Introduction to Databases **Fall-Winter 2010/11**

Exercises

Lina Lubyte
Marius Kaminskas
Werner Nutt

Indexing and Query Execution (I)

We consider in these exercises queries posed over the employee database that you already know from the SQL exercises. Here is a short reminder of the schema of that database. It contains the following three relations:

- `emp(empno, ename, job, mgr, hiredate, sal, comm, deptno)`
- `dept(deptno, dname, loc)`
- `salgrade(grade, losal, hisal)`.

We assume that no primary keys or uniqueness constraints have been declared.

For these exercises, you will need some resources that you find on the web pages of the course labs:

- a script `comptest.sql` that creates three tables `emp1`, `emp2`, and `emp3` with the same schema as `emp` and three tables `dept1`, `dept2`, and `dept3` with the same schema as `dept`;
- files `emp1.dat`, `emp2.dat`, and `emp3.dat` with dummy data for the corresponding `emp` tables and files files `dept1.dat`, `dept2.dat`, and `dept3.dat` with dummny data for the corresponding `dept` tables;
- Java classes by which you can generate similar dummy data for the `emp` and `dept` relations.

These resources will be needed to run queries on database instances with different sizes. You will be asked to experiment with indexes and to see how they influence query run time.
Before starting the exercises, run the script `comptest.sql` in your group database.
Then download the file `emp3.dat` into your home directory (or a subdirectory) on the university intranet. Note the size of the file, which is more than 60 MB.
With that size, it is better to load it into your database directly from the intranet.
The data in the file can be loaded into the table `emp3` using the PostgreSQL bulk loader. The bulk loader can be called with the `COPY` command and from within

`psql` with the command `\copy`. Syntax and explanations can be found in the documentation at

`http://www.postgresql.org/docs/manuals/.`

Note that you have to be a superuser if you want to use `COPY`, while `\copy` is accessible to everyone. Examples for the usage of `\copy` are shown on the the lab page.

1. Indexes Supporting Selections

Consider the following query:

```
SELECT E.ename, E.sal, E.hiredate
FROM   emp3 E
WHERE  E.ename >= 'A' AND ename <'B' AND
       E.sal <= 1000 AND
       E.hiredate >= '01-OCT-2009';
```

For this and the following queries you have to use the pgAdmin client.

1. Using `Explain` and `ExplainAnalyze`, find out how the query is executed and what is the execution time.
2. You are now allowed to create an index for a single attribute. Analyze the statistics of the table `emp3` in pgAdmin to find out which is probably the best index. Then create it and check that, in fact, it gives the best performance improvement.
3. For the next step, you are allowed to create an index on two attributes. Again, consult the statistics to come up with an hypothesis and then verify it. How much has the performance improved?
4. Now, choose one index (or several indexes) with as many attributes as you wish to increase query performance. Compare whether the results are better and, if so, how much.
5. Check whether and how much clustering reduces the run time of the query. How does the query plan change? Explain!

Hint: If you run the `CLUSTER` command, run `ANALYZYE` immediately after so that the optimizer is aware of the changes.
Consider now the query

```
SELECT E.ename, E.sal, E.hiredate
FROM   emp3 E
WHERE  E.ename >= 'A' AND ename <'B' AND
       E.sal = 1000 AND
       E.hiredate >= '01-OCT-2009';
```

where the comparison in the condition on salary has been changed from a comparison to an equality.
Now, drop the multi-attribute indexes and to undo the clustering (how?).

6. You are allowed to create as many *single-attribute* indexes as you wish. How does the plan created for this query differ from the ones you have seen for the previous query?

7. Change the parameters in the conditions and observe how the execution plans change. What are the reasons for the changes?

Hint: To experiment with the queries, write them into the SQL Editor pane and execute them by marking them and activating the symbols on top of the SQL Editor. Similarly, when experimenting with the creation and dropping of indexes, write `CREATE` and `DROP` statements for the various indexes into the SQL Editor pane and execute them by marking them and activating the *Execute Query* symbol (green triangle).
Comment: The example queries have been formulated with text comparisons instead of `LIKE` conditions because the query optimizer cannot deal with the `LIKE` operator.

2. Indexes Supporting Joins

For this exercise, you are asked to load data into all three sets of employee and department tables. At this stage you should still have the indexes that you created in the previous exercise.
Consider the following join query:

```
SELECT *
FROM   emp NATURAL JOIN dept
```

1. Find out how PostgreSQL executes this query over the three versions of the employee and department tables. (Of course, you have to submit the query with $\texttt{emp}_i$ instead of `emp` and $\texttt{dept}_i$ instead of `dept`.)

2. Create indexes on the attribute `deptno` of each of the `dept` tables. These indexes would be created automatically by PostgreSQL if we had declared `deptno` the primary key. Check whether the execution plans change.

Consider now the query

```
SELECT *
FROM   emp E NATURAL JOIN dept
WHERE  E.ename >= 'A' AND ename <'B' AND
       E.sal = 1000 AND
       E.hiredate >= '01-OCT-2009'
```

which has the same `WHERE` clause as the selection queries of the previous exercise.

3. Again, find out how PostgreSQL executes this query over the three versions of the employee and department tables. Does the plan depend on the size of the tables? Explain your observations.