
// 4. Filter countries with more than 10 rivers
{
  $match: {
    $expr: {
      $gt: [{ $size: "$rivers" }, 10]
    }
  }
},
// 5. Unwind river list to show each river separately and sort results
{
  $unwind: "$rivers"
},
{
  $project: {
    _id: 0,
    country: "$_id",
    river: "$rivers"
  }
},
{
  $sort: {
    country: 1,
    river: 1
  }
}
])
```

## 1.2. Query 2 - Shtetet e NATO që ndajnë të njëjtin det dhe liqen

```
db.ismember.aggregate([
```

```
{
  $match: {
    Organization: "NATO"
  }
},
{
  $lookup: {
    from: "geo_sea",
    localField: "Country",
    foreignField: "Country",
    as: "seas"
  }
},
{
  $lookup: {
    from: "geo_lake",
    localField: "Country",
    foreignField: "Country",
    as: "lakes"
  }
},
{
  $unwind: "$seas"
},
{
  $unwind: "$lakes"
},
{
  $project: {
    Country1: "$Country",
    Sea: "$seas.Sea",
    Lake: "$lakes.Lake"
  }
},
{
  $lookup: {
    from: "geo_sea",
    localField: "Sea",
    foreignField: "Sea",
    as: "seaMatches"
  }
},
{
  $lookup: {
    from: "geo_lake",
    localField: "Lake",
    foreignField: "Lake",
    as: "lakeMatches"
  }
```

```
},
{
  $unwind: "$seaMatches"
},
{
  $unwind: "$lakeMatches"
},
{
  $match: {
    $expr: {
      $and: [
        { $eq: ["$seaMatches.Country", "$lakeMatches.Country"] },
        { $ne: ["$Country1", "$seaMatches.Country"] },
        { $lt: ["$Country1", "$seaMatches.Country"] }
      ]
    }
  }
},
{
  $lookup: {
    from: "ismember",
    let: { c: "$seaMatches.Country" },
    pipeline: [
      {
        $match: {
          $expr: {
            $and: [
              { $eq: ["$Organization", "NATO"] },
              { $eq: ["$Country", "$$c"] }
            ]
          }
        }
      }
    ],
    as: "memberCheck"
  }
},
{
  $match: {
    "memberCheck.0": { $exists: true }
  }
},
// Deduplicate combinations of Country1, Country2, Sea, Lake
{
  $group: {
    _id: {
      Country1: "$Country1",
      Country2: "$seaMatches.Country",
      Sea: "$Sea",
```

```
      Lake: "$Lake"
    }
   }
  },
  {
   $project: {
    _id: 0,
    Country1: "$_id.Country1",
    Country2: "$_id.Country2",
    Sea: "$_id.Sea",
    Lake: "$_id.Lake"
   }
  }
 ])
 db.ismember.aggregate([
  {
   $match: {
    Organization: "NATO"
   }
  },
  {
   $lookup: {
    from: "geo_sea",
    localField: "Country",
    foreignField: "Country",
    as: "seas"
   }
  },
  {
   $lookup: {
    from: "geo_lake",
    localField: "Country",
    foreignField: "Country",
    as: "lakes"
   }
  },
  {
   $unwind: "$seas"
  },
  {
   $unwind: "$lakes"
  },
  {
   $project: {
    Country1: "$Country",
    Sea: "$seas.Sea",
    Lake: "$lakes.Lake"
   }
  },
```

```
{
  $lookup: {
    from: "geo_sea",
    localField: "Sea",
    foreignField: "Sea",
    as: "seaMatches"
  }
},
{
  $lookup: {
    from: "geo_lake",
    localField: "Lake",
    foreignField: "Lake",
    as: "lakeMatches"
  }
},
{
  $unwind: "$seaMatches"
},
{
  $unwind: "$lakeMatches"
},
{
  $match: {
    $expr: {
      $and: [
        { $eq: ["$seaMatches.Country", "$lakeMatches.Country"] },
        { $ne: ["$Country1", "$seaMatches.Country"] },
        { $lt: ["$Country1", "$seaMatches.Country"] }
      ]
    }
  }
},
{
  $lookup: {
    from: "ismember",
    let: { c: "$seaMatches.Country" },
    pipeline: [
      {
        $match: {
          $expr: {
            $and: [
              { $eq: ["$Organization", "NATO"] },
              { $eq: ["$Country", "$$c"] }
            ]
          }
        }
      }
    ],
```

```
    as: "memberCheck"
  }
},
{
  $match: {
    "memberCheck.0": { $exists: true }
  }
},
// Deduplicate combinations of Country1, Country2, Sea, Lake
{
  $group: {
    _id: {
      Country1: "$Country1",
      Country2: "$seaMatches.Country",
      Sea: "$Sea",
      Lake: "$Lake"
    }
  }
},
{
  $project: {
    _id: 0,
    Country1: "$_id.Country1",
    Country2: "$_id.Country2",
    Sea: "$_id.Sea",
    Lake: "$_id.Lake"
  }
}
])
```

## 1.3. Query 3 - Detet pa ishuj të kufizuara nga shtetet e NATO ose BE

```
db.sea.aggregate([
// 1. Left lookup into islandin to find islands in this sea
{
  $lookup: {
    from: "islandin",
    localField: "Name",
    foreignField: "Sea",
    as: "islands"
  }
},
// 2. Filter out seas that have any islands
{
  $match: {
    islands: { $size: 0 }
  }
},
```

```
// 3. Lookup geo_sea to get bordering countries
{
  $lookup: {
    from: "geo_sea",
    localField: "Name",
    foreignField: "Sea",
    as: "borderingCountries"
  }
},
// 4. Filter out seas with no bordering countries
{
  $match: {
    "borderingCountries.0": { $exists: true }
  }
},
// 5. For each bordering country, we need to check if the country is member of NATO or EU
// We can $lookup into country and then ismember to check memberships
// Unwind bordering countries to check each country separately
{
  $unwind: "$borderingCountries"
},
// Lookup country document by country code (borderingCountries.Country)
{
  $lookup: {
    from: "country",
    localField: "borderingCountries.Country",
    foreignField: "Code",
    as: "countryDoc"
  }
},
{
  $unwind: "$countryDoc"
},
// Lookup ismember for this country to get all memberships
{
  $lookup: {
    from: "ismember",
    localField: "countryDoc.Code",
    foreignField: "Country",
    as: "memberships"
  }
},
// Filter memberships only to NATO or EU
{
  $addFields: {
    isMemberNATOorEU: {
      $gt: [
        {
```

```
        $size: {
          $filter: {
            input: "$memberships",
            as: "m",
            cond: { $in: ["$$m.Organization", ["NATO", "EU"]] }
          }
        }
      },
      0
    ]
  }
 }
},
// 6. Group back by sea, gather all isMemberNATOorEU flags to check if all countries qualify
{
 $group: {
  _id: "$Name",
  allMembersNATOorEU: { $min: "$isMemberNATOorEU" }  // if any false, min will be false
 }
},
// 7. Only keep seas where all bordering countries are NATO or EU members
{
 $match: {
  allMembersNATOorEU: true
 }
},
// 8. Project output
{
 $project: {
  _id: 0,
  Sea_Name: "$_id"
 }
}
]);
```

# 2. Big Data

## 2.1. Konfigurimi në Google Colab

```
# Install Java
!apt-get install openjdk-8-jdk-headless -qq > /dev/null

# Download Spark 3.1.2 with Hadoop 2.7
```

```
!wget
https://archive.apache.org/dist/spark/spark-3.1.2/spark-3.1.2-bin-hadoop2.
7.tgz

# Extract the archive
!tar -xvzf spark-3.1.2-bin-hadoop2.7.tgz

# Install findspark
!pip install -q findspark
```

## 2.2. Importimi i datasetit Mondial

```
from google.colab import files
uploaded = files.upload()

!mkdir mondial
mondial_path = "mondial"

for file in os.listdir(mondial_path):
    if file.endswith(".json"):
        table_name = file.replace(".json", "")
        full_path = os.path.join(mondial_path, file)
        df = spark.read.json(full_path)
        df.createOrReplaceTempView(table_name)
        print(f"✅ Loaded table: {table_name}")
```

## 2.3. Queries në datasetin Mondial

Query 1 - Kryeqytetet që nuk kalohen nga lumenj por janë pjesë e organizatave

```
# Reading the files from drive
df_city                                                                =
spark.read.json("/content/drive/MyDrive/big_data_project/mondial/city.json
")
```

```
df_country =
spark.read.json("/content/drive/MyDrive/big_data_project/mondial/country.j
son")
df_organization =
spark.read.json("/content/drive/MyDrive/big_data_project/mondial/organizat
ion.json")
df_located =
spark.read.json("/content/drive/MyDrive/big_data_project/mondial/located.j
son")
df_located.createOrReplaceTempView("located")


df_city.createOrReplaceTempView("city")
df_country.createOrReplaceTempView("country")
df_organization.createOrReplaceTempView("organization")

spark.sql("""
WITH Kryeqytetet AS (
  SELECT DISTINCT c.Name AS Kryeqytetet
  FROM city c
  RIGHT JOIN country co ON c.Name = co.Capital
  WHERE c.Name IS NOT NULL
),
Kryeqytetet_Organizat AS (
  SELECT k.Kryeqytetet AS Kryeqytetet_O
  FROM Kryeqytetet k
  WHERE EXISTS (
    SELECT 1 FROM organization o WHERE o.City = k.Kryeqytetet
  )
)
SELECT ko.Kryeqytetet_O
FROM Kryeqytetet_Organizat ko
WHERE NOT EXISTS (
  SELECT 1 FROM located l WHERE l.City = ko.Kryeqytetet_O
)
""").show()
```

Query 2 - Pesë malet më të larta në Ballkan përjashtuar ato të Serbisë

```
df_mountain =
spark.read.json("/content/drive/MyDrive/big_data_project/mondial/mountain.
json")
df_geo_mountain =
spark.read.json("/content/drive/MyDrive/big_data_project/mondial/geo_mount
ain.json")

df_mountain.createOrReplaceTempView("mountain")
df_geo_mountain.createOrReplaceTempView("geo_mountain")

spark.sql("""
  SELECT m.Name AS Mountain_Name,
         MAX(m.Height) AS Height
  FROM mountain m
  JOIN geo_mountain g ON m.Name = g.Mountain
   WHERE g.Country IN ('AL', 'BIH', 'BG', 'HR', 'GR', 'XK', 'MNE', 'MK',
'RO', 'SI')
  GROUP BY m.Name
  ORDER BY Height DESC
  LIMIT 5
""").show()
```

## 2.4. Importimi i dataset-it COVID-19

```
from google.colab import files
uploaded = files.upload()
# Load the CSV file with header and inferred data types
covid_df = spark.read.csv("usa_county_wise.csv", header=True,
inferSchema=True)

# Register it as a SQL table
covid_df.createOrReplaceTempView("covid")
# Preview the schema
covid_df.printSchema()
```

Query 1 - Top 10 shtete për nga rastet e konfirmuara

```
df_covid =
spark.read.csv("/content/drive/MyDrive/big_data_project/usa_county_wise.csv", header=True, inferSchema=True)
df_covid.createOrReplaceTempView("covid")


spark.sql("""
  SELECT Province_State AS State,
         SUM(Confirmed) AS Total_Confirmed
  FROM covid
  GROUP BY Province_State
  ORDER BY Total_Confirmed DESC
  LIMIT 10
""").show()
```

Query 2 - Shtetet me përqindjen më të lartë të vdekjeve

```
df_covid =
spark.read.csv("/content/drive/MyDrive/big_data_project/usa_county_wise.csv",header=True,inferSchema=True)
df_covid.createOrReplaceTempView("covid")

spark.sql("""
  SELECT Province_State AS State,
         SUM(Deaths) AS Total_Deaths,
         SUM(Confirmed) AS Total_Confirmed,
                  ROUND(SUM(Deaths) * 100.0 / SUM(Confirmed), 2) AS Death_Rate_Percent
  FROM covid
  WHERE Confirmed > 0
  GROUP BY Province_State
  ORDER BY Death_Rate_Percent DESC
  LIMIT 10
""").show()
```

Query 3 - Korelacioni në mes të rasteve të konfirmuara dhe vdekjeve

```
df_covid =
spark.read.csv("/content/drive/MyDrive/big_data_project/usa_county_wise.csv", header=True, inferSchema=True)
df_covid.createOrReplaceTempView("covid")

spark.sql("""
  SELECT Admin2 AS County, Province_State AS State,
         SUM(Confirmed) AS Confirmed, SUM(Deaths) AS Deaths
  FROM covid
  GROUP BY Admin2, Province_State
  ORDER BY Confirmed DESC
  LIMIT 10
""").show()
```

### 2.5. Vizualizimi 1 - Korelacioni mes rasteve të konfirmuara dhe vdekjeve

```
scatter_df = spark.sql("""
  SELECT SUM(Confirmed) AS Confirmed, SUM(Deaths) AS Deaths
  FROM covid
  GROUP BY Province_State
""").toPandas()

plt.scatter(scatter_df["Confirmed"], scatter_df["Deaths"], alpha=0.6)
plt.title("Deaths vs Confirmed Cases by State")
plt.xlabel("Confirmed Cases")
plt.ylabel("Deaths")
plt.grid(True)
plt.tight_layout()
plt.show()
```

### 2.6. Vizualizimi 2 - Shtetet me më së shumti raste të konfirmuara

```
import pandas as pd
import matplotlib.pyplot as plt

top_states = spark.sql("""
  SELECT Province_State AS State, SUM(Confirmed) AS Confirmed
  FROM covid
  GROUP BY Province_State
```

```
  ORDER BY Confirmed DESC
  LIMIT 10
""").toPandas()

top_states.plot(kind='bar', x='State', y='Confirmed', legend=False)
plt.title("Top 10 States by Confirmed Cases")
plt.ylabel("Confirmed Cases")
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

### 2.7. Mount për Google Drive

```
from google.colab import drive
drive.mount('/content/drive')
```

### 2.8. Ngarkimi i dataseteve nga Google Drive

```
# For Mondial
df_mondial =
spark.read.json("/content/drive/MyDrive/big_data_project/mondial/continent
.json")

# For COVID CSV dataset
df_covid =
spark.read.csv("/content/drive/MyDrive/big_data_project/usa_county_wise.cs
v", header=True, inferSchema=True)
```