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

A key issue is the use of multi-value columns in the bad\_posts table, where the upvotes and downvotes are stored as comma-separated values. This practice deviates from the principles of database normalization and can complicate data retrieval and manipulation.

Additionally, there is a notable lack of foreign key constraints, particularly in the bad\_posts table where the post\_id column is un-referenced. This absence could lead to data integrity issues, such as orphan records. Another concern is the repetition of the username column in both the bad\_posts and bad\_comments tables, which suggests a potential redundancy and inefficiency in the schema design.

### Proposed Schema Improvements

There are several aspects that could be improved:

* Data Normalization:
  + **Multi-values Column:** comma-separated values - Table: bad\_posts, Columns: [upvotes, downvotes]
  + Separate Votes information from bad\_posts table
* Data Integrity:
  + **Lack of Foreign Key Constraints:** un-referenced column - Table bad\_posts, Column: post\_id
  + **Column Repetition:** username column repeated in both tables bad\_posts and bad\_comments
* Data Consistency:
  + Add CHECK constraint to ensure valid URLs
  + Add created\_at column to track the records timeline
* Query Optimization:
  + Add appropriate indexes to speed up query search and retrieval

## Part II: Create the DDL for your new schema

Having done this initial investigation and assessment, before diving deep into the heart of the problem and create a new schema for Udiddit. A few guidelines are provided to consider any modelling or querying concerns. [[Schema Guidelines](https://github.com/Sasa94s/udiddit/blob/main/GUIDELINES.md)]

|  |
| --- |
| -- Users Table CREATE TABLE users (  id SERIAL PRIMARY KEY,  username VARCHAR(25) UNIQUE NOT NULL,  last\_login TIMESTAMP,  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,   CONSTRAINT "nonempty\_username" CHECK ( COALESCE(TRIM(username), '') <> '' ) );  CREATE INDEX idx\_users\_last\_login ON users (last\_login); CREATE INDEX idx\_users\_username ON users (username);  -- Topics Table CREATE TABLE topics (  id SERIAL PRIMARY KEY,  name VARCHAR(30) UNIQUE NOT NULL,  description VARCHAR(500) DEFAULT NULL,  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP );  CREATE UNIQUE INDEX unique\_idx\_topics\_name ON topics (TRIM(name)); CREATE UNIQUE INDEX idx\_topics\_name ON topics (LOWER(name) VARCHAR\_PATTERN\_OPS);  -- Posts Table CREATE TABLE posts (  id SERIAL PRIMARY KEY,  user\_id INT REFERENCES users (id) ON DELETE SET NULL,  topic\_id INT REFERENCES topics (id) ON DELETE CASCADE,  title VARCHAR(150) NOT NULL,  url VARCHAR(4000) DEFAULT NULL,  text\_content TEXT DEFAULT NULL,  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,   CONSTRAINT "text\_or\_url\_only" CHECK ( (url IS NOT NULL OR text\_content IS NOT NULL) AND  (url IS NULL OR text\_content IS NULL)) );  CREATE INDEX idx\_posts\_user ON posts (user\_id); CREATE INDEX idx\_posts\_topic ON posts (topic\_id);  -- Comments Table CREATE TABLE comments (  id SERIAL PRIMARY KEY,  user\_id INT REFERENCES users (id) ON DELETE SET NULL,  post\_id INT REFERENCES posts (id) ON DELETE CASCADE,  parent\_comment\_id INT REFERENCES comments (id) ON DELETE CASCADE,  text\_content TEXT NOT NULL,  created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP );  CREATE INDEX idx\_comments\_post ON comments (post\_id); CREATE INDEX idx\_comments\_user ON comments (user\_id);  -- Votes Table  CREATE TABLE votes (  user\_id INT REFERENCES users (id) ON DELETE SET NULL,  post\_id INT REFERENCES posts (id) ON DELETE CASCADE,  vote\_value SMALLINT CHECK (vote\_value IN (-1, 1)),  PRIMARY KEY (user\_id, post\_id) ); |

## Part III: Migrate the provided data

|  |
| --- |
| -- Users INSERT INTO users (username) SELECT DISTINCT bp.username FROM bad\_posts AS bp WHERE COALESCE(TRIM(bp.username), '') <> '' UNION SELECT DISTINCT bc.username FROM bad\_comments AS bc WHERE COALESCE(TRIM(bc.username), '') <> '' UNION SELECT username FROM (SELECT DISTINCT UNNEST(string\_to\_array(upvotes, ',') || string\_to\_array(downvotes, ',')) AS username  FROM bad\_posts) AS v WHERE COALESCE(TRIM(v.username), '') <> '';  -- Topics INSERT INTO topics (name) SELECT DISTINCT topic FROM bad\_posts WHERE COALESCE(TRIM(topic), '') <> '';  -- Posts INSERT INTO posts (id, user\_id, topic\_id, title, url, text\_content) SELECT bp.id, u.id, t.id, bp.title, bp.url, bp.text\_content FROM bad\_posts bp  LEFT JOIN users u ON bp.username = u.username  LEFT JOIN topics t ON bp.topic = t.name;  -- Comments INSERT INTO comments (id, user\_id, post\_id, text\_content) SELECT bc.id, u.id, p.id, bc.text\_content FROM bad\_comments bc  LEFT JOIN users u ON bc.username = u.username  LEFT JOIN posts p ON bc.post\_id = p.id;  -- Votes INSERT INTO votes (user\_id, post\_id, vote\_value) SELECT v.user\_id, v.post\_id, SUM(v.vote\_value) AS vote\_value FROM (SELECT DISTINCT u.id AS user\_id, downvotes.id AS post\_id, -1 AS vote\_value  FROM (SELECT id, regexp\_split\_to\_table(downvotes, ',') AS username  FROM bad\_posts) AS downvotes  LEFT JOIN users u ON downvotes.username = u.username  WHERE COALESCE(TRIM(u.username), '') <> ''  UNION ALL  SELECT DISTINCT u.id AS user\_id, upvotes.id AS post\_id, 1 AS vote\_value  FROM (SELECT id, regexp\_split\_to\_table(upvotes, ',') AS username  FROM bad\_posts) AS upvotes  LEFT JOIN users u ON upvotes.username = u.username  WHERE COALESCE(TRIM(u.username), '') <> '') AS v GROUP BY user\_id, post\_id HAVING SUM(vote\_value) <> 0; |

### Validation

The summary of the tables shows the deprecated bad\_posts table has identical rows count of 50,000 as the newly migrated posts table, while the deprecated bad\_comments has identical rows count of 100,000 as the newly migrated comments table.

#### Deprecated Tables Summary

Here is the summary of the rows count per table for the deprecated tables:

| **Table** | **Rows Count** |
| --- | --- |
| Bad Posts | 50000 |
| Bad Comments | 100000 |

#### Migrated Tables Summary

Here is the summary of the rows count per table for the newly migrated tables:

| **Table** | **Rows Count** |
| --- | --- |
| Users | 11077 |
| Topics | 89 |
| Posts | 50000 |
| Comments | 100000 |
| Votes | 499710 |