UNIVERSITÀ DELLA CALABRIA
DIP. DI MATEMATICA E INFORMATICA

CORSO DI LAUREA IN INFORMATICA
A.A. 2019/20120

Progetto per il Corso di

BASI DI DATI

studente 161798 *ANGOTTI LUIGI*

Prof. P. Rullo

ESERCITATORI

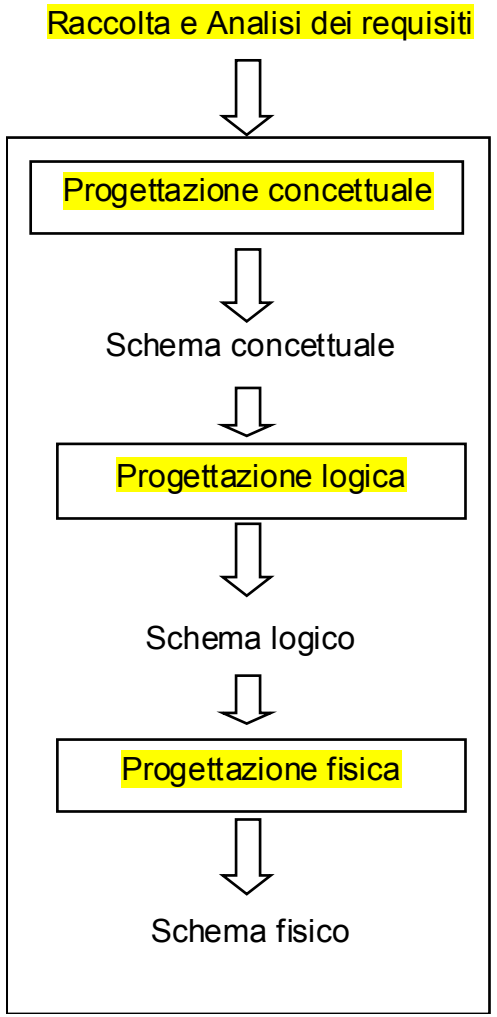
Ing. G. Laboccetta

Dott.ssa D. Angilica

DOCENTE

Introduzione

Le fasi di progettazione di una base di dati sono le seguenti:



1. Tematica Progettuale

La progettazione del sistema informatico in esame riguarda la base di dati per la gestione dei dati relative ad un centro commerciale.

2. Raccolta e Analisi dei Requisiti

L'attività che precede la progettazione vera e propria è la raccolta e l'analisi dei requisiti, che consiste nell'individuare i problemi che l'applicazione intende risolvere e le caratteristiche che tale applicazione dovrà avere.

2.1. Raccolta dei requisiti

In questa fase il progettista deve stabilire "quali dati il sistema informativo deve contenere e cosa deve fare il sistema per gestire questi dati". Questa attività viene realizzata tramite le seguenti attività:

1) interviste agli utenti del sistema
2) visione di documentazione interna
3) analisi di prodotti sw e base dati preesistenti che si intendono rimpiazzare e/o integrare.

Il risultato finale di questa attività consiste in un documento che descrive senza ambiguità e/o fraintendimenti ciò che l'applicazione deve risolvere. L'obiettivo finale consiste, dunque, in una descrizione "precisa" delle specifiche del problema in esame, in termini di dati del problema e operazioni di manipolazione su di essi.

	REQUISITI RICHIESTI
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18	Il social network Quarantine vuole rivedere il proprio database. In occasione dell'introduzione di nuove funzionalità, vuole approfittarne per rendere più efficiente il sistema. Ogni utente che si registra sul sito fornisce la propria anagrafica (nome, cognome, data di nascita), una username che lo identifica, un indirizzo e-mail che può essere usato per una sola registrazione e una password lunga almeno 8 e massimo 16 caratteri. Al momento della registrazione, l'utente decide se tenere il proprio profilo pubblico o privato (scelta che può essere modificata in seguito). In quest'ultimo caso solo utenti presenti nella sua lista di amici potranno visualizzare il suo profilo e ciò che viene pubblicato. Sul profilo si possono trovare solo post testuali, ognuno con una data, un orario, eventuali tag ed eventuali commenti (di chi lo ha postato o di altri utenti) ciascuno dei quali con data e orario. Se il profilo è privato, solo gli amici possono commentare. Il sistema, ogni giorno propone all'utente i post da egli pubblicati in quel giorno negli anni precedenti. Gli utenti possono scambiare tra loro anche dei messaggi privati. Ogni messaggio ha un testo, una data e un orario e si vuole sapere se è stato letto oppure no.

2.2. Analisi dei requisiti

Il documento di raccolta dei requisiti potrebbe presentare delle ambiguità e delle imprecisioni essendo stato scritto in un linguaggio non formale. Il linguaggio naturale, infatti, è ambiguo, essendo soggetto ad interpretazioni personali. L'obiettivo è quello di individuare tutte le ambiguità, i sinonimi e/o gli omonimi presenti, in modo da filtrare tutte le inesattezze derivanti da una strutturazione dei requisiti scritti in un linguaggio naturale. Le attività presenti in questa fase sono difficilmente standardizzabili; tuttavia esistono alcune regole pratiche che si possono seguire:

- **SCEGLIERE IL CORRETTO LIVELLO DI ASTRAZIONE:**
  - ➢ Non usare termini troppo generici o troppo specifici.
- **STANDARDIZZARE LA STRUTTURA DELLE FRASI:**
  - ➢ Usare sempre lo stesso stile sintattico.
- **EVITARE FRASI CONTORTE:**
  - ➢ Usare definizioni semplici e chiare.
- **INDIVIDUARE SINONIMI/OMONIMI E UNIFICARE I TERMINI:**
  - ➢ Sinonimi: termini diversi con lo stesso significato ⇨ unificare i termini.
  - ➢ Omonimi: termini uguali con significato diverso ⇨ termini diversi.
- **RENDERE ESPLICITO IL RIFERIMENTO TRA TERMINI:**
  - ➢ Esplicitare il riferimento (collegamento) tra termini.
- **COSTRUIRE UN GLOSSARIO DEI TERMINI:**
  - ➢ Per ogni termine una breve descrizione e possibili sinonimi.

2.2.1. Eliminazione delle Ambiguità

LINEA	TERMINE	SINONIMI	MOTIVAZIONE CORREZIONE
1	Social network	Rete sociale	Sinonimo di social network
1	Database	Basi di dati	Sinonimo di database

13	tag	Tag ad utente	Ambiguità con “hashtag”

2.2.2. Ristrutturazione dei Requisiti Richiesti

	SPECIFICHE RISTRUTTURATE
1 2 3 4 5 6 7 8	La rete sociale Quarantine vuole rivedere la propria base di dati. In occasione dell’introduzione di nuove funzionalità, vuole approfittarne per rendere più efficiente il sistema. Ogni utente che si registra sul sito fornisce la propria anagrafica (nome, cognome, data di nascita), una username che lo identifica, un indirizzo e-mail che può essere usato per una sola registrazione e una password lunga almeno 8 e massimo 16 caratteri. Al momento della registrazione, l’utente decide se tenere il proprio profilo pubblico o privato (scelta che può essere modificata in seguito). In

2.2.3. Raffinamento delle Specifiche e Individuazione dei Concetti di Base

Dopo aver eliminato tutte le fonti di ambiguità presenti nel testo, si può eseguire un'analisi più approfondita dei requisiti al fine di individuare i concetti di base, ossia le entità coinvolte nella realtà di interesse. I requisiti del sistema vengono, pertanto, suddivisi in ulteriori sottoparti ognuna delle quali raggruppa le informazioni relative ad una specifico concetto.

	FRASI DI CARATTERE GENERALE
1-3	La rete sociale Quarantine vuole rivedere la propria base di dati. In occasione dell'introduzione di nuove funzionalità, vuole approfittarne per rendere più efficiente il sistema.

	FRASI RELATIVE AI COMMENTI
12-14	commenti (di chi lo ha postato o di altri utenti) ciascuno dei quali con data e orario

	FRASI RELATIVE AGLI UTENTI
4-7	Ogni utente che si registra sul sito fornisce la propria anagrafica (nome, cognome, data di nascita), una username che lo identifica, un indirizzo e-mail che può essere usato per una sola registrazione e una password lunga almeno 8 e massimo 16 caratteri.
8-11	l'utente decide se tenere il proprio profilo pubblico o privato (scelta che può essere modificata in seguito). In quest'ultimo caso solo utenti presenti nella sua lista di amici potranno visualizzare il suo profilo e ciò che viene pubblicato.

Progettazione Concettuale

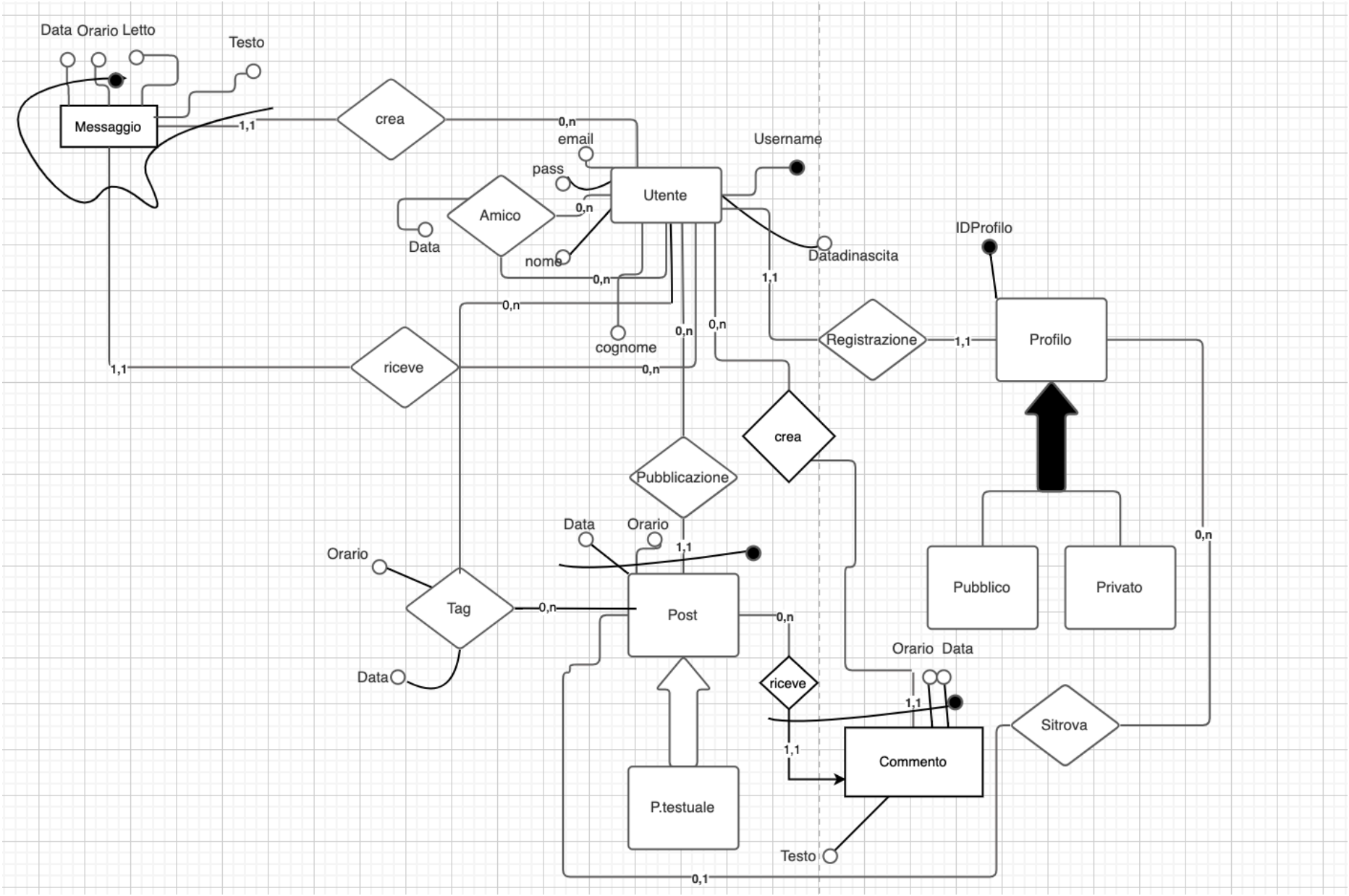
1. Schemi E-R

	FRASI RELATIVE A TAG
11-12	Sul profilo si possono trovare solo post testuali, ognuno con una data, un orario, eventuali tag ad utente

	FRASI AMICI
8-11	l'utente decide se tenere il proprio profilo pubblico o privato (scelta che può essere modificata in seguito). In quest'ultimo caso solo utenti presenti nella sua lista di amici potranno visualizzare il suo profilo
14	Se il profilo è privato, solo gli amici possono commentare.

2.3. Specifica delle Operazioni sui dati previste

1. Aggiornare il profilo dell'utente 'Tizio' e renderlo pubblico.
2. Trovare gli amici dell'utente 'Caio'.
3. Trovare i profili pubblici con meno di 50 amici.
4. Trovare i post dei profili pubblici con il minor numero di commenti.
5. Trovare per ogni profilo i post con il più alto numero di commenti.
6. Controllare che se un profilo è privato allora solo un amico può commentare



2.4. Documentazione dello Schema E-R

2.4.1. Dizionario dei Dati

ENTITÀ	IDENTIFICATORE	ATTRIBUTI	DESCRIZIONE
utente	username	data_nascita,cognome,nome, username,mail,password	Contiene le informazioni di ogni utente
Profilo	IdProfilo		
profilo privato			Verifica se profilo è private
profilo pubblico			Verifica se profilo è pubblico
commento	post,username,data,orario	post,username,data,orario,Testo	Contiene un commento testuale
post	usernameUtente,data,orario	usernameUtente,data,orario	Contiene un post
posttestuale			Contiene un post testuale
Messaggio	usernamemitt,usernamedest,data,orario	usernamemitt,usernamedest,data,orario,testo,letto	Contiene le informazioni dei messaggi privati

2.4.2. Descrizione Entità

ENTITA' utente			
DESCRIZIONE	Rappresenta l'entità dell'account utente che descrive ogni singolo utente		
NOME ATTRIBUTO	**TIPO DI DATO**	**VINCOLO**	**DESCRIZIONE**
Cognome	Alfanumerico	Obbligatorio	Cognome dell'utente
Nome	Alfanumerico	Obbligatorio	Nome dell'utente

Username	Alfanumerico	Obbligatorio	Username dell'utente e che lo identifica univocamente
Mail	Alfanumerico	Unico e obbligatorio	Mail che identifica univocamente un profilo
password	Alfanumerico	Obbligatorio	Parola chiave dell'utente
datadinascita	date		data di nascita utente

ENTITA' commento			
DESCRIZIONE			
NOME ATTRIBUTO	**TIPO DI DATO**	**VINCOLO**	**DESCRIZIONE**
Orario	Date		ora del commento
Data	Time		data commento
Testo	alfanumerico		testo contenuto nel commento

ENTITA' Profilo			
DESCRIZIONE			
NOME ATTRIBUTO	**TIPO DI DATO**	**VINCOLO**	**DESCRIZIONE**
IdProfilo	alfanumerico		descrive ed identifica l'entità Profilo

ENTITA' MESSAGGIO			
DESCRIZIONE			
NOME ATTRIBUTO	**TIPO DI DATO**	**VINCOLO**	**DESCRIZIONE**
Data	date		data invio messaggio
Orario	time		orario invio messaggio
Testo	alfanumerico		contenuto testuale messaggio
Letto	boolean		indica se il mess è stato letto

ENTITA' Post			
DESCRIZIONE			
NOME ATTRIBUTO	**TIPO DI DATO**	**VINCOLO**	**DESCRIZIONE**

Data	date		data del post
Ora	time		orario post

2.4.3. Descrizione Relazioni

Tag		
DESCRIZIONE	SERVE PER INDICARE UN MESSAGGIO PRIVATO	
Entità Coinvolte		
ENTITÀ	CARDINALITÀ	
UTENTE	(0, N)	
POST	(0,N)	
ATTRIBUTI		
NOME	TIPO DIDATI	DESCRIZIONE

DATA	DATE	data del tag
ORARIO	TIME	Orario del tag

AMICO		
DESCRIZIONE	SERVE PER INDICARE UN MESSAGGIO PRIVATO	
Entità Coinvolte		
ENTITÀ	CARDINALITÀ	
UTENTE	(0, N)	
UTENTE	(0,N)	
ATTRIBUTI		
NOME	TIPO DIDATI	DESCRIZIONE

DATA	DATE	data Amicizia

SITROVA	
DESCRIZIONE	Possiamo capire se il post è riferito a qualche utente oppure no
Entità Coinvolte	
ENTITÀ	CARDINALITÀ
POST	(0,1)
PROFILO	(0,N)

REGISTRAZIONE	
DESCRIZIONE	INDICA CHE PER LA REGISTRAZIONE è EFFETTUABILE DA UN UTENTE SPECIFICO
Entità Coinvolte	

ENTITÀ	CARDINALITÀ
UTENTE	(1,1)
PROFILO	(1,1)

Crea	
DESCRIZIONE	Possiamo capire se il post è riferito a qualche utente oppure no
Entità Coinvolte	
ENTITÀ	CARDINALITÀ
UTENTE	(0,N)
MESSAGGI	(1,1)

RICEVE	
DESCRIZIONE	Possiamo capire se il post è riferito a qualche utente oppure no
Entità Coinvolte	
ENTITÀ	CARDINALITÀ
UTENTE	(0,N)
MESSAGGI	(1,1)

RICEVECommento	
DESCRIZIONE	Possiamo capire se il post è riferito a qualche utente oppure no
Entità Coinvolte	
ENTITÀ	CARDINALITÀ
UTENTE	(0,N)
COMMENTI	(1,1)

2.4.4. Vincoli non espressi dallo schema E/R

REGOLE DI VINCOLO
Email deve essere valida ed unica per ogni utente
Lunghezza password deve essere compresa tra 8 e 16 caratteri e deve essere criptata
Se profilo privato solo gli amici possono commentare o vedere profilo

4. Progettazione Logica

L'obiettivo della fase di progettazione logica è quello di "tradurre" lo schema concettuale, prodotto in fase di progettazione concettuale, in uno schema logico che rappresenti gli stessi dati in maniera corretta ed efficiente.
La progettazione logica si articola in due fasi:

- Ristrutturazione del modello Entità-Relazione: è una fase indipendente dal modello logico scelto e si basa su criteri di ottimizzazione dello schema;
- Traduzione verso il modello logico: fa riferimento ad un particolare modello logico e può includere una ulteriore ottimizzazione.

Il modello logico utilizzato in questo progetto didattico è il Modello **Relazionale**.

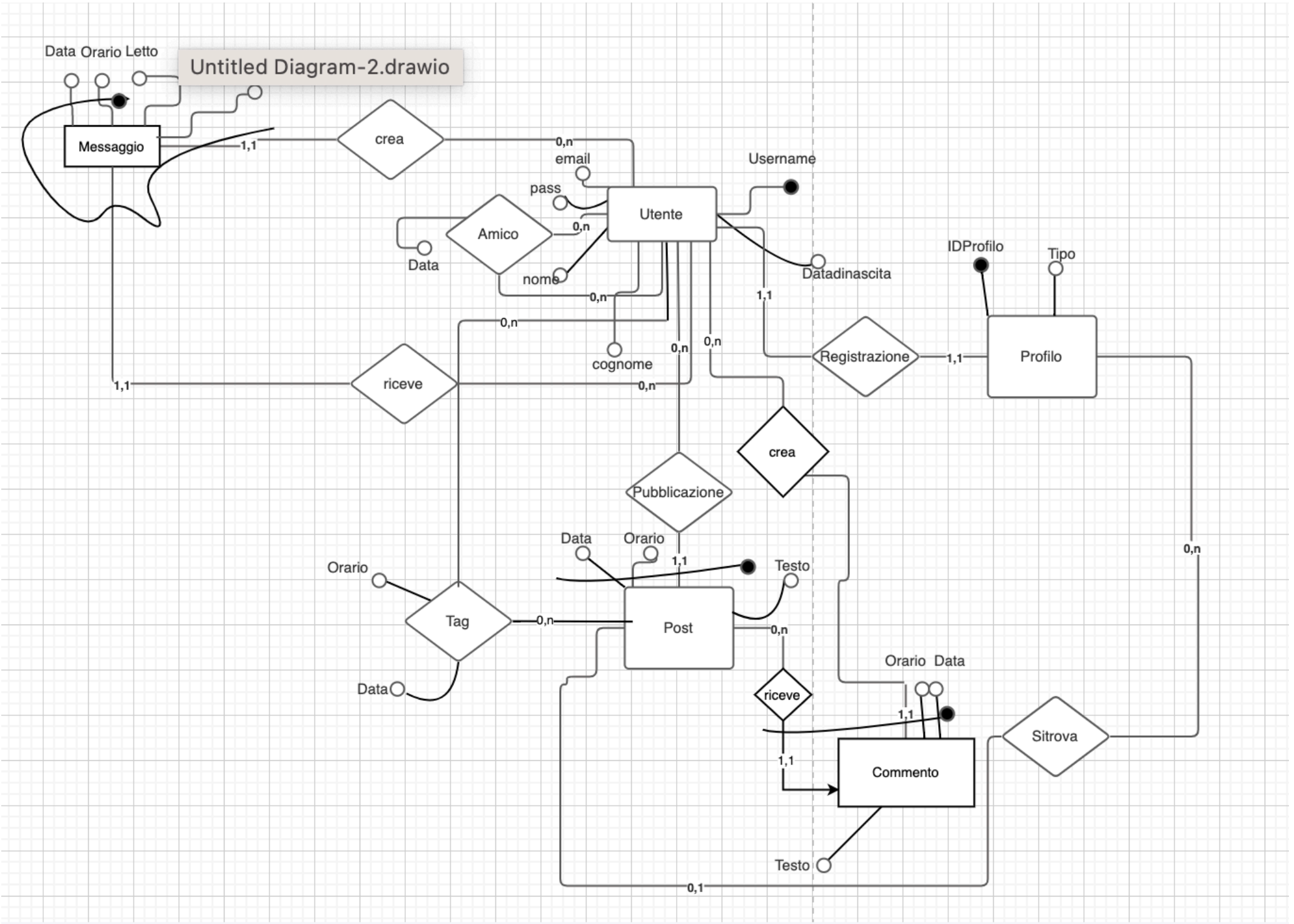
1. RISTRUTTURAZIONE DELLO SCHEMA E-R

Si articola in quattro passi:

- Analisi delle ridondanze
- Eliminazione delle generalizzazioni
- Partizionamento/accorpamento di entità e relazioni
- Scelta degli identificatori primari

La fase di ristrutturazione dello schema E-R ottenuto nella fase di progettazione concettuale è necessaria perché non tutti i costrutti del modello E-R sono direttamente rappresentabili nel modello logico utilizzato, ossia nel modello relazionale. In particolare, il problema si pone per le generalizzazioni presenti nello schema. Occorre documentare opportunamente le scelte adottate.

Figura 2- Schema E-R ristrutturato


Untitled Diagram-2.drawio
Data Orario Letto
Messaggio
1,1
crea
0,n
email
pass
Utente
Username
Amico
0,n
Data
nome
Datadinascita
IDProfilo
Tipo
0,n
1,1
0,n
cognome
0,n
0,n
1,1
riceve
0,n
Registrazione
1,1
Profilo
crea
Pubblicazione
Data
Orario
1,1
Testo
Orario
Tag
0,n
Post
0,n
Data
Orario Data
riceve
1,1
Sitrova
0,n
1,1
Commento
Testo
0,1

2. DOCUMENTAZIONE DELLO SCHEMA LOGICO

2.1. DESCRIZIONE ENTITÀ

ENTITA' Profilo			
DESCRIZIONE			
NOME ATTRIBUTO	**TIPO DI DATO**	**VINCOLO**	**DESCRIZIONE**
IdProfilo	alfanumerico		descrive ed identifica l'entità Profilo
Pubblico	boolean		specifica se l'id è pubblico o privato

ENTITA' Post			
DESCRIZIONE			
NOME ATTRIBUTO	**TIPO DI DATO**	**VINCOLO**	**DESCRIZIONE**
Data	date		data del post
Ora	time		orario post
Testo	alfanumerico		testo del post

3. TRADUZIONE DELLO SCHEMA E-R

4.1. Traduzione Entità

UTENTE(username, email, password, nome, cognome, dataNascita, idProfilo*)

con vincolo di integrità referenziale tra l'attributo id profilo e la relazione profilo

COMMENTO(usernameUtentePost*, dataPost*, orarioPost*, usernameUtenteCommentante*, data, orario, testo)

con vincolo di integrità referenziale tra gli attributi usernameUtentePost , dataPost,orarioPost e le relazione post e tra l'attributo usurnameUtentecommentante e la relazione utente

MESSAGGIO(usernameMittente*, usernameDestinatario*, data, orario, testo, letto)

con vincolo di integrità referenziale tra l'attributo usurnameMittente e la relazione utente

con vincolo di integrità referenziale tra l'attributo usurnameDestinatario e la relazione utente

PROFILO(idProfilo,pubblico)

Post(usernameUtente*, data, orario, testo)

con vincolo di integrità referenziale tra l'attributo usernameUtente e la relazione utente

POST(USERNAMEUTENTE*, DATA, ORARIO, TESTO)

con vincolo di integrità referenziale tra l'attributo usernameUtente e la relazione utente

4.2. Traduzione Relazioni

AMICO(USERNAME1*, USERNAME2*, DATAINIZIOAMICIZIA)

con vincolo di integrità referenziale tra l'attributo username1 e la relazione utente

con vincolo di integrità referenziale tra l'attributo username2 e la relazione utente

SITROVA(USERNAMEUTENTEPOST*, DATAPOST*, ORARIOPOST*, IDPROFILO*)

con vincolo di integrità referenziale tra gli attributi usernameutentepost,datapost,orario post e la relazione post

con vincolo di integrità referenziale tra l'attributo idrofilo e la relazione profilo

TAG(USERNAMEUTENTEPOST*, DATAPOST*, ORARIOPOST*, USERNAMEUTENTETAGGATO*, DATA, ORARIO)

con vincolo di integrità referenziale tra gli attributi usernameutentepost,datapost,orario post e la relazione post
con vincolo di integrità referenziale tra l'attributo usernameutentetaggato e la relazione utente

5. Progettazione Fisica

5.1.Definizione dello schema della base di dati

```
CREATE TABLE `Q`.`post` (
  `usernameutente` VARCHAR(45) NOT NULL,
  `Data` DATE NOT NULL,
  `Orario` TIME NOT NULL,
  `testo` VARCHAR(45) NULL,
  PRIMARY KEY (`usernameutente`, `Data`, `Orario`),
  CONSTRAINT `usernameutentepost_fk`
    FOREIGN KEY (`usernameutente`)
    REFERENCES `Q`.`utente` (`username`)
    ON DELETE NO ACTION
```

Creazione tabella Profilo

```
CREATE TABLE `Q`.`profilo` (
  `idprofilo` VARCHAR(45) NOT NULL,
  `pubblico` TINYINT(1) NULL,
  PRIMARY KEY (`idprofilo`));
```

5.2. Definizione delle interrogazioni per la visualizzazione dei dati

```
CREATE TABLE `Q`.`sitrova` (
  `usernameutentepost` VARCHAR(45) NOT NULL,
  `DataPost` DATE NOT NULL,
  `OrarioPost` TIME NOT NULL,
  `idProfilo` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`usernameutentepost`, `DataPost`, `OrarioPost`, `idProfilo`),
  INDEX `idProfilo_fk_idx` (`idProfilo` ASC) VISIBLE,
  CONSTRAINT `post_fk`
    FOREIGN KEY (`usernameutentepost` , `DataPost` , `OrarioPost`)
    REFERENCES `Q`.`post` (`usernameutente` , `Data` , `Orario`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `idProfiloSitrova_fk`
    FOREIGN KEY (`idProfilo`)
    REFERENCES `Q`.`profilo` (`idprofilo`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
```

```
CREATE TABLE `Q`.`tag` (
  `usernameUtentePost` VARCHAR(45) NOT NULL,
  `DataPost` DATE NOT NULL,
  `OrarioPost` TIME NOT NULL,
  `usernameUtenteTaggato` VARCHAR(45) NOT NULL,
  `Data` DATE NULL,
  `Orario` TIME NULL,
  PRIMARY KEY (`usernameUtentePost`, `DataPost`, `OrarioPost`, `usernameUtenteTaggato`),
  INDEX `usernameUtenteTaggato_idx` (`usernameUtenteTaggato` ASC) VISIBLE,
  CONSTRAINT `Post_tag_fk`
    FOREIGN KEY (`usernameUtentePost` , `DataPost` , `OrarioPost`)
    REFERENCES `Q`.`post` (`usernameutente` , `Data` , `Orario`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `usernameUtenteTaggato`
    FOREIGN KEY (`usernameUtenteTaggato`)
    REFERENCES `Q`.`utente` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
```

```
CREATE TABLE `Q`.`messaggio` (
  `usernameMittente` VARCHAR(45) NOT NULL,
  `usernameDestinatario` VARCHAR(45) NOT NULL,
  `Data` DATE NOT NULL,
  `Orario` TIME NOT NULL,
  `testo` VARCHAR(1000) NULL,
  `Letto` TINYINT(1) NULL,
  PRIMARY KEY (`usernameMittente`, `usernameDestinatario`
  INDEX `usernameDestinatario_fk_idx` (`usernameDestinatar
  CONSTRAINT `usernameMittente_fk`
    FOREIGN KEY (`usernameMittente`)
    REFERENCES `Q`.`utente` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `usernameDestinatario_fk`
    FOREIGN KEY (`usernameDestinatario`)
    REFERENCES `Q`.`utente` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
```

```
CREATE TABLE `Q`.`Segue` (
  `username1` VARCHAR(45) NOT NULL,
  `username2` VARCHAR(45) NOT NULL,
  `DataInizioFollow` DATE NULL,
  PRIMARY KEY (`username1`, `username2`),
  INDEX `usernamefollow2_fk_idx` (`username2` AS
  CONSTRAINT `usernamefollow1_fk`
    FOREIGN KEY (`username1`)
    REFERENCES `Q`.`utente` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `usernamefollow2_fk`
    FOREIGN KEY (`username2`)
    REFERENCES `Q`.`utente` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
```

```
CREATE TABLE `Q`.`amico` (
  `username1` VARCHAR(45) NOT NULL,
  `username2` VARCHAR(45) NOT NULL,
  `DataInizioAmicizia` DATE NULL,
  PRIMARY KEY (`username1`, `username2`),
  INDEX `username2_fk_idx` (`username2` ASC) VISIBLE,
  CONSTRAINT `username1_fk`
    FOREIGN KEY (`username1`)
    REFERENCES `Q`.`utente` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `username2_fk`
    FOREIGN KEY (`username2`)
    REFERENCES `Q`.`utente` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
DROP TRIGGER IF EXISTS `Q`.`Amico_BEFORE_INSERT`;
```

```
CREATE TABLE `Q`.`commento` (
  `usernameUtentePost` VARCHAR(45) NOT NULL,
  `dataPost` DATE NOT NULL,
  `orarioPost` TIME NOT NULL,
  `usernameUtenteCommentante` VARCHAR(45) NOT NULL,
  `data` DATE NOT NULL,
  `orario` TIME NOT NULL,
  `testo` VARCHAR(1000) NULL,
  PRIMARY KEY (`usernameUtentePost`, `dataPost`, `orarioPost`, `usernameUtenteCommentante`,
`data`, `orario`),
  CONSTRAINT `postCommento_fk`
    FOREIGN KEY (`usernameUtentePost`, `dataPost` , `orarioPost`)
    REFERENCES `Q`.`post` (`usernameUtente` , `data` , `orario`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `usernameUtenteCommentante_fk`
    FOREIGN KEY (`usernameUtentePost`)
    REFERENCES `q`.`utente` (`username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
```

Creazione tabella

```
CREATE TABLE `Q`.`utente` (
  `username` VARCHAR(45) NOT NULL,
  `email` VARCHAR(45) NULL,
  `password` VARCHAR(45) NULL,
  `nome` VARCHAR(45) NULL,
  `cognome` VARCHAR(45) NULL,
  `dataNascita` DATE NULL,
  `idProfilo` VARCHAR(45) NULL,
  PRIMARY KEY (`username`),
  INDEX `idProfilo_fk_idx` (`idProfilo` ASC) VISIBLE,
  CONSTRAINT `idProfilo_fk`
    FOREIGN KEY (`idProfilo`)
    REFERENCES `Q`.`profilo` (`idprofilo`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
DROP TRIGGER IF EXISTS `Q`.`utente_BEFORE_INSE
```

Query1

```
# Rendo pubblico il profilo se l'id è l'id dell'utente tizio.
UPDATE PROFILO SET pubblico=TRUE WHERE idProfilo IN (
    SELECT  U.idProfilo
    FROM    UTENTE U
           WHERE U.nome='Tizio'
);
```

Query2

```
#Se utilizzo solo una tabella utente ottengo le username degli utenti che sono Caio ed anche amici di caio
Select UA.idProfilo
From Amico A, Utente UA,Utente U
Where A.username2=UA.username and A.username1=U.username and  U.nome='Caio'

UNION

Select UA.idProfilo
From Amico A, Utente UA, Utente U
Where A.username1=UA.username and A.username2=U.username and  U.nome='Caio';
```

T

```sql
SELECT  NumAmici.idProfilo
FROM    (
# Prendo l'id profilo della persona pubblica la cui username compare nella prima tabella di amico   ( A.username1=U. username) insieme al suo numero di amici

    select  P.idProfilo,count(*) as nAmici
    from    Utente U, Profilo P, Amico A
    where   A.username1=U.username and U.idProfilo=P.idProfilo and P.pubblico=true
    Group by P.idProfilo

    UNION all

    select  P.idProfilo,count(*) as nAmici
    from    Utente U, Profilo P, Amico A
    where   A.username2=U.username and U.idProfilo=P.idProfilo and P.pubblico=true
    Group by P.idProfilo

    Union all
    select  P.idProfilo, 0
    from    Utente U, Profilo P
  #  Se non esiste una tupla in AMICO che ha come username 1 o come username2 la username del profilo fissato

where  U.idProfilo=P.idProfilo and P.pubblico = true and not exists
                    (   select  *
                        From    Amico A
                        where   A.username1=U.username or A.username2=U.username)   ) as NumAmici

#Infine ottengo una tabella virtuale che è composta da idProfilo e numero amici. In numAmici potrebbero esserci due tuple con lo stesso id Profilo
#Id1   4. La prima tupla deriva dal fatto che la username di id1 è presente 4 volte in amico come username1
#Id1   3 La  seconda tupla deriva dal fatto che la username di id1 è presente 3 volte in amico come username2


   group by     NumAmici.idProfilo
   having       sum(NumAmici.namici) <50;
```

```sql
#Numero commenti dei post dei profili pubblici
create view numCommenti (usernameUtentePost,dataPost,orarioPost,numerocommenti) as
    select  usernameUtentePost,dataPost,orarioPost,count(*) as numerocommenti
    From    Commento C, Utente U, Profilo P
    Where   C.usernameUtentePost=U.username and U.idProfilo=P.idProfilo and P.pubblico=True
    Group By usernameUtentePost,dataPost,orarioPost

    UNION ALL
       # (Prendo in considerazione I post con 0 commenti sempre dei profili pub)
    Select  P.usernameUtente as usernameUtentePost , P.data as dataPost , P.orario as orarioPost, 0
#Se non esiste un tapla in commento che contiene il post pubblico fissato
from    Post P, Utente U, Profilo Pr
    where   P.usernameUtente=U.username and U.idProfilo=Pr.idProfilo and Pr.pubblico=True and
            not exists (    select *
                            From    Commento C
                            where   C.usernameUtentePost=P.usernameUtente and C.dataPost=P.data and C.orarioPost=P.orario)


            # un post che è preso dalla vista lo restituisco se ha un numero commmenti  = al minimo globale
Select  NC.usernameUtentePost, NC.dataPost, NC.orarioPost
From    NumCommenti NC
Where   NC.numerocommenti = (   Select  min(numerocommenti)
                    From    NumCommenti         );

DROP VIEW NumCommenti;
```

```sql
#Qua prendo per ogni post il numero commenti
create view numCommenti (idProfilo,usernameUtentePost,dataPost,orarioPost,numerocommenti) as
  select    P.idProfilo, C.usernameUtentePost, C.dataPost, C.orarioPost, count(*) as numerocommenti
    From    Commento C, Utente U, Profilo P
    Where   C.usernameUtentePost=U.username and U.idProfilo=P.idProfilo
    Group By usernameUtentePost,dataPost,orarioPost

    UNION ALL
     #   Se non esiste un tapla in commento che contiene il post fissato(faccio questo perché è probabile che tutti post di un profilo abbiano 0 commenti)
    Select  Pr.idProfilo, P.usernameUtente as usernameUtentePost , P.data as dataPost , P.orario as orarioPost, 0
    from    Post P, Utente U, Profilo Pr
    where   P.usernameUtente=U.username and U.idProfilo=Pr.idProfilo and
            not exists (    select *
                            From    Commento C
                            where   C.usernameUtentePost=P.usernameUtente and C.dataPost=P.data and C.orarioPost=P.orario);

#per ogni profilp trovo il massimo numero commenti trai suoi post
create view ProfiloMaxCommenti(idProfilo, numMaxCommenti) as
    select idProfilo, max(numerocommenti) as numMaxCommenti
    from   NumCommenti
Group By idProfilo;


  select    NC.idProfilo, NC.usernameUtentePost,NC.dataPost,NC.orarioPost
  from  numCommenti NC, ProfiloMaxCommenti PMC
  where NC.idProfilo=PMC.idProfilo and NC.numeroCommenti=PMC.numMaxCommenti;

  # NC              PMC             RIS
  # a p1 3          a 4             a p2 4
  # a p2 4          b 7             b p3 7
  # a p5 4                          a p5 4
  # b p3 7
  # b p4 0
  drop view numCommenti ;
drop view ProfiloMaxCommenti;
```

TRIGGER AGGIUNTIVI

```
#CONTROLLARE CHE SE UN PROFILO è PRIVATO ALLORA SOLO UN AMICO PUO COMMENTARE
CREATE DEFINER = CURRENT_USER TRIGGER `Q`.`commento_BEFORE_INSERT` BEFORE INSERT ON `commento` FOR EACH ROW
BEGIN
#(se l'utente del post= utente post commentante è possibile perché è la stessa persona)
#Mi chiedo se utente del post è divrerso dall'utente commentante AND se l'utente del post ha il profilo privato AND verifico se inf
    if (new.usernameUtentePost<>new.usernameUtenteCommentante
        and false=(select pubblico from profilo p, utente u where u.idProfilo=p.idProfilo and u.username=New.UsernameUtentePost)
        and new.usernameUtenteCommentante not in (select username1 from Amico where username2=new.usernameUtentePost)
        and new.usernameUtenteCommentante not in (select username2 from Amico where username1=new.usernameUtentePost)   ) then
        signal sqlstate '45000'
        set message_text = 'Non sono amici.';
    end if;

END
```

EVENTI

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `postAnniPrecedenti`()
BEGIN
    select  pr.idProfilo, p.*
    from    profilo pr, utente u, post p
    where   pr.idProfilo=u.idProfilo and u.username=p.usernameUtente and
            day(now())=day(p.data) and month(now()) = month(p.data) and year(now()) > year(p.data);
END
```

TRIGGER PER LUNGHEZZA PASSWORD

```
DELIMITER $$
USE `Q`$$
CREATE DEFINER=`root`@`localhost` TRIGGER `utente_BEFORE_INSERT` BEFORE INSERT ON
`utente` FOR EACH ROW BEGIN
        declare minPwdLength int unsigned;
  declare maxPwdLength int unsigned;
  declare pwdLength int unsigned;
  set minPwdLength = 8;
  set maxPwdLength = 16;
  set pwdLength = (select length(new.password));
  if (pwdLength < minPwdLength) then
    signal sqlstate '45000'
      set message_text = 'Password breve (meno di 8 caratteri).';
  end if;
  if (pwdLength > maxPwdLength) then
    signal sqlstate '45000'
      set message_text = 'Password lunga (più di 16 caratteri).';
  end if;

END$$
```

```
DELIMITER $$
USE `Q`$$
CREATE DEFINER = CURRENT_USER TRIGGER `Q`.`Amico_BEFORE_INSERT` BEFORE INSERT ON
`Amico` FOR EACH ROW
BEGIN
if (new.username1 = new.username2) then
        signal sqlstate '45000'
            set message_text = 'Stesso username.';
    end if;
SE NOI STIAMO CERCANDO DI INSERIRE B-A ED ESISTE B-A DEVE DARE ERRORE
    if (exists ( select * from amico a where new.username1 = a.username2 and new.username2 =
a.username1) ) then
        signal sqlstate '45000'
            set message_text = 'Coppia già presente.';
    end if;

END$$
DELIMITER ;
```