Investigating Udiddit's database

By Ivan Babkin

Introduction

Udiddit, a social news aggregation, web content rating, and discussion website, is currently using a risky and unreliable Postgres database schema to store the forum posts, discussions, and votes made by their users about different topics.

The schema allows posts to be created by registered users on certain topics, and can include a URL or a text content. It also allows registered users to cast an upvote (like) or downvote (dislike) for any forum post that has been created. In addition to this, the schema also allows registered users to add comments on posts.

Here is the DDL used to create the existing schema:

```sql
CREATE TABLE bad_posts (
        id SERIAL PRIMARY KEY,
        topic VARCHAR(50),
        username VARCHAR(50),
        title VARCHAR(150),
        url VARCHAR(4000) DEFAULT NULL,
        text_content TEXT DEFAULT NULL,
        upvotes TEXT,
        downvotes TEXT
);

CREATE TABLE bad_comments (
        id SERIAL PRIMARY KEY,
        username VARCHAR(50),
        post_id BIGINT,
        text_content TEXT
);
```

1 Investigating the existing schema

After a thorough investigation of the provided schema as well as the sample data provided on Udacity's website, a number of issues with the current schema were identified.

The following issues should be improved in the new schema:

1. The current schema is not normalised. There should be separate tables for users, topics, posts, comments and votes, where each table contains a unique id. Considering the data storing needs of Udiddit, having only the two tables will simply not be enough to store all the required data.

2. There are no foreign key relationships between the tables in the schema. Essentially under this schema, a comment, which is meant to be dependent on the post, could exist without the post itself.

3. The username should be unique in order to allow other users to uniquely identify any particular user who made a post.

4. In 'bad_posts' table, the 'url' column allows a 'url' of up to 4,000 characters. This is unnecessary as the maximum character limit for a URL is around 2,000. The number of characters allowed for this column should therefore be reduced.

5. In 'bad_posts' table, the 'upvotes' and 'downvotes' columns contain text data. This should be changed as a numeric data type will be the most appropriate to store this information.

6. A further problem with 'upvotes' and 'downvotes' columns is that there are no constraints on user's voting on a particular post allowing the users to submit multiple votes. It is unlikely that this was the goal of Udiddit, and therefore a constraint should be added allowing a user to only vote once.

2 Proposing a new schema

The new schema structure is proposed based on the shortcomings of the initial schema outlined above as well as a number of provided guidelines that are described below.

1. **Guideline #1:** the new schema should at least include a list of features and specifications that Udiddit needs in order to support its website and administrative interface:

   (a) Allow new users to register:

   - Each username has to be unique
   - Usernames can be composed of 25 characters at most
   - Usernames can't be empty
   - User passwords will be ignored in the context of this project

   (b) Allow registered users to create new topics:

   - Topic names have to be unique
   - The topic's name is 30 characters at most
   - The topic's name can't be empty
   - Topics can have an optional description of 500 characters at most

   (c) Allow registered users to create new posts on existing topics:

   - Posts have a required title of 100 characters at most
   - The title of a post can't be empty
   - Posts should contain either a URL or text content, **but not both**
   - If a topic gets deleted, all the posts associated with it should be automatically deleted too
   - If the user who created the post gets deleted, then the post will remain, but it will become dissociated from that user

   (d) Allow registered users to comment on existing posts:

   - A comment's text content can't be empty
   - Contrary to the current linear comments, the new structure should allow comment threads at arbitrary levels
   - If a post gets deleted, all comments associated with it should be automatically deleted too
   - If the user who created the comment gets deleted, then the comment will remain, but it will become dissociated from that user

- If a comment gets deleted, then all its descendants in the thread structure should be automatically deleted too

(e) Make sure that a given user can only vote once on a given post:

- Store the (up/down) value of the vote as the values 1 and -1 respectively
- If the user who cast a vote gets deleted, then all their votes will remain, but will become dissociated from the user
- If a post gets deleted, then all the votes for that post should be automatically deleted too

2. **Guideline #2:** the new schema should support the following list of query objectives that Udiddit needs in order to support its website and administrative interface:

   (a) List all users who haven't logged in during the last year

   (b) List all users who haven't created any post

   (c) Find a user by their username

   (d) List all topics that don't have any posts

   (e) Find a topic by its name

   (f) List the latest 20 posts for a given topic

   (g) List the latest 20 posts made by a given user

   (h) Find all posts that link to a specific URL, for moderation purposes

   (i) List all the top-level comments (those that don't have a parent comment) for a given post

   (j) List all the direct children of a parent comment

   (k) List the latest 20 comments made by a given user

   (l) Compute the score of a post, defined as the difference between the number of upvotes and the number of downvotes

3. **Guideline #3:** the new schema should be *normalized* and should include:

   (a) Named constraints

   (b) Named indexes

4. **Guideline #4:** the new schema should be composed of five tables that should have an auto-incrementing id as their primary key.

The proposed schema outlined using DDL:

```sql
-- "users" table
CREATE TABLE "users" (
  "id" SERIAL PRIMARY KEY,
  "username" VARCHAR(25) NOT NULL,
  "phone_number" VARCHAR(30),
  "time_created" TIMESTAMP,
  "last_login" TIMESTAMP,
  CONSTRAINT "not_empty_username" CHECK (LENGTH(TRIM("username")) > 0)
);

CREATE UNIQUE INDEX "lower_username" ON "users"(
  LOWER(TRIM("username")));
CREATE INDEX "reverse_phone_search" ON "users" (
  REGEXP_REPLACE("phone_number", '[^0-9]+', '', 'g'));

-- "topics" table
CREATE TABLE "topics" (
  "id" SERIAL PRIMARY KEY,
  "name" VARCHAR(30) UNIQUE NOT NULL,
  "description" VARCHAR(500),
  "creator_id" INTEGER REFERENCES "users" ("id") ON DELETE SET NULL,
  "time_created" TIMESTAMP,
  "time_updated" TIMESTAMP,
  CONSTRAINT "not_empty_name" CHECK (
    LENGTH(TRIM("name")) > 0)
);

CREATE INDEX "lower_topic_name" ON "topics" (
  LOWER("name") VARCHAR_PATTERN_OPS);

-- "posts" table
CREATE TABLE "posts" (
  "id" BIGSERIAL PRIMARY KEY,
  "title" VARCHAR(100) NOT NULL,
  "url" VARCHAR(2000) DEFAULT NULL,
  "text_content" TEXT DEFAULT NULL,
  "creator_id" INTEGER REFERENCES "users" ("id") ON DELETE SET NULL,
  "topic_ref" INTEGER REFERENCES "topics" ("id") ON DELETE CASCADE,
  "time_created" TIMESTAMP,
  CONSTRAINT "not_empty_title" CHECK (LENGTH(TRIM("title")) > 0),
  CONSTRAINT "url_or_text" CHECK (
    (NULLIF("url",'') IS NULL OR NULLIF("text_content",'') IS NULL)
    AND NOT
    (NULLIF("url",'') IS NULL AND NULLIF("text_content",'') IS NULL))
);

CREATE INDEX "url_search" ON "posts" ("url" VARCHAR_PATTERN_OPS);

-- "comments" table
CREATE TABLE "comments" (
  "id" BIGSERIAL PRIMARY KEY,
  "text_content" TEXT NOT NULL,
  "creator_id" INTEGER REFERENCES "users" ("id") ON DELETE SET NULL,
  "post_ref" INTEGER REFERENCES "posts" ("id") ON DELETE CASCADE,
  "parent_comment_id" INTEGER REFERENCES "comments" ("id")
    ON DELETE CASCADE,
  "time_created" TIMESTAMP,
  CONSTRAINT "not_empty_text_content" CHECK (
    LENGTH(TRIM("text_content")) > 0)
);

-- "votes" table
CREATE TABLE "votes" (
  "id" BIGSERIAL PRIMARY KEY,
  "user_id" INTEGER REFERENCES "users" ("id") ON DELETE SET NULL,
  "post_ref" INTEGER REFERENCES "posts" ("id") ON DELETE CASCADE,
  "vote" SMALLINT NOT NULL,
  CONSTRAINT "one_vote_per_user" UNIQUE ("user_id", "comment_id"),
  CONSTRAINT "vote_value" CHECK ("vote" = 1 OR "vote" = -1)
);
```

3 Migrating existing data

Now that the new schema is created, the existing data, stored using the original schema, must be migrated to this proposed schema. The data migration is carried out in the project's SQL Workspace on Udacity's website. A set of guidelines described below should be followed during the migration process.

1. Topic descriptions can all be empty.
2. Since the bad_comments table doesn't have the threading feature, all comments can be migrated as top-level comments, i.e. without a parent.
3. Postgres string function **regexp_split_to_table** should be used to unwind the comma-separated votes values into separate rows.
4. Users who only vote or comment, without creating any posts should also be created.
5. The order of migrations matters as posts depend on topics and users, the latter must be migrated first.
6. The data in the SQL Workspace contains thousands of posts and comments. Hence the DML queries may take at least 10-15 seconds to run.

DML code used to migrate the current data stored using bad_posts and bad_comments tables to the proposed database schema:

```sql
-- migrating data to "users" table
INSERT INTO "users" ("username")
SELECT DISTINCT username
FROM bad_comments
UNION
SELECT DISTINCT username
FROM bad_posts
UNION
SELECT DISTINCT REGEXP_SPLIT_TO_TABLE(upvotes, ',')::VARCHAR
FROM bad_posts
UNION
SELECT DISTINCT REGEXP_SPLIT_TO_TABLE(downvotes, ',')::VARCHAR
FROM bad_posts;

-- migrating data to "topics" table
```

```sql
INSERT INTO "topics" ("name")
SELECT DISTINCT topic
FROM bad_posts;

-- migrating data to "posts" table
INSERT INTO "posts" (
  "id", "title", "url", "text_content", "creator_id", "topic_ref")
SELECT bp.id, LEFT(bp.title, 100), bp.url, bp.text_content,
  users.id, topics.id
FROM bad_posts bp
JOIN users
ON bp.username = users.username
JOIN topics
ON bp.topic = topics.name;

-- migrating data to "comments" table
INSERT INTO "comments" (
  "id", "text_content", "creator_id", "post_ref")
SELECT bc.id, bc.text_content, users.id, posts.id
FROM bad_comments bc
JOIN users
ON bc.username = users.username
JOIN posts
ON bc.post_id = posts.id;

-- migrating data to "votes" table
INSERT INTO "votes" ("user_id", "post_ref", "vote")
SELECT users.id, bp_up.id, 1 upvote
FROM (
  SELECT id , REGEXP_SPLIT_TO_TABLE(upvotes, ',') usernames
  FROM bad_posts) bp_up
JOIN users
ON users.username = bp_up.usernames;

INSERT INTO "votes" ("user_id", "post_ref", "vote")
SELECT users.id, bp_dw.id, 1 downvote
FROM (
  SELECT id , REGEXP_SPLIT_TO_TABLE(downvotes, ',') usernames
  FROM bad_posts) bp_dw
JOIN users
ON users.username = bp_dw.usernames;
```