Università di Federico II

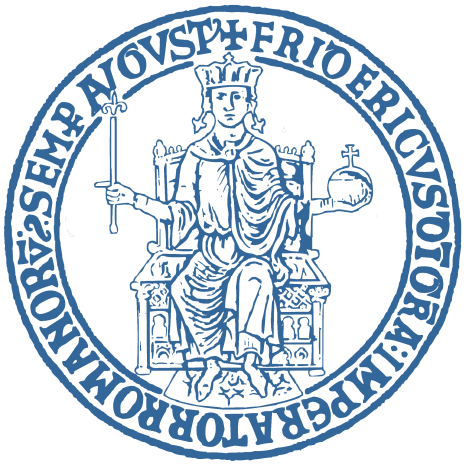
Basi di Dati

Progettazione e Sviluppo di una Rubrica Telefonica Avanzata

Authors
Oleksandr Sosovskyy
Francesco Magri

Professor
Prof. Silvio Barra

Ottobre 2022



Indice

1 Requisiti identificati **3**
- 1.1 Introduzione . . . 3
- 1.2 Requisiti sui dati . . . 3
- 1.3 Requisiti sulle operazioni . . . 4

2 Progettazione concettuale **5**
- 2.1 Class Diagram . . . 5
- 2.2 Analisi della ristrutturazione del Class Diagram . . . 6
  - 2.2.1 Analisi delle ridondanze . . . 6
  - 2.2.2 Analisi degli identificativi . . . 6
  - 2.2.3 Rimozione degli attributi multipli . . . 6
  - 2.2.4 Rimozione degli attributi composti . . . 6
  - 2.2.5 Rimozione delle gerarchie . . . 7
  - 2.2.6 Partizione/Accorpamento di entità e associazioni . . . 7
  - 2.2.7 Rimozione delle Enumeration . . . 7
- 2.3 Class Diagram ristrutturato . . . 8
- 2.4 Dizionario delle classi . . . 9
- 2.5 Dizionario delle associazioni . . . 11
- 2.6 Dizionario dei vincoli . . . 13

3 Progettazione logica **14**
- 3.1 Schema logico . . . 14

4 Progettazione Fisica **16**
- 4.1 Eccezioni . . . 16
- 4.2 Domini . . . 16
- 4.3 Tabelle . . . 17
  - 4.3.1 Rubrica . . . 17
  - 4.3.2 Gruppo . . . 17
  - 4.3.3 Contatto . . . 17
  - 4.3.4 Account . . . 18
  - 4.3.5 Email . . . 18
  - 4.3.6 Indirizzo . . . 18
  - 4.3.7 Telefono . . . 19

4.3.8 Associa . . . 19
4.3.9 Composizione . . . 19
4.4 Trigger . . . 20
4.4.1 Introduzione . . . 20
4.4.2 group_membership_coherency . . . 20
4.4.3 block_homonymous_groups_in_same_rubric . . . 21
4.4.4 block_void_groups_insertion . . . 22
4.4.5 unique_email . . . 22
4.4.6 undeletable_main_address . . . 23
4.4.7 unchangeable_main_address_description . . . 23
4.4.8 block_contact_without_both_numbers . . . 24
4.4.9 block_contact_without_main_address . . . 25
4.4.10 check_mobile_landline_numbers_existence . . . 25
4.5 Automazioni . . . 26
4.5.1 Introduzione . . . 26
4.5.2 automatic_void_groups_deletion . . . 26
4.5.3 automatic_email_association . . . 27
4.5.4 automatic_account_association . . . 27
4.6 Funzioni . . . 28
4.6.1 Introduzione . . . 28
4.6.2 generate_new_contatto_id . . . 29
4.6.3 generate_new_gruppo_id . . . 29
4.6.4 coherent_insertion_f . . . 29
4.6.5 reindirizza . . . 30
4.6.6 cerca . . . 31
4.6.7 cerca_per_nome . . . 32
4.6.8 cerca_per_numero . . . 32
4.6.9 cerca_per_email . . . 32
4.6.10 cerca_per_account . . . 33
4.7 Codice SQL e Popolamento . . . 33

5 Manuale d'uso per l'Applicativo **34**
5.1 Installazione . . . 34
5.2 Esempi di funzionalità notevoli . . . 35

Capitolo 1

Requisiti identificati

1.1 Introduzione

Si progetterà una base di dati relazionale per la gestione di una Rubrica Telefonica Avanzata.

A seguito di alcune problematicità riscontrate (come la possibilità per più persone di usare lo stesso dispositivo, e quindi il medesimo applicativo), si è deciso consapevolmente di optare per un'implementazione "multi-rubrica": è ammessa la possibilità per più persone di gestire una rubrica personale, i cui contatti mantengono la completa indipendenza informativa rispetto a eventuali omonimi presenti in altre rubriche.

Ogni contatto dispone obbligatoriamente di almeno due numeri di telefono (di cui uno deve essere mobile e l'altro fisso, in modo che il primo possa reindirizzare sempre il secondo, e viceversa) e di un indirizzo principale. Altri numeri di telefono, indirizzi secondari o email possono essere associati facoltativamente. Dei gruppi possono raccogliere da uno a più contatti (i gruppi vuoti non sono ammessi).

Infine, un indirizzo elettronico associa un contatto ai suoi account presso i sistemi di messaging. La Base di Dati ignora il modo in cui le suddette informazioni esterne vengano reperite: si ammette che siano recuperate dal dispositivo in uso magari attraverso delle API esterne, inserite nella base di dati e, alla dichiarazione della corretta email, associate al contatto corrispondente. La Base di Dati si occupa solo del terzo e ultimo passaggio indicato.

1.2 Requisiti sui dati

La Base di dati gestisce le seguenti classi di dati:

- **Rubrica** indica l'insieme delle *Rubriche* aggiunte dagli utenti;

- **Elemento** indica la generalizzazione degli elementi presenti in una *Rubrica*, vale a dire i *contatti* e i *gruppi*: senza almeno un *elemento* la *Rubrica* è vuota;
- **Contatto** indica gli *elementi* di una *Rubrica* che possiedono dei campi informativi, come i numeri di telefono, indirizzi ed email. Un contatto ha da due a più numeri di telefono e a ciascuno di questi sono associabili possibili descrizioni (come: mobile, fisso, ufficio, ecc.);
- **Gruppo** indica gli *elementi* che sono collezioni di *contatti*;
- **Account** indica le informazioni associate a una email registrata su qualche sistema di messaging. Tali informazioni sono il nome del fornitore, l'indirizzo email, il nickname e la frase di benvenuto.

1.3 Requisiti sulle operazioni

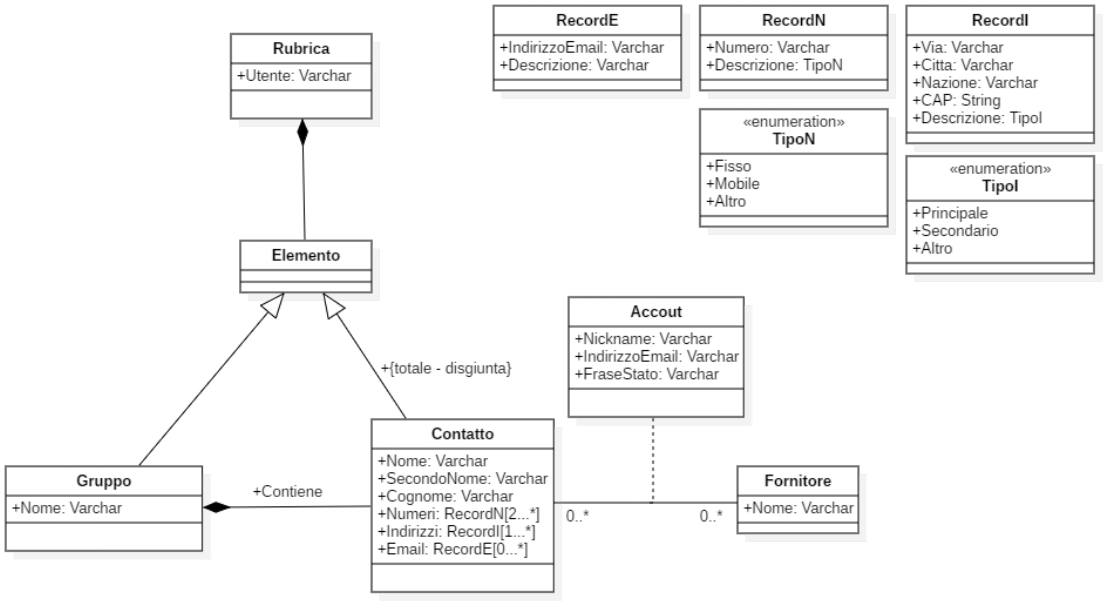
La Base di dati consente le seguenti operazioni sui dati:

- Aggiunta e cancellazione di *gruppi*, *contatti* e di tutte le informazioni relative;
- Manipolazione dei dati di un *Contatto* o di un *Gruppo*;
- Ricerca di *contatti* per nome, email, account di messaging e numero di telefono;
- Reindirizzamento della chiamata da un numero a un altro.

Capitolo 2

Progettazione concettuale

2.1 Class Diagram



2.2 Analisi della ristrutturazione del Class Diagram

2.2.1 Analisi delle ridondanze

Per quanto riguarda le email, un Contatto condivide gli stessi indirizzi di posta elettronica dei propri Account presso i sistemi di messanging. Tuttavia la ridondanza non è eliminabile, per come funziona il sistema: l'aggiunta di un nuovo account comporta la sua associazione ai rispettivi contatti, e viceversa. Rimuovendo questa apparente ridondanza, l'associazione tra contatti e account non può avvenire in alcun altro modo.

Altra ridondaza è quella relativa alle informazioni di un contatto, quali numeri, indirizzi ed email, condivise con altri contatti, della stessa Rubrica o meno. Se le ridondaze venissero rimosse, allora la modifica o eliminazione voluta per un campo informativo di un certo contatto si ripercuoterebbe in modo indesiderato sugli altri contatti che lo avrebbero in comune. Anche in questo caso, la ridondanza non può essere rimossa.

2.2.2 Analisi degli identificativi

Per il modo in cui si è scelto di implementare il sistema, è indispensabile che rubriche diverse consentano la presenza di omonimi (gruppi, contatti, numeri, indirizzi ed email). A queste entità sono aggiunti i codici identificativi.

Si procede in modo diverso per le seguenti due entità:

- *Account*, può essere identificato dal nome del fornitore e dall'indirizzo email che è unico per ogni account.
- *Rubrica*, può essere identificata dal nome del proprietario della rubrica. Questa scelta impedisce che vengano create rubriche diverse con lo stesso nome.

2.2.3 Rimozione degli attributi multipli

Nella classe Contatto sono presenti i seguenti attributi multipli: numeri, email e indirizzi. Questi attributi portano informazioni non eliminabili nella ristrutturazione e possono esistere in numero arbitrariamente grande. Sono state create delle classi per ciascuno degli attributi multipli.

2.2.4 Rimozione degli attributi composti

Gli attributi composti coincidono con gli attributi multipli. È dunque indispensabile l'introduzione di classi per ciascuno di questi.

2.2.5 Rimozione delle gerarchie

La classe Elemento rappresenta una generalizzazione delle classi Contatto e Gruppo. Queste due sono troppo diverse per essere identificate da una classe Elemento. Dunque si elimina la generalizzazione e si introducono due classi.

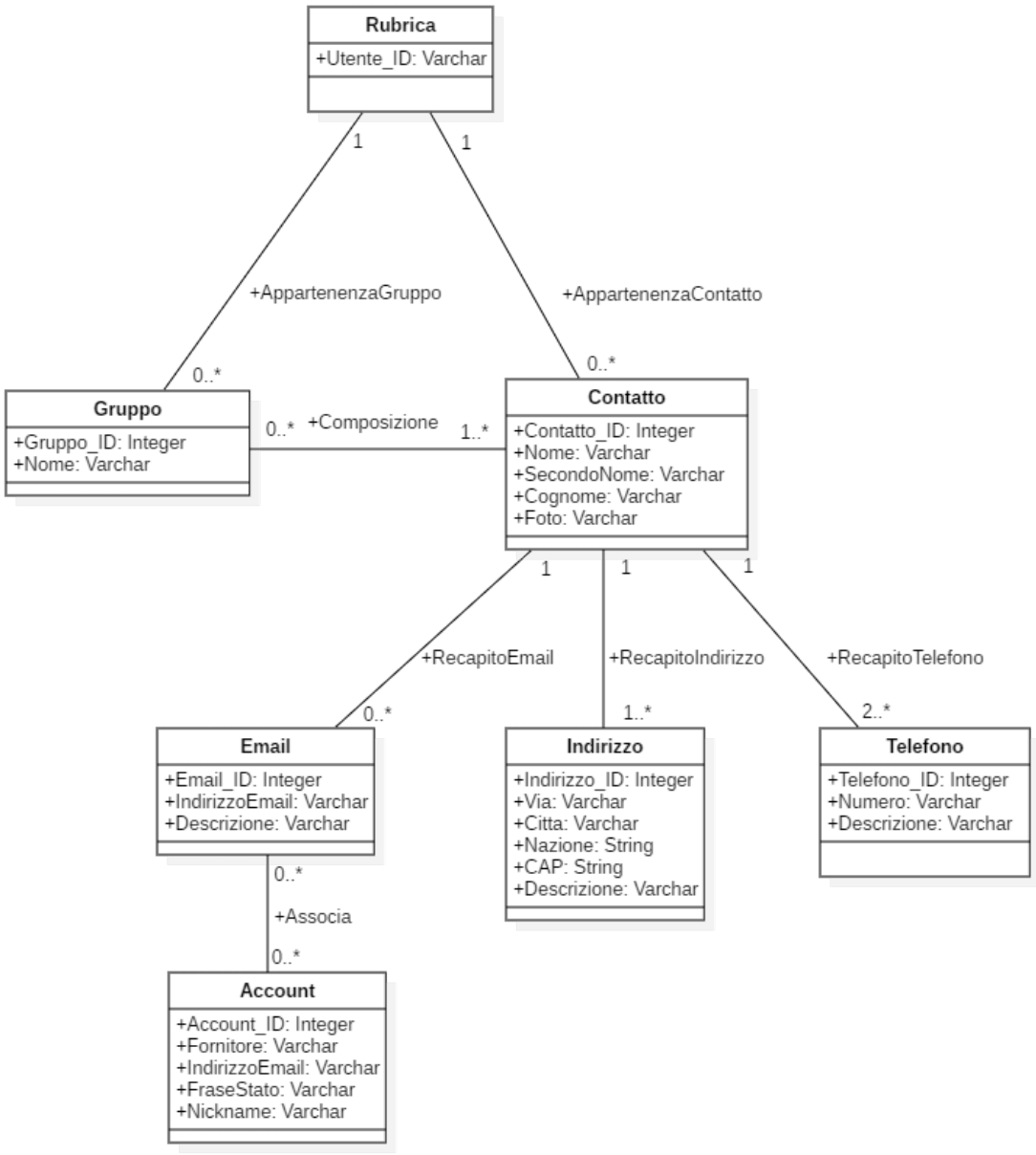
2.2.6 Partizione/Accorpamento di entità e associazioni

Si decide di accorpare le entità *Account* e *Fornitore* poiché il nome del fornitore può essere inteso come un attributo dell'account, recuperato con tutte le sue altre informazioni.

2.2.7 Rimozione delle Enumeration

Si sceglie di rimuovere tutte le enumeration del diagramma concettuale per garantire la massima flessibilità d'uso. Tutti i vincoli relativi a una certa descrizione di qualche entità (ad esempio, numero fisso o mobile, oppure indirizzo principale o secondario) sono gestiti da trigger e altri costrutti di Postgres.

2.3 Class Diagram ristrutturato



2.4 Dizionario delle classi

Classe	*Descrizione*	*Attributi*
Rubrica	Descrittore di un'entità rubrica aggiunta da qualche utente al sistema	*Utente_ID* (varchar): Chiave tecnica. Identifica l'utente proprietario di una rubrica.
Gruppo	Descrittore di una collezione di contatti in rubrica	*Gruppo_ID* (integer): Chiave tecnica. Identifica univocamente ciascun gruppo; *Nome* (varchar): Nome del gruppo.
Contatto	Descrittore di un'entità contatto di una rubrica	*Contatto_ID* (integer): Chiave tecnica. Identifica univocamente ogni contatto; *Nome* (varchar): indica il primo nome del contatto; *SecondoNome* (varchar): indica il secondo nome del contatto; *Cognome* (varchar): indica il cognome del contatto; Foto (varchar): percorso sul disco dove può essere reperita la foto del contatto.
Email	Descrittore di un entità indirizzo di posta elettronica di un contatto	*Email_ID* (integer): Chiave tecnica. Identifica univocamente un'email di un contatto; *IndirizzoEmail* (varchar): stringa dell'e-mail; *Descrizione* (varchar): informazione aggiuntiva di una email (privata, ufficio, ecc.).

Classe	*Descrizione*	*Attributi*
Indirizzo	Descrittore di un entità indirizzo fisico di un contatto	*Indirizzo_ID* (integer): Chiave tecnica. Identifica univocamente un indirizzo fisico di un contatto; *Via* (varchar): indica la via dell'indirizzo; *Città* (varchar): indica la città dell'indirizzo; *Nazione* (string): indica la nazione dell'indirizzo; *CAP* (string): indica il codice di avviamento postale dell'indirizzo; *Descrizione*(varchar): indica se l'indirizzo è principale, secondario, o altro
Telefono	Descrittore di un'entità numero telefonico di un contatto	*Telefono_ID* (integer): Chiave tecnica. Identifica univocamente un numero di telefono di un contatto; *Numero* (varchar): stringa del numero; *Descrizione* (varchar): informazione aggiuntiva di un numero di telefono (mobile, fisso, ufficio ecc).
Account	Descrittore di un account recuperato da qualche sistema di messanging	*Account_ID* (varchar): Chiave tecnica. Identifica l'utente proprietario di una rubrica; *Fornitore* (varchar): indica il nome del fornitore da cui è recuperato l'account (Whatsapp, Telegram, Teams, ecc..); *IndirizzoEmail* (varchar): stringa dell'indirizzo email associato a un account; *FraseStato* (varchar): frase di benvenuto associata a un account; *Nickname* (varchar): indica il nickname associato a un account.

2.5 Dizionario delle associazioni

Associazione	*Descrizione*	*Classi coinvolte*
AppartenenzaGruppo	Esprime l'appartenenza di un gruppo a una rubrica	*Rubrica* [1]: indica la rubrica che contiene un gruppo; *Gruppo* [0..*]: indica il gruppo appartenente a una rubrica.
AppartenenzaContatto	Esprime l'appartenenza di un contatto a una rubrica	*Rubrica* [1]: indica la rubrica che contiene un contatto; *Contatto* [0..*]: indica il contatto appartenente a una rubrica.
Composizione	Esprime l'appartenenza di un contatto a un gruppo	*Gruppo* [0...*]: indica il gruppo che contiene un contatto; *Contatto* [1..*]: indica il contatto appartenente a un gruppo.
Associa	Esprime l'associazione di un account a un contatto	*Email* [0...*]: indica l'e-mail a cui è associato un account; *Account* [0..*]: indica l'account che è associato a una email.
RecapitoEmail	Esprime l'appartenenza di una email a un contatto	*Email* [0..*]: indica l'e-mail appartenente ad un contatto; *Contatto* [1]: indica il contatto a cui è associata una email.
RecapitoIndirizzo	Esprime l'appartenenza di un indirizzo fisico a un contatto	*Indirizzo* [1..*]: indica l'indirizzo fisico associato ad un contatto; *Contatto* [1]: indica il contatto a cui è associato un indirizzo fisico.

Associazione	*Descrizione*	*Classi coinvolte*
RecapitoTelefono	Esprime l'appartenenza di un numero di telefono a un contatto	*Telefono* [2..*]: indica il numero di telefono associato ad un contatto; *Contatto* [1]: indica il contatto a cui è associato un numero di telefono.

2.6 Dizionario dei vincoli

Vincolo	*Descrizione*
Group Membership Coherency	Un gruppo può contenere solo i contatti appartenenti alla rubrica del gruppo.
Block Homonymous Groups	In una rubrica non ci possono essere due gruppi aventi lo stesso nome
Block Void Groups	Non possono esistere gruppi vuoti
Mobile And Landline Number Existence	Ogni contatto deve avere almeno un telefono mobile e uno fisso.
Not redundant Telephone Number	Un contatto non può registrare più volte lo stesso numero telefonico.
Not redundant Email	Un contatto non può registare più volte la stessa email.
Email Uniqueness	Non possono esistere due contatti distinti in una stessa rubrica a cui sia associata la stessa email.
Unique Provider-Email	Le informazioni di un'email rilasciate da un fornitore sono uniche.
Main Address Existence	Ogni contatto deve avere un indirizzo principale
Undeletable Main Address	L'indirizzo principale associato a un contatto non può essere eliminato, se non con tutto il contatto, ma può essere soltanto modificato
Unchangeable Main Address Description	Gli indirizzi principali non possono diventare secondari, e viceversa. Il vincolo garantisce che esista sempre un singolo indirizzo principale associato a un contatto.

Capitolo 3

Progettazione logica

3.1 Schema logico

Rubrica	(**Utente_ID**)
Contatto	(**Contatto_ID**, Nome, SecondoNome, Cognome, Foto, Rubrica_FK) Rubrica_FK → Rubrica.Utente_ID
Gruppo	(**Gruppo_ID**, Nome, Rubrica_FK) Rubrica_FK → Rubrica.Utente_ID
Composizione	(Contatto_FK, Gruppo_FK) Contatto_FK → Contatto.Contatto_ID Gruppo_FK → Gruppo.Gruppo_ID
Account	(**Account_ID**, Fornitore, IndirizzoEmail, FraseStato, Nickname)
Associa	(Email_FK, FornitoreAccount, IndirizzoEmailAccount) Email_FK → Email.Email_ID FornitoreAccount → Account.Fornitore IndirizzoEmailAccount → Account.IndirizzoEmail
Email	(**Email_ID**, IndirizzoEmail, Descrizione, Contatto_FK)

	Contatto_FK → Contatto.Contatto_ID
Indirizzo	(**Indirizzo_ID**, Via, Città, CAP, Nazione, Descrizione, Contatto_FK)
	Contatto_FK → Contatto.Contatto_ID
Telefono	(**Telefono_ID**, Numero, Descrizione, Contatto_FK)
	Contatto_FK → Contatto.Contatto_ID

Capitolo 4

Progettazione Fisica

4.1 Eccezioni

Col fine di garantire una gestione più efficace degli errori, sia a livello della Base di dati che dell'Applicativo, le eccezione sono identificate da codici:

- **GMCOH** → *Group Membership COHerency*
- **BLKHG** → *BLocK Homonymous Groups in same rubric*
- **BLKVG** → *BLocK Void Groups insertion*
- **UNIQE** → *UNIQue Email*
- **UDMAD** → *UnDeletable Main ADdress*
- **UMADD** → *Unchangeable Main ADdress Description*
- **BCWBN** → *Block Contact Without Both Numbers*
- **BCWMA** → *Block Contact Without Main Address*
- **CMLNE** → *Check Mobile Landline Numbers Existence*

4.2 Domini

```
1 --Dominio per una Email:
2 --un indirizzo email deve essere strutturato in questo modo: testo
     +"@"+testo+"."+testo.
3 CREATE DOMAIN EmailType AS VARCHAR(256)
4 CHECK (VALUE LIKE '_%@_%.__%');
5
6 --Dominio per il CAP:
7 --il CAP deve essere una stringa di 5 cifre
8  CREATE DOMAIN CAPType AS CHAR(5)
9  CHECK (VALUE ~ '^[0-9]*$');
```

```
10
11 --Dominio per un Numero telefonico:
12 --un numero di telefono ha una lunghezza compresa tra 2 e 15 cifre.
13 CREATE DOMAIN NUMType AS VARCHAR(15)
14  CHECK (VALUE ~ '^[0-9]*$' AND
15       (length(VALUE)) > 2
16        );
```

4.3 Tabelle

4.3.1 Rubrica

```
1 --Tabella Rubrica
2 CREATE TABLE Rubrica(
3 Utente_ID VARCHAR(30)
4 );
5
6 ALTER TABLE Rubrica
7  --Chiave primaria
8  ADD CONSTRAINT rubrica_pk PRIMARY KEY(Utente_ID);
```

4.3.2 Gruppo

```
1 --Tabella Gruppo
2 CREATE TABLE Gruppo(
3 Gruppo_ID SERIAL,
4 Nome VARCHAR(20) NOT NULL,
5 Rubrica_FK VARCHAR(30) NOT NULL
6 );
7
8 ALTER TABLE Gruppo
9  --Chiave primaria
10 ADD CONSTRAINT gruppo_pk PRIMARY KEY(Gruppo_ID),
11 --Chiave esterna
12 ADD CONSTRAINT gruppo_rubrica_fk FOREIGN KEY(Rubrica_FK)
      REFERENCES Rubrica(Utente_ID)
13 ON UPDATE CASCADE ON DELETE CASCADE;
```

4.3.3 Contatto

```
1 --Tabella Contatto
2 CREATE TABLE Contatto(
3 Contatto_ID SERIAL,
4 Nome VARCHAR(20) NOT NULL,
5 SecondoNome VARCHAR(20),
6 Cognome VARCHAR(20) NOT NULL,
7 Foto VARCHAR(1000),
8 Rubrica_FK VARCHAR(20) NOT NULL
9 );
10
11 ALTER TABLE Contatto
12  --Chiave primaria
13  ADD CONSTRAINT contatto_pk PRIMARY KEY(Contatto_ID),
14  --Chiave esterna
15  ADD CONSTRAINT contatto_rubica_fk FOREIGN KEY(Rubrica_FK)
      REFERENCES Rubrica(Utente_ID)
```

```
 ON UPDATE CASCADE ON DELETE CASCADE;
```

4.3.4 Account

```
--Tabella Account
CREATE TABLE Account(
Fornitore VARCHAR(20) NOT NULL,
IndirizzoEmail EmailType NOT NULL,
FraseStato VARCHAR(256),
Nickname VARCHAR(30)
);

ALTER TABLE Account
 --Chiave primaria
 ADD CONSTRAINT account_pk PRIMARY KEY(Fornitore, IndirizzoEmail);
```

4.3.5 Email

```
--Tabella Email
CREATE TABLE Email(
Email_ID SERIAL,
IndirizzoEmail EmailType NOT NULL,
Descrizione VARCHAR(30) NOT NULL,
Contatto_FK SERIAL
);

ALTER TABLE Email
 --Chiave primaria
 ADD CONSTRAINT email_pk PRIMARY KEY(Email_ID),
 --Chiave esterna
 ADD CONSTRAINT email_contatto_fk FOREIGN KEY(Contatto_FK)
    REFERENCES Contatto(Contatto_ID)
 ON UPDATE CASCADE ON DELETE CASCADE,
 --Un contatto non può avere associata due volte la stessa email
 ADD CONSTRAINT not_redundant_email UNIQUE (Contatto_FK,
    indirizzoEmail);
```

4.3.6 Indirizzo

```
--Tabella Indirizzo
CREATE TABLE Indirizzo(
Indirizzo_ID SERIAL ,
Via VARCHAR(20) NOT NULL,
Città VARCHAR(30) NOT NULL,
Nazione VARCHAR(30) NOT NULL,
CAP CAPType NOT NULL,
Descrizione VARCHAR(20) NOT NULL,
Contatto_FK SERIAL
);

ALTER TABLE Indirizzo
 --Chiave primaria
 ADD CONSTRAINT indirizzo_pk PRIMARY KEY(Indirizzo_ID),
 --Chiave esterna
 ADD CONSTRAINT indirizzo_contatto_fk FOREIGN KEY(Contatto_FK)
    REFERENCES Contatto(Contatto_ID)
 ON UPDATE CASCADE ON DELETE CASCADE;
```

4.3.7 Telefono

```
--Tabella Telefono
CREATE TABLE Telefono (
Telefono_ID SERIAL,
Numero NUMType NOT NULL,
Descrizione VARCHAR(20) NOT NULL,
Contatto_FK SERIAL
);

ALTER TABLE Telefono
 --Chiave primaria
 ADD CONSTRAINT telefono_pk PRIMARY KEY(Telefono_ID),
 --Chiave esterna
 ADD CONSTRAINT telefono_contatto_fk FOREIGN KEY(Contatto_FK)
    REFERENCES Contatto(Contatto_ID)
 ON UPDATE CASCADE ON DELETE CASCADE,
 --Un contatto non può avere associato due volte lo stesso numero
    di telefono
 ADD CONSTRAINT not_redundant_telephone_number UNIQUE(Contatto_FK,
    Numero);
```

4.3.8 Associa

```
--Tabella Associa
CREATE TABLE Associa (
Email_FK SERIAL,
FornitoreAccount VARCHAR(20),
IndirizzoEmailAccount EmailType
);

ALTER TABLE Associa
 --Chiave esterna sulla tabella Email
 ADD CONSTRAINT associa_email_fk FOREIGN KEY(Email_FK) REFERENCES
    Email(Email_ID)
 ON UPDATE CASCADE ON DELETE CASCADE,
 --Chiave esterna sulla tabella Account
 ADD CONSTRAINT associa_account_fk FOREIGN KEY(FornitoreAccount,
    IndirizzoEmailAccount) REFERENCES Account(Fornitore,
    IndirizzoEmail)
 ON UPDATE CASCADE ON DELETE CASCADE;
```

4.3.9 Composizione

```
--Tabella Composizione
CREATE TABLE Composizione(
Contatto_FK SERIAL,
Gruppo_FK   SERIAL
);

ALTER TABLE Composizione
 --Chiave esterna sulla tabella Contatto
 ADD CONSTRAINT composizione_contatto_fk FOREIGN KEY(Contatto_FK)
    REFERENCES Contatto(Contatto_ID)
 ON UPDATE CASCADE ON DELETE CASCADE,
 --Chiave esterna sulla tabella Gruppo
```

```
12 ADD CONSTRAINT composizione_gruppo_fk   FOREIGN KEY(Gruppo_FK)
      REFERENCES Gruppo(Gruppo_ID)
13 ON UPDATE CASCADE ON DELETE CASCADE;
```

4.4 Trigger

4.4.1 Introduzione

I vincoli più complessi sono stati implementati mediante l'uso di trigger. Nella sezione presente i trigger individuati sono:

- **group_membership_coherency** : prima dell'inserimento di un contatto in un gruppo, si verifica se i due appartengono alla stessa rubrica;
- **block_homonymous_groups_in_same_rubric** : all'inserimento di un gruppo, si verifica se è già presente un omonimo nella stessa rubrica. Nel caso affermativo, l'inserimento è bloccato;
- **block_void_groups_insertion** : blocca ogni transazione che prevede l'inserimento di un gruppo senza contatti;
- **unique_email** : blocca l'associazione della stessa email a due contatti diversi appartenenti alla stessa rubrica;
- **undeletable_main_address** : blocca la cancellazione di un indirizzo principale se il contatto corrispondente esiste;
- **unchangeable_main_address_description** : blocca ogni tentativo di modifica della descrizione di un indirizzo principale, o di un indirizzo di cui si cerca di cambiare la descrizione in principale;
- **block_contact_without_both_numbers** : blocca ogni transazione che cerca di inserire un contatto senza un numero mobile e uno fisso;
- **block_contact_without_main_address** : blocca ogni transazione che cerca di inserire un contatto senza un indirizzo principale;
- **check_mobile_landline_numbers_existence** : ad ogni cancellazione o modifica di un numero di telefono, blocca ogni operazione che possa lasciare un contatto senza almeno un numero mobile e uno fisso.

4.4.2 group_membership_coherency

```
1 --I membri di un gruppo devono appartenere alla stessa rubrica
2 CREATE OR REPLACE FUNCTION group_membership_coherency_f()
3   RETURNS TRIGGER
4   LANGUAGE PLPGSQL
5   AS $$
6   DECLARE
7     rubrica_c Rubrica.Utente_ID%TYPE;
8     rubrica_g   Rubrica.Utente_ID%TYPE;
```

```
  BEGIN
    SELECT Rubrica_FK INTO rubrica_c
    FROM Contatto
    WHERE Contatto_ID = NEW.Contatto_FK;

    SELECT Rubrica_FK INTO rubrica_g
    FROM   Gruppo
    WHERE  Gruppo_ID = NEW.Gruppo_FK;
    --Se la rubrica del contatto è diversa da quella del gruppo,
    blocca l'inserimento
    IF (rubrica_c <> rubrica_g) THEN
      RAISE EXCEPTION USING ERRCODE = 'GMCOH';
    ELSE
      RETURN NEW;
    END IF;
   EXCEPTION
    WHEN SQLSTATE 'GMCOH' THEN
    RAISE 'Il contatto di ID % non appartiene alla stessa rubrica
         del gruppo di ID %, pertanto l''inserimento è annullato ',
     NEW.Contatto_FK, NEW.Gruppo_FK USING ERRCODE = 'GMCOH';
    RETURN OLD;
  END; $$;

CREATE OR REPLACE TRIGGER group_membership_coherency
  BEFORE INSERT ON Composizione
  FOR EACH ROW
  EXECUTE PROCEDURE group_membership_coherency_f();
```

4.4.3 block_homonymous_groups_in_same_rubric

```
--Blocca l'inserimento di più gruppi aventi lo stesso nome, che
    stanno nella stessa rubrica
CREATE OR REPLACE FUNCTION block_homonymous_groups_in_same_rubric_f
    ()
  RETURNS TRIGGER
  LANGUAGE PLPGSQL
  AS $$
  BEGIN
    -- verifico se esistono degli omonimi nella stessa rubrica
    IF (SELECT COUNT(*) FROM gruppo WHERE nome = NEW.nome AND
    rubrica_fk = NEW.rubrica_fk) > 1
    THEN
      RAISE EXCEPTION USING ERRCODE = 'BLKHG';
    ELSE
      RETURN NEW;
    END IF;
  EXCEPTION
    WHEN SQLSTATE 'BLKHG' THEN
    RAISE 'Si sta provando a inserire gruppo già presente in questa
     rubrica' USING ERRCODE = 'BLKHG';
    RETURN NULL;
  END; $$;

CREATE CONSTRAINT TRIGGER block_homonymous_groups_in_same_rubric
  AFTER INSERT ON Gruppo
  DEFERRABLE
  FOR EACH ROW
```

```
  EXECUTE PROCEDURE block_homonymous_groups_in_same_rubric_f();
```

4.4.4 block_void_groups_insertion

```
--Un gruppo deve avere sempre almeno un contatto
CREATE OR REPLACE FUNCTION block_void_groups_insertion_f()
  RETURNS TRIGGER
  LANGUAGE PLPGSQL
  AS $$
  BEGIN
    -- controllo se esistono gruppi vuoti, cioè senza contatti
    IF (SELECT COUNT(*) FROM Gruppo AS Gr WHERE (SELECT COUNT(*)
    FROM Composizione AS Comp WHERE Comp.gruppo_fk = Gr.gruppo_id)
    < 1) > 0
    THEN
      RAISE EXCEPTION USING ERRCODE = 'BLKVG';
    ELSE
      return NEW;
    END IF;
  EXCEPTION
    WHEN SQLSTATE 'BLKVG' THEN
    RAISE 'Si sta provando a inserire un gruppo vuoto: si richiede
    che un gruppo abbia almeno un contatto' USING ERRCODE = 'BLKVG'
    ;
    RETURN NULL;
  END; $$;

CREATE CONSTRAINT TRIGGER block_void_groups_insertion
  AFTER INSERT ON Gruppo
  DEFERRABLE
  FOR EACH ROW
  EXECUTE PROCEDURE block_void_groups_insertion_f();
```

4.4.5 unique_email

```
--Vincolo per l'unicità dell'email per rubrica
CREATE OR REPLACE FUNCTION unique_email_f()
  RETURNS TRIGGER
  LANGUAGE PLPGSQL
  AS $$
  DECLARE
    cerca_email_uguali CURSOR FOR(
      SELECT *
      FROM Contatto AS CNEW, Contatto AS C1
      WHERE CNEW.Contatto_ID = NEW.Contatto_FK AND
          CNEW.Contatto_ID <> C1.Contatto_ID AND
          CNEW.Rubrica_FK  =  C1.Rubrica_FK  AND
          C1.Contatto_ID IN
          (   SELECT E1.Contatto_FK
              FROM   Email AS E1
              --Confronto CASE INSENSITIVE
              WHERE  lower(E1.IndirizzoEmail) = lower(NEW.
    IndirizzoEmail) )
          );
    email_uguali cerca_email_uguali%TYPE;
  BEGIN
    OPEN cerca_email_uguali;
```

```
22      FETCH cerca_email_uguali INTO email_uguali;
23      --FOUND è una variabile di sistema che, in questo caso,
        verifica
24      --se la fetch ha restituito almeno una tupla, nel qual caso l'
        email già esiste
25      IF FOUND THEN
26        RAISE EXCEPTION USING ERRCODE ='UNIQE';
27      ELSE
28        CLOSE cerca_email_uguali;
29        RETURN NEW;
30      END IF;
31    EXCEPTION
32      WHEN SQLSTATE 'UNIQE' THEN
33      RAISE 'L''email che si intende aggiungere è già in uso da
        qualche altro contatto della stessa rubrica' USING ERRCODE = '
        UNIQE';
34      CLOSE cerca_email_uguali;
35      RETURN OLD;
36    END; $$;
37
38  CREATE OR REPLACE TRIGGER unique_email
39    BEFORE INSERT ON Email
40    FOR EACH ROW
41    EXECUTE PROCEDURE unique_email_f();
```

4.4.6 undeletable_main_address

```
1   --Vincolo che impedisce l'eliminazione di indirizzi con
2   --descrizione "Principale'
3   CREATE OR REPLACE FUNCTION undeletable_main_address_f()
4     RETURNS TRIGGER
5     LANGUAGE PLPGSQL
6     AS $$
7     BEGIN
8       -- Se l'indirizzo è principale e corrisponde a un contatto
        esistente allora blocca la cancellazione dell'indirizzo
9       IF (OLD.Descrizione = 'Principale') AND ((SELECT COUNT(*) FROM
        Contatto WHERE Contatto_ID = OLD.Contatto_FK ) <> 0) THEN
10        RAISE EXCEPTION USING ERRCODE ='UDMAD';
11      ELSE
12        RETURN OLD;
13      END IF;
14    EXCEPTION
15      WHEN SQLSTATE 'UDMAD' THEN
16      RAISE 'Questo indirizzo è principale e non può essere eliminato
        ' USING ERRCODE = 'UDMAD';
17      RETURN NULL;
18    END;$$;
19
20  CREATE OR REPLACE TRIGGER undeletable_main_address
21    BEFORE DELETE ON Indirizzo
22    FOR EACH ROW
23    EXECUTE PROCEDURE undeletable_main_address_f();
```

4.4.7 unchangeable_main_address_description

```
--Vincolo che impedisce di modificare la descrizione degli
    indirizzi principali
CREATE OR REPLACE FUNCTION unchangeable_main_address_description_f
    ()
  RETURNS TRIGGER
  LANGUAGE PLPGSQL
  AS $$
  BEGIN
    IF (OLD.Descrizione='Principale' AND NEW.Descrizione<>'
    Principale') OR (OLD.Descrizione<>'Principale' AND NEW.
    Descrizione='Principale') THEN
      RAISE EXCEPTION USING ERRCODE ='UMADD';
    ELSE
      RETURN NEW;
    END IF;
  EXCEPTION
    WHEN SQLSTATE 'UMADD' THEN
    RAISE 'Tentativo di modificare un indirizzo principale in uno
    secondario, o viceversa, bloccato' USING ERRCODE = 'UMADD';
    RETURN NULL;
  END; $$;

CREATE OR REPLACE TRIGGER unchangeable_main_address_description
  BEFORE UPDATE ON Indirizzo
  FOR EACH ROW
  EXECUTE PROCEDURE unchangeable_main_address_description_f();
```

4.4.8 block_contact_without_both_numbers

```
--Vincolo che verifica se alla fine di una transazione di
    inserimento di un contatto
--questi abbia sempre almeno un numero mobile e uno fisso
CREATE OR REPLACE FUNCTION block_contact_without_both_numbers_f()
  RETURNS TRIGGER
  LANGUAGE PLPGSQL
  AS $$
  BEGIN
    IF (SELECT COUNT(*) FROM telefono WHERE Contatto_FK = NEW.
    Contatto_ID AND descrizione = 'Fisso' ) < 1
      OR (SELECT COUNT(*) FROM telefono WHERE Contatto_FK = NEW.
    Contatto_ID AND descrizione = 'Mobile') < 1
    THEN
      RAISE EXCEPTION USING ERRCODE ='BCWBN';
    ELSE
      RETURN NEW;
    END IF;
  EXCEPTION
    WHEN SQLSTATE 'BCWBN' THEN
    RAISE EXCEPTION 'Si blocca l''inserimento di un contatto di cui
     non è stato dichiarato
            almeno un numero mobile e uno fisso' USING ERRCODE = '
    BCWBN';
    RETURN NULL;
  END; $$;

CREATE CONSTRAINT TRIGGER block_contact_without_both_numbers
  AFTER INSERT ON Contatto
```

```
  DEFERRABLE
  FOR EACH ROW
  EXECUTE PROCEDURE block_contact_without_both_numbers_f();
```

4.4.9 block_contact_without_main_address

```
--Vincolo che verifica se alla fine di una transazione di inserimento di un contatto
--questi abbia sempre un indirizzo principale associato
CREATE OR REPLACE FUNCTION block_contact_without_main_address_f()
  RETURNS TRIGGER
  LANGUAGE PLPGSQL
  AS $$
  BEGIN
    IF (SELECT COUNT(*) FROM Indirizzo WHERE Contatto_FK = NEW.Contatto_ID AND descrizione = 'Principale') < 1
    THEN
      RAISE EXCEPTION USING ERRCODE = 'BCWMA';
      ROLLBACK;
      RETURN NULL;
    ELSE
      RETURN NEW;
    END IF;
  EXCEPTION
    WHEN SQLSTATE 'BCWMA' THEN
    RAISE EXCEPTION 'Si blocca l''inserimento di un contatto di cui non è stato dichiarato
             un indirizzo principale' USING ERRCODE = 'BCWMA';
    RETURN NULL;
  END; $$;

CREATE CONSTRAINT TRIGGER block_contact_without_main_address
  AFTER INSERT ON Contatto
  DEFERRABLE
  FOR EACH ROW
  EXECUTE PROCEDURE block_contact_without_main_address_f();
```

4.4.10 check_mobile_landline_numbers_existence

```
--Vincolo che assicura che nessuna modifica possa lasciare un contatto
--senza almeno un numero mobile e uno fisso
CREATE OR REPLACE FUNCTION check_mobile_landline_numbers_existence_f()
  RETURNS TRIGGER
  LANGUAGE PLPGSQL
  AS $$
  BEGIN
    --Se il numero di telefono interessato è l'ultimo con descrizione
    --"Fisso" o "Mobile" e il contatto esiste, allora l'operazione è bloccata
    IF ((SELECT COUNT(*) FROM Telefono WHERE  Contatto_fk = OLD.Contatto_fk
                                        AND  Descrizione = OLD.Descrizione
```

```
12                                      AND (OLD.Descrizione = 'Mobile
    ' OR OLD.Descrizione = 'Fisso')) = 1)
13          AND ((SELECT COUNT(*) FROM Contatto WHERE Contatto_ID = OLD
    .Contatto_FK ) <> 0) THEN
14        RAISE EXCEPTION USING ERRCODE = 'CMLNE';
15      ELSE
16        --Se l'operazione è UPDATE
17        IF tg_op = 'UPDATE' THEN
18          RETURN NEW;
19        --Se l'operazione è DELETE
20        ELSE
21          RETURN OLD;
22        END IF;
23      END IF;
24    EXCEPTION
25      WHEN SQLSTATE 'CMLNE' THEN
26      --tg_op è una metavariabile che indica l'operazione che innesca
         il trigger
27      RAISE EXCEPTION 'operazione di % del numero % del contatto di
        ID % non consentita',tg_op, OLD.Numero, OLD.Contatto_fk USING
        ERRCODE = 'CMLNE';
28      RETURN NULL;
29    END; $$;
30
31  CREATE OR REPLACE TRIGGER check_mobile_landline_numbers_existence
32    BEFORE DELETE OR UPDATE ON Telefono
33    FOR EACH ROW
34    EXECUTE PROCEDURE check_mobile_landline_numbers_existence_f();
```

4.5 Automazioni

4.5.1 Introduzione

La Base di dati garantisce le seguenti automazioni:

- **automatic_void_groups_deletion** : quando un gruppo resta senza contatti, viene automaticamente cancellato dalla Base di dati;
- **automatic_email_association** : quando viene aggiunta una nuova email, questa viene automaticamente associata a tutti gli account che hanno in comune lo stesso indirizzo di posta elettronica;
- **automatic_account_association** : quando viene aggiunto un nuovo account, questo viene automaticamente associato a tutte le email che hanno in comune lo stesso indirizzo di posta elettronica dell'account.

4.5.2 automatic_void_groups_deletion

```
1  -- Regola attiva che cancella automaticamente quei gruppi che
       restano senza contatti
2  CREATE OR REPLACE FUNCTION automatic_void_groups_deletion_f()
3    RETURNS TRIGGER
4    LANGUAGE PLPGSQL
5    AS $$
```

```
  BEGIN
    IF (SELECT count(*) FROM composizione AS CP WHERE OLD.gruppo_fk
     = CP.gruppo_fk) = 0 THEN
      DELETE FROM Gruppo WHERE Gruppo_ID = OLD.gruppo_fk;
      RAISE NOTICE 'Gruppo di ID % è stato eliminato perché senza
    contatti', OLD.Gruppo_fk;
      RETURN NEW;
    END IF;
  END; $$;

CREATE CONSTRAINT TRIGGER automatic_void_groups_deletion
  AFTER DELETE ON Composizione
  DEFERRABLE
  FOR EACH ROW
  EXECUTE PROCEDURE automatic_void_groups_deletion_f();
```

4.5.3 automatic_email_association

```
--Regola attiva che dopo ogni inserimento di una nuova email
--la associa a tutti i suoi account
CREATE OR REPLACE FUNCTION automatic_email_association_f()
  RETURNS TRIGGER
  LANGUAGE PLPGSQL
  AS $$
  DECLARE
    cerca_account_da_associare CURSOR FOR
      SELECT *
      FROM Account
      WHERE IndirizzoEmail = NEW.IndirizzoEmail;
  BEGIN
    FOR account_da_associare IN cerca_account_da_associare LOOP
      INSERT INTO Associa VALUES
      (NEW.Email_ID, account_da_associare.Fornitore,
    account_da_associare.IndirizzoEmail);
      RAISE NOTICE 'La nuova email è stata associata all''account
    di indirizzo "%" con fornitore "%"', NEW.indirizzoEmail,
    account_da_associare.Fornitore;
    END LOOP;
    RETURN NEW;
  END; $$;

CREATE OR REPLACE TRIGGER automatic_email_association
  AFTER INSERT ON Email
  FOR EACH ROW
  EXECUTE PROCEDURE automatic_email_association_f();
```

4.5.4 automatic_account_association

```
--Regola attiva che dopo ogni inserimento di un nuovo account
--il suo indirizzo elettronico è associato alle email che lo
    condividono
CREATE OR REPLACE FUNCTION automatic_account_association_f()
  RETURNS TRIGGER
  LANGUAGE PLPGSQL
  AS $$
  DECLARE
    cerca_email_da_associare CURSOR FOR
```

```
9       SELECT *
10      FROM Email
11      WHERE IndirizzoEmail = NEW.IndirizzoEmail;
12    BEGIN
13      FOR email_da_associare IN cerca_email_da_associare LOOP
14        INSERT INTO Associa VALUES
15        (email_da_associare.Email_ID, NEW.Fornitore, NEW.
      IndirizzoEmail);
16        RAISE NOTICE 'Il nuovo account di fornitore "%" è stato
      associato all''email di indirizzo "%"', NEW.Fornitore,
      email_da_associare.indirizzoEmail;
17      END LOOP;
18      RETURN NEW;
19    END; $$;
20
21  CREATE OR REPLACE TRIGGER automatic_account_association
22    AFTER INSERT ON Account
23    FOR EACH ROW
24    EXECUTE PROCEDURE automatic_account_association_f();
```

4.6 Funzioni

4.6.1 Introduzione

La Base di dati fornisce le seguenti funzioni:

- **generate_new_contatto_id** : genera un ID valido per un nuovo contatto;
- **generate_new_gruppo_id** : genera un ID valido per un nuovo gruppo;
- **coherent_insertion_f** : garantisce l'inserimento di un nuovo contatto con tutte le informazioni essenziali (come anche un numero mobile, un numero fisso e un indirizzo principale) in una rubrica;
- **reindirizza** : dato un contatto e un numero chiamato, restituisce un altro numero del contatto a cui chiamare in caso di mancata risposta;
- **cerca** : dato un criterio (per nome, numero, email o nickname di account) e un testo di ricerca, restituisce una tabella di ID di contatti che soddisfano la ricerca per sottostringa;
- **cerca_per_nome** : dato un testo di ricerca, restituisce una tabella di contatti i cui nomi soddisfano la ricerca;
- **cerca_per_numero** : dato un testo di ricerca, restituisce una tabella di telefoni e ID contatto corrispondenti i cui numeri soddisfano la ricerca;
- **cerca_per_email** : dato un testo di ricerca, restituisce una tabella di email e ID contatto corrispondenti i cui indirizzi di posta elettronica soddisfano la ricerca;
- **cerca_per_account** : dato un testo di ricerca, restituisce una tabella di account e ID contatto corrispondenti i cui nickname soddisfano la ricerca;

4.6.2 generate_new_contatto_id

```
--Funzione che genera un id valido per un contatto
CREATE OR REPLACE FUNCTION generate_new_contatto_id()
  RETURNS INTEGER
  LANGUAGE PLPGSQL
  AS $$
  DECLARE
    --Seleziono e conservo un nuovo identificativo per il contatto
    codice_contatto     INTEGER;
    nuovo_codice_valido INTEGER := (SELECT max(contatto_id) FROM
    Contatto) + 1;
  BEGIN
    IF (nuovo_codice_valido <> -1) THEN
      codice_contatto := nuovo_codice_valido;
      RAISE NOTICE 'Si inserisce il nuovo max id';
    ELSE
      codice_contatto := 1;
      RAISE NOTICE 'Si inserisce 1';
    END IF;
    RETURN codice_contatto;
  END; $$;
```

4.6.3 generate_new_gruppo_id

```
--Funzione che genera un id valido per un gruppo
CREATE OR REPLACE FUNCTION generate_new_gruppo_id()
  RETURNS INTEGER
  LANGUAGE PLPGSQL
  AS $$
  DECLARE
    --Seleziono e conservo un nuovo identificativo per il gruppo
    codice_gruppo     INTEGER;
    nuovo_codice_valido INTEGER := (SELECT max(gruppo_id) FROM
    Gruppo) + 1;
  BEGIN
    IF (nuovo_codice_valido <> -1) THEN
      codice_gruppo := nuovo_codice_valido;
      RAISE NOTICE 'Si inserisce il nuovo max id';
    ELSE
      codice_gruppo := 1;
      RAISE NOTICE 'Si inserisce 1';
    END IF;
    RETURN codice_gruppo;
  END; $$;
```

4.6.4 coherent_insertion_f

```
--Funzione che garantisce il corretto inserimento di un contatto
    con le informazioni principali:
--un numero fisso, uno mobile e un indirizzo fisico principale.
--coherent_insertion_f(utente_rubrica, nome_contatto,
    secondonome_contatto, cognome_contatto,
--                     num_mobile, num_fisso, via, citta, nazione,
    cap, indirizzo_email, descrizione_email, contatto_id)
CREATE OR REPLACE PROCEDURE coherent_insertion_f(rubrica_par
    Rubrica.utente_id%TYPE,     nome_par Contatto.nome%TYPE,
```

```
6                       sec_no_par Contatto.secondonome%TYPE,
    cognome_par Contatto.cognome%TYPE,
7                       numero_mobile_par Telefono.numero%TYPE,
    numero_fisso_par Telefono.numero%TYPE,
8                       via_par Indirizzo.via%TYPE,
    citta_par Indirizzo.città%TYPE,
9                       nazione_par Indirizzo.nazione%TYPE,
    cap_par Indirizzo.cap%TYPE,
10                      codice_contatto INTEGER)
11  --RETURNS INTEGER
12  LANGUAGE PLPGSQL
13  AS $$
14  BEGIN
15    SET CONSTRAINTS ALL DEFERRED;
16    INSERT INTO Contatto (contatto_id, nome, secondonome, cognome,
    rubrica_fk)
17                  VALUES (codice_contatto, nome_par, NULLIF(
    sec_no_par,''), cognome_par, rubrica_par);
18    INSERT INTO Telefono (numero, descrizione, contatto_fk)
19                  VALUES (numero_mobile_par, 'Mobile',
    codice_contatto);
20    INSERT INTO Telefono (numero, descrizione, contatto_fk)
21                  VALUES (numero_fisso_par, 'Fisso',
    codice_contatto);
22    INSERT INTO Indirizzo (via, descrizione, città, nazione, cap,
    contatto_fk)
23              VALUES (via_par, 'Principale', citta_par, nazione_par
    , cap_par, codice_contatto);
24  END; $$;
```

4.6.5 reindirizza

```
1   --Nel caso del fallimento di una chiamata a un certo contatto, la
      funzione ritorna un numero alternativo dello stesso contatto a
      cui reindirizzare la chiamata (mobile->fisso; fisso->mobile;
      non fisso e non mobile->fisso o mobile)
2   CREATE OR REPLACE FUNCTION reindirizza(contatto_chiamato Contatto.
      contatto_id%TYPE, numero_chiamato Telefono.numero%TYPE)
3     RETURNS Telefono.numero%TYPE
4     LANGUAGE PLPGSQL
5     AS $$
6     DECLARE
7       descrizione_num_chiamato Telefono.descrizione%TYPE;
8       numero_reindirizzato Telefono.numero%TYPE;
9     BEGIN
10      --Si cerca la descrizione del numero chiamato
11      SELECT descrizione INTO descrizione_num_chiamato
12      FROM Telefono
13      WHERE numero = trim(numero_chiamato) AND
14          contatto_chiamato = contatto_fk;
15      --Se il numero chiamato non esiste tra quelli del contatto, si
      ritorna NULL
16      IF descrizione_num_chiamato IS NULL THEN
17        numero_reindirizzato := NULL;
18        RAISE NOTICE 'Il numero chiamato non è presente tra quelli
      del contatto di ID %', contatto_chiamato;
19      ELSE
```

```
20      --Se il numero chiamato è fisso, ritorna il primo numero
    mobile del contatto
21      IF     descrizione_num_chiamato = 'Fisso'  THEN
22        SELECT numero INTO numero_reindirizzato
23        FROM   Telefono
24        WHERE  descrizione = 'Mobile' AND
25             contatto_fk = contatto_chiamato
26        LIMIT 1;
27      --Se il numero chiamato è mobile, ritorna il primo numero
    fisso del contatto
28      ELSEIF descrizione_num_chiamato = 'Mobile' THEN
29        SELECT numero INTO numero_reindirizzato
30        FROM   Telefono
31        WHERE  descrizione = 'Fisso' AND
32             contatto_fk = contatto_chiamato
33        LIMIT 1;
34      --Se il numero chiamato non è mobile o fisso, ritorna il
    primo numero di uno dei due tipi del contatto
35      ELSE
36        SELECT numero INTO numero_reindirizzato
37        FROM   Telefono
38        WHERE  (descrizione = 'Mobile' OR descrizione = 'Fisso')
    AND
39             contatto_fk = contatto_chiamato
40        LIMIT 1;
41      END IF;
42    END IF;
43    RETURN numero_reindirizzato;
44  END; $$;
```

4.6.6 cerca

```
1  --Funzione che generalizza la ricerca dei contatti secondo un
       criterio indicato (nome, numero, email o account) e restituisce
        una tabella di codici di contatti che soddisfano il match
2  CREATE OR REPLACE FUNCTION cerca(criterio_ricerca VARCHAR, rubrica
       Rubrica.utente_id%TYPE, testo_ricerca VARCHAR)
3    RETURNS TABLE(
4          contatto_id Contatto.contatto_id%TYPE)
5    LANGUAGE PLPGSQL
6    AS $$
7    BEGIN
8      IF     criterio_ricerca = 'Nome'    THEN
9        RETURN QUERY
10         SELECT CNO.contatto_id
11         FROM cerca_per_nome(rubrica, testo_ricerca) AS CNO;
12     ELSEIF criterio_ricerca = 'Numero'  THEN
13       RETURN QUERY
14         SELECT CNU.contatto_id
15         FROM cerca_per_numero(rubrica, testo_ricerca) AS CNU;
16     ELSEIF criterio_ricerca = 'Email'   THEN
17       RETURN QUERY
18         SELECT CEM.contatto_id
19         FROM cerca_per_email(rubrica, testo_ricerca) AS CEM;
20     ELSEIF criterio_ricerca = 'Account' THEN
21       RETURN QUERY
22         SELECT CAC.contatto_id
```

```
        FROM cerca_per_account(rubrica, testo_ricerca) AS CAC;
    END IF;
  END; $$;
```

4.6.7 cerca_per_nome

```
--Funzione che cerca i contatti per nome e restituisce una tabella di contatti
CREATE OR REPLACE FUNCTION cerca_per_nome(rubrica Rubrica.utente_id%TYPE, nome_cercato Contatto.nome%TYPE)
  RETURNS TABLE(
        contatto_id Contatto.contatto_id%TYPE,
        nome Contatto.nome%TYPE,
        secondonome Contatto.secondonome%TYPE,
        cognome Contatto.cognome%TYPE,
        foto Contatto.foto%TYPE,
        rubrica_fk Contatto.rubrica_fk%TYPE)
  LANGUAGE PLPGSQL
  AS $$
  BEGIN
    RETURN QUERY
      SELECT C1.contatto_id, C1.nome, C1.secondonome, C1.cognome, C1.foto, C1.rubrica_fk
      FROM Contatto AS C1
      WHERE rubrica = C1.rubrica_fk AND (position(lower(nome_cercato) in lower(C1.nome)) <> 0);
  END; $$;
```

4.6.8 cerca_per_numero

```
--Funzione che cerca i contatti per numero di telefono e restituisce una tabella di numeri di telefono
CREATE OR REPLACE FUNCTION cerca_per_numero(rubrica Rubrica.utente_id%TYPE, numero_cercato Telefono.numero%TYPE)
  RETURNS TABLE(
        contatto_id Telefono.contatto_fk%TYPE,
        telefono_id Telefono.telefono_id%TYPE,
        numero Telefono.numero%TYPE,
        descrizione Telefono.descrizione%TYPE
        )
  LANGUAGE PLPGSQL
  AS $$
  BEGIN
    RETURN QUERY
      SELECT T1.contatto_fk, T1.telefono_id, T1.numero, T1.descrizione
      FROM Telefono AS T1, Contatto AS C1
      WHERE rubrica = C1.rubrica_fk AND (position(lower(numero_cercato) in lower(T1.numero)) <> 0) AND
            C1.contatto_id = T1.contatto_fk;
  END; $$;
```

4.6.9 cerca_per_email

```
--Funzione che cerca i contatti per indirizzo email e restituisce una tabella di indirizzi email
```

```
CREATE OR REPLACE FUNCTION cerca_per_email(rubrica Rubrica.
    utente_id%TYPE, email_cercata VARCHAR)
  RETURNS TABLE(
        contatto_id Email.contatto_fk%TYPE,
        email_id Email.email_id%TYPE,
        indirizzoEmail Email.indirizzoEmail%TYPE,
        descrizione Email.descrizione%TYPE)
  LANGUAGE PLPGSQL
  AS $$
  BEGIN
    RETURN QUERY
      SELECT E1.contatto_fk, E1.email_id, E1.indirizzoEmail, E1.
    descrizione
      FROM Email AS E1, Contatto AS C1
      WHERE rubrica = C1.rubrica_fk AND (position(lower(
    email_cercata) in lower(E1.indirizzoEmail)) <> 0) AND
          C1.contatto_id = E1.contatto_fk;
  END; $$;
```

4.6.10 cerca_per_account

```
--Funzione che cerca i contatti per nickname dell'account e
    restituisce una tabella di nickname di account associati a
    rispetivi contatti
CREATE OR REPLACE FUNCTION cerca_per_account(rubrica Rubrica.
    utente_id%TYPE, nickname_cercato Account.nickname%TYPE)
  RETURNS TABLE(
        contatto_id Contatto.contatto_id%TYPE,
        nickname Account.nickname%TYPE,
        fornitore Account.fornitore%TYPE,
        indirizzoEmail Account.indirizzoEmail%TYPE,
        frasestato Account.frasestato%TYPE)
  LANGUAGE PLPGSQL
  AS $$
  BEGIN
    RETURN QUERY
      SELECT E1.contatto_fk, A1.nickname, A1.fornitore, A1.
    indirizzoEmail, A1.frasestato
      FROM Account AS A1, Associa AS AS1, Email AS E1, Contatto AS
    C1
      WHERE rubrica = C1.rubrica_fk AND (position(lower(
    nickname_cercato) in lower(A1.nickname)) <> 0) AND
          C1.contatto_id = E1.contatto_fk AND E1.email_id = AS1.
    email_fk AND AS1.FornitoreAccount = A1.Fornitore AND
          AS1.indirizzoEmailAccount = A1.indirizzoEmail;
  END; $$;
```

4.7 Codice SQL e Popolamento

Per visionare il file SQL cliccare questo link.
Per visionare li popolamento cliccare questo link.

Capitolo 5

Manuale d'uso per l'Applicativo

5.1 Installazione

Per usare l'applicativo, si suggerisce di seguire i seguenti passaggi:

1. importare il progetto da GitHub seguendo questo link;

2. se non presente, scaricare PostgreSQL;

3. creare un Database di nome **Rubrica** con owner **postgres** e password **1234** , oppure modificare gli attributi *nomeutente*, *password* e *url* della classe ConnessioneDatabase con i propri valori;

4. eseguire questa query per costuire lo schema del Database;

5. eseguire questa query per popolare il Database.

5.2 Esempi di funzionalità notevoli

Si consiglia di provare le seguenti funzionalità:

- per il reindirizzamento della chiamata si prema il pulsante "Visualizza contatto" presso la lista di contatti per aprire la scheda contatto da cui poi si può chiamare il numero fisso o il numero mobile (l'uno reindirizza sempre l'altro);

- per verificare l'associazione degli account a un contatto tramite email si provi ad aggiungere a un contatto uno dei seguenti indirizzi di posta elettronica: **altra@gmail.com**, **a@gmail.com** o **test@libero.com**;

- per la ricerca di contatti secondo un criterio, accedere alla lista di contatti di una rubrica, selezionare il criterio di ricerca presso la combobox collocata in alto e nell'adiacente barra di ricerca digitare la parola chiave (può anche non essere esatta perché il match avviene per sottostringa).