In-class activities: Oct 16, 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 1: ER diagrams

Your friend is impressed by your skills in 345, and asks you to help her design a database to model doctors' offices across the Pioneer Valley. She started the ER diagram below, which already contains all the entity sets, their attributes and relationships she wants you to consider. You need to finish the diagram and define a number of constraints to ensure that you model the semantics of an office as closely as possible.

Make sure that you do not impose additional constraints not defined by the model.

1. An office may be managed by at most one doctor. A doctor is uniquely identified by their badgeId, and may manage more than one office. Draw the necessary constraints to capture this requirement.

2. An office is identified by its address, and contains one or more exam rooms. An exam room can be identified by its room number and the office that it is in. Capture these constraints by adding to or modifying the diagram where necessary.

3. When a patient visits an office, they have a consultation with a doctor in an examination room. Each patient is uniquely identified by their SSN.
Step 2: ER to relational

Write the SQL DDL commands (CREATE TABLE statements) to convert the ER diagram below to a database schema. Assume all attributes are integers. Pay attention to the constraints you need to declare (primary keys and foreign keys). When you are done, check your solution with the other teams at your table.
Step 3: Understanding key constraints

The goal of this activity is to understand the differences among 3 types of constraints: primary keys, foreign keys, and unique constraints. Start postgres and connect to your practice database: \c practicedb. Create simple tables and add a few tuples as needed to test and answer the following:

For each type of constraint (PRIMARY KEY / FOREIGN KEY /UNIQUE):

1. Can the constraint be defined on a single attribute? Can the constraint be defined on multiple attributes?

2. Can you have multiple constraints of the same type on the same table? (e.g., two primary keys, or two unique constraints)

3. Can two tuples in the table have the same value for the constraint?

4. Can tuples take NULL values for the constraint?

5. Can constraints of the same or different types overlap? For example, can part of a foreign key be defined as unique? Can two foreign keys overlap? Can the same attribute have a foreign key constraint referencing multiple tables?

*** Team-to-table ***

Understanding join cardinality with foreign keys

Create the following two simple tables in your practice database and populate them with a few tuples:

CREATE TABLE T1 (a int PRIMARY KEY, b int);
CREATE TABLE T2 (c int PRIMARY KEY,
    d int NOT NULL REFERENCES T1(a));

Let’s say that T1 has x tuples and T2 has y tuples. How many tuples are there in the result of each of the following queries?

SELECT * FROM T1, T2 WHERE a=d;
SELECT * FROM T1 LEFT OUTER JOIN T2 ON a=d;
SELECT * FROM T1 RIGHT OUTER JOIN T2 ON a=d;

Do you understand how the foreign key impacts the size of the result? Can you reason about the minimum and maximum result size for each of these queries under any random, valid data instance where T1 has x tuples and T2 has y tuples?
Step 4: Constraints

A customer’s database has the following tables:

- Product, listing product ids and names.
- Inventory, listing the quantity for each product.
- Supplier, listing the supplier id, name and address.
- Supplies, indicating which supplier (sid) supplies which product (pid).
- PurchaseOrder, indicating the quantity that is on order for each product.

You have the following CREATE TABLE statements, but they are incomplete.

```
CREATE TABLE Product (pid INT PRIMARY KEY, name VARCHAR(20))
CREATE TABLE Inventory (pid INT PRIMARY KEY, quantity INT)
CREATE TABLE Supplier (sid INT PRIMARY KEY, name VARCHAR(20), address VARCHAR(50))
CREATE TABLE Supplies (sid INT, pid INT, PRIMARY KEY (sid, pid))
CREATE TABLE PurchaseOrder (pid INT, quantity INT)
```

Make the appropriate modifications to the statements above to enforce the following constraints:

1. Indicate all the appropriate foreign keys. For example, ensure that a purchase order refers to a product that exists in the Product table.

2. Ensure that the value of the quantity attribute from table Inventory is always greater than or equal to 0.

3. Whenever someone deletes a tuple in Supplier, any tuple in Supplies that referred to it should also be deleted.
**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.

```
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:

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

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

```sql
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:

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

What changed? Do you understand why the query is now faster?

---

### Team-to-table

Check how long it takes to run the following query:

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

Use EXPLAIN to understand what 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, please drop any indexes you created on the IMDB database, so that another team who may work on this machine next time can have the chance to start fresh. For example:

```sql
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:

```sql
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:

```sql
--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
```