A6: Índices, gatilhos, procedimentos e povoamento

1. Carga da Base de Dados

1.1. Estimativa de Tuplos

Relação		Ordem de grandeza dos tuplos	Crescimento estimado (/dia)
R01	Localizacao	10^2 (centenas)	constante
R02	UserRole	3	constante
R03	RegistosModeracao	10^3 (milhares)	unidades
R04	Aviso	10^3 (milhares)	unidades
R05	Ban	10^1 (dezenas)	unidades
R06	Utilizador	10^6 (milhões)	dezenas
R07	UserBadge	10^6 (dezenas de milhão)	milhares
R08	Badge	10^1 (dezenas)	constante
R09	Publicacao	10^8 (centenas de milhão)	dezenas de milhar
R10	Voto	10^8 (centenas de milhão)	dezenas de milhar
R11	Pergunta	10^7 (dezenas de milhão)	milhares
R12	Resposta	10^7 (dezenas de milhão)	dezenas de milhar
R13	Comentario	10^7 (dezenas de milhão)	dezenas de milhar
R14	ComentarioPergunta	10^7 (dezenas de milhão)	dezenas de milhar
R15	ComentarioResposta	10^7 (dezenas de milhão)	dezenas de milhar
R16	Categoria	10^1 (dezenas)	unidades
R17	QuestionTag	10^6 (milhões)	milhares
R18	Tag	10^6 (milhões)	milhares

1.2. Interrogações mais Frequentes

SELECT01	Ver perfil pessoal

```
SELECT username, email, fullname, about, website, locationid, roleid
FROM users
WHERE userid = $1; -- passado por argumento
```

SELECT02	Respostas a uma pergunta

```
SELECT publicationid
FROM answers
WHERE answer.questionid = $1; -- passado por argumento
```

SELECT03	Perguntas não respondidas mais recentes

```
SELECT questionid
FROM questions, publications
WHERE questions.publicationid = publications.publicationid
 AND solved_date IS NULL
ORDER BY creation_date DESC
LIMIT 20
```

SELECT04	Obter o nome do autor de uma pergunta

```
SELECT users.username
FROM publications
 INNER JOIN questions ON publications.publicationid =
questions.publicationid
 INNER JOIN users ON publications.userid = users.userid
WHERE questions.publicationid = $1; -- passado por argumento
```

SELECT05	Obter as perguntas de um utilizador

```
SELECT questions.publicationid
FROM publications
 INNER JOIN questions ON publications.publicationid =
questions.publicationid
 INNER JOIN users ON publications.userid = users.userid
WHERE users.userid = $1; -- passado por argumento
```

1.3. Modificações mais Frequentes

UPDATE01	Aceitar respostas (frequente)

```
UPDATE answers
SET solved_date = now()
WHERE publicationid = '<answerid>'
```

UPDATE02	Edição de fullname (ocasional)

```
UPDATE users
SET fullname = '<newfullname>'
WHERE userid = '<userid>'
```

UPDATE03	Edição de password (ocasional)

```
UPDATE users
SET password = '<newpassword>'
WHERE userid = '<userid>'
```

UPDATE04	Edição de about (ocasional)

```
UPDATE users
SET about = '<newabout>'
WHERE userid = '<userid>'
```

UPDATE05	Edição de website (ocasional)

```
UPDATE users
SET website = '<newwebsite>'
WHERE userid = '<userid>
```

DELETE01	Apagar user (raro)

```
DELETE users
WHERE userid = '<userid>'
```

DELETE02	Apagar publicação (raro)

```
DELETE publications
WHERE publicationid = '<publicationid>'
```

2. Índices Propostos

As escolhas dos índices são importantes no que toca à optimização da base de dados, para que futuros problemas de escalabilidade sejam minimizados. Para isso criamos alguns indices que nos vão optimizar certas funcionalidades como a *Full Text Search*.

Full Text Search

Para a pesquisa **FTS** foi implementado um index *GIN*, dado que as colunas não vão ser actualizadas muitas vezes e em termos de performance é melhor que *GiST*. A criação deste índice, constitui os seguintes passos:

Criação da coluna full_text_index_col

```
ALTER TABLE questions ADD COLUMN full_text_index_col tsvector;
UPDATE questions
SET full_text_index_col = to_tsvector('english', COALESCE(title,'') || ' '
|| COALESCE(body,''));
```

Atualização sempre que uma das colunas que o constituem também for atualizada

```
CREATE TRIGGER questions_tsvector_update_trigger
BEFORE INSERT OR UPDATE ON questions
FOR EACH ROW
EXECUTE PROCEDURE
tsvector_update_trigger_column('full_text_index_col','pg_catalog.english',
'title', 'body');
```

Criação de dois index GIN, um na coluna criada (full_text_index_col) e outro na coluna body da tabela answers

```
CREATE INDEX questions_question_search_idx ON questions
USING gin(full_text_index_col);
CREATE INDEX answers_question_search_idx ON answers USING gin(body);
```

Função para obter as questões filtradas

```
CREATE OR REPLACE FUNCTION search_questions(psearch text)
RETURNS TABLE (questionid INTEGER) AS $func$
BEGIN
  RETURN QUERY
  SELECT DISTINCT publications.publicationid
  FROM questions, publications
  WHERE to_tsvector(COALESCE(questions.title,'') || ' ' ||
COALESCE(publications.body,'')) @@ to_tsquery(psearch)
        OR questionid IN (
    SELECT DISTINCT(answers.questionid) FROM answers INNER JOIN publications
ON answers.publicationid = publications.publicationid
    WHERE to_tsvector(COALESCE(publications.body)) @@ to_tsquery(psearch)
  )
  ;
```

```
END
$func$  LANGUAGE plpgsql;
```

Outros índices

```
CREATE INDEX publications_search_idx ON publications USING
gin(to_tsvector('english', body));
CREATE INDEX questions_question_search_idx ON questions USING
gin(to_tsvector('english', COALESCE(title)));
```

Btree index

Na tabela users há colunas que beneficiam deste índice pois serão obtidos os seus dados através de igualdades. Optou-se por uma btree devido à sua eficiente implementação no PostgreSQL.

```
CREATE INDEX users_username ON users USING btree(username);
CREATE INDEX users_email ON users USING btree(email);
CREATE INDEX ixfk_modregister_target_user ON modregisters USING
btree(userid_target);
CREATE INDEX ixfk_modregister_author_user ON modregisters USING
btree(userid_author);
CREATE INDEX questions_updated_at ON publications USING
btree(last_edit_date);
CREATE INDEX ixfk_publications_users ON publications USING btree(userid);
CREATE INDEX ixfk_answers_questions ON answers USING btree(questionid);
CREATE INDEX ixfk_user_votes ON votes USING btree(userid);
CREATE INDEX ixfk_questions_tags_tags ON questiontags USING btree(tagid);
CREATE INDEX ixfk_user_badges ON userbadges USING btree(badgeid);
```

3. Gatilhos

RN01	Marcar pergunta como resolvida

`CREATE TRIGGER t_mark_question_as_solved`
`BEFORE UPDATE OF solved_date ON answers`
`EXECUTE PROCEDURE mark_question_as_resolved();`	

RN02	Verificar se pergunta já tinha sido resolvida

`CREATE TRIGGER t_check_question_is_solved`
`BEFORE UPDATE OF solved_date ON questions`
`EXECUTE PROCEDURE check_question_is_solved();`	

RN03	Actualização do timestamp de uma pergunta

`CREATE TRIGGER answer_update_question_timestamp`
`BEFORE INSERT OR UPDATE ON publications`
`FOR EACH ROW`
`EXECUTE PROCEDURE trigger_update_question_timestamp();`	

RN04	Uma publicação do tipo pergunta não pode ser de outro tipo

```
CREATE TRIGGER check_question
BEFORE INSERT ON questions
FOR EACH ROW
EXECUTE PROCEDURE check_question();
```

RN05	Uma publicação do tipo resposta não pode ser de outro tipo

```
CREATE TRIGGER check_answer
BEFORE INSERT ON answers
FOR EACH ROW
EXECUTE PROCEDURE check_answer();
```

RN06	Uma publicação do tipo comentário não pode ser de outro tipo

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

RN07	Um utilizador é banido após atingir um determinado número de warnings

```
CREATE TRIGGER auto_ban_on_warning_limit
AFTER INSERT ON warnings
FOR EACH ROW
EXECUTE PROCEDURE trigger_auto_ban_on_warning_limit();
```

RN08	Após um voto, os rankings dos utilizadores são actualizados

```
CREATE TRIGGER auto_rank_up
AFTER INSERT OR UPDATE ON votes
FOR EACH ROW
EXECUTE PROCEDURE user_badges_ranking();
```

RN09	Um utilizador não pode votar no seu próprio conteúdo

```
CREATE TRIGGER own_content_vote_trigger
AFTER INSERT OR UPDATE ON votes
FOR EACH ROW
EXECUTE PROCEDURE own_content_vote();
```

RN10	Uma pergunta não pode ser uma resposta ou comentário

```
CREATE TRIGGER check_question
BEFORE INSERT ON questions
FOR EACH ROW
EXECUTE PROCEDURE check_question();
```

RN11	Uma resposta não pode ser uma pergunta ou comentário

```
CREATE TRIGGER check_answer
BEFORE INSERT ON answers
FOR EACH ROW
EXECUTE PROCEDURE check_answer();
```

RN12	Um comentário não pode ser uma perfunta ou resposta

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

4. Código SQL

TABLES / TRIGGERS / USER FUNCTIONS / INDEXES

lbaw1641_create.sql

```
CREATE TABLE badges
(
    badgeid SERIAL PRIMARY KEY,
    name VARCHAR(50),
    description VARCHAR(100) NOT NULL,
    CONSTRAINT badge_description CHECK(CHAR_LENGTH(description) >= 2
AND CHAR_LENGTH(description) <= 100)
);

CREATE TABLE categories
(
    categoryid SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    CONSTRAINT valid_category CHECK(CHAR_LENGTH(name) >= 3 AND
CHAR_LENGTH(name) <= 50)
);

CREATE TABLE userroles
(
    roleid SERIAL PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    CONSTRAINT user_role CHECK(name IN ('Admin', 'Editor',
'Authenticated'))
);

CREATE TABLE locations
(
    locationid SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL
);

CREATE TABLE users
(
    userid SERIAL PRIMARY KEY,
    username VARCHAR(50) NOT NULL,
    email VARCHAR(70) NOT NULL,
    password VARCHAR(50) NOT NULL,
    fullname VARCHAR(200),
    about VARCHAR(500),
    website VARCHAR(150),
    signup_date DATE DEFAULT CURRENT_DATE NOT NULL,
    last_login TIMESTAMP,
    locationid INTEGER,
    roleid INTEGER,
    CONSTRAINT valid_date CHECK(last_login > signup_date),
    CONSTRAINT valid_password CHECK(CHAR_LENGTH(password) >= 6 AND
CHAR_LENGTH(password) < 50),
    CONSTRAINT valid_username CHECK(CHAR_LENGTH(username) >= 1 AND
CHAR_LENGTH(username) < 20),
    CONSTRAINT valid_fullname CHECK(CHAR_LENGTH(fullname) >= 6 AND
CHAR_LENGTH(fullname) <= 50),
    CONSTRAINT valid_email CHECK(CHAR_LENGTH(email) >= 6 AND
CHAR_LENGTH(email) <= 50),
    CONSTRAINT "FK_User_Location"
    FOREIGN KEY ("locationid") REFERENCES locations ("locationid") ON
DELETE SET NULL ON UPDATE CASCADE,
    CONSTRAINT "FK_User_UserRole"
    FOREIGN KEY ("roleid") REFERENCES userroles ("roleid") ON DELETE
SET NULL ON UPDATE CASCADE
);

CREATE INDEX users_username ON users USING btree(username);
CREATE INDEX users_email ON users USING btree(email);

CREATE TABLE modregisters
(
    modregisterid SERIAL PRIMARY KEY,
    date_creation TIMESTAMP DEFAULT now() NOT NULL,
    reason VARCHAR(200) NOT NULL,
    userid_author INTEGER NOT NULL,
    userid_target INTEGER NOT NULL,
    CONSTRAINT author
    FOREIGN KEY ("userid_author") REFERENCES users ("userid") ON DELETE
SET NULL ON UPDATE CASCADE,
    CONSTRAINT target
    FOREIGN KEY ("userid_target") REFERENCES users ("userid") ON DELETE
CASCADE ON UPDATE CASCADE
);

CREATE INDEX ixfk_modregister_target_user ON modregisters USING btree
(userid_target);
CREATE INDEX ixfk_modregister_author_user ON modregisters USING btree
(userid_author);

CREATE TABLE warnings
(
    warningid SERIAL PRIMARY KEY,
    CONSTRAINT "FK_warnings_modregisters"
    FOREIGN KEY ("warningid") REFERENCES modregisters ("modregisterid")
ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE bans
(
    banid SERIAL PRIMARY KEY,
    end_date TIMESTAMP,
    CONSTRAINT "FK_Ban_ModRegister"
    FOREIGN KEY ("banid") REFERENCES modregisters ("modregisterid") ON
DELETE CASCADE ON UPDATE CASCADE
);
```

```
CREATE TABLE publications
(
    publicationid SERIAL PRIMARY KEY,
    body text NOT NULL ,
    creation_date TIMESTAMP DEFAULT now() NOT NULL,
    userid INTEGER NOT NULL,
    last_edit_date TIMESTAMP,
    CONSTRAINT body_length CHECK (CHAR_LENGTH(body) >= 10 AND
CHAR_LENGTH(body) <= 1000),
    CONSTRAINT "FK_publications_users"
    FOREIGN KEY ("userid") REFERENCES users ("userid") ON DELETE SET
NULL ON UPDATE CASCADE
);

CREATE INDEX publications_search_idx ON publications USING
gin(to_tsvector('english', body));
CREATE INDEX questions_updated_at ON publications USING
btree(last_edit_date);
CREATE INDEX ixfk_publications_users ON publications USING btree
(userid);

CREATE TABLE questions
(
    publicationid SERIAL PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    categoryid INTEGER NOT NULL,
    solved_date TIMESTAMP,
    CONSTRAINT title_length CHECK (CHAR_LENGTH(title) >= 3 AND
CHAR_LENGTH(title) <= 50),
    CONSTRAINT "FK_Question_Category"
    FOREIGN KEY ("categoryid") REFERENCES categories ("categoryid") ON
DELETE SET NULL ON UPDATE CASCADE,
    CONSTRAINT "FK_Question_Publication"
    FOREIGN KEY ("publicationid") REFERENCES publications
("publicationid") ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE INDEX questions_question_search_idx ON questions USING
gin(to_tsvector('english', COALESCE(title)));

CREATE TABLE comments
(
    publicationid SERIAL PRIMARY KEY,
    CONSTRAINT "FK_Comment_Publication"
    FOREIGN KEY ("publicationid") REFERENCES publications
("publicationid") ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE questioncomments
(
    commentid SERIAL PRIMARY KEY,
    questionid INTEGER NOT NULL,
    CONSTRAINT "FK_questioncomments_comments"
    FOREIGN KEY ("commentid") REFERENCES comments ("publicationid") ON
DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT "FK_questioncomments_questions"
    FOREIGN KEY ("questionid") REFERENCES questions ("publicationid")
ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE answers
(
    publicationid SERIAL PRIMARY KEY,
    questionid INTEGER NOT NULL,
    solved_date TIMESTAMP,
    CONSTRAINT "FK_answers_questions"
    FOREIGN KEY ("questionid") REFERENCES questions ("publicationid")
ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT "FK_answers_publications"
    FOREIGN KEY ("publicationid") REFERENCES publications
("publicationid") ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE INDEX ixfk_answers_questions ON answers USING btree
(questionid);

CREATE TABLE answercomments
(
    commentid SERIAL PRIMARY KEY,
    answerid INTEGER NOT NULL,
    CONSTRAINT "FK_answercomments_comments"
    FOREIGN KEY ("commentid") REFERENCES comments ("publicationid") ON
DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT "FK_answercomments_answers"
    FOREIGN KEY ("answerid") REFERENCES answers ("publicationid") ON
DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE tags
(
    tagid SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    CONSTRAINT valid_tag CHECK(CHAR_LENGTH(name) >= 3 AND
CHAR_LENGTH(name) <= 30)
);

CREATE TABLE votes
(
    voteid SERIAL PRIMARY KEY,
    VALUES INTEGER DEFAULT  NOT NULL,
    publicationid INTEGER NOT NULL,
    userid INTEGER NOT NULL, -- user that voted
    CONSTRAINT vote_values CHECK(VALUES =  OR VALUES = 1 OR VALUES =
-1),
    CONSTRAINT "FK_Vote_Publication"
    FOREIGN KEY ("publicationid") REFERENCES publications
("publicationid") ON DELETE CASCADE ON UPDATE CASCADE,
    CONSTRAINT "FK_Vote_User"
    FOREIGN KEY ("userid") REFERENCES users ("userid") ON DELETE
CASCADE ON UPDATE CASCADE
);

CREATE INDEX ixfk_user_votes ON votes USING btree (userid);

CREATE TABLE questiontags (
    questionid INTEGER NOT NULL,
    tagid INTEGER NOT NULL,
    PRIMARY KEY(questionid,tagid),
    CONSTRAINT "Tag"
    FOREIGN KEY ("tagid") REFERENCES tags ("tagid") ON DELETE CASCADE
ON UPDATE CASCADE,
    CONSTRAINT "Question"
    FOREIGN KEY ("questionid") REFERENCES questions ("publicationid")
ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE INDEX ixfk_questions_tags_tags ON questiontags USING btree
(tagid);

CREATE TABLE userbadges (
    userid INTEGER NOT NULL,
    badgeid INTEGER NOT NULL,
    PRIMARY KEY(userid,badgeid),
    CONSTRAINT "Badge"
    FOREIGN KEY ("badgeid") REFERENCES badges ("badgeid") ON DELETE
CASCADE ON UPDATE CASCADE,
    CONSTRAINT "User"
    FOREIGN KEY ("userid") REFERENCES users ("userid") ON DELETE
CASCADE ON UPDATE CASCADE
);


CREATE INDEX ixfk_user_badges ON userbadges USING btree (badgeid);



--------------------------FUNCTIONS-------------------------------------
----

---- Function that returns the tags of a question
CREATE OR REPLACE FUNCTION question_tags(pquestion_id INT)
    RETURNS TABLE (tag CHARACTER VARYING(10)) AS $func$
```

```
BEGIN
    RETURN QUERY
    SELECT tags.name
    FROM tags INNER JOIN questiontags ON tags.tagid =
questiontags.tagid
    WHERE questiontags.questionid = pquestion_id;
END
$func$  LANGUAGE plpgsql;


---- Function that returns the answers of a question and aditional /
optional info
CREATE OR REPLACE FUNCTION question_answers(pquestion_id INT)
    RETURNS TABLE (
        id INTEGER,
        user_id INTEGER,
        username CHARACTER VARYING(50),
        body TEXT,
        created_at TIMESTAMP
    ) AS $func$
BEGIN
    RETURN QUERY
    SELECT answers.publicationid, users.userid, users.username,
publications.body, publications.creation_date
    FROM answers INNER JOIN publications ON answers.publicationid =
publications.publicationid
        RIGHT JOIN users ON publications.userid = users.userid
    WHERE answers.questionid = pquestion_id;
END
$func$  LANGUAGE plpgsql;


---- Function that returns the questions of a user
CREATE OR REPLACE FUNCTION user_questions(puser_id INT)
    RETURNS TABLE (
        publicationid INTEGER,
        title CHARACTER VARYING(100),
        body TEXT,
        solved_date TIMESTAMP,
        creation_date TIMESTAMP,
        last_edit_date TIMESTAMP,
        count_answers BIGINT

    ) AS $func$
BEGIN
    RETURN QUERY
    SELECT questions.publicationid, questions.title, publications.body,
questions.solved_date, publications.creation_date,
        publications.last_edit_date, (SELECT COUNT(*) FROM answers
WHERE questionid = questions.publicationid)
    FROM questions INNER JOIN publications ON questions.publicationid =
publications.publicationid
    WHERE publications.userid = puser_id;
END
$func$  LANGUAGE plpgsql;


---- Function that counts the votes one user received
CREATE OR REPLACE FUNCTION count_vote_rating_received_user(puser_id
INT)
    RETURNS INTEGER AS $func$
DECLARE publicationvotecount INTEGER;
BEGIN
    SELECT COUNT(*) FROM votes INNER JOIN publications ON
votes.publicationid = publications.publicationid
        RIGHT JOIN users ON publications.userid = users.userid WHERE
users.userid = puser_id
    INTO publicationvotecount;

    IF publicationvotecount IS NULL THEN
        publicationvotecount := ;
    END IF;

    RETURN publicationvotecount;
END
$func$  LANGUAGE plpgsql;


---- Function that returns important info about one user puser_id
CREATE OR REPLACE FUNCTION user_profile(puser_id INT)
    RETURNS TABLE (
        username CHARACTER VARYING(50),
        email CHARACTER VARYING(100),
        TYPE CHARACTER VARYING(10),
        badge CHARACTER VARYING(50),
        created_at DATE,
        count_votes_rating_received INT,
        count_questions BIGINT,
        count_answers BIGINT,
        count_votes_made BIGINT
    ) AS $func$
BEGIN
    RETURN QUERY
    SELECT users.username, users.email,
        (SELECT name FROM users INNER JOIN userroles ON users.roleid =
userroles.roleid WHERE userid = puser_id),
        users.signup_date,
        count_vote_rating_received_user(puser_id),
        (SELECT COUNT(*) FROM publications INNER JOIN questions ON
questions.publicationid = publications.publicationid
            RIGHT JOIN users ON publications.userid = users.userid
WHERE users.userid = puser_id),
        (SELECT COUNT(*) FROM publications INNER JOIN answers ON
answers.publicationid = publications.publicationid
            RIGHT JOIN users ON publications.userid = users.userid
WHERE users.userid = puser_id),
        (SELECT COUNT(*) FROM votes WHERE votes.userid = puser_id)
    FROM users
    WHERE users.userid = puser_id;
END
$func$  LANGUAGE plpgsql;


---- Function that creates a new register on User Badges table
associating the User total points to the badge he deserves
CREATE OR REPLACE FUNCTION user_badges_ranking()
    RETURNS TRIGGER AS $func$
DECLARE target_user INTEGER;
BEGIN
    SELECT publications.userid FROM publications INNER JOIN votes ON
publications.publicationid = votes.publicationid
    WHERE publications.publicationid = NEW.publicationid INTO
target_user;
    IF count_vote_rating_received_user(target_user) = 1 THEN
        INSERT INTO userbadges(userid, badgeid) VALUES (target_user,
1);
    END IF;
    IF count_vote_rating_received_user(target_user) = 3 THEN
        INSERT INTO userbadges(userid, badgeid) VALUES (target_user,
2);
    END IF;
    IF count_vote_rating_received_user(target_user) = 15 THEN
        INSERT INTO userbadges(userid, badgeid) VALUES (target_user,
3);
    END IF;
    IF count_vote_rating_received_user(target_user) = 30 THEN
        INSERT INTO userbadges(userid, badgeid) VALUES (target_user,
4);
    END IF;
    IF count_vote_rating_received_user(target_user) = 50 THEN
        INSERT INTO userbadges(userid, badgeid) VALUES (target_user,
5);
    END IF;
    RETURN NULL;
END
$func$ LANGUAGE plpgsql;

--
DROP TRIGGER IF EXISTS t_mark_question_as_solved ON public.answers;
CREATE OR REPLACE FUNCTION mark_question_as_solved() RETURNS TRIGGER AS
$$
BEGIN
  UPDATE questions
  SET solved_date = NEW.solved_date
  WHERE NEW.questionid = question.publicationid
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER t_mark_question_as_solved
BEFORE UPDATE OF solved_date ON answers
EXECUTE PROCEDURE mark_question_as_resolved();


--
DROP TRIGGER IF EXISTS t_check_question_is_solved ON public.questions;
CREATE OR REPLACE FUNCTION check_question_is_solved() RETURNS TRIGGER
AS $$
BEGIN
 IF (OLD.solved_date IS NOT NULL) THEN
 RAISE EXCEPTION 'Question already solved!';
END IF;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER t_check_question_is_solved
BEFORE UPDATE OF solved_date ON questions
EXECUTE PROCEDURE check_question_is_solved();

---- Trigger that updates ranking
CREATE TRIGGER auto_rank_up AFTER INSERT OR UPDATE ON votes
FOR EACH ROW EXECUTE PROCEDURE user_badges_ranking();


---- Function that checks if the user is voting in his own content
CREATE OR REPLACE FUNCTION own_content_vote()
    RETURNS TRIGGER AS $func$
DECLARE target_user INTEGER;
BEGIN
    SELECT publications.userid FROM publications INNER JOIN votes ON
publications.publicationid = votes.publicationid
    WHERE publications.publicationid = NEW.publicationid INTO
target_user;
    IF target_user = NEW.userid THEN
        RAISE EXCEPTION 'You cant vote on your own publications';
    END IF;
    RETURN NULL;
END
$func$  LANGUAGE plpgsql;

CREATE TRIGGER own_content_vote_trigger AFTER INSERT OR UPDATE ON votes
    FOR EACH ROW EXECUTE PROCEDURE own_content_vote();




--------------------------------------------------------------------
```

```
---- This function adds a user to the ban table when he exceeds the
warning limit (3)
DROP TRIGGER IF EXISTS auto_ban_on_warning_limit ON public.warnings;
CREATE OR REPLACE FUNCTION trigger_auto_ban_on_warning_limit()
    RETURNS "trigger" AS $func$
BEGIN
    IF (SELECT COUNT(*)
        FROM modregisters INNER JOIN users ON
modregisters.userid_author = users.userid
            INNER JOIN warnings ON modregisters.modregisterid =
warnings.warningid
        GROUP BY userid_target) = 3 THEN
        INSERT INTO bans(banid) VALUES(NEW.warningid);
    END IF;
    RETURN NULL;
END;
$func$  LANGUAGE plpgsql;

CREATE TRIGGER auto_ban_on_warning_limit AFTER INSERT ON warnings
FOR EACH ROW EXECUTE PROCEDURE trigger_auto_ban_on_warning_limit();

--- This function updates the column last_edit_date with the current
timestamp everytime there is an update on the table
DROP TRIGGER IF EXISTS answer_update_question_timestamp ON
public.publications;
CREATE OR REPLACE FUNCTION trigger_update_question_timestamp()
    RETURNS TRIGGER AS $func$
BEGIN
    NEW.last_edit_date := now();
    RETURN NEW;
END;
$func$  LANGUAGE plpgsql;

CREATE TRIGGER answer_update_question_timestamp BEFORE INSERT OR UPDATE
ON publications
FOR EACH ROW EXECUTE PROCEDURE trigger_update_question_timestamp();


---- This function returns the username of the user of a given question
CREATE OR REPLACE FUNCTION getusernamefromquestion(questionid INTEGER)
    RETURNS VARCHAR AS $$
BEGIN
    SELECT users.username
    FROM publications
        INNER JOIN questions ON publications.publicationid =
questions.publicationid
        INNER JOIN users ON publications.userid = users.userid
    WHERE questions.publicationid = $1;
END;
```

```
$$ LANGUAGE plpgsql;

---- This is the Full text search function
CREATE OR REPLACE FUNCTION search_questions(psearch text)
    RETURNS TABLE (questionid INTEGER) AS $func$
BEGIN
    RETURN QUERY
    SELECT DISTINCT publications.publicationid
    FROM questions, publications
    WHERE to_tsvector(COALESCE(questions.title,'') || ' ' ||
COALESCE(publications.body,'')) @@ to_tsquery(psearch)
          OR questions.publicationid IN (
        SELECT DISTINCT(answers.questionid) FROM answers INNER JOIN
publications ON answers.publicationid = publications.publicationid
        WHERE to_tsvector(COALESCE(publications.body)) @@
to_tsquery(psearch)
    )
    ;
END
$func$  LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS check_question ON public.questions;
CREATE OR REPLACE FUNCTION check_question()
 RETURNS "trigger" AS $func$
BEGIN
 IF (
  SELECT COUNT(*)
  FROM answers, comments
  WHERE NEW.publicationid = answers.publicationid
   OR NEW.publicationid = comments.publicationid
  ) >  THEN
    RAISE EXCEPTION 'A question cant be an answer or a comment!';
  END IF;
  RETURN NULL;
END;
$func$  LANGUAGE plpgsql;

CREATE TRIGGER check_question
BEFORE INSERT ON questions
FOR EACH ROW
EXECUTE PROCEDURE check_question();

--
DROP TRIGGER IF EXISTS check_answer ON public.answers;
CREATE OR REPLACE FUNCTION check_answer()
 RETURNS "trigger" AS $func$
BEGIN
 IF (
  SELECT COUNT(*)
  FROM comments, questions
  WHERE NEW.publicationid = questions.publicationid
```

```
    OR NEW.publicationid = comments.publicationid
   ) >  THEN
    RAISE EXCEPTION 'An answer cant be a question or a comment!';
   END IF;
   RETURN NULL;
END;
$func$  LANGUAGE plpgsql;

CREATE TRIGGER check_answer
BEFORE INSERT ON answers
FOR EACH ROW
EXECUTE PROCEDURE check_answer();

--
DROP TRIGGER IF EXISTS check_comment ON public.comments;
CREATE OR REPLACE FUNCTION check_comment()
 RETURNS "trigger" AS $func$
BEGIN
 IF (
   SELECT COUNT(*)
   FROM answers, questions
   WHERE NEW.publicationid = questions.publicationid
    OR NEW.publicationid = answers.publicationid
   ) >  THEN
    RAISE EXCEPTION 'A comment cant be a question or an answer!';
   END IF;
   RETURN NULL;
END;
$func$  LANGUAGE plpgsql;

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

INSERTS

lbaw1641_data.sql

```
INSERT INTO badges(name, description) VALUES ('Newbie', 'New User. No
correct answers');
INSERT INTO badges(name, description) VALUES ('Starter', 'Made one
question');
INSERT INTO badges(name, description) VALUES ('Helper', 'Answered a
topic');
INSERT INTO badges(name, description) VALUES ('Loyal', 'Have one
question marked as accepted');
INSERT INTO badges(name, description) VALUES ('Master', '10 correct
answers');


INSERT INTO categories(name) VALUES ('Sports');
INSERT INTO categories(name) VALUES ('Science');
INSERT INTO categories(name) VALUES ('Informatic');
INSERT INTO categories(name) VALUES ('Technology');
INSERT INTO categories(name) VALUES ('Culture');
INSERT INTO categories(name) VALUES ('Others');
INSERT INTO categories(name) VALUES ('Travel');
INSERT INTO categories(name) VALUES ('Health');

INSERT INTO userroles(name) VALUES ('Authenticated');
INSERT INTO userroles(name) VALUES ('Admin');
INSERT INTO userroles(name) VALUES ('Editor');

INSERT INTO locations(name) VALUES ('Porto');
INSERT INTO locations(name) VALUES ('Viseu');
INSERT INTO locations(name) VALUES ('Sao Paulo');
INSERT INTO locations(name) VALUES ('Paredes');

INSERT INTO users (username, email, password, locationid, roleid)
VALUES ('editor', 'editor@editor.pt', '12345678', 1, 3);
INSERT INTO users (username, email, password, locationid, roleid)
VALUES ('warneduser', 'warneduser@user.pt', '12345678', 4, 1);
INSERT INTO users (username, email, password, locationid, roleid)
VALUES ('luistelmocosta', 'luistelmocosta@gmail.com', '12345678', 4,
2);
INSERT INTO users (username, email, password, locationid, roleid)
VALUES ('admin', 'admin@admin.pt', '12345678', 4, 2);
INSERT INTO users (username, email, password, locationid, roleid)
VALUES ('member', 'member@member.pt', '12345678', 3, 1);
INSERT INTO users (username, email, password, locationid, roleid)
VALUES ('ronaldo7', 'ronaldo@ronaldo.pt', '12345678', 3, 1);

INSERT INTO modregisters(reason, userid_author, userid_target) VALUES
('Offensive Speech', 3, 4);
INSERT INTO modregisters(reason, userid_author, userid_target) VALUES
('Wrong question structure', 3, 4);
INSERT INTO modregisters(reason, userid_author, userid_target) VALUES
('Bad username', 3, 4);
INSERT INTO modregisters(reason, userid_author, userid_target) VALUES
('Third Warning', 4, 4);

INSERT INTO warnings(warningid) VALUES (2);
INSERT INTO warnings(warningid) VALUES (3);

INSERT INTO bans(banid, end_date) VALUES (2, '2018-03-29 01:05:00');
INSERT INTO bans(banid, end_date) VALUES (4, '2018-04-04 01:05:00');

INSERT INTO publications(body, userid) VALUES ('When is the next full
moon?', 3);
INSERT INTO publications(body, userid) VALUES ('What is the cheapest
way to go to London?',3);
INSERT INTO publications(body, userid) VALUES ('Do I get pregnant from
sitting in public toilets?', 3);
INSERT INTO publications(body, userid) VALUES ('Whos the best football
player in the world?', 3);
INSERT INTO publications(body, userid) VALUES ('The next full moon will
be on 22nd June 2017', 4);
INSERT INTO publications(body, userid) VALUES ('You should explore
Ryanair/EasyJet flight promotions', 4);
INSERT INTO publications(body, userid) VALUES ('Please tell me youre
not serious, mate', 4);
INSERT INTO publications(body, userid) VALUES ('That question has no
rational answer. Both players are great! Enjoy them while you can!',
4);
INSERT INTO publications(body, userid) VALUES ('Cristiano Ronaldo only
thinks about himself', 5);
INSERT INTO publications(body, userid) VALUES ('Cristiano Ronaldo
carries his National Team', 6);

INSERT INTO questions(publicationid ,title, categoryid) VALUES (1,'Full
Moon', 6);
INSERT INTO questions(publicationid, title, categoryid) VALUES (2, 'Fly
to London', 7);
INSERT INTO questions(publicationid, title, categoryid) VALUES
(3,'Pregnancy Alarm', 8);
INSERT INTO questions(publicationid, title, categoryid) VALUES (4,'Best
FootballPlayer', 1);


INSERT INTO comments(publicationid) VALUES (9);
INSERT INTO comments(publicationid) VALUES (10);

INSERT INTO questioncomments(commentid, questionid) VALUES (9, 4);

INSERT INTO answers(publicationid, questionid) VALUES (5, 1);
INSERT INTO answers(publicationid, questionid) VALUES (6, 2);
INSERT INTO answers(publicationid, questionid) VALUES (7, 3);


INSERT INTO answercomments(commentid, answerid) VALUES (10, 5);

INSERT INTO tags(name) VALUES ('real madrid');
INSERT INTO tags(name) VALUES ('moon');
INSERT INTO tags(name) VALUES ('flights');
INSERT INTO tags(name) VALUES ('pregnancy');
INSERT INTO tags(name) VALUES ('health');
INSERT INTO tags(name) VALUES ('football');

INSERT INTO votes("value", publicationid, userid) VALUES (1, 1, 1);
INSERT INTO votes("value", publicationid, userid) VALUES (, 1, 2);
INSERT INTO votes("value", publicationid, userid) VALUES (-1, 2, 2);
INSERT INTO votes("value", publicationid, userid) VALUES (1, 2, 3);
INSERT INTO votes("value", publicationid, userid) VALUES (1, 3, 3);


INSERT INTO questiontags(questionid, tagid) VALUES (1, 2);
INSERT INTO questiontags(questionid, tagid) VALUES (2, 3);
INSERT INTO questiontags(questionid, tagid) VALUES (3, 4);
INSERT INTO questiontags(questionid, tagid) VALUES (3, 5);
INSERT INTO questiontags(questionid, tagid) VALUES (4, 1);
INSERT INTO questiontags(questionid, tagid) VALUES (4, 6);


INSERT INTO userbadges(userid, badgeid) VALUES (1, 1);
INSERT INTO userbadges(userid, badgeid) VALUES (1, 2);
INSERT INTO userbadges(userid, badgeid) VALUES (1, 3);
INSERT INTO userbadges(userid, badgeid) VALUES (1, 4);
INSERT INTO userbadges(userid, badgeid) VALUES (1, 5);
INSERT INTO userbadges(userid, badgeid) VALUES (2, 1);
INSERT INTO userbadges(userid, badgeid) VALUES (3, 1);
INSERT INTO userbadges(userid, badgeid) VALUES (3, 2);
INSERT INTO userbadges(userid, badgeid) VALUES (4, 1);
INSERT INTO userbadges(userid, badgeid) VALUES (4, 2);
INSERT INTO userbadges(userid, badgeid) VALUES (4, 3);
INSERT INTO userbadges(userid, badgeid) VALUES (4, 5);
```

Revisão

- actualizado
  - em vez de índices hash, foram usados índices btree
  - função search_questions (FTS)
  - função que retorna informação importante sobre um utilizador
- adicionado
  - função / trigger answer_update_question_timestamp
  - interrogação para obter nome do autor de uma pergunta
  - interrogação para obter as perguntas de um utilizador
  - índices para foreign keys
  - função para obter o nome do autor de uma pergunta
  - função para obter as perguntas de um utilizador
  - função para obter as tags de uma pergunta
  - função para obter as respostas a uma pergunta
  - função / trigger que conta os votos que um utilizador recebeu e atribui o badge apropriado
  - verificação se o utilizador está a votar no próprio conteúdo
  - função / trigger para banir o utilizador após 3 warnings
  - funções / triggers para assegurar que uma publicação tem apenas um tipo