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. bad_posts.upvotes and bad_posts.downvotes are incorrectly set to the Text type, which can not be incremented numerically. In order to remedy this, setting both to the Integer type, then setting each equal to 0, should cover any situation in terms of scale (unless somehow a third of the global population subscribed to the same person and all liked their post at once.

2. The VARCHAR LIMIT on bad_posts.url is currently set to 4000, which could pose issues if a user overloaded the space with characters. A character limit of roughly 2050 characters is a generally agreed upon standard, so I would change the character limit to 2050. (Source: https://mywebshosting.com/what-is-the-maximum-url-length-limit-in-browsers/)

3. Both bad_posts and bad_comments have an id and username column. These should be moved to a separate "Users" table that would contain the username and id for each person. In that same regard, I would also change the data type of id to SERIAL so that each user is unique, and set it as the Primary Key.

4. Both bad_posts.username and bad_posts.title are set up to allow NULL values. A NULL title could be allowed, depending on preference for the site (Facebook doesn't have titles for posts, but Tumblr does). However, a NULL username would allow for those without an account to interact with posts. I would set the username column to NOT NULL to ensure that each person interacting has a username.

5. As it currently stands, the upvotes and downvotes columns of bad_posts contain a long string of usernames. To make this easier to manage, I would separate both into a Votes table and split the list up so that there is only one entry per row. This makes it easier to count votes, as well as keeps the data normalized.

6. Bad_comments.post_id is currently set to BIGINT, which drastically exceeds the number of posts that are available in this database. For a massive social media company like Facebook, this is maybe appropriate, but a majority of the post_ids in this dataset don't break the 10,000 mark. I would thus change it to INTEGER.

7. The text_content columns for both tables do not have a limit, and thus are filled with a seemingly-infinite chain of "Lorem ipsum" filler text. This is good for user freedom, but can easily allow them to exceed the 2GB limit if they wish to do so. Udiddit is meant to be a discussion website, so I would set the VARCHAR LIMIT of text_content columns to a generous 60,000 and comment columns to 8,000, which I found to be roughly the standard post limit for Facebook (Source: https://sproutsocial.com/insights/social-media-character-counter/)

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:

```
                -- Part II: Create the DDL for Your New Schema


-- a. Allow new Users to Register
-- Satisfies Part 1: #3 and #4

CREATE TABLE "users" (
    "user_id" SERIAL PRIMARY KEY,
    "username" VARCHAR(25) UNIQUE NOT NULL,
```

```
    "last_login" TIMESTAMP,
    CONSTRAINT "unique_username" UNIQUE ("username"),
    CONSTRAINT "non_null_username" CHECK (LENGTH(TRIM("username")) > 0 )
);

CREATE INDEX ON "users"("username");

-- b. Allow registered users to create new topics:
-- Satisfies Part 1: #7

CREATE TABLE "topics" (
    "topic_id" SERIAL PRIMARY KEY,
    "topic_name" VARCHAR(30) NOT NULL,
    "topic_description" VARCHAR(500),
    CONSTRAINT "unique_topics" UNIQUE (topic_name),
    CONSTRAINT "non_null_topic" CHECK (LENGTH(TRIM("topic_name")) > 0 )
);

CREATE INDEX ON "topics"("topic_name");
-- c. Allow registered users to create new posts on existing topics
-- Satisfies Part 1: #2, #4, #6, and #7

CREATE TABLE "posts" (
    "post_id" SERIAL PRIMARY KEY,
    "post_title" VARCHAR(100) NOT NULL,
    "posted_on" TIMESTAMP,
    "topic_id" INTEGER REFERENCES "topics"("topic_id") ON DELETE
CASCADE,
    "user_id" INTEGER REFERENCES "users"("user_id") ON DELETE SET NULL,
    "url" VARCHAR(2050),
    "post_text" VARCHAR(60000),
    CONSTRAINT "non_null_topic_id" CHECK (LENGTH(TRIM("topic_id")) > 0 )
    CONSTRAINT "non_null_title" CHECK (LENGTH(TRIM("post_title")) > 0 ),
    CONSTRAINT "url_or_text_content" CHECK (
        ((LENGTH(TRIM("url")) > 0)  AND (LENGTH(TRIM("post_text")) = 0
)) OR
```

```
        ((LENGTH(TRIM("url")) = 0) AND (LENGTH(TRIM("post_text")) > 0 ))
    )
);

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


-- d. Allow registered users to comment on existing posts
-- Satisfies Part 1: #6 and #7

CREATE TABLE "comments" (
    "comment_id" SERIAL PRIMARY KEY,
    "comment_text" VARCHAR(8000) NOT NULL,
    "commented_on" TIMESTAMP,
    "post_id" INTEGER REFERENCES "posts"("post_id") ON DELETE CASCADE,
    "user_id" INTEGER REFERENCES "users"("user_id") ON DELETE SET NULL,
    "comment_id_parent" INTEGER REFERENCES "comments"("comment_id") ON
DELETE CASCADE,
    CONSTRAINT "non_null_text" CHECK (LENGTH(TRIM("comment_text")) > 0
),
    CONSTRAINT "non_null_post_id" CHECK (LENGTH(TRIM("post_id")) > 0 )
    CONSTRAINT "non_null_parent" CHECK
(LENGTH(TRIM("comment_id_parent")) > 0 )
);


-- e. Make sure that a given user can only vote once on a given post
-- Satisfies Part 1: #1 and #5

CREATE TABLE "votes" (
    "vote_id" SERIAL PRIMARY KEY,
    "vote" INTEGER NOT NULL,
    "post_id" INTEGER REFERENCES "posts"("post_id") ON DELETE CASCADE,
    "user_id" INTEGER REFERENCES "users"("user_id") ON DELETE SET NULL,
    CONSTRAINT "non_null_post_id" CHECK (LENGTH(TRIM("post_id")) > 0 )
    CONSTRAINT "upvote_or_downvote" CHECK ("vote" = 1 or "vote" = -1),
```

```
    CONSTRAINT "limit_one_vote" UNIQUE ("user_id", "post_id")
);
```

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:

```
 -- Part III: Migrate the Provided Data

-- Migrate users info from bad_posts & bad_comments into the Users table

INSERT INTO "users"("username")
    SELECT DISTINCT "username"
    FROM "bad_posts"
    UNION
    SELECT DISTINCT "username"
    FROM "bad_comments"
    UNION
    SELECT DISTINCT regexp_split_to_table("upvotes", ',')
    FROM "bad_posts"
```

```
    UNION
    SELECT DISTINCT regexp_split_to_table("downvotes", ',')
    FROM "bad_posts";


-- Migrate Topic Info from bad_posts into the Topics table
INSERT INTO "topics"("topic_name")
    SELECT DISTINCT "topic"
    FROM "bad_posts";


-- Migrate posts info from bad_posts to Posts table
INSERT INTO "posts"(
    "post_title",
    "topic_id",
    "user_id",
    "url",
    "post_text"
)

SELECT SUBSTRING("bad_posts"."title", 1, 100),
    "topics"."topic_id",
    "users"."user_id",
    "bad_posts"."url",
    "bad_posts"."text_content"
FROM "bad_posts"
JOIN "topics"
ON "bad_posts"."topic" = "topics"."topic_name"
JOIN "users"
ON "bad_posts"."username" = "users"."username";


-- Migrate upvotes from bad_posts to Votes table

INSERT INTO "votes" (
    "post_id",
```

```
    "user_id",
    "vote"
)

SELECT "bp_up"."id", "users"."user_id", 1 AS "upvote"
FROM(
    SELECT "id", REGEXP_SPLIT_TO_TABLE("upvotes", ',') AS "upvotes"
    FROM "bad_posts") AS "bp_up"
JOIN "users"
ON "bp_up"."upvotes" = "users"."username";


-- Migrate downvote from bad_posts to Votes table

INSERT INTO "votes" (
    "post_id",
    "user_id",
    "vote"
)

SELECT "bp_down"."id", "users"."user_id", -1 AS "downvote"
FROM (
    SELECT "id", REGEXP_SPLIT_TO_TABLE("downvotes", ',') AS "downvotes"
    FROM "bad_posts") AS "bp_down"
JOIN "users"
ON "bp_down"."downvotes" = "users"."username";


-- Migrate comments from bad_comments to Comments table

INSERT INTO "comments"(
    "post_id",
    "user_id",
    "comment_text"
)
```

```
SELECT "posts"."post_id", "users"."user_id",
"bad_comments"."text_content"
FROM "bad_comments"
JOIN "posts"
ON "bad_comments"."post_id" = "posts"."post_id"
JOIN "users"
ON "bad_comments"."username" = "users"."username";
```