SQL test 1 25.04.2023

For each of the following tasks, write appropriate SQL commands. We expect the solution to be in the form of a file containing the SQL **commands** themselves, not the results obtained. Any syntactically incorrect queries will not be checked, so please check your solution using, for example, the command `\i file.sql` You can send the file multiple times, but only the latest version will be checked. Remember that before executing any database-modifying operations, you can issue the command `BEGIN;`, and then `ROLLBACK;` (if you want to undo all changes) or `COMMIT;` (if you want to save them).

Load the `offers.sql` file into your database. Send the solution via the form at `https://dbserv.stud.ii/`. Do it as often as possible! All data on your computer will be erased after restarting. In case of any problems, please contact the course instructor before restarting your computer.

The format of the first line of the solution should be: `-- mabi-firstname-lastname`, e.g., `mabi-Jan-Kowalski`. The required format of the entire solution file is:

```
-- mabi-firstname-lastname
-- Task 1
<query>

-- Task 2
<query>
...
```

Task 1 (1 pt) Write a query that adds exactly one entry to the `company_branch` table. Practice using `BEGIN;` and then `ROLLBACK;` to avoid making any permanent changes to the database.

Task 2 (2 pts)

List (without duplicates) the names of companies looking for people with knowledge of PostgreSQL (i.e., the appropriate value of `name` in `skill` contains the substring `'PostgreSQL'`). Take letter case into account.

The sample query returns 47 tuples.

Task 3 (1 pt)

Unfortunately, sometimes there are typos in job postings - modify the query from the previous task to also include all requirements that contain the substring `'postres'` (case-insensitive). Sort the results alphabetically.

The sample query returns 50 tuples.

Rozwiązanie

```
SELECT DISTINCT c.name
FROM offer o JOIN
      skill s ON (o.id=s.offer_id) JOIN
      company c ON (c.id=o.company_id)
WHERE s.name LIKE '%PostgreSQL%' OR
      s.name ILIKE '%postres%'
ORDER BY 1;
```

Task 4 (4 pts)

Some people are annoyed by companies that do not provide the salary range in their job advertisements or the provided range is very wide. Let's investigate this phenomenon. Consider only offers containing the salary range for B2B contracts published on Saturday, April 22, 2023 or later, indicating the salary for B2B in Polish zlotys (PLN) whose lower bound is non-zero. List tuples consisting of: company name `name`, job title `title`, experience level `experience_level`, lower bound `salary_from_b2b`, upper bound `salary_to_b2b`, the difference between the upper and lower bounds, and the percentage of the lower bound that difference represents.

Sort the results by the last value in ascending order, and then alphabetically by company name. Round the percentages to the nearest integer (see the mathematical functions section in the documentation). Is the floating-point type appropriate for monetary data?

Rozwiązanie

```
SELECT c.name, title, experience_level,
       salary_from_b2b::decimal, salary_to_b2b::decimal,
       round(salary_to_b2b::decimal - salary_from_b2b::decimal) AS diff,
       round(100*(salary_to_b2b::decimal-salary_from_b2b::decimal)
                  /salary_from_b2b::decimal) AS "%"
FROM offer JOIN
     company c ON c.id = offer.company_id
WHERE salary_from_b2b::decimal>0 AND
      salary_currency_b2b='pln' AND
      published_at::date >= '22.04.2023'
ORDER BY 7,1
```

The sample query returns 2749 tuples and uses type casts (e.g. `salary_from_b2b::decimal`, `published_at::date`).
Selected rows from the result:

```
IN Team    | Node.js Developer      | mid     |  21840 | 26880 | 5040 | 23
ITDS       | Application Engineer   | junior  |  13000 | 16000 | 3000 | 23
Superdevs  | Senior Python Engineer | senior  |  26000 | 32000 | 6000 | 23
```

Task 5 (2 pts)

Create a table called `salary` that stores data about salaries for individual job offers. The table should have the following columns: the lower and upper salary threshold, `salary_from` and `salary_to`, of type `decimal`, an offer identifier `offer_id` of type `int` (referring to the `id` of the respective job offer - foreign key), `type` of type `text` that takes on values `'b2b'`, `'permanent'`, or `'mandate'` (but you don't have to impose a constraint limiting the allowed contents of this field to these 3 strings), and `currency` of type `text` containing the currency in which the salary thresholds are given (you also don't have to limit the contents of this field).

Task 6 (2 pts)

Insert data on salaries in the B2B context into the `salary` table (you can use the `INSERT INTO salary SELECT ...` construction). Skip offers from the `offer` table for which the currency is unknown or one of the salary thresholds is zero.

Task 7 (2 pts)

Change the data type of the column `salary_from_b2b` to `decimal`.
Hint: You can do the task in three steps by adding a new column, copying values, and deleting the old column, or check the documentation on how to use the `ALTER TABLE ... SET DATA TYPE ... USING` command.

Task 8 (1 pt)

Remove the column `if_b2b` from the `offer` table.

Rozwiązanie

```
CREATE TABLE salary(salary_from decimal NOT NULL,
                    salary_to decimal NOT NULL,
                    offer_id int REFERENCES offer(id) NOT NULL,
                    type text,
                    currency text);

INSERT INTO salary
   SELECT round(salary_from_b2b::decimal),
         round(salary_to_b2b::decimal),
         id,
         'b2b',
         salary_currency_b2b
   FROM offer
   WHERE salary_currency_b2b<>'unknown' AND
         salary_from_b2b::decimal>0;

ALTER TABLE offer
```

```
    ALTER COLUMN salary_from_b2b
    SET DATA TYPE decimal
    USING salary_from_b2b::decimal;

ALTER TABLE offer DROP COLUMN if_b2b;
```