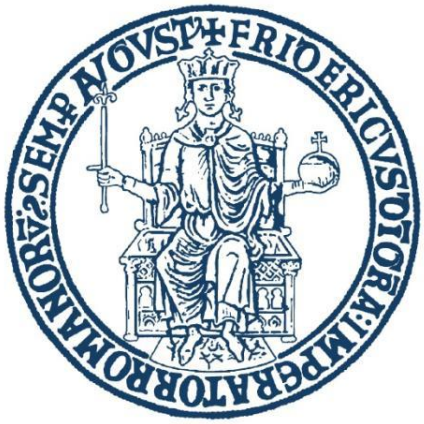


UNIVERSITÀ DEGLI STUDI
DI NAPOLI FEDERICO II

UninaSocialGroup

Esame di Basi di Dati I

Anno Accademico
2023/2024

Gruppo
SIMONE CATENACCIO N86004634
LUIGI DE FALCHI N86004546

Indice

1 Presentazione del progetto

1.1 UninaSocialGroup in due parole

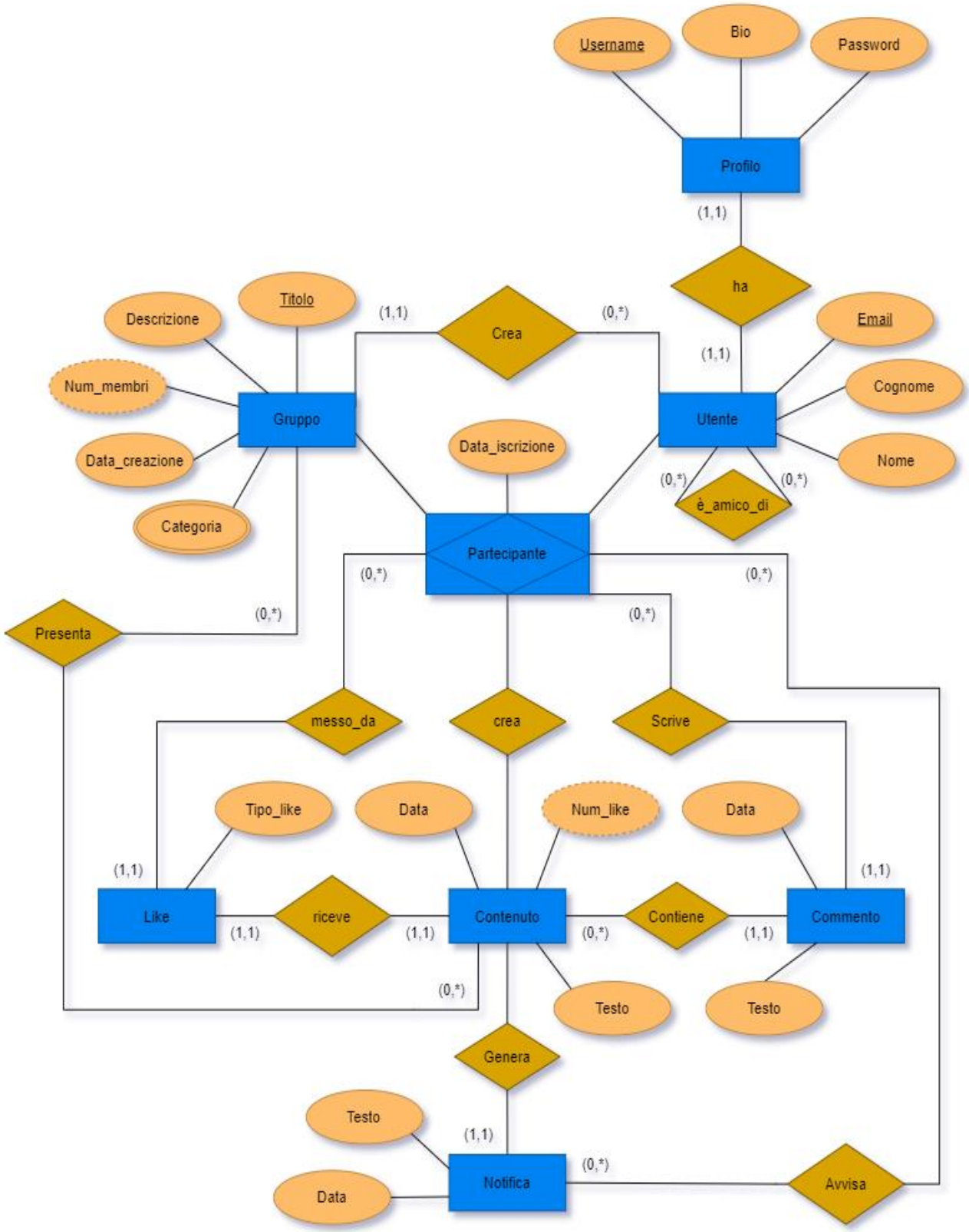
UninaSocialGroup è un social network messo a disposizione per gli studenti della Federico II, un modo per condividere con semplicità i propri interessi, pareri e domande nei diversi gruppi disponibili. Gli utenti potranno iscriversi o creare gruppi affinché rispecchino al meglio i propri interessi creando anche contenuti, commentando e lasciando “mi piace”.

1.2 Un primo sguardo

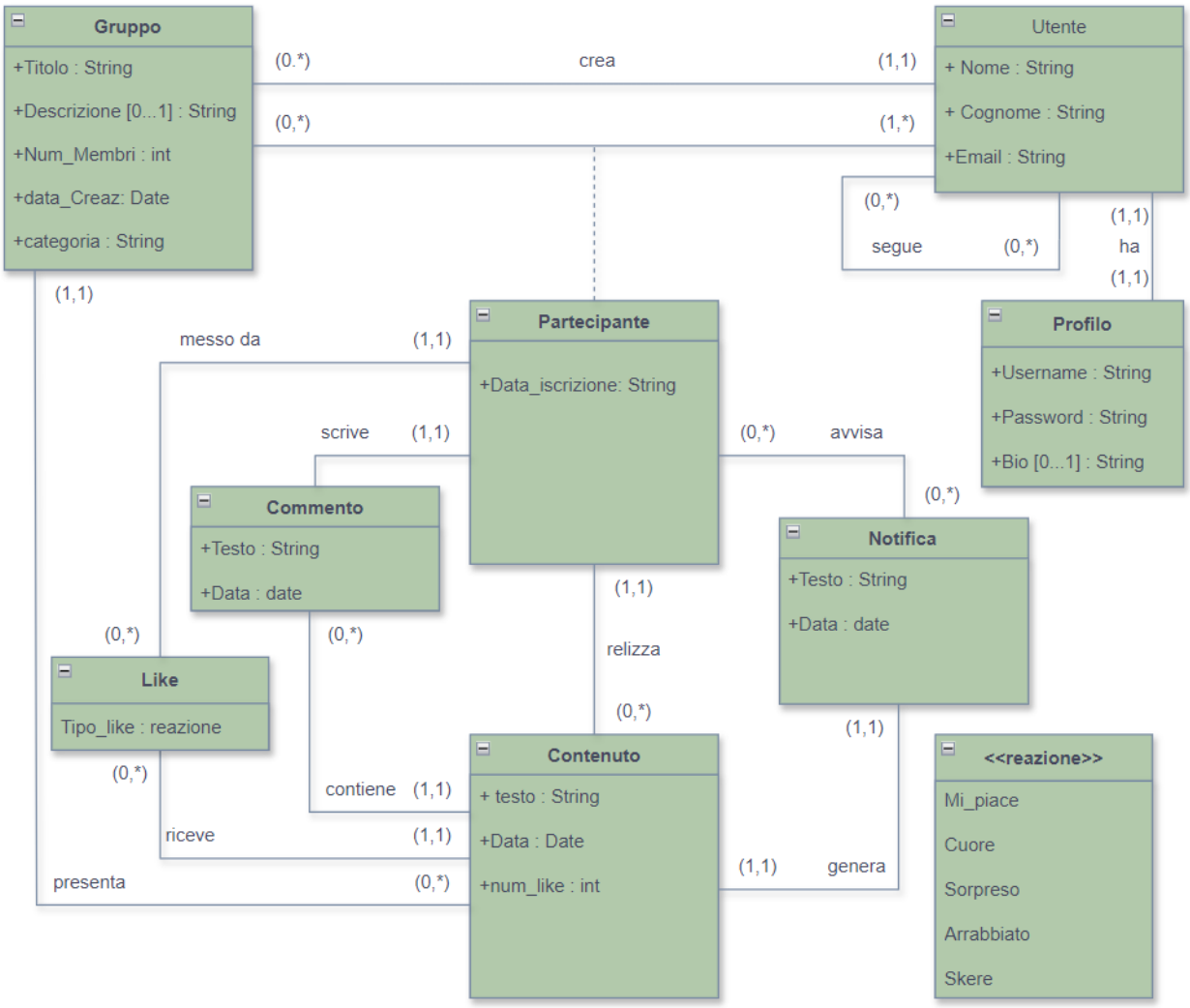
Un individuo ha la possibilità di aderire a diversi gruppi o di formarne di nuovi. Ciascun gruppo è identificato da un nome unico e uno o più argomenti generali, che aiutano gli utenti a identificare la natura dei contenuti condivisi all’interno del gruppo. I membri di un gruppo hanno la possibilità di visualizzare e condividere post, oltre a interagire con i post degli altri utenti attraverso “mi piace” o commenti. L’utente riceve una notifica ogni volta che un nuovo post viene pubblicato in un gruppo di cui è membro. Ogni post è composto da informazioni scritte, che possono essere eventualmente integrate con immagini.

2 Progettazione Concettuale

2.1 Diagramma ER



2.2 Diagramma UML

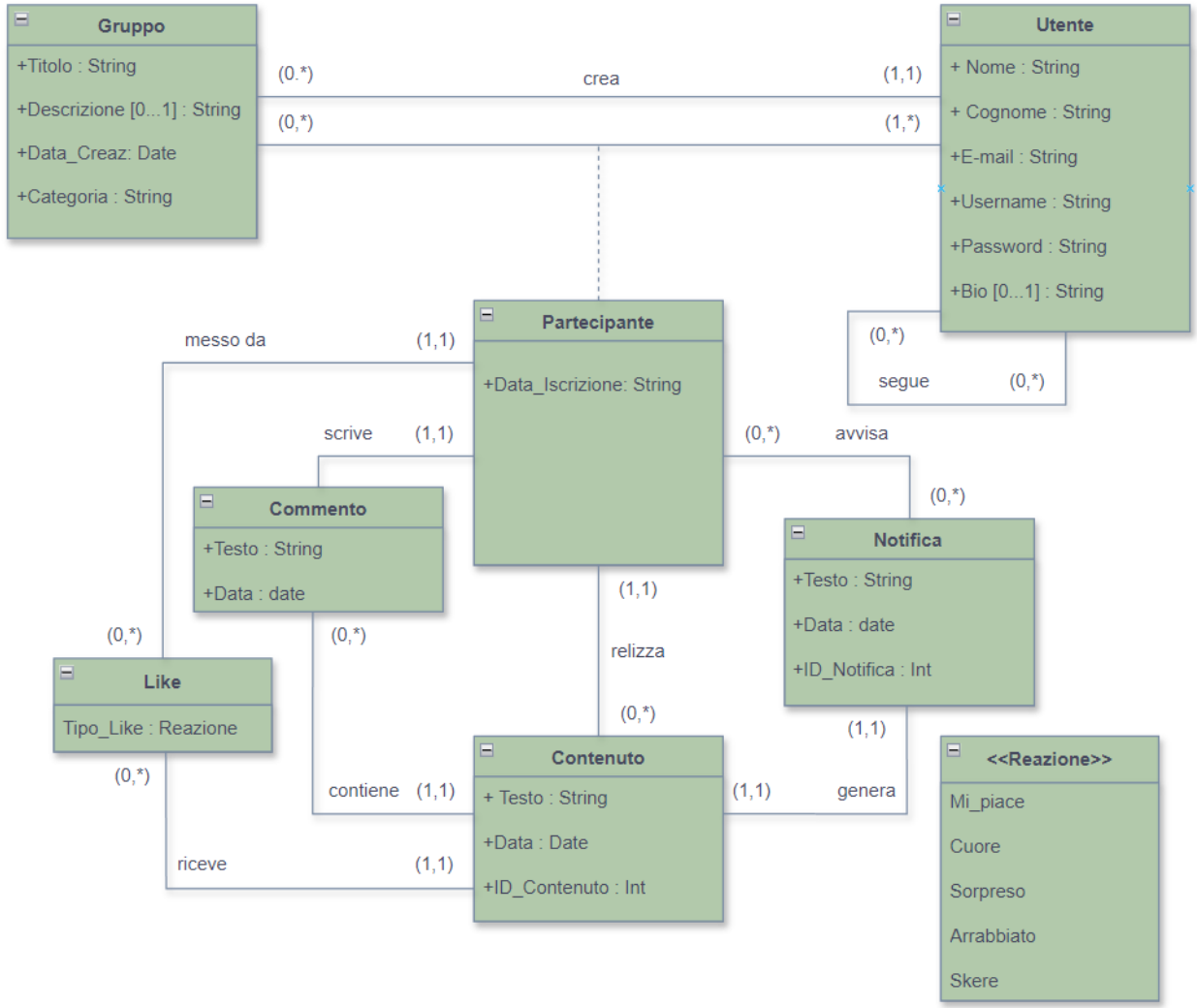


2.2.1 Precisazioni sul Class Diagram

“Partecipante” è una classe di associazione : un Utente può creare o partecipare ad uno o più gruppi contemporaneamente .

“Reazione” è una classe che si riferisce a “like” specificando i diversi tipi di “like” che si possono inserire.

2.3 Ristrutturazione del Class Diagram



2.3.1 Accorpamento entità Utente-Profilo

Durante il processo di ristrutturazione, abbiamo accorpato la classe ''Profilo'' in ''Utente''. Essendo una relazione 1 a 1, ed entrambi sono indispensabili per le informazioni di un utente, abbiamo preso in considerazione questa scelta.

2.3.2 Eliminazione attributi derivabili da altre entità/associazioni

Nella rappresentazione concettuale, abbiamo inserito degli attributi che possono essere facilmente calcolabili da alcune entità.
Nel dettaglio abbiamo l'attributo ''Num_Membri'' della classe Gruppo : è un contatore che aggiorna in tempo reale quanti membri ci sono nel gruppo. Può essere tranquillamente verificato controllando le istanze nella tabella Partecipante.

Il discorso è analogo per l'attributo ''Num_Like'' della tabella Contenuto, che può essere facilmente calcolato dalle istanze della tabella Like.

2.3.3 Eliminazione associazione "Presenta" tra ''Gruppo'' e ''Contenuto''

Nonostante l'associazione in questione sia importante per comprendere che i contenuti vengono visualizzati all'interno dei gruppi, abbiamo comunque valutato di eliminare quest' ultima, perché potrebbe avere un impatto sulle prestazioni. Dato che il contenuto ricava il gruppo di appartenenza dal partecipante che lo crea, avere quest'associazione poterebbe ad avere due volte l'attributo gruppo, una volta dal partecipante, e l'altra dal gruppo; dunque, abbiamo valuto più conveniente farne a meno.

3 Dizionari

3.1 Dizionario delle Classi

Classe	*Descrizione*	*Attributi*
Utente	Descrittore del profilo di una persona per interagire con il sito.	**Username** (*string*): Chiave tecnica. Identifica univocamente ciascuna istanza di username per ogni utente. **Nome** (*string*): Nome reale associato a utente. **Cognome** (*string*): Cognome reale associate a utente. **E-mail** (*string*): Chiave tecnica. Identifica univocamente ciascuna istanza di e-mail per ogni utente. **Password** (*String*): Insieme di caratteri per accedere al proprio profilo. **Bio** (*string*): Insieme di informazioni che l'utente sceglie di mettere.
Gruppo	Insieme di utenti che interagiscono tra di loro, condividendo interessi, valori e obbiettivi comuni.	**Titolo** (*string*): Chiave tecnica. Identificatore univoco per il titolo del gruppo. **Descrizione** (*string*): Insieme di informazioni che descrivono regole e comportamenti da rispettare nel gruppo. **Data_Creazione** (*date*): Data di creazione del gruppo. **Categoria** (*string*): Argomento di dibattito del gruppo.
Partecipante	Utente che entra a far parte di un gruppo.	**Data_Iscrizione** *(string)*: Data in cui l'utente partecipa al gruppo.
Contenuto	Contenuto che ogni utente presente in un gruppo può condividere.	**ID_Contenuto** (*int*): Chiave tecnica. Identifica univocamente l'ID di ogni contenuto pubblicato nel gruppo. **Testo** (*string*): Testo che l'utente può scrivere all'interno del contenuto condiviso. **Data** (*date*): Data di pubblicazione del contenuto.

Commento	Risposta di un utente al contenuto di un altro.	**Testo** (*string*): Testo che inserisce l'utente come risposta ad un contenuto. **Num_Like** (*int*): Numero di like che riceve il commento postato dall'utente.
Notifica	Notifica pop-up che arriva all'utente quando ci sono interazioni con il gruppo	**ID_Notifica** *(int)*: Chiave tecnica. Identifica univocamente l'ID di ogni notifica che avvisa gli utenti. **Testo** (*string*): Testo che avvisa l'utente della pubblicazione di un contenuto. **Data** *(date)*: data di arrivo della notifica.

3.2 Dizionario delle Associazioni

Associazione	*Descrizione*	*Classi coinvolte*
Segue	Un utente può seguire zero o più utenti.	Utente [0…*] - Utente [0…*]
Partecipa		
Messo_da	Un like è messo da un solo partecipante, e un partecipante può mettere più like (o nessuno).	Like [1] - Partecipante [0…*]
Riceve	Un like è messo ad un solo contenuto, e un contenuto può ricevere più like (o nessuno).	Like [1] - Contenuto [0…*]
Crea	Un partecipante può creare molti contenuti (o nessuno), e un contenuto è creato da un solo partecipante.	Partecipante [0…*] - Contenuto [1]
Scrive	Un partecipante può scrivere molti commenti (o nessuno), e un commento è scritto da un solo partecipante.	Partecipante [0…*] - Commento [1]
Contiene	Un contenuto può contenere molti	Contenuto [0…*] - Commento [1]

	commenti (o nessuno), e un commento è presente in un solo contenuto.	
Avvisa	Una notifica avvisa molti partecipanti (o nessuno), e un partecipante può essere avvisato da molte notifiche (o nessuna).	Notifica [0…1] - Partecipante [0…1]
Genera	La creazione di un contenuto genere una notifica, e una notifica parte alla creazione di un contenuto.	Notifica [1] - Contenuto [1]

3.3 Dizionario dei Vincoli

Vincolo	***Tipo***	***Descrizione***
Nome_Not_Null	Dominio	L'attributo ''Nome'' in Utente, non può essere NULL.
Cognome_Not_Null	Dominio	L'attributo ''Cognome'' in Utente, non può essere NULL.
Categoria_Not_Null	Dominio	L'attributo ''categoria'' in gruppo non può essere NULL.
Testo_Contenuto_Not_NULL	Dominio	L'attributo ''Testo in contenuto non può essere NULL.
Testo_Commento_Not_Null	Dominio	L'attributo ''Testo'' in commento non può essere NULL.
Tipo_Like_Not_Null	Dominio	L'attributo ''Tipo_like'' in Like non può essere NULL.
Lunghezza_Password	Dominio	La stringa dell'attributo ''password'' in Utente deve essere lunga almeno otto caratteri.
Partecipanti_Unici	Intrarelazionale	Non possono esistere due partecipanti in un gruppo con la stessa e-mail.

Admin_Partecipante	interrelazionale	Alla creazione di un gruppo, l'admin ne sarà automaticamente partecipante.
Aggiorna_E-mail_Admin	Interrelazionale	Se il partecipante ''admin abbandona il gruppo, il partecipante meno recente dello stesso gruppo diventa l'admin.
Elimina_Gruppo_Vuoto	Interrelazionale	Se l'unico partecipante esce dal gruppo, questo si elimina.
Genera_Notifica	Interrelazionale	Se un partecipante di un gruppo crea un contenuto, si crea una notifica.
Avvisa_Utenti	Interrelazionale	Quando si genera una notifica, essa avrà il compito di avvisare tutti gli utenti di quel gruppo, tranne il creatore del post stesso.
Verifica_Commento	Interrelazionale	Se la persona che commenta un post, non partecipa al gruppo in cui è stato condiviso, impedirà l'inserimento e segnalerà un errore.
Verifica_Like	Interrelazionale	Se la persona che mette like a un post, non partecipa al gruppo in cui è stato condiviso, impedirà l'inserimento e segnalerà un errore.
Solo_Un_Like	Intrarelazionale	Un partecipante può mettere like a un contenuto una sola volta.

4 Progettazione Logica

4.1 Tabelle

Leggenda:

- Gli attributi in **grassetto** sono le chiavi primarie;
- Gli attributi **sottolineati** sono le chiavi esterne;

- Gli attributi sottolineati sono le chiavi con più attributi;
- Le parole [……] sono le tabelle di riferimento;

Nome	*Attributi*
Utente	**E-mail**, Nome, Cognome, Username, Password, Bio
Gruppo	**Titolo**, Descrizione, Num_Membri, Data_Creaz, Categoria, **E-mail_Admin**
Partecipante	**Titolo_Gruppo** [Gruppo], **E-mail** [Utente], Data_Iscrizione
Contenuto	**ID_Contenuto**, Testo, Data, Num_Like, **E-mail_Creator**, **Gruppo_App**
Commento	**ID_Contenuto**, **E-mail_Partecipante**, **Gruppo_Rifer** [Partec], Testo, Data
Notifica	**ID_Notifica**, **ID_Contenuto**, Data, Testo
Avvisa	**E-mail_Destinatario**, **Gruppo_di_Avviso**, **ID_Notifica**
Like	**ID_Contenuto**, **Titolo_Gruppo**, **E-mail_Partecipante**, Tipo_Like
Segue	**E-mail_Utente_1**, **E-mail_Utente_2**

4.2 Traduzione delle associazioni in tabelle

Associazione	**Implementazione**
crea	Chiave esterna in Gruppo → Utente
Partecipante ←	Chiave esterna in Partecipante → Gruppo
Partecipante →	Chiave esterna in Partecipante → Utente
Messo_da	Chiave esterna in Like → Partecipante
riceve	Chiave esterna in Like → Contenuto
scrive	Chiave esterna in Commento → Partecipante

contiene	Chiave esterna in Commento → Contenuto
realizza	Chiave esterna in Contenuto → Partecipante
genera	Chiave esterna in Notifica → Contenuto
Avvisa →	Chiave esterna in Avvisa → Notifica
Avvisa ←	Chiave esterna in Avvisa →Partecipante
Segue ←	Chiave esterna in Segue → Utente
Segue →	Chiave esterna in Segue → Utente

4.3 Scelta delle Primary Key

In alcune tabelle avremmo potuto utilizzare un insieme di attributi per identificare un'istanza dall'altra, ma per comodità, abbiamo deciso di introdurre una chiave surrogata, ovvero un semplice numero (ID) che viene utilizzato come identificativo univoco tra le istanze di una tabella. Ci siamo mossi in questo modo; con le seguenti tabelle "Contenuto" e "Notifica".

Abbiamo cercato di evitare il più possibile quest'approccio, per esempio abbiamo scelto di lasciare ''Titolo'' in Gruppo come chiave tecnica, dato che nel nostro database non possono esistere più gruppi con lo stesso Titolo.

Per comodità, in "Partecipante", abbiamo scelto di usare le chiavi esterne, ricavate dalle tabelle "Utente" e "Gruppo" come sue chiavi tecniche.

5 Progettazione Fisica

Per la progettazione fisica è stato utilizzato DataGrip, un'interfaccia grafica per l'amministrazione di database di PostgreSQL.

5.1 Creazione tabelle

5.1.1 Utente

```
create table utente
(
    email    varchar(100)
        primary key,
    nome     varchar(30)  not null,
    cognome  varchar(30)  not null,
    username varchar(100)
        unique,
    password varchar(30)  not null,
    bio      varchar(100)
);

alter table utente
    owner to postgres;
```

5.1.2 Gruppo

```
create table gruppo
(
    titolo varchar(30) primary key,
    descrizione    varchar(255),
    data_creazione date not null,
    categoria      varchar(50) not null,
    email_admin    varchar(100),
    constraint data_creazione check (data_creazione <= CURRENT_DATE),
    constraint fk_email_admin foreign key (email_admin) references utente(email)
            on delete set null
);

alter table gruppo
    owner to postgres;
```

5.1.3 Partecipante

```
create table partecipante
(
    data_iscrizione    date not null ,
    email_partecipante varchar(100),
    titolo_gruppo      varchar(30),
    constraint fk_titolo_gruppo foreign key (titolo_gruppo) references gruppo(titolo)
            on delete cascade,
     constraint fk_email_partecipante foreign key(email_partecipante) references utente (email)
            on delete cascade,
    constraint pk_partecipante
        primary key (email_partecipante, titolo_gruppo)
);

alter table partecipante
    owner to postgres;

```

5.1.4 Commento

```
create table commento
(
    id_contenuto       integer,
    email_partecipante varchar(100),
    gruppo_riferimento varchar(30),
    testo              varchar(30)  not null,
     constraint fk_id_contenuto foreign key (id_contenuto)references contenuto(id_contenuto)
           on delete cascade,
    constraint fk_partecipante foreign key (email_partecipante, gruppo_riferimento) references partecipante
           on delete cascade
);

alter table commento
    owner to postgres;

```

5.1.5 Contenuto

```
create table contenuto
(
    id_contenuto serial
        primary key,
    testo        varchar(300) not null,
    data         date         not null,
    gruppo_app   varchar(30),
    email_utente varchar(100),
    constraint fk_contenuto
        foreign key (gruppo_app, email_utente) references partecipante (titolo_gruppo,email_partecipante)
            on delete cascade
);

alter table contenuto
    owner to postgres;
```

5.1.6 Like

La funzione “Like” è stata chiamata “Mi_Piace” per problemi di codice in DataGrip.

```
create table mi_piace
(
    id_contenuto       integer,
    titolo_gruppo      varchar(30),
    email_partecipante varchar(100) ,
    tipo_like          social_group.reazione not null,
    constraint uq_like unique (id_contenuto, titolo_gruppo, email_partecipante),
    constraint fk_partecipante foreign key (titolo_gruppo, email_partecipante) references partecipante (titolo_gruppo,email_partecipante),
     constraint fk_id_contenuto foreign key (id_contenuto)references contenuto (id_contenuto)
);

alter table mi_piace
    owner to postgres;
```

5.1.7 Tipo “Like”[enum]

```
create domain reazione as varchar(20)
    constraint reazione_check check ((VALUE)::text = ANY
                                     ((ARRAY ['Mi_piace'::character varying,
                                       'Cuore'::character varying, 'Sorpreso'::character varying,
                                       'Arrabbiato'::character varying])::text[]));

alter domain reazione owner to postgres;
```

5.1.8 Notifica

```
create table notifica
(
    id_notifica  serial primary key,
    id_contenuto integer,
    testo        varchar(100),
    data         date not null
        constraint controllo_data
            check (data <= CURRENT_DATE),
     constraint fk_id_contenuto foreign key (id_contenuto) references contenuto(id_contenuto)
            on delete cascade
);

alter table notifica
    owner to postgres;
```

5.1.9 Avvisa

```
create table avvisa
(
    email_destinatario     varchar(100),
    gruppo_da_cui_avvisato varchar(30),
    id_notifica integer,
    constraint fk_partecipante foreign key (email_destinatario, gruppo_da_cui_avvisato) references partecipante
            on delete cascade,
     constraint fk_id_notifica foreign key (id_notifica) references notifica(id_notifica)
            on delete cascade,
    constraint uq_avvisa unique (email_destinatario,id_notifica)
);

alter table avvisa
    owner to postgres;

```

5.1.10 Segue

```
CREATE TABLE segue
(
    email_utente1 VARCHAR(100),
    email_utente2 VARCHAR(100),
    constraint fk_email_utente1 foreign key (email_utente1) REFERENCES utente(email)
        on delete cascade ,
    constraint fk_email_utente2 foreign key (email_utente2) REFERENCES utente(email)
        on delete cascade ,
    constraint uq_segue unique (email_utente1,email_utente2)
);

```

5.2 Funzioni e Triggers

5.2.1 Admin_Partecipante

```
create function admin_partecipante() returns trigger
    language plpgsql
as
$$
    BEGIN
        --inserisco nella tabella partecipante l'admin, non appena ha creato il gruppo
    INSERT INTO partecipante
        VALUES(new.data_Creazione,new.email_admin, new.titolo);
        return new;
    end;
$$;

alter function admin_partecipante() owner to postgres;

```

5.2.2 Tr_Admin_Partecipante

```
create trigger tr_admin_partecipante
    after insert
    on gruppo
    for each row
execute procedure admin_partecipante();

```

5.2.3 Aggiorna_E-mail_Admin

```
create function aggiorna_email_admin() returns trigger
    language plpgsql
as
$$
    DECLARE
      partecipanti INT := 0;
BEGIN
        --mi faccio un count sui partecipanti
    SELECT COUNT(*) INTO partecipanti
    FROM social_group.partecipante
    WHERE Titolo_gruppo = old.titolo_gruppo;

    IF EXISTS (
        SELECT 1
        FROM social_group.gruppo
        WHERE titolo = old.titolo_gruppo
          AND email_admin = OLD.email_partecipante
        --controllo se è uscito il partecipande admin
    )
        AND partecipanti > 0
        --controllo se c'è almeno un partecipante

        THEN
            --aggiorno admin
        UPDATE social_group.gruppo
        SET Email_admin = (
            SELECT email_partecipante
            FROM social_group.partecipante
            WHERE Titolo_gruppo = OLD.Titolo_Gruppo
            ORDER BY data_iscrizione DESC
            LIMIT 1)
        WHERE titolo = old.Titolo_gruppo;

    END IF;
    RETURN NEW;
END;
```

5.2.4 Tr_Aggiorna_E-mail_Admin

```
create trigger tr_aggiorna_email_admin
    after delete
    on partecipante
    for each row
execute procedure aggiorna_email_admin();
```

5.2.5 Date_Coerenti

```
create function date_coerenti() returns trigger
    language plpgsql
as
$$
BEGIN
    -- Ottieni la data del contenuto corrispondente
    DECLARE
        data_contenuto DATE;
    BEGIN
        SELECT data INTO data_contenuto
        FROM contenuto
        WHERE id_contenuto = NEW.id_contenuto;

        -- Verifica se la data del commento è più recente
        IF NEW.data <= data_contenuto THEN
            RAISE EXCEPTION 'La data del commento deve essere più recente della data del contenuto';
        END IF;
    END;
    RETURN NEW;
END;
$$;

alter function date_coerenti() owner to postgres;
```

5.2

```
-- auto-generated definition
create trigger tr_date_coerenti
    before insert
    on commento
    for each row
execute procedure date_coerenti();
```

5.2.7 Elimina_Gruppo_Vuoto

```
create function elimina_gruppo_vuoto() returns trigger
    language plpgsql
as
$$
DECLARE
    partecipanti INT := 0;
BEGIN
    --controllo quanti partecipanti ci sono.
    SELECT COUNT(*) INTO partecipanti
    FROM social_group.partecipante
    WHERE Titolo_gruppo = old.titolo_gruppo;

--se sono 0, elimino il gruppo
    IF partecipanti = 0 THEN
        DELETE FROM social_group.gruppo
        WHERE Titolo = old.titolo_gruppo;
    END IF;

    RETURN NEW;
END;
$$;

alter function elimina_gruppo_vuoto() owner to postgres;
```

5.2.8 Tr_Elimina_Gruppo_Vuoto

```
create trigger tr_elimina_gruppo_vuoto
    after delete
    on partecipante
    for each row
execute procedure elimina_gruppo_vuoto();

```

5.2.9 Genera_Notifica

```
create function genera_notifica() returns trigger
    language plpgsql
as
$$

        DECLARE
        gruppo varchar(30);

        BEGIN
            --mi salvo la variabile per printare il gruppo nella stringa testo
           SELECT gruppo_app INTO gruppo
            FROM social_group.contenuto
            WHERE id_contenuto =new.id_contenuto;

        INSERT INTO notifica(id_contenuto,testo,data)VALUES
        (new.id_contenuto,'è appena stato caricato un contenuto in: '||gruppo||'.',now());


        RETURN NEW;



        end;
      $$;

alter function genera_notifica() owner to postgres;

```

5.2.10 Tr_Genera_Notifica

```
-- auto-generated definition
create trigger tr_genera_notifica
    after insert
    on contenuto
    for each row
execute procedure genera_notifica();

```

5.2.11 Like_Coerenti

```
create function like_coerenti() returns trigger
    language plpgsql
as
$$
BEGIN

    DECLARE
        data_like DATE;
    BEGIN
        SELECT data INTO data_like
        FROM contenuto
        WHERE id_contenuto = NEW.id_contenuto;


        IF NEW.data <= data_like THEN
            RAISE EXCEPTION 'La data del like deve essere più recente della data del contenuto';
        END IF;

        RETURN NEW;
    END;
END;
$$;

alter function like_coerenti() owner to postgres;
```

5.2.12 Tr_Like_Coerenti

```
create trigger tr_like_coerenti
    before insert
    on mi_piace
    for each row
execute procedure like_coerenti();
```

5.2.13 Lunghezza_Password

```
create function lunghezza_password() returns trigger
    language plpgsql
as
$$
BEGIN
    --controlla se la password è più lunga di 8 caratteri, altrimenti segnala un errore
    IF LENGTH(NEW.password) < 8 THEN
        RAISE EXCEPTION 'La password deve essere lunga almeno 8 caratteri';
    END IF;
    RETURN NEW;
END;
$$;

alter function lunghezza_password() owner to postgres;
```

5.2.14 Tr_Lunghezza_Password

```
create trigger tr_lunghezza_password
    before insert
    on utente
    for each row
execute procedure lunghezza_password();
```

5.2.15 Verifica_Commento

```
create function verifica_commento() returns trigger
    language plpgsql
as
$$
  DECLARE gruppo varchar(30);

BEGIN
    --mi salvo il gruppo di appartenenza del contenuto
    SELECT gruppo_app INTO gruppo
    FROM social_group.contenuto
    WHERE id_contenuto = NEW.id_contenuto;

    --se il gruppo di rifremento del commento è diverso da quello del contenuto, segnala l'errore
    IF NEW.gruppo_riferimento <> gruppo THEN
        RAISE EXCEPTION 'Prima di commentare, devi iscriverti al gruppo in cui è stato postato il contenuto';
    END IF;

    RETURN NEW;
END;
$$;

alter function verifica_commento() owner to postgres;
```

5.2.16 Tr_Verifica_Commento

```
create trigger tr_verifica_commento
    before insert
    on commento
    for each row
execute procedure verifica_commento();
```

5.2.17 Verifica_Mi_Piace

```
create function verifica_mi_piace() returns trigger
    language plpgsql
as
$$
    DECLARE gruppo varchar(30);
BEGIN

    SELECT gruppo_app INTO gruppo
    FROM social_group.contenuto
    WHERE id_contenuto = NEW.id_contenuto;

    -- Verifica se il gruppo di riferimento del partecipante è uguale al gruppo di riferimento del contenuto
    IF NEW.titolo_gruppo <> gruppo THEN
        RAISE EXCEPTION 'Prima di inserire una reazione, devi iscriverti al gruppo in cui è stato postato il contenuto';
    END IF;

    RETURN NEW;
END;
$$;

alter function verifica_mi_piace() owner to postgres;
```

5.2.18 Tr_Verifica_Mi_Piace

```
create trigger tr_verifica_mi_piace
    before insert
    on mi_piace
    for each row
execute procedure verifica_mi_piace();
```

5.2.19 Aggiorna_E-mail_Admin

```
create function aggiorna_email_admin() returns trigger
    language plpgsql
as
$$
    DECLARE
      partecipanti INT := 0;
BEGIN
        --mi faccio un count sui partecipanti
    SELECT COUNT(*) INTO partecipanti
    FROM social_group.partecipante
    WHERE Titolo_gruppo = old.titolo_gruppo;

    IF EXISTS (
        SELECT 1
        FROM social_group.gruppo
        WHERE titolo = old.titolo_gruppo
          AND email_admin = OLD.email_partecipante
        --controllo se è uscito il partecipande admin
    )
        AND partecipanti > 0
        --controllo se c'è almeno un partecipante

            THEN
                --aggiorno admin
            UPDATE social_group.gruppo
            SET Email_admin = (
                SELECT email_partecipante
                FROM social_group.partecipante
                WHERE Titolo_gruppo = OLD.Titolo_Gruppo
                ORDER BY data_iscrizione DESC
                LIMIT 1)
            WHERE titolo = old.Titolo_gruppo;

        END IF;
        RETURN NEW;
END;
$$;

alter function aggiorna_email_admin() owner to postgres;

```

5.2.20 Tr_Aggiorna_E-mail_Admin

```
create trigger tr_aggiorna_email_admin
    after delete
    on partecipante
    for each row
execute procedure aggiorna_email_admin();

```