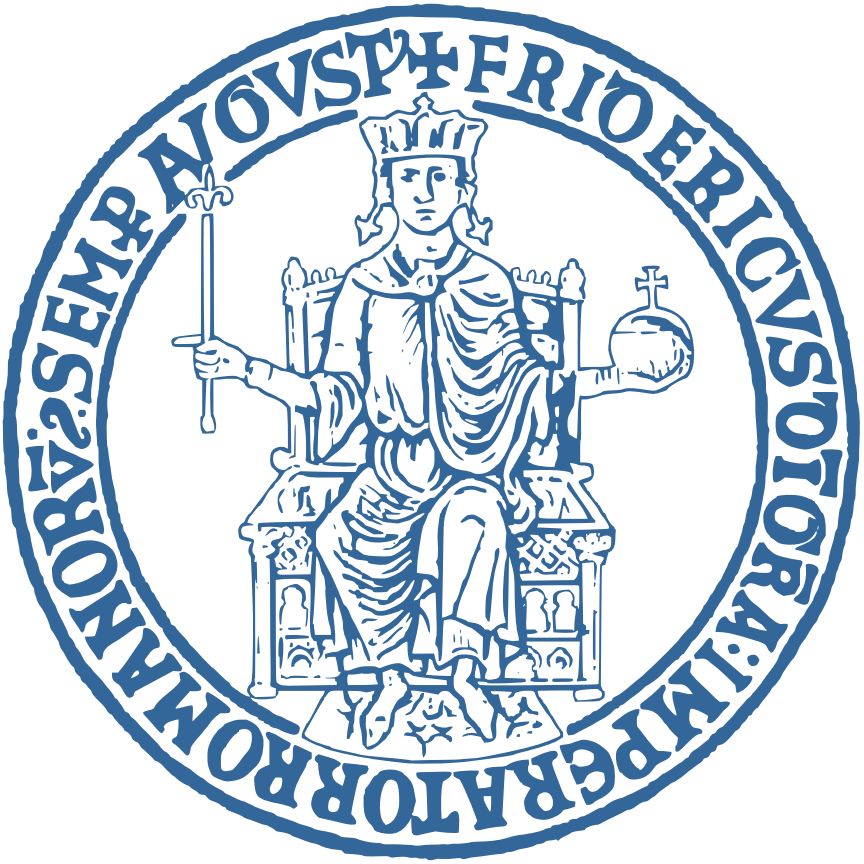
Progetto Basi di dati 2023/2024

Università degli Studi di Napoli Federico II



Traccia 2

Realizzato da:
Antonio De Martino N86004701
Andrea Capitelli N86004646
Davide Gatta N86004143

Prof.ssa: *Mara Sangiovanni*

Gruppo: OOBD2324_17

INDICE:

1.Descrizione progetto……3

2.Class Diagram……

2.1 Schema concettuale…….....4

2.2 Diagramma ER ……...5

2.3 Schema ristrutturato……..6

3.Dizionari…….....

3.1 Dizionario delle classi…….....7

3.2 Dizionario delle associazioni……...9

3.3 Dizionario dei vincoli……..13

4.Schema logico……15

5.Progettazione fisica……16

1 Descrizione progetto

La seguente base di dati ha lo scopo di rappresentare un social network, suddiviso in gruppi tematici.
All'interno dei gruppi le interazioni tra gli utenti sono possibili tramite la creazione di post, commenti e like.

Le funzionalità principali che l'applicativo deve implementare sono:

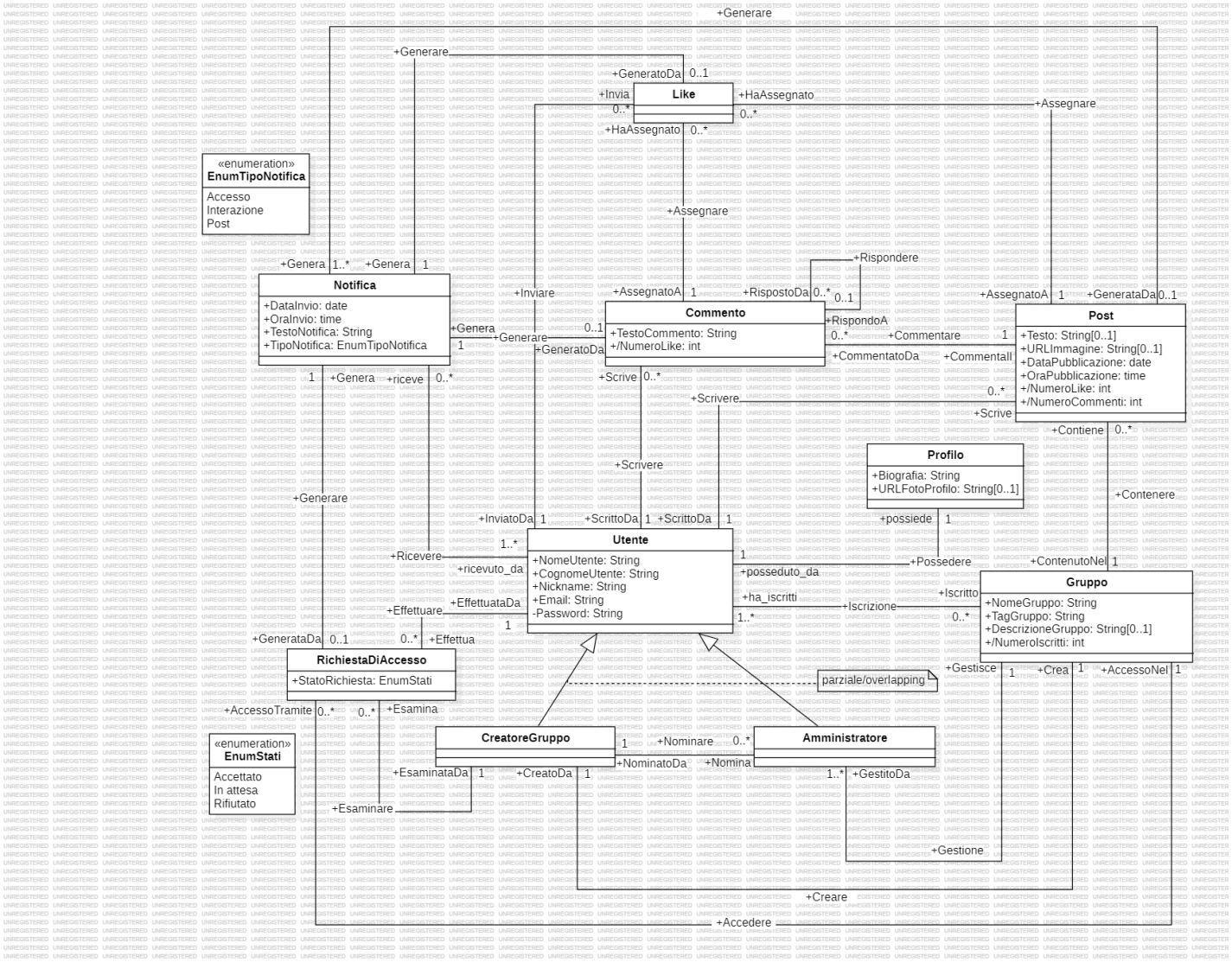
-La possibilità di accedere ai gruppi tramite richieste di accesso.

-Visualizzare report statistici relativi ad un gruppo.

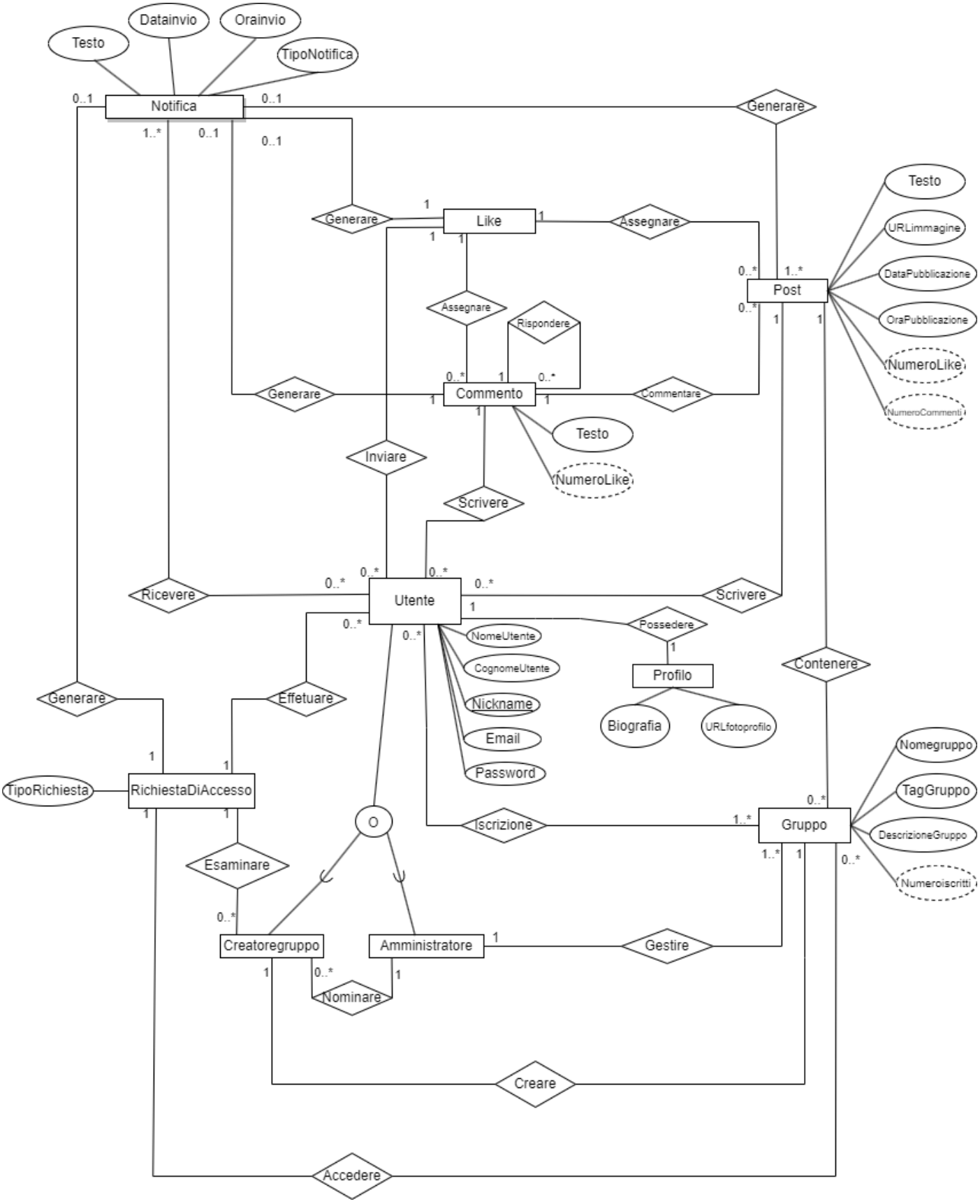
-Creazione di notifiche specifiche relative ad i seguenti eventi: nuova richiesta di accesso, creazione nuovo post, nuova interazione.

2 Class Diagram

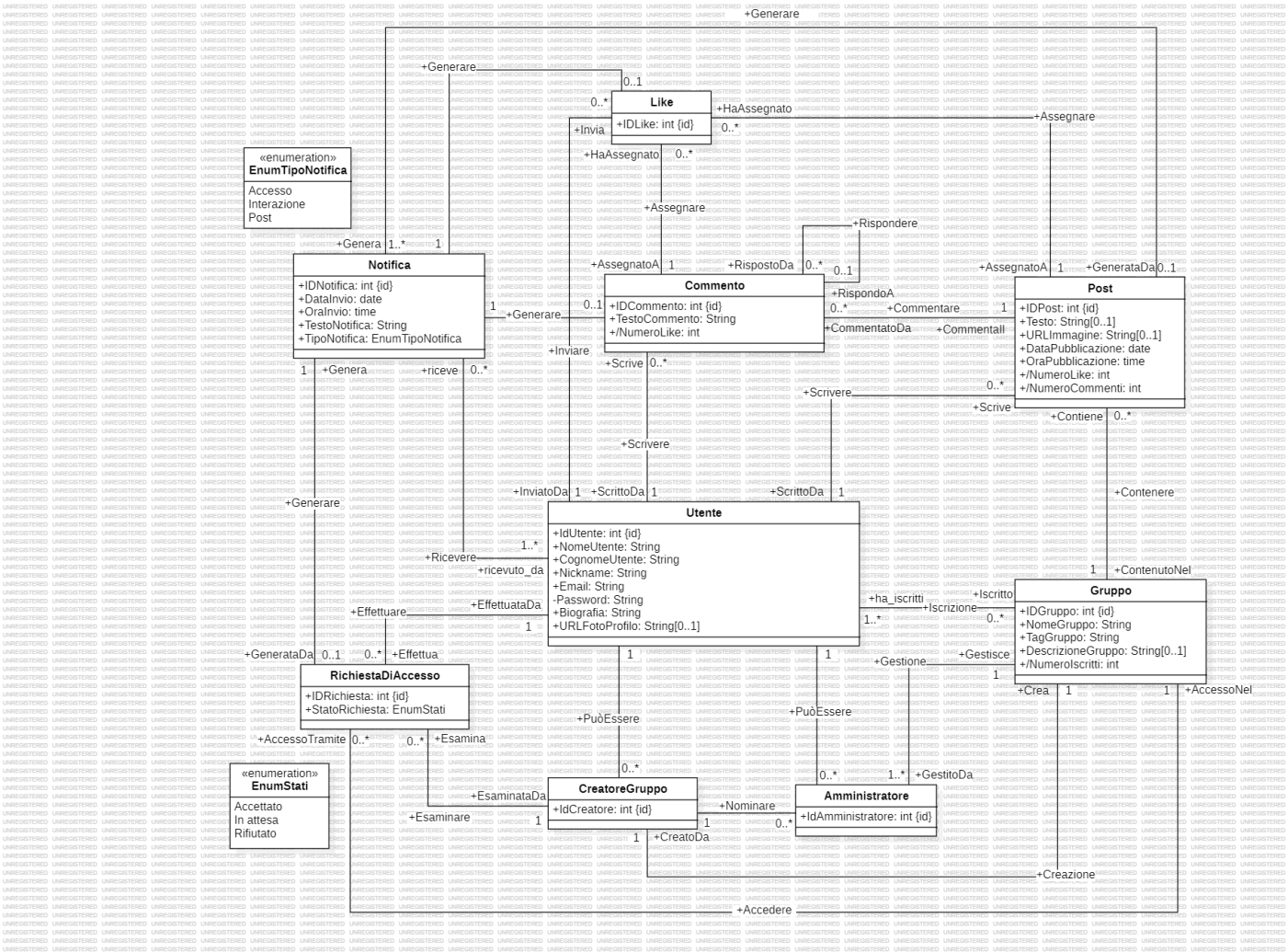
2.1 Schema concettuale



2.3 Diagramma ER



2.3 Schema ristrutturato



3 Dizionari
3.1 Dizionario delle classi

Classe	Attributi	Descrizione
Utente	**NomeUtente**(string): Nome anagrafico dell'utente. **CognomeUtente**(string): Cognome dell'utente. **Nickname**(string): Soprannome con il quale l'utente si registra. **Email**(string): Email con la quale l'utente si registra. **Password**(string): Stringa per validare l'accesso. **Biografia**(string): Breve descrizione dell'utente. **URLFotoProfilo**(string): Stringa contente l'URL dell'immagine profilo. **IdUtente**(int): Chiave surrogata. Identifica univocamente l'utente.	Informazioni riguardanti un utente registrato al social network.
Amministratore	**IdAmministratore**(int): Chiave surrogata. Identifica univocamente un amministratore.	Elenco di tutti gli amministratori dei gruppi.
Creatore Gruppo	**IdCreatore**(int): Chiave surrogata. Identifica univocamente un creatore gruppo.	Elenco di tutti i creatori dei gruppi.
Gruppo	**IdGruppo**(int): Chiave surrogata. Identifica univocamente un gruppo. **NomeGruppo**(string): Nome del gruppo. **TagGruppo**(string): Temi trattati all'interno del gruppo. **DescrizioneGruppo**(string): Breve descrizione del gruppo. **NumeroIscritti**(int): Numero totale di iscritti ad un gruppo.	Informazioni riguardanti i gruppi.
Notifica	**IdNotifica**(int): Chiave surrogata. Identifica univocamente una notifica. **DataInvio**(date): Data di invio della notifica. **OraInvio**(time): Ora di invio della notifica.	Informazioni riguardanti una notifica.

	TestoNotifica(string): Testo esplicativo della notifica. **TipoNotifica**(EnumTipoNotifica) Tipo di notifica ricevuta dall'utente.	
Like	**IdLike**(int): Chiave surrogata. Identifica univocamente un like.	Rappresenta l'interazione di un utente con un post o un commento.
Commento	**IdCommento**(int): Chiave surrogata. Identifica univocamente un commento. **TestoCommento**(string): Messaggio testuale con il quale un utente ha intenzione di commentare. **NumeroLike**(int): Conteggio del numero di like a un commento.	Informazioni riguardanti un determinato commento.
Post	**IdPost**(int): Chiave surrogata. Identifica univocamente un post. **Testo**(string): Contenuto testuale del post. **URLImmagine**(string): Stringa contente l'URL dell'immagine del post. **DataPubblicazione**(date): Data in cui viene pubblicato il post. **OraPubblicazione**(time): Orario in cui viene pubblicato il post. **NumeroLike**(int): Conteggio del numero di like al post. **NumeroCommenti**(int): Conteggio del numero di commenti al post.	Informazioni riguardanti un determinato post.
Richiesta di accesso	**IdRichiesta**(int): Chiave surrogata. Identifica univocamente una richiesta di accesso. **StatoRichiesta**(EnumStati): Stato in cui si trova una richiesta di accesso.	Informazioni riguardanti la richiesta di accesso per accedere ad un gruppo.

3.2 Dizionario delle associazioni

Associazioni	Classi coinvolte	Descrizione
Scrivere (1)	**Utente[0,*]** ruolo **(Scrive)**:Indica chi scrive il Commento. **Commento[1]** ruolo**(ScrittoDa)**: Indica cosa scrive un utente.	*Esprime il legame tra Utente e Commento.*
Scrivere (2)	**Post[1]** ruolo **(ScrittoDa)**: Indica cosa scrive un utente. **Utente[0,*]** ruolo **(Scrive)**:Indica chi scrive il Post.	*Esprime il legame tra Utente e Post.*
Iscrizione	**Utente[0,*]** ruolo **(Iscritto)**: Indica i gruppi a cui un utente è iscritto. **Gruppo[1,*]** ruolo **(HaIscritti)**: Indica gli utenti che sono iscritti al gruppo.	*Esprime il legame tra Utente e Gruppo.*
Effettuare	**Utente[0,*]** ruolo **(Effettua)**: Indica le richieste di accesso effettuate da un utente. **RichiestaDiAccesso[1]** ruolo **(EffettuataDa)**: Indica gli utenti che hanno fatto richiesta per iscriversi ad un gruppo.	*Esprime il legame tra Utente e Richiesta di accesso.*
Ricevere	**Utente[0,*]** ruolo **(Riceve)**: Indica le notifiche ricevute da un utente. **Notifica[1,*]** ruolo **(RicevutoDa)**: Indica quali utenti hanno ricevuto una determinata notifica.	*Esprime il legame tra Notifica ed Utente.*
Inviare	**Utente[0,*]** ruolo **(invia)**: Indica le interazioni tramite like effettuate da un utente. **Like[1]** ruolo **(InviatoDa)**: Indica l'utente che ha interagito con un like.	*Esprime il legame tra Utente e Like.*
PuòEssere (1)	**Utente[0,*]** ruolo **(è)**: Indica gli utenti che sono amministratori in un determinato gruppo. **CreatoreGruppo[1]** ruolo **(rappresentatoDa)**:	*Esprime il legame tra Utente e CreatoreGruppo.*

	Indica quali utenti sono creatori di gruppi.	
PuòEssere (2)	**Utente[0,*]** ruolo **(è)**: Indica gli utenti che sono amministratori in un determinato gruppo. **Amministratore[1]** ruolo **(rappresentatoDa)**: Indica quale utente è un amministratore di un gruppo.	*Esprime il legame tra Utente e Amministratore.*
Nominare	**CreatoreGruppo[0,*]** ruolo **(nomina)**: Indica il creatore gruppo che nomina gli amministratori **Amministratore[1]** ruolo **(nominatoda)**: Indica da chi gli utenti sono stati nominati amministratori	*Esprime il legame tra CreatoreGruppo e Amministratore.*
Creazione	**CreatoreGruppo[1]** ruolo **(Crea)**:Indica il gruppo creato dal CreatoreGruppo. **Gruppo[1]** ruolo **(CreatoDa)**: Indica chi ha creato il gruppo.	*Esprime il legame tra il Gruppo creato e il suo creatore: CreatoreGruppo.*
Esaminare	**CreatoreGruppo[0,*]** ruolo **(Esamina)**:Indica le richieste di accesso che devono essere valutate da un CreatoreGruppo. **RichiestaDiAccesso[1]** ruolo **(EsaminataDa):**Indica da quale CreatoreGruppo è stata valutata una richiesta.	*Esprime il legame tra il CreatoreGruppo e le RichiesteDiAccesso che devono essere valutate, per permettere l'accesso di un utente ad un gruppo.*
Gestione	**Amministratore[1]** ruolo **(Gestisce)**: Indica il gruppo gestito da un determinato amministratore. **Gruppo[1,*]** ruolo **(GestitoDa)**: Indica gli amministratori che gestiscono un gruppo.	*Esprime il legame tra un amministratore e il gruppo che gestisce.*
Accedere	**RichiestaDiAccesso[1]** ruolo (**AccessoNel)**:Indica in quale gruppo la RichiestaDiAccesso permette di accedere. **Gruppo[0,*]** ruolo **(AccessoTramite)**:Indica con quali RichiesteDiAccesso è possibile accedere al gruppo.	*Esprime il legame tra le RichiesteDiAccesso e il rispettivo Gruppo in cui è possibile accedere.*

Contenere	**Gruppo[0,*]** ruolo **(contiene)**: Indica i post contenuti in un determinato gruppo. **Post[1]** ruolo **(ContenutoNel)**: Indica il gruppo in cui è contenuto il post.	*Esprime il legame tra un gruppo e i post che contiene.*
Generare (1)	**RichiestaDiAccesso[1]** ruolo **(Genera)**:Indica quale notifica ha generato la richiesta. **Notifica[0,1]** ruolo **(GeneratoDa)**:Indica che la notifica è stata generata da una richiesta di accesso.	*Esprime il legame tra una richiesta di accesso ad un gruppo e le notifiche che genera la richiesta.*
Assegnare (1)	**Post[0,*]** ruolo **(HaAssegnato)**: Indica i like che sono stati assegnati ad un determinato post. **Like[1]** ruolo **(AssegnatoA)** : Indica il post che ha ricevuto un interazione tramite like.	*Esprime il legame tra un post e i like che ha ricevuto come interazioni.*
Commentare	**Post[0,*]** ruolo **(commentatoDa)**: Indica i commenti che sono stati scritti per interagire con un post. **Commento[1]** ruolo **(CommentaIl)** : Indica il Post che è stato commentato.	*Esprime il legame tra un post ed i suoi commenti.*
Generare (2)	**Post[1,*]** ruolo **(Genera)**:Indica le notifiche che l'inserimento di un nuovo Post genera. **Notifica[0,1]** ruolo **(GeneratoDa)**:Indica che la notifica è stata generata da un nuovo Post.	*Esprime il legame tra un Post e le notifiche che esso genera all'inserimento.*
Generare (3)	**Like[1]** ruolo **(Genera)**:Indica la notifica che l'interazione con un Like genera. **Notifica[0,1]** ruolo **(GeneratoDa)**:Indica che la notifica è stata generata da un Like.	*Esprime il legame tra un Like e la Notifica che il creatore di un Post riceve, quando viene inserito uno nuovo like.*
Generare (4)	**Commento[1]** ruolo **(Genera)**:Indica la notifica generata all'inserimento di un nuovo Commento. **Notifica[0,1]** ruolo **(GeneratoDa)**:Indica che la	*Esprime il legame tra un Commento e la Notifica che il creatore di un Post riceve, quando ne viene scritto uno nuovo coommento.*

	notifica è stata generata da un Commento.	
Assegnare (2)	**Like[1]** ruolo **(AssegnatoA)**: Indica il commento a cui si è assegnato un like. **Commento[0,*]** ruolo **(HaAssegnato)** : Indica i like relativi ad un determinato commento.	*Esprime la relazione tra un commento ed i suoi like.*
Rispondere	**Commento[0,1]** ruolo **(RispondoA)**: Indica il commento con cui si è interagito tramite un altro commento. **Commento[0,*] (RispostoDa)** : Indica i commenti scritti per rispondere ad un determinato commento.	*Esprime la relazione ricorsiva tra un commento e le sue risposte.*

3.3 Dizionario dei vincoli

Nome vincolo	Tipo vincolo	Descrizione
UnicoLikePerPost	**Vincolo di unicità**	*All'interno della classe Like gli attributi IdUtente,IdPost sono unique, per assicurarci che un utente inserisca un solo Like per lo stesso Post.*
UnicoLikePerPostCommento	**Vincolo di unicità**	*All'interno della classe Like gli attributi IdUtente, IdCommento sono unique, per assicurarci che un utente inserisca un solo Like per lo stesso Commento.*
UnicitàNickname	**Vincolo di unicità**	*Un utente non può registrarsi con un Nickname già utilizzato da un altro utente.*
UnicitàUtenteGruppo InAmministratore	**Vincolo di unicità**	*All'interno della classe Amministratore la coppia di attributi IdUtente e IdGruppo sono unique.*
UnicitàRichiestaAccesso PerGruppo	**Vincolo di unicità**	*All'interno della classe RichiestaDiAccesso la coppia di attributi IdUtenteRichiesta e IdGruppoRichiesta sono unique.*
ValiditàData OraNotifica	**Vincolo intrarelazionale**	*La data e l'ora di invio di una notifica corrispondano alla data e all'ora correnti del sistema al momento dell'inserimento dei dati.*
ControlloPostNo TestoNoFoto	**Vincolo intrarelazionale**	*Non può esistere un Post senza testo e immagine.*
ControlloRiferimentoLike	**Vincolo interrelazionale**	*Un like deve fare riferimento ad un post o a un commento esistente.*
UnicitàGruppoIn CreatoreGruppo	**Vincolo di unicità**	*All'interno della classe CreatoreGruppo l'attributo IdGruppo è unique.*
ControlloNumeroLike CommentiInPost	**Vincolo intrarelazionale**	*All'inserimento di un nuovo post i valori NumeroLike e NumeroCommenti sia settato a quello di default 0.*
ControlloNumeriLike InCommento	**Vincolo intrarelazionale**	*All'inserimento di un nuovo commento il numero di like sia settato al valore di default 0.*
CreatoreGruppoè Amministratore	**Vincolo interrelazionale**	*Il creatore gruppo è automaticamente anche un amministratore.*

NuovoAdmin	**Vincolo interrelazionale**	*Il creatore gruppo può nominare amministratore solo utenti iscritti al gruppo.*
CreatorePostInGruppo	**Vincolo interrelazionale**	*L'utente che scrive un post lo sta scrivendo per un gruppo in cui è iscritto.*
NotificaAccessoGruppo	**Vincolo interrelazionale**	*Le notifiche di tipo <<Accesso>> arrivano unicamente al creatore del gruppo, a cui si vuole accedere.*
VerificaStatoRichiesta	**Vincolo interrelazionale**	*Controllo se ci sono richieste di accesso da parte di un utente. In caso affermativo, verifico se la richiesta è stata classificata come "Accettato",in caso contrario, impedisce l'iscrizione al gruppo.*
ValiditàCreatoreGruppo InRichiesta	**Vincolo interrelazionale**	*Controllo se l'id del creatore gruppo nella richiesta è corretto.*
GestioneUscitaCreatore Gruppo	**Vincolo interrelazionale**	*Se il creatore di un gruppo, abbandona un gruppo, allora il gruppo deve essere eliminato.*
GestioneEliminazioneUtente CreatoreGruppo	**Vincolo interrelazionale**	*Se viene eliminato un utente, vengono eliminati tutti i gruppi in cui era un creatore.*
ValiditàNotifica InterazionePost	**Vincolo intrarelazionale**	*Solo uno tra gli attributi IdNuovoPost,IdNuovoCommento e IdNuovoLike deve essere NOT NULL.*
ValiditàNotificaAccesso	**Vincolo intrarelazionale**	*Controllo che gli attributi IdNuovoPost,IdNuovoLike,IdNuovoCommento sono NULL per le notifiche di tipo <<Accesso>>.*
ControlloUtenteLikeIscritto AlGruppo	**Vincolo interrelazionale**	*L'IdUtente di una riga da inserire in Like_ deve corrispondere a quello di un utente iscritto al gruppo in cui è stato scritto il post o il commento, altrimenti si blocca l'inserimento del like.*
ControlloUtenteCommento IscrittoAlGruppo	**Vincolo interrelazionale**	*Un Utente che vuole inserire un commento deve essere iscritto al gruppo in cui commenta.*
ControllaDataOraPost	**Vincolo interrelazionale**	*L'inserimento di un post deve avvenire rispettando la data e l'orario attuali*

4 Schema logico

Utente(<u>IdUtente</u>,NomeUtente,CognomeUtente,Nickname,Email,Password,Biografia,URLFotoProfilo)

CreatoreGruppo(<u>IdCreatore</u>,IdUtente,IdGruppo)
CreatoreGruppo(IdUtente)→Utente(IdUtente)
CreatoreGruppo(IdGruppo)→Gruppo(IdGruppo)

Amministratore(<u>IdAmministratore</u>,IdCreatore,IdUtente,IdGruppo)
Amministratore(IdCreatore)→CreatoreGruppo(IdCreatore)
Amministratore(IdUtente)→Utente(IdUtente)
Amministratore(IdGruppo)→Gruppo(IdGruppo)

Gruppo(<u>IdGruppo</u>,NomeGruppo,TagGruppo,DescrizioneGruppo,NumeroIscritti)

Post(<u>IdPost</u>,Testo,URLImmagine,DataPubblicazione,OraPubblicazione,NumeroLike, NumeroCommenti,IdUtente,IdGruppo,IdNotifica)
Post(IdUtente)→Utente(IdUtente)
Post(IdGruppo)→Gruppo(IdGruppo)
Post(IdNotifica)→Notifica(IdNotifica)

Like_(<u>IdLike</u>,IdUtente,IdCommento,IdPost)
Like_(IdUtente)→Utente(IdUtente)
Like_(IdCommento)→Commento(IdCommento)
Like_(IdPost)→Post(IdPost)

Commento(<u>IdCommento</u>,TestoCommento,NumeroLike,IdUtente,IdCommentoRisp, IdPostCommentato)
Commento(IdUtente)→Utente(IdUtente)
Commento(IdCommentoRisp)→Commento(IdCommento)
Commento(IdPostCommentato)→Post(IdPost)

Notifica(<u>IdNotifica</u>,DataInvio,OraInvio,TestoNotifica,TipoNotifica,IdNuovoPost, IdNuovoCommento,IdNuovoLike)
Notifica(IdNuovPost)→Post(IdPost)
Notifica(IdNuovoCommento)→Commento(IdCommento)
Notifica(IdNuovoLike)→Like_(IdLike)

RichiestaDiAccesso(<u>IdRichiesta</u>,StatoRichiesta,IdUtenteRichiesta,IdCreatore, IdGruppoRichiesta,IdNotificaGenerata)
RichiestaDiAccesso(IdUtenteRichiesta)→Utente(IdUtente)
RichiestaDiAccesso(IdCreatore)→CreatoreGruppo(IdCreatore)
RichiestaDiAccesso(IdNotificaGenerata)→Notifica(IdNotifica)
RichiestaDiAccesso(IdGruppoRichiesta)→Gruppo(IdGruppo)

Iscrizione(<u>IdUtente,IdGruppo</u>)
Iscrizione(IdUtente)→Utente(IdUtente)
Iscrizione(IdGruppo)→Gruppo(IdGruppo)

Ricevere(<u>IdUtente,IdNotifica</u>)
Ricevere(IdUtente)→Utente(IdUtente)
Ricevere(IdNotifica)→Notifica(IdUtente)

5 Progettazione fisica

5.1 Definizione enumerazioni

Di seguito è possibile visionare la documentazione relativa ai tipi enumerativi che sono stati rappresentati all'interno del diagramma ristrutturato:

```
CREATE TYPE EnumStati AS ENUM ('Accettato','In attesa','Rifiutato');
CREATE TYPE EnumTipoNotifica AS ENUM ('Accesso','Interazione','Post');
```

5.2 Documentazione tabelle

Di seguito è possibile visionare la documentazione relativa alla creazione delle tabelle:

```
CREATE TABLE Utente (
        IdUtente SERIAL PRIMARY KEY,
          NomeUtente VARCHAR(30) NOT NULL
                        CHECK (NomeUtente ~ '^[A-Za-zÀ-ÖØ-öø-ÿ  ]+$'),
          CognomeUtente VARCHAR(30) NOT NULL
                        CHECK (CognomeUtente ~ '^[A-Za-zÀ-ÖØ-öø-ÿ ] +$'),
        Nickname VARCHAR(20) NOT NULL CHECK(LENGTH(Nickname)>=5),
        Email VARCHAR(100) UNIQUE NOT NULL,
        Password VARCHAR(50) NOT NULL CHECK(LENGTH(Password)>=6),
        Biografia VARCHAR(350),
        URLFotoProfilo VARCHAR(2000)
);


CREATE TABLE Gruppo (
        IdGruppo SERIAL PRIMARY KEY,
        NomeGruppo VARCHAR(50) NOT NULL,
         TagGruppo VARCHAR(200) NOT NULL CHECK (TagGruppo ~ '^([a-zA-
        ZàèìòùÀÈÌÒÙ]+,)*
        [a-zA-ZàèìòùÀÈÌÒÙ]+$'),
        DescrizioneGruppo TEXT,
        NumeroIscritti INT DEFAULT 0
);
```

```
CREATE TABLE CreatoreGruppo (
        IdCreatore SERIAL PRIMARY KEY,
        IdUtente INT NOT NULL,
        IdGruppo INT NOT NULL,
        FOREIGN KEY (IdUtente) REFERENCES Utente(IdUtente) ON DELETE CASCADE,
        FOREIGN KEY (IdGruppo) REFERENCES Gruppo(IdGruppo) ON DELETE
CASCADE
);

CREATE TABLE Amministratore (
        IdAmministratore SERIAL PRIMARY KEY,
        IdCreatore INT NOT NULL,
        IdUtente INT NOT NULL,
        IdGruppo INT NOT NULL,
        FOREIGN KEY (IdCreatore) REFERENCES CreatoreGruppo(IdCreatore) ON DELETE
        CASCADE,
        FOREIGN KEY (IdUtente) REFERENCES Utente(IdUtente) ON DELETE CASCADE,
        FOREIGN KEY (IdGruppo) REFERENCES Gruppo(IdGruppo) ON DELETE
        CASCADE
);

CREATE TABLE Post (
        IdPost SERIAL PRIMARY KEY,
        Testo TEXT,
        URLImmagine VARCHAR(2000),
        DataPubblicazione DATE NOT NULL DEFAULT CURRENT_DATE,
        OraPubblicazione TIME NOT NULL DEFAULT CURRENT_TIME,
        NumeroLike INT DEFAULT 0,
        NumeroCommenti INT DEFAULT 0,
        IdUtente INT NOT NULL,
        IdGruppo INT NOT NULL,
        IdNotifica INT,
        FOREIGN KEY (IdUtente) REFERENCES Utente(IdUtente) ON DELETE CASCADE,
        FOREIGN KEY (IdGruppo) REFERENCES Gruppo(IdGruppo) ON DELETE
        CASCADE
        --Aggiungere dopo Notifica la FK su IdNotifica
);

CREATE TABLE Notifica (
        IdNotifica SERIAL PRIMARY KEY,
        DataInvio DATE NOT NULL,
        OraInvio TIME NOT NULL,
        TestoNotifica TEXT NOT NULL,
        TipoNotifica EnumTipoNotifica NOT NULL,
        IdNuovoPost INT,
        IdNuovoCommento INT,
        IdNuovoLike INT,
        FOREIGN KEY (IdNuovoPost) REFERENCES Post(IdPost) ON DELETE CASCADE,
```

```
        FOREIGN KEY (IdNuovoCommento) REFERENCES Commento(IdCommento)
        ON DELETE CASCADE,
        FOREIGN KEY (IdNuovoLike) REFERENCES Like_(IdLike) ON DELETE CASCADE
);

CREATE TABLE Iscrizione (
        IdUtente INT,
        IdGruppo INT,
        PRIMARY KEY (IdUtente, IdGruppo),
        FOREIGN KEY (IdUtente) REFERENCES Utente(IdUtente) ON DELETE CASCADE,
        FOREIGN KEY (IdGruppo) REFERENCES Gruppo(IdGruppo) ON DELETE
        CASCADE
);

CREATE TABLE Like_ (
        IdLike SERIAL PRIMARY KEY,
        IdUtente INT NOT NULL,
        IdCommento INT,
        IdPost INT,
        FOREIGN KEY (IdUtente) REFERENCES Utente(IdUtente) ON DELETE CASCADE,
        FOREIGN KEY (IdCommento) REFERENCES Commento(IdCommento)
        ON DELETE CASCADE,
        FOREIGN KEY (IdPost) REFERENCES Post(IdPost) ON DELETE CASCADE
);

CREATE TABLE Commento (
        IdCommento SERIAL PRIMARY KEY,
        TestoCommento TEXT NOT NULL,
        NumeroLike INT DEFAULT 0,
        IdUtente INT NOT NULL,
        IdCommentoRisp INT,
        IdPostCommentato INT NOT NULL,
        FOREIGN KEY (IdUtente) REFERENCES Utente(IdUtente) ON DELETE CASCADE,
        FOREIGN KEY (IdPostCommentato) REFERENCES Post(IdPost)
        ON DELETE CASCADE
);

CREATE TABLE Ricevere (
        IdUtente INT,
        IdNotifica INT,
        PRIMARY KEY (IdUtente, IdNotifica),
        FOREIGN KEY (IdUtente) REFERENCES Utente(IdUtente) ON DELETE CASCADE ,
        FOREIGN KEY (IdNotifica) REFERENCES Notifica(IdNotifica)
        ON DELETE CASCADE
);
```

```
CREATE TABLE RichiestaDiAccesso (
        IdRichiesta SERIAL PRIMARY KEY,
        StatoRichiesta EnumStati DEFAULT 'In attesa',
        IdUtenteRichiesta INT NOT NULL,
        IdCreatore INT NOT NULL,
        IdGruppoRichiesta INT NOT NULL,
        IdNotificaGenerata INT,
        FOREIGN KEY (IdUtenteRichiesta) REFERENCES Utente(IdUtente) ON DELETE
        CASCADE,
        FOREIGN KEY (IdCreatore) REFERENCES CreatoreGruppo(IdCreatore) ON DELETE
        CASCADE,
        FOREIGN KEY (IdNotificaGenerata) REFERENCES Notifica(IdNotifica) ON DELETE
        CASCADE,
        FOREIGN KEY (IdGruppoRichiesta) REFERENCES Gruppo(IdGruppo) ON DELETE
        CASCADE
);
```

5.3 Documentazione vincoli ,trigger e funzioni/procedure

Di seguito è possibile visionare la documentazione relativa all'insieme delle funzioni e dei trigger che permettono la gestione del database e di implementare i vincoli, che nel Capitolo 3.3 e durante la ristrutturazione abbiamo identificato :

- **PostIdNotificafkey**

```
ALTER TABLE Post
ADD CONSTRAINT post_IdNotifica_fkey
FOREIGN KEY (IdNotifica) REFERENCES Notifica(IdNotifica) ON DELETE SET NULL;
```

- **CommentoIdCommentoFKey**

```
ALTER TABLE
ADD CONSTRAINT commento_IdCommentoRisp_fkey
FOREIGN KEY (IdCommentoRisp) REFERENCES Commento(IdCommento) ON DELETE
CASCADE;
```

- **UnicoLikePerPost**

```
ALTER TABLE Like_ ADD CONSTRAINT UnicoLikePerPost
UNIQUE(IdUtente, IdPost);
```

- **UnicoLikePerCommento**

```
ALTER TABLE Like_ ADD CONSTRAINT UnicoLikePerCommento
UNIQUE(IdUtente, IdCommento);
```

- **UnicitàNickname**

```
ALTER TABLE Utente ADD CONSTRAINT UnicitàNickname
UNIQUE(Nickname);
```

- **UnicitàUtenteGruppoInAmministratore**

```
ALTER TABLE Amministratore ADD CONSTRAINT UnicitàUtenteGruppoInAmministratore
UNIQUE(IdGruppo,IdUtente);
```

- **UnicitàRichiestaAccessoPerGruppo**

```
ALTER TABLE RichiestaDiAccesso ADD CONSTRAINT UnicitàRichiestaAccessoPerGruppo
UNIQUE (IdUtenteRichiesta,IdGruppoRichiesta)
```

- **ValiditàDataOraNotifica**

```
ALTER TABLE Notifica
ADD CONSTRAINT ValiditàDataOraNotifica
CHECK (DataInvio=CURRENT_DATE AND OraInvio=CURRENT_TIME)
```

- **ControlloPostNoTestoNoFoto**

```
ALTER TABLE Post ADD CONSTRAINT ControlloPostNoTestoNoFoto
CHECK(
        ( URLImmagine IS NULL AND Testo IS NOT NULL )
        OR
        ( Testo IS NULL AND URLImmagine IS NOT NULL )
        OR
        ( Testo IS NOT NULL AND URLImmagine IS NOT NULL )
);
```

- **ControlloRiferimentoLike**

```
--Controllo che un like è riferito ad un Post o a un Commento
ALTER TABLE Like_
ADD CONSTRAINT ControlloRiferimentoLike
CHECK(
        (IdCommento IS NULL AND IdPost IS NOT NULL)
        OR
        (IdCommento IS NOT NULL AND IdPost IS NULL)
);
```

- **UnicitàGruppoInCreatoreGruppo**

```
ALTER TABLE CreatoreGruppo
ADD CONSTRAINT UnicitaGruppoInCreatoreGruppo UNIQUE (IdGruppo);
```

- **ControlloNumeroLikeCommentiInPost**

```
--La seguente funzione controlla che all'inserimento di un nuovo post
--i valori NumeroLike e NumeroCommenti sia settato a quello di default 0

CREATE OR REPLACE FUNCTION ControlloNumeroLikeCommentiInPost()
RETURNS TRIGGER AS $$
BEGIN
    IF (NEW.NumeroCommenti <> 0 OR NEW.NumeroLike <> 0) THEN
        RAISE EXCEPTION 'Impossibile inserire un post che abbia già dei like e commenti';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER ControlloNumeroLikeCommentiInPost
BEFORE INSERT ON Post
FOR EACH ROW
EXECUTE FUNCTION ControlloNumeroLikeCommentiInPost();
```

- **ControlloNumeriLikeInCommento**

```
--La seguente funzione controlla che all'inserimento di un nuovo commento
--il numero di like sia settato al valore di default 0

CREATE OR REPLACE FUNCTION ControlloNumeroLikeInCommento()
RETURNS TRIGGER AS $$
BEGIN
        IF (NEW.NumeroLike <> 0) THEN
                RAISE EXCEPTION 'Impossibile inserire un commento che abbia un numero di like
                diverso da zero';
        END IF;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER ControlloNumeroLikeInCommento
BEFORE INSERT ON Commento
FOR EACH ROW
EXECUTE FUNCTION ControlloNumeroLikeInCommento();
```

- **CreatoreGruppoèAmministratore**

--La seguente funzione dopo l'inserimento di un creatore gruppo inserisce una tupla anche nella
--tabella amministratore, relativa allo stesso utente.

```
CREATE OR REPLACE FUNCTION CreatoreAdmin()
RETURNS TRIGGER AS $$

BEGIN
        INSERT INTO Iscrizione  (idUtente, idGruppo)
        VALUES (NEW.idUtente , NEW.idGruppo);

        INSERT INTO Amministratore (idCreatore , idUtente , idGruppo)
        VALUES (NEW.idCreatore , NEW.idUtente ,NEW.idGruppo);
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER CreatoreGruppoèAmministratore
AFTER INSERT ON CreatoreGruppo
FOR EACH ROW
EXECUTE FUNCTION CreatoreAdmin();
```

- **NuovoAdmin**

--La seguente funzione si occupa di controllare che l'utente nominato amministratore
-- sia effettivamente iscritto al gruppo

```
CREATE OR REPLACE FUNCTION NuovoAdmin()
RETURNS TRIGGER AS $$
BEGIN
        IF NOT EXISTS (SELECT 1
                                FROM Iscrizione
                                WHERE New.idUtente=idUtente AND New.idGruppo=idGruppo)
        THEN
                RAISE EXCEPTION  'L''utente non è iscritto';
    END IF;
      RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER NominaAmministratore
BEFORE INSERT ON Amministratore
FOR EACH ROW
EXECUTE FUNCTION NuovoAdmin();
```

- **CreatorePostInGruppo**

```
--La seguente funzione si occupa di controllare che un utente che scrive un post, lo stia scrivendo
--per un gruppo in cui è iscritto

CREATE OR REPLACE FUNCTION CreatorePostInGruppo()
RETURNS TRIGGER AS $$
BEGIN
        IF NOT EXISTS (SELECT 1
                        FROM Iscrizione I
                        WHERE I.IdUtente=NEW.IdUtente AND I.IdGruppo=NEW.IdGruppo)
        THEN
                RAISE EXCEPTION 'L''utente che ha scritto il post non è iscritto al gruppo';
        END IF;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER CreatorePostInGruppo
BEFORE INSERT ON Post
FOR EACH ROW
EXECUTE FUNCTION CreatorePostInGruppo();
```

- **NotificaAccessoCreatore**

```
--La seguente funzione invia una notifica di tipo <<Accesso>> al CreatoreGruppo del gruppo a cui
--un utente vuole accedere

CREATE OR REPLACE FUNCTION InviaNotificaAccesso()
RETURNS TRIGGER AS $$
DECLARE
        varCreatoreGruppo CreatoreGruppo.IdCreatore%TYPE;
        varNewIdNotifica Notifica.IdNotifica%TYPE;
        varNomeUtenteRichiesta Utente.NomeUtente%TYPE;
        varCognomeUtenteRichiesta Utente.CognomeUtente%TYPE;
        varNomeGruppoRichiesta Gruppo.NomeGruppo%TYPE;
        varIdUtenteCreatore Utente.IdUtente%TYPE;
        Messaggio TEXT;
BEGIN
        --Salvo il Creatore del gruppo dove viene effettuata una nuova richiesta
        SELECT IdCreatore
        INTO varCreatoreGruppo
        FROM RichiestaDiAccesso
        WHERE IdRichiesta = NEW.IdRichiesta;

        --Salvo il nome e cognome dell'utente che effettua la richiesta di accesso
        SELECT NomeUtente
        INTO varNomeUtenteRichiesta
        FROM Utente
        WHERE IdUtente = NEW.IdUtenteRichiesta;
```

```
        SELECT CognomeUtente
        INTO varCognomeUtenteRichiesta
        FROM Utente
        WHERE IdUtente = NEW.IdUtenteRichiesta;
        --Salvo il nome del gruppo in cui si effettua la richiesta di accesso
        SELECT NomeGruppo
        INTO varNomeGruppoRichiesta
        FROM Gruppo
        WHERE IdGruppo = New.IdGruppoRichiesta;
        --Testo della notifica
        Messaggio := 'L"utente ' || varNomeUtenteRichiesta || ' ' || varCognomeUtenteRIchiesta || ' ha
        effettuato una nuova richiesta di accesso al gruppo: ' || varNomeGruppoRichiesta || '.';
        --Creo e salvo l'id della nuova notifica
        INSERT INTO Notifica(DataInvio, OraInvio, TestoNotifica, TipoNotifica)
        VALUES (CURRENT_DATE, CURRENT_TIME,Messaggio,'Accesso')
        RETURNING IdNotifica INTO varNewIdNotifica;
        UPDATE RichiestaDiAccesso SET IdNotificaGenerata=varNewIdNotifica
        WHERE IdRichiesta=NEW.IdRichiesta;
        --Recupero l'id dell'utente che è il creatore
        SELECT IdUtente
        INTO varIdUtenteCreatore
        FROM CreatoreGruppo
        WHERE IdCreatore=NEW.IdCreatore;
        --Invio la notifica al'utente che è il  creatore
        INSERT INTO Ricevere VALUES (varIdUtenteCreatore, varNewIdNotifica);
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE TRIGGER NotificaAccessoCreatore
AFTER INSERT ON RichiestaDiAccesso
FOR EACH ROW
EXECUTE FUNCTION InviaNotificaAccesso();
```

- **VerificaStatoRichiesta**

```
--La seguente funzione si occupa di controllare se esistono richieste di accesso per un utente
--e se esiste, controlla se è esitata come <<Accettato>> altrimenti blocca l'iscrizione al gruppo
CREATE OR REPLACE FUNCTION VerificaStatoRichiesta()
RETURNS TRIGGER AS $$
BEGIN
        IF NEW.IdUtente NOT IN (SELECT IdUtente
                                FROM CreatoreGruppo
                                WHERE IdGruppo=NEW.IdGruppo)
      THEN
            IF NOT EXISTS(SELECT 1
                          FROM RichiestaDiAccesso
                          WHERE IdUtenteRichiesta=NEW.IdUtente AND
                          IdGruppoRichiesta=NEW.IdGruppo)
            THEN
                  RAISE EXCEPTION 'L'utente non ha fatto richiesta di accedere al gruppo.';
            ELSIF NOT EXISTS(SELECT 1
                             FROM RichiestaDiAccesso
                             WHERE StatoRichiesta='Accettato' AND
                             IdUtenteRichiesta=NEW.IdUtente AND
                             IdGruppoRichiesta=NEW.IdGruppo)
            THEN
                  RAISE EXCEPTION 'La richiesta di accesso dell'utente non è stata
                  accettata.';
            END IF;
      END IF;
      RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER VerificaStatoRichiesta
BEFORE INSERT ON Iscrizione
FOR EACH ROW
EXECUTE FUNCTION VerificaStatoRichiesta();
```

- **ControllaRichiestaUtenteIscritto**

--La seguente funzione ha il compito di evitare che un utente
--possa mandare una richiesta di accesso ad un gruppo in cui è già iscritto

```
CREATE OR REPLACE FUNCTION ControllaRichiestaUtenteIscritto()
RETURNS TRIGGER AS $$
BEGIN
        IF EXISTS(SELECT 1
                        FROM Iscrizione
                        WHERE NEW.idUtenteRichiesta=idUtente AND
                                NEW.idGruppoRichiesta=idGruppo)
        THEN
                        RAISE EXCEPTION  'Utente già iscritto al gruppo';

        END IF;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE  TRIGGER ValiditàRichiestaDiAccesso
BEFORE INSERT ON RichiestaDiAccesso
FOR EACH ROW
EXECUTE FUNCTION ControllaRichiestaUtenteIscritto();
```

- **IscriviUtenteInGruppo**

--La seguente funzione non appena la richiesta di accesso a un di un utente viene accettata, --quest'ultimo viene iscritto nel gruppo a cui aveva fatto richiesta.

```
CREATE OR REPLACE FUNCTION IscriviUtenteInGruppo()
RETURNS TRIGGER AS $$
BEGIN
        INSERT INTO Iscrizione VALUES (NEW.IdUtenteRichiesta,NEW.IdGruppoRichiesta);
        RETURN NEW;
END;

CREATE TRIGGER IscriviUtenteInGruppo
AFTER UPDATE ON RichiestaDiAccesso
FOR EACH ROW
WHEN (NEW.StatoRichiesta='Accettato')
EXECUTE FUNCTION IscriviUtenteInGruppo();
```

- **ValiditàCreatoreGruppoInRichiesta**

--La seguente funzione controlla se l'id del creatore gruppo nella richiesta è corretto.

```
CREATE OR REPLACE FUNCTION ValiditàCreatoreInRichiesta()
RETURNS TRIGGER AS $$
BEGIN
        IF NOT EXISTS(SELECT 1
                        FROM CreatoreGruppo CG
                        WHERE CG.IdCreatore=NEW.IdCreatore AND
                        CG.IdGruppo=NEW.IdGruppoRichiesta)
        THEN
                RAISE EXCEPTION 'L"idCreatore non corrisponde con quello del creatore
                del gruppo.';
        END IF;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE TRIGGER ValiditàCreatoreInRichiesta
BEFORE INSERT ON RichiestaDiAccesso
FOR EACH ROW
EXECUTE FUNCTION ValiditàCreatoreInRichiesta();
```

- **EliminaRichiestaAccesso**

--La seguente funzione ha il comito di eliminare una richiesta di accesso dalla relativa tabella
--appena viene esitata come <<Rifiutato>>

```
CREATE OR REPLACE FUNCTION EliminaRichiestaAccesso()
RETURNS TRIGGER AS $$
BEGIN
        DELETE FROM RichiestaDiAccesso WHERE NEW.IdRichiesta=IdRichiesta;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER ControlloRichiestaRifiutata
AFTER UPDATE ON RichiestaDiAccesso
FOR EACH ROW
WHEN(NEW.StatoRichiesta='Rifiutato')
EXECUTE FUNCTION EliminaRichiestaAccesso();
```

- **EliminaNotificaRichiestaAccesso**

--Dopo aver eliminato una richiesta di accesso, la seguente funzione elimina anche la notifica che
--ha generato la corrispondente richiesta di accesso.

```
CREATE OR REPLACE FUNCTION EliminaNotificaRichiestaAccesso()
RETURNS TRIGGER AS $$
BEGIN
        DELETE FROM  Notifica WHERE IdNotifica=OLD.IdNotificaGenerata;
        RETURN OLD;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE TRIGGER EliminaNotificaRichiestaAccesso
AFTER DELETE ON RichiestaDiAccesso
FOR EACH ROW
EXECUTE FUNCTION EliminaNotificaRichiestaAccesso();
```

- **GestioneUscitaCreatoreGruppo**

--Se il creatore di un gruppo, abbandona un gruppo, allora il gruppo deve essere eliminato

```
CREATE OR REPLACE FUNCTION GestioneUscitaCreatoreGruppo()
RETURNS TRIGGER AS $$
BEGIN
        IF EXISTS (SELECT 1
                        FROM CreatoreGruppo C
                        WHERE C.IdGruppo=OLD.IdGruppo AND C.IdUtente=OLD.IdUtente)
        THEN
                DELETE FROM Gruppo WHERE IdGruppo=OLD.IdGruppo;
        END IF;
        RETURN OLD;
END;
$$ LANGUAGE plpgsql;


CREATE OR REPLACE TRIGGER GestioneUscitaCreatoreGruppo
AFTER DELETE ON Iscrizione
FOR EACH ROW
EXECUTE FUNCTION GestioneUscitaCreatoreGruppo();
```

- **GestioneEliminazioneUtenteCreatoreGruppo**

--Se viene eliminato un utente, vengono eliminati tutti i gruppi in cui era un creatore

```
CREATE OR REPLACE FUNCTION GestioneEliminazioneUtenteCreatoreGruppo()
RETURNS TRIGGER AS $$
BEGIN
        DELETE FROM Gruppo WHERE IdGruppo IN (SELECT C.IdGruppo
                                              FROM CreatoreGruppo C
                                              WHERE C.IdUtente=OLD.IdUtente);
        RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER GestioneEliminazioneUtenteCreatoreGruppo
BEFORE DELETE ON Utente
FOR EACH ROW
EXECUTE FUNCTION GestioneEliminazioneUtenteCreatoreGruppo();
```

- **InviaNotificaNuovoPost**

--La seguente funzione permette di inserire una nuova Notifica, che deve essere inviata a tutti e soli
--gli iscritti al gruppo in cui è stato scritto il post.
--La funzione inserisce nella tabella Ricevere, l'utente che riceve la notifica con il rispettivo
--IdNotifica e aggiorna l'attributo IdNotifica del Post che è stato scritto.

```
CREATE OR REPLACE FUNCTION InserisciNotificaPost(CreatorePost
Utente.IdUtente%TYPE, UtenteNotifica Utente.IdUtente%TYPE,IdGruppoIN
Gruppo.IdGruppo%TYPE,IdPostIN INT)
RETURNS  VOID AS $$
DECLARE
        Messaggio TEXT;
        NewIdNotifica INT;
        NomeGruppoIN Gruppo.NomeGruppo%TYPE;
        varNome Utente.NomeUtente%TYPE;
        varCognome Utente.CognomeUtente%TYPE;
BEGIN
        --Estraiamo il nome del gruppo
        SELECT  G.NomeGruppo
        INTO NomeGruppoIN
        FROM Gruppo G
        WHERE G.IdGruppo=IdGruppoIN;

        --Estraiamo nome e cognome dell'utente creatore del post
        SELECT U.NomeUtente
        INTO varNome
        FROM Utente U
        WHERE U.IdUtente=CreatorePost;
```

```
        SELECT U.CognomeUtente
        INTO varCognome
        FROM Utente U
        WHERE U.IdUtente=CreatorePost;

        Messaggio := 'L''utente ' ||varNome|| ' '||varCognome|| ' ha scritto un nuovo post in:
        '||NomeGruppoIN||'.';
        INSERT INTO Notifica (DataInvio, OraInvio, TestoNotifica, TipoNotifica, IdNuovoPost)
        VALUES (CURRENT_DATE, CURRENT_TIME, Messaggio, 'Post', IdPostIN)
        RETURNING IdNotifica INTO NewIdNotifica;    --Recupera l'ultimo ID generato
        UPDATE Post SET IdNotifica=NewIdNotifica WHERE IdPost=IdPostIN;
        INSERT INTO Ricevere VALUES(UtenteNotifica,NewIdNotifica);
END;
$$ LANGUAGE plpgsql;

--La seguente funzione di trigger permette di estrarre tutti gli utenti iscritti al gruppo in cui è stato
-- scritto il post e richiama la funzione InserisciNotificaPost

CREATE OR REPLACE FUNCTION InviaNotificaNuovoPost()
RETURNS TRIGGER AS $$
DECLARE
        EstraiUtentiGruppo CURSOR FOR
                SELECT I.IdUtente
                FROM Iscrizione I
                WHERE I.IdGruppo = NEW.IdGruppo AND I.IdUtente<>NEW.IdUtente;
BEGIN
        FOR RigaUtenti IN EstraiUtentiGruppo
        LOOP
                -- Chiamata alla funzione per ogni utente nel gruppo
                PERFORM InserisciNotificaPost(NEW.IdUtente, RigaUtenti.IdUtente,
                NEW.IdGruppo,NEW.IdPost);
                RETURN NEW;
        END LOOP;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER InviaNotificaNuovoPost
AFTER INSERT ON Post
FOR EACH ROW
EXECUTE FUNCTION InviaNotificaNuovoPost ();
```

- **ValiditàNotificaInterazionePost**

--La seguente funzione di trigger si occupa di controllare la validità di una notifica, una notifica per essere valida deve avere una sola chiave esterna diversa da null e le altre due uguali a null quando è di tipo <<Interazione>> o <<Post>>.

```
CREATE OR REPLACE FUNCTION ValiditàNotificaInterazionePost()
RETURNS TRIGGER AS $$
BEGIN
        IF (
                (NEW.idNuovoLike IS NULL AND NEW.idNuovoCommento IS NULL AND
                NEW.idNuovoPost IS NULL)
                 OR
                (NEW.idNuovoLike IS NULL AND NEW.idNuovoCommento IS NOT NULL AND
                NEW.idNuovoPost IS NOT NULL)
                OR
                (NEW.idNuovoLike IS NOT NULL AND NEW.idNuovoCommento IS NOT NULL
                AND NEW.idNuovoPost IS NULL)
                OR
                (NEW.idNuovoLike IS NOT NULL AND NEW.idNuovoCommento IS NULL AND
                NEW.idNuovoPost IS NOT NULL)
        )THEN
                RAISE EXCEPTION 'La notifica non è valida';
        END IF;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;


CREATE TRIGGER NotificaInterazionePost
BEFORE INSERT ON Notifica
FOR EACH ROW
WHEN (NEW.TipoNotifica='Interazione' OR NEW.TipoNotifica ='Post')
EXECUTE FUNCTION ValiditàNotificaInterazionePost();
```

- **ValiditàNotificaAccesso**

```
--La seguente funzione controlla che gli attributi IdNuovoPost,IdNuovoLike,IdNuovoCommento sono
--NULL per le notifiche di tipo <<Accesso>>
CREATE OR REPLACE FUNCTION ValiditàNotificaAccesso()
RETURNS TRIGGER AS $$
BEGIN
        IF(NEW.IdNuovoCommento IS NOT NULL OR NEW.IdNuovoLike IS NOT NULL
            OR NEW.IdNuovoPost  IS NOT NULL)
        THEN
                RAISE EXCEPTION 'Attributi della notifica non validi.';
        END IF;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER ValiditàNotificaAccesso
BEFORE INSERT ON Notifica
FOR EACH ROW
WHEN (NEW.TipoNotifica='Accesso')
EXECUTE FUNCTION ValiditàNotificaAccesso();
```

- **AggiungiIscritto**

```
--La seguente funzione permette di aggiornare il numero di iscritti quando un utente entra in un
--gruppo.

CREATE OR REPLACE FUNCTION AumentaNumeroIscritti(GruppoIN IN INT)
RETURNS VOID AS $$
BEGIN
        --Aggiorno il numero di iscritti
        UPDATE Gruppo SET NumeroIscritti=NumeroIscritti+1
        WHERE IdGruppo=GruppoIN;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION AggiungiIscritto()
RETURNS TRIGGER AS $$
BEGIN
        PERFORM AumentaNumeroIscritti(NEW.IdGruppo);
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE  OR REPLACE TRIGGER AggiungiIscritto
AFTER INSERT ON Iscrizione
FOR EACH ROW
EXECUTE FUNCTION AggiungiIscritto();
```

- **RimuoviIscritto**

--La seguente funzione permette di aggiornare il numero di iscritti quando un utente esce da un
--gruppo.

```
CREATE OR REPLACE FUNCTION DiminuisciNumeroIscritti(GruppoIN IN INT)
RETURNS VOID AS $$
BEGIN
        --Aggiorno il numero di iscritti
        UPDATE Gruppo SET NumeroIscritti=NumeroIscritti-1
        WHERE IdGruppo=GruppoIN;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION RimuoviIscritto()
RETURNS TRIGGER AS $$
BEGIN
        PERFORM DiminuisciNumeroIscritti(NEW.IdGruppo);
        RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER RimuoviIscritto
BEFORE DELETE ON Iscrizione
FOR EACH ROW
EXECUTE FUNCTION RimuoviIscritto();
```

- **AggiungiLike**

--La seguente funzione permette di aumentare il numero di Like per un commento o un post

```
CREATE OR REPLACE FUNCTION(CommentoIN IN Commento.IdCommento%TYPE,
PostIN IN Post.IdPost%TYPE)
RETURNS VOID AS $$
BEGIN
        --Solo uno tra IdCommento e IdPost è NOT NULL in una riga di Like_ poiché uno stesso like
        --non può fare riferimento ad un Post e ad un Commento
        IF(CommentoIN IS NOT NULL)
        THEN
                UPDATE Commento SET NumeroLike=NumeroLike+1
                WHERE IdCommento=CommentoIN;
        ELSE
                UPDATE Post SET NumeroLike=NumeroLike+1
                WHERE IdPost=PostIN;
        END IF;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE FUNCTION AggiungiLike()
RETURNS TRIGGER AS $$
BEGIN
        PERFORM IncrementaLike(NEW.IdCommento,NEW.IdPost);
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER AggiungiLike
AFTER INSERT ON Like_
FOR EACH ROW
EXECUTE FUNCTION AggiungiLike();
```

- **RimuoviLike**

```
--La seguente funzione permette di diminuire il numero di Like per un commento o un post

CREATE OR REPLACE FUNCTION DecrementaLike(CommentoIN IN
Commento.IdCommento%TYPE,
PostIN IN Post.IdPost%TYPE)
RETURNS VOID AS $$
BEGIN
    --Solo uno tra IdCommento e IdPost è NOT NULL in una riga di Like_ poiché uno stesso like
    --non può fare riferimento ad un Post e ad un Commento
    IF(CommentoIN IS NOT NULL)THEN
        UPDATE Commento SET NumeroLike=NumeroLike-1
        WHERE IdCommento=CommentoIN;
    ELSE
        UPDATE Post SET NumeroLike=NumeroLike-1
        WHERE IdPost=PostIN;
    END IF;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION SottraiLike()
RETURNS TRIGGER AS $$
BEGIN
        PERFORM DecrementaLike(OLD.IdCommento,OLD.IdPost);
        RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER SottraiLike
BEFORE DELETE ON Like_
FOR EACH ROW
EXECUTE FUNCTION SottraiLike();
```

- **ControlloUtenteLikeIscrittoAlGruppo**

--La seguente funzione controlla che l'IdUtente di una riga da inserire in Like_ corrisponda a
-- quello di un utente iscritto al gruppo in cui è stato scritto il post o il commento, altrimenti
--blocca l'inserimento del like.

```
CREATE OR REPLACE FUNCTION ControlloUtenteLikeIscrittoAlGruppo()
RETURNS TRIGGER AS $$
DECLARE
        varIdGruppo Gruppo.IdGruppo%TYPE;
BEGIN
        --Recupero l'id del gruppo in cui il like è stato inserito
        IF NEW.IdCommento IS NULL THEN   --Il like è per un post
                SELECT IdGruppo
                INTO varIdGruppo
                FROM Post
                WHERE IdPost=NEW.IdPost;
        ELSE                                    --Il like è per un commento
                SELECT IdGruppo
                INTO varIdGruppo
                FROM Commento C JOIN Post P
                ON C.IdPostCommentato=P.IdPost
                WHERE IdCommento=NEW.IdCommento;
        END IF;
        IF NEW.IdUtente NOT IN (SELECT IdUtente
                                FROM Iscrizione
                                WHERE IdGruppo=varIdGruppo)
        THEN
                RAISE EXCEPTION 'L''utente non è iscritto al gruppo. ';
        END IF;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER ControlloUtenteLikeIscrittoAlGruppo
BEFORE INSERT ON Like_
FOR EACH ROW
EXECUTE FUNCTION ControlloUtenteLikeIscrittoAlGruppo();
```

- **AggiungiCommento**

```
--La seguente funzione permette di incrementare il numero di commenti sotto a un post
--non appena viene inserita una nuova riga in Commento

CREATE OR REPLACE FUNCTION AggiungiCommento()
RETURNS TRIGGER AS $$
BEGIN
        UPDATE Post SET NumeroCommenti=NumeroCommenti+1
        WHERE IdPost=NEW.IdPostCommentato;
        RETURN NEW;
END;
    $$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER AggiungiCommento
AFTER INSERT ON Commento
FOR EACH ROW
EXECUTE FUNCTION AggiungiCommento();
```

- **RimuoviCommento**

```
--La seguente funzione permette di decrementare il numero di commenti sotto a un post
--non appena viene eliminata una  riga in Commento

CREATE OR REPLACE FUNCTION RimuoviCommento()
RETURNS TRIGGER AS $$
BEGIN
        UPDATE Post SET NumeroCommenti=NumeroCommenti-1
        WHERE IdPost=OLD.IdPostCommentato;
        RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER RimuoviCommento
AFTER DELETE ON Commento
FOR EACH ROW
EXECUTE FUNCTION RimuoviCommento();
```

- **ControlloUtenteCommentoIscrittoAlGruppo**

--La seguente funzione permette di controllare che un Utente che vuole inserire un commento sia
--iscritto al gruppo in cui commenta

```
CREATE OR REPLACE FUNCTION ControlloUtenteCommentoIscrittoAlGruppo ()
RETURNS TRIGGER AS $$
DECLARE
        varIdGruppo Gruppo.IdGruppo%TYPE;
BEGIN
        --Recupero l'IdGruppo in cui si trova il post (o il commento sotto il post) commentato
        SELECT IdGruppo
        INTO varIdGruppo
        FROM Post
        WHERE IdPost=NEW.IdPostCommentato;

        IF NEW.IdUtente NOT IN (SELECT IdUtente
                                FROM Iscrizione
                                WHERE IdGruppo=varIdGruppo)
        THEN
                RAISE EXCEPTION 'L''utente non è iscritto al gruppo. ';
        END IF;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER ControlloUtenteCommentoIscrittoAlGruppo
BEFORE INSERT ON Commento
FOR EACH ROW
EXECUTE FUNCTION ControlloUtenteCommentoIscrittoAlGruppo();
```

- **CreaGruppo**

--La seguente funzione, che prende in input l'id di un utente che vuole creare un gruppo e le
-- informazioni necessarie per la creazione del gruppo, permette di creare un gruppo, nominare
--creatore gruppo un utente.

```
CREATE OR REPLACE FUNCTION CreaGruppo(UtenteCreatoreIN IN
Utente.IdUtente%TYPE,NomeGruppoIN IN Gruppo.NomeGruppo%TYPE,TagIN IN
Gruppo.TagGruppo%TYPE,DescrIN IN Gruppo.DescrizioneGruppo%TYPE)
RETURNS INT AS $$
DECLARE
        varIdGruppo Gruppo.IdGruppo%TYPE;
BEGIN
        --Inserisci dati del nuovo gruppo
        INSERT INTO Gruppo(NomeGruppo,TagGruppo,DescrizioneGruppo)
        VALUES(NomeGruppoIN,TagIN,DescrIN)
        RETURNING IdGruppo INTO varIdGruppo;
```

```
        --Inserisci quale utente ha creato il gruppo
        INSERT INTO CreatoreGruppo(IdUtente,IdGruppo) VALUES
        (UtenteCreatoreIN,varIdGruppo);
        RETURN varIdGruppo;
END;
$$ LANGUAGE plpgsql;
```

- **InviaNotificaInterazioneLike**

```
--La seguente funzione, all'inserimento di una nuova riga nella tabella Like_ per un Post, consente
-- di inviare la relativa notifica al creatore del Post, questo si traduce con l'inserimento di una
--nuova riga nella tabella Notifica.

CREATE OR REPLACE FUNCTION NotificaInterazioneLike()
RETURNS TRIGGER AS $$
DECLARE
        NomeUtenteInterazione Utente.NomeUtente%TYPE;
        CognomeUtenteInterazione Utente.NomeUtente%TYPE;
        NewIdNotifica Notifica.IdNotifica%TYPE;
        CreatorePost Post.IdUtente%TYPE;
        Messaggio TEXT;
BEGIN
        --Salvo il nome dell'utente che ha interagito
        SELECT NomeUtente
        INTO NomeUtenteInterazione
        FROM Utente
        WHERE IdUtente = NEW.IdUtente;
        --Salvo il cognome dell'utente che ha interagito
        SELECT CognomeUtente
        INTO CognomeUtenteInterazione
        FROM Utente
        WHERE IdUtente = NEW.IdUtente;

        --Salvo l'id del Creatore del Post
        SELECT IdUtente
        INTO CreatorePost
        FROM Post
        WHERE IdPost = NEW.IdPost;

        --Se il creatore post ha messo like al suo post non deve ricevere la notifica
        IF CreatorePost = NEW.IdUtente THEN
                RETURN NULL;
        END IF;

        Messaggio:= 'L"utente '|| NomeUtenteInterazione ||' '|| CognomeUtenteInterazione||'ha messo
        "Mi piace" al tuo post.';
```

```
        --Creo e salvo l'id della notifica appena generata
        INSERT INTO Notifica(DataInvio,OraInvio,TestoNotifica,TipoNotifica,IdNuovoLike)
        VALUES (CURRENT_DATE,CURRENT_TIME,Messaggio,'Interazione',NEW.IdLike)
        RETURNING IdNotifica INTO NewIdNotifica;

         --Invio della notifica al creatore del post
        INSERT INTO Ricevere
        VALUES(CreatorePost, NewIdNotifica);
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER InviaNotificaInterazioneLike
AFTER INSERT ON Like_
FOR EACH ROW
WHEN (NEW.idPost IS NOT NULL )
EXECUTE FUNCTION NotificaInterazioneLike();
```

- **InviaNotificaInterazioneCommento**

```
CREATE OR REPLACE FUNCTION NotificaInterazioneCommento()
RETURNS TRIGGER AS $$
DECLARE
        NomeUtenteInterazione Utente.NomeUtente%TYPE;
        CognomeUtenteInterazione Utente.NomeUtente%TYPE;
        NewIdNotifica Notifica.IdNotifica%TYPE;
        CreatorePost Post.IdUtente%TYPE;
        Messaggio TEXT;
BEGIN
        --Salvo il nome dell'utente che ha interagito
        SELECT NomeUtente
        INTO NomeUtenteInterazione
        FROM Utente
        WHERE IdUtente = NEW.IdUtente;

        --Salvo il cognome dell'utente che ha interagito
        SELECT CognomeUtente
        INTO CognomeUtenteInterazione
        FROM Utente
        WHERE IdUtente = NEW.IdUtente;

        --Salvo l'id del Creatore del Post
        SELECT IdUtente
        INTO CreatorePost
        FROM Post
        WHERE IdPost= NEW.IdPostCommentato;
```

```
        --Se il creatore post ha commentato sotto il suo post non deve ricevere la notifica
        IF CreatorePost = NEW.IdUtente THEN
                RETURN NULL;
        END IF;

        Messaggio:= 'L''utente '|| NomeUtenteInterazione ||' '|| CognomeUtenteInterazione||'ha
        commentato il tuo post.';

        --Creo e salvo l'id della notifica appena generata
        INSERT INTO
        Notifica(DataInvio,OraInvio,TestoNotifica,TipoNotifica,IdNuovoCommento)
        VALUES
        (CURRENT_DATE,CURRENT_TIME,Messaggio,'Interazione',NEW.IdCommento)
        RETURNING IdNotifica INTO NewIdNotifica;

        --Invio della notifica al creatore del post
        INSERT INTO Ricevere
        VALUES(CreatorePost, NewIdNotifica);
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER InviaNotificaInterazioneCommento
AFTER INSERT ON Commento
FOR EACH ROW
EXECUTE FUNCTION NotificaInterazioneCommento();
```

- **OttieniStatisticheGruppo**

```
--La seguente procedure si occupa di fornire delle statistiche mensili relative ad un gruppo, i
--parametri di ingresso sono idGruppo ed un mese e anno di riferimento, la funzione restituirà
--numero di post, commenti, like ai post , relativi al periodo di riferimento

CREATE OR REPLACE PROCEDURE OttieniStatisticheGruppo(IN idGruppoin
Gruppo.idGruppo%TYPE,IN mese INT,IN anno INT,OUT numero_post INT,
OUT numero_commenti INT,OUT numero_like INT)
AS $$
BEGIN
        SELECT COUNT(*) INTO numero_post
        FROM Post
        WHERE IdGruppo = idGruppoin
         AND EXTRACT(MONTH FROM DataPubblicazione) = mese
         AND EXTRACT(YEAR FROM DataPubblicazione) = anno;

        SELECT COUNT(*) INTO numero_commenti
        FROM Commento AS C , Post AS P
        WHERE P.IdGruppo = idGruppoin
         AND C.idPost=P.idPost
         AND EXTRACT(MONTH FROM P.DataPubblicazione) = mese
         AND EXTRACT(YEAR FROM P.DataPubblicazione) = anno;

        SELECT COUNT(*) INTO numero_like
        FROM Like_
        WHERE IdPost IN (SELECT IdPost FROM Post WHERE IdGruppo = idGruppo
                         AND EXTRACT(MONTH FROM DataPubblicazione) = mese
                         AND EXTRACT(YEAR FROM DataPubblicazione) = anno);
        RAISE NOTICE 'Numero commenti : %  Numero post : %  Numero like per i post:
        %',numero_commenti,numero_post,numero_like;
END;
$$ LANGUAGE plpgsql;
```

- **RecuperaIdUtenteConNickname**

--La seguente funzione la utilizziamo per popolare il database

```
CREATE OR REPLACE FUNCTION RecuperaIdUtenteConNickname(NicknameIN IN
Utente.Nickname%TYPE)
RETURNS INT AS $$
DECLARE
	varIdUtente Utente.IdUtente%TYPE;
BEGIN
	SELECT IdUtente
	INTO varIdUtente
	FROM Utente
	WHERE Nickname=NicknameIN;

	IF varIdUtente IS NULL
	THEN
		RAISE EXCEPTION 'L''utente % non esiste nel database',NicknameIN;
	END IF;
	RETURN varIdUtente;
END;
$$ LANGUAGE plpgsql;
```

- **RecuperaIdCreatoreConNickname**

--La seguente funzione è utilizzata nel popolamento del DB

```
CREATE OR REPLACE FUNCTION RecuperaIdCreatoreConNickname(NicknameIN IN
Utente.Nickname%TYPE,IdGruppoIN IN Gruppo.IdGruppo%TYPE)
RETURNS INT AS $$
DECLARE
	varIdUtente Utente.IdUtente%TYPE;
	varIdCreatore CreatoreGruppo.IdCreatore%TYPE;
BEGIN
	SELECT RecuperaIdUtenteConNickname(NicknameIN)
	INTO varIdUtente;

	SELECT IdCreatore
	INTO varIdCreatore
	FROM CreatoreGruppo
	WHERE IdUtente=varIdUtente AND IdGruppo=IdGruppoIN;

	IF varIdCreatore IS NULL
	THEN
		RAISE EXCEPTION 'L''utente inserito non è un creatore di un gruppo';
	END IF;
	RETURN varIdCreatore;
END;
$$ LANGUAGE plpgsql;
```

- **IscriviUtente**

--La seguente funzione è utilizzata nel popolamento del DB, per iscrivere un utente ad un gruppo

```
CREATE OR REPLACE FUNCTION IscriviUtente(IdUtenteIN IN
Utente.IdUtente%TYPE,IdCreatoreIN IN CreatoreGruppo.IdCreatore%TYPE,IdGruppoIN IN
Gruppo.IdGruppo%TYPE)
RETURNS VOID AS $$
BEGIN
        INSERT INTO RichiestaDiAccesso(StatoRichiesta,IdUtenteRichiesta,IdCreatore,
         IdGruppoRichiesta)
        VALUES ('Accettato',IdUtenteIN,IdCreatoreIN,IdGruppoIN);
END;
$$ LANGUAGE plpgsql;
```

- **ControllaDataOraPost**

--La seguente funzione si occupa di controllare che l'inserimento di un post avvenga rispettando la
--data e l'orario attuali, di conseguenza vieta inserimenti di post in date passate o future.

```
CREATE OR REPLACE FUNCTION ControllaDataOraPost()
RETURNS TRIGGER AS $$
BEGIN
        IF (NEW.DataPubblicazione <> CURRENT_DATE) THEN
                RAISE EXCEPTION  'Impossibile inserire un post che non abbia la data odierna';
        ELSIF (NEW.OraPubblicazione <> CURRENT_TIME) THEN
                RAISE EXCEPTION 'Impossibile inserire un post che non abbia l''orario corrente';
        END IF;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER ControllaDataOraPost
BEFORE INSERT ON Post
FOR EACH ROW
EXECUTE FUNCTION ControllaDataOraPost();
```

- **DataOraPostNonModificabile**

```
--La data e l'orario di un post non possono essere modificati una volta
--che sono stati inseriti in fase di creazione del post

CREATE OR REPLACE FUNCTION DataOraPostNonModificabile()
RETURNS TRIGGER AS $$
BEGIN
        IF (OLD.OraPubblicazione IS DISTINCT FROM NEW.OraPubblicazione
        OR
        OLD.DataPubblicazione IS DISTINCT FROM NEW.DataPubblicazione )
        THEN
                RAISE EXCEPTION 'Non è consentito modificare data o orario.';
        END IF;
        RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE TRIGGER DataOraPostNonModificabile
BEFORE UPDATE ON Post
FOR EACH ROW
EXECUTE FUNCTION DataOraPostNonModificabile();
```