Esercitazione di laboratorio

Corso di Gestione dell'Informazione

A.A. 2010/2011

Turno di appartenenza: A

Docente di riferimento: Sergio Mascetti

Titolo esercitazione: *MiniFacebook*

Gruppo composto da:

Francesca Madeddu francesca.madeddu@studenti.unimi.it [referente]

Alberto Camillo alberto.camillo@studenti.unimi.it

Data consegna: *{07/07/2011}*

Studenti che intendono partecipare alla prova:

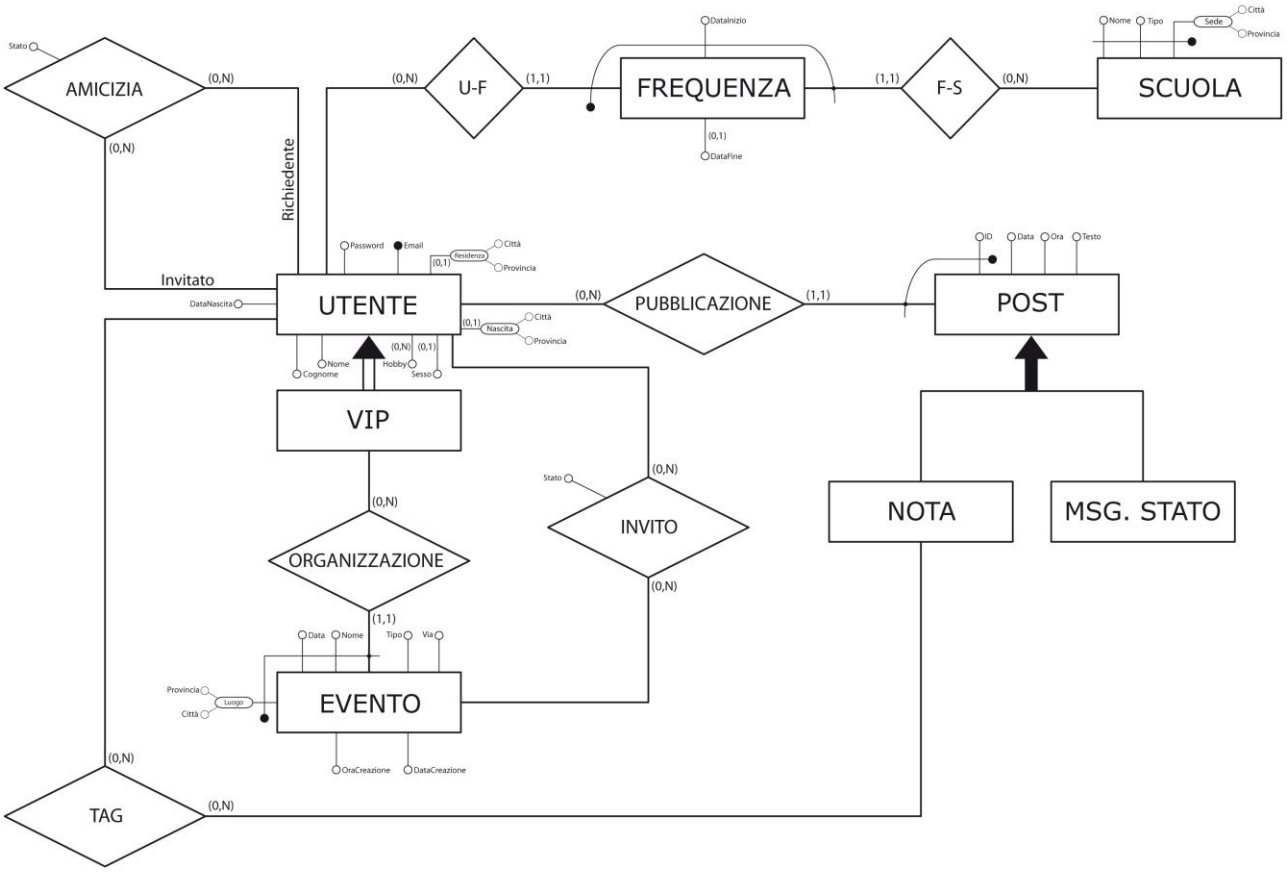
Francesca Madeddu

Alberto Camillo

Il progetto è già stato consegnato in precedenza? {no}

1. Progettazione concettuale

1.1 Schema ER



(presente in .pdf nella cartella del progetto)

Dizionario delle entità

Nome	Descrizione	Attributi	Identificatore
Utente	Utente iscritto al servizio miniFB.	**Nome** **Cognome** **Sesso** (opzionale) **Email** **Password** **Residenza** (Città, Provincia) **Nascita** (Città, Provincia) **Hobby** (opzionale, multivalore) **DataNascita**	**Email**
Vip	Utente con più di dieci amici.		
Frequenza	Frequenza di un utente in una determinata scuola.	**DataInizio** **DataFine** (opzionale)	**Utente** **Scuola** **DataInizio**

Scuola	Scuola frequentata da un utente.	**Nome** **Tipo** **Sede** (Città, Provincia)	**Nome** **Tipo** **Sede**
Evento	Evento organizzato da un utente VIP.	**Nome** **Tipo** **Data** **Ora** **Luogo** (Città, Provincia) **Via**	**Nome** **Vip** **Data**
Post	Post pubblicato sulla bacheca di un utente. Può essere di vari tipi espressi nella generalizzazione.	**ID** **Data** **Ora** **Testo**	**ID** **Utente**
Nota	Tipo di post in cui possono essere "taggati" altri utenti.	→ **Post**	→ **Post**
Messaggio di stato	Messaggio di stato dell'utente.	→ **Post**	→ **Post**

Dizionario delle associazioni

Nome	Descrizione	Entità coinvolte	Attributi
Amicizia	Mette in relazione due utenti (un richiedente e un invitato) attraverso un'amicizia.	**RICHIEDENTE** (0,n), **INVITATO** (0,n)	**Stato**
U-F	Mette in relazione un utente con una frequenza presso una determinata scuola.	**UTENTE** (0,n), **FREQUENZA** (1,1)	
F-S	Mette in relazione una frequenza di un utente con una scuola.	**FREQUENZA** (1,1), **SCUOLA** (0,n)	
Pubblicazione	Pubblicazione dei post in bacheca da parte di un utente, mette quindi in relazione un utente con un determinato post.	**UTENTE** (0,n), **POST** (1,1)	
Invito	Invito all'evento organizzato dall'utente VIP.	**UTENTE** (0,n), **EVENTO** (0,n)	**Stato**
Tag	Associa ad una nota uno o più utenti in essa "taggati".	**NOTA** (0,n), **UTENTE** (0, n)	
Organizzazione	Associa ad un utente VIP l'evento da lui creato.	**VIP** (0,n), **EVENTO** (1, 1)	

1.2 Vincoli di dominio e implementazione in SQL (o soluzione alternativa)

<table>
<tr><td>1</td><td>

Un utente è VIP se il suo numero di amici è maggiore di 10 (il passaggio da uno stato all'altro avviene al momento del login).

Nella pagina *login.php* viene eseguita la seguente query al fine di verificare il numero di amicizie dell'utente loggato:

```
SELECT count(*)
FROM amicizia
WHERE(richiedente='$usr' OR invitato='$usr') AND stato='s';
```

A seconda del risultato viene in seguito aggiornata la tupla dell'utente loggato. Ad esempio in caso di `count(*)>=10` la query sarà:

```
UPDATE utente
SET vip = true
WHERE email = '$usr'
```

</td></tr>
<tr><td>2</td><td>

Un VIP può invitare un utente ad un evento solo se sono amici.

L'invito a un evento è gestito tramite la funzione

```
invito(IN inv VARCHAR(35), IN eve integer)
```

la quale verifica che l'organizzatore dell'evento e l'invitato siano amici e solo in quel caso aggiunge l'invito.

</td></tr>
<tr><td>3</td><td>

Un utente può "taggare" un altro utente in una nota solo se sono amici.

Come nel caso precedente abbiamo creato una funzione

```
ins_tag(IN taggante VARCHAR(35), IN taggato VARCHAR(35))
```

la quale verifica che l'utente taggante e l'utente taggato siano amici prima dell'inserimento del tag.

In ogni caso lato php l'azione del taggare in una nota è vincolata da un menù a tendina in cui si possono scegliere gli utenti da taggare tra i soli amici. Risulta quindi impossibile selezionare utenti che non lo siano.

</td></tr>
<tr><td>4</td><td>

Il tipo di evento può essere "concerto", "festa" o "sport".

Nella definizione della tabella evento abbiamo inserito il vincolo:

```
CONSTRAINT check_tipo CHECK (tipo::text = ANY (ARRAY['sport'::character
varying::text, 'concerto'::character varying::text, 'festa'::character
varying::text]))
```

</td></tr>
</table>

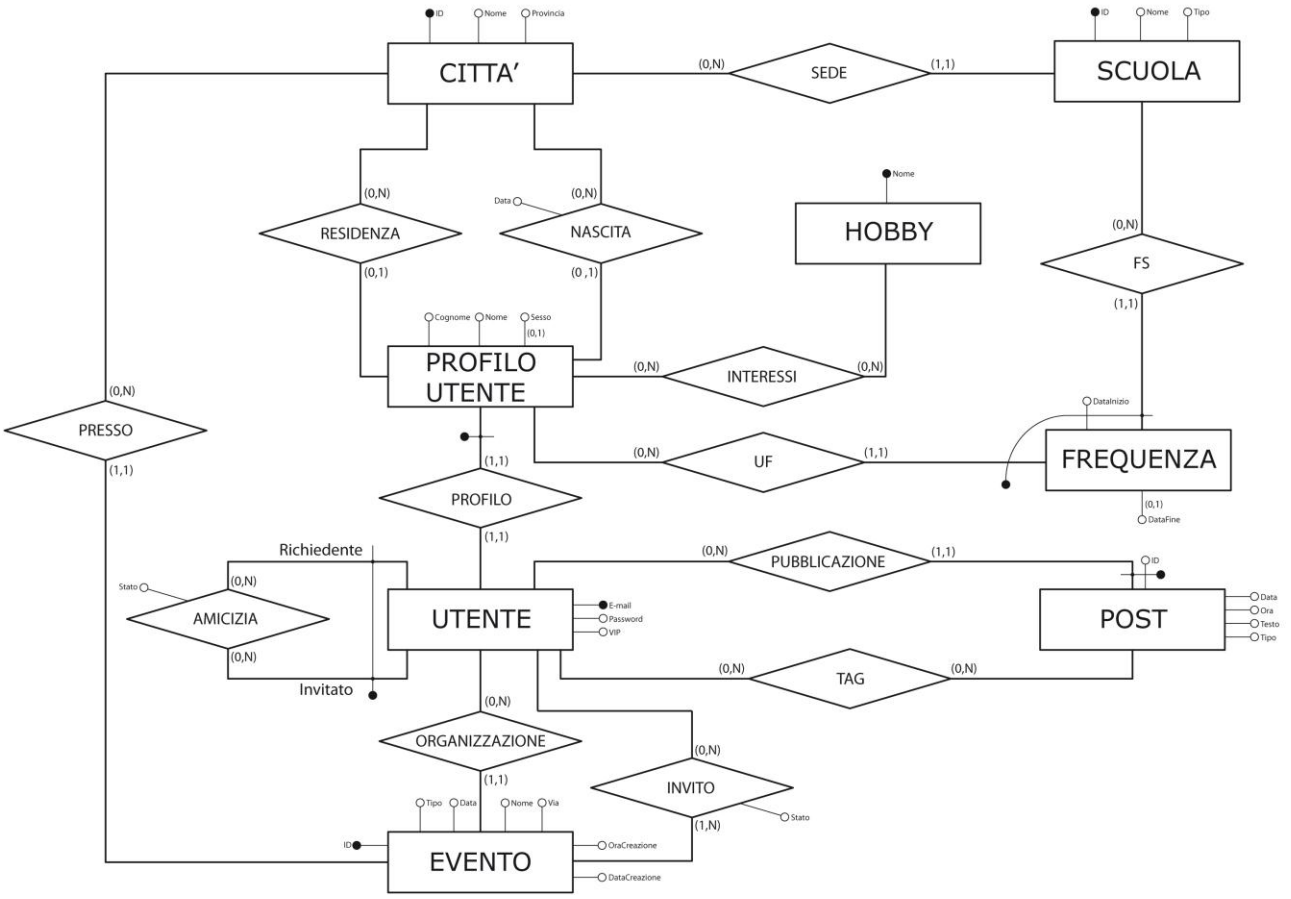
5	**Lo stato di conferma di un invito e di una richiesta di amicizia può essere: accettato (s), rifiutato (n) o in attesa (a).** Nella definizione della tabella amicizia abbiamo inserito il vincolo: `CONSTRAINT check_stato CHECK (stato::text = ANY (ARRAY['s'::character varying::text, 'n'::character varying::text, 'a'::character varying::text]))`

6	**Quando un evento viene creato, il relativo annuncio compare sia sulla bacheca del creatore che degli invitati.**
	
	Questo vincolo è stato implementato utilizzando la stessa soluzione adottata per la creazione delle note, le quali se associate a tag appaiono sia sulla bacheca di chi le ha create che sulla bacheca di chi è stato taggato. Tale soluzione prevede, in fase di visualizzazione della bacheca, una query che seleziona tutti i post dell'utente loggato più tutti i post in cui è stato taggato:
	
	`SELECT idpost, utente, data, ora, testo, tipo`
	`FROM post WHERE utente = '$usr'`
	`UNION`
	`SELECT idpost, utente, data, ora, testo, tipo`
	`FROM post`
	`WHERE (idpost, utente) IN (SELECT idpost, taggante`
	`FROM tag`
	`WHERE taggato = '$usr')`
	`ORDER BY data DESC, ora DESC"`

7	**È possibile aggiungere scuole non presenti.** Questo vincolo è stato implementato nella pagina *inserisciscuola.php* dove tramite la compilazione di un form e una INSERT è possibile aggiungere la nuova scuola nel db.
8	**La data iniziale di una frequenza deve essere antecedente alla date di fine ed entrambe le date non possono essere posteriori alla data di creazione.** Per il confronto tra data di inizio e data di fine abbiamo introdotto il vincolo: `CONSTRAINT anno CHECK (data_fine > data_inizio)` Per il controllo sulla correttezza della data inserita, abbiamo agito direttamente nella pagina *gest_profilo.php* utilizzando la funzione php `$date = date("Y-m-d")` la quale restituisce la data corrente.
9	**Un utente non può richiedere l'amicizia a un altro utente finchè la tupla non viene cancellata.** La possibilità di richiedere un'amicizia ad un altro utente è implementata attraverso l'interazione con un bottone nella pagina *home.php?action=ricerca.* Finchè l'amicizia non viene accettata o rifiutata tale pulsante resta inattivo. Inoltre, se la richiesta viene rifiutata, l'utente che ha inoltrato la richiesta di amicizia non troverà più l'utente a cui ha richiesto l'amicizia utilizzando il form di ricerca (a meno che quest'ultimo non cancelli tale utente dalla sua lista delle "Amicizie rifiutate").

10	**Il sesso di un utente può essere maschio (m) o femmina (f) o non specificato (null).** Abbiamo introdotto nel db il seguente vincolo: `CONSTRAINT sesso CHECK (sesso = ANY (ARRAY['m'::bpchar, 'f'::bpchar]))`
11	**Un utente per registrarsi al servizio deve essere maggiorenne (valore dell'età maggiore di 18).** Abbiamo effettuato un semplice controllo al momento della registrazione nella pagina *index.php.*

1.3 Schema ER ristrutturato



(presente in .pdf nella cartella del progetto)

Ristrutturazione dello schema ER

In seguito all'analisi abbiamo ristrutturato lo schema ER nel seguente modo:

- **Analisi delle ridondanze**
  Non sono presenti ridondanze.

- **Eliminazione delle generalizzazioni**
  Abbiamo risolto le generalizzazioni come segue:

  - L'entità VIP è stata accorpata nell'entità padre UTENTE. E' stato inserito un attributo booleano VIP all'utente che permette di distinguere l'utente normale dal VIP (numero di amici maggiore di dieci).

  - Le entità MSG DI STATO e NOTA sono state accorpate nell'entità POST alla quale è stato aggiunto un attributo *tipo* che ci permette di distinguerle. Questa scelta è stata dettata dal fatto che, ad eccezione della possibilità di "taggare" altri utenti in una nota, le due entità hanno le stesse caratteristiche.

- **Partizionamento/accorpamento delle entità**
  Nella ristrutturazione abbiamo partizionato/creato le seguenti entità:

  - UTENTE → PROFILO UTENTE e UTENTE: questa scelta ci ha permesso di dividere le informazioni relative agli utenti da quelle dei servizi messi a disposizione da *mini-facebook*.

  - CITTA': infatti durante la ristrutturazione ci siamo accorti che tale entità risultava fondamentale poiché richiamata numerose volte in altre entità**.**

  - L'attributo *hobby* è stato trasformato in una nuova entità in quanto multivalore**.**

- **Scelta degli identificatori principali**
  Abbiamo infine deciso di modificare gli identificatori principali le entità CITTA', SCUOLA ed EVENTO: tale scelta è giustificata dal fatto che che entrambe le entità sono spesso referenziate in numerose istanze di altre entità.

  Prendiamo il caso della CITTA' relativamente al PROFILO UTENTE.
  Poniamo quindi di voler memorizzare le informazioni relative a 10.000 utenti e che una città sia identificata mediante un nome (10 caratteri) e una provincia (2 caratteri). Analizziamo quindi i due casi possibili:

  - **senza l'identificatore ID**: per rappresentare tutte le città di residenza e di nascita dei profili di tutti gli utenti ho bisogno di: 12 caratteri * 2 città * 10.000 utenti = 240kb. Per rappresentare 100 città ho bisogno di 12 caratteri * 100 città = 1,2kb.
    Costo totale: 241,2 kb.

  - **con l'identificatore ID**: per rappresentare tutte le città di residenza e nascita di tutti gli utenti ho bisogno di 2 caratteri per la città quindi 10.000 * 2 * 2 = 40kb.
    Per contro devo memorizzare l'informazione circa l'ID della città che avrà un costo di (12 caratteri + 2 caratteri) * 100 città = 1,4kb.
    Costo totale: 41,4 kb.

  Consideriamo ora il caso dell'entità SCUOLA relativamente alla FREQUENZA e analizziamo i costi dei due casi possibili:

  - **senza l'identificatore ID**:
    per rappresentare tutte le scuole ho bisogno del nome della scuola (15 caratteri) più l'ID della città (2 caratteri). Ipotizzando di avere 100 scuole sarebbero necessari (15 + 2) * 100 = 1,7kb. Se in media ogni utente frequenta una scuola, per rappresentare tutte le frequenze serviranno: nome della scuola (15 caratteri) più ID della città (2 caratteri) * 10.000 = 170kb.
    Costo totale: 171,7kb.

  - **con l'identificatore ID**:
    per rappresentare tutte le scuole ho bisogno del nome della scuola più l'ID della città più l'ID della scuola quindi (15 + 2 + 2) * 100 = 1,9kb. Per rappresentare tutte le frequenze però ora mi bastano i due caratteri dell'ID quindi 10.000 * 2 = 20kb.
    Costo totale: 21,9kb.

L'ultimo caso da analizzare è quello relativo all'entità EVENTO. Anche qui vediamo le due possibilità:

- **senza l'identificatore ID:**
  per memorizzare le informazioni degli eventi che supponiamo essere circa 1000 servono il nome (15 caratteri) e l'organizzatore (25 caratteri) per un totale di 1000 * (15 + 25) = 40kb. Per rappresentare gli inviti, immaginando che ogni evento abbia una media di 5 inviti, servono la mail dell'invitato (25 caratteri), il nome dell'evento (15 caratteri) e la mail dell'organizzatore (25 caratteri). In totale si tratta di (25 + 15 + 25) * 5 * 1000 = 325kb.
  Costo totale: 365kb.

- **con l'identificatore ID**:
  in questo caso per memorizzare i 1000 eventi ho bisogno di 2 caratteri aggiuntivi per ogni ID, quindi 2kb in più rispetto al caso precedente ovvero 42kb. Per contro per memorizzare i 5000 inviti mi basteranno 135kb (cioè 2 caratteri dell'ID più 25 caratteri dell'invitato per 5000).
  Costo totale: 177kb.

1.4 Nuovi vincoli di dominio e implementazione in SQL (o soluzione alternativa)

<table>
<tr><td>1</td><td>

I post possono essere note (n) o messaggi di stato (m). Solo se sono di tipo nota è possibile taggare altri utenti (purché siano amici).

Nella definizione della tabella evento abbiamo inserito il vincolo:

```
CONSTRAINT check_tipo CHECK (tipo::text = ANY (ARRAY['n'::character
varying::text, m::character varying::text]))
```

L'inserimento dei tag esclusivamente associati alle note è vincolato grazie all'implementazione php del pulsante "gestisci tag" presente solo nel caso in cui il posto sia di tipo nota.

```
switch($row['tipo']){

case('n')
$str_bacheca .=
"<form
action='home.php?action=gestpost&tgnt=$utente&postid=$idpost'
method='post'>
 <input type='submit' name='tagga' value='Gestisci Tag'>
 <input type='submit' name='eliminapost' value='Elimina
Nota'></form><br/>";
 break;

case('m'):
$str_bacheca .=
"<form
action='home.php?action=gestpost&tgnt=$utente&postid=$idpost'
method='post'>
<input type='submit' name='eliminapost' value='Elimina
Messaggio'></form><br/>";
              break;
      }
```

</td></tr>
<tr><td>2</td><td>

Un evento può essere creato solo da un utente di tipo VIP.

Poiché il passaggio da utente normale e VIP avviene contestualmente al login, interrogare il db al momento della creazione dell'evento circa il numero di amici dell'utente non sarebbe significativo.
Per questo nella pagina php dedicata alla creazione dell'evento sarebbe necessario un controllo sulla variabile di sessione `$_SESSION['vip']` creata al login.

</td></tr>
</table>

Schema Relazionale

Utente (email, password, vip)

Amicizia (richiedente[utente], invitato[utente], stato)

Citta (IDcitta, nome, provincia)

Evento (IDevento, nome, organizzatore[utente], tipo, data, ora, via, luogo[citta,] ,data_creazione, ora_creazione)

Scuola (IDscuola, nome, tipo, sede[citta])

Profilo (utente[utente], nome, cognome, sesso, citta_residenza[citta], citta_nascita[citta], DataNascita)

Frequenza (studente[profilo], scuola[scuola], DataInizio, DataFine)

Hobby (nome)

Interessi (nome_hobby[hobby], utente[profilo])

Post (IDpost, utente[utente], data,ora, testo, tipo)

Invito (invitato[utente], stato, IDevento[evento])

Tag (taggante[post], IDpost[post], taggato[utente])

Forma Normale

Il modello rispetta la 1NF in quanti tutto gli attributi dello schema hanno domini atomici. Rispetta la 2NF in quanto ogni attributo non primo (quindi non appartenente a nessuna chiave) dipende completamente da ogni chiave. Non sono presenti dipendenze funzionali non banali.

Scelte di dettagli di implementazione php.

L'utente può *loggarsi* attraverso un form di registrazione oppure inserendo e-mail e password di una precedente registrazione. Nel caso in cui abbia dimenticato la password ha la possibilità di effettuare il recupero via e-mail.
Rispetto alle specifiche abbiamo reso obbligatorio anche il campo "data di nascita" al fine di proibire l'accesso ad utenti minorenni.

L'utente registrato ha la possibilità di vedere e modificare il proprio *profilo*, aggiungere nuove frequenze scolastiche scegliendo gli istituti da una lista di scuole messe a disposizione. Nel caso l'istituto cercato non fosse presente può essere inserito. Delle frequenze inserite l'utente può modificare solo la data di fine e non è possibile inserire nuove frequenze che si sovrappongano a frequenze passate nello stesso istituto. Nel caso in cui si voglia cambiare il nome della scuola è necessario cancellare la frequenza e inserirne una nuova. In ogni caso, la data di inizio di una frequentazione è sempre obbligatoria.

L'utente ha inoltre la possibilità di visualizzare la propria *bacheca* e di creare nuovi messaggi di stato o note nelle quali può taggare i propri amici. I nomi degli utenti taggati compaiono alla fine della nota mentre nella bacheca di ogni utente taggato compare solo il testo nella nota. L'utente taggato ha la possibilità di eliminare tale tag, il che comporta una cancellazione della nota dalla propria bacheca.

Le note nella bacheca degli amici sono "virtuali" in quanto le uniche note presenti nel db sono quelle dell'utente che le ha create. La nota presente nella bacheca dell'amico è quindi un semplice collegamento, fisicamente corrispondente ad una tupla della tabella 'tag'.

Tutti gli utenti possono vedere reciprocamente i propri profili, mentre per visualizzare le bacheche è necessario che la richiesta di amicizia sia stata accettata. Poiché la visualizzazione della bacheca degli amici è implementata mediante 'GET' la mail dell'amico è visibile nella barra degli indirizzi nel momento in cui si accede alla sua bacheca. Tuttavia se si prova a sostituire la sua mail con quella di un utente non amico, tale richiesta viene bloccata. Lo stesso controllo viene effettuato anche nella pagina dedicata alla modifica degli eventi creati.

Le *richieste di amicizia* sono gestite in una apposita pagina dove l'utente può accettare e rifiutare amicizie o eliminare richieste precedentemente rifiutate.

Ha inoltre la possibilità di *cercare* nuovi utenti nella sezione "Ricerca" tramite un apposito form. Tale ricerca è implementata utilizzando il costrutto *like* e restituisce una lista di utenti con alcune informazioni di base ai quali è possibile richiedere l'amicizia. Nel caso in cui tali amici abbiano già accettato l'amicizia è possibile visualizzare il loro profilo e la loro bacheca. Nel caso in cui la richiesta di amicizia sia già stata inviata ma non ancora accettata viene visualizzato un pulsante inattivo "richiesta in corso".
Inoltre, in tale ricerca, non è più possibile trovare gli utenti che hanno precedentemente rifiutato l'amicizia dell'utente che la effettua (a meno che non si elimini dall'apposita sezione il rifiuto della richiesta). Quindi se Alice chiede l'amicizia a Bob e Bob la rifiuta, Alice non potrà più trovare Bob attraverso la sezione ricerca (a meno che Bob non elimini il rifiuto nella sezione 'amici').

Nel form di ricerca è possibile utilizzare vari parametri (nome, cognome, sesso, ecc). Nel caso in cui si scelga di fare una ricerca che includa la città è necessario selezionale anche la provincia e viceversa.

La bacheca e il profilo degli amici sono anche raggiungibili attraverso la pagina dedicata alla gestione delle richieste di amicizia, ovviamente solo per quegli utenti che hanno accettato tale richiesta.

L'utente ha la possibilità di creare *eventi* a patto che sia VIP, in caso contrario il tasto di creazione degli eventi è disabilitato. Nella stessa sezione in cui gli eventi sono creati sono anche visualizzati. Gli eventi creati possono essere modificati, cancellati ed è ovviamente prevista una funzione per l'invito di amici.
Se un amico viene invitato ad un evento ed in un secondo momento l'amicizia viene revocata l'utente rimane invitato all'evento a meno che l'utente invitante non revochi anche l'invito.

Se un utente VIP crea un evento ed in seguito perde lo status di VIP non può più creare eventi ma può comunque ancora gestire quelli creati.

Gli *inviti* sono visualizzati in una sezione separata. L'utente invitato ad un evento vede le informazioni circa gli invitati solo se accetta l'invito. In caso contrario può vedere solo i dettagli generici. Egli può accettare o rifiutare l'invito ed eliminare un invito precedentemente rifiutato o accettato.

Contestualmente alla creazione di un invito sulla bacheca di chi ha creato un evento e su quella di chi è stato invitato compare un post con la segnalazione della creazione/invito. Tale post si contraddistingue dalle note e dai messaggi di stato per via del colore diverso. Inoltre sotto a tale post sono presenti dei link che conducono ai dettagli di creazione dell'evento (se sulla bacheca dell'organizzatore) o a quelli dell'invito (se sulla bacheca dell'invitato).

Ovunque siano presenti menù a tendina concettualmente vincolati (ad esempio Milano – MI) l'applicativo lascia all'utente la libertà di scegliere il valore desiderato; tuttavia nel caso egli scegliesse un valore non corretto (es. Milano – VR) si limita a segnalare l'errore. Si parte quindi dal presupposto che un utente nato o residente in una città ne conosca il nome e la sigla della provincia. Lo stesso discorso è valido anche nel caso della scelta delle scuole.

L'eliminazione dell'utente può essere effettuata nella sezione "modifica utente" solo nel caso in cui l'utente non abbia amicizie confermate o richieste di amicizia in sospeso.

In generale tutti i controlli sulla correttezza e validità dei dati inseriti sono stati implementati sia lato client che lato server attraverso opportuni vincoli specificati sul db.

In particolare, ovunque sia presente l'inserimento di testo attraverso caselle di testo, viene sempre effettuato un ulteriore controllo per il cross-site scripting (**XSS**) al fine di impedire l'inserimento di codice malevolo.

(Per la visione del progetto si consiglia l'utilizzo di Chrome come browser e dell'utente adalberto.muffa@gmail.com, psw:password).

Query SQL

1) Utente

a) Registrazione di un nuovo utente

```
INSERT INTO utente (email, psw, vip) VALUES
('alberto.camillo@mail.com','acami', false);

INSERT INTO profilo (utente, nome, cognome, sesso, citta_residenza,
citta_nascita, data_nascita)
VALUES
('alberto.camillo@mail.com','alberto','camillo','m','01','01','1985-01-
01')
```

b) Modifca del proflo di un utente.

```
INSERT INTO interessi (utente, nome_hobby) VALUES
('alberto.camillo@mail.com', 'tennis')

UPDATE profilo
SET citta_residenza = 1
WHERE utente = 'francesca.mad@gmail.com';
```

c) Richiesta di amicizia.

```
INSERT INTO  amicizia (richiedente, invitato, stato) VALUES
('alberto.camillo@mail.com','francesca.mad@gmail.com','a');
```

d) Accettazione/rifiuto di una richiesta di amicizia.

```
UPDATE amicizia SET stato = 's' WHERE richiedente =
'alberto.camillo@mail.com' AND invitato = 'francesca.mad@gmail.com';


UPDATE amicizia SET stato = 'n' WHERE richiedente =
'alberto.camillo@mail.com' AND invitato = 'francesca.mad@gmail.com';
```

e) Eliminazione di un utente (un utente può essere eliminato se non e' amico di nessuno).

```
DELETE FROM utente WHERE email = 'alberto.camillo@mail.com';
```

f) Visualizzazione della bacheca.

```
SELECT  utente, data, tipo, testo, taggato
FROM post p JOIN tag t ON utente = taggante AND p.idpost = t.idpost
WHERE tipo = 'n' AND utente = 'adalberto.muffa@gmail.com'
UNION
SELECT  utente, data, tipo, testo, taggato
FROM post p LEFT JOIN tag t ON p.idpost = t.idpost
WHERE tipo = 'm' AND utente = 'adalberto.muffa@gmail.com'
UNION
SELECT  organizzatore AS utente, data, tipo, nome, invitato
FROM evento JOIN invito ON organizzatore = 'adalberto.muffa@gmail.com'
UNION
SELECT  invitato AS utente, data, tipo, nome, organizzatore
FROM invito JOIN evento ON invitato = 'adalberto.muffa@gmail.com'
ORDER BY data;
```

Tale query seleziona:
- tutti post (note o messaggi di stato) creati dall'utente stesso;
- tutte le note in cui si è stati taggati;
- tutti gli eventi organizzati;
- tutti eventi a cui si è stati invitati.

g) Ricerca di utenti.

Es: nati a Napoli e frequentano o hanno frequentato l'universita.

```
SELECT utente, profilo.nome, cognome
FROM profilo JOIN citta ON citta_nascita = idcitta
WHERE citta.nome='napoli'
      AND provincia ='na'
      AND utente IN (SELECT studente
                     FROM frequenza JOIN scuola ON scuola = idscuola
                     AND tipo = 'universita');
```

2) Scuole.

a) Gestione della frequentazione di una scuola in un certo periodo.

(a1)Inserimento di una frequentazione.

```
INSERT INTO frequenza VALUES('alessia.guidi@gmail.com',4,'2003-05-05',
null);
```

(a2) Modifica di una frequentazione.

```
UPDATE frequenza
SET data_fine = '2004-01-01'
WHERE studente = 'alessia.guidi@gmail.com' AND
                  scuola = 4 AND data_inizio = '2003-05-05';
```

(a3) Visualizzazione di una frequentazione.

```
SELECT studente, nome, tipo, data_inizio, data_fine
FROM frequenza natural join scuola
WHERE studente='alessia.guidi@gmail.com' and scuola=idscuola;
```

(a4) Eliminazione di una frequentazione.

```
DELETE
FROM frequenza
WHERE studente='alessia.guidi@gmail.com' AND scuola = 4 AND data_inizio
= '2003-05-05';
```

b) Data una citta', restituire il numero di persone che vi sono nate, che vi abitano.

```
SELECT count(utente) as Milanesi
FROM profilo
WHERE citta_nascita = citta_residenza AND
       citta_residenza IN (SELECT idcitta
                           FROM citta
                           WHERE provincia = 'mi' AND nome = 'milano');
```

c) Scelta di una scuola dall'elenco delle scuole gia inserite.

```
SELECT  s.nome, tipo, c.nome, provincia
FROM scuola s JOIN citta c ON sede = idcitta
ORDER BY c.nome, tipo;
```

d) Inserimento di una scuola.

```
INSERT INTO scuola (nome, tipo, sede)
VALUES ('alessandro valenti','universita',get_idcitta('milano','mi'));
```

3. Post.

a)

(a1) Inserimento di un messaggio di stato

```
INSERT INTO post(idpost, utente, data, ora, testo) VALUES
('31','adalberto.muffa@gmail.com','2011-06-26','18:55','messaggio di
stato','m');
```

(a2) Rimozione.

```
DELETE FROM post WHERE utente = 'gatto.pixel@gmail.com' AND idpost = 1;
```

b) Inserimento di una nota più tag

```
INSERT INTO post(idpost, utente, data, ora, testo, tipo) VALUES
(22,'francesca.mad@gmail.com','2011-06-26','18:55','nota','n');

INSERT INTO TAG(taggante, taggato, idpost) VALUES
('francesca.mad@gmail.com','adalberto.muffa@gmail.com',22);
```

c) Eliminazione di un tag.

```
DELETE FROM tag
WHERE taggante = 'francesca.mad@gmail.com' AND idpost = 22 AND taggato
= 'adalberto.muffa@gmail.com';
```

4. Eventi.

a) Inserimento/rimozione di un EVENTO.

```
INSERT INTO evento (nome, organizzatore, tipo, data, via, luogo) VALUES
('compleanno',alberto.cami@gmail.com','festa','2012-2-2','via roma',2);

DELETE FROM evento WHERE idevento = 2;
```

b)

(b1) Inserimento di un invito

```
INSERT INTO invito('francesca.mad@gmail.com', 2);
```

(b2) Visualizza lo stato di un invito

```
SELECT invitato, stato
FROM invito
WHERE evento = 2 AND invitato = 'francesca.mad@gmail.com';
```

(b3) Accettazione/Rifiuto di un invito (es. dell'ultimo invito ricevuto)

```
UPDATE invito SET stato = 's'
WHERE invitato ='francesca.mad@gmail.com' AND evento = 02);

UPDATE invito SET stato = 'n'
WHERE invitato ='francesca.mad@gmail.com' AND evento = 02);
```

c) Visualizzazione gli utenti che hanno accettato/rifutato l'invito (es. dell'ultimo evento creato).

```
SELECT invitato AS accettato
FROM invito
WHERE evento = max_idevento() AND stato = 's';

SELECT invitato AS rifiutato
FROM invito
WHERE evento = max_idevento() AND stato = 'n';
```

5. Altre procedure di gestione (di base).

a) Elenco, dato un utente, dei suoi amici diretti che hanno la stessa eta e che vivono nella stessa citta.

```
SELECT nome, cognome
FROM amicizia JOIN profilo ON utente = richiedente AND invitato =
'francesca.mad@gmail.com' AND stato ='s'
WHERE (citta_nascita,extract(year from data_nascita)) IN
                  (SELECT citta_nascita, extract(year from data_nascita)
                   FROM profilo
                   WHERE utente = 'francesca.mad@gmail.com')
UNION
SELECT nome, cognome
FROM amicizia JOIN profilo ON utente = invitato AND richiedente =
'francesca.mad@gmail.com' AND stato ='s'
WHERE (citta_nascita,extract(year from data_nascita)) IN
                  (SELECT citta_nascita, extract(year from data_nascita)
                   FROM profilo
                   WHERE utente = 'francesca.mad@gmail.com');
```

L'unione serve per considerare sia quando l'utente è richiedente che invitato nell'amicizia. Successivamente si selezionano solo gli utenti che hanno in comune determinate caratteristiche, ovvero quelle racchiude nelle parentesi subito dopo la clausola `WHERE`.

b) Per ogni citta' trovare il numero di utenti che vi abitano.

```
SELECT get_nomecitta(citta_residenza), count(utente) AS Abitanti
FROM profilo
GROUP BY get_nomecitta(citta_residenza);
```

c) Trovare gli utenti che hanno un numero di amici superiore alla media.

```
SELECT u2.email
FROM utente u1, utente u2
WHERE (u1.email IN (SELECT richiedente
                    FROM amicizia
```

```
                    WHERE invitato = u1.email AND stato ='s'
                    )
    OR u1.email IN (SELECT invitato
                    FROM amicizia
                    WHERE richiedente = u2.email AND stato ='s'
                    ))
GROUP BY u2.email
HAVING count(*) > ( SELECT avg(amici.namici)
                    FROM(
                    SELECT count(*) AS namici
                    FROM utente u1, utente u2
                    WHERE u1.email IN
                                      (SELECT richiedente
                                       FROM amicizia
                                       WHERE invitato = u1.email
                                       AND stato ='s')
                       OR u1.email IN (SELECT invitato
                                       FROM amicizia
                                       WHERE richiedente = u2.email
                                       AND stato ='s')
                                       GROUP BY u2.email) num_amici);
```

Si calcola il numero di amici degli utenti con amicizie confermate (considerando quindi entrambi i casi: utente invitato e utente richiedente) e si calcola una media di tale numero.

Si calcolano nuovamente il numero di amici degli utenti con amicizie confermate ma questa volta si selezionano solo quelli con numeri di amici superiori alla media appena calcolata.

d) Restituire l'elenco degli utenti con le informazioni relative alle scuole che hanno frequentato, se ve ne sono.

```
SELECT p.nome, p.cognome, data_nascita, s.tipo, s.nome, c.nome,
data_inizio, data_fine
FROM ((profilo p JOIN frequenza ON p.utente = studente)
                     JOIN scuola s ON  scuola = s.idscuola)
                        JOIN citta c ON sede = c.idcitta;
```