Домашнее задание по лекции Базы Данных и SQL

Все задания нужно оформить в виде отдельных SQL скриптов, с именем по номеру соответствующей задачи (например, `task_1.sql`). Прислать лектору (Яркин Станислав) ссылку на пулл реквест до 22.12.2019 включительно в слак. Все скрипты должны отрабатывать без ошибок на чистой базе, если запускать их последовательно на postgres версии 9.6 и выше.

1. (*2 балла*) Разработать схему данных для хранения информации о вакансии, резюме и откликах (можно опустить поле user_id и записывать отклик как пару резюме и вакансии). написать SQL скрипт создания соответствующих таблиц. Нужно отразить только основные поля, которые видны на странице вакансии и резюме. Схему хранения адреса можно не создавать. Пример создания таблицы вакансий, разобранный на лекции, приведен в Приложении 1. В таблицах необходимо отразить поля, которые понадобятся для запросов из пунктов 3-6, а также из задания 8 варианта а) Также нужно добавить FOREIGN KEY связь между `vacancy` и `vacancy_body` по `vacancy_body_id`.
2. (*1 балл*) Наполнить полученные таблицы случайными данными: 10000 вакансий, 100000 резюме и 50000 откликов. Пример, как заполнить таблицу вакансий приведен в Приложении 2. В примере есть ошибки - их нужно исправить самостоятельно.

   В заданиях 3-6 нужно написать один `SELECT` запрос:
3. (*1 балл*) Вывести среднюю величину предлагаемой зарплаты по каждому региону (area_id): средняя нижняя граница, средняя верхняя граница и средняя средних. Нужно учесть поле `compensation_gross`, а также возможность отсутствия значения в обоих или одном из полей со значениями зарпаты.
4. (*1 балл*) В каком месяце было опубликовано больше всего резюме? В каком месяце было опубликовано больше всего вакансий? Вывести оба значения в одном запросе.
5. (*1 балл*) Вывести названия вакансий в алфавитном порядке, на которые было меньше 5 откликов за первую неделю после публикации вакансии. В случае, если на вакансию не было ни одного отклика она также должна быть выведена.
6. (*1 балл*) Для каждого резюме вывести его идентификатор, массив из его специализаций, а также самую частую специализацию у вакансий, на которые он откликался (`NULL` если он не откликался ни на одну вакансию).

Для агрегации специализаций в массив воспользоваться функцией `array_agg`.

7. (*1 балл*) Проанализировать план выполнения каждого из полученных запросов, найти самый медленный из них. Для этого запроса создать один наиболее оптимальный индекс, который максимально ускорит выполнение этого запроса. Выполнить запрос и проверить, что индекс используется.
8. (*2 балла*) Сделать так, чтобы при обновлении или удалении информации в основной таблице резюме, информация о предыдущих значениях не перетиралась, а в каком то виде сохранялась. Написать запрос, в котором по resume_id выводилась бы история изменения названия резюме в виде: (resume_id, last_change_time, old_title, new_title). Возможно выбрать одну из реализаций, если не можете выбрать, попросите вариант у лектора:
    a. Создать столбец active в таблице резюме. Написать триггер который при любом изменении строки из таблицы (DELETE, UPDATE) пометит изменяемую запись в таблице как active = False, и при UPDATE создаст новую запись. Во всех запросах выше нужно будет учесть флаг active - то есть работать только с активными записями.
    b. При любых изменениях в основной таблице формировать JSONB объект соответствующей строки и записывать его в отдельную таблицу с полями (resume_id, last_change_time, json). В этом варианте не придется менять запросы из пп 3-6, но придется разобраться с тем как работать с JSON в postgres.

   Ссылки на материалы для работы над этим заданием приведены в Приложении 3.

Приложение 1.

```
CREATE TABLE vacancy_body (
   vacancy_body_id serial PRIMARY KEY,
   company_name varchar(150) DEFAULT ''::varchar NOT NULL,
   name varchar(220) DEFAULT ''::varchar NOT NULL,
   text text,
   area_id integer,
   address_id integer,
   work_experience integer DEFAULT 0 NOT NULL,
   compensation_from bigint DEFAULT 0,
   compensation_to bigint DEFAULT 0,
   test_solution_required boolean DEFAULT false NOT NULL,
   work_schedule_type integer DEFAULT 0 NOT NULL,
   employment_type integer DEFAULT 0 NOT NULL,
   compensation_gross boolean,
   driver_license_types varchar(5)[],
   CONSTRAINT vacancy_body_work_employment_type_validate CHECK
((employment_type = ANY (ARRAY[0, 1, 2, 3, 4]))),
   CONSTRAINT vacancy_body_work_schedule_type_validate CHECK
((work_schedule_type = ANY (ARRAY[0, 1, 2, 3, 4])))
);

CREATE TABLE vacancy (
   vacancy_id serial PRIMARY KEY,
   creation_time timestamp NOT NULL,
   expire_time timestamp NOT NULL,
   employer_id integer DEFAULT 0 NOT NULL,
   disabled boolean DEFAULT false NOT NULL,
   visible boolean DEFAULT true NOT NULL,
   vacancy_body_id integer,
   area_id integer
);

CREATE TABLE vacancy_body_specialization (
   vacancy_body_specialization_id integer NOT NULL,
   vacancy_body_id integer DEFAULT 0 NOT NULL,
   specialization_id integer DEFAULT 0 NOT NULL
);
```

Приложение 2.

```
INSERT INTO vacancy (creation_time, expire_time, employer_id, disabled, visible,
area_id)
SELECT
   -- random in last 5 years
   now()-(random() * 365 * 24 * 3600 * 5) * '1 second'::interval AS creation_time,
   now()-(random() * 365 * 24 * 3600 * 5) * '1 second'::interval AS expire_time,
   (random() * 1000000)::int AS employer_id,
   (random() > 0.5) AS disabled,
   (random() > 0.5) AS visible,
   (random() * 1000)::int AS area_id
FROM generate_series(1, 100) AS g(i);

-- Delete invalid records
DELETE FROM vacancy WHERE expire_time <= creation_time;

INSERT INTO vacancy_body(
   company_name, name, text, area_id, address_id, work_experience,
   compensation_from, test_solution_required,
   work_schedule_type, employment_type, compensation_gross
)
SELECT
   (SELECT string_agg(
      substr(
         '
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789',
         (random() * 77)::integer + 1, 1
      ),
      '')
   FROM generate_series(1, 1 + (random() * 250 + i % 10)::integer)) AS company_name,

   (SELECT string_agg(
      substr(
         '
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789',
         (random() * 77)::integer + 1, 1
      ),
      '')
```

```
    FROM generate_series(1, 1 + (random() * 25 + i % 10)::integer)) AS name,

    (SELECT string_agg(
       substr(
          '
abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789',
          (random() * 77)::integer + 1, 1
       ),
       '')
    FROM generate_series(1, 1 + (random() * 50 + i % 10)::integer)) AS text,

    (random() * 1000)::int AS area_id,
    (random() * 50000)::int AS address_id,
    NULL AS work_experience,
    25000 + (random() * 150000)::int AS compensation_from,
    (random() > 0.5) AS test_solution_required,
    floor(random() * 5)::int AS work_schedule_type,
    floor(random() * 6)::int AS employment_type,
    (random() > 0.5) AS compensation_gross
FROM generate_series(1, 100) AS g(i);
```

Приложение 3.

Тут описывается с примерами как создавать и работать с триггерами: https://www.postgresql.org/docs/current/plpgsql-trigger.html

В статье https://www.postgresql.org/docs/9.5/functions-json.html описана работа с JSON объектами в Postgresql, в частности для решения этой задачи понадобится функция `row_to_json()`