



UNIVERSITÀ DI FEDERICO II

BASI DI DATI

Progettazione e sviluppo di una base di dati relazionale per una rubrica telefonica avanzata

Authors

Alessandro MAURIELLO	N86003603
Giovanni ZAMPETTI	N86003787
Antonio TODISCO	N86003642

Febbraio 2022

Questa pagina è stata lasciata intenzionalmente bianca.

Indice

1	Introduzione	4
1.1	Analisi del problema	4
1.2	Analisi dei requisiti	4
2	Progettazione Concettuale	6
2.1	Introduzione	6
2.2	Class Diagram	6
3	Ristrutturazione del class diagram	7
3.1	Analisi delle ridondanze	7
3.2	Eliminazione delle generalizzazioni	7
3.3	Eliminazione degli attributi multivalore	7
3.4	Eliminazione degli attributi composti	7
3.5	Scelta degli identificatori primari	7
3.6	Class diagram ristrutturato	8
3.7	Dizionario delle classi	9
3.8	Dizionario delle associazioni	10
3.9	Dizionario dei vincoli	11
4	Progettazione Logica	12
4.1	Schema Logico	12
5	Progettazione Fisica	13
5.1	Creazione dei domini	13
5.1.1	Valid email	13
5.1.2	Valid name	13
5.1.3	Valid Lastname	13
5.2	Creazione Tabelle	13
5.2.1	Contact	13
5.2.2	Messaging Account	14
5.2.3	Address	14
5.2.4	Landline	15
5.2.5	Mobile	15
5.2.6	Group	16

5.2.7	Contact Group	16
5.2.8	E-mail	16
5.3	Funzioni e trigger	17
5.3.1	Check existing email	17
5.3.2	Check duplicate email	17
5.3.3	Check duplicate account mail	18
5.3.4	Check primary address	18
5.3.5	Create contact	19
5.3.6	Insert Email	19
5.3.7	Insert account	19
5.3.8	Insert Contact group	20
5.3.9	Insert address	21
5.3.10	insert mobile	21
5.3.11	insert landline	22
5.3.12	Modify contact	22
5.3.13	Modify contact address	23
5.3.14	Modify contact email	23
5.3.15	Modify contact mobile	24
5.3.16	Modify contact landline	24
5.3.17	Modify contact account	25
5.3.18	Delete contact	25
5.3.19	Filter First Name	26
5.3.20	Filter email	26
5.3.21	Filter mobile	27
5.3.22	Filter landline	27
5.3.23	View private contact	27
5.3.24	Popolazione	28

Capitolo 1

Introduzione

1.1 Analisi del problema

Si progetterà una base una base di dati relazionale per la gestione di una rubrica telefonica avanzata. La rubrica sarà in grado di memorizzare dati riguardanti contatti, dove ogni contatto sarà individuato da un nome ed un cognome e può eventualmente appartenere ad un gruppo. Andremo inoltre a memorizzare per ogni contatto numeri di telefono (fissi o mobili), indirizzi fisici primari e secondari, eventuali indirizzi di posta elettronica e account a sistemi di messaging. Infine andremo a definire il tipo di un contatto, se pubblico o privato.

1.2 Analisi dei requisiti

Abbiamo identificato i seguenti requisiti per la base di dati :

- Un'entità CONTACT che ci permette di memorizzare il nome, il cognome, la foto e il tipo, se pubblico o privato del contatto, e un id per indentificarlo univocamente;
- Un'entità ADDRESS che ci permette di individuare l'indirizzo fisico del contatto, avente i seguenti attributi : via, città, codice postale, nazione e il tipo se primario o secondario;
- Un'entità MESSAGING ACCOUNT che si specializzerà nel mantenere dati riguardanti account di messaggistica, (ad esempio whatsapp, telegram, ecc.), che ha come attributi un id dell'account, il nome del fornitore, il nickname, la bio e la mail;
- Un'entità GROUP che permette di dividere la rubrica in sottogruppi, aventi un nome e una descrizione;
- Un'entità E-MAIL che sarà caratterizzata dal solo attributo e-mail per mantenere un'insieme (eventualmente vuoto) di mail riguardanti un contatto;

- Un'entità TELEPHONE NUMBER che ci permetterà di memorizzare numeri mobili o fissi, caratterizzata da due attributi quali prefix e number.

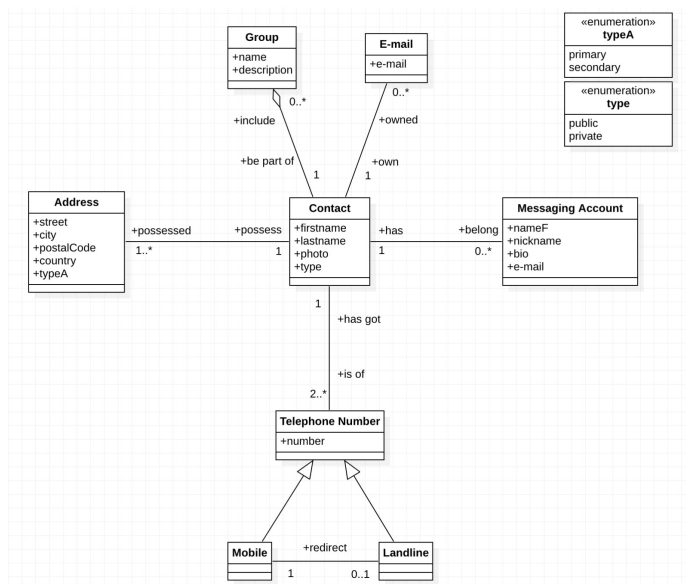
Capitolo 2

Progettazione Concettuale

2.1 Introduzione

Per la progettazione di questa base di dati abbiamo individuato come classe principale CONTACT, che è in relazione con tutte le altre entità. In particolare avremo un'aggregazione tra CONTACT e GROUP, per dire che non esistono dei gruppi senza contatti. E' stato poi scelto di implementare MOBILE e LANDLINE come generalizzazione di una classe TELEPHONE NUMBER.

2.2 Class Diagram



Capitolo 3

Ristrutturazione del class diagram

3.1 Analisi delle ridondanze

Non sono presenti ridondanze nel class diagram.

3.2 Eliminazione delle generalizzazioni

Nel class diagram è presente una generalizzazione di Telephone Number, che verrà rimossa nel class diagram ristrutturato dove Mobile e Landline diventeranno due classi separate.

3.3 Eliminazione degli attributi multivalore

Non sono presenti attributi multivalore nel class diagram.

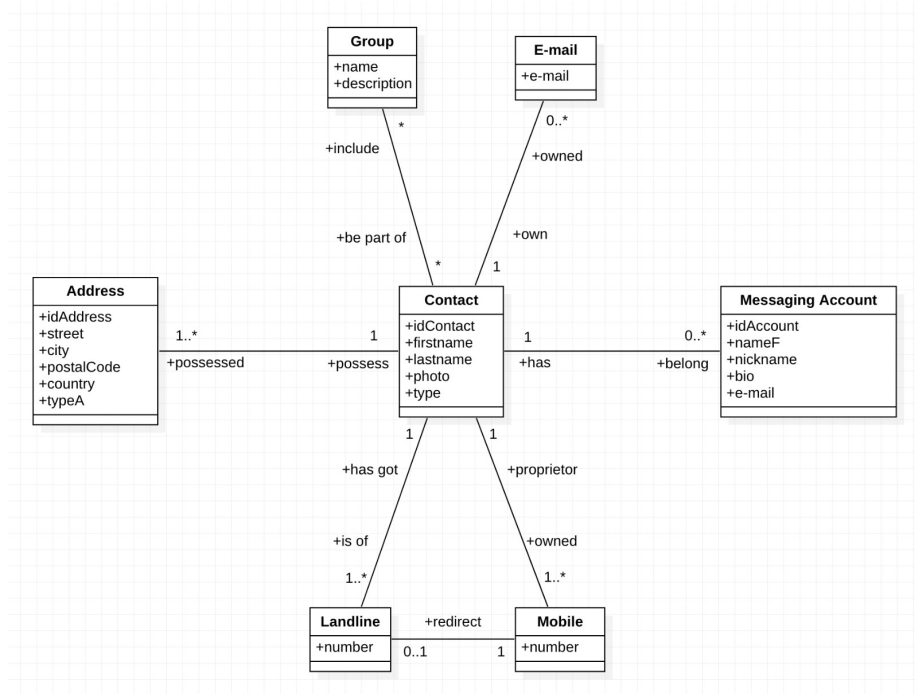
3.4 Eliminazione degli attributi composti

Non sono presenti attributi composti nel class diagram.

3.5 Scelta degli identificatori primari

Per quanto riguarda le classi CONTACT, ADDRESS e MESSAGING ACCOUNT si è scelto di usare rispettivamente una chiave idContact, idAddress, idAccount per non utilizzare chiavi primarie composte. Per GROUP è stato scelto di utilizzare name come chiave primaria. Per quanto riguarda LANDLINE e MOBILE, si è scelto di utilizzare due chiavi primarie composte, formate da number e idContact.

3.6 Class diagram ristrutturato



3.7 Dizionario delle classi

Viene presentato qui il dizionario delle classi, che contiene il nome di ogni classe, i rispettivi attributi, ed il tipo, con una breve descrizione.

NOME CLASSE	DESCRIZIONE
Contact	idContact (<i>Serial</i>) : l'id del contatto firstName (<i>String</i>) : Il nome del contatto lastName (<i>String</i>) : Il cognome del contatto photo (<i>String</i>) : Percorso della foto del contatto (opzionale) type (<i>String</i>) : Il tipo del contatto (public o private)
Messaging Account	idAccount (<i>Serial</i>) : L'id dell'account nameF (<i>String</i>) : Il nome del fornitore nickname (<i>String</i>) : Il nickname dell'account bio (<i>String</i>) : Lo stato dell'account e-mail (<i>String</i>) : La mail dell'account
Address	street (<i>String</i>) : La via dell'indirizzo city (<i>String</i>) : La città dell'indirizzo postalCode (<i>Int</i>) : Il codice postale dell'indirizzo country (<i>String</i>) : La nazione dell'indirizzo typeA (<i>String</i>) : Il tipo dell'indirizzo (primary or secondary)
Group	name (<i>String</i>) : Il nome del gruppo description (<i>String</i>) : la descrizione del gruppo
Mobile	number (<i>String</i>) : Il numero del mobile
Landline	number (<i>String</i>) : Il numero del landline
E-mail	e-mail (<i>String</i>) : La mail del contatto

3.8 Dizionario delle associazioni

Viene qui presentato il dizionario delle associazioni, che contiene tutte le associazioni, i rispettivi ruoli, ed una breve descrizione dell'associazione.

NOME ASSOCIAZIONE	DESCRIZIONE
Contact Account	Contact [1] <i>ruolo</i> has : Rappresenta il contatto del messaging account Messaging Account [0...*] <i>ruolo</i> belong : Rappresenta l'account del contatto
Contact Address	Contact [1] <i>ruolo</i> possess : Rappresenta il contatto dell'indirizzo. Address [1...*] <i>ruolo</i> possessed : Rappresenta l'indirizzo del contatto.
Contact Group	Contact [*] <i>ruolo</i> be part of : Rappresenta il contatto nel gruppo. Group [*] <i>ruolo</i> include : Rappresenta il gruppo del contatto.
Contact e-mail	Contact [1] <i>ruolo</i> own : Rappresenta il contatto della mail. E-mail [0...*] <i>ruolo</i> owned : Rappresenta la mail del contatto.
Contact Mobile	Contact [1] <i>ruolo</i> proprietor : Rappresenta il contatto del mobile. Mobile [1...*] <i>ruolo</i> owned : Rappresenta il mobile del contatto.
Contact Landline	Contact [1] <i>ruolo</i> has got : Rappresenta il contatto del landline. Landline [1...*] <i>ruolo</i> is of : Rappresenta il landline del contatto.

3.9 Dizionario dei vincoli

Vengono qui proposti i vincoli che saranno presenti nel database, con la rispettiva descrizione.

NOME VINCOLO	DESCRIZIONE
Unique Email	Non devono esistere più contatti con la stessa mail.
Valid Email	La mail deve rispettare la forma x@y.z dove x y e z sono stringhe non nulle.
Valid MobileNumber	Il mobile number ha lunghezza compresa tra cinque e dieci.
Valid LandlineNumber	Il landline ha lunghezza compresa tra cinque e dieci.
Valid postalCode	Il postalCode rispetta la forma xxxxx.
Valid FirstName	La lunghezza del nome del contatto deve essere compreso tra zero e venti e non deve contenere numeri.
Valid LastName	La lunghezza del cognome del contatto deve essere compreso tra zero e venti e non devono contenere numeri.
Valid GroupName	La lunghezza del nome del gruppo deve essere compresa tra zero e trenta.
Check Duplicate Contact	I contatti non devono essere duplicati.

Capitolo 4

Progettazione Logica

4.1 Schema Logico

In questo schema relazionale le chiavi primarie sono state scritte in **grassetto**, mentre le chiavi esterne vengono scritte in *corsivo*.

CONTACT	(idContact , firstName, lastName, photo, type)
E-MAIL	(e-mail , <i>idContact</i>) <i>idContact</i> \mapsto CONTACT.idContact
GROUP	(name ,description)
CONTACT_GROUP	(name , <i>idContact</i>) <i>name</i> \mapsto GROUP.name <i>idContact</i> \mapsto CONTACT.idContact
ADDRESS	(idAddress , <i>idContact</i> , street, city, PostalCode, Country, typeA) <i>idContact</i> \mapsto CONTACT.idContact
MESSAGING_ACCOUNT	(idAccount , <i>idContact</i> , NameF, nickname, bio, e-mail) <i>idContact</i> \mapsto CONTACT.idContact <i>e-mail</i> \mapsto E-MAIL.e-mail
LANDLINE	(Number , <i>idContact</i> , <i>Mobile</i>) <i>idContact</i> \mapsto CONTACT.idContact <i>Mobile</i> \mapsto MOBILE.Number
MOBILE	(Number , <i>idContact</i> , <i>Landline</i>) <i>idContact</i> \mapsto CONTACT.idContact <i>Landline</i> \mapsto LANDLINE.Number

Capitolo 5

Progettazione Fisica

5.1 Creazione dei domini

5.1.1 Valid email

```
1 CREATE DOMAIN E_MAIL AS VARCHAR(42)
2 CHECK ( VALUE LIKE '_%@_%.%' );
```

5.1.2 Valid name

```
1 CREATE DOMAIN FIRST_NAME AS VARCHAR(20)
2 CHECK ( VALUE <> '' AND VALUE NOT SIMILAR TO '%[0-9]++' );
```

5.1.3 Valid Lastname

```
1 CREATE DOMAIN LAST_NAME AS VARCHAR(20)
2 CHECK ( VALUE <> '' AND VALUE NOT SIMILAR TO '%[0-9]++' );
```

5.2 Creazione Tabelle

5.2.1 Contact

```
1 CREATE TABLE CONTACT(
2   idContact SERIAL NOT NULL,
3   First_Name FIRST_NAME NOT NULL,
4   Last_Name LAST_NAME NOT NULL,
5   Photo Varchar(30),
6   Type VARCHAR(10) NOT NULL
7 );
8
9 ALTER TABLE CONTACT
10 -- Aggiunta del vincolo di chiave primaria
11 ADD CONSTRAINT Contact_pk PRIMARY KEY(idContact);
12
13 -- Modifica dell'inizio del serial
14 ALTER SEQUENCE contact_idContact_seq RESTART WITH 1 INCREMENT BY
   1;
```

5.2.2 Messaging Account

```
1 CREATE TABLE MESSAGING_ACCOUNT(  
2   idAccount SERIAL NOT NULL,  
3   idContact SERIAL NOT NULL,  
4   NameF VARCHAR(30) NOT NULL,  
5   Nickname VARCHAR(20) NOT NULL,  
6   Bio Varchar(40),  
7   Email E_MAIL NOT NULL  
8 );  
9 ALTER TABLE MESSAGING_ACCOUNT  
10 -- Aggiunta del vincolo di chiave primaria  
11   ADD CONSTRAINT msg_pk PRIMARY KEY(idAccount),  
12 -- Aggiunta del vincolo di chiave esterna sulla tabella CONTACT  
13   ADD CONSTRAINT msg_fk FOREIGN KEY(idContact) REFERENCES CONTACT(  
14     idContact)  
15     ON UPDATE CASCADE,  
16     ON DELETE CASCADE,  
17 -- Aggiunta del vincolo di chiave esterna sulla tabella EMAIL  
18   ADD CONSTRAINT msg_email_fk FOREIGN KEY(Email) REFERENCES EMAIL(  
19     Email)  
20     ON UPDATE CASCADE  
21     ON DELETE RESTRICT;
```

5.2.3 Address

```
1 CREATE TABLE ADDRESS(  
2   idAddress SERIAL NOT NULL,  
3   idContact SERIAL NOT NULL,  
4   Street VARCHAR(40) NOT NULL,  
5   City VARCHAR(40) NOT NULL,  
6   Postal_Code Integer NOT NULL,  
7   Country VARCHAR(15) NOT NULL,  
8   TypeA VARCHAR(15) NOT NULL  
9 );  
10  
11 ALTER TABLE ADDRESS  
12 -- Aggiunta del vincolo di chiave primaria  
13   ADD CONSTRAINT address_pk PRIMARY KEY(idAddress),  
14 -- Aggiunta del vincolo di chiave esterna sulla tabella CONTACT  
15   ADD CONSTRAINT address_fk FOREIGN KEY(idContact) REFERENCES  
16     CONTACT(idContact)  
17     ON UPDATE CASCADE  
18     ON DELETE CASCADE;
```

5.2.4 Landline

```
1 CREATE TABLE LANDLINE(  
2   Number VARCHAR(15) NOT NULL,  
3   idContact SERIAL NOT NULL,  
4   Mobile VARCHAR(15) NOT NULL,  
5  
6   PRIMARY KEY(Number,idContact)  
7 );  
8  
9 ALTER TABLE LANDLINE  
10 -- Aggiunta del vincolo di chiave esterna sulla tabella CONTACT  
11 ADD CONSTRAINT landline_fk FOREIGN KEY(idContact) REFERENCES  
12   CONTACT(idContact)  
13   ON UPDATE CASCADE  
14   ON DELETE CASCADE,  
15 -- Aggiunta del vincolo di chiave esterna sulla tabella MOBILE  
16 ADD CONSTRAINT landline_mobile_fk FOREIGN KEY(idContact,Mobile)  
17 REFERENCES MOBILE(idContact,Number)  
18   ON UPDATE CASCADE  
19   ON DELETE CASCADE;
```

5.2.5 Mobile

```
1 CREATE TABLE MOBILE(  
2   Number VARCHAR(15) NOT NULL,  
3   idContact SERIAL NOT NULL,  
4   Landline VARCHAR(15),  
5  
6   PRIMARY KEY(Number,idContact)  
7 );  
8 ALTER TABLE MOBILE  
9 -- Aggiunta del vincolo di chiave esterna sulla tabella CONTACT  
10 ADD CONSTRAINT mobile_fk FOREIGN KEY(idContact) REFERENCES  
11   CONTACT(idContact)  
12   ON UPDATE CASCADE  
13   ON DELETE CASCADE,  
14 -- Aggiunta del vincolo di chiave esterna sulla tabella LANDLINE  
15 ADD CONSTRAINT mobile_landline_fk FOREIGN KEY(idContact,Landline)  
16 REFERENCES LANDLINE(idContact,Number)  
17   ON UPDATE CASCADE  
18   ON DELETE CASCADE;
```


5.2.6 Group

```
1 CREATE TABLE GROUPS(  
2     name VARCHAR(15) NOT NULL,  
3     description VARCHAR(50)  
4 );  
5  
6 ALTER TABLE GROUPS  
7 -- Aggiunta del vincolo di chiave primaria  
8     ADD CONSTRAINT groups_pk PRIMARY KEY(name);  
9  
10 -- *****
```

5.2.7 Contact Group

```
1     name VARCHAR(15) NOT NULL,  
2     idContact SERIAL NOT NULL  
3 );  
4  
5 ALTER TABLE CONTACT_GROUP  
6 -- Aggiunta del vincolo di chiave primaria  
7     ADD CONSTRAINT group_pk PRIMARY KEY(name,idContact),  
8 -- Aggiunta del vincolo di chiave esterna sulla tabella CONTACT  
9     ADD CONSTRAINT group_fk FOREIGN KEY(idContact) REFERENCES CONTACT  
10         (idContact)  
11         ON UPDATE CASCADE  
12         ON DELETE CASCADE;
```

5.2.8 E-mail

```
1 CREATE TABLE EMAIL(  
2     Email E_MAIL NOT NULL,  
3     idContact SERIAL NOT NULL  
4 );  
5  
6 ALTER TABLE EMAIL  
7 -- Aggiunta del vincolo di chiave primaria  
8     ADD CONSTRAINT email_pk PRIMARY KEY(Email),  
9 -- Aggiunta del vincolo di chiave esterna sulla tabella CONTACT  
10     ADD CONSTRAINT email_fk FOREIGN KEY(idContact) REFERENCES CONTACT  
11         (idContact)  
12         ON UPDATE CASCADE  
13         ON DELETE CASCADE;
```

5.3 Funzioni e trigger

5.3.1 Check existing email

```
1  LANGUAGE 'plpgsql'
2  AS $$
3  BEGIN
4      IF NOT EXISTS (SELECT * FROM email where new.email=email) THEN
5          RAISE NOTICE 'l email del fornitore non presente tra gli
6              indirizzi mail del contatto';
7          return null;
8          END IF;
9          return new;
10 END;
11 $$;
12
13 CREATE TRIGGER check_existing_email BEFORE INSERT ON
14     Messaging_account
15     FOR EACH ROW
16     EXECUTE PROCEDURE Func_CEE();
17
18 --- check_duplicate_email: controllo l'esistenza di un altro
19     contatto duplicato (il contatto sar duplicato quando avr la
20     stessa email)
```

5.3.2 Check duplicate email

```
1  LANGUAGE 'plpgsql'
2  AS $$
3  BEGIN
4      IF EXISTS (SELECT * FROM email where new.email=email and
5          idcontact<>new.idcontact) THEN RAISE NOTICE 'l email gia
6          stata utilizzata da un altro contatto usare la funzione per
7          eliminare il contatto o modificarlo';
8          RETURN NULL;
9          END IF;
10         return new;
11 END;
12 $$;
13
14 CREATE TRIGGER check_duplicate_email BEFORE INSERT ON email
15     FOR EACH ROW
16     EXECUTE PROCEDURE func_CDE();
17
18 --- check_duplicate_account_mail: controllo l'esistenza di un altro
19     account di un altro contatto con la stessa mail
```

5.3.3 Check duplicate account mail

```
1  LANGUAGE 'plpgsql'
2  AS $$
3  BEGIN
4      IF EXISTS (SELECT * FROM messaging_account where new.email=
5                  email and idcontact<>new.idcontact) THEN RAISE NOTICE 'l email
6                  gia stata utilizzata da un altro account di un altro
7                  contatto, usare la funzione per eliminare l account o
8                  modificarlo';
9      RETURN NULL;
10     END IF;
11     return new;
12 END;
13 $$;
14
15 CREATE TRIGGER check_duplicate_account_mail BEFORE INSERT ON
16     messaging_account
17 FOR EACH ROW
18 EXECUTE PROCEDURE func_CDM();
```

5.3.4 Check primary address

```
1  LANGUAGE 'plpgsql'
2  AS $$
3  BEGIN
4      IF (SELECT typeA FROM address where new.idContact = address.
5            idContact ORDER BY TypeA LIMIT 1) <> 'primary' THEN
6      RAISE NOTICE 'non presente un indirizzo principale associato a
7      questo contatto';
8      RETURN NULL;
9      END IF;
10     return new;
11 END;
12 $$;
13
14 CREATE TRIGGER check_primary_address BEFORE INSERT ON ADDRESS
15 FOR EACH ROW
16 EXECUTE PROCEDURE func_CPA();
```

5.3.5 Create contact

```
1  Type VARCHAR(10),Email E_MAIL,Street VARCHAR(40),City VARCHAR(40)
   ,Postal_Code Integer, Country VARCHAR(15), TypeA VARCHAR(15),
   Number_L VARCHAR(15),
2  Landline_R VARCHAR(15),Number_M VARCHAR(15),Mobile_R VARCHAR(15))
3  LANGUAGE 'plpgsql'
4  AS $$
5      DECLARE
6      query varchar(500);
7      idC int;
8  BEGIN
9      query := 'INSERT INTO CONTACT(First_Name,Last_Name,Photo,Type)
   VALUES (' || quote_literal(first_name) || ',' || quote_literal(
   last_name) || ',' || quote_literal(Photo) || ',' ||
   quote_literal(Type) || ')';
10     EXECUTE query;
11     SELECT idcontact into idC from CONTACT C where C.first_name =
   quote_literal(first_name) and C.last_name = quote_literal(
   last_name) LIMIT 1;
12     query := 'INSERT INTO EMAIL(email,idContact) VALUES (' ||
   quote_literal(Email) || ',' || quote_literal(idC) || ')';
13     EXECUTE query;
14     query := 'INSERT INTO ADDRESS(idContact, street, city,
   Postal_Code, Country, typeA) VALUES (' || quote_literal(idC) ||
   ',' || quote_literal(street) || ',' || quote_literal(city) ||
   ',' || quote_literal(Postal_Code) || ',' || quote_literal(
   Country) || ',' || quote_literal(TypeA) || ')';
15     EXECUTE query;
16     query := 'INSERT INTO MOBILE(Number, idContact) VALUES (' ||
   quote_literal(Number_M) || ',' || quote_literal(idC) || ')';
17     EXECUTE query;
18     query := 'INSERT INTO MOBILE(Number, idContact,Mobile) VALUES ('
   || quote_literal(Number_L) || ',' || quote_literal(idC) || ','
   || quote_literal(Landline_R) || ')';
19     EXECUTE query;
20 END;
21 $$;
22
23 -- insert_email: procedura per l'inserimento di una mail
```

5.3.6 Insert Email

```
1  LANGUAGE 'plpgsql'
2  AS $$
3      DECLARE
4      query varchar(500);
5  BEGIN
6      query := 'INSERT INTO EMAIL(email,idContact) VALUES (' ||
   quote_literal(Email) || ',' || quote_literal(idContact) || ')';
7      EXECUTE query;
8  END;
9  $$;
10
11 -- insert_account: procedura per l'inserimento di una mail
```

5.3.7 Insert account

```

1
2 CREATE OR REPLACE PROCEDURE insert_account(idContact VARCHAR(500),
      NameF VARCHAR(30), Nickname VARCHAR(20), Bio Varchar(40), Email
      E_MAIL)
3 LANGUAGE 'plpgsql'
4 AS $$
5 DECLARE
6     query varchar(500);
7 BEGIN
8     query := 'INSERT INTO MESSAGING_ACCOUNT(idContact, NameF,
      Nickname, Bio, Email) VALUES (' || quote_literal(idContact) ||
      ', ' || quote_literal(idContact) || ', ' || quote_literal(NameF)
      || ', ' || quote_literal(Nickname) || ', ' || quote_literal(Bio)
      || ', ' || quote_literal(Email) || ')';
9     EXECUTE query;
10 END;
11 $$;

```

5.3.8 Insert Contact group

```

1
2 CREATE OR REPLACE PROCEDURE insert_contact_group(idContact VARCHAR
      (500), name VARCHAR(15))
3 LANGUAGE 'plpgsql'
4 AS $$
5 DECLARE
6     query varchar(500);
7 BEGIN
8     query := 'INSERT INTO CONTACT_GROUP(name, idContact) VALUES (' ||
      quote_literal(name) || ', ' || quote_literal(idContact) || ')';
9     EXECUTE query;
10 END;
11 $$;

```

5.3.9 Insert address

```
1
2 CREATE OR REPLACE PROCEDURE insert_address(idContact VARCHAR(500),
3       Street VARCHAR(40), City VARCHAR(40), Postal_Code Integer,
4       Country VARCHAR(15), TypeA VARCHAR(15))
5 LANGUAGE 'plpgsql'
6 AS $$
7 DECLARE
8     query varchar(500);
9 BEGIN
10     query := 'INSERT INTO ADDRESS(idContact, street, city,
11           Postal_Code, Country, typeA) VALUES (' || quote_literal(
12           idContact) || ',' || quote_literal(street) || ',' ||
13           quote_literal(city) || ',' || quote_literal(Postal_Code) || ',' ||
14           quote_literal(Country) || ',' || quote_literal(TypeA) || ')'
15           ';
16 EXECUTE query;
17 END;
18 $$;
```

5.3.10 insert mobile

```
1
2 CREATE OR REPLACE PROCEDURE insert_mobile(idContact VARCHAR(500),
3       Number_M VARCHAR(15), Mobile_R VARCHAR(15))
4 LANGUAGE 'plpgsql'
5 AS $$
6 DECLARE
7     query varchar(500);
8 BEGIN
9     IF Mobile_R = 'NULL' THEN
10         query := 'INSERT INTO MOBILE(Number, idContact) VALUES (' ||
11               quote_literal(Number_M) || ',' || quote_literal(idContact) || '
12               )';
13     ELSE
14         query := 'INSERT INTO MOBILE(Number, idContact, Landline) VALUES (
15               ' || quote_literal(Number_M) || ',' || quote_literal(idContact)
16               || ',' || quote_literal(Mobile_R) || ')';
17     EXECUTE query;
18 END IF;
19 END;
20 $$;
```

5.3.11 insert landline

```
1
2 CREATE OR REPLACE PROCEDURE insert_landline(idContact VARCHAR(500),
3       Number_L VARCHAR(15), Landline_R VARCHAR(15))
4   LANGUAGE 'plpgsql'
5   AS $$
6     DECLARE
7       query varchar(500);
8   BEGIN
9     query := 'INSERT INTO LANDLINE(Number, idContact, Landline) VALUES
10      (' || quote_literal(Number_M) || ', ' || quote_literal(
11      idContact) || ', ' || quote_literal(Landline_R) || ')';
12   EXECUTE query;
13 END;
14 $$;
```

5.3.12 Modify contact

```
1
2 CREATE OR REPLACE PROCEDURE modify_contact(row_name VARCHAR(15),
3       idcontact VARCHAR(10), update_var VARCHAR(30))
4   LANGUAGE 'plpgsql'
5   AS $$
6     DECLARE
7       query varchar(500);
8   BEGIN
9     query = 'UPDATE Contact SET ' || row_name || ' = ' ||
10      quote_literal(update_var) || ' where idcontact = ' || idcontact
11      ;
12   EXECUTE query;
13 END;
14 $$;
```

5.3.13 Modify contact address

```
1
2 CREATE OR REPLACE PROCEDURE modify_contact_address(row_name VARCHAR
  (15), idContact VARCHAR(10), idAddress VARCHAR(10), update_var
  VARCHAR(30))
3   LANGUAGE 'plpgsql'
4   AS $$
5   DECLARE
6     query varchar(500);
7 BEGIN
8   query = 'UPDATE address SET ' || row_name || ' = ' ||
  quote_literal(update_var) || ' WHERE idcontact = ' || idcontact
  || ' AND idaddress = ' || idaddress;
9   EXECUTE query;
10 END;
11 $$;
```

5.3.14 Modify contact email

```
1 CREATE OR REPLACE PROCEDURE modify_contact_email(idContact VARCHAR
  (10), email VARCHAR(30), update_var VARCHAR(30))
2   LANGUAGE 'plpgsql'
3   AS $$
4   DECLARE
5     query varchar(500);
6 BEGIN
7   IF EXISTS (SELECT email from email where email=quote_literal(
  update_var)) THEN RAISE NOTICE 'la nuova email    gia presente
  nelle informazioni di un altro contatto';
8   ELSE
9     query = 'UPDATE email SET email = ' || quote_literal(update_var
  ) || ' where idcontact = ' || idcontact || ' AND email = ' ||
  quote_literal(email);
10    EXECUTE query;
11  END IF;
12 END;
13 $$;
```


5.3.15 Modify contact mobile

```
1 CREATE OR REPLACE PROCEDURE modify_contact_mobile(row_name VARCHAR
  (15), idContact VARCHAR(10), update_var VARCHAR(30))
2 LANGUAGE 'plpgsql'
3 AS $$
4 DECLARE
5     query varchar(500);
6 BEGIN
7     query = 'UPDATE mobile SET ' || row_name || ' = ' ||
      quote_literal(update_var) || ' where idcontact = ' || idcontact
8     ;
9     EXECUTE query;
10 END;
11 $$;
12 -- modify_contact_landline: modifica di un numero fisso del
    contatto
```

5.3.16 Modify contact landline

```
1 CREATE OR REPLACE PROCEDURE modify_contact_landline(row_name
  VARCHAR(15), idContact VARCHAR(10), update_var VARCHAR(30))
2 LANGUAGE 'plpgsql'
3 AS $$
4 DECLARE
5     query varchar(500);
6 BEGIN
7     query = 'UPDATE landline SET ' || row_name || ' = ' ||
      quote_literal(update_var) || ' where idcontact = ' || idcontact
8     ;
9     EXECUTE query;
10 END;
11 $$;
```

5.3.17 Modify contact account

```
1
2 CREATE OR REPLACE PROCEDURE modify_contact_account(row_name VARCHAR
  (15), idContact VARCHAR(10), idAccount VARCHAR(10), update_var
  VARCHAR(30))
3   LANGUAGE 'plpgsql'
4   AS $$
5   DECLARE
6     query varchar(500);
7 BEGIN
8   query = 'UPDATE messaging_account SET ' || row_name || ' = ' ||
  quote_literal(update_var) || ' WHERE idcontact = ' || idcontact
  || ' AND idaccount = ' || idaccount;
9   EXECUTE query;
10 END;
11 $$;
```

5.3.18 Delete contact

```
1
2 CREATE OR REPLACE PROCEDURE delete_contact(idContact VARCHAR(10))
3   LANGUAGE 'plpgsql'
4   AS $$
5   DECLARE
6     query varchar(500);
7 BEGIN
8   query = 'DELETE FROM Contact where idcontact = ' || idcontact;
9   EXECUTE query;
10 END;
11 $$;
```

5.3.19 Filter First Name

```
1
2 CREATE OR REPLACE PROCEDURE filter_first_name(first_name VARCHAR
3         (25))
4     LANGUAGE 'plpgsql'
5     AS $$
6     DECLARE
7         query varchar(500);
8     BEGIN
9         query = 'create or replace view filter_first_name as select *
10        from CONTACT where first_name = ' || quote_literal(first_name);
11        EXECUTE query;
12    END;
13    $$;
```

5.3.20 Filter email

```
1
2 CREATE OR REPLACE PROCEDURE filter_email(Email E_MAIL)
3     LANGUAGE 'plpgsql'
4     AS $$
5     DECLARE
6         query varchar(500);
7     BEGIN
8         query = 'create or replace view filter_email as select C.
9        first_name,C.last_name from CONTACT C, EMAIL E where C.
10        idContact=E.idContact AND E.email = ' || quote_literal(Email);
11        EXECUTE query;
12    END;
13    $$;
```

5.3.21 Filter mobile

```
1
2 CREATE OR REPLACE PROCEDURE filter_mobile(Number varchar(15))
3     LANGUAGE 'plpgsql'
4     AS $$
5     DECLARE
6         query varchar(500);
7 BEGIN
8     query = 'create or replace view filter_mobile as select C.
9             first_name,C.last_name from CONTACT C join Mobile M on C.
10            idContact = M.idContact where M.Number = ' || quote_literal(
11            Number);
12     EXECUTE query;
13 END;
14 $$;
```

5.3.22 Filter landline

```
1
2 CREATE OR REPLACE PROCEDURE filter_landline(Number varchar(15))
3     LANGUAGE 'plpgsql'
4     AS $$
5     DECLARE
6         query varchar(500);
7 BEGIN
8     query = 'create or replace view filter_landline as select C.
9             first_name,C.last_name from CONTACT C join Landline L on C.
10            idContact = L.idContact where L.Number = ' || quote_literal(
11            Number);
12     EXECUTE query;
13 END;
14 $$;
```

5.3.23 View private contact

```
1 CREATE OR REPLACE PROCEDURE view_private_contact()
2     LANGUAGE 'plpgsql'
3     AS $$
4     DECLARE
5         query varchar(500);
6 BEGIN
7     query = 'create or replace view view_private_contact as select
8             * from contact C,address A,landline L where A.idcontact=L.
9             idContact AND L.idContact=C.idContact AND C.type=' ||
10            quote_literal(private);
11     EXECUTE query;
12 END;
13 $$;
```

5.3.24 Popolazione

La popolazione del database é presente nel file sql su github ([Clicca qui](#)).