Ghent University - TELIN - DDCM
2021-2022

Assignment 2: SQL Advanced

In this assignment, you will test your skills related to advanced SQL acquired during the last lectures and lab sessions. The SQL questions will feature subqueries and aggregation in particular, but still, knowledge about basic functionality and combination of multiple tables is expected.

All practical information related to this assignment is listed in the SQL introductory document (`sqlintroduction.pdf`) on Ufora. We recommend to read carefully through this document before starting the assignment and, maybe a second time, before the deadline (March 25th, 2022, 10pm) in order to submit correctly. If you have any additional questions, do not hesitate to contact us.

Part 1: Interpretation of advanced SQL

1. Consider the following `SELECT`-query.

```
SELECT DISTINCT license_plate
FROM registration r1
WHERE NOT EXISTS(SELECT 1
                 FROM registration r2
                 WHERE r1.email != r2.email
                    AND r1.license_plate = r2.license_plate);
```

   Describe, in your own words, what this `SELECT`-query achieves. You should not give the result table of this query or explain this query in technical terms (i.e. we do not expect a literal translation of the operations that are performed in the query), but explain what the semantical outcome is of this query when executed on data that is stored in the `rollsrobin` database. Provide your answer in a short report.

2. Rewrite the following SELECT-query such that it does not use aggregation functions, grouping and having. However, it is important that the result table retrieved by your query equals exactly the result table retrieved by the original query, so try to understand the original query first. Add a file with the name `studentcode_firstname_lastname_1_2.sql`, in which you substitute 'studentcode', 'firstname' and 'lastname' by resp. your studentcode, firstname and lastname, to the .zip file containing the rewritten SELECT-query.

```
SELECT DISTINCT r.email FROM registration r
    INNER JOIN car c USING (license_plate)
GROUP BY r.email
HAVING COUNT(DISTINCT c.enterprisenumber) = 1;
```

3. Consider the following task for which a SELECT-query should be written.

   *Give a list of email addresses of all persons (one or multiple) who registered a car for the **longest** rental period present in the database. Here, the length of a rental period is based on the number of days from `period_begin` until `period_end` (boundaries inclusive).*

   *Only persons who started this (longest) rental period the **earliest** (based on `period_begin`) of all people renting for the longest period should be returned by your query. In the result table, only one column `email` of datatype `varchar` is expected.*

   In order to illustrate this task, consider the example data given in Table 1. Given these data, the query should return only one email address, i.e. 'person1@ugent.be'. The reason for this is that both person 1 and person 3 had the longest rental period in the database (i.e. 12 days), but the begin date of this rental period of person 1 (i.e. 2018-06-12) was earlier than the begin date of this rental period of person 3 (i.e. 2020-02-14).

email	period_begin	period_end	# days
person1@ugent.be	2018-06-12	2018-06-23	12
person1@ugent.be	2017-06-27	2017-06-28	2
person2@ugent.be	2018-05-03	2018-05-04	2
person2@ugent.be	2019-12-15	2020-12-19	5
person3@ugent.be	2020-02-14	2020-02-25	12

Table 1: Example data part 1, exercise 3.

   Now have a look at the following SELECT-query.

```
SELECT DISTINCT email FROM registration INNER JOIN person USING (email)
WHERE period_end - period_begin >= ALL (
        SELECT period_end - period_begin FROM registration)
    AND period_begin <= ALL (SELECT period_begin FROM registration);
```

Does this query solve the task that was described above in all possible situations? If not, explain in your own words what is wrong with the query and how you can solve this. Provide your answer in a short report.

Part 2: Advanced SQL

Provide SQL `SELECT`-queries as solutions to the following exercises related to the `rollsrobin` database. Each query should be added to a separate .sql file with filename `studentcode_firstname_lastname_2_X.sql` in which you substitute 'studentcode', 'firstname' and 'lastname' with your studentcode, firstname and lastname respectively, and 'X' with the number of the question (1 to 5). Add these .sql files to your final .zip file.

1. Give for each car in the database that was rented (registered) at least once, the number of nights that passed by between the first time that the car was rented (based on `period_begin` of the first registration, exclusive) and the last time that the car was rented (based on `period_begin` of the last registration, inclusive). In the result table, we expect two columns with corresponding datatype: `license_plate` (`varchar`) and `passed_nights` (`integer`).

   Example: Table 2 shows a list of registrations (only the license plate and begin of the registration period are shown). Given this data, the expected result table is shown in Table 3.

license_plate	**period_begin**
1-AEK-885	2018-06-12
1-AEK-885	2018-06-27
1-DKA-429	2018-05-03
1-TZH-641	2019-12-15
1-TZH-641	2020-02-15
1-TZH-641	2020-02-18

Table 2: Example data part 2, exercise 1.

license_plate	**passed_nights**
1-AEK-885	15
1-DKA-429	0
1-TZH-641	65

Table 3: Example result table part 2, exercise 1.

2. Give the employee who rented the highest number of *unique cars* (so, not the employee that did the highest number of *unique car rentals*). In the result

table, we expect one column with name `email` and datatype `varchar`. Make sure that your query takes into account ex aequos.

Example: Table 4 shows a list of unique cars rented per person (for both non-employees and employees). Given this data, we only expect the employee with email address 'person1@ugent.be' in the end result, because this employee rented the highest number of unique cars of all employees.

license_plate	**email**	**is_employee**
1-AAA-111	person1@ugent.be	true
2-BBB-222	person1@ugent.be	true
3-CCC-333	person1@ugent.be	true
4-DDD-444	person2@ugent.be	true
5-EEE-555	person2@ugent.be	true
1-AAA-111	person3@ugent.be	false
3-CCC-333	person3@ugent.be	false
4-DDD-444	person3@ugent.be	false
5-EEE-555	person3@ugent.be	false

Table 4: Example data part 2, exercise 2.

3. Give, for each *car brand* (so not *car model*), the total number of times that an employee rented a car of this brand himself/herself during his/her contract at the same company that owns the rented car (i.e. `registration.period_begin` should be between `contract.period_begin` and `contract.period_end`, boundaries inclusive). In the result table, we expect two columns with corresponding datatype: `brand` (`varchar`) and `amount` (`integer`). Also include brands that are persisted in the database and for which no car rental meets the requirements, with an amount of 0. Sort the results first on amount (numerically descending) and then on brand (alphabetically ascending).

   Example: Table 5 shows a list of contracts of employees and Table 6 shows a list of registrations done by these employees. Given this data and the fact that 'Renault', 'Peugeot', 'Citroën, 'Volkswagen' and 'Opel' are stored in the database, the expected result table is given in Table 7

4. Calculate, for each email domain used by an employee, two different numerical values, which are

   - the percentage share of unique employees using this email domain on the total number of unique persons (i.e. employees and non-employees) using this email domain (column `percentage_employees`), and

email	begin contract	end contract	enterprisenumber contract
person1@ugent.be	2022-02-01	2023-02-01	101
person2@ugent.be	2022-01-16	2022-04-16	101
person2@ugent.be	2022-04-17	2022-05-31	102

Table 5: Example contract data part 2, exercise 3.

email	begin registration	end registration	enterprisenumber rental car	brand rental car
person1@ugent.be	2022-02-05	2022-02-07	101	Renault
person1@ugent.be	2022-02-01	2022-02-03	102	Peugeot
person1@ugent.be	2022-02-01	2022-02-10	101	Citroën
person1@ugent.be	2022-07-12	2022-07-15	101	Peugeot
person2@ugent.be	2022-01-18	2022-01-18	101	Volkswagen
person2@ugent.be	2022-02-01	2022-02-03	101	Volkswagen
person2@ugent.be	2022-04-15	2022-04-17	102	Volkswagen
person2@ugent.be	2022-05-31	2022-06-18	102	Citroën

Table 6: Example registration data part 2, exercise 3.

- the percentage share of unique cars rented by employees using this email domain on the total number of unique cars rented by persons (i.e. employees and non-employees) using this email domain (column `percentage_cars`).

You may assume that domains in an email address start right after the '@' symbol (and that there is only one '@' symbol in an email address) and end right before the final '.' symbol (e.g. gmail, hotmail,...). Besides that, you may also assume that all persons did at least one registration. So, in the result table, we expect three columns with corresponding datatype: `email_domain` (`varchar`), `percentage_employees` (`numeric`) and `percentage_cars` (`numeric`). The values in columns `percentage_employees` and `percentage_cars` should be between 0 and 100 (boundaries inclusive) and should be rounded up to two decimals.

Example: Table 8 shows a list of unique cars rented per person (for both non-employees and employees). Given this data, the expected result table is shown in Table 9. As an example, of the five persons using the 'gmail' domain, four of them are employees (i.e. $4/5 = 80\%$) and of the four unique cars rented by persons using the 'gmail' domain, three of these cars are also rented by an employee (i.e. $3/4 = 75\%$). if there is a non-employee using email domain 'hotmail', but no employee is using this domain, 'hotmail should not be listed

brand	amount
Citroën	2
Renault	1
Volkswagen	1
Opel	0
Peugeot	0

Table 7: Example result table part 2, exercise 3.

in the end result.

email_domain	email	license_plate	is_employee
gmail	person1@gmail.com	1-HLM-435	false
gmail	person2@gmail.com	1-DAK-198	true
gmail	person3@gmail.com	1-BGP-156	true
outlook	person4@outlook.com	1-AAA-111	true
outlook	person5@outlook.com	1-AAA-111	true
gmail	person3@gmail.com	1-FFV-642	true
gmail	person6@gmail.com	1-DAK-198	true
telenet	person7@telenet.be	1-FFV-642	false
outlook	person8@outlook.com	1-AAA-111	false

Table 8: Example data part 2, exercise 4.

email_domain	percentage_employees	percentage_cars
gmail	80	75
outlook	66.66	100

Table 9: Example result data part 2, exercise 4.

5. Return the license plate of the car of which the total number of times that this car was rented, is the closest to the average number of times that a car is rented (computed over all cars present in the database). Be aware of the fact that, in order to calculate the average, you should also take into account the cars that have never been rented before. In the result table, only one column `license_plate` of datatype `varchar` is expected. Make sure that your query takes into account ex aequos.

   Example: Assume that there are only four cars in the database. Table 10 shows a list of rentals by specific persons of these cars (and also shows cars for which no persons exist that rented the car). The average number of times

that a car is rented based on these data is 1.25 ($= (1+1+3+0)/4$). As such, the according result table should look like Table 11.

email	**license_plate**
person1@gmail.com	1-HLM-435
person2@gmail.com	1-DAK-198
person1@outlook.com	1-AAA-111
person1@outlook.com	1-AAA-111
person3@outlook.com	1-AAA-111
/	1-FFV-642

Table 10: Example data part 2, exercise 5.

license_plate
1-HLM-435
1-DAK-198

Table 11: Example result data part 2, exercise 5.