Lab 15.1: Analysing and Optimising Queries
IN705 Databases Three

Introduction

A lot of execution time can be gained or lost in a database driven application on SQL query execution time. In order to do this effectively we need to ba able to analyze the manner in which our DBMS executes queries. Nearly every DBMS provides some capacity to do this. In this lab we will use the PostgreSQL `EXPLAIN` utility for this.

1 Preparation

We need to load a sample database to use. Carry out the following steps on your EC2 server (note that htis requires you to have completed the previous lab).

```
curl http://www.commandprompt.com/ppbook/booktown.sql > booktown.sql

sudo su postgres

psql < booktown.sql
```

You will get some warnnigs and errors in the output, but they are of no consequence to us. Once the database is loaded, start your Postgres shell with the command

```
psql booktown
```

You will perfomr the rest of this lab inside this shell. One little tip: if you forget to type a semicolon when it's needed, you don't get any obvious cue that something has gone wrong. About the only symptom is that you won't get any output at times when you expect it. If this happens to you, just enter a semicolon, hit enter, suck up the error message and drive on.

2 Analysing queries

To begin, enter the two queries below and compare the results:

```
select title, last_name, first_name from books, authors
  where (books.author_id = authors.id) and last_name='Geisel';

select title, last_name, first_name from books as b
  inner join authors as a on (b.author_id=a.id) where last_name='Geisel';
```

Not surprisingly, they both return the same results. But which one is more efficient? We can use Postgres' `EXPLAIN` tool to find an answer.

```
explain analyze select title, last_name, first_name from books, authors
  where (books.author_id = authors.id) and last_name='Geisel';

explain analyze select title, last_name, first_name from books as b
  inner join authors as a on (b.author_id=a.id) where last_name='Geisel';
```

`EXPLAIN` makes Postgres show the execution plan the will be used to perform the query. Adding `ANALYZE` makes Postgres actually perform the query and show the execution time and costs. Note that, since the query is actually performed, it may not always be wise to use `ANALYZE` if the query changes data.

It's not completely predictable what you will see, but in a test run I saw that the execution plans for these two queries was the same. We can conclude that neither query provides a performance advantage in this case.

3 Analysing tables

PostgreSQL make estimates of execution plan costs based on statistics it maintains about the database. These cost estimates are more accurate, and hence more useful, if the statistics are accurate. If you perform an `EXPLAIN ANALYZE` on a query and the actual statistics differ greatly from the estimates, then it would be useful to update the statistics. We can force Postgres to update its statistics with the commands

```
analyze books;
analyze authors;
```

Then re-run the `EXPLAIN ANALYZE` queries again and see if you notice any difference. You should see that the cost estimates are somewhat improved.

4 Indexes

Our exectution plan shows that Postgres performs "sequence scans", scanning the table rows in order. Perhaps we could add indexes to speed this up.

```
create index books_author_id_idx on books (author_id);
create index author_id_idx on authors (id);
create index author_name_idx on author (last_name);
```

Sure the queries will be faster now. Run the `EXPLAIN ANALYZE` queries again to see.

It turns out that Postgres doesn't use our indexes! This is because the query engine knows that our tables are actually very small, so it would actually take *longer* to use the indexes. One the other hand, if we had many books and authors, then the indexes would be used.

So, our analysis shows that at present the indexes are not helping. Since there is a cost to maintaining them, it makes sense to remove them until we have much more data.