In-class activities: Oct 23, 2017

Today’s activities include some exercises that you need to do on paper or your whiteboard, and some problems that you will work on using our databases. Work through these problems with your 3-person team and check your solutions with the other teams at your table.

Pacing:

In a flipped classroom, the activities give you some structure, but the objective is that everyone can adjust the pace according to their needs. That means that you are in charge of your pace. However, we urge you to maintain focus and efficiency through the activities to ensure that you don't fall behind.

Quiz:

As usual, there will be a short end-of-class quiz. As before, you should indicate your 3-person team on the quiz. We keep track of that based on table and laptop numbers. Note your table number, indicated on the whiteboard and monitor assigned to your table. There are two laptop numbers, one on yellow sticker and one on white sticker. Indicate both if possible, but at least one is necessary. If you cannot find these numbers, please talk to the instructors or TAs before you turn in your quiz.
Step 5: Indexes

Postgres creates some indexes by default (for primary keys and unique constraints). Connect to your IMDB database and use the command \di to see the list of available indexes.

We will now observe query runtimes. For this, we will turn on timing with this command:

\timing on

Now, when you run a query, postgres will report to you the time it took in milliseconds.

Try the following query and observe how long it takes to run:

```sql
SELECT * FROM Actor WHERE lname = 'Bacon';
```

You may note that the time varies somewhat if you execute the query again. This is normal.

We will now take an inside look at how the query is executed, by adding the keyword EXPLAIN before the query. This will not execute the query, but will show you how the database intends to execute this query.

```sql
EXPLAIN SELECT * FROM Actor WHERE lname = 'Bacon';
```

Examine the output of the explain command. What does the database do to find the records the query requests?
Let's now create an index on the actor last name:

```
CREATE INDEX actorlname
ON Actor(lname);
```

Creating the index may take a little while to complete. Once it is done, run the query again:

```
SELECT *
FROM Actor
WHERE lname = 'Bacon';
```

How long did it take to run this time?

Use the explain command again to see what the database is doing this time:

```
EXPLAIN
SELECT *
FROM Actor
WHERE lname = 'Bacon';
```

What changed? Do you understand why the query is now faster?
What if you had created a multi-attribute index on (fname,lname)? Drop the previous index to try this out.

---

**Team-to-table**

Check how long it takes to run the following query:

```
SELECT C.role
FROM Movie M, Casts C
WHERE name = 'Titanic' AND M.id = C.mid;
```

Use EXPLAIN to understand how the query is executed and how you can make it faster. Create the appropriate indexes and try to improve runtime. How do you compare with the other teams at your table?

---

**IMPORTANT NOTE:**

When you are done with this activity, or at any other time you need to, please drop any indexes you created on the IMDB database, with the DROP INDEX command. For example:

```
DROP INDEX actorlname;
```
Step 6: Tuning

You are the database administrator for the UMass Memorial Medical Center (UMassMMC).

The database has the following schema:

Patient(pid, fname, lname, age, street, city, zipcode)
Disease(pid, disease)
Doctor(did, fname, lname, specialty)
Sees(pid, did)
Product(eid, description)
Stock(eid, quantity)
Supplier(sid, name, street, city, zipcode)
Supplies(eid, sid)

The Patient, and Doctor tables contain information about the patients and doctors, respectively. The Disease table denotes which patient has which disease(s). The Sees table says which patient is seeing which doctor(s). The Product table contains information about the medical supplies and equipment that the UMassMMC has. The Stock table maintains the remaining quantity of each product. The Supplier table contains information about the supplier companies, whereas the Supplies table says which products are supplied by which supplier companies.

For each relation, the attributes that form the primary key are underlined.

Additionally:

- Disease.pid is a foreign key referring to Patient.pid
- Sees.pid is a foreign key referring to Patient.pid
- Sees.did is a foreign key referring to Doctor.did
- Stock.eid is a foreign key referring to Product.eid
- Supplies.eid is a foreign key referring to Product.eid
- Supplies.sid is a foreign key referring to Supplier.sid
Loading the data

First, create a new database and connect to it:

```
CREATE DATABASE umassmcc;
\c umassmcc
```

Unzip the data file from this week’s activities and save the two included .sql files in your home directory.

Make sure you are connected to your new database (umassmcc) and load the data from `medData.sql`.

If you have saved the file in your home directory, you should be able to load it within `psql` with the following command:

```
\i medData.sql
```

If your file is elsewhere, you may need to provide the absolute path for `psql` to find it.

If this step is successful, you should get no errors and you should have all the tables loaded in your database.
Tuning

The staff at UMassMMC access the database extensively. There are six kinds of queries they run very often:

--look up a doctor with a given specialty.
SELECT fname, lname
FROM Doctor WHERE specialty = ?;

--look up all doctors for a patient
SELECT D.fname, D.lname
FROM Doctor D, Sees S, Patient P
WHERE D.did = S.did AND S.pid = P.pid AND P.fname = ? AND P.lname = ?;

--look up all patients for a doctor
SELECT P.fname, P.lname
FROM Doctor D, Sees S, Patient P
WHERE D.did = S.did AND S.pid = P.pid AND D.fname = ? AND D.lname = ?;

--check how many patients have a certain disease
SELECT Di.disease, count(*)
FROM Disease Di
WHERE Di.disease = ?
GROUP BY Di.disease;

--count number of patients within an age range
--(used by Marketing team for targeted advertising)
SELECT count(*)
FROM Patient
WHERE age > ? AND age < ?;

--count number of patients living in a specific zipcode
--(used by Marketing team for targeted advertising)
SELECT count(*)
FROM Patient
WHERE zipcode = ?;

You are in charge of administrating the database and optimizing it for performance. The UMassMMC would like these queries to run about an order of magnitude faster than they do now. Can you create the appropriate indexes to do that?

You can test your work by randomly populating each of those queries and running them. You can check if an index you created is used by a query using the EXPLAIN keyword.

For more extensive testing, we provide queries.sql. The file contains a set of 6,000 queries generated from the above templates. You can run the file and time its execution from a terminal window (not inside psql):

```
time psql -U postgresadmin -d umassmcc <queries.sql >/dev/null
```