ebd

Last edited by **Rodrigo de Sousa** 20 hours ago

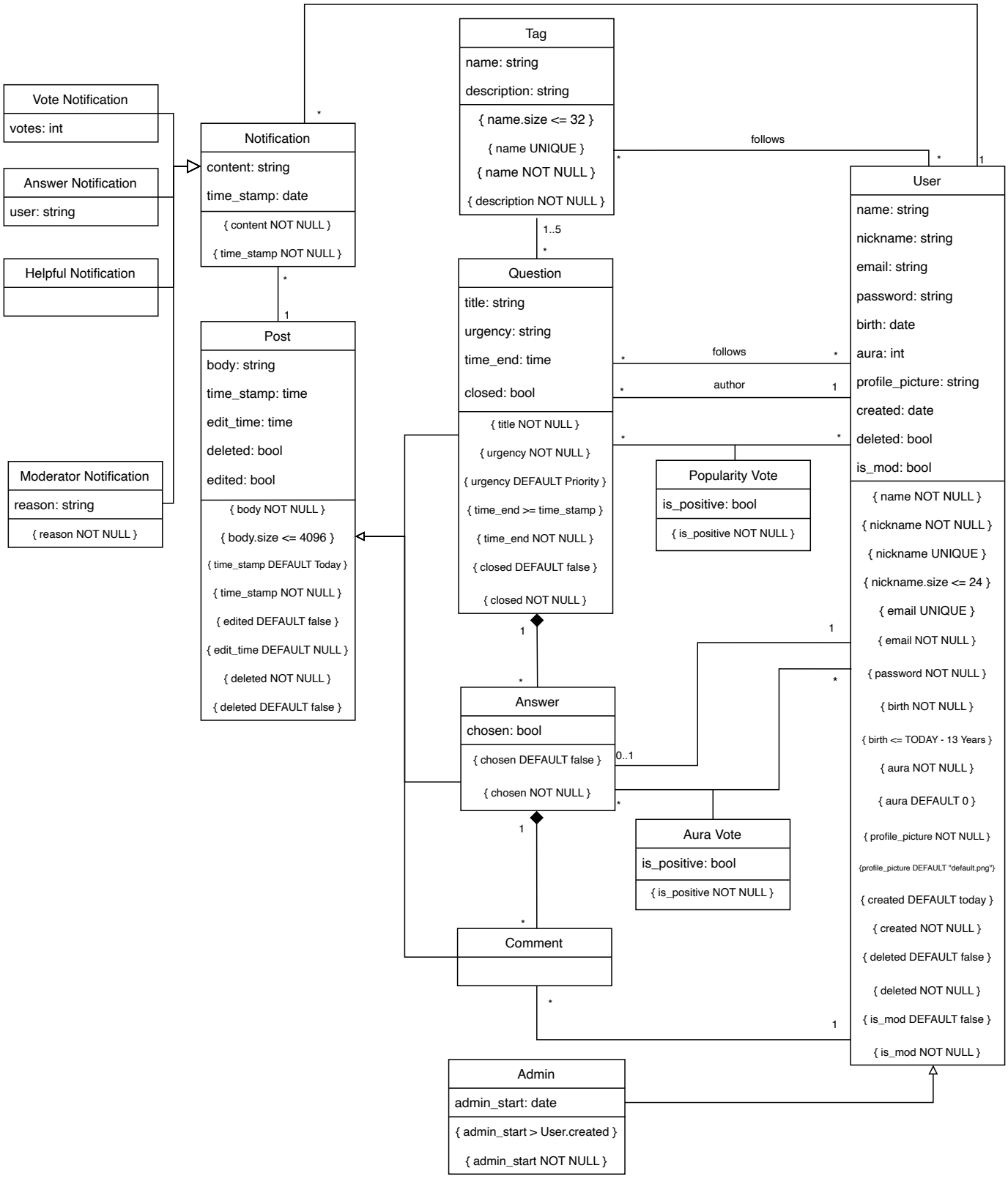
EBD: Database Specification Component

Our project aims to redefine what it means to be online and have questions. Our goal is to ensure that every user feels comfortable asking questions and confident that they will receive fast and constructive answers in a friendly environment. We will have a rating system to ensure only the best and most helpful answers according to the community get to the top.

A4: Conceptual Data Model

The goal for this artefact is to define how data will work on our website. It also serves as a guideline for features.

1. Class diagram



2. Additional Business Rules

2.1 User Rules

Attribute	Rule	Default/Constraint
`nickname`	Must be unique	`nickname.size >= 24`
`email`	Must be unique	N/A
`birth`	User must be at least 13 years old	`birth <= TODAY - 13 Years`
`profile_picture`	Defaults to "default.png" if not provided	`profile_picture DEFAULT 'default.png'`

`created`	Set to current date when account is created	`created DEFAULT today`
`isMod`	Defaults to `false`	`is_mod DEFAULT false`

2.2 Tag Rules

Attribute	Rule	Default/Constraint
`name`	Must be unique and cannot exceed 32 characters	`name.size <= 32`, `name UNIQUE`
`description`	Cannot be null	`description NOT NULL`
`relationships`	Each tag must be linked to at least one question	1..* relationship with `Question`

2.3 Question and Answer Rules

Attribute	Rule	Default/Constraint
`title`	Cannot be empty	`title NOT NULL`
`urgency`	Defaults to "Priority" if not specified	`urgency DEFAULT Priority`
`time_end`	Must be after `time_stamp`	`time_end >= time_stamp`
`closed`	Indicates if the question is closed; no new answers allowed if `true`	`closed DEFAULT false`, `closed NOT NULL`
`chosen` (Answer)	Only one answer can be marked as "chosen" per question	`chosen DEFAULT false`

2.4 Post and Comment Rules

Attribute	Rule	Default/Constraint
`body` (Post)	Cannot exceed 4096 characters	`body.size <= 4096`
`time_stamp`	Set to current date when created	`time_stamp DEFAULT Today`
`edited`	Indicates if post is edited	`edited DEFAULT false`
`deleted`	Indicates if post is deleted	`deleted DEFAULT false`, `deleted NOT NULL`
`relationship`	Comments must be linked to a `Post` or `Answer`	N/A

2.5 Notifications and Moderation Rules

Notification Type	Specific Attribute	Rule	Default/Constraint
Vote Notification	`votes`	Number of votes counted	`votes INT`
Answer Notification	`user`	User associated with answer	`user STRING`
Helpful Notification	N/A	Notification for helpful posts	N/A
Moderator Notification	`reason`	Reason must be provided	`reason NOT NULL`
General Notification	`content`, `time_stamp`	Cannot be null	`content NOT NULL`, `time_stamp NOT NULL`

2.6 Voting Rules

Attribute	Rule	Default/Constraint
`is_positive` (Popularity Vote)	Indicates if vote is positive or negative	`is_positive NOT NULL`

`is_positive` (Aura Vote)	Indicates if aura is positive or negative	`is_positive NOT NULL`
Relationships	User can follow other users or tags	Many-to-many with `User` and `Tag`

2.7 Admin Rules

Attribute	Rule	Default/Constraint
`admin_start`	Date when user was made an admin	`admin_start > User.created`, `admin_start NOT NULL`
Relationship	Admin must be a previously created user	`User.created < TODAY`

2.8 Data Integrity and Consistency Rules

Rule	Description
Only users can edit their own posts/comments	`edited DEFAULT false` in `Post`
Deleted items are not visible to regular users	`deleted` attribute controls visibility
Comments are tied to a specific post or answer	Comments cannot exist independently

A5: Relational Schema, validation and schema refinement

The goal of this artefact is to provide a detailed specification of the relational schema, including validation and refinement processes. It ensures that the database design is robust, normalized, and adheres to best practices. This section will cover the relational schema, domains, schema validation, and normalization steps to guarantee data integrity and efficiency.

1. Relational Schema

1.1 Entity-Relation Tables

Relation reference	Relation Compact Notation
R01	user(id, name **NN**, nickname **NN UK CK nickname.size <= 24**, email **NN UK**, password **NN**, birth **NN CK birth <= TODAY - 13 Years**, aura **NN DEFAULT 0**, profile_picture **NN DEFAULT default.png**, created **NN DEFAULT today**, deleted **NN DEFAULT false**, is_mod **NN DEFAULT false**)
R02	admin(id → user, admin_start **NN CK admin_start >= user.created**)
R03	tag(id, name **UK NN CK name.size <= 32**, description **NN**)
R04	post(id, body **NN CK body.size <= 4096**, time_stamp **NN**, deleted **NN DEFAULT false**, edited **NN DEFAULT false**, edit_time **DEFAULT null**)
R05	question(id, title **NN**, urgency **NN default Priority**, time_end **NN CK time_end >= time_stamp**, closed **NN DEFAULT false**, author_id → user **NN**, post_id → post **NN**)
R06	answer(id, chosen **NN DEFAULT false** , question_id → question **NN**, author_id → user **NN**, post_id → post **NN**)
R07	comment(id, answer_id → answer **NN**, author_id → user **NN**, post_id → post **NN**)
R08	popularity_vote(id, is_positive **NN**, user_id → user **NN**, question_id → question **NN**)
R09	aura_vote(id, is_positive **NN**, user_id → user **NN**, answer_id → answer **NN**)
R10	notification(id, content **NN**, time_stamp **NN**, post_id → post **NN**, user_id → user **NN**)
R11	vote_notification(id → notification, votes **int**)
R12	answer_notification(id → notification, user **NN DEFAULT "A User"**)

R13	helpful_notification(id → notification)
R14	moderator_notification(id → notification, reason **NN**)

1.2 Many-to-Many Mapping Tables:

Relation reference	Relation Compact Notation
R15	user_follows_tag(user_id → user, tag_id → tag)
R16	user_follows_question(user_id → user, question_id → question)
R17	question_tags(question_id → question, tag_id → tag)

2. Domains

Domain Name	Domain Specification
Today	DATE DEFAULT CURRENT_DATE
Priority	ENUM ('Red', 'Orange', 'Yellow', 'Green')

3. Schema validation

Table	Keys	Functional Dependencies	Normal Form
User	`{id}`, `{email}`	`{id} → {name, nickname, email, password, birth, aura, profile_picture, created, deleted, is_mod}`, `{email} → {id, name, nickname, password, birth, aura, profile_picture, created, deleted, is_mod}`	BCNF
Admin	`{id}`	`{id} → {admin_start}`	BCNF
Tag	`{id}`, `{name}`	`{id} → {name, description}`, `{name} → {id, description}`	BCNF
Post	`{id}`	`{id} → {body, time_stamp, deleted, edited, edit_time}`	BCNF
Question	`{id}`	`{id} → {title, urgency, time_end, closed, author_id, post_id}`	BCNF
Answer	`{id}`	`{id} → {chosen, question_id, author_id, post_id}`	BCNF
Comment	`{id}`	`{id} → {answer_id, author_id, post_id}`	BCNF
Popularity Vote	`{id}`	`{id} → {is_positive, user_id, question_id}`	BCNF
Aura Vote	`{id}`	`{id} → {is_positive, user_id, answer_id}`	BCNF
Notification	`{id}`	`{id} → {content, time_stamp, post_id, user_id}`	BCNF
Vote Notification	`{id}`	`{id} → {votes}`	BCNF
Answer Notification	`{id}`	`{id} → {user_id}`	BCNF
Helpful Notification	`{id}`	`{id} → {}`	BCNF
Moderator Notification	`{id}`	`{id} → {reason}`	BCNF

3.1. Candidate Keys:

In the `User` table, both `id` (primary key) and `email` (unique key) are candidate keys.

- **Keys**: `{id}`, `{email}`

3.2. Functional Dependencies:

- **FD0101**: `id → {email, name, nickname, password, birth, aura, profile_picture, created, deleted}`
- **FD0102**: `email → {id, name, nickname, password, birth, aura, profile_picture, created, deleted}`

3.3. Normal Form Analysis:

- **1NF**: The table is in 1NF as all attributes are atomic.
- **2NF**: The table is in 2NF since all non-key attributes fully depend on the primary key (`id`).
- **3NF**: The table is in 3NF because there are no transitive dependencies; all attributes depend directly on the primary key.
- **BCNF**: The table satisfies BCNF as both functional dependencies (`id → ...` and `email → ...`) involve superkeys.

3.4. Conclusion:

The `User` table is already in **BCNF**, with no need for further normalization or decomposition. This applies to all the other tables. However, to standardise the schema, we will only use the `id` as the primary key for all tables.

A6: Indexes, triggers, transactions and database population

This artefact establishes a foundation for efficient data handling and integrity in the project by defining optimized indexing, triggers, and transaction management. These elements are crucial for maintaining performance, data consistency, and scalability as user interactions and data volume grow.

1. Database Workload

Relation reference	Relation Name	Order of magnitude	Estimated growth
R01	user	thousands	tens per day
R02	admin	units	units per month
R03	tag	hundreds	units per day
R04	post	thousands	hundreds per day
R05	question	thousands	tens per day
R06	answer	hundreds	tens per day
R07	comment	hundreds	tens per day
R08	popularity_vote	thousands	hundreds per day
R09	aura_vote	thousands	hundreds per day
R10	notification	hundreds	hundreds per day
R11	vote_notification	hundreds	hundreds per day
R12	answer_notification	hundreds	hundreds per day
R13	helpful_notification	hundreds	hundreds per day
R14	moderator_notification	hundreds	hundreds per day
R15	user_follows_tag	hundreds	tens per day
R16	user_follows_question	hundreds	tens per day
R17	question_tags	hundreds	tens per day

2. Proposed Indices

2.1. Performance Indices

Index	IDX01
Relation	user
Attribute	nickname
Type	B-tree
Cardinality	Low, if few distinct nicknames; Medium to High, if many unique nicknames
Clustering	No clustering unless nickname is frequently queried in sorted order; if so, clustering could enhance performance
Justification	The B-tree index will efficiently support equality and range queries on nickname. Hash indices could also be considered but are only effective for exact matches, while B-trees support a broader range of operations and are generally more versatile.
`SQL code`	`CREATE INDEX IDX01 ON user(nickname);`

Index	IDX02
Relation	tag
Attribute	name
Type	B-tree
Cardinality	Likely High, as `name` in `tag` may contain many unique values (depending on the dataset)
Clustering	Not clustered unless `name` is frequently queried in sorted order; clustering could help in such cases
Justification	A B-tree index on `name` will support efficient lookups, especially for exact matches and range-based queries. This index can improve query performance for filtering and joining on `name`. Hash indexing might be considered if only exact matches are needed, but B-tree provides more flexibility overall.
`SQL code`	`CREATE INDEX IDX02 ON tag(name);`

2.2. Full-text Search Indices with Field Weighting

Index	IDX03
Relation	post
Attribute	body
Type	GIN
Clustering	No clustering
Justification	Supports fast full-text search for post content. Field weighting helps prioritize search results for better relevance.
`SQL code`	`CREATE INDEX IDX03 ON post USING GIN ( setweight(to_tsvector('english', body), 'A'));`

Index	IDX04
Relation	question
Attribute	title
Type	GIN
Clustering	No clustering

Justification	Supports fast full-text search for question content with a focus on relevance.
`SQL code`	`CREATE INDEX IDX04 ON question USING GIN ( setweight(to_tsvector('english', title), 'A'));`

3. Triggers

Trigger	**TRIGGER01: Update edited field on post when body changes**
Description	This trigger sets the edited field to TRUE and updates edit_time when a post is modified.

```
CREATE OR REPLACE FUNCTION update_post_edit_time()
    RETURNS TRIGGER AS $$
        BEGIN
            NEW.edited := TRUE;
            NEW.edit_time := CURRENT_TIMESTAMP;
            RETURN NEW;
        END;
    $$ LANGUAGE plpgsql;

CREATE TRIGGER TRIGGER01
BEFORE UPDATE ON post
FOR EACH ROW
WHEN (OLD.body IS DISTINCT FROM NEW.body)
EXECUTE FUNCTION update_post_edit_time();
```

Trigger	**TRIGGER02: Update aura on user when aura_vote is added**
Description	This trigger updates the aura value on the user when a new aura_vote is added, reflecting the change based on is_positive.

```
CREATE OR REPLACE FUNCTION update_user_aura()
RETURNS TRIGGER AS $$
BEGIN
    IF NEW.is_positive THEN
        UPDATE lbaw24112.user SET aura = aura + 1 WHERE id = NEW.user_id;
    ELSE
        UPDATE lbaw24112.user SET aura = aura - 1 WHERE id = NEW.user_id;
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER TRIGGER02
AFTER INSERT ON aura_vote
FOR EACH ROW
EXECUTE FUNCTION update_user_aura();
```

4. Transactions

SQL Reference	**Transaction01**
Justification	Ensures that deleting a user marks all their posts as deleted to maintain referential integrity without hard deletion of data.
Isolation level	SERIALIZABLE

```
BEGIN;

UPDATE lbaw24112.user
```

```
SET deleted = TRUE
WHERE id = :user_id;

UPDATE post
SET deleted = TRUE
WHERE id IN (SELECT post_id FROM question WHERE author_id = :user_id)
   OR id IN (SELECT post_id FROM answer WHERE author_id = :user_id)
   OR id IN (SELECT post_id FROM comment WHERE author_id = :user_id);

COMMIT;
```

Annex A. SQL Code

DB SCHEMA CREATION SCRIPT

DB POPULATE SCRIPT

DB TRIGGERS SCRIPT

A.1. Database schema

```
SET search_path TO lbaw24112;


DROP TABLE IF EXISTS lbaw24112.user;
DROP TABLE IF EXISTS admin;
DROP TABLE IF EXISTS tag;
DROP TABLE IF EXISTS question;
DROP TABLE IF EXISTS answer;
DROP TABLE IF EXISTS comment;
DROP TABLE IF EXISTS popularity_vote;
DROP TABLE IF EXISTS aura_vote;
DROP TABLE IF EXISTS notification;
DROP TABLE IF EXISTS moderator_notification;
DROP TABLE IF EXISTS vote_notification;
DROP TABLE IF EXISTS helpful_notification;
DROP TABLE IF EXISTS answer_notification;
DROP TABLE IF EXISTS post;
DROP TABLE IF EXISTS user_follows_tag;
DROP TABLE IF EXISTS user_follows_question;
DROP TABLE IF EXISTS question_tags;

CREATE DOMAIN Today DATE DEFAULT CURRENT_DATE;

CREATE TYPE priority_level AS ENUM ('Red', 'Orange', 'Yellow', 'Green');
CREATE DOMAIN Priority AS priority_level;


-- R01
CREATE TABLE IF NOT EXISTS lbaw24112.user (
  id SERIAL PRIMARY KEY UNIQUE NOT NULL,
  name VARCHAR NOT NULL,
  nickname VARCHAR UNIQUE CHECK (LENGTH(nickname) <= 24) NOT NULL,
  email VARCHAR UNIQUE NOT NULL,
  password VARCHAR NOT NULL,
  birth_date DATE CHECK (birth_date <= CURRENT_DATE - INTERVAL '13 years') NOT NULL,
  aura INT DEFAULT 0 NOT NULL,
  profile_picture VARCHAR DEFAULT 'default.png' NOT NULL,
  created DATE DEFAULT CURRENT_DATE NOT NULL,
  deleted BOOLEAN DEFAULT FALSE NOT NULL,
  is_mod BOOLEAN DEFAULT FALSE NOT NULL
);
```

```sql
-- R02
CREATE TABLE IF NOT EXISTS admin (
  id SERIAL PRIMARY KEY REFERENCES lbaw24112.user (id) UNIQUE NOT NULL,
  admin_start DATE NOT NULL
);

-- R03
CREATE TABLE IF NOT EXISTS tag (
  id SERIAL PRIMARY KEY UNIQUE NOT NULL,
  name VARCHAR(32) UNIQUE CHECK (LENGTH(name) <= 32) NOT NULL,
  description TEXT NOT NULL
);

-- R04
CREATE TABLE IF NOT EXISTS post (
  id SERIAL PRIMARY KEY UNIQUE NOT NULL,
  body TEXT CHECK (LENGTH(body) >= 4096) NOT NULL,
  time_stamp TIMESTAMP NOT NULL,
  deleted BOOLEAN DEFAULT FALSE NOT NULL,
  edited BOOLEAN DEFAULT FALSE NOT NULL,
  edit_time TIMESTAMP DEFAULT NULL
);

-- R05
CREATE TABLE IF NOT EXISTS question (
  id SERIAL PRIMARY KEY UNIQUE NOT NULL,
  title TEXT NOT NULL,
  urgency TEXT NOT NULL,
  time_end TIMESTAMP NOT NULL,
  closed BOOLEAN DEFAULT FALSE NOT NULL,
  author_id INTEGER REFERENCES lbaw24112.user (id) NOT NULL,
  post_id INTEGER REFERENCES post (id) NOT NULL
);

-- R06
CREATE TABLE IF NOT EXISTS answer (
  id SERIAL PRIMARY KEY UNIQUE NOT NULL,
  chosen BOOLEAN DEFAULT FALSE NOT NULL,
  question_id INTEGER NOT NULL REFERENCES question(id),
  author_id INTEGER REFERENCES lbaw24112.user (id) NOT NULL,
  post_id INTEGER REFERENCES post (id) NOT NULL
);

-- R07
CREATE TABLE IF NOT EXISTS comment (
  id SERIAL PRIMARY KEY UNIQUE NOT NULL,
  answer_id  INTEGER NOT NULL REFERENCES answer(id),
  author_id INTEGER REFERENCES lbaw24112.user (id) NOT NULL,
  post_id INTEGER REFERENCES post (id) NOT NULL
);

-- R08
CREATE TABLE IF NOT EXISTS popularity_vote (
  id SERIAL PRIMARY KEY UNIQUE NOT NULL,
  is_positive BOOLEAN NOT NULL,
  user_id INTEGER REFERENCES lbaw24112.user (id) NOT NULL,
  question_id  INTEGER REFERENCES question(id) NOT NULL
);

-- R09
CREATE TABLE IF NOT EXISTS aura_vote (
  id SERIAL PRIMARY KEY UNIQUE NOT NULL,
  is_positive BOOLEAN NOT NULL,
```

```
  user_id INTEGER REFERENCES lbaw24112.user (id) NOT NULL,
  answer_id  INTEGER REFERENCES answer(id) NOT NULL
);

-- R10
CREATE TABLE IF NOT EXISTS notification (
  id SERIAL PRIMARY KEY UNIQUE NOT NULL,
  content TEXT NOT NULL,
  time_stamp TIMESTAMP NOT NULL,
  post_id INTEGER REFERENCES post (id) NOT NULL,
  user_id INTEGER REFERENCES lbaw24112.user (id) NOT NULL
);

-- R11
CREATE TABLE IF NOT EXISTS vote_notification (
  notification_id SERIAL PRIMARY KEY REFERENCES notification(id) NOT NULL,
  votes INTEGER NOT NULL
);

-- R12
CREATE TABLE IF NOT EXISTS answer_notification (
  notification_id SERIAL PRIMARY KEY REFERENCES notification(id) NOT NULL,
  user_ TEXT DEFAULT 'A user' NOT NULL
);

-- R13
CREATE TABLE IF NOT EXISTS helpful_notification (
  notification_id SERIAL PRIMARY KEY REFERENCES notification(id) NOT NULL
);

-- R14
CREATE TABLE IF NOT EXISTS moderator_notification (
  notification_id SERIAL PRIMARY KEY REFERENCES notification(id) NOT NULL,
  reason TEXT NOT NULL
);

-- R15
CREATE TABLE IF NOT EXISTS user_follows_tag (
  user_id INTEGER REFERENCES lbaw24112.user(id) NOT NULL,
  tag_id INTEGER REFERENCES tag(id) NOT NULL
);

-- R16
CREATE TABLE IF NOT EXISTS user_follows_question (
  user_id INTEGER REFERENCES lbaw24112.user(id) NOT NULL,
  question_id INTEGER REFERENCES question(id) NOT NULL
);

-- R17
CREATE TABLE IF NOT EXISTS question_tags (
  question_id INTEGER REFERENCES question(id) NOT NULL,
  tag_id INTEGER REFERENCES tag(id) NOT NULL
);
```

A.2. Database population

```
INSERT INTO lbaw24112.user(name, nickname, email, password, birth_date, profile_picture)
VALUES ('Leonor', 'Nónó', 'leonoremail@fake.com', '8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc
VALUES ('Rodrigo', 'Rodri_5', 'rodrigoemail@fake.com', '8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c9
VALUES ('Pedro', 'Puka', 'pedroemail@fake.com', '8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c
VALUES ('Afonso', 'Osnofa', 'afonsoemail@fake.com', '8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923a
```

```
INSERT INTO tag(name, description)
VALUES ('computers', 'all things related to the little machines that we control (or atleast think we do)');
VALUES ('cookies', 'from the savoury to the yummy, all things cookies');
VALUES ('music', 'all bangers included from mozart to ksi')

INSERT INTO post(body)
VALUES ('my computer crashed tooday, it was driving me to school and now i am lost.');
VALUES ('I love biscuits. Hungaros and belgas are the best.');
VALUES ('I love music, it is the best thing in the world.');
VALUES ('I dont know why my tummy hurts, I ate a lot of cookies this morning but I was hungry and my tummy was


INSERT INTO question(title, urgency, time_end, author_id ,post_id)
VALUES ('I need help fixing my computer!!', 'Red', '2021-06-01 00:00:00', 1, 1);
VALUES ('Any new biscuit recomendation?', 'Green', '2021-06-01 00:00:00', 4, 2);
VALUES ('What is the best music genre?', 'Yellow', '2021-06-01 00:00:00', 3, 3);
VALUES ('Why does my tummy hurt?' 'Orange', '2014-03-16 16:31:54', 2, 4)

INSERT INTO question_tags(question_id, tag_id)
VALUES (1, 1);
VALUES (2, 2);
VALUES (3, 3);
VALUES (4, 2);


BEGIN;

UPDATE lbaw24112.user
SET deleted = TRUE
WHERE id = :user_id;

UPDATE post
SET deleted = TRUE
WHERE id IN (SELECT post_id FROM question WHERE author_id = :user_id)
   OR id IN (SELECT post_id FROM answer WHERE author_id = :user_id)
   OR id IN (SELECT post_id FROM comment WHERE author_id = :user_id);

COMMIT;
```

Revision history

Changes made to the first submission:

1.

GROUP112, 04/11/2024

- Afonso Castro, up202208026@up.pt
- Leonor Couto, up202205796@up.pt
- Pedro Santos, up202205900@up.pt
- Rodrigo de Sousa, up202205751@up.pt (editor)