2. Advanced Queries

December 3, 2013

Directions

- Work in groups of 2 students.
- Write-up a short note (max 2 pages long) to sum up your experiments and focus on singular difficulties of the subject.
- Please, add at the end of your report, the following sentence:

  > "We, ⟨student1⟩ and ⟨student2⟩, attest this is our own work and accept the consequences if it is not."

- Deliverables must be packaged into a *tar.gz* or *zip* file containing a single directory having your short report (PDF file) and the SQL scripts to play back the entire story. Name of the directory must satisfy the pattern:

  ```
  hw1_<name of student1>_<name of student2>
  ```

- The archive must be uploaded on the Moodle web site within the due date.

Requirements

PostgreSQL (>=9) database system is required to achieve this assignment. It is an Open Source software available for download at `http://www.postgresql.org/download/`. The abundant documentation and active community could help you installing and setting the system if necessary. The *PgAdmin III* client included into the regular install package is the preferred way for the connection to the database.

Due date

2013-12-15 11:59pm

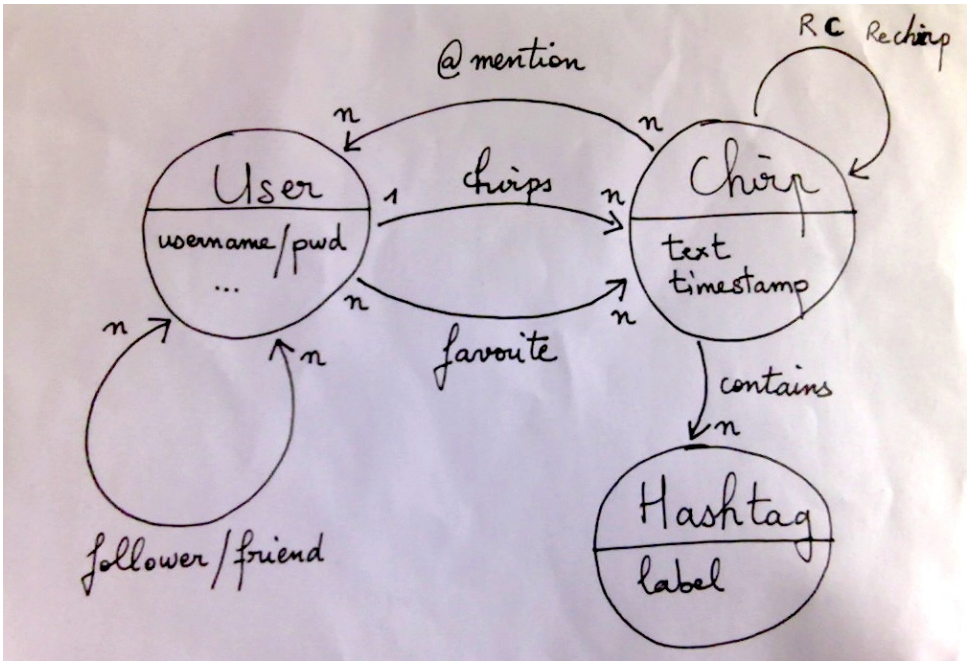

Figure 1: *Birdie* — A (Not So) Simplified Model of Twitter

Introduction

The playground is Twitter, the famous micro-blogging platform. We are trying to develop a restricted copy of Twitter, coined *Birdie*.

The drawing on Figure gives a model for *Birdie*. There are three main entities (user, chirp, hashtag) and several binary relationships among them. A registered user (with username/password) post chirps, i.e. short messages of less than 80 characters each. Chirps may actually be rechirps (`RC <chirp_id>`) of previous messages from registred users. RC is idempotent, i.e. `RC RC <chirp_id>` = `RC <chirp_id>`. Users might follow any other registered users, so-called friends. Friend's chirps are highlighted. Chirps admit hashtags and mentions. Hashtags (`#Nantes`) are basically keywords for chirp categorization purpose. Mentions such like `@me` are references to registered users and allow for instance to answer a previous chirp or to focus on your audience.

The above model translates into the set of following relations:

```
User       (uid, username, password, date_of_registration, geotag, gravatar)
Chirp      (cid, text, timestamp, author)
Hashtag    (hid, label)
Taginchirp (tag, chirp)
Mention    (chirp, user)
Rechirp    (chirp, rechirp)
Follower   (user, friend)
Favorite   (chirp, user)
```

Geotag is a GPS-like geo-localization info. *Gravatar* is a sort of self-portrait materialized by a thumbnail (small picture).

1 Warm-up

As you probably noticed, the database schema slightly changed! Then, you must ugrade the production database to the new schema above.

1. Basically, you are required to follow on the SQL script of Homework 1 and first, adapt database declaration to the updated model. Your main allie in this task is the `ALTER TABLE` SQL statement. Take care of nor loosing any piece of data, neither breaking integrity constraints.

2. Continue to populate the new database with any tuple that helps illustrating the questions below.

2 Advanced SQL Queries

2.1 Window function

SQL provides a sort of extended `GROUP BY` facility where the tuples of a single group are not aggregated (for counting, avg, sum, etc.). It is coined *Window function* and is implemented b.t.w. of the `PARTITION ... ORDER BY` clause.

1. List for each day, the 3 most popular hashtags in descending order. The answer must look like:

```
date        | hashtag   | count | rank
--------------------------------------
2013-12-03   #Nantes     5       1
2013-12-03   #Paris      3       2
2013-12-03   #Lyon       3       2
2013-12-02   #Nantes     6       1
2013-12-02   #Lyon       5       2
2013-12-02   #Paris      2       3
```

Tricks: use the rank() function to build the last column.

2.2 Recursive queries

Common Table Expression (CTE) introduces the `WITH` clause to define temporary tables within SQL `SELECT-FROM-WHERE` statements. The key point is that it allows for recursive definition of tables that obeyes the Datalog stratified semantics.

Write CTE-based SQL statements to answer the following queries.

1. List in descending order the top-10 chirps, i.e. chirps that have been rechirped in the farest community (w.r.t. the friend's network).

2. List in descending order the *betweeness centrality*[1] of every user w.r.t. the friend's relationship.

Tricks: one possible way to storing paths relies on array construction in the recursive table.

3 Server-side programming

3.1 Database triggers

Triggers are database stored procedures that are automatically fired on events. They mainly contribute to database resilience by enforcing integrity constraints that cannot be declaratively defined.

The native procedural language for writting triggers within PostgreSQL is PL/pgSQL.

1. Write PL/pgSQL *triggers* to implement the following global integrity constraints of the *Birdie* project.
   (a) insert a `Mention` tuple on `@<username>` pattern in chirps.
   (b) insert a `Taginchirp` tuple each time a chirp has a `#tag`. If tag is new, then insert in the `Hashtag` table as well.

Tricks: use pattern matching and regexp to detect mentions and hashtags into chirps.

3.2 More on design issues

1. Split content of the `Chirp` table into several chirp-by-month tables such like `chirp-2013-10`, `chirp-2013-11`, `chirp-2013-12`, etc.

Tricks: use the *partitioning* feature of PostgreSQL implemented through the inheritance capability.

2. Allow for right insertion (depending on the timestamp) when adding a new tuple into the `Chirp` table.

Tricks: require triggers.

3. Write a 'conceptual' view for chirp in *Birdie* by the way of the following schema definition:

```
Birdie( username, chirp, timestamp,
        set of #hashtag, set of @mention, rechirp_from )
```

Tricks: use a CREATE VIEW statement and the ARRAY type.

4. Make it updatable!

Tricks: require triggers.

[1] betweeness centrality counts the number of shortest paths that pass through a given node.

Resources & References

1. Chapter *Relational Model & Languages* of Lecture Notes (and SQL counterparts)
2. Chapter *Datalog* of Lecture Notes (and SQL counterparts)
3. Basic SQL tutorials and quick reference guide
   `http://www.w3schools.com/sql/`
4. Homepage of the PostgreSQL 9.3 documentation
   `http://www.postgresql.org/docs/9.3/static/index.html`
5. ALTER TABLE PostgreSQL document page entry
   `http://www.postgresql.org/docs/9.3/static/sql-altertable.html`
6. Window function PostgreSQL document page entry
   `http://www.postgresql.org/docs/9.3/static/tutorial-window.html`
7. CTE PostgreSQL document page entry
   `http://www.postgresql.org/docs/9.3/static/queries-with.html`
8. PL/pgSQL language PostgreSQL document page entry
   `http://www.postgresql.org/docs/9.3/static/plpgsql.html`
9. Pattern matching PostgreSQL document page entry
   `http://www.postgresql.org/docs/9.3/static/functions-matching.html`
10. Trigger PostgreSQL document page entry
    `http://www.postgresql.org/docs/9.3/static/plpgsql-trigger.html`
11. Partitioning PostgreSQL document page entry
    `http://www.postgresql.org/docs/9.3/static/ddl-partitioning.html`
12. CREATE VIEW PostgreSQL document page entry
    `http://www.postgresql.org/docs/9.3/static/sql-createview.html`
13. Array PostgreSQL document page entry
    `http://www.postgresql.org/docs/9.3/static/arrays.html`