Brandeis University

International Business School

Report 4: Advanced SQL

Bus 211A-1 - Foundations of Data Analytics

Linoy Noikas, Salman Haider, Sana Ijaz, Kyle Allsopp

November 12th, 2023

Table of Contents

Part 1: Business Proposition 2
Part 2: Focus and Growth 4
Part 3: Views 6
Part 4: Window Function 10
Part 5: Explainer Diagram 12
Part 6: Speed Queries 13
Part 7: Procedures 15
Part 8: Triggers 21

Part 1: Business Proposition

Our consultancy specializes in public health and health care data analysis. Our goal is to understand health care and educate the public on the health care situation across the globe. Our aim is to provide recommendations to improve healthcare outcomes and improve population well being at large through our data exploration and analysis.

Our consultancy wanted to focus on an important social and wellbeing issue and knowing that a common interest was Healthcare Analytics, we decided to focus on this credible dataset from the World Bank. This source is widely recognized and respected in the global health area and has an international reputation that we can trust. We can be rest assured that this data is legitimate and captures most countries of the world. We have extracted several databases from the World Bank such as Life Expectancy by Countries, and Nominal GDP by Countries and Access to Clean Water. Using this data from the World Bank will provide us with a more accurate, and well trusted output.

We will be investigating human life expectancy by country from the years 2001-2019 and looking at different factors that affect it (Prevelence of Undernourishment, CO2, Health Expenditure %, Education Expenditure %, Unemployment, GDP, Injuries, Access to Clean Water, Communicable, NonCommunicable). We will be predicting life expectancy based on the variables we have in the dataset. We will be providing different graphs, and visualizations to show our analysis. Our consultancy will be mainly using Tableau Prep to clean our data, and produce visualization analysis. In order to provide a better understanding of the data, our consultancy will also be using Python to generate histograms to understand the data better.

Part 2: Focus and Growth

2.1 Audience

Our potential clients include international organizations such as the World Bank or WHO, non-governmental organizations, charities, or even just general businesses looking for the next place to expand their business. This set of potential clients will all be able to gain useful insights out of our data to help determine where to distribute their resources or operate as a business. We are also hoping to work with Moderna and Pfizer to see if we can provide specific interventions to regions or specific countries such as medicine and vaccines. With our analyses we hope to assist these organizations in optimizing resource allocation, shaping policy development, and addressing disparities among countries.

What sets us apart is our commitment to utilizing reputable data sources, such as the World Bank, ensuring the legitimacy and reliability of our analyses. Our ability to generate actionable recommendations and provide clear, data-driven insights empowers our clients to make informed decisions, optimize resource allocation, and contribute to improved healthcare and population well-being on a global scale.

2.2 Potential Areas of Growth

Our consultancy project, specializing in public health and healthcare data analysis, has significant growth potential in serving a diverse range of clients and organizations. These potential clients include healthcare providers, insurers, pharmaceutical companies, government health agencies, NGOs, health tech startups, academic institutions, corporate wellness programs, global health organizations, epidemiological research organizations, and environmental agencies. These entities would be interested in collaborating with our consultancy to leverage our data insights for various purposes, including improving patient care, enhancing insurance offerings, informing policy decisions, advancing research, and addressing public health challenges. By tailoring our services to meet the unique needs of each client group, we can position our consultancy as a valuable partner in improving healthcare outcomes and public well-being.

In addition to the previously mentioned growth avenues, our consultancy can further expand its services by leveraging data on unemployment rates, access to clean water, GDP, and education expenditure. This data enables us to offer labor market analysis, economic impact assessments, and support for water and sanitation initiatives. Moreover, we can specialize in evaluating education policies, align with Sustainable Development Goals, aid regional development strategies, and assess social impacts for investors and philanthropic organizations. Our expertise can also assist global businesses in expansion efforts, partner with community development programs, and facilitate sustainability reporting for companies, demonstrating the breadth of our consultancy's capabilities and its potential to make a meaningful impact on diverse sectors.

Going forward with this project we will continue to work on refining and expanding our dataset to better achieve these goals. We look to incorporate data on areas such as crime, birth rates, pollution, and potentially mental health among others. Adding additional data could help us to refine general recommendations for countries, or specific recommendations for clients our consultancy is working with at a given time. This could also help to address issues we have encountered with a few columns of our data that needed to be removed due to having too many nulls. We will continue to analyze our data as we begin working more with My SQL and make additional changes to our dataset as needed

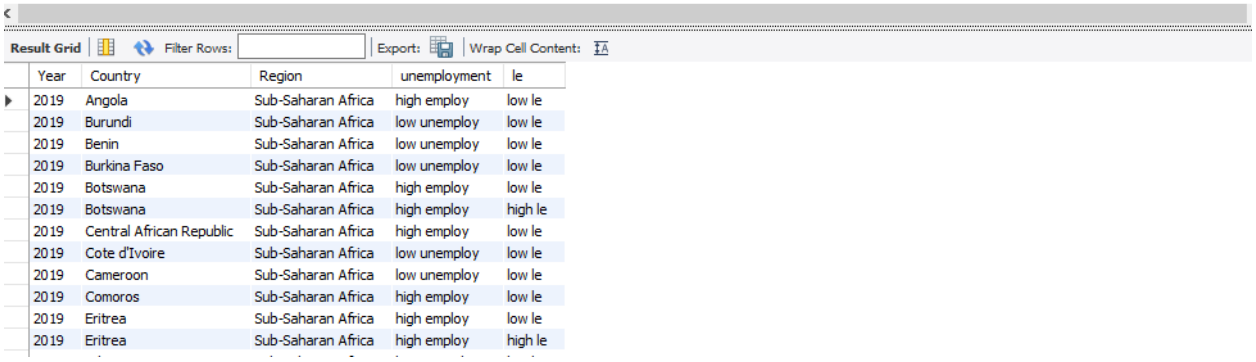
Part 3: Views

In this section, we created 3 views from multiple tables to compare Life Expectancy to Unemployment rates, Access to clean water, and GDP in billions by country. We wanted to focus on the Sub Saharan Africa region to see if there is any correlation to low life expectancy and high unemployment rate, low access to clean water, or low GDP value. Below are our queries for the 3 views along with the output from MySQL.

View 1

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `admin`@`localhost`
    SQL SECURITY DEFINER
VIEW `le`.`le_sub_saharan_africa` AS
    SELECT DISTINCT
        `le`.`economy_table`.`Year` AS `Year`,
        `le`.`countries_table`.`Country` AS `Country`,
        `le`.`regions_table`.`Region` AS `Region`,
        IF((`le`.`economy_table`.`Unemployment` < 5),
            'low unemploy',
            'high employ') AS `unemployment`,
        IF((`le`.`lifeexpectancy_table`.`LifeExpectancyValue` <
65),
            'low le',
            'high le') AS `le`
    FROM
        (((`le`.`economy_table`
        JOIN `le`.`countries_table` ON
((`le`.`economy_table`.`CountryID` =
`le`.`countries_table`.`CountryID`)))
        JOIN `le`.`lifeexpectancy_table` ON
((`le`.`economy_table`.`CountryID` =
`le`.`lifeexpectancy_table`.`CountryID`)))
        JOIN `le`.`regions_table` ON
((`le`.`countries_table`.`RegionID` =
`le`.`regions_table`.`RegionID`)))
    WHERE
        ((`le`.`economy_table`.`Year` = '2019')
            AND (`le`.`regions_table`.`Region` = 'Sub-Saharan
Africa'))
```

```
1 • USE le;
2 • SELECT * FROM `le_sub_saharan_africa`;
```



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Year	Country	Region	unemployment	le
2019	Angola	Sub-Saharan Africa	high employ	low le
2019	Burundi	Sub-Saharan Africa	low unemploy	low le
2019	Benin	Sub-Saharan Africa	low unemploy	low le
2019	Burkina Faso	Sub-Saharan Africa	low unemploy	low le
2019	Botswana	Sub-Saharan Africa	high employ	low le
2019	Botswana	Sub-Saharan Africa	high employ	high le
2019	Central African Republic	Sub-Saharan Africa	high employ	low le
2019	Cote d'Ivoire	Sub-Saharan Africa	low unemploy	low le
2019	Cameroon	Sub-Saharan Africa	low unemploy	low le
2019	Comoros	Sub-Saharan Africa	high employ	low le
2019	Eritrea	Sub-Saharan Africa	high employ	low le
2019	Eritrea	Sub-Saharan Africa	high employ	high le

View 2

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `admin`@`localhost`
    SQL SECURITY DEFINER
VIEW `le`.`le_sub_saharan_africa_2` AS
    SELECT DISTINCT
        `le`.`environment_table`.`Year` AS `Year`,
        `le`.`countries_table`.`Country` AS `Country`,
        `le`.`regions_table`.`Region` AS `Region`,
        IF((`le`.`environment_table`.`AccessToCleanWater` < 10),
            'lessthan10%',
            'none') AS `AccessToCleanWater`,
        IF((`le`.`lifeexpectancy_table`.`LifeExpectancyValue` <
65),
            'lowle',
            'highle') AS `le`
    FROM
        (((`le`.`environment_table`
        JOIN `le`.`countries_table` ON
((`le`.`environment_table`.`CountryID` =
`le`.`countries_table`.`CountryID`)))
        JOIN `le`.`lifeexpectancy_table` ON
((`le`.`environment_table`.`CountryID` =
`le`.`lifeexpectancy_table`.`CountryID`)))
        JOIN `le`.`regions_table` ON
((`le`.`countries_table`.`RegionID` =
`le`.`regions_table`.`RegionID`)))
    WHERE
```

```
        ((`le`.`environment_table`.`AccessToCleanWater` =
'lessthan10%')
            AND (`le`.`environment_table`.`Year` = 2019)
            AND (`le`.`regions_table`.`Region` = 'Sub-Saharan
Africa')
            AND
(`le`.`lifeexpectancy_table`.`LifeExpectancyValue` < 65))
    ORDER BY `le` DESC
```

```
1 • USE le;
2 • SELECT * FROM `le_sub_saharan_africa_2`;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Year	Country	Region	AccessToCleanWater	le
2019	Burkina Faso	Sub-Saharan Africa	lessthan10%	lowle
2019	Gabon	Sub-Saharan Africa	lessthan10%	lowle
2019	Mauritania	Sub-Saharan Africa	lessthan10%	lowle
2019	Namibia	Sub-Saharan Africa	lessthan10%	lowle
2019	Sudan	Sub-Saharan Africa	lessthan10%	lowle
2019	South Sudan	Sub-Saharan Africa	lessthan10%	lowle
2019	South Africa	Sub-Saharan Africa	lessthan10%	lowle
2019	Zambia	Sub-Saharan Africa	lessthan10%	lowle

View 3

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `admin`@`localhost`
    SQL SECURITY DEFINER
VIEW `le`.`le_sub_saharan_africa_3` AS
    SELECT DISTINCT
        `le`.`economy_table`.`Year` AS `Year`,
        `le`.`countries_table`.`Country` AS `Country`,
        `le`.`regions_table`.`Region` AS `Region`,
        `le`.`economy_table`.`GDP_Billions` AS `GDP_Billions`,
        `le`.`lifeexpectancy_table`.`LifeExpectancyValue` AS
`Le`
    FROM
        (((`le`.`economy_table`
        JOIN `le`.`countries_table` ON
((`le`.`economy_table`.`CountryID` =
`le`.`countries_table`.`CountryID`)))
        LEFT JOIN `le`.`lifeexpectancy_table` ON
((`le`.`economy_table`.`CountryID` =
`le`.`lifeexpectancy_table`.`CountryID`)))
```

```
        JOIN `le`.`regions_table` ON
((`le`.`countries_table`.`RegionID` =
`le`.`regions_table`.`RegionID`)))
    WHERE
        (`le`.`regions_table`.`Region` = 'Sub-Saharan Africa')
    ORDER BY `Le` , `le`.`economy_table`.`GDP_Billions` DESC
```

```
17
18 •  select * from `le_sub_saharan_africa_3`
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

Year	Country	Region	GDP_Billions	Le
2014	Sierra Leone	Sub-Saharan Africa	5.015	40.369
2013	Sierra Leone	Sub-Saharan Africa	4.92	40.369
2015	Sierra Leone	Sub-Saharan Africa	4.252	40.369
2018	Sierra Leone	Sub-Saharan Africa	4.085	40.369
2019	Sierra Leone	Sub-Saharan Africa	4.077	40.369
2016	Sierra Leone	Sub-Saharan Africa	3.863	40.369
2012	Sierra Leone	Sub-Saharan Africa	3.789	40.369
2017	Sierra Leone	Sub-Saharan Africa	3.719	40.369
2011	Sierra Leone	Sub-Saharan Africa	2.932	40.369
2010	Sierra Leone	Sub-Saharan Africa	2.578	40.369

Part 4: Window Function

Using the views we have created in part 3, we utilized the window function to aggregate the data. The first query uses AVG, MIN, MAX for GDP calculations for overall Sub Saharan Africa region for the year 2019. In the second query, AVG, and STDDEV functions are used for overall Sub Saharan Africa region life expectancy and GDP values. We used the ROUND function to round our values for a clearer understanding of the data.

```
USE `le`;
SELECT DISTINCT Region, Year,
     AVG(`GDP_Billions`) OVER (PARTITION BY `Year`)  AS
average_GDP,
    MIN(`GDP_Billions`) OVER (PARTITION BY `Year`) AS
minimum_GDP,
    MAX(`GDP_Billions`) OVER (PARTITION BY `Year`) AS
maximum_GDP
FROM `le_sub_saharan_africa_3`
WHERE Year = 2019;
```

```
23 •  USE `le`;
24 •  SELECT DISTINCT Region, Year,
25        AVG(`GDP_Billions`) OVER (PARTITION BY `Year`)  AS average_GDP,
26        MIN(`GDP_Billions`) OVER (PARTITION BY `Year`) AS minimum_GDP,
27        MAX(`GDP_Billions`) OVER (PARTITION BY `Year`) AS maximum_GDP
28    FROM `le_sub_saharan_africa_3`
29    WHERE Year = 2019;
30
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Region	Year	average_GDP	minimum_GDP	maximum_GDP
Sub-Saharan Africa	2019	40.12614148681053	0.427	474.517

```
USE `le`;
SELECT DISTINCT Region, Year,
     ROUND(AVG(`GDP_Billions`) OVER (PARTITION BY `Year`))  AS
average_GDP,
     ROUND(AVG(`Le`) OVER (PARTITION BY `Year`)) AS average_Le,
    ROUND(STDDEV(GDP_Billions) OVER (PARTITION BY `Year`)) AS
stddev_GDP
FROM `le_sub_saharan_africa_3`;
```

```sql
USE `le`;
SELECT DISTINCT Region, Year,
    ROUND(AVG(`GDP_Billions`) OVER (PARTITION BY `Year`))  AS average_GDP,
    ROUND(AVG(`Le`) OVER (PARTITION BY `Year`)) AS average_Le,
    ROUND(STDDEV(GDP_Billions) OVER (PARTITION BY `Year`)) AS stddev_GDP
FROM `le_sub_saharan_africa_3`;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

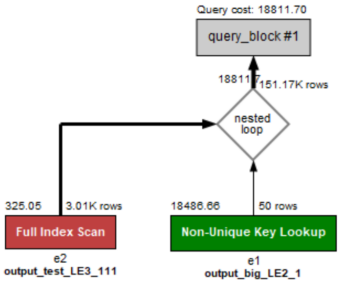
Region	Year	average_GDP	average_Le	stddev_GDP
Sub-Saharan Africa	2001	9	58	22
Sub-Saharan Africa	2002	10	58	23
Sub-Saharan Africa	2003	13	58	32
Sub-Saharan Africa	2004	16	58	42
Sub-Saharan Africa	2005	18	58	49
Sub-Saharan Africa	2006	22	58	56
Sub-Saharan Africa	2007	25	58	63
Sub-Saharan Africa	2008	28	58	68
Sub-Saharan Africa	2009	27	58	64
Sub-Saharan Africa	2010	32	58	81
Sub-Saharan Africa	2011	37	58	90
Sub-Saharan Africa	2012	38	58	93
Sub-Saharan Africa	2013	40	58	96
Sub-Saharan Africa	2014	42	58	101
Sub-Saharan Africa	2015	37	58	88
Sub-Saharan Africa	2016	35	58	76
Sub-Saharan Africa	2017	38	58	79
Sub-Saharan Africa	2018	39	58	85
Sub-Saharan Africa	2019	40	58	89

Part 5: Explainer Diagram

The explainer tool on MySQL shows a visualization of queries, and which tables, indexes, or views are used. Below we tested a query using our two tables, output table, with around 3000 rows, and our output big table, with around 150,000 rows. We used the explainer diagram to identify which table will be better for creating indexed columns to improve speed of queries. The explainer diagram, shows that the output big is green and the regular output table is red. The red full index scan means it is high especially for large indexes, and the green non unique key lookup means it is low to medium, low if the number of matching rows is small; higher as the number of rows increases. We have also texted the explainer tool in a query with an additional table to see if the results have changed. See below the Visual Explain Diagrams.

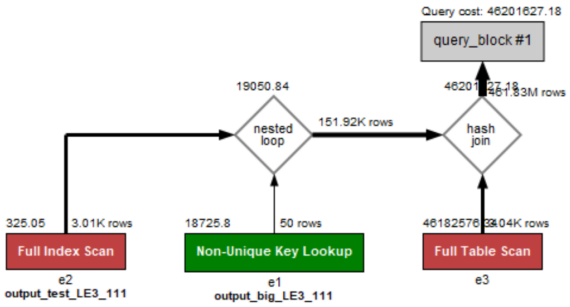
```
1 •  SELECT e1.LifeExpectancy, e2.Year, e2.CO2emissions
2    FROM output_big AS e1
3    JOIN `output - test` AS e2
4    ON e1.LifeExpectancy = e2.LifeExpectancy;
5
```

Visual Explain | Display Info: Read + Eval cost | Overview: | View Source:




```
1 •  SELECT e1.LifeExpectancy, e2.Year, e2.CO2emissions, e2.LifeExpectancy, e1.Year, e1.CO2emissions, e3.LifeExpectancyValue
2    FROM output_big AS e1
3    JOIN `output - test` AS e2
4    JOIN `lifeexpectancy_table` AS e3
5    ON e1.LifeExpectancy = e2.LifeExpectancy;
```

Visual Explain | Display Info: Read + Eval cost | Overview: | View Source:



Part 6: Speed Queries

In this section we created different indexes 1-3 columns using our output big table and our regular output table. See below queries for index creation.

```
CREATE INDEX  output_big_LE ON output_big(LifeExpectancy);
CREATE INDEX output_big_LE2_1 ON output_big(LifeExpectancy,
Year);
CREATE INDEX output_big_LE3_111 ON output_big(LifeExpectancy,
Year, CO2emissions);

CREATE INDEX output_test_LE ON `output - test`(LifeExpectancy);
CREATE INDEX output_test_LE2_1 ON `output -
test`(LifeExpectancy, Year);
CREATE INDEX output_test_LE3_111 ON `output -
test`(LifeExpectancy, Year, CO2emissions);
```

After creating the indexes we tested the speeds of our indexes and compared the speed improvement between the three indexes created from each table. We have found that the more columns are used in the index, the faster the query is, however overall the output_big table indexes are faster than the output little table indexes.

Testing Index Speed:
Output_big

Columns Used	Speed	Speed % Improvement
1	0.559	
2	0.314	44%
3	0.335	-7%

Output_little

Columns Used	Speed	Speed % Improvement
1	0.094	
2	0.093	1%
3	0.093	0%

We also tested complex queries using JOIN with vs. without our indexed columns from Output_big and from Output_little. Below our results for speed improvement. Overall there was an increase in speed using indexed columns rather than no indexed columns.

Testing Complex Query with JOIN with no indexes

Output_big

Big Table	Speed	Speed % Improvement
No Indexes	0.0056	
With indexes	0.0028	50%

Output_little

Little Table	Speed	Speed % Improvement
No indexes	0.013	
With Indexes	0.0075	42%

Part 7: Procedures

Stored Procedure for Adding a New Country

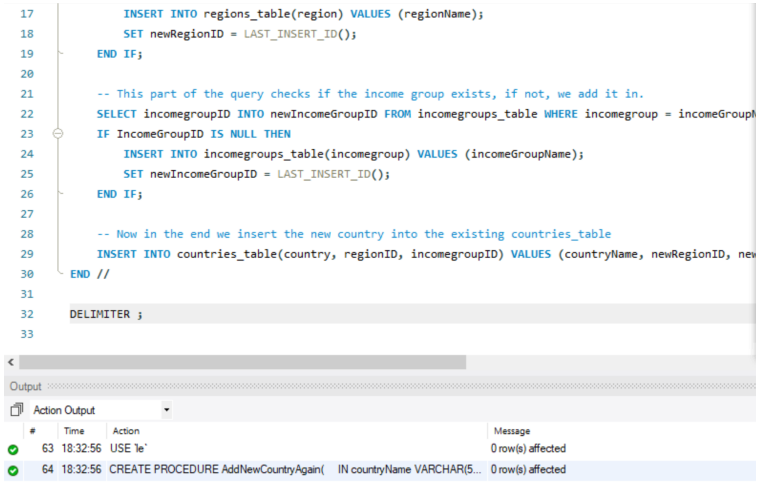
```
DELIMITER //
CREATE PROCEDURE AddNewCountryAgain(
    IN countryName VARCHAR(50),
    IN regionName VARCHAR(50),
    IN incomeGroupName VARCHAR(50)
)
BEGIN
    DECLARE newRegionID INT;
    DECLARE newIncomeGroupID INT;

    -- This part of the query checks if the region entered exists, if
not, we add it in.
    SELECT regionID INTO newRegionID FROM regions_table WHERE region
= regionName;
    IF newRegionID IS NULL THEN
        INSERT INTO regions_table(region) VALUES (regionName);
        SET newRegionID = LAST_INSERT_ID();
    END IF;

    -- This part of the query checks if the income group exists, if
not, we add it in.
    SELECT incomegroupID INTO newIncomeGroupID FROM
incomegroups_table WHERE incomegroup = incomeGroupName;
    IF IncomeGroupID IS NULL THEN
        INSERT INTO incomegroups_table(incomegroup) VALUES
(incomeGroupName);
        SET newIncomeGroupID = LAST_INSERT_ID();
    END IF;

    -- Now in the end we insert the new country into the existing
countries_table
    INSERT INTO countries_table(country, regionID, incomegroupID)
VALUES (countryName, newRegionID, newIncomeGroupID);
END //

DELIMITER
```

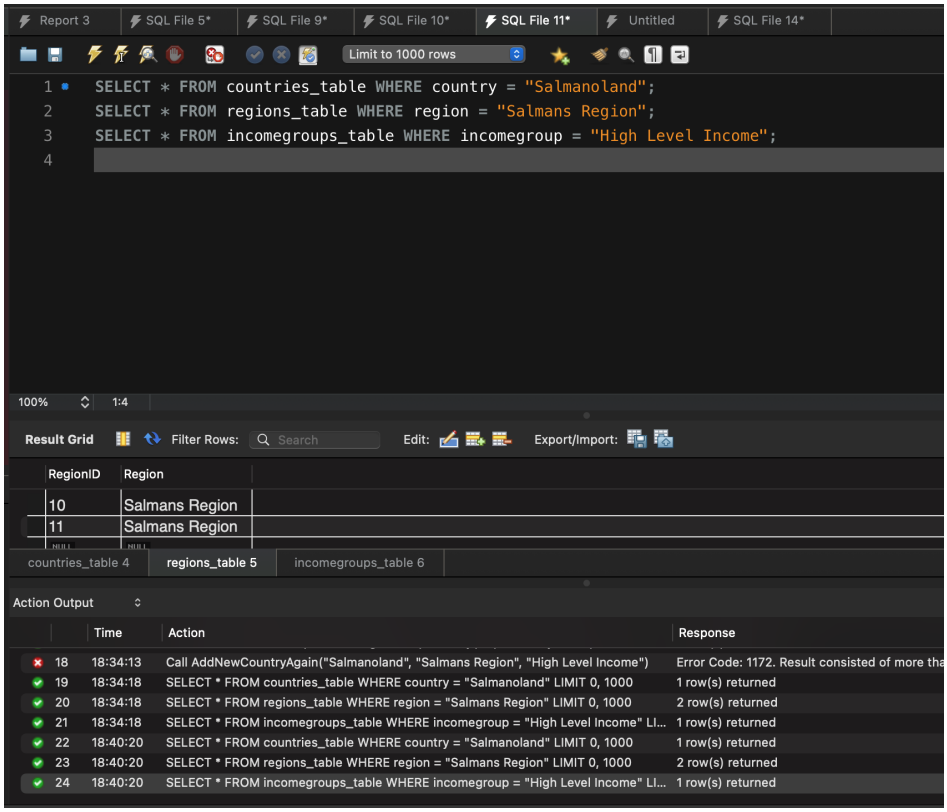
This procedure adds a new country to our database. It checks if a region and income group exists. If they don't exist, it adds them, auto increments them and then adds the country, regionID and incomegroupID to the countries_table

This is an example of how this Stored Procedure would be used:

```
CALL AddNewCountryAgain("Salmanistan", "Salmano Region", "High Income");
```

This Procedure is adding a new country, its region and income group to the countries_table.





Stored Procedure to Calculate and Update Average Life Expectancy

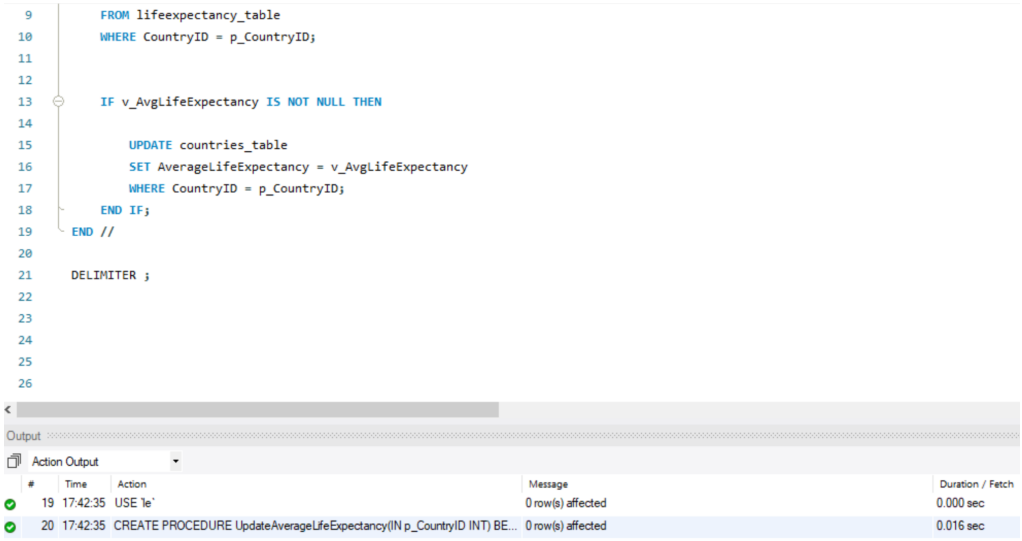
```
DELIMITER //

CREATE PROCEDURE UpdateAverageLifeExpectancy(IN p_CountryID INT)
BEGIN
    DECLARE v_AvgLifeExpectancy DOUBLE;
    SELECT AVG(LifeExpectancyValue) INTO v_AvgLifeExpectancy
    FROM lifeexpectancy_table
    WHERE CountryID = p_CountryID;
    IF v_AvgLifeExpectancy IS NOT NULL THEN
        UPDATE countries_table
        SET AverageLifeExpectancy = v_AvgLifeExpectancy
        WHERE CountryID = p_CountryID;
    END IF;
END //

DELIMITER ;
```

This stored procedure calculates the average life expectancy for a specific country based on the life expectancy data from the lifeexpectancy_table. It processes all available records for the given CountryID to compute the mean value of the life expectancy over the years. Once calculated, the average is then updated in a designated column (AverageLifeExpectancy) within the countries_table. This procedure is particularly useful for analyses that require a quick reference to a country's overall life expectancy without the need for real-time calculation. This gives us the average of it.

This is how you would call this stored procedure:

```
CALL UpdateAverageLifeExpectancy(123);
```

We would replace 123 with the actual CountryID for the country we are interested in. Before calling this procedure, we would make sure that the countries_table has a column to store the average life expectancy data.



Stored Procedure for adding record into Environment Table

```
DELIMITER //

CREATE PROCEDURE InsertEnvironmentRecord(
    IN p_CountryID INT,
    IN p_Year INT,
    IN p_AccessToCleanWater DOUBLE
)
BEGIN
```

```
    INSERT INTO environment_table (CountryID, Year,
AccessToCleanWater)
    VALUES (p_CountryID, p_Year, p_AccessToCleanWater);
END //

DELIMITER ;
```

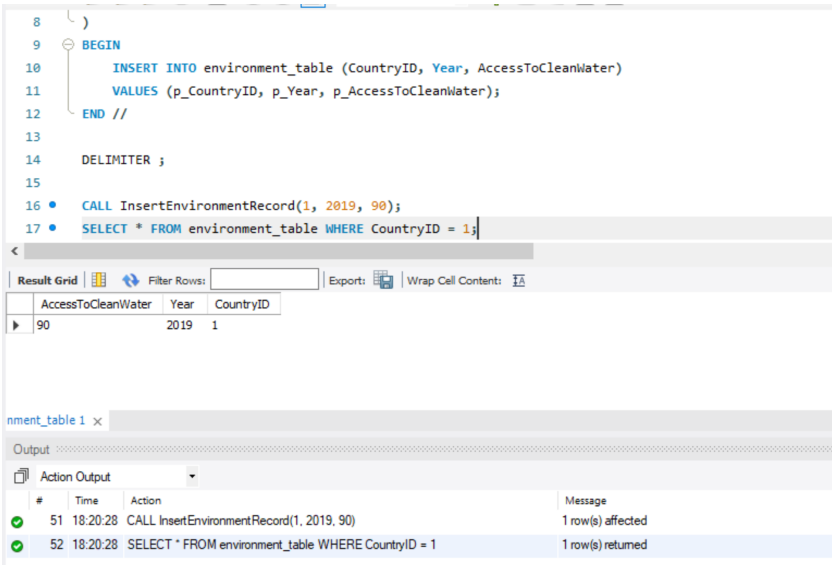
```
USE `le`;
DELIMITER //

CREATE PROCEDURE InsertEnvironmentRecord(
    IN p_CountryID INT,
    IN p_Year INT,
    IN p_AccessToCleanWater DOUBLE
)
BEGIN
    INSERT INTO environment_table (CountryID, Year, AccessToCleanWater)
    VALUES (p_CountryID, p_Year, p_AccessToCleanWater);
END //

DELIMITER ;
```

This stored procedure inserts a new record into the environment_table. It's designed to input data about a country's access to clean water for a specific year. This simplifies the process of adding new environmental data by ensuring that all necessary fields are populated correctly.

Here is how you would run this Procedure:

To insert a new environmental record for a country with a CountryID of 123 for the year 2021, where 95.5% of the population has access to clean water, you would use the following SQL command:

```
CALL InsertEnvironmentRecord(123, 2021, 95.5);
```


)
BEGIN
INSERT INTO environment_table (CountryID, Year, AccessToCleanWater)
VALUES (p_CountryID, p_Year, p_AccessToCleanWater);
END //
DELIMITER ;
CALL InsertEnvironmentRecord(1, 2019, 90);
SELECT * FROM environment_table WHERE CountryID = 1;
Result Grid
Filter Rows:
Export:
Wrap Cell Content:
AccessToCleanWater
Year
CountryID
90
2019
1
nment_table 1
Output
Action Output
#
Time
Action
Message
51 18:20:28 CALL InsertEnvironmentRecord(1, 2019, 90) 1 row(s) affected
52 18:20:28 SELECT * FROM environment_table WHERE CountryID = 1 1 row(s) returned

Part 8: Triggers

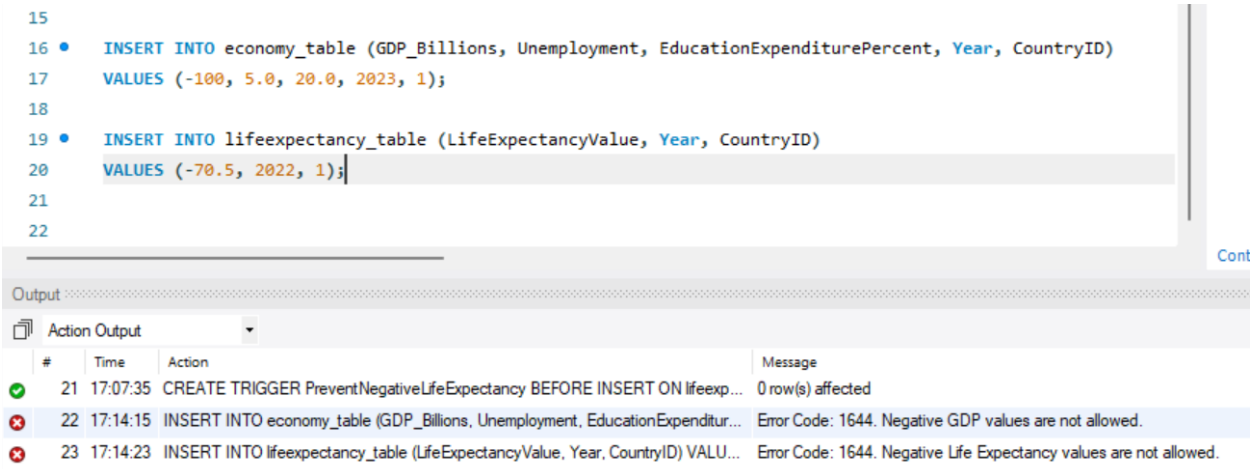
Insert Trigger

Preventing the addition of negative value of Life Expectancy and GDP using INSERT Trigger

```
DELIMITER //

CREATE TRIGGER PreventNegativeGDP
BEFORE INSERT ON economy_table
FOR EACH ROW
BEGIN
    IF NEW.GDP_Billions < 0 THEN
        SIGNAL SQLSTATE '45000'
             SET MESSAGE_TEXT = 'Negative GDP values are not
allowed.';
    END IF;
END;
//

CREATE TRIGGER PreventNegativeLifeExpectancy
BEFORE INSERT ON lifeexpectancy_table
FOR EACH ROW
BEGIN
    IF NEW.LifeExpectancyValue < 0 THEN
        SIGNAL SQLSTATE '45000'
         SET MESSAGE_TEXT = 'Negative Life Expectancy values are
not allowed.';
    END IF;
END;
//

DELIMITER ;
```

The "PreventNegativeGDP" and "PreventNegativeLifeExpectancy" triggers are designed to maintain data integrity in both the "Economy" and "LifeExpectancy" tables by preventing the insertion of records with negative values in the "GDP_Billions" and "LifeExpectancyValue" columns, respectively. This trigger proactively intercepts any insert operation and checks the values being inserted. If either of the values is negative, the trigger generates a custom error message specific to the table being modified, signaling that "Negative GDP values" or "Negative

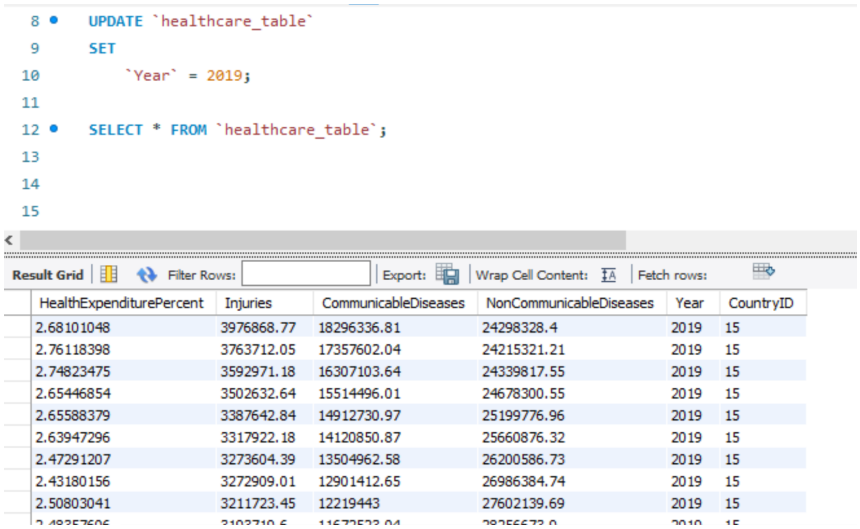
Life Expectancy values" are not allowed. By implementing these triggers, potential data entry errors or anomalies related to GDP and life expectancy are preemptively addressed, contributing to the accuracy and reliability of the database and supporting data quality standards.



In this query, we attempted to insert negative values for GDP and Life Expectancy into the database. The system promptly detected these invalid entries and generated custom error messages, signaling that negative values are not permitted. This demonstrates the successful operation of the triggers designed to maintain data integrity by preventing the insertion of negative values and validating the effectiveness of the implemented trigger mechanisms.

Update Trigger

```
USE `le`;
CREATE TRIGGER      before_2019_update_hc
BEFORE UPDATE ON `healthcare_table`
FOR EACH ROW INSERT INTO `Year`
SET
operation = 'update';

UPDATE `healthcare_table`
SET
     `Year` = 2019;

SELECT * FROM `healthcare_table`;
```



```
8  UPDATE `healthcare_table`
9  SET
10     `Year` = 2019;
11
12 SELECT * FROM `healthcare_table`;
13
14
15
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: | Fetch rows:

HealthExpenditurePercent	Injuries	CommunicableDiseases	NonCommunicableDiseases	Year	CountryID
2.68101048	3976868.77	18296336.81	24298328.4	2019	15
2.76118398	3763712.05	17357602.04	24215321.21	2019	15
2.74823475	3592971.18	16307103.64	24339817.55	2019	15
2.65446854	3502632.64	15514496.01	24678300.55	2019	15
2.65588379	3387642.84	14912730.97	25199776.96	2019	15
2.63947296	3317922.18	14120850.87	25660876.32	2019	15
2.47291207	3273604.39	13504962.58	26200586.73	2019	15
2.43180156	3272909.01	12901412.65	26986384.74	2019	15
2.50803041	3211723.45	12219443	27602139.69	2019	15

Creating the trigger, will be fired before any update on the table, and will insert a record into the year column with the operation set as update. The health care table will then be updated setting the year to 2019 in the second query. The last query is to show that the update trigger has worked successfully.

Delete Trigger

Creating an Archive Table using Delete Trigger to keep track of deleted data

First, we create the "ArchiveCountries" table that mirrors the structure of the "countries_table" and includes additional columns for archiving purposes, such as an "ArchivedAt" timestamp to record when the data was archived.

```
CREATE TABLE ArchiveCountries (
    CountryID INT PRIMARY KEY,
    Country VARCHAR(50) NOT NULL,
    RegionID INT,
    IncomeGroupID INT,
    ArchivedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

As the next step, we create a trigger that activates before records are deleted from the "countries_table"and copies the deleted records to the "ArchiveCountries" table.

```
DELIMITER //

CREATE TRIGGER CountryDeleteTrigger BEFORE DELETE ON
countries_table
FOR EACH ROW
BEGIN
    -- Insert the deleted record into the ArchiveCountries table
    INSERT INTO ArchiveCountries (CountryID, Country, RegionID,
IncomeGroupID)
    VALUES (OLD.CountryID, OLD.Country, OLD.RegionID,
OLD.IncomeGroupID);
END;
//

DELIMITER ;
```

This trigger, named "CountryDeleteTrigger," is set to activate before a record is deleted from the "countries_table" (BEFORE DELETE ON countries_table). It inserts the deleted record into the "ArchiveCountries" table, preserving the data along with the deletion timestamp.

Now, when someone deletes a record from the "countries_table", it will be archived in the "ArchiveCountries" table with the deletion timestamp, allowing us to retain historical data.

Checking the Delete Trigger:

In the screenshot below, we successfully deleted a specific record from the "countries_table" and confirmed its entry in the "ArchiveCountries" table, ensuring the preservation of historical data.

```
1 •  DELETE FROM countries_table WHERE CountryID = 161;
2
3 •  SELECT * FROM ArchiveCountries;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

CountryID	Country	RegionID	IncomeGroupID	ArchivedAt
161	New Country	1	1	2023-11-08 16:28:26
NULL	NULL	NULL	NULL	NULL