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. Table “bad_comments’ has a column called ‘post_id’, which datatype is BIGINT and it’s a bigger number which is unnecessary in our case. Therefore, I would recommend the datatype ‘INT’ and it has a storage of 4 bytes.
2. Table “bad_posts” there are no constraints on any columns like: topic, username or url, which could have multiples formats without constraints.
3. In both tables there are no indexes. In addition, primary keys in both tables will make querying data slow.
4. Finally, table “bad_posts” the column “downvotes” and “upvotes” have more than one value per row. So, it must be parted (divided) to normal form.

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
-- CREATING USERS DATABASE
-------------------------------------------------------------------------------
CREATE TABLE users (
-- 2.b
  username_id SERIAL PRIMARY KEY,
          -- 1.a.i, 1.a.ii, 1.a.iii, and 1.e
  username VARCHAR(25) CONSTRAINT required_unique_username UNIQUE NOT NULL,
          log_in TIMESTAMP WITH TIME ZONE
);

-- 2.a
CREATE INDEX log_in_index ON users (log_in);
-- 2.c
CREATE INDEX username_index ON users (username VARCHAR_PATTERN_OPS);




-- CREATING TOPICS DATABASE
-------------------------------------------------------------------------------
CREATE TABLE topics (
-- 2.d
          topic_id SERIAL PRIMARY KEY,
          -- 1.b.i, 1.b.ii, and 1.b.iii
          topic VARCHAR(30) CONSTRAINT required_unique_topic UNIQUE NOT NULL,
          -- 1.b.iv
          topic_description VARCHAR(500)
);

-- 2.e
CREATE INDEX topic_index ON topics (topic VARCHAR_PATTERN_OPS);




-- CREATING POSTS DATABASE
-------------------------------------------------------------------------------
CREATE TABLE posts (
          id SERIAL PRIMARY KEY,
          -- 1.c and 1.c.iv
          topic_id INTEGER REFERENCES topics ON DELETE CASCADE,
          -- 1.c and 1.c.v
          user_id INTEGER REFERENCES users ON DELETE SET NULL,
          time_stamp_post TIMESTAMP WITH TIME ZONE,
          -- 1.c.i and 1.c.ii
          title VARCHAR(100) CONSTRAINT required_title NOT NULL,
          url VARCHAR(4000) DEFAULT NULL,
          text_content TEXT DEFAULT NULL,
          -- 1.c.iii
          CONSTRAINT url_or_text
          CHECK(url IS NOT NULL AND text_content IS NULL OR
              url IS NULL AND text_content IS NOT NULL)
);
```

```
-- 2.f
CREATE INDEX latest_posts_per_topic ON posts (topic_id,time_stamp_post);
-- 2.g
CREATE INDEX latest_posts_per_user ON posts (topic_id,user_id);
-- 2.h
CREATE INDEX post_url_moderation ON posts (url VARCHAR_PATTERN_OPS);



-- CREATING COMMENTS DATABASE
-------------------------------------------------------------------------
CREATE TABLE comments (
        id SERIAL PRIMARY KEY,
        -- 1.d and 1.d.iv
        user_id INTEGER REFERENCES users ON DELETE SET NULL,
        -- 1.d.iii
        post_id INTEGER REFERENCES posts ON DELETE CASCADE,
        -- 1.d.i
        text_content TEXT CONSTRAINT required_text_content NOT NULL,
        -- 2.k
        time_stamp_comment TIMESTAMP WITH TIME ZONE,
        -- 1.d.ii and 1.d.v
        level INTEGER REFERENCES comments ON DELETE CASCADE
);

-- 2.i
CREATE INDEX level_index ON comments (level);
-- 2.j
CREATE INDEX parent_id_index ON comments (post_id);
-- 2.k
CREATE INDEX comments_by_user ON comments (user_id,time_stamp_comment);



------ CREATING VOTES DATABASE
-------------------------------------------------------------------------
CREATE TABLE votes (
        PRIMARY KEY (user_id, post_id),
        -- 1.e.iii
        user_id INTEGER REFERENCES users ON DELETE SET NULL,
        -- 1.e.ii
        post_id INTEGER REFERENCES posts ON DELETE CASCADE,
        -- 1.e.i
        vote INTEGER CONSTRAINT up_down_vote CHECK(vote=1 OR vote=-1)
);

-- 2.l
CREATE INDEX score_post 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:

```
-- INSERTING DATA INTO USER DATABASE (USERS WHO HAVE MADE POSTS)

INSERT INTO users (username)
        SELECT DISTINCT username
        FROM bad_posts;


--USERS WHO HAVE MADE ONLY COMMENTED

INSERT INTO users (username)
        SELECT DISTINCT bad_com.username
        FROM bad_comments bad_com
        LEFT JOIN users u
        ON bad_com.username = u.username
        WHERE u.username IS NULL;
```

```sql
--USERS WHO HAVE ONLY UP VOTED

INSERT INTO users (username)
        WITH table1 AS (SELECT REGEXP_SPLIT_TO_TABLE(upvotes, ',')
                        AS upvote
                        FROM bad_posts)
        SELECT DISTINCT upvote
        FROM table1
        LEFT JOIN users u
        ON table1.upvote = u.username
        WHERE u.username IS NULL;

--USERS WHO HAVE ONLY DOWN VOTED
INSERT INTO users (username)
        WITH table1 AS (SELECT REGEXP_SPLIT_TO_TABLE(downvotes, ',')
                        AS downvote
                        FROM bad_posts)
        SELECT downvote
        FROM table1
        LEFT JOIN users u
        ON table1.downvote = u.username
        WHERE u.username IS NULL;


-- INSERTING DATA INTO TOPICS DATABASE (ALL TOPIC NAMES)

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


-- INSERTING DATA INTO POSTS DATABASE (ALL POST DATA INTO POSTS
DATABASE)

INSERT INTO posts (id, topic_id, user_id, title, url, text_content)
        SELECT bad_post.id, t.topic_id, u.username_id,
                LEFT(bad_post.title,100), bad_post.url, bad_post.text_content
        FROM bad_posts bad_post
        JOIN topics t
        ON bad_post.topic = t.topic
        JOIN users u
        ON bad_post.username = u.username;
```

```sql
-- INSERTING DATA INTO COMMENTS DATABASE

INSERT INTO comments (user_id, post_id, text_content, level)
        SELECT u.username_id, p.id, bad_com.text_content,
             ROW_NUMBER() OVER(PARTITION BY p.id)
        FROM bad_comments bad_com
        JOIN posts p
        ON bad_com.post_id = p.id
        JOIN users u
        ON bad_com.username = u.username;


-- INSERTING DATA INTO VOTES DATABASE

INSERT INTO votes (user_id, post_id, vote)
        WITH table1 AS (SELECT id, REGEXP_SPLIT_TO_TABLE(downvotes, ',')
                        AS downvote
                        FROM bad_posts)
        SELECT u.username_id, table1.id, -1 AS vote
        FROM table1
        JOIN users u
        ON u.username = table1.downvote;

INSERT INTO votes (user_id, post_id, vote)
        WITH table1 AS (SELECT id, REGEXP_SPLIT_TO_TABLE(upvotes, ',')
                        AS upvote
                        FROM bad_posts)
        SELECT u.username_id, table1.id, 1 AS vote
        FROM table1
        JOIN users u
        ON u.username = table1.upvote;
```