The project files for the step three are uploaded to the following folders in the GitHub repository:

1. ‘Cs306_last_step3_routines.sql’ is uploaded to the ‘MySQL’ folder
2. ‘Step3_log_file.pdf’ is uploaded to the ‘Log Files’ folder.

GitHub repository for the project: https://github.com/ihagverdi/Database-Systems-Project

Group 2:

Hagverdi Ibrahimli - 30014

Raman Afravi - 30061

Amil Kazımoğlu - 27891

Ayçelen Kaptan - 28826

Kaan Özyer - 26837

SQL statements:

View statement 1:

Purpose of the following ‘create view’ statement is to select countries with high affordability of purchasing cigarettes and then we project the columns: country name, GDP affordability, year the data corresponds to. The result of the statement exactly returns the projected column values.

```
‘CREATE VIEW high_affordability_countries AS
SELECT c.country_name, a.gdp_affordability, a.data_year
FROM affordability a
JOIN countries c ON a.iso_code = c.iso_code
WHERE a.gdp_affordability >= 20
GROUP BY c.country_name, a.gdp_affordability, a.data_year
ORDER BY a.gdp_affordability DESC’
```

View statement 2:

In the following statement we want to find low affordability of buying cigarettes and then we visualize in the columns: country, gdp affordability, data year that related to. The result is visualized in the column values.

```
‘CREATE VIEW: low_affordability_countries AS
```

```
SELECT c.country_name, a.gdp_affordability, a.data_year
FROM affordability a
JOIN countries c ON a.iso_code = c.iso_code
WHERE a.gdp_affordability < 1
GROUP BY c.country_name, a.gdp_affordability, a.data_year
ORDER BY a.gdp_affordability DESC’
```

<u>View Statement 3:</u>

In this statement we add two columns of death rates, death rate of males and death rates of females to find the total death rate for each country and year and then we project the columns: iso_code, year information the data corresponds to, total death rate per hundred thousand people.

```
‘CREATE VIEW most_lung_cancer_deaths AS
SELECT iso_code, data_year, ( death_rate_male_100k + death_rate_female_100k ) AS
total_death_rate_100k
FROM deaths_lung_cancer
GROUP BY iso_code, data_year
ORDER BY total_death_rate_100k DESC’
```

<u>View Statement 4:</u>

In this statement we aim to find the average death rates from lung cancer for each country and then we project the columns: iso_code, average deaths.

```
‘CREATE VIEW avg_lung_cancer_deaths AS
SELECT iso_code, AVG( total_death_rate_100k ) AS avg_death
FROM most_lung_cancer_deaths
GROUP BY iso_code
ORDER BY avg_death DESC’
```

View Statement 5:

Purpose of the following ‘create view’ statement is to create a view of countries with cigarette consumptions that are higher than the average and then we project the columns: iso_code, average consumption.

```
‘CREATE VIEW high_consumption_country_year_view AS
SELECT iso_code, AVG(consumption_per_smoker_daily) AS avg_con
FROM cig_consumption_per_smoker_daily
GROUP BY iso_code
HAVING AVG(consumption_per_smoker_daily) > (SELECT
AVG(consumption_per_smoker_daily) FROM cig_consumption_per_smoker_daily)’
```

View Statement 6:

In this statement we aim to acquire the minimum cigarette sales in a specific country or countries by selecting the minimum value from the sales_daily column from the cig_sales_daily view and then we project the columns: iso_code, minimum sales.

```
‘CREATE VIEW min_cig_sales_country_year_view AS
SELECT iso_code, MIN(sales_daily) AS min_sales
FROM cig_sales_daily
GROUP BY iso_code
HAVING MIN(sales_daily) = (SELECT MIN(sales_daily) FROM cig_sales_daily)’
```

View Statement 7:

In this statement we aim to acquire the maximum cigarette sales in a specific country or countries by selecting the maximum value from the sales_daily column from the cig_sales_daily view and then we project the columns: iso_code, maximum sales.

```
‘CREATE VIEW max_cig_sales_country_year_view AS
```

```
SELECT iso_code,MAX(sales_daily) AS max_sales
FROM cig_sales_daily
GROUP BY iso_code
HAVING MAX(sales_daily) = (SELECT MAX(sales_daily) FROM cig_sales_daily)’
```

<u>View Statement 8:</u>
Purpose of the following ‘create view’ statement is to calculate the average tobacco deaths for each country and project the columns: iso code, avg_tobacco_deaths and then we project the columns: iso_code, average tobacco deaths.

```
‘CREATE VIEW avg_tobacco_usage_SUMCOUNT AS
SELECT iso_code, SUM(tobacco_deaths) / count(tobacco_deaths) as avg_tobacco_deaths
from deaths_tobacco_usage
group by iso_code’
```

<u>View Statement 9:</u>
In this statement we take the avg deaths of each country from the avg_lung_cancer_deaths then by taking the countries with average deaths greater than the average of avg_deaths column we get the countries with high lung cancer deaths.and then we project the columns: iso_code, average number of deaths.

```
‘CREATE VIEW high_avg_lung_cancer_deaths AS
SELECT iso_code, AVG(avg_death) as avg_death
FROM avg_lung_cancer_deaths
GROUP BY iso_code
HAVING AVG(avg_death) > (SELECT AVG(avg_death) FROM avg_lung_cancer_deaths)
ORDER BY avg_death DESC’
```

<u>View Statement 10:</u>
In the following ‘create view’ statement we are creating a view of the cigarette consumption of countries that have a population higher than 50 million and then we project the columns:

iso_code, year information the data corresponds to, cigarette consumption per smoker on a daily basis.

```
‘CREATE VIEW cc_filtered_except AS
SELECT cc.iso_code, cc.data_year, cc.consumption_per_smoker_daily
FROM cig_consumption_per_smoker_daily cc
WHERE cc.iso_code NOT IN (
  SELECT c.iso_code
  FROM countries c
  WHERE c.population < 50000000
)’
```

<u>View Statement 11:</u>
The given SQL statement creates a view called "cc_outer_join" that selects data from the "cig_consumption_per_smoker_daily" table and performs a left outer join with the "countries" table to filter out countries with a population less than 50 million. This will return a table with three columns: iso_code, data_year, and consumption_per_smoker_daily. The iso_code column contains the country codes, data_year column contains the year of the data, and consumption_per_smoker_daily column contains the average daily consumption per smoker for countries with a population greater than or equal to 50 million.

```
‘CREATE VIEW cc_outer_join AS
SELECT cc.iso_code, cc.data_year, cc.consumption_per_smoker_daily
FROM cig_consumption_per_smoker_daily cc
LEFT OUTER JOIN (
  SELECT iso_code
  FROM countries
  WHERE population < 50000000
) c ON cc.iso_code = c.iso_code
WHERE c.iso_code IS NULL’
```

<u>Check Constraint Statement:</u>

The following statement creates a constraint on the ‘cig_sales_daily’ table such that whenever a new value to the ‘sales_daily’ is inserted, the database management system will have to ensure that the value is in the [0.1, 12] range. Unless it violates the constraint, the value is being inserted into the table.

```
‘ALTER TABLE cig_sales_daily
ADD CONSTRAINT check_sales_daily
CHECK (sales_daily >= 0.1 AND sales_daily <= 12)’
```

After creating the constraint we try inserting a value that is outside of the range to make sure that the constraint is working as expected.

```
‘Insert into cig_sales_daily(iso_code,data_year,sales_daily) values("AUS", 3000, 13)’
```

Trigger Statement:

The following trigger statement aims to create a specialized type of stored procedure that is called every time a database server event occurs. In our case, we are creating a trigger that checks (after the certain database server event) whether ‘sales_daily’ column values are out of the [1, 12] range and if true (the value is out of bounds), the trigger sets the value to 1 if the value is smaller than 1 or 12 if the values is greater than 12.

```
‘CREATE TRIGGER cigs_sales_trigger BEFORE INSERT ON cig_sales_daily
FOR EACH ROW
BEGIN
   IF NEW.sales_daily <= 1 THEN
      SET NEW.sales_daily = 1;
   END IF;

   IF NEW.sales_daily >= 12 THEN
      SET NEW.sales_daily = 12;
   END IF;
END’
```

After creating the Trigger, we again insert the same out of range value and see that this time it inserts the value to the table but changes the value as we intended.

```
‘Insert into cig_sales_daily(iso_code,data_year,sales_daily) values("AUS", 3000, 13)’
```

And we would eventually delete it again from the table to remove unneeded data from our dataset.

```
‘delete from cig_sales_daily where iso_code="AUS" and data_year=3000’
```

General constraints are easy to create and maintain, and they ensure that data is validated and the rules are enforced automatically. However, they may be restrictive and require updates to existing data. Such as in our case we want to add new maximum values which are greater than 12, however it is restricted to insert that data and gives an error message about it.

Trigger methods can enforce more complex rules and restrictions, but they can be complex to create and maintain, and may have a performance impact on the database. They can also perform custom actions, which may be beneficial in certain scenarios.

Lastly, the choice between general constraints and trigger methods and deciding on whichever is more advantageous depends on the specific needs of the database and the data being stored.

<u>Procedure Statement:</u>
The aim of this procedure is to create a function that returns the affordability values of the related countries when it is called.

```
‘CREATE PROCEDURE get_affordability(
  IN p_iso_code VARCHAR(3)
)
BEGIN
  SELECT gdp_affordability
  FROM affordability
  WHERE iso_code = p_iso_code;
END’

‘CALL get_affordability('USA')’‘CALL get_affordability('AUS')’
```