EBD: Database Specification Component

The Super Legit Collaborative News (SLCN) is a project headed by a small group of developers with the main goal of free, open, and accessible news sharing for and by users.

This will allow all users to view and browse all types of news and comments on any topic, with access to text search and tag selection.

A4: Conceptual Data Model

The Conceptual Data Model artifact identifies and describes the entities and relations relevant to the database through the use of a UML diagram.

1. Class diagram

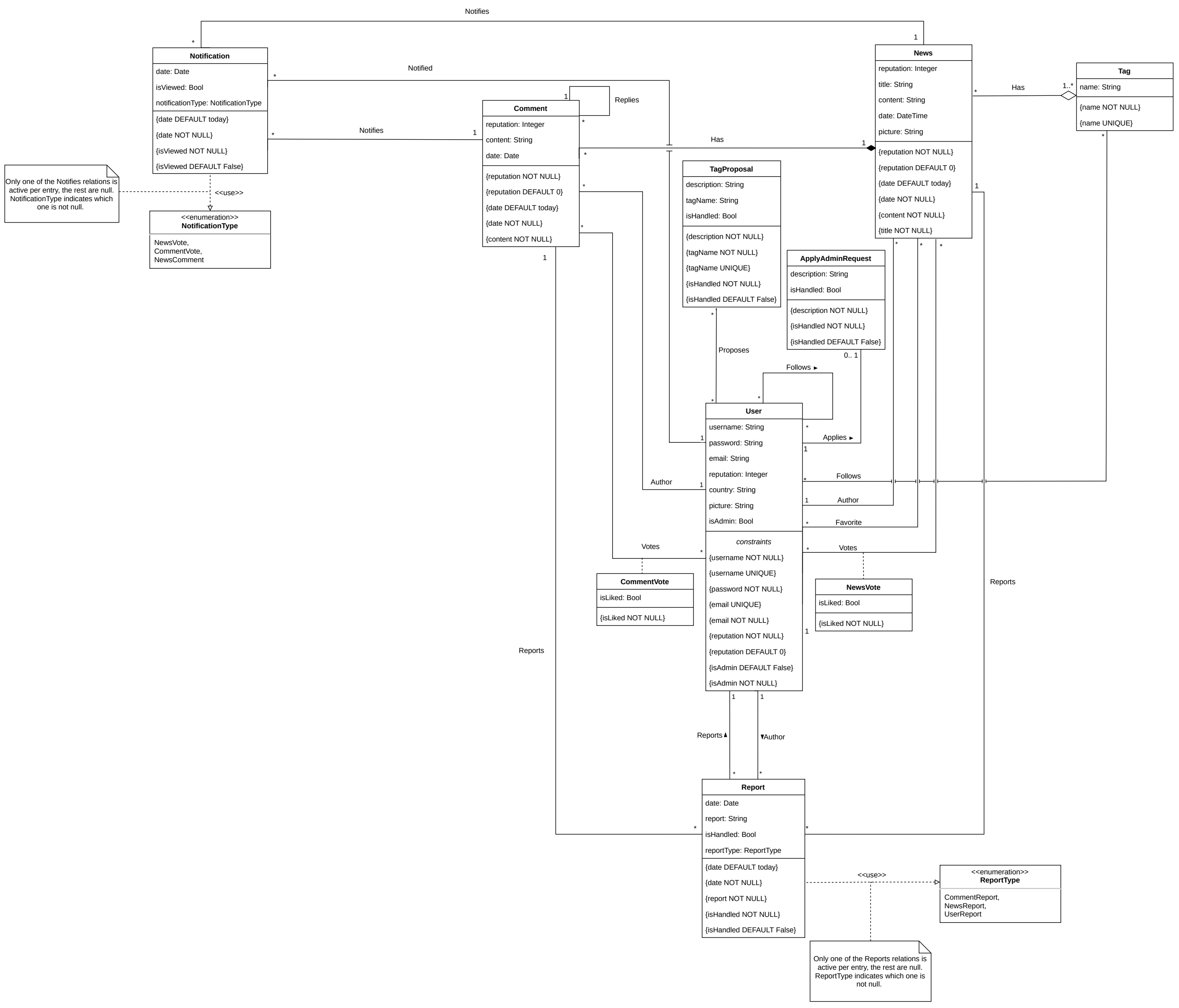

Figure 1: Conceptual Data Model

2. Additional Business Rules

- BR07: When a given tag is proposed for the first time, an entry in the TagProposal table and the many-to-many relation table associated with it is created for that tag. When there's another proposal for the same tag, the entry is only created for the many-to-many relation table.
- BR08: A comment can have comments as a reply but those replies can't have comments of their own.

A5: Relational Schema, validation and schema refinement

This artifact contains the Relational Schema created from the Conceptual Model UML. It includes attributes, domains, primary keys, foreign keys and restrictions like UNIQUE, DEFAULT and NOT NULL. It also includes the schema validations through functional depency analysis.

1. Relational Schema

Relation reference	**Relation Compact Notation**
R01	user (id, username **UK NN**, email **UK NN**, password **NN**, reputation **NN DF** 0, country, picture, is_admin **NN DF** False)
R02	follows(id1 -> user, id2 -> user)
R03	apply_admin_request(id, description **NN**, is_handled **NN DF** False, id_user -> user **UK NN**)
R04	news(id, reputation **NN DF** 0, title **NN**, content **NN**, date **NN DF** Today, picture, id_author -> user **NN**)
R05	news_favorite(id_user -> user, id_news -> news)
R06	news_vote(id_user -> user, id_news -> news, is_liked **NN**)
R07	news_tag(id_news -> news, id_tag -> tag)
R08	comment(id, reputation **NN DF** 0, content **NN**, date **NN DF** Today, id_news -> news **NN**, id_comment -> comment, id_author -> user **NN**)
R09	comment_vote(id_user -> user, id_comment -> comment, is_liked **NN**)
R10	tag(id, tag_name **UK NN**)
R11	tag_follow (id_user -> user, id_tag -> tag)
R12	tag_proposal(id, tag_name **UK NN**, description **NN**, is_handled **NN DF** False)
R13	tag_proposal_user (id_user -> user, id_tag -> tag_proposal)
R14	report(id_report, report_type **NN**, date **NN DF** Today, report_text **NN**, is_handled **NN DF** False, id_author -> user **NN**, id_user -> user, id_news -> news, id_comment -> comment)
R15	notification(id_notification, notification_type **NN**, date **NN DF** Today, is_viewed **NN DF** False, id_user -> user **NN**, id_news -> news, id_comment -> comment)

Legend:

- **UK** = UNIQUE KEY

- **NN** = NOT NULL
- **DF** = DEFAULT

2. Domains

Domain Name	**Domain Specification**
Today	DATE DEFAULT CURRENT_DATE
ReportType	ENUM('UserReport', 'NewsReport', 'CommentReport')
NotificationType	ENUM('NewsVote', 'CommentVote', 'NewsComment')

3. Schema validation

TABLE R01	**user**
Keys	{id}, {email}, {username}
Functional Dependencies:	
FD0101	{id} -> {username, email, password, country, picture, is_admin}
FD0102	{email} -> {id, username, password, reputation, country, picture, is_admin}
FD0103	{username} -> {id, email, password, reputation, country, picture, is_admin}
NORMAL FORM	BCNF

TABLE R02	**follows**
Keys	{id1, id2}
Functional Dependencies:	*none*
NORMAL FORM	BCNF

TABLE R03	**apply_admin_request**
Keys	{id}, {id_user}
Functional Dependencies:	
FD0401	{id} → {description, is_handled, id_user}
FD0402	{id_user} -> {id, description, is_handled}
NORMAL FORM	BCNF

TABLE R04	**news**
Keys	{id}
Functional Dependencies:	

TABLE R04	**news**
FD0501	{id} -> {reputation, title, content, date, picture, id_author}

NORMAL FORM BCNF

TABLE R05	**news_favorite**
Keys	{id_user, id_news}
Functional Dependencies:	*none*

NORMAL FORM BCNF

TABLE R06	**news_vote**
Keys	{id_user, id_news}
Functional Dependencies:	
FD0701	{id_user, id_news} -> {is_liked}

NORMAL FORM BCNF

TABLE R07	**news_tag**
Keys	{id_news, id_tag}
Functional Dependencies:	*none*

NORMAL FORM BCNF

TABLE R08	**comment**
Keys	{id}
Functional Dependencies:	
FD0901	{id} -> {reputation, content, date}

NORMAL FORM BCNF

TABLE R09	**comment_vote**
Keys	{id_user, id_comment}
Functional Dependencies:	
FD1001	{id_user, id_comment} -> {is_liked}

NORMAL FORM BCNF

TABLE R10	**tag**
Keys	{id}, {tag_name}
Functional Dependencies:	
FD1101	{id} -> {name}
FD1102	{tag_name} -> {id}

TABLE R10	**tag**
NORMAL FORM	BCNF

TABLE R11	**tag_follow**
Keys	{id_user, id_tag}
Functional Dependencies:	*none*
NORMAL FORM	BCNF

TABLE R12	**tag_proposal**
Keys	{id}, {tag_name}
Functional Dependencies:	
FD1201	{id} -> {tag_name, description, is_handled}
FD1202	{tag_name} -> {id, description, is_handled}
NORMAL FORM	BCNF

TABLE R13	**tag_proposal_user**
Keys	{id_user, id_tag}
Functional Dependencies:	*none*
NORMAL FORM	BCNF

TABLE R14	**report**
Keys	{id_report}
Functional Dependencies	
FD1401	{id_report} -> {report_type, date, report_text, is_handled, id_author, id_user, id_news, id_comment}
NORMAL FORM	BCNF

TABLE R15	**notification**
Keys	{id_notification}
Functional Dependencies:	
FD1501	{id_notification} -> {notification_type, date, is_viewed, id_user, id_news, id_comment}
NORMAL FORM	BCNF

Considering all the tables are in the BCNF, the Schema is in the BCNF.

A6: Indexes, triggers, transactions and database population

This artefact contains the Database Workload, the proposed indices, triggers and transactions we created for our database. There is also the complete database creation and population scripts, in the annex.

1. Database Workload

Relation reference	Relation Name	Order of magnitude	Estimated growth
R01	user	10 M	10 k / day
R02	follows	100 M	100 k / day
R03	apply_admin_request	1 k	10 / day
R04	news	1 M	10 k / day
R05	news_favorite	100 k	1 k / day
R06	news_vote	1 B	1 M / day
R07	news_tag	1 M	10 k / day
R08	comment	10 M	100 k / day
R09	comment_vote	100 M	1 M / day
R10	tag	100	1 / day
R11	tag_follow	10 M	10 k / day
R12	tag_proposal	10	1 / day
R13	tag_proposal_user	100 k	100 / day
R14	report	10 k	100 / day
R15	notification	1 B	1 M / day

2. Proposed Indices

2.1. Performance Indices

Index	**IDX01**
Relation	comment
Attribute	id_news
Type	Hash
Cardinality	Medium
Clustering	No

Index	IDX01
Justification	Table ‘comment’ is very large. Everytime we open a news, we need to filter access to the comments by its corresponding news. Filtering is done by exact match, thus an hash type index is best suited.

SQL code

```
CREATE INDEX news_comments ON comment USING hash (id_news);
```

Index	IDX02
Relation	news
Attribute	reputation
Type	B-tree
Cardinality	Medium
Clustering	No
Justification	Table ‘news’ is frequently accessed for news filtered by popularity (reputation). A B-tree index allows for faster order search queries based on the reputation.

SQL code

```
CREATE INDEX news_by_popularity ON news USING btree (reputation);
```

Index	IDX03
Relation	notification
Attribute	id_user
Type	Hash
Cardinality	High
Clustering	No
Justification	Table 'notification' is very large. Everytime a user sees his notifications, we need to filter access to the notifications by the user they correspond to. Filtering is done by exact match, thus an hash type index is best suited.

SQL code

```
CREATE INDEX user_notifications ON notification USING hash (id_user);
```

2.2. Full-text Search Indices

Index	**IDX01**
Relation	news
Attribute	title, content
Type	GiST
Clustering	No
Justification	To provide full-text search features to look for news based on matching titles or content. The index type is GiST because the indexed fields are not expected to change often.

SQL code

```
SET search_path TO lbaw2286;

ALTER TABLE news
ADD COLUMN tsvectors TSVECTOR;

CREATE FUNCTION news_search_update() RETURNS TRIGGER AS $$
BEGIN
 IF TG_OP = 'INSERT' THEN
        NEW.tsvectors = (
         setweight(to_tsvector('english', NEW.title), 'A') ||
         setweight(to_tsvector('english', NEW.content), 'B')
        );
 END IF;
 IF TG_OP = 'UPDATE' THEN
         IF (NEW.title <> OLD.title OR NEW.content <> OLD.content) THEN
           NEW.tsvectors = (
             setweight(to_tsvector('english', NEW.title), 'A') ||
             setweight(to_tsvector('english', NEW.content), 'B')
           );
         END IF;
 END IF;
 RETURN NEW;
END $$
LANGUAGE plpgsql;

CREATE TRIGGER news_search_update
 BEFORE INSERT OR UPDATE ON news
 FOR EACH ROW
 EXECUTE PROCEDURE news_search_update();

CREATE INDEX search_news ON news USING GiST (tsvectors);
```

3. Triggers

Trigger	TRIGGER01
Description	Trigger that updates comment and user reputation when a new vote is issued on the comment. BR01

SQL code

```
SET search_path TO lbaw2286;

CREATE FUNCTION add_comment_reputation() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF (NEW.is_liked) THEN
        UPDATE comment
        SET reputation = reputation+1
        WHERE id = NEW.id_comment;

        UPDATE users
        SET reputation = reputation+1
        WHERE id = (
            SELECT id_author FROM comment WHERE id = NEW.id_comment
        );
    ELSE
        UPDATE comment
        SET reputation = reputation-1
        WHERE id = NEW.id_comment;

        UPDATE users
        SET reputation = reputation-1
        WHERE id = (
            SELECT id_author FROM comment WHERE id = NEW.id_comment
        );

    END IF;
    RETURN NULL;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER add_comment_reputation
    AFTER INSERT ON comment_vote FOR EACH ROW
    EXECUTE PROCEDURE add_comment_reputation();
```

Trigger	TRIGGER02
Description	Trigger that updates news and user reputation when a new vote is issued on the news. BR01

SQL code

```
SET search_path TO lbaw2286;

CREATE FUNCTION add_news_reputation() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF (NEW.is_liked) THEN
        UPDATE news
        SET reputation = reputation+1
        WHERE id = NEW.id_news;

        UPDATE users
        SET reputation = reputation+1
        WHERE id = (
            SELECT id_author FROM news WHERE id = NEW.id_news
        );
    ELSE
        UPDATE news
        SET reputation = reputation-1
        WHERE id = NEW.id_news;

        UPDATE users
        SET reputation = reputation-1
        WHERE id = (
            SELECT id_author FROM news WHERE id = NEW.id_news
        );
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER add_news_reputation
    AFTER INSERT ON news_vote FOR EACH ROW
    EXECUTE PROCEDURE add_news_reputation();
```

Trigger	**TRIGGER03**
Description	Trigger that updates a user and comment reputation when a vote is removed from said comment. BR01

SQL code

```
SET search_path TO lbaw2286;

CREATE FUNCTION remove_comment_reputation() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF (OLD.is_liked) THEN
        UPDATE comment
        SET reputation = reputation-1
        WHERE id = OLD.id_comment;

        UPDATE users
        SET reputation = reputation-1
        WHERE id = (
            SELECT id_author FROM comment WHERE id = NEW.id_comment
        );
    ELSE
        UPDATE comment
        SET reputation = reputation+1
        WHERE id = OLD.id_comment;

        UPDATE users
        SET reputation = reputation+1
        WHERE id = (
            SELECT id_author FROM comment WHERE id = NEW.id_comment
        );
    END IF;
    RETURN NULL;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER remove_comment_reputation
    BEFORE DELETE ON comment_vote FOR EACH ROW
    EXECUTE PROCEDURE remove_comment_reputation();
```

Trigger	**TRIGGER04**
Description	Trigger that updates a user and news reputation when a vote is removed from said news. BR01

SQL code

```
SET search_path TO lbaw2286;

CREATE FUNCTION remove_news_reputation() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF (OLD.is_liked) THEN
        UPDATE news
        SET reputation = reputation-1
        WHERE id = OLD.id_news;

        UPDATE users
        SET reputation = reputation-1
        WHERE id = (
            SELECT id_author FROM news WHERE id = NEW.id_news
        );
    ELSE
```

```
        UPDATE news
        SET reputation = reputation+1
        WHERE id = OLD.id_news;

        UPDATE users
        SET reputation = reputation+1
        WHERE id = (
            SELECT id_author FROM news WHERE id = NEW.id_news
        );
    END IF;
    RETURN NULL;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER remove_news_reputation
    BEFORE DELETE ON news_vote FOR EACH ROW
    EXECUTE PROCEDURE remove_news_reputation();
```

Trigger	**TRIGGER05**
Description	Trigger that updates a user and comment reputation when a vote is updated on said comment. BR01

SQL code

```
SET search_path TO lbaw2286;

CREATE FUNCTION update_comment_reputation() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF (NEW.is_liked) THEN
        UPDATE comment
        SET reputation = reputation+2
        WHERE id = NEW.id_comment;

        UPDATE users
        SET reputation = reputation+2
        WHERE id = (
            SELECT id_author FROM comment WHERE id = NEW.id_comment
        );
    ELSE
        UPDATE comment
        SET reputation = reputation-2
        WHERE id = NEW.id_comment;

        UPDATE users
        SET reputation = reputation-2
        WHERE id = (
            SELECT id_author FROM comment WHERE id = NEW.id_comment
        );
    END IF;
    RETURN NULL;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER update_comment_reputation
    AFTER UPDATE ON comment_vote FOR EACH ROW
    EXECUTE PROCEDURE update_comment_reputation();
```

Trigger	TRIGGER06
Description	Trigger that updates a user and news reputation when a vote is updated on said news. BR01

SQL code

```
SET search_path TO lbaw2286;

CREATE FUNCTION update_news_reputation() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF (NEW.is_liked) THEN
        UPDATE news
        SET reputation = reputation+2
        WHERE id = NEW.id_news;

        UPDATE users
        SET reputation = reputation+2
        WHERE id = (
            SELECT id_author FROM news WHERE id = NEW.id_news
        );
    ELSE
        UPDATE news
        SET reputation = reputation-2
        WHERE id = NEW.id_news;

        UPDATE users
        SET reputation = reputation-2
        WHERE id = (
            SELECT id_author FROM news WHERE id = NEW.id_news
        );
    END IF;
    RETURN NULL;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER update_news_reputation
    AFTER UPDATE ON news_vote FOR EACH ROW
    EXECUTE PROCEDURE update_news_reputation();
```

Trigger	TRIGGER07
Description	Trigger that replaces user data with anonymous data on user account deletion. BR03

SQL code

```
SET search_path TO lbaw2286;

-- user id 5 is anonymous

CREATE FUNCTION anonymous_user() RETURNS TRIGGER AS
$BODY$
BEGIN
        UPDATE news SET id_author=5 WHERE OLD.id = id_author;
        UPDATE comment SET id_author=5 WHERE OLD.id = id_author;
        UPDATE apply_admin_request SET id_user = 5 WHERE OLD.id = id_user;
        RETURN NULL;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER anonymous_user
        BEFORE DELETE ON users
        FOR EACH ROW
        EXECUTE PROCEDURE anonymous_user();
```

Trigger	TRIGGER08
Description	Trigger that garantees that a comment can't be a reply to another reply. BR08

SQL code

```
SET search_path TO lbaw2286;

CREATE FUNCTION comment_on_comment() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF EXISTS (select id_comment from comment where id_comment = NEW.id)
THEN-- se comentário já for resposta a comentário não pode ser comentado
        RAISE EXCEPTION 'Comments that are commented on other comment cant
have comments';
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER comment_on_comment
        BEFORE INSERT ON comment
        FOR EACH ROW
        EXECUTE PROCEDURE comment_on_comment();
```

Trigger	TRIGGER09
Description	Trigger that garantees that a comment can't be deleted if it has replies or votes. BR02

SQL code

```
SET search_path TO lbaw2286;

CREATE FUNCTION delete_comment() RETURNS TRIGGER AS
$BODY$
BEGIN
       IF EXISTS (SELECT * FROM comment WHERE id_comment = OLD.id ) THEN
            RAISE EXCEPTION 'You cant delete a comment with comments in
it';
    END IF;
        IF NOT (OLD.reputation = 0) THEN
            RAISE EXCEPTION 'You cant delete a comment with votes in it.';
        END IF;
    RETURN NULL;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER delete_comment
        BEFORE DELETE ON comment
        FOR EACH ROW
        EXECUTE PROCEDURE delete_comment();
```

Trigger	TRIGGER10
Description	Trigger that garantees that a news item can't be deleted if it has comments or votes. BR02

SQL code

```
SET search_path TO lbaw2286;

CREATE FUNCTION delete_news() RETURNS TRIGGER AS
$BODY$
BEGIN
    IF EXISTS (SELECT * FROM comment WHERE id_news = OLD.id ) THEN
        RAISE EXCEPTION 'You cant delete news with comments in it';
    END IF;
    IF NOT (OLD.reputation = 0) THEN
        RAISE EXCEPTION 'You cant delete news with votes in it';
    END IF;
    RETURN NEW;
END
$BODY$
LANGUAGE plpgsql;

CREATE TRIGGER delete_news
        BEFORE DELETE ON news
        FOR EACH ROW
        EXECUTE PROCEDURE delete_news();
```

4. Transactions

SQL Reference	**newstag**
Justification	When news are created, news_tag entries are also created to associate the news and the chosen tags. In the middle of the tansaction, new rows can be inserted in the news table, which could result in currval returning a wrong id. To prevent these non-repeatable reads, we chose isolation level Repeatable Read.
Isolation level	REPEATABLE READ

Complete SQL Code

```
SET search_path TO lbaw2286;

BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-- Insert news
INSERT INTO news(title, content, picture, id_author) VALUES ($title,
$content, $picture, $id_author);

-- Insert news_tag
INSERT INTO news_tag(id_news, id_tag) VALUES (currval('news_id_seq'),
$id_tag);

END TRANSACTION;
```

SQL Reference	**tagproposal**

SQL Reference	**tagproposal**
Justification	When a tag is proposed for the first time, an entry on the tag_proposal table is created, containing the tag information. It is also necessary to create an entry in the tag_proposal_user to associate the user to its proposal. In the middle of the tansaction, new rows can be inserted in the tag_proposal table, which could result in currval returning a wrong id. To prevent these non-repeatable reads, we chose isolation level Repeatable Read.
Isolation level	REPEATABLE READ

SQL code

```sql
SET search_path TO lbaw2286;

BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

-- Insert tag_proposal
INSERT INTO tag_proposal (tag_name, description) VALUES ($tag_name,
$description);

-- Insert tag_proposal_user
INSERT INTO tag_proposal_user (id_user, id_tag) VALUES ($id_user,
currval('tag_proposal_id_seq'));

END TRANSACTION;
```

Annex A. SQL Code

A.1. Database schema

The full database schema creation script can be found here.

SQL code

```sql
-- SCHEMA: lbaw2286

DROP SCHEMA IF EXISTS lbaw2286 CASCADE;

CREATE SCHEMA IF NOT EXISTS lbaw2286
    AUTHORIZATION postgres;


SET search_path TO lbaw2286;
```

```
-----------------------------------------
-- Drop old schema
-----------------------------------------

DROP TABLE IF EXISTS users CASCADE; --R01
DROP TABLE IF EXISTS follows CASCADE; --R02
DROP TABLE IF EXISTS apply_admin_request CASCADE; --R03
DROP TABLE IF EXISTS news CASCADE; --R04
DROP TABLE IF EXISTS news_favorite CASCADE; --R05
DROP TABLE IF EXISTS news_vote CASCADE; --R06
DROP TABLE IF EXISTS news_tag CASCADE; --R07
DROP TABLE IF EXISTS comment CASCADE; --R08
DROP TABLE IF EXISTS comment_vote CASCADE; --R09
DROP TABLE IF EXISTS tag CASCADE; --R10
DROP TABLE IF EXISTS tag_follow CASCADE; --R11
DROP TABLE IF EXISTS tag_proposal CASCADE; --R12
DROP TABLE IF EXISTS tag_proposal_user CASCADE; --R13
DROP TABLE IF EXISTS report CASCADE; --R14
DROP TABLE IF EXISTS notification CASCADE; --R15

-----------------------------------------
-- Types
-----------------------------------------
DROP TYPE IF EXISTS ReportType CASCADE;
DROP TYPE IF EXISTS NotificationType CASCADE;

CREATE TYPE ReportType AS ENUM ('UserReport', 'NewsReport',
'CommentReport');
CREATE TYPE NotificationType AS ENUM ('NewsVote', 'CommentVote',
'NewsComment');

-----------------------------------------
-- Tables
-----------------------------------------

-- Note that a plural 'users' name was adopted because user is a reserved
word in PostgreSQL.

--R01
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username TEXT NOT NULL UNIQUE,
    email TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL,
    reputation INTEGER NOT NULL DEFAULT 0,
    country TEXT,
    picture TEXT,
    isAdmin BOOLEAN NOT NULL
);

--R02
CREATE TABLE follows (
    id1 INTEGER NOT NULL REFERENCES users (id) ON UPDATE CASCADE ON DELETE
```

```
CASCADE,
    id2 INTEGER NOT NULL REFERENCES users (id) ON UPDATE CASCADE ON DELETE
CASCADE,
    PRIMARY KEY (id1, id2)
);

--R03
CREATE TABLE apply_admin_request (
    id SERIAL PRIMARY KEY,
    id_user INTEGER NOT NULL REFERENCES users (id) ON UPDATE CASCADE,
    description TEXT NOT NULL,
    is_handled BOOL NOT NULL DEFAULT False
);

--R10
CREATE TABLE tag(
    id SERIAL PRIMARY KEY,
    tag_name TEXT UNIQUE NOT NULL
);

--R04
CREATE TABLE news (
    id SERIAL PRIMARY KEY,
    reputation INTEGER NOT NULL DEFAULT 0,
    title TEXT NOT NULL,
    content TEXT NOT NULL,
    date TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL,
    picture TEXT,
    id_author INTEGER NOT NULL REFERENCES users (id) ON UPDATE CASCADE
);

--R05
CREATE TABLE news_favorite (
    id_user INTEGER NOT NULL REFERENCES users (id) ON UPDATE CASCADE ON
DELETE CASCADE,
    id_news INTEGER NOT NULL REFERENCES news (id) ON UPDATE CASCADE ON
DELETE CASCADE,
    PRIMARY KEY (id_user, id_news)
);

--R06
CREATE TABLE news_vote (
    id_user INTEGER NOT NULL REFERENCES users (id) ON UPDATE CASCADE ON
DELETE CASCADE,
    id_news INTEGER NOT NULL REFERENCES news (id) ON UPDATE CASCADE,
    is_liked BOOL NOT NULL,
    PRIMARY KEY (id_user, id_news)
);

--R07
CREATE TABLE news_tag (
    id_news INTEGER NOT NULL REFERENCES news (id) ON UPDATE CASCADE ON
DELETE CASCADE,
    id_tag INTEGER NOT NULL REFERENCES tag (id) ON UPDATE CASCADE ON DELETE
```

```
CASCADE,
    PRIMARY KEY (id_news, id_tag)
);

--R08
CREATE TABLE comment (
    id SERIAL PRIMARY KEY,
    reputation INTEGER NOT NULL DEFAULT 0,
    content TEXT NOT NULL,
    date TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL,
    id_news INTEGER NOT NULL REFERENCES news (id) ON UPDATE CASCADE,
    id_author INTEGER NOT NULL REFERENCES users (id) ON UPDATE CASCADE,
    id_comment INTEGER REFERENCES comment (id) ON UPDATE CASCADE
);

--R09
CREATE TABLE comment_vote (
    id_user INTEGER NOT NULL REFERENCES users (id) ON UPDATE CASCADE ON
DELETE CASCADE,
    id_comment INTEGER NOT NULL REFERENCES comment (id) ON UPDATE CASCADE,
    is_liked BOOL NOT NULL,
    PRIMARY KEY (id_user, id_comment)
);

--R11
CREATE TABLE tag_follow (
    id_user INTEGER NOT NULL REFERENCES users (id) ON UPDATE CASCADE ON
DELETE CASCADE,
    id_tag INTEGER NOT NULL REFERENCES tag (id) ON UPDATE CASCADE ON DELETE
CASCADE,
    PRIMARY KEY (id_user, id_tag)
);

--R12
CREATE TABLE tag_proposal (
    id SERIAL PRIMARY KEY,
    tag_name TEXT UNIQUE NOT NULL,
    description TEXT NOT NULL,
    is_handled BOOLEAN DEFAULT False
);

--R13
CREATE TABLE tag_proposal_user (
    id_user INTEGER NOT NULL REFERENCES users (id) ON UPDATE CASCADE ON
DELETE CASCADE,
    id_tag INTEGER NOT NULL REFERENCES tag_proposal (id) ON UPDATE CASCADE,
    PRIMARY KEY (id_user, id_tag)
);

--R14
CREATE TABLE report (
    id_report SERIAL PRIMARY KEY,
    report_type ReportType NOT NULL,
    date TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL,
    report_text TEXT,
    is_handled BOOLEAN DEFAULT False,
    id_author INTEGER NOT NULL REFERENCES users (id) ON UPDATE CASCADE,
    id_user INTEGER REFERENCES users (id) ON UPDATE CASCADE,
    id_news INTEGER REFERENCES news (id) ON UPDATE CASCADE,
    id_comment INTEGER REFERENCES comment (id) ON UPDATE CASCADE,
    CHECK((id_user IS NOT NULL AND id_news IS NULL AND id_comment IS NULL)
OR (id_user IS NULL AND id_news IS NOT NULL AND id_comment IS NULL) OR
(id_user IS NULL AND id_news IS NULL AND id_comment IS NOT NULL))
);

--R15
CREATE TABLE notification (
    id_notification SERIAL PRIMARY KEY,
    notification_type NotificationType NOT NULL,
    date TIMESTAMP WITH TIME ZONE DEFAULT now() NOT NULL,
    is_viewed BOOLEAN NOT NULL DEFAULT False,
    id_user INTEGER NOT NULL REFERENCES users (id) ON UPDATE CASCADE,
    id_news INTEGER REFERENCES news (id) ON UPDATE CASCADE,
    id_comment INTEGER REFERENCES comment (id) ON UPDATE CASCADE,
    CHECK((id_news IS NOT NULL AND id_comment IS NULL) OR (id_news IS NULL
AND id_comment IS NOT NULL))
);
```

A.2. Database population

The full database schema population script can be found here.

SQL code

```
-------------------------------
-- Users
-------------------------------

SET search_path TO lbaw2286;

INSERT INTO users (id, username, email, password, country, picture,
isAdmin) VALUES(1, 'André Morais', 'andre@legitmail.com', 'legitandre',
'Portugal', './path/to/picture.png', true);
INSERT INTO users (id, username, email, password, country, picture,
isAdmin) VALUES(2, 'João Teixeira', 'joao@legitmail.com', 'legitjoao',
'Portugal', './path/to/picture.png', true);
INSERT INTO users (id, username, email, password, country, picture,
isAdmin) VALUES(3, 'Lucas Sousa', 'lucas@legitmail.com', 'legitlucas',
'Portugal', './path/to/picture.png', true);
INSERT INTO users (id, username, email, password, country, picture,
isadmin) VALUES(4, 'Rui Soares', 'rui@legitmail.com', 'legitrui',
'Portugal', './path/to/picture.png', true);
INSERT INTO users (id, username, email, password, country, picture,
isAdmin) VALUES(5, '[redacted]', 'redac@legitmail.com', 'legitredac',
```

```sql
'Zimbabue', './path/to/default.png', false); --id 5 is deleted user


-------------------------------
-- Follows
-------------------------------
INSERT INTO follows (id1, id2) VALUES (1, 2);
INSERT INTO follows (id1, id2) VALUES (1, 3);
INSERT INTO follows (id1, id2) VALUES (1, 4);
INSERT INTO follows (id1, id2) VALUES (2, 1);
INSERT INTO follows (id1, id2) VALUES (2, 3);
INSERT INTO follows (id1, id2) VALUES (2, 4);
INSERT INTO follows (id1, id2) VALUES (3, 1);
INSERT INTO follows (id1, id2) VALUES (3, 2);

-------------------------------
-- Apply admin request
-------------------------------
INSERT INTO apply_admin_request(description, is_handled, id_user) VALUES
('I would like to be an admin to help manage news',false,1);
INSERT INTO apply_admin_request(description, is_handled, id_user) VALUES
('I would like to be an admin please!',true,2);
INSERT INTO apply_admin_request(description, is_handled, id_user) VALUES
('I would like to be an admin to manage reports',false,3);
INSERT INTO apply_admin_request(description, is_handled, id_user) VALUES
('I would like to be an admin to help manage ags',false,4);

-------------------------------
-- tag
-------------------------------
INSERT INTO tag(id, tag_name) VALUES (1, 'Gaming'); -- 1
INSERT INTO tag(id, tag_name) VALUES (2, 'Politics'); -- 2
INSERT INTO tag(id,tag_name) VALUES (3, 'Academia'); -- 3
INSERT INTO tag(id, tag_name) VALUES (4, 'Memes'); -- 4
INSERT INTO tag(id, tag_name) VALUES (5, 'Food'); -- 5
INSERT INTO tag(id, tag_name) VALUES (6, 'Animals'); -- 6
INSERT INTO tag(id, tag_name) VALUES (7, 'Celebrities'); -- 7
INSERT INTO tag(id, tag_name) VALUES (8, 'Movies'); -- 8
INSERT INTO tag(id, tag_name) VALUES (9, 'TV'); -- 9
INSERT INTO tag(id, tag_name) VALUES (10, 'Books'); -- 10
INSERT INTO tag(id, tag_name) VALUES (11, 'Technology'); -- 11
INSERT INTO tag(id, tag_name) VALUES (12, 'Hardware'); -- 12
INSERT INTO tag(id, tag_name) VALUES (13, 'Software'); -- 13
INSERT INTO tag(id, tag_name) VALUES (14, 'Sci-Fi'); -- 14
INSERT INTO tag(id, tag_name) VALUES (15, 'Fantasy'); -- 15
INSERT INTO tag(id, tag_name) VALUES (16, 'Sports'); -- 16
INSERT INTO tag(id, tag_name) VALUES (17, 'Photography'); -- 17
INSERT INTO tag(id, tag_name) VALUES (18, 'Science'); -- 18
INSERT INTO tag(id, tag_name) VALUES (19, 'DIY'); -- 19
INSERT INTO tag(id, tag_name) VALUES (20, 'Music'); -- 20
INSERT INTO tag(id, tag_name) VALUES (21, 'Anime'); -- 21

-------------------------------
-- News
```

```sql
-------------------------------
INSERT INTO news (id, title, content, date, picture, id_author) VALUES (1,
'Overwatch Fan Makes LEGO Bastion Figure for Their Brother', 'Lorem ipsum
dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Euismod
lacinia at quis risus sed vulputate odio ut.
Dignissim convallis aenean et tortor. Eu feugiat pretium nibh ipsum
consequat nisl. Interdum consectetur libero id faucibus.
Erat velit scelerisque in dictum non consectetur a.', '2022.10.20',
'./path/to/picture.png', 1);

INSERT INTO news (id, title, content, date, picture, id_author) VALUES (2,
'Here’s What to Expect from Season 3 of The Witcher', 'Lorem ipsum dolor
sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Euismod
lacinia at quis risus sed vulputate odio ut.
Dignissim convallis aenean et tortor. Eu feugiat pretium nibh ipsum
consequat nisl. Interdum consectetur libero id faucibus.
Erat velit scelerisque in dictum non consectetur a.', '2022.10.20',
'./path/to/picture.png', 2);

INSERT INTO news (id, title, content, date, picture, id_author) VALUES (3,
'The State Of Destiny 2s Festival Of The Lost Is Unacceptable','Lorem ipsum
dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Euismod
lacinia at quis risus sed vulputate odio ut.
Dignissim convallis aenean et tortor. Eu feugiat pretium nibh ipsum
consequat nisl. Interdum consectetur libero id faucibus.
Erat velit scelerisque in dictum non consectetur a.', '2022.10.20',
'./path/to/picture.png', 3);

INSERT INTO news (id, title, content, date, picture, id_author) VALUES (4,
'Bleach TYBW shocks fans with brutal character deaths in episode 2', 'Lorem
ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Euismod
lacinia at quis risus sed vulputate odio ut.
Dignissim convallis aenean et tortor. Eu feugiat pretium nibh ipsum
consequat nisl. Interdum consectetur libero id faucibus.
Erat velit scelerisque in dictum non consectetur a.', '2022.10.20',
'./path/to/picture.png', 4);

-------------------------------
-- news_favorite
-------------------------------
INSERT INTO news_favorite(id_user, id_news) VALUES (1, 1);
INSERT INTO news_favorite(id_user, id_news) VALUES (2, 2);
INSERT INTO news_favorite(id_user, id_news) VALUES (3, 4);
INSERT INTO news_favorite(id_user, id_news) VALUES (4, 3);

-------------------------------
-- news_vote
-------------------------------
INSERT INTO news_vote(id_user, id_news, is_liked) VALUES (1, 1, true);
INSERT INTO news_vote(id_user, id_news, is_liked) VALUES (1, 2, false);
INSERT INTO news_vote(id_user, id_news, is_liked) VALUES (2, 2, true);
INSERT INTO news_vote(id_user, id_news, is_liked) VALUES (2, 3, false);
INSERT INTO news_vote(id_user, id_news, is_liked) VALUES (3, 1, true);
INSERT INTO news_vote(id_user, id_news, is_liked) VALUES (3, 4, false);

-------------------------------
-- news_tag
-------------------------------
INSERT INTO news_tag (id_news, id_tag) VALUES (1, 1); -- gaming
INSERT INTO news_tag (id_news, id_tag) VALUES (2, 9); -- TV
INSERT INTO news_tag (id_news, id_tag) VALUES (2, 7); -- Celebrities
INSERT INTO news_tag (id_news, id_tag) VALUES (2, 1); -- Gaming
INSERT INTO news_tag (id_news, id_tag) VALUES (3, 1); -- Gaming
INSERT INTO news_tag (id_news, id_tag) VALUES (4, 8); -- Movies
INSERT INTO news_tag (id_news, id_tag) VALUES (4, 9); -- TV
INSERT INTO news_tag (id_news, id_tag) VALUES (4, 21); -- Anime

-------------------------------
-- comment
-------------------------------
INSERT INTO comment(id, content, id_news, id_comment, id_author) VALUES (1,
'Fake news!', 1, NULL, 1);
INSERT INTO comment(id, content, id_news, id_comment, id_author) VALUES (2,
'Very informative', 2, NULL, 2);
INSERT INTO comment(id, content, id_news, id_comment, id_author) VALUES (3,
'Loved it!', 2, 1, 3);
INSERT INTO comment(id, content, id_news, id_comment, id_author) VALUES (4,
'Source?', 3, 2, 4);

-------------------------------
-- comment_vote
-------------------------------
INSERT INTO comment_vote(id_user, id_comment, is_liked) VALUES (1, 1,
true);
INSERT INTO comment_vote(id_user, id_comment, is_liked) VALUES (1, 2,
false);
INSERT INTO comment_vote(id_user, id_comment, is_liked) VALUES (1, 3,
true);
INSERT INTO comment_vote(id_user, id_comment, is_liked) VALUES (2, 2,
true);
INSERT INTO comment_vote(id_user, id_comment, is_liked) VALUES (3, 1,
false);
INSERT INTO comment_vote(id_user, id_comment, is_liked) VALUES (4, 1,
true);

-------------------------------
-- tag_follow
-------------------------------
INSERT INTO tag_follow(id_user, id_tag) VALUES (1,1);
INSERT INTO tag_follow(id_user, id_tag) VALUES (2,2);
INSERT INTO tag_follow(id_user, id_tag) VALUES (3,3);
INSERT INTO tag_follow(id_user, id_tag) VALUES (4,4);
INSERT INTO tag_follow(id_user, id_tag) VALUES (1,4);
INSERT INTO tag_follow(id_user, id_tag) VALUES (2,5);
```

```
-------------------------------
-- tag_proposal
-------------------------------
INSERT INTO tag_proposal(tag_name, description, is_handled) VALUES
('Wholesome','I want to tag my cat pictures',false);
INSERT INTO tag_proposal(tag_name, description, is_handled) VALUES
('Mystery','I want to tag some books with this tag',false);
INSERT INTO tag_proposal(tag_name, description, is_handled) VALUES
('Manga','I want to tag my favorite manga without using the "books"
tag',false);
INSERT INTO tag_proposal(tag_name, description, is_handled) VALUES
('Cars','This important tag is missing',false);
INSERT INTO tag_proposal(tag_name, description, is_handled) VALUES
('Anime','I want to tag my favorite anime shows without using the "TV"
tag', true);

---------------------INSERT INTO report(report_type, report_text,
is_handled, id_author, id_user, id_news, id_comment) VALUES
('UserReport','User insulted me', false,1,1,NULL, NULL);
----------
-- tag_proposal_user
-------------------------------
INSERT INTO tag_proposal_user(id_user, id_tag) VALUES (1, 1);
INSERT INTO tag_proposal_user(id_user, id_tag) VALUES (2, 1);
INSERT INTO tag_proposal_user(id_user, id_tag) VALUES (3, 1);
INSERT INTO tag_proposal_user(id_user, id_tag) VALUES (1, 2);
INSERT INTO tag_proposal_user(id_user, id_tag) VALUES (2, 2);
INSERT INTO tag_proposal_user(id_user, id_tag) VALUES (3, 3);
INSERT INTO tag_proposal_user(id_user, id_tag) VALUES (4, 3);

-------------------------------
-- report --
--'UserReport', 'NewsReport', 'CommentReport'
-------------------------------
INSERT INTO report(report_type, report_text, is_handled, id_author,
id_user, id_news, id_comment) VALUES ('UserReport','User insulted me',
false,1,1,NULL, NULL);
INSERT INTO report(report_type, report_text, is_handled, id_author,
id_user, id_news, id_comment) VALUES ('NewsReport','Wrong use of tags',
true,2,NULL,2, NULL);
INSERT INTO report(report_type, report_text, is_handled, id_author,
id_user, id_news, id_comment) VALUES ('CommentReport','Offensive comment',
false,3,NULL,NULL,1);
INSERT INTO report(report_type, report_text, is_handled, id_author,
id_user, id_news, id_comment) VALUES ('CommentReport','Spam', false,4,
NULL, NULL, 2);

-------------------------------
-- notification
-- 'NewsVote', 'CommentVote', 'NewsComment'
-------------------------------
INSERT INTO notification(notification_type, is_viewed, id_user, id_news,
id_comment) VALUES ('NewsVote', false, 1, 1, NULL);
```

```
INSERT INTO notification(notification_type, is_viewed, id_user, id_news,
id_comment) VALUES ('CommentVote', false, 2, NULL, 3);
INSERT INTO notification(notification_type, date, is_viewed, id_user,
id_news, id_comment) VALUES ('NewsComment', '2022.10.20', true, 2, NULL,
1);
INSERT INTO notification(notification_type, date, is_viewed, id_user,
id_news, id_comment) VALUES ('NewsVote', '2022.10.20', true, 4, 4, NULL);
```

Revision history

First Submission

- A4: Conceptual Data Model
- A5: Relational Schema, validation and schema refinement
- A6: Indexes, triggers, transactions and database population

---

- André Morais, up202005303@edu.fe.up.pt (editor)
- João Teixeira, up202005437@edu.fe.up.pt
- Lucas Sousa, up202004682@edu.fe.up.pt
- Rui Soares, up202103631@edu.fe.up.pt

lbaw2223-t8g6, 29/10/22