RISK FACTOR OF DEATHS
Project Step 3

Nur Ayca İlhan 26701, Kaan Akçay 27928, Elif Özvarış 26752, Kutay Yüceak 28337, Muammer Tunahan Yıldız 27968

Github Repo: https://github.com/kaanakcay/CS306-Project
SQL FILENAME: DATABASE_STEP3.sql
Log files are in the CS306-STEP3 file.

CREATE VIEW statements

We stated 5 CREATE VIEW sentences, which are *env_factor_view*, *child_mortality_diet_view*, *addiction_air_pol_view*, *diet_env_view*, and *addiction_env_child_mortality_view*.

- env_factor_view: This view displays environmental factors related to water and sanitation for each country. The view combines data from two tables, *countries* and *env_factor*, using a join operation based on the common *iso_code* column. The resulting view includes columns for country code, country name, unsafe water, unsafe sanitation, and hand washing data. By creating this view, users can easily analyze the environmental data for different countries whenever they need information about finding corresponding countries of environmental factors.

  ```
  CREATE VIEW env_factor_view AS
  SELECT c.iso_code, c.countries_name, ef.unsafe_water, ef.unsafe_sanitation, ef.hand_washing
  FROM countries c
  JOIN env_factor ef ON c.iso_code = ef.iso_code;
  ```

- child_mortality_diet_view: Displays information about child mortality rates and dietary factors for each country. The purpose of creating this view is to access information about the relationship between child health and nutrition across different countries. The view combines data from three tables, *countries*, *child_mortality*, and *diet*, using JOIN operations based on the common *iso_code* column.

  ```
  CREATE VIEW child_mortality_diet_view AS
  SELECT cm.iso_code, c.countries_name, cm.child_wasting, cm.non_bfeeding,
  cm.low_birth_weight, cm.d_bfeeding, d.high_sodium, d.low_fruits, d.low_nuts_seeds,
  d.low_whole_grain, d.low_vegetables
  FROM countries c
  JOIN child_mortality cm ON c.iso_code = cm.iso_code
  JOIN diet d ON c.iso_code = d.iso_code;
  ```

- addiction_air_pol_view: Displays information about addiction and air pollution factors for each country. The view combines data from three tables, *countries*, *addiction*, and *air_pol*, using JOIN operations based on the common *iso_code* column.

```
CREATE VIEW addiction_air_pol_view AS
SELECT a.iso_code, c.countries_name, a.smoke, a.drug_use, a.alcohol, ap.indoor, ap.outdoor
FROM countries c
JOIN addiction a ON c.iso_code = a.iso_code
JOIN air_pol ap ON c.iso_code = ap.iso_code;
```

- diet_env_view: Displays information about dietary factors and environmental factors for each country. The view combines data from three tables, *countries*, *diet*, and *env_factor*, using JOIN operations based on the common *iso_code* column.

```
CREATE VIEW diet_env_view AS
SELECT d.iso_code, c.countries_name, d.high_sodium, d.low_fruits, d.low_nuts_seeds,
d.low_whole_grain, d.low_vegetables, ef.unsafe_water, ef.unsafe_sanitation, ef.hand_washing,
ef.air_pollution
FROM countries c
JOIN diet d ON c.iso_code = d.iso_code
JOIN env_factor ef ON c.iso_code = ef.iso_code;
```

- Addiction_env_child_mortality_view: Displays information about addiction, environmental factors, and child mortality rates for each country. The view combines data from four tables, *countries*, *addiction*, *env_factor*, and *child_mortality*, using JOIN operations based on the common *iso_code* column.

```
CREATE VIEW addiction_env_child_mortality_view AS
SELECT a.iso_code, c.countries_name, a.smoke, a.drug_use, a.alcohol, ef.unsafe_water,
ef.unsafe_sanitation, ef.hand_washing, ef.air_pollution, cm.child_wasting, cm.non_bfeeding,
cm.low_birth_weight, cm.d_bfeeding
FROM countries c
JOIN addiction a ON c.iso_code = a.iso_code
JOIN env_factor ef ON c.iso_code = ef.iso_code
JOIN child_mortality cm ON c.iso_code = cm.iso_code;
```

Joins and Set Operators

- The "INTERSECT" keyword is used in this SQL query to merge the result sets from two different SELECT statements. The "countries_name" column from the "env_factor_view" table is chosen in the first SELECT query if the "unsafe_water" value is larger than 50,000. The "countries_name" column from the "air_pol_view" table is chosen from the second SELECT statement if either the "indoor" value is less than 3,000 OR the "outdoor" value is larger than 12,000 in the table. Only the countries that meet both criteria will be returned when the "INTERSECT" keyword is applied between these two SELECT statements.

```
SELECT countries_name

FROM env_factor_view

WHERE unsafe_water > 50000

INTERSECT

SELECT countries_name

FROM air_pol_view

WHERE indoor < 3000 OR outdoor > 12000;
```

- This SQL query combines data from various tables using JOIN and filters the results using WHERE according to specific criteria. It chooses the names of the countries where the "high_sodium" value in the "diet" table equals 1, and then it deducts the countries where the "child_wasting" value in the "child_mortality" table equals 1. The names of the countries that meet the first requirement but not the second are the final outcome.

```
SELECT c.countries_name

FROM countries c

JOIN diet d ON c.iso_code = d.iso_code

JOIN child_mortality cm ON c.iso_code = cm.iso_code

WHERE d.high_sodium = 1

EXCEPT

SELECT c.countries_name

FROM countries c

JOIN child_mortality cm ON c.iso_code = cm.iso_code

WHERE cm.child_wasting = 1;
```

- By combining data from various tables using an LEFT JOIN, this SQL query will return all entries from the left table, only matching records from the right tables, and NULL values for records that do not match. If the "high_sodium" value is found in the "diet" table but not in the "child_mortality" table, the query restricts the results to only those countries.

```
SELECT c.countries_name

FROM countries c
```

```
LEFT JOIN diet d ON c.iso_code = d.iso_code AND d.high_sodium = 1

LEFT JOIN child_mortality cm ON c.iso_code = cm.iso_code AND cm.child_wasting = 1

WHERE d.high_sodium IS NOT NULL AND cm.child_wasting IS NULL;
```

- In response to a subquery that picks the "iso_code" values from the "diet" dataset where the "high_sodium" value is more than 50,000, this SQL query retrieves all columns from the "child_mortality" table where the "iso_code" values are present. In other words, it returns information on child mortality for those countries where more than 50,000 people consume a lot of sodium.

```
SELECT *

FROM child_mortality cm

WHERE cm.iso_code IN (

        SELECT iso_code

        FROM diet

        WHERE high_sodium > 50000

);
```

- In response to a subquery that picks the "iso_code" values from the "diet" dataset when the "high_sodium" value is equal to 1, this SQL query selects all columns from the "child_mortality" table where the "iso_code" values are present. So, it provides information on child mortality for those countries where the high sodium intake is equal to 1.

```
SELECT *

FROM child_mortality cm

WHERE cm.iso_code IN (

        SELECT iso_code

        FROM diet

        WHERE high_sodium = 1

);
```

"In" and "Exists"

The IN operator returns true if any of the values in a subquery match any of the values in a list. In the first query, all countries that have an iso_code that matches any iso_code in a subquery that selects all iso_codes from child_mortality_diet_view where high_sodium = 1 are selected.

The EXISTS operator returns true if a subquery returns at least one row. In the second query, we select countries where there exists at least one row in child_mortality_diet_view where iso_code = c.iso_code and high_sodium = 1, basically the same selection as the first query.

IN:

```
SELECT c.iso_code, c.countries_name

FROM countries c

WHERE c.iso_code IN (

        SELECT cmdv.iso_code

        FROM child_mortality_diet_view cmdv

        WHERE cmdv.high_sodium = 1

);
```

EXISTS:

```
SELECT c.iso_code, c.countries_name

FROM countries c

WHERE EXISTS (

        SELECT 1

        FROM child_mortality_diet_view cmdv

        WHERE cmdv.iso_code = c.iso_code AND cmdv.high_sodium = 1

);
```

Aggregate Operators

A.O -> GROUP BY

```
SELECT cmdv.countries_name, COUNT(*) AS countries_with_high_sodium_and_unsafe_water

FROM child_mortality_diet_view cmdv

JOIN env_factor_view efv ON cmdv.iso_code = efv.iso_code

WHERE cmdv.high_sodium = 1 AND efv.unsafe_water > 50

GROUP BY cmdv.countries_name

HAVING countries_with_high_sodium_and_unsafe_water > 0;
```

This SQL query joins and filters data from two views, groups the results by country name, and filters the results to only include countries with at least one of these occurrences. It then counts the number of countries, which have both high sodium intake and unsafe water conditions.

–

A.O -> GROUP BY

```
SELECT aapv.countries_name, AVG(aapv.smoke) AS avg_smoke, AVG(cmdv.child_wasting) AS
avg_child_wasting

FROM addiction_air_pol_view aapv

JOIN child_mortality_diet_view cmdv ON aapv.iso_code = cmdv.iso_code

GROUP BY aapv.countries_name

HAVING avg_smoke > 20 AND avg_child_wasting > 5;
```

This query calculates the average of smoking and child wasting for each country by joining two views. Then it filters the outcomes to only include countries where the average smoking is more than 20 and the average child wasting is more than 5.

—

A.O -> GROUP BY

```
SELECT apv.countries_name, SUM(apv.indoor) AS sum_indoor, SUM(aapv.alcohol) AS sum_alcohol

FROM air_pol_view apv

JOIN addiction_air_pol_view aapv ON apv.iso_code = aapv.iso_code
```

```
GROUP BY apv.countries_name

HAVING sum_indoor > 100 AND sum_alcohol > 50;
```

This query calculates the sum of indoor air pollution and also alcohol consumption for each country by joining two views. Then it filters the outcomes to only include countries where the sum of indoor air pollution is more than 100 and the sum of alcohol consumption is more than 50.

–

A.O -> GROUP BY

```
SELECT efv.countries_name, MIN(efv.unsafe_sanitation) AS min_unsafe_sanitation, MIN(apv.outdoor) AS min_outdoor_pollution

FROM env_factor_view efv

JOIN air_pol_view apv ON efv.iso_code = apv.iso_code

GROUP BY efv.countries_name

HAVING min_unsafe_sanitation < 30 AND min_outdoor_pollution < 40;
```

MIN(): Returns the minimum value from a set of values. In this query, it is used to find the minimum value of deaths caused by the risk factors of unsafe sanitation and outdoor pollution for each country.

–

A.O -> GROUP BY

```
SELECT dev.countries_name, MAX(dev.low_vegetables) AS max_low_vegetables, MAX(cmdv.low_birth_weight) AS max_low_birth_weight

FROM diet_env_view d

JOIN child_mortality_diet_view cmdv ON dev.iso_code = cmdv.iso_code

GROUP BY dev.countries_name

HAVING max_low_vegetables > 0 AND max_low_birth_weight > 5;
```

MAX(): Returns the maximum value from a set of values. In this query, it is used to find the maximum value of deaths caused by the risk factors of a diet low in vegetables and low birth weight for each country.

STORED PROCEDURE statement

We created a STORED PROCEDURE called *country_health_data*. This stored procedure can be used to retrieve predetermined risk factors data for a specific country using *iso_code*.

If the *iso_code* passed is 'JPN', the stored procedure returns data related to addiction (smoking, drug use, and alcohol consumption) from the *addiction* table.

If the *iso_code* passed is 'ITA', the stored procedure returns data related to diet (high sodium, low fruits, low nuts and seeds, low whole grain, and low vegetables) from the *diet* table.

If the *iso_code* passed is not 'JPN' or 'ITA', the stored procedure returns data related to air pollution (indoor and outdoor) from the *air_pol* table.

```
CREATE PROCEDURE country_health_data (IN p_iso_code VARCHAR(5))
BEGIN
  IF p_iso_code = 'JPN' THEN
    SELECT a.iso_code, a.smoke, a.drug_use, a.alcohol
    FROM addiction a
    WHERE a.iso_code = p_iso_code;
  ELSEIF p_iso_code = 'ITA' THEN
    SELECT d.iso_code, d.high_sodium, d.low_fruits, d.low_nuts_seeds, d.low_whole_grain,
d.low_vegetables
    FROM Diet d
    WHERE d.iso_code = p_iso_code;
  ELSE
    SELECT ap.iso_code, ap.indoor, ap.outdoor
    FROM air_pol ap
    WHERE ap.iso_code = p_iso_code;
  END IF;
END
```

CONSTRAINT statements

We added 3 CONSTRAINT statements to the *addiction* table.

```
ALTER TABLE addiction
```

- ck_smoke_range: Checks that the value in the *smoke* column is greater than or equal to 0 and less than or equal to 1,700,000. This constraint ensures that the *smoke* column only contains valid values.
  ```
  ADD CONSTRAINT ck_smoke_range CHECK (smoke >= 0 AND smoke <= 1700000),
  ```

- ck_drug_use_range: Checks that the value in the *drug_use* column is greater than or equal to 0 and less than or equal to 100,000. This constraint ensures that the *drug_use* column only contains valid values.
  ADD CONSTRAINT ck_drug_use_range CHECK (drug_use >= 0 AND drug_use <= 100000),

- ck_alcohol_range: Same as above. Checks that the value in the *alcohol* column is greater than or equal to 0 and less than or equal to 400,000.
  ADD CONSTRAINT ck_alcohol_range CHECK (alcohol >= 0 AND alcohol <= 400000);

TRIGGER statements

2 TRIGGER statements, one to insert to addiction and one for update(?):

- Addiction_before_insert: Executed before a new row is inserted into the *addiction* table. It checks the values of the *smoke*, *drug_use*, and *alcohol* columns in the new row, and if any of these values are less than 0, it sets them to 0. Similarly, if any of these values are greater than 100, it sets them to 100. This trigger ensures that the values in these columns remain within a valid range.

```
CREATE TRIGGER addiction_before_insert
BEFORE INSERT ON addiction
FOR EACH ROW
BEGIN
 IF NEW.smoke < 0 THEN
   SET NEW.smoke = 0;
 ELSEIF NEW.smoke > 100 THEN
   SET NEW.smoke = 100;
 END IF;

 IF NEW.drug_use < 0 THEN
   SET NEW.drug_use = 0;
 ELSEIF NEW.drug_use > 100 THEN
   SET NEW.drug_use = 100;
 END IF;

 IF NEW.alcohol < 0 THEN
   SET NEW.alcohol = 0;
 ELSEIF NEW.alcohol > 100 THEN
   SET NEW.alcohol = 100;
 END IF;
END
```

- Addiction_before_update: Executed before an existing row in the *addiction* table is updated. It performs the same checks and on the same columns as *addiction_before_insert* in the updated

row. This trigger ensures that the values in these columns remain within a valid range even after updates are made to the table.

```
CREATE TRIGGER addiction_before_update
BEFORE UPDATE ON addiction
FOR EACH ROW
BEGIN
  IF NEW.smoke < 0 THEN
    SET NEW.smoke = 0;
  ELSEIF NEW.smoke > 100 THEN
    SET NEW.smoke = 100;
  END IF;

  IF NEW.drug_use < 0 THEN
    SET NEW.drug_use = 0;
  ELSEIF NEW.drug_use > 100 THEN
    SET NEW.drug_use = 100;
  END IF;

  IF NEW.alcohol < 0 THEN
    SET NEW.alcohol = 0;
  ELSEIF NEW.alcohol > 100 THEN
    SET NEW.alcohol = 100;
  END IF;
END
```

Trigger and General Constraints Comparisons

At the column or table level, general constraints are simpler to develop, administer, and enforce, but they are unable to handle complex data validation requirements. On the other side, triggers provide additional flexibility and can handle complex data modification, row-level enforcement, and validation across several tables.