Databases for text storage

Jonathan Ronen

New York University

jr4069@nyu.edu

December 1, 2014
Overview

1 Introduction

2 PostgreSQL

3 MongoDB
Why Databases?

- Structured way to store your data
- Accessible, shareable
- Manage growing volumes of data
- You cannot keep all of your data in working memory...
- indexing
Basic issues with databases

- Inserting data
- Schema
- Querying
- Indexing
I’ll show you how to do this in

- PostgreSQL
- MongoDB
PostgreSQL

- Relational DB
- Which means we define tables with columns **and relations**
- Queried using Structured Query Language
- ES-QUE-ELL, or SEQUEL, but not SQUEAL
- opensource, free, very fast, advanced text search capabilities
- Friendly elephant logo
<table>
<thead>
<tr>
<th>id</th>
<th>user_id</th>
<th>rt_of</th>
<th>timestamp</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>539473</td>
<td>416532680</td>
<td>5392...</td>
<td>2014-12-01 17:37:02</td>
<td>RT @XoOverDosed: Enough said #Ferguson <a href="http://t.co/47CCsFNY4q">http://t.co/47CCsFNY4q</a></td>
</tr>
<tr>
<td>539473</td>
<td>2239548626</td>
<td>5394...</td>
<td>2014-12-01 17:37:02</td>
<td>RT @DineshDSouza: When we hear about the police as an &quot;occupying force&quot; in Ferguson, that's the distinctive voice of the anti-colonial idea...</td>
</tr>
<tr>
<td>539473</td>
<td>2899949066</td>
<td></td>
<td>2014-12-01 17:37:02</td>
<td>Eavesdropping on convos around me. Everyone AA are talking Mike and #Ferguson, Caucasian set talking about being inconvenienced by protests.</td>
</tr>
<tr>
<td>539473</td>
<td>2338665151</td>
<td>5394...</td>
<td>2014-12-01 17:37:02</td>
<td>RT @sarahkendzior: &quot;Ferguson is not 'over,' because Ferguson never really 'began.'&quot; My latest from #STL for @Politico <a href="http://t.co/iRNVUMarwE">http://t.co/iRNVUMarwE</a></td>
</tr>
</tbody>
</table>
text

```sql
\d tweets;

Table "public.tweets"

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Modifiers</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>bigint</td>
<td>not null</td>
</tr>
<tr>
<td>user_id</td>
<td>bigint</td>
<td></td>
</tr>
<tr>
<td>retweet_of</td>
<td>bigint</td>
<td></td>
</tr>
<tr>
<td>timestamp</td>
<td>timestamp without time zone</td>
<td></td>
</tr>
<tr>
<td>text</td>
<td>text</td>
<td></td>
</tr>
</tbody>
</table>

Indexes:
- "tweets_pkey" PRIMARY KEY, btree (id)

Foreign-key constraints:
- "tweets_user_id_fkey" FOREIGN KEY (user_id) REFERENCES users(id)
Basics of SQL

texts=> \d users;
    Table "public.users"
       Column | Type    | Modifiers
-------------------------
       id     | bigint  | not null
  screen_name | text    |
description    | text    |
Indexes:
  "users_pkey" PRIMARY KEY, btree (id)
Referenced by:
   TABLE "tweets" CONSTRAINT "tweets_user_id_fkey" FOREIGN KEY (user_id) REFERENCES users(id)
texts=>
SELECT statement
SELECT * FROM tweets WHERE user_id=2170941466;

SELECT statement with time range
SELECT * FROM tweets WHERE timestamp >'2014-12-2';

SELECT statement with LIKE
SELECT * FROM tweets WHERE lower(text) LIKE '%obama%';
Indexing

Imagine searching through a table:

<table>
<thead>
<tr>
<th>id</th>
<th>user_id</th>
<th>timestamp</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2014-11-30 10:23:40</td>
<td>I love the biebsssss!</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2014-11-30 11:33:44</td>
<td>Bieberboy make me a baby!</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2014-11-30 10:23:23</td>
<td>God if biebs dont come i shoot myself!</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2014-11-30 9:12:11</td>
<td>I love bieber so much i have bieber sani</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2014-11-30 12:33:10</td>
<td>RT if you love biebsbs as much ias me!</td>
</tr>
</tbody>
</table>

Find me all tweets since noon.
Imagine searching through a table:

<table>
<thead>
<tr>
<th>id</th>
<th>user_id</th>
<th>timestamp</th>
<th>text</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>2014-11-30 9:12:11</td>
<td>I love bieber so much i have bieber sandwich!</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2014-11-30 10:23:23</td>
<td>God if biebs dont come i shoot myself!</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2014-11-30 10:23:40</td>
<td>I love the biebssssss!</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2014-11-30 11:33:44</td>
<td>Bieberboy make me a baby!</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>2014-11-30 12:33:10</td>
<td>RT if you love biebsbs as much ias me!</td>
</tr>
</tbody>
</table>

Easy! Sort by time!
An index is a sorted copy of a column.

<table>
<thead>
<tr>
<th>timestamp</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014-11-30 9:12:11</td>
<td>4</td>
</tr>
<tr>
<td>2014-11-30 10:23:23</td>
<td>3</td>
</tr>
<tr>
<td>2014-11-30 10:23:40</td>
<td>1</td>
</tr>
<tr>
<td>2014-11-30 11:33:44</td>
<td>2</td>
</tr>
<tr>
<td>2014-11-30 12:33:10</td>
<td>5</td>
</tr>
</tbody>
</table>

(Or really, it’s usually a btree...)

Indexing
Text search in postgres

SELECT statement using PG text search

```sql
SELECT * FROM tweets WHERE to_tsvector('english', text) @@ to_tquery('obama');
```

- to_tsvector
- to_tquery
- (show these in the terminal...)
CREATE INDEX text_idx ON tweets USING gin(to_tsvector('english', text));

SELECT * FROM tweets WHERE to_tsvector('english', text) @@ to_tsquery('obama');
GROUP BY statement

```sql
SELECT user_id, count(*) FROM tweets GROUP BY user_id;
```
- Document store
- noSQL doesn’t mean query language isn’t structured (but it’s different..)
- opensource, free, really fast (sometimes)
Functions that return JSON documents

```json
{
  "created_at": "Wed Aug 13 15:20:46 +0000 2014",
  "lang": "en",
  "retweet_count": 0,
  "text": "Pennsylvania USA Philadelphia \u00bb Mike Brown 545 Mike Brown: St. Louis Police Shoot and Kill Unarmed 18-Year-Old...
  "user": {
    "name": "Jeff",
    "screen_name": "jeffersondol",
    "statuses_count": 207845,
    "description": "#android, #androidgames, #iphone, #ipad, #ipadgames, #app",
    "followers_count": 810,
    "lang": "en",
    "geo_enabled": false,
    "location": "Florida",
  }
}
```
MongoDB is a document database

- MongoDB lets you store these documents directly
- No need to flatten to tabular form!
- Comes with its own query syntax
- Also uses indexing to speed queries

<table>
<thead>
<tr>
<th>SQL</th>
<th>Mongo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Database</td>
</tr>
<tr>
<td>Table</td>
<td>Collection</td>
</tr>
<tr>
<td>Row</td>
<td>Document</td>
</tr>
<tr>
<td>Index</td>
<td>Index</td>
</tr>
</tbody>
</table>
MongoDB Query Syntax

**Regex matching**

```javascript
db.collection.find({'text': /obama/})
```

**Date range**

```javascript
db.collection.find({timestamp: {
  $gt: new Date(2014,10,6)
}})
```
Creating a text index

```javascript
db.tweets.ensureIndex({text: "text" })
```

Using text search

```javascript
db.tweets.findOne({text : {search: "obama"}})
```
Aggregation in MongoDB

Aggregation framework

```javascript
db.tweets.aggregate({
  $group: {
    _id: "$user.screen_name",
    number: { $sum: 1 }
  }
})
```
Some info on the smapp backend:

- MongoDB with index on tweet id, timestamp, random number (for sampling)
- No text index (yet!)
- New!: multiple collection for smappler indexes (smapptoolkit)
The End