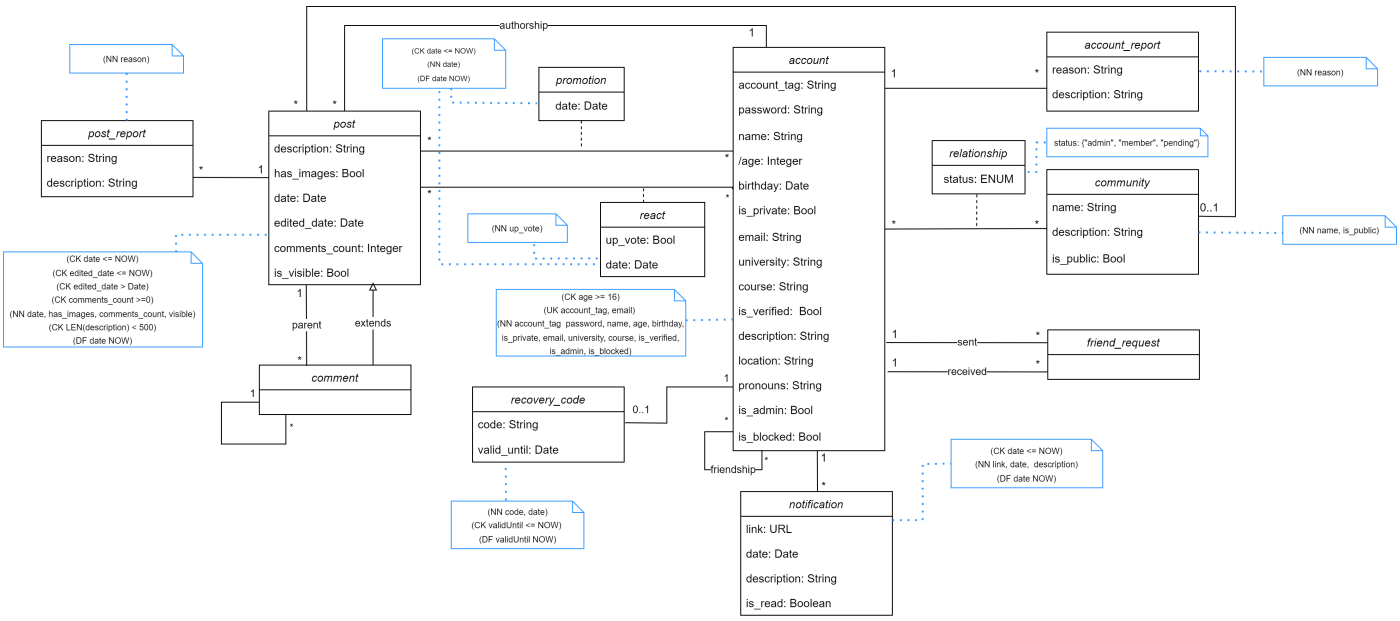
A4: Conceptual Data Model

This artifact contains the identification and description of the entities and relationships that are relevant to the database specification.

1. Class diagram



2. Additional Business Rules

Identifier	Name	Description
BR001	Upon Account Deletion	Upon account deletion, shared user data (e.g. comments, reactions) is kept but made anonymous.
BR002	Upon Group Creation	Upon group creation, the group's creator is made it's first group administrator.
BR003	Users's Self Interaction	A user can react, promote and comment under their own post.
BR004	Chronological Order	An authenticated user's post date must be more recent than the account's creation date.
BR005	Content Moderation	NSFW, extremist or hateful content will be deleted with the possibility of the author's account being terminated.
BR006	Reacting and Promoting Posts	Each user can only react and/or promote each post once.

Identifier	Name	Description
BR007	Friend Requests Duality	Once a user befriends another user neither of them can resend a friend request to each other.
BR008	Group Administrator	Each group has at least one administrator

A5: Relational Schema, validation and schema refinement

This artifact contains the Relational Schema obtained by mapping from the Conceptual Data Model.

1. Relational Schema

Relation reference	Relation Compact Notation
R01	account(id, account_tag **NN UK**, password **NN**, name **NN**, age **NN (CK age >= 16)**, birthday **NN**, is_private **NN**, email **UK NN**, university **NN**, course **NN**, is_verified **NN**, description, location, pronouns, is_admin **NN**, is_blocked **NN**)
R02	account_report(id, reason **NN**, description, id_account_reported -> account **NN**)
R03	post(id, parent->post, owner->account **NN**, group->group, description **(CK LEN(description) < 500)**, has_images **NN**, date **NN (CK date <= NOW) (DF date NOW)**, edited_date **(CK editedDate <= NOW) (CK editedDate > Date)**, comments_count **NN (CK commentsCount >=0)**, is_visible **NN**)
R04	community(id, name **NN**, description, is_public **NN**)
R05	friend_request(id_sender->account, id_receiver->account)
R06	notification(id, id_account->account **NN**, link **NN**, date **NN (CK date <= NOW) (DF date NOW)**, description **NN**, is_read **NN**)
R07	recovery_code(id, id_account->account **NN**, code **NN**, valid_until **NN (CK valid_until <= NOW) (DF valid_until NOW)**)
R08	post_report(id, id_post->post **NN**, reason **NN**, description
R09	relationship(id_group-> group, id_account-> account, status **NN**)
R10	post_promotion(id_account-> account, id_post-> post, date **NN (CK date <= NOW) (DF date NOW)**)
R11	post_react(id_account-> account, id_post-> post, date **NN (CK date <= NOW) (DF date NOW)**, up_vote **NN**)

Relation reference	Relation Compact Notation
R12	friendship(id_account_1-> account, (id_account_2-> account)

Legend:

- UK = UNIQUE KEY
- NN = NOT NULL
- DF = DEFAULT
- CK = CHECK

2. Domains

Domain Name	Domain Specification
Status	ENUM ('Admin', 'Member', 'Pending')

3. Schema validation

TABLE R01	account
Keys	{id}
Functional Dependencies:	
FD0101	id → {account_tag, password, name, age, birthday, is_private, email, university, course, verified, description, location, pronouns, is_admin, is_blocked}
NORMAL FORM	BCNF

TABLE R02	community
Keys	{id}
Functional Dependencies:	
FD401	id → {name, description, is_public}
NORMAL FORM	BCNF

TABLE R03	post
Keys	{id}

TABLE R03	post
Functional Dependencies:	
FD0301	id → {parent, owner, group, description, has_images, date, edited_date, comments_count, visible}
NORMAL FORM	BCNF

TABLE R04	account_report
Keys	{id}
Functional Dependencies:	
FD0201	id → {reason, description, account_id}
NORMAL FORM	BCNF

TABLE R05	friend_request
Keys	{sender_id, receiver_id}
Functional Dependencies:	-
NORMAL FORM	BCNF

TABLE R06	notification
Keys	{id}
Functional Dependencies:	
FD0601	id → {id_account, link, date, description}
NORMAL FORM	BCNF

TABLE R07	recovery_code
Keys	{id}
Functional Dependencies:	
FD0701	id → {id_account, code, valid_until}
NORMAL FORM	BCNF

TABLE R08	post_report

TABLE R08	post_report
Keys	{id}
Functional Dependencies:	
FD0801	id → {id_post, reason, description}
NORMAL FORM	BCNF

TABLE R09	relationship
Keys	{id_group, id_account}
Functional Dependencies:	
FD0901	{id_group, id_account} → {status}
NORMAL FORM	BCNF

TABLE R10	post_promotion
Keys	{id_account, id_post}
Functional Dependencies:	
FD1001	{id_account, id_post} → {date}
NORMAL FORM	BCNF

TABLE R11	post_reaction
Keys	{id_account, id_post}
Functional Dependencies:	
FD1101	{id_account, id_post} → {date, upvote}
NORMAL FORM	BCNF

TABLE R12	friendship
Keys	{id_account_1, id_account_2}
Functional Dependencies:	-
NORMAL FORM	BCNF

All relations are in the Boyce–Codd Normal Form (BCNF) because for every one of its dependencies X → Y, X is a superkey for schema R.

Due to this, the relational schema is also in the BCNF and, therefore, the schema does not need to be further normalised.

A6: Indexes, triggers, transactions and database population

1. Database Workload

Relation reference	Relation Name	Order of magnitude	Estimated growth
R01	account	10k (tens of thousands)	10 (tens) / day
R02	notification	1k (thousands)	100 (hundreds) / day
R03	account report	100 (hundreds)	1 (units) / day
R04	community	1k (thousands)	10 (tens) / day
R05	relationship	1k (thousands)	10 (tens) / day
R06	friend request	10k (tens of thousands)	100 (hundreds) / day
R07	recovery code	100 (hundreds)	1 (units) / day
R08	react	1M (milions)	1k (thousands) / day
R09	promotion	10k (tens of thousands)	100 (hundreds) / day
R10	post	100k (hundreds of thousands)	100 (hundreds) / day
R11	comment	10k (tens of thousands)	100 (hundreds) / day
R12	post report	1k (thousands)	10 (tens) / day

2. Proposed Indices

2.1. Performance Indices

Index	IDX01
Relation	post
Attribute	publication_date
Type	B-tree
Cardinality	High
Clustering	No

Index	IDX01
Justification	Dates are often going to be searched, for example when searching for a post's children comments
`SQL code`	CREATE INDEX post_date_idx ON post USING btree(publication_date);

Index	IDX02
Relation	post
Attribute	(owner_id, publication_date)
Type	B-tree
Cardinality	High
Clustering	Yes
Justification	It is often necessary to get all the posts by a specific user and, most of the time, we'll want to order them by date
`SQL code`	CREATE INDEX post_owner_date_idx ON post USING BTREE(owner_id, publication_date);

2.2. Full-text Search Indices

Index	IDX03
Relation	post
Attribute	tsvectors (derived from description)
Type	GIN (important short look-up time)
Clustering	Clustering of the index
Justification	Justification for the proposed index

SQL code

```
--Add tsvector column to post
ALTER TABLE post
ADD COLUMN tsvectors TSVECTOR;


--Create a function to automatically update ts_vectors
CREATE FUNCTION post_tsv_update() RETURNS TRIGGER AS $$
BEGIN
  IF TG_OP = 'INSERT' THEN
```

```
      NEW.tsvectors = to_tsvector('portuguese', NEW.description);
  END IF;
  IF TG_OP = 'UPDATE' THEN
      IF !

      END IF;
  END IF;
  RETURN NEW;
END
$$ LANGUAGE 'plpgsql';

--Create a trigger
CREATE TRIGGER post_tsv_update
BEFORE INSERT OR UPDATE ON post
FOR EACH ROW
EXECUTE PROCEDURE post_tsv_update();

--Create Index
CREATE INDEX tsv_idx ON post USING GIN(tsvectors);
```

3. Triggers

Trigger	TRIGGER01
Description	BR006 - Each user can only react and/or promote each post once.
SQL code	

```
-- Promotion
CREATE FUNCTION promotion_once() RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'UPDATE' THEN
       RAISE EXCEPTION 'Not possible to update this table';
    END IF;
    IF TG_OP = 'INSERT' THEN
       IF EXISTS (SELECT * FROM post_promotion WHERE id_account = NEW.id_account AND id_post=
           THEN RAISE EXCEPTION 'Already promoted by this user';
       END IF;
    END IF;
RETURN NEW;
END
$$ LANGUAGE plpgsql;


CREATE TRIGGER promotion_once
BEFORE INSERT OR UPDATE ON post_promotion
FOR EACH ROW
EXECUTE PROCEDURE promotion_once();

-- React
```

```
CRE
BEG
    IF TG_OP = 'UPDATE' THEN
       IF  NEW.id_account <> OLD.id_account OR NEW.id_post <> OLD.id_post THEN
           RAISE EXCEPTION 'Not possible to update id_account or id_post';
       END IF;
    END IF;
    IF TG_OP = 'INSERT' THEN
       IF EXISTS (SELECT * FROM post_reaction WHERE id_account = NEW.id_account AND id_post= N
           THEN RAISE EXCEPTION 'Already reacted by this user';
       END IF;
    END IF;
RETURN NEW;
END
$$ LANGUAGE plpgsql;


CREATE TRIGGER react_once
BEFORE INSERT OR UPDATE ON post_reaction
FOR EACH ROW
EXECUTE PROCEDURE react_once();
```

Trigger	TRIGGER02
Description	BR007 - Once a user befriends another user neither of them can resend a friend request to each other, any user can send a friend request to himself and if user "a" sends a friend request to user "b", then user "b" cannot send user "a" another friend request
SQL code	

```
-- Friend Request
CREATE FUNCTION friends_t() RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'UPDATE' THEN
        RAISE EXCEPTION 'Cannot update this table';
    END IF;

    IF TG_OP = 'INSERT' THEN
        IF NEW.id_sender = NEW.id_receiver
            THEN RAISE EXCEPTION 'Cannot request yourself';
        END IF;

        IF EXISTS (SELECT * FROM friend_request WHERE (id_sender = NEW.id_sender AND  id_recei
            THEN RAISE EXCEPTION 'Friend already requested';
        END IF;

        IF EXISTS (SELECT * FROM friendship WHERE (id_account_1 = NEW.id_account_1 AND  id_acc
```

```
                THEN RAISE EXCEPTION '2 users are friends';
        END IF;
    END IF;
RETURN NEW;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER friends_t
BEFORE INSERT OR UPDATE ON friend_request
FOR EACH ROW
EXECUTE PROCEDURE friends_t();
```

Trigger	TRIGGER03
Description	BR008 - Each group has at least one administrator
SQL code	

```
-- Community Administrator
CREATE FUNCTION administrator_t() RETURNS TRIGGER AS $$
BEGIN
    --UPDATE
    IF TG_OP = 'UPDATE' THEN
        IF NEW.status <> OLD.status AND OLD.status = 'admin' AND (SELECT COUNT(*) FROM relatio
            THEN RAISE EXCEPTION 'User is the only administrator';
        END IF;
    END IF;

    --DELETE
    IF TG_OP = 'DELETE' THEN
        IF OLD.status = 'admin' AND (SELECT COUNT(*) FROM relationship WHERE id_group = OLD.id
            THEN RAISE EXCEPTION 'User is the only administrator';
        END IF;
    END IF;
RETURN NEW;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER administrator_t
BEFORE INSERT ON relationship
FOR EACH ROW
EXECUTE PROCEDURE administrator_t();
```

4. Transactions

SQL Reference	Increasing comment_counter

SQL Reference	Increasing comment_counter
Justification	When a user decides to comment under a post this post's comment_counter is going to increase by 1. In order to keep the integrity of this variable we need a transaction to prevent simultaneous changes to it's value
Isolation level	Serializable
Complete SQL Code	

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

--comment on a poost
INSERT INTO post WHERE parent_post = NEW.parent_post_id
    VALUES(NEW.description, NEW.has_image, CURRENT_TIMESTAMP(2), CURRENT_TIMESTAMP(2), "0");
UPDATE post WHERE id_post = NEW.parent_post_id
    SET comments_count = comments_count + 1;

END TRANSACTION;
```

Annex A. SQL Code

A.1. Database schema

createdb.sql

```
--Index drops
DROP INDEX IF EXISTS tsv_idx;
DROP INDEX IF EXISTS post_owner_date_idx;
DROP INDEX IF EXISTS post_date_idx;

--Trigger drops
DROP TRIGGER IF EXISTS administrator_t on relationship;
DROP FUNCTION IF EXISTS administrator_t();

DROP TRIGGER IF EXISTS friends_t ON friend_request;
DROP FUNCTION IF EXISTS friends_t();

DROP TRIGGER IF EXISTS react_once ON post_reaction;
DROP FUNCTION IF EXISTS react_once();
```

```
DROP TRIGGER IF EXISTS promotion_once ON post_promotion;
DROP FUNCTION IF EXISTS promotion_once();

DROP TRIGGER IF EXISTS post_tsv_update ON post;
DROP FUNCTION IF EXISTS post_tsv_update();

--Table drops
DROP TABLE IF EXISTS friendship;
DROP TABLE IF EXISTS post_reaction;
DROP TABLE IF EXISTS post_promotion;
DROP TABLE IF EXISTS relationship;
DROP TABLE IF EXISTS post_report;
DROP TABLE IF EXISTS recovery_code;
DROP TABLE IF EXISTS notification;
DROP TABLE IF EXISTS friend_request;
DROP TABLE IF EXISTS account_report;
DROP TABLE IF EXISTS post;
DROP TABLE IF EXISTS community;
DROP TABLE IF EXISTS account;

--Tables

CREATE TABLE account (
    id_account SERIAL PRIMARY KEY,
    account_tag TEXT CONSTRAINT null_account_account_tag NOT NULL CONSTRAINT unique_account_ac
    password TEXT CONSTRAINT null_account_password NOT NULL,
    name TEXT CONSTRAINT null_account_name NOT NULL,
    age NUMERIC(3,0) CONSTRAINT null_account_age NOT NULL CONSTRAINT check_account_age CHECK (
    birthday DATE CONSTRAINT null_account_birthdate NOT NULL,
    is_private BOOLEAN CONSTRAINT null_account_is_private NOT NULL,
    email TEXT CONSTRAINT null_account_email NOT NULL CONSTRAINT unique_account_email UNIQUE,
    university TEXT CONSTRAINT null_account_university NOT NULL,
    course TEXT CONSTRAINT null_account_course NOT NULL,
    is_verified BOOLEAN CONSTRAINT null_account_verified NOT NULL,
    description TEXT,
    location TEXT,
    pronouns TEXT,
    is_admin BOOLEAN CONSTRAINT null_account_is_admin NOT NULL,
    is_blocked BOOLEAN CONSTRAINT null_account_is_blocked NOT NULL
);

CREATE TABLE community (
    id_community SERIAL PRIMARY KEY,
    name TEXT CONSTRAINT null_Community_name NOT NULL,
    description TEXT,
    is_public BOOLEAN CONSTRAINT null_Community_is_public NOT NULL
);

CREATE TABLE post (
    id_post SERIAL PRIMARY KEY,
    parent_post INTEGER REFERENCES post(id_post),
    owner_id INTEGER CONSTRAINT null_Post_owner NOT NULL REFERENCES account (id_account) ON DE
    group_id INTEGER CONSTRAINT null_Post_group REFERENCES community (id_community) ON DELETE
```

```
    description TEXT CONSTRAINT null_Post_description NOT NULL CONSTRAINT check_Post_descripti
    has_images BOOLEAN CONSTRAINT null_Post_has_images NOT NULL,
    publication_date TIMESTAMP(2) CONSTRAINT null_Post_date NOT NULL DEFAULT CURRENT_TIMESTAMP
    edited_date TIMESTAMP(2) CONSTRAINT check_Post_edited_date CHECK (edited_date <= CURRENT_T
    comments_count INTEGER CONSTRAINT null_Post_comments_count NOT NULL CONSTRAINT check_Post_
    is_visible BOOLEAN CONSTRAINT null_Post_is_private NOT NULL
);

CREATE TABLE account_report (
    id_report SERIAL PRIMARY KEY,
    reason INTEGER CONSTRAINT null_account_report_reason NOT NULL,
    description TEXT,
    id_account_reporting INTEGER CONSTRAINT null_account_id_account_reporting NOT NULL REFEREN
    id_account_reported INTEGER CONSTRAINT null_account_report_id_account_reported NOT NULL RE
);

CREATE TABLE friend_request (
    id_sender INTEGER CONSTRAINT null_Friend_request_id_sender NOT NULL REFERENCES account (id
    id_receiver INTEGER CONSTRAINT null_Friend_request_id_receiver NOT NULL REFERENCES account
    PRIMARY KEY (id_sender, id_receiver)
);

CREATE TABLE notification (
    id_notification SERIAL PRIMARY KEY,
    id_receiver INTEGER CONSTRAINT null_Notification_id_receiver NOT NULL REFERENCES account (
    url TEXT CONSTRAINT null_Notification_url NOT NULL,
    notification_date timestamp(2) CONSTRAINT null_Notification_date NOT NULL CONSTRAINT check
    description TEXT CONSTRAINT null_Notification_description NOT NULL,
    is_read BOOLEAN CONSTRAINT null_Notification_is_read NOT NULL
);

CREATE TABLE recovery_code (
    id_recovery_code SERIAL PRIMARY KEY,
    id_account INTEGER CONSTRAINT null_Recovery_code_id_account NOT NULL REFERENCES account (i
    code TEXT CONSTRAINT null_Recovery_code_code NOT NULL UNIQUE,
    valid_until TIMESTAMP(2) CONSTRAINT null_Recovery_code_valid_until NOT NULL CONSTRAINT che
);

CREATE TABLE post_report (
    id_report SERIAL PRIMARY KEY,
    id_post INTEGER CONSTRAINT null_Post_report_id_post NOT NULL REFERENCES post (id_post) ON
    reason INTEGER CONSTRAINT null_Post_report_reason NOT NULL,
    description TEXT
);

CREATE TABLE relationship (
    id_community INTEGER CONSTRAINT null_Relationship_id_community NOT NULL REFERENCES communi
    id_account INTEGER CONSTRAINT null_Relationship_account_id NOT NULL REFERENCES account (id
    status TEXT CONSTRAINT null_Relationship_status NOT NULL CONSTRAINT check_Relationship_sta
    PRIMARY KEY (id_community, id_account)
);

CREATE TABLE post_promotion (
```

```
    id_account INTEGER CONSTRAINT null_Post_promotion_id_account NOT NULL REFERENCES account (
    id_post INTEGER CONSTRAINT null_Post_promotion_id_post NOT NULL REFERENCES post (id_post)
    promotion_date TIMESTAMP(2) CONSTRAINT null_Post_promotion_date NOT NULL CONSTRAINT check_
    PRIMARY KEY (id_account, id_post)
);

CREATE TABLE post_reaction (
    id_account INTEGER CONSTRAINT null_Post_reaction_id_account NOT NULL REFERENCES account (i
    id_post INTEGER CONSTRAINT null_Post_reaction_id_post NOT NULL REFERENCES post (id_post) C
    react_date TIMESTAMP(2) CONSTRAINT null_Post_reaction_date NOT NULL CONSTRAINT check_Post_
    up_vote BOOLEAN CONSTRAINT null_Post_reaction_up_vote NOT NULL,
    PRIMARY KEY (id_account, id_post)
);

CREATE TABLE friendship (
    account1_id INTEGER CONSTRAINT null_Friendship_account1_id NOT NULL REFERENCES account (id
    account2_id INTEGER CONSTRAINT null_Friendship_account2_id NOT NULL REFERENCES account (id
    CHECK (account1_id <> account2_id),
    PRIMARY KEY (account1_id, account2_id)
);


--Triggers

--Add tsvector column to post
ALTER TABLE post
ADD COLUMN tsvectors TSVECTOR;

--Create a function to automatically update ts_vectors
CREATE FUNCTION post_tsv_update() RETURNS TRIGGER AS $$
BEGIN
  IF TG_OP = 'INSERT' THEN
      NEW.tsvectors = to_tsvector('portuguese', NEW.description);
  END IF;
  IF TG_OP = 'UPDATE' THEN
      IF NEW.description <> OLD.description THEN
          NEW.tsvectors = to_tsvector('portuguese',NEW.description);
      END IF;
  END IF;
  RETURN NEW;
END
$$ LANGUAGE 'plpgsql';

--Create a trigger
CREATE TRIGGER post_tsv_update
BEFORE INSERT OR UPDATE ON post
FOR EACH ROW
EXECUTE PROCEDURE post_tsv_update();


-- Promotion
CREATE FUNCTION promotion_once() RETURNS TRIGGER AS $$
BEGIN
```

```
    IF TG_OP = 'UPDATE' THEN

    E
    IF TG_OP =  'INSERT'  THEN
       IF EXISTS (SELECT * FROM post_promotion WHERE id_account = NEW.id_account AND id_post=
           THEN RAISE EXCEPTION 'Already promoted by this user';
       END IF;
    END IF;
RETURN NEW;
END
$$ LANGUAGE plpgsql;


CREATE TRIGGER promotion_once
BEFORE INSERT OR UPDATE ON post_promotion
FOR EACH ROW
EXECUTE PROCEDURE promotion_once();

-- React

CREATE FUNCTION react_once() RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'UPDATE' THEN
       IF  NEW.id_account <> OLD.id_account OR NEW.id_post <> OLD.id_post THEN
           RAISE EXCEPTION 'Not possible to update id_account or id_post';
       END IF;
    END IF;
    IF TG_OP = 'INSERT' THEN
       IF EXISTS (SELECT * FROM post_reaction WHERE id_account = NEW.id_account AND id_post= N
           THEN RAISE EXCEPTION 'Already reacted by this user';
       END IF;
    END IF;
RETURN NEW;
END
$$ LANGUAGE plpgsql;


CREATE TRIGGER react_once
BEFORE INSERT OR UPDATE ON post_reaction
FOR EACH ROW
EXECUTE PROCEDURE react_once();

-- Friend Request
CREATE FUNCTION friends_t() RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'UPDATE' THEN
        RAISE EXCEPTION 'Cannot update this table';
    END IF;

    IF TG_OP = 'INSERT' THEN
        IF NEW.id_sender = NEW.id_receiver
            THEN RAISE EXCEPTION 'Cannot request yourself';
        END IF;
```

```
        IF EXISTS (SELECT * FROM friend_request WHERE (id_sender = NEW.id_sender AND  id_recei
            THEN RAISE EXCEPTION 'Friend already requested';
        END IF;

        IF EXISTS (SELECT * FROM friendship WHERE (account1_id = NEW.id_sender AND  account2_i
                THEN RAISE EXCEPTION '2 users are friends';
        END IF;
    END IF;
RETURN NEW;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER friends_t
BEFORE INSERT OR UPDATE ON friend_request
FOR EACH ROW
EXECUTE PROCEDURE friends_t();


-- Group Administrator
CREATE FUNCTION administrator_t() RETURNS TRIGGER AS $$
BEGIN
    --UPDATE
    IF TG_OP = 'UPDATE' THEN
        IF (NEW.status <> OLD.status AND OLD.status = 'admin' AND (SELECT COUNT(*) FROM relati
            THEN RAISE EXCEPTION 'User is the only administrator';
        END IF;
    END IF;

    --DELETE
    IF TG_OP = 'DELETE' THEN
        IF (OLD.status = 'admin' AND (SELECT COUNT(*) FROM relationship WHERE id_community = O
            THEN RAISE EXCEPTION 'User is the only administrator';
        END IF;
    END IF;
RETURN NEW;
END
$$ LANGUAGE plpgsql;

CREATE TRIGGER administrator_t
BEFORE INSERT ON relationship
FOR EACH ROW
EXECUTE PROCEDURE administrator_t();




--Indexes

CREATE INDEX post_date_idx ON post USING btree(publication_date);

CREATE INDEX post_owner_date_idx ON post USING BTREE(owner_id, publication_date);
```

```
--Full text search trigger

CREATE INDEX tsv_idx ON post USING GIN(tsvectors);
```

A.2. Database population

Below there's a sample of the SQL population code, which is fully available in the populate.sql file available in the repository and in the **Annex A. Section** aswell

```
INSERT INTO account (account_tag, password, name, age, birthday, is_private, email, university
VALUES
 ('AvilaAndre', 'pg!password', 'André Ávila', 20, '2002-07-01', false, 'up202006767@edu.fe.up.
 ('rspencock0', '3MYlqie', 'Rickert Spencock', 16, '1983-06-03', false, 'rspencock0@java.com',
 ('fo1', 'IKYukrzIy', 'Felike O'' Liddy', 20, '1977-03-25', true, 'fo1@wikia.com', 'Wenzhou Un
 ('gwhilder2', 'RHgQE5FU', 'Ginny Whilder', 23, '1972-09-13', true, 'gwhilder2@technorati.com'
 ('nratt3', 'OWZlLLjIz', 'Nedda Ratt', 24, '1983-07-05', true, 'nratt3@weebly.com', 'Hogeschoo
 ('aroache4', 'HKBpHqT5', 'Alano Roache', 25, '1998-09-27', true, 'aroache4@redcross.org', 'Pa
 ('svidineev5', 'JodahnBzO9', 'Stephannie Vidineev', 26, '1973-10-12', false, 'svidineev5@pbs.
 ('babbatini6', 'GJs0uHzo', 'Bartholomeo Abbatini', 17, '1996-03-02', false, 'babbatini6@yelp.
...
```

Revision history

Changes made to the first submission: None so far.

GROUP2222, 23/10/2022

- André Ismael Ferraz Ávila, up202006767@edu.fe.up.pt
- Francisco Maria Lopes Pinto Pimentel Serra, up202007723@edu.fe.up.pt (Editor)
- Diogo Miguel Ferreira da Costa, up202007770@edu.fe.up.pt
- Manuel João Gomes Alves Amorim, up202007485@edu.fe.up.pt