

CS306 Project

Project Repository: [SleepyCoffy/CS-306-BasedData-Project](https://github.com/SleepyCoffy/CS-306-BasedData-Project): CS 306 Database project that will use data from www.ourworldindata.com and MySQL as a main tool. Project Group name: [BasedData](https://github.com/BasedData). (github.com)

Group Name: **BasedData**

Group Members:

Name-Surname	Student ID	E-mail
Emir Aikan	28614	aikan@sabanciuniv.edu
Safa Abdullah Söğütlügil	29214	safasogutlugil@sabanciuniv.edu
Yunus Emre Şencan	29562	sencanyunus@sabanciuniv.edu
Yusuf Erkut Bayram	28976	yusuferkut@sabanciuniv.edu
Yusuf Kılıç	29431	yusufk@sabanciuniv.edu

Agriculture and Food Supply Database

Question 1.a:

```
CREATE VIEW ExportToProductRatio as
SELECT c.country_name AS country, c.year, c.production_t AS production,
c.export_t AS export,
(c.export_t / c.production_t) AS export_to_production_ratio
FROM crops as c
WHERE (c.export_t / c.production_t) >= 0.5 AND (c.export_t /
c.production_t) <= 1;
```

This view returns countries that exported more than 50% of their corn production in a given year. The view has the columns country name, year, production in tonnes, export in tonnes and ratio of these two. The export to production ratio is also upper limited to 1 to filter out any erroneous data that shows more export than production.

```

CREATE VIEW countries_above_avg_production AS
SELECT p.country_name, p.year, c.production_t / p.size AS
production_per_person
FROM population p
JOIN crops c ON p.country_name = c.country_name AND p.year = c.country_year
WHERE c.production_t / p.size > (
    SELECT AVG(production_t / size)
    FROM crops
    JOIN population ON crops.country_name = population.country_name AND
crops.country_year = population.year
);

```

This view shows countries which produced more crops than the average in certain years. It joins the population table and crops table to show a percentage which describes how countries produce crops proportionally to the population they have. Higher percentage describes that the country is more self sufficient.

```

create view PFR as
select P.Year, P.country_name as country, P.size as population, F.weight_t,
ifnull(P.size/F.weight_t,0) as PFR
from population P, food F
where P.country_name = F.country_name and P.Year = F.year and F.weight_t !=
0
and (ifnull(P.size/F.weight_t,0))>50;

```

This view represents the ratio of population over food_weight. The purpose is to find out the efficiency of food production of countries each year. It contains columns 'Year', 'Country_name', 'Population Size' from table 'population' and column 'weight_t' from table 'food'. Last column contains the ratio of population/food_weight_t for a specific country in a unique year. In some countries-years food_weight_t is reported as 0, these rows are discarded with the ifnull() function. This view accepts only ratios that are higher than 50. There is a tendency for the ratio to increase each year for all countries.

```

CREATE VIEW waste_to_production_ratio AS
SELECT s.country, SUM(s.weight_t) AS supply_chain_waste,
SUM(c.production_t) AS production, SUM(s.weight_t)/SUM(c.production_t) AS
waste_to_production_ratio
FROM supply_chain_waste AS s
JOIN crops AS c ON c.country_name = s.country AND c.year = s.year
WHERE s.weight_t IS NOT NULL AND
c.production_t IS NOT NULL AND
s.weight_t/c.production_t IS NOT NULL AND
s.weight_t/c.production_t < 1
GROUP BY s.country;

```

The view `waste_to_production_ratio` shows the ratio of the total supply chain waste (tons) over the total production, considering all of the years in which data is available. It has columns 'country', 'supply_chain_waste', 'production' and 'waste_to_production_ratio'. We discarded the null values in the columns 'weight_t' and 'production_t'. We also discarded the cases where the denominator is zero. To eliminate the false data, we limited the ratio to be lower than 1. Since waste cannot be greater than production, they are eliminated.

```

CREATE VIEW PopulationIncreaseRate AS
SELECT
    p1.country_name,
    p2.year AS year_before,
    p2.size AS population_before,
    p1.year AS year_after,
    p1.size AS population_after,
    CONCAT((CAST(p1.size AS SIGNED) - CAST(p2.size AS SIGNED)) / CAST(p2.size
AS SIGNED) * 100, '%') AS increase_rate
FROM
    population p1
JOIN population p2
    ON p1.country_name = p2.country_name AND p1.year = p2.year + 1;

```

This query joins the population table on itself and calculates the yearly population increase rate. The result shows the country name, the previous year, what the population was, the year after and what the population became in that year. In the last column, the population increase rate for each country and for each year is given.

Question 1.b:

```
create view PFR1 as
select P.Year, P.country_name as country, P.size as population, F.weight_t,
ifnull(P.size/F.weight_t,0) as PFR
from population P, food F
where P.country_name = F.country_name and P.Year = F.year and F.weight_t !=
0
and (ifnull(P.size/F.weight_t,0))>=0;

SELECT Year, Country, Population, weight_t, PFR
FROM PFR1
WHERE (Year, Country) NOT IN (SELECT Year, Country FROM PFR);

SELECT PFR1.Year, PFR1.Country, PFR1.Population, PFR1.weight_t, PFR1.PFR
FROM PFR1
LEFT JOIN PFR ON PFR1.Year = PFR.Year AND PFR1.Country = PFR.Country
WHERE PFR.Year IS NULL AND PFR.Country IS NULL;
```

This code creates an additional view which contains population/food_weight_t ratio but accepts all ratios ranging from 0 to infinity. The goal here is to find out the countries that contain ratios lower than 50 by using the SET operators and EXCEPTION operators. Firstly, by using operator NOT IN we have extracted countries and years with ratios lower than 50 from PFR1 and discarded countries and years which were present in view PFR. In the second method, LEFT JOIN operator was used to do the same operation.

Question 1.c:

```
SELECT scw.country_name, scw.year, scw.weight_t AS SupplyChainWaste_t
FROM supply_chain_waste scw
WHERE EXISTS (SELECT * FROM crops c
              WHERE c.export_t > c.import_t AND scw.country_name =
c.country_name
              AND scw.year = c.year);
```

```
SELECT scw.country_name, scw.year, scw.weight_t AS SupplyChainWaste_t
FROM supply_chain_waste scw
WHERE scw.country_name IN (SELECT c.country_name FROM crops c
                          WHERE c.export_t > c.import_t AND scw.year = c.year);
```

These queries retrieve data from the supply_chain_waste table for countries and years where the export was more than the import and show the amount of supply chain waste in those years. When we use EXISTS, the country name check is moved inside the nested query's WHERE clause. When we use IN, the country name check is written before the IN keyword, but essentially they do the same thing and return the same results in different ways.

Question 1.d:

```
SELECT food.country_name, MAX(food.protein_g_per_capita * population.size /
food.weight_t) as max_protein_grams_per_product_tonne
FROM food
JOIN population ON food.country_name = population.country_name AND
food.year = population.year
GROUP BY food.country_name
HAVING max_protein_grams_per_product_tonne > 0
ORDER BY max_protein_grams_per_product_tonne DESC;
```

This query gets the maximum yearly protein production per ton of product for each country. It orders this production in a descending order. The results of this query shows there are big differences in protein production of each country, which may be caused by different types of corn produced by different countries.

```

SELECT PFR.Country, MIN(PFR.PFR) AS Minimum_Ratio
FROM PFR
INNER JOIN food
ON PFR.Country = food.country_name
GROUP BY food.country_name
HAVING AVG(PFR.PFR) > 0;

```

This query finds the specific country's minimum_ratio for population/food_weight_t columns. One constraint here is that the country should have an average ratio over 0.

```

SELECT pir.country_name, COUNT(*) AS num_years
FROM populationincreaserate pir
JOIN crops c ON pir.country_name = c.country_name AND pir.year_before =
c.year
WHERE pir.increase_rate > 0 AND c.export_t < c.import_t
GROUP BY pir.country_name
HAVING num_years >= 10;

```

This query counts the number of years for each country, where they had a positive population increase rate(population was increasing) and less export than the imports and shows the ones that fit these conditions for at least 10 years.

```

SELECT c.country_name, SUM(c.export_t) AS exports_sum, MAX(p.size) AS
max_population
FROM crops AS c
JOIN population AS p ON p.country_name = c.country_name AND p.year = c.year
GROUP BY c.country_name
HAVING exports_sum > (SELECT AVG(total_exports) FROM (SELECT SUM(export_t)
AS total_exports FROM crops GROUP BY country_name) AS avg_exports)
AND max_population < 100000000;

```

The select statement shows the countries which have a population less than 100 Million, and which have their export_sum greater than the average of the total_exports. To select average value from the sum of exports, we used nested select statements. It has columns 'country_name', 'exports_sum', 'max_population'. We can see countries which exports more than average, but have relatively less population.

```
SELECT c.country_name, AVG(c.production_t) AS avg_production
FROM crops c
JOIN population p ON c.country_name = p.country_name AND c.country_year =
p.year
GROUP BY c.country_name
HAVING avg_production > 3312915;
```

This query shows countries above general crop production. The number at the end is calculated with the help of excel table and it describes the average production. All in all this query gives us the countries which produce the most crops throughout the years.

Question 2:

```
ALTER TABLE supply_chain_waste
ADD CONSTRAINT chk_weight_t_range_of_supply_chain_waste
CHECK (weight_t >= 0 AND weight_t <= 19111000);
```

```
DELIMITER //
CREATE TRIGGER trigger_before_insert_scw
BEFORE INSERT ON supply_chain_waste
FOR EACH ROW
BEGIN
    IF NEW.weight_t < 0 THEN
        SET NEW.weight_t = 0;
    ELSEIF NEW.weight_t > 19111000 THEN
        SET NEW.weight_t = 19111000;
    END IF;
END//
DELIMITER ;
```

```
DELIMITER //
CREATE TRIGGER trigger_before_update_scw
BEFORE UPDATE ON supply_chain_waste
FOR EACH ROW
BEGIN
    IF NEW.weight_t < 0 THEN
        SET NEW.weight_t = 1;
    ELSEIF NEW.weight_t > 19111000 THEN
        SET NEW.weight_t = 19111000;
    END IF;
END//
DELIMITER ;
```

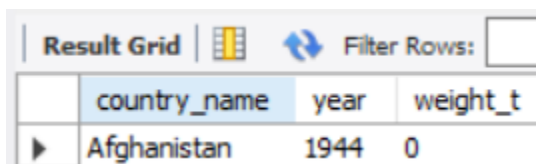
For the table called 'supply_chain_waste' two triggers were created, one for insert operation and another one for update operation. Both triggers activate when the inserted value for column 'weight_t' of table 'supply_chain_waste' is out of range, range here spans from 0 to 19111000. If the value was inserted as lower than 0, then each trigger sets the default value of 1, and if inserted or updated value was entered as higher than 19111000, then both triggers set it to 19111000.

```
insert into supply_chain_waste(country_name, year, weight_t) values
('Afghanistan', 1944, -12);
```

This insert statement gives an error when there is only the constraint in place since weight value is out of limits. It inserts a row successfully after adding the triggers.

```
select *
from supply_chain_waste as s
where s.country_name = "Afghanistan" and s.year = 1944
```

This query returns one row after the insert command after adding triggers:



	country_name	year	weight_t
▶	Afghanistan	1944	0

The result of the select query shows that the trigger worked correctly to fix the weight value.

Question 3:

```
delimiter //
CREATE PROCEDURE `get_production`(cname varchar(50))
BEGIN
    SELECT *
    FROM crops AS C
    WHERE C.country_name = cname;
END//
delimiter ;
```

This procedure returns production data for the given country name.

General Constraints vs. Triggers:

Constraints block data that is outside the limits to be inserted to the table. This can both be an advantage and a disadvantage. It is an advantage because there is no erroneous data that can break calculations. It is a disadvantage because a row that is blocked by a general constraint due to an out of bounds column may have other valuable columns that are necessary to have.

Triggers, on the other hand, ensure that even if some value of a data row is outside the limits, it can still be inserted to the table, which is an advantage. However, some columns may not have default values so it is hard to fix data errors and insert them into the table.