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!

1) Ideally, generating a key that is really unique is combination of a primary key and a not null constraint; here 'topic' and 'username' could be set to "unique NOT NULL"; otherwise new users and topics can be generated without any content; this could end up in having multiple NULL users in the database;
2) Upvotes and downvotes are numeric counts; there is no need to choose TEXT as data type; in a TEXT field an unlimited number of characters can be stored; this can consume storage for nothing;
3) Attribute 'Post_id' (table 'bad_posts') could be linked to bad_posts table as a foreign key or will at least be used to join both tables together. However, data types do not match. **SERIAL** is an auto-incremented **integer** column that takes 4 bytes (int) while **BIGSERIAL** is an auto-incremented **bigint** column taking 8 bytes. The id of 'bad_posts' should be converted to **BIGSERIAL** to match 'bad_comments' table's **BIGINT**.
4) 'Text_content' is currently set to a TEXT data type. This data type allows unlimited characters input and can thus be misused. Choosing varchar with a limited amount of characters as maximum allowed input would be better, pose less risk, reduce lifecycle costs plus providing more control and scalability over the database itself.
5) 'URL' allows input of 4000 characters. This can lead to inputs of more than 2083 characters which is e.g. the current limit of URL length for Internet Explorer. Thus, it would be better to allow only the smallest length of URL of all browsers (here: 2083 characters). So this URL can be opened anywhere and there will be no browser issues like that some URLs can only be opened with special browsers.

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:
    a. Allow new users to register:
        i. Each username has to be unique
        ii. Usernames can be composed of at most 25 characters
        iii. Usernames can't be empty
        iv. We won't worry about user passwords for this project
    b. Allow registered users to create new topics:
        i. Topic names have to be unique.
        ii. The topic's name is at most 30 characters
        iii. The topic's name can't be empty
        iv. Topics can have an optional description of at most 500 characters.
    c. Allow registered users to create new posts on existing topics:
        i. Posts have a required title of at most 100 characters
        ii. The title of a post can't be empty.
        iii. Posts should contain either a URL or a text content, **but not both**.
        iv. If a topic gets deleted, all the posts associated with it should be automatically deleted too.
        v. If the user who created the post gets deleted, then the post will remain, but it will become dissociated from that user.
    d. Allow registered users to comment on existing posts:
        i. A comment's text content can't be empty.
        ii. Contrary to the current linear comments, the new structure should allow comment threads at arbitrary levels.
        iii. If a post gets deleted, all comments associated with it should be automatically deleted too.
        iv. If the user who created the comment gets deleted, then the comment will remain, but it will become dissociated from that user.
        v. If a comment gets deleted, then all its descendants in the thread structure should be automatically deleted too.
    e. Make sure that a given user can only vote once on a given post:
        i. Hint: you can store the (up/down) value of the vote as the values 1 and -1 respectively.
        ii. If the user who cast a vote gets deleted, then all their votes will remain, but will become dissociated from the user.

iii. If a post gets deleted, then all the votes for that post should be automatically deleted too.

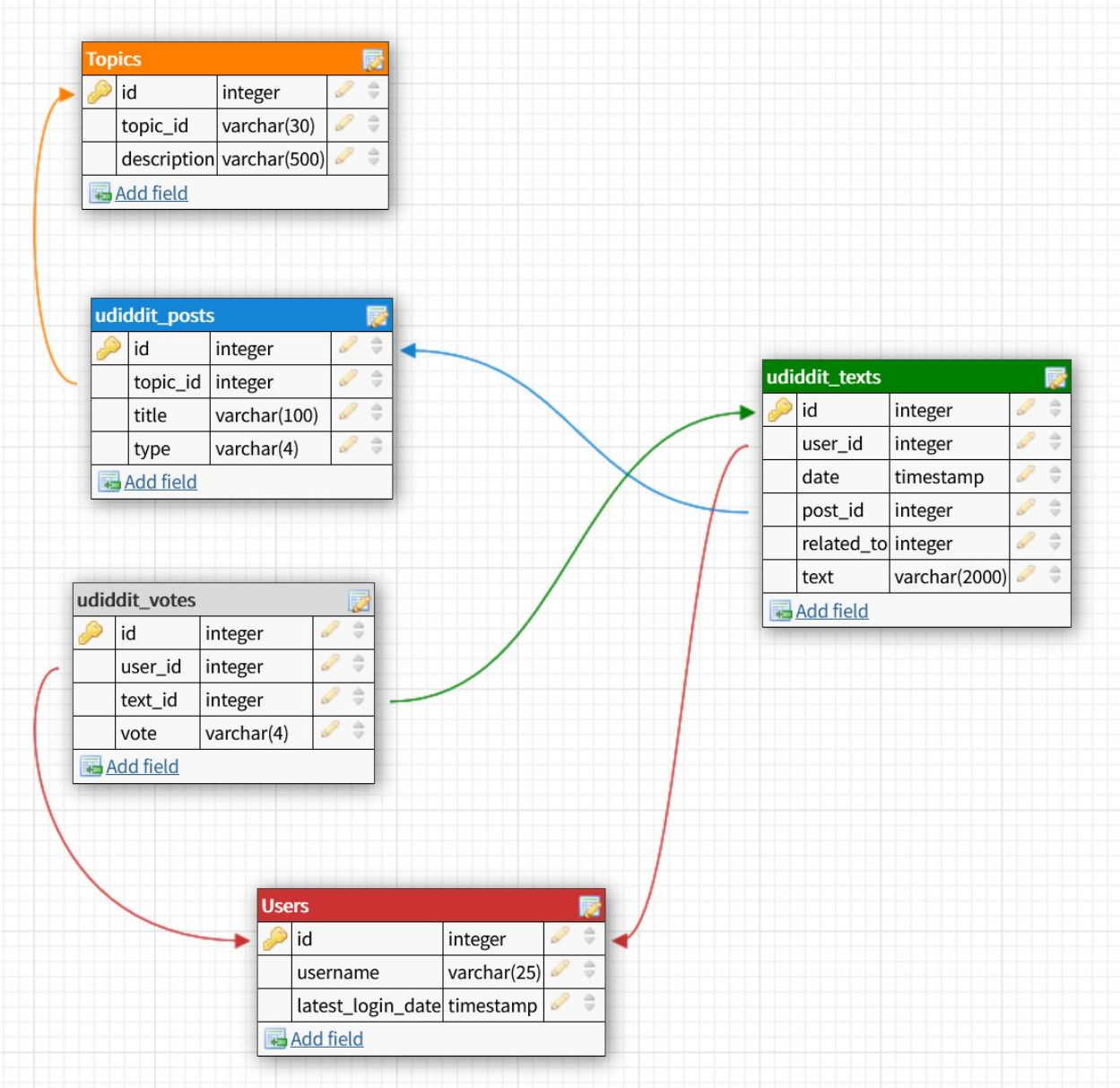
2. Guideline #2: here is a list of queries that Udiddit needs in order to support its website and administrative interface. Note that you don't need to produce the DQL for those queries: they are only provided to guide the design of your new database schema.
    a. List all users who haven't logged in 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: 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:

As an introduction I have added my planned database schema. Please review this (simplified) graphical illustration in combination with DDL details listed below.



```
'Table (1) USERS'
CREATE TABLE "users" (
 "id" SERIAL,
 "username" VARCHAR(25) NOT NULL,
 "latest_login_date" TIMESTAMP,
 CONSTRAINT "users_PK" PRIMARY KEY ("id"),
 CONSTRAINT "unique_usernames" UNIQUE ("username"),
 CONSTRAINT "username_not_empty" CHECK (LENGTH(TRIM("username"))>0)
);

'Index'
CREATE INDEX "lower_username" ON "users" (LOWER("username") VARCHAR_PATTERN_OPS);
CREATE TABLE "topics" (
 "id" SERIAL,
 "topic" varchar(30) NOT NULL,
 "description" VARCHAR(500),
 CONSTRAINT "topics_PK" PRIMARY KEY ("id"),
 CONSTRAINT "unique_topics" UNIQUE ("topic"),
 CONSTRAINT "topic_not_empty" CHECK (LENGTH(TRIM("topic"))>0)
);

CREATE INDEX "topic" ON "topics" (LOWER("topic") VARCHAR_PATTERN_OPS);

CREATE TABLE "udiddit_posts" (
 "id" BIGSERIAL,
 "topic_id" INTEGER NOT NULL,
 "title" VARCHAR(100) NOT NULL,
 "type" VARCHAR(7) NOT NULL,
 CONSTRAINT "posts" PRIMARY KEY ("id"),
 CONSTRAINT "title" UNIQUE ("title"),
 CONSTRAINT "title_not_empty" CHECK (LENGTH(TRIM("title"))>0),
 CONSTRAINT "type" CHECK (LOWER(TRIM("type"))='url' OR LOWER(TRIM("type"))='text'),
 FOREIGN KEY ("topic_id") REFERENCES "topics" ("id") ON DELETE CASCADE
);

CREATE TABLE "udiddit_texts" (
 "id" BIGSERIAL,
 "user_id" INTEGER,
 "date" TIMESTAMP,
 "post_id" BIGINT UNIQUE,
 "related_to" BIGINT,
 "text" VARCHAR(2000) NOT NULL,
 CONSTRAINT "texts" PRIMARY KEY ("id"),
 CONSTRAINT "unique_user_with_post" UNIQUE ("post_id", "user_id"),
 CONSTRAINT "text_not_empty" CHECK (LENGTH(TRIM("text"))>0),
 CONSTRAINT "parent_text" CHECK ("post_id" IS NULL OR "related_to" IS NULL),
```

```
  FOREIGN KEY ("user_id") REFERENCES "users" ("id") ON DELETE SET NULL,
  FOREIGN KEY ("post_id") REFERENCES "udiddit_posts" ("id") ON DELETE CASCADE,
  FOREIGN KEY ("related_to") REFERENCES "udiddit_texts" ("id") ON DELETE CASCADE
);

CREATE INDEX "children_of_parent_comments" ON "udiddit_texts" ("related_to");

CREATE TABLE "udiddit_votes" (
  "id" BIGSERIAL,
  "user_id" INTEGER,
  "text_id" INTEGER,
  "vote" VARCHAR(4),
  CONSTRAINT "votes" PRIMARY KEY ("id"),
  CONSTRAINT "user_id" UNIQUE ("user_id", "text_id"),
  CONSTRAINT "votes_valid" CHECK (LOWER(TRIM("vote")) = 'up' or
LOWER(TRIM("vote"))='down'),
  FOREIGN KEY ("user_id") REFERENCES "users" ("id") ON DELETE SET NULL,
  FOREIGN KEY ("text_id") REFERENCES "udiddit_texts" ("id") ON DELETE CASCADE
);

CREATE INDEX "post_votes" ON "udiddit_votes" ("text_id","vote");
CREATE INDEX "post_votes" ON "udiddit_votes" ("vote") VARCHAR_PATTERN_OPS;
```

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:

(1) Table “Users”

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

INSERT 0 100

INSERT INTO "users" ("username")
  SELECT DISTINCT "username" FROM  "bad_comments";

ERROR:  duplicate key value violates unique constraint
"unique_usernames"
DETAIL:  Key (username)=(Luz45) already exists.

SELECT DISTINCT "username" FROM  "bad_comments"
   WHERE "username" <> 'Luz45';
```

Comment:
100 users were inserted from table “bad_posts”. User ‘Luz45’ was part of the “bad_posts” as well as “bad_comments” table. This led to a violation because no “username” can be stored twice. Thus I migrated “username” values explicitly without ‘Luz45’. However, I do not lose any information because ‘Luz45’ is already part of my new table as the violation shows. As an alternative I could also say ***SELECT DISTINCT "username" FROM "bad_comments"***
WHERE "username" NOT IN (SELECT DISTINCT “username” FROM “bad_posts”);

(2) Table “Topics”

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

INSERT 0 89
```

Comment:
89 topics were inserted from table “bad_posts”.

(3) Table “Udiddit_Posts”

```
ALTER TABLE "bad_posts" ADD COLUMN "topic_id" INTEGER;

UPDATE "bad_posts" SET "topic_id" = (SELECT "id"
                                     FROM "topics"
                                    WHERE "topics"."topic"="bad_posts"."topic");

UPDATE 50000

INSERT INTO "udiddit_posts" ("id","topic_id", "title","type")
  SELECT DISTINCT "id",
                  "topic_id",
                  LEFT("title",100), 'url'
             FROM  "bad_posts"
            WHERE "url" IS NOT NULL AND "title" IS NOT NULL;

INSERT 0 37506

INSERT INTO "udiddit_posts" ("id","topic_id", "title","type")
  SELECT DISTINCT "id",
                  "topic_id",
                  LEFT("title",100), 'text'
             FROM  "bad_posts"
            WHERE "url" IS NULL AND "title" IS NOT NULL;

INSERT 0 12494
```

Comment:
50,000 posts were inserted from table “bad_posts”. I categorized them into ‘urls’ and ‘texts’ as both within one entity are not allowed. As I can see for my first migration step related to this table (insertion of topic_ids) there seem to be no doubles. Because, if so, I should have received a violation because ‘title’ has a unique constraint. Thus, if I would have an entry with ‘url’ and ‘text_contents’ the title would have to be inserted twice which would not work out.

It is important to note that insertion of a new post goes always together with an insertion in new text. The order is important: First the post is generated in “udiddit_posts” and then the text_content belonging to it is added in a second step to “udiddit_texts”. If the topic does not exist this has to be even done before. We will have to use transactions to ensure consistency.

(4) Table “Udiddit_Texts”

```
ALTER TABLE "bad_posts" ADD COLUMN "user_id" INTEGER;

ALTER TABLE

ALTER TABLE "bad_comments" ADD COLUMN "user_id" INTEGER;

ALTER TABLE

UPDATE "bad_posts"
    SET "user_id" = (SELECT "id"
                       FROM "users"
                      WHERE "users"."username"="bad_posts"."username");

UPDATE 50000

UPDATE "bad_comments”
    SET "user_id" = (SELECT "id"
                       FROM "users"
                      WHERE "users"."username"="bad_comments"."username");

UPDATE 100000

INSERT INTO "udiddit_texts" (“user_id”,"post_id","text")
  SELECT DISTINCT “user_id”,
                  "id",
                  "url"
             FROM "bad_posts"
            WHERE "url" IS NOT NULL;

INSERT 0 37506

INSERT INTO "udiddit_texts" (“user_id”,"post_id","text")
  SELECT DISTINCT “user_id”,
                  "id",
                  "text_content"
             FROM "bad_posts"
            WHERE "url" IS NULL;

INSERT 0 12494
INSERT INTO "udiddit_texts" ("user_id","related_to","text")
  SELECT DISTINCT "user_id",
```

```
                "Post_id",
                LEFT("text_content",2000)
            FROM "bad_comments";

INSERT 0 100000
```

Comment:
12,494 posts with *url* were inserted from table “bad_posts”. 37,506 posts with *text* were inserted from table “bad_posts”. These posts and its 100,000 comments related to them were inserted from table “bad_comments”. So 150.000 texts are now part of the ‘udiddit_texts’ table.

It is possible within this schema to comment on a post directly (which are top level comments where entry in ‘related_to’ is the ‘id’ in “udiddit_texts” where the text for the relevant “post_id” is shown). Furthermore, it is possible to comment on every existing comment thus having unlimited thread options. One thread can be regenerated by using “date” in combination with “related_to”-column ordered by date.

It is also important to note that insertion of a new post goes always together with an insertion in new text (so two tables will get a new entity, data is split up). The order is important: First the post is generated in “udiddit_posts” and then the text_content belonging to it is added in a second step to “udiddit_texts”. If the topic does not exist this has to be even done before. We will have to use transactions to ensure consistency. A new comment on an existing post is just inserted in table “udiddit_texts”.

(5) Table “Udiddit_Votes”

```
CREATE TABLE "votes_help" (
  "id" BIGSERIAL,
  “text_id” BIGINTEGER,
  "usernames_upvotes" varchar(25),
  "usernames_downvotes" varchar(25),
  "upvotes" VARCHAR(4),
  "downvotes" VARCHAR(4)
);

CREATE TABLE
```

```
INSERT INTO "votes_help" (“text_id”,"usernames_upvotes")
        SELECT “id”,regexp_split_to_table("upvotes", ',') FROM "bad_posts";

INSERT 0 249799

UPDATE "votes_help"
     SET "upvotes" = ‘up’  WHERE "votes_help"."usernames_upvotes" IS NOT NULL;

UPDATE 249799

INSERT INTO "votes_help" (“text_id”,"usernames_downvotes")
        SELECT “id”,regexp_split_to_table("downvotes", ',') FROM "bad_posts";

INSERT 0 249911

UPDATE "votes_help"
     SET "downvotes" = ‘down’  WHERE "votes_help"."usernames_upvotes" IS NOT NULL;

UPDATE 249911

ALTER TABLE "votes_help" ADD COLUMN "user_id" INTEGER;

ALTER TABLE

UPDATE "votes_help"
   SET "user_id" = (SELECT "id"
                    FROM "users"
                   WHERE “users"."username"="votes_help"."usernames_upvotes")

UPDATE 499710



UPDATE "votes_help"
   SET "user_id" = (SELECT "id"
                    FROM "users"
                   WHERE “users"."username"="votes_help"."usernames_downvotes")

UPDATE 499710

/*ADD USERS WHICH HAVE NEVER COMMENTED OR POSTED ANYTHING AND THUS
ARE STILL MISSING IN USERS TABLE */
```

```
INSERT INTO "users" ("username")
  SELECT DISTINCT "usernames_upvotes"
             FROM  "votes_help" WHERE "user_id" IS NULL AND
                    "usernames_upvotes" IS NOT NULL AND
                    "usernames_upvotes" NOT IN (SELECT "username" FROM users);

INSERT 0 1000

INSERT INTO "users" ("username")
  SELECT DISTINCT "usernames_downvotes"
             FROM  "votes_help" WHERE "user_id" IS NULL AND
                    "usernames_downvotes" IS NOT NULL AND
                    "usernames_downvotes" NOT IN (SELECT "username" FROM users);

INSERT 0 0

UPDATE "votes_help"
  SET "user_id" = (SELECT "user_id"
                   FROM "users"
                  WHERE "users"."username"="votes_help"."usernames_upvotes");

UPDATE 499710

UPDATE "votes_help"
  SET "user_id" = (SELECT "user_id"
                   FROM "users"
                  WHERE "users"."username"="votes_help"."usernames_downvotes");

UPDATE 45479




/*Migration Upvotes*/

INSERT INTO "udiddit_votes" ("text_id","user_id","vote")
  SELECT DISTINCT "text_id", "user_id","upvotes"
             FROM "votes_help"
            WHERE "upvotes" IS NOT NULL;

INSERT 0 249799

/*Migration Downvotes*/
```

```
INSERT INTO "udiddit_votes" ("text_id","user_id","vote")
  SELECT DISTINCT "text_id", "user_id","downvotes"
             FROM "votes_help"
            WHERE "downvotes" IS NOT NULL;

INSERT 0 249911
```

Comment:
First, a table “votes_help” is created after having created the new, later relevant table “udiddit_votes”. Here, four columns are generated with usernames that have upvoted and one column for usernames which have downvoted.

This is realized by ‘regexp_split_to_table’ accessing data in “upvotes” column in “bad_posts”. Then, in a second step, for each entity “up” is added. As there are not any “downvotes” yet, this can be done like this - otherwise this would not work. Then, in a third step, ‘regexp_split_to_table’ is used for fetching usernames from “downvotes” column in “bad_posts”. Then, in a fourth step, “down” is added for each entity where “upvotes” IS NULL as these are the only ones remaining without any “vote” flag because they were migrated in step 3.

Then, a column ‘user_id’ is added.

```
postgres=# SELECT * from "votes_help2" LIMIT 10;
   id   | text_id | usernames_upvotes | usernames_downvotes | upvotes | downvotes | user_id
--------+---------+-------------------+---------------------+---------+-----------+---------
 249788 |   49998 | Travon55          |                     | up      |           |
 249789 |   49998 | Albina_Lemke      |                     | up      |           |
 249790 |   49998 | Keagan3           |                     | up      |           |
 249791 |   49998 | Major65           |                     | up      |           |
 249792 |   49998 | Adriel50          |                     | up      |           |
 249793 |   49998 | Rosendo_Simonis   |                     | up      |           |
 249794 |   49998 | Mikayla.Bednar78  |                     | up      |           |
 249795 |   50000 | Rosanna.Bogan     |                     | up      |           |
 249796 |   50000 | Sammie29          |                     | up      |           |
 249797 |   50000 | Felicity.Cremin   |                     | up      |           |
(10 rows)
```

“Username_upvotes” and “usernames_downvotes” are filled in by using a ‘hidden join’:

```
postgres=# SELECT * FROM votes_help2 LIMIT 19;
   id   | text_id |  usernames_upvotes  | usernames_downvotes | upvotes | downvotes | user_id
--------+---------+---------------------+---------------------+---------+-----------+---------
-
 249799 |   50000 | Christophe.Terry78  |                     | up      |           |    8764
     67 |      22 | Hallie_Kuhn         |                     | up      |           |    9217
  14713 |    2969 | Jaydon7             |                     | up      |           |    9249
  14714 |    2969 | Arturo.Murray       |                     | up      |           |    8768
  11454 |    2290 | Tate58              |                     | up      |           |    8694
  11444 |    2287 | Amir78              |                     | up      |           |    8396
  12089 |    2420 | Alfonso47           |                     | up      |           |    9293
  14715 |    2969 | Katelin_Lubowitz1   |                     | up      |           |    8945
  14716 |    2969 | Elmo_Aufderhar31    |                     | up      |           |    9244
  16726 |    3385 | Anais60             |                     | up      |           |    8643
  14717 |    2969 | Chris_Graham61      |                     | up      |           |    9015
  14718 |    2969 | Terence.Ortiz94     |                     | up      |           |    9008
  14719 |    2970 | Chris.Ferry         |                     | up      |           |      98
  14720 |    2970 | Kasandra29          |                     | up      |           |    9176
  14721 |    2970 | Craig78             |                     | up      |           |    8908
  14722 |    2970 | Milo_Wehner         |                     | up      |           |    9093
  14723 |    2970 | Augustine.Schmitt22 |                     | up      |           |    9077
  14724 |    2970 | Eudora_Dickinson68  |                     | up      |           |    8610
  14725 |    2970 | Eric25              |                     | up      |           |    8832
(19 rows)
```

However, some user_ids remain with NULL. Why? This is because only users from “bad_posts” and “bad_comments” have been migrated to “users” table in part I. Thus, we have to add these users which have only voted but not commented or posted up to now to “users” table - so these 1000 users will also get their IDs after doing an UPDATE on “votes_help” table because these names are also part of “users” table now. So every entity has a “text_id”, a “user_id” and either an “upvote” or a “downvote”.
We can now migrate these three columns to the final table “udiddit_votes”. For querying the count of upvotes and downvotes we can use a SELECT COUNT(*) WHERE “vote” = ‘up’ for retrieving all upvotes and use a SELECT COUNT(*) WHERE “vote” = ‘down’ for getting all downvotes/dislikes for a certain post or comment.

```
postgres=# SELECT * FROM "udiddit_votes" LIMIT 19;
 id | user_id | text_id | vote
----+---------+---------+------
  1 |         |   24797 | up
  2 |         |   47114 | up
  3 |         |    8495 | up
  4 |         |   44952 | up
  5 |         |   23979 | up
  6 |         |   23220 | up
  7 |         |    7163 | up
  8 |         |   15602 | up
  9 |         |   28925 | up
 10 |         |   30239 | up
 11 |         |   45718 | up
 12 |         |    4600 | up
 13 |         |    5785 | up
 14 |         |    2481 | up
 15 |         |   32394 | up
 16 |         |   45494 | up
 17 |         |   17328 | up
 18 |         |    4555 | up
 19 |         |   28405 | up
(19 rows)

postgres=# SELECT * FROM  "udiddit_votes" WHERE "user_id" IS NOT NULL LIMIT 19;
  id   | user_id | text_id | vote
-------+---------+---------+------
 45480 |    9044 |   25132 | down
 45481 |    9320 |   48237 | down
 45482 |    9254 |   48626 | down
 45483 |    8480 |    7644 | down
 45484 |    9003 |   15539 | down
 45485 |    8875 |   49507 | down
 45486 |    8413 |   15314 | down
 45487 |    8685 |   34286 | down
 45488 |    8868 |   27998 | down
 45489 |    9047 |   39625 | down
 45490 |    8719 |   11110 | down
 45491 |    9223 |   19911 | down
 45492 |    8535 |   10080 | down
```