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; |