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 create_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. Data Type: Upvotes and Downvotes should not be text. They should be INT data type.
2. Bad_posts Table: This is not normalized. There should be another table- user_info containing Users_id which, with username, will form primary keys. This User_id can be included in Bad_posts table referenced as a foreign key.
3. Bad_comments table: Similarly, instead of username, user_id from user_info table should be included as foreign key.

Part II: Create the DDL for your new schema

Once you've taken the time to think about your new schema, write the DDL for it in the space provided here:

Indexes are automatically created for primary key constraints and unique constraints.

```
CREATE TABLE user_info (
id SERIAL PRIMARY KEY,
username VARCHAR (25) UNIQUE NOT NULL,
last_login TIMESTAMP,
CONSTRAINT len CHECK (LENGTH(TRIM(username)) > 0)
);

CREATE TABLE topics(
id SERIAL PRIMARY KEY,
name VARCHAR(30) UNIQUE NOT NULL,
description VARCHAR(500),
CONSTRAINT len2 CHECK (LENGTH(TRIM(name)) > 0)
);



CREATE TABLE create_posts (
id SERIAL PRIMARY KEY,
title VARCHAR(100)  UNIQUE NOT NULL,
created_on TIMESTAMP,
url VARCHAR(400),
text_content TEXT,
topic_id INTEGER REFERENCES topics(id) ON DELETE CASCADE NOT NULL,
user_id INTEGER REFERENCES user_info(id) ON DELETE SET NULL,
CONSTRAINT len3 CHECK (LENGTH(TRIM(title)) > 0),
CONSTRAINT len4 CHECK ((LENGTH(TRIM(url)) > 0 AND LENGTH(TRIM(text_content)) = 0) OR
(LENGTH(TRIM(url)) = 0 AND LENGTH(TRIM(text_content)) > 0))
);

CREATE INDEX index1 ON create_posts (url VARCHAR_PATTERN_OPS);
CREATE INDEX index2 ON create_posts( text_content TEXT_PATTERN_OPS);

CREATE TABLE comments(
id SERIAL PRIMARY KEY,
created_on TIMESTAMP,
post_id INTEGER REFERENCES create_posts(id) ON DELETE CASCADE,
```

```
user_id INTEGER REFERENCES user_info(id) ON DELETE SET NULL,
text_content TEXT NOT NULL,
parent_comment_id INTEGER REFERENCES comments(id) ON DELETE CASCADE,
CONSTRAINT len5 CHECK(LENGTH(TRIM(text_content)) > 0)
 );

CREATE INDEX index3 ON comments (text_content TEXT_PATTERN_OPS);


CREATE TABLE votes(
id SERIAL PRIMARY KEY,
user_id INTEGER REFERENCES user_info(id) ON DELETE SET NULL,
post_id INTEGER,
vote SMALLINT NOT NULL,
CONSTRAINT len6 CHECK(vote = 1 OR vote = -1),
CONSTRAINT one_vote_per_user UNIQUE (user_id, post_id)
);
```

Part III: Migrate the provided data

Write the DML to migrate the current data in bad_posts and bad_comments to your new database schema:

```
INSERT INTO user_info(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;
```

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

INSERT INTO create_posts(user_id,  topic_id,  title,  url,  text_content)
SELECT  user_info.id,  topics.id,  LEFT(bad_posts.title, 100),bad_posts.url, bad_posts.text_content
FROM bad_posts
JOIN user_info
ON bad_posts.username = user_info.username
JOIN topics
ON bad_posts.topic = topics.name;

INSERT INTO comments( post_id,  user_id,  text_content)
SELECT   create_posts.id,   user_info.id,   bad_comments.text_content
FROM bad_comments
JOIN user_info
ON bad_comments.username = user_info.username
JOIN create_posts
ON create_posts.id = bad_comments.post_id;

With cte as
(SELECT id, REGEXP_SPLIT_TO_TABLE (upvotes, ',')  AS upvote_users FROM bad_posts)
INSERT INTO votes (post_id,  user_id,  vote)
SELECT cte.id, user_info.id,1 AS vote_up
FROM cte
JOIN user_info
ON user_info.username=cte.upvote_users;


With cte2 AS
(SELECT id, REGEXP_SPLIT_TO_TABLE(downvotes, ',') AS downvote_users FROM bad_posts)
INSERT INTO votes ( post_id, user_id, vote)
SELECT cte2.id, user_info.id, -1 AS vote_down
FROM cte2
JOIN user_info
ON user_info.username=cte2.downvote_users;
```