Fabian Pascal's Experiment Report

Question 1

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.

Answer:

Let's show the main ideas in this PL/pgSQL function. First, `EXPLAIN ANALYZE Q` is used to get the execution time of query `Q`. Second, since `EXPAIN ANALYZE` returns multiple results of `Q`, we set the output format of `EXPLAIN ANALYZE` as json in order to extract the execution time of `Q`. Third, we add the execution time to the total execution time in each step. Finally, we return the average execution time `time_total/N`.

The source code is shown below:

```
CREATE or REPLACE FUNCTION test(Q TEXT, N INTEGER) RETURNs FLOAT AS
$$
DECLARE
time_single FLOAT :=0.0;
time_total FLOAT :=0.0;
i INTEGER :=0;
plan_json JSON;
BEGIN
	FOR i IN 1..N
	LOOP
		EXECUTE 'EXPLAIN(ANALYZE TRUE, TIMING, FORMAT JSON)' || Q INTO plan_json;
		SELECT plan_json -> 0 ->'Execution Time' INTO time_single;
		--extract execution to time_single;
		time_total := time_total + time_single;     --add single time
	END LOOP;
	RETURN time_total/N;
END;
$$ LANGUAGE plpgsql;
```

Question 2

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.

Question 2.1

a simple query with `OUTER JOIN` and only `IS NULL` or `IS NOT NULL` conditions in the `WHERE` clause

Answer:

Source code is show below:

```
SELECT e.empid, e.lname
FROM employee e FULL JOIN payroll p
ON e.empid = p.empid         -- outer join employee and payroll on employee's id
WHERE p.salary = 199170
AND e.lname IS NOT NULL;     --search for those whose salary is 199170 and last name is not
empty
```

Test results is shown below:

Execution time (average over 1000): 8.095ms

Question 2.2

a nested query with a correlated subquery in the `WHERE` clause

Answer:

Source code is show below:

```
SELECT e.empid, e.lname
FROM employee e
WHERE EXISTS (
       SELECT *
       FROM payroll p
       WHERE e.empid = p.empid      --correlated condition
       AND p.salary = 199170);
```

Test results:

Execution time (average over 1000 times): 7.607 ms

Question 2.3

a nested query with an uncorrelated subquery in the `WHERE` clause

Answer:

Source code is show below:

```
SELECT e.empid, e.lname
FROM employee e
WHERE e.empid IN (
       SELECT p.empid
       FROM payroll p
       WHERE p.salary = 199170);    --uncorrelated subquery
```

Test result:

Execution time (average over 1000 times): 7.490ms

Question 2.4

a nested query with an uncorrelated subquery in the `FROM` clause

Answer:

Source code is show below:

```
SELECT e.empid, e.lname
FROM employee e, (
               SELECT empid
               FROM payroll p
               WHERE p.salary = 199170      --uncorrelated subquery
               ) temp
WHERE  e.empid = temp.empid;
```

Test result:

Execution time (average over 1000 times): 7.003ms

Question 2.5

a double-negative nested query with a correlated subquery in the `WHERE` clause

Answer:

Source code is show below:

```
SELECT e.empid, e.lname
FROM employee e
WHERE NOT EXISTS (                          --1st negation
               SELECT *
               FROM payroll p
               WHERE e.empid = p.empid      --correlated subquery
               AND p.salary <> 199170       --2nd negation
               );
```

Test result:

Execution time (average over 1000 times): 16.128ms

Question 3

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.

Answer:

To put up a slow query, we try several queries and come to an interesting phenomenon. We put up Query 1 as below.

Query 1

```
SELECT e.empid, e.lname
FROM employee e
WHERE  e.empid NOT IN(
                      SELECT p.empid
                      FROM payroll p
                      WHERE p.salary <> 199170
                      p.empid = e.empid
                      );
```

To get more insights of Query 1, we also test Query 2 for comparison.

Query 2

```
SELECT e.empid, e.lname
FROM employee e
WHERE  e.empid NOT IN(
                      SELECT p.empid
                      FROM payroll p
                      WHERE p.salary <> 199170
                      );
```

Execution times (average over 10 times) are shown as following:

- Query 1: 19357.771ms
- Query 2: 15.8997ms

Combining the above results with that of **Question 2.5** (here, we call it Query 3 for convince), we find that both Query 1 and Query 2 use `NOT IN`, and both Query 1 and Query 3 use correlated subquery. However, when Query 1 use both `NOT IN` and correlated subquery, the execution time grows about 1000 times larger. One possible explanation is that subquery need to find every records satisfying `p.empid = e.empid` and the query also need to find out if `e.empid` is in the every returns of the subquery, which means that the execution cost of Query 1 is about execution cost of Query 2 multiplied by that of query in **Question 2.5**.