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: Investigating the existing schema

Things that could be improved about this schema:

1. The varchar limit of topic should be increased for topic. 50 is very small.
2. The varchar limit for title should be increased. 150 seems small.
3. Post_id is referenced from bad_posts table. So, identifying post_id as a foreign key in bad _comments table will be like this:
   Post_id BIGINT REFERENCES “bad_posts”
4. The data type of post_id in bad_comments table has to be int, to keep the consistency with its reference key.
5. There are more than one values in upvotes and downvotes in the bad_posts table. That goes against the normalization rule. We have to create 2 more tables: one for upvoting and another is for downvoting. Both tables will contain user id and post id.

Part II: Creating the DDL for new schema

Having done this initial investigation and assessment, my next goal is to dive deep into the heart of the problem and to create a new schema for Udiddit. My new schema will reflect fixes to the shortcomings that was pointed to in part 1.

1. **Users table:**

```
CREATE TABLE "users" (
     user_id SERIAL,
     username VARCHAR(25) NOT NULL CHECK (LENGTH(TRIM("username"))> 0),
     last_login_date TIMESTAMP,
     CONSTRAINT "user_id_pk" PRIMARY KEY ("user_id"),
     CONSTRAINT "unique_username" UNIQUE ("username")
     );
```

2. **Topics table:**

```
CREATE TABLE "topics" (
     topic_id SERIAL,
     topic_name VARCHAR(30) NOT NULL CHECK (LENGTH(TRIM("topic_name"))
     > 0),
     topic_text VARCHAR (500),
     CONSTRAINT "topic_id_pk" PRIMARY KEY ("topic_id"),
     CONSTRAINT "unique_topicname" UNIQUE ("topic_name")
     );
```

3. **Posts table:**

```
CREATE TABLE "posts" (
      post_id SERIAL,
      post_title VARCHAR(100) NOT NULL CHECK (LENGTH(TRIM("post_title"))
      > 0),
      post_url VARCHAR,
      post_text TEXT,
      post_topic_id INTEGER REFERENCES "topics" ("topic_id") ON DELETE
      CASCADE,
      user_id INTEGER REFERENCES "users" ("user_id") ON DELETE SET NULL,
      posting_date TIMESTAMP,
      CONSTRAINT "post_id_pk" PRIMARY KEY ("post_id"),
      CONSTRAINT "post_content_type" CHECK ( "post_url" IS NULL AND
      "post_text" IS NOT NULL OR "post_url" IS NOT NULL AND "post_text"
      is null)
      );
```

Indexes related to posts table:

```
      CREATE INDEX "post_url_search" ON "posts" ("post_url");
```

4. **Comments table:**

```
CREATE TABLE "comments" (
      comment_id SERIAL,
      parent_comment_id INTEGER REFERENCES comments(comment_id) ON
      DELETE CASCADE,
      comment_text TEXT NOT NULL CHECK (LENGTH(TRIM("comment_text"))>
      0),
      comment_date TIMESTAMP,
      post_id INTEGER REFERENCES "posts" ("post_id") ON DELETE CASCADE,
      user_id INTEGER REFERENCES "users" ("user_id") ON DELETE SET NULL,
      CONSTRAINT "comment_id_pk" PRIMARY KEY ("comment_id")
      );
```

5. **User vote table:**

```
CREATE TABLE "user_vote" (
      vote_id SERIAL,
      user_id INTEGER REFERENCES "users" ("user_id") ON DELETE SET NULL,
      post_id INTEGER REFERENCES "posts" ("post_id") ON DELETE CASCADE,
      vote_value SMALLINT CHECK (vote_value IN (-1,1)),
      CONSTRAINT "user_vote_pk" PRIMARY KEY
      ("vote_id","user_id","post_id"),
      CONSTRAINT one_time_vote UNIQUE (vote_id, user_id, post_id)
      );
```

Part III: Migrate the provided data

Now that our new schema is created, it is time to migrate the data to our own schema:

1. **Getting all the usernames:**
   a) **From bad_posts table:**

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

   b) **From bad_comments table:**

```
INSERT INTO "users" ("username")
SELECT DISTINCT "username" FROM "bad_comments"
WHERE "username" NOT IN (SELECT DISTINCT "username"
                         FROM "bad_posts");
```

   c) **Unique users who never created a post nor commented on any post, but they did give upvotes and downvotes:**

```
INSERT INTO "users" ("username")
WITH cte AS (
select "id"
       ,regexp_split_to_table("upvotes",',') as names
FROM "bad_posts"
UNION
SELECT "id"
,regexp_split_to_table("downvotes",',') as names
FROM "bad_posts")
SELECT DISTINCT names
FROM cte
WHERE "names" NOT IN (Select DISTINCT "bad_posts"."username"
                      FROM "bad_posts")
AND "names" NOT IN (SELECT DISTINCT "bad_comments"."username"
                    FROM "bad_comments");
```

2. **Getting all the topics:**

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

3. **Getting columns for posts table:**

```
INSERT INTO "posts"
("post_id","post_title","post_url","post_text","user_id",
"post_topic_id")
SELECT id
      ,LEFT("title",100)
      ,"url"
      ,"text_content"
      ,u."user_id"
      ,t."topic_id"
FROM "bad_posts" bs
INNER JOIN "users" u ON u. "username" = bs. "username"
INNER JOIN "topics" t ON t. "topic_name" = LEFT (bs. "topic",30);
```

4. **Getting columns for comments table:**

```
INSERT INTO "comments" ("comment_text", "post_id", "user_id")
SELECT "text_content", "post_id", "user_id"
FROM "bad_comments" bc
INNER JOIN "users" u ON u."username" = bc."username";
```

5. **Getting columns for user_vote table:**

```
INSERT INTO "user_vote" ("user_id", "post_id", "vote_value")
SELECT "user_id"
      ,"id"
      ,1 AS "vote_value"
FROM (SELECT "id"
            ,regexp_split_to_table("upvotes",',') AS like_names
      FROM "bad_posts") like_vote
      INNER JOIN "users" u ON u."username" = like_vote.like_names
UNION
SELECT "user_id"
      ,"id"
      ,-1 AS "vote_value"
FROM (SELECT "id"
            ,regexp_split_to_table("downvotes",',') AS dislike_names
FROM "bad_posts") dislike_vote
INNER JOIN "users" u ON u."username" = dislike_vote.dislike_names;
```

Conclusion

Our Data migration is complete. So, we have successfully solved the bad data issue of Udiddit, a social news aggregator. In this process, we initially created the DDL schemas with proper constraints and indexes. After that, we successfully completed the task by migrating all the data to this newly created tables.