UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II
SCUOLA POLITECNICA E DELLE SCIENZE DI BASE
DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

CORSO DI LAUREA IN INFORMATICA

Documentazione del Database UninaSocialGroup

Luglio 2024

Autori:
Gennaro De Luca (N86004471)
Gabriele Cifuni (N86004765)
Antonio Caruso (N86004571)

Gruppo OOBD2324_40

Indice

1 Presentazione ed Analisi del Progetto

1.1 Introduzione

UninaSocialGroup è un social network che offre agli utenti la possibilità di creare e partecipare a gruppi tematici, descritti da tag (es: sport, fantasy, anime, ecc.). Una volta accettati dal creatore del gruppo gli utenti possono condividere i loro pensieri, esperienze o interessi relativi alla categoria del gruppo pubblicando contenuti come foto e/o testo. Gli altri membri iscritti al gruppo hanno la possibilità di interagire con i contenuti attraverso commenti e "like". All'inserimento di un nuovo contenuto tutti gli iscritti al gruppo ricevono una notifica.

Specifiche solo per i gruppi con 3 membri

Aggiungere la possibilità di gestire le notifiche, un utente riceve una notifica ogni volta che:

- un utente accede ad un gruppo di cui si è il creatore;
- un utente ha creato un nuovo contenuto ad un gruppo di cui si è amministratori;
- un utente ha interagito ad un contenuto di cui si è autori.

Come primo approccio bisogna individuare le varie entità che compongono la nostra base di dati e le relazioni che hanno tra loro. Andremo quindi ad esaminare la traccia fornita, passo per passo, per scroprire le varie entità, associazioni ed eventuali vincoli.

1.2 Analisi della traccia

Dalle prime righe del testo notiamo subito l'apparizione degli utenti che possegono un proprio **Profilo**.
Succesivamente ci viene detto che gli utenti possono partecipare a dei **Gruppi** dove quest'ultimi possiedono dei **Tag** per descrivere il tema del gruppo.
Per partecipare ai gruppi il **Creatore** del gruppo deve accettare la richiesta mandata dall'utente, da qui possiamo già notare una **distinzione** tra Utente e Creatore, inoltre si devono gestire le **Notifiche delle richieste** di partecipazione ad gruppo.
Una volta accettati, gli utenti possono inserire nei gruppi dei **contenuti** dove è possibile interagire con quest'ultimi tramite **Like** e/o **Commenti**.
Quando un utente pubblica un contenuto su un gruppo, gli utenti che partecipano a quel gruppo devono ricevere una **notifica generale** che informa dell'insermineto del contenuto da parte dell'utente. Quando un creatore effettua il login e và online, tutti gli utenti che partecpiano ai gruppi creati dal creatore ricevono una notifica che avvisa che quest'ultimo è online.
La traccia ci fa notare un altro tipo di distinzione di un utente, cioè **l'amministratore**, una figura che gestisce i contenuti presenti nel gruppo.
Infine, un utente riceve una **notifica relativa al contenuto** se altri utenti interagiscono con like e/o commenti sul suo contenuto.

Osservando questa analisi otteniamo:

- L'entità "Profili" che si suddivide in Utenti, Amministratori, Creatori
- L'entità "Gruppi"
- L'entità "Tags"
- L'entità "Notifiche_Richieste"
- L'entità "Contenuti"
- L'entità "Likes"
- L'entità "Commenti"
- L'entità "Notifiche_Gruppi"
- L'entità "Notifiche_Contenuti"

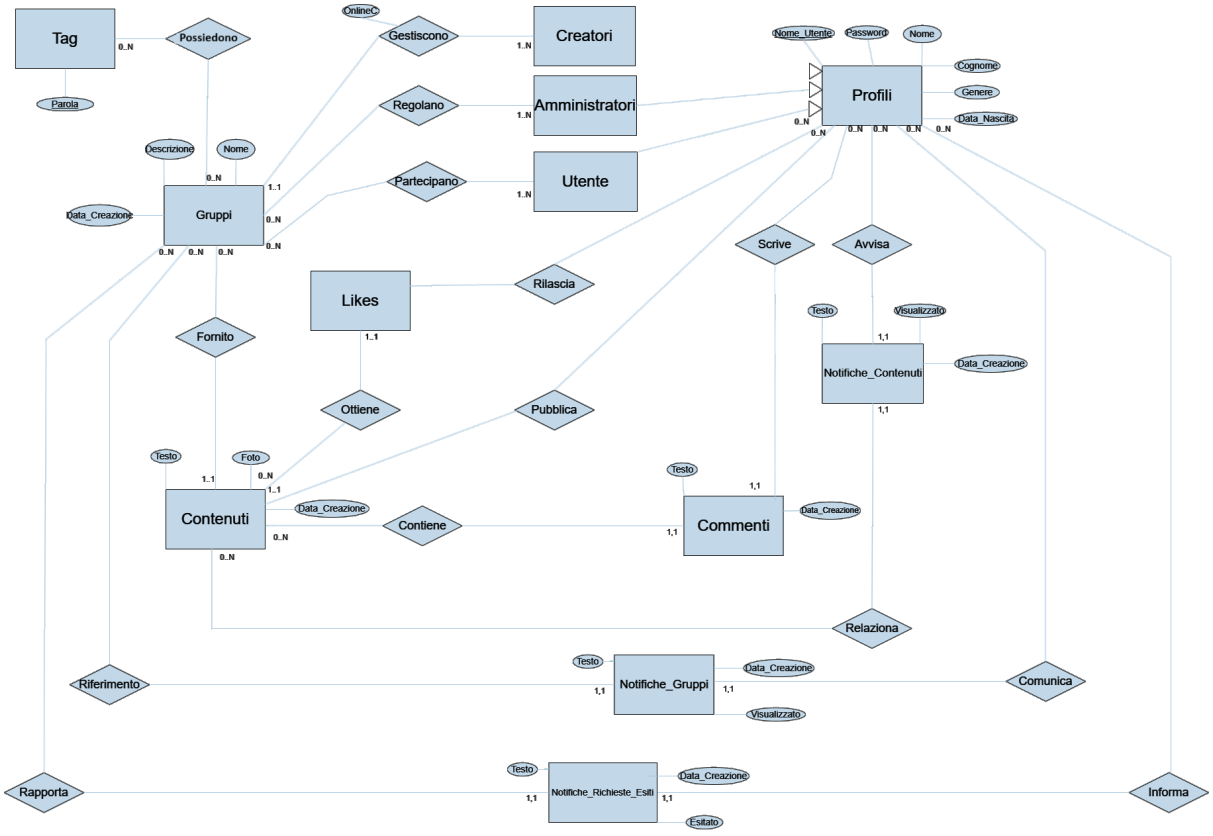
Con queste informazioni possiamo già comporre una prima versione del diagramma UML (non ristrutturato) e il Diagramma ER.

2 Progettazione concettuale

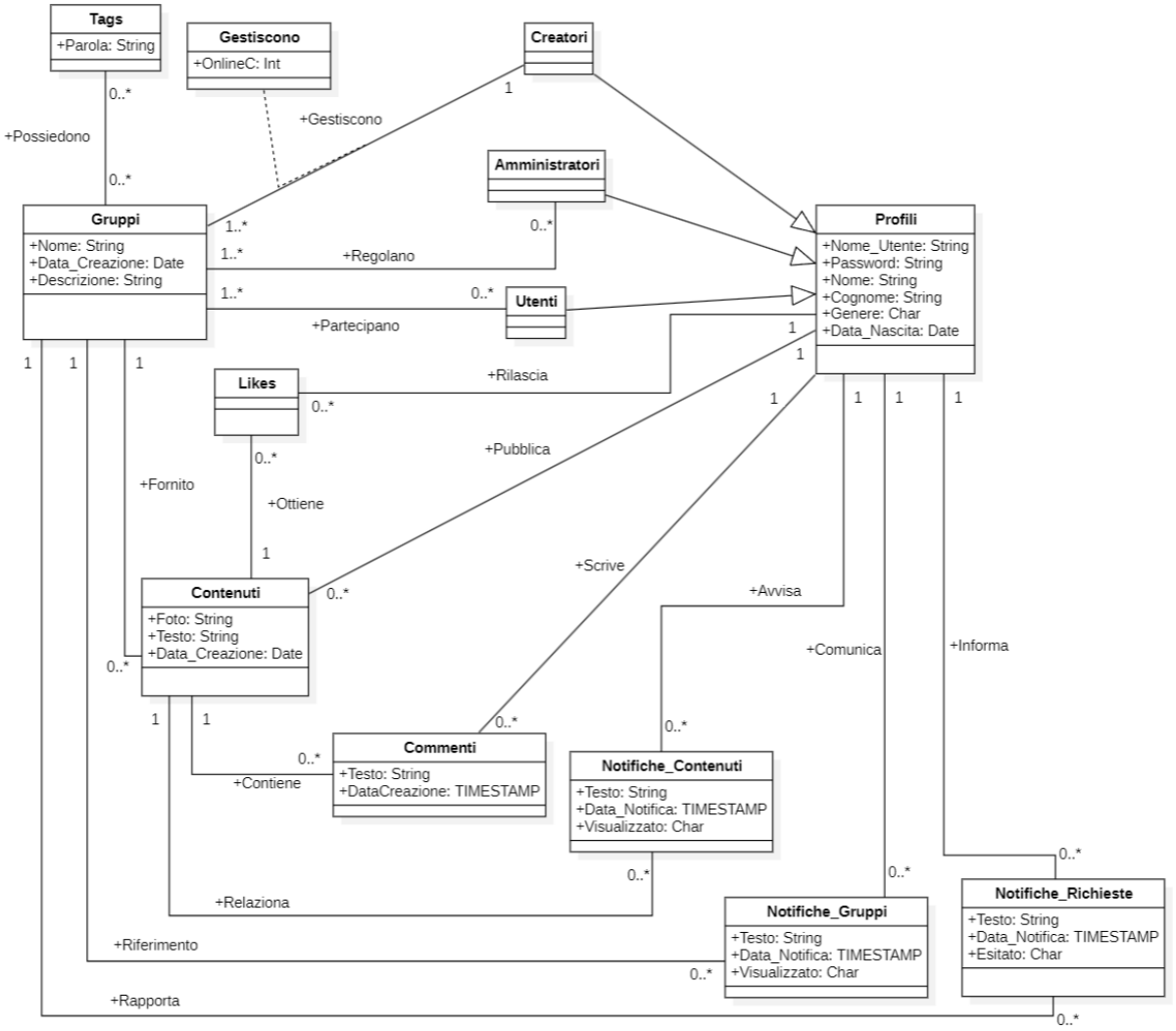
In questa frase andremo a progettare i vari schemi (UML, ER) che ci servono per fornire una rappresentazione chiara e concisa della struttura del database, mostrando entità, attributi e relazioni, ignorando dettagli implementativi. Questi schemi ci permettono anche di identificare i vari vincoli e di descriverli.

Le entità, gli attributi, le relazioni e i vincoli verranno descritti in maniera dettagliatta nei vari **Dizionari** presenti nel capitolo successivo.

2.1 Diagramma ER



2.2 Diagramma UML



3 Ristrutturazione del modello concettuale

3.1 Procedimento

In questa fase, si ha lo scopo di rendere il Diagramma UML idoneo per la traduzione in schemi relazionali e di migliorare l'efficienza dell'implementazione, si procede alla **ristrutturazione** dello stesso. Al termine di questo operazione il Diagramma UML non conterrà alcun attributo multiplo, composto, eventuali specializzazioni o generalizzazioni e si procederà all'inserimento di chiavi surrogate.

3.1.1 Ricerca degli identificativi

Adesso si vanno ad identificare e scegliere gli attributi per identificare univocamente le varie entità:

- In **Profili** è già presente un attributo, cioè *Nome_Utente* che rispecchia una chiave primaria.
- In **Gruppi** non è presente un attributo candidabile come chiave primaria, quindi aggiungiamo l'attributo *Id_Gruppo*.
- In **Tags** è già presente un attributo, cioè *Parola* che rispecchia una chiave primaria essendo l'unico attributo dell'entità.
- In **Contenuti** non è presente un attributo candidabile come chiave primaria, quindi aggiungiamo l'attributo *Id_Contenuto*.
- In **Likes** non è possibile indicare una chiave primaria, quest'ultima comparirà con lo schema logico.
- In **Commenti** non è presente un attributo candidabile come chiave primaria, quindi aggiungiamo l'attributo *Id_Commenti*.
- In **Notifiche_Contenuti** non è presente un attributo candidabile come chiave primaria, quindi aggiungiamo l'attributo *Id_Notifica_C*.
- In **Notifiche_Gruppi** non è presente un attributo candidabile come chiave primaria, quindi aggiungiamo l'attributo *Id_Notifica_G*.
- In **Notifiche_Richieste** non è presente un attributo candidabile come chiave primaria, quindi aggiungiamo l'attributo *Id_Notifica_RC*.

3.1.2 Eliminazione delle gerarchie

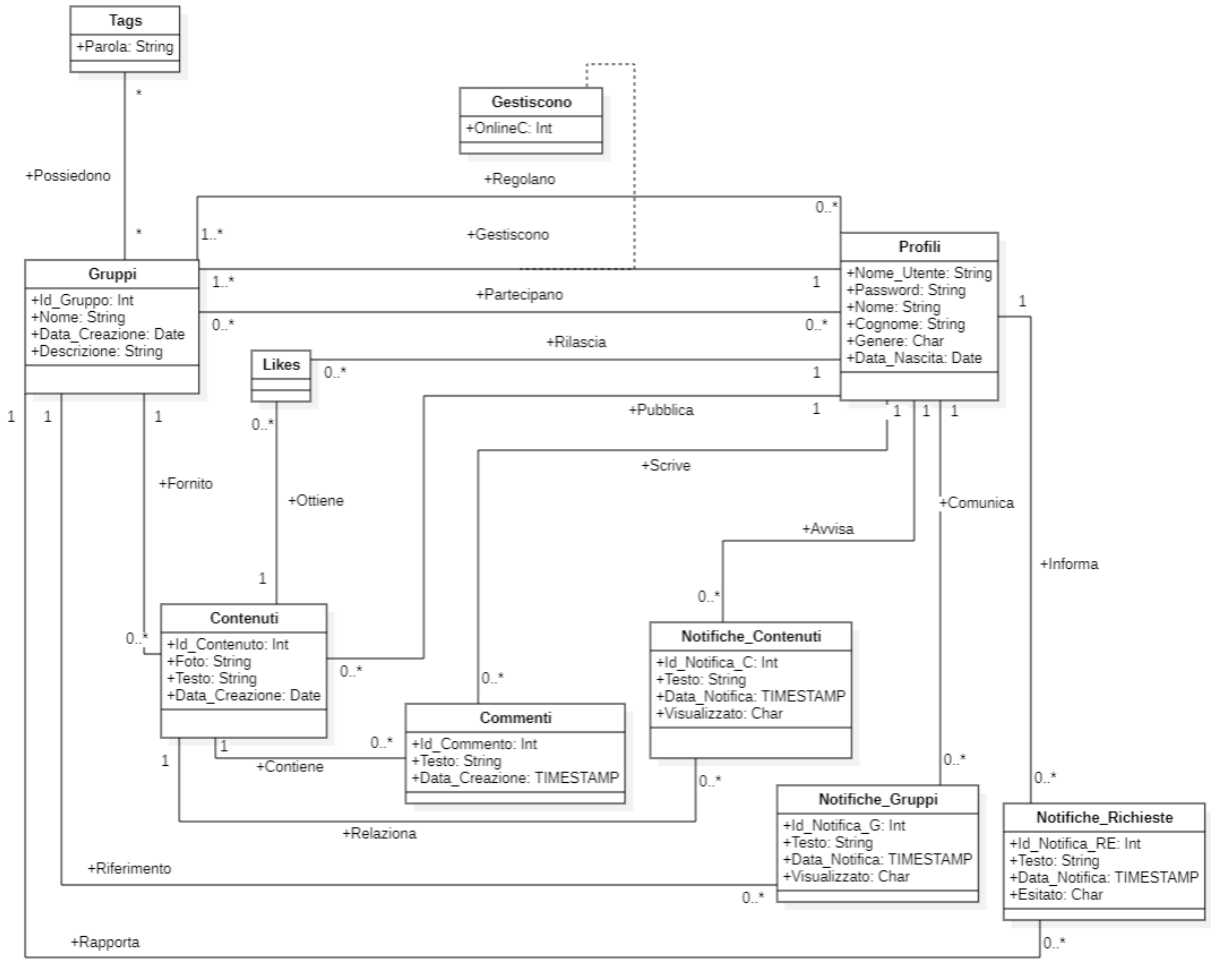
In questo diagramma è presente una generalizzazione con 3 specializzazioni.
La generalizzazione è **Totale/Overlapping**:

- Totale perchè un profilo deve essere necessariamente posseduto da un utente, o da un amministratore, o un creatore.
- Overlapping perchè un utente può essere sia un amministarore che un creatore.

Per risolvere la generalizzazione **Profili**, si è scelto di includere le entità figlie nell'entità padre, quindi infine avremo un'unica entità Profili, a quest'ultima non gli apportiamo nessuna modifica siccome le entità figlie non hanno attributi e grazie alle relazioni possiamo specificare il ruolo di un profilo nel gruppo.

3.2 Class Diagram ristrutturato

3.2.1 UML Diagram



3.2.2 Specifiche del diagramma UML

- Apparentemente le enitità **Notifiche_Gruppi** e **Notifiche_Contenuti** sono uguali, ma per avere una gestione migliore delle notifiche abbiamo deciso di avere due enità separate

3.3 Dizionario delle classi

Leggenda

- Gli attributi **in grassetto** corrispondono alle chiavi primarie

Classi	Descrizione	Attributi
Profilo	Un utente generico del Social Network.	**Nome_Utente** : Nome identificativo dell'utente. Password: La password dell'utente Nome: Nome giuridico (o legale) dell'utente Cognome: Cognome giuridico (o legale) dell'utente Genere: Esprime l'appartenenza ad un sesso. Data_Di_Nascita: Data in cui è nato l'utente.
Gruppi	Gruppi tematici a cui gli utenti possono parteciare.	**Id_Gruppo**: Numero identificativo del gruppo. Nome: Nome assegnato al gruppo. Data_Creazione: Data di Creazione del gruppo. Descrizione: Breve presentazione del gruppo.
Contenuti	Contenuti che vengono pubblicati dagli utenti.	**Id_Contenuto**: Numero identificativo del contenuto. Foto: Immagine allegata al contenuto. Testo: Testo del contenuto. Data_Creazione: Data di Creazione del contenuto.
Commenti	Commenti che vengono pubblicati dagli utenti relativi ai vari contenuti.	**Id_Commento**: Numero identificativo del commento. Testo: Commento espresso dall'utente. Data_Creazione: Data di Creazione del commento.
Tags	Parola chiave del tema del gruppo.	Parola: Termine associato al tema del gruppo.
Likes	Numero di 'mi piace' del contenuto.	Nessun attributo.
Notifiche_Contenuti	Notifiche relative ai contenuti.	**Id_Notifica_C**: Numero identificativo della notifica. Testo: Testo della notifica. Data_Notifica: Data di Creazione della notifica. Visualizzato: Controllo sulla eventuale visualizzazione della notifica.
Notifiche_Gruppi	Notifiche relative ai gruppi.	**Id_Notifica_G**: Numero identificativo della notifica. Testo: Testo della notifica. Data_Notifica: Data di Creazione della notifica. Visualizzato: Controllo sulla eventuale visualizzazione della notifica.
Notifiche_richieste	Notifiche relative alle richieste per l'accesso ai gruppi.	**Id_Notifica_G**: Numero identificativo della notifica.

Classi	Descrizione	Attributi
		Testo: Testo della notifica. Data_Notifica: Data di Creazione della notifica. Esitato: Esito del creatore per l'accesso al gruppo.

3.4 Dizionario delle relazioni

Leggenda

- Le Classi **in grassetto**.
- Gli attributi Sottolineati.

Relazione	**Descrizione**
Possiedono	Relazione molti a molti tra "**Gruppi**" e "**Tags**". Un Gruppo può avere più tag e un tag può essere associato a più gruppi.
Partecipano	Relazione uno a molti tra "**Gruppi**"e "**Profili**". Un gruppo può avere più utenti e un Utente può partecipare a più gruppi.
Gestiscono	Relazione uno a molti tra "**Gruppi**" e "**Profili**." Un creatore può avere più gruppi e un gruppo può avere un solo creatore. OnlineC: Attributo per verificare quando il creatore è online.
Regolano	Relazione molti a molti tra "**Gruppi**" e "**Profili**". Un Amministratore può avere più gruppi e un gruppo può avere più amministratori.
Rilascia	Relazione uno a molti tra "**Likes**" e "**Profili**". Un profilo può rilasciare più Like e Like può essere rilasciato da un solo. profilo.
Pubblica	Relazione uno a molti tra "**Contenuti**" e "**Profili**". Un profilo può pubblicare più contenuti e un contenuto può essere pubblicato da un solo profilo.
Scrive	Relazione uno a molti tra "**Commenti**" e "**Profili**". Un profilo può scrivere più commenti e un commento può essere pubblicato da un solo profilo.
Avvisa	Relazione molti a uno tra "**Notifiche_contenuti**" e "**Profili**". Un profilo può essere avvisato con più notifiche e una notifica deve avvisare un profilo.

Leggenda

- Le Classi **In Grassetto**.
- Gli attributi Sottolineati.

Relazione	**Descrizione**
Comunica	Relazione molti a uno tra "**Notifiche_gruppi**" e "**Profili**". Un profilo può avere più notifiche da un gruppo e una notifica deve appartenere ad un solo profilo.
Informa	Relazione molti a uno tra "**Notifiche_richieste_esiti**" e "**Profili**". Un profilo può avere più notifiche di esito e una notifica può informare un solo profilo.
Relaziona	Relazione molti a uno tra "**Notifiche_Contenuti**" e "**Contenuti**". Un contenuto può relazionarsi con più notifiche e una notifica può relazionarsi con un solo contenuto.
Riferimento	Relazione molti a uno tra "**Notifiche_Gruppi**" e "**Gruppi**". Un gruppo può fare riferimento a più notifiche e una notifica può fare riferimento a un solo gruppo.
Rapporta	Relazione uno a molti tra "**Notifiche_richieste_esiti**" e "**Gruppi**". Un Gruppo può avere più notifiche di esito e una notifica può rapportare un solo gruppo.
Fornito	Relazione uno a molti tra "**Contenuti**" e "**Gruppi**". Un Gruppo può fornire più contenuti e un contenuto può fornire un solo gruppo
Ottiene	Relazione uno a molti tra "**Contenuti**" e "**like**". Un like può essere ottenuto da un contenuto e un contenuto può ottenere più likes.

3.5 Dizionario dei vincoli

Nome del vincolo	Tipo del vincolo	Descrizione
Lunghezza_Password	Dominio	La password deve essere lunga minimo 8 caratteri
Verifica_Genere	Dominio	Il genere deve corrispondere alle lettere "M" o "F" o "N"
Verifica_Data_Nascita	Dominio	La data di nascita deve essere precedente alla data attuale del sistema
Imposta_Data_Creazione_G	Dominio	La data di creazione di un gruppo deve corrispondere alla data attuale del sistema
Imposta_OnlineC	Dominio	L'utente di base è offline
Imposta_Data_Creazione_C	Dominio	La data di creazione di un contenuto deve corrispondere alla data attuale del sistema
Imposta_Data_Creazione_Commenti	Dominio	La data e ora di creazione di un commento deve corrispondere alla data e ora del sistema a quell'istante di tempo
Imposta_Visualizzato_Notifiche_G	Dominio	Il visualizzato di una notifica del gruppo inizialmente è pari a 0
Imposta_Visualizzato_Notifiche_C	Dominio	Il visualizzato di una notifica del contenuto inizialmente è pari a 0
Imposta_Visualizzato_Notifiche_R	Dominio	Il visualizzato di una notifica di richiesta inizialmente è pari a 0

Nome del vincolo	Tipo del vincolo	Descrizione
Verifica_Elementi_Contenuto	N-UPLA	Un contenuto deve avere almeno il testo o la foto
Verifica_Nome_Gruppo	Intrarelazionale	Il nome di un gruppo è unico
Invia_Richiesta_Partecipazione	Interrelazionale	Se un utente ha già inviato una richiesta di partecipazione ad un gruppo, potrà inviare un'altra richiesta a quel gruppo solo quando viene rifiutato o accettato
Invia_Richiesta_Partecipazione_Iscritto	Interrelazionale	Se un utente partecipa già al gruppo, non potrà inviare un'altra richiesta a quel gruppo
Mostra_Notifiche_Gruppi_Disiscritti	Interrelazionale	Un utente non può ricevere le nuove notifiche di un gruppo se non partecipa a quel gruppo
Mostra_Notifiche_Contenuti_Disiscritti	Interrelazionale	Un utente non può ricevere le nuove notifiche di un contenuto se non partecipa a quel gruppo
Crea_Contenuti_Disiscritti	Interrelazionale	Un utente non può creare un contenuto se non partecipa a quel gruppo
Crea_Commenti_Disiscritti	Interrelazionale	Un utente non può creare un commento se non partecipa a quel gruppo
Inserisci_Like_Disiscritti	Interrelazionale	Un utente non può inserire un like se non partecipa a quel gruppo
Modifica_Gruppo	Interrelazionale	Solo il creatore del gruppo può modificare gli attributi del gruppo
Modifica_Contenuto	Interrelazionale	Solo il creatore del contenuto può modificare modificare il contenuto
Modifica_Commento	Interrelazionale	Solo il creatore del commento può modificare il commento
Elimina_Gruppo	Interrelazionale	Solo il creatore del grupo può eliminare il gruppo

3.6 Progettazione Logica

Nella fase di progettazione logica, analizzeremo i procedimenti che ci permetteranno di passare da uno schema concettuale (ristrutturato) a uno **schema logico**, per ottenere una rappresentazione più vicina al database.

3.6.1 Schema logico

Legenda

- Chiavi primarie **in grassetto**.
- Chiavi esterne sottolineate
- Chiavi primarie esterne **sottolineate e in grassetto**.

Profili (**Nome_Utente**, Password, Nome, Cognome, Genere, Data_Nascita)

Gruppi (**Id_Gruppo**, Nome, Data_Creazione, Descrizione, FK_Nome_Utente)

Tags (**Parola**)

Contenuti (**Id_Contenuto**, Foto, Testo,Data_Creazione, FK_Nome_Utente, FK_Id_Gruppo)

Commenti (**Id_Commento**, Testo, Data_Creazione, FK_Nome_Utente, FK_Id_Contenuto)

Likes (**FK_Nome_Utente, FK_Id_Contenuto**)

Notifiche_Contenuti (**Id_Notifica_C**, Testo, Data_Creazione, Visualizzato, FK_Nome_Utente, FK_Id_Contenuto)

Notifiche_Gruppi (**Id_Notifica_G**, Testo, Data_Creazione, Visualizzato, FK_Nome_Utente, FK_Id_Gruppo)

Notifiche_Richieste (**Id_Notifica_RE**, Testo, Data_Creazione, Esitato, FK_Nome_Utente, FK_Id_Gruppi)

Partecipano (**FK_Nome_Utente, FK_Id_Gruppo**)

Regolano(**FK_Nome_Utente, FK_Id_Gruppo**)

Possiedono(**Parola, FK_Id_Gruppo**)

3.6.2 Traduzione delle relazioni

- **Relazione "Gestiscono"**: Si tratta di una relazione uno-a-molti, l'entità con cardinalità 1 è **Gruppi**, quella con cardinalità molti è **Profili**, date le cardinalità si deve passare in gruppi la chiave "**FK_Nome_Utente**".
- **Relazione "Regolano"**: Si tratta di una relazione molti-a-molti, date le cardinalità si deve creare una nuova entità "**Regolano**" che ha come attributti le chiavi esterne di **Gruppi** e **Profili**, entrambe saranno le chiavi primarie della nuova entità.
- **Relazione "Partecipano"**: Si tratta di una relazione molti-a-molti, date le cardinalità si deve creare una nuova entità "**Partecipano**" che ha come attributti le chiavi esterne di **Gruppi** e **Profili**, entrambe saranno le chiavi primarie della nuova entità.
- **Relazione "Possiedono"**: Si tratta di una relazione molti-a-molti, date le cardinalità si deve creare una nuova entità "**Possiedono**" che ha come attributti le chiavi esterne di **Gruppi** e **Tags**, entrambe saranno le chiavi primarie della nuova entità.
- **Relazione "Fornito"**: Si tratta di una relazione uno-a-molti, l'entità con cardinalità 1 è **Contenuti**, quella con cardinalità molti è **Gruppi**, date le cardinalità si deve passare in contenuti la chiave "**FK_Id_Gruppi**".
- **Relazione "Riferimento"**: Si tratta di una relazione uno-a-molti, l'entità con cardinalità 1 è **Notifiche_Gruppi**, quella con cardinalità molti è **Gruppi**, date le cardinalità si deve passare in Notifiche_Gruppi la chiave "**FK_Id_Gruppi**".
- **Relazione "Rapporta"**: Si tratta di una relazione uno-a-molti, l'entità con cardinalità 1 è **Notifiche_Richieste**, quella con cardinalità molti è **Gruppi**, date le cardinalità si deve passare in Notifiche_Richieste la chiave "**FK_Id_Gruppi**".
- **Relazione "Contiene"**: Si tratta di una relazione uno-a-molti, l'entità con cardinalità 1 è **Commenti**, quella con cardinalità molti è **Contenuti**, date le cardinalità si deve passare in Commenti la chiave "**FK_Id_Contenuti**".
- **Relazione "Pubblica"**: Si tratta di una relazione uno-a-molti, l'entità con cardinalità 1 è **Contenuti**, quella con cardinalità molti è **Profili**, date le cardinalità si deve passare in contenuti la chiave "**FK_Nome_Utente**".
- **Relazione "Ottiene"**: Si tratta di una relazione uno-a-molti, l'entità con cardinalità 1 è **Likes**, quella con cardinalità molti è **Contenuti**, date le cardinalità si deve passare in Likes la chiave "**FK_Id_Contenuti**".
- **Relazione "Rilascia"**: Si tratta di una relazione uno-a-molti, l'entità con cardinalità 1 è **Likes**, quella con cardinalità molti è **Profili**, date le cardinalità si deve passare in Likes la chiave "**FK_Nome_Utente**".
- **Relazione "Avvisa"**: Si tratta di una relazione uno-a-molti, l'entità con cardinalità 1 è **Notifica_Contenuti**, quella con cardinalità molti è **Profili**, date le cardinalità si deve passare in Notifica_Contenuti la chiave "**FK_Nome_Utente**".
- **Relazione "Relaziona"**: Si tratta di una relazione uno-a-molti, l'entità con cardinalità 1 è **Notifica_Contenuti**, quella con cardinalità molti è **Contenuti**, date le cardinalità si deve passare in Notifica_Contenuti la chiave "**FK_Id_Contenuti**".
- **Relazione "Comunica"**: Si tratta di una relazione uno-a-molti, l'entità con cardinalità 1 è **Notifica_Gruppi**, quella con cardinalità molti è **Profili**, date le cardinalità si deve passare in Notifica_Gruppi la chiave "**FK_Nome_Utente**".
- **Relazione "Informa"**: Si tratta di una relazione uno-a-molti, l'entità con cardinalità 1 è **Notifica_Richieste**, quella con cardinalità molti è **Profili**, date le cardinalità si deve passare in Notifica_Richieste la chiave "**FK_Nome_Utente**".
- **Relazione "Scrive"**: Si tratta di una relazione uno-a-molti, l'entità con cardinalità 1 è **Commenti**, quella con cardinalità molti è **Profili**, date le cardinalità si deve passare in commenti la chiave "**FK_Nome_Utente**"

4 Progettazione Fisica

In questa ultima fase analizzeremo i meccanismi di traduzione da uno schema logico ad uno **schema fisico**. A questo punto possiamo implementare **tabelle**, **funzioni**, **procedure**, **trigger** e i **vincoli** all'interno del database.
Considereremo come database di riferimento "**ORACLE_XE**" nella versione "**21c**".

4.1 Definizione delle tabelle e viste

Si procede implementando tutte le tabelle e le eventuali viste all'interno del database.
Con la creazione delle varie tabelle si adranno ad implementare i vincoli di **Dominio**, di **N-Upla** e **Intrarelazionali**.

4.1.1 Definizione della tabella Profili

```
--CREAZIONE TABELLA PROFILI
 CREATE TABLE Profili(
 Nome_Utente VARCHAR2(30),
 Password VARCHAR2(30) NOT NULL CHECK(length(Password)>=8),
 Nome VARCHAR2(30) NOT NULL,
 Cognome VARCHAR2(30) NOT NULL,
 Genere CHAR(1) NOT NULL CHECK(Genere='M' OR Genere='F' OR Genere='N'),
 Data_Nascita Date NOT NULL,
 Primary key (Nome_Utente)
 );
```

4.1.2 Definizione della tabella Gruppi

```
--CREAZIONE TABELLA GRUPPI
 CREATE TABLE Gruppi(
 Id_Gruppo NUMBER GENERATED ALWAYS AS IDENTITY (START WITH 1 INCREMENT BY 1),
 Nome VARCHAR2(30) NOT NULL UNIQUE,
 Data_Creazione Date DEFAULT SYSDATE,
 Descrizione VARCHAR2(100) NOT NULL,
 OnlineC NUMBER(1) DEFAULT 0, --0 offline, 1 online
 FK_Nome_Utente VARCHAR2(30) NOT NULL,
 Primary key (Id_Gruppo),
 FOREIGN KEY (FK_Nome_Utente) REFERENCES Profili(Nome_Utente) ON DELETE CASCADE
);
```

4.1.3 Definizione della tabella Tags

```
--CREAZIONE TABELLA TAGS
 CREATE TABLE Tags(
 Parola VARCHAR2(20),
 Primary key (Parola)
 );
```

4.1.4 Definizione della tabella Contenuti

```
--CREAZIONE TABELLA CONTENUTI
 CREATE TABLE Contenuti(
 Id_Contenuto NUMBER GENERATED ALWAYS AS IDENTITY (START WITH 1 INCREMENT BY 1),
 Data_Creazione Date DEFAULT SYSDATE,
 Foto VARCHAR2(2000),
 Testo VARCHAR2(1000),
 FK_Id_Gruppo NUMBER NOT NULL,
 FK_Nome_Utente VARCHAR2(30) NOT NULL,
```

```
Primary key (Id_Contenuto),
FOREIGN KEY (FK_Nome_Utente) REFERENCES Profili(Nome_Utente) ON DELETE CASCADE,
FOREIGN KEY (FK_Id_Gruppo) REFERENCES Gruppi(Id_Gruppo) ON DELETE CASCADE,
CONSTRAINT Check_Contenuto Check(Foto<>NULL OR Testo<>NULL)
);
```

4.1.5 Definizione della tabella Commenti

```
--CREAZIONE TABELLA COMMENTI
CREATE TABLE Commenti(
Id_Commento NUMBER GENERATED ALWAYS AS IDENTITY (START WITH 1 INCREMENT BY 1),
Data_Creazione TIMESTAMP DEFAULT SYSTIMESTAMP,
Testo VARCHAR2(1000) NOT NULL,
FK_Id_Contenuto NUMBER NOT NULL,
FK_Nome_Utente VARCHAR2(30) NOT NULL,
Primary key (Id_Commento),
FOREIGN KEY (FK_Nome_Utente) REFERENCES Profili(Nome_Utente) ON DELETE CASCADE,
FOREIGN KEY (FK_Id_Contenuto) REFERENCES Contenuti(Id_Contenuto) ON DELETE CASCADE
);
```

4.1.6 Definizione della tabella Notifiche_Gruppi

```
--CREAZIONE TABELLA NOTIFICHE GRUPPI
CREATE TABLE Notifiche_Gruppi(
Id_Notifica_G NUMBER GENERATED ALWAYS AS IDENTITY (START WITH 1 INCREMENT BY 1),
Testo VARCHAR2(1000) NOT NULL,
Data_Notifica TIMESTAMP DEFAULT SYSTIMESTAMP,
Visualizzato CHAR(1) DEFAULT '0' CHECK(Visualizzato='0' OR Visualizzato='1'), --0
    non visualizzato, 1 visualizzato
FK_Id_Gruppo NUMBER NOT NULL,
FK_Nome_Utente VARCHAR2(30) NOT NULL,
Primary key (Id_Notifica_G),
FOREIGN KEY (FK_Nome_Utente) REFERENCES Profili(Nome_Utente) ON DELETE CASCADE,
FOREIGN KEY (FK_Id_Gruppo) REFERENCES Gruppi(Id_Gruppo) ON DELETE CASCADE
);
```

4.1.7 Definizione della tabella Notifiche_Contenuti

```
--CREAZIONE TABELLA NOTIFICHE CONTENUTI
CREATE TABLE Notifiche_Contenuti(
Id_Notifica_C NUMBER GENERATED ALWAYS AS IDENTITY (START WITH 1 INCREMENT BY 1),
Testo VARCHAR2(1000) NOT NULL,
Data_Notifica TIMESTAMP DEFAULT SYSTIMESTAMP,
Visualizzato CHAR(1) DEFAULT '0' CHECK(Visualizzato='0' OR Visualizzato='1'), --0
    non visualizzato, 1 visualizzato
FK_Id_Contenuto NUMBER NOT NULL,
FK_Nome_Utente VARCHAR2(30) NOT NULL,
Primary key (Id_Notifica_C),
FOREIGN KEY (FK_Nome_Utente) REFERENCES Profili(Nome_Utente) ON DELETE CASCADE,
FOREIGN KEY (FK_Id_Contenuto) REFERENCES Contenuti(Id_Contenuto) ON DELETE CASCADE
);
```

4.1.8 Definizione della tabella Notifiche_Richieste

```
--CREAZIONE TABELLA NOTIFICHE RICHIESTE
CREATE TABLE Notifiche_Richieste(
Id_Notifica_RE NUMBER GENERATED ALWAYS AS IDENTITY (START WITH 1 INCREMENT BY 1),
Testo VARCHAR2(1000) NOT NULL,
```

```
 Data_Notifica TIMESTAMP DEFAULT SYSTIMESTAMP,
 Esitato CHAR(1) DEFAULT '0' CHECK(Esitato='0' OR Esitato='1' OR Esitato='2'), --0
     non risposto, 1 accettato, 2 rifutato
 FK_Id_Gruppo NUMBER NOT NULL,
 FK_Nome_Utente VARCHAR2(30) NOT NULL,
 Primary key (Id_Notifica_RE),
 FOREIGN KEY (FK_Nome_Utente) REFERENCES Profili(Nome_Utente) ON DELETE CASCADE,
 FOREIGN KEY (FK_Id_Gruppo) REFERENCES Gruppi(Id_Gruppo) ON DELETE CASCADE
);
```

4.1.9 Definizione della tabella Partecipano

```
 --CREAZIONE TABELLA PARTECIPANO
create table Partecipano (
 FK_Nome_Utente VARCHAR2(30),
 FK_Id_Gruppo NUMBER,
 Primary key(FK_Nome_Utente, FK_Id_Gruppo),
 FOREIGN KEY (FK_Nome_Utente) REFERENCES Profili(Nome_Utente) ON DELETE CASCADE,
 FOREIGN KEY (FK_Id_Gruppo) REFERENCES Gruppi(Id_Gruppo) ON DELETE CASCADE
);
```

4.1.10 Definizione della tabella Regolano

```
--CREAZIONE TABELLA REGOLANO (Tabella per gli amministartori)
create table Regolano (
 FK_Nome_Utente VARCHAR2(30),
 FK_Id_Gruppo NUMBER,
 Primary key(FK_Nome_Utente, FK_Id_Gruppo),
 FOREIGN KEY (FK_Nome_Utente) REFERENCES Profili(Nome_Utente) ON DELETE CASCADE,
 FOREIGN KEY (FK_Id_Gruppo) REFERENCES Gruppi(Id_Gruppo) ON DELETE CASCADE
);
```

4.1.11 Definizione della tabella Possiedono

```
--CREAZIONE TABELLA POSSIEDONO
create table Possiedono(
FK_Id_Gruppo NUMBER,
FK_Parola VARCHAR2(20),
Primary key (FK_Id_Gruppo, FK_Parola),
FOREIGN KEY (FK_Parola) REFERENCES TAGS(Parola) ON DELETE CASCADE,
FOREIGN KEY (FK_Id_Gruppo) REFERENCES Gruppi(Id_Gruppo) ON DELETE CASCADE
);
```

4.1.12 Definizione della tabella Likes

```
--CREAZIONE TABELLA LIKES
CREATE TABLE LIKES(
FK_Nome_Utente VARCHAR2(30),
FK_Id_Contenuto NUMBER,
Primary key (FK_Nome_Utente, FK_Id_Contenuto),
FOREIGN KEY (FK_Nome_Utente) REFERENCES Profili(Nome_Utente) ON DELETE CASCADE,
FOREIGN KEY (FK_Id_Contenuto) REFERENCES Contenuti(Id_Contenuto) ON DELETE CASCADE
);
```

4.1.13 Definizione della vista Contenuti_con_Likes

```
--Viste
CREATE View Contenuti_con_Likes AS (
   SELECT Contenuti.*, (SELECT COUNT(*) FROM Likes WHERE likes.fk_Id_Contenuto=
       Contenuti.Id_Contenuto) AS N_LIKE FROM Contenuti
);
```

4.2 Definizione dei trigger

Si procede implementando tutti i trigger.
Con la creazione dei vari trigger si adranno ad implementare i vincoli **Interrelazionali** e di **Dominio**.

4.2.1 Definizione del trigger Verifica_DataNascita

Questo trigger implementa il vincolo Verifica_Data_Nascita

```
--Trigger per verificare la Data di nascita degli utenti
create or replace TRIGGER Verifica_DataNascita
BEFORE INSERT ON Profili
FOR EACH ROW
WHEN (NEW.Data_Nascita>SYSDATE)

BEGIN
   RAISE_APPLICATION_ERROR(-20001, 'La data di nascita deve essere minore o uguale
       alla data attuale'); -- Eccezione che ci permette di avere un messaggio
       personalizzato
END;
/
```

4.2.2 Definizione del trigger Invia_Notifica_OnlineC

```
--Trigger che avvisa gli utenti di un gruppo quando il creatore  online
create or replace TRIGGER Invia_Notifica_OnlineC
AFTER UPDATE ON Gruppi
FOR EACH ROW
WHEN (NEW.OnlineC = 1 AND OLD.OnlineC = 0)

DECLARE

CURSOR Rec_Utente IS (SELECT FK_Nome_Utente FROM partecipano Where FK_Id_Gruppo= :NEW
    .Id_Gruppo);
TMP_Utente Partecipano.FK_Nome_Utente%TYPE;

BEGIN

OPEN Rec_Utente;

LOOP

   FETCH Rec_Utente INTO TMP_Utente;
   EXIT WHEN Rec_Utente%NOTFOUND;

   INSERT INTO Notifiche_Gruppi (Testo, FK_Id_Gruppo, FK_Nome_Utente) VALUES ('Il
       creatore del gruppo '|| :NEW.Nome ||'  Online!', :NEW.Id_Gruppo, TMP_Utente);

END LOOP;

CLOSE Rec_Utente;

END;
```

```
/
```

4.2.3 Definizione del trigger Partecipazione_Creatore

```
--Trigger che aggiunge il creatore quando crea un gruppo
create or replace TRIGGER Partecipazione_Creatore
AFTER INSERT ON GRUPPI
FOR EACH ROW

BEGIN

   CREA_PARTECIPANO(:NEW.Fk_Nome_Utente, :NEW.Id_Gruppo);

END;
/
```

4.2.4 Definizione del trigger InviNotifica_G

```
--Trigger che avvisa gli utenti iscritti ad un gruppo che un utente ha messo un
    contenuto
create or replace TRIGGER Invia_Notifica_G
AFTER INSERT ON Contenuti
FOR EACH ROW

DECLARE

CURSOR Rec_Utente IS (SELECT FK_Nome_Utente FROM partecipano Where fk_Id_Gruppo= :NEW
    .fk_Id_Gruppo AND FK_Nome_Utente<>:NEW.FK_Nome_Utente);
TMP_Utente Partecipano.FK_Nome_Utente%TYPE;

BEGIN

OPEN Rec_Utente;

LOOP

   FETCH Rec_Utente INTO TMP_Utente;
   EXIT WHEN Rec_Utente%NOTFOUND;

   INSERT INTO Notifiche_Gruppi (Testo, fk_Id_Gruppo, fk_Nome_Utente) VALUES ('Utente
        '|| :NEW.FK_Nome_Utente ||' ha inserito un nuovo contenuto!', :NEW.FK_Id_Gruppo
       , TMP_Utente);
END LOOP;

CLOSE Rec_Utente;

END;
/
```

4.2.5 Definizione del trigger Notifica_Like

Questo trigger implementa il vincolo Mostra_Notifiche_Contenuti_Disiscritti

```
--Trigger che avvisa l'utente che un altro utente ha interagito con un like al suo
    post
create or replace TRIGGER Notifica_Like
AFTER INSERT ON Likes
FOR EACH ROW

DECLARE
```

```
TMP_Utente Profili.Nome_Utente%TYPE;
TMP_Testo Contenuti.Testo%TYPE;
Verifica_Notifica NUMBER;


BEGIN
   SELECT FK_Nome_Utente INTO TMP_Utente FROM Contenuti WHERE Id_Contenuto=:NEW.FK_Id_
       Contenuto;

   -- recupera l'id del gruppo e il nome creatore, del contenuto, del commento e
       con questi verifica se partecipa (contando se esiste almeno 1 riga)
   SELECT COUNT(*) INTO Verifica_Notifica FROM Partecipano WHERE FK_ID_GRUPPO IN (
       SELECT FK_ID_GRUPPO FROM CONTENUTI WHERE Id_Contenuto = :NEW.FK_Id_Contenuto)
       AND FK_NOME_UTENTE IN (FK_Nome_Utente);

   --se  presente almeno una riga allora...
   IF(Verifica_Notifica <>0) THEN
      SELECT Testo INTO TMP_Testo FROM Contenuti WHERE Id_Contenuto=:NEW.FK_Id_
          Contenuto;
      IF(TMP_Utente<>:NEW.FK_Nome_Utente) THEN
         IF(TMP_Testo=NULL) THEN
            INSERT INTO notifiche_contenuti (Testo, fk_Id_Contenuto, fk_Nome_Utente)
                VALUES (:NEW.FK_Nome_Utente ||' ha messo mi piace alla tua foto', :NEW
                .FK_Id_Contenuto, TMP_Utente);
         ElSE
            INSERT INTO notifiche_contenuti (Testo, fk_Id_Contenuto, fk_Nome_Utente)
                VALUES (:NEW.FK_Nome_Utente ||' ha messo mi piace al tuo contenuto: '||
                 TMP_Testo, :NEW.FK_Id_Contenuto, TMP_Utente);
         END IF;
      END IF;
   END IF;

END;
/
```

4.2.6 Definizione del trigger Notifica_Commento

Questo trigger implementa il vincolo Mostra_Notifiche_Contenuti_Disiscritti

```
--Trigger che avvisa l'utente che un altro utente ha interagito con un commento al
    suo post
create or replace TRIGGER Notifica_Commento
AFTER INSERT ON Commenti
FOR EACH ROW

DECLARE
TMP_Utente Profili.Nome_Utente%TYPE;
Verifica_Notifica NUMBER;
BEGIN
   SELECT FK_Nome_Utente INTO TMP_Utente FROM Contenuti WHERE Id_Contenuto=:NEW.FK_Id_
       Contenuto;

   -- recupera l'id del gruppo e il nome creatore del contenuto del commento e con
       questi verifica se partecipa (contando se esiste almeno 1 riga)
   SELECT COUNT(*) INTO Verifica_Notifica FROM Partecipano WHERE FK_ID_GRUPPO IN (
       SELECT FK_ID_GRUPPO FROM CONTENUTI WHERE Id_Contenuto = :NEW.FK_Id_Contenuto)
       AND FK_NOME_UTENTE IN (FK_Nome_Utente);

   --se  presente almeno una riga allora...
   IF(Verifica_Notifica <>0) THEN

      IF(TMP_Utente<>:NEW.FK_Nome_Utente) THEN
```

```
        INSERT INTO notifiche_contenuti (Testo, fk_Id_Contenuto, fk_Nome_Utente)
            VALUES (:NEW.FK_Nome_Utente ||' ha commentato il tuo contenuto: '|| :NEW.
            Testo, :NEW.FK_Id_Contenuto, TMP_Utente);
      END IF;
   END IF;
END;
/
```

4.2.7 Definizione del trigger Notifica_Eliminazione

```
--Trigger che avvisa l'utente che  stato eliminato da un gruppo
create or replace TRIGGER Notifica_Eliminazione
AFTER DELETE ON Partecipano
FOR EACH ROW

DECLARE
TMP_Nome_Gruppo Gruppi.Nome%TYPE;

BEGIN

   SELECT Nome INTO TMP_Nome_Gruppo
   FROM Gruppi
   WHERE Id_Gruppo=:OLD.FK_Id_Gruppo;

   INSERT INTO Notifiche_Gruppi (Testo, FK_Id_Gruppo, FK_Nome_Utente) VALUES (:OLD.FK_
       Nome_Utente||' non fa pi parte del gruppo ' ||TMP_Nome_Gruppo, :OLD.FK_Id_Gruppo,
        :OLD.FK_Nome_Utente);

END;
/
```

4.2.8 Definizione del trigger Verifica_Like

```
--Trigger che verifica se un utente ha gi messo like ad un contenuto
CREATE OR REPLACE TRIGGER Verifica_Like
BEFORE INSERT ON Likes
FOR EACH ROW

DECLARE
Check_Like NUMBER;

BEGIN

   SELECT Count(*) INTO Check_Like
   FROM Likes
   WHERE fk_id_contenuto = :NEW.fk_id_contenuto AND fk_nome_utente = :NEW.fk_nome_
       utente;

   IF(Check_Like = 1) THEN
      RAISE_APPLICATION_ERROR(-20001, 'Hai gi messo like a questo contenuto'); --
          Eccezione che ci permette di avere un messaggio personalizzato
   END IF;

END;
/
```

4.2.9 Definizione del trigger Accettazione_Richiesta

```
--INSERISCE GLI UTENTI IN PARTECIPANO DOPO AVER ACCETTATO LA RICHIESTA
create or replace TRIGGER Accettazione_Richiesta
AFTER UPDATE ON Notifiche_richieste
FOR EACH ROW
WHEN (NEW.Esitato = '1' AND OLD.Esitato <>'1')

BEGIN

   CREA_PARTECIPANO(:OLD.Fk_Nome_Utente, :OLD.Fk_Id_Gruppo);

END;
/
```

4.2.10 Definizione del trigger Notifica_Contenuto_Eliminato

```
--TRIGGER CHE MANDA UNA NOTIFICA A TUTTI GLI UTENTI DEL GRUPPO QUANDO UN UTENTE
    ELIMINA UN CONTENUTO
create or replace TRIGGER Notifca_Contenuto_Eliminato
AFTER DELETE ON Contenuti
FOR EACH ROW

BEGIN

   crea_notifica_gruppo(:OLD.FK_Nome_Utente ||' Ha eliminato un contenuto', :OLD.FK_Id_
       Gruppo, :OLD.FK_Nome_Utente);

END;
/
```

4.2.11 Definizione del trigger Invia_Notifica_Esito

```
-- INVIA NOTIFICA ESITO
create or replace TRIGGER Invia_Notifica_Esito
AFTER UPDATE ON Notifiche_richieste
FOR EACH ROW
WHEN (OLD.Esitato<>'1' AND NEW.Esitato='1')

DECLARE

CURSOR Rec_Nome_Utente IS (SELECT FK_Nome_Utente FROM Partecipano WHERE FK_Id_Gruppo =
     :NEW.FK_Id_Gruppo);

TMP_Nome_Utente Gruppi.FK_Nome_Utente%TYPE;

testo_Msg VARCHAR2(1000);

BEGIN

   OPEN Rec_Nome_Utente;

   LOOP

      FETCH Rec_Nome_Utente INTO TMP_Nome_Utente;
      EXIT WHEN Rec_Nome_Utente%NOTFOUND;

      INSERT INTO Notifiche_Gruppi(Testo, Visualizzato, FK_Id_Gruppo, FK_Nome_Utente)
           VALUES (:NEW.FK_Nome_Utente ||'  stato aggiunto nel gruppo', 0 , :NEW.Fk_ID
          _Gruppo, TMP_Nome_Utente);

   END LOOP;
```

```
    ClOSE Rec_Nome_Utente;

    INSERT INTO Notifiche_Gruppi(Testo, Visualizzato, FK_Id_Gruppo, FK_Nome_Utente)
        VALUES ('Sei stato aggiunto nel gruppo', 0 , :NEW.Fk_ID_Gruppo, :NEW.FK_Nome_
        Utente);


END;
/
```

4.3 Definizione delle procedure

Si procede implementando tutte le procedure.
Con la creazione delle varie procedure si adranno ad implementare i vincoli **Interrelazionali** e di **Dominio**.

4.3.1 Definizione della procedure Rimozione_Commento

```
--PROCEDURE PER LA RIMOZIONE COMMENTO
create or replace PROCEDURE Rimozione_Commento(P_Id_Commento IN COMMENTI.Id_Commento%
    TYPE)
AS

BEGIN

   DELETE FROM COMMENTI WHERE ID_COMMENTO = P_Id_Commento;

END Rimozione_Commento;
/
```

4.3.2 Definizione della procedure Rimozione_Like

```
--PROCEDURE PER LA RIMOZIONE LIKE
create or replace PROCEDURE Rimozione_Like(P_Nome_Utente IN LIKES.FK_NOME_UTENTE%TYPE
    , P_ID_CONTENUTO IN LIKES.FK_ID_CONTENUTO%TYPE)
AS

BEGIN

   DELETE FROM LIKES WHERE FK_ID_CONTENUTO = P_ID_CONTENUTO AND FK_NOME_UTENTE = P_
       Nome_Utente;

END Rimozione_Like;
/
```

4.3.3 Definizione della procedure Ricerca_Gruppo

```
-- RICERCA GRUPPO DA UNA STRINGA
create or replace PROCEDURE Ricerca_Gruppo(P_NOME IN VARCHAR2)
AS

CURSOR Rec_Gruppi IS SELECT NOME FROM GRUPPI WHERE NOME LIKE '%' ||P_NOME ||'%';


TMP_Gruppo Gruppi.Nome%TYPE;

BEGIN
   OPEN Rec_Gruppi;

   LOOP
      FETCH Rec_Gruppi INTO TMP_Gruppo;
      EXIT WHEN Rec_Gruppi%NOTFOUND;

      DBMS_OUTPUT.PUT_LINE(TMP_Gruppo);

   END LOOP;
   CLOSE Rec_Gruppi;

END Ricerca_Gruppo;
/
```

4.3.4 Definizione della procedure Rimozione_Contenuto

```
create or replace PROCEDURE Rimozione_Contenuto(P_Id_Contenuto IN CONTENUTI.Id_
    Contenuto%TYPE)
AS

BEGIN

   DELETE FROM CONTENUTI WHERE ID_CONTENUTO = P_Id_Contenuto;

END Rimozione_Contenuto;
/
```

4.3.5 Definizione della procedure Modifica_Profilo

```
--MODIFICA UN SINGOLO ATTRIBUTO DELLA TABELLA PROFILI
create or replace PROCEDURE Modifica_Profilo(Campo IN VARCHAR2, Val_NEW IN VARCHAR2,
     P_Nome_Utente IN Profili.Nome_Utente%TYPE)
AS

Comando VARCHAR(1000);


BEGIN
   Comando:='UPDATE Profili SET '||Campo||' = '''||Val_New||''' WHERE Nome˙Utente = '''||
       P_Nome_Utente||''''; -- Si usano le virgole (") prima e dopo gli —— per ogni
       variabile che ha bisogno degli apici ('') nel comando
   EXECUTE IMMEDIATE Comando;
END Modifica_Profilo;
/
```

4.3.6 Definizione della procedure Mostra_Richiesta

```
--MOSTRA TUTTE LE NOTIFICHE DELLE RICHIESTE DI PARTECIPAZIONE AL CREATORE DEL
    GRUPPO
create or replace PROCEDURE Mostra_Richiesta (P_Nome_Utente IN Profili.Nome_Utente%
    TYPE)
AS

CURSOR Rec_Gruppo_C IS (SELECT Id_Gruppo From Gruppi Where FK_Nome_Utente = P_Nome_
    Utente);

TMP_Id_Gruppo Gruppi.Id_Gruppo%TYPE;
TMP_Testo Notifiche_richieste.Testo%TYPE;

CURSOR Rec_Testo IS (SELECT Testo FROM Notifiche_richieste WHERE TMP_Id_Gruppo =
    Notifiche_richieste.fk_id_gruppo AND Notifiche_richieste.Esitato = '0');

BEGIN

   OPEN Rec_Gruppo_C;

   LOOP

      FETCH Rec_Gruppo_C INTO TMP_Id_Gruppo;
      EXIT WHEN Rec_Gruppo_C%NOTFOUND;


      OPEN Rec_Testo;

      LOOP
         FETCH Rec_Testo INTO TMP_Testo;
         EXIT WHEN Rec_Testo%NOTFOUND;

         DBMS_OUTPUT.PUT_LINE(TMP_Testo);

      END LOOP;
      CLOSE Rec_Testo;



   END LOOP;

   CLOSE Rec_Gruppo_C;

EXCEPTION
WHEN OTHERS THEN
NULL;


END Mostra_Richiesta;
/
```

4.3.7 Definizione della procedure Mostra_Like_Commento

```
--MOSTRA TUTTI I LIKE E I COMMENTI DI UN ID COMMNETO
create or replace PROCEDURE Mostra_Like_Commento (P_Id_Contenuto IN CONTENUTI_CON_
    LIKES.ID_CONTENUTO%TYPE)
AS


CURSOR Rec_Commento IS SELECT Testo, FK_Nome_Utente FROM COMMENTI WHERE FK_ID_
    CONTENUTO = P_Id_Contenuto;



TMP_N_LIKE CONTENUTI_CON_LIKES.N_LIKE%TYPE;
TMP_Testo CONTENUTI_CON_LIKES.Testo%TYPE;
TMP_Nome_Utente CONTENUTI_CON_LIKES.FK_Nome_Utente%TYPE;

BEGIN
   SELECT N_LIKE INTO TMP_N_LIKE FROM CONTENUTI_CON_LIKES WHERE ID_CONTENUTO = P_Id_
       Contenuto;

   DBMS_OUTPUT.PUT_LINE('LIKES : ' ||TMP_N_LIKE);

   OPEN Rec_Commento;
   LOOP
      FETCH Rec_Commento INTO TMP_Testo, TMP_Nome_Utente;
      EXIT WHEN Rec_Commento%NOTFOUND;

      DBMS_OUTPUT.PUT_LINE( TMP_Nome_Utente ||' : '|| TMP_Testo);

   END LOOP;
   CLOSE Rec_Commento;

END Mostra_Like_Commento;
/
```

4.3.8 Definizione della procedure Rifiuta_Profilo

```
-- PROCEDURE PER RIFIUTARE UN PROFILO
create or replace NONEDITIONABLE PROCEDURE Rifiuta_Profilo(P_FK_Nome_Utente IN
    NOTIFICHE_RICHIESTE.FK_Nome_Utente%TYPE, P_FK_Id_Gruppo IN NOTIFICHE_RICHIESTE.FK_
    Id_Gruppo%TYPE)
AS

Comando VARCHAR(1000);
Tmp_Notifica NOTIFICHE_RICHIESTE.id_notifica_re%TYPE;

TMP_Nome_Gruppo GRUPPI.Nome%TYPE;
BEGIN


   SELECT NOME INTO TMP_Nome_Gruppo FROM GRUPPI WHERE ID_GRUPPO = P_FK_ID_GRUPPO;

   -- recupero l'id della notifica dell'utente che ha fatto richiesta al gruppo e
       che deve ancora avere una risposta (esitato = 0)
   SELECT id_notifica_re INTO TMP_Notifica
   FROM NOTIFICHE_RICHIESTE
   WHERE FK_Id_Gruppo=P_FK_Id_Gruppo AND fk_nome_utente=P_FK_Nome_Utente AND Esitato='0
       ';

   Comando := 'UPDATE NOTIFICHE˙RICHIESTE SET Esitato = ''2'', TESTO = ''Rifiutato
       ' ||P_FK_Nome_Utente ||' nel gruppo :' ||TMP_Nome_Gruppo ||''' WHERE id˙notifica˙re
        = '''||TMP_Notifica||'''';
   EXECUTE IMMEDIATE Comando;

END Rifiuta_Profilo;
/
```

4.3.9 Definizione della procedure Crea_Partecipano

```
-- CREA PARTECIPANO
create or replace PROCEDURE Crea_Partecipano (P_Nome_Utente IN PARTECIPANO.FK_NOME_
    UTENTE%TYPE, P_Id_Gruppo IN PARTECIPANO.FK_Id_Gruppo%TYPE)
AS

BEGIN
   INSERT INTO Partecipano VALUES (P_Nome_Utente, P_Id_Gruppo);
END Crea_Partecipano;
/
```

4.3.10 Definizione della procedure Crea_Regolano

```
-- CREA REGOLANO
create or replace PROCEDURE Crea_Regolano (P_Nome_Utente IN Regolano.FK_NOME_UTENTE%
    TYPE, P_Id_Gruppo IN Regolano.FK_Id_Gruppo%TYPE)
AS

BEGIN
   INSERT INTO Regolano VALUES (P_Nome_Utente, P_Id_Gruppo);
END Crea_Regolano;
/
```

4.3.11 Definizione della procedure Crea_Likes

Questa procedure implementa il vincolo Inserisci_Like_Disiscritti

```
-- CREA LIKES
create or replace PROCEDURE Crea_Like (P_FK_Nome_Utente IN Likes.FK_NOME_UTENTE%TYPE,
    P_FK_Id_Contenuto IN Likes.FK_Id_Contenuto%TYPE)
AS

Verifica_Partecipano NUMBER;
Trova_Gruppo Gruppi.Id_Gruppo%TYPE;

BEGIN

   SELECT FK_Id_Gruppo INTO Trova_Gruppo
   FROM Contenuti
   WHERE Id_Contenuto = P_FK_Id_Contenuto;

   --Verifico se l'utente che mette LIKE partecipa effettivamente al gruppo (se si
       conta 1 riga)
   SELECT COUNT(*) INTO Verifica_Partecipano
   FROM Partecipano
   WHERE FK_Id_Gruppo = Trova_Gruppo AND FK_Nome_Utente = P_FK_Nome_Utente;

  IF (Verifica_Partecipano = 1) THEN
    INSERT INTO Likes VALUES (P_FK_Nome_Utente, P_FK_Id_Contenuto);
  ELSE
    DBMS_OUTPUT.PUT_LINE('Non partecipi al gruppo');
  END IF;

END Crea_Like;
/
```

4.3.12 Definizione della procedure Crea_Possiedono

```
-- CREA POSSIEDONO

create or replace PROCEDURE Crea_Possiedono (P_Id_Gruppo IN Possiedono.FK_Id_Gruppo%
    TYPE, P_FK_Parola IN Possiedono.FK_Parola%TYPE )
AS

BEGIN
   INSERT INTO Possiedono VALUES (P_Id_Gruppo, P_FK_Parola);
END Crea_Possiedono;
/
```

4.3.13 Definizione della procedure Rimozione_Commento_Profilo

```
-- RIMOZIONE DI TUTTI I COMMENTI DI UN PROFILO IN UN GRUPPO
create or replace NONEDITIONABLE PROCEDURE Rimozione_Commento_Profilo(P_Nome_Utente
    COMMENTI.FK_Nome_Utente%TYPE, P_ID_GRUPPO GRUPPI.ID_GRUPPO%TYPE)
AS


CURSOR Rec_Id_Con IS SELECT ID_CONTENUTO FROM CONTENUTI WHERE FK_ID_GRUPPO = P_ID_
    GRUPPO;

TMP_Id_Con COMMENTI.FK_ID_CONTENUTO%TYPE;

BEGIN

   OPEN Rec_Id_Con;

   LOOP
      FETCH Rec_Id_Con INTO TMP_Id_Con;
      EXIT WHEN Rec_Id_Con%NOTFOUND;

      DELETE FROM COMMENTI WHERE FK_ID_CONTENUTO = TMP_Id_Con AND FK_NOME_UTENTE = P_
          Nome_Utente;

   END LOOP;
   CLOSE Rec_Id_Con;

END Rimozione_Commento_Profilo;
/
```

4.3.14 Definizione della procedure Rimozione_Like_Profilo

```
-- RIMOZIONE DI TUTTI I LIKE DI UN PROFILO IN UN GRUPPO
create or replace PROCEDURE Rimozione_Like_Profilo(P_Nome_Utente COMMENTI.FK_Nome_
    Utente%TYPE, P_ID_GRUPPO GRUPPI.ID_GRUPPO%TYPE)
AS


CURSOR Rec_Id_Con IS SELECT ID_CONTENUTO FROM CONTENUTI WHERE FK_ID_GRUPPO = P_ID_
    GRUPPO;

TMP_Id_Con LIKES.FK_ID_CONTENUTO%TYPE;

BEGIN

   OPEN Rec_Id_Con;

   LOOP
      FETCH Rec_Id_Con INTO TMP_Id_Con;
      EXIT WHEN Rec_Id_Con%NOTFOUND;

      DELETE FROM LIKES WHERE FK_ID_CONTENUTO = TMP_Id_Con AND FK_NOME_UTENTE = P_Nome_
          Utente;

   END LOOP;
   CLOSE Rec_Id_Con;

END Rimozione_Like_Profilo;
/
```

4.3.15 Definizione della procedure Rimozione_Contenuto_Profilo

```
-- RIMOZIONE DI TUTTI I CONTENUTI DI UN PROFILO IN UN GRUPPO
create or replace PROCEDURE Rimozione_Contenuto_Profilo(P_Nome_Utente CONTENUTI.FK_
    Nome_Utente%TYPE, P_ID_GRUPPO GRUPPI.ID_GRUPPO%TYPE)
AS

BEGIN

   DELETE FROM CONTENUTI WHERE FK_NOME_UTENTE = P_Nome_Utente AND FK_ID_GRUPPO = P_ID_
       GRUPPO;

END Rimozione_Contenuto_Profilo;
/
```

4.3.16 Definizione della procedure Crea_Profilo

```
-- PROCEDURE PER L'AGGIUNTA DI UN UTENTE NELLA TABALLA PROFILI
create or replace PROCEDURE Crea_Profilo (P_Nome_Utente IN Profili.Nome_Utente%TYPE,
    P_Password IN Profili.Password%TYPE, P_Nome IN Profili.Nome%TYPE, P_Cognome IN
    Profili.Cognome%TYPE, P_Genere IN Profili.Genere%TYPE, P_Data_Nascita IN Profili.
    Data_Nascita%TYPE)
AS

BEGIN
   INSERT INTO Profili VALUES (P_Nome_Utente, P_Password, P_Nome, P_Cognome, P_Genere,
       P_Data_Nascita);
END Crea_Profilo;
/
```

4.3.17 Definizione della procedure Crea_Gruppo

```
-- PROCEDURE PER L'AGGIUNTA DI UN GRUPPO NELLA TABALLA GRUPPI
create or replace PROCEDURE Crea_Gruppo (P_Nome IN Gruppi.Nome%TYPE, P_Descrizione IN
    Gruppi.Descrizione%TYPE, P_FK_Nome_Utente IN Gruppi.FK_Nome_Utente%TYPE)
AS

BEGIN
   INSERT INTO Gruppi (Nome, Descrizione, FK_Nome_Utente) VALUES (P_Nome, P_
       Descrizione, P_FK_Nome_Utente);
END Crea_Gruppo;
/
```

4.3.18 Definizione della procedure Crea_Tag

```
-- PROCEDURE PER L'AGGIUNTA DI UN TAG NELLA TABALLA TAGS
create or replace PROCEDURE Crea_Tag (P_Parola IN Tags.Parola%TYPE)
AS

BEGIN
   INSERT INTO Tags (Parola) VALUES (P_Parola);
END Crea_Tag;
/
```

4.3.19 Definizione della procedure Crea_Contenuto

Questa procedure implementa il vincolo Inserisci_Contenuti_Disiscritti

```
-- PROCEDURE PER L'AGGIUNTA DI UN CONTENUTO NELLA TABALLA CONTENUTI
create or replace PROCEDURE Crea_Contenuto (P_Foto IN Contenuti.Foto%TYPE, P_Testo IN
     Contenuti.Testo%TYPE, P_FK_Id_Gruppo IN Contenuti.FK_Id_Gruppo%TYPE, P_FK_Nome_
    Utente IN Contenuti.FK_Nome_Utente%TYPE)
AS

Verifica_Partecipano NUMBER;

BEGIN


   --Verifico se l'utente partecipa effettivamente al gruppo (se si conta 1 riga)
   SELECT COUNT(*) INTO Verifica_Partecipano
   FROM Partecipano
   WHERE FK_Id_Gruppo = P_FK_Id_Gruppo AND FK_Nome_Utente = P_FK_Nome_Utente;

   IF (Verifica_Partecipano = 1) THEN
      INSERT INTO Contenuti (Foto, Testo, FK_Id_Gruppo, FK_Nome_Utente) VALUES (P_
          Foto, P_Testo, P_FK_Id_Gruppo, P_FK_Nome_Utente);
   ELSE
      DBMS_OUTPUT.PUT_LINE('Non partecipi al gruppo');
   END IF;

END Crea_Contenuto;

/
```

4.3.20 Definizione della procedure Crea_Commento

Questa procedure implementa il vincolo Inserisci_Commento_Disiscritti

```
--Procedure per la creazione dei commenti
create or replace PROCEDURE Crea_Commento(P_Testo IN Commenti.Testo%TYPE, P_FK_Id_
    Contenuto IN Commenti.FK_Id_Contenuto%TYPE, P_FK_Nome_Utente IN Commenti.FK_Nome_
    Utente%TYPE)
AS

Verifica_Partecipano NUMBER;
Trova_Gruppo Gruppi.Id_Gruppo%TYPE;

BEGIN

   SELECT FK_Id_Gruppo INTO Trova_Gruppo
   FROM Contenuti
   WHERE Id_Contenuto = P_FK_Id_Contenuto;

   --Verifico se l'utente che mette il commento partecipa effettivamente al gruppo
       (se si conta 1 riga)
   SELECT COUNT(*) INTO Verifica_Partecipano
   FROM Partecipano
   WHERE FK_Id_Gruppo = Trova_Gruppo AND FK_Nome_Utente = P_FK_Nome_Utente;

  IF (Verifica_Partecipano = 1) THEN
   INSERT INTO Commenti (Testo, FK_Id_Contenuto, FK_NOME_UTENTE) VALUES (P_Testo, P_FK
       _Id_Contenuto, P_FK_Nome_Utente);
  ELSE
   DBMS_OUTPUT.PUT_LINE('Non partecipi al gruppo');
  END IF;

END Crea_Commento;
```

```
/
```

4.3.21 Definizione della procedure Crea_Notifica_Gruppo

```
--Procedure per la creazione delle notifiche gruppo
create or replace PROCEDURE Crea_Notifica_Gruppo(P_Testo IN Notifiche_Gruppi.Testo%
    TYPE, P_FK_Id_Gruppo IN Notifiche_Gruppi.FK_Id_Gruppo%TYPE, P_FK_Nome_Utente IN
    Notifiche_Gruppi.FK_Nome_Utente%TYPE)
AS

BEGIN

  INSERT INTO Notifiche_Gruppi(Testo,FK_Id_Gruppo, FK_Nome_Utente) VALUES (P_Testo,P_
      FK_Id_Gruppo, P_FK_Nome_Utente);

END Crea_Notifica_Gruppo;
/
```

4.3.22 Definizione della procedure Crea_Notifica_Contenuto

```
--Procedure per la creazione delle notifiche contenuto
create or replace PROCEDURE Crea_Notifica_Contenuto(P_Testo IN Notifiche_Contenuti.
    Testo%TYPE, P_FK_Id_Contenuto IN Notifiche_Contenuti.FK_Id_Contenuto%TYPE, P_FK_
    Nome_Utente IN Notifiche_Contenuti.FK_Nome_Utente%TYPE)
AS

BEGIN

  INSERT INTO Notifiche_Contenuti (Testo, FK_Id_Contenuto, FK_NOME_UTENTE) VALUES (P_
      Testo, P_FK_Id_Contenuto, P_FK_Nome_Utente);

END Crea_Notifica_Contenuto;
/
```

4.3.23 Definizione della procedure Crea_Richiesta

Questa procedure implementa i vincoli Invia_Richiesta_Partecipazione e Invia_Richiesta_Partecipazione_Iscritto

```
--Procedure per creare la notifica della richiesta
create or replace PROCEDURE Crea_Richiesta(P_FK_Id_Gruppo IN Notifiche_richieste.FK_Id
    _Gruppo%TYPE , P_FK_Nome_Utente IN Notifiche_richieste.FK_Nome_Utente%TYPE)
AS

TMP_Nome Gruppi.Nome%TYPE;
Verifica_Richiesta NUMBER;
Verifica_Richiesta2 NUMBER;
BEGIN
   SELECT Nome INTO TMP_Nome FROM Gruppi WHERE P_FK_Id_Gruppo = Id_Gruppo;


   --Verifico Se ho gi mandato 1 richiesta che deve ancora avere risposta al gruppo
       (se si conta 1 riga)
   SELECT Count(*) INTO Verifica_Richiesta
   FROM Notifiche_richieste
   WHERE
   (Notifiche_richieste.fk_nome_utente = p_fk_nome_utente
   AND Notifiche_richieste.fk_id_gruppo = p_fk_id_gruppo
   AND Notifiche_richieste.Esitato = '0');

   --Verifico Se sto mandando una richiesta ad un gruppo di cui gi faccio parte (se
       si conta 1 riga)
   SELECT Count(*) INTO Verifica_Richiesta2
   FROM Partecipano
   WHERE
   (Partecipano.fk_nome_utente = p_fk_nome_utente
   AND Partecipano.fk_id_gruppo = p_fk_id_gruppo);


   IF (Verifica_Richiesta2 <>0) THEN
      DBMS_OUTPUT.PUT_LINE('partecipi gi al gruppo');
   ELSIF (Verifica_Richiesta <>0) THEN
      DBMS_OUTPUT.PUT_LINE('Hai gi inviato una richiesta al gruppo');
   ELSE
      INSERT INTO Notifiche_richieste(Testo, FK_Id_Gruppo, FK_Nome_Utente) VALUES (P_
          FK_Nome_Utente ||' Ha inviato una richiesta al gruppo: ' ||TMP_Nome, P_FK_Id_
          Gruppo, P_FK_Nome_Utente);
   END IF;


END Crea_Richiesta;
/
```

4.3.24 Definizione della procedure Visualizzato_Notifica_Contenuto

```
-- Procedure per Visualizzare le notifiche dei contenuti di un utente
create or replace PROCEDURE Visualizzato_Notifica_Contenuto(P_Id_Notifica IN
    Notifiche_Contenuti.Id_Notifica_C%TYPE)
AS

BEGIN
   UPDATE Notifiche_Contenuti SET Visualizzato = '1' WHERE Id_Notifica_C = P_Id_
       Notifica;
END Visualizzato_Notifica_Contenuto;
/
```

4.3.25 Definizione della procedure Visualizzato_Notifica_Gruppi

```
-- Procedure per Visualizzare le notifiche dei gruppi di un utente
create or replace PROCEDURE Visualizzato_Notifica_Gruppo(P_Id_Notifica IN Notifiche_
    Gruppi.Id_Notifica_G%TYPE)
AS

BEGIN
   UPDATE Notifiche_Gruppi SET Visualizzato = '1' WHERE Id_Notifica_G = P_Id_Notifica
       ;
END Visualizzato_Notifica_Gruppo;
/
```

4.3.26 Definizione della procedure Mostra_Notifica

```
-- Procedure per Mostrare le notifiche dei gruppi e dei contenuti di un utente
create or replace NONEDITIONABLE PROCEDURE Mostra_Notifica(P_Nome_Utente IN Notifiche
    _Contenuti.FK_Nome_Utente%TYPE)
AS

-- Creo un cursore ad una tabella che ah SIA Tutte le notifiche dei contenuti di un
     utente SIA tutte le notifiche dei gruppi di un utente e ordino per la data
CURSOR Rec_Notifica IS SELECT NULL AS Id_Notifica_G, Id_Notifica_C ,Visualizzato,
    Testo, Data_Notifica FROM Notifiche_Contenuti WHERE fk_nome_utente LIKE P_Nome_
    Utente
                    UNION ALL
                    SELECT Id_Notifica_G, NULL AS Id_Notifica_C,Visualizzato, Testo,
                        Data_Notifica FROM Notifiche_Gruppi WHERE FK_NOME_UTENTE LIKE P
                        _Nome_Utente
                    ORDER BY Data_Notifica DESC;


TMP_Testo Notifiche_Contenuti.Testo%TYPE;
TMP_Data Notifiche_Contenuti.Data_Notifica%TYPE;
TMP_Visualizzato Notifiche_Contenuti.Visualizzato%TYPE;
TMP_Id_Notifica_C Notifiche_Contenuti.Id_Notifica_C%TYPE;
TMP_Id_Notifica_G Notifiche_Gruppi.Id_Notifica_G%TYPE;


BEGIN
   OPEN Rec_Notifica;

   LOOP
      FETCH Rec_Notifica INTO TMP_Id_Notifica_G,TMP_Id_Notifica_C,TMP_Visualizzato,TMP_
          Testo, TMP_Data;
      EXIT WHEN Rec_Notifica%NOTFOUND;

      DBMS_OUTPUT.PUT_LINE(TMP_Visualizzato ||' - ' ||TMP_Testo ||' - ' ||TMP_Data);

      IF TMP_Id_Notifica_C IS NOT NULL AND TMP_Id_Notifica_C <>0 THEN
         Visualizzato_Notifica_Contenuto(TMP_Id_Notifica_C);
      ELSIF TMP_Id_Notifica_G IS NOT NULL AND TMP_Id_Notifica_G <>0 THEN
         Visualizzato_Notifica_Gruppo(TMP_Id_Notifica_G);
      END IF;


   END LOOP;
   CLOSE Rec_Notifica;

  COMMIT;

END Mostra_Notifica;
```

```
/
```

4.3.27 Definizione della procedure Modifica_Gruppo

Questa procedure implementa il vincolo Modifcia_Gruppo

```
--MODIFICA IL GRUPPO SOLO SE SEI IL CREATORE
create or replace PROCEDURE Modifica_Gruppo(Campo IN VARCHAR2, Val_NEW IN VARCHAR2,
    P_FK_Nome_Utente IN Gruppi.FK_Nome_Utente%TYPE, P_Id_Gruppo IN Gruppi.Id_Gruppo%
    TYPE)
AS

Comando VARCHAR(1000);
TMP_Creatore Gruppi.FK_Nome_Utente%TYPE;

BEGIN

   SELECT FK_Nome_Utente INTO TMP_Creatore
   FROM Gruppi
   WHERE Id_Gruppo = P_Id_Gruppo;


   IF (TMP_Creatore LIKE P_FK_Nome_Utente) THEN
      Comando:='UPDATE Gruppi SET '||Campo||' = '''||Val_New||''' WHERE Id˙Gruppo = '''||
          P_Id_Gruppo||''''; -- Si usano le virgole (”) prima e dopo gli —— per ogni
          variabile che ha bisogno degli apici ('') nel comando
      EXECUTE IMMEDIATE Comando;
   ELSE
      DBMS_OUTPUT.PUT_LINE('Non sei il creatore del gruppo gruppo');
   END IF;

END Modifica_Gruppo;
/
```

4.3.28 Definizione della procedure Modifica_Contenuto

Questa procedure implementa il vincolo Modifcia_Contenuto

```
--MODIFICA IL CONTENUTO SOLO SE SEI IL CREATORE
create or replace PROCEDURE Modifica_Contenuto(Campo IN VARCHAR2, Val_NEW IN
    VARCHAR2, P_FK_Nome_Utente IN Contenuti.FK_Nome_Utente%TYPE, P_Id_Contenuto IN
    Contenuti.Id_Contenuto%TYPE)
AS

Comando VARCHAR(1000);
TMP_Creatore Contenuti.FK_Nome_Utente%TYPE;

BEGIN

   SELECT FK_Nome_Utente INTO TMP_Creatore
   FROM Contenuti
   WHERE Id_Contenuto = P_Id_Contenuto;


   IF (TMP_Creatore LIKE P_FK_Nome_Utente) THEN
      Comando:='UPDATE Contenuti SET '||Campo||' = '''||Val_New||''' WHERE Id˙Contenuto
          = '''||P_Id_Contenuto||''''; -- Si usano le virgole (") prima e dopo gli ––
          per ogni variabile che ha bisogno degli apici ('') nel comando
      EXECUTE IMMEDIATE Comando;
   ELSE
      DBMS_OUTPUT.PUT_LINE('Non sei il creatore del contenuto');
   END IF;

END Modifica_Contenuto;
/
```

4.3.29 Definizione della procedure Modifica_Commento

Questa procedure implementa il vincolo Modifcia_Commento

```
--MODIFICA IL COMMENTO SOLO SE SEI IL CREATORE
create or replace PROCEDURE Modifica_Commento(Val_NEW IN VARCHAR2, P_FK_Nome_Utente
    IN Commenti.FK_Nome_Utente%TYPE, P_Id_Commento IN Commenti.Id_Commento%TYPE)
AS

Comando VARCHAR(1000);
TMP_Creatore Contenuti.FK_Nome_Utente%TYPE;

BEGIN

   SELECT FK_Nome_Utente INTO TMP_Creatore
   FROM Commenti
   WHERE Id_Commento = P_Id_Commento;


   IF (TMP_Creatore LIKE P_FK_Nome_Utente) THEN
      Comando:='UPDATE Commenti SET Testo = '''||Val_New||''' WHERE Id˙Commento = '''||
          P_Id_Commento||''''; -- Si usano le virgole (") prima e dopo gli –– per ogni
           variabile che ha bisogno degli apici ('') nel comando
      EXECUTE IMMEDIATE Comando;
   ELSE
      DBMS_OUTPUT.PUT_LINE('Non sei il creatore del commento');
   END IF;

END Modifica_Commento;
/
```

4.3.30 Definizione della procedure Accetta_Profilo

```
--ACCETTA LA RICHIESTA DI PARTECIPAZIONE AD UN GRUPPO DA PARTE DELL'UTENTE
create or replace PROCEDURE Accetta_Profilo(P_FK_Nome_Utente IN NOTIFICHE_RICHIESTE.
    FK_Nome_Utente%TYPE, P_FK_Id_Gruppo IN NOTIFICHE_RICHIESTE.FK_Id_Gruppo%TYPE)
AS

Comando VARCHAR(1000);
Tmp_Notifica NOTIFICHE_RICHIESTE.id_notifica_re%TYPE;

TMP_Nome_Gruppo GRUPPI.Nome%TYPE;
BEGIN


   SELECT NOME INTO TMP_Nome_Gruppo FROM GRUPPI WHERE ID_GRUPPO = P_FK_ID_GRUPPO;

   --Recupero l'id della notifica dell utente X che non ha ancora avuto risposta (
       esitato = 0)
   SELECT id_notifica_re INTO TMP_Notifica
   FROM NOTIFICHE_RICHIESTE
   WHERE FK_Id_Gruppo=P_FK_Id_Gruppo AND fk_nome_utente=P_FK_Nome_Utente AND Esitato='0
       ';

   Comando := 'UPDATE NOTIFICHE˙RICHIESTE SET Esitato = ''1'', TESTO = ''Accettato
       ' ||P_FK_Nome_Utente ||' nel gruppo :' ||TMP_Nome_Gruppo ||''' WHERE id˙notifica˙re
        = '''||TMP_Notifica||'''';
   EXECUTE IMMEDIATE Comando;

END Accetta_Profilo;

/
```

4.3.31 Definizione della procedure Abbandona_Gruppo

```
--L'UTENTE ABBANDONA O VIENE RIMOSSO DAL GRUPPO
create or replace PROCEDURE Abbandona_Gruppo(P_FK_Nome_Utente IN Partecipano.FK_Nome_
    Utente%TYPE, P_FK_Id_Gruppo IN Partecipano.FK_Id_Gruppo%TYPE)
AS

Comando VARCHAR(1000);
TMP_Creatore Partecipano.FK_Nome_Utente%TYPE;

BEGIN

   SELECT FK_Nome_Utente INTO TMP_Creatore
   FROM Gruppi
   WHERE Id_Gruppo = P_FK_Id_Gruppo;

      IF(TMP_Creatore LIKE P_FK_Nome_Utente) THEN
         Comando:='DELETE FROM Gruppi WHERE Id'Gruppo = '''||P_FK_Id_Gruppo||'''AND
             fk'nome'utente ='''|| P_FK_Nome_Utente||''''; -- Si usano le virgole ('')
             prima e dopo gli -- per ogni variabile che ha bisogno degli apici ('')
             nel comando
         EXECUTE IMMEDIATE Comando;
      ELSE
         Comando:='DELETE FROM Partecipano WHERE FK'Id'Gruppo = '''||P_FK_Id_Gruppo||''
             'AND fk'nome'utente ='''|| P_FK_Nome_Utente||''''; -- Si usano le virgole
             ('') prima e dopo gli -- per ogni variabile che ha bisogno degli apici
             ('') nel comando
         EXECUTE IMMEDIATE Comando;
      END IF;

END Abbandona_Gruppo;
/
```

4.3.32 Definizione della procedure Rimozione_Gruppo

Questa procedure implementa il vincolo Elimina_Gruppo

```
--IL CREATORE ELIMINA IL GRUPPO
create or replace PROCEDURE Rimozione_Gruppo(P_Id_Gruppo IN Gruppi.Id_Gruppo%TYPE, P_
    FK_Nome_Utente IN Gruppi.FK_Nome_Utente%TYPE)
AS

TMP_Creatore Gruppi.FK_Nome_Utente%TYPE;

BEGIN
   -- trovo il creatore del gruppo
   SELECT FK_Nome_Utente INTO TMP_Creatore
   FROM Gruppi
   WHERE Id_Gruppo = P_Id_Gruppo;


   IF (TMP_Creatore LIKE P_FK_Nome_Utente) THEN
       DELETE FROM Gruppi WHERE ID_Gruppo = P_Id_Gruppo;
   END IF;

END Rimozione_Gruppo;
/
```

4.3.33 Definizione della procedure Rimozione_Profilo

```
--L'UTENTE ELIMINA IL PROFOILO
create or replace PROCEDURE Rimozione_Profilo(P_Nome_Utente IN Profili.Nome_Utente%
    TYPE)
AS

BEGIN

   DELETE FROM Profili WHERE nome_utente = p_nome_utente;

END Rimozione_Profilo;
/
```

4.4 Definizione delle Funzioni

Si procede implementando tutte le funzioni.
Alcune delle procedure elencate hanno una mera funzione di stampa , utilizzando il comando "DBMS_OUTPUT.PUT_LINE".
Questo non crea un ottimale collegamento con java, per ovviare alla suddetta problematica si dovrebbe utilizzare una function per ognuna di queste procedure.
Questo approccio causa una difficoltà nel chiamare le suddette funzioni dal database, dunque abbiamo ritenuto più opportuno lasciare le procedure.

Un esempio di struttura di soluzione con le funzioni è come segue:

4.4.1 Definizione della Funzione Mostra_Archiviata_F

```
create or replace NONEDITIONABLE FUNCTION Mostra_Archiviata_F(P_Nome_Utente IN
    Profili.Nome_Utente%TYPE)
RETURN SYS_REFCURSOR AS
   rc SYS_REFCURSOR;
BEGIN
   OPEN rc FOR
   SELECT Testo
   FROM (
      (SELECT Testo, FK_Id_gruppo
       FROM NOTIFICHE_RICHIESTE
       WHERE Esitato <>'0'
       AND FK_Nome_Utente <>P_Nome_Utente)

      UNION ALL

      (SELECT Testo, FK_Id_gruppo
       FROM NOTIFICHE_RICHIESTE
       WHERE Esitato <>'0'
       AND FK_Nome_Utente = P_Nome_Utente)
   )
   GROUP BY FK_Id_gruppo, Testo
   ORDER BY FK_Id_gruppo;

   RETURN rc;
END;
```

4.4.2 Definizione del blocco anonimo per utilizzare la funzione Mostra_Archiviata_F

La chiamata alla funzione nel database sarebbe del tipo:

```
DECLARE
   rc SYS_REFCURSOR;
   v_testo NOTIFICHE_RICHIESTE.Testo%TYPE;
BEGIN
   rc := Mostra_archiviata_F('Genny03cry');

   LOOP
      FETCH rc INTO v_testo;
      EXIT WHEN rc%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(v_testo);
   END LOOP;

   CLOSE rc;
END;
/
```

Mentre nel caso di una procedure dobbiamo solamente utilizzare il comando "CALL" come segue:

```
CALL Mostra_Archiviata('Genny03cry');
```