Università 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

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'indirzzo 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.

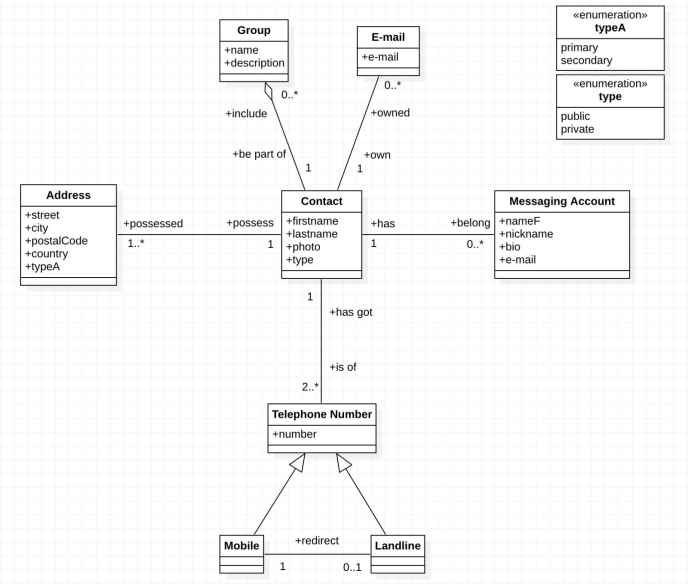
Capitolo 2

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 erediterenno gli attributi di Telephone Number e diventeranno due classsi separate.

3.3 Eliminazione degli attributi multivalore

Non sono presenti attributi multivalore nel class diagram.

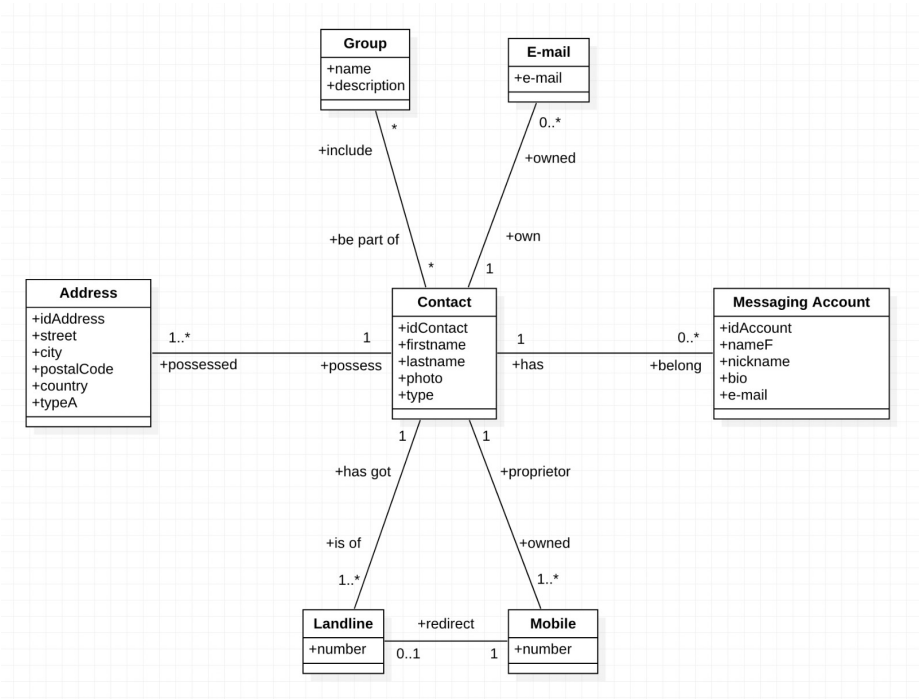
3.4 Eliminazione degli attributi composti

Non sono presenti attributi composti nel class diagram.

3.5 Scelta degli identificatori primari

Per quanto riguarda le classi **CONTACT**, **ADDRESS** e **MESSAGGING 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** (*Serial*) : l'id del contatto **firstName** (*String*) : Il nome del contatto **LastName** (*String*) : Il cognome del contatto **photo** (*String*) : Percorso della foto del contatto (opzionale) **type** (*String*) : Il tipo del contatto (public o private)
Messaging Account	**idAccount** (*Serial*) : L'id dell'account **nameF** (*String*) : Il nome del fornitore **nickname** (*String*) : Il nickname dell'account **bio** (*String*) : Lo stato dell'account **e-mail** (*String*) : La mail dell'account
Address	**street** (*String*) : La via dell'indirizzo **city** (*String*) : La città dell'indirizzo **postalCode** (*Int*) : Il codice postale dell'indirizzo **country** (*String*) : La nazione dell'indirizzo **typeA** (*String*) : Il tipo dell'indirizzo (primary or secondary)
Group	**name** (*String*) : Il nome del gruppo **description** (*String*) : La descrizione del gruppo
Mobile	**number** (*String*) : Il numero del mobile
Landline	**number** (*String*) : Il numero del landline
E-mail	**e-mail** (*String*) : 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] *ruolo* **has** : Rappresenta il contatto del messaging account **Messaging Account** [0...*] *ruolo* **belong** : Rappresenta l'account del contatto
Contact Address	**Contact** [1] *ruolo* **possess** : Rappresenta il contatto dell'indirizzo. **Address** [1...*] *ruolo* **possessed** : Rappresenta l'indirizzo del contatto.
Contact Group	**Contact** [*] *ruolo* **be part of** : Rappresenta il contatto nel gruppo. **Group** [*] *ruolo* **include** : Rappresenta il gruppo del contatto.
Contact e-mail	**Contact** [1] *ruolo* **own** : Rappresenta il contatto della mail. **E-mail** [0...*] *ruolo* **owned** : Rappresenta la mail del contatto.
Contact Mobile	**Contact** [1] *ruolo* **proprietor** : Rappresenta il contatto del mobile. **Mobile** [1...*] *ruolo* **owned** : Rappresenta il mobile del contatto.
Contact Landline	**Contact** [1] *ruolo* **has got** : Rappresenta il contatto del landline. **Landline** [1...*] *ruolo* **is of** : Rappresenta il landline del contatto.
Number Redirect	**Landline** [0.1] *redirect* **has got** : Rappresenta il numero fisso a cui viene reinderizzata una chiama. **Mobile** [1] *redirect* **is of** : Rappresenta il numero di telefono a cui viene reinderizzata una chiama.

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**, *idContact*) *idContact* $\mapsto$ CONTACT.idContact
GROUP	(**name**,description)
CONTACT_GROUP	(**name**,*idContact*) *name* $\mapsto$ GROUP.name *idContact* $\mapsto$ CONTACT.idContact
ADDRESS	(**idAddress**, *idContact*, street, city, PostalCode, Country, typeA) *idContact* $\mapsto$ CONTACT.idContact
MESSAGING_ACCCOUNT	(**idAcccount**, *idContact*, NameF, nickaname, bio, e-mail) *idContact* $\mapsto$ CONTACT.idContact *e-mail* $\mapsto$ E-MAIL.e-mail
LANDLINE	(**Number**, ***idContact***,*Mobile*) *idContact* $\mapsto$ CONTACT.idContact *Mobile* $\mapsto$ MOBILE.Number
MOBILE	(**Number**, ***idContact***,*Landline*) *idContact* $\mapsto$ CONTACT.idContact *Landline* $\mapsto$ LANDLINE.Number

Capitolo 5

Progettazione Fisica

5.1 Creazione dei domini

5.1.1 Valid email

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

5.1.2 Valid name

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

5.1.3 Valid Lastname

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

5.2 Creazione Tabelle

5.2.1 Contact

```
CREATE TABLE CONTACT(
  idContact SERIAL NOT NULL,
  First_Name FIRST_NAME NOT NULL,
  Last_Name LAST_NAME NOT NULL,
  Photo Varchar(30),
  Type VARCHAR(10) NOT NULL
);

ALTER TABLE CONTACT
-- Aggiunta del vincolo di chiave primaria
  ADD CONSTRAINT Contact_pk PRIMARY KEY(idContact);

-- Modifica dell'inizio del serial
   ALTER SEQUENCE contact_idContact_seq RESTART WITH 1 INCREMENT BY
     1;
```

5.2.2 Messaging Account

```
CREATE TABLE MESSAGING_ACCOUNT(
  idAccount SERIAL NOT NULL,
  idContact SERIAL NOT NULL,
  NameF VARCHAR(30) NOT NULL,
  Nickname VARCHAR(20) NOT NULL,
  Bio Varchar(40),
  Email E_MAIL NOT NULL
  );
ALTER TABLE MESSAGING_ACCOUNT
-- Aggiunta del vincolo di chiave primaria
  ADD CONSTRAINT msg_pk PRIMARY KEY(idAccount),
-- Aggiunta del vincolo di chiave esterna sulla tabella CONTACT
  ADD CONSTRAINT msg_fk FOREIGN KEY(idContact) REFERENCES CONTACT(
    idContact)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
-- Aggiunta del vincolo di chiave esterna sulla tabella EMAIL
  ADD CONSTRAINT msg_email_fk FOREIGN KEY(Email) REFERENCES EMAIL(
    Email)
    ON UPDATE CASCADE
    ON DELETE RESTRICT;
```

5.2.3 Address

```
CREATE TABLE ADDRESS(
  idAddress SERIAL NOT NULL,
  idContact SERIAL NOT NULL,
  Street VARCHAR(40) NOT NULL,
  City VARCHAR(40) NOT NULL,
  Postal_Code Integer NOT NULL,
  Country VARCHAR(15) NOT NULL,
  TypeA VARCHAR(15) NOT NULL
);

ALTER TABLE ADDRESS
-- Aggiunta del vincolo di chiave primaria
  ADD CONSTRAINT address_pk PRIMARY KEY(idAddress),
-- Aggiunta del vincolo di chiave esterna sulla tabella CONTACT
  ADD CONSTRAINT address_fk FOREIGN KEY(idContact) REFERENCES
    CONTACT(idContact)
    ON UPDATE CASCADE
    ON DELETE CASCADE;

```

5.2.4 Landline

```
CREATE TABLE LANDLINE(
  Number VARCHAR(15) NOT NULL,
  idContact SERIAL NOT NULL,
  Mobile VARCHAR(15) NOT NULL,

  PRIMARY KEY(Number,idContact)
);

ALTER TABLE LANDLINE
-- Aggiunta del vincolo di chiave esterna sulla tabella CONTACT
  ADD CONSTRAINT landline_fk FOREIGN KEY(idContact) REFERENCES
    CONTACT(idContact)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
-- Aggiunta del vincolo di chiave esterna sulla tabella MOBILE
  ADD CONSTRAINT landline_mobile_fk FOREIGN KEY(idContact,Mobile)
    REFERENCES MOBILE(idContact,Number)
    ON UPDATE CASCADE
    ON DELETE CASCADE;
```

5.2.5 Mobile

```
CREATE TABLE MOBILE(
  Number VARCHAR(15) NOT NULL,
  idContact SERIAL NOT NULL,
  Landline VARCHAR(15),

  PRIMARY KEY(Number,idContact)
);
ALTER TABLE MOBILE
-- Aggiunta del vincolo di chiave esterna sulla tabella CONTACT
  ADD CONSTRAINT mobile_fk FOREIGN KEY(idContact) REFERENCES
    CONTACT(idContact)
    ON UPDATE CASCADE
    ON DELETE CASCADE,
-- Aggiunta del vincolo di chiave esterna sulla tabella LANDLINE
  ADD CONSTRAINT mobile_landline_fk FOREIGN KEY(idContact,Landline)
     REFERENCES LANDLINE(idContact,Number)
    ON UPDATE CASCADE
    ON DELETE CASCADE;
```

5.2.6 Group

```
CREATE TABLE GROUPS(
  name VARCHAR(15) NOT NULL,
  description VARCHAR(50)
);

ALTER TABLE GROUPS
-- Aggiunta del vincolo di chiave primaria
  ADD CONSTRAINT groups_pk PRIMARY KEY(name);

-- ****************************
```

5.2.7 Contact Group

```
  name VARCHAR(15) NOT NULL,
  idContact SERIAL NOT NULL
);

ALTER TABLE CONTACT_GROUP
-- Aggiunta del vincolo di chiave primaria
  ADD CONSTRAINT group_pk PRIMARY KEY(name,idContact),
-- Aggiunta del vincolo di chiave esterna sulla tabella CONTACT
  ADD CONSTRAINT group_fk FOREIGN KEY(idContact) REFERENCES CONTACT
    (idContact)
    ON UPDATE CASCADE
    ON DELETE CASCADE;
```

5.2.8 E-mail

```
  CREATE TABLE EMAIL(
  Email E_MAIL NOT NULL,
  idContact SERIAL NOT NULL
);

ALTER TABLE EMAIL
-- Aggiunta del vincolo di chiave primaria
  ADD CONSTRAINT email_pk PRIMARY KEY(Email),
-- Aggiunta del vincolo di chiave esterna sulla tabella CONTACT
  ADD CONSTRAINT email_fk FOREIGN KEY(idContact) REFERENCES CONTACT
    (idContact)
    ON UPDATE CASCADE
    ON DELETE CASCADE;

```

5.3 Funzioni e trigger

5.3.1 Check existing email

```
    LANGUAGE 'plpgsql'
  AS $$
BEGIN
    IF NOT EXISTS (SELECT * FROM email where new.email=email) THEN
    RAISE NOTICE 'l email del fornitore non    presente tra gli
    indirizzi mail del contatto';
  return null;
    END IF;
  return new;
END;
$$;

CREATE TRIGGER check_existing_email BEFORE INSERT ON
    Messaging_account
FOR EACH ROW
EXECUTE PROCEDURE Func_CEE();

--- check_duplicate_email: controllo l'esistenza di un altro
    contatto duplicato (il contatto sar  duplicato quando avr  la
     stessa email)
```

5.3.2 Check duplicate email

```
    LANGUAGE 'plpgsql'
  AS $$
BEGIN
    IF EXISTS (SELECT * FROM email where new.email=email and
    idcontact<>new.idcontact) THEN RAISE NOTICE 'l email    gia
    stata utilizzata da un altro contatto usare la funzione per
    eliminare il contatto o modificarlo';
  RETURN NULL;
    END IF;
  return new;
END;
$$;

CREATE TRIGGER check_duplicate_email BEFORE INSERT ON email
FOR EACH ROW
EXECUTE PROCEDURE func_CDE();

--- check_duplicate_account_mail: controllo l'esistenza di un altro
     account di un altro contatto con la stessa mail
```

5.3.3 Check duplicate account mail

```
    LANGUAGE 'plpgsql'
  AS $$
BEGIN
    IF EXISTS (SELECT * FROM messaging_account where new.email=
    email and idcontact<>new.idcontact) THEN RAISE NOTICE 'l email
       gia stata utilizzata da un altro account di un altro
    contatto, usare la funzione per eliminare l account o
    modificarlo';
  RETURN NULL;
    END IF;
  return new;
END;
$$;

CREATE TRIGGER check_duplicate_account_mail BEFORE INSERT ON
    messaging_account
FOR EACH ROW
EXECUTE PROCEDURE func_CDAM();
```

5.3.4 Check primary address

```
    LANGUAGE 'plpgsql'
  AS $$
BEGIN
    IF (SELECT typeA FROM address where new.idContact = address.
    idContact ORDER BY TypeA LIMIT 1) <> 'primary' THEN
  RAISE NOTICE 'non    presente un indirizzo principale associato a
     questo contatto';
  RETURN NULL;
    END IF;
  return new;
END;
$$;

CREATE TRIGGER check_primary_address BEFORE INSERT ON ADDRESS
FOR EACH ROW
EXECUTE PROCEDURE func_CPA();
```

5.3.5 Create contact

```
  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),
  Landline_R VARCHAR(15),Number_M VARCHAR(15),Mobile_R VARCHAR(15))
  LANGUAGE 'plpgsql'
  AS $$
    DECLARE
    query varchar(500);
  idC int;
BEGIN
    query := 'INSERT INTO CONTACT(First_Name,Last_Name,Photo,Type)
    VALUES (' || quote_literal(first_name) || ',' || quote_literal(
    last_name) || ',' || quote_literal(Photo) || ',' ||
    quote_literal(Type) || ')';
    EXECUTE query;
  SELECT idcontact into idC from CONTACT C where C.first_name =
    quote_literal(first_name) and C.last_name = quote_literal(
    last_name) LIMIT 1;
  query := 'INSERT INTO EMAIL(email,idContact) VALUES (' ||
    quote_literal(Email) || ',' || quote_literal(idC) || ')';
  EXECUTE query;
  query := 'INSERT INTO ADDRESS(idContact, street, city,
    Postal_Code, Country, typeA) VALUES (' || quote_literal(idC) ||
     ',' || quote_literal(street) || ',' || quote_literal(city) ||
    ',' || quote_literal(Postal_Code) || ',' || quote_literal(
    Country) || ',' || quote_literal(TypeA) || ')';
  EXECUTE query;
  query := 'INSERT INTO MOBILE(Number, idContact) VALUES (' ||
    quote_literal(Number_M) || ',' || quote_literal(idC) || ')';
  EXECUTE query;
  query := 'INSERT INTO MOBILE(Number, idContact,Mobile) VALUES ('
    || quote_literal(Number_L) || ',' || quote_literal(idC) || ','
    || quote_literal(Landline_R) || ')';
  EXECUTE query;
END;
$$;

-- insert_email: procedura per l'inserimento di una mail
```

5.3.6 Insert Email

```
  LANGUAGE 'plpgsql'
  AS $$
    DECLARE
    query varchar(500);
BEGIN
  query := 'INSERT INTO EMAIL(email,idContact) VALUES (' ||
    quote_literal(Email) || ',' || quote_literal(idContact) || ')';
  EXECUTE query;
END;
$$;

-- 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),
       Street VARCHAR(40), City VARCHAR(40), Postal_Code Integer,
       Country VARCHAR(15), TypeA VARCHAR(15))
3    LANGUAGE 'plpgsql'
4    AS $$
5      DECLARE
6      query varchar(500);
7  BEGIN
8    query := 'INSERT INTO ADDRESS(idContact, street, city,
       Postal_Code, Country, typeA) VALUES (' || quote_literal(
       idContact) || ',' || quote_literal(street) || ',' ||
       quote_literal(city) || ',' || quote_literal(Postal_Code) || ','
        || quote_literal(Country) || ',' || quote_literal(TypeA) || ')
       ';
9    EXECUTE query;
10 END;
11 $$;
```

5.3.10 insert mobile

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

5.3.11 insert landline

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

5.3.12 Modify contact

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

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

```
CREATE OR REPLACE PROCEDURE modify_contact_mobile(row_name VARCHAR
    (15), idContact VARCHAR(10), update_var VARCHAR(30))
    LANGUAGE 'plpgsql'
  AS $$
    DECLARE
    query varchar(500);
BEGIN
    query = 'UPDATE mobile SET '|| row_name || ' = ' ||
    quote_literal(update_var) || ' where idcontact = ' || idcontact
     ;
    EXECUTE query;
END;
$$;

-- modify_contact_landline: modifica di un numero fisso del
    contatto
```

5.3.16 Modify contact landline

```
CREATE OR REPLACE PROCEDURE modify_contact_landline(row_name
    VARCHAR(15), idContact VARCHAR(10), update_var VARCHAR(30))
    LANGUAGE 'plpgsql'
  AS $$
    DECLARE
    query varchar(500);
BEGIN
    query = 'UPDATE landline SET '|| row_name || ' = ' ||
    quote_literal(update_var) || ' where idcontact = ' || idcontact
     ;
    EXECUTE query;
END;
$$;
```

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

5.3.20 Filter email

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

5.3.21 Filter mobile

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

5.3.22 Filter landline

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

5.3.23 View private contact

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

5.3.24 Popolazione

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