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!

- The existing database schema is in a denormalized structure, indicating that the fields from both tables could be divided into more smaller tables structures.
- Currently, there are no indexes created except for primary keys in both tables which may cause slower data retrieval times during queries.
- The database could benefit from more effective use of constraints. Implementing a UNIQUE constraint would prevent duplicate values in a column.
- The existing schema does not include any columns for dates or times.
- The "upvotes" and "downvotes" columns in the "bad_posts" table contain multiple values within individual rows, indicating a need to normalize these fields to meet the first normal form requirements.

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
-- 1.a Create Table users
CREATE TABLE "users"
(
  "id" SERIAL PRIMARY KEY,
  "username" VARCHAR(25) UNIQUE NOT NULL,
  "login_time" TIMESTAMP,

  CONSTRAINT "ck_username_cant_be_empty" CHECK (Length(Trim("username"))> 0)
);
```

```sql
-- 1.b Create Table topics
CREATE TABLE "topics"
(
  "id" SERIAL PRIMARY KEY,
  "user_id" INT,
  "topic_name" VARCHAR(30) UNIQUE NOT NULL,
  "description" VARCHAR(500),

  CONSTRAINT "ck_topics_cant_be_empty" CHECK (Length(Trim("topic_name"))> 0)
);
```

```sql
-- 1.c Create Table posts
CREATE TABLE "posts"
(
  "id" SERIAL PRIMARY KEY,
  "user_id" INT,
  "topic_id" INT NOT NULL,
  "title" VARCHAR(100) NOT NULL,
  "url" TEXT,
  "post_content" TEXT,
  "post_time" DATE NOT NULL DEFAULT CURRENT_DATE,

  CONSTRAINT "ck_posts_cant_be_empty" CHECK (Length(Trim("title")) > 0),
  CONSTRAINT "ck_text_or_URL"
       CHECK ((("url") IS NULL AND ("post_content") IS NOT NULL)
           OR (("url") IS NOT NULL AND ("post_content") IS NULL)),
  CONSTRAINT "fk_user_id" FOREIGN KEY ("user_id") REFERENCES "users" ("id")
  ON DELETE SET NULL,
  CONSTRAINT "fk_topic_id" FOREIGN KEY ("topic_id") REFERENCES "topics"
  ("id") ON DELETE CASCADE
);
```

```sql
-- 1.d Create Table comments
CREATE TABLE "comments"
(
  "id" SERIAL PRIMARY KEY,
  "user_id" INT,
  "parent_id" INT DEFAULT NULL,
  "topic_id" INT NOT NULL,
  "post_id" INT NOT NULL,
  "comment" TEXT NOT NULL,

  CONSTRAINT "ck_comment_cant_be_empty" CHECK (Length(Trim("comment"))> 0 ),
  CONSTRAINT "parent_comment_id" FOREIGN KEY ("parent_id") REFERENCES
  "comments"("id")ON DELETE CASCADE,
  CONSTRAINT "fk_user_id" FOREIGN KEY ("user_id") REFERENCES "users" ("id")
  ON DELETE SET NULL,
  CONSTRAINT "fk_topic_id" FOREIGN KEY ("topic_id") REFERENCES "topics"
  ("id") ON DELETE CASCADE,
  CONSTRAINT "fk_post_id" FOREIGN KEY ("post_id") REFERENCES "posts" ("id")
  ON DELETE CASCADE
);
```

```sql
-- 1.e Create Table votes
CREATE TABLE "votes"
(
  "id" SERIAL PRIMARY KEY,
  "user_id" INT,
  "post_id" INT NOT NULL,
  "vote" INT NOT NULL,

  CONSTRAINT "fk_user_id" FOREIGN KEY ("user_id") REFERENCES "users" ("id")
  ON DELETE SET NULL,
  CONSTRAINT "fk_post_id" FOREIGN KEY ("post_id") REFERENCES "posts" ("id")
  ON DELETE CASCADE,
  CONSTRAINT "votes_values" CHECK ("vote" = 1 OR "vote" = -1),
  CONSTRAINT "vote_once" UNIQUE ("user_id", "post_id")
);
```

```sql
-- 2.a Create an INDEX to find all users who haven't logged in in the ast
year
CREATE INDEX "idx_login" ON "users" ("username",
"login_time");
```

```sql
-- 2.f Create Index to list the latest posts for a given topic.
CREATE INDEX "idx_latest_post_by_topic" ON "posts" ("topic_id", "url",
"post_content", "post_time");
```

```sql
-- 2.g Create an Index to list the latest posts made by a given user.
CREATE INDEX "idx_latest_post_by_user" ON "posts" ("user_id", "url",
"post_content", "post_time");
```

```sql
-- 2.h Create an Index to find posts that link to specific URL.
CREATE INDEX "idx_posts_url" ON "posts" ("url");
```

```sql
-- 2.j Create an Index to list all the direct children of a parent comment
CREATE INDEX "idx_parent_id" ON "comments" ("parent_id");

-- 2.k Create an Index to list the latest comments made by a given user.
CREATE INDEX "idx_latest_comment_by_user" ON "comments" ("user_id",
"comment");

-- 2.l Create an Index to find score of post votes.
CREATE INDEX "post_score" ON "votes" ("post_id", "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:

```
-- Inserting all usernames data from both initial tables to users table.
INSERT INTO "users" ("username")
SELECT DISTINCT "username"
FROM (SELECT "username" FROM "bad_posts"
      UNION
      SELECT "username" FROM "bad_comments"
      UNION
      SELECT regexp_split_to_table(upvotes, ',')
      FROM "bad_posts" WHERE upvotes IS NOT NULL
      UNION
      SELECT regexp_split_to_table(downvotes, ',')
      FROM "bad_posts" WHERE downvotes IS NOT NULL
      )AS unique_usernames;
```

```sql
-- Inserting all topic & user_id from bad_posts table into topics table
INSERT INTO "topics" ("user_id", "topic_name")
SELECT DISTINCT ON (topic) u.id, bp.topic
FROM "bad_posts" AS bp
JOIN "users" AS u ON u.username = bp.username;

-- Inserting data into table posts
INSERT INTO "posts" ("id", "topic_id", "user_id", "title", "url",
"post_content")
SELECT bp.id, t.id, u.id, LEFT(bp.title,100), bp.url, bp.text_content
FROM "bad_posts" AS bp
JOIN "topics" AS t ON bp.topic = t.topic_name
JOIN "users" AS u ON bp.username = u.username;

-- Inserting data into table comments
INSERT INTO "comments" ("user_id", "topic_id", "post_id", "comment")
SELECT u.id, p.topic_id, p.id, bc.text_content
FROM "bad_comments" AS bc
JOIN "users" AS u ON bc.username = u.username
JOIN "posts" AS p ON bc.post_id = p.id;

-- Inserting upvote data from bad_posts to table votes
WITH bp_upvote AS
    (SELECT id, regexp_split_to_table(upvotes, ',') AS upvoter_username
     FROM bad_posts)
INSERT INTO "votes" ("user_id", "post_id", "vote")
SELECT u.id, bpu.id, 1 AS vote
FROM bp_upvote AS bpu
JOIN "users" AS u ON u.username = bpu.upvoter_username;

-- Inserting downvote data from bad_posts to table votes
WITH bp_downvote AS
    (SELECT id, regexp_split_to_table(downvotes, ',') AS downvoter_username
     FROM bad_posts)
INSERT INTO "votes" ("user_id", "post_id", "vote")
SELECT u.id, bpd.id, -1 AS vote
FROM "bp_downvote" AS bpd
JOIN "users" AS u ON u.username = bpd.downvoter_username;
```