2022

Social News Aggregator

UDIDDIT

Prepared by

MARTA ESPINOSA GARCÍA

DATA ANALYST / DATA SCIENTIST

UDIDDIT

Follow

Udiddit is a social news aggregator, content rating, and discussions website. On Udiddit, registered users are able to link to external content or post their own text content about various topics, ranging from common topics such as photography and food, to more arcane ones such as horse masks or birds with arms. In turn, other users can comment on these posts, and each user is allowed to cast a vote about each post, either in an up (like) or down (dislike) direction.

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 schema:

```
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
);
```

Part I: Investigate the existing schema

As a first step, investigate this schema and some of the sample data in the project's SQL workspace. Then, in your own words, outline three (3) specific things that could be improved about this schema. Don't hesitate to outline more if you want to stand out!

1.From table "bad_posts" we have a syntax error because there are different formats, and the constraints were not used.
·The up_votes should be an INTEGER and not TEXT
·The down_votes should be an INTEGER and not TEXT
2.In the Table "bad_comments" there is a column named "post_id" with datatype "BIGINT" however, the "INT" datatype can be used as the numbers are smaller.
3.The are no foreign keys or indexes.
4.The data is not normalized.

Part II: Create the DDL for your new schema

Having done this initial investigation and assessment, your next goal is to dive deep into the heart of the problem and create a new schema for Udiddit. Your new schema should at least reflect fixes to the shortcomings you pointed to in the previous exercise. To help you create the new schema, a few guidelines are provided to you:

1. Guideline #1: here is a list of features and specifications that Udiddit needs in order to support its website and administrative interface:
    a. Allow new users to register:
        i. Each username has to be unique
        ii. Usernames can be composed of at most 25 characters
        iii. Usernames can't be empty
        iv. We won't worry about user passwords for this project
    b. Allow registered users to create new topics:
        i. Topic names have to be unique.
        ii. The topic's name is at most 30 characters
        iii. The topic's name can't be empty
        iv. Topics can have an optional description of at most 500 characters.
    c. Allow registered users to create new posts on existing topics:
        i. Posts have a required title of at most 100 characters
        ii. The title of a post can't be empty.
        iii. Posts should contain either a URL or a text content, but not both.
        iv. If a topic gets deleted, all the posts associated with it should be automatically deleted too.
        v. 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:
    i. A comment's text content can't be empty.
    ii. Contrary to the current linear comments, the new structure should allow comment threads at arbitrary levels.
    iii. If a post gets deleted, all comments associated with it should be automatically deleted too.
    iv. If the user who created the comment gets deleted, then the comment will remain, but it will become dissociated from that user.
    v. 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:
    i. Hint: you can store the (up/down) value of the vote as the values 1 and -1 respectively.
    ii. If the user who cast a vote gets deleted, then all their votes will remain, but will become dissociated from the user.
    iii. If a post gets deleted, then all the votes for that post should be automatically deleted too.

2. Guideline #2: here is a list of queries that Udiddit needs in order to support its website and administrative interface. Note that you don't need to produce the DQL for those queries: they are only provided to guide the design of your new database schema.
    a. List all users who haven't logged in in 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: you'll need to use normalization, various constraints, as well as indexes in your new database schema. You should use named constraints and indexes to make your schema cleaner.
4. Guideline #4: your new database schema will be composed of five (5) tables that should have an auto-incrementing id as their primary key.

Once you've taken the time to think about your new schema, write the DDL for it in the space provided here:

```
CREATE TABLE users (
user_id SERIAL PRIMARY KEY,
user_name VARCHAR (25) CONSTRAINT user_name_required UNIQUE NOT NULL
  CONSTRAINT user_name_not_empty CHECK(LENGTH(TRIM("user_name")) > 0),
last_login TIMESTAMP
);
CREATE INDEX login_index ON users (last_login);
CREATE INDEX find_user_by_user_name ON users (user_name);

CREATE TABLE topics (
  topic_id SERIAL PRIMARY KEY,
  user_id INTEGER REFERENCES users ON DELETE SET NULL,
  topic_name VARCHAR (30) CONSTRAINT topic_name_required UNIQUE NOT
NULL
    CONSTRAINT topic_name_not_empty CHECK(LENGTH(TRIM("topic_name")) >
0),
  topic_description VARCHAR (500)
);
CREATE INDEX find_topic_name ON topics (topic_name);

CREATE TABLE posts (
  post_id SERIAL PRIMARY KEY,
  title VARCHAR (100) NOT NULL
    CONSTRAINT title_not_empty CHECK(LENGTH(TRIM("title")) > 0),
  url TEXT,
  post_content TEXT,
  topic_id INTEGER REFERENCES topics ON DELETE CASCADE CONSTRAINT
topic_required NOT NULL,
  user_id INTEGER REFERENCES users ON DELETE SET NULL,
  time_stamp_post TIMESTAMP WITH TIME ZONE,
  CONSTRAINT url_or_post_content
  CHECK (url IS NOT NULL AND post_content IS NULL OR
    url IS NULLAND post_content IS NOT NULL)
);
CREATE INDEX latest_posts_topic ON posts (topic_id,time_stamp_post);
CREATE INDEX latest_posts_user ON posts (topic_id,user_id);
CREATE INDEX post_url ON posts (url);
```

```
CREATE TABLE comments (
  id SERIAL PRIMARY KEY,
  text_content TEXT CONSTRAINT text_content_required NOT NULL
    CONSTRAINT text_content_not_empty CHECK(LENGTH(TRIM("text_content")) >
0),
  parent_id INTEGER REFERENCES comments ON DELETE CASCADE CONSTRAINT
parent_required NOT NULL,
  post_id INTEGER REFERENCES posts ON DELETE CASCADE CONSTRAINT
post_required NOT NULL,
  user_id INTEGER REFERENCES users ON DELETE SET NULL,
  time_stamp_comment TIMESTAMP WITH TIME ZONE,
  top_level INTEGER REFERENCES comments ON DELETE CASCADE CONSTRAINT
top_level_required NOT NULL
);

CREATE INDEX top_level_index ON comments (top_level);
CREATE INDEX parent_id ON comments (post_id);
CREATE INDEX comments_by_user ON comments
(user_id,time_stamp_comment);

CREATE TABLE votes (
  user_id INTEGER REFERENCES users ON DELETE SET NULL,
  post_id INTEGER REFERENCES posts ON DELETE CASCADE CONSTRAINT
post_required NOT NULL,
  PRIMARY KEY(user_id, post_id),
  vote INTEGER CONSTRAINT up_down CHECK(vote=1 OR vote=-1)
);
CREATE INDEX post_score ON votes (vote);
```

Part III: Migrate the provided data

Now that your new schema is created, it's time to migrate the data from the provided schema in the project's SQL Workspace to your own schema. This will allow you to review some DML and DQL concepts, as you'll be using INSERT...SELECT queries to do so. Here are a few guidelines to help you in this process:

1. Topic descriptions can all be empty
2. Since the bad_comments table doesn't have the threading feature, you can migrate all comments as top-level comments, i.e. without a parent
3. You can use the Postgres string function regexp_split_to_table to unwind the comma-separated votes values into separate rows
4. Don't forget that some users only vote or comment, and haven't created any posts. You'll have to create those users too.
5. The order of your migrations matter! For example, since posts depend on users and topics, you'll have to migrate the latter first.
6. Tip: You can start by running only SELECTs to fine-tune your queries, and use a LIMIT to avoid large data sets. Once you know you have the correct query, you can then run your full INSERT...SELECT query.
7. NOTE: The data in your SQL Workspace contains thousands of posts and comments. The DML queries may take at least 10-15 seconds to run.

Write the DML to migrate the current data in bad_posts and bad_comments to your new database schema:

```
INSERT INTO users (user_name)
  SELECT bp.username
    FROM bad_posts AS bp
  UNION
  SELECT bc.username
    FROM bad_comments AS bc;

INSERT INTO topics (topic_name, user_id)
  SELECT bp.topic, u.user_id
    FROM bad_posts AS bp
    JOIN users AS u
      ON u.user_name = bp.username
    GROUP BY u.user_id, bp.topic;
```

```
INSERT INTO posts (title, url, post_content, topic_id, user_id)
 SELECT LEFT(bp.title, 100), bp.url, bp.text_content, t.topic_id, u.user_id
  FROM bad_posts AS bp
 JOIN topics AS t
  ON bp.topic = t.topic_name
 JOIN users AS u
  ON bp.username = u.user_name;

INSERT INTO comments(text_content, post_id, user_id)
 SELECT bc.text_content, p.post_id, u.user_id
  FROM bad_comments AS bc
 JOIN bad_posts AS bp
  ON bc.post_id = bp.id
 JOIN posts AS p
  ON p.title = bp.title
 JOIN users AS u
  ON bc.username = u.user_name;
```