Project 1 - Fabian Pascal's Experiment

INSTRUCTIONS

1. This project is an individual project.
2. Submit the "`project1-[studentid].zip`" ZIP file (for example: "`project1-A012345L.zip`") containing only the completed SQL file "`answers.sql`" to the "`Project 1: Submission`" folder under Luminus "`Files > Project 1: Fabian Pascal` by **Friday 7 February 2020, 18:30**.
3. Do not submit other SQL and Django files.
4. Past this deadline and before **Friday 14 February 2020, 18:30**, you may submit to the "`Project 1: Late Submission`" folder (penalties apply).
5. Follow the naming rules and change the SQL file "`answer.sql`" as instructed.

In 1988, Fabian Pascal, a database designer and programmer (and prolific blogger on database issues, see `http://www.dbdebunk.com`, published the article "SQL Redundancy and DBMS Performance" in the journal Database Programming & Design (V1, N12). He compared and discussed the plan and performance of seven equivalent SQL queries with different database management systems. For the experiment he proposed a schema and a synthetic instance on which the seven queries are executed.

At the time, the different systems could or could not execute all the queries and the performances significantly differed among and within individual systems while one would expect the DBMS optimizer to choose the same optimal execution plan for these queries.

In this project, we propose to replay Fabian Pascal's experiment with PostgreSQL.

1. Create a PostgreSQL database called "`cs4221_p1`". The schema consists of a table `employee` and a table `payroll`. The table `employee` records information about employees of a fictitious company. Employees have an employee identifier, a first name and a last name, an address recorded as a street address, a city, a state and a zip code. The table `payroll` records, for each employee, her bonus and salary. The following SQL creates the tables `employee` and `payroll` with the domains in Fabian Pascal's original article.

```
CREATE TABLE employee
(
  empid CHAR(9),
  lname CHAR(15),
  fname CHAR(12),
  address CHAR(20),
  city CHAR(20),
  state CHAR(2),
  ZIP CHAR(5)
);
```

```
CREATE TABLE payroll
(
  empid CHAR(9),
  bonus INTEGER,
  salary INTEGER
);
```

PL/pgSQL is procedural code that can be executed by the PostgreSQL server directly. You may use or modify the following PL/pgSQL function to generate random string of upper case alphabetical characters of a fixed length.

```
CREATE or REPLACE FUNCTION random_string(length INTEGER) RETURNS TEXT AS
$$
DECLARE
  chars TEXT[] := '{A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z}';
  result TEXT := '';
  i INTEGER := 0;
BEGIN
  IF length < 0 then
    RAISE EXCEPTION 'Given_length_cannot_be_less_than_0';
  END IF;
  FOR i IN 1..length
  LOOP
    result := result || chars[1+random()*(array_length(chars, 1)-1)];
  END LOOP;
  RETURN result;
END;
$$ LANGUAGE plpgsql;
```

You may use or modify the following SQL DML code to insert data into the two tables.

```
INSERT INTO employee
SELECT
  TO_CHAR(g, '09999') AS empid,
  random_string(15) AS lname,
  random_string(12) AS fname,
  '500_ORACLE_PARKWAY' AS address,
  'REDWOOD_SHORES' AS city,
  'CA' AS state,
  '94065' AS zip
FROM
  generate_series(0, 9999) g;

INSERT INTO payroll(empid, bonus, salary)
SELECT
  per.empid,
  0 as bonus,
  99170 + ROUND(random() * 1000)*100 AS salary
FROM
  employee per;
```

Create and populate the `employee` and `payroll` tables with the code given above in the "`cs4221_p1`" database. Run the SQL query and PL/pgSQL function above with the Query Tool in pgAdmin 4.

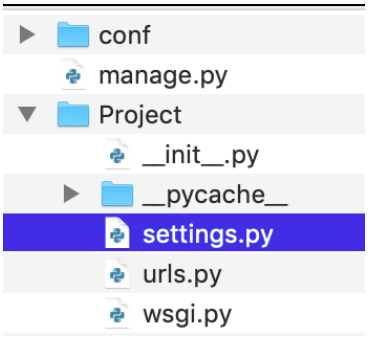
2. We use a Django web application to replay Fabian Pascal's experiment. Download the zip file "`project1.zip`" from Luminus Files "`Project 1: Fabian Pascal`" folder. It contains the `FabianPascal` folder and `answer.sql`.

   Run the `answer.sql` file in the pgAdmin 4.

   We want to find the identifier and the last name of the employees earning a salary of $189170. All of the queries in the `answer.sql` is a simple query with `INNER JOIN`. You will need to change it later to answer the next questions.

   Follow the following instructions to set up your Django web application.

   - Go to your Bitnami-Django stack installation folder. This is the folder that you specified during the installation process.
   - Go to `apps > django > django_projects > Project`. For Windows, you need to find the location of your `django_projects` (by default it should be in `C:\Users\Username\`). You should see two folders named `conf` and `Project` and a file named `manage.py`. Copy the `FabianPascal` folder (extracted from the "`project1.zip`") and paste it here.
   - Go to the `Project` folder and locate `settings.py` file as shown in Figure below.



   - Edit `settings.py` in jEdit. Change line 28 in the file `settings.py` from (the actual value inside the array might be different):

```
ALLOWED_HOSTS = ['127.0.1.1']
```

     to:

```
ALLOWED_HOSTS = ['127.0.1.1','localhost','127.0.0.1']
```

   - Edit `settings.py` in jEdit. Add the `FabianPascal` to the INSTALLED_APPS.
     Change line 33 in that file from:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

     to:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'FabianPascal'
]
```

- Go to your project folder and open the `Project/settings.py` and change the database name by changing the following line:
  From:

```
DATABASES = {
        'default': {
                'ENGINE': 'django.db.backends.postgresql_psycopg2',
                'NAME': 'djangostack',
                'HOST': '/home/cs4221/djangostack-2.2.3-4/postgresql',
                'PORT': '5432',
                'USER': 'bitnami',
                'PASSWORD': '9dfd01e5a1'
        }
}
```

  To:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql_psycopg2',
        'NAME': 'cs4221_p1',
        'HOST': 'localhost',
        'PORT': '5432',
        'USER': 'postgres', ## fill with your username
        'PASSWORD': 'postgres' ## fill with your password
    }
}
```

  Change the username and password accordingly to the password you set during the installation.

- Save and close the `settings.py` file.
- We need to add the FabianPascal URL to the project. Edit `urls.py` file in jEdit. Change the following lines from:

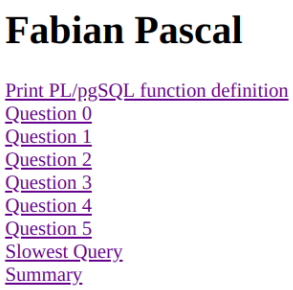
```
urlpatterns = [
    re_path(r'^admin/', admin.site.urls),
    re_path(r'^$', default_urlconf),
]
```

  to:

```
urlpatterns = [
    re_path(r'^admin/', admin.site.urls),
    re_path(r'^$', default_urlconf),
    url(r'^FabianPascal/', include('FabianPascal.urls'))
]
```

- Save and close the `urls.py` file.
- Start the `use_djangostack` terminal. The script (`use_djangostack.bat` or `use_djangostack.bash`) is in your Bitnami-Django stack installation folder.
  For Windows, click the `use_djangostack.bat` file.
  For Mac, open `Terminal` from Launchpad, and execute `cd /Applications/djangostack-2.2.4-0` to go to the bitnami-django stack installation folder (assuming you installed the stack in `/Applications/djangostack-2.2.4-0`). If you installed bitnami-django stack in a different folder change the line accordningly. On the terminal, execute `./use_djangostack`.
  For Linux, open a new terminal, go to the installation folder and type `./use_djangostack`
- Go to the Django `Project` directory on the `use_djangostack` terminal with the command: `cd apps/django/django_projects/Project/`. For Windows, you need to find the location of your `django_projects` (by default it should be in `C:\Users\Username\`) and change all "`/`" to "`\`".

- Go back to the `Project` folder that contains the "`manage.py`" file and run the Django server with the following command:
  `python3 manage.py runserver`
- Open `http://127.0.0.1:8000/FabianPascal/` in your browser. Click on each question to view your answer and results.



- (Optional) If your Apache server is running, you cannot open the `Project1` directly. You need to restart the server in the Bitnami Django app by clicking restart on the Apache webserver. Open your browser and open the following link `http://localhost:8080/Project/FabianPascal/`

In the Django web application, we have created a PL/pgSQL function called `test` that takes an SQL query `Q` and a number N as its parameters and returns the **average execution time**, as reported by `EXPLAIN ANALYZE Q` over N executions of the query `Q`.To see the function definition, go to "`Print PL/pgSQL function definition`" in the menu or look at the source code.The function is called with $N = 100$.

3. (5 points) We want to find the identifier and the last name of the employees earning a salary of $189170.

   One possible SQL query to answer this question is `query_0` as follows.

   ```
   SELECT per.empid, per.lname
   FROM employee per, payroll pay
   WHERE per.empid = pay.empid AND pay.salary = 189170;
   ```

   Write five different but equivalent SQL queries that answer the same question and meet the requirements indicated below, respectively. Unless otherwise indicated, do not use INNER JOIN, OUTER JOIN and NATURAL JOIN. Only use CROSS JOIN represented with comma in the FROM clause. Unless otherwise indicated, do not use subqueries in the FROM clause. Other unnecessary constructs are also penalized.

   1. a simple query with OUTER JOIN and only IS NULL or IS NOT NULL conditions in the WHERE clause,
   2. a nested query with a correlated subquery in the WHERE clause,
   3. a nested query with an uncorrelated subquery in the WHERE clause,
   4. a nested query with an uncorrelated subquery in the FROM clause,
   5. a double-negative nested query with a correlated subquery in the WHERE clause.

   Write your answers in the SQL file "`answer.sql`" by modifying the code of the five corresponding SQL queries, namely `query_1` to `query_5`, respectively.

   Do not remove any of the view definitions `CREATE OR REPLACE VIEW query_<n> (empid, lname) AS`

   Queries that do not produce the correct results will automatically receive zero mark. Beware of minor typos and errors.

   Do not remove the code for `query_0` and use it to check your queries. All queries should have the same results.

   Indicate the measured time for 100 executions of each of the queries (replace `<time>` by the average execution time reported by the Web page) in the indicated space in the file "`answer.sql`".

4. (5 points) Propose a new query (different from the above five queries) that is non-trivially[1] as slow as possible. Do not modify the schema and the data. Measure and indicate its average execution time over 100 executions. Give your answer in the corresponding question in the SQL file "`answer.sql`". This question is marked competitively first on the speed and then the originality of the answer.

   Indicate the measured time for 100 executions of `query_slowest` (replace `<time>` by the average execution time reported by the Web page) in the indicated space in the file "`answer.sql`". Make sure that you are able to run the 100 executions on your computer and report the average execution time. You may be called to demosntrate the execution o fthe query on your machine.

Beyond this project, we recommend that you experiment with different schemas, constraints and indexes and create the environment for the fastest query.

– END OF PAPER –

[1] Trivial constructions include repeated tables and repeated subqueries and other devices as arbitrated at the discretion of the marking team.