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]

```
-- 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,

    CONSTRAINT "nonempty_topic_name" CHECK ( COALESCE(TRIM(name), '') <> ''
)
);
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 "nonempty_post_title" CHECK ( COALESCE(TRIM(title), '') <> ''
),
    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,

    CONSTRAINT "nonempty_comment_text_content" CHECK (
COALESCE(TRIM(text_content), '') <> '' )
);
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