Coop Pank

Rakendusadministraator

SQL

Vadim Kööp +372 53811395

Tallinn 2023

Table of Contents

1 Task

Is given a database named "Hospital" from which the work process would be carried out. The following conditions are also known with which the work will be carried out:

- Keyword = ALL,
- Difficulty = Hard,
- Completion = ALL.

1.1 Task description

Is given a database named "Hospital" from which the work process would be carried out. The following conditions are also known with which the work will be carried out:

- Keyword = ALL,
- Difficulty = Hard,
- Completion = ALL.

1.2 SQL Database Overview

1.2.1 Table “patients”

- patient_id (INT) - Primary Key
- first_name (TEXT)
- last_name (TEXT)
- gender (CHAR(1))
- birth_date (DATE)
- city (TEXT)
- province_id (CHAR(2)) – Secondary key
- allergies (TEXT)
- height (INT)weight (INT)1.2.2 Table “admissions”
-

1.2.2 Table “admissions”

- patient_id (INT) - Secondary key
- admission_date (DATE)
- discharge_date (DATE)
- diagnosis (TEXT)
- attending_doctor_id (INT) - Secondary key

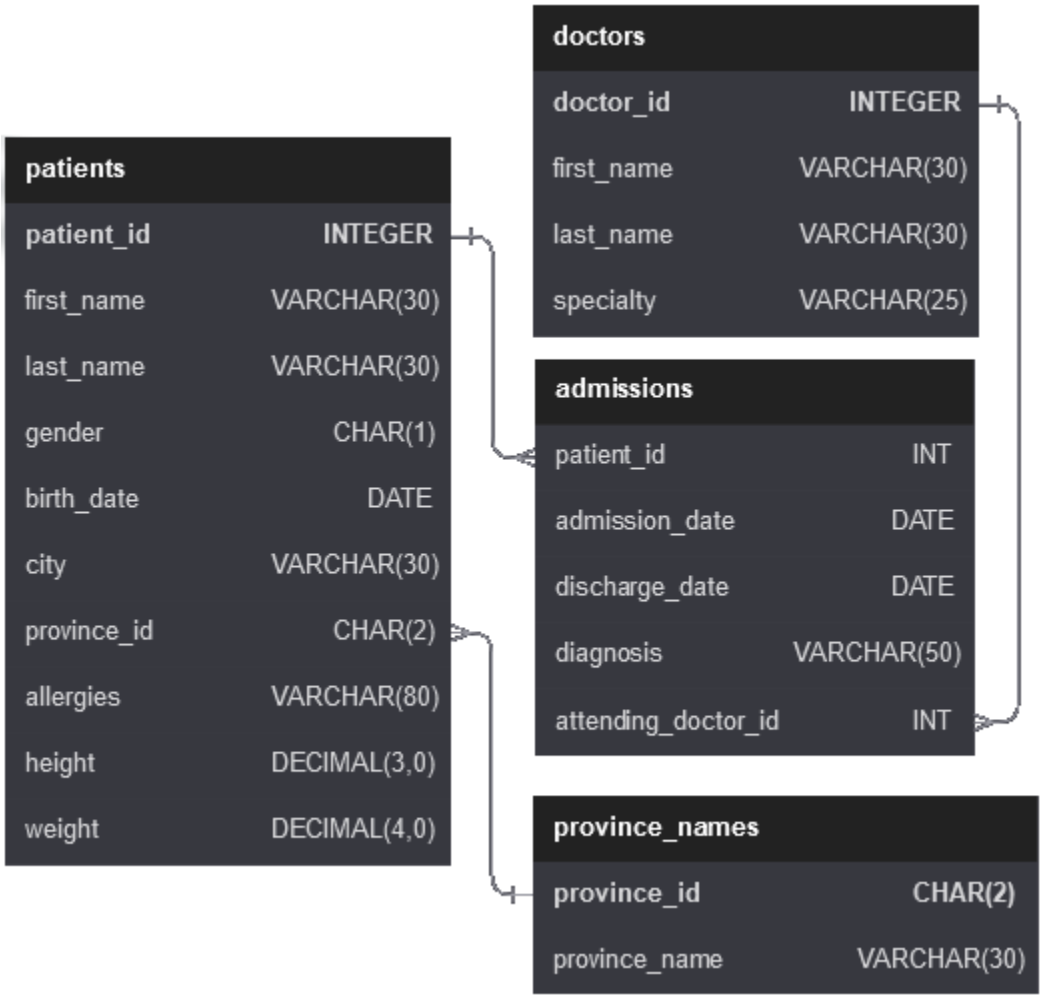
1.2.3 Table “doctors”

- doctor_id (INT) - Primary Key
- first_name (TEXT)
- last_name (TEXT)
- specialty (TEXT)

1.2.4 Table “province_names”

- province_id (CHAR(2)) - Primary Key
- province_name (TEXT)

1.2.5 1.4 SQL Database schema



2 Questions

2.1 Question number 1 (hard)

Show all of the patients grouped into weight groups.

Show the total amount of patients in each weight group.

Order the list by the weight group decending.

For example, if they weight 100 to 109 they are placed in the 100 weight group, 110-119 = 110 weight group, etc.

2.2 Solution for question number 1


```
SELECT (weight - weight % 10) AS new_weight_group,
  COUNT(*) AS total_patients
FROM patients GROUP BY new_weight_group ORDER BY new_weight_group DESC;
```

Explanation:

1) I calculate the weight group by subtracting the remainder of the weight divided by 10 (weight - weight % 10). This effectively rounds down each patient's weight to the nearest multiple of 10, creating weight groups.

2) Then count the number of patients in each weight group using the **COUNT**(*) function.

3) Finally, the result is grouped by the new_weight_group and ordered in descending order of the weight group.

This query will group patients by weight in 10-unit increments and display the total number of patients in each weight group, sorted in descending order.

2.3 Question 2 (hard)

Show patient_id, weight, height, isObese from the patients table.

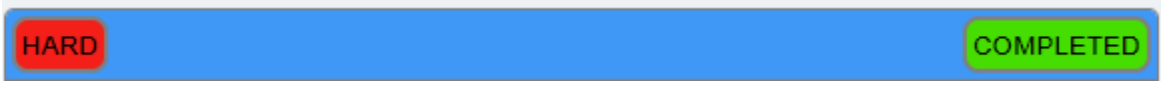
Display isObese as a boolean 0 or 1.

Obese is defined as weight(kg)/(height(m)2) >= 30.

weight is in units kg.

height is in units cm.

2.4 Solution for 2 question




```
SELECT patient_id, weight, height,
  CASE WHEN (weight / (height / 100.0) / (height / 100.0)) >= 30 THEN 1
    ELSE 0
  END AS isObese FROM patients;
```

Explanation:

1) I calculate the Body Mass Index (BMI) by dividing the weight in kilograms by the square of height in meters. To convert height from centimeters to meters, I divide it by 100 (height / 100.0).

2) I use a **CASE** statement to check if the calculated BMI is greater than or equal to 30, which is the threshold for obesity. If the condition is met, it assigns 1 to isObese; otherwise, it assigns 0.

3) The result includes the patient_id, weight, height, and the isObese flag for each patient in the patients table. The isObese column will be displayed as 0 for non-obese patients and 1 for obese patients.

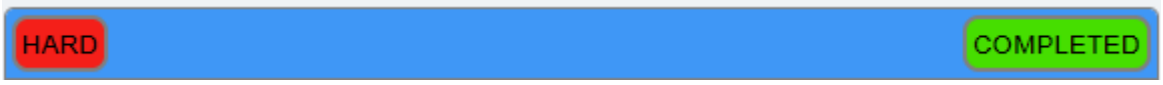
2.5 Question 3 (hard) (There is mistake with Epilepsy and Dementia diagnosis.

Show patient_id, first_name, last_name, and attending doctor's specialty.

Show only the patients who has a diagnosis as 'Epilepsy' and the doctor's first name is 'Lisa'

Check patients, admissions, and doctors tables for required information.

2.6 Solution for 3 question



SELECT p.patient_id, p.first_name, p.last_name, d.specialty **FROM** patients p

INNER JOIN admissions a **ON** p.patient_id = a.patient_id

INNER JOIN doctors d **ON** a.attending_doctor_id = d.doctor_id

WHERE a.diagnosis = 'Epilepsy' **AND** d.first_name = 'Lisa';

Explanation:
1) I select the following columns:

- p.patient_id: The patient's ID.
- p.first_name: The patient's first name.
- p.last_name: The patient's last name.
- d.specialty: The specialty of the attending doctor.

2) I perform **INNER JOIN** operations to combine data from multiple tables:

- I join the "patients" table (aliased as "p") with the "admissions" table (aliased as "a") based on the patient's ID.
- I join the "doctors" table (aliased as "d") with the "admissions" table (aliased as "a") based on the attending doctor's ID.

3) In the **WHERE** clause, I specify the conditions to filter the results:

- a.diagnosis = 'Epilepsy' ensures that only admissions with a diagnosis of 'Epilepsy' are selected.
- d.first_name = 'Lisa' ensures that only doctors with the first name 'Lisa' are selected.

This query will return the patient information for patients diagnosed with 'Epilepsy' and whose attending doctor's first name is 'Lisa.'

2.7 Question 4 (hard)

All patients who have gone through admissions, can see their medical documents on our site. Those patients are given a temporary password after their first admission. Show the patient_id and temp_password.

The password must be the following, in order:

1. patient_id
2. the numerical length of patient's last_name
3. year of patient's birth_date

2.8 Solution for 4 question

SELECT p.patient_id, **CONCAT**(p.patient_id, **LENGTH**(p.last_name), **YEAR**(p.birth_date)) **AS** new_temp_password

FROM patients p **WHERE** p.patient_id **IN** (**SELECT DISTINCT** patient_id **FROM** admissions);

Explanation:

1) I select the patient_id from the patients table.

2) I use the **CONCAT** function to concatenate the following components to create the temporary password:

- patient_id: The patient's unique identifier.
- **LENGTH**(p.last_name): The numerical length of the patient's last name.
- **YEAR**(p.birth_date): The year of the patient's birth date.

3) I filter the patients by checking if their patient_id exists in the list of distinct patient IDs from the admissions table using a subquery.

This query will return the patient_id and the corresponding temporary password for patients who have gone through admissions.

2.9 Question 5 (hard)

Each admission costs $50 for patients without insurance, and $10 for patients with insurance. All patients with an even patient_id have insurance.

Give each patient a 'Yes' if they have insurance, and a 'No' if they don't have insurance. Add up the admission_total cost for each has_insurance group.

2.10 Solution for 5 question


```
SELECT CASE WHEN p.patient_id % 2 = 0 THEN 'Yes'
    ELSE 'No'  END AS has_insurance,
  SUM(CASE WHEN p.patient_id % 2 = 0 THEN 10 -- Patients who have insurance
       ELSE 50 -- Patients who do not have insurance
      END) AS admission_total_cost
FROM patients p JOIN admissions a ON p.patient_id = a.patient_id GROUP BY has_insurance;
```

Explanation:

1) I use a **CASE** statement to determine the insurance status for each patient based on their patient_id. If the patient_id is even, it is considered 'Yes,' indicating that the patient has insurance; otherwise, it is 'No,' indicating that the patient does not have insurance.

2) I calculate the total admission cost for each patient based on their insurance status. Patients with insurance are charged $10, and patients without insurance are charged $50. I use another **CASE** statement to calculate this cost.

3) I join the patients and admissions tables based on the patient_id.

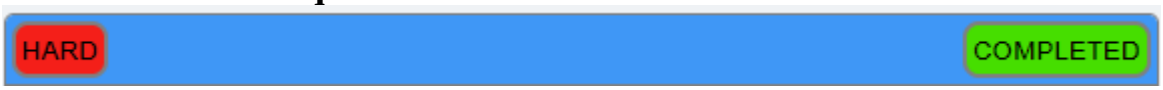
4) Finally, I group the results by the has_insurance column, providing the total admission cost for each insurance group.

This query will give a result with two rows, one for patients with insurance and one for patients without insurance, along with the total admission cost for each group.

2.11 Question 6 (hard)

Show the provinces that has more patients identified as 'M' than 'F'. Must only show full province_name

2.12 Solution for 6 question



SELECT pn.province_name **FROM** province_names pn

JOIN patients p ON pn.province_id = p.province_id

GROUP BY pn.province_name

HAVING SUM(**CASE WHEN** p.gender = 'M' **THEN** 1 **ELSE** 0 **END**) > **SUM**(**CASE WHEN** p.gender = 'F' **THEN** 1 **ELSE** 0 **END**);

Explanation:

1) I join the province_names and patients tables based on the province_id to link provinces to patients in those provinces.

2) I use the **GROUP BY** clause to group the result by province_name.

3) The **HAVING** clause is used to filter the results. It compares the count of male ('M') patients (using **SUM**(**CASE WHEN** p.gender = 'M' **THEN** 1 **ELSE** 0 **END**)) to the count of female ('F') patients (using **SUM**(**CASE WHEN** p.gender = 'F' **THEN** 1 **ELSE** 0 **END**)), and it selects only the provinces where there are more male patients than female patients.

This query will return the full province names for provinces where there are more male patients than female patients.

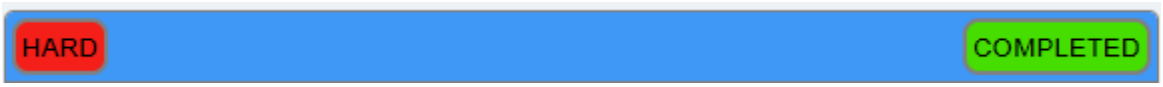
2.13 Question 7 (hard)

We are looking for a specific patient. Pull all columns for the patient who matches the following criteria:

- First_name contains an 'r' after the first two letters.
- Identifies their gender as 'F'
- Born in February, May, or December
- Their weight would be between 60kg and 80kg
- Their patient_id is an odd number
- They are from the city 'Kingston'

2.14 Solution for 7 question

My solution which should be right, but website does not accept it:




```
SELECT * FROM patients WHERE first_name LIKE '__r%'

  AND gender = 'F'

  AND MONTH(birth_date) IN (2, 5, 12)

  AND weight BETWEEN 60 AND 80

  AND patient_id % 2 = 1

  AND city = 'Kingston';
```

Explanation:

1) first_name **LIKE** '__r%': This condition checks if the "first_name" has 'r' as the third character. The double underscores represent any two characters, so this condition ensures that there are two characters followed by 'r' in the "first_name."

2) gender = 'F': It checks that the patient's gender is 'F', indicating a female patient.

3) **MONTH**(birth_date) **IN** (2, 5, 12): This condition checks if the birth month (extracted from "birth_date") is either February (2), May (5), or December (12).

4) weight **BETWEEN** 60 **AND** 80: It ensures that the patient's weight falls between 60 and 80 kilograms.

5) patient_id % 2 = 1: This condition checks if the "patient_id" is an odd number.

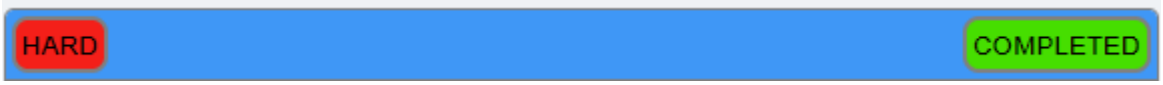
6) city = 'Kingston': It filters patients who are from the city 'Kingston'.

This query combines all these conditions using the **AND** operator, ensuring that only patients meeting all the specified criteria are returned in the result set.

2.15 Question 8 (hard)

Show the percent of patients that have 'M' as their gender. Round the answer to the nearest hundreth number and in percent form.

2.16 Solution for 8 question



SELECT CONCAT(ROUND((COUNT(*) * 100.0 / (**SELECT COUNT**(*) **FROM** patients)), 2), '%') **AS** percent_male

FROM patients **WHERE** gender = 'M';

Explanation:

1) I calculate the percentage of male patients by dividing the count of male patients by the total count of all patients in the subquery.

2) The **ROUND** function is used to round the result to two decimal places.

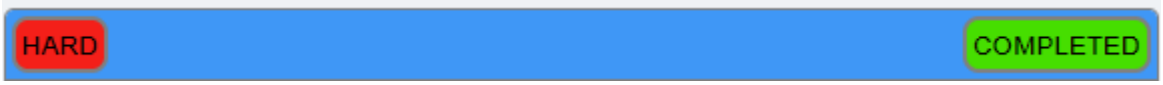
3) I use **CONCAT** to append the percentage symbol '%' to the result, converting it to percent form.

This query should provide with the percentage of patients with 'M' as their gender, rounded to the nearest hundredth, and in percent form.

2.17 Question 9 (hard)

For each day display the total amount of admissions on that day. Display the amount changed from the previous date.

2.18 Solution for 9 question



SELECT admission_date,

COUNT(admission_date) **AS** new_admission_day,

COUNT(admission_date) - **LAG**(count(admission_date)) **OVER**(**ORDER BY** admission_date) **AS** new_admission_count_change

FROM admissions **GROUP BY** admission_date

Explanation:

1) **SELECT** admission_date, count(admission_date) as new_admission_day, ...: In this part of the query, I am selecting the "admission_date" and counting the number of admissions for each "admission_date." I give this count the alias "admission_day."

2) count(admission_date): This part calculates the total number of admissions for each day. It uses the **COUNT** function to count the occurrences of each unique "admission_date."

3) **LAG**(count(admission_date)) **OVER**(**ORDER BY** admission_date) **AS** new_admission_count_change: This is the key part of the query. It calculates the change in admissions from the previous day.

- **LAG**(count(admission_date)) calculates the value of "count(admission_date)" for the previous row based on the specified **ORDER BY** clause (which is ordered by "admission_date").
- **OVER**(**ORDER BY** admission_date) is the window function clause that defines the window over which the **LAG** function operates.
- **AS** admission_count_change assigns the calculated change to the alias "admission_count_change."

4) **FROM** admissions: This specifies the "admissions" table as the source of data for the query.

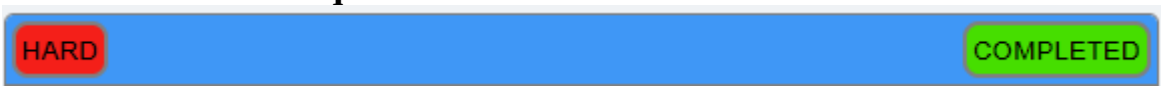
5) **GROUP BY** admission_date: The **GROUP BY** clause groups the results by "admission_date," gets a count for each unique admission date.

The end result of this query will be a list of "admission_date" values, the total number of admissions ("admission_day") for each date, and the change in admissions ("admission_count_change") from the previous day. This information allows to track the daily admission count and understand how it changes over time.

2.19 Question 10 (hard)

Sort the province names in ascending order in such a way that the province 'Ontario' is always on top.

2.20 Solution for 10 question



SELECT province_name **FROM** province_names **ORDER BY**

CASE WHEN province_name = 'Ontario' **THEN** 1

ELSE province_name **END**

Explanation:

1) **SELECT** province_name: This part of the query selects the "province_name" column from the "province_names" table. It specifies that I want retrieve only the names of the provinces.

2) **FROM** province_names: This clause specifies the source table from which the data is selected, which is the "province_names" table.

3) **ORDER BY**: This clause is used to define the sorting order for the result set.

4) **CASE** ... **END**: Within the **ORDER BY** clause, a **CASE** expression is used to control the sorting. It assigns a sort value to each province name based on the condition provided.

- **WHEN** province_name = 'Ontario' **THEN** 1: When the province name is 'Ontario,' it assigns a sort value of 1. This ensures that 'Ontario' will have the lowest sort value and will appear at the top of the sorted list.

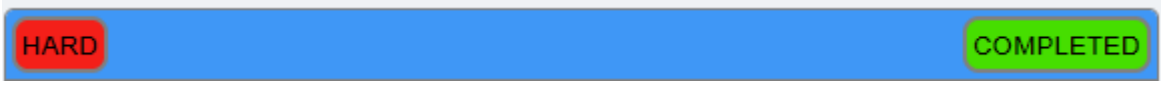
- **ELSE** province_name: For all other province names, it assigns the province name itself as the sort value. This means that other province names will be sorted alphabetically.

By using this **CASE** expression within the **ORDER BY** clause, the query creates a custom sorting order. It places 'Ontario' at the top (sort value 1) and then sorts the remaining provinces alphabetically. This way, I get the desired sorting with 'Ontario' on top and other provinces in ascending alphabetical order.

2.21 Question 11 (hard)

We need a breakdown for the total amount of admissions each doctor has started each year. Show the doctor_id, doctor_full_name, specialty, year, total_admissions for that year.

2.22 Solution for 11 question



SELECT d.doctor_id as new_doctor_id,

CONCAT(d.first_name,' ', d.last_name) **AS** new_doctor_full_name,

d.specialty,

YEAR(a.admission_date) **AS** new_selected_year,

COUNT(*) **AS** new_total_admissions

FROM doctors **AS** d

LEFT JOIN admissions as a **ON** d.doctor_id = a.attending_doctor_id

GROUP BY new_doctor_full_name, new_selected_year

ORDER BY doctor_id, new_selected_year

Solution:

1) In the **SELECT** clause, I am selecting the following columns and using aliases for them:

- d.doctor_id as new_doctor_id: The doctor's **ID** with a new alias.
- **CONCAT**(d.first_name, ' ', d.last_name) **AS** new_doctor_full_name: The doctor's full name with a new alias.
- d.specialty: The doctor's specialty.
- **YEAR**(a.admission_date) **AS** new_selected_year: The year from the "admission_date" with a new alias.
- **COUNT**(*) **AS** new_total_admissions: The count of admissions with a new alias.

2) I use a **LEFT JOIN** to join the "doctors" table (aliased as "d") with the "admissions" table (aliased as "a") based on the doctor's **ID**.

3) In the **GROUP BY** clause, I group the results by "new_doctor_full_name" and "new_selected_year," using the aliases I've defined.

4) The **ORDER BY** clause orders the results first by "doctor_id" and then by "new_selected_year."

This query should work as intended and provide the desired breakdown of the total number of admissions each doctor has started each year. The use of aliases helps make the query more readable.