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. The current schema is not normalized:
    a. It contains multiple value columns: upvotes and downvotes. It makes it harder to query and quickly access upvote/downvote data.
    b. There are repetitions. For example, in case there is a need to change username, it will take a long time to update the username in both tables. It should be in a separate table with a unique id. Same goes for topics, which are going to be duplicated. They should be identified by a unique primary key.

2. The foreign key data type in the bad_comments table does not match the primary key data type in bad_posts table. It should be INTEGER instead of BIGINT, as the primary key will never be bigger than INTEGER.

3. There are no constraints in the table in case the post is deleted the comments will still exist in bad_comments table, however they should be deleted with the post. The bad_comments table should only contain valid ids from bad_posts table.

4. The following columns: upvotes and downvotes in bad_posts table contain usernames; they are not uniquely constrained and can be duplicated enabling a user upvote and downvote multiple times.

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:

```sql
CREATE TABLE users (
      id SERIAL PRIMARY KEY,
      username VARCHAR(25) NOT NULL CHECK (LENGTH(TRIM(username)) > 0),
      last_logon TIMESTAMP DEFAULT NULL,
      CONSTRAINT unique_usernames UNIQUE (username)
);
CREATE INDEX ON users(last_logon);

CREATE TABLE topics (
      id SERIAL PRIMARY KEY,
      name VARCHAR(30) NOT NULL CHECK (LENGTH(TRIM(name)) > 0),
      description VARCHAR(500) DEFAULT NULL,
      CONSTRAINT unique_names UNIQUE (name)
);

CREATE TABLE posts (
      id SERIAL PRIMARY KEY,
      user_id INTEGER REFERENCES users ON DELETE SET NULL,
      topic_id INTEGER NOT NULL REFERENCES topics ON DELETE CASCADE,
      title VARCHAR(100) NOT NULL CHECK (LENGTH(TRIM(title)) > 0),
      url VARCHAR(4000) DEFAULT NULL,
      content TEXT DEFAULT NULL,
      created_on TIMESTAMP DEFAULT NULL,
      CONSTRAINT only_url_or_content
            CHECK ((url IS NOT NULL AND content IS NULL)
                  OR (content IS NOT NULL AND url IS NULL)
      )
);
CREATE INDEX ON posts(created_on);
CREATE INDEX ON posts(url);

CREATE TABLE comments (
      id SERIAL PRIMARY KEY,
      top_level_comment_id INTEGER DEFAULT NULL,
      post_id INTEGER NOT NULL REFERENCES posts ON DELETE CASCADE,
      user_id INTEGER REFERENCES users ON DELETE SET NULL,
      content TEXT NOT NULL CHECK (LENGTH(TRIM(content)) > 0),
      created_on TIMESTAMP DEFAULT NULL,
      CONSTRAINT top_level_comment
            FOREIGN KEY (top_level_comment_id)
            REFERENCES comments (id)
            ON DELETE CASCADE
);
CREATE INDEX ON comments(created_on);
```

```sql
CREATE TABLE votes (
      id SERIAL PRIMARY KEY,
      post_id INTEGER NOT NULL REFERENCES posts ON DELETE CASCADE,
      user_id INTEGER REFERENCES users ON DELETE SET NULL,
      vote SMALLINT CHECK(vote = 1 OR vote = -1),
      CONSTRAINT one_vote_per_user UNIQUE(post_id, user_id)
);
CREATE INDEX ON votes(post_id);
CREATE INDEX 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:

```
WITH usernames AS (
     SELECT username
     FROM bad_posts
     UNION
     SELECT username
     FROM bad_comments
     UNION
     SELECT regexp_split_to_table(upvotes, ',')
     FROM bad_posts
     UNION
     SELECT regexp_split_to_table(downvotes, ',')
     FROM bad_posts
)
```

```sql
INSERT INTO users(username)
     SELECT DISTINCT username FROM usernames;

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

INSERT INTO posts(user_id, topic_id, title, url, content)
     SELECT u.id, t.id, SUBSTR(bp.title, 1, 100), bp.url,
          bp.text_content
     FROM bad_posts AS bp
     JOIN topics AS t
     ON t.name = bp.topic
     JOIN users AS u
     ON u.username = bp.username;

INSERT INTO comments(post_id, user_id, content)
     SELECT u.id, bc.post_id, bc.text_content
     FROM bad_comments AS bc
     JOIN users AS u
     ON bc.username = u.username;

WITH likes AS (
     SELECT id AS post_id,
          REGEXP_SPLIT_TO_TABLE(upvotes, ',') AS username
     FROM bad_posts
)
INSERT INTO votes(post_id, user_id, vote)
     SELECT l.post_id, u.id, 1
     FROM likes AS l
     JOIN users AS u
     ON u.username = l.username;

WITH dislikes AS (
     SELECT id AS post_id,
          REGEXP_SPLIT_TO_TABLE(downvotes, ',') AS username
     FROM bad_posts
)
INSERT INTO votes(post_id, user_id, vote)
     SELECT l.post_id, u.id, -1
     FROM dislikes AS l
     JOIN users AS u
     ON u.username = l.username;
```