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

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!

|  |
| --- |
| The current schema of the bad\_posts and bad\_comments tables can be improved in the following ways:  **1.Normalization of Votes**:   1. **Problem**: The upvotes and downvotes columns in bad\_posts are stored as comma-separated strings, which is inefficient and difficult to query. This design violates the principles of database normalization, specifically the first normal form, which requires atomicity in columns. 2. **Improvement**: Create a separate votes table to store each vote as a separate record. This will make it easier to perform operations such as counting votes or determining if a specific user has voted.   **2.Introduction of a Users Table**:   * **Problem**: There is no users table to ensure unique usernames and manage user data effectively. Usernames are repeated in both bad\_posts and bad\_comments, leading to redundancy and potential data inconsistencies. * **Improvement**: Create a users table to store user-related data, ensuring usernames are unique and making it easier to manage user-specific information.   **3.Support for Threaded Comments**:   * **Problem**: The current bad\_comments table structure does not support threaded comments, limiting the ability to create discussion threads or replies within comments. * **Improvement**: Add a parent\_id column to the comments table, allowing comments to reference parent comments and support nested threads. |

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

Having done this initial investigation and assessment, your next goal is to dive deep into the heart of the problem and create a new schema for Udiddit. Your new schema should at least reflect fixes to the shortcomings you pointed to in the previous exercise. To help you create the new schema, a few guidelines are provided to you:

1. Guideline #1: here is a list of features and specifications that Udiddit needs in order to support its website and administrative interface:
   1. Allow new users to register:
      1. Each username has to be unique
      2. Usernames can be composed of at most 25 characters
      3. Usernames can’t be empty
      4. We won’t worry about user passwords for this project
   2. Allow registered users to create new topics:
      1. Topic names have to be unique.
      2. The topic’s name is at most 30 characters
      3. The topic’s name can’t be empty
      4. Topics can have an optional description of at most 500 characters.
   3. Allow registered users to create new posts on existing topics:
      1. Posts have a required title of at most 100 characters
      2. The title of a post can’t be empty.
      3. Posts should contain either a URL or a text content, **but not both**.
      4. If a topic gets deleted, all the posts associated with it should be automatically deleted too.
      5. If the user who created the post gets deleted, then the post will remain, but it will become dissociated from that user.
   4. Allow registered users to comment on existing posts:
      1. A comment’s text content can’t be empty.
      2. Contrary to the current linear comments, the new structure should allow comment threads at arbitrary levels.
      3. If a post gets deleted, all comments associated with it should be automatically deleted too.
      4. If the user who created the comment gets deleted, then the comment will remain, but it will become dissociated from that user.
      5. If a comment gets deleted, then all its descendants in the thread structure should be automatically deleted too.
   5. Make sure that a given user can only vote once on a given post:
      1. Hint: you can store the (up/down) value of the vote as the values 1 and -1 respectively.
      2. If the user who cast a vote gets deleted, then all their votes will remain, but will become dissociated from the user.
      3. If a post gets deleted, then all the votes for that post should be automatically deleted too.
2. Guideline #2: here is a list of queries that Udiddit needs in order to support its website and administrative interface. Note that you don’t need to produce the DQL for those queries: they are only provided to guide the design of your new database schema.
   1. List all users who haven’t logged in in the last year.
   2. List all users who haven’t created any post.
   3. Find a user by their username.
   4. List all topics that don’t have any posts.
   5. Find a topic by its name.
   6. List the latest 20 posts for a given topic.
   7. List the latest 20 posts made by a given user.
   8. Find all posts that link to a specific URL, for moderation purposes.
   9. List all the top-level comments (those that don’t have a parent comment) for a given post.
   10. List all the direct children of a parent comment.
   11. List the latest 20 comments made by a given user.
   12. Compute the score of a post, defined as the difference between the number of upvotes and the number of downvotes
3. Guideline #3: you’ll need to use normalization, various constraints, as well as indexes in your new database schema. You should use named constraints and indexes to make your schema cleaner.
4. Guideline #4: your new database schema will be composed of five (5) tables that should have an auto-incrementing id as their primary key.

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

|  |
| --- |
| -- Drop tables if they already exist to ensure clean creation  DROP TABLE IF EXISTS votes;  DROP TABLE IF EXISTS comments;  DROP TABLE IF EXISTS posts;  DROP TABLE IF EXISTS topics;  DROP TABLE IF EXISTS users;  -- 1. Guideline #1: Create Tables with Features and Specifications  -- a. Allow new users to register  CREATE TABLE users (      id SERIAL PRIMARY KEY,  -- Auto-incrementing primary key      username VARCHAR(25) UNIQUE NOT NULL,  -- Unique and non-empty username with a maximum of 25 characters      created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP -- Timestamp for user creation  );  -- b. Allow registered users to create new topics  CREATE TABLE topics (      id SERIAL PRIMARY KEY,  -- Auto-incrementing primary key      name VARCHAR(30) UNIQUE NOT NULL,  -- Unique and non-empty topic name with a maximum of 30 characters      description VARCHAR(500) DEFAULT NULL,  -- Optional description with a maximum of 500 characters      created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP -- Timestamp for topic creation  );  -- c. Allow registered users to create new posts on existing topics  CREATE TABLE posts (      id SERIAL PRIMARY KEY,  -- Auto-incrementing primary key      topic\_id INT NOT NULL,  -- Foreign key to topics      user\_id INT DEFAULT NULL,  -- Foreign key to users, nullable to dissociate if user is deleted      title VARCHAR(100) NOT NULL,  -- Required non-empty title with a maximum of 100 characters      url VARCHAR(4000) DEFAULT NULL,  -- URL or text content must be provided, not both      text\_content TEXT DEFAULT NULL,      created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, -- Timestamp for post creation      FOREIGN KEY (topic\_id) REFERENCES topics(id) ON DELETE CASCADE, -- Cascade delete if topic is deleted      FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE SET NULL,  -- Set NULL if user is deleted      CHECK ((url IS NOT NULL AND text\_content IS NULL) OR (url IS NULL AND text\_content IS NOT NULL)), -- Ensure only one of url or text\_content is present      CONSTRAINT chk\_post\_content CHECK (url IS NOT NULL OR text\_content IS NOT NULL) -- Ensure at least one content is present  );  -- d. Allow registered users to comment on existing posts  CREATE TABLE comments (      id SERIAL PRIMARY KEY,  -- Auto-incrementing primary key      post\_id INT NOT NULL,  -- Foreign key to posts      user\_id INT DEFAULT NULL,  -- Foreign key to users, nullable to dissociate if user is deleted      parent\_id INT DEFAULT NULL,  -- Self-referential foreign key for comment threads      text\_content TEXT NOT NULL,  -- Required non-empty text content      created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, -- Timestamp for comment creation      FOREIGN KEY (post\_id) REFERENCES posts(id) ON DELETE CASCADE, -- Cascade delete if post is deleted      FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE SET NULL, -- Set NULL if user is deleted      FOREIGN KEY (parent\_id) REFERENCES comments(id) ON DELETE CASCADE -- Cascade delete if parent comment is deleted  );  -- e. Make sure that a given user can only vote once on a given post  CREATE TABLE votes (      id SERIAL PRIMARY KEY,  -- Auto-incrementing primary key      post\_id INT NOT NULL,  -- Foreign key to posts      user\_id INT DEFAULT NULL,  -- Foreign key to users, nullable to dissociate if user is deleted      value INT NOT NULL CHECK (value IN (1, -1)),  -- Vote value must be 1 (upvote) or -1 (downvote)      created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, -- Timestamp for vote creation      FOREIGN KEY (post\_id) REFERENCES posts(id) ON DELETE CASCADE, -- Cascade delete if post is deleted      FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE SET NULL, -- Set NULL if user is deleted      UNIQUE (post\_id, user\_id),  -- Ensure a user can only vote once per post      CONSTRAINT chk\_vote\_value CHECK (value IN (1, -1)) -- Ensure valid vote values  );  -- 2. Guideline #2: Queries (Conceptual)  -- a. List all users who haven’t logged in in the last year.  -- b. List all users who haven’t created any post.  -- c. Find a user by their username.  -- d. List all topics that don’t have any posts.  -- e. Find a topic by its name.  -- f. List the latest 20 posts for a given topic.  -- g. List the latest 20 posts made by a given user.  -- h. Find all posts that link to a specific URL, for moderation purposes.  -- i. List all the top-level comments (those that don’t have a parent comment) for a given post.  -- j. List all the direct children of a parent comment.  -- k. List the latest 20 comments made by a given user.  -- l. Compute the score of a post, defined as the difference between the number of upvotes and the number of downvotes.  -- 3. Guideline #3: Normalization, Constraints, and Indexes  -- Indexes for optimization  CREATE INDEX idx\_user\_username ON users(username); -- Index for quick username lookup  CREATE INDEX idx\_topic\_name ON topics(name); -- Index for quick topic name lookup  CREATE INDEX idx\_post\_title ON posts(title); -- Index for quick post title lookup  CREATE INDEX idx\_comment\_hierarchy ON comments(parent\_id); -- Index for quick comment hierarchy traversal  CREATE INDEX idx\_vote\_post\_user ON votes(post\_id, user\_id); -- Index for quick vote lookup by post and user  -- Add comments to explain constraints  COMMENT ON CONSTRAINT chk\_post\_content ON posts IS 'Ensures posts have either a URL or text content, but not both.';  COMMENT ON CONSTRAINT chk\_vote\_value ON votes IS 'Ensures votes have a value of either 1 (upvote) or -1 (downvote).';  -- 4. Guideline #4: Use of Auto-incrementing Primary Key  -- The primary keys for all tables are set as SERIAL, which is an auto-incrementing integer in PostgreSQL. |

## Part III: Migrate the provided data

Now that your new schema is created, it’s time to migrate the data from the provided schema in the project’s SQL Workspace to your own schema. This will allow you to review some DML and DQL concepts, as you’ll be using INSERT...SELECT queries to do so. Here are a few guidelines to help you in this process:

1. Topic descriptions can all be empty
2. Since the bad\_comments table doesn’t have the threading feature, you can migrate all comments as top-level comments, i.e. without a parent
3. You can use the Postgres string function **regexp\_split\_to\_table** to unwind the comma-separated votes values into separate rows
4. Don’t forget that some users only vote or comment, and haven’t created any posts. You’ll have to create those users too.
5. The order of your migrations matter! For example, since posts depend on users and topics, you’ll have to migrate the latter first.
6. Tip: You can start by running only SELECTs to fine-tune your queries, and use a LIMIT to avoid large data sets. Once you know you have the correct query, you can then run your full INSERT...SELECT query.
7. **NOTE**: The data in your SQL Workspace contains thousands of posts and comments. The DML queries may take at least 10-15 seconds to run.

Write the DML to migrate the current data in bad\_posts and bad\_comments to your new database schema:

|  |
| --- |
| -- Migrate users from bad\_posts and bad\_comments  INSERT INTO users (username)  SELECT DISTINCT username  FROM (      SELECT username FROM bad\_posts      UNION      SELECT username FROM bad\_comments      UNION      SELECT unnest(string\_to\_array(upvotes, ',')) AS username FROM bad\_posts      UNION      SELECT unnest(string\_to\_array(downvotes, ',')) AS username FROM bad\_posts  ) AS all\_users  WHERE username IS NOT NULL;  -- Migrate topics  INSERT INTO topics (name)  SELECT DISTINCT topic  FROM bad\_posts;  -- Migrate posts  INSERT INTO posts (topic\_id, user\_id, title, url, text\_content)  SELECT      t.id AS topic\_id,      u.id AS user\_id,      b.title,      b.url,      b.text\_content  FROM      bad\_posts b  JOIN      topics t ON b.topic = t.name  LEFT JOIN      users u ON b.username = u.username;  -- Migrate comments  INSERT INTO comments (post\_id, user\_id, text\_content)  SELECT      p.id AS post\_id,      u.id AS user\_id,      c.text\_content  FROM      bad\_comments c  JOIN      posts p ON c.post\_id = p.id  LEFT JOIN      users u ON c.username = u.username;  -- Migrate votes (upvotes)  INSERT INTO votes (post\_id, user\_id, value)  SELECT      p.id AS post\_id,      u.id AS user\_id,      1 AS value  FROM (      SELECT          title,          unnest(string\_to\_array(upvotes, ',')) AS username      FROM          bad\_posts  ) AS votes  JOIN      posts p ON votes.title = p.title  LEFT JOIN      users u ON votes.username = u.username;  -- Migrate votes (downvotes)  INSERT INTO votes (post\_id, user\_id, value)  SELECT      p.id AS post\_id,      u.id AS user\_id,      -1 AS value  FROM (      SELECT          title,          unnest(string\_to\_array(downvotes, ',')) AS username      FROM          bad\_posts  ) AS votes  JOIN      posts p ON votes.title = p.title  LEFT JOIN      users u ON votes.username = u.username; |

**Explanation**

**User Migration**: We extract all unique usernames from bad\_posts and bad\_comments and include those from upvotes and downvotes, using unnest to split the comma-separated values.

**Topic Migration**: Extract unique topic names from bad\_posts.

**Post Migration**: Associate each post with a topic and user using the titles and usernames.

**Comment Migration**: Insert comments with post and user associations. All comments are migrated as top-level comments due to the lack of threading in the old schema.

**Vote Migration**: Use unnest on the upvotes and downvotes to split the values into individual rows, and then associate them with posts and users. We use a value of 1 for upvotes and -1 for downvotes.