Causes of Death Tracker Database Application

GROUP 24

Berk Karaca 24954

Ege Karamelek 28020

Emre Gökay Kılınç 28086

Ali Taylan Kıran 29040

#1a)

```
CREATE VIEW low_disease_death AS

SELECT d_name, d_code, d_year,
d_meningitis,d_alzheimers_disease,d_parkinsons_disease,d_malaria,d_maternal_disorders,d_hiv_aids,d_tuberculosis,d_cardiovascular_diseases,d_diabetes_mellitus,d_id

FROM causesofdeath.diseases

WHERE d_meningitis<1500 AND d_alzheimers_disease<5000 AND d_parkinsons_disease<1000 AND d_malaria<0.2 AND d_maternal_disorders<2000 AND d_hiv_aids<1000 AND d_tuberculosis<200000 AND d_cardiovascular_diseases<100000 AND d_diabetes_mellitus<5000;


CREATE VIEW high_nature_death AS

SELECT f_name, f_code, f_year, f_death, f_id

FROM causesofdeath.exposuretoforcesofnature

WHERE f_death>1;


CREATE VIEW high_terrorism_activity AS

SELECT t_name, t_code, t_year, t_death, t_id

FROM causesofdeath.terrorism

WHERE t_death>30;


CREATE VIEW high_selfharm_activity AS

SELECT s_name, s_code, s_year, s_death, s_id

FROM causesofdeath.selfharm

WHERE t_death>10;
```

```
USE causesOfDeath;

CREATE VIEW low_democracy AS

SELECT dem_name, dem_code, dem_year, dem_mean, dem_id

FROM causesofdeath.democracy

WHERE dem_mean<0.5;


USE causesOfDeath;

CREATE VIEW high_gdp AS

SELECT g_name, g_code, g_year, g_mean, g_id

FROM causesofdeath.gdp

WHERE g_mean>18381;
```

We have created “VIEW” for six different tables to analyze whether the death values of the countries were above or below the values we determined. After creating the views, we’ve found some correlations between the democracy index and gdp index. We found out that the countries with higher GDP have more self harms. Also, the countries with lower democracy rates have higher terrorism activities. Moreover, the countries with lower GDP have more “exposure to nature” deaths. Furthermore, the countries with higher GDP have less deaths due to diseases.

```
#1b)

SELECT * FROM low_democracy AS ld

UNION

SELECT * FROM high_terrorism_activity AS hta;

SELECT ld.dem_name AS name, ld.dem_code AS code, ld.dem_year AS year, ld.dem_mean AS
mean,ld.dem_low AS low, ld.dem_high AS high, ld.dem_id AS id

FROM low_democracy ld

LEFT JOIN

high_terrorism_activity hta ON ld.dem_code = hta.t_code AND ld.dem_year = hta.t_year

WHERE

hta.t_code IS NULL

UNION

SELECT hta.t_name AS name, hta.t_code AS code, hta.t_year AS year, hta.t_death AS death, hta.t_id
AS id

FROM low_democracy ld
```

```
RIGHT JOIN high_terrorism_activity hta ON ld.dem_code = hta.t_code AND ld.dem_year = hta.t_year

WHERE ld.dem_code IS NULL;
```

We have used set operators between low_democracy and high_terrorism_activity.

This SQL query contains three parts.

The first part is simply selecting all the records from the low_democracy view and high_terrorism_activity view using the UNION operator. The low_democracy view contains records of countries with a democracy score lower than 0.5, and the high_terrorism_activity view contains records of countries with more than 10 terrorism-related deaths. The result of this query will be a combined list of records from both views.

The second part of the query retrieves data from the low_democracy view, filtering out those that also appear in the high_terrorism_activity view. This is achieved by using a LEFT JOIN and a WHERE clause with the condition hta.t_code IS NULL. The output of this part of the query will be a list of countries with low democracy scores that do not have high terrorism-related deaths. The result will contain columns for the country's name (name), code (code), year (year), democracy score mean (mean), democracy score low (low), democracy score high (high), and an identifier (id).

The third part of the query retrieves data from the high_terrorism_activity view, filtering out those that also appear in the low_democracy view. This is achieved by using a RIGHT JOIN and a WHERE clause with the condition ld.dem_code IS NULL. The output of this part of the query will be a list of countries with high terrorism-related deaths that do not have low democracy scores. The result will contain columns for the country's name (name), code (code), year (year), the number of terrorism-related deaths (death), and an identifier (id).

The final output of the entire query will be a combined list of countries from both the second and third parts. This means it will display records for countries that either have low democracy scores but not high terrorism-related deaths, or high terrorism-related deaths but not low democracy scores.

```
#1c)

SELECT * FROM causesofdeath.countries c

WHERE c.c_year IN (

   SELECT hta.t_year

   FROM high_terrorism_activity hta

   WHERE c.c_code = hta.t_code

);
```

```
SELECT * FROM causesofdeath.countries c
WHERE EXISTS (
   SELECT hta.t_year
   FROM high_terrorism_activity hta
   WHERE c.c_code = hta.t_code AND c.c_year = hta.t_year
);

SELECT COUNT(*)
FROM (SELECT * FROM causesofdeath.countries c
      WHERE c.c_year IN (
      SELECT hta.t_year
      FROM high_terrorism_activity hta
      WHERE c.c_code = hta.t_code
   )
) AS in_result;

SELECT COUNT(*)
FROM (
   SELECT *
   FROM causesofdeath.countries c
   WHERE EXISTS (
      SELECT 1
      FROM high_terrorism_activity hta
      WHERE c.c_code = hta.t_code AND c.c_year = hta.t_year
   )
) AS exists_result;
```

These SQL queries aim to retrieve records from the causesofdeath.countries table that have a matching record in the high_terrorism_activity view, which contains records of countries with more than 10 terrorism-related deaths. They demonstrate two different ways of achieving this goal: using the IN operator and the EXISTS operator.

The first query uses the IN operator to find records in the causesofdeath.countries table with a c_year value that matches the t_year value from the high_terrorism_activity view for the same c_code (country code). The output of this query will be a list of records from the causesofdeath.countries table that have high terrorism-related deaths in a specific year.

The second query uses the EXISTS operator to achieve the same result. It selects records from the causesofdeath.countries table where a corresponding record exists in the high_terrorism_activity view for the same c_code (country code) and c_year. The output of this query will be the same as the output of the first query.

The third query counts the number of records returned by the first query (using the IN operator). It wraps the first query in a subquery and applies the COUNT(*) aggregate function to calculate the total number of records returned.

The fourth query counts the number of records returned by the second query (using the EXISTS operator). It wraps the second query in a subquery and applies the COUNT(*) aggregate function to calculate the total number of records returned.

Both the third and fourth queries should return the same count of tuples, demonstrating that both the IN and EXISTS operators can be used interchangeably to achieve the same result in this specific case.

#1d)

```
#1- Sum and count
SELECT c.c_year,
   COUNT(*) as country_count,
   SUM(d.d_malaria) as total_malaria_deaths
FROM causesofdeath.countries c
JOIN causesofdeath.diseases d ON c.c_code = d.d_code AND c.c_year = d.d_year
GROUP BY c.c_year
HAVING total_malaria_deaths > 1000;

#2- Avg and min
        SELECT c.c_year,
           AVG(d.d_malaria) as average_malaria_deaths,
           MIN(d.d_malaria) as min_malaria_deaths
        FROM causesofdeath.countries c
        JOIN causesofdeath.diseases d ON c.c_code = d.d_code AND c.c_year = d.d_year
```

```
    GROUP BY c.c_year
    HAVING average_malaria_deaths > 10;
```

#3- Max and count:

```
SELECT c.c_year,
   MAX(d.d_malaria) as max_malaria_deaths,
   COUNT(*) as country_count
FROM causesofdeath.countries c
JOIN causesofdeath.diseases d ON c.c_code = d.d_code AND c.c_year = d.d_year
GROUP BY c.c_year
HAVING max_malaria_deaths > 100;
```

#4- Sum,avg,count,min,max:

```
    SELECT c.c_year,
       SUM(d.d_malaria) as total_malaria_deaths,
       AVG(d.d_malaria) as average_malaria_deaths,
       COUNT(*) as country_count,
       MIN(d.d_malaria) as min_malaria_deaths,
       MAX(d.d_malaria) as max_malaria_deaths
    FROM causesofdeath.countries c
    JOIN causesofdeath.diseases d ON c.c_code = d.d_code AND c.c_year = d.d_year
    GROUP BY c.c_year
    HAVING total_malaria_deaths > 1000 AND average_malaria_deaths > 10;
```

These SQL queries demonstrate the use of aggregate functions (SUM, AVG, COUNT, MIN, MAX) by joining the causesofdeath.countries and causesofdeath.diseases tables. Each query groups the results by the c_year column and uses various aggregate functions and HAVING clauses to filter the output.

The first query calculates the total number of countries (country_count) and the total number of malaria deaths (total_malaria_deaths) per year. The HAVING clause filters the results to only include years with more than 1000 total malaria deaths.

The second query calculates the average number of malaria deaths (average_malaria_deaths) and the minimum number of malaria deaths (min_malaria_deaths) per year. The HAVING clause filters the results to only include years with an average of more than 10 malaria deaths.

The third query calculates the maximum number of malaria deaths (max_malaria_deaths) and the total number of countries (country_count) per year. The HAVING clause filters the results to only include years with more than 100 maximum malaria deaths.

The fourth query combines all the aggregate functions used in the previous queries. It calculates the total, average, minimum, and maximum number of malaria deaths, as well as the total number of countries, per year. The HAVING clause filters the results to only include years with more than 1000 total malaria deaths and an average of more than 10 malaria deaths.

When you run these queries, the output will show various statistics about malaria deaths per year based on the conditions specified in the HAVING clauses.

#2)

```
   SELECT MIN(d_malaria) AS min_malaria, MAX(d_malaria) AS max_malaria
FROM causesofdeath.diseases;


ALTER TABLE causesofdeath.diseases
ADD CONSTRAINT chk_malaria_range CHECK (d_malaria >= 0 AND d_malaria <= 1000000);


INSERT INTO causesofdeath.diseases (d_name, d_code, d_year,d_meningitis, d_alzheimers_disease, d_parkinsons_disease,d_malaria, d_maternal_disorders, d_hiv_aids, d_tuberculosis, d_cardiovascular_diseases, d_diabetes_mellitus, d_id)
VALUES ('Test Country', 'TC', 2022, 1500, 0, 0, 0, 0, 0, 0, 0, 0, 6893);


DELIMITER //
CREATE TRIGGER tr_fix_malaria_value_before_insert
BEFORE INSERT ON causesofdeath.diseases
FOR EACH ROW
BEGIN
   IF NEW.d_malaria < 0 THEN
      SET NEW.d_malaria = 0;
   ELSEIF NEW.d_malaria > 1000 THEN
      SET NEW.d_malaria = 1000;
   END IF;
```

```
END;
//
DELIMITER ;

DELIMITER //
CREATE TRIGGER tr_fix_malaria_value_before_update
BEFORE UPDATE ON causesofdeath.diseases
FOR EACH ROW
BEGIN
   IF NEW.d_malaria < 0 THEN
      SET NEW.d_malaria = 0;
   ELSEIF NEW.d_malaria > 1000 THEN
      SET NEW.d_malaria = 1000;
   END IF;
END;
//
DELIMITER ;

INSERT INTO causesofdeath.diseases (d_name, d_code, d_year,d_meningitis, d_alzheimers_disease,
d_parkinsons_disease,d_malaria, d_maternal_disorders, d_hiv_aids, d_tuberculosis,
d_cardiovascular_diseases, d_diabetes_mellitus, d_id)
VALUES ('Test Country3', 'TCT', 2022, 0, 0, 0, 1500, 0, 0, 0, 0, 0, 6895);

UPDATE causesofdeath.diseases
SET d_malaria = 1500
WHERE d_year = 1990 AND d_code = 'AFG';
```

These SQL queries demonstrate the use of constraints and triggers to control the values that can be inserted or updated in the causesofdeath.diseases table, specifically for the d_malaria column.

The first query selects the minimum and maximum values of the d_malaria column. This information is useful when defining a constraint for the column.

The second query adds a constraint named chk_malaria_range to the d_malaria column, restricting its values between 0 and 1,000,000.

The third query attempts to insert a new row into the causesofdeath.diseases table with a d_malaria value of 0. This value is within the allowed range, so the insertion should succeed.

The fourth query creates a "before insert" trigger named tr_fix_malaria_value_before_insert. This trigger checks if the d_malaria value of a new row is outside the range of 0 to 1,000. If the value is less than 0, it sets the value to 0; if it's greater than 1,000, it sets the value to 1,000.

The fifth query creates a "before update" trigger named tr_fix_malaria_value_before_update. This trigger works similarly to the previous trigger but checks the d_malaria value when updating an existing row instead of inserting a new row.

The sixth query attempts to insert a new row with a d_malaria value of 1,500, which is outside the allowed range. The "before insert" trigger should adjust the value to 1,000 before inserting the row.

The seventh query attempts to update an existing row with a d_malaria value of 1,500, which is outside the allowed range. The "before update" trigger should adjust the value to 1,000 before updating the row.

When you run these queries, the output will show the results of inserting and updating rows with different d_malaria values. The triggers will ensure that the values stay within the defined range (0 to 1,000) even if the constraint allows a broader range (0 to 1,000,000).

#3)

```
USE causesOfDeath;

DELIMITER //

CREATE PROCEDURE get_disease_data(IN d_code VARCHAR(50))

BEGIN

  SELECT

    d.d_year,

    d.d_meningitis,

    d.d_malaria

  FROM causesofdeath.diseases d

  JOIN causesofdeath.countries c ON c.c_code = d.d_code AND c.c_year = d.d_year

  WHERE c.c_code = d_code;

END;

//

DELIMITER ;


CALL get_disease_data('USA');
```

```
CALL get_disease_data('GBR');
```

These SQL queries demonstrate the creation and usage of a stored procedure named get_disease_data with one input parameter d_code. The stored procedure retrieves disease data from the causesOfDeath database, specifically the causesofdeath.diseases and causesofdeath.countries tables.

The USE causesOfDeath; statement selects the causesOfDeath database to work with.

The DELIMITER // command changes the delimiter from the default semicolon (;) to a double forward slash (//). This is necessary because the stored procedure contains semicolons within its body, and we don't want the SQL client to interpret those as the end of the entire statement.

The CREATE PROCEDURE statement defines a new stored procedure named get_disease_data with one input parameter d_code of type VARCHAR(50). The stored procedure body contains a SELECT query that retrieves the year, meningitis, and malaria death counts for the specified country code. The query joins the causesofdeath.diseases and causesofdeath.countries tables on their respective code and year columns, and filters the result by the given country code.

The DELIMITER ; command resets the delimiter back to the default semicolon (;), so subsequent queries can use the standard delimiter.

The CALL get_disease_data('USA'); statement calls the get_disease_data stored procedure with the 'USA' parameter, which will retrieve disease data for the United States.

The CALL get_disease_data('GBR'); statement calls the get_disease_data stored procedure with the 'GBR' parameter, which will retrieve disease data for the United Kingdom.

When you run these queries, the output will display the disease data (year, meningitis, and malaria death counts) for the specified country codes (USA and GBR) from the causesofdeath.diseases and causesofdeath.countries tables.