**Universiteti i Prishtinës**

**Fakulteti Inxhinierisë Elektrike dhe Kompjuterike**



**Dokumentim teknik i projektit**

**Big Data**

**Projekt Pjesa I**

**Emri dhe Mbiemri ID-**

| 1. Daris Dragusha | 220756100113 |
| --- | --- |
| 2. Art Jashari | 220756100114 |
| 3. Artin Rexhepi | 220756100130 |
| 4. Ag Hamiti | 220756100026 |
| 5. Gent Podvorica | 220756100109 |

Prishtinë, 2024/25

[**Hyrje 3**](#_blxfr21z26ik)

[**Structured Data Querying 3**](#_kbgs7buwcg5a)

[Queries/Views 3](#_kr920z4dgxtl)

[Query / View 1: 3](#_t9zytrrfc8z3)

[Query / View 2: 3](#_rvx5zmnrk4c8)

[Query / View 3: 4](#_y2qg0qwbzm9w)

[Query / View 4: 4](#_2zd8v153nop7)

[Query / View 5: 5](#_ycuuwajdqtyd)

[Stored Procedures: 7](#_uwwnw2x9qfkf)

[Dataset I Jashtëm 7](#_hfeuk85zyh2r)

[Gjetja dhe Përgatitja e Dataset-it 8](#_setnzg3u9mhj)

[Krijimi i Databazës dhe Strukturës së Tabelës 8](#_fvduzih50n6a)

[Ngarkimi i të Dhënave në MySQL 8](#_5jvrmhikxepl)

[Krijimi i një Funksioni për Llogaritjen e Jetëgjatësisë Mesatare 9](#_vyftnu5pc8xc)

[Krijimi i një View për Anëtarët e NATO-s 9](#_ou6ozn1yhwav)

[Ndryshimet në PostgreSQL 10](#_tmbw882pjn2c)

[**Data Warehouse 12**](#_5qyrc59se046)

[Krijimi i strukturës 12](#_1gn6h03to4rf)

[Skripta për krijimin e strukturës 12](#_71rkjsmuj351)

[Mbushja e të dhënave 13](#_13b0ubov86ea)

[Automatizimi i procesit të mbushjes së DWH-së 14](#_teitxhh2x216)

[Përfundim 15](#_fonviua0m97y)

# 

# Hyrje

Në këtë projekt kemi përdorur dy sisteme për menaxhimin e bazave të të dhënave (DBMS): MySQL dhe PostgreSQL, së bashku me mjetet përkatëse MySQL Workbench dhe PGAdmin. Fillimisht kemi shkarkuar MySQL nga<https://dev.mysql.com/downloads/> dhe PostgreSQL nga<https://www.postgresql.org/download/>. Pas shkarkimit, kemi ndjekur hapat e instalimit përmes instaluesve (installation wizards) të secilit sistem, dhe çdo gjë është instaluar me sukses pa ndonjë problem. Më pas kemi marrë skemën përkatëse nga repository në GitHub per Mondial database dhe kemi ekzekutuar skriptet SQL në të dy platformat për të krijuar strukturën e bazës së të dhënave.

Të gjithë skriptet dhe kodi i përdorur në këtë projekt mund të gjendet në repository-n tonë në GitHub (<https://github.com/artinrexhepii/BigData_Project>). Për lehtësi, pjesët më të rëndësishme të kodit do të paraqiten edhe brenda këtij dokumenti, së bashku me shpjegimet përkatëse.

# Structured Data Querying

## Queries/Views

### **Query / View 1:**

Qëllimi i kësaj kërkese është të listohen kryeqytetet dhe qyteti më i madh për nga popullsia për secilin vend anëtar të NATO-s. Janë përdorur tabelat country, isMember dhe city, ku përmes bashkimeve dhe filtrave përkatës është krijuar view-i NATO\_Largest\_Cities:

CREATE VIEW NATO\_Largest\_Cities AS

SELECT co.Name AS Country, co.Capital AS CapitalCity,

maxcity.Name AS LargestCity, maxcity.Population AS LargestCityPopulation

FROM country co

JOIN isMember m ON co.Code = m.Country

JOIN city maxcity ON maxcity.Country = co.Code

WHERE m.Organization = 'NATO'

AND maxcity.Population = (

SELECT MAX(c2.Population) FROM city c2 WHERE c2.Country = co.Code

);

Logjika përfshin filtrimin e shteteve që janë anëtare të NATO-s, gjetjen e qytetit më të populluar për secilin shtet dhe bashkimin e të dhënave përmes kolonës Code.

### **Query / View 2:**

Kërkesa e dytë synon të listojë shtetet anëtare të NATO-s që nuk kanë dalje në det dhe që kanë një popullsi prej të paktën një milion banorë. Janë përdorur tabelat country, isMember dhe geo\_sea, dhe është ndërtuar view me query-n e mëposhtëm:

SELECT co.Name AS Country, co.Capital, co.Population

FROM country co

JOIN isMember m ON co.Code = m.Country

WHERE m.Organization = 'NATO'

AND co.Population >= 1000000

AND co.Code NOT IN (SELECT DISTINCT Country FROM geo\_sea);

Këtu, bashkimi i country me isMember siguron përzgjedhjen vetëm të vendeve të NATO-s, ndërsa filtrat përjashtojnë ato me dalje në det.

### **Query / View 3:**

Kjo pyetje identifikon malin më të lartë për çdo shtet në kontinentin Amerika, duke përfshirë emrin e shtetit, malin më të lartë, lartësinë dhe provincën. Përdoren tabelat country, encompasses, geo\_mountain dhe mountain. View-i ndërtohet si më poshtë:

SELECT co.Name AS Country, m.Name AS Mountain, m.Height, gm.Province

FROM country co

JOIN encompasses e ON co.Code = e.Country

JOIN geo\_mountain gm ON gm.Country = co.Code

JOIN mountain m ON m.Name = gm.Mountain

WHERE e.Continent = 'America'

AND m.Height = (

SELECT MAX(m2.Height)

FROM geo\_mountain gm2

JOIN mountain m2 ON m2.Name = gm2.Mountain

WHERE gm2.Country = co.Code

);

Kjo strukturë siguron që për çdo shtet në Amerikë të shfaqet vetëm mali më i lartë.

### **Query / View 4:**

Kjo është një pyetje bazë që synon të listojë informacionet themelore për të gjitha shtetet në bazën e të dhënave – emri, kryeqyteti dhe popullsia. View-i All\_Countries\_Basic\_Info krijohet si më poshtë:

CREATE VIEW All\_Countries\_Basic\_Info AS

SELECT Name AS Country, Capital AS CapitalCity, Population

FROM country;

Ky view e bën të lehtë përdorimin e këtyre të dhënave në analizat e mëvonshme.

### **Query / View 5:**

Përmes këtyre pyetjeve SQL kemi identifikuar trupat ujore – liqenet dhe detet – që ndahen nga të paktën dy vende anëtare të NATO-s. Në mënyrë të veçantë, query-t përzgjedhin liqenet dhe detet ku janë të pranishme të paktën dy shtete të NATO-s, duke u bazuar në tabelat geo\_lake dhe geo\_sea, të lidhura me informacionin për anëtarësimin në organizata nga tabela isMember. Për çdo trup ujor të përbashkët, lista përfundimtare tregon emrin e ujit, shtetin përkatës dhe llojin e trupit ujor (liqen apo det). Gjithashtu, në një query të veçantë janë nxjerrë shtetet që ndajnë njëkohësisht si liqene ashtu edhe dete me vende të tjera të NATO-s, duke i lidhur të dhënat mbi bazën e përbashkët të kodit të shtetit. Këto pyetje ndihmojnë në analizën e përbashkësive gjeografike midis vendeve anëtare të NATO-s.

**Për rastin kur ndajnë *të dyja*:**

-- Lakes shared by at least 2 NATO countries

SELECT

'Lake' AS WaterType,

gl.Lake AS WaterBody,

c.Name AS Country

FROM (

SELECT DISTINCT gl.Lake, gl.Country

FROM geo\_lake gl

JOIN isMember m ON gl.Country = m.Country

JOIN organization o ON m.Organization = o.Abbreviation

WHERE o.Name = 'North Atlantic Treaty Organization'

) gl

JOIN (

SELECT Lake

FROM geo\_lake gl

JOIN isMember m ON gl.Country = m.Country

JOIN organization o ON m.Organization = o.Abbreviation

WHERE o.Name = 'North Atlantic Treaty Organization'

GROUP BY Lake

HAVING COUNT(DISTINCT gl.Country) >= 2

) shared\_lakes ON gl.Lake = shared\_lakes.Lake

JOIN country c ON gl.Country = c.Code

UNION

-- Seas shared by at least 2 NATO countries

SELECT

'Sea' AS WaterType,

gs.Sea AS WaterBody,

c.Name AS Country

FROM (

SELECT DISTINCT gs.Sea, gs.Country

FROM geo\_sea gs

JOIN isMember m ON gs.Country = m.Country

JOIN organization o ON m.Organization = o.Abbreviation

WHERE o.Name = 'North Atlantic Treaty Organization'

) gs

JOIN (

SELECT Sea

FROM geo\_sea gs

JOIN isMember m ON gs.Country = m.Country

JOIN organization o ON m.Organization = o.Abbreviation

WHERE o.Name = 'North Atlantic Treaty Organization'

GROUP BY Sea

HAVING COUNT(DISTINCT gs.Country) >= 2

) shared\_seas ON gs.Sea = shared\_seas.Sea

JOIN country c ON gs.Country = c.Code

ORDER BY WaterType, WaterBody, Country:

**Për rastin kur ndajnë *njërën ose dyjat*:**

SELECT DISTINCT

c.Name AS Country,

gl.Lake AS SharedLake,

gs.Sea AS SharedSea

FROM

-- Get NATO countries with distinct lake

(

SELECT DISTINCT gl.Country, gl.Lake

FROM geo\_lake gl

JOIN isMember m ON gl.Country = m.Country

JOIN organization o ON m.Organization = o.Abbreviation

WHERE o.Name = 'North Atlantic Treaty Organization'

AND gl.Lake IN (

SELECT Lake

FROM geo\_lake gl2

JOIN isMember m2 ON gl2.Country = m2.Country

JOIN organization o2 ON m2.Organization = o2.Abbreviation

WHERE o2.Name = 'North Atlantic Treaty Organization'

GROUP BY Lake

HAVING COUNT(DISTINCT gl2.Country) >= 2

)

) AS gl

JOIN

-- Get NATO countries with distinct sea

(

SELECT DISTINCT gs.Country, gs.Sea

FROM geo\_sea gs

JOIN isMember m ON gs.Country = m.Country

JOIN organization o ON m.Organization = o.Abbreviation

WHERE o.Name = 'North Atlantic Treaty Organization'

AND gs.Sea IN (

SELECT Sea

FROM geo\_sea gs2

JOIN isMember m2 ON gs2.Country = m2.Country

JOIN organization o2 ON m2.Organization = o2.Abbreviation

WHERE o2.Name = 'North Atlantic Treaty Organization'

GROUP BY Sea

HAVING COUNT(DISTINCT gs2.Country) >= 2

)

) AS gs

ON gl.Country = gs.Country

JOIN country c ON gl.Country = c.Code

ORDER BY SharedLake, SharedSea, Country;

Në të dyja rastet përdoret krahasimi m1.Country < m2.Country për të shmangur përsëritje të panevojshme. Në versionin e dytë përdoren LEFT JOIN dhe COALESCE për të përfshirë rastet ku ekziston vetëm njëra lidhje ujore.

## Stored Procedures:

Për të shtuar fleksibilitetin e kërkimit brenda view-it American\_Countries\_Highest\_Mountains, është ndërtuar një stored procedure me emrin SearchAmericanMountains. Kjo procedurë lejon përdoruesin të kërkojë malet më të larta në vendet e kontinentit të Amerikës bazuar në kritere specifike si lartësia minimale, emri i shtetit, emri i provincës dhe mundësia për kërkim të pjesshëm (partial match).

Parametrat që procedura pranon janë katër: minHeight, që përcakton lartësinë minimale të malit; countryName, që mund të përmbajë emrin e plotë ose pjesërisht të një shteti; provinceName, që funksionon në mënyrë të ngjashme për provincat; si dhe partialMatch, një parametër logjik që tregon nëse kërkimi duhet të bëhet me përputhje të pjesshme (LIKE) apo të saktë (=).

Logjika brenda procedurës është ndërtuar në mënyrë që, nëse parametri për vendin ose provincën lihet bosh (''), atëherë ai filtër anashkalohet dhe përfshihen të gjitha vlerat përkatëse. Nëse partialMatch është i aktivizuar (TRUE), kërkimi realizohet me LIKE, që mundëson kërkime më fleksibile, si për shembull për të gjetur të gjitha provincat që përmbajnë fjalën “Andes”. Në të kundërt, nëse kërkohet përputhje e saktë (FALSE), përdoret operatori = për të krahasuar vlerat.

Nje thirrje e kësaj mund të jetë:  
CALL SearchAmericanMountains(5000, '', '', FALSE);

## Dataset I Jashtëm

### Gjetja dhe Përgatitja e Dataset-it

Kjo pjes e projektit filloi me kërkimin e një dataset-i voluminoz dhe gjithëpërfshirës që përmban indikatorë të zhvillimit për vende të ndryshme. Pas shqyrtimit të disa burimeve, vendosëm të përdorim dataset-in **Global Development**, i cili është në formatin CSV dhe përmban të dhëna si jetëgjatësia, rritja e popullsisë, përdorimi i telefonisë mobile, toka bujqësore dhe të tjera. Dataset-i është marrë nga [CORGIS Dataset Project](https://corgis-edu.github.io/corgis/csv/global_development/)  
Pas shkarkimit, CSV-ja u vendos në pathin **C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/**, pra brenda folderit Uploads,i cili është një vend i dedikuar nga MySQL për të lehtësuar ngarkimin e file-ve në databazë. Kjo siguron që skedari të jetë i aksesueshëm përmes SQL queries dhe shmang problemet me lejet e aksesit.

### Krijimi i Databazës dhe Strukturës së Tabelës

Sipas kërkesave të projektit, krijuam një databazë të re për të ruajtur të dhënat përmes queryt:

*CREATE DATABASE BigData\_Datasets;*

*USE BigData\_Datasets;*

Më pas analizuam strukturën e CSV-së dhe krijuam një tabelë që përputhet me kolonat dhe llojet e të dhënave të saj. Tabela u krijua si më poshtë:

*CREATE TABLE country\_data (*

*country VARCHAR(255),*

*year INT,*

*. . .*

### Ngarkimi i të Dhënave në MySQL

Pasi tabela u krijua, të dhënat nga CSV-ja u ngarkuan në tabelën *country\_data* duke përdorur komandën *LOAD DATA INFILE*, e cila mundëson importimin masive të të dhënave. Kjo komandë u ekzekutua si më poshtë:

*LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/global\_development.csv'*

*INTO TABLE country\_data*

*FIELDS TERMINATED BY ','*

*ENCLOSED BY '"'*

*LINES TERMINATED BY '\n'*

*IGNORE 1 ROWS*

*(country, year, birth\_rate, death\_rate, fertility\_rate, life\_expectancy\_female, . . .);*

Përdorimi i FIELDS TERMINATED BY dhe ENCLOSED BY siguron që çdo kolonë të ndahet saktë, ndërsa IGNORE 1 ROWS e anashkalon rreshtin e parë që përmban emrat e kolonave.

### Krijimi i një Funksioni për Llogaritjen e Jetëgjatësisë Mesatare

Konform kërkesave, krijuam një funksion që llogarit jetëgjatësinë mesatare për një vend të caktuar në një periudhë të përcaktuar kohore. Funksioni merr tre parametra: emrin e vendit, vitin e fillimit dhe vitin e mbarimit. Ai përdor funksionin AVG() mbi kolonën life\_expectancy\_total dhe filtron të dhënat për vitet dhe vendin e dhënë. Ja struktura e funksionit:

CREATE FUNCTION calculate\_life\_expectancy\_average(

country\_name VARCHAR(100),

start\_year INT,

end\_year INT

)

RETURNS DECIMAL(10,2)

DETERMINISTIC

BEGIN

DECLARE avg\_life\_expectancy DECIMAL(10,2);

SELECT AVG(life\_expectancy\_total)

INTO avg\_life\_expectancy

FROM country\_data

WHERE country = country\_name

AND year BETWEEN start\_year AND end\_year;

RETURN avg\_life\_expectancy;

END;

Ky funksion u përdor më pas në mënyrë të përsëritur në pyetjet tona analitike pa pasur nevojë për të përsëritur kodin.

### Krijimi i një View për Anëtarët e NATO-s

Në fazën e fundit, synuam të lidhim të dhënat e *global development* me tabelat nga databaza *Mondial*, për të analizuar vendet anëtare të NATO-s. Përdorëm funksionin e përmendur më lart dhe krijuam një pamje (*view*) që bashkon tabelën country\_data me tabelat country, isMember dhe organization nga databaza *Mondial*. E kemi të qartë që bashkimi me tabelën organization është i tepërt dhe jo i domosdoshëm, pasi emri i organizatës mund të filtrohet drejtpërdrejt përmes isMember. Megjithatë, për qëllime edukative dhe për të praktikuar bashkimin me tabela të shumta, vendosëm ta përfshijmë atë gjithsesi.

Kjo pamje filtron vendet që janë anëtare të NATO-s dhe llogarit jetëgjatësinë mesatare (dhe rangon në bazë të saj), popullsinë mesatare dhe sipërfaqen mesatare për çdo vend nga viti 1980 deri më 2020:

CREATE VIEW country\_life\_expectancy\_nato\_view AS

SELECT

cd.country,

calculate\_life\_expectancy\_average(cd.country, 1980, 2020) AS avg\_life\_expectancy,

AVG(cd.total\_population) AS avg\_population,

AVG(cd.surface\_area) AS avg\_surface\_area

FROM

country\_data cd

JOIN

mondial.country c ON cd.country = c.Name

JOIN

mondial.isMember im ON c.Code = im.Country

JOIN

mondial.organization o ON im.Organization = o.Abbreviation

WHERE

o.Name = 'North Atlantic Treaty Organization'

GROUP BY

cd.country

ORDER BY

avg\_life\_expectancy DESC;

### Ndryshimet në PostgreSQL

Për të realizuar të njëjtin proces në PostgreSQL, patëm disa ndryshime që duhej t’i përshtatnim krahasuar me MySQL. Fillimisht, databazën BigData\_Dataset e krijuam duke përdorur ndërfaqen grafike (GUI) të pgAdmin, pasi kjo metodë ishte më e lehtë dhe më intuitive për këtë rast. Pasi databaza u krijua, ngarkimi i file-t .csv u bë pa pasur nevojë të ndryshonim vendndodhjen e tij — ndryshe nga ajo që prisnim, PostgreSQL e pranoi rrugën e plotë të file-t të përdorur më herët në MySQL, pa kërkuar që file të ndodhej në një path të veçantë të shpërndarë të PostgreSQL.

Më pas, krijimi i funksionit për të llogaritur jetëgjatësinë mesatare kishte një ndryshim të vogël në sintaksë, por logjika dhe struktura mbetën të ngjashme me versionin në MySQL.

Pjesa më sfiduese ishte realizimi i bashkimit të tabelave midis dy databazave të ndryshme, pasi PostgreSQL nuk e mbështet këtë funksionalitet në mënyrë të drejtpërdrejtë. Për ta zgjidhur këtë, përdorëm një mekanizëm të quajtur **Foreign Data Wrappers (FDW)**, i cili lejon që një databazë PostgreSQL të lexojë tabela nga një databazë tjetër PostgreSQL si të ishin lokale. Procesi u realizua me komandat në vijim:

CREATE EXTENSION postgres\_fdw;

CREATE SERVER mondial\_server

FOREIGN DATA WRAPPER postgres\_fdw

OPTIONS (host 'localhost', dbname 'mondial', port '5432');

CREATE USER MAPPING FOR postgres

SERVER mondial\_server

OPTIONS (user 'postgres', password '. . .');

-- Kufizuam importin vetëm në tabelat e nevojshme për të shmangur probleme me tipin e personalizuar geocoord

IMPORT FOREIGN SCHEMA public LIMIT TO (country, isMember, organization)

FROM SERVER mondial\_server

INTO public;

Fillimisht aktivizuam extension postgres\_fdw. Më pas, krijuam një “server të jashtëm” me emrin mondial\_server, duke specifikuar hostin, portin dhe emrin e databazës së jashtme. Përcaktuam edhe kredencialet për t’u lidhur me këtë server. Me komandën IMPORT FOREIGN SCHEMA, importuam vetëm tabelat që na nevojiteshin (country, isMember, dhe organization) për të shmangur konflikte me tipe të personalizuara të dhënash që ekzistonin në databazën Mondial, si geocoord.

Pas këtij konfigurimi, krijimi i pamjes (view) ishte pothuajse identik me versionin në MySQL. Ndryshimi i vetëm ishte që tani nuk kishim më nevojë të përdornim prefiksin mondial. për të referuar tabelat, pasi ato tashmë ishin bërë pjesë e skemës publike përmes FDW dhe mund të thirreshin drejtpërdrejt me emrin e tyre.

# Data Warehouse

Në këtë fazë të projektit, kemi ndërtuar një **Data Warehouse (DWH)** të dedikuar për analizimin e ngjarjeve gjeopolitike që lidhen me vendet dhe organizatat ndërkombëtare të regjistruara në databazën burimore Mondial. Qëllimi kryesor ka qenë ndarja e qartë midis të dhënave transaksionale dhe të dhënave të strukturuara për analizë.

### Krijimi i strukturës

Fillimisht është krijuar databaza DWH\_Mondial, e cila përmban një model Star Schema me tri tabela dimensionale dhe një tabelë faktike:

* Dim\_Country: përmban informacion për shtetet (kodi, emri dhe forma e qeverisjes)
* Dim\_Organization: përmban organizatat ndërkombëtare, qytetin dhe vendin ku ndodhen
* Dim\_Date: përmban datat reale të pavarësive dhe themelimeve të organizatave
* Facts\_GeopoliticalEvents: përmban ngjarje të lidhura me vendet dhe organizatat në një datë të caktuar

Tabela faktike lidhet me të gjitha dimensionet përmes çelësave të huaj dhe përfshin tipin e ngjarjes (si p.sh. “Independence” ose “Organization Founded”).

### Skripta për krijimin e strukturës

Për krijimin e tabelave është përdorur skripti në vijim:

| DROP DATABASE IF EXISTS DWH\_Mondial; CREATE DATABASE DWH\_Mondial; USE DWH\_Mondial;  CREATE TABLE Dim\_Country (  CountryCode VARCHAR(4) PRIMARY KEY,  CountryName VARCHAR(35),  Government VARCHAR(120) );  CREATE TABLE Dim\_Organization (  OrgCode VARCHAR(12) PRIMARY KEY,  OrgName VARCHAR(80),  City VARCHAR(35),  CountryCode VARCHAR(4),  FOREIGN KEY (CountryCode) REFERENCES Dim\_Country(CountryCode) );  CREATE TABLE Dim\_Date (  DateID DATE PRIMARY KEY,  Year INT,  Month INT,  Day INT );  CREATE TABLE Facts\_GeopoliticalEvents (  EventID INT AUTO\_INCREMENT PRIMARY KEY,  CountryCode VARCHAR(4),  OrgCode VARCHAR(12),  DateID DATE,  EventType VARCHAR(50),  FOREIGN KEY (CountryCode) REFERENCES Dim\_Country(CountryCode),  FOREIGN KEY (OrgCode) REFERENCES Dim\_Organization(OrgCode),  FOREIGN KEY (DateID) REFERENCES Dim\_Date(DateID) ); |
| --- |

Pas krijimit të strukturës, janë shtuar edhe **indekse për performancë më të mirë** gjatë analizave:

| CREATE INDEX idx\_date ON Dim\_Date(DateID); CREATE INDEX idx\_event\_country ON Facts\_GeopoliticalEvents(CountryCode); CREATE INDEX idx\_event\_org ON Facts\_GeopoliticalEvents(OrgCode); |
| --- |

### Mbushja e të dhënave

Të dhënat janë importuar nga databaza burimore Mondial përmes komandave INSERT ... SELECT. Janë përdorur vetëm të dhëna që përmbajnë datë reale, pa përdorur snapshot-e apo të dhëna statike.

| INSERT INTO Dim\_Country (CountryCode, CountryName, Government) SELECT c.Code, c.Name, p.Government FROM Mondial.country c LEFT JOIN Mondial.politics p ON c.Code = p.Country ON DUPLICATE KEY UPDATE  CountryName = VALUES(CountryName),  Government = VALUES(Government);  INSERT INTO Dim\_Organization (OrgCode, OrgName, City, CountryCode) SELECT Abbreviation, Name, City, Country FROM Mondial.organization WHERE Country IS NOT NULL ON DUPLICATE KEY UPDATE  OrgName = VALUES(OrgName),  City = VALUES(City),  CountryCode = VALUES(CountryCode);  INSERT INTO Dim\_Date (DateID, Year, Month, Day) SELECT DISTINCT Independence,  YEAR(Independence),  MONTH(Independence),  DAY(Independence) FROM Mondial.politics WHERE Independence IS NOT NULL ON DUPLICATE KEY UPDATE  Year = VALUES(Year),  Month = VALUES(Month),  Day = VALUES(Day);  INSERT INTO Dim\_Date (DateID, Year, Month, Day) SELECT DISTINCT Established,  YEAR(Established),  MONTH(Established),  DAY(Established) FROM Mondial.organization WHERE Established IS NOT NULL ON DUPLICATE KEY UPDATE  Year = VALUES(Year),  Month = VALUES(Month),  Day = VALUES(Day);    INSERT INTO Facts\_GeopoliticalEvents (CountryCode, DateID, EventType) SELECT p.Country, p.Independence, 'Independence' FROM Mondial.politics p WHERE p.Independence IS NOT NULL ON DUPLICATE KEY UPDATE  EventType = VALUES(EventType);  INSERT INTO Facts\_GeopoliticalEvents (OrgCode, CountryCode, DateID, EventType) SELECT o.Abbreviation, o.Country, o.Established, 'Organization Founded' FROM Mondial.organization o WHERE o.Established IS NOT NULL AND o.Country IS NOT NULL ON DUPLICATE KEY UPDATE |
| --- |

### Automatizimi i procesit të mbushjes së DWH-së

Për të siguruar që DWH të përditësohet automatikisht me të dhëna të reja, është krijuar një skript Python, i cili ekzekutohet periodikisht çdo 5 minuta përmes një cron job.

Skripti lidhet me databazat MySQL dhe për çdo herë që ekzekutohet, kontrollon dhe shton të dhëna të reja në dimensione dhe tabelën faktike duke përdorur INSERT IGNORE.

Për t’u siguruar që ky proces të jetë i automatizuar, është krijuar një **skript (bash)** që konfiguron cron për ta ekzekutuar këtë skript çdo 5 minuta dhe ta logojë rezultatin.

### Përfundim

Me ndërtimin e këtij Data Warehouse dhe automatizimin e tij, kemi arritur të krijojmë një sistem të qëndrueshëm për analizimin e ngjarjeve historike dhe strukturave ndërkombëtare në mënyrë të organizuar dhe të përditësuar. Kjo strukturë lejon integrime të mëtejshme me dataset-e të jashtme për analiza më të avancuara dhe raportime të detajuara.