# EBD: Database Specification Component

Collaborative News is a web platform for news sharing where you can also interact with other users.

## A4: Conceptual Data Model

The purpose of this Conceptual Data Model is to provide a high level view of the static aspects and structures of our application. It shows the classes of our system and the main relationships between them. It's usefull when implementing other aspects of the system, such as the database and during it's inception system flaws can be discovered and corrected.

### 1. Class diagram

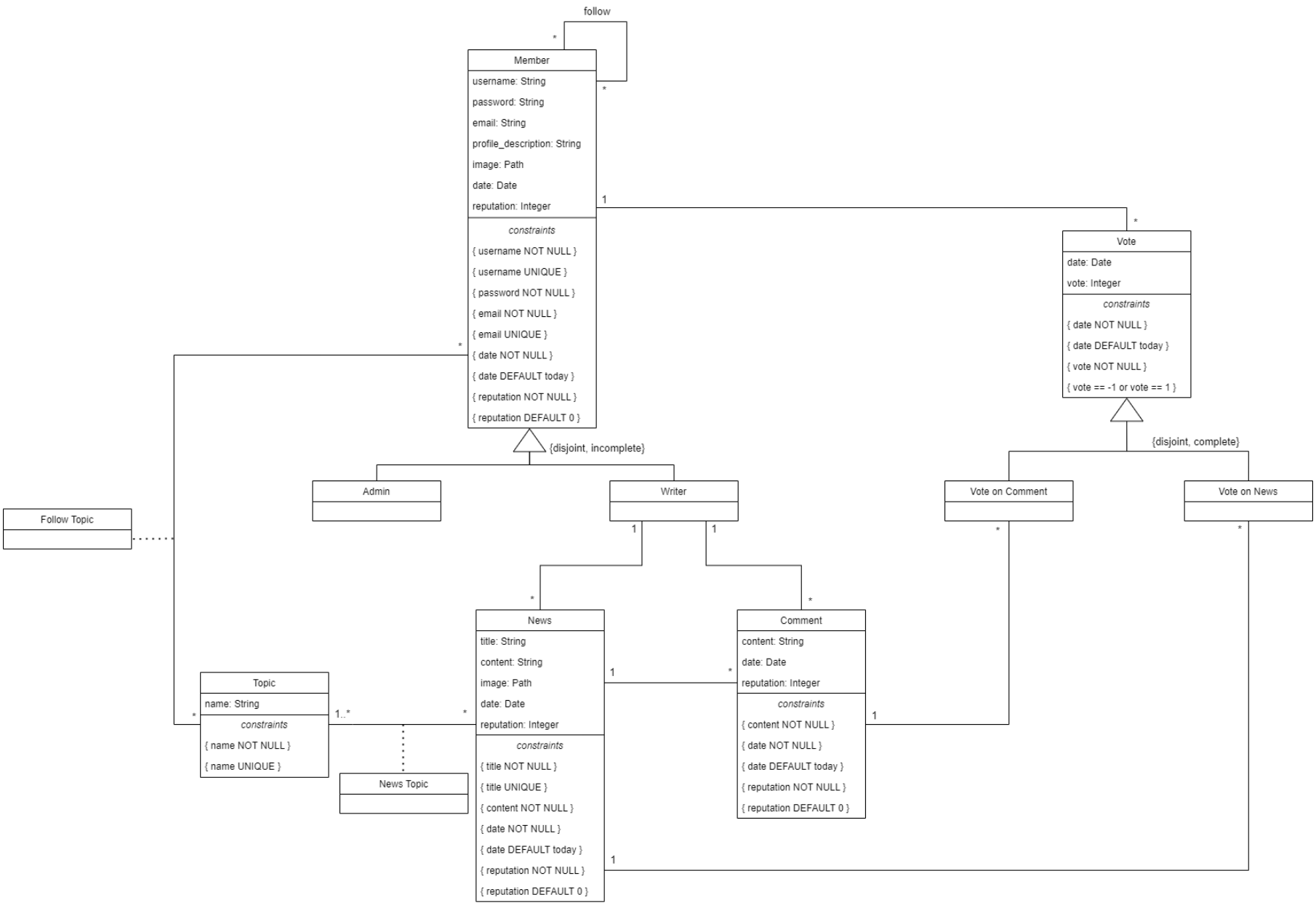


Figure 1: Collaborative News conceptual data model in UML.

### 2. Additional Business Rules

- BR01. When user account is deleted, shared user data (news items, comments and votes) is kept but is made anonymous
- BR02. User reputation depends on the upvotes and downvotes received on his posts or comments
- BR03. A post or comment cannot be deleted if it has votes or comments
- BR04. The date of a comment or a vote on a post must be greater than the item it refers to (Comment.date > News.date, VoteOnNews.date > News.date, VoteOnComment.date > Comment.date)
- BR05. A user can´t vote on his own posts or comments but he can comment his own news items
- BR06. A writer has to have at least one comment or news
- BR07. A user can't vote twice on an item
- BR08. An item's reputation is the sum of votes for that item (News.reputation = SUM(VoteOnNews.vote), Comment.reputation = SUM(VoteOnComment.vote))

---

## A5: Relational Schema, validation and schema refinement

The goal of the Relational Schema is to represent the data to be entered into the database and describe how that data is structured in tables. The tables are constructed properly and efficiently each representing a single subject and keeping redundant data to an absolute minimum while ensuring data integrity and suporting business rules.

## 1. Relational Schema

For the vote generalization, the best option is to use the Object-oriented strategy and eliminate the relation for the superclass.

| **Relation reference** | **Relation Compact Notation** |
|---|---|
| R01 | member(id_member, username **UK NN** , password **NN**, email **UK NN**, profile_description, image **UK**, date **NN DF** Today, reputation **NN DF** 0, is_admin **NN**) |
| R02 | news(id_news, writer -> member, title **UK NN**, content **NN**, image **UK**, date **NN DF** Today, reputation **NN DF** 0) |
| R03 | comment(id_comment, writer -> member, news -> news, content **NN**, date **NN DF** Today, reputation **NN DF** 0) |
| R04 | follow_member(follower -> member, followed -> member) |
| R05 | topic(id_topic, name **UK NN**) |
| R06 | news_topic(news -> news, topic -> topic) |
| R07 | follow_topic(member -> member, topic -> topic) |
| R08 | vote_on_news(member -> member, news -> news, vote **NN CK** vote = 1 OR vote = -1, date **NN DF** Today) |
| R09 | vote_on_comment(member -> member, comment -> comment, vote **NN CK** vote = 1 OR vote = -1, date **NN DF** Today) |

Legend:

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

## 2. Domains

| **Domain Name** | **Domain Specification** |
|---|---|
| Today | DATE DEFAULT CURRENT_DATE |

## 3. Schema validation

| **TABLE R01** | **member** |
|---|---|
| **Keys** | { id_member }, { username }, { email } |
| **Functional Dependencies:** | |
| FD0101 | id_member → {username, password, email, profile_description, image, data, reputation, is_admin} |
| FD0102 | username → {id_member, password, email, profile_description, image, data, reputation, is_admin} |
| FD0103 | email → {id_member, username, password, profile_description, image, data, reputation, is_admin} |
| **NORMAL FORM** | BCNF |

| **TABLE R02** | **news** |
|---|---|
| **Keys** | { id_news }, { título } |
| **Functional Dependencies:** | |
| FD0201 | id_news → {writer, title, content, image, date, reputation} |
| FD0202 | título → {id_news, writer, content, image, date, reputation} |
| **NORMAL FORM** | BCNF |

| **TABLE R03** | **comment** |
|---|---|
| **Keys** | { id_comment } |
| **Functional Dependencies:** | |
| FD0301 | id_comment → {writer, news, content, date, reputation} |
| **NORMAL FORM** | BCNF |

| **TABLE R04** | **follow_member** |
|---|---|
| **Keys** | { follower, followed } |
| **Functional Dependencies:** | none |
| **NORMAL FORM** | BCNF |

| **TABLE R05** | **topic** |
|---|---|
| **Keys** | { id_topic }, { name } |
| **Functional Dependencies:** | |
| FD0501 | id_topic → {name} |
| FD0502 | name → {id_topic} |
| **NORMAL FORM** | BCNF |

| **TABLE R06** | **news_topic** |
|---|---|
| **Keys** | { news, topic } |
| **Functional Dependencies:** | none |
| **NORMAL FORM** | BCNF |

| **TABLE R07** | **follow_topic** |
|---|---|
| **Keys** | { member, topic } |
| **Functional Dependencies:** | none |
| **NORMAL FORM** | BCNF |

| **TABLE R08** | **vote_on_news** |
|---|---|
| **Keys** | { member, news } |
| **Functional Dependencies:** | |
| FD0801 | member, news → {vote, date} |
| **NORMAL FORM** | BCNF |

| **TABLE R09** | **vote_on_comment** |
|---|---|
| **Keys** | { member, comment } |
| **Functional Dependencies:** | |
| FD0901 | member, comment → {vote, date} |
| **NORMAL FORM** | BCNF |

Upon analysis, the relations in our schema were already in Boyce-Codd Normal Form. Therefore, the entire schema itself is in BCNF. This in turn reduces data modification anomalies and reduces data redundancy.

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

DROP TABLE IF EXISTS member CASCADE;
DROP TABLE IF EXISTS news CASCADE;
DROP TABLE IF EXISTS comments CASCADE;
DROP TABLE IF EXISTS follow_member CASCADE;
DROP TABLE IF EXISTS topic CASCADE;
DROP TABLE IF EXISTS news_topic CASCADE;
DROP TABLE IF EXISTS follow_topic CASCADE;
DROP TABLE IF EXISTS vote_on_news CASCADE;
DROP TABLE IF EXISTS vote_on_comment CASCADE;



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

-- Note that plural 'comments' and 'dates' names were adopted because comment and date are reserved words in PostgreSQL.

CREATE TABLE "member"(
  id_member SERIAL PRIMARY KEY,
  username  VARCHAR NOT NULL CONSTRAINT member_username_uk UNIQUE,
  passwords VARCHAR NOT NULL,
  email     VARCHAR NOT NULL CONSTRAINT member_email_uk UNIQUE,
  profile_description VARCHAR,
  image     VARCHAR UNIQUE,
  dates     TIMESTAMP WITHOUT TIME ZONE DEFAULT now() NOT NULL,
  reputation INTEGER DEFAULT 0 NOT NULL,
  is_admin  BOOLEAN NOT NULL
);

CREATE TABLE "news"(
  id_news SERIAL PRIMARY KEY,
  writer  INTEGER REFERENCES member (id_member) ON DELETE SET NULL ON UPDATE CASCADE,
  title   VARCHAR NOT NULL CONSTRAINT news_title_uk UNIQUE,
  content VARCHAR NOT NULL,
  image   VARCHAR UNIQUE,
  dates   TIMESTAMP WITHOUT TIME ZONE NOT NULL DEFAULT now(),
  reputation INTEGER DEFAULT 0 NOT NULL
);

CREATE TABLE "comments"(
  id_comment  SERIAL PRIMARY KEY,
  writer      INTEGER REFERENCES member (id_member) ON DELETE SET NULL ON UPDATE CASCADE,
  news        INTEGER REFERENCES news (id_news) ON DELETE RESTRICT ON UPDATE CASCADE,
  content     VARCHAR NOT NULL,
  dates       TIMESTAMP WITHOUT TIME ZONE DEFAULT now() NOT NULL,
  reputation  INTEGER DEFAULT 0 NOT NULL
);

CREATE TABLE "follow_member"(
  follower INTEGER REFERENCES member (id_member) ON DELETE CASCADE ON UPDATE CASCADE,
  followed INTEGER REFERENCES member (id_member) ON DELETE CASCADE ON UPDATE CASCADE,
  PRIMARY KEY (follower, followed)
);

CREATE TABLE "topic"(
  id_topic  SERIAL PRIMARY KEY,
  name      VARCHAR NOT NULL CONSTRAINT topic_name_uk UNIQUE
);

CREATE TABLE "news_topic"(
  id_news INTEGER REFERENCES news (id_news) ON DELETE CASCADE ON UPDATE CASCADE,
  topic   INTEGER REFERENCES topic (id_topic) ON DELETE RESTRICT ON UPDATE CASCADE,
  PRIMARY KEY (id_news, topic)
);
```

```
CREATE TABLE "follow_topic"(
  member  INTEGER REFERENCES member (id_member) ON DELETE CASCADE ON UPDATE CASCADE,
  topic   INTEGER REFERENCES topic (id_topic) ON DELETE CASCADE ON UPDATE CASCADE,
  PRIMARY KEY (member, topic)
);

CREATE TABLE "vote_on_news"(
  member  INTEGER REFERENCES member (id_member) ON DELETE SET NULL ON UPDATE CASCADE,
  news    INTEGER REFERENCES news (id_news) ON DELETE RESTRICT ON UPDATE CASCADE,
  vote    INTEGER NOT NULL,
  dates   TIMESTAMP WITHOUT TIME ZONE DEFAULT now() NOT NULL,
  PRIMARY KEY (member, news),
  CONSTRAINT vote_ck CHECK ((vote = 1) OR (vote = -1))
);

CREATE TABLE "vote_on_comment"(
  member    INTEGER REFERENCES member (id_member) ON DELETE SET NULL ON UPDATE CASCADE,
  comments  INTEGER REFERENCES comments (id_comment) ON DELETE RESTRICT ON UPDATE CASCADE,
  vote      INTEGER NOT NULL,
  dates     TIMESTAMP WITHOUT TIME ZONE DEFAULT now() NOT NULL,
  PRIMARY KEY (member, comments),
  CONSTRAINT vote_ck CHECK ((vote = 1) OR (vote = -1))
);
```

## A6: Indexes, triggers, transactions and database population

This artifact aims to estimate the amount of data which the database need to support, create indexes in order to retrieve information faster, even at the expense of some space, establish triggers so business rules are followed and the integrity of information uncompromised and implement transactions that provide isolation between concurrent accesses to the database and keep the database consistent even when failrue occurs.

### 1. Database Workload

| Relation reference | Relation Name | Order of magnitude | Estimated growth |
|---|---|---|---|
| R01 | member | 10 k (tens of thousands) | 10 (tens) / day |
| R02 | news | 10 k | 10 / day |
| R03 | comment | 100 k (hundreds of thousands) | 100 (hundreds) / day |
| R04 | follow_member | 100 k | 100 / day |
| R05 | topic | 100 (hundreds) | 1 (units) / day |
| R06 | news_topic | 10 k | 10 / day |
| R07 | follow_topic | 10 k | 10 / day |
| R08 | vote_on_news | 100 k | 100 / day |
| R09 | vote_on_comment | 1 M (millions) | 1 k (thousands) / day |

### 2. Proposed Indices

#### 2.1. Performance Indices

| Index | IDX01 |
|---|---|
| **Relation** | news |
| **Attribute** | writer |
| **Type** | Hash |
| **Cardinality** | Medium |
| **Clustering** | No |

| **Index** | **IDX01** |
|---|---|
| **Justification** | The access to "writer" from news has to be done fast. Thus, it is necessary to create an index for "writer" to match it with the member quickly. Filtering is done through an exact match so hash is the appropriate structure to use. The cardinality is medium given the fact that "writer" can appear several times in the table. Clustering is not possible since the update frequency is high. |
| SQL code | `CREATE INDEX write_news ON news USING hash (writer);` |

| **Index** | **IDX02** |
|---|---|
| **Relation** | news |
| **Attribute** | date |
| **Type** | B-tree |
| **Cardinality** | Medium |
| **Clustering** | No |
| **Justification** | The access to table "news" filtered by a range query based on the "date" attribute will be done frequently. Thus, B-Tree structure is the approriate choice. The cardinality is medium given the fact that "date" can appear several times in the table. Clustering is not possible since the update frequency is high. |
| SQL code | `CREATE INDEX news_date ON news USING BTREE (dates);` |

| **Index** | **IDX03** |
|---|---|
| **Relation** | news |
| **Attribute** | reputation |
| **Type** | B-tree |
| **Cardinality** | Medium |
| **Clustering** | No |
| **Justification** | The access to table "reputation" filtered by a range query based on the "reputation" attribute will be done frequently. Thus, B-Tree structure is the approriate choice. The cardinality is medium given the fact that "reputation" can appear several times in the table. Clustering is not possible since the update frequency is high. |
| SQL code | `CREATE INDEX news_reputation ON news USING BTREE (reputation);` |

| **Index** | **IDX04** |
|---|---|
| **Relation** | comment |
| **Attribute** | writer |
| **Type** | Hash |
| **Cardinality** | Medium |
| **Clustering** | No |
| **Justification** | The access to "writer" from comment has to be done fast. Thus, it is necessary to create an index for "writer" to match it with the member quickly. Filtering is done through an exact match so hash is the appropriate structure to use. The cardinality is medium given the fact that "writer" can appear several times in the table. Clustering is not possible since the update frequency is high. |
| SQL code | `CREATE INDEX comment_writer ON comments USING hash (writer);` |

| **Index** | **IDX05** |
|---|---|
| **Relation** | comment |
| **Attribute** | news |

| **Index** | **IDX05** |
|---|---|
| **Type** | Hash |
| **Cardinality** | Medium |
| **Clustering** | No |
| **Justification** | The access to "news" from comment has to be done fast. Thus, it is necessary to create an index for "news" to match it with the news associated quickly. Filtering is done through an exact match so hash is the appropriate structure to use. The cardinality is medium given the fact that "news" can appear several times in the table. Clustering is not possible since the update frequency is high. |
| SQL code | `CREATE INDEX comment_news ON comments USING hash (news);` |

| **Index** | **IDX06** |
|---|---|
| **Relation** | follow_member |
| **Attribute** | follower |
| **Type** | Hash |
| **Cardinality** | Medium |
| **Clustering** | No |
| **Justification** | The access to "follower" from follow_member has to be done fast. Thus, it is necessary to create an index for "follower" to match it with the member we want to view who his he following quickly. Filtering is done through an exact match so hash is the appropriate structure to use. The cardinality is medium given the fact that "follower" can appear several times in the table. Clustering is not possible since the update frequency is high. |
| SQL code | `CREATE INDEX order_by_follower ON follow_member USING hash (follower);` |

| **Index** | **IDX07** |
|---|---|
| **Relation** | follow_member |
| **Attribute** | followed |
| **Type** | Hash |
| **Cardinality** | Medium |
| **Clustering** | No |
| **Justification** | The access to "followed" from follow_member has to be done fast. Thus, it is necessary to create an index for "followed" to match it with the member we want to view who follows him quickly. Filtering is done through an exact match so hash is the appropriate structure to use. The cardinality is medium given the fact that "followed" can appear several times in the table. Clustering is not possible since the update frequency is high. |
| SQL code | `CREATE INDEX order_by_followed ON follow_member USING hash (followed);` |

| **Index** | **IDX08** |
|---|---|
| **Relation** | news_topic |
| **Attribute** | topic |
| **Type** | Hash |
| **Cardinality** | Medium |
| **Clustering** | No |
| **Justification** | The access to "topic" from news_topic has to be done fast. Thus, it is necessary to create an index for "topic" to match the news with the topic which it is associated quickly. Filtering is done through an exact match so hash is the appropriate structure to use. The cardinality is medium given the fact that "writer" can appear several times in the table. Clustering is not possible since the update frequency is high. |

| Index | IDX08 |
|---|---|
| SQL code | `CREATE INDEX news_topics ON news_topic USING hash (topic);` |

| Index | IDX09 |
|---|---|
| **Relation** | follow_topic |
| **Attribute** | member |
| **Type** | Hash |
| **Cardinality** | Medium |
| **Clustering** | No |
| **Justification** | The access to "member" from follow_topic has to be done fast. Thus, it is necessary to create an index for "member" to match the member with the topics that he follows quickly. Filtering is done through an exact match so hash is the appropriate structure to use. The cardinality is medium given the fact that "writer" can appear several times in the table. Clustering is not possible since the update frequency is high. |
| SQL code | `CREATE INDEX member_topic ON follow_topic USING hash (member);` |

| Index | IDX10 |
|---|---|
| **Relation** | vote_on_news |
| **Attribute** | news |
| **Type** | Hash |
| **Cardinality** | Medium |
| **Clustering** | No |
| **Justification** | The access to "news" from vote_on_news has to be done fast. Thus, it is necessary to create an index for "news" to match the vote with the news associated quickly. Filtering is done through an exact match so hash is the appropriate structure to use. The cardinality is medium given the fact that "news" can appear several times in the table. Clustering is not possible since the update frequency is high. |
| SQL code | `CREATE INDEX news_vote ON vote_on_news USING hash (news);` |

| Index | IDX11 |
|---|---|
| **Relation** | vote_on_comment |
| **Attribute** | comment |
| **Type** | Hash |
| **Cardinality** | Medium |
| **Clustering** | No |
| **Justification** | The access to "comment" from vote_on_comment has to be done fast. Thus, it is necessary to create an index for "comment" to match the vote with the comment associated quickly. Filtering is done through an exact match so hash is the appropriate structure to use. The cardinality is medium given the fact that "comment" can appear several times in the table. Clustering is not possible since the update frequency is high. |
| SQL code | `CREATE INDEX comment_vote ON vote_on_comment USING hash (comments);` |

### 2.2. Full-text Search Indices

| Index | IDX12 |
|---|---|
| **Relation** | news |
| **Attribute** | title, content |

| **Index** | **IDX12** |
|---|---|
| **Type** | GIN |
| **Clustering** | No |
| **Justification** | To provide full-text search features to look for news based on matching titles or content. The index type is GIN because the indexed fields are not expected to change often. **Respective trigger (TRIGGER01) is provided below in Trigger's section.** |
| SQL Code | `CREATE INDEX search_idx ON work USING GIN (tsvectors);` |

## 3. Triggers

| **Trigger** | **TRIGGER01** |
|---|---|
| Description | This trigger is used when a news is created or updated, and it is then necessary to update the tsvector associated with it |

```
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();
```

| **Trigger** | **TRIGGER02** |
|---|---|
| Description | This trigger is used when a new vote is inserted on vote_on_news, and it is necessary to keep track of total reputation from the news |

```
CREATE OR REPLACE FUNCTION vote_on_news_insert() RETURNS TRIGGER AS $$
DECLARE
    aux_writer INTEGER;
    aux_id_member INTEGER;
BEGIN
    SELECT writer FROM news WHERE NEW.news = id_news INTO aux_writer;
    SELECT id_member FROM member WHERE NEW.member = id_member INTO aux_id_member;
    IF (aux_writer = aux_id_member) THEN
        RAISE EXCEPTION 'Writer cannot vote on own news';
    END IF;
    UPDATE news SET reputation = reputation + NEW.vote  WHERE news.id_news = NEW.news;
     RETURN NEW;
END $$
LANGUAGE plpgsql;


CREATE TRIGGER vote_on_news_insert
        BEFORE INSERT ON vote_on_news
        FOR EACH ROW
        EXECUTE PROCEDURE vote_on_news_insert();
```

| **Trigger** | **TRIGGER03** |
|---|---|
| Description | This trigger is used when a vote is updated on vote_on_news, and it is necessary to keep track of total reputation from the news |

```
CREATE OR REPLACE FUNCTION vote_on_news_update() RETURNS TRIGGER AS $$
BEGIN
    IF (NEW.vote <> OLD.vote) THEN
        UPDATE news SET reputation = reputation - OLD.vote + NEW.vote WHERE news.id_news = NEW.news;
    END IF;
    RETURN NEW;
END $$
LANGUAGE plpgsql;


CREATE TRIGGER vote_on_news_update
        AFTER UPDATE ON vote_on_news
        FOR EACH ROW
        EXECUTE PROCEDURE vote_on_news_update();
```

| **Trigger** | **TRIGGER04** |
|---|---|
| Description | This trigger is used when a vote is deleted on vote_on_news, and it is necessary to keep track of total reputation from the news |

```
CREATE OR REPLACE FUNCTION vote_on_news_delete() RETURNS TRIGGER AS $$
BEGIN
     UPDATE news SET reputation = reputation - OLD.vote WHERE news.id_news = OLD.news;
     RETURN OLD;
END $$
LANGUAGE plpgsql;


CREATE TRIGGER vote_on_news_delete
        AFTER DELETE ON vote_on_news
        FOR EACH ROW
        EXECUTE PROCEDURE vote_on_news_delete();
```

| **Trigger** | **TRIGGER05** |
|---|---|
| Description | This trigger is used when a new vote is inserted on vote_on_comment, and it is necessary to keep track of total reputation from the comment |

```
CREATE OR REPLACE FUNCTION vote_on_comment_insert() RETURNS TRIGGER AS $$
DECLARE
    aux_writer INTEGER;
    aux_id_member INTEGER;
BEGIN
    SELECT writer FROM comments WHERE NEW.comments = id_comment INTO aux_writer;
    SELECT id_member FROM member WHERE NEW.member = id_member INTO aux_id_member;
    IF (aux_writer = aux_id_member) THEN
        RAISE EXCEPTION 'Writer cannot vote on own comment';
    END IF;
    UPDATE comments SET reputation = reputation + NEW.vote WHERE comments.id_comment = NEW.comments;
     RETURN NEW;
END $$
LANGUAGE plpgsql;


CREATE TRIGGER vote_on_comment_insert
        BEFORE INSERT ON vote_on_comment
        FOR EACH ROW
        EXECUTE PROCEDURE vote_on_comment_insert();
```

| **Trigger** | **TRIGGER06** |
|---|---|
| Description | This trigger is used when a vote is updated on vote_on_comment, and it is necessary to keep track of total reputation from the comment |

```
CREATE OR REPLACE FUNCTION vote_on_comment_update() RETURNS TRIGGER AS $$
BEGIN
    IF (NEW.vote <> OLD.vote) THEN
                UPDATE comments SET reputation = reputation - OLD.vote + NEW.vote WHERE comments.id_comment = NEW.com
ments;
    END IF;
    RETURN NEW;
END $$
LANGUAGE plpgsql;


CREATE TRIGGER vote_comment_update
        AFTER UPDATE ON vote_on_comment
        FOR EACH ROW
        EXECUTE PROCEDURE vote_on_comment_update();
```

| **Trigger** | **TRIGGER07** |
|---|---|
| Description | This trigger is used when a vote is deleted on vote_on_comment, and it is necessary to keep track of total reputation from the comment |

```
CREATE OR REPLACE FUNCTION vote_on_comment_delete() RETURNS TRIGGER AS $$
BEGIN
    UPDATE comments SET reputation = reputation - OLD.vote  WHERE comments.id_comment = NEW.comments;
    RETURN OLD;
END $$
LANGUAGE plpgsql;


CREATE TRIGGER vote_on_comment_delete
        AFTER DELETE ON vote_on_comment
        FOR EACH ROW
        EXECUTE PROCEDURE vote_on_comment_delete();
```

| **Trigger** | **TRIGGER08** |
|---|---|
| Description | This trigger is used when a news is updated, and it is necessary to keep track of total reputation of its writer |

```
CREATE OR REPLACE FUNCTION user_news_reputation() RETURNS TRIGGER AS $$
BEGIN
    UPDATE member SET reputation = reputation - OLD.reputation + NEW.reputation WHERE NEW.writer = member.id_member;
    RETURN NEW;
END $$
LANGUAGE plpgsql;


CREATE TRIGGER user_news_reputation
        AFTER UPDATE ON news
        FOR EACH ROW
        EXECUTE PROCEDURE user_news_reputation();
```

| **Trigger** | **TRIGGER09** |
|---|---|
| Description | This trigger is used when a comment is updated, and it is necessary to keep track of total reputation of its writer |

```
CREATE OR REPLACE FUNCTION user_comment_reputation() RETURNS TRIGGER AS $$
BEGIN
    UPDATE member SET reputation = reputation - OLD.reputation + NEW.reputation WHERE NEW.writer = member.id_member;
    RETURN NEW;
END $$
LANGUAGE plpgsql;


CREATE TRIGGER user_comment_reputation
        AFTER UPDATE ON comments
        FOR EACH ROW
        EXECUTE PROCEDURE user_comment_reputation();
```

| **Trigger** | **TRIGGER10** |
|---|---|
| Description | This trigger is used when a comment is inserted, and it checks if the date of the comment is valid (>= date of the news it refers to) |

```
CREATE OR REPLACE FUNCTION check_comment_date() RETURNS TRIGGER AS $$
BEGIN
        IF NEW.dates < (SELECT news.dates FROM news     WHERE news.id_news = NEW.news)
        THEN
                RAISE EXCEPTION 'One can´t make a comment on a news before being released';
        END IF;
    RETURN NEW;
END $$
LANGUAGE plpgsql;


CREATE TRIGGER comment_date
        BEFORE INSERT ON comments
        FOR EACH ROW
        EXECUTE PROCEDURE check_comment_date();
```

| **Trigger** | **TRIGGER11** |
|---|---|
| Description | This trigger is used when a vote on news is inserted, and it checks if the date of the vote is valid (>= date of the news it refers to) |

```
CREATE OR REPLACE FUNCTION check_vote_news_date() RETURNS TRIGGER AS $$
BEGIN
        IF NEW.dates < (SELECT news.dates FROM news WHERE news.id_news = NEW.news)
        THEN
                RAISE EXCEPTION 'One can´t vote on a news before being released';
        END IF;
    RETURN NEW;
END $$
LANGUAGE plpgsql;


CREATE TRIGGER vote_news_date
        BEFORE INSERT ON vote_on_news
        FOR EACH ROW
        EXECUTE PROCEDURE check_vote_news_date();
```

| **Trigger** | **TRIGGER12** |
|---|---|
| Description | This trigger is used when a vote on comment is inserted, and it checks if the date of the vote is valid (>= date of the comment it refers to) |

```
CREATE OR REPLACE FUNCTION check_vote_comment_date() RETURNS TRIGGER AS $$
BEGIN
        IF NEW.dates < (SELECT comments.dates FROM comments WHERE comments.id_comment = NEW.comments)
        THEN
                RAISE EXCEPTION 'One can´t vote on a comment before being released';
        END IF;
    RETURN NEW;
END $$
LANGUAGE plpgsql;


CREATE TRIGGER vote_comment_date
        BEFORE INSERT ON vote_on_comment
        FOR EACH ROW
        EXECUTE PROCEDURE check_vote_comment_date();
```

| **Trigger** | **TRIGGER13** |
|---|---|
| Description | This trigger is used to secure that no member can follow himself |

```
CREATE OR REPLACE FUNCTION follow_member_insert() RETURNS TRIGGER AS $$
BEGIN
    IF (NEW.follower = NEW.followed)
        THEN
        RAISE EXCEPTION 'Member cannot follow himself';
    END IF;
        RETURN NEW;
END $$
LANGUAGE plpgsql;


CREATE TRIGGER follow_member_insert
        BEFORE INSERT ON follow_member
        FOR EACH ROW
        EXECUTE PROCEDURE follow_member_insert();
```

## 4. Transactions

| **Transaction** | **TRAN01 Add vote on news** |
|---|---|
| Justification | To maintain consistency, it's necessary to use a transaction to ensure that all the code executes without errors. A serializable isolation level is used so that so that it is not possible for the reputation value to be calculated incorrectly, due to the fact that two insertion or update operations are performed in the vote_on_news table |

| Isolation level | Serializable |
| --- | --- |

```
BEGIN TRANSACTION;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE


UPDATE news SET reputation = reputation + NEW.vote  WHERE news.id_news = NEW.news;


COMMIT;


BEGIN TRANSACTION;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE


UPDATE news SET reputation = reputation - OLD.vote + NEW.vote WHERE news.id_news = NEW.news;


COMMIT;
```

```
BEGIN TRANSACTION;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE


UPDATE news SET reputation = reputation - OLD.vote WHERE news.id_news = OLD.news;


COMMIT;
```

| **Transaction** | **TRAN02 Add vote on comment** |
| --- | --- |
| Justification | To maintain consistency, it's necessary to use a transaction to ensure that the all the code executes without errors. An serializable isolation level is used so that so that it is not possible for the reputation value to be calculated incorrectly, due to the fact that two insertion or update operations are performed in the vote_on_comment table |
| Isolation level | Serializable |

```
BEGIN TRANSACTION;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE


UPDATE comments SET reputation = reputation + NEW.vote WHERE comments.id_comment = NEW.comments;


COMMIT;


BEGIN TRANSACTION;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE


UPDATE comments SET reputation = reputation - OLD.vote + NEW.vote WHERE comments.id_comment = NEW.comments;


COMMIT;


BEGIN TRANSACTION;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE


UPDATE comments SET reputation = reputation - OLD.vote  WHERE comments.id_comment = NEW.comments;


COMMIT;
```

# Annex A. SQL Code

- Create.sql
- Populate.sql

## A.1. Database schema

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

DROP TABLE IF EXISTS member CASCADE;
DROP TABLE IF EXISTS news CASCADE;
DROP TABLE IF EXISTS comments CASCADE;
DROP TABLE IF EXISTS follow_member CASCADE;
DROP TABLE IF EXISTS topic CASCADE;
DROP TABLE IF EXISTS news_topic CASCADE;
DROP TABLE IF EXISTS follow_topic CASCADE;
DROP TABLE IF EXISTS vote_on_news CASCADE;
DROP TABLE IF EXISTS vote_on_comment CASCADE;


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

-- Note that plural 'comments' and 'dates' names were adopted because comment and date are reserved words in PostgreSQL.

CREATE TABLE "member"(
  id_member SERIAL PRIMARY KEY,
  username  VARCHAR NOT NULL CONSTRAINT member_username_uk UNIQUE,
  passwords VARCHAR NOT NULL,
  email     VARCHAR NOT NULL CONSTRAINT member_email_uk UNIQUE,
  profile_description VARCHAR,
  image     VARCHAR UNIQUE,
  dates     TIMESTAMP WITHOUT TIME ZONE DEFAULT now() NOT NULL,
  reputation INTEGER DEFAULT 0 NOT NULL,
  is_admin  BOOLEAN NOT NULL
);
```

```sql
CREATE TABLE "news"(
  id_news SERIAL PRIMARY KEY,
  writer  INTEGER REFERENCES member (id_member) ON DELETE SET NULL ON UPDATE CASCADE,
  title   VARCHAR NOT NULL CONSTRAINT news_title_uk UNIQUE,
  content VARCHAR NOT NULL,
  image   VARCHAR UNIQUE,
  dates   TIMESTAMP WITHOUT TIME ZONE NOT NULL DEFAULT now(),
  reputation INTEGER DEFAULT 0 NOT NULL
);

CREATE TABLE "comments"(
  id_comment  SERIAL PRIMARY KEY,
  writer      INTEGER REFERENCES member (id_member) ON DELETE SET NULL ON UPDATE CASCADE,
  news        INTEGER REFERENCES news (id_news) ON DELETE RESTRICT ON UPDATE CASCADE,
  content     VARCHAR NOT NULL,
  dates       TIMESTAMP WITHOUT TIME ZONE DEFAULT now() NOT NULL,
  reputation  INTEGER DEFAULT 0 NOT NULL
);

CREATE TABLE "follow_member"(
  follower INTEGER REFERENCES member (id_member) ON DELETE CASCADE ON UPDATE CASCADE,
  followed INTEGER REFERENCES member (id_member) ON DELETE CASCADE ON UPDATE CASCADE,
  PRIMARY KEY (follower, followed)
);

CREATE TABLE "topic"(
  id_topic  SERIAL PRIMARY KEY,
  name      VARCHAR NOT NULL CONSTRAINT topic_name_uk UNIQUE
);

CREATE TABLE "news_topic"(
  id_news INTEGER REFERENCES news (id_news) ON DELETE CASCADE ON UPDATE CASCADE,
  topic   INTEGER REFERENCES topic (id_topic) ON DELETE RESTRICT ON UPDATE CASCADE,
  PRIMARY KEY (id_news, topic)
);

CREATE TABLE "follow_topic"(
  member  INTEGER REFERENCES member (id_member) ON DELETE CASCADE ON UPDATE CASCADE,
  topic   INTEGER REFERENCES topic (id_topic) ON DELETE CASCADE ON UPDATE CASCADE,
  PRIMARY KEY (member, topic)
);

CREATE TABLE "vote_on_news"(
  member  INTEGER REFERENCES member (id_member) ON DELETE SET NULL ON UPDATE CASCADE,
  news    INTEGER REFERENCES news (id_news) ON DELETE RESTRICT ON UPDATE CASCADE,
  vote    INTEGER NOT NULL,
  dates   TIMESTAMP WITHOUT TIME ZONE DEFAULT now() NOT NULL,
  PRIMARY KEY (member, news),
  CONSTRAINT vote_ck CHECK ((vote = 1) OR (vote = -1))
);

CREATE TABLE "vote_on_comment"(
  member    INTEGER REFERENCES member (id_member) ON DELETE SET NULL ON UPDATE CASCADE,
  comments  INTEGER REFERENCES comments (id_comment) ON DELETE RESTRICT ON UPDATE CASCADE,
  vote      INTEGER NOT NULL,
  dates     TIMESTAMP WITHOUT TIME ZONE DEFAULT now() NOT NULL,
  PRIMARY KEY (member, comments),
  CONSTRAINT vote_ck CHECK ((vote = 1) OR (vote = -1))
);

----------------------------------------
-- Index
----------------------------------------

DROP INDEX IF EXISTS write_news;
DROP INDEX IF EXISTS news_date;
DROP INDEX IF EXISTS news_reputation;
DROP INDEX IF EXISTS comment_writer;
DROP INDEX IF EXISTS comment_news;
DROP INDEX IF EXISTS order_by_follower;
DROP INDEX IF EXISTS order_by_followed;
```

```
DROP INDEX IF EXISTS news_topics;
DROP INDEX IF EXISTS member_topic;
DROP INDEX IF EXISTS news_vote;
DROP INDEX IF EXISTS comment_vote;
DROP INDEX IF EXISTS search_idx;

CREATE INDEX write_news ON news USING hash (writer);
CREATE INDEX news_date ON news USING BTREE (dates);
CREATE INDEX news_reputation ON news USING BTREE (reputation);
CREATE INDEX comment_writer ON comments USING hash (writer);
CREATE INDEX comment_news ON comments USING hash (news);
CREATE INDEX order_by_follower ON follow_member USING hash (follower);
CREATE INDEX order_by_followed ON follow_member USING hash (followed);
CREATE INDEX news_topics ON news_topic USING hash (topic);
CREATE INDEX member_topic ON follow_topic USING hash (member);
CREATE INDEX news_vote ON vote_on_news USING hash (news);
CREATE INDEX comment_vote ON vote_on_comment USING hash (comments);



-----------------------------------------
-- User-defined Functions and Triggers
-----------------------------------------

---------------------- TRIGGER_01
DROP TRIGGER IF EXISTS news_search_update ON news;

ALTER TABLE news
ADD COLUMN tsvectors TSVECTOR;
CREATE OR REPLACE 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();

-------------------- Related Index

CREATE INDEX search_idx ON news USING GIN (tsvectors);



---------------------- TRIGGER_02
DROP TRIGGER IF EXISTS vote_on_news_insert ON vote_on_news;

CREATE OR REPLACE FUNCTION vote_on_news_insert() RETURNS TRIGGER AS $$
DECLARE
    aux_writer INTEGER;
    aux_id_member INTEGER;
BEGIN
    SELECT writer FROM news WHERE NEW.news = id_news INTO aux_writer;
    SELECT id_member FROM member WHERE NEW.member = id_member INTO aux_id_member;
    IF (aux_writer = aux_id_member) THEN
        RAISE EXCEPTION 'Writer cannot vote on own news';
    END IF;
    UPDATE news SET reputation = reputation + NEW.vote  WHERE news.id_news = NEW.news;
```

```
     RETURN NEW;
END $$
LANGUAGE plpgsql;

CREATE TRIGGER vote_on_news_insert
        BEFORE INSERT ON vote_on_news
        FOR EACH ROW
        EXECUTE PROCEDURE vote_on_news_insert();

-------------------- TRIGGER_03
DROP TRIGGER IF EXISTS vote_on_news_update ON vote_on_news;

CREATE OR REPLACE FUNCTION vote_on_news_update() RETURNS TRIGGER AS $$
BEGIN
    IF (NEW.vote <> OLD.vote) THEN
        UPDATE news SET reputation = reputation - OLD.vote + NEW.vote WHERE news.id_news = NEW.news;
    END IF;
    RETURN NEW;
END $$
LANGUAGE plpgsql;

CREATE TRIGGER vote_on_news_update
        AFTER UPDATE ON vote_on_news
        FOR EACH ROW
        EXECUTE PROCEDURE vote_on_news_update();

-------------------- TRIGGER_04
DROP TRIGGER IF EXISTS vote_on_news_delete ON vote_on_news;

CREATE OR REPLACE FUNCTION vote_on_news_delete() RETURNS TRIGGER AS $$
BEGIN
     UPDATE news SET reputation = reputation - OLD.vote WHERE news.id_news = OLD.news;
     RETURN OLD;
END $$
LANGUAGE plpgsql;

CREATE TRIGGER vote_on_news_delete
        AFTER DELETE ON vote_on_news
        FOR EACH ROW
        EXECUTE PROCEDURE vote_on_news_delete();

-------------------- TRIGGER_05
DROP TRIGGER IF EXISTS vote_on_comment_insert ON vote_on_comment;

CREATE OR REPLACE FUNCTION vote_on_comment_insert() RETURNS TRIGGER AS $$
DECLARE
    aux_writer INTEGER;
    aux_id_member INTEGER;
BEGIN
    SELECT writer FROM comments WHERE NEW.comments = id_comment INTO aux_writer;
    SELECT id_member FROM member WHERE NEW.member = id_member INTO aux_id_member;
    IF (aux_writer = aux_id_member) THEN
        RAISE EXCEPTION 'Writer cannot vote on own comment';
    END IF;
    UPDATE comments SET reputation = reputation + NEW.vote WHERE comments.id_comment = NEW.comments;
     RETURN NEW;
END $$
LANGUAGE plpgsql;

CREATE TRIGGER vote_on_comment_insert
        BEFORE INSERT ON vote_on_comment
        FOR EACH ROW
        EXECUTE PROCEDURE vote_on_comment_insert();

-------------------- TRIGGER_06
DROP TRIGGER IF EXISTS vote_on_comment_update ON vote_on_comment;

CREATE OR REPLACE FUNCTION vote_on_comment_update() RETURNS TRIGGER AS $$
BEGIN
    IF (NEW.vote <> OLD.vote) THEN
                UPDATE comments SET reputation = reputation - OLD.vote + NEW.vote WHERE comments.id_comment = NEW.comments;
    END IF;
```

```
    RETURN NEW;
END $$
LANGUAGE plpgsql;

CREATE TRIGGER vote_comment_update
        AFTER UPDATE ON vote_on_comment
        FOR EACH ROW
        EXECUTE PROCEDURE vote_on_comment_update();

-------------------- TRIGGER_07
DROP TRIGGER IF EXISTS vote_on_comment_delete ON vote_on_comment;

CREATE OR REPLACE FUNCTION vote_on_comment_delete() RETURNS TRIGGER AS $$
BEGIN
    UPDATE comments SET reputation = reputation - OLD.vote  WHERE comments.id_comment = NEW.comments;
    RETURN OLD;
END $$
LANGUAGE plpgsql;

CREATE TRIGGER vote_on_comment_delete
        AFTER DELETE ON vote_on_comment
        FOR EACH ROW
        EXECUTE PROCEDURE vote_on_comment_delete();
```

```
-------------------- TRIGGER_08
DROP TRIGGER IF EXISTS user_news_reputation ON news;

CREATE OR REPLACE FUNCTION user_news_reputation() RETURNS TRIGGER AS $$
BEGIN
    UPDATE member SET reputation = reputation - OLD.reputation + NEW.reputation WHERE NEW.writer = member.id_member;
    RETURN NEW;
END $$
LANGUAGE plpgsql;

CREATE TRIGGER user_news_reputation
        AFTER UPDATE ON news
        FOR EACH ROW
        EXECUTE PROCEDURE user_news_reputation();

-------------------- TRIGGER_09
DROP TRIGGER IF EXISTS user_comment_reputation ON comments;

CREATE OR REPLACE FUNCTION user_comment_reputation() RETURNS TRIGGER AS $$
BEGIN
    UPDATE member SET reputation = reputation - OLD.reputation + NEW.reputation WHERE NEW.writer = member.id_member;
    RETURN NEW;
END $$
LANGUAGE plpgsql;

CREATE TRIGGER user_comment_reputation
        AFTER UPDATE ON comments
        FOR EACH ROW
        EXECUTE PROCEDURE user_comment_reputation();

-------------------- TRIGGER_10
DROP TRIGGER IF EXISTS comment_date ON comments;

CREATE OR REPLACE FUNCTION check_comment_date() RETURNS TRIGGER AS $$
BEGIN
        IF NEW.dates < (SELECT news.dates FROM news     WHERE news.id_news = NEW.news)
        THEN
                RAISE EXCEPTION 'One can´t make a comment on a news before being released';
        END IF;
    RETURN NEW;
END $$
LANGUAGE plpgsql;

CREATE TRIGGER comment_date
        BEFORE INSERT ON comments
        FOR EACH ROW
        EXECUTE PROCEDURE check_comment_date();
```

```
------------------ TRIGGER_11
DROP TRIGGER IF EXISTS vote_news_date ON vote_on_news;

CREATE OR REPLACE FUNCTION check_vote_news_date() RETURNS TRIGGER AS $$
BEGIN
        IF NEW.dates < (SELECT news.dates FROM news WHERE news.id_news = NEW.news)
        THEN
                RAISE EXCEPTION 'One can´t vote on a news before being released';
        END IF;
    RETURN NEW;
END $$
LANGUAGE plpgsql;

CREATE TRIGGER vote_news_date
        BEFORE INSERT ON vote_on_news
        FOR EACH ROW
        EXECUTE PROCEDURE check_vote_news_date();

---------------------- TRIGGER_12
DROP TRIGGER IF EXISTS vote_comment_date ON vote_on_comment;

CREATE OR REPLACE FUNCTION check_vote_comment_date() RETURNS TRIGGER AS $$
BEGIN
        IF NEW.dates < (SELECT comments.dates FROM comments WHERE comments.id_comment = NEW.comments)
        THEN
                RAISE EXCEPTION 'One can´t vote on a comment before being released';
        END IF;
    RETURN NEW;
END $$
LANGUAGE plpgsql;

CREATE TRIGGER vote_comment_date
        BEFORE INSERT ON vote_on_comment
        FOR EACH ROW
        EXECUTE PROCEDURE check_vote_comment_date();

---------------------- TRIGGER_13
DROP TRIGGER IF EXISTS follow_member_insert ON follow_member;

CREATE OR REPLACE FUNCTION follow_member_insert() RETURNS TRIGGER AS $$
BEGIN
     IF (NEW.follower = NEW.followed)
         THEN
         RAISE EXCEPTION 'Member cannot follow himself';
     END IF;
         RETURN NEW;
END $$
LANGUAGE plpgsql;

CREATE TRIGGER follow_member_insert
        BEFORE INSERT ON follow_member
        FOR EACH ROW
        EXECUTE PROCEDURE follow_member_insert();
```

## A.2. Database population

```
DELETE FROM vote_on_news CASCADE;
DELETE FROM vote_on_comment CASCADE;
DELETE FROM news_topic CASCADE;
DELETE FROM comments CASCADE;
DELETE FROM follow_member CASCADE;
DELETE FROM follow_topic CASCADE;
DELETE FROM topic CASCADE;
DELETE FROM news CASCADE;
DELETE FROM member CASCADE;


INSERT INTO
        member(id_member, username, passwords, email, profile_description, image, dates, reputation, is_admin)
VALUES
 (1, 'jcundict0', '3zOxwZ368o', 'rcashman0@alexa.com', 'I am a 53-year-old semi-professional sports person who enjoys swimmi
```

```
 (2, 'lface1', 'tntOnY', 'edelia1@usatoday.com', 'I am a 24-year-old sous chef who enjoys extreme ironing, watching televisi
 (3, 'odemattia2', 'UODTRl3E', 'dpreedy2@wordpress.com', 'I am a 23-year-old health centre receptionist who enjoys recycling
 (4, 'tharmstone3', 'JIZH0NdwZNkY', 'dbroggelli3@apache.org', 'I am a 29-year-old chef at chain restaurant who enjoys extrem
 (5, 'cpillington4', 'hlNxfR', 'rscrimshaw4@globo.com', 'I am a 73-year-old former government politician who enjoys reading,
 (6, 'avasser5', 'jAwSbTp', 'glaurenz5@1und1.de', 'I am a 29-year-old tradesperson assistant who enjoys helping old ladies a
 (7, 'mstokey6', 'gwnHiyoAwU', 'mgroger6@dmoz.org', 'I am a 35-year-old personal trainer who enjoys cookery, horse riding an
 (8, 'fmilbourne7', 'MVys1p90', 'breide7@cnbc.com', 'I am a 40-year-old intelligence researcher who enjoys meditation, badmi
 (9, 'bgrut8', 'JZBxw3', 'mabadam8@merriam-webster.com', 'I am a 26-year-old former town counsellor who enjoys watching YouT
 (10, 'bholbury9', 'dhyqgCJj', 'vguerner9@businesswire.com', 'I am a 70-year-old former clerk who enjoys reading, theatre an

INSERT INTO
        news(id_news, writer, title, content, image, dates, reputation)
VALUES
 (1, 1, 'Religious discrimination bill introduced to parliament', 'Prime Minister Scott Morrison has introduced proposed law
 (2, 2, 'Thousands of NSW teachers to strike over unsustainable workloads and uncompetitive salaries', 'NSW public school te
 (3, 1, 'myGov website back online after widespread outage', 'The widespread outage that downed government services website
 (4, 1, 'The fear of missing out and the psychology of Black Friday', 'From its somewhat chequered origins in the 1950s when
 (5, 1, 'Fears mount over Russian invasion of Ukraine', 'The US may send military advisors and new weaponry to Ukraine as Ru
 (6, 1, 'Australian fashion label becomes nations first to be carbon neutral', 'Menswear brand MJ Bale has become the nation
 (7, 1, 'Britains army chief warns risk of accidental war with Russia is greater than during Cold War', 'The risk of an acci
 (8, 1, 'Millions of COVID-19 home testing kits made by Aussie company recalled in US', 'More than two million at-home COVID

INSERT INTO
        comments(id_comment, writer, news, content, dates, reputation)
VALUES
 (1, 2, 1, 'NEVER EVER vote for Labor or Liberal parties EVER Again.', '02/01/2021', 0),
 (2, 3, 1, 'What about medical discrimination!!!!!', '02/01/2021', 0),
 (3, 4, 1, 'How about a freedom of speech bill.', '02/01/2021', 0),
 (4, 5, 1, 'Just remember this bill was Morrisons entire ambition as prime minister and its a complete joke.', '02/01/2021',
 (5, 1, 1, 'Morrison is clearly not a Christian by his behaviour.', '02/01/2021', 0),
 (6, 6, 1, 'We are anything but a democracy.', '02/01/2021', 0),
 (7, 7, 1, 'Despite Scotty saying he is a Christian, his actions since early 2020 show that he is more of a devil worshipper
 (8, 8, 1, 'Are politicians needed?', '02/01/2021', 0),
 (9, 9, 1, 'Never voting mainstream parties again , time for a change and new start in Australia.', '02/01/2021', 0),
 (10, 10, 1, 'Nice speech. Took him his ENTIRE term to write it. Perspective.', '02/01/2021', 0);

INSERT INTO
    follow_member(follower, followed)
VALUES
 (6, 9),
 (3, 9),
 (8, 3),
 (2, 10),
 (1, 6);

INSERT INTO
    topic(id_topic, name)
VALUES
 (1, 'War'),
 (2, 'Government'),
 (3, 'Politics'),
 (4, 'Education'),
 (5, 'Health'),
 (6, 'The Environment'),
 (7, 'Economy'),
 (8, 'Business'),
 (9, 'Fashion'),
 (10, 'Entertainment');

INSERT INTO
    news_topic(id_news, topic)
VALUES
 (1, 3),
 (7, 1),
 (3, 2),
 (4, 7),
 (5, 1),
 (2, 4),
 (6, 9),
 (8, 5);

INSERT INTO
```

```
    follow_topic(member, topic)
VALUES
 (3, 1),
 (6, 1),
 (2, 2),
 (9, 5),
 (10, 8),
 (10, 7),
 (1, 7),
 (3, 2),
 (7, 3),
 (4, 4);

INSERT INTO
        vote_on_news(member, news, vote, dates)
VALUES
 (10, 1, 1, '01/01/2021'),
 (2, 1, 1, '01/01/2021'),
 (3, 1, 1, '01/01/2021'),
 (4, 1, -1, '01/01/2021'),
 (5, 1, 1, '01/01/2021');

INSERT INTO
        vote_on_comment(member, comments, vote, dates)
VALUES
 (1, 1, 1, '02/01/2021'),
 (5, 1, 1, '02/01/2021'),
 (3, 1, 1, '02/01/2021'),
 (4, 1, -1, '02/01/2021');
```

## Revision history

Nothing changed so far.

GROUP2103, 29/11/2021

- Diogo Pinto, up201906067@up.pt
- Guilherme Garrido, up201905407@up.pt
- Luís Lucas, up201904624@up.pt (Editor)
- Pedro Pinheiro, up201906788@up.pt