CS306 PROJECT STEP3
GROUP ARTISAN

Demir Boğa
Adil Yıldız
Baturalp Kabadayı
Mustafa Kulak
Eren Özdil
Cemal Efe Yılmaz

Views:

1) High_population_countries:
   In this view, we wanted to present the countries which have populations over 20 million.

```
CREATE VIEW high_population_countries AS
SELECT *
FROM country
WHERE population > 20000000;
```

2) low_life_expectancy_countries:
   In this view, we wanted to present the countries which have life expectancy lower than 65.

```
CREATE VIEW low_life_expectancy_countries AS
SELECT country_code, age
FROM life_expectancy
WHERE age < 65;
```

3) high_kid_HIV_death_rate_countries:
   In this view, we wanted to present the countries which have a kid death rate caused by HIV over 10%.

```
CREATE VIEW high_kid_HIV_death_rate_countries AS
SELECT death_rate_age_5_14, country_code
FROM HIV_deaths
WHERE death_rate_age_5_14 > 10;
```

4) menengitis_deaths:
In this view, we wanted to show the total amount of deaths caused by meningitis.

```
CREATE VIEW menengitis_deaths AS
SELECT country_code, disease_type, death_num
FROM deaths
WHERE disease_type = 'menengitis';
```

5) high_ART_cure:
In this view, we wanted to show countries which cured more than 300 000 HIV patients with ART treatment.

```
CREATE VIEW high_ART_cure AS
SELECT *
FROM cures_by_ART
WHERE num_deaths_averted > 300000;
```

6) developed_asia_countries:
In this view, we wanted to show the Asian countries which have a development index more than 0.9.

```
CREATE VIEW developed_asia_countries AS
SELECT *
FROM country
WHERE continent = 'Asia' AND development_index > 0.9;
```

Joins and Set Operations:

1) Select developed Asia countries whose population > 20.000.000 with using INTERSECT between high_population_countries, developed_asia_countries.

```
SELECT * from high_population_countries
INTERSECT
SELECT * from developed_asia_countries;
```

2) Select low life expectancy country_code's whose Hiv death rates among childs are > 10 with using INTERSECT between low_life_expectancy_countries, high_kid_HIV_death_rate_countries.

```
SELECT country_code from low_life_expectancy_countries
INTERSECT
SELECT country_code from high_kid_HIV_death_rate_countries;
```

3) Select countries with High amount of art cures but low populations using EXCEPT between high_ART_cure, high_population_countries.

```
SELECT country_code from high_ART_cure
EXCEPT
SELECT country_code from high_population_countries;
```

4) Select countries with High amount of art cures but low populations using FULL OUTER JOIN.

```
SELECT country_code
FROM country
FULL OUTER JOIN cures_by_ART
ON country.country_code = cures_by_ART.country_code
WHERE country.population > 20000000 AND cures_by_ART.num_deaths_averted > 300000;
```

IN and EXIST Operations:

1) Select high population countries who are also in low life expectancy countries using “IN”

```
SELECT * FROM high_population_countries
WHERE country_code IN (SELECT country_code FROM low_life_expectancy_countries);
```

2) Select high population countries who are also in low life expectancy countries using “EXIST”

```
SELECT * FROM high_population_countries
WHERE EXISTS (
    SELECT
    country_code FROM low_life_expectancy_countries
    WHERE high_population_countries.country_code = low_life_expectancy_countries.country_code
);
```

AGGREGATE OPERATORS:

1) Calculate the average life expectancy for each continent using INNER JOIN.

```
SELECT country.continent, AVG(life_expectancy.age) AS average_life_expectancy
FROM life_expectancy
INNER JOIN country ON life_expectancy.country_code = country.country_code
GROUP BY country.continent;
```

2) Calculate total deaths for each disease using SUM.

```
SELECT disease_type,
       SUM(deaths.death_num) AS death_count
FROM deaths
GROUP BY disease_type
HAVING COUNT(*) >= 2;
```

3) Find the disease which kills the most for each country using MAX.

```
SELECT country_code,
       SUM(deaths.death_num) AS death_count
FROM deaths
GROUP BY country_code
HAVING COUNT(*) >= 2;
```

4) Find the countries that have the minimum ART cure rate for each continent using MIN.

```
SELECT country_code, disease_type, MAX(death_num) AS max_death_num
FROM deaths
GROUP BY country_code
HAVING disease_type = (SELECT disease_type FROM deaths WHERE death_num = max_death_num)
```

5) Find average development index and average ART cure rate for each continent using AVG and INNER JOIN.

```
SELECT
    country.continent,
    AVG(country.development_index) AS avg_dev_index,
    AVG(cures_by_ART.num_deaths_averted) AS avg_art_cure
FROM
    country
    INNER JOIN HIV_deaths ON country.country_code = HIV_deaths.country_code
    INNER JOIN cures_by_ART ON HIV_deaths.country_code = cures_by_ART.country_code
GROUP BY
    country.continent;
```

Constraints and Triggers:

We added a population constraint to country table for population to be in range (1775,1421894100)

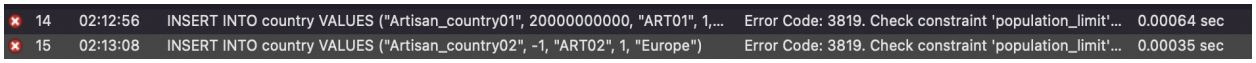
```
ALTER TABLE country
ADD CONSTRAINT population_limit CHECK (population >= 1775 AND population <= 1421864100);

INSERT INTO country VALUES ("Artisan_country01", 2000000000, "ART01", 1, "Europe");
INSERT INTO country VALUES ("Artisan_country02", 1000, "ART02", 1, "Europe");
```

Before adding the constraints:

Artisan_country1	2000000000	ART1	1	Europe
Artisan_country2	1000	ART2	1	Europe

After adding the constraints:



Stored Procedure:

This stored procedure takes a country_code parameter as input, and returns the name, population, development index, life expectancy, and HIV death rate for the specified country. The stored procedure uses a LEFT JOIN to combine the country, life_expectancy, and HIV_deaths tables and retrieve the relevant information.

```
CREATE PROCEDURE get_country_info (IN country_code VARCHAR(10))
  SELECT c.name, c.population, c.development_index, le.age, hd.death_rate_age_5_14
  FROM country c
  LEFT JOIN life_expectancy le ON c.country_code = le.country_code
  LEFT JOIN HIV_deaths hd ON c.country_code = hd.country_code
  WHERE c.country_code = country_code;
```

Creating General Constraints

Pros:

1) Constraints are easy to define and modify using standard SQL syntax.
2) Constraints are automatically applied by the database engine, which is reliable and consistent.
3) Constraints can be used to apply a wide range of data integrity rules, such as uniqueness, referential integrity, and data type constraints.

Cons:

1) Constraints can be inflexible and may not be able to apply complex business rules or validations.
2) Constraints can be difficult to debug and correct if they fail to enforce the desired data integrity rules.

Creating Triggers

Pros:

1)Triggers can be used to apply complex business rules or validations that cannot be easily expressed as constraints.
2)Triggers can be used to customize the behavior of a database application by executing application-specific code in response to database events.
3) Triggers can be used to provide auditing and logging capabilities by tracking changes to specific tables or columns.

Cons:

1)Triggers can be more difficult to define and maintain than constraints.
2)Triggers can have a performance impact on the database engine, especially if they are complex or executed frequently.
3) Triggers can be a source of errors and unexpected behavior if they are not carefully designed and tested.