SQL Exam No. 4 23.05.2024

For each of the following tasks, write appropriate SQL commands. We expect solutions in the form of one file containing **the content** of SQL commands, not the results of the query. Any queries with incorrect syntax will not be checked. Check your solution, for example, using `\i file.sql`! You can send the file multiple times; only the latest version will be checked. Remember that before executing a database-modifying operation, you can issue the command `BEGIN;` and then `ROLLBACK;` (if you want to revert all changes) or `COMMIT;` (if you want to save them).

Load the file `jjit.sql` into your database. Send your solutions through the form at `https://dbserv.stud.ii/`. Do this as often as possible! All data on your computers is erased after a restart. In case of any problems, be sure to contact the course instructor before restarting your computer.

The format of the first line of the solution: `-- group-firstname-lastname`, where `group` are the initials of the instructor leading your group (`jotop/mabi/plg/pwi`), e.g., `pwi-Jan-Kowalski`.

You can earn a total of 16 points but the maximum includes 14.

For this test, the previous database has been modified with the following commands (they are already included in the provided database dump):

```
ALTER TABLE offer ALTER COLUMN company_id DROP not null;
UPDATE offer SET company_id = null
   WHERE company_id IN (19278, 19756, 19138,19092, 19290, 19675, 19382);
UPDATE employment_details SET currency='eur' WHERE salary_from IS NULL;
```

Task 1 (4 points). For each of the known companies, list: its name; the number of stationary job offers it has (i.e., such that the `remote` field has value "false"); the number of countries where these offers are available; and the publication times of the oldest and newest such offer. Order the results in ascending order of company name. The result should also include companies that offer remote work only.

ARCHE CONSULTING has five such offers from one country, published in the last three days of August 2023.

Solution

```
SELECT name, COUNT(offer.id), COUNT(DISTINCT country_code),
             MIN(published_at), MAX(published_at)
  FROM company LEFT JOIN
       offer ON (NOT remote AND company.id = company_id)
  GROUP BY company.id
  ORDER BY 1;
```

Task 2 (4 points). Write out company names that are prefixes of some other company names. When checking this criterion, take into account letter case, but omit spaces and tabs at both ends of the name strings (but not in the values that are returned

– these are different names of different companies!). Order the results first by name length (ascending), then alphabetically by name.

The reference query returns 20 names ranging from `'GN'` *to* `'Raiffeisen Bank International'`*, including two (different) looking like* `'Accenture'`*. The name* `'Kaczmarski'` *does not appear in the result. A string containing escape characters must be preceded by the character* `E`*, i.e., a string containing a single tab character is* `E'\t'`*.*

Relevant PostgreSQL documentation: 9.4 String Functions and Operators.

Solution

```
SELECT name FROM company AS super
  WHERE EXISTS (
    SELECT 1 FROM company AS sub
      WHERE TRIM(TRAILING E' \t' FROM sub.name) ^@
            TRIM(TRAILING E' \t' FROM super.name)
        AND TRIM(TRAILING E' \t' FROM sub.name)
            != TRIM(TRAILING E' \t' FROM super.name)
  )
  ORDER BY LENGTH(name), name;

  -- zamiast operatora ^@ można użyć funkcji starts_with()
```

Task 3 (4 points). When asked to write a query that calculates the median of minimum salaries offered in euros, an AI tool provided the following solution:

```
WITH ordered_salaries AS (
   SELECT
      salary_from,
      RANK() OVER (ORDER BY salary_from) AS rn,
      COUNT(*) OVER () AS total_count
   FROM
   employment_details
   WHERE currency='eur'
)
SELECT
  AVG(salary_from) AS median_salary
FROM
  ordered_salaries
WHERE
  rn >= (total_count + 1) / 2 AND rn <= (total_count + 2) / 2;
```

This query does not work correctly – figure out why and correct it. In your solution, include the corrected query and a comment on what has changed.

Relevant PostgreSQL documentation: 3.5, 9.22.

Solution We need to filter out tuples without a specified minimum salary by adding the condition `salary_from IS NOT NULL` and change `rank()` to `row_number()`. The reason is the way the `rank()` function numbers positions "with gaps" – details can be found in the linked tutorial and in the function description. Another interesting solution is to eliminate all ties by changing the sorting within the group to `ORDER BY salary_from, offer.id`.

Task 4 (4 points). Write a `CREATE INDEX` command that creates an index (or indexes) that should speed up the deletion of tuples from the `company` table that are not referenced (via `company_id`) from the `offer` table. Assume that (few) such tuples are present, as in the initial state of the database.

Write a query that deletes such tuples. Test your proposed changes by running this query and displaying its schedule and execution times, before and after the changes. Comment on why the created index improved the performance time.

Consider what steps the engine needs to perform in order to execute your query. Start by comparing the runtime when there are some tuples in the `company` table that meet the specified delete condition and when they are no longer there. Carefully read the measurement results, do not limit yourself to the total execution time. Why does the query run faster in the latter case? When experimenting, remember that the command whose running time you are measuring really is executed. If you commit such changes then reload the database dump.

Solution

```
CREATE INDEX ON  offer(company_id);

EXPLAIN ANALYZE
DELETE FROM company c WHERE c.id NOT IN (
  SELECT company_id FROM offer WHERE company_id IS NOT NULL
);
```

Adding the index defined above allows us to quickly determine that deleting a tuple from `company` is safe because no tuple from `offer` references it anymore. This could be discovered by analyzing the results of `EXPLAIN ANALYZE DELETE ...`. We see that when we actually delete a tuple, the `Trigger for constraint offer_company_id_fkey` is triggered, and its operation takes (relatively) a lot of time. When there are no tuples to delete, the above trigger is not activated.

Defining such an index for each foreign key is a good design practice and can be omitted only if we have reasons to believe that it will not cause performance issues.