Progetto di Basi di Dati

(Social Network)

ANNO ACCADEMICO 2017-2018

Sciacco Mariano - 1142498
Buratto Enrico - 1142644

Abstract

qbreak è un servizio di Social Networking, nato originariamente dall'idea di due studenti di informatica di condividere codici e programmi tra gli studenti padovani. Successivamente si è evoluto e definito come social network specifico per programmatori ed esperti informatici; al momento il sito conta più di 10.000 iscritti da tutto il mondo.
Evolvendosi da semplice sito accessibile a pochi eletti, ***qbreak*** si è adattato a tutti gli standard degli odierni social network: negli ultimi anni sono state introdotte svariate funzionalità, tra le quali la creazione di account specializzata in pagine e utenti e la possibilità di seguire pagine o utenti specifici e di avviare discussioni private (chat) con essi. Oltre a questo, il sito ora permette anche di scrivere post pubblici e di commentarne a propria volta, di caricare foto, di votare positivamente o negativamente i post degli altri utenti o delle altre pagine, andando a formare così un meccanismo di classifica all'interno del network, e di creare un proprio portfolio con tutti i progetti a cui si sta lavorando (o si ha lavorato in passato), visibile agli altri utenti. Queste ultime due funzionalità, inoltre, si sono rivelate particolarmente utili agli utenti per farsi notare dalle aziende informatiche, le quali hanno trovato il sito un utile strumento per l'assunzione di personale altamente qualificato e in grado di darne prova.
Inizialmente, il progetto non possedeva una base di dati molto avanzata, poiché gli iscritti erano relativamente pochi, e pertanto la gestione dei dati degli utenti e dei codici condivisi era affidata a un database molto scarno ed essenziale, sebbene fosse comunque veloce e richiedesse poche risorse. Tuttavia, l'aumento inaspettato degli utenti ha reso necessaria l'estensione dell'attuale struttura della base di dati così che da un lato si potesse ampliare il sito con nuove funzionalità, migliorando la gestione e il mantenimento dei dati attuali, e dall'altro si potesse tenere traccia di tutti i movimenti interni al sito, garantendo così molta più sicurezza ed efficienza nelle operazioni.

Analisi dei requisiti

Introduzione

Si vuole realizzare una base di dati per organizzare le informazioni di utenti, pagine e attività svolte per un Social Network incentrato sull'informatica.
Al momento della creazione di un nuovo account, l'utente dovrà scegliere se registrarsi come semplice utente (nel caso di una persona reale) o come pagina (nel caso si tratti di un'azienda o di un ente).
Al momento dell'iscrizione, verrà chiesto di fornire un indirizzo eMail e di scegliere uno username e una password; colui che si sta iscrivendo potrà scegliere poi se inserire o meno:

- Nel caso si tratti di un utente:
  - Il suo nome e il suo cognome;
  - La sua data di nascita;
  - La formazione acquisita;
  - Le abilità particolari;
  - Una foto del profilo e una foto di copertina.
- Nel caso si tratti di una pagina:
  - Il titolo e una descrizione;
  - Una foto del profilo e una foto di copertina.

La base di dati inoltre dovrà registrare la data di iscrizione e creare un ID univoco per l'account in questione.
Un utente o una pagina avrà poi la possibilità di cancellare la propria iscrizione: in questo caso la base di dati dovrà permetterlo cancellandolo alla vista degli utenti, mantenendo però in memoria i dati.
Dopo l'iscrizione, ogni account potrà:

- Creare dei post caratterizzati da tags;
- Commentare e/o votare positivamente o negativamente altri post;
- Caricare delle foto all'interno dei post;
- Seguire (follow) un altro utente o un'altra pagina. In questo caso, l'account che segue si chiama "follower", quello che invece viene seguito "following";
- Creare dei progetti (software o siti web);
- Scambiare messaggi privati con altri utenti o pagine (chat).

I post sono caratterizzati da tags: essi servono a relazionare in una o più tematiche vari post, come se fossero ordinati in categorie. I tags vengono inseriti dall'autore stesso del post.
Ciascun autore di un post riceverà una notifica per ogni voto ricevuto e per ogni commento inserito nel post.
Il social network fornisce inoltre la possibilità di gestione di un portfolio, che è l'insieme di tutti i progetti realizzati o in sviluppo da parte di un utente o pagina.
Ciascun progetto, che in genere è identificato come software, presenta un titolo, una descrizione, una versione corrente ed un URL di riferimento (es: repository da cui poterlo scaricare, sito web, ecc.).
Oltre alle funzioni disponibili a ogni account, il sito vuole tenere traccia di tutte le operazioni svolte al suo interno; per fare ciò, la base di dati deve registrare tutte le informazioni all'interno di un log. Le informazioni che vengono registrate sono:

- L'ID univoco di chi ha svolto l'operazione;
- L'indirizzo IP dal quale viene svolta;
- La data e l'ora;
- La descrizione dell'evento.

Glossario dei termini

Termine	Descrizione	Collegamento
Account	Utente che si iscrive e si collega al social network. Può essere di due tipologie: Persona reale, con nome, cognome e data di nascita, o Pagina, che può rappresentare una azienda, entità o relazione.	Foto, Post, Follower, Following, Voto, Progetto, Chat, Logs, Notifica
Foto	Immagine che può essere relativa a un post, una foto profilo o ad una copertina dell'account.	Account (foto profilo, copertina), Post
Post	Messaggio inserito da parte di un utente identificato tramite account. Può contenere foto e tags.	Account, Foto, Commento, Tags, Voto
Commento	Messaggio di risposta ad un post.	Account, Post
Voto	Può essere di due tipologie: positivo (+1) o negativo (-1). E' rilasciato dagli utenti in relazione ad un post.	Account, Post
Follower	Account che segue un altro account per poter visualizzare i suoi post in rilievo.	Account
Following	Account che viene seguito da parte di un altro account.	Account
Progetto	Software / Sito Web realizzato o in sviluppo da parte di un account.	Account, Portfolio
Portfolio	Insieme di tutti i progetti realizzati o in sviluppo da parte di un account.	Account, Progetto
Chat	Messaggi privati inviati tra due account.	Account
Logs	Informazione sui movimenti e sulle azioni significative intraprese da parte di account all'interno del social network. **Vengono escluse:** messaggi di chat, votazioni post, following.	Account
Tag	Parola che identifica un tema o una categoria di un post.	Post
Notifica	Informazioni inviata all'autore di un post che riceve nuovi voti, menzioni o commenti	Commento, Voto

Operazioni previste sulla base di dati

Operazione	**Tipo**	**Frequenza**
1. Iscrizione di un nuovo utente	I	50 volte al giorno
2. Accesso di un utente	I	2500 volte al giorno
3. Inserimento post da parte di un account	I	300 volte al giorno
4. Inserimento commento in un post da parte di un account	I	800 volte al giorno
5. Votazione di un post da parte di un account	I	1850 volte al giorno
6. Creazione di progetti nel proprio portfolio	I	60 volte al giorno
7. Aggiunta di una foto nel proprio profilo	I	150 volte al giorno
8. Eliminazione del proprio account	I	3 volte al giorno
9. Eliminazione di un post da parte di un account	I	20 volte al giorno
10. Eliminazione di un commento da parte di un account	I	50 volte al giorno
11. Following account	I	500 volte al giorno

12. Invio messaggio privato di chat	I	35.000 volte al giorno
13. Inserimento Log per ogni azione account	B	4068 volte al giorno
14. Visualizzazione logs da parte di un admin	I	250 volte al giorno
15. Modifica informazioni account	I	50 volte al giorno
16. Modifica contenuto post	I	75 volte al giorno
17. Eliminazione foto da parte di un account	I	10 volte al giorno
18. Aggiornamento n° messaggi (post + commenti) account	B	1100 volte al giorno
19. Aggiornamento n° voti di un post	B	1850 volte al giorno
20. Visualizzazione di un post da parte di un account	I	(30 post / utente) 75.000 volte al giorno
21. Visualizzazione di un profilo da parte di un account	I	(5 profili / utente) 12.500 volte al giorno
22. Aggiornamento n° voti ottenuti da un account	B	1850 volte al giorno
23. Aggiunta tags ad un post, durante il suo inserimento	I	180 volte al giorno
24. Invio di una notifica	B	2150 volte al giorno

Strutturazione dei requisiti

FRASI RELATIVE AD *ACCOUNT*

Al momento della creazione di un nuovo **account**, l'utente dovrà scegliere se registrarsi come semplice utente o come pagina…
[...] Dopo l'iscrizione, ogni account potrà:

- Creare dei post caratterizzati da tags;
- Commentare e/o votare positivamente o negativamente altri post;
- Caricare delle foto all'interno dei post;
- Seguire (follow) un altro utente o un'altra pagina. In questo caso, l'account che segue si chiama "follower", quello che invece viene seguito "following";
- Creare dei progetti (software o siti web);
- Scambiare messaggi privati con altri utenti o pagine (chat).

La base di dati inoltre dovrà registrare la data di iscrizione e creare un ID univoco per l'account in questione.
Un utente o una pagina, infine, avrà la possibilità di cancellare la propria iscrizione: in questo caso la base di dati dovrà permetterlo.

FRASI RELATIVE A *FOTO*

Nel caso si tratti di un utente (o di una pagina):

- [...]
- Una **foto del profilo e una foto di copertina**.

Dopo l'iscrizione, ogni account potrà:

- […]
- Caricare delle **foto all'interno dei post**;

FRASI RELATIVE A *POST*

Dopo l'iscrizione, ogni account potrà:

- Creare dei **post** caratterizzati da tags;
- Caricare delle foto all'interno dei **post**;
- Commentare e/o votare positivamente o negativamente altri **post**;

FRASI RELATIVE A *COMMENTO*

Dopo l'iscrizione, ogni account potrà:

- [...]
- **Commentare** e/o votare positivamente o negativamente altri post;

FRASI RELATIVE A *VOTO*

Dopo l'iscrizione, ogni account potrà:

- [...]
- Commentare e/o **votare** positivamente o negativamente altri post;

I voti di tutti i post pubblicati da un account vanno a formare un meccanismo di ranking (classifica), il quale si basa su una semplice differenza tra il numero di voti positivi (+1) e il numero di messaggi e voti negativi (-1). Il valore risultante è quello che identifica la posizione dell'utente nella classifica globale.

FRASI RELATIVE A *FOLLOWER / FOLLOWING*

[...] Seguire (follow) un altro utente o un'altra pagina. In questo caso, l'account che segue si chiama "**follower**", quello che invece viene seguito "**following**"; [...]

FRASI RELATIVE A *PROGETTO*

Dopo l'iscrizione, ogni account potrà:

- [...]
- Creare dei **progetti** (software o siti web);

Ciascun progetto, che in genere è identificato come software, presenta un titolo, una descrizione, una versione corrente ed un URL di riferimento (es: repository da cui poterlo scaricare, sito web, ecc.).

FRASI RELATIVE A *PORTFOLIO*

Il social network fornisce inoltre la possibilità di gestione di un **portfolio**, che è l'insieme di tutti i progetti realizzati o in sviluppo da parte di un utente o pagina.

FRASI RELATIVE A *CHAT*

Dopo l'iscrizione, account potrà:

- [...]
- Scambiare messaggi privati con altri utenti o pagine (chat).

FRASI RELATIVE A *LOGS*

Oltre alle funzioni disponibili a ogni account, il sito vuole tenere traccia di tutte le operazioni svolte al suo interno; per fare ciò, la base di dati deve registrare tutte le informazioni all'interno di un log. Le informazioni che vengono registrate sono:

- L'ID univoco di chi ha svolto l'operazione;
- L'indirizzo IP dal quale viene svolta;
- La data e l'ora;
- La descrizione dell'evento.

FRASI RELATIVE A *TAGS*

I post sono caratterizzati da tags: essi servono a relazionare in una o più tematiche vari post, come se fossero ordinati in categorie. I tags vengono inseriti dall'autore stesso del post.

FRASI RELATIVE A *NOTIFICA*

Ciascun autore di un post riceverà una notifica per ogni voto ricevuto e per ogni commento inserito nel post.

Progettazione concettuale

Entità della base di dati

Entità	Descrizione	Attributi
account	Un account può essere un utente o una pagina, e indica un utente o un'azienda che si iscrive al social network.	accountID {PK} DataReg, Username, Email, Password, NumMessaggi, NumVoti
utente	Entità figlia esclusiva di account. Un utente può essere un utente semplice o un amministratore; può essere inoltre attivo o cancellato.	Nome, Cognome, DataNascita, Formazione,Abilità
utente_admin	Entità figlia opzionale di utente; indica un utente con privilegi di amministrazione all'interno del sito.	*Nessun attributo*
utente_attivo		*Nessun attributo*
utente_cancellato	Entità per mantenere in memoria i dati di un utente che ha deciso di cancellarsi.	*Nessun attributo*
pagina	Entità figlia esclusiva di account. Anch'essa può essere attiva o cancellata.	Titolo Descrizione
pagina_attiva		*Nessun attributo*
pagina_cancellata	Entità per mantenere in memoria i dati di una pagina che ha deciso di cancellarsi.	*Nessun attributo*
foto	Immagini che vengono caricate sul sito; esse possono essere immagini del profilo (foto_profilo), della copertina (foto_copertina), di un post (foto_post) oppure non appartenenti a nessuna di queste categorie.	fotoID {PK} accountID, Data, Titolo, File
foto_attiva		*Nessun attributo*
foto_cancellata	Entità per mantenere in memoria una foto che viene cancellata dal sito.	*Nessun attributo*
foto_profilo	Entità figlia che identifica una foto del profilo.	*Nessun attributo*
foto_copertina	Entità figlia che identifica una foto di copertina.	*Nessun attributo*
foto_post	Entità figlia che identifica una foto di un post.	*Nessun attributo*
post	Post che vengono scritti sul sito, memorizzando anche chi ha scritto un determinato post e il numero dei voti che ha ricevuto.	postID {PK} accountID {FK}, Data, Titolo, Contenuto, NumVoti
post_cancellato	Entità per mantenere in memoria un post che viene cancellato dal sito.	*Nessun attributo*
post_attivo		*Nessun attributo*
commento	Commenti che vengono fatti ai post, identificati da un ID. Memorizza anche chi ha scritto un commento sotto un determinato post.	commentID {PK} accountID {FK}, postID {FK}, DataInserimento, Contenuto
commento_cancellato	Entità per mantenere in memoria un commento che viene cancellato da un post sul sito.	*Nessun attributo*

commento_attivo		*Nessun attributo*
voto	Voti dati a un determinato post, memorizzando il post e chi lo ha commentato.	postID {PK} Voto, accountID {FK}
progetto	Progetti o lavori realizzati da un account.	(Titolo + accountID {Fk}) {FK} DataInserimento, DataUltimaModifica, VersioneAttuale, Descrizione, Url, accountID {Fk}
portfolio	Contiene tutti i progetti di un determinato account.	portfolioID {PK}, accountID
messaggio	Messaggi tra due utenti; l'entità è identificata dagli account che comunicano e dalla data e ora dei singoli messaggi.	Data {PK}, Contenuto, accountID {FK}
log	Contiene tutte le operazioni che avvengono sul sito, memorizzando anche chi ha svolto una determinata azione, da che indirizzo e la data.	logID {PK}, accountID, IP,DataOra, Descrizione
azione_admin	Entità figlia che, insieme ad azione_utente, determina le operazioni che un utente può fare a seconda che sia un utente amministratore o semplice.	*Nessun attributo*
azione_utente		*Nessun attributo*

Approfondimenti

Generalizzazioni

- Account è generalizzazione completa di utente e pagina;
- Utente è generalizzazione parziale di utente_admin;
- Utente è anche generalizzazione completa di utente_cancellato e utente_attivo;
- Pagina è generalizzazione completa di pagina_cancellata e pagina_attiva;
- Post è generalizzazione completa di post_cancellato e post_attivo;
- Commento è generalizzazione completa di commento_cancellato e commento_attivo;
- Foto è generalizzazione completa di foto_cancellata e foto_attiva;
- Foto è generalizzazione parziale di foto_profilo, foto_copertina e foto_post;
- Log è generalizzazione completa di azione_admin e azione_utente.

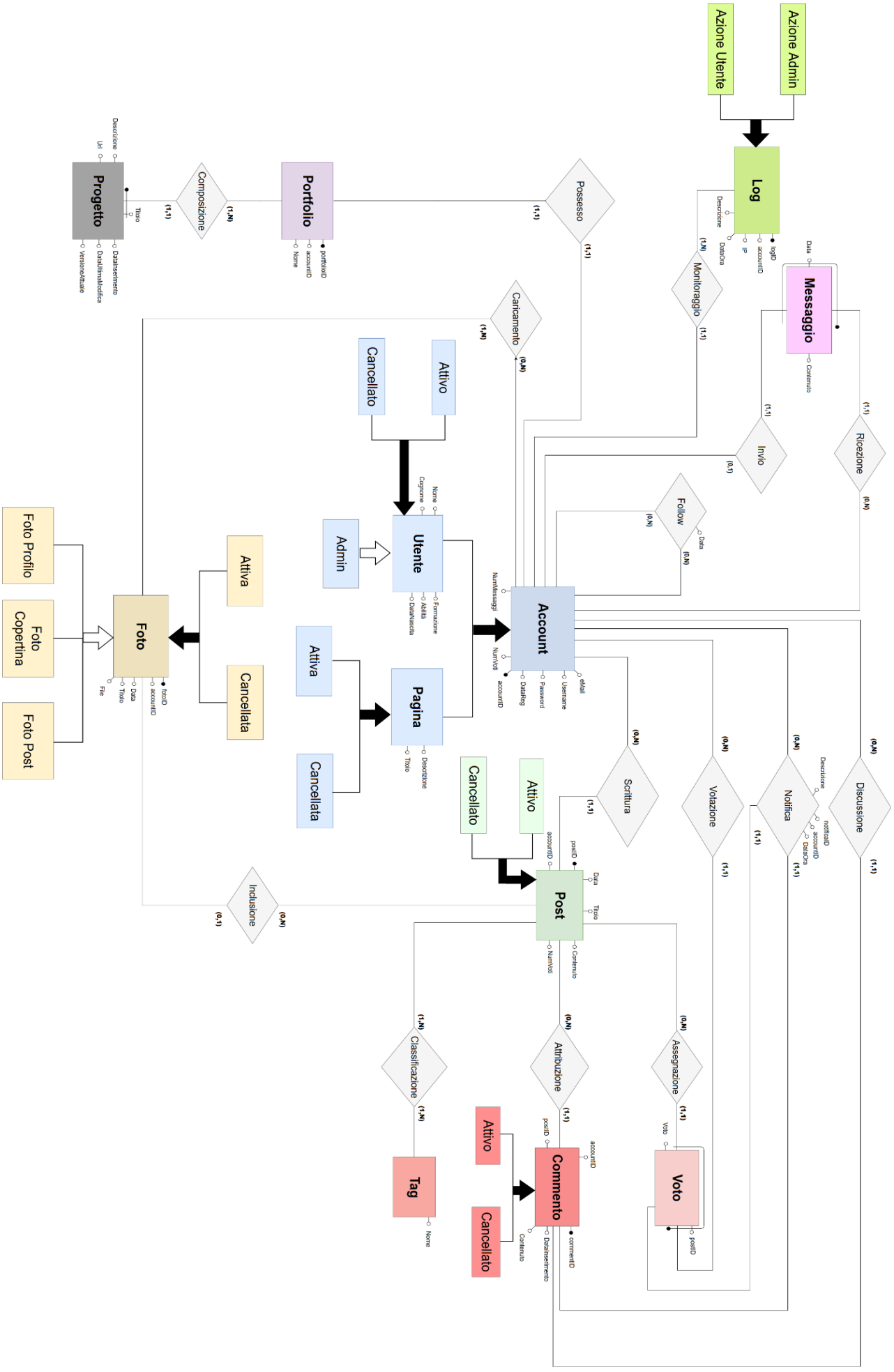
Entità principale	Entità figlie	Tipo di generalizzazione	Descrizione
Account	utente pagina	Completa	Un account è un utente oppure una pagina.
Utente	utente_admin	Parziale	Un utente può essere un amministratore.
Utente	utente_cancellato utente_attivo	Completa	Un utente è attivo oppure è cancellato.
Pagina	pagina_cancellata pagina_attiva	Completa	Una pagina è attiva oppure è cancellata.
Post	post_cancellato post_attivo	Completa	Un post è attivo oppure è cancellato.
Commento	commento_cancellato commento_attivo	Completa	Un commento è attivo oppure è cancellato.
Foto	foto_cancellata foto_attiva	Completa	Una foto è cancellata oppure è attiva.

Foto	foto_profilo foto_copertina foto_post	Parziale	Una foto può essere una foto profilo, una foto di copertina o una foto in un post; tuttavia, possono esserci foto che non appartengono a nessna di queste categorie.
Log	azione_admin azione_utente	Completa	Utenti amministratori e semplici hanno accessi diversi alle logs.

Tabella delle relazioni

Relazione	Descrizione	Entità coinvolte	Attributi
Invio	Associa un messaggio di chat inviato da un account ad un altro account.	account (0,1) messaggio (1,1)	*Nessun attributo*
Ricezione	Associa un messaggio di chat ricevuto da un account al proprio account.	account (0,N) messaggio (1,1)	*Nessun attributo*
Possesso	Relazione che associa il portfolio all'account.	account (1,1) portfolio (1,1)	*Nessun attributo*
Composizione	Relazione che associa più progetti ad un portfolio	portfolio (1,N) progetti (1,1)	*Nessun attributo*
Caricamento	Relazione che associa il caricamento di una foto ad un account.	account (0,N) foto (1,N)	*Nessun attributo*
Inclusione	Relazione che include una o più foto in un post.	foto (0,1) post (0,N)	*Nessun attributo*
Scrittura	Associa un post ad un account che ne diventa l'autore.	account (0,N) post (1,1)	*Nessun attributo*
Votazione	Associa il voto di un account ad un post.	account (0,N) voto (1,1)	*Nessun attributo*
Discussione	Associa un commento ad un account.	account (0,N) commento (1,1)	*Nessun attributo*
Assegnazione	Relazione che associa voti ad un post.	voto(1,1) post(0,N)	*Nessun attributo*
Attribuzione	Associa ad un post dei commenti.	commento (1,1) post (0,N)	*Nessun attributo*
Classificazione	Associa dei tag ad un post.	tag (1,N) post (1,N)	*Nessun attributo*
Follow	Relazione che associa due account, non in maniera reciproca.	account (0,N) account (0,N)	*Data* [timestamp]
Monitoraggio	Relazione che associa le azioni di un account a delle logs.	account (1,1) log (1,N)	*Nessun attributo*
Notifica	Relazione che notifica all'acount quando un suo post viene commentato o votato	voto (1,1) commento (1,1) account (0,N)	*notificaID* [INT] *accountID* [INT] *DataOra* [timestamp] *Descrizione* [TEXT]

Diagramma E/R



Progettazione logica

Analisi delle ridondanze

Nel database sono state riscontrate 4 ridondanze:

1. **Ridondanza - Attributo NumVoti in Account**
   L'attributo *NumVoti* sta per il numero totale di voti ricevuti dall'account attraverso i post inseriti.
   Questo attributo, combinato con il *NumMessaggi*, viene utilizzato per determinare la classifica dell'account (Rank) che è dato dalla seguente formula *((NumVoti+NumMessaggi) / 10)*.
   La ridondanza si presenza dal momento che il numero di voti è facilmente ricavabile alla visualizzazione del profilo dell'account. Si procederà pertanto ad analizzare la tabella dei volumi con le relative operazioni.

2. **Ridondanza - Attributo NumMessaggi in Account**
   L'attributo *NumMessaggi* riguarda il numero totale di Messaggi inseriti da parte di un account (come totale di *Post* e *Commenti*).
   Esso viene utilizzato in combinazione di *NumVoti* per calcolare la classifica (Rank) dell'account. La ridondanza si presenta dal momento che l'attributo NumMessaggi può essere banalmente calcolato alla visualizzazione dell'account, come nel caso del *NumVoti*. Si procederà ad una analisi della ridondanza più specifica.

3. **Ridondanza - Attributo NumVoti in Post**
   L'attributo *NumVoti* nell'entità Post riguarda la differenza tra i voti positivi e i voti positivi assegnati al post. Attraverso questa informazione, è possibile applicare un algoritmo di ranking virtuale ai post, per cui alcuni post vengono visualizzati con maggior rilievo rispetto agli altri, in base al valore dell'attributo.
   La ridondanza sta nel fatto che il numero di voti relativo al post sarebbe banalmente ricavabile alla visualizzazione del post senza bisogno di utilizzare un attributo apposito che dovrebbe venire costantemente aggiornato. Verrà fatta una analisi della ridondanza.

Tabella dei volumi

Concetto	Tipo	Volume Totale approssimativo*
Account	E	18.250
Post	E	109.500 (~ 11 post / account)
Voti	E	675.250 (~ 6 voti / post)
Commenti	E	292.000 (~ 3 comm. / post)
Foto	E	54.750 (~ 3 foto / account)
Progetto	E	21.900 (~ 1,2 progetti / account)
Messaggio (chat)	E	12.775.000 (~ 700 messaggi / account)
Log	E	1.484.820 (~ 82 azioni significative / account)
Follow	R	182.500 (~ 10 followers / account)
Classificazione (tag)	R	109.500 (~ 1 tag / post)

** Il volume calcolato si basa sulle operazioni previste giornaliere e calcolate in un anno (365 giorni)*

Tabelle delle operazioni

1. Attributo NumVoti in Accounts

Operazione	Tipo	Frequenza
5. Votazione di un post da parte di un account	I	1850 volte al giorno
21. Visualizzazione di un profilo da parte di un account (che implica, tra le varie informazioni, la visualizzazione del **Numero dei voti ottenuti**)	I	(5 profili / utente) 12.500 volte al giorno

OP. 5 - CON RIDONDANZA				OP. 5 - SENZA RIDONDANZA			
Concetto	**Costrutto**	**Accesso**	**Tipo**	**Concetto**	**Costrutto**	**Accesso**	**Tipo**
Voti	E	1	S	Voti	E	1	S
Post	E	1	L				
Account	E	1	S				
L'operazione di votazione di un post (analizzando questo caso particolare) implica l'aggiunta del voto ad un post, la selezione dell'account che ha inserito il post e il relativo incremento del contatore NumVoti presente in Account.				In questo caso, l'operazione di votazione di un post implica solamente l'inserimento del voto relativo al post.			
TOTALE ACCESSI [(2 * 1850) + (1 * 1850) + (2 * 1850)] = 9250				**TOTALE ACCESSI** (2 * 1850) = 3700			

OP. 21 - CON RIDONDANZA				OP. 21 - SENZA RIDONDANZA			
Concetto	**Costrutto**	**Accesso**	**Tipo**	**Concetto**	**Costrutto**	**Accesso**	**Tipo**
Account	E	1	L	Account	E	1	L
				Post	E	x=11	L
				Voti	E	x*y=66	L
Poiché l'attributo NumVoti è presente e già contato settato al valore giusto è sufficiente reperire individualmente questo dato.				Questa operazione implica il reperimento dell'account che che si vuole visualizzare e il conteggio dei voti totali ottenuto dalla somma dei voti totali (x) per ogni post (y) inserito dall'account. *Riportando le medie dalla tabella dei volumi avremo 6 voti per ogni post, ossia x*y = 6*11 = 66, sebbene il numero di voti per post e il numero di post stesso possa essere variabile per ogni account.* Accediamo dunque una volta per recuperare l'account, **x volte** per recuperare tutti i post dell'account e **x*y volte** per recuperare i voti (**y = voti per post)** di ciascun post.			
TOTALE ACCESSI (12.500*1) = 12.500				**TOTALE ACCESSI** [(1 * 12.500) + (11*12.500) + (66 * 12.500)] = 12.500 +137.500 + 825.000 = 975.000			

2. Attributo NumMessaggi in Accounts

Operazione	Tipo	Frequenza
3. Inserimento post da parte di un account	I	300 volte al giorno
4. Inserimento commento in un post da parte di un account	I	800 volte al giorno
21. Visualizzazione di un profilo da parte di un account (che implica, tra le varie informazioni, la visualizzazione del **Numero dei messaggi**, ossia commenti e post, **inseriti**)	I	(5 profili / utente) 12.500 volte al giorno

OP. 3 e 4 - CON RIDONDANZA				OP. 3 e 4 - SENZA RIDONDANZA			
Concetto	**Costrutto**	**Accesso**	**Tipo**	**Concetto**	**Costrutto**	**Accesso**	**Tipo**
Post / Comm.	E	1	S	Post / Comm.	E	1	S
Account	E	1	S				

L'operazione include l'inserimento del Post o del commento all'interno della base di dati e l'incremento della variabile contatore NumMessaggi.	L'operazione include solamente l'inserimento del Post o del commento all'interno della base di dati.
TOTALE ACCESSI [Post] (300*2) + (300*2) = 1200 **TOTALE ACCESSI [Commenti]** (800*2) + (800*2) = 3200	**TOTALE ACCESSI [Post]** (300*2) = 600 **TOTALE ACCESSI [Commenti]** (800*2) = 1600

OP. 21 - CON RIDONDANZA				**OP. 21 - SENZA RIDONDANZA**			
Concetto	**Costrutto**	**Accesso**	**Tipo**	**Concetto**	**Costrutto**	**Accesso**	**Tipo**
Account	E	1	L	Account	E	1	L
				Post	E	11	L
				Commenti	E	16	L
Poiché l'attributo NumMessaggi è presente sarà sufficiente reperire individualmente questo dato.				Questa operazione implica il reperimento dell'account che che si vuole visualizzare e il conteggio dei commenti (y) e dei post (x) totali inviati dall'account. *Riportando le medie dalla tabella dei volumi una media di 11 post per account e 16 commenti per account, ossia (x+y) = 11+16 = 27, sebbene il numero di post e il numero di commenti possa essere variabile per ogni account*. Si accede dunque la prima volta per reperire l'account, la seconda volta per trovare il numero di post di quel account e la terza per recuperare il numero di commenti di quel account.			
TOTALE ACCESSI (12.500*1) = 12.500				**TOTALE ACCESSI** [(1*12.500) + (11*12.500) + (16*12.500)] = 350.000			

3. Attributo NumVoti in Post

Operazione	**Tipo**	**Frequenza**
5. Votazione di un post da parte di un account	I	1850 volte al giorno
20. Visualizzazione di un post da parte di un account	I	(30 post / utente) 75.000 volte al giorno

OP. 5 - CON RIDONDANZA				**OP. 5 - SENZA RIDONDANZA**			
Concetto	**Costrutto**	**Accesso**	**Tipo**	**Concetto**	**Costrutto**	**Accesso**	**Tipo**
Voti	E	1	S	Voti	E	1	S
Post	E	1	S				
L'operazione di votazione di un post (analizzando questo caso particolare) implica l'aggiunta del voto ad un post e l'incremento del contatore NumVoti per il post.				In questo caso, l'operazione di votazione di un post implica solamente l'inserimento del voto relativo al post.			
TOTALE ACCESSI [(2 * 1850) + (2 * 1850)] = 7400				**TOTALE ACCESSI** (2 * 1850) = 3700			

OP. 20 - CON RIDONDANZA				OP. 20 - SENZA RIDONDANZA			
Concetto	**Costrutto**	**Accesso**	**Tipo**	**Concetto**	**Costrutto**	**Accesso**	**Tipo**
Post	E	1	L	Post	E	1	L
				Voti	E	x=6	L
Poiché l'attributo NumVoti è presente nel post sarà sufficiente reperire individualmente questo dato.				Questa operazione implica il reperimento del post e dei voti relativi al post. Ponendo, in base alla tabella dei volumi, una media di 6 voti (x voti) per post, eseguiremo 6 volte l'accesso ai voti per effettuare quindi il calcolo dei (voti positivi - voti negativi).			
TOTALE ACCESSI (75.000*1) = 75.000				**TOTALE ACCESSI** [(1*75.000) + (6*75.000)] = 525.000			

Risultati finali

1. Ridondanza - Attributo NumVoti in Account

- Operazioni totali con ridondanza (9250 + 12.500)
- Operazioni totali senza ridondanza (3700 + 975.000)

Analizzando l'attributo NumVoti e le relative operazioni si è potuto constatare che, al fine di ottenere il numero di voti per ciascun post scritto dall'account analizzato, sia meglio **mantenere la ridondanza** dal momento che, sebbene siano richieste più operazioni durante l'inserimento di un voto, è possibile beneficiarne nel momento in cui viene visualizzato il profilo di un utente essendo il numero di operazioni previste per la visualizzazione di tale dato nettamente inferiore, ma soprattutto molto dipendente dall'attività dell'utente. Difatti, più l'utente inserisce post e più voti può ottenere.

2. Ridondanza - Attributo NumMessaggi in Account

- Operazioni totali con ridondanza (1200 + 3200 + 12.500)
- Operazioni totali senza ridondanza (600 + 1600 + 350.000)

Analizzando l'attributo NumMessaggi e le relative operazioni si è optato per **mantenere la ridondanza** per il seguente motivo: sebbene ad ogni inserimento di un post o di un commento si debba incrementare il numero di messaggi effettuati, questo non causa un gap sufficientemente alto tra le due operazioni con e senza ridondanza. Questo è dovuto al fatto che l'operazione successiva che è correlata al reperimento del numero di messaggi inseriti è fortemente dipendente dal numero di commenti e di post che l'utente inserisce giornalmente. Anche evitando la ridondanza, un account con un numero di post e di co
mmenti nella media richiede sicuramente troppe operazioni al fine di reperire un dato piuttosto banale, ma comunque importante per fini statistici e per i fini del social network. Per questi motivi, si è preferito pesare di più sulla velocità di visualizzazione di questo dato rispetto al suo modo per essere ottenuto.

3. Ridondanza - Attributo NumVoti in Post

- Operazioni totali con ridondanza (7400 + 75.000)
- Operazioni totali senza ridondanza (3700 + 525.000)

Analizzando l'attributo NumVoti e le relative operazioni si è optato per **mantenere la ridondanza**, anche in questo caso. Il NumVoti relativo ad un post viene utilizzato per determinare, attraverso un algoritmo apposito di post ranking, i post più popolari da far visualizzare nella bacheca dell'utente. Dal momento che il dato per essere ottenuto richiede la differenza tra i voti positivi e i voti negativi e dunque l'accesso a ciascun voto, si è osservato che all'aumentare del numero di voti che corrispondono al post si eseguiranno meno accessi usando la ridondanza.

Eliminazione delle generalizzazioni

Lo schema concettuale sviluppato finora presenta svariate generalizzazioni, non rappresentabili nel modello relazionale e quindi non utilizzabili come implementazione nel database. Si procede quindi a reificare le generalizzazioni tramite utilizzo di attributi e ulteriori entità.

utente

utente possiede tre entità figlie disgiunte: da un lato le entità *utente_attivo* e *utente_cancellato* e dall'altro l'entità *utente_admin*.

utente_attivo* e *utente_cancellato: esse posseggono gli stessi attributi dell'entità padre, e anzi sono la stessa entità che concettualmente si divide in utenti cancellati e ancora attivi. Si è deciso di accorpare le due entità figlie all'interno dell'entità padre tramite l'utilizzo di un attributo booleano *Cancellato* che identifica lo stato dell'utente: 0 per attivo e 1 per cancellato.

utente_admin: *utente_admin* è sottoinsieme di *utente*, e possiede i medesimi attributi di quest'ultima; l'unica differenza concettuale è l'accesso a impostazioni riservate agli amministratori del sito. Si è deciso quindi di accorpare

l'entità all'interno di *utente* tramite l'utilizzo di un attributo booleano *IsAdmin* che identifica la differenza tra utente semplice (0) e utente amministratore (1).
La motivazione di queste scelte, oltre a un'ovvia semplificazione dello schema e alla sua facile rappresentabilità nella base di dati, è che con questa reificazione si può rappresentare tutte quelle che sarebbero tabelle separate all'interno della stessa tabella, con l'aggiunta di due soli attributi booleani.

pagina

Le stesse motivazioni che hanno portato alla scelta di reificazioni di *utente* sono applicabili all'entità *pagina*: le entità figlie *pagina_attiva* e *pagina_cancellata* sono quindi state accorpate all'entità padre *pagina*, con l'utilizzo di un attributo booleano *Cancellato.*

account

Dopo aver eliminato le generalizzazioni di *utente* e *pagina*, si è deciso di accorpare queste due all'interno della stessa entità *account*. Tutti gli attributi di *account*, infatti, sono anche attributi di *utente* e *pagina*, e non sono presenti relazioni specifiche per queste ultime due.
Sebbene le entità figlie abbiano attributi propri tra loro non comuni, si è optato per unirle comunque nell'entità padre con l'utilizzo di un valore booleano *isPage* che identifica l'account come utente (0) o come pagina (1). In questo modo vengono accorpati anche gli attributi propri, prevedendo che gli attributi non appartenenti alla sottoclasse selezionata siano nulli.
Sebbene possa sembrare una scelta controintuitiva, la motivazione alla base è solida: nel caso la reificazione fosse stata fatta attraverso trasformazione in entità correlate, le operazioni sulla base di dati sarebbero state più complicate con conseguente perdita di efficienza, in quanto *account* è l'entità sulla quale agisce la maggior parte delle operazioni.

log

L'entità *log* possiede due entità figlie: *azione_admin* e *azione_utente*. Queste due possiedono gli stessi attributi di *log*, in quanto questa generalizzazione è solo una divisione concettuale in ciò che un utente, admin o semplice, può o non può vedere. Si è quindi deciso, anche in questo caso, di accorpare le entità figlie nell'entità padre con l'aggiunta di un attributo booleano *IsAdminAction*: 0 per utente semplice e 1 per utente amministratore.

post e commento

Sia l'entità *post* che l'entità *commento* sono generalizzazioni di entità che gestiscono la cancellazione (*post_attivo* e *post_cancellato* per *post*, *commento_attivo* e *commento_cancellato* per *commento*). Anche in questo caso, la soluzione è stata l'aggiunta di un valore booleano *Cancellato* (per entità) che divide i post e i commenti attivi (0) da quelli cancellati (1).

foto

Anche l'entità *foto* è generalizzata in due entità figlie che gestiscono la possibilità di cancellazione; essa, inoltre, presenta una generalizzazione parziale per indicare che una foto può essere del profilo, di copertina, in un post o non appartenente a nessuno di questi casi.
***foto_attiva* e *foto_cancellata*:** Entrambe queste entità figlie possiedono i medesimi attributi; anche in questo caso si è deciso di reificare la generalizzazione con l'introduzione di un attributo booleano *Cancellato*, il quale indica se la foto è attiva (0) o cancellata (1).
foto_profilo*, *foto_copertina*, *foto_post: In questo caso la generalizzazione è parziale; la soluzione adottata, in questo caso, è stata duplice: da un lato è stata creata una nuova entità *foto_in_post*, la quale funge da collegamento tra l'entità *foto*, nella quale rimangono memorizzate tutte le foto caricate sul sito, e l'entità *post*. Dall'altro lato sono stati aggiunti due attributi *propicID* e *coverpicID* all'interno di *account*; questi attributi servono a collegare a ogni account la propria immagine del profilo e la propria immagine di copertina.

portfolio

L'entità *portfolio* è un concetto utile solo a livello di sito, mentre a livello di database non è utile e, anzi, rischia di complicare ulteriormente le applicazioni e di inficiare sull'efficienza di esse. È stato perciò deciso di eliminarla, tenendo solo l'entità *progetto* come tabella contenitrice di tutti i progetti, identificati in base all'ID dell'account.
N.B. È necessario fare una precisazione sulla cancellazione di account, di foto, di post e di commento. Quando un utente cancella uno di questi elementi, i dati presenti in memoria non vengono immediatamente cancellati, ma vengono solo resi invisibili a tutti gli utenti. La base di dati si serve poi di un *cron job* per la cancellazione definitiva di tutti gli elementi dalla memoria.

Ulteriori reificazioni

follow: relazione -> entità

La relazione *follow* non è rappresentabile nel modello relazionale, né tantomeno implementabile in una base di dati reale. Si è quindi deciso di reificarla sostituendola con un'entità *follow*. Quest'entità è così definita:

follow			
followerID	INTEGER	Id dell'account (accountID) che segue il following.	Chiave
followingID	INTEGER	Id dell'account (accountID) che viene seguito dal follower.	Chiave
DataFollow	TIMESTAMP	Data dalla quale il follower segue il following.	

messaggio
Per come è schematizzata nel modello concettuale, l'entità messaggio non è utile a rappresentare una vera conversazione tra due account nel modello logico. Vengono quindi aggiunti due attributi *accountMittenteID* e *accountDestinatarioID* al posto di *accountID* per rappresentare la vera conversazione.
log
L'indirizzo IP dell'utente è reperibile soltanto tramite interfaccia web, e l'unico modo per permettere la registrazione di una riga nella tabella *log,* incluso l'attributo IP, è utilizzare un attributo *currentIP* in *account* che serve, a ogni accesso dell'utente, a determinare l'indirizzo IP in uso.
notifica: relazione -> entità
La relazione *notifica* non è rappresentabile nel modello logico dei dati; viene perciò trasformata in un'entità *notifica* così definita:

notifica			
notificaID	INTEGER	Id univoco della notifica.	Chiave
accountID	INTEGER	Id univoco dell'account che riceve la notifica.	
DataOra	TIMESTAMP	Ora in cui viene mandata la notifica.	
Descrizione	TEXT	Descrizione della notifica.	
Visualizzata	BOOL	Identifica una notifica come visualizzata (1) o non visualizzata (0)	

Vincoli di integrità referenziale

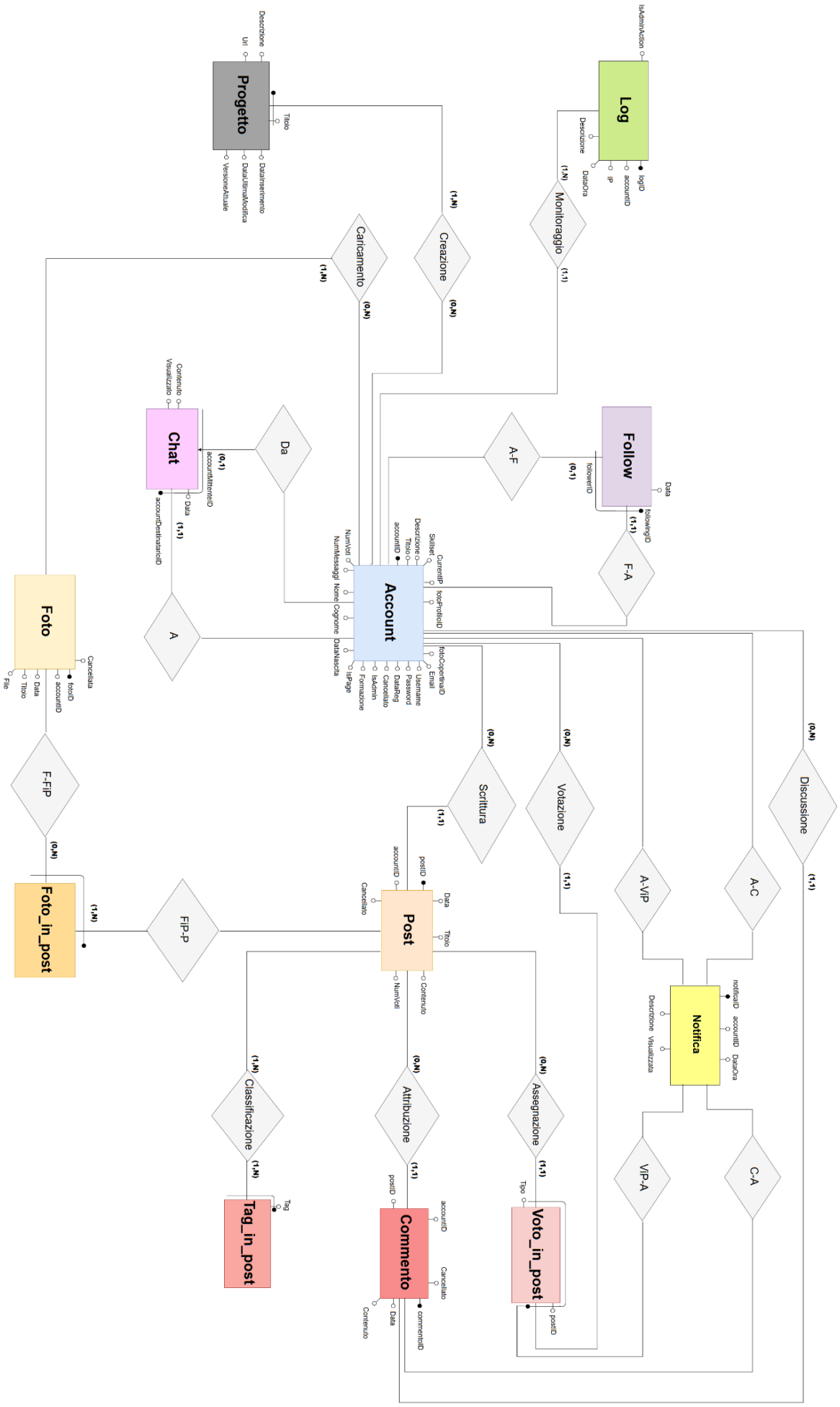
account - foto: In *account* sono presenti due vincoli di integrità referenziale tra *fotoProfiloID* e *fotoID* di *foto*, e tra *fotoCopertinaID* e *fotoID* di *foto*.
chat - account: In *chat* è presente un vincolo di integrità referenziale tra *accountMittenteID* e *accountID* di *account*, e tra *accountDestinatarioID* e *accountID* di *account.*
commento - account: In *commento* è presente un vincolo di integrità referenziale tra *accountID* e *accountID* di *account*.
commento - post: In *commento* è presente un vincolo di integrità referenziale tra *postID* e *postID* di *post*.
follow - account: In *follow* è presente un vincolo di integrità referenziale tra *followerID* e *accountID* di *account*, e tra *followingID* e *accountID* di *account*.
foto - account: In *foto* è presente un vincolo di integrità referenziale tra *accountID* e *accountID* di *account.*
foto_in_post - foto: In *foto_in_post* è presente un vincolo di integrità referenziale tra *fotoID* e *fotoID* di *foto*.
foto_in_post - post: In *foto_in_post* è presente un vincolo di integrità referenziale tra *postID* e *postID* di *post*.
log - account: In *log* è presente un vincolo di integrità referenziale tra *accountID* e *accountID* di *account*.
post - account: In *post* è presente un vincolo di integrità referenziale tra *accountID* e *accountID* di *account*.
progetto - account: In *progetto* è presente un vincolo di integrità referenziale tra *accountID* e *accountID* di *account*.
tag_in_post - post: In *tag_in_post* è presente un vincolo di integrità referenziale tra *postID* e *postID* di *post*.
voto_in_post - post: In *voto_in_post* è presente un vincolo di integrità referenziale tra *postID* e *postID* di *post*.
voto_in_post - account: In *voto_in_post* è presente un vincolo di integrità referenziale tra *accountID* e *accountID* di *account*.

notifica - account: In *notifica* è presente un vincolo di integrità referenziale tra *accountID* e *accountID* di *account.*

Traduzione verso il modello relazionale

ACCOUNT(accountID, DataReg, CurrentIP, Username, Email, Password, NumMessaggi, NumVoti, IsPage, Nome, Cognome, DataNascita, Formazione, Skillset, Titolo, Descrizione, fotoCopertinaID, fotoProfiloID, IsAdmin, Cancellato)
FOTO(fotoID, accountID, Data, Titolo, File, Cancellata)
POST(postID, accountID, Data, Titolo, Contenuto, NumVoti, Cancellato)
FOTO_IN_POST(postID, fotoID)
COMMENTO(commentoID, accountID, postID, Data, Contenuto, Cancellato)
VOTO_IN_POST(postID, accountID, Tipo)
FOLLOW(followerID, followingID, Data)
PROGETTO(Titolo, accountID, DataInserimento, DataUltimaModifica, VersioneAttuale, Descrizione, Url)
CHAT(accountMittenteID, accountDestinatarioID, Data, Contenuto,Visualizzato)
LOG(logID, accountID, IP, DataOra, Descrizione, IsAdminAction)
TAG_IN_POST(postID, Tag)
NOTIFICA(notificaID, accountID, Visualizzata, Descrizione, DataOra)

Schema E/R ristrutturato



Si vuole creare una funzione che dato l'ID di un account controlli se un conflitto di Indirizzo IP.

```
DELIMITER ||
CREATE FUNCTION ConflittoIP(aid integer) RETURNS boolean
BEGIN
  DECLARE ip varchar(24);
  SELECT CurrentIP
  INTO ip
  FROM account
  WHERE accountID = aid;
  IF(SELECT COUNT(*) FROM account a WHERE a.CurrentIP = ip AND a.accountID != aid) > 0
  THEN
    RETURN true;
  ELSE
    RETURN false;
  END IF;
END ||
DELIMITER ;
```

Si vuole creare una funzione che dato l'ID di un account ritorni il tag più utilizzato nei suoi post e in particolare si vuole sapere o il tag in sé (type=1) o la frequenza con cui è stato usato (type!=1).

```
DELIMITER ||
CREATE FUNCTION MostUsedTag(aid integer, type tinyint(1)) RETURNS varchar(32)
BEGIN
  DECLARE tag varchar(32);
  DECLARE ntag int(11);

  SELECT tip.Tag, COUNT(tip.Tag)
  INTO tag, ntag
  FROM tag_in_post tip JOIN post p ON tip.postID = p.postID
  WHERE p.accountID = aid
  GROUP BY tip.Tag
  ORDER BY tip.postID DESC
  LIMIT 1;

  IF(type > 0)
  THEN
    RETURN tag;
  ELSE
    RETURN ntag;
  END IF;

END ||
DELIMITER ;
```

Si vuole calcolare il numero di account attivi, di utenti amministratori e di account cancellati

```
CREATE VIEW attivi AS
        SELECT COUNT(accountID) as "Utenti attivi"
        FROM account
        WHERE Cancellato='0';
```

```
CREATE VIEW admins AS
        SELECT COUNT(accountID) as "Utenti amministratori"
        FROM account
        WHERE IsAdmin='1';
CREATE VIEW cancellati AS
        SELECT COUNT(accountID) as "Utenti cancellati"
        FROM account
        WHERE Cancellato='1';

SELECT * FROM attivi, admins, cancellati;
```

Utenti attivi	Utenti amministratori	Utenti cancellati
7	3	1

Si vogliono visualizzare gli utenti con un conflitto di indirizzo IP.

```
CREATE VIEW conflittoIP AS
    SELECT a.IsPage as "Pagina(1) - Utente(0)", a.Username, a.CurrentIP as "IP"
    FROM account a
    WHERE ConflittoIP(a.accountID)=1;
```

Pagina(1) – Utente(0)	Username	IP
0	Maxel	192.168.0.1
1	Spritz	192.168.0.1

Si vogliono conoscere i post degli utenti che sono stati cancellati.

```
SELECT a.accountID AS "ID Account", a.Username AS "Username", p.postID AS "ID Post"
FROM post p, account a
WHERE a.Cancellato='1' AND a.accountID=p.accountID;
```

ID Account	Username	ID Post	Titolo Post
8	xyzPwner	5	Buy Arduino for 9,99$!

Si vogliono visualizzare i commenti relativi ad un post

```
DELIMITER ||
CREATE PROCEDURE commenti_post(idp INT(11))
BEGIN
SELECT a.Username, com.Data, com.Contenuto
FROM commento com, post p, account a
WHERE p.postID=idp AND com.postID=p.postID AND a.accountID=com.accountID AND
p.Cancellato=0;
END ||
DELIMITER ;
```

Username	Data	Contenuto
Maxel	2018-01-21 19:57:42	E non dimenticate di followarci!
BreakTheWall	2018-01-21 20:01:22	Ciao a tutti! Bel social network!
Hikaria	2018-01-21 20:25:42	Continuate così :)

Si vuole visualizzare la chat tra due account.

```
DELIMITER ||
CREATE PROCEDURE visualizza_chat (id1 INT(11), id2 INT(11))
BEGIN
SELECT a.Username, c.Data, c.Contenuto, c.Visualizzato
FROM account a, chat c
WHERE (c.accountMittenteID=id1 AND c.accountDestinatarioID=id2 AND a.accountID=id1) OR
(c.accountMittenteID=id2 AND c.accountDestinatarioID=id1 AND a.accountID=id2)
ORDER BY c.Data ASC;
END ||
DELIMITER ;

CALL visualizza_chat(1, 2);
```

Username	Data	Contenuto	Visualizzato
Maxel	2018-01-21 18:54:25	Ehy sembra funzionare la chat!	1
Enricobu6	2018-01-21 18:54:53	A quanto pare si, domani finiamo il progetto e lo mandiamo.	1
Enricobu6	2018-01-21 18:56:06	E non dimenticarti di mandarmi il link della cyberchallenge che la proviamo.	0

Si vuole visualizzare un post con il numero di voti

```
DELIMITER ||
CREATE PROCEDURE visualizza_post(idp INT(11))
BEGIN
SELECT p.postID AS "ID Post", p.Titolo as "Titolo", p.NumVoti AS "Numero voti"
FROM post p
WHERE p.postID=idp;
END ||
DELIMITER ;
```

ID Post	Titolo	Numero voti
1	Primo post	2

Si vogliono visualizzare i tags più popolari

```
SELECT Tag, COUNT(Tag) as "Numero Tag"
FROM tag_in_post
GROUP BY Tag
ORDER BY COUNT(Tag) DESC;
```

Tag	Numero Tag
tecnologia	13
proposte di lavoro	2
php	1
html	1
manutenzione	1

Si vogliono visualizzare i post degli account seguiti (follow)

```
DELIMITER ||
CREATE PROCEDURE bacheca (id INT(11))
BEGIN
SELECT p.accountID, p.postID AS “ID Post”, p.Titolo AS “Titolo”
FROM post p JOIN follow fol ON fol.followingID=p.accountID
WHERE p.Cancellato=0 AND fol.followerID=id
ORDER BY p.Data,p.NumVoti DESC;
END ||
DELIMITER ;
```

accountID	ID Post	Titolo
2	2	Problemi al server

Si vogliono visualizzare il numero medio di voti e il numero medio di messaggi inviati da utenti o pagine non cancellati.

```
CREATE VIEW Statistiche AS
SELECT a.IsPage AS "Utente (0) o pagina (1)",
       AVG(a.NumVoti) AS "Numero medio voti",
       AVG(a.NumMessaggi) AS "Numero medio messaggi"
FROM account a
WHERE a.Cancellato=0
GROUP BY a.IsPage
```

Utente (0) o pagina (1)	Numero medio voti	Numero medio messaggi
0	1,833	2,333
1	1,500	0,500

Si vuole visualizzare il numero di tag inseriti (anche ripetuti) per ogni utente

```
SELECT a.Username, COUNT(*) AS “Numero Tags”
FROM account a, tag_in_post tip, post p
WHERE p.accountID=a.accountID AND tip.postID=p.postID
GROUP BY a.accountID;
```

Username	Numero Tags
Maxel	3
Enricobu6	1
ZephirCorporation	2

Si vuole controllare attraverso un Trigger sulla tabella *account* che l’iscrizione dell’account avvenga correttamente.

```
DROP TRIGGER IF EXISTS ControlloIscrizione;

DELIMITER ||
CREATE TRIGGER ControlloIscrizione
BEFORE INSERT ON account
FOR EACH ROW
BEGIN
	DECLARE msg varchar(200);
	DECLARE n int(11);

	IF (New.IsPage > 0)
	THEN
		IF	(New.Nome IS NOT NULL) OR
			(New.Cognome IS NOT NULL) OR
			(New.DataNascita IS NOT NULL) OR
			(New.Skillset IS NOT NULL) OR
			(New.Formazione IS NOT NULL)
		THEN
			SET msg = "La pagina non può avere come attributi Nome, Cognome, Data di
nascita, Abilità e Formazione.";
			SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
		END IF;
	ELSE
		IF	(New.Titolo IS NOT NULL) OR
			(New.Descrizione IS NOT NULL)
		THEN
			SET msg = "Un utente non può avere come attributi Titolo e Descrizione.";
			SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
		END IF;
	END IF;


	SELECT COUNT(*) INTO n FROM account WHERE Username = New.Username AND accountID !=
New.accountID;
```

```
	IF(n <> 0)
	THEN
		SET msg = "Lo username inserito risulta già presente nel database.";
		SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
	END IF;

	SELECT COUNT(*) INTO n FROM account WHERE Email = New.Email AND accountID != 
New.accountID;
	IF(n <> 0)
	THEN
		SET msg = "L'email inserita risulta già presente nel database.";
		SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
	END IF;

END ||
DELIMITER ;
```

Si vuole controllare attraverso un Trigger sulla tabella *account* che l'aggiornamento di un account avvenga correttamente, senza ripetizioni di Username e Email (essendo i campi settati UNIQUE) e senza che un account cancellato possa diventare Admin.

```
DROP TRIGGER IF EXISTS ControlloUtente;

DELIMITER ||
CREATE TRIGGER ControlloUtente
BEFORE UPDATE ON account
FOR EACH ROW
BEGIN
	DECLARE msg varchar(200);
	DECLARE n int(11);

	IF (New.IsAdmin > 0)
	THEN
		IF (New.Cancellato > 0)
		THEN
			SET msg = "Un admin deve essere un utente attivo!";
			SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
		END IF;
	END IF;

	IF(New.Username <> Old.Username)
	THEN
		SELECT COUNT(*) INTO n FROM account WHERE Username = New.Username AND accountID != 
New.accountID;
		IF(n <> 0)
		THEN
			SET msg = "Il nuovo username per l'utente risulta già presente nel 
database.";
			SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
		END IF;
	END IF;

	IF(New.Email <> Old.Email)
	THEN
		SELECT COUNT(*) INTO n FROM account WHERE Email = New.Email AND accountID != 
New.accountID;
```

```
            IF(n <> 0)
            THEN
                  SET msg = "La nuova email per l'utente risulta già presente nel database.";
                  SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = msg;
            END IF;
      END IF;
END ||
DELIMITER ;
```

Si vuole aggiungere un log attraverso un Trigger sulla tabella *account* nel momento in cui l'utente si registra.

```
DROP TRIGGER IF EXISTS LogUserSubscription;

DELIMITER ||
CREATE TRIGGER LogUserSubscription
AFTER INSERT ON account
FOR EACH ROW
BEGIN
      INSERT INTO log (accountID, IP, Descrizione)
      VALUES (New.accountID, New.CurrentIP, 'USER_SUBSCRIPTION');
END ||
DELIMITER ;
```

Attraverso un Trigger sulla tabella *post* nel momento in cui viene inserito un nuovo post, si vuole tenere traccia dell'azione inserendo una nuova log e incrementando il numero di messaggi per l'utente che ha inserito il post.

```
DROP TRIGGER IF EXISTS SendPost;

DELIMITER ||
CREATE TRIGGER SendPost
AFTER INSERT ON post
FOR EACH ROW
BEGIN
      DECLARE UserIP varchar(24);
      DECLARE Azione varchar(64);

      SELECT a.CurrentIP
      INTO UserIP
      FROM account a
      WHERE a.accountID = New.accountID
      LIMIT 1;

      UPDATE account
      SET NumMessaggi = NumMessaggi+1
      WHERE accountID = New.accountID
      LIMIT 1;

      SET Azione = CONCAT('USER_SEND_POST (ID = ', New.postID, ')');
      INSERT INTO log (accountID, IP, Descrizione)
      VALUES (New.accountID, UserIP, Azione);
END ||
DELIMITER ;
```

Attraverso un Trigger sulla tabella *commento* nel momento in cui viene inserito un nuovo commento, si vuole tenere traccia dell'azione inserendo una nuova log e incrementando il numero di messaggi per l'utente che ha inserito il commento. Inoltre si vuole notificare l'autore del post della presenza di un nuovo commento

```
DROP TRIGGER IF EXISTS SendComment;

DELIMITER ||
CREATE TRIGGER SendComment
AFTER INSERT ON commento
FOR EACH ROW
BEGIN
	DECLARE AuthorID int(11);
	DECLARE UserIP varchar(24);
	DECLARE Azione varchar(64);
	DECLARE DescNot varchar(256);
	DECLARE FromUsername varchar(24);
	DECLARE TitoloPost varchar(128);

	SELECT a.CurrentIP
	INTO UserIP
	FROM account a
	WHERE a.accountID = New.accountID
	LIMIT 1;

	UPDATE account
	SET NumMessaggi = NumMessaggi+1
	WHERE accountID = New.accountID
	LIMIT 1;

	SELECT accountID, Titolo
	INTO AuthorID, TitoloPost
	FROM post
	WHERE postID = New.postID
	LIMIT 1;

	SELECT Username
	INTO FromUsername
	FROM account
	WHERE accountID = New.accountID
	LIMIT 1;

	IF (AuthorID <> New.accountID)
	THEN
		SET DescNot = CONCAT(FromUsername, ' ha risposto al tuo post (#', New.postID, ')
', TitoloPost);
		INSERT INTO notifica (accountID, DataOra, Descrizione) VALUES (AuthorID, New.Data,
DescNot);
	END IF;

	SET Azione = CONCAT('USER_SEND_COMMENT (ID = ', New.commentoID, ')', ' (PID = ',
New.postID, ')');
	INSERT INTO log (accountID, IP, Descrizione)
	VALUES (New.accountID, UserIP, Azione);
END ||
DELIMITER ;
```

Attraverso un Trigger sulla tabella *foto* nel momento in cui viene inserita una nuova foto, si vuole tenere traccia dell'azione inserendo una nuova log.

```
DROP TRIGGER IF EXISTS LogAddFoto;

DELIMITER ||
CREATE TRIGGER LogAddFoto
AFTER INSERT ON foto
FOR EACH ROW
BEGIN
	DECLARE UserIP varchar(24);
	DECLARE Azione varchar(64);

	SELECT a.CurrentIP
	INTO UserIP
	FROM account a
	WHERE a.accountID = New.accountID
	LIMIT 1;

	SET Azione = CONCAT('USER_ADD_FOTO (ID = ', New.fotoID, ')');
	INSERT INTO log (accountID, IP, Descrizione)
	VALUES (New.accountID, UserIP, Azione);
END ||
DELIMITER ;
```

Attraverso un Trigger sulla tabella *progetto* nel momento in cui viene inserito un nuovo progetto, si vuole tenere traccia dell'azione inserendo una nuova log.

```
DROP TRIGGER IF EXISTS LogAddProject;

DELIMITER ||
CREATE TRIGGER LogAddProject
AFTER INSERT ON progetto
FOR EACH ROW
BEGIN
	DECLARE UserIP varchar(24);
	DECLARE Azione varchar(256);

	SELECT a.CurrentIP
	INTO UserIP
	FROM account a
	WHERE a.accountID = New.accountID
	LIMIT 1;

	SET Azione = CONCAT('USER_ADD_PROJECT (NAME = ', New.Titolo, ')');
	INSERT INTO log (accountID, IP, Descrizione)
	VALUES (New.accountID, UserIP, Azione);
END ||
DELIMITER ;
```

Si vuole aggiornare il contatore NumMessaggi attraverso un Trigger sulla tabella *post* nel momento in cui viene eseguito un update sul campo Cancellato.

```
DROP TRIGGER IF EXISTS UpdateNumMessaggiByPost;

DELIMITER ||
CREATE TRIGGER UpdateNumMessaggiByPost
AFTER UPDATE ON post
FOR EACH ROW
BEGIN
	IF (New.Cancellato = 1) AND (Old.Cancellato != New.Cancellato)
	THEN
		UPDATE account
		SET NumMessaggi = NumMessaggi-1
		WHERE accountID = New.accountID
		LIMIT 1;
	ELSEIF (New.Cancellato = 0) AND (Old.Cancellato != New.Cancellato)
	THEN
		UPDATE account
		SET NumMessaggi = NumMessaggi+1
		WHERE accountID = New.accountID
		LIMIT 1;
	END IF;
END ||
DELIMITER ;
```

Si vuole aggiornare il contatore NumMessaggi attraverso un Trigger sulla tabella *commento* nel momento in cui viene eseguito un update sul campo Cancellato.

```
DROP TRIGGER IF EXISTS UpdateNumMessaggiByPost;

DELIMITER ||
CREATE TRIGGER UpdateNumMessaggiByComment
AFTER UPDATE ON commento
FOR EACH ROW
BEGIN
	IF (New.Cancellato = 1) AND (Old.Cancellato != New.Cancellato)
	THEN
		UPDATE account
		SET NumMessaggi = NumMessaggi-1
		WHERE accountID = New.accountID
		LIMIT 1;
	ELSEIF (New.Cancellato = 0) AND (Old.Cancellato != New.Cancellato)
	THEN
		UPDATE account
		SET NumMessaggi = NumMessaggi+1
		WHERE accountID = New.accountID
		LIMIT 1;
	END IF;
END ||
DELIMITER ;
```

Si vuole aggiornare il contatore NumVoti di *account* e *post* attraverso un Trigger sulla tabella *voto_in_post* nel momento in cui viene inserito un nuovo dato. In particolare, si vuole aggiornare il contatore dei voti del post (che è dato dalla differenza tra voti positivi e voti negativi) e il contatore dei voti dell'utente (che è il totale dei voti ricevuti). Infine, si vuole notificare l'utente autore del post che ha ricevuto un voto (positivo o negativo) al proprio post.

```
DROP TRIGGER IF EXISTS AddNumVoti;

DELIMITER ||
CREATE TRIGGER AddNumVoti
AFTER INSERT ON voto_in_post
FOR EACH ROW
BEGIN
	DECLARE AuthorID int(11);
	DECLARE DescrizioneNot text;
	DECLARE FromUsername varchar(24);
	DECLARE TitoloPost varchar(24);
	DECLARE t int(1);
	IF (New.Voto > 0) THEN
		SET t = 1;
	ELSE
		SET t = -1;
	END IF;

	UPDATE post
	SET NumVoti = (NumVoti+t)
	WHERE postID = New.postID
	LIMIT 1;

	SELECT p.accountID, p.Titolo
	INTO AuthorID,TitoloPost
	FROM post p
	WHERE p.postID = New.postID
	LIMIT 1;

	UPDATE account
	SET NumVoti = (NumVoti+1)
	WHERE accountID = AuthorID
	LIMIT 1;

	SELECT Username
	INTO FromUsername
	FROM account
	WHERE accountID = New.accountID
	LIMIT 1;

	IF (AuthorID <> New.accountID)
	THEN
		IF( t > 0)
		THEN
			SET DescrizioneNot = CONCAT(FromUsername, ' ha valutato positivamente il
tuo post (#', New.postID, ') ', TitoloPost);
		ELSE
			SET DescrizioneNot = CONCAT(FromUsername, ' ha valutato negativamente il
tuo post (#', New.postID, ') ', TitoloPost);
		END IF;
```

```
            INSERT INTO notifica (accountID, Descrizione) VALUES (AuthorID, DescrizioneNot);
      END IF;
END ||
DELIMITER ;
```

Si vuole aggiornare il contatore NumVoti di *account* e *post* attraverso un Trigger sulla tabella *voto_in_post* nel momento in cui viene rimosso un dato. In particolare, si vuole aggiornare il contatore dei voti del post e quello dell'utente.

```
DROP TRIGGER IF EXISTS UpdateNumMessaggiByPost;

DELIMITER ||
CREATE TRIGGER RemoveNumVoti
AFTER DELETE ON voto_in_post
FOR EACH ROW
BEGIN
      DECLARE AuthorID int(11);
      DECLARE t int(1);

      IF (Old.Voto > 0)
      THEN
            SET t = -1;
      ELSE
            SET t = 1;
      END IF;

      UPDATE post
      SET NumVoti = (NumVoti+t)
      WHERE postID = Old.postID
      LIMIT 1;

      SELECT p.accountID INTO AuthorID FROM post p WHERE p.postID = Old.postID LIMIT 1;

      UPDATE account
      SET NumVoti = (NumVoti-1)
      WHERE accountID = AuthorID
      LIMIT 1;

END ||
DELIMITER ;
```

Nota dei relatori: Logs

Nella Base di Dati sono stati omessi volontariamente alcuni trigger che fungevano per il sistema logs implementato dal momento che determinati dati (es. cancellazione account, post, commento) possono essere eseguiti sia da parte di un admin che da parte di un utente (autore). Ai fini di questa completezza, il sistema logs si può ritenere completo solo con una adeguata implementazione web che permette l'identificazione (es. attraverso apposite sessioni PHP) di colui che effettivamente esegue l'azione.

FINE