ebd

Last edited by **JoAPSo Vitor Pereira Ventura** 8 months ago

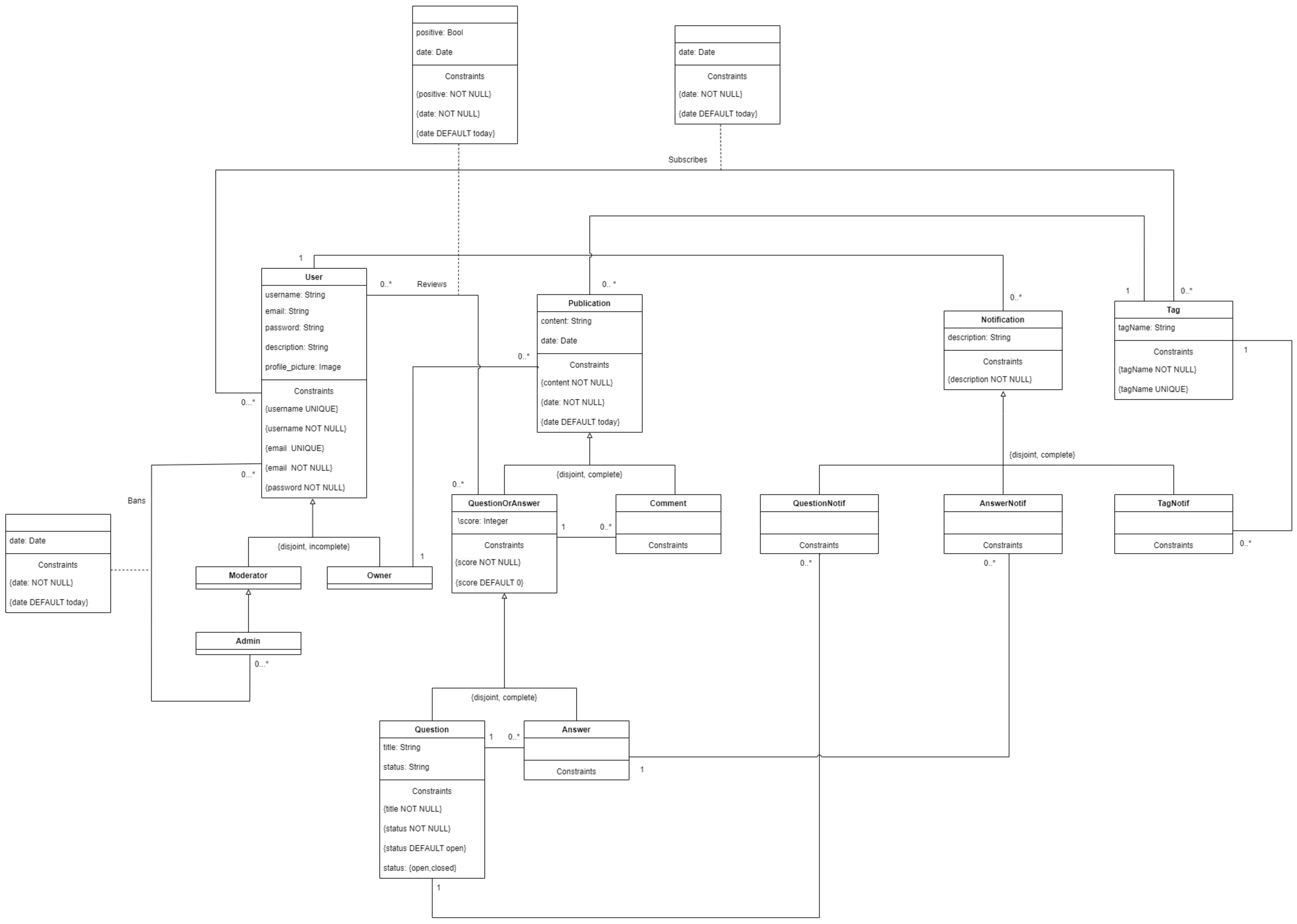
EBD: Database Specification Component

> Brain share seeks to use the internet's power to connect people towards developing knowledge collaboratively.

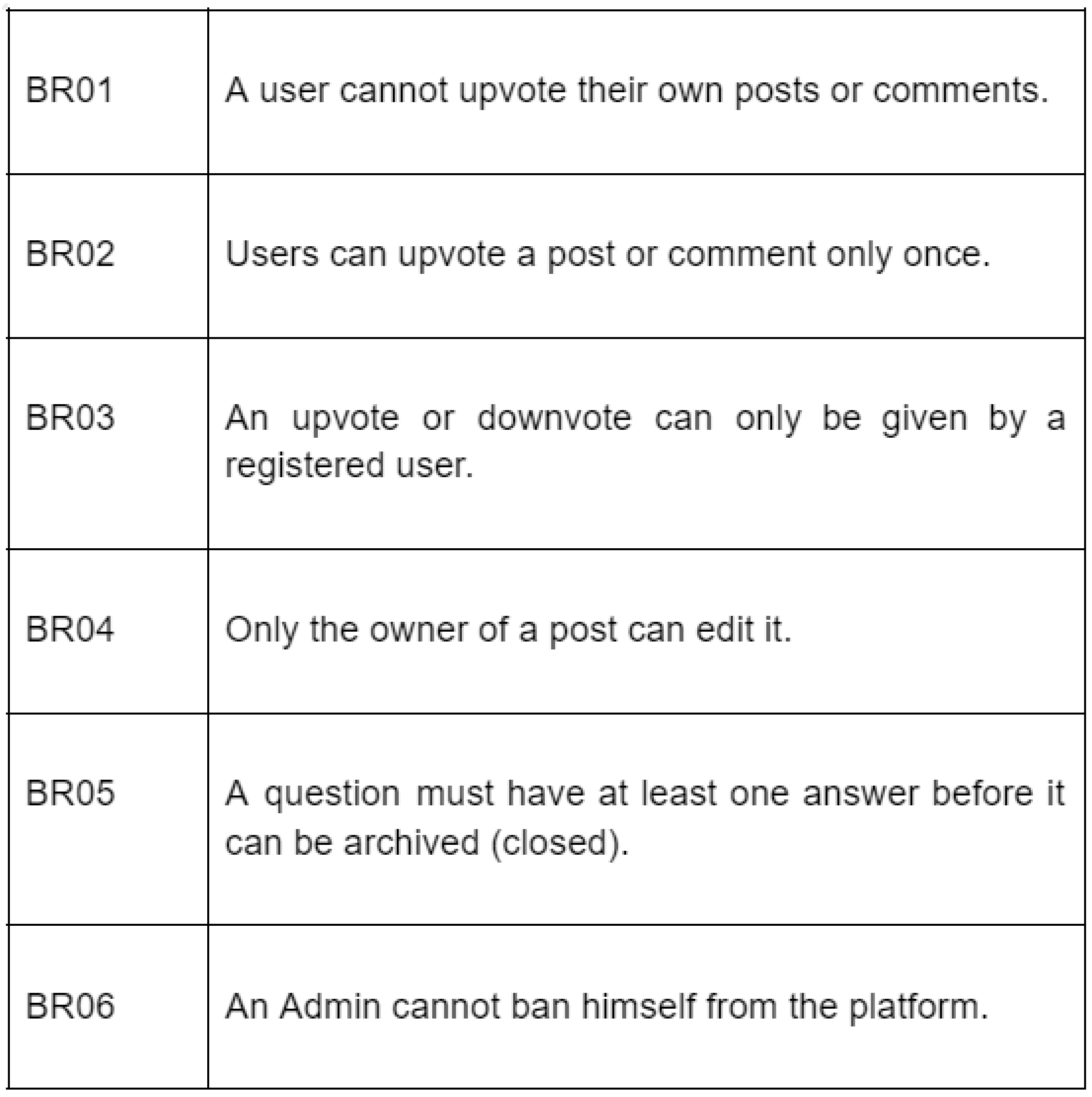
A4: Conceptual Data Model

> This artifact has the goal of visually representing the tables in the database by describing their components and the relationships between them.

1. Class diagram



2. Additional Business Rules



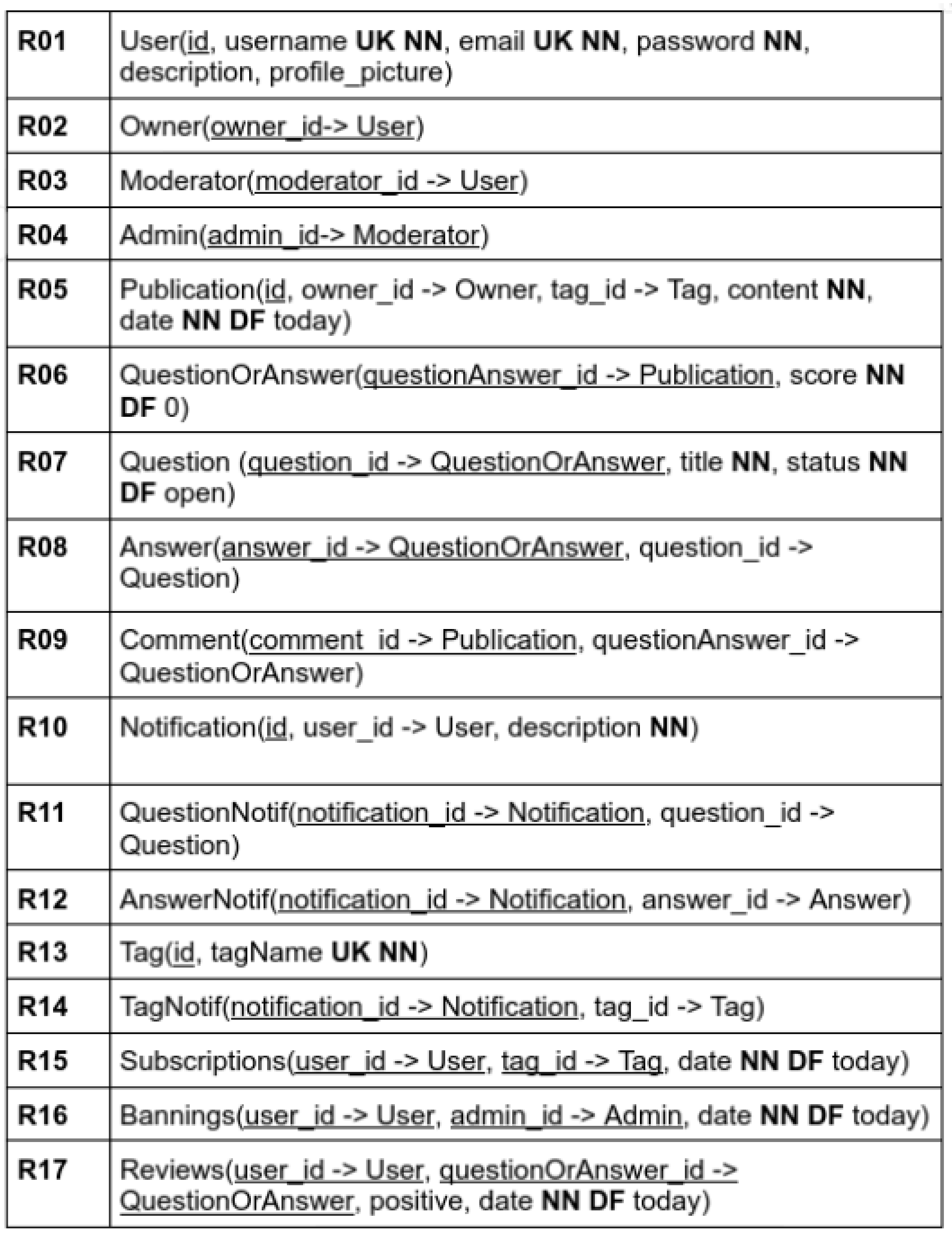
BR01	A user cannot upvote their own posts or comments.
BR02	Users can upvote a post or comment only once.
BR03	An upvote or downvote can only be given by a registered user.
BR04	Only the owner of a post can edit it.
BR05	A question must have at least one answer before it can be archived (closed).
BR06	An Admin cannot ban himself from the platform.

Table 1: UML Business Rules

A5: Relational Schema, validation and schema refinement

> We seek to initially create some artifacts to work on the creation of our database on a higher level to finally get to the base SQL code.

1. Relational Schema



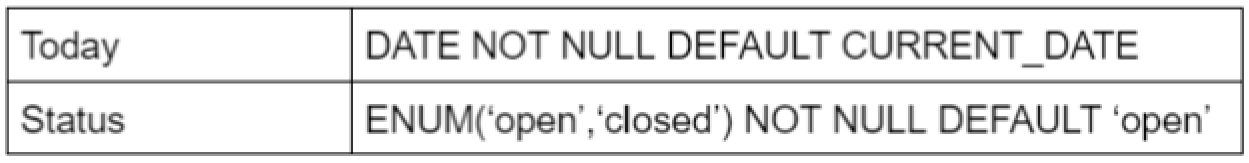
R01	User(id, username **UK NN**, email **UK NN**, password **NN**, description, profile_picture)
R02	Owner(owner_id-> User)
R03	Moderator(moderator_id -> User)
R04	Admin(admin_id-> Moderator)
R05	Publication(id, owner_id -> Owner, tag_id -> Tag, content **NN**, date **NN DF** today)
R06	QuestionOrAnswer(questionAnswer_id -> Publication, score **NN DF** 0)
R07	Question (question_id -> QuestionOrAnswer, title **NN**, status **NN DF** open)
R08	Answer(answer_id -> QuestionOrAnswer, question_id -> Question)
R09	Comment(comment_id -> Publication, questionAnswer_id -> QuestionOrAnswer)
R10	Notification(id, user_id -> User, description **NN**)
R11	QuestionNotif(notification_id -> Notification, question_id -> Question)
R12	AnswerNotif(notification_id -> Notification, answer_id -> Answer)
R13	Tag(id, tagName **UK NN**)
R14	TagNotif(notification_id -> Notification, tag_id -> Tag)
R15	Subscriptions(user_id -> User, tag_id -> Tag, date **NN DF** today)
R16	Bannings(user_id -> User, admin_id -> Admin, date **NN DF** today)
R17	Reviews(user_id -> User, questionOrAnswer_id -> QuestionOrAnswer, positive, date **NN DF** today)

Table 1: Relational Schema

> Legend:

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

2. Domains



Today	DATE NOT NULL DEFAULT CURRENT_DATE
Status	ENUM('open','closed') NOT NULL DEFAULT 'open'

3. Schema validation

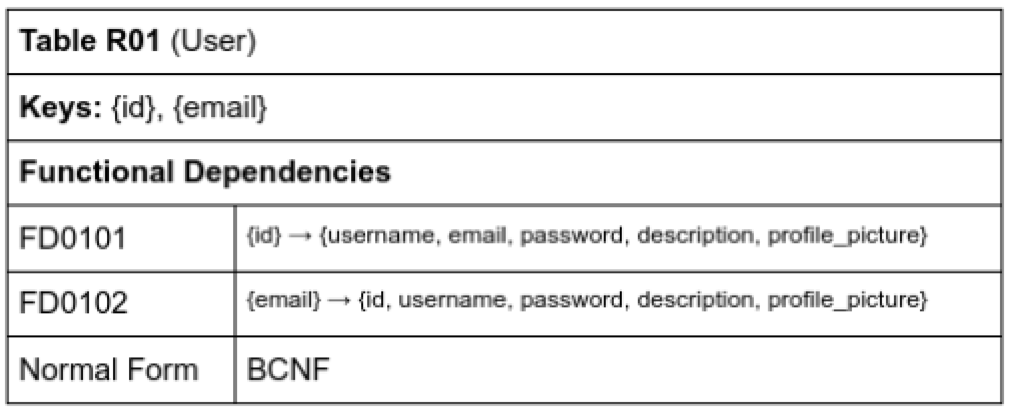


Table R01 (User)	
Keys: {id}, {email}	
Functional Dependencies	
FD0101	{id} → {username, email, password, description, profile_picture}
FD0102	{email} → {id, username, password, description, profile_picture}
Normal Form	BCNF

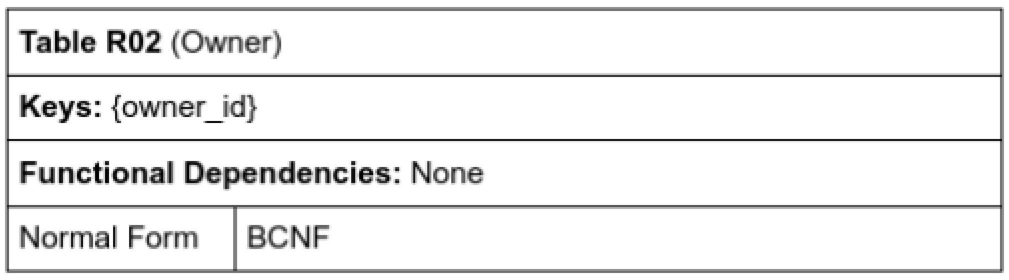


Table R02 (Owner)	
Keys: {owner_id}	
Functional Dependencies: None	
Normal Form	BCNF

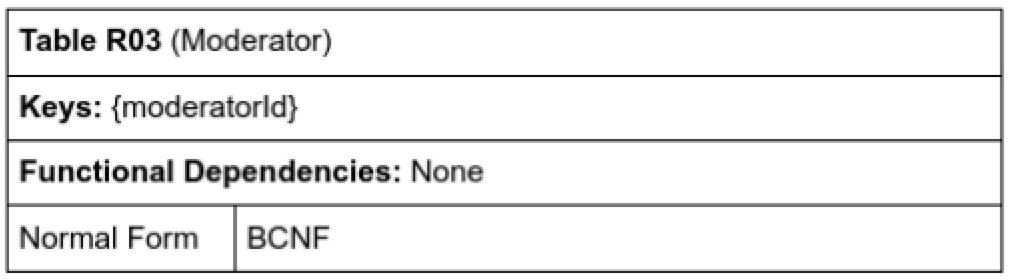


Table R03 (Moderator)	
Keys: {moderatorId}	
Functional Dependencies: None	
Normal Form	BCNF

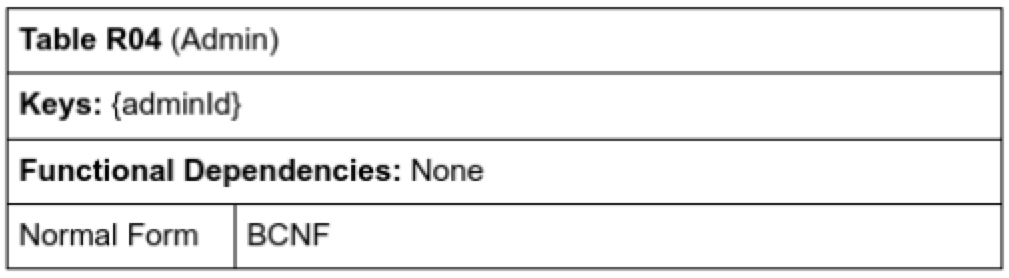


Table R04 (Admin)	
Keys: {adminId}	
Functional Dependencies: None	
Normal Form	BCNF

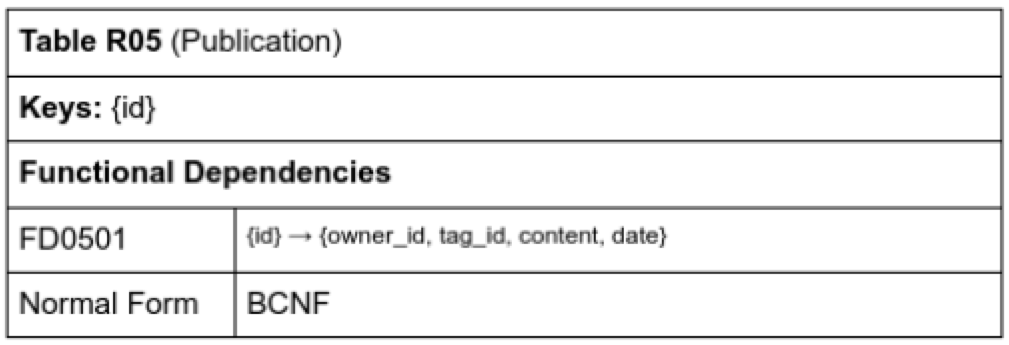


Table R05 (Publication)	
Keys: {id}	
Functional Dependencies	
FD0501	{id} → {owner_id, tag_id, content, date}
Normal Form	BCNF

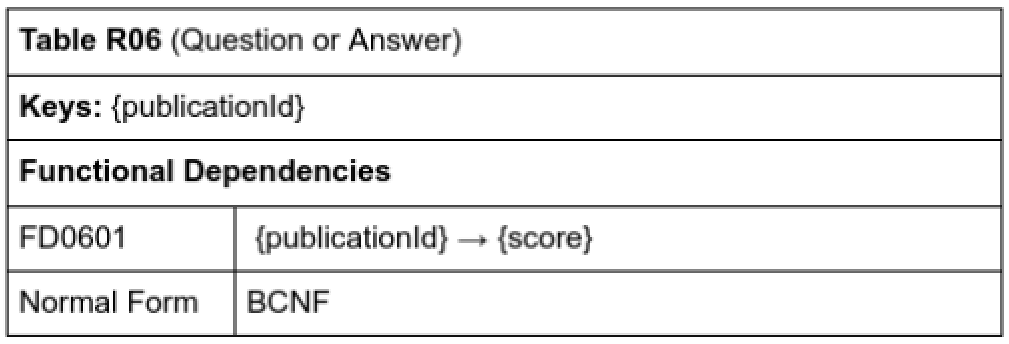


Table R06 (Question or Answer)	
Keys: {publicationId}	
Functional Dependencies	
FD0601	{publicationId} → {score}
Normal Form	BCNF

Table R07 (Question)	
Keys: {questionId}	
Functional Dependencies:	
FD0701	{questionId} → {title, status}
Normal Form	BCNF

Table R08 (Answer)	
Keys: {answerId}	
Functional Dependencies: None	
Normal Form	BCNF

Table R09 (Comment)	
Keys: {commentId}	
Functional Dependencies: None	
Normal Form	BCNF

Table R10 (Notification)	
Keys: {id}	
Functional Dependencies	
FD1001	{id} → {description}
Normal Form	BCNF

Table R11 (QuestionNotif)	
Keys: {notificationId}	
Functional Dependencies: None	
Normal Form	BCNF

Table R12 (AnswerNotif)	
Keys: {notificationId}	
Functional Dependencies: None	
Normal Form	BCNF

Table R13 (TagNotif)	
Keys: {notificationId}	
Functional Dependencies: None	
Normal Form	BCNF

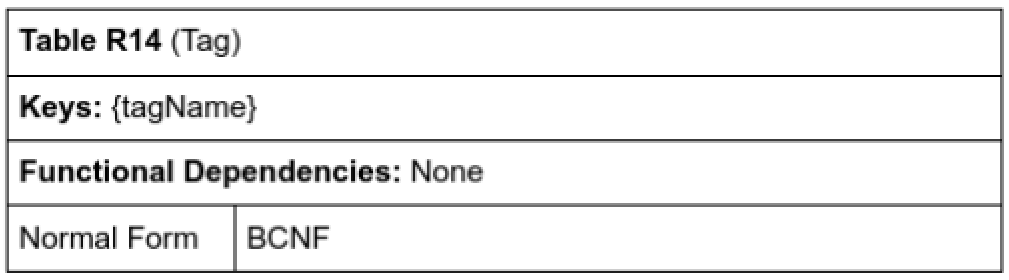


Table R14 (Tag)	
Keys: {tagName}	
Functional Dependencies: None	
Normal Form	BCNF

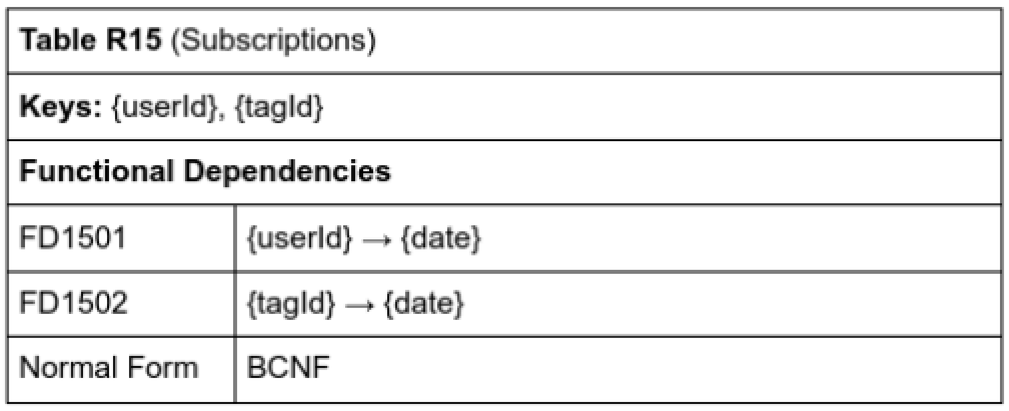


Table R15 (Subscriptions)	
Keys: {userId}, {tagId}	
Functional Dependencies	
FD1501	{userId} → {date}
FD1502	{tagId} → {date}
Normal Form	BCNF

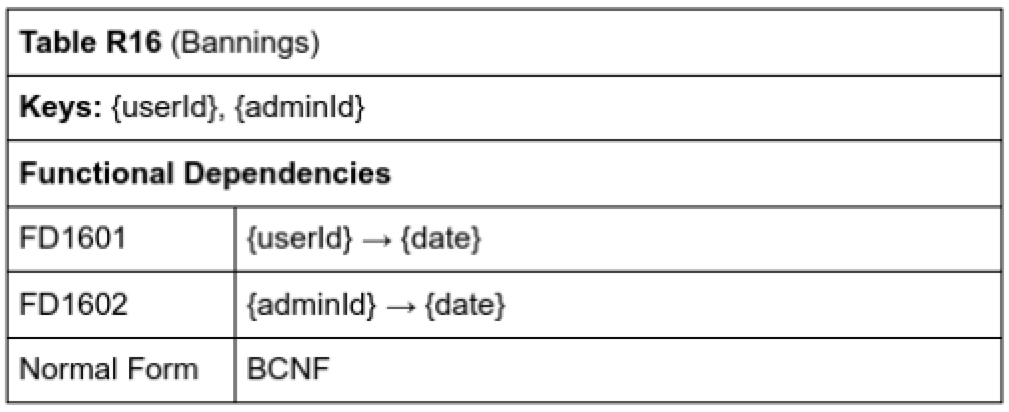


Table R16 (Bannings)	
Keys: {userId}, {adminId}	
Functional Dependencies	
FD1601	{userId} → {date}
FD1602	{adminId} → {date}
Normal Form	BCNF

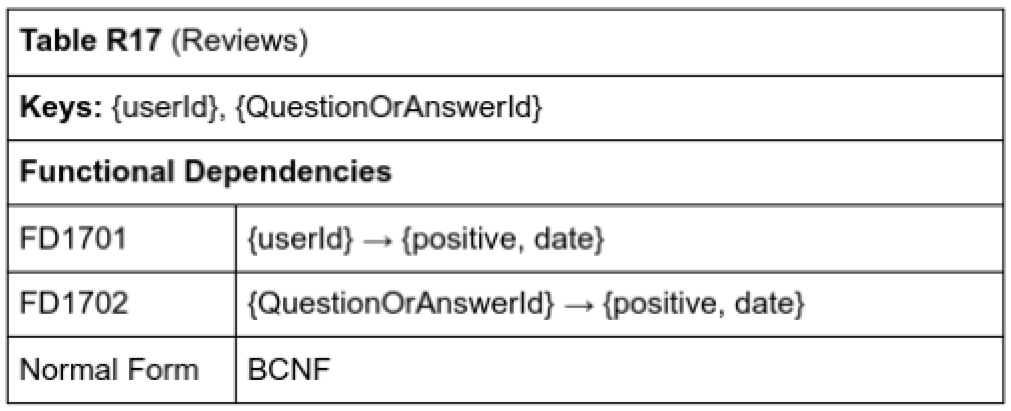


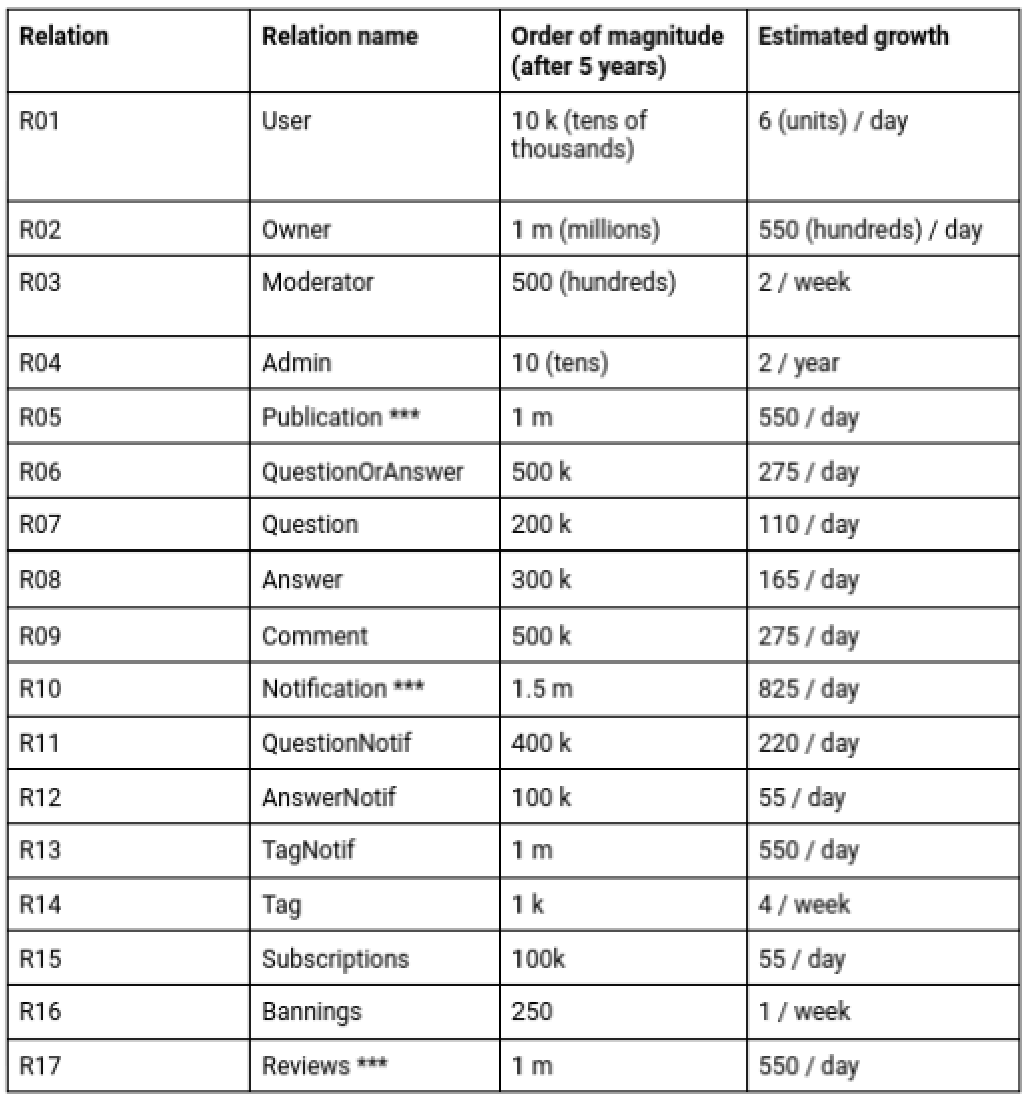
Table R17 (Reviews)	
Keys: {userId}, {QuestionOrAnswerId}	
Functional Dependencies	
FD1701	{userId} → {positive, date}
FD1702	{QuestionOrAnswerId} → {positive, date}
Normal Form	BCNF

> Since all relations adhere to the Boyce–Codd Normal Form (BCNF), the relational schema itself is inherently in BCNF, eliminating the necessity for additional normalization.

A6: Indexes, triggers, transactions and database population

> We seek to think the evolution of our project over time and develop further our SQL code to satisfy the needs we identify through this projection.

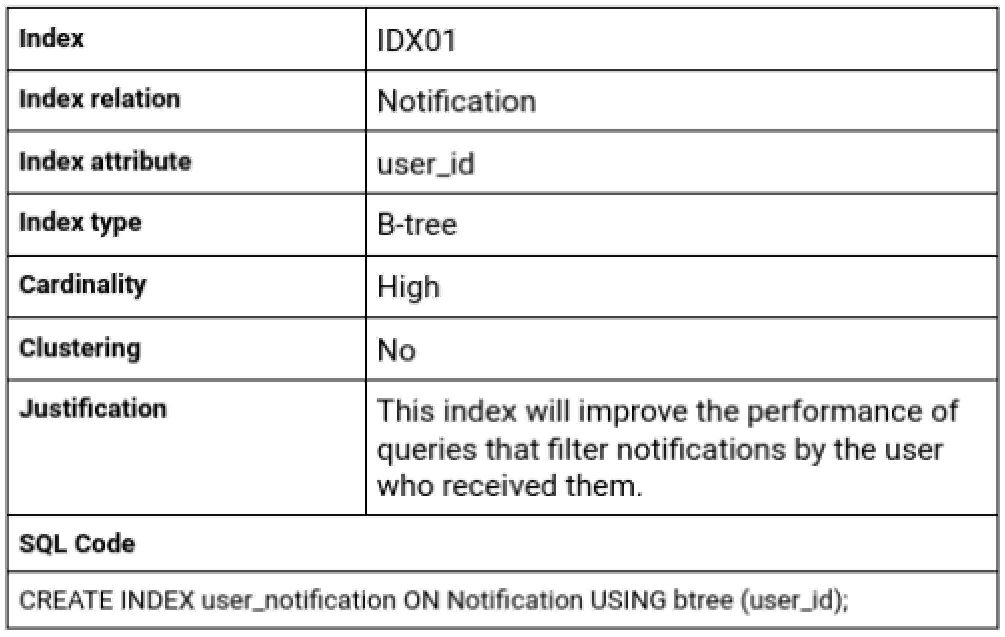
1. Database Workload



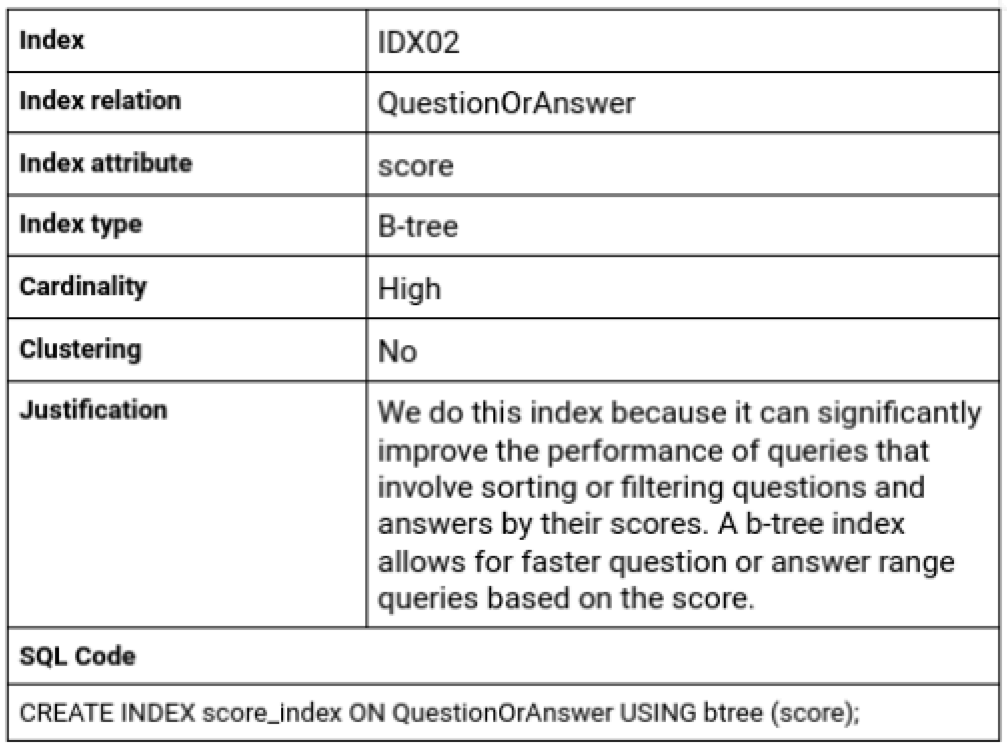
Relation	Relation name	Order of magnitude (after 5 years)	Estimated growth
R01	User	10 k (tens of thousands)	6 (units) / day
R02	Owner	1 m (millions)	550 (hundreds) / day
R03	Moderator	500 (hundreds)	2 / week
R04	Admin	10 (tens)	2 / year
R05	Publication ***	1 m	550 / day
R06	QuestionOrAnswer	500 k	275 / day
R07	Question	200 k	110 / day
R08	Answer	300 k	165 / day
R09	Comment	500 k	275 / day
R10	Notification ***	1.5 m	825 / day
R11	QuestionNotif	400 k	220 / day
R12	AnswerNotif	100 k	55 / day
R13	TagNotif	1 m	550 / day
R14	Tag	1 k	4 / week
R15	Subscriptions	100k	55 / day
R16	Bannings	250	1 / week
R17	Reviews ***	1 m	550 / day

2. Proposed Indices

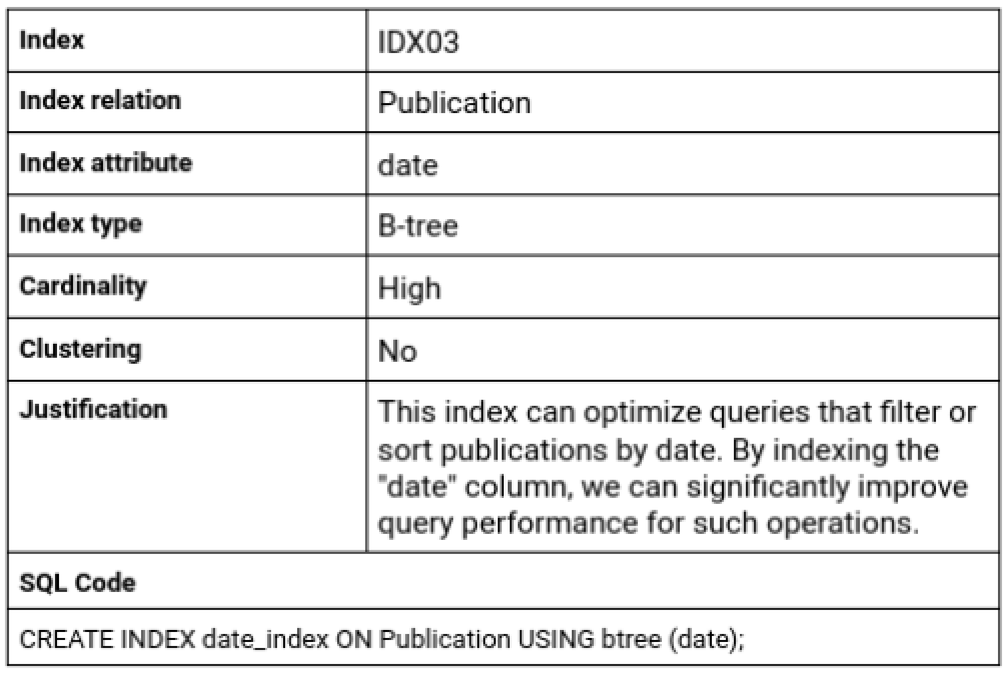
2.1. Performance Indices



Index	IDX01
Index relation	Notification
Index attribute	user_id
Index type	B-tree
Cardinality	High
Clustering	No
Justification	This index will improve the performance of queries that filter notifications by the user who received them.
SQL Code	
CREATE INDEX user_notification ON Notification USING btree (user_id);	



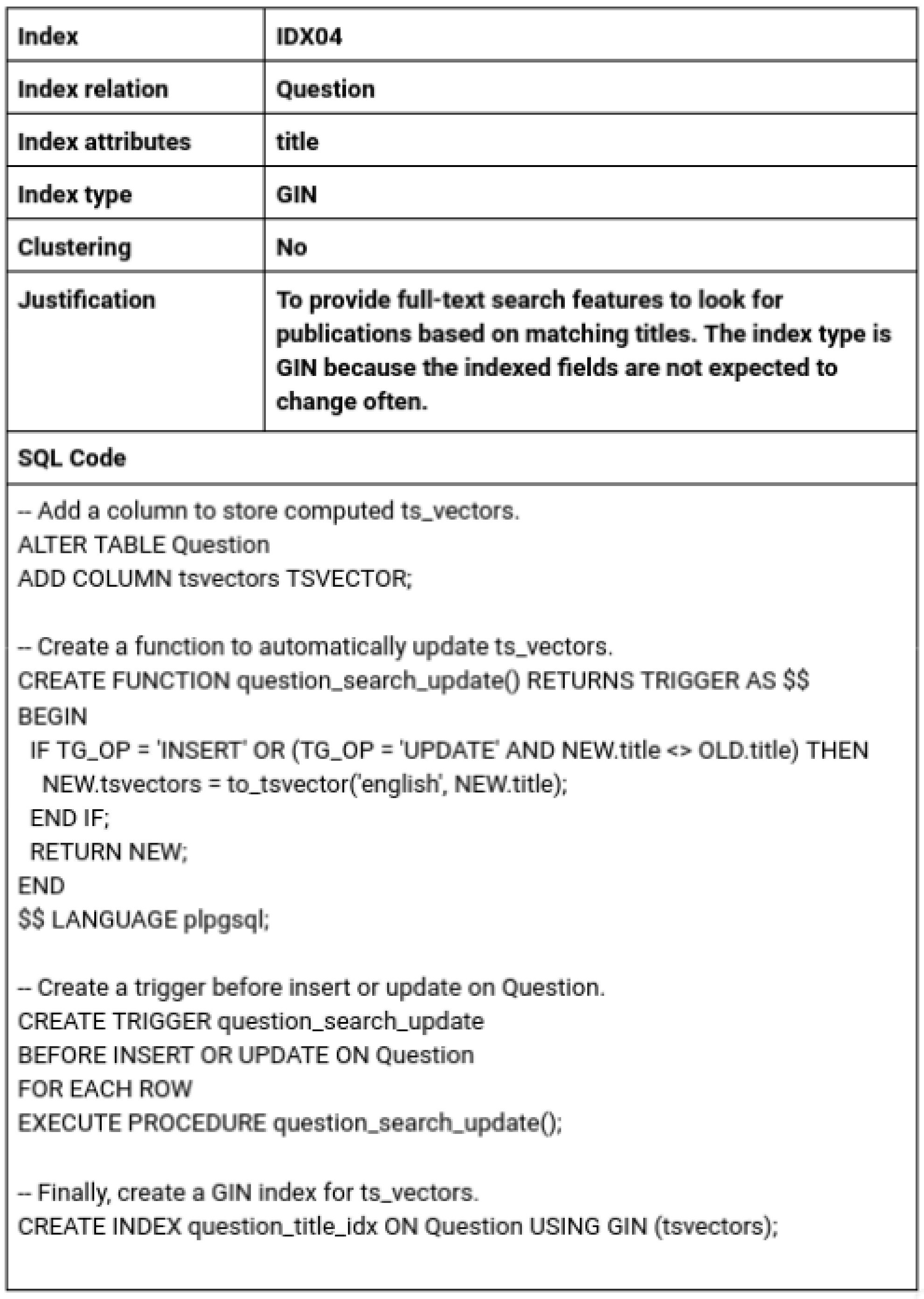
Index	IDX02
Index relation	QuestionOrAnswer
Index attribute	score
Index type	B-tree
Cardinality	High
Clustering	No
Justification	We do this index because it can significantly improve the performance of queries that involve sorting or filtering questions and answers by their scores. A b-tree index allows for faster question or answer range queries based on the score.
SQL Code	
CREATE INDEX score_index ON QuestionOrAnswer USING btree (score);	



Index	IDX03
Index relation	Publication
Index attribute	date
Index type	B-tree
Cardinality	High
Clustering	No
Justification	This index can optimize queries that filter or sort publications by date. By indexing the "date" column, we can significantly improve query performance for such operations.
SQL Code	

```
CREATE INDEX date_index ON Publication USING btree (date);
```

2.2. Full-text Search Indices



Index	**IDX04**
Index relation	**Question**
Index attributes	**title**
Index type	**GIN**
Clustering	**No**
Justification	**To provide full-text search features to look for publications based on matching titles. The index type is GIN because the indexed fields are not expected to change often.**
SQL Code	

```
-- Add a column to store computed ts_vectors.
ALTER TABLE Question
ADD COLUMN tsvectors TSVECTOR;

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

-- Create a trigger before insert or update on Question.
CREATE TRIGGER question_search_update
BEFORE INSERT OR UPDATE ON Question
FOR EACH ROW
EXECUTE PROCEDURE question_search_update();

-- Finally, create a GIN index for ts_vectors.
CREATE INDEX question_title_idx ON Question USING GIN (tsvectors);
```

3. Triggers

Trigger	TRIGGER01
Description	Update the score based on review's "positive" attribute
SQL code	

```
-- Create a trigger to update the score of a question or answer after a review
CREATE OR REPLACE FUNCTION update_score_after_review() RETURNS TRIGGER
AS $$
BEGIN
 IF NEW.positive = 1 THEN
  -- Increase the score by 1 if the review is positive
  UPDATE QuestionOrAnswer
  SET score = score + 1
  WHERE questionAnswer_id = NEW.questionOrAnswer_id;
 ELSIF NEW.positive = 0 THEN
  -- Decrease the score by 1 if the review is not positive
  UPDATE QuestionOrAnswer
  SET score = score - 1
  WHERE questionAnswer_id = NEW.questionOrAnswer_id;
 END IF;
 RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Create a trigger to execute the update_score_after_review function
CREATE TRIGGER update_score_trigger
AFTER INSERT ON Reviews
FOR EACH ROW
EXECUTE FUNCTION update_score_after_review();
```

Trigger	TRIGGER02
Description	Notifications on Tag, Question and Answer
SQL code	

```
-- Create a trigger to insert a notification after a new publication
CREATE OR REPLACE FUNCTION trigger_notifications_function() RETURNS
TRIGGER AS $$
BEGIN
  IF NEW.user_id IS NOT NULL THEN
    -- Insert a notification of type 'QuestionNotif'
    INSERT INTO Notification (user_id, description)
    VALUES (NEW.user_id, 'New answer or comment on your question.');
  END IF;

  IF NEW.questionAnswer_id IS NOT NULL THEN
    -- Insert a notification of type 'AnswerNotif'
    INSERT INTO Notification (user_id, description)
    VALUES (NEW.user_id, 'New comment on your answer');
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_notifications
AFTER INSERT ON QuestionOrAnswer
FOR EACH ROW
EXECUTE FUNCTION trigger_notifications_function();
```

4. Transactions

> Transactions needed to assure the integrity of the data.

Transaction	TRAN01
Description	User Registration
Justification	We use the “READ” isolation level, which ensures that the data read by this transaction is committed.
Isolation level	READ

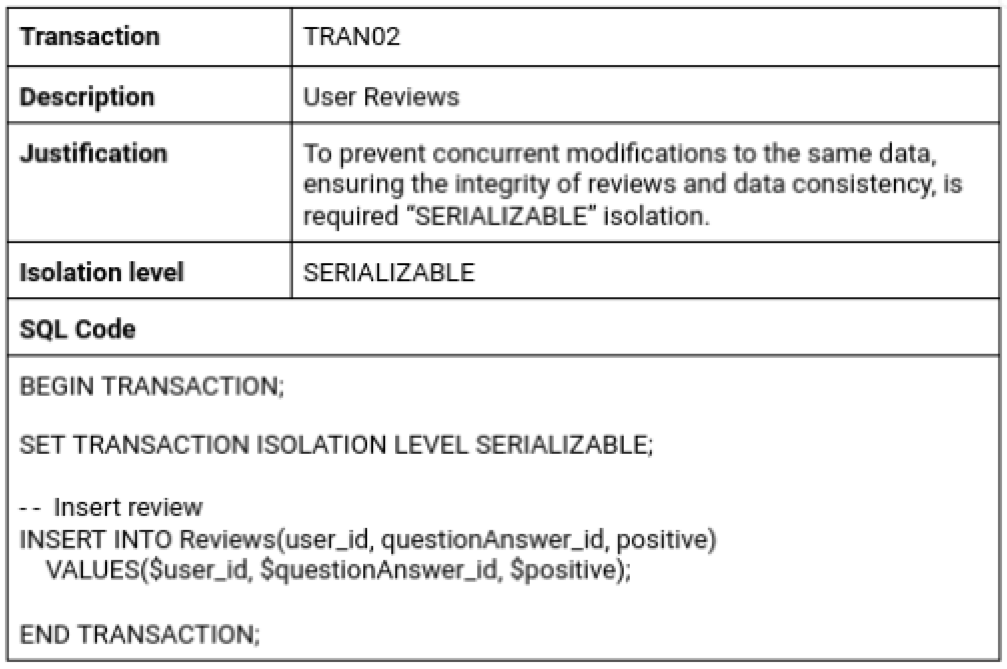
SQL Code

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL READ;

- -  Insert new user
INSERT INTO Users(username, email, password, description)
   VALUES($username, $email, $password, $description);

END TRANSACTION;
```



Transaction	TRAN02
Description	User Reviews
Justification	To prevent concurrent modifications to the same data, ensuring the integrity of reviews and data consistency, is required “SERIALIZABLE” isolation.
Isolation level	SERIALIZABLE

SQL Code

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

- -  Insert review
INSERT INTO Reviews(user_id, questionAnswer_id, positive)
   VALUES($user_id, $questionAnswer_id, $positive);

END TRANSACTION;
```

Transaction	TRAN03
Description	Publication of content
Justification	We need to prevent other transactions from modifying the data read by this transaction to ensure data consistency and to avoid issues related to concurrent updates.
Isolation level	REPEATABLE READ

SQL Code

```
BEGIN TRANSACTION;

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

- -  Insert publication
INSERT INTO Publication(owner_id, tag_id, content, date)
   VALUES($owner_id,$, tag_id, $content, $date);

- -  Insert question
INSERT INTO Question(question_id, title, status)
   VALUES($question_id, $title, $status);

- -  Insert answer
INSERT INTO Answer(answer_id, question_id)
   VALUES($answer_id, $question_id)

END TRANSACTION;
```

A.1. Database schema

```
DROP SCHEMA lbaw23136 CASCADE;
CREATE SCHEMA lbaw23136;
SET search_path TO lbaw23136;
DROP TABLE IF EXISTS TagNotif CASCADE;
```

```sql
DROP TABLE IF EXISTS Tag CASCADE;
DROP TABLE IF EXISTS AnswerNotif CASCADE;
DROP TABLE IF EXISTS QuestionNotif CASCADE;
DROP TABLE IF EXISTS Notification CASCADE;
DROP TABLE IF EXISTS Comment CASCADE;
DROP TABLE IF EXISTS Answer CASCADE;
DROP TABLE IF EXISTS Question CASCADE;
DROP TABLE IF EXISTS QuestionOrAnswer CASCADE;
DROP TABLE IF EXISTS Publication CASCADE;
DROP TABLE IF EXISTS Admin CASCADE;
DROP TABLE IF EXISTS Moderator CASCADE;
DROP TABLE IF EXISTS Owner CASCADE;
DROP TABLE IF EXISTS Users CASCADE;
DROP TABLE IF EXISTS Subscription CASCADE;
DROP TABLE IF EXISTS Bannings CASCADE;
DROP TABLE IF EXISTS Reviews CASCADE;

-----------------------------------------
-- Domains
-----------------------------------------
DROP DOMAIN IF EXISTS TODAY;
DROP DOMAIN IF EXISTS STATUS;

CREATE DOMAIN TODAY AS DATE NOT NULL DEFAULT CURRENT_DATE;
CREATE DOMAIN STATUS AS VARCHAR(255) NOT NULL CHECK (VALUE IN ('open', 'closed')) DEFAULT 'open';

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

-- We modified the table "User" to "Users", otherwise PostgreSQL gives an error
CREATE TABLE Users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(255) NOT NULL UNIQUE,
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    description TEXT,
    profile_picture BYTEA
);

CREATE TABLE Owner (
    owner_id INTEGER PRIMARY KEY,
    FOREIGN KEY (owner_id) REFERENCES Users(id) ON DELETE CASCADE
);

CREATE TABLE Moderator (
    moderator_id INTEGER PRIMARY KEY,
    FOREIGN KEY (moderator_id) REFERENCES Users(id) ON DELETE CASCADE
);

CREATE TABLE Admin (
    admin_id INTEGER PRIMARY KEY,
    FOREIGN KEY (admin_id) REFERENCES Users(id) ON DELETE CASCADE
);

CREATE TABLE Tag (
    id SERIAL PRIMARY KEY,
    tagName VARCHAR NOT NULL UNIQUE
);

CREATE TABLE Publication (
    id SERIAL PRIMARY KEY,
    owner_id INTEGER REFERENCES Owner(owner_id) ON DELETE CASCADE,
    tag_id INTEGER REFERENCES Tag(id) ON DELETE CASCADE,
    content TEXT NOT NULL,
    date TODAY
);
```

```
CREATE TABLE QuestionOrAnswer(
    questionAnswer_id INTEGER PRIMARY KEY,
    FOREIGN KEY (questionAnswer_id) REFERENCES Publication(id) ON DELETE CASCADE,
    score INTEGER NOT NULL DEFAULT 0
);

CREATE TABLE Question(
    question_id INTEGER PRIMARY KEY,
    FOREIGN KEY (question_id) REFERENCES QuestionOrAnswer(questionAnswer_id) ON DELETE CASCADE,
    title VARCHAR(255) NOT NULL,
    status STATUS
);

CREATE TABLE Answer(
    answer_id INTEGER PRIMARY KEY,
    FOREIGN KEY (answer_id) REFERENCES QuestionOrAnswer(questionAnswer_id) ON DELETE CASCADE,
    question_id INTEGER NOT NULL REFERENCES Question(question_id) ON DELETE CASCADE
);

CREATE TABLE Comment(
    comment_id INTEGER PRIMARY KEY,
    FOREIGN KEY (comment_id) REFERENCES Publication(id) ON DELETE CASCADE,
    questionAnswer_id INTEGER NOT NULL REFERENCES QuestionOrAnswer(questionAnswer_id) ON DELETE CASCADE
);

CREATE TABLE Notification(
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES Users(id) ON DELETE CASCADE,
    description TEXT NOT NULL
);

CREATE TABLE QuestionNotif(
    notification_id INTEGER PRIMARY KEY,
    FOREIGN KEY (notification_id) REFERENCES Notification(id) ON DELETE CASCADE,
    question_id INTEGER NOT NULL REFERENCES Question(question_id) ON DELETE CASCADE
);

CREATE TABLE AnswerNotif(
    notification_id INTEGER PRIMARY KEY,
    FOREIGN KEY (notification_id) REFERENCES Notification(id) ON DELETE CASCADE,
    answer_id INTEGER NOT NULL REFERENCES Answer(answer_id) ON DELETE CASCADE
);

CREATE TABLE TagNotif(
    notification_id INTEGER PRIMARY KEY,
    FOREIGN KEY (notification_id) REFERENCES Notification(id) ON DELETE CASCADE,
    tag_id INTEGER NOT NULL REFERENCES Tag(id) ON DELETE CASCADE
);

CREATE TABLE Subscription(
    user_id INTEGER NOT NULL REFERENCES Users(id) ON DELETE CASCADE,
    tag_id INTEGER NOT NULL REFERENCES Tag(id) ON DELETE CASCADE,
    PRIMARY KEY (user_id, tag_id),
    date TODAY
);

CREATE TABLE Bannings(
    user_id INTEGER NOT NULL REFERENCES Users(id) ON DELETE CASCADE,
    admin_id INTEGER NOT NULL REFERENCES Admin(admin_id) ON DELETE CASCADE,
    PRIMARY KEY (user_id, admin_id),
    date TODAY
);

CREATE TABLE Reviews(
    user_id INTEGER NOT NULL REFERENCES Users(id) ON DELETE CASCADE,
    questionAnswer_id INTEGER NOT NULL REFERENCES QuestionOrAnswer(questionAnswer_id) ON DELETE CASCADE,
    PRIMARY KEY (user_id, questionAnswer_id),
    positive BOOLEAN,
    date TODAY
);




-----------------------------------------
-- INDEXES
-----------------------------------------

CREATE INDEX user_notification ON Notification USING btree (user_id);

CREATE INDEX score_index ON QuestionOrAnswer USING btree (score);

CREATE INDEX date_index ON Publication USING btree (date);

-- FTS INDEXES

-- Add a column to store computed ts_vectors.
ALTER TABLE Question
ADD COLUMN tsvectors TSVECTOR;

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

-- Create a trigger before insert or update on Question.
CREATE TRIGGER question_search_update
BEFORE INSERT OR UPDATE ON Question
FOR EACH ROW
EXECUTE PROCEDURE question_search_update();

-- Finally, create a GIN index for ts_vectors.
CREATE INDEX question_title_idx ON Question USING GIN (tsvectors);




-----------------------------------------
-- TRIGGERS
-----------------------------------------
-- TRIGGER01
-- Create a trigger to update the score of a question or answer after a review
CREATE OR REPLACE FUNCTION update_score_after_review() RETURNS TRIGGER AS $$
BEGIN
  IF NEW.positive = 1 THEN
    -- Increase the score by 1 if the review is positive
    UPDATE QuestionOrAnswer
    SET score = score + 1
    WHERE questionAnswer_id = NEW.questionOrAnswer_id;
  ELSIF NEW.positive = 0 THEN
    -- Decrease the score by 1 if the review is not positive
    UPDATE QuestionOrAnswer
    SET score = score - 1
    WHERE questionAnswer_id = NEW.questionOrAnswer_id;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Create a trigger to execute the update_score_after_review function
```

```
CREATE TRIGGER update_score_trigger
AFTER INSERT ON Reviews
FOR EACH ROW
EXECUTE FUNCTION update_score_after_review();


-- TRIGGER02
-- Create a trigger to insert a notification after a new publication
CREATE OR REPLACE FUNCTION trigger_notifications_function() RETURNS TRIGGER AS $$
BEGIN
    IF NEW.user_id IS NOT NULL THEN
        -- Insert a notification of type 'QuestionNotif'
        INSERT INTO Notification (user_id, description)
        VALUES (NEW.user_id, 'New answer or comment on your question.');
    END IF;

    IF NEW.questionAnswer_id IS NOT NULL THEN
        -- Insert a notification of type 'AnswerNotif'
        INSERT INTO Notification (user_id, description)
        VALUES (NEW.user_id, 'New comment on your answer');
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trigger_notifications
AFTER INSERT ON QuestionOrAnswer
FOR EACH ROW
EXECUTE FUNCTION trigger_notifications_function();
```

database.sql

A.2. Database population

```
-- Insert sample data into the Users table (with email as username + @example.com)
INSERT INTO Users (id, username, email, password, description, profile_picture)
VALUES
    (1, 'JohnAppeased', 'johnappeased@example.com', 'password123', 'John is a software developer with a passion for cod
    (2, 'AliceSmith', 'alicesmith@example.com', 'password456', 'Alice is an artist who loves painting and sculpting.', 
    (3, 'RobertJohnson', 'robertjohnson@example.com', 'password789', 'Robert is a scientist exploring the mysteries of 
    (4, 'EmmaBrown', 'emmabrown@example.com', 'password987', 'Emma is a travel enthusiast and a food blogger.', NULL),
    (5, 'WilliamClark', 'williamclark@example.com', 'password654', 'William is a fitness coach and nutrition expert.', 
    (6, 'OliviaAnderson', 'oliviaanderson@example.com', 'password321', 'Olivia is a musician with a passion for playing
    (7, 'SophiaWhite', 'sophiawhite@example.com', 'password654', 'Sophia is a veterinarian who adores animals.', NULL),
    (8, 'LiamHarris', 'liamharris@example.com', 'password123', 'Liam is an avid hiker and nature enthusiast.', NULL),
    (9, 'CharlotteDavis', 'charlottedavis@example.com', 'password987', 'Charlotte is a bookworm and literature lover.',
    (10, 'JamesMiller', 'jamesmiller@example.com', 'password456', 'James is a chef specializing in international cuisin
```

populate.sql

Revision history

Changes made to the first submission:

1. Item 1
2. ..

GROUP23136, 24/10/2023

- Inês Martin Soares, up202108852@fe.up.pt(Editor)
- João Padrão, up202108766@fe.up.pt
- Samuel Maciel, up202108697@fe.up.pt
- João Ventura, up202302063@fe.up.pt(Editor)