




Foundations of Databases, A.Y. 2020/2021
Master Degree in Computer Engineering
Master Degree in ICT for Internet and Multimedia
Homework 1 – Requirements Analysis

Deadline: October 23, 2020

Group HyperTeam	Project  HyperU HyperU	
Last Name Alecci Martinelli Zirollo	First Name Marco Luca Elia	Student Number 2013384 2005837 2019297

Objectives of the System

HyperU is a web application in which users can share project ideas and create teams based on required skills to realize the project. The users can customize their profile, select their skills and can post ideas on the main page of the application. Each post can be liked and commented

by other users that can ask to work on the project. The author of the post can create a team, and each one of them will have its own page to discuss the project and share contents and links.

The application needs a database for the management of the user's accounts (registration, login, skills, and editing of personal information), the ideas shared (posting, editing, comments, likes) and the creation of the teams with a group chat for each project.



Interviews

A technology company specialized in online services and products wants to launch a new web application called HyperU.

Interviews were conducted with the Head Of Product Development of the company, who told us how the application should work and should be like.

Users and Stakeholders of the System

The following people will make use of the system:

- **Application's users:** they will register to the application and use it
- **Moderators:** they will check and hide inappropriate users' posts and comments
- **Administrators:** they have to add new skills and topics and choose the moderators to guarantee the correct functioning of the application

Natural Language Sentences

HyperU is a web application in which users can share project ideas and create teams based on required skills to realize the project. This way people with good ideas but without the competences to realize them can meet the most suitable person to work with. Or vice versa people that are good at doing something but maybe lack creativity, can join a team to help other people develop their ideas. This can, for instance, lead to the creation of a start-up, but can also be used by students to create group works for university and school projects.

Everyone who wants to use HyperU needs to sign up with a form entering name, surname, date of birth and email. Then the user will have to pick a unique username and a password. For legal reasons, users can subscribe to the application only if they are sixteen or older.

After being registered, the user can login using his username and password. The password must be stored safely in the database.

The new user can customize his personal page by editing the information he entered in the register form (e.g. name, surname, username, email, password etc...). He can also add more information such as a profile picture, his gender and a small biography. The biography must support markdown language (e.g. adding links, setting texts in bold, underline or italic, etc...).

One of the most important thing that a user has to do is to specify his skills in the profile page. Examples of skills could be problem solving, the knowledge of a specific programming language like Java, the languages he speaks, etc.... These skills can be found and chosen from a list.

If the skill the user is looking for isn't on the list, he can request the administrators to add it.

The main purpose of the app is to share ideas among all the users and create teams to work on them. In order to do it each user can create a post about his idea, including a title, an image (like a small presentation or a schema of the project), a description, a list of skills required to build the project and a list of topics the system will use to classify the post. Topics, just like skills, can be found and chosen from a list. Administrators can update this list adding more topics. The description supports markdown language.

A user can also add in his personal page a list of topics he is interested in, so the application can purposely show him on the main page only ideas that he would find stimulating. Otherwise, the posts aren't filtered by topics and they are listed by chronological order.

The application also has a search page, thanks to which a user can search other profiles. He can also see other posts that can be filtered by topics or skills. Users' profiles are public and on their pages are shown their posts, profile picture, username, name, surname, and a small part of the biography. By clicking the "more information" button, the whole biography and the list of the user's skills become visible. A post can also be liked and commented, but the user can undo these actions whenever he wants.

When a user is interested in one specific idea, he can request to join the team that will realize that project. He can also attach a small message, for example explaining why he should be part of the team. Users whose skills aren't required to realize the project can send the request too. The user who posted the idea, in fact, can accept or refuse that request, after checking the requestor profile and his message.

After at least one user is accepted by the post's author, the members of the team have access to a private group chat in which they can discuss the project, share links, media and other types of files.

A user can edit his own posts, changing the title, the image, the description, the list of the required skills, and he can set if other users can ask to join the team.

Some users, called moderators, have more privileges than common users, in fact they can hide inappropriate posts and comments. Moderators are manually set by the administrators of the application. Administrators also have the same privileges of moderators and they are users too.

Filtered Sentences

HyperU is a web application that needs a database to manage the users and let all the features to work properly:

- **Users** are those who use the application:
 - They have to sign up to the application giving name, surname, birthdate and email
 - During the registration process they have to choose a unique username and a password that they will use to log in

- Each user has a **personal page**:
 - the page contains the main information (name, surname, username, gender, profile picture, biography, **skills**) about the user and the ideas he has shared
 - the users can also add a profile picture, their gender and a biography
 - all of this information can be edited anytime by the owner of the profile
- **Moderators** are users with more privileges than common users, in fact:
 - They can hide inappropriate comments
 - They can hide inappropriate posts
- **Administrators** are the most powerful users, in fact they have the same privileges of the moderators, but they can also add new skills and topics and choose the moderators to guarantee the correct functioning of the application
- **Skills** show the field of competences of the users, allowing the users to find the right teammates for each specific project. Users can select their skills from a list and if the one they are looking for isn't on the list, they can request the **administrators** to add it to the list
- **Topics** are used to classify **ideas**. Users can add in their profile topics they are interested in; this way the application will show in the main page only the ideas that each user would like
- **Ideas** are the main concept of the application:
 - Each user can post ideas on the main page of the application
 - Each post can include the title of the idea, an image (like a small presentation or a schema of the project) and a small description
 - Each post can also be liked and commented by other users
 - Each post includes also a list of the **skills** required to build the project
 - Each post has to be classified selecting the related **topics** from a list
 - Each user can edit the data of the post (title, image, description, skills, topics) and allow other users to request to be part of the team that will realize the project
 - The users can ask to join a project if they are interested in it
- **Teams** are associated to a specific idea and represents a group of people working on the same project:
 - When an idea's owner accepts the request of at least another user, a group chat is created between all the members
 - The group chat allows the members to discuss the project and share links, media and other types of file

Term Glossary

Term	Description	Synonyms	Connection
User	Who uses the application		Idea, Personal Page, Skill
Moderator	User with more privileges than common users		User, Idea, Comment
Administrator	Users with more privileges than moderators		User, Skill, Topic, Idea, Comment
Registration	Act of creating a new account	Sign up	User
Login	Act of logging into the account	Sign in	User
Personal Page	The page containing the main information about the user	Profile page	User
Main Page	The page viewed by the user after the login. In this page all the ideas related to topics that the user has checked as interesting are listed		User, Idea, Topic
Search Page	The page used by users to find ideas and other users. Ideas can be filtered by topics or skills		User, Idea, Topic, Skill
Skill	Represents a specific competence (e.g. knowing a certain language)	Competence	User, Idea
Topic	It is used to classify and filter the ideas		User, Idea
Idea	It is posted by a user on the main page	Post, Project	User, Skill, Team
Like	Reaction given to a post that the user likes		User, Idea
Comment	A small sentence attached by users under a post		User, Idea
Team	Group of people working on the same idea	Group	User, Idea
Request	Act of asking to join another user's team, can come with an attached message		User, Idea
Group Chat	Chat of the team, in which users talk about the project and share files, links or media.	Chat	User, Team, Idea

Functional Requirements

The database must store all the data needed for the correct functioning of the application. In particular:

- The system must store:
 - Users data and their role
 - A list of selectable skills
 - A list of selectable topics
 - All the data related to the posts, likes and comments
 - All the data related to the group chats
- The system must provide the following functions:
 - Let the user create and modify their own profile page
 - Let the user create, modify and remove their own posts
 - Let the user create and remove their own comments and likes to other posts
 - Let the user send messages, files and medias
 - Let the moderators delete other users' posts and comments

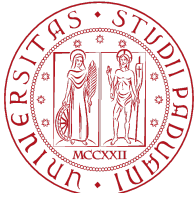
Non Functional Requirements

- The system must be accessible through multiple platforms (desktop, mobile)
- Login operations should be carried out in a secure way using SSL protocols
- Password must be stored securely, after an hashing cryptation
- The biography of a user must be less or equal to 700 characters
- The biography and the post description have to support markdown language
- The message attached to a request must be less or equal to 250 characters
- The image of a post must be less or equal to 5MB
- Medias and files shared in the group chat must be less or equal to 2GB

Constraints

The DBMS should satisfy the following constraints:


- Be implemented with PostgreSQL
- The client side will be implemented with HTML, CSS, JavaScript, and JQuery
- The server side will be implemented with Tomcat, java servlet, JSP, MVC and REST web services
- Operating System: Linux



Foundations of Databases, A.Y. 2020/2021
Master Degree in Computer Engineering
Master Degree in ICT for Internet and Multimedia

Homework 2 – Conceptual and Logical Design

Deadline: November 27, 2020

Group HyperTeam	Project	
	 HyperU HyperU	
Last Name	First Name	Student Number
Alecci	Marco	2013384
Martinelli	Luca	2005837
Zioldo	Elia	2019297

Conceptual Design

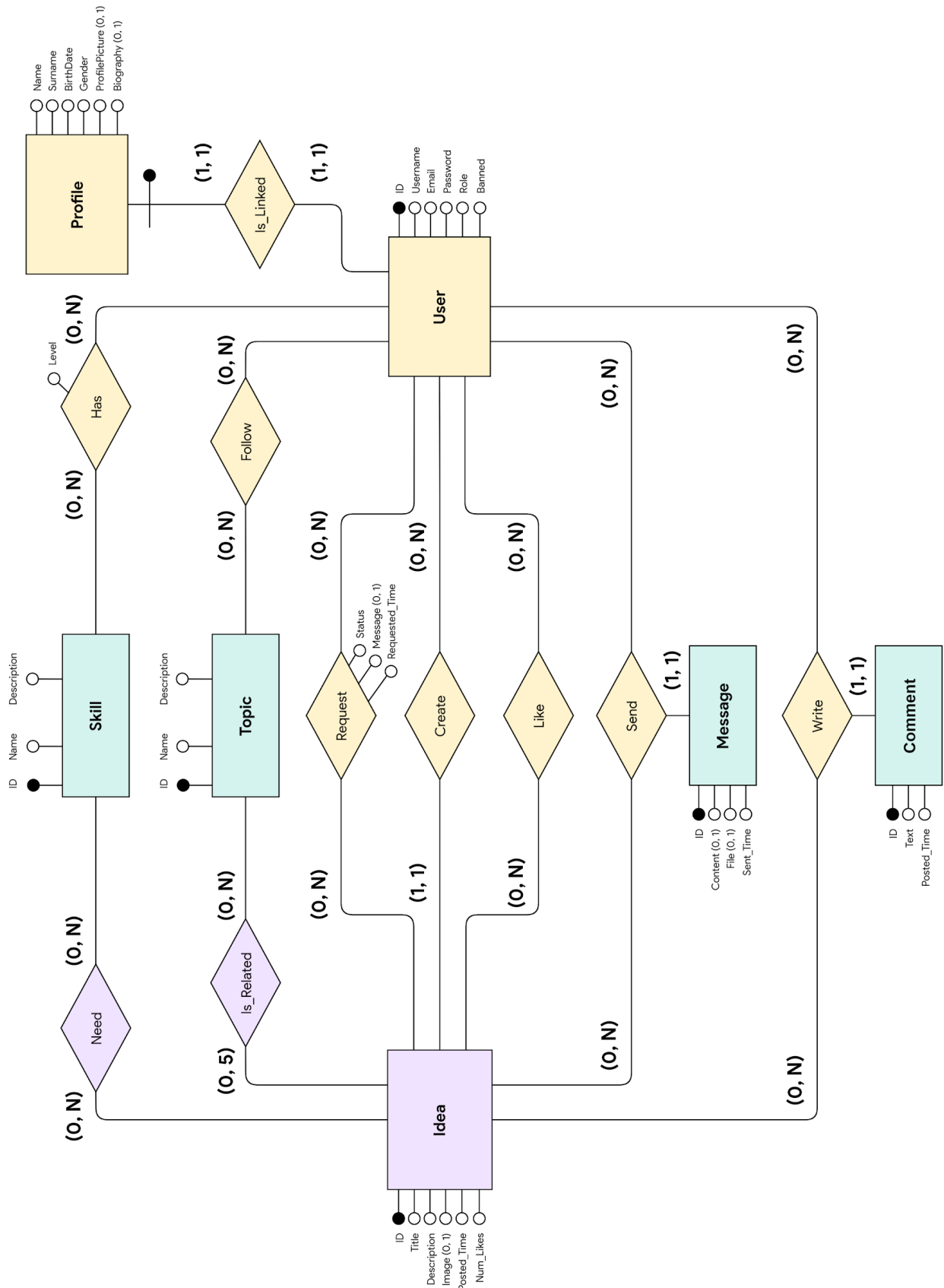
Variations to the Requirement Analysis

The main and the search page are the result of a query operation, if the user wants to filter the ideas by topics, another query has to be executed. In the post it's also shown the post creator. If a user wants to quit a team, he can do it and he will no longer participate in the group chat. The teams can be as big as we want, but of course, it's the creator who decides if continue to accept other people. Moderators can ban or readmit users, except for other moderators or administrators. The idea creator can't invite other users, only a user can request to participate in a project, anyway, the creator can contact a user using the contact information on his profile page. The image of the post is not mandatory. If the post doesn't have an image the title will be shown bigger, with a colored background.

The functional requirement also comprehends search operations likes:

- search idea by title, topic, or skills required
- search user by username, name, or skill

Entity-Relationship Schema



Data Dictionary

Entities Table

Entity	Description	Attributes	Identifier
User	Represents data about a user that uses the application	<ul style="list-style-type: none">• ID: a user identifier, serial• Username: unique username chosen by the user, text• Email: email of the user, text• Password: login password is chosen by the user, text• Role: role of the user in the application, enumeration among (Common User, Moderator, Administrator)• Banned: define if a user is banned, boolean	ID
Profile	The profile page of a specific user	<ul style="list-style-type: none">• Name: the first name of the user, text• Surname: last name of the user, text• Birthdate: date of birth, Date• Gender: the gender indicated by the user, enumeration among (Male, Female, Not declared)• Profile Picture (0,1): an image chose by the user, blob• Biography (0,1): small biography of the user, text	External Identifier: ID from USER
Idea	The idea posted by a user on the main page	<ul style="list-style-type: none">• ID: identifier of the post, serial	ID

		<ul style="list-style-type: none"> • Title: the title of the project, text • Description: small description of the project, text • Image (0,1): image associated with the post, blob • Posted_Time: date and time relative to the creation of the post, Datetime • Num_Likes: number of total likes to the post, integer 	
Comment	A small sentence attached by users under a post	<ul style="list-style-type: none"> • ID: identifier of the comment, serial • Text: the content of the comment, text • Posted_Time: date and time relative to the creation of the comment, Datetime 	ID
Message	Message in the group chat of an idea	<ul style="list-style-type: none"> • ID: identifier of the message, serial • Content (0,1): the content of the message, text • File (0,1): a file sent in the group chat, FILE_TYPE • Sent_Time: date and time relative to the sending of a message, Datetime 	ID
Skill	Represent the competences of a user	<ul style="list-style-type: none"> • ID: identifier of a skill, serial • Name: name representing the skill, text • Description: the description of the skill, text 	ID
Topic	Information used to classify and filter ideas	<ul style="list-style-type: none"> • ID: identifier of a topic, serial 	ID

		<ul style="list-style-type: none"> • Name: name representing a topic, text • Description: description of a topic, text 	
--	--	--	--

Relationship Table

Relationship	Description	Component Entities	Attributes
Is_Linked	Associates the user to his profile page	<ul style="list-style-type: none"> • User (1,1) • Profile (1,1) 	
Has	Relates each user to the skill he owns	<ul style="list-style-type: none"> • User (0,N) • Skill (0,N) 	<ul style="list-style-type: none"> • Level: represent the level of the skill, integer between 1 and 5
Is_Related	Associates project with skills that are needed	<ul style="list-style-type: none"> • Topic (0,N) • Idea (0,N) 	
Need	Manages projects with topics that are related to	<ul style="list-style-type: none"> • Skill (0,N) • Idea (0,N) 	
Follow	Associates the users with the topics they are following	<ul style="list-style-type: none"> • User (0,N) • Topic (0,N) 	
Request	Links users to the project they participate in	<ul style="list-style-type: none"> • User (0,N) • Idea (0,N) 	<ul style="list-style-type: none"> • Status: describe the status of the request, enumeration among (Pending, Accepted) • Message (0,1): the content of the message associated with the request, text • Requested_Time: time and date of the request, Datetime
Create	Links users to the project they have created	<ul style="list-style-type: none"> • User (0,N) • Idea (1,1) 	

Like	Keeps track of the likes given by the user	<ul style="list-style-type: none"> • User (0,N) • Idea (0,N) 	
Send	Keeps track of the messages in the group chat by the user	<ul style="list-style-type: none"> • User (0,N) • Idea (0,N) • Message (1,1) 	
Write	Keeps track of the comments in the group chat by the user	<ul style="list-style-type: none"> • User (0,N) • Idea (0,N) • Message (1,1) 	

External Constraints

- A MESSAGE must contain some text or one file, but not both simultaneously.

Functional Requirements Satisfaction Check

Store users' data and their role: the entity *USER* records this information through its attributes. With the attribute *ROLE* is possible to specify if a user is a common user, a moderator, or an administrator.

Store a list of selectable skills: the entity *SKILL* records this information through its attributes.

Store a list of selectable topics: the entity *TOPIC* records this information through its attributes.

Store data about comments: the entity *COMMENT* is associated with a ternary relationship to *USER* and *IDEA* and contains the content of the comment and when it was written.

Store data about the post: the entity *IDEA* contains all the information about the post like the title, the description, the image, and the posted date. Since the relationship *CREATE* has cardinality (1,1) to *USER*, it is possible to easily retrieve the post's creator.

Store data about likes: the relationship *LIKES* between *USER* and *IDEA* records all the likes given by a user.

Store data related to the group chat: the entity *MESSAGE* is associated with a ternary relationship to *USER* and *IDEA*. The attributes *Content* and *File* are optional because the message can contain a text or a file, but at least one of the two.

Allow the user to create and modify their profile page: Since the relationship *IS_LINKED* relates the entities *USER* and *PROFILE* with cardinality (1, 1), each user can modify his profile page.

Allow the user to create and modify and remove their post: The user can edit his posts changing the attributes of the entity *IDEA*. The creator of the post is saved into the relationship *CREATE*, which has cardinality (1, 1) from *IDEA* to *USER*.

Allow the user to create and remove their comments to other posts: The relationship COMMENT stores all the information needed to identify which USER has posted the COMMENT on a POST.

Allow the user to likes and unlike other posts: The relationship LIKE stores all the information needed to identify which USER has liked a POST.

Allow the user to send messages and files to the group chat: The relationship SEND stores the information about the MESSAGE of the group chat of a specific IDEA. The members (USER) of the group are the ones in the relationship REQUEST with attribute Status “accepted”.

Allow the user to search ideas by title, topic, or skill: The title is stored in the entity IDEA, meanwhile the lists of SKILLS and TOPICS are stored respectively in the relationships NEED and IS_RELATED.

Allow the user to search other users by username, name, or skill: The username is stored in the entity USER, the name is stored in the entity PROFILE, which is related with cardinality (1, 1) to the USER, meanwhile, the list of SKILLS is stored in the relationship IS_LINKED.

Logical Design

Transformation of the Entity-Relationship Schema

1. Redundancy Analysis

The schema does not contain any cycle of entities but presents the derivative attribute Num_Likes in the Idea entity. We report below the analysis of the database load to check whether to keep this attribute or not.

2. Choice of Principal Identifiers

The schema does not contain external identification cycles and the main identifiers comply with the selection criteria.

Analysis of Database Load

In this section, we report the analysis of the database to justify the presence of redundancies in the entity-relationship schema. Consider the following two example operations that involve the redundant attribute Num_Likes

- O1: Store a new like to a post.
- O2: Show the total number of likes of a post.

Table 1 reports the description of each operation, its frequency, and type. Both O1 and O2 are online operations since they are interactive operations done while the user is

viewing the post. The frequencies in the table are referred to the operation done by a single user. It has been estimated that a user likes an average of 30 posts a day, and he sees an average of 100 posts during an entire day.

Operation	Description	Frequency	Type
O1 New like	Store a new like to a post	30/Day	Online
O2 Total number of likes	Show the total number of likes of a post	100/Day	Online

Table 1: Frequency table

Table 2 reports the access/volume data related to operation O1 with redundancy.

Concept	Construct	Access	Type	Average Access
Like	Relationship	1	W	$1 \times 30 \times 2 = 60$
Idea	Entity	1	R	$1 \times 30 \times 1 = 30$
Idea	Entity	1	W	$1 \times 30 \times 2 = 60$
Total Access			150	

Table 2: Access/Volume for operation O1 with redundancy

With the redundancy when a user likes a post, we need first to store the information in the relationship Like, and then to update the attribute Num_Likes

Table 3 reports the access/volume data related to operation O2 with redundancy.

Concept	Construct	Access	Type	Average Access
Idea	Entity	1	R	$1 \times 30 \times 1 = 30$
Total Access			30	

Table 3: Access/Volume for operation O2 with redundancy

With the redundant attributes Num_Likes, we need only one access to read the total number of likes.

Table 4 reports the access/volume data related to operation O1 without redundancy.

Concept	Construct	Access	Type	Average Access
Like	Relationship	1	W	$1 \times 30 \times 2 = 60$
Total Access			60	

Table 4: Access/Volume for operation O1 without redundancy

Without the redundancy, when a user likes a post, we need only one access to the relationship Like to store the information.

Table 5 reports the access/volume data related to operation O2 without redundancy. It has been estimated that a post has an average number of 150 likes.

Concept	Construct	Access	Type	Average Access
Like	Relationship	150	R	$150 \times 30 \times 1 = 450$
Total Access				450

Table 5: Access/Volume for operation O2 without redundancy

Without the redundant attribute Num_Likes, when we want to count to the total number of likes, we need to have a lot of access to the relationship Like.

Operation	With Redundancy	Without Redundancy
O1	150	60
O2	30	450
Total Access/Day	180	510

Table 6: Comparison of the Number of Accesses for each Operation

As shown in Table 6, the Num_Likes attribute significantly improves operations performance. For this reason, we decided to keep the redundant attribute in our schema.

Relational Schema

Send, *Write*, and *Create* are one to many relationships.

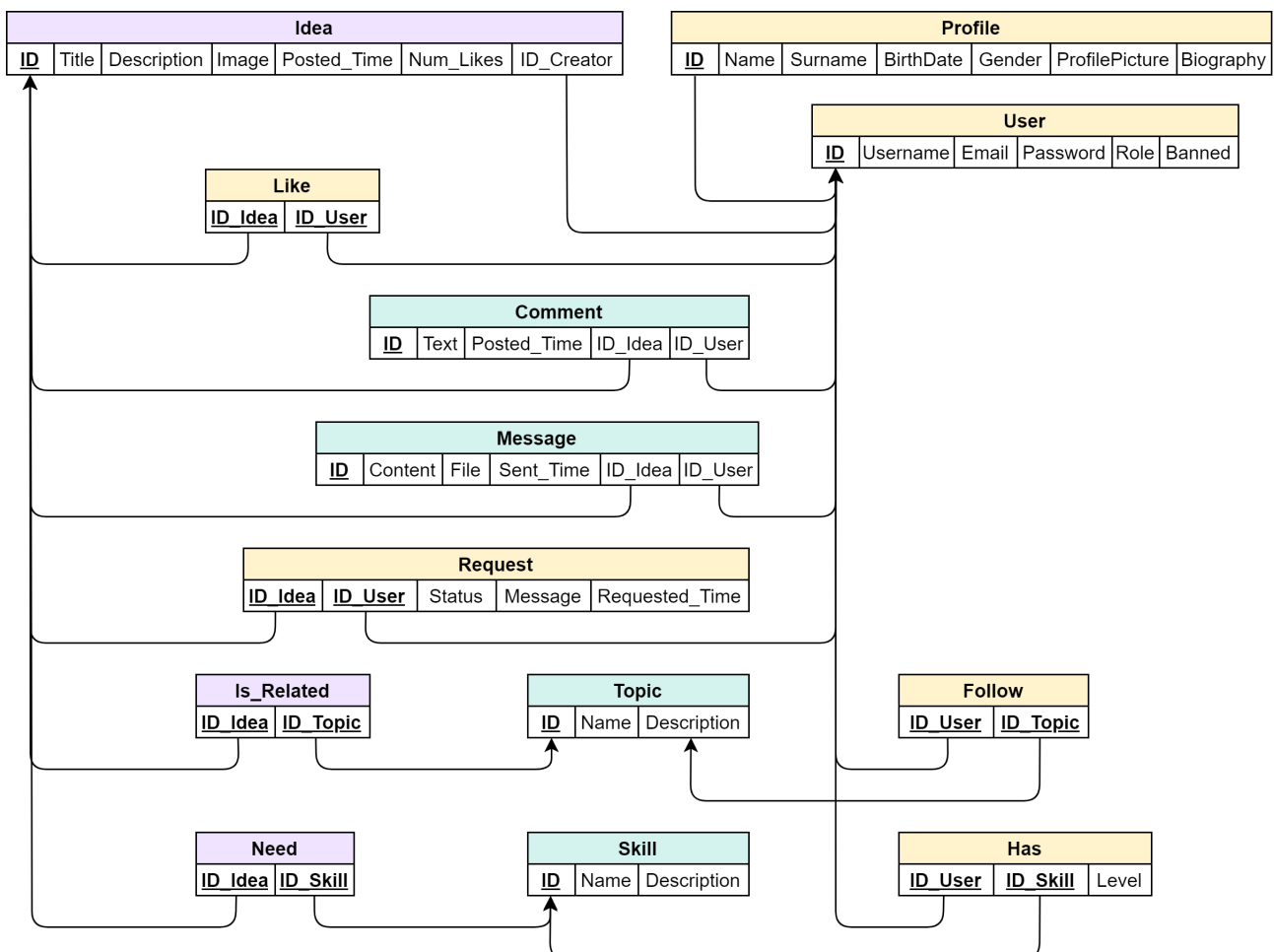
For *Send*, we choose not to introduce a new relation and to directly add in the *Message* relation the ID_User/ID_Idea information as Foreign Key to *User* and *Idea* relations. This is because when we access the *Message* relation, we want to know directly the information about who has sent the message and about which idea.

The same consideration can be done for *Write* and *Create* for analogue reasons.

Is_Linked is the only one to one relationship, and since *Profile* is externally identified by it, we add the Foreign Key ID_User in the *Profile* relation.

All the other relationships are many to many relationships, and since there are not many different ways to transform them, they are transformed straightforwardly.

A *Message* cannot both contain a text message and a file so at least one of the two fields will be NULL, but this does not affect the system performance.



Data Dictionary

Relation	Attribute	Description	Domain	Constraints
User	ID	User identifier	Serial	Primary Key
	Username	A unique username is chosen by the user	Text	Not NULL
	Email	Email of the user	Text	Not NULL
	Password	The login password is chosen by the user	Text	Not NULL
	Role	Role of the user in the application	Enumerate	Not NULL, Enumeration in { Common User, Moderator, Administrator }
	Banned	Define if a user is banned	Boolean	Not NULL
Profile	ID	User identifier	Serial	Primary Key, Foreign Key to User
	Name	Name of the user	Text	Not NULL
	Surname	The surname of the user	Text	Not NULL
	BirthDate	Date of birth of the user	Date	Not NULL
	Gender	Gender of the user	Enumerate	Not NULL, Enumeration in { Male, Female, Not Declared }
	ProfilePicture	The profile picture was chosen by the user	Blob	
	Biography	Some text about the user	Text	
Skill	ID	Skill identifier	Serial	Primary Key
	Name	Name of the skill	Text	Not NULL
	Description	Description of the skill	Text	Not NULL
Topic	ID	Topic identifier	Serial	Primary Key
	Name	Name of the topic	Text	Not NULL
	Description	Description of the topic	Text	Not NULL
Idea	ID	Idea identifier	Serial	Primary Key
	Title	Title of the idea chosen by the user	Text	Not NULL
	Description	Description of the idea	Text	Not NULL
	Image	Image attached to the idea	Blob	
	Posted_Time	The exact time of when the idea was posted	DateTime	Not NULL
	Num_Likes	The number of likes to the idea	Int	Not NULL

	ID_Creator	Identifier of the user who created the post	Serial	Not NULL, Foreign key to User,
Message	ID	Message identifier	Serial	Primary Key
	Content	Text message	Text	
	File	File attached to the message	Blob	
	Sent_Time	The exact time of when the message was sent	DateTime	Not NULL
	ID_Idea	Identificator of the Idea linked to the group chat	Serial	Not NULL, Foreign Key to Idea
	ID_User	Identificator of the User who sent the message	Serial	Not NULL, Foreign Key to User
Comment	ID	Comment identifier	Serial	Primary Key
	Text	The string of the comment	Text	Not NULL
	Posted_Time	The exact time of when the comment was sent	DateTime	Not NULL
	ID_Idea	Identificator of the Idea user wants to comment	Serial	Not NULL, Foreign Key to Idea
	ID_User	Identificator of the User who writes the comment	Serial	Not NULL, Foreign Key to User
Has	ID_User	Identifier of a specific user	Serial	Primary Key with ID_Skill, Foreign Key to User
	ID_Skill	The skill of the specified user	Serial	Primary Key with ID_User, Foreign Key to Skill
	Level	Level of the skill of the user	Int between (0,5)	Not NULL
Follow	ID_User	Identifier of a specific user	Serial	Primary Key with ID_Topic, Foreign key to User
	ID_Topic	Topics followed by the specified user	Serial	Primary Key with ID_User, Foreign key to Topic
Request	ID_Idea	The idea (project) for which has been asked a request	Serial	Primary Key with ID_User, Foreign key to Idea
	ID_User	The user that request to enter the idea (project)	Serial	Primary Key with ID_Idea, Foreign key to User
	Status	Status of the request	Enumerate	Not NULL, Enumeration in { Accepted, Pending }

	Message	Text message attached to the request	Text	
	Requested_Time	The exact time of when the request was sent	Datetime	Not NULL
Like	ID_Idea	Identifier of a specific idea	Serial	Primary Key with ID_User, Foreign key to Idea
	ID_User	The user that has put "like" on the idea	Serial	Primary Key with ID_Idea, Foreign key to User
Need	ID_Idea	Identifier of a specific idea	Serial	Primary Key with ID_Skill, Foreign key to Idea
	ID_Skill	The skill needed to realize the idea	Serial	Primary Key with ID_Idea, Foreign key to Skill
Is_Related	ID_Idea	Identifier of a specific idea	Serial	Primary Key with ID_Topic, Foreign key to Idea
	ID_Topic	Topic related to the idea	Serial	Primary Key with ID_Idea, Foreign key to Topic


External Constraints on Relational

- Only Moderators and Administrator can change the attribute Banned from the User Relation
- The pair (ID_Idea, ID_User) in the Message Relation has to be in the Request Relation with attribute Status set to "Accepted". In fact, a user can't send a message in a group in which is not inserted
- The pair (ID_Idea, ID_User) in the Request Relation is not allowed if the Idea associated with ID_Idea has as ID_Creator the same ID_User



Foundations of Databases, A.Y. 2020/2021
Master Degree in Computer Engineering
Master Degree in ICT for Internet and Multimedia
Homework 3 – Physical Design

Deadline December 16, 2020

Group HyperTeam	Project  HyperU HyperU	
Last Name Alecci Martinelli Ziroldo	First Name Marco Luca Elia	Student Number 2013384 2005837 2019297

Variations to the Relational Schema

We've added two new Entities called Team and Group chat to decompose the 'Idea' entity into different entities to better clarify the distinction between Idea, Team, and Group Chat. We've decided to distinguish the roles of the users with an enumeration attribute instead of creating new entities because we think that no other roles will be necessary, so, for simplicity, we've chosen to use enumeration.

These are the modifications made to the **Data Dictionary** (the rest of the Data Dictionary remain the same as in the HW2) :

Relation	Attribute	Description	Domain	Constraints
Team	ID	Team identifier	Serial	Primary Key
	Creation_Time	Date and Time of creation of the team	Datetime	Not NULL
	ID_Idea	Identifier of the Idea related to the team	Serial	Foreign Key to Idea
	Accept_Requests	Define if other users can send requests for join the team	Boolean	Not NULL

Group_Chat	ID	Group identifier	Serial	Primary Key, Foreign Key to Team
	Image	Image of the group chat	Blob	
	Description	Text description showed in the chat page	Text	
Message	ID	Message identifier	Serial	Primary Key
	Content	Text message	Text	
	File	File attached to the message	Blob	
	Sent_Time	The exact time of when the message was sent	DateTime	Not NULL
	ID_Group	Identificator of the Group Chat where the message was sent	Serial	Not NULL, Foreign Key to Group Chat
	ID_User	Identificator of the User who sent the message	Serial	Not NULL, Foreign Key to User
Request	ID_Team	The team for which has been asked a request	Serial	Primary Key with ID_User, Foreign key to Team
	ID_User	The user that request to enter the idea (project)	Serial	Primary Key with ID_Idea, Foreign key to User
	Status	Status of the request	Enumerate	Not NULL, Enumeration in { Accepted, Pending }
	Message	Text message attached to the request	Text	
	Requested_Time	The exact time of when the request was sent	Datetime	Not NULL

Figure 1 shows the updated relational schema.

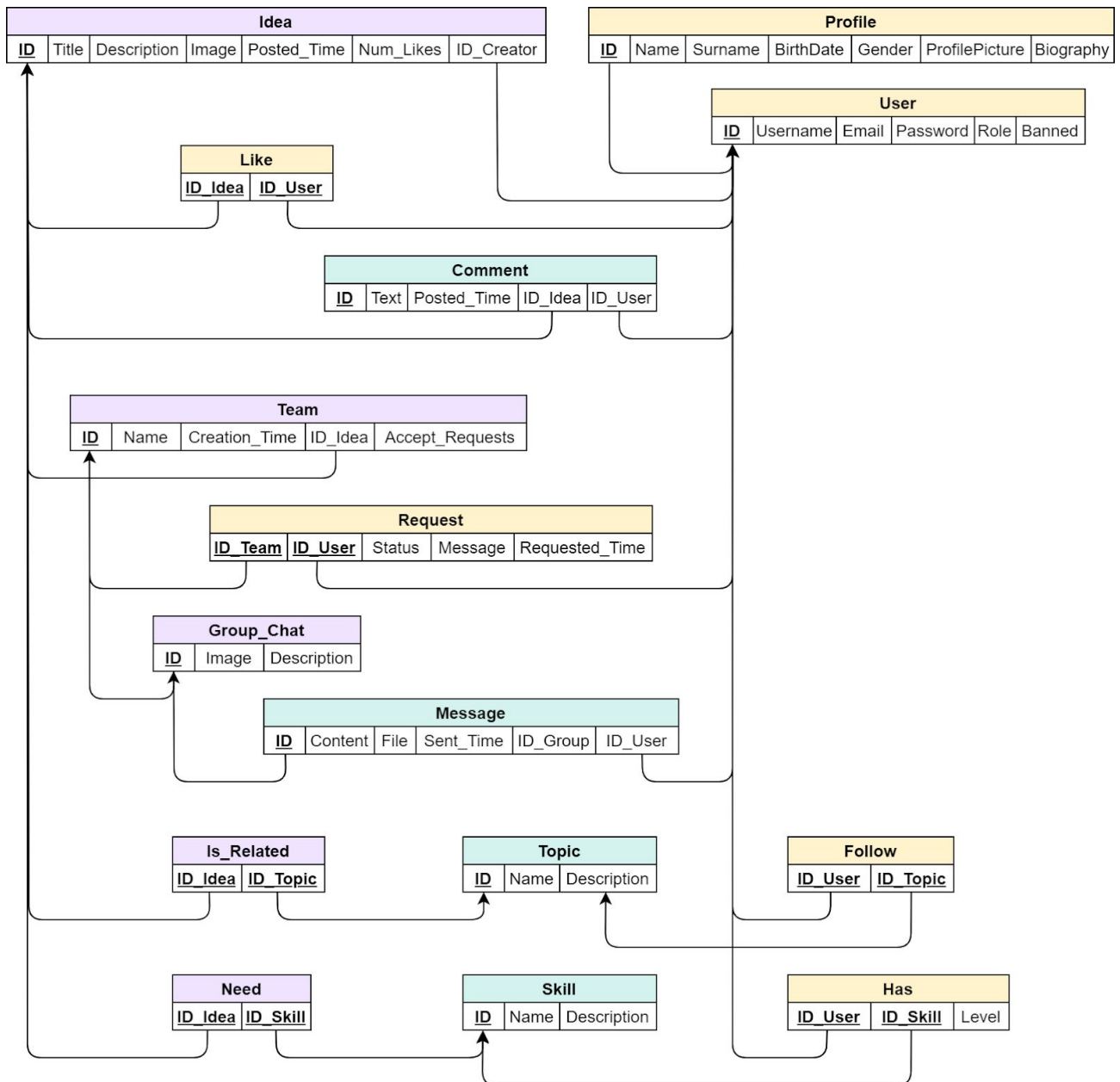


Figure 1: New Logical Schema

Physical Schema

In the following the SQL instructions to build the database in Figure 1 are reported.

```
-- Database creations
CREATE DATABASE HyperU_DB OWNER POSTGRES ENCODING='UTF8';

-- Connect to HyperU_DB db to create data for its 'public' schema
\c HyperU_DB
```

```

-- Create new data types
CREATE TYPE gendertype AS ENUM (
    'Not Declared',
    'Male',
    'Female'
);

CREATE TYPE usertype AS ENUM (
    'Common User',
    'Moderator',
    'Administrator'
);

CREATE TYPE statustype AS ENUM (
    'Pending',
    'Accepted'
);

-- Create username domain
CREATE DOMAIN username AS VARCHAR
CHECK (((VALUE)::text ~ '^[a-zA-Z0-9._-]*$'));

-- Create email domain
CREATE DOMAIN email AS VARCHAR
CHECK (((VALUE)::text ~
'^[a-zA-Z0-9.!#$%&'"+/=?^_`{|}~-]+@[a-zA-Z0-9]([a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?((?!\.)[a-zA-Z0-9]([a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)*$'));

-- Password has CHAR(32) because hash function return a string of 32 characters
-- Users
CREATE TABLE Users (
    ID SERIAL,
    Username USERNAME NOT NULL,
    Email EMAIL NOT NULL,
    Password CHAR(32),
    Role USERTYPE NOT NULL DEFAULT 'Common User',
    Banned BOOLEAN NOT NULL DEFAULT FALSE,

    PRIMARY KEY (ID),
    UNIQUE (Username)
);

-- Profile
CREATE TABLE Profile (
    ID SERIAL,
    Name VARCHAR NOT NULL,
    Surname VARCHAR NOT NULL,
    BirthDate DATE DEFAULT NULL,
    Gender GENDERTYPE DEFAULT 'Not Declared',
    Profile_Picture BYTEA DEFAULT NULL,
    Biography VARCHAR(700) DEFAULT '',

```

```

        PRIMARY KEY (ID),
        FOREIGN KEY (ID) REFERENCES Users(ID) ON UPDATE CASCADE ON DELETE CASCADE
    );

-- Skill
CREATE TABLE Skill (
    ID SERIAL,
    Name VARCHAR NOT NULL,
    Description VARCHAR NOT NULL,

    PRIMARY KEY (ID)
);

-- Topic
CREATE TABLE Topic (
    ID SERIAL,
    Name VARCHAR NOT NULL,
    Description VARCHAR NOT NULL,

    PRIMARY KEY (ID)
);

-- Idea
CREATE TABLE Idea (
    ID SERIAL,
    Title VARCHAR NOT NULL,
    Description VARCHAR NOT NULL,
    Image BYTEA DEFAULT NULL,
    Posted_Time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    Num_Likes INTEGER CHECK (Num_Likes >= 0) DEFAULT 0,
    ID_Creator SERIAL,

    PRIMARY KEY (ID),
    FOREIGN KEY (ID_Creator) REFERENCES Users(ID) ON UPDATE CASCADE ON DELETE CASCADE
);

-- Team
CREATE TABLE Team (
    ID SERIAL,
    Name VARCHAR NOT NULL,
    Creation_Time TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    ID_Idea SERIAL,
    Accept_Requests BOOLEAN NOT NULL DEFAULT True,

    PRIMARY KEY (ID),
    FOREIGN KEY (ID_Idea) REFERENCES Idea(ID) ON UPDATE CASCADE ON DELETE SET NULL,
    UNIQUE(ID_Idea)
);

-- Group Chat

```



```

CREATE TABLE Group_Chat (
    ID SERIAL,
    Image BYTEA DEFAULT NULL,
    Description VARCHAR DEFAULT NULL,

    PRIMARY KEY (ID),
    FOREIGN KEY (ID) REFERENCES Team(ID) ON UPDATE CASCADE ON DELETE CASCADE
);

-- Message
CREATE TABLE Message (
    ID SERIAL,
    Content VARCHAR DEFAULT NULL,
    File BYTEA DEFAULT NULL,
    Sent_Time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ID_Group SERIAL,
    ID_User SERIAL,

    CONSTRAINT MessageConstraint CHECK ((Content IS NULL AND File IS NOT NULL) OR (File IS
NULL AND Content IS NOT NULL)),

    PRIMARY KEY (ID),
    FOREIGN KEY (ID_Group) REFERENCES Group_Chat(ID) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (ID_User) REFERENCES Users(ID) ON UPDATE CASCADE ON DELETE CASCADE
);

-- Comment
CREATE TABLE Comment (
    ID SERIAL,
    Text VARCHAR NOT NULL,
    Sent_Time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    ID_Idea SERIAL,
    ID_User SERIAL,

    PRIMARY KEY (ID),
    FOREIGN KEY (ID_Idea) REFERENCES Idea(ID) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (ID_User) REFERENCES Users(ID) ON UPDATE CASCADE ON DELETE CASCADE
);

-- Need
CREATE TABLE Need (
    ID_Idea SERIAL,
    ID_Skill SERIAL,

    PRIMARY KEY (ID_Idea, ID_Skill),
    FOREIGN KEY (ID_Idea) REFERENCES Idea(ID) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (ID_Skill) REFERENCES Skill(ID) ON UPDATE CASCADE ON DELETE CASCADE
);

-- Is_Related
CREATE TABLE Is_Related (

```

```

ID_Idea SERIAL,
ID_Topic SERIAL,

PRIMARY KEY (ID_Idea, ID_Topic),
FOREIGN KEY (ID_Idea) REFERENCES Idea(ID) ON UPDATE CASCADE ON DELETE CASCADE,
FOREIGN KEY (ID_Topic) REFERENCES TOPIC(ID) ON UPDATE CASCADE ON DELETE CASCADE
);

-- Has
CREATE TABLE Has (
    ID_User SERIAL,
    ID_Skill SERIAL,
    Level INTEGER CHECK (Level BETWEEN 0 AND 5),

    PRIMARY KEY (ID_User, ID_Skill),
    FOREIGN KEY (ID_User) REFERENCES Users(ID) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (ID_Skill) REFERENCES Skill(ID) ON UPDATE CASCADE ON DELETE CASCADE
);

-- Follow
CREATE TABLE Follow (
    ID_User SERIAL,
    ID_Topic SERIAL,

    PRIMARY KEY (ID_User, ID_Topic),
    FOREIGN KEY (ID_User) REFERENCES Users(ID) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (ID_Topic) REFERENCES TOPIC(ID) ON UPDATE CASCADE ON DELETE CASCADE
);

-- Likes
CREATE TABLE Likes (
    ID_Idea SERIAL,
    ID_User SERIAL,

    PRIMARY KEY (ID_User, ID_Idea),
    FOREIGN KEY (ID_Idea) REFERENCES Idea(ID) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (ID_User) REFERENCES Users(ID) ON UPDATE CASCADE ON DELETE CASCADE
);

-- Request
CREATE TABLE Request (
    ID_Team SERIAL,
    ID_User SERIAL,
    Status STATUSTYPE NOT NULL DEFAULT 'Pending',
    Message VARCHAR(250) DEFAULT NULL,
    Requested_Time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    PRIMARY KEY (ID_User, ID_Team),
    FOREIGN KEY (ID_User) REFERENCES Users(ID) ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (ID_Team) REFERENCES Team(ID) ON UPDATE CASCADE ON DELETE CASCADE
);

```

Populate the Database: Example

In the following, there are some examples of SQL instructions to insert data in the created tables.

```
-- Users
INSERT INTO Users (Username, Email, Password, Role, Banned) VALUES
  ('FlamingoMark', 'marcoalecci98@gmail.com', 'd67d9fc418505ea8115c148646dc38d5',
  'Administrator', False),
  ('elia.zirolido', 'eliazir@gmail.com', '583cb78aaed75c23217df86262f9fb84',
  'Administrator', False),
  ('luca.martinelli.09', 'martinelliluca98@gmail.com',
  '4a902cb1bb06b427cec7b3ffecba6277', 'Administrator', False);

-- Profile
INSERT INTO Profile (Name, Surname, BirthDate, Gender) VALUES
  ('Marco', 'Alecci', '1998-11-26', 'Male'),
  ('Elia', 'Zirolido', '1998-08-21', 'Male'),
  ('Luca', 'Martinelli', '1998-09-24', 'Male');

-- Skill
INSERT INTO Skill (Name, Description) VALUES
  ('Java', 'Knowledge of the Java programming language'),
  ('SQL', 'Knowledge of the SQL language');

-- Topic
INSERT INTO Topic (Name, Description) VALUES
  ('Web Application', 'Computer programs that utilizes web browsers and web technology
to perform tasks over the Internet');

-- Idea
INSERT INTO Idea (Title, Description, Posted_Time, Num_Likes, ID_Creator) VALUES
  ('HyperU', 'An application to share ideas', '2020-10-05 13:00:00', 6, 1);

--Team
INSERT INTO Team (Name, Creation_Time, ID_Idea) VALUES
  ('HyperGroup', '2020-10-06 17:00:00', 1);

--Group Chat
INSERT INTO Group_Chat (ID, Description) VALUES
  (1, 'The chat of the HyperGroup!');

-- Message
INSERT INTO Message (Content, Sent_Time, ID_Group, ID_User) VALUES
  ('I think this is a good Idea ', '2020-10-07 15:00:00', 1, 3);

-- Comment
INSERT INTO Comment (Text, Sent_Time, ID_Idea, ID_User) VALUES
  ('It will be such a beautiful game!', '2020-10-05 18:15:36', 3, 2);
```

```

-- Need
INSERT INTO Need (ID_Idea, ID_Skill) VALUES (5, 12);

-- Is_Related
INSERT INTO Is_Related (ID_Idea, ID_Topic) VALUES (3, 15);

-- Has
INSERT INTO Has (Id_User, Id_Skill, Level) VALUES (3, 7, 4);

-- Follow
INSERT INTO Follow (Id_User, Id_Topic) VALUES (3, 10);

-- Likes
INSERT INTO Likes (Id_Idea, Id_User) VALUES (6, 9);

-- Request
INSERT INTO Request (Id_Team, Id_User, Status, Message, Requested_Time) VALUES
    (3, 8, 'Pending', 'Can I join?', '2019-06-12 14:24:00');

```

Principal Queries

In this section we report the queries needed to perform some operations on the database:

1. Show the group chat of a specific Group with the message in chronological order
2. Show the home page of a specific user, displaying all the ideas that are related to a topic followed by that user. The ideas are displayed in chronological order. The ideas posted by banned users are excluded.
3. Show the profile of a specific user, displaying all his information and also the number of ideas posted, and the total number of likes received.
4. Show the ideas with an above average number of likes (excluding ideas posted by banned users)

```

-- QUERY 1: GROUP CHAT (messages and username in a group chat)
SELECT
    Message.Sent_Time,
    Message.Content,
    Message.File,
    Users.Username
FROM
    Message
    INNER JOIN Users ON Message.ID_User = Users.ID
WHERE
    Message.ID_Group = 1
ORDER BY
    Message.Sent_Time ASC
LIMIT
    30;

```

	sent_time timestamp without time zone	content character varying	file bytea	username character varying
1	2020-10-07 15:00:00	I think this is a good Id...	[null]	luca.martinelli.09
2	2020-10-07 15:12:00	Yes, for me too	[null]	elia.zioldo
3	2020-10-07 15:16:00	I am so glad you liked it!	[null]	FlamingoMark
4	2020-10-07 15:17:00	We also need to create...	[null]	FlamingoMark
5	2020-10-07 15:21:00	Ok, no problem I follow...	[null]	elia.zioldo
6	2020-10-07 15:24:00	Nice!	[null]	FlamingoMark
7	2020-10-07 15:28:00	Maybe, I can take care ...	[null]	luca.martinelli.09
8	2020-10-07 15:31:00	Yeah, sure!	[null]	FlamingoMark

Figure 2: Result of the first query

```
-- QUERY 2: HOME PAGE (Ideas with topics followed by the user, excluding ideas posted by
banned users)
-- Subquery 1: retrieve ideas
SELECT
    Idea.*
FROM
    Idea
    INNER JOIN Is_Related ON Idea.ID = Is_Related.ID_Idea
    INNER JOIN Follow ON Is_Related.ID_Topic = Follow.ID_Topic
    INNER JOIN Users ON Follow.ID_User = Users.ID
WHERE
    Users.ID = 1;

-- Subquery 2: retrieve information about the creators (check if user is banned)
SELECT
    DISTINCT Users.ID,
    Users.Username
FROM
    Idea
    INNER JOIN Users ON Idea.ID_Creator = Users.ID
WHERE
    Users.Banned = False;

-- Full query
SELECT
    Ideas.*,
    Creator.Username
FROM
    (
        SELECT
            Idea.*
        FROM
```

```

    Idea
    INNER JOIN Is_Related ON Idea.ID = Is_Related.ID_Idea
    INNER JOIN Follow ON Is_Related.ID_Topic = Follow.ID_Topic
    INNER JOIN Users ON Follow.ID_User = Users.ID
WHERE
    Users.ID = 1
ORDER BY
    Idea.Posted_Time
LIMIT
    30
) AS Ideas
INNER JOIN (
    SELECT
        DISTINCT Users.ID,
        Users.Username
    FROM
        Idea
        INNER JOIN Users ON Idea.ID_Creator = Users.ID
    WHERE
        Users.Banned = False
) AS Creator ON Ideas.ID_Creator = Creator.ID;

```

--Query 2: Alternative method without using subqueries

```

SELECT
    Idea.ID,
    Idea.Title,
    Idea.Description,
    Idea.Image,
    Idea.Posted_Time,
    Idea.Num_Likes,
    Creator.Username
FROM
    Idea
    INNER JOIN Is_Related ON Idea.ID = Is_Related.ID_Idea
    INNER JOIN Follow ON Is_Related.ID_Topic = Follow.ID_Topic
    INNER JOIN Users ON Follow.ID_User = Users.ID
    INNER JOIN Users AS Creator ON Idea.ID_Creator = Creator.ID
WHERE
    Users.ID = 1
    AND Creator.Banned = False
ORDER BY
    Idea.Posted_Time
LIMIT
    30;

```

	id integer	title character varying	description character varying	image bytea	posted_time timestamp without time zone	num_likes integer	username character varying
1	7	OnePlus	A new mobile phone high tech startup	[null]	2020-03-23 21:23:00	0	FilippoolV
2	1	HyperU	An application to share ideas	[null]	2020-10-05 13:00:00	6	FlamingoMark

Figure 3: Result of the second query

```

-- QUERY 3: USER PROFILE (infos, total likes, ideas posted)
SELECT
    Users.ID,
    Users.Username,
    Profile.Name,
    Profile.Surname,
    Profile.Gender,
    Profile.Birthdate,
    Profile.Profile_Picture,
    Profile.Biography,
    COUNT(Users.ID) AS Num_Ideas,
    SUM(Idea.Num_Likes) AS Total_Likes
FROM
    Users
    INNER JOIN Profile ON Users.ID = Profile.ID
    INNER JOIN Idea ON Idea.ID_Creator = Users.ID
WHERE
    Users.ID = 6
GROUP BY
    Users.ID,
    Profile.ID;

```

	id	username	name	surname	gender	birthdate	profile_picture	biography	num_ideas	total_likes
	integer	character varying	character varying	character varying	gendertype	date	bytea	character varying (700)	bigint	bigint
1	6	FilippoolV	Filippo	Mattena	Male	1995-09-13	[null]		3	4

Figure 4: Result of the third query

```

-- QUERY 4: IDEAS (info and username) with an above average number of likes (excluding
ideas posted by banned users)
SELECT
    Users.Username,
    Idea.Title,
    Idea.Description,
    Idea.Image,
    Idea.Num_Likes,
    Idea.Posted_Time
FROM
    Idea
    INNER JOIN Users ON Users.ID = Idea.ID_Creator
WHERE
    Users.Banned = False
    AND Idea.Num_Likes >= (
        SELECT
            AVG(Num_Likes)
        FROM
            Idea
            INNER JOIN Users ON Users.ID = Idea.ID_Creator
        WHERE

```

```

        Users.Banned = False
    );

```

	username character varying	title character varying	description character varying	image bytea	num_likes integer	posted_time timestamp without time zone
1	FlamingoMark	HyperU	An application to share...	[null]	6	2020-10-05 13:00:00
2	FilippoolV	SuperCharge	A new project for a uni...	[null]	4	2019-06-08 18:45:00

Figure 5: Result of the fourth query

JDBC Implementations of the Principal Queries and Visualization

Hereafter, we report a java class which reads the list of the first three users and prints the ideas that have topics followed by each of them. The result is shown on the terminal output.

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class HyperU {

    // Declaration of constants for accessing the DBMS
    private static final String DRIVER = "org.postgresql.Driver";
    private static final String DATABASE = "jdbc:postgresql://localhost/HyperU_DB";
    private static final String USER = "postgres";
    private static final String PASSWORD = "admin";

    public static void main(String[] args) {
        // JDBC objects to access the DBMS
        Connection con = null;
        Statement stmt = null;
        ResultSet rsUsers = null;
        ResultSet rsIdeas = null;
        long start;
        long end;

        // Registration of JDBC driver to be used
        try {
            Class.forName(DRIVER);
            System.out.printf("Driver %s successfully registered.%n", DRIVER);
        } catch (ClassNotFoundException e) {
            System.out.printf("Driver %s not found: %s.%n", DRIVER, e.getMessage());
            System.exit(-1);
        }

        try {
            // Connection to the database
            start = System.currentTimeMillis();
            con = DriverManager.getConnection(DATABASE, USER, PASSWORD);
            end = System.currentTimeMillis();
            System.out.printf("Connection to database %s successfully established in %d milliseconds.%n", DATABASE,

```



```

        end - start);

// Creation of the statement
start = System.currentTimeMillis();
stmt = con.createStatement();
end = System.currentTimeMillis();
System.out.printf("Statement successfully created in %d milliseconds.%n", end - start);

// QUERY: Get first three users
String sql = "SELECT ID, Username, Email FROM Users LIMIT 3";
start = System.currentTimeMillis();
rsUsers = stmt.executeQuery(sql);

System.out.println("\nHome Pages for users:");

String userID;
String username;
String email;

// For each user get and print his Home Page
while (rsUsers.next()) {
    // get user information
    userID = rsUsers.getString("ID");
    username = rsUsers.getString("Username");
    email = rsUsers.getString("Email");

    if (!rsUsers.isFirst()) System.out.println("\n");
    System.out.printf("User %s - Username: %s, Email: %s%n%n", userID, username, email);

    // QUERY: Get Ideas in the Home Page (Ideas with a topic the user follows)
    sql = "SELECT Idea.ID, Idea.Title, Idea.Description, Idea.Posted_Time,
Idea.Num_Likes, Creator.Username FROM Idea INNER JOIN Is_Related ON Idea.ID = Is_Related.ID_Idea
INNER JOIN Follow ON Is_Related.ID_Topic = Follow.ID_Topic INNER JOIN Users ON Follow.ID_User =
Users.ID INNER JOIN Users AS Creator ON Idea.ID_Creator = Creator.ID WHERE Users.ID = " + userID
+ " AND Creator.Banned = False ORDER BY Idea.Posted_Time";

    rsIdeas = stmt.executeQuery(sql);
    String IdeaID;
    String Title;
    String Description;
    String PostedTime;
    String NumLikes;
    String CreatorUsername;

    if (!rsIdeas.isBeforeFirst()) { // if there are no ideas
        System.out.printf("Empty Home Page for user %s%n", username);
    } else {
        // header
        System.out.printf("    ID\tPOSTED TIME\t\tNUM LIKES\tCREATOR\t\tTITLE\t\t
DESCRIPTION%n");

        while (rsIdeas.next()) { // if there is at least one idea
            IdeaID = rsIdeas.getString("ID");
            Title = rsIdeas.getString("Title");
            Description = rsIdeas.getString("Description");
            PostedTime = rsIdeas.getString("Posted_Time");
            NumLikes = rsIdeas.getString("Num_Likes");

```

```

        CreatorUsername = rsIdeas.getString("Username");

        // print idea
        System.out.printf("    %s\t%s\t%s\t%s\t%s\t%s\t    %s\n", IdeaID, PostedTime,
NumLikes,
                                CreatorUsername, Title, Description);
    }
}
rsIdeas.close(); // release results for Ideas
}
rsUsers.close(); // release results for Users
stmt.close(); // release statement
con.close(); // release connection

end = System.currentTimeMillis();
System.out.printf("\nData correctly extracted and visualized in %d milliseconds.\n", end
- start);
} catch (SQLException e) {
    System.out.printf("Database access error:\n");

    // cycle in the exception chain
    while (e != null) {
        System.out.printf("- Message: %s\n", e.getMessage());
        System.out.printf("- SQL status code: %s\n", e.getSQLState());
        System.out.printf("- SQL error code: %s\n", e.getErrorCode());
        System.out.printf("\n");
        e = e.getNextException();
    }
} finally {
    try {
        // close the used resources
        if ((rsUsers != null && rsIdeas != null) && (!rsUsers.isClosed() ||
!rsIdeas.isClosed())) {
            start = System.currentTimeMillis();
            rsUsers.close();
            rsIdeas.close();
            end = System.currentTimeMillis();

            System.out.printf("Result set successfully closed in finally block in %,d
milliseconds.\n", end - start);
        }
        if (stmt != null && !stmt.isClosed()) {
            start = System.currentTimeMillis();
            stmt.close();
            end = System.currentTimeMillis();

            System.out.printf("Statement successfully closed in finally block in %,d
milliseconds.\n", end - start);
        }
        if (con != null && !con.isClosed()) {
            start = System.currentTimeMillis();
            con.close();
            end = System.currentTimeMillis();

            System.out.printf("Connection successfully closed in finally block in %,d
milliseconds.\n", end - start);
        }
    }
}

```

```

    } catch (SQLException e) {
        System.out.printf("Error while releasing resources in finally block:%n");

        // cycle in the exception chain
        while (e != null) {
            System.out.printf("- Message: %s%n", e.getMessage());
            System.out.printf("- SQL status code: %s%n", e.getSQLState());
            System.out.printf("- SQL error code: %s%n", e.getErrorCode());
            System.out.printf("%n");
            e = e.getNextException();
        }
    } finally {
        // release resources to the garbage collector
        rsUsers = null;
        stmt = null;
        con = null;

        System.out.printf("Resources released to the garbage collector.%n");
    }
}

System.out.printf("Program end.%n");
}
}

```

Driver org.postgresql.Driver successfully registered.
 Connection to database jdbc:postgresql://localhost/HyperU_DB successfully established in 122 milliseconds.
 Statement successfully created in 4 milliseconds.

Home Pages for users:

User 1 - Username: FlamingoMark, Email: marcoalecci98@gmail.com

ID	POSTED TIME	NUM LIKES	CREATOR	TITLE	DESCRIPTION
7	2020-03-23 21:23:00	0	FilippooIV	OnePlus	A new mobile phone high tech startup
1	2020-10-05 13:00:00	6	FlamingoMark	HyperU	An application to share ideas

User 2 - Username: elia.zirollo, Email: eliazir@gmail.com

Empty Home Page for user elia.zirollo

User 3 - Username: luca.martinelli.09, Email: martinelliluca98@gmail.com

ID	POSTED TIME	NUM LIKES	USERNAME	TITLE	DESCRIPTION
7	2020-03-23 21:23:00	0	FilippooIV	OnePlus	A new mobile phone high tech startup

Data correctly extracted and visualized in 41 milliseconds.
 Resources released to the garbage collector.
 Program end.