

Universiteti i Prishtinës “Hasan Prishtina”

Fakulteti Inxhinierisë Elektrike dhe Kompjuterike



Dokumentim teknik i projektit

Lënda: Big Data

Titulli i projektit: Structured data querying dhe dizajn i Data Warehouse

Emri profesorit/Asistentit

Emri & mbiemri studentëve / email adresa

Prof. Dr. Vigan Raça PhD Cand. Rafet Duriqi	1. Dionis Sylejmani	dionis.sylejmani1@student.uni-pr.edu
	2. Fjolla Kadriu	fjolla.kadriu@student.uni-pr.edu
	3. Leutrim Morina	leutrim.morina13@student.uni-pr.edu
	4. Linda Hasanaj	linda.hasanaj@student.uni-pr.edu
	5. Lum Hoxha	lum.hoxha@student.uni-pr.edu

Prishtinë, 2024/25

Përmbajtja

Hyrje.....	3
Pjesa kryesore.....	4
Structured Data Querying.....	4
Queries.....	4
Views.....	6
Stored procedure.....	7
Importimi i datasetit.....	7
Funksioni për konvertim.....	8
View në kuadër të datasetit.....	9
Dizajni i Data Warehouse.....	10
Databaza transaksionale.....	10
Data Warehouse.....	11
Automatizimi.....	12
Konkluzione (apo Përfundim).....	14

Hyrje

Ky projekt ka për qëllim ndihmimin në zhvillimin e një platforme të fuqishme për menaxhimin dhe analizimin e të dhënave, duke përdorur dy DBMS, MySQL dhe SQL Server. Projektit i janë caktuar dy objekte kryesore: përgatitja e mjedisit dhe realizimi i analizave strukturore të të dhënave, për të siguruar një rrjedhë të qëndrueshme dhe të automatizuar të të dhënave nga burimet origjinale në një Data Warehouse analitik.

Në pjesën e parë të projektit, ne kemi konfiguruar dy DBMS-të, MySQL dhe SQL Server. Më pas kemi importuar databazën Mondial nga burime të besueshme. Megjithatë, është e rëndësishme të theksohet se databaza Mondial përmban vetëm të dhëna deri në vitin 1990 dhe nuk është përditësuar që nga ajo kohë, mirëpo ne jemi siguruar që struktura e të dhënave të jetë plotësisht e përputhshme me kërkesat e projektit, pavarësisht këtij kufizimi kohor.

Në pjesën e dytë, projektit i kërkohet të zhvillojë një Data Warehouse me përfshirjen e tabelave Fact dhe Dimension, si dhe implementimin e një procesi të automatizuar për mbushjen e të dhënave nga databaza burimore në Data Warehouse. Po ashtu, do të implementohet një Data Flow që automatizon mbushjen e të dhënave në intervale të caktuara kohore të cilën e kemi implementuar edhe me MySQL edhe me python.

Ky dokumentim synon të prezantojë detajet e përgatitjes dhe implementimit të projektit, duke ofruar një pasqyrë të plotë të procesit të punës, metodologjisë dhe teknologjive të përdorura.

Pjesa kryesore

Structured Data Querying

Queries

Në këtë seksion janë zhvilluar 5 queries që shërbejnë për të nxjerrë të dhëna specifike nga databaza Mondial, bazuar në kërkesat për analizë të avancuar, këto queries i kemi zbatuar edhe në SQL Server edhe në MySQL Workbench.

Query 1: Ky query shfaq të gjitha kryeqytetet për të cilat nuk ka të dhëna se kalon ndonjë lum përmes tyre, mirëpo janë pjesë e të paktën një organizate ndërkombëtare. Ky query përdor CTE për të ndarë procesin në dy hapa: Kryeqytetet (Krijohet një listë e kryeqyteteve duke përdorur një RIGHT JOIN në mes të tabelave City dhe Country, për të marrë kryeqytetet) dhe pjesa tjetër Kryeqytetet_Organizata (filtron kryeqytetet që janë pjesë e një organizate ndërkombëtare duke përdorur EXISTS me tabelën Organization), këtë query po e paraqesim më poshtë se si e kemi strukturuar në SQL Server:

```
WITH Kryeqytetet AS (  
    SELECT DISTINCT c.[Name] AS Kryeqytetet  
    FROM Mondial.dbo.City c  
    RIGHT JOIN Mondial.dbo.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 Mondial.dbo.Organization o WHERE o.City = k.Kryeqytetet  
    )  
)  
  
SELECT ko.Kryeqytetet_O  
FROM Kryeqytetet_Organizat ko  
WHERE NOT EXISTS (  
    SELECT 1 FROM Mondial.dbo.located l WHERE l.City = ko.Kryeqytetet_O  
)  
);
```

Rezultati i këtij query është paraqitur në figurën më poshtë:

	Kryqytetet_O
1	Addis Ababa
2	Amman
3	Bangkok
4	Brussels
5	Caracas
6	Damascus
7	Gaborone
8	Guatemala City
9	Jerusalem
10	Kathmandu
11	Lima
12	Luxembourg
13	Madrid
14	Mexico City
15	Minsk
16	Prague
17	Pretoria
18	Riyadh
19	Santiago
20	Tegucigalpa
21	Tehran
22	Washington

Figura 1: Rezultati nga ekzekutimi i query-t të parë.

Query 2: Ky query liston lumenjtë që rrjedhin nëpër shtet të cilat nuk janë anëtarë të NATO-s, kanë dalje në det dhe kanë më shumë se 10 lumenjë që kalojnë nëpër to. Query është implementuar duke përdorur subqueries për të filtruar të dhënat nga tabela geo_river dhe country. Përdor JOIN mes country dhe geo_sea për të marrë vetëm ato shtete që kanë dalje në det. Përfshin shtetet anëtare të NATO-s përmes një subquery që referon tabelën isMember. Përfshin vetëm ato shtete që kanë më shumë se 10 lumenjë, duke përdorur GROUP BY dhe HAVING COUNT mbi tabelën geo_river.

Query 3: Ky query liston të gjitha detet në të cilat nuk ka asnjë ishull mirëpo shtetet të cilat kufizohet ai det janë anëtare në NATO ose BE. Ky query në databazën Mondial shfaq emrat e deteve që nuk përmbajnë asnjë ishull dhe kufizohen vetëm nga shtete që janë anëtare të NATO-s ose BE-së. Për ta arritur këtë, përdoret funksioni NOT EXISTS për të përjashtuar detet me ishuj nga tabela islandIn, dhe për të siguruar që çdo shtet që kufizon detin është pjesë e organizatave ndërkombëtare NATO ose EU, përmes tabelave geo_Sea, Country dhe isMember. Gjithashtu, për të siguruar që deti të jetë i lidhur me të paktën një shtet, kontrollohet ekzistenca e tij në tabelën geo_Sea.

Query 4: Ky query paraqet çiftet e shteteve të cilat janë anëtare të NATO-s dhe kufizohen me të njëjtin det dhe liqen. Duke përdorur JOIN mbi tabelat isMember, geo_Sea, geo_Lake dhe Organization, sigurohet që të dy shtetet në çift të jenë pjesë e NATO-s, të kenë akses në të njëjtin det nga tabela geo_Sea, dhe të kufizohen me të njëjtin liqen nga tabela geo_Lake. Kushti im1.Country < im2.Country përdoret për të shmangur përsëritjen e çifteve në rend të kundërt.

Query 5: Ky query shfaq 5 malet më të larta që ndodhen në vendet e Ballkanit, duke përjashtuar Serbinë. Duke përdorur një JOIN mes tabelave mountain dhe geo_mountain, identifikohen

malet që gjenden në shtetet e specifikuara me kode ISO. Përdoret funksioni MAX() për të marrë lartësinë maksimale për secilin mal dhe GROUP BY për t'i grupuar sipas emrit. Rezultatet renditen në mënyrë zbritëse sipas lartësisë dhe kufizohen me TOP 5 për të marrë vetëm pesë më të lartat.

Në përgjithësi, këto queries ilustrojnë aftësinë për të strukturuar queries komplekse mbi një databazë reale gjeografike, duke shfrytëzuar SQL për kërkime të avancuara dhe filtrime logjike të të dhënave.

Views

Në këtë pjesë janë krijuar 5 views dhe duke pasur parasysh se jo çdo query mund të kthehet automatikisht në view, na është dashur që 3 nga 5 views t'ua ndryshojmë strukturën dhe t'i përshtatim si views të rregullta, këto views i kemi zbatuar edhe në SQL Server edhe në MySQL Workbench.

Secila view e kthen rezultatin e njëjtë me queryn përkatëse, ndërsa më poshtë kemi paraqitur View 3:

```
CREATE VIEW View3 AS

SELECT s.Name AS Sea_Name

FROM Sea s

WHERE

    s.Name NOT IN (SELECT DISTINCT Sea FROM islandIn WHERE Sea IS NOT NULL)

    AND s.Name IN (

        SELECT gs.Sea

        FROM geo_Sea gs

        JOIN (

            SELECT DISTINCT Country

            FROM isMember

            WHERE Organization IN ('NATO', 'EU')

        ) AS valid_countries ON gs.Country = valid_countries.Country

    GROUP BY gs.Sea

    HAVING COUNT(*) = (

        SELECT COUNT(*)

        FROM geo_Sea gs2

        WHERE gs2.Sea = gs.Sea
```

```
)
);
```

Kurse rezultati i kësaj view është paraqitur në figurën e mëposhtme:

Results		Messages	
	Sea_Name		
1	Skagerrak		

Figura 2: Rezultati nga ekzekutimi i view-së së 3-të.

Stored procedure

Procedura Procedura është ndërtuar për të filtruar të dhënat nga View4, bazuar në tre parametra hyrës: SeaName, LakeName, dhe Country1. Këta parametra mund të përdoren individualisht ose të kombinuar për të gjetur shtetet anëtare të NATO-s që ndajnë të njëjtin det dhe liqe. Nëse ndonjë parametër lihet NULL, ai filtrohet automatikisht jashtë kërkesës përmes kushteve IS NULL OR ... LIKE, duke e bërë procedurën fleksibile dhe të përshtatshme për kërkesa të ndryshme.

Stored procedure është paraqitur më poshtë:

```
CREATE PROCEDURE dbo.Procedura
    @SeaName NVARCHAR(100),
    @LakeName NVARCHAR(100),
    @Country1 NVARCHAR(100)
AS
BEGIN
    SELECT v.Country1, v.Country2, v.Sea, v.Lake
    FROM View4 v
    WHERE (@SeaName IS NULL OR v.Sea LIKE '%' + @SeaName + '%')
        AND (@LakeName IS NULL OR v.Lake LIKE '%' + @LakeName + '%')
        AND (@Country1 IS NULL OR v.Country1 LIKE '%' + @Country1 + '%')
    ORDER BY v.Country1, v.Country2;
END;
```

Importimi i datasetit

Në këtë pjesë të projektit, kemi zgjedhur një dataset të shkarkuar nga Kaggle, i cili përmban të dhëna ditore të temperaturave në qytete të ndryshme të botës nga viti 1995 deri në vitin 2020.

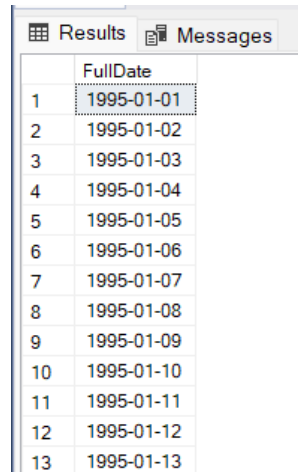
Dataset-i është në format CSV dhe përmban rreth 2 milionë rekorde, me këto kolona: Region, Country, State, City, Month, Day, Year, AvgTemperature. Ky dataset është importuar në bazën e të dhënave BigData_Datasets. Procesi i importimit u realizua përmes Microsoft SQL Import Wizard, e njëjta procedurë është ndjekur edhe në MySQL Workbench mirëpo këtu për shkak të shpejtësisë së DBMS na është dashur të importojmë vetëm 700k rekorde, duke ndjekur hapat kryesor të zgjedhjes së file-it, konfigurimit të kolonave dhe importimit të të dhënave në mënyrë të strukturuar.

Funksioni për konvertim

Qëllimi i funksionit të krijuar është të konvertojë vitin, muajin dhe ditën të cilat janë të dhëna të tipit int në një format të vetëm të tipit DATE, i cili mund të përdoret për analiza të mëtejshme mbi të dhënat kohore. Funksioni ConvertToDate pranon tre parametra hyrës: Year, Month dhe Day dhe rikthen një vlerë të tipit DATE të formatit YYYY-MM-DD. Ky funksion është paraqitur në vazhdim:

```
CREATE FUNCTION dbo.ConvertToDate (  
    @Year INT,  
    @Month INT,  
    @Day INT  
)  
RETURNS DATE  
AS  
BEGIN  
    DECLARE @FullDate DATE;  
    SET @FullDate = TRY_CAST(  
        CONCAT(  
            @Year, '-',  
            RIGHT('0' + CAST(@Month AS VARCHAR), 2), '-',  
            RIGHT('0' + CAST(@Day AS VARCHAR), 2)  
        ) AS DATE  
    );  
  
    RETURN @FullDate;  
END;
```


Kurse, kur thirret ky funksion në rezultat na shfaqet një kolonë e cila i ka këto të dhëna të konvertuara në tipin DATE. Në figurën e mëposhtme është paraqitur rezultati:



	FullDate
1	1995-01-01
2	1995-01-02
3	1995-01-03
4	1995-01-04
5	1995-01-05
6	1995-01-06
7	1995-01-07
8	1995-01-08
9	1995-01-09
10	1995-01-10
11	1995-01-11
12	1995-01-12
13	1995-01-13

Figura 3: Rezultati nga ekzekutimi i funksionit të krijuar.

View në kuadër të datasetit

Ky view realizon join në mes të tabelës city_temperature që është pjesë e databazës BigData_Datasets dhe tabelave Country dhe isMember nga databaza Mondial. Qëllimi i view-it është të paraqesë temperaturën mesatare në muajin prill për vitin 1995 në kryeqytetet e shteteve që janë anëtare të NATO-s. View-i për zgjedh qytetet që përputhen si kryeqytete në të dy burimet, dhe llogarit mesataren e temperaturave për ato që plotësojnë kriterin e organizatës. View-i është paraqitur në vazhdim:

```
CREATE VIEW Temperature AS
SELECT
    c.Capital AS Capital,
    ROUND(AVG(t.AvgTemperature), 2) AS AvgAprilTempFahrenheit
FROM
    city_temperature t
JOIN
    Mondial.dbo.Country c ON t.City = c.Capital AND t.Country = c.Name
JOIN
    Mondial.dbo.isMember m ON c.Code = m.Country
WHERE
    m.Organization = 'NATO'
```

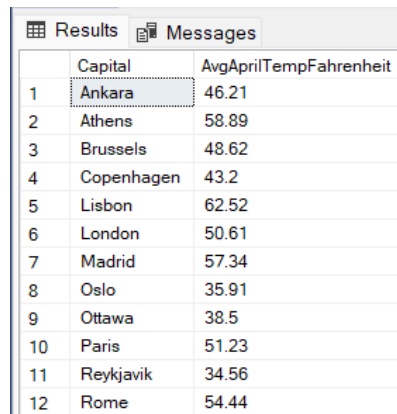
AND t.Month = 4

AND t.Year = 1995

GROUP BY

c.Capital;

Kurse rezultati është paraqitur në figurën e mëposhtme:



	Capital	AvgAprilTempFahrenheit
1	Ankara	46.21
2	Athens	58.89
3	Brussels	48.62
4	Copenhagen	43.2
5	Lisbon	62.52
6	London	50.61
7	Madrid	57.34
8	Oslo	35.91
9	Ottawa	38.5
10	Paris	51.23
11	Reykjavik	34.56
12	Rome	54.44

Figura 4: Rezultati nga ekzekutimi i view.

Dizajni i Data Warehouse

Databaza transaksionale

Ne kemi dizajnuar një databazë transaksionale BankDB e cila është projektuar për të menaxhuar aspektet kryesore të operacioneve bankare. Ajo përfshin tabela që përfaqësojnë entitetet kryesore si klientët, llogaritë bankare, degët, punonjësit, transaksionet, kreditë dhe pagesat. Marrëdhëniet midis këtyre entiteteve janë ndërtuar nëpërmjet foreign keys për të siguruar integritet të të dhënave. Ndërsa skema relacionale e paraqet një mënyrë më të qartë sa i përket lidhjeve midis tabelave, duke mundësuar një strukturë të organizuar dhe funksionale për menaxhimin efikas të proceseve. Në figurën e mëposhtme është paraqitur skema relacionale e databazës transaksionale:

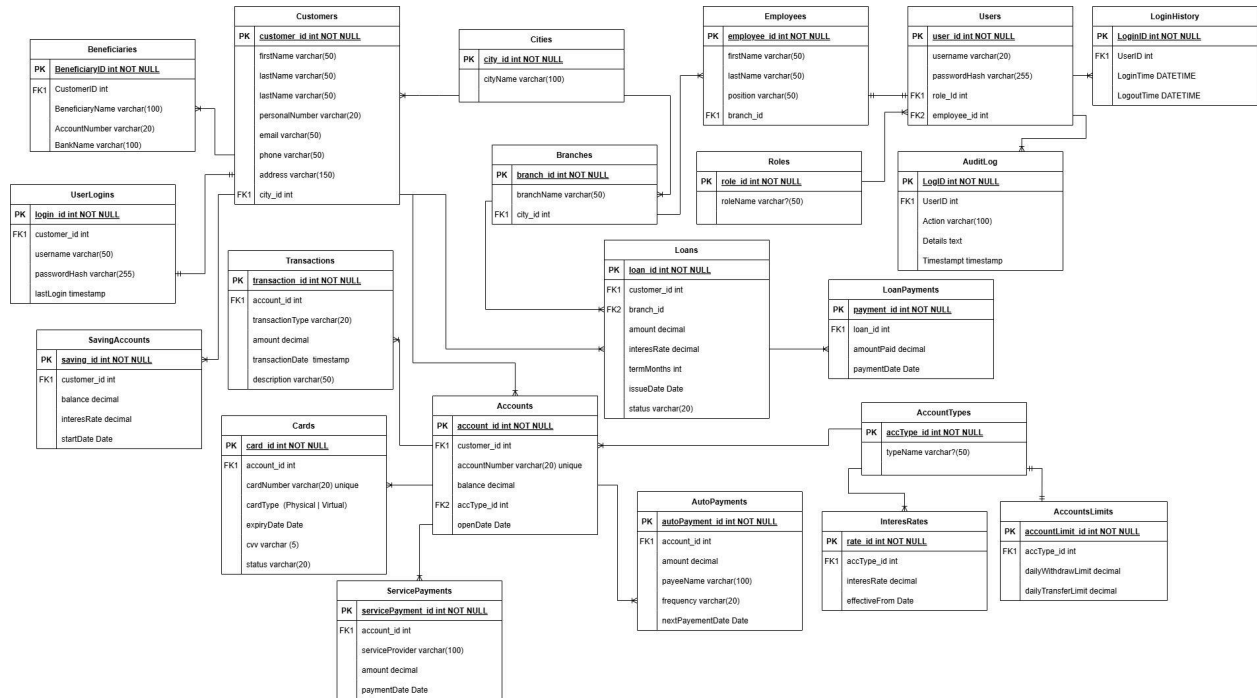


Figura 5: Skema relacionale e databazës transaksionale

Data Warehouse

Për ndërtimin e sistemit analitik është krijuar një Data Warehouse me emrin `dwh_bankdb`, i cili përmban të dhëna të përpunuara dhe të organizuar për analizë të avancuar dhe raportim. Kjo databazë është krijuar mbi modelin Star Schema, ku një tabelë qendrore (Fact Table) lidhet me disa tabela dimensionale.

Fact Table: `FactTransactions` përmban informacionin mbi transaksionet e bankës, përfshirë tipin e transaksionit, shumën, datën dhe përshkrimin. Kjo tabelë përmban lidhje me tabelat dimensionale për të mundësuar analizë për çdo transaksion në kontekstin e klientit, llogarisë dhe tipit të llogarisë.

Dimension Tables: `DimCustomers` përmban informacione bazë të klientëve si emri, mbiemri dhe qyteti. Kjo tabelë na ndihmon në analizat e transaksioneve sipas klientëve dhe vendndodhjes së tyre.

`DimAccounts` përmban të dhëna për llogaritë bankare, na ndihmon të analizojmë transaksionet sipas llogarive.

`DimAccountTypes` përmban të dhëna mbi llojet e llogarive bankare, na ndihmon analizën sipas tipeve të llogarisë.

Në këtë pjesë të projektit janë përdorur unclustered indexes mbi fushat si CityName, LastName, AccountNumber dhe TypeName në tabelat dimensionale. Këto fusha janë të rëndësishme për kërkime dhe filtrime gjatë analizave të të dhënave, por nuk janë fusha unike ose të përshtatshme për indeksim clustered. Për këtë arsye, përdorimi i indekseve unclustered ndihmon në përmirësimin e performancës së kërkesave pa ndikuar në mënyrën se si ruhen fizikisht të dhënat në tabelë.

Automatizimi

Për të realizuar automatizimin e rrjedhës së të dhënave nga databaza burimore BankDB në Data Warehouse-in analitik dwh_bankdb, është ndërtuar një procedurë e personalizuar me emrin autodataflow(). Kjo procedurë ka për qëllim që në mënyrë periodike të fshijë të dhënat ekzistuese në tabelat dimensionale dhe tabelën faktike përmes komandës TRUNCATE, dhe më pas të mbushë ato me të dhëna të përditësuara nga burimi. Gjatë ekzekutimit të saj, përkohësisht çaktivizohen kontrollet e çelësave të huaj (FOREIGN_KEY_CHECKS = 0) për të shmangur konflikte të mundshme gjatë fshirjes dhe mbushjes së të dhënave, dhe ato aktivizohen sërish në fund të procedurës.

Konkretisht, të dhënat për klientët, llogaritë, dhe tipet e llogarive ruhen në tabelat dimensionale DimCustomers, DimAccounts dhe DimAccountTypes, ndërsa transaksionet ruhen në tabelën faktike FactTransactions, duke ruajtur edhe lidhjet me dimensionet përkatëse. Për të siguruar ekzekutim automatik të kësaj procedure pa ndërhyrje manuale, është krijuar një event (scheduler) ev_autodataflow, i cili planifikohet të ekzekutohet çdo 5 minuta duke përdorur mekanizmin e ngulitur të MySQL EVENT SCHEDULER. Ky automatizim i vazhdueshëm garanton që të dhënat në Data Warehouse të jenë gjithmonë të përditësuara, gjë që përmirëson saktësinë e analizave dhe vizualizimeve të bazuara në këtë strukturë.

Përveç job-it të krijuar në MySQL, ne kemi krijuar edhe një job me anë të python-t i cili e automatizon procesin e përditësimit të data warehouse-it. Skripti Python (run_autodataflow.py) lidhet me SQL Server dhe ekzekuton procedurën e ruajtur autodataflow për të automatizuar proceset e përditësimit të të dhënave në bazën e të dhënave DWH_BankDB. Skripti siguron përditësime të suksesshme. Task Scheduler përdoret për të planifikuar dhe automatizuar ekzekutimin e këtij skripti në kohë të caktuara, duke mundësuar automatizimin e procesit pa ndërhyrje manuale. Skripta në python është paraqitur më poshtë:

```
import pyodbc
from datetime import datetime

try:
    conn = pyodbc.connect(
        "Driver={SQL Server};"
        "Server=FJOLLA\\SQLEXPRESS;"
        "Database=DWH_BankDB;"
        "Trusted_Connection=yes;"
    )
```

```
cursor = conn.cursor()
cursor.execute("EXEC autodataflow;")
conn.commit()

print(f'[{datetime.now()}] Procedura 'autodataflow' u ekzekutua me sukses.")

except Exception as e:
    print(f'[{datetime.now()}] Gabim gjatë ekzekutimit: {e}')

finally:
    try:
        cursor.close()
        conn.close()
    except:
        pass
```

Konkluzione (apo Përfundim)

Në përfundim, ky projekt arriti me sukses qëllimin e tij për të ndërtuar një mjedis të qëndrueshëm për menaxhimin dhe analizimin e të dhënave duke përdorur MySQL dhe SQL Server. Gjatë punës, u aplikuan teknika të avancuara si integrimi i të dhënave, normalizimi, ndërtimi i një Data Warehouse me fact tables dhe dimension tables, si dhe automatizimi i rrjedhës së të dhënave përmes stored procedures dhe event scheduler. Rezultatet treguan që platforma mund të analizojë të dhënat në mënyrë të shpejtë dhe efikase, duke siguruar bazën për raportim dhe vendimmarrje të informuar.

Punët e ardhshme përfshijnë integrimin e më shumë burimeve të të dhënave, zhvillimin e një dashboard-i vizualizues për analizë të avancuar, si dhe përmirësimin e mekanizmave të mirëmbajtjes dhe refresh-it të të dhënave në Data Warehouse në kohë reale. Këto zgjerime do të kontribuojnë në rritjen e shkallës dhe performancës së sistemit analitik.