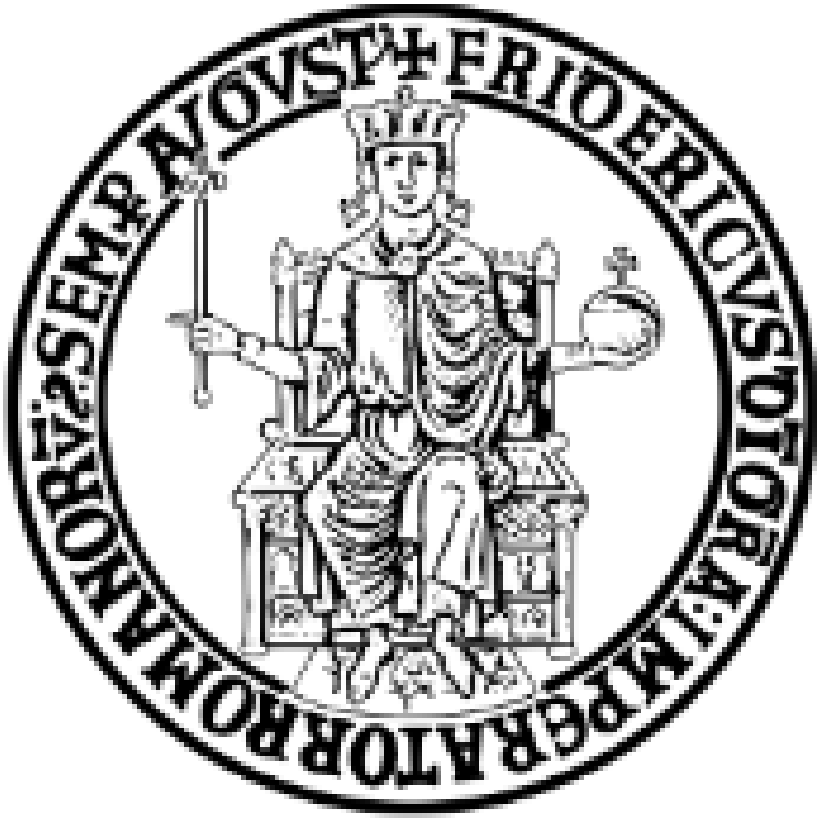


Progettazione e descrizione di una base di dati relazionale per la gestione e memorizzazione di rubriche telefoniche avanzate.

Lembo Gianmarco N86003940
Veniero Simone N86003919
25/07/2022

Indice

Capitolo 1

Introduzione

1.1 Analisi del problema

Si vuole progettare una base di dati che permetta all'utente di memorizzare e gestire una rubrica telefonica avanzata. Per espandere ulteriormente il dominio del problema abbiamo optato per l'inserimento dell'entità utente, in questo modo è permessa l'esistenza di più rubriche corrispondenti a diversi utenti. I contatti saranno infatti associati all'utente che li ha salvati, dando la possibilità di accedere alla propria rubrica tramite delle credenziali di accesso.
L'utente potrà quindi accedere alla propria rubrica oppure registrarsi per poterne creare una nuova, per effettuare l'accesso e la registrazione sarà necessario fornire e-mail e password valide. La rubrica sarà costituita da contatti, in cui verranno mantenute tutte le informazioni principali, e ad esso saranno associati indirizzi di posta elettronica, indirizzi fisici, numeri di telefono e account di messaging. I contatti della rubrica potranno essere organizzati in gruppi. Nella rubrica sarà inoltre salvato il log delle chiamate che potranno essere effettuate, ricevute o perse.

1.2 Requisiti identificati

Una rubrica è un insieme di contatti gestiti da un utente, a cui vi potrà accedere tramite e-mail e password. Inoltre, all'utente sarà data la possibilità di scrivere delle note o eventuali memo. La rubrica deve essere in grado di consentire la memorizzazione delle informazioni riguardanti:

- I contatti, ovvero nome, cognome, una o più e-mail ed eventualmente una foto profilo. Un contatto può essere inserito tra i preferiti e può essere associato ad un indirizzo fisico principale e uno o più secondari, che specifichino via, città, zipcode (CAP) e nazione. Un contatto può avere una sola e-mail principale ed un solo indirizzo fisico principale e tutti i contatti devono essere dotati di almeno un numero di telefono mobile ed un numero di telefono fisso, e per ogni telefono mobile può essere indicato un telefono fisso a cui verranno reindirizzate eventuali chiamate senza risposta e, analogamente per ogni telefono fisso può essere indicato un telefono mobile per il reindirizzamento. Agli stessi indirizzi fisici e numeri di telefono può corrispondere anche più di un contatto.

- La cronologia delle chiamate effettuate, ricevute e perse. Per ogni chiamata saranno memorizzati il numero di telefono, data e ora della chiamata e la durata della stessa (in caso di chiamata persa sarà 0). Inoltre la chiamata avrà un nome del contatto a cui si riferisce, se non esiste nessun contatto a cui può essere associata il campo avrà valore "Unknown".

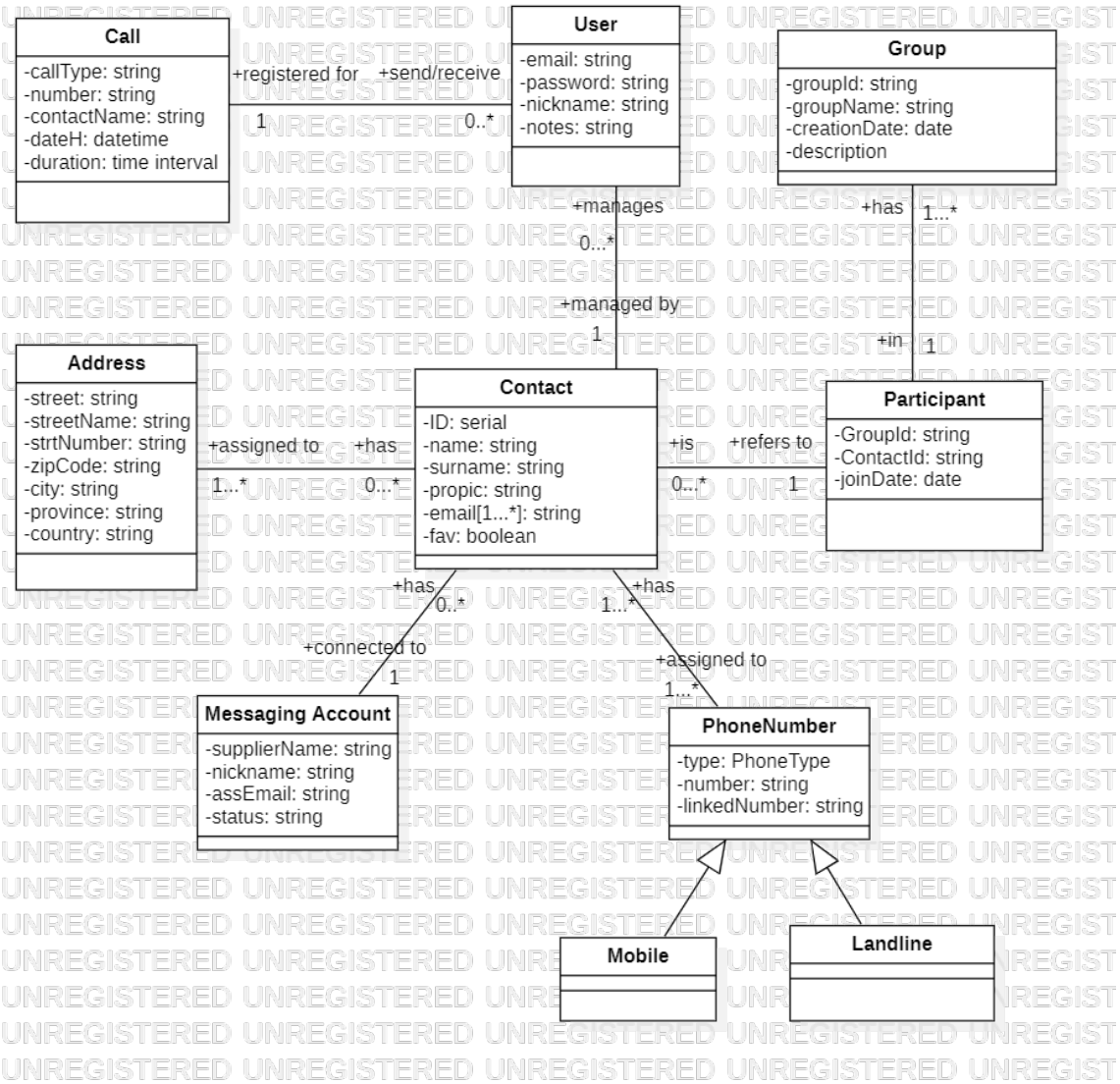
- I gruppi, ovvero un sottoinsieme dei contatti. Un contatto può essere partecipante a uno o più gruppi, e un gruppo ha uno o più partecipanti. I gruppi avranno un nome, una data di creazione e una descrizione.

- Gli account associati a sistemi di messaging. Per ognuno di questi account verrà memorizzato il fornitore, il nickname, la frase di benvenuto (status) e l'indirizzo e-mail collegato che dovrà necessariamente esistere tra gli account di posta già salvati per il contatto.

Capitolo 2

Progettazione concettuale

2.1 Class Diagram



2.2 Analisi della ristrutturazione del Class Diagram

2.2.1 Analisi delle ridondanze

Nell'analisi non sono evidenti particolari ridondanze, tuttavia si è deciso di aggiungere un'associazione tra la relazione User e la relazione Group per snellire le query che richiedessero un accesso a Group necessitando dell'User, operazione che potrebbe essere piuttosto frequente.

2.2.2 Analisi degli identificativi

Analizzando la relazione Address, si nota che la chiave candidata per tale relazione è composta da ben quattro attributi (street, streetName, number, zipCode). Al fine di alleggerire gli accessi sulla relazione, si è deciso quindi di accorpare gli attributi street, streetName e number in un unico attributo di tipo string denominato street. Inoltre, per gli stessi motivi si aggiunge una chiave tecnica all'interno della relazione call.

2.2.3 Rimozione degli attributi multipli

L'unico attributo multiplo presente è email nella relazione Contact, al suo posto è stata creata una relazione aggiuntiva nella quale è stato inserito l'attributo main di tipo boolean. Si è preferita questa soluzione per poter aggiungere un'associazione tra la relazione Messaging e la nuova relazione Email.

2.2.4 Rimozione degli attributi composti

Nella progettazione concettuale non si è reso necessario l'utilizzo di attributi composti.

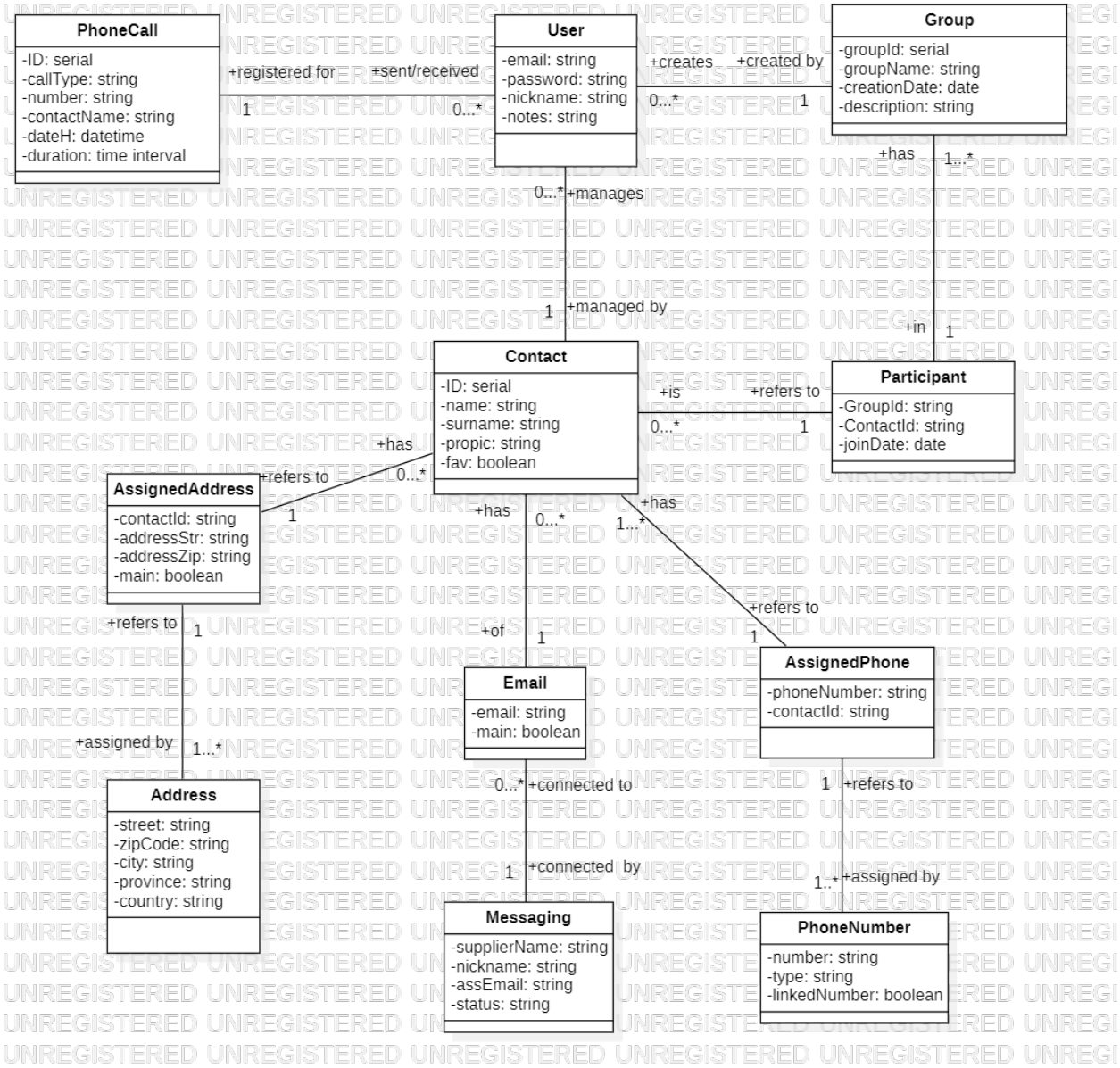
2.2.5 Partizione/Accorpamento delle associazioni

Si è preferito inoltre eliminare l'associazione tra Contact e Social. In questo modo implementando un vincolo di foreign key sull'attributo assEmail della tabella Messaging, si rispetta la condizione secondo la quale per ogni account di messaging, la e-mail collegata dovrà necessariamente esistere tra gli account di posta già salvati per il contatto a cui si riferisce.
Vengono aggiunte per completezza le relazioni necessarie ad esprimere le associazioni di tipo molti a molti, ovvero quella tra Contact e Address, e quella tra Contact e PhoneNumber. Le associazioni prenderanno rispettivamente il nome di AssignedAddress e AssignedPhone. Inoltre, in AssignedAddress viene inserita la variabile main per identificare quale indirizzo associato è principale per quel contatto.

2.2.6 Rimozione delle gerarchie

Le gerarchie da eliminare sono le specializzazioni della relazione PhoneNumber. Si eliminano tali specializzazioni e si inserisce un attributo Type di tipo string all'interno della relazione che può assumere solamente i valori "MOBILE" oppure "LANDLINE".

2.3 Class Diagram ristrutturato



2.4 Dizionario delle classi

Classe	Descrizione	Attributi
User	Utente proprietario della rubrica	**Email** (string): e-mail valida del proprietario della rubrica. **Password** (string): password di accesso alla rubrica. Deve essere di almeno 8 caratteri. **Nickname** (string): nome identificativo del proprietario della rubrica. Deve essere almeno di 3 caratteri. **Notes** (string): note o memo scritte dall'utente.
Contact	Istanza di un contatto salvato nella rubrica	**ContactID** (serial): chiave tecnica. Codice identificativo di un contatto gestito da un utente. **Name** (string): nome del contatto. **Surname** (string): cognome del contatto. **Propic** (string): percorso sul disco del file contenente l'immagine profilo del contatto. **Favorite** (boolean): indica se il contatto è inserito tra i preferiti.
Address	Indirizzo fisico associato ad uno o più contatti	**Street** (string): stringa contenente nome della strada e numero civico dell'indirizzo. **ZipCode** (string): codice postale formato da 5 caratteri (CAP) dell'indirizzo. **City** (string): nome della città all'interno dell'indirizzo. **Province** (string): identificativo di 2 caratteri della provincia relativa alla città. **Country** (string): nazione dell'indirizzo.

Email	Indica uno degli indirizzi di posta elettronica associati al contatto.	**Email** (string): stringa contenente l'indirizzo di posta elettronica. **Main** (boolean): indica se è l'e-mail principale del contatto. Può esistere solo una e-mail principale per contatto.
Messaging	Descrive un eventuale account di messaging collegato ad un contatto	**SupplierName** (string): nome del fornitore dell'app di messaging. **Nickname** (string): nome identificativo dell'account del contatto. **AssEmail** (string): e-mail associata all'account del contatto. Deve esistere tra le e-mail già salvate per il contatto. **Status** (string): frase di benvenuto o stato personale dell'account.
Phone Number	Numero di telefono associato ad uno o più contatti	**PhoneType** (string): tipo di riferimento telefonico. Può essere mobile (MOBILE) oppure fisso (LANDLINE). **PhoneNumber** (string): stringa contenente il numero di telefono. **LinkedNumber** (boolean): indica se è un numero di reindirizzamento. Per ogni contatto deve esistere un numero di reindirizzamento tra i numeri fissi, e uno tra i numeri mobili.
Group	Gruppo a cui possono appartenere uno o più contatti all'interno di una rubrica	**GroupID** (serial): chiave tecnica. Codice identificativo del gruppo creato da un utente. **GroupName** (string): nome del gruppo all'interno della rubrica. **CreationDate** (date): data della creazione del gruppo, formato GG/MM/AAAA. **Description** (string): breve descrizione del gruppo.

Participant	Descrittore dell'associazione tra i contatti ed il gruppo a cui appartengono	**GroupID** (integer): ID del gruppo a cui partecipa il contatto. **ContactID** (integer): ID del contatto partecipante. **JoinDate** (date): data di inizio partecipazione al gruppo, formato GG/MM/AAAA.
PhoneCall	Istanza di una chiamata effettuata, ricevuta o persa	**CallID** (serial): chiave tecnica. Codice identificativo di una chiamata archiviata per un utente. **CallType** (string): tipo della chiamata. Può essere effettuata (SENT), ricevuta (ENTERED) o persa (MISSED). **CallNumber** (string): stringa contenente il numero di telefono relativo alla chiamata. **ContactName** (string): nome e cognome del contatto o dei contatti associati al numero di telefono. Vale sconosciuto (UNKNOWN) se il numero non è presente in rubrica. **DateH** (datetime): data e ora della chiamata. **Duration** (time interval): durata della chiamata. Vale 0 se la chiamata è persa.
Assigned Address	Esprime l'assegnazione di un indirizzo fisico ad un contatto e viceversa	**ContactID** (integer): ID del contatto a cui è assegnato l'indirizzo. **AddressStr** (string): nome della strada e numero civico dell'indirizzo da assegnare al contatto. **AddressZip** (string): codice postale dell'indirizzo da assegnare al contatto. **Main** (boolean): indica se è l'indirizzo principale del contatto. Può esistere solo un indirizzo principale per contatto.

Assigned Phone	Esprime l'assegnazione di un numero di telefono ad un contatto e viceversa	**PhoneNumber** (string): numero di telefono da associare al contatto. **ContactID** (integer): ID del contatto da associare al numero di telefono.

2.5 Dizionario delle associazioni

Associazione	Descrizione	Tabelle Coinvolte
User-Contact	Esprime l'appartenenza di un contatto ad una rubrica gestita da un utente	**User** ruolo manage [0...*]: un utente può gestire più contatti all'interno della rubrica. **Contact** ruolo managed by [1]: il contatto è gestito da un utente.
User-Group	Esprime l'appartenenza ad una rubrica di un gruppo creato da un utente	**User** ruolo creates [0...*]: un utente può creare più gruppi all'interno della rubrica. **Group** ruolo created by [1]: il gruppo viene creato dell'utente.
Group-Participant	Indica a quale a quale gruppo si riferisce la partecipazione di un contatto	**Group** ruolo has [1...*]: un gruppo ha uno o più partecipanti. **Participant** ruolo in [1]: un partecipante è presente all'interno di un gruppo.
Contact-Participant	Indica quale contatto è partecipante ad un gruppo	**Contact** ruolo is [0...*]: un contatto può essere partecipante a più gruppi. **Participant** ruolo refers to [1]: un partecipante si riferisce ad un contatto esistente.
Contact-AssignedAddress	Indica quali indirizzi fisici assegnare ad un contatto	**Contact** ruolo has [0...*]: un contatto può avere più indirizzi assegnati. **AssignedAddress** ruolo refers to [1]: l'indirizzo assegnato fa riferimento ad un contatto esistente.
Address-AssignedAddress	Indica quali contatti assegnare ad un indirizzo fisico	**Address** ruolo assigned by [1...*]: un indirizzo è assegnato da una o più istanze di AssignedAddress. **AssignedAddress** ruolo refers to [1]: un indirizzo assegnato fa riferimento ad un indirizzo esistente.
Contact-Email	Indica a quale contatto si riferisce una e-mail	**Contact** ruolo has [0...*]: un contatto può avere più e-mail AssignedAddress. **Email** ruolo of [1]: una e-mail può essere di un solo contatto.

Messaging-Email	Indica a quale e-mail è collegato un account di messaging	**Messaging** ruolo connected by [1]: un account messaging è collegato tramite una e-mail. **Email** ruolo connected to [0...*]: una e-mail può essere collegato a più account di messaging.
Contact-AssignedPhone	Indica quali numeri di telefono assegnare ad un contatto	**Contact** ruolo has [1...*]: un contatto ha uno o più numeri di telefono assegnati. **AssignedPhone** ruolo refers to [1]: un numero di telefono assegnato fa riferimento ad un contatto esistente.
AssignedPhone-PhoneNumber	Indica quali contatti assegnare ad un numero di telefono	**AssignedPhone** ruolo assigned by [1...*]: un numero di telefono è assegnato da una o più istanze di AssignedPhone. **PhoneNumber** ruolo refers to [1]: un numero di telefono assegnato fa riferimento ad un numero di telefono esistente.
PhoneCall-User	Indica per quale utente è stata registrata una chiamata	**PhoneCall** ruolo registered for [1]: una chiamata viene registrata per un utente. **User** ruolo sent or received [0...*]: un utente può effettuare o ricevere più chiamate.

2.6 Dizionario dei vincoli

Nome	Tipo	Descrizione
emailFormat	n-upla	Il formato di un e-mail deve essere così composto: almeno un carattere, una @, almeno un carattere, un punto e almeno un carattere.
passwordLen	n-upla	Una password deve contenere almeno 8 caratteri.
usernameLen	n-upla	Il nickname dell'user deve contenere almeno 3 caratteri.
checkNumberType	Dominio	La variabile Type di PhoneNumber può assumere solo i valori "MOBILE" e "LANDLINE".
checkCallType	Dominio	La variabile Type di Call può assumere solo i valori "SENT", "ENTERED" o "MISSED".
PK_messaging	Intra-relazionale (primary key)	Può esistere un solo tipo di account social collegato ad una email.
distinctEmail	Intra-relazionale	Non possono esistere due contatti associati alla stessa e-mail.
FK_messsagingEmail	Inter-relazionale (foreign key)	L'indirizzo e-mail di un account di messaging di un contatto deve esistere necessariamente tra gli account di posta già salvati per il contatto.
uniqueMainEmail	Inter-relazionale	Ad un contatto può essere associata una sola e-mail di tipo main.
uniqueMainAddress	Inter-relazionale	Ad un contatto può essere associato un solo indirizzo fisico di tipo main.
uniqueLinked Number	Inter-relazionale	Ad un contatto può essere associato un solo numero di telefono MOBILE di reindirizzamento, ed un solo numero LANDLINE di reindirizzamento.
checkContact Numbers	Inter-relazionale	Ogni contatto deve avere almeno un telefono fisso e un telefono mobile.
timeConsistancy Group	Inter-relazionale	La joinDate di un partecipante in un gruppo deve essere successiva alla data di creazione del gruppo.

Capitolo 3

Schema logico

*alcuni nomi sono stati cambiati poiché parole chiave di postgresql

R_USER	(**email**, passW, nickName, notes)
CONTACT	(**contactID**, contactName, surname, propic, favorite, r_user) *r_user → R_User.email*
R_GROUP	(**groupID**, groupName, creationDate, Description, r_user) *r_user → R_User.email*
PARTICIPANT	(contactID, groupID, joinDate) *contactID → Contact.contactID* *groupID → Group.groupID*
ADDRESS	(**street, zipCode**, city, province, country)
EMAIL	(**email**, main, contactID) *contactID → Contact.contactID*
MESSAGING	(**supplierName, assEmail**, nickName, status) *assEmail → Email.email*
PHONENUMBER	(**phoneNumber**, phoneType, linkedNumber)
PHONECALL	(**callID**, callType, phoneNumber, contactName, dateH, duration, r_user) *r_user → R_User.email*
ASSIGNEDADDRESS	(contactID, addressStr, addressZip, main) *contactID → Contact.contactID* *(addressStr, addressZip) → (Address.street, Address.zipCode)*

ASSIGNEDPHONE	(<u>contactID</u>, <u>phoneNumber</u>) *contactID* → *Contact.contactID* *phoneNumber* → *PhoneNumber.phoneNumber*

Capitolo 4

Progettazione Fisica

4.1 Definizione Tabelle

```
CREATE TABLE R_User(
  email VARCHAR(64) NOT NULL PRIMARY KEY,
  passW VARCHAR(64) NOT NULL,
  nickname VARCHAR(64),
  notes VARCHAR(1024)
);

CREATE TABLE Contact(
  contactID SERIAL NOT NULL PRIMARY KEY,
  contactName VARCHAR(32) NOT NULL,
  surname VARCHAR(32) NOT NULL,
  propic VARCHAR(1024) DEFAULT 'C:\Users\Velios\Desktop\Uni\cd\
    defaultpic',
  favorite BOOLEAN NOT NULL DEFAULT 'false',
  r_user VARCHAR(64) NOT NULL,
  CONSTRAINT FK_userContact FOREIGN KEY (r_user) REFERENCES R_User(
    email) ON DELETE CASCADE
);

CREATE TABLE R_Group(
  groupID SERIAL NOT NULL PRIMARY KEY,
  groupName VARCHAR(32) NOT NULL,
  creationDate DATE NOT NULL,
  description VARCHAR(64),
  r_user VARCHAR(64) NOT NULL,
  CONSTRAINT FK_userGroup FOREIGN KEY (r_user) REFERENCES R_User(
    email) ON DELETE CASCADE
);

CREATE TABLE Participant(
  groupID INTEGER NOT NULL,
  contactID INTEGER NOT NULL,
  joinDate DATE NOT NULL,
  CONSTRAINT FK_groupIDParticipant FOREIGN KEY (groupID) REFERENCES
    R_Group(groupID) ON DELETE CASCADE,
  CONSTRAINT FK_contactIDParticipant FOREIGN KEY (contactID)
    REFERENCES Contact(contactID) ON DELETE CASCADE
```

```
);

CREATE TABLE Address(
  street VARCHAR(1024) NOT NULL,
  zipCode CHAR(5) NOT NULL,
  city VARCHAR(32),
  province CHAR(2),
  country VARCHAR(32),
  CONSTRAINT PK_Address PRIMARY KEY (street, zipcode)
);

CREATE TABLE Email(
  email VARCHAR(1024) NOT NULL PRIMARY KEY,
  main BOOLEAN NOT NULL DEFAULT 'false',
  contactID INTEGER,
  CONSTRAINT FK_contactEmail FOREIGN KEY (contactID) REFERENCES
    Contact(contactID) ON DELETE SET NULL
);

CREATE TABLE Messaging(
  supplierName VARCHAR(64) NOT NULL,
  nickname VARCHAR(64) NOT NULL,
  assEmail VARCHAR(1024) NOT NULL,
  status VARCHAR(1024),
  CONSTRAINT PK_Messaging PRIMARY KEY (supplierName, assEmail),
  CONSTRAINT FK_AssEmail FOREIGN KEY (assEmail) REFERENCES Email(
    email) ON DELETE CASCADE
);

CREATE TABLE PhoneNumber(
  phoneType VARCHAR(8) NOT NULL,
  phoneNumber CHAR(10) NOT NULL PRIMARY KEY,
  linkedNumber BOOLEAN NOT NULL DEFAULT 'false'
);

CREATE TABLE PhoneCall(
  callID SERIAL NOT NULL PRIMARY KEY,
  callType VARCHAR(32) NOT NULL,
  dateH TIMESTAMP NOT NULL,
  phoneNumber CHAR(10) NOT NULL,
  duration INTERVAL,
  contactName VARCHAR(64) DEFAULT 'UNKNOWN',
  r_user VARCHAR(64) NOT NULL,
  CONSTRAINT FK_userCall FOREIGN KEY (r_user) REFERENCES R_User(
    email) ON DELETE CASCADE
);

CREATE TABLE AssignedAddress(
  contactID INTEGER NOT NULL,
  addressStr VARCHAR(126) NOT NULL,
  addressZip VARCHAR(5) NOT NULL,
  main BOOLEAN DEFAULT 'false',
  CONSTRAINT FK_ADContactID FOREIGN KEY (contactID) REFERENCES
    Contact(contactID) ON DELETE CASCADE,
  CONSTRAINT FK_ADAddress FOREIGN KEY (addressStr, addressZip)
    REFERENCES Address(street, zipCode) ON DELETE CASCADE
```

```
84 );
85
86 CREATE TABLE AssignedPhone(
87   phoneNumber CHAR(10) NOT NULL,
88   contactID INTEGER,
89   CONSTRAINT FK_APContactID FOREIGN KEY (ContactId) REFERENCES
      Contact(contactID) ON DELETE SET NULL,
90   CONSTRAINT FK_phoneNumber FOREIGN KEY(phoneNumber) REFERENCES
      PhoneNumber(phoneNumber) ON DELETE CASCADE
91 );
```

Listing 4.1: PostgresSQL

4.2 Definizione dei vincoli

```
-- Implementazione del vincolo emailFormat
ALTER TABLE Email ADD CONSTRAINT emailFormat CHECK ( email LIKE '_%@_%.__%');
ALTER TABLE R\_user ADD CONSTRAINT emailFormat CHECK ( email LIKE '_%@_%.__\%');

-- Implementazione dei vincoli passwordLen e usernameLen
ALTER TABLE R\_User
ADD CONSTRAINT passwordLen CHECK (LENGTH(passw)>7),
ADD CONSTRAINT usernameLen CHECK (LENGTH(nickname)>2);

-- Implementazione del vincolo checkNumberType
ALTER TABLE PhoneNumber
ADD CONSTRAINT checkPhoneNumberType CHECK (phonetype in ('MOBILE', 'LANDLINE'));

-- Implementazione del vincolo checkCallType
ALTER TABLE PhoneCall
ADD CONSTRAINT checkCallType CHECK (callType in ('SENT', 'ENTERED', 'MISSED'));

-- Implementazione del vincolo distinctEmail
ALTER TABLE Email
ADD CONSTRAINT distinctEmail UNIQUE (email, contactID);

--Implementazione del vincolo uniqueMainEmail
CREATE FUNCTION uniqueMainEmail()
RETURNS TRIGGER AS $uniqueMainEmail$
  BEGIN
    IF NEW.main=true AND NEW.contactID IN (SELECT contactID FROM Email WHERE main=true) THEN
        RAISE EXCEPTION 'A main e-mail for this contact already exists!';
      END IF;
RETURN NEW;
END;
$uniqueMainEmail$ LANGUAGE plpgsql;

CREATE TRIGGER uniqueMainEmail
BEFORE INSERT OR UPDATE ON Email
FOR EACH ROW
EXECUTE PROCEDURE uniqueMainEmail();

-- Implementazione del vincolo uniqueMainAddress
CREATE FUNCTION uniqueMainAddress()
RETURNS TRIGGER AS $uniqueMainAddress$
  BEGIN
    IF NEW.main = true AND NEW.contactID IN (SELECT contactID FROM AssignedAddress WHERE main = true) THEN
      RAISE EXCEPTION 'A main Address for this contact already exists!';
    END IF;
  RETURN NEW;
```

```
  END;
$uniqueMainAddress$ LANGUAGE plpgsql;

CREATE TRIGGER uniqueMainAddress
BEFORE INSERT OR UPDATE ON AssignedAddress
FOR EACH ROW
EXECUTE PROCEDURE uniqueMainAddress();

-- Implementazione del vincolo uniqueLinkedNumber
CREATE FUNCTION uniqueLinkedNumber()
RETURNS TRIGGER AS $uniqueLinkedNumber$
DECLARE
    Condizione boolean;
    Tipo varchar(10);
BEGIN
    SELECT PN.linkedNumber INTO Condizione FROM PhoneNumber AS PN
   WHERE  PN.phoneNumber = new.phoneNumber;
    SELECT PN.phoneType INTO Tipo FROM PhoneNumber AS PN WHERE PN.
   phoneNumber = new.phoneNumber;
  IF(Condizione = true) THEN
    IF EXISTS (SELECT * FROM AssignedPhone AS AP, PhoneNumber AS PN
    WHERE PN.linkedNumber = true AND AP.contactID = new.contactID
   AND PN.phoneType = Tipo AND AP.phoneNumber = PN.phoneNumber)
   THEN
        RAISE EXCEPTION 'A linked number already exists for this
   contact !';
    END IF;
  END IF;
  RETURN NEW;
END;
$uniqueLinkedNumber$ LANGUAGE plpgsql;

CREATE TRIGGER uniqueLinkedNumber
BEFORE INSERT OR UPDATE ON AssignedPhone
FOR EACH ROW
EXECUTE PROCEDURE uniqueLinkedNumber();

-- Implementazione del vincolo checkContactNumbers
CREATE FUNCTION checkContactNumbers()
RETURNS TRIGGER AS $checkContactNumbers$
    DECLARE
        ConID integer := OLD.contactID;
  BEGIN
      IF NOT EXISTS (SELECT *
                     FROM AssignedPhone as AP, PhoneNumber as PN
                     WHERE AP.phoneNumber = PN.phoneNumber and
   ConID = AP.contactID and PN.phoneType = 'MOBILE')
    OR NOT EXISTS (SELECT *
                     FROM AssignedPhone as AP, PhoneNumber as PN
                     WHERE AP.phoneNumber = PN.phoneNumber and
   ConID = AP.contactID and PN.phoneType = 'LANDLINE')
      THEN
            RAISE EXCEPTION 'A contact must have at least a
   landline number and a mobile number';
    END IF;
  RETURN NEW;
```

```
93    END;
94  $checkContactNumbers$ LANGUAGE plpgsql;
95
96  CREATE TRIGGER checkContactNumbers
97  AFTER DELETE OR UPDATE ON AssignedPhone
98  FOR EACH ROW
99  EXECUTE PROCEDURE checkContactNumbers();
100
101 -- Implementazione del vincolo timeConsistancyGroup
102 CREATE FUNCTION timeConsistancyGroup()
103 RETURNS TRIGGER AS $timeConsistancyGroup$
104 DECLARE
105   DataCreateGrp DATE;
106 BEGIN
107   SELECT creationDate INTO DataCreateGrp FROM R\_GROUP WHERE
      groupID = new.groupID;
108   IF(DataCreateGrp > new.joinDate) THEN
109     RAISE EXCEPTION 'Participant join date shouldnt be older
      than group creation date !';
110   END IF;
111   RETURN NEW;
112 END;
113 $timeConsistancyGroup$ LANGUAGE plpgsql;
114
115 CREATE TRIGGER timeConsistancyGroup
116 BEFORE INSERT OR UPDATE ON Participant
117 FOR EACH ROW
118 EXECUTE PROCEDURE timeConsistancyGroup()
```

Listing 4.2: PostgresSQL

4.3 Trigger e Automazioni

```
-- Alla rimozione di un contatto da un Gruppo, se quel Gruppo non ha pi  partecipanti viene eliminato

CREATE FUNCTION AutoDeleteGroup()
RETURNS TRIGGER AS $AutoDeleteGroup$
  BEGIN
    IF NOT EXISTS (SELECT * FROM Participant as Par WHERE OLD.groupID = Par.groupID)
    THEN
      DELETE FROM R_Group as rg
      WHERE OLD.groupID = rg.groupID;
    END IF;
  RETURN NEW;
  END;
$AutoDeleteGroup$ LANGUAGE plpgsql;

CREATE TRIGGER AutoDeleteGroup
AFTER DELETE ON Participant
FOR EACH ROW
EXECUTE PROCEDURE AutoDeleteGroup();



-- La seguente procedura prende in input un numero telefonico e se non ci sono contatti associati ad esso lo elimina

CREATE PROCEDURE AutoDeletePhone(PNumber char(10))
LANGUAGE plpgsql AS $AutoDeletePhone$
  BEGIN
    IF NOT EXISTS (SELECT * FROM AssignedPhone as AP WHERE PNumber = AP.phoneNumber and AP.contactID <> NULL)
    THEN
      DELETE FROM PhoneNumber as PN
      WHERE PN.phoneNumber = PNumber;
    END IF;
  END;
$AutoDeletePhone$;



-- il seguente trigger chiama la procedura AutoDeletePhone alla rimozione di un contatto dalla tabella AssignedPhone

CREATE FUNCTION CallDeletePhone()
RETURNS TRIGGER AS $CallDeletePhone$
  BEGIN
    CALL AutoDeletePhone(OLD.phonenumber);
  RETURN NEW;
  END;
$CallDeletePhone$ LANGUAGE plpgsql;

CREATE TRIGGER CallDeletePhone
AFTER UPDATE ON AssignedPhone
```

```
49 FOR EACH ROW
50 WHEN (NEW.contactID = NULL)
51 EXECUTE PROCEDURE CallDeletePhone();
52 -- NOTA: Quando un contatto viene eliminato, la foreign key
       a l l interno  della tabella AssignedPhone viene impostata a NULL
       (si veda tabella di definizione) per evitare conflitti con i
      trigger che implementano i vincoli
53
54
55
56 -- Alla rimozione di un assegnazione di un indirizzo ad un contatto
      , se  q u e l l indirizzo  non ha altre assegnazioni viene eliminato
57
58 CREATE FUNCTION AutoDeleteAddress()
59 RETURNS TRIGGER AS $AutoDeleteAddress$
60   BEGIN
61     IF NOT EXISTS (SELECT * FROM AssignedAddress as AA WHERE OLD.
      addressStr = AA.addressStr and OLD.addressZip = AA.addressZip)
62     THEN
63       DELETE FROM Address
64       WHERE Address.street = OLD.addressStr and Address.zipcode =
      OLD.addressZip;
65     END IF;
66   RETURN NEW;
67   END;
68 $AutoDeleteAddress$ LANGUAGE plpgsql;
69
70 CREATE TRIGGER AutoDeleteAddress
71 AFTER DELETE ON AssignedAddress
72 FOR EACH ROW
73 EXECUTE PROCEDURE AutoDeleteAddress();
74
75
76
77 -- Alla creazione di un Gruppo, la data di creazione viene
      impostata a quella corrente
78
79 CREATE FUNCTION SetGroupDate()
80 RETURNS TRIGGER AS $SetGroupDate$
81   BEGIN
82     UPDATE R_Group as RG
83     SET NEW.creationDate = CURRENT_DATE
84     WHERE NEW.groupID = RG.groupID;
85     RETURN NEW;
86   END;
87 $SetGroupDate$ LANGUAGE plpgsql;
88
89 CREATE TRIGGER SetGroupDate
90 AFTER INSERT ON R_Group
91 FOR EACH ROW
92 EXECUTE PROCEDURE SetGroupDate();
93
94
95
96 -- La seguente procedura riceve in ingresso un numero telefonico e
      un utente, cerca il nome e il cognome dei contatti associati ad
```

```
    essi, e li scrive nella colonna contactName di PhoneCall dove il
     numero telefonico corrisponde. Se non ci sono contatti
    associati al numero telefonico, scrive     Unknown
97
98  CREATE PROCEDURE UpdCallContactName(PNumber char(10), usr R_User.
    email%type)
99  LANGUAGE plpgsql AS $UpdCallContactName$
100 DECLARE
101   resultString varchar(1024) := '';
102   namesur CURSOR  FOR (SELECT contactName, surname FROM PhoneNumber
      as PN, AssignedPhone as AP, Contact as Con WHERE PN.phoneNumber
      = AP.phoneNumber and AP.contactID = Con.contactID and PN.
    phoneNumber = PNumber and Con.r_user = usr);
103 BEGIN
104   FOR nome IN namesur LOOP
105     resultString := resultString || nome.contactName || ' ' || nome
    .surname || ', ';
106   END LOOP;
107   resultString := rtrim(resultString, ', ');
108   IF resultString = '' THEN
109     resultString := 'Unknown';
110   END IF;
111   UPDATE PhoneCall as PC
112   SET contactName = resultString
113   WHERE PC.phoneNumber = PNumber;
114 END;
115 $UpdCallContactName$
116
117
118 -- Chiamata della procedura UpdCallContactName  a l l inserimento  di
     una nuova riga nella tabella PhoneCall
119
120 CREATE FUNCTION UpdPhoneCall()
121 RETURNS TRIGGER AS $UpdPhoneCall$
122   BEGIN
123     CALL UpdCallContactName(NEW.phoneNumber, NEW.r_user);
124     RETURN NEW;
125   END;
126 $UpdPhoneCall$ LANGUAGE plpgsql;
127
128 CREATE TRIGGER UpdPhoneCall
129 AFTER INSERT ON PhoneCall
130 FOR EACH ROW
131 EXECUTE PROCEDURE UpdPhoneCall();
132
133
134 -- Chiamata della procedura UpdCallContactName  a l l assegnazione
    di un numero di telefono ad un contatto
135
136 CREATE FUNCTION UpdCallFromAP()
137 RETURNS TRIGGER AS $UpdCallFromAP$
138   DECLARE
139     currentUser varchar(10);
140   BEGIN
141     SELECT Con.r_user into currentUser
142     FROM Contact as Con
```

```
143     WHERE NEW.contactID = Con.contactID;
144     CALL UpdCallContactName(NEW.phoneNumber, currentUser);
145     RETURN NEW;
146   END;
147 $UpdCallFromAP$ LANGUAGE plpgsql;
148 
149 CREATE TRIGGER UpdCallFromAP
150 AFTER INSERT or DELETE ON AssignedPhone
151 FOR EACH ROW
152 EXECUTE PROCEDURE UpdCallFromAP();
```

Listing 4.3: PostgresSQL