Fabian Pascal's Experiment

INSTRUCTIONS

1. This project is an individual project.
2. Submit only one ZIP file, `<student number>.zip` (for example: "`A012345L.zip`"), containing your report (PDF file) and the completed python file "`project1.py`" to the folder "Fabian Pascal Submissions" in Luminus Files by **Friday 1 February at 18:30**.
3. After the deadline and until Friday 8 February at 18:30, you can submit to the folder "Fabian Pascal Late Submissions" in Luminus Workbin (Files) (penalties apply).
4. Make sure to write your **name** and your **student number** on the front page of the report.
5. Structure your report according to the questions and parts. Use clear headings for the questions and parts, e.g. `Question 1` and `Question 1.b`.
6. Make sure to clearly highlight the answer to the question from other discussions and comments.
7. Keep your code as you may be asked to demonstrate that you can reproduce the results that you are reporting.

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 assignment, we propose to replay Fabian Pascal's experiment with PostgreSQL.

We require that you use the PostgreSQL installation (LAPP and pgAdmin) available in the virtual machine that we made available. Please follow the instructions for the virtual machine installation from Luminus Files. In this project, we use Python 2.7 with the SQLAlchemy library to connect to and interact with the PostgreSQL database. Use the Python file "`project1.py`" from Luminus Files to answer the questions.

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 Pascals 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 PLPGSQL 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 DML and PL/pgSQL code given above.

Question 1 [4 marks]

Write 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`. Modify the Python file "`project1.py`" by writing the PL/pgSQL function in the space indicated.

Question 2 [10 marks]

We want to find the identifier and the last name of the employees earning a salary of $199170.

Write the following different but equivalent SQL queries that answer the above query. 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 will also be 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.

Modify the Python file "`project1.py`" by writing the five SQL queries in the space indicated. Tabulate the average execution time over 1000 executions for each of the five queries in the report.

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

Question 3 [6 marks]

Propose a new query (different from the above five queries) that is non-trivially (the marking team reserves the right to define non-trivial on the fly) as slow as possible. Do not modify the schema and the data. Measure and indicate its average execution time over 1000 executions. Give your answer and briefly explain your answer in the report. Give your answer in the corresponding question in the Python file "`project1.py`". This question is marked competitively based on the speed and originality of your answer. You may receive less than the average mark if your query is not as slow or not as interesting as other queries.

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 –