Udiddit, a social news aggregator

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) For ease of administration and scalability, user information should be in its own table.
2) Registered users should not be able to create comments on non-existent posts.
3) The upvotes and downvotes would be better represented as numeric values.
4) The text_content on both of the bad_posts and bad_comments tables could be improved by adding a size limitation for the column. This would reduce the ability for someone to abuse the table by inserting extremely large volumes of content, with the intent of simply consuming excessive disk space.
5) The use of indexes, beyond just primary keys, will benefit queries based on commonly used fields, such as post name and usernames.

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:

```
-- file: udiddit_ddl.sql

-- GUIDELINE #4
-- Your new database schema will be composed of five (5) tables that should
have an auto-incrementing id as their primary key.

-------------------------------------------------------------------------------
-------------------------
-- drop the tables, if exist
DROP TABLE IF EXISTS votes CASCADE;
DROP TABLE IF EXISTS comments CASCADE;
DROP TABLE IF EXISTS posts CASCADE;
DROP TABLE IF EXISTS topics CASCADE;
```

```sql
DROP TABLE IF EXISTS users CASCADE;

-------------------------------------------------------------------------------
-------------------------
-- users
CREATE TABLE users (
   id SERIAL PRIMARY KEY,
   username VARCHAR(25) UNIQUE CHECK ( LENGTH("username") > 0 ),
   last_login DATE DEFAULT NULL,
   created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX users_last_login_date_idx ON users (last_login);
CREATE INDEX users_username_idx ON users (username);

-------------------------------------------------------------------------------
-------------------------
-- topics
CREATE TABLE topics (
   id SERIAL PRIMARY KEY,
   name VARCHAR(30) UNIQUE CHECK ( LENGTH("name") > 0 ),
   description VARCHAR(500) DEFAULT NULL,
   -- According to this comment https://knowledge.udacity.com/questions/340724
a user_id is not required.
   created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX topics_topic_name_idx ON topics (name);

-------------------------------------------------------------------------------
-------------------------
-- posts
CREATE TABLE posts (
   id SERIAL PRIMARY KEY,
   title VARCHAR(100) CHECK ( LENGTH("title") > 0 ),
   url VARCHAR(4000) DEFAULT NULL,
   text_content TEXT DEFAULT NULL,
   topic_id INTEGER REFERENCES topics (id) ON DELETE CASCADE,
   user_id INTEGER REFERENCES users (id) ON DELETE SET NULL,
   created_at TIMESTAMP DEFAULT NOW(),
   -- "Posts should contain either a URL or a text content, but not both."
   CONSTRAINT posts_url_text_content_check CHECK (
       -- url and text_content cannot both be null
       NOT ( url IS NULL AND text_content IS NULL) AND
       -- url and text_content cannot both be non-null
       NOT ( LENGTH("url") > 0 AND LENGTH("text_content") > 0)
   )
);

CREATE INDEX posts_title_idx ON posts (title);
CREATE INDEX posts_user_id_idx ON posts (user_id);
CREATE INDEX posts_topic_id_idx ON posts (topic_id);
CREATE INDEX posts_url_idx ON posts (url);
```

```
CREATE INDEX posts_post_id_created_at_idx ON posts (id, created_at);
CREATE INDEX posts_user_id_created_at_idx ON posts (user_id, created_at);

--------------------------------------------------------------------------------
-------------------------
-- comments
CREATE TABLE comments (
   id SERIAL PRIMARY KEY,
   comment_text VARCHAR(100) CHECK ( LENGTH("comment_text") > 0 ),
   post_id INTEGER REFERENCES posts (id) ON DELETE CASCADE,
   user_id INTEGER REFERENCES users (id) ON DELETE SET NULL,
   parent_comment_id INTEGER REFERENCES comments (id) ON DELETE CASCADE
DEFAULT NULL,
   created_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX comments_comment_text_idx ON comments (comment_text);
CREATE INDEX comments_parent_comment_id_idx ON comments (parent_comment_id);
CREATE INDEX comments_user_id_created_at_idx ON comments (user_id,
created_at);

--------------------------------------------------------------------------------
-------------------------
-- votes
CREATE TABLE votes (
   id SERIAL PRIMARY KEY,
   -- only -1 and 1 are acceptable values
   vote SMALLINT CHECK ( vote IN (-1, 1) ),
   post_id INTEGER REFERENCES posts (id) ON DELETE CASCADE,
   user_id INTEGER REFERENCES users (id) ON DELETE SET NULL,
   CONSTRAINT votes_post_id_user_id_key UNIQUE (post_id, user_id),
   created_at TIMESTAMP DEFAULT NOW()
);
```

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:

```
-- file: udiddit_migration.sql

-- the following code will migrate the data from the old tables into the new
tables

-------------------------------------------------------------------------------
-------------------------
-- populate users table
-- get usernames from bad_comments.username
INSERT INTO users (username)
SELECT DISTINCT left(trim(bc.username),25)
FROM bad_comments bc
LEFT OUTER JOIN users u ON left(trim(bc.username),25) = u.username
WHERE trim(u.username) IS NULL;

-- get usernames from bad_posts.username
INSERT INTO users (username)
SELECT DISTINCT left(trim(bp.username),25)
FROM bad_posts bp
LEFT OUTER JOIN users u ON left(trim(bp.username),25) = u.username
WHERE trim(u.username) IS NULL;

-- get usernames from bad_posts.upvotes
INSERT INTO users (username)
WITH split_table AS (
   SELECT DISTINCT left(trim(regexp_split_to_table(bp.upvotes ,',')),25)
username
   FROM bad_posts bp
)
SELECT st.username
FROM split_table st
LEFT OUTER JOIN users u ON st.username = u.username
WHERE trim(u.username) IS NULL;

-- get usernames from bad_posts.downvotes
INSERT INTO users (username)
WITH split_table AS (
   SELECT DISTINCT left(trim(regexp_split_to_table(bp.downvotes ,',')),25)
```

```
username
   FROM bad_posts bp
)
SELECT st.username
FROM split_table st
LEFT OUTER JOIN users u ON st.username = u.username
WHERE trim(u.username) IS NULL;

-------------------------------------------------------------------------------
-------------------------
-- populate topics table
INSERT INTO topics (name)
SELECT DISTINCT left(trim(bp.topic),30)
FROM bad_posts bp
JOIN users u ON left(trim(bp.username),25) = u.username;

-------------------------------------------------------------------------------
-------------------------
-- populate posts table
INSERT INTO posts (id, title, url, text_content, topic_id, user_id)
SELECT bp.id, left(trim(bp.title), 100) title, trim(bp.url) url,
trim(bp.text_content) text_content, t.id topic_id, u.id user_id
FROM bad_posts bp
JOIN users u on left(trim(bp.username),25) = u.username
JOIN topics t on t.name = left(trim(bp.topic),30);

-------------------------------------------------------------------------------
-------------------------
-- populate comments table
INSERT INTO comments (comment_text, post_id, user_id)
SELECT left(trim(bc.text_content), 100) text_content, bc.post_id post_id, u.id
user_id
FROM bad_comments bc
JOIN users u on left(trim(bc.username),25) = u.username;

-------------------------------------------------------------------------------
-------------------------
-- populate votes table
-- upvotes
INSERT INTO votes (vote, post_id, user_id)
WITH upvoter_names AS
        (
          SELECT bp1.id AS post_id,
left(trim(regexp_split_to_table(bp1.upvotes, ',')), 25) AS username
          FROM bad_posts bp1
        )
SELECT 1 AS vote, uvn.post_id AS post_id, u.id AS user_id
FROM users u
JOIN upvoter_names uvn ON uvn.username = u.username;

-- downvotes
INSERT INTO votes (vote, post_id, user_id)
WITH downvoter_names AS
```

```
        (
            SELECT bp1.id AS post_id,
left(trim(regexp_split_to_table(bp1.downvotes, ',')), 25) AS username
            FROM bad_posts bp1
        )
SELECT -1 AS vote, dvn.post_id AS post_id, u.id AS user_id
FROM users u
JOIN downvoter_names dvn ON dvn.username = u.username;
```