EBD: Database Specification Component

Gameorama is a web application that provides a place for gamers to post, discuss and rate the latest news on their favorite videogames and topics.

A4: Conceptual Data Model

1. Class diagram

The Conceptual Domain Model contains the identification and description of the entities of the domain and the relationships between them in a UML class diagram.

The diagram of **Figure 1** represents the main organizational entities, the relationships between them, attributes and their domains, and the multiplicity of relationships for the **Gameorama** website.

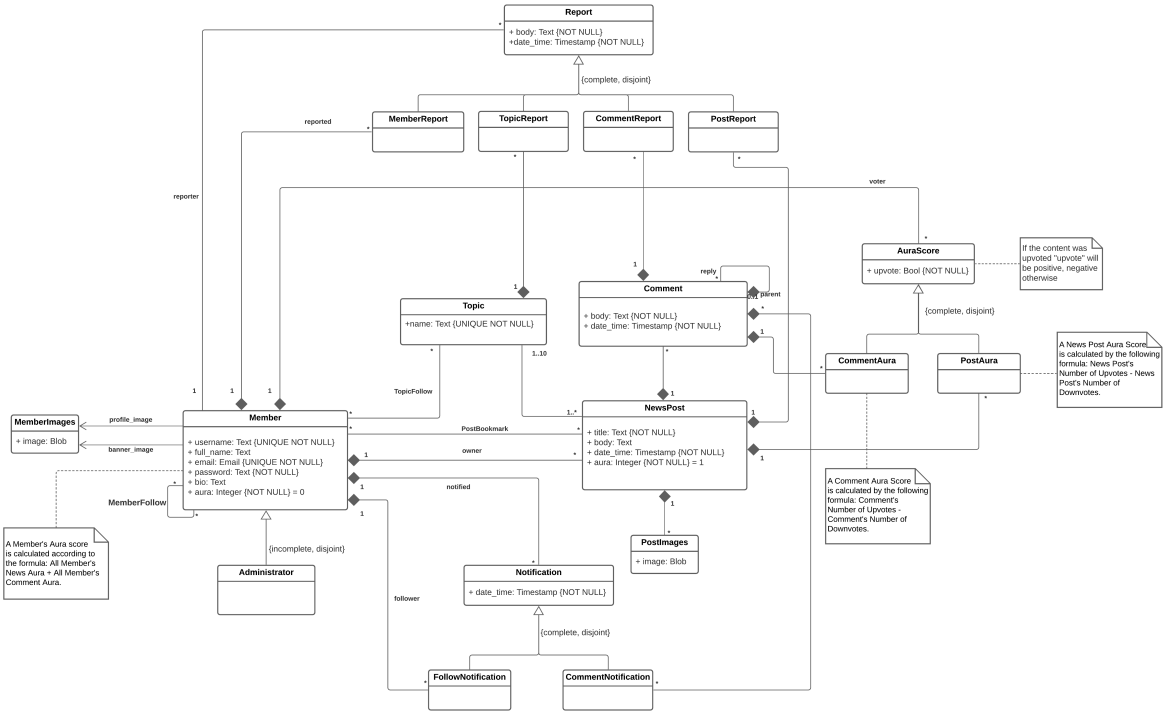

Figure 1: Class diagram

2. Additional Business Rules

- Members cannot follow themselves;
- Members cannot report themselves;
- When a news post is deleted, all comments and replies associated with it are deleted;
- When a comment is deleted, all replies associated with it are deleted;
- A newly created member has a default profile picture and banner image;

A5: Relational Schema, validation and schema refinement

This artifact contains the Relational Schema obtained by mapping from the Conceptual Data Model. The Relational Schema includes the relation schema, attributes, domains, primary keys, foreign keys and other integrity rules: UNIQUE, DEFAULT, NOT NULL, CHECK.

1. Relational Schema

Relation schemas are specified in the compact notation:

Relation reference	Relation Compact Notation
R01	member_image(id, file **NN**)
R02	member(id, username **UK NN**, full_name **NN**, email **UK NN**, password **NN**, bio, id_profile_image → member_image **NN**, id_banner_image → member_image **NN**, aura **NN DF** 0)
R03	administrator(id → member)
R04	member_follow(id_followed → member, id_follower → member **CK** id_follower <> id_followed)
R05	news_post(id, title **NN**, body, date_time **NN**, aura **NN DF** 0, id_owner → member **NN**)
R06	topic(id, name **UK NN**)
R07	topic_follow(id_topic → topic, id_member → member)
R08	post_topic(id_post → news_post, id_topic → topic)
R09	comment(id, body **NN**, date_time **NN**, aura **NN DF** 0, id_owner → member **NN**, id_post → news_post **NN**)
R10	reply(id_comment → comment, id_parent → comment **NN CK** id_parent <> id_comment)
R11	post_image(id, id_post → news_post **NN**, file **NN**)
R12	post_aura(id_post → news_post, id_voter → member , upvote **NN**)
R13	comment_aura(id_comment → Comment, id_voter → member, upvote)
R14	follow_notification(id_notified → member, id_follower → member **CK** id_notified <> id_follower, date_time **NN**)
R15	comment_notification(id_notified → member, id_comment → comment, date_time **NN**)
R16	reply_notification(id_notified → member, id_reply → reply, date_time **NN**)
R17	post_report(id_reporter → member, id_post → news_post, body **NN**, date_time **NN**)
R18	comment_report(id_reporter → member, id_comment → comment, body **NN**, date_time **NN**)
R19	topic_report(id_reporter → member, id_topic → topic, body **NN**, date_time **NN**)
R20	member_report(id_reporter → member, id_reported → member **CK** id_reported <> id_reporter, body **NN**, date_time **NN**)
R21	post_bookmark(id_post → news_post, id_bookmarker → member

Note: UK means UNIQUE KEY, NN means NOT NULL, DF means DEFAULT and CK means CHECK.

2. Domains

For this project, there was no need to specify any additional domains.

3. Functional Dependencies and schema validation

To validate the Relational Schema obtained from the Conceptual Model, all functional dependencies are identified and the normalization of all relation schemas is accomplished.

Table R01 (member_image)	
Keys	{id}
Functional Dependencies	
FD0101	{id}:- {file}
Normal form	BCNF

Table R02 (member)	
Keys	{id}, {username}, {email}
Functional Dependencies	
FD0201	{id}:- {username, full_name ,email, password, bio, id_profile_image, id_banner_image, aura}
FD0202	{username}:- {id, full_name ,email, password, bio, id_profile_image, id_banner_image, aura}
FD0203	{email}:- {id, username, full_name, password, bio, id_profile_image, id_banner_image, aura}
Normal form	BCNF

Table R03 (administrator)	
Keys	{id_member}
Functional Dependencies	
Normal form	BCNF

Table R04 (member_follow)	
Keys	{id_followed, id_follower}
Functional Dependencies	
Normal form	BCNF

Table R05 (news_post)	
Keys	{id}
Functional Dependencies	
FD0501	{id}:- {title, body, date, aura, id_owner}
Normal form	BCNF

Table R06 (topic)	
Keys	{id}
Functional Dependencies	
FD0601	{id}:- {name}
Normal form	BCNF

Table R07 (topic_follow)	
Keys	{id_topic, id_member}
Functional Dependencies	
Normal form	BCNF

Table R08 (post_topic)	
Keys	{id_post, id_topic}
Functional Dependencies	
Normal form	BCNF

Table R09 (comment)	
Keys	{id}
Functional Dependencies	
FD0901	{id}:- {body, date_time, aura, id_owner, id_post}
Normal form	BCNF

Table R10 (reply)	
Keys	{id_comment}
Functional Dependencies	
FD1001	{id_comment}:- {id_parent}
Normal form	BCNF

Table R11 (post_image)	
Keys	{id}
Functional Dependencies	
FD1101	{id}:- {id_post, file}
Normal form	BCNF

Table R12 (post_aura)	
Keys	{id_post, id_voter}
Functional Dependencies	
FD1201	{id_post, id_voter}:- {upvote}
Normal form	BCNF

Table R13 (comment_aura)	
Keys	{id_comment, id_voter}
Functional Dependencies	
FD1301	{id_comment, id_voter}:- {upvote}
Normal form	BCNF

Table R14 (follow_notification)	
Keys	{id_notified, id_follower}
Functional Dependencies	
FD1401	{id_follower, id_followed}:- {date_time}
Normal form	BCNF

Table R15 (comment_notification)	
Keys	{id_notified, id_comment}
Functional Dependencies	
FD1501	{id_notified, id_comment}:- {date_time}
Normal form	BCNF

Table R16 (reply_notification)	
Keys	{id_notified, id_reply}
Functional Dependencies	
FD1601	{id_notified, id_reply}:- {date_time}
Normal form	BCNF

Table R17 (post_report)	
Keys	{id_reporter, id_post}
Functional Dependencies	
FD1701	{id_reporter, id_post}:- {body, date_time}
Normal form	BCNF

Table R18 (comment_report)	
Keys	{id_reporter, id_comment}
Functional Dependencies	
FD1801	{id_reporter, id_comment}:- {body, date_time}
Normal form	BCNF

Table R19 (topic_report)	
Keys	{id_reporter, id_topic}
Functional Dependencies	
FD1901	{id_reporter, id_topic}:- {body, date_time}
Normal form	BCNF

Table R20 (member_report)	
Keys	{id_reporter, id_reported}
Functional Dependencies	
FD2001	{id_reporter, id_reported}:- {body, date_time}
Normal form	BCNF

Table R21 (post_bookmark)	
Keys	{id_post, id_bookmarker}
Functional Dependencies	
Normal form	BCNF

As all relations schemas are in the Boyce–Codd Normal Form (BCNF), the relational schema is also in the BCNF and therefore there is no need to be refined using normalization.

A6: Indexes, triggers, user functions, transactions and population

This artefact contains the physical schema of the database, the identification and characterization of the indexes, the support of data integrity rules with triggers and the definition of the database user-defined functions. This artefact also shows the database transactions needed to assure the integrity of the data in the presence of concurrent accesses. For each transaction, the isolation level is explicitly stated and justified and read-only transactions to improve global performance are identified and justified. Finally, it also contains the database's workload as well as the complete database creation script, including all SQL necessary to define all integrity constraints, indexes and triggers.

1. Database Workload

Understanding the nature of the workload for the application and the performance goals is essential to developing a good database design. The workload includes:

- the most important queries (SELECT) and how often they arise
- the most important modifications (UPDATE, DELETE) and how often they arise
- the desired performance for these queries and updates
- an estimate of the number of tuples for each relation

1.1. Tuple Estimation

The following table includes an estimation of the number of tuples and growth for each relation

Relation reference	Relation Name	Order of magnitude	Estimated growth
R01	member_image	tens of thousands	tens per day
R02	member	tens of thousands	tens per day
R03	member_follow	millions	thousands per day
R04	administrator	tens	units per year
R05	news_post	hundreds of thousands	hundreds per day
R06	topic	thousands	tens per day
R07	topic_follow	hundreds of thousands	hundreds per day
R08	post_topics	hundreds of thousands	hundreds per day
R09	comment	millions	thousands per day
R10	reply	millions	thousands per day
R11	post_images	tens of thousands	hundreds per day
R12	post_aura	millions	tens of thousands per day
R13	comment_aura	millions	tens of thousands per day
R14	follow_notification	tens of thousands	thousands per day
R15	comment_notification	tens of thousands	thousands per day
R16	reply_notification	tens of thousands	thousands per day
R17	post_report	thousands	units per day
R18	comment_report	tens of thousands	tens per day
R19	topic_report	hundreds	units per day
R20	member_report	hundreds	units per day
R21	post_bookmark	tens of thousands	thousands per day

1.2. Frequent Queries

The following tables contain the most important queries (SELECT) and their frequency.

Query	SELECT01
Description	Selects a member's information
Frequency	tens of thousands per day

SQL Code:

```
SELECT id, username, full_name, bio, aura, profile_image, banner_image
FROM member
NATURAL JOIN
(SELECT id AS id_profile_image, file AS profile_image FROM member_image) AS
member_profile_image
NATURAL JOIN
(SELECT id AS id_banner_image, file AS banner_image FROM member_image) AS
member_banner_image
WHERE member.id = $id_member;
```

Query	SELECT02
Description	Selects a member's password hash
Frequency	tens of thousands per day

SQL Code:

```
SELECT password
FROM member
WHERE username = $username;
```

Query	SELECT03
Description	Select all news posts and respective aura score and number of comments, sorted by recent
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT news_post.id, title, body, date_time, news_post.aura, id_owner, username
FROM news_post
INNER JOIN member ON id_owner = member.id
ORDER BY news_post.date_time DESC;
```

Query	SELECT04
Description	Select main page trending posts - posts from last 2 weeks with the most amount of aura score
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT news_post.id, title, body, date_time, news_post.aura, id_owner, username
FROM news_post
INNER JOIN member ON id_owner = member.id
WHERE date_time BETWEEN NOW()::DATE-EXTRACT(DOW FROM NOW())::INTEGER - 14
AND NOW()::DATE-EXTRACT(DOW from NOW())::INTEGER + 1
ORDER BY aura DESC;
```

Query	SELECT05
Description	Select all news_posts for a member feed
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT news_post.id, title, body, date_time, news_post.aura, id_owner, username
FROM news_post
INNER JOIN member ON id_owner = member.id
WHERE news_post.id IN
(
    SELECT DISTINCT news_post.id FROM news_post
    INNER JOIN post_topic ON news_post.id = post_topic.id_post
    INNER JOIN topic ON post_topic.id_topic = topic.id
    INNER JOIN member_follow ON member_follow.id_follower = $id_member
    WHERE topic.name IN
    (
        SELECT name FROM topic
        INNER JOIN topic_follow ON topic.id = topic_follow.id_topic
        WHERE topic_follow.id_member = $id_member
    )
    OR
    member_follow.id_followed = id_owner
) ORDER BY date_time DESC;
```

Query	SELECT06
Description	Check if a post is bookmarked for a member
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT COUNT(*)
FROM post_bookmark
WHERE id_post = $id_post AND id_bookmarker = $id_member
```

Query	SELECT07
Description	Select all member's bookmarked posts
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT news_post.id, title, body, date_time, news_post.aura, id_owner, username
FROM news_post
INNER JOIN member ON id_owner = member.id
INNER JOIN post_bookmark ON news_post.id = post_bookmark.id_post
WHERE id_bookmarker = $id_member
ORDER BY news_post.date_time DESC;
```

Query	SELECT08
Description	Selects all topics from a post
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT id, name
FROM post_topic
INNER JOIN topic ON topic.id = post_topic.id_topic
WHERE id_post = $id_post;
```

Query	SELECT09
Description	Selects all images from a post
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT id, file
FROM post_image
WHERE id_post = $id_post;
```

Query	SELECT10
Description	Select number of comments of a post
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT COUNT(*)
FROM comment
WHERE id_post = $id_post;
```

Query	SELECT11
Description	Select all parent comments from a post (not replies)
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT id, body, date_time, aura, id_owner, id_post
FROM comment
WHERE id_post = $id_post
AND id NOT IN (SELECT id_comment as id FROM reply WHERE id_post = $id_post);
```

Query	SELECT12
Description	Select all replies to a comment
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT id, body, date_time, aura, id_owner, id_post
FROM reply
INNER JOIN comment ON id_comment = id
WHERE id_parent = $id_comment;
```

Query	SELECT13
Description	Select the 5 members with the most aura score (for the Hall of Fame)
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT id, username, aura
FROM member
ORDER BY aura DESC LIMIT 5;
```

Query	SELECT14
Description	Select the 5 topics with the most followers (for the Most Popular Topics)
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT id, name, num_followers
FROM topic
NATURAL JOIN
(SELECT id_topic AS id, COUNT(*) AS num_followers FROM topic_follow GROUP BY
id_topic) AS num_followers_topics
ORDER BY num_followers DESC
LIMIT 5;
```

Query	SELECT15
Description	Select all posts from a specific topic
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT * FROM news_post
INNER JOIN post_topic ON news_post.id = post_topic.id_post
INNER JOIN topic ON post_topic.id_topic = topic.id
WHERE name = $name_topic;
```

Query	SELECT16
Description	Select all posts from a member
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT * FROM news_post
WHERE id_owner = $id_member;
```

Query	SELECT17
Description	Select all comments from a member
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT * FROM comment
WHERE id_owner = $id_member;
```

Query	SELECT18
Description	Select topics followed by a member
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT id, name
FROM topic
INNER JOIN topic_follow
ON id_topic = id
WHERE id_member = $id_member;
```

Query	SELECT19
Description	Select members followed by a member
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT *
FROM member_follow
INNER JOIN member
ON id_followed = id
NATURAL JOIN
(
    SELECT id_followed AS id, COUNT(*) AS num_followers FROM member_follow GROUP
BY id_followed
) AS num_followers
WHERE id_follower = $id_member;
```

Query	SELECT20
Description	Select members that follow a member
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT *
FROM member_follow
INNER JOIN member
ON id_follower = id
NATURAL JOIN
(
    SELECT id_follower AS id, COUNT(*) AS num_followers FROM member_follow GROUP
BY id_follower
) AS num_followers
WHERE id_followed = $id_member;
```

Query	SELECT21
Description	Select posts bookmarked by a member
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT * FROM post_bookmark
INNER JOIN news_post ON news_post.id = post_bookmark.id_post
INNER JOIN member M ON news_post.owner = M.id
WHERE post_bookmark.id_bookmarker = $id_member;
```

Query	SELECT22
Description	Select the most reported members
Frequency	hundreds per day

SQL Code:

```
SELECT * FROM member
NATURAL JOIN (SELECT id_reported AS id, COUNT(*) AS num_reports FROM
member_report GROUP BY id_reported) AS reported_members
ORDER BY num_reports DESC;
```

Query	SELECT23
Description	Select the most reported posts
Frequency	hundreds per day

SQL Code:

```
SELECT * FROM news_post
NATURAL JOIN (SELECT id_post AS id, COUNT(*) AS num_reports FROM post_report
GROUP BY id_post) AS reported_posts
ORDER BY num_reports DESC;
```

Query	SELECT24
Description	Select the most reported topics
Frequency	hundreds per day

SQL Code:

```
SELECT * FROM topic
NATURAL JOIN (SELECT id_topic AS id, COUNT(*) AS num_reports FROM topic_report
GROUP BY id_topic) AS reported_topics
ORDER BY num_reports DESC;
```

Query	SELECT25
Description	Select the most reported comments
Frequency	hundreds per day

SQL Code:

```
SELECT * FROM comment
NATURAL JOIN (SELECT id_comment AS id, COUNT(*) AS num_reports FROM
comment_report GROUP BY id_comment) AS reported_comments
ORDER BY num_reports DESC;
```

Query	SELECT26
Description	Select news posts with a title or content similar to the input searched
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT *
FROM news_post
WHERE title @@ plainto_tsquery('english', $search);
```

Query	SELECT27
Description	Select members with a username similar to the input searched
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT *
FROM member
WHERE username @@ plainto_tsquery('english', $search);
```

Query	SELECT29
Description	Select topics with a name similar to the input searched
Frequency	hundreds of thousands per day

SQL Code:

```
SELECT id, name
FROM topic
WHERE name @@ plainto_tsquery('english', $search);
```

1.3. Frequent Updates

The following tables contain the most important updates (INSERT, UPDATE, DELETE) and their frequency.

Query	UPDATE01
Description	Update member information (Edit profile page)
Frequency	thousands per day

SQL Code:

```
UPDATE member SET full_name = $full_name, bio = $bio
WHERE id = $id_member
```

Query	UPDATE02
Description	Update account settings (Account settings page)
Frequency	thousands per day

SQL Code:

```
UPDATE member SET password = $password_hash, email = $email
WHERE id = $id_member;
```

Query	UPDATE03
Description	Update news post's information (Edit a post)
Frequency	thousands per day

SQL Code:

```
UPDATE news_post SET title = $title, body = $body
WHERE id = $id_post;
```

Query	UPDATE04
Description	Update a vote from a post
Frequency	thousands per day

SQL Code:

```
UPDATE post_aura SET upvote = $upvote
WHERE id_post = $id_post
AND id_voter = $id_voter
```

Query	UPDATE05
Description	Update a vote from a comment
Frequency	thousands per day

SQL Code:

```
UPDATE comment_aura SET upvote = $upvote
WHERE id_comment = $id_comment
AND id_voter = $id_voter
```

Query	INSERT01
Description	Create a member image
Frequency	tens per day

SQL Code:

```
insert into member_image (file) values ($file));
```

Query	INSERT02
Description	Register a new member
Frequency	tens per day

SQL Code:

```
INSERT INTO member (username, full_name, email, password, bio, id_profile_image,
id_banner_image) VALUES ($username, $full_name, $email, $password, $bio,
$id_profile_image, $id_banner_image);
```

Query	INSERT03
Description	Register a new administrator
Frequency	units per year

SQL Code:

```
INSERT INTO administrator (id) VALUES ($id);
```

Query	INSERT04
Description	Member follows another member
Frequency	thousands per day

SQL Code:

```
INSERT INTO member_follow (id_followed, id_follower) VALUES ($id_followed,
$id_follower);
```

Query	INSERT05
Description	Create a news post
Frequency	hundreds per day

SQL Code:

```
INSERT INTO news_post (title, body, date_time, id_owner) VALUES ($title, $body,
$date_time, $id_owner);
```

Query	INSERT06
Description	Create a topic
Frequency	tens per day

SQL Code:

```
INSERT INTO topic (name) VALUES ($name) ON CONFLICT DO NOTHING;
```

Query	INSERT07
Description	Member follows a topic
Frequency	hundreds per day

SQL Code:

```
INSERT INTO topic_follow (id_topic, id_member) VALUES ($id_topic, $id_member);
```

Query	INSERT08
Description	Assign a topic to a post
Frequency	hundreds per day

SQL Code:

```
INSERT INTO post_topic (id_topic, id_post) VALUES ($id_topic, $id_post);
```

Query	INSERT09
Description	Create a comment to a post
Frequency	thousands per day

SQL Code:

```
INSERT INTO comment (body, id_owner, id_post) VALUES ($body, $id_owner,
$id_post);
```

Query	INSERT10
Description	Create a reply to a comment
Frequency	thousands per day

SQL Code:

```
INSERT INTO reply (id_comment, id_parent) VALUES ($id_comment, $id_parent);
```

Query	INSERT11
Description	Create an image for a post
Frequency	hundreds per day

SQL Code:

```
INSERT INTO post_image (id_post, file) VALUES ($id_post, $file);
```

Query	INSERT12
Description	Member bookmarks a post
Frequency	hundreds per day

SQL Code:

```
INSERT INTO post_bookmark(id_post, id_bookmarker) VALUES($id_post,
$id_bookmarker);
```

Query	INSERT13
Description	Member votes on a post
Frequency	tens of thousands per day

SQL Code:

```
INSERT INTO post_aura (id_post, id_voter, upvote) VALUES ($id_post, $id_voter, $upvote);
```

Query	INSERT14
Description	Member votes on a comment
Frequency	tens of thousands per day

SQL Code:

```
INSERT INTO comment_aura (id_comment, id_voter, upvote) VALUES ($id_comment, $id_voter, $upvote);
```

Query	INSERT15
Description	Create a notification for a follow
Frequency	thousands per day

SQL Code:

```
INSERT INTO follow_notification (id_followed, id_follower) VALUES ($id_followed, $id_follower);
```

Query	INSERT16
Description	Create a notification for a comment
Frequency	thousands per day

SQL Code:

```
INSERT INTO comment_notification (id_notified, id_comment) VALUES ($id_notified, $id_comment);
```

Query	INSERT17
Description	Create a notification for a reply
Frequency	thousands per day

SQL Code:

```
INSERT INTO reply_notification (id_notified, id_comment) VALUES ($id_notified,
$id_comment);
```

Query	INSERT18
Description	Create a report for a post
Frequency	units per day

SQL Code:

```
INSERT INTO post_report (id_reporter, id_post, body) VALUES ($id_reporter,
$id_post, $body);
```

Query	INSERT19
Description	Create a report for a comment
Frequency	tens per day

SQL Code:

```
INSERT INTO comment_report (id_reporter, id_comment, body) VALUES ($id_reporter,
$id_comment, $body);
```

Query	INSERT20
Description	Create a report for a topic
Frequency	units per day

SQL Code:

```
INSERT INTO topic_report (id_reporter, id_topic, body) VALUES ($id_reporter,
$id_topic, $body);
```

Query	INSERT21
Description	Create a report for a member
Frequency	units per day

SQL Code:

```
INSERT INTO member_report (id_reporter, id_reported, body) VALUES ($id_reporter, $id_reported, $body);
```

Query	DELETE01
Description	Delete a member
Frequency	units per day

SQL Code:

```
DELETE FROM member WHERE id = $id_member;
```

Query	DELETE02
Description	Delete an image from a member
Frequency	units per day

SQL Code:

```
DELETE FROM member_image WHERE id = $id_image;
```

Query	DELETE03
Description	Member unfollows another member
Frequency	hundreds per day

SQL Code:

```
DELETE FROM member_follow
WHERE id_follower = $id_follower
AND id_followed = $id_followed;
```

Query	DELETE04
Description	Member unfollows a topic
Frequency	tens per day

SQL Code:

```
DELETE FROM topic_follow
WHERE id_member = $id_member
AND id_topic = $id_topic;
```

Query	DELETE05
Description	Delete a news post
Frequency	tens per day

SQL Code:

```
DELETE FROM news_post WHERE id = $id_post;
```

Query	DELETE06
Description	Delete a comment
Frequency	tens per day

SQL Code:

```
DELETE FROM comment WHERE id = $id_comment;
```

Query	DELETE07
Description	Delete a vote from a post
Frequency	thousands per day

SQL Code:

```
DELETE FROM post_aura
WHERE id_post = $id_post
AND id_voter = $id_voter
```

Query	DELETE08
Description	Delete a vote from a comment
Frequency	thousands per day

SQL Code:

```
DELETE FROM comment_aura
WHERE id_comment = $id_comment
AND id_voter = $id_voter
```

Query	DELETE09
Description	Delete a bookmark from a post
Frequency	tens per day

SQL Code:

```
DELETE FROM post_bookmark
WHERE id_post = $id_post
AND id_bookmarker = $id_bookmarker;
```

2. Proposed Indices

2.1. Performance Indexes

The following tables display the indexes proposed to improve the performance of the identified queries.

Index	IDX01
Related queries	SELECT03
Relation	news_post
Attribute	date_time
Type	B-tree
Cardinality	Low
Clustering	No
Justification	This index allows the news posts to be searched by date faster. Allowing the main page information to be loaded quickly.

SQL Code:

```
DROP INDEX IF EXISTS news_post_date;
CREATE INDEX post_date ON news_post USING btree (date_time);
```

Index	IDX02
Related queries	SELECT11
Relation	comment
Attribute	id_post
Type	Hash
Cardinality	Medium
Clustering	No
Justification	Due to the number of comments present in the table, this index reduces the query execution time allowing the comment to be fetched fast for the news post's page.

SQL Code:

```
DROP INDEX IF EXISTS search_comment_post;
CREATE INDEX search_comment_post ON comment USING hash (id_post);
```

Index	IDX03
Related queries	SELECT16
Relation	news_post
Attribute	id_owner
Type	Hash
Cardinality	Medium
Clustering	No
Justification	Due to the number of posts present in the table, this index reduces the query execution time allowing the posts to be fetched fast for the user's profile page.

SQL Code:

```
DROP INDEX IF EXISTS search_post_member;
CREATE INDEX search_post_member ON news_post USING hash (id_owner);
```

Index	IDX04
Related queries	SELECT17
Relation	comment
Attribute	id_owner
Type	Hash
Cardinality	Medium
Clustering	No
Justification	Due to the number of comments present in the table, this index reduces the query execution time allowing the comments to be fetched fast for the user's profile page.

SQL Code:

```
DROP INDEX IF EXISTS search_comment_owner;
CREATE INDEX search_comment_owner ON comment USING hash (id_owner);
```

Index	IDX05
Related queries	SELECT18
Relation	topic_follow
Attribute	id_member
Type	Hash
Cardinality	Medium
Clustering	No
Justification	Due to a large number of members following topics, this index reduces the query execution time allowing the followed topics to be fetched fast for the user's profile page.

SQL Code:

```
DROP INDEX IF EXISTS topic_follow_member;
CREATE INDEX topic_follow_member ON topic_follow USING hash (id_member);
```

Index	IDX06
Related queries	SELECT19
Relation	member_follow
Attribute	id_followed
Type	Hash
Cardinality	Medium
Clustering	No
Justification	Due to a large number of members following each other, this index reduces the query execution time allowing the followed members to be fetched fast for the user's profile page.

SQL Code:

```
DROP INDEX IF EXISTS member_id_followed;
CREATE INDEX member_id_followed ON member_follow USING hash (id_followed);
```

Index	IDX07
Related queries	SELECT20
Relation	member_follow
Attribute	id_follower
Type	Hash
Cardinality	Medium
Clustering	No
Justification	Due to a large number of members following each other, this index reduces the query execution time allowing the followers to be fetched fast for the user's profile page.

SQL Code:

```
DROP INDEX IF EXISTS member_id_follower;
CREATE INDEX member_id_follower ON member_follow USING hash (id_follower);
```

2.2. Full-text Search Indexes

The following tables specify the indexes created in order to achieve full-text search and its features.

Index	**IDX08**
Related queries	SELECT26
Relation	news_post
Attribute	title
Type	GiST
Clustering	No
Justification	To improve the performance of full-text searches while searching for news posts' titles. The type of this index is GiST because it's better for dynamic data.

SQL Code:

```
DROP INDEX IF EXISTS search_posts;
CREATE INDEX search_posts ON news_post USING gist (to_tsvector('english',
title));
```

Index	**IDX09**
Related queries	SELECT27
Relation	member
Attribute	username
Type	GiST
Clustering	No
Justification	To improve the performance of full-text searches while searching for members' usernames. The type of this index is GiST because it's better for dynamic data.

SQL Code:

```
DROP INDEX IF EXISTS member_username;
CREATE INDEX IF NOT EXISTS member_username ON member USING gist
(to_tsvector('english', username));
```

Index	IDX10
Related queries	SELECT28
Relation	topic
Attribute	name
Type	GIN
Clustering	No
Justification	To improve the performance of full-text searches while searching for topics' names. Seeing that this table is infrequently updated the type of index used is GIN, which takes longer to create/update but leads to faster lookups.

SQL Code:

```
DROP INDEX IF EXISTS topic_name;
CREATE INDEX topic_name ON topic USING gin (to_tsvector('english', name));
```

2.3. Constraint-enforcing Indexes

The following table specifies the index used to enforce special unique constraints, such as guaranteeing uniqueness of case insensitive usernames, emails and topic names.

```
CREATE INDEX unique_lowercase_username ON member (lower(username));
CREATE INDEX unique_lowercase_email ON member (lower(email));
CREATE INDEX unique_lowercase_topic ON topic (lower(name));
```

3. Triggers

The following tables define the triggers and user-defined functions developed in order to maintain the system's data integrity.

Trigger	TRIGGER01
Description	When a post already has a vote (upvote or downvote) and a member changes its vote, the old vote is removed and the post's and its owner's aura score is updated (+2 on upvotes or -2 on downvotes)

SQL Code:

```
DROP FUNCTION IF EXISTS update_post_aura CASCADE;
CREATE FUNCTION update_post_aura() RETURNS TRIGGER AS $$
  BEGIN
    IF NEW.upvote AND NOT OLD.upvote THEN
      UPDATE news_post
        SET aura = aura + 2
        WHERE NEW.id_post = news_post.id;
      UPDATE member
        SET aura = aura + 2
```

```
        WHERE (SELECT id_owner FROM news_post WHERE NEW.id_post = news_post.id)
= member.id;

    ELSIF NOT NEW.upvote AND OLD.upvote THEN
      UPDATE news_post
        SET aura = aura - 2
        WHERE NEW.id_post = news_post.id;
      UPDATE member
        SET aura = aura - 2
        WHERE (SELECT id_owner FROM news_post WHERE NEW.id_post = news_post.id)
= member.id;
    END IF;
    RETURN NEW;
  END;
  $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS update_post_aura ON post_aura CASCADE;
CREATE TRIGGER update_post_aura
    BEFORE UPDATE ON post_aura
    FOR EACH ROW EXECUTE PROCEDURE update_post_aura();
```

Trigger	TRIGGER02
Description	When a post doesn't have a vote yet and a member votes, the new vote is added and the post's and its owner's aura score is updated (+1 on upvotes or -1 on downvotes)

SQL Code:

```
DROP FUNCTION IF EXISTS insert_post_aura CASCADE;
CREATE FUNCTION insert_post_aura() RETURNS TRIGGER AS $$
  BEGIN
    IF NEW.upvote THEN
      UPDATE news_post
        SET aura = aura + 1
        WHERE NEW.id_post = news_post.id;
      UPDATE member
        SET aura = aura + 1
        WHERE (SELECT id_owner FROM news_post WHERE NEW.id_post = news_post.id)
= member.id;

    ELSIF NOT NEW.upvote THEN
      UPDATE news_post
        SET aura = aura - 1
        WHERE NEW.id_post = news_post.id;
      UPDATE member
        SET aura = aura - 1
        WHERE (SELECT id_owner FROM news_post WHERE NEW.id_post = news_post.id)
= member.id;
    END IF;
    RETURN NEW;
  END;
  $$ LANGUAGE plpgsql;
```

```
DROP TRIGGER IF EXISTS insert_post_aura ON post_aura CASCADE;
CREATE TRIGGER insert_post_aura
  BEFORE INSERT ON post_aura
  FOR EACH ROW EXECUTE PROCEDURE insert_post_aura();
```

Trigger	TRIGGER03
Description	When a post already has a vote and the member removes it, the vote is removed and the post's and its owner's aura score is updated (-1 on removing an upvote or +1 on removing a downvote)

SQL Code:

```
DROP FUNCTION IF EXISTS delete_post_aura CASCADE;
CREATE FUNCTION delete_post_aura() RETURNS TRIGGER AS $$
  BEGIN
    IF OLD.upvote THEN
      UPDATE news_post
        SET aura = aura - 1
        WHERE OLD.id_post = news_post.id;
      UPDATE member
        SET aura = aura - 1
        WHERE (SELECT id_owner FROM news_post WHERE OLD.id_post = news_post.id)
= member.id;

    ELSIF NOT OLD.upvote THEN
      UPDATE news_post
        SET aura = aura + 1
        WHERE OLD.id_post = news_post.id;
      UPDATE member
        SET aura = aura + 1
        WHERE (SELECT id_owner FROM news_post WHERE OLD.id_post = news_post.id)
= member.id;
    END IF;
    RETURN NEW;
  END;
  $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS delete_post_aura ON post_aura CASCADE;
CREATE TRIGGER delete_post_aura
    BEFORE DELETE ON post_aura
    FOR EACH ROW EXECUTE PROCEDURE delete_post_aura();
```

Trigger	TRIGGER04
Description	When a comment already has a vote (upvote or downvote) and a member changes its vote, the old vote is removed and the comment's and its owner's aura score is updated (+2 on upvotes or -2 on downvotes)

SQL Code:

```
DROP FUNCTION IF EXISTS update_comment_aura CASCADE;
CREATE FUNCTION update_comment_aura() RETURNS TRIGGER AS $$
  BEGIN
    IF NEW.upvote AND NOT OLD.upvote THEN
      UPDATE comment
        SET aura = aura + 2
        WHERE NEW.id_comment = comment.id;
      UPDATE member
        SET aura = aura + 2
        WHERE (SELECT id_owner FROM comment WHERE NEW.id_comment = comment.id) =
member.id;

    ELSIF NOT NEW.upvote AND OLD.upvote THEN
      UPDATE comment
        SET aura = aura - 2
        WHERE NEW.id_comment = comment.id;
      UPDATE member
        SET aura = aura - 2
        WHERE (SELECT id_owner FROM comment WHERE NEW.id_comment = comment.id) =
member.id;
    END IF;
    RETURN NEW;
  END;
  $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS update_comment_aura ON comment_aura CASCADE;
CREATE TRIGGER update_comment_aura
    BEFORE UPDATE ON comment_aura
    FOR EACH ROW EXECUTE PROCEDURE update_comment_aura();
```

Trigger	**TRIGGER05**
Description	When a comment doesn't have a vote yet and a member votes, the new vote is added and the comment's and its owner's aura score is updated (+1 on upvotes or -1 on downvotes)

SQL Code:

```
DROP FUNCTION IF EXISTS insert_comment_aura CASCADE;
CREATE FUNCTION insert_comment_aura() RETURNS TRIGGER AS $$
  BEGIN
    IF NEW.upvote THEN
      UPDATE comment
        SET aura = aura + 1
        WHERE NEW.id_comment = comment.id;
      UPDATE member
        SET aura = aura + 1
        WHERE (SELECT id_owner FROM comment WHERE NEW.id_comment = comment.id) =
member.id;

    ELSIF NOT NEW.upvote THEN
      UPDATE comment
```

```
        SET aura = aura - 1
        WHERE NEW.id_comment = comment.id;
      UPDATE member
        SET aura = aura - 1
        WHERE (SELECT id_owner FROM comment WHERE NEW.id_comment = comment.id) =
member.id;
    END IF;
    RETURN NEW;
  END;
  $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS insert_comment_aura ON comment_aura CASCADE;
CREATE TRIGGER insert_comment_aura
  BEFORE INSERT ON comment_aura
  FOR EACH ROW EXECUTE PROCEDURE insert_comment_aura();
```

Trigger	TRIGGER06
Description	When a comment already has a vote and the member removes it, the vote is removed and the comment's and its owner's aura score is updated (-1 on removing an upvote or +1 on removing a downvote)

SQL Code:

```
DROP FUNCTION IF EXISTS delete_comment_aura CASCADE;
CREATE FUNCTION delete_comment_aura() RETURNS TRIGGER AS $$
  BEGIN
    IF OLD.upvote THEN
      UPDATE comment
        SET aura = aura - 1
        WHERE OLD.id_comment = comment.id;
      UPDATE member
        SET aura = aura - 1
        WHERE (SELECT id_owner FROM comment WHERE OLD.id_comment = comment.id) =
member.id;

    ELSIF NOT OLD.upvote THEN
      UPDATE comment
        SET aura = aura + 1
        WHERE OLD.id_comment = comment.id;
      UPDATE member
        SET aura = aura + 1
        WHERE (SELECT id_owner FROM comment WHERE OLD.id_comment = comment.id) =
member.id;
    END IF;
    RETURN NEW;
  END;
  $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS delete_comment_aura ON comment_aura CASCADE;
CREATE TRIGGER delete_comment_aura
  BEFORE DELETE ON comment_aura
  FOR EACH ROW EXECUTE PROCEDURE delete_comment_aura();
```

Trigger	TRIGGER07
Description	Guarantees that a post has a minimum of 1 and a maximum of 10 topics

SQL Code:

```
DROP FUNCTION IF EXISTS check_topics CASCADE;
CREATE FUNCTION check_topics() RETURNS TRIGGER AS $$
  DECLARE num_topics SMALLINT;
  DECLARE current RECORD;
  BEGIN
    IF TG_OP = 'INSERT' THEN
      current = NEW;
    ELSE
      current = OLD;
    END IF;
    SELECT INTO num_topics count(*)
    FROM post_topic
    WHERE current.id_post = post_topic.id_post;
    IF num_topics > 10 THEN
      RAISE EXCEPTION 'A post can only have a maximum of 10 topics';
    ELSIF num_topics < 1 THEN
      RAISE EXCEPTION 'A post must have at least 1 topic';
    END IF;
    RETURN NEW;
  END;
  $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS check_topics ON post_topic CASCADE;
CREATE TRIGGER check_topics
  AFTER INSERT OR DELETE ON post_topic
  FOR EACH ROW EXECUTE PROCEDURE check_topics();
```

Trigger	TRIGGER08
Description	Guarantees that a comment's date is after its parent post's date

SQL Code:

```
DROP FUNCTION IF EXISTS check_date_post CASCADE;
CREATE FUNCTION check_date_post() RETURNS TRIGGER AS $$
  BEGIN
    IF EXISTS (SELECT news_post.date_time FROM comment INNER JOIN news_post
      ON comment.id_post = news_post.id
      WHERE comment.date_time < news_post.date_time)
      THEN
        RAISE EXCEPTION 'A comment can only be posted after the post it refers
to.';
    END IF;
    RETURN NEW;
  END;
  $$ LANGUAGE plpgsql;
```

```
DROP TRIGGER IF EXISTS check_date_post ON comment CASCADE;
CREATE TRIGGER check_date_post
  BEFORE INSERT ON comment
  FOR EACH ROW
  EXECUTE PROCEDURE check_date_post();
```

Trigger	TRIGGER09
Description	Guarantees that a reply's date is after its parent comment's date

SQL Code:

```
DROP FUNCTION IF EXISTS check_date_comment CASCADE;
CREATE FUNCTION check_date_comment() RETURNS TRIGGER AS $$
  BEGIN
    IF EXISTS (SELECT C1.date_time FROM comment C1 INNER JOIN reply
      ON C1.id = reply.id_parent
      INNER JOIN comment C2
      ON C2.id = reply.id_comment
      WHERE C2.date_time < C1.date_time)
      THEN
        RAISE EXCEPTION 'A reply can only be posted after the comment it refers
to.';
    END IF;
    RETURN NEW;
  END;
  $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS check_date_comment ON comment CASCADE;
CREATE TRIGGER check_date_comment
  BEFORE INSERT ON comment
  FOR EACH ROW
  EXECUTE PROCEDURE check_date_comment();
```

Trigger	TRIGGER10
Description	Creates a follow notification for the member that got followed

SQL Code:

```
DROP FUNCTION IF EXISTS create_follow_notification CASCADE;
CREATE FUNCTION create_follow_notification() RETURNS TRIGGER AS $$
  BEGIN
    INSERT INTO follow_notification (id_notified, id_follower, date_time) VALUES
(NEW.id_followed, NEW.id_follower, now());
    RETURN NULL;
  END;
  $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS create_follow_notification ON member_follow CASCADE;
CREATE TRIGGER create_follow_notification
  AFTER INSERT ON member_follow
  FOR EACH ROW EXECUTE PROCEDURE create_follow_notification();
```

Trigger	**TRIGGER11**
Description	Creates a comment notification for a member if someone comments on a post made by them

SQL Code:

```
DROP FUNCTION IF EXISTS create_comment_notification CASCADE;
CREATE FUNCTION create_comment_notification() RETURNS TRIGGER AS $$
  BEGIN
    INSERT INTO comment_notification (id_notified, id_comment, date_time) VALUES
((SELECT id_owner FROM news_post WHERE news_post.id=NEW.id_post), NEW.id,
now());
    RETURN NULL;
  END;
  $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS create_comment_notification ON comment CASCADE;
CREATE TRIGGER create_comment_notification
  AFTER INSERT ON comment
  FOR EACH ROW EXECUTE PROCEDURE create_comment_notification();
```

Trigger	**TRIGGER12**
Description	Creates a reply notification for a member if someone replies to a comment made by them

SQL Code:

```
DROP FUNCTION IF EXISTS create_reply_notification CASCADE;
CREATE FUNCTION create_reply_notification() RETURNS TRIGGER AS $$
  BEGIN
    INSERT INTO reply_notification (id_notified, id_reply, date_time) VALUES
((SELECT id_owner FROM comment WHERE comment.id=NEW.id_parent), NEW.id_comment,
now());
    RETURN NULL;
  END;
  $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS create_reply_notification ON reply CASCADE;
CREATE TRIGGER create_reply_notification
  AFTER INSERT ON reply
  FOR EACH ROW EXECUTE PROCEDURE create_reply_notification();
```

Trigger	**TRIGGER13**
Description	Makes the owner upvote a newly created post

SQL Code:

```
DROP FUNCTION IF EXISTS auto_post_upvote CASCADE;
CREATE FUNCTION auto_post_upvote() RETURNS TRIGGER AS $$
  BEGIN
    INSERT INTO post_aura(id_post, id_voter, upvote) VALUES(NEW.id,
NEW.id_owner, 'true');
    RETURN NULL;
  END;
  $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS auto_post_upvote ON news_post CASCADE;
CREATE TRIGGER auto_post_upvote
  AFTER INSERT ON news_post
  FOR EACH ROW EXECUTE PROCEDURE auto_post_upvote();
```

Trigger	**TRIGGER14**
Description	Makes the owner upvote a newly created comment

SQL Code:

```
DROP FUNCTION IF EXISTS auto_comment_upvote CASCADE;
CREATE FUNCTION auto_comment_upvote() RETURNS TRIGGER AS $$
  BEGIN
    INSERT INTO comment_aura(id_comment, id_voter, upvote) VALUES(NEW.id,
NEW.id_owner, 'true');
    RETURN NULL;
  END;
  $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS auto_comment_upvote ON comment CASCADE;
CREATE TRIGGER auto_comment_upvote
  AFTER INSERT ON comment
  FOR EACH ROW EXECUTE PROCEDURE auto_comment_upvote();
```

4. Transactions

The following tables display the transactions needed to assure the integrity of the data in the presence of concurrent accesses.

T01	**Insert replies**
Justification	It is necessary to ensure a ROLLBACK in case any of the transaction's instructions result in an error. The isolation level is Repeatable Read, because, otherwise, an update of comment_id_seq could happen, due to an insert in the table comment committed by a concurrent transaction, and as a result, inconsistent data would be stored.
Isolation level	REPEATABLE READ

SQL Code:

```
BEGIN TRANSACTION;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

    INSERT INTO comment (body, id_member, id_post) VALUES ($body, $id_member,
$id_post);
    INSERT INTO reply (id_comment, id_parent) VALUES (currval('comment_id_seq'),
$id_parent);

COMMIT;
```

Annex A. SQL Code

A.1. Database schema

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

DROP TABLE IF EXISTS member CASCADE;
DROP TABLE IF EXISTS member_image CASCADE;
DROP TABLE IF EXISTS member_follow CASCADE;
DROP TABLE IF EXISTS administrator CASCADE;
DROP TABLE IF EXISTS news_post CASCADE;
DROP TABLE IF EXISTS topic CASCADE;
DROP TABLE IF EXISTS topic_follow CASCADE;
DROP TABLE IF EXISTS post_topic CASCADE;
DROP TABLE IF EXISTS comment CASCADE;
DROP TABLE IF EXISTS reply CASCADE;
DROP TABLE IF EXISTS post_image CASCADE;
DROP TABLE IF EXISTS post_aura CASCADE;
DROP TABLE IF EXISTS comment_aura CASCADE;
DROP TABLE IF EXISTS post_bookmark CASCADE;
DROP TABLE IF EXISTS follow_notification CASCADE;
DROP TABLE IF EXISTS comment_notification CASCADE;
DROP TABLE IF EXISTS reply_notification CASCADE;
DROP TABLE IF EXISTS post_report CASCADE;
DROP TABLE IF EXISTS comment_report CASCADE;
DROP TABLE IF EXISTS topic_report CASCADE;
DROP TABLE IF EXISTS member_report CASCADE;


-----------------------------------------
-- Create tables
-----------------------------------------

CREATE TABLE member_image (
    id serial PRIMARY KEY,
    file bytea NOT NULL
);

CREATE TABLE member (
    id serial PRIMARY KEY,
    username text NOT NULL UNIQUE,
    full_name text NOT NULL,
    email text NOT NULL UNIQUE,
    password text NOT NULL,
    bio text,
    id_profile_image integer NOT NULL DEFAULT 1 REFERENCES member_image(id) ON
DELETE SET DEFAULT,
    id_banner_image integer NOT NULL DEFAULT 2 REFERENCES member_image(id) ON
DELETE SET DEFAULT,
    aura integer DEFAULT 0 NOT NULL
);

CREATE TABLE administrator (
    id integer PRIMARY KEY REFERENCES member(id)
```

```
);

CREATE TABLE member_follow (
    id_followed integer REFERENCES member(id) ON DELETE CASCADE,
    id_follower integer REFERENCES member(id) ON DELETE CASCADE,
    PRIMARY KEY(id_follower, id_followed),
    CONSTRAINT follow_ids CHECK (id_followed <> id_follower)
);

CREATE TABLE news_post (
    id serial PRIMARY KEY,
    title text NOT NULL,
    body text,
    date_time timestamp NOT NULL DEFAULT now(),
    aura integer DEFAULT 0 NOT NULL,
    id_owner integer NOT NULL REFERENCES member(id) ON DELETE CASCADE
);

CREATE TABLE topic (
    id serial PRIMARY KEY,
    name text NOT NULL UNIQUE
);

CREATE TABLE topic_follow (
    id_topic integer REFERENCES topic(id) ON DELETE CASCADE,
    id_member integer REFERENCES member(id) ON DELETE CASCADE,
    PRIMARY KEY(id_topic, id_member)
);

CREATE TABLE post_topic (
    id_post integer REFERENCES news_post(id) ON DELETE CASCADE,
    id_topic integer REFERENCES topic(id) ON DELETE CASCADE,
    PRIMARY KEY(id_post, id_topic)
);

CREATE TABLE comment (
    id serial PRIMARY KEY,
    body text NOT NULL,
    date_time timestamp NOT NULL DEFAULT now(),
    aura integer DEFAULT 0 NOT NULL,
    id_owner integer NOT NULL REFERENCES member(id) ON DELETE CASCADE,
    id_post integer NOT NULL REFERENCES news_post(id) ON DELETE CASCADE
);

CREATE TABLE reply (
    id_comment integer PRIMARY KEY REFERENCES comment(id) ON DELETE CASCADE,
    id_parent integer NOT NULL REFERENCES comment(id) ON DELETE CASCADE,
    CONSTRAINT reply_ids CHECK (id_comment <> id_parent)
);

CREATE TABLE post_image (
    id serial PRIMARY KEY,
    id_post integer NOT NULL REFERENCES news_post(id) ON DELETE CASCADE,
    file bytea NOT NULL
);

CREATE TABLE post_aura (
    id_post integer REFERENCES news_post(id) ON DELETE CASCADE,
```

```
    id_voter integer REFERENCES member(id) ON DELETE CASCADE,
    upvote boolean NOT NULL,
    PRIMARY KEY(id_post, id_voter)
);

CREATE TABLE comment_aura (
    id_comment integer REFERENCES comment(id) ON DELETE CASCADE,
    id_voter integer REFERENCES member(id) ON DELETE CASCADE,
    upvote boolean NOT NULL,
    PRIMARY KEY(id_comment, id_voter)
);

CREATE TABLE post_bookmark (
    id_post integer REFERENCES news_post(id) ON DELETE CASCADE,
    id_bookmarker integer REFERENCES member(id) ON DELETE CASCADE,
    PRIMARY KEY(id_post, id_bookmarker)
);

CREATE TABLE follow_notification (
    id_notified integer REFERENCES member(id) ON DELETE CASCADE,
    id_follower integer REFERENCES member(id) ON DELETE CASCADE,
    date_time timestamp NOT NULL DEFAULT now(),
    PRIMARY KEY(id_notified, id_follower),
    CONSTRAINT follow_notification_ids CHECK (id_follower <> id_notified)
);

CREATE TABLE comment_notification (
    id_notified integer REFERENCES member(id) ON DELETE CASCADE,
    id_comment integer REFERENCES comment(id) ON DELETE CASCADE,
    date_time timestamp NOT NULL DEFAULT now(),
    PRIMARY KEY(id_notified, id_comment)
);

CREATE TABLE reply_notification (
    id_notified integer REFERENCES member(id) ON DELETE CASCADE,
    id_reply integer REFERENCES reply(id_comment) ON DELETE CASCADE,
    date_time timestamp NOT NULL DEFAULT now(),
    PRIMARY KEY(id_notified, id_reply)
);

CREATE TABLE post_report (
    id_reporter integer REFERENCES member(id) ON DELETE SET NULL,
    id_post integer REFERENCES news_post(id) ON DELETE CASCADE,
    body text NOT NULL,
    date_time timestamp NOT NULL DEFAULT now(),
    PRIMARY KEY(id_reporter, id_post)
);

CREATE TABLE comment_report (
    id_reporter integer REFERENCES member(id) ON DELETE SET NULL,
    id_comment integer REFERENCES comment(id) ON DELETE CASCADE,
    body text NOT NULL,
    date_time timestamp NOT NULL DEFAULT now(),
    PRIMARY KEY(id_reporter, id_comment)
);

CREATE TABLE topic_report (
    id_reporter integer REFERENCES member(id) ON DELETE SET NULL,
```

```
    id_topic integer REFERENCES topic(id) ON DELETE CASCADE,
    body text NOT NULL,
    date_time timestamp NOT NULL DEFAULT now(),
    PRIMARY KEY(id_reporter, id_topic)
);

CREATE TABLE member_report (
    id_reporter integer REFERENCES member(id) ON DELETE SET NULL,
    id_reported integer REFERENCES member(id) ON DELETE CASCADE,
    body text NOT NULL,
    date_time timestamp NOT NULL DEFAULT now(),
    PRIMARY KEY(id_reporter, id_reported),
    CONSTRAINT member_report_ids CHECK (id_reporter <> id_reported)
);
```

A.2. Database population

Because populate.sql is too large to fit in the wiki, the link to its separate file with the complete content is present here: [populate.sql]

Below are some fragments of populate.sql:

```
insert into member_image (id, file) values (1, bytea('profile_image'));
insert into member_image (id, file) values (2, bytea('banner_image'));


insert into member (id, username, full_name, email, password, bio,
id_profile_image, id_banner_image) values (1, 'BrotherSena', 'Gustavo Sena',
'up201806078@fe.up.pt',
'EF92B778BAFE771E89245B89ECBC08A44A4E166C06659911881F383D4473E94F', 'Virei
Monge!', 1, 2);
insert into member (id, username, full_name, email, password, bio,
id_profile_image, id_banner_image) values (2, 'El_biden', 'Andre Nascimento',
'up201806461@fe.up.pt',
'EF92B778BAFE771E89245B89ECBC08A44A4E166C06659911881F383D4473E94F', 'Da-me tu
sardita :P', 1, 2);
insert into member (id, username, full_name, email, password, bio,
id_profile_image, id_banner_image) values (3, 'kaka34', 'Caio Nogueira',
'up201806218@fe.up.pt',
'EF92B778BAFE771E89245B89ECBC08A44A4E166C06659911881F383D4473E94F', 'Nao mandas
em mim nao es minha mae', 1, 2);
insert into member (id, username, full_name, email, password, bio,
id_profile_image, id_banner_image) values (4, 'WanWan', 'Diogo Almeida',
'up201806630@fe.up.pt',
'EF92B778BAFE771E89245B89ECBC08A44A4E166C06659911881F383D4473E94F', 'Ta mal, ta
errado', 1, 2);
insert into member (id, username, full_name, email, password, bio,
id_profile_image, id_banner_image) values (5, 'tmc0', 'Thalia Mc Dermid',
'tmc0@arizona.edu',
'802b289be8283f1110f30cf2f7188dff437d01c72b8c129ad5dae89805b09839', 'Stand-alone
4th generation moratorium', 1, 2);
insert into member (id, username, full_name, email, password, bio,
id_profile_image, id_banner_image) values (10, 'hmerryweather5', 'Hally
Merryweather', 'hmerryweather5@ibm.com',
'87e899bb60ab87c207a63e769ccc0cb595550e1366c25171b3a64ba2cd211e44', 'Profit-
focused tangible internet solution', 1, 2);
```

```
insert into member (id, username, full_name, email, password, bio,
id_profile_image, id_banner_image) values (35, 'sbrosnanu', 'Shea Brosnan',
'sbrosnanu@epa.gov',
'a615b8fe9be681734664cc2c2a782775efef51e4624e4c19d97ac0869884d532', 'Versatile
bifurcated alliance', 1, 2);
insert into member (id, username, full_name, email, password, bio,
id_profile_image, id_banner_image) values (37, 'rdaenenw', 'Randie Daenen',
'rdaenenw@ox.ac.uk',
'a2bcb6ddc52c3cf0dac0b04fe9bc394527aa49dde54f876c46531d7a67d967d5', 'Total
global hierarchy', 1, 2);
insert into member (id, username, full_name, email, password, bio,
id_profile_image, id_banner_image) values (70, 'gpollandf', 'Gonzales Polland',
'gpollandf@tuttocitta.it', '6l5zqk', 'Fundamental systemic hub', 1, 2);



insert into administrator (id) values (1);
insert into administrator (id) values (2);
insert into administrator (id) values (3);
insert into administrator (id) values (4);


insert into member_follow (id_followed, id_follower) values (1, 3);
insert into member_follow (id_followed, id_follower) values (1, 5);
insert into member_follow (id_followed, id_follower) values (2, 4);
insert into member_follow (id_followed, id_follower) values (3, 5);
insert into member_follow (id_followed, id_follower) values (4, 1);


insert into news_post (id, title, body, date_time, id_owner) values (9, '9 Free
PS4 And PSVR Games Up For Grabs Now, Including Abzu And Enter The Gungeon', 'If
you''re looking for something new to play on PS4 or PS5 this weekend, you''re in
luck, as Sony''s Play at Home 2021 has made nine PS4 and PSVR games free to claim
for a limited time. Available now until April 22 at 8 PM PT / 11 PM ET, the list
includes an assortment of id, titles from indies like Abzu and Enter the Gungeon
to VR experiences like Moss and Thumper. Other games on the list include Rez
Infinite, Subnautica, The Witness, Astro Bot Rescue Mission, and Paper Beast. The
most enticing game in Sony''s Play at Home giveaway, however, is still yet to
come. Guerrilla Games'' Horizon Zero Dawn: Complete Edition will be available for
all PS4 players to download starting April 19 at 8 PM PT / 11 PM ET. You''ll have
until May 14 to claim Zero Dawn on PS4 or PS5.', TIMESTAMP '2020-03-26
23:47:42', 4);
insert into news_post (id, title, body, date_time, id_owner) values (16, 'Final
Fantasy 11 Mobile Reboot Cancelled 6 Years After Announcement', 'Six years after
it was announced, Final Fantasy XI Reboot, a mobile version of Square''s early
2000s MMORPG, has officially been cancelled.\nThe cancellation came as co-
developers Square Enix and Nexon decided the mobile version didn''t meet the
creative standards Final Fantasy fans have come to expect from the series,
according to Gamebiz.jp (via Gematsu). The development team has reportedly been
moved to other projects.', TIMESTAMP '2020-02-23 12:27:05', 3);
```

```
insert into news_post (id, title, body, date_time, id_owner) values (45,
'Fortnite Is Getting Its First Single-Player Story Event', 'Fortnite will soon
provide players with a new single-player story event that acts as a conclusion to
the current season of the battle royale.\nWhen Fortnite Chapter 2 Season 6
launches on March 16, the first thing players will be met with is the Zero Crisis
Finale. Described as a “solo experience” by developer Epic, it is the conclusion
of Agent Jones’ mission that formed the basis of the Season 5 story. Epic
promises that the aftermath of this single-player event will “shape Reality as we
know it”, suggesting that the event will push forward and make changes to
Fortnite’s storyline.', TIMESTAMP '2019-08-20 11:17:18', 5);
insert into news_post (id, title, body, date_time, id_owner) values (57, 'New
Smash Bros. Ultimate Freebie Available For Switch Online Members', 'NSO
subscribers can grab a free Spirits Set, which contains a random Legend-class and
Ace-class Spirit.\nNintendo Switch Online subscribers can claim another Super
Smash Bros. Ultimate freebie for a limited time. The latest exclusive offer for
NSO members is the Spirits Set 1. This free pack unlocks two random Spirits in
your game: one Ace-class Primary Spirit, and one Legend-class Support Spirit.\nTo
grab the Spirits Set, select the Nintendo Switch Online icon from your system''s
home screen and open the Special Offers tab. The Spirits Set will be listed among
the other free offers. Alternately, you can grab the freebie directly from the
Nintendo eShop by selecting Nintendo Switch Online from the left sidebar.',
TIMESTAMP '2018-03-03 17:28:34', 3);
insert into news_post (id, title, body, date_time, id_owner) values (67, 'Sonic
2 The Movie Starts Production -- "Lights, Camera, Hedgehog"', 'Director Jeff
Fowler announces that production has begun on the sequel.\nProduction has
officially begun on Sonic the Hedgehog 2 the movie, director Jeff Fowler has
announced. He confirmed the news on Twitter where he wrote, "lights, camera,
hedgehog."\nThat a sequel to Sonic is in the works is no surprise, as the first
Sonic the Hedgehog became the highest-grossing video game movie of all time in
the United States. Detective Pikachu, Warcraft, Rampage, and Prince of Persia:
Sands of Time did better globally, but Sonic is No. 1 in the US.Sonic 2 is
scheduled to release in theaters in April 2022. It''s expected to bring back Ben
Schwartz as the voice of Sonic, while other cast members like Tika Sumpter may
also return. We don''t know if Jim Carrey will be back as Dr. Robotnik or if the
rumors of Tails appearing in the sequel are true.', TIMESTAMP '2020-02-16
19:31:25', 2);


insert into topic (id, name) values (7, 'Console Gaming');
insert into topic (id, name) values (9, 'Nintendo');
insert into topic (id, name) values (30, 'Playstation');
insert into topic (id, name) values (34, 'Battle Royale');
insert into topic (id, name) values (47, 'Mobile Gaming');


insert into topic_follow (id_topic, id_member) values (30, 1);
insert into topic_follow (id_topic, id_member) values (30, 3);
insert into topic_follow (id_topic, id_member) values (34, 2);
insert into topic_follow (id_topic, id_member) values (34, 4);
insert into topic_follow (id_topic, id_member) values (47, 3);


insert into post_topic (id_topic,id_post) values (7,9);
insert into post_topic (id_topic,id_post) values (7,57);
insert into post_topic (id_topic,id_post) values (9,67);
insert into post_topic (id_topic,id_post) values (30,9);
insert into post_topic (id_topic,id_post) values (34,45);
insert into post_topic (id_topic,id_post) values (47,16);
```

```
insert into comment (id, body, date_time, id_owner, id_post) values (12, 'Nice',
TIMESTAMP '2020-03-26 23:52:12', 10, 9);
insert into comment (id, body, date_time, id_owner, id_post) values (46, 'Today
I learned there was mobile port of XI in the works, and it has been canceled.',
TIMESTAMP '2020-02-23 14:32:13', 37, 16);
insert into comment (id, body, date_time, id_owner, id_post) values (47, 'Today
was a roller coaster ride.', TIMESTAMP '2020-02-23 15:22:25', 35, 16);
insert into comment (id, body, date_time, id_owner, id_post) values (48, 'They
should remake 11''s story as a single player game.', TIMESTAMP '2020-02-23
16:23:16', 35, 16);
insert into comment (id, body, date_time, id_owner, id_post) values (49, 'No
they really shouldn''t. I say that as an FFXI player since 2003 launch to current
day. It''s best experienced with friends.', TIMESTAMP '2020-02-23 17:45:53', 70,
16);
insert into comment (id, body, date_time, id_owner, id_post) values (196, 'I''m
not a big Fortnite guy, but honestly I respect how Epic has been able to keep
this game relevant for years.', TIMESTAMP '2019-08-20 11:47:00', 3, 45);
insert into comment (id, body, date_time, id_owner, id_post) values (281, 'All
the Smash related packs have been a waste of what was initially promised with the
NSO launch. ', TIMESTAMP '2018-03-03 18:00:00', 2, 57);


insert into reply (id_comment, id_parent) values (47, 46);
insert into reply (id_comment, id_parent) values (49, 48);


insert into post_aura (id_post, id_voter, upvote) values (9, 1, 'true');
insert into post_aura (id_post, id_voter, upvote) values (9, 2, 'true');
insert into post_aura (id_post, id_voter, upvote) values (16, 5, 'false');
insert into post_aura (id_post, id_voter, upvote) values (45, 3, 'true');
insert into post_aura (id_post, id_voter, upvote) values (57, 2, 'false');
insert into post_aura (id_post, id_voter, upvote) values (67, 3, 'false');


insert into comment_aura (id_comment, id_voter, upvote) values (12, 1, 'true');
insert into comment_aura (id_comment, id_voter, upvote) values (12, 5, 'false');
insert into comment_aura (id_comment, id_voter, upvote) values (47, 3, 'true');
insert into comment_aura (id_comment, id_voter, upvote) values (49, 3, 'true');
insert into comment_aura (id_comment, id_voter, upvote) values (196, 5,
'false');
insert into comment_aura (id_comment, id_voter, upvote) values (281, 1, 'true');


insert into post_bookmark (id_post, id_bookmarker) values (57, 3);
insert into post_bookmark (id_post, id_bookmarker) values (67, 1);
insert into post_bookmark (id_post, id_bookmarker) values (67, 5);


insert into post_report (id_reporter, id_post, body, date_time) values (2, 9,
'This is misinformation', TIMESTAMP '2019-03-20 09:42:33');
insert into comment_report (id_reporter, id_comment, body, date_time) values (1,
49, 'This is abusive or harassing', TIMESTAMP '2019-11-10 22:02:30');
insert into topic_report (id_reporter, id_topic, body, date_time) values (4, 47,
'Posts are offensive', TIMESTAMP '2020-03-29 17:29:45');
insert into member_report (id_reporter, id_reported, body, date_time) values (5,
2, 'Impersonating someone', TIMESTAMP '2020-03-29 17:29:33');
```

Revision history

Changes made to the first submission: N/A

GROUP2133, 14/04/2021

- André Nascimento, up201806461@fe.up.pt
- Caio Nogueira, up201806218@fe.up.pt
- Diogo Almeida (editor), up201806630@fe.up.pt
- Gustavo Sena, up201806078@fe.up.pt