Database Tuning – Assignment 2

Query Tuning

Group Name A3

Platzer Hugo, 1421579

Strohmeier Mario, 1422959

October 21, 2016

Experimental Data

Creating Tables and Indexes SQL statements used to create the tables `Employee`, `Student`, and `Techdept`, and the indexes on the tables:

```
CREATE TABLE employee
(
  ssnum integer,
  name character varying,
  manager character varying,
  dept character varying,
  salary integer,
  numfriends integer,
  CONSTRAINT employee_name_key UNIQUE (name),
  CONSTRAINT employee_ssnum_key UNIQUE (ssnum)
)

CREATE INDEX dept_index ON employee (dept)

CREATE TABLE student
(
  ssnum integer,
  name character varying,
  course character varying,
  grade integer,
  CONSTRAINT student_name_key UNIQUE (name),
  CONSTRAINT student_ssnum_key UNIQUE (ssnum)
)

CREATE TABLE techdept
(
  dept character varying,
  manager character varying,
  location character varying,
  CONSTRAINT techdept_dept_key UNIQUE (dept)
)
```

The CONSTRAINTs make Postgres automatically generate an index on the unique attribute.

Populating the Tables ssnum: Ascending int values from the inserting for-loop
name: Random 20 character string with characters a-z, to minimize errors from duplicates
manager: Random letter from A-Z, to have a good chance of duplicates
dept: Random letter from A-J for 10% working in a tech. dept., to have a good chance of duplicates
Letter Z for rest of employees not working in a tech. dept.
salary: Random integer between 1-100000
numfriends: Random integer between 1-100
course: Random 5 character string with character a-z
grade: Random integer between 1-5
location: Random 3 character string with character a-z

Query 1

Intention This query should count the number of employees for which the total number of employees in their department exceeds 1000.

Original Query

```
SELECT COUNT(*) FROM Employee e1 WHERE
    (SELECT COUNT(*) FROM Employee e2 WHERE e2.dept = e1.dept) > 1000;
```

Rewritten Query

```
SELECT dept, COUNT(dept) AS cnt INTO Temp FROM Employee GROUP BY dept;
SELECT COUNT(*) FROM Employee, Temp WHERE
    Employee.dept = Temp.dept AND Temp.cnt > 1000;
```

Evaluation of the Execution Plans Execution plan original query:

```
 Aggregate  (cost=211475925.33..211475925.34 rows=1 width=0)
   ->  Seq Scan on employee e1  (cost=0.00..211475842.00 rows=33333 width=0)
         Filter: ((SubPlan 1) > 1000)
         SubPlan 1
           ->  Aggregate  (cost=2114.73..2114.74 rows=1 width=0)
                 ->  Seq Scan on employee e2  (cost=0.00..2092.00 rows=9091 width=0)
                       Filter: ((dept)::text = (e1.dept)::text)
```

This works like a nested for loop: For every employee e1 go over all Employees e2 and filter these where dept matches. Then count the number of Employees e1 where the number of matches was > 1000.

Execution plan rewritten query:

```
HashAggregate  (cost=2342.00..2342.11 rows=11 width=2)
  Group Key: dept
  ->  Seq Scan on employee  (cost=0.00..1842.00 rows=100000 width=2)

Aggregate  (cost=6281.68..6281.69 rows=1 width=0)
  ->  Merge Join  (cost=41.43..5797.93 rows=193500 width=0)
        Merge Cond: ((employee.dept)::text = (temp.dept)::text)
        ->  Index Only Scan using dept_index on employee  (cost=0.29..2604.29 rows=1000
        ->  Sort  (cost=41.13..42.10 rows=387 width=32)
              Sort Key: temp.dept
              ->  Seq Scan on temp  (cost=0.00..24.50 rows=387 width=32)
                    Filter: (cnt > 1000)
```

In the first step, for every department the number of employees is counted, this (presumably) also uses the index on dept. The result, a small table (11 rows) is stored in Temp.

A merge join is then performed checking for each employee whether its dept is in the temp table, only considering depts with > 1000 employees. The number of rows returned by the merge join is counted.

Runtime It becomes obvious that the rewritten query has to be a lot faster when you need n^2 operations (n is number of employees) for the original nested-loop approach but only around kn (k is number of departments) for the nested-loop approach: create counter for each department, iterate over employees, find their departments counter, increment, then iterate over employees again and compare department to every department in list and if you find a match, check if departments counter is > 1000, if so increment total counter.

	Runtime [sec]
Original query	> 300
Rewritten query	0.2

Query 2

Intent Count how many employees are having a salary higher than the average salary of all employees in their department.

Original Query

```
SELECT COUNT(DISTINCT ssnum) FROM Employee e1 WHERE
    salary > (SELECT AVG(salary) FROM Employee e2 WHERE e2.dept = e1.dept);
```

Rewritten Query

```
SELECT AVG(salary) AS avsalary, dept INTO Temp
FROM Employee GROUP BY Employee.dept;

SELECT COUNT(*) FROM Employee, Temp WHERE
    salary > avsalary and Employee.dept = Temp.dept;
```

Evaluation of the Execution Plans Execution plan original query:

```
  ->  Seq Scan on employee e1  (cost=0.00..115314867.00 rows=33333 width=4)
        Filter: ((salary)::numeric > (SubPlan 1))
        SubPlan 1
          ->  Aggregate  (cost=1153.12..1153.13 rows=1 width=4)
                ->  Bitmap Heap Scan on employee e2  (cost=174.75..1130.39 rows=9091 wi
                      Recheck Cond: ((dept)::text = (e1.dept)::text)
                      ->  Bitmap Index Scan on dept_index  (cost=0.00..172.47 rows=9091
                            Index Cond: ((dept)::text = (e1.dept)::text)
```

A nested loop approach: Iterate over all employees e1, iterate over all employees e2 and check if in same department as e1, if so, add to avg. salary, after inner loop is done, check if e1 salary greater than average, if so increment counter.

Execution plan rewritten query:

```
 HashAggregate  (cost=2342.00..2342.14 rows=11 width=6)
   Group Key: dept
   ->  Seq Scan on employee  (cost=0.00..1842.00 rows=100000 width=6)

 Aggregate  (cost=13553.10..13553.11 rows=1 width=0)
   ->  Merge Join  (cost=60.81..13194.77 rows=143333 width=0)
         Merge Cond: ((employee.dept)::text = (temp.dept)::text)
         Join Filter: ((employee.salary)::numeric > temp.avsalary)
         ->  Index Scan using dept_index on employee  (cost=0.29..4284.25 rows=100000 wi
         ->  Sort  (cost=60.52..62.67 rows=860 width=64)
               Sort Key: temp.dept
               ->  Seq Scan on temp  (cost=0.00..18.60 rows=860 width=64)
```

First phase: Group employees by department (using index), compute avg. salary for each department, store in temp.

Second phase: Iterate over employees, look up their departments avg. salary, increment counter if salary > avg.

Runtime Similar to the last query, a nested loop is avoided.

	Runtime [sec]
Original query	> 300
Rewritten query	0.5

Time in hours per person: **2.71828**