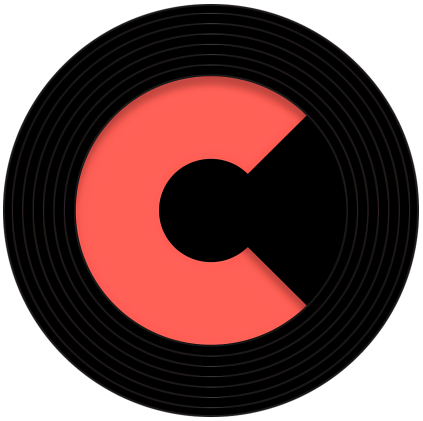
UNIVERSITÀ DEGLI STUDI DELL'AQUILA

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica.

Corso di Laurea in Informatica

Insegnamento: Laboratorio di Basi di Dati



Progetto: Collectors

Data di consegna: 19/07/2023

Membri del team		
Cognome e nome	**Matricola**	**Indirizzo e-mail**
Di Cresce Manuel	279650	manuel.dicresce@student.univaq.it
Paolocci Giacomo	278662	giacomo.paolocci@student.univaq.it
Lorusso Raffaele	279064	raffaele.lorusso@student.univaq.it

A.A 2022/2023

Sommario

A - Contributi al progetto ed altre informazioni

Lo sviluppo del progetto si è articolato in egual modo tra i vari membri del gruppo, riportiamo di seguito una tabella che cerca di riassumere molto sinteticamente le attività svolte da ognuno di noi.

Di Cresce Manuel	● Realizzazione del DDL; ● Realizzazione dei triggers; ● Realizzazione delle query; ● Implementazione JDBC; ● Implementazione delle query in Java; ● Implementazione interfaccia grafica JavaFX, contributo maggiore; ● Ottimizzazione dell'app; ● Realizzazione della documentazione, contributo maggiore.
Paolocci Giacomo	● Realizzazione del DDL; ● Realizzazione dei triggers; ● Realizzazione delle query; ● Implementazione JDBC; ● Implementazione delle query in Java con contributo particolare sulla query 13; ● Implementazione interfaccia grafica JavaFX; ● Realizzazione della documentazione.
Lorusso Raffaele	● Realizzazione del DDL; ● Realizzazione dei triggers; ● Realizzazione delle query con contributo particolare sulla query n°8; ● Realizzazione dei test cases per popolare il database; ● Implementazione delle query in Java con contributo particolare

	sulla query 13; ● Implementazione interfaccia grafica JavaFX; ● Realizzazione della documentazione.

Alcune informazioni riguardo il progetto:
Abbiamo tentato di stabilire un ordine tra le cartelle in modo tale da consentire una migliore visualizzazione dei vari contenuti.

Abbiamo allegato il dump del database e un unico script SQL che contiene l'intero progetto, dal DDL ai triggers, query (operazioni) e test cases nella cartella "**Progetto**". Questa cartella contiene anche l'implementazione Java con il JDBC e una semplice applicazione sviluppata con JavaFX che consente di usare almeno le operazioni richieste dalla specifica.

Sono inoltre disponibili per la visualizzazione anche gli script SQL separati (e quindi non riuniti in un unico file), questi ultimi si trovano nella cartella "**Old**" e sono rispettivamente: "**struttura.sql**", "**operazioni.sql**", "**triggers.sql**" e "**testcases.sql**".

1 - Formalizzazione ed analisi dei requisiti

1.1 - Specifiche del progetto

Il database Collectors memorizza informazioni relative a collezioni di dischi (anche se lo stesso tipo di database potrebbe adattarsi quasi a qualunque tipo di collezione, useremo i dischi come caso di studio in modo da poter aggiungere più dettagli alla specifica).

Nel database andranno prima di tutto registrati i dati (saranno sufficienti nickname e indirizzo email) relativi ai collezionisti , e poi i dati relativi alle loro collezioni di dischi (dovete prevedere che ogni collezionista possa creare più collezioni, ciascuna con un nome distinto). Per ogni disco in una collezione, dovranno essere specificati gli autori, il titolo, l'anno di uscita, l'etichetta (casa editrice), il genere (scelto da una lista predefinita di generi musicali), lo stato di conservazione dell'oggetto (scelto da una lista predefinita), il formato (vinile, CD, digitale,...), il numero di barcode, se disponibile (i codici a barre garantiscono l'identificazione univoca dell'elemento), e poi ovviamente la lista delle tracce, ciascuna con titolo, durata, ed eventuali informazioni su compositore ed esecutore (cantante, musicista), se diverso da quelli dell'intero disco. Infine, ogni disco potrebbe essere associato a una o più immagini (copertina, retro, eventuali facciate interne o libretti, ecc.). Insomma, cercate di essere il più realistici possibile.

Per ogni disco, il collezionista potrà inoltre indicare l'eventuale numero di doppioni a sua disposizione (spesso si hanno più copie dello stesso disco, magari a seguito di scambi, e magari anche perché se ne prevede la rivendita).

I collezionisti potranno decidere di condividere la propria collezione con specifici utenti o in maniera pubblica. Ogni collezione avrà quindi associato un flag privato/pubblico e la lista di collezionisti con i quali è stata condivisa.

Ci sono indubbiamente svariati vincoli che possono essere applicati ai contenuti di questa base di dati. L'individuazione dei vincoli e la loro implementazione (con vincoli sulle tabelle, trigger o quantomeno definendo il codice e le query necessari a

effettuarne il controllo) costituiscono un requisito importante per lo sviluppo di un progetto realistico, e ne verrà tenuto conto durante la valutazione finale.

1.1.1 - Operazioni da realizzare

Di seguito sono illustrate schematicamente le operazioni previste sulla base di dati, ciascuna da realizzare tramite una query (o, se necessario, tramite più query, opzionalmente racchiuse in una stored procedure). Ovviamente, ogni ulteriore raffinamento o arricchimento di queste specifiche aumenterà il valore del progetto.

1. Inserimento di una nuova collezione.

2. Aggiunta di dischi a una collezione e di tracce a un disco.

3. Modifica dello stato di pubblicazione di una collezione (da privata a pubblica e viceversa) e aggiunta di nuove condivisioni a una collezione.

4. Rimozione di un disco da una collezione.

5. Rimozione di una collezione.

6. Lista di tutti i dischi in una collezione.

7. Track list di un disco.

8. Ricerca di dischi in base a nomi di autori/compositori/interpreti e/o titoli. Si potrà decidere di includere nella ricerca le collezioni di un certo collezionista e/o quelle condivise con lo stesso collezionista e/o quelle pubbliche. (Suggerimento: potete realizzare diverse query in base alle varie combinazioni di criteri di ricerca. Usate la UNION per unire i risultati delle ricerche effettuate sulle collezioni private, condivise e pubbliche)

9. Verifica della visibilità di una collezione da parte di un collezionista. (Suggerimento: una collezione è visibile a un collezionista se è sua, condivisa con lui o pubblica)

10. Numero dei brani (tracce di dischi) distinti di un certo autore (compositore, musicista) presenti nelle collezioni pubbliche.

11. Minuti totali di musica riferibili a un certo autore (compositore, musicista) memorizzati nelle collezioni pubbliche.

12. Statistiche (una query per ciascun valore): numero di collezioni di ciascun collezionista, numero di dischi per genere nel sistema.

13. Opzionalmente, dati un numero di barcode, un titolo e il nome di un autore, individuare tutti i dischi presenti nelle collezioni che sono più coerenti con questi dati (funzionalità utile, ad esempio, per individuare un disco già presente nel sistema prima di inserirne un doppione). L'idea è che il barcode è univoco, quindi i dischi con lo stesso barcode sono senz'altro molto coerenti, dopodichè è possibile cercare dischi con titolo simile e/o con l'autore dato, assegnando maggior punteggio di somiglianza a quelli che hanno più corrispondenze. Questa operazione può essere svolta con una stored procedure o implementata nell'interfaccia Java/PHP.

È possibile inserire procedure di gestione addizionali che si ritengano utili.

1.2 - Analisi dei requisiti

Dall’analisi della specifica del progetto sono state identificate le seguenti entità:

- Collezione;
- Collezionista;
- Disco;
- Genere;
- Autore;
- Traccia;
- Immagine;
- Etichetta (casa produttrice).

Per ogni entità, rispettivamente, abbiamo individuato diverse caratteristiche che ognuna di loro deve avere, per cui le riportiamo qui in ordine:

Collezione:

- Nome
- Flag

La flag serve ad indicare se la collezione è condivisa o meno con altri utenti.

Collezionista:

- E-mail
- Nickname

Disco:

- Titolo
- Anno di uscita
- Barcode
- Formato
- Stato conservazione
- Descrizione conservazione

Genere:

- Nome

Abbiamo pensato di introdurre un'entità "Genere" in modo tale da poter aggiungere nuovi generi in futuro senza dover mettere mano al database.

Autore:

- Nome d'autore
- Info
- Ruolo
- Nome
- Cognome
- Data di nascita

Abbiamo riflettuto sull'univocità del nome d'autore e siamo giunti alla conclusione che normalmente esso è un marchio registrato, per cui nessuno può utilizzare tale nome al di fuori del detentore del marchio stesso.

Traccia:

- Titolo
- Durata

Immagine:

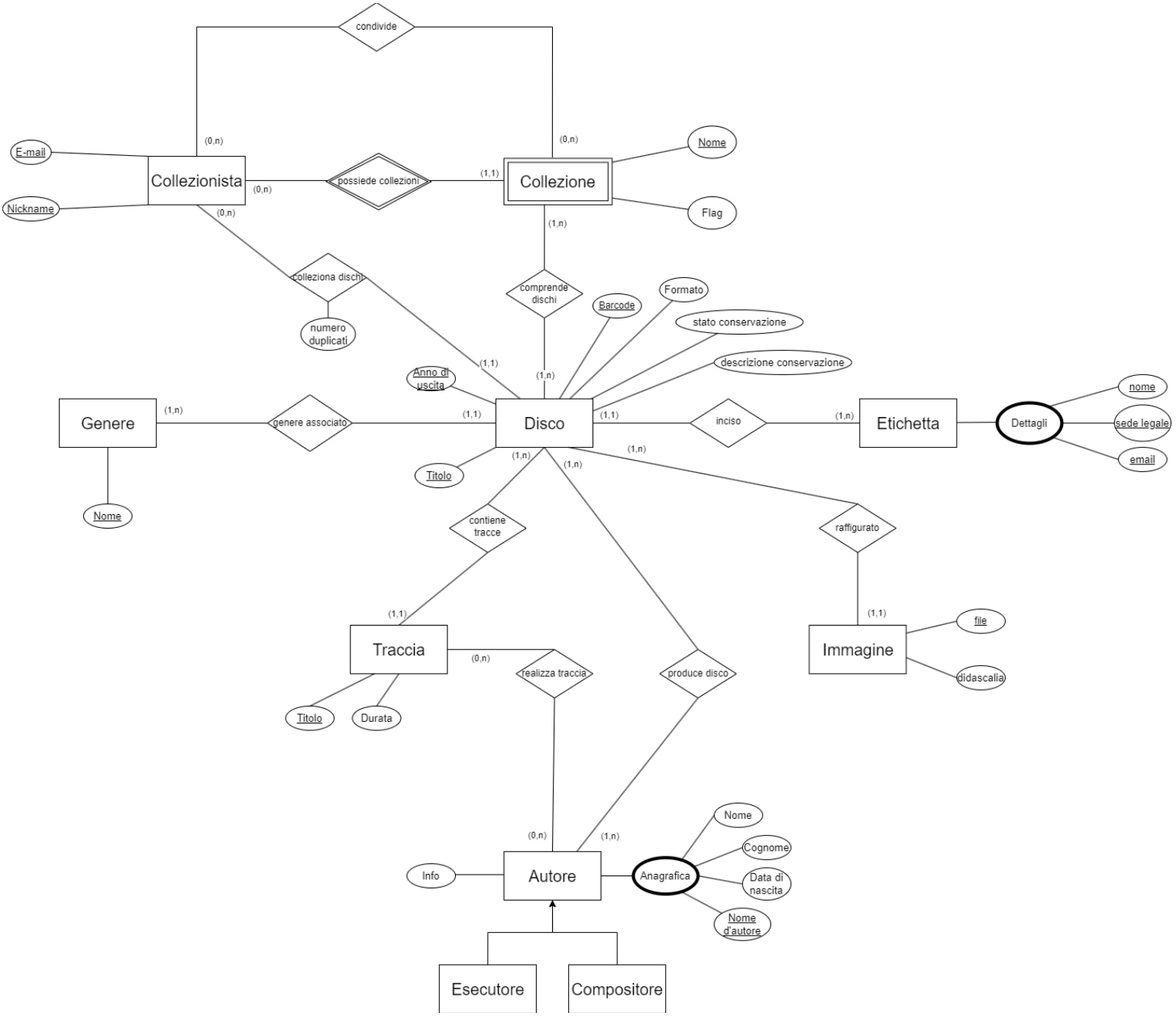
- ha un path che la identifica univocamente.
- didascalia

Questa entità mi permette di modellare i concetti di “Copertina”, “Retro della copertina” e “Libretto”, pensando il libretto come un insieme di immagini.

Etichetta:

- Nome
- Sede legale
- E-mail.

2 - Modello Entità-Relazione



Dato che la lettura del modello entità-relazione potrebbe risultare difficoltosa da questa documentazione, abbiamo deciso di includere nella cartella **Modelli**, il modello in questione in alta qualità, sotto il nome di **ER.png**.

3 - Formalizzazione dei vincoli non esprimibili nel modello ER

Di seguito elenchiamo i vincoli che abbiamo identificato durante la stesura del diagramma ER:

Collezione:

- Il suo attributo primo **nome** deve essere non nullo e unico per ogni collezione del dato collezionista.
- L'attributo **flag** è un boolean che viene utilizzato per rendere una collezione pubblica oppure privata, tale attributo va specificato sempre.

Collezionista:

- I suoi attributi **nickname** ed **e-mail** devono essere unici e non-nulli in quanto chiavi primarie.

Disco:

- I suoi attributi **titolo**, **anno di uscita** e **barcode** *(quest'ultimo quando presente)* nel complesso formano la chiave primaria di tale entità, sono quindi tutti non nulli e univoci.
- Gli attributi **formato** e **stato_conservazione** sono stati pensati come valori preesistenti nella base di dati, ma attenzione in quanto non è richiesto che essi siano non nulli.

Genere:

- Ha un unico attributo che lo **identifica univocamente**, quindi non può essere nullo né avere duplicati.

Autore:

- Il suo attributo **nome d'autore** identifica la chiave primaria dell'autore, quindi non può essere nullo e non possono esistere autori con lo stesso nome d'autore.
- Vi è una decomposizione dell'entità in due sottoentità che rappresentano rispettivamente l'**esecutore** e il **compositore**; l'idea è quella di sfruttare un tipo enumerativo per indicare il ruolo dell'autore, rispettivamente **compositore**, **esecutore** e **poliedrico** (per indicare che l'autore ricopre entrambi i ruoli); in questo modo possiamo ridurre il carico del modello ER, ma questa osservazione verrà applicata successivamente con la ristrutturazione di

tale modello.

Traccia:

- Il suo attributo **titolo** è la chiave primaria di tale tabella, quindi è sicuramente non nullo e unico.
- L'attributo **durata** è di tipo time.

Immagine:

- Il suo attributo **file** oltre ad essere chiave primaria e quindi not null e unique, è una stringa contenente il path dell'immagine.

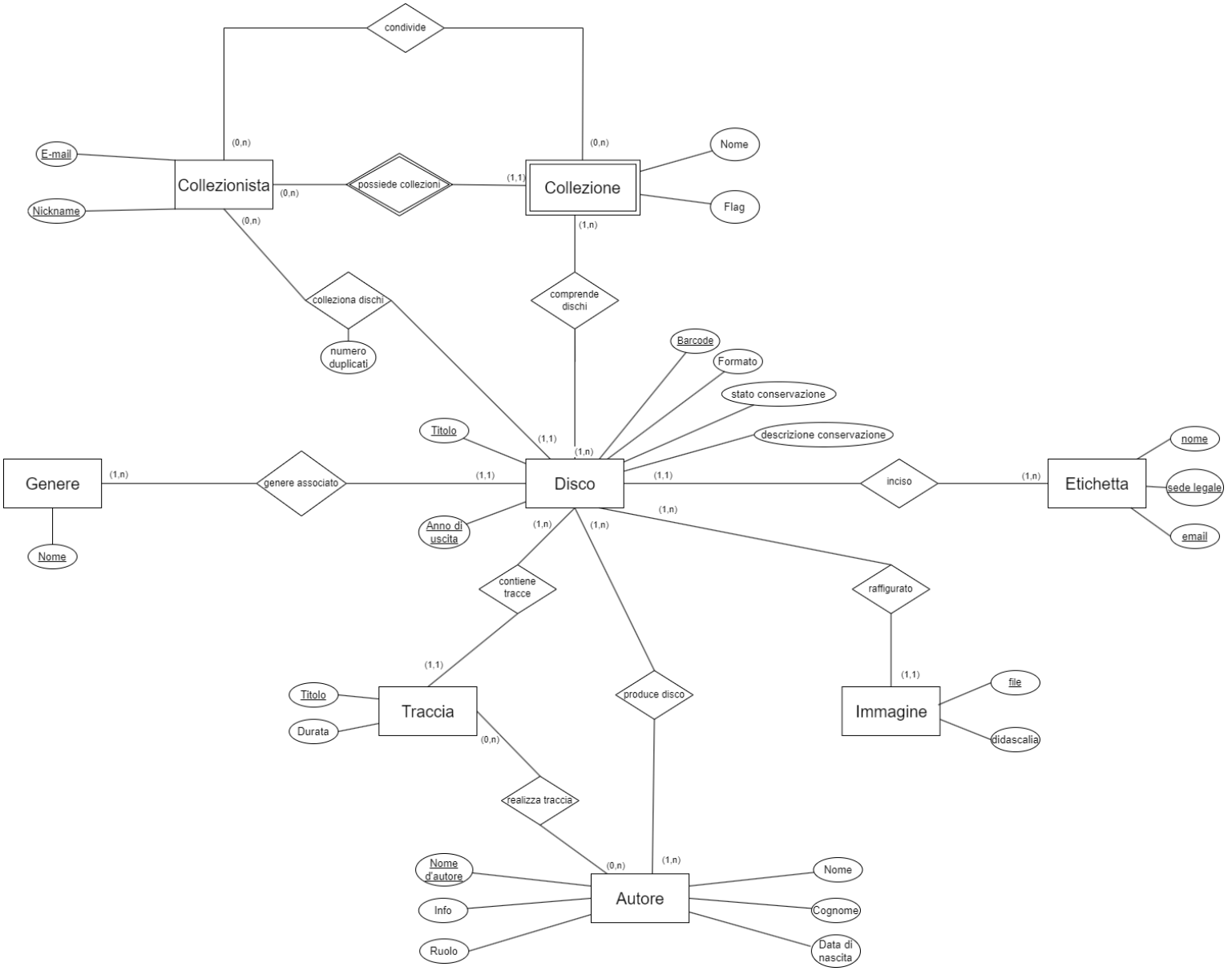
Etichetta:

- I suoi attributi **nome**, **sede_legale** ed **e-mail** sono chiave primaria della tabella, di conseguenza sono anche loro non nulli e senza duplicati.

Colleziona Dischi:

- Tale relazione modella il concetto di collezionamento di dischi da parte del collezionista, senza l'obbligo che tali dischi debbano essere all'interno di una collezione. La particolarità di tale relazione è che ha un attributo denominato: "**numero duplicati**". Attraverso tale attributo intero, è possibile tenere traccia di quante copie di un determinato disco ha il collezionista.
- All'inserimento del primo disco nel sistema da parte del collezionista, l'attributo avrà **numero duplicati = 0** e aumenterà ad **1** nel momento in cui egli inserirà una copia di tale disco e così via, non c'è un limite alle copie che egli può avere.

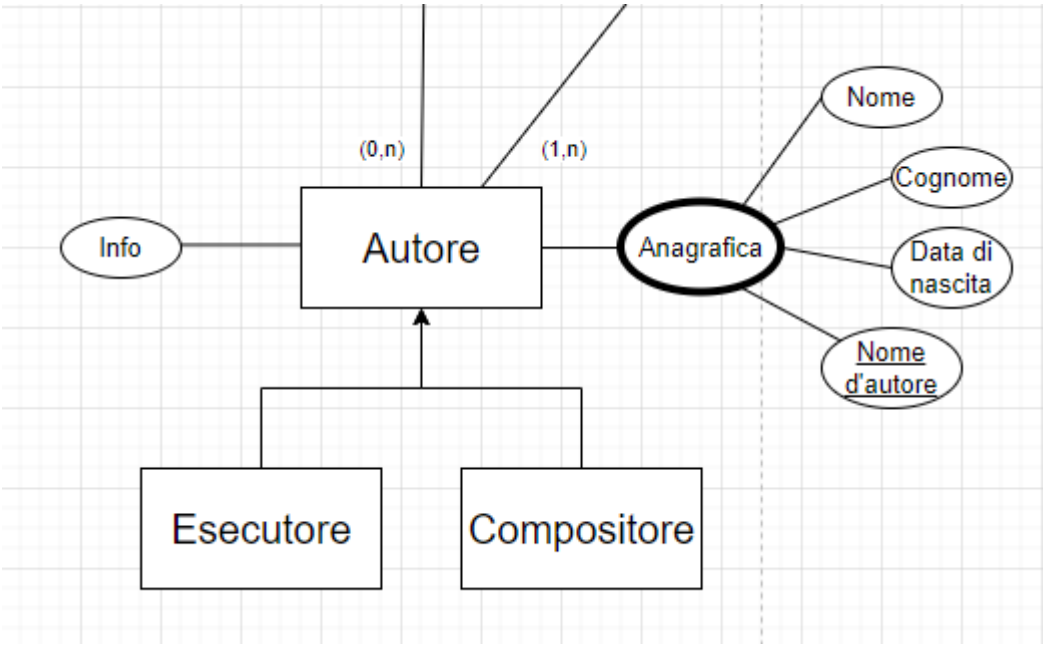
4 - Ristrutturazione Modello ER



Dato che la lettura del modello entità-relazione potrebbe risultare difficoltosa da questa documentazione, abbiamo deciso di includere nella cartella **Modelli**, il modello in questione in alta qualità, sotto il nome di **ER-Ristrutturato.png**.

Discutiamo in dettaglio i cambiamenti relativi alle entità: “Autore”, “Compositore” ed “Esecutore”.

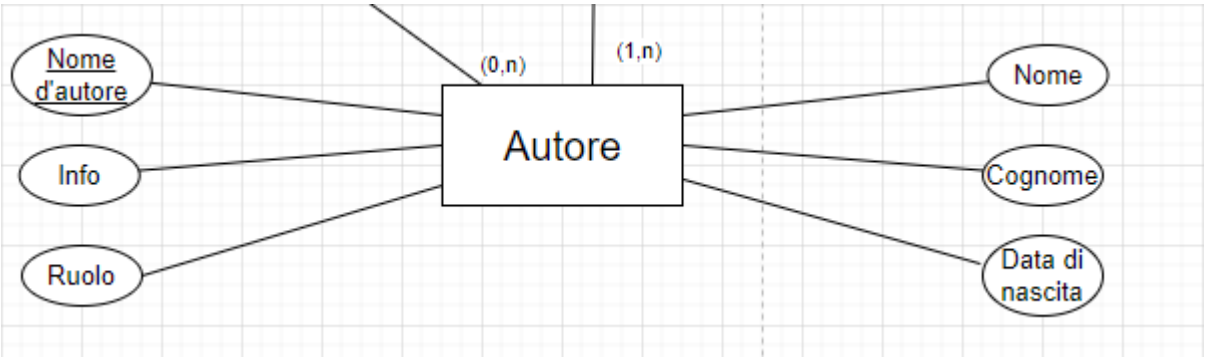
Inizialmente le varie entità erano così poste all’interno del diagramma ER:



Ciò che abbiamo deciso di attuare è una fusione verso l’alto (**figli-genitore**), tale decisione è stata presa in seguito ad una analisi delle entità Esecutore e Compositore che, come si può notare nella figura in alto, oltre gli attributi ereditati dal padre non hanno altri attributi, quindi effettivamente tali entità possono essere condensate in un’unica entità Autore che avrà un attributo aggiuntivo, denominato “Ruolo”.
Come ultimo passo per la ristrutturazione e l’ottimizzazione del diagramma ER, abbiamo deciso di rimuovere l’attributo composto “Anagrafica” e collegare direttamente i vari attributi all’entità “Autore”, in quanto l’entità finale non risultava poi così tanto carica.

Notare che la generalizzazione di partenza era totale, quindi l’Autore o è Esecutore o è Compositore; da qui è nata una riflessione riguardante il ruolo dell’autore, in quanto potrebbero esserci casi in cui l’autore sia esecutore ed anche compositore…
Adesso, da questo punto in poi avendo adottato un attributo “Ruolo”, abbiamo pensato di attribuire la denominazione “Poliedrico” ad un autore quando egli ricopre ambo i ruoli.

Prendendo in considerazione tali riflessioni, quel che otteniamo è la seguente porzione di diagramma ER:



5 - Modello relazionale

Per migliorare la leggibilità del modello relazione, indichiamo con una sottolineatura le chiavi primarie, mentre con una *sottolineatura in corsivo* le chiavi esterne.

Entità:
Collezionista(ID, email, nickname)
Collezione(ID, nome , flag, *ID_Collezionista*)
Disco(ID, titolo, genere, anno_uscita, barcode, formato, stato_conservazione, descrizione_conservazione, *ID_Genere*)
Traccia(ID, titolo, durata)
Autore(ID, nome_d'autore, info, ruolo, nome, cognome, data_nascita)
Immagine(ID, file, didascalia, *ID_Disco*)
Etichetta(ID, nome, sede_legale, email)
Genere(ID, nome)

Relazioni:
Condivide(*ID_Collezionista*, *ID_Collezione*)
Colleziona_dischi(*ID_Collezionista*, *ID_Disco*, numero_duplicati)
Comprende_dischi(*ID_Collezione*, *ID_Disco*)
Inciso(*ID_Disco*, *ID_Etichetta*)
Contiene_tracce(*ID_Disco*, *ID_Traccia*)
Produce_disco(*ID_Disco*, *ID_Autore*)
Realizza_traccia(*ID_Traccia*, *ID_Autore*)

6 - Progettazione fisica

6.1 - Implementazione del modello relazionale

Riportiamo di seguito lo script SQL per la creazione della struttura del database.

Tutto il codice di seguito riportato è situato nello script SQL denominato "**struttura.sql**".

```
drop schema if exists collectors;

create schema if not exists collectors;
use collectors;

drop table if exists collezionista;
drop table if exists collezione;
drop table if exists etichetta;
drop table if exists genere;
drop table if exists disco;
drop table if exists traccia;
drop table if exists autore;
drop table if exists immagine;
drop table if exists condivide;
drop table if exists colleziona dischi;
drop table if exists comprende_dischi;
drop table if exists produce_disco;
drop table if exists realizza_traccia;

drop user if exists 'admin'@'localhost';

create user 'admin'@'localhost' identified by 'admin';
grant all privileges on collectors.* to 'admin'@'localhost';

create table collezionista(
ID integer primary key auto_increment not null,
email varchar(320) unique not null check (email REGEXP
'[a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]+$'),
nickname varchar(25) unique not null
);
```

```sql
create table collezione(
ID integer primary key auto_increment not null,
nome varchar(25) not null,
flag boolean,
ID_collezionista integer not null,
foreign key (ID_collezionista) references collezionista(ID) on update
cascade on delete cascade
);

create table etichetta(
ID integer primary key auto_increment not null,
nome varchar(25) not null,
sede_legale varchar(255) not null,
email varchar(320) not null check (email REGEXP
'[a-zA-Z0-9._%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]+$')
);

create table genere(
ID integer primary key auto_increment not null,
nome varchar(25) unique not null
);

create table disco(
ID integer primary key auto_increment not null,
titolo varchar(35) not null,
anno_uscita smallint not null,
barcode varchar(128), /*Lunghezza massima barcode esistenti*/
formato enum('vinile', 'cd', 'digitale', 'cassetta'),
stato_conservazione enum ('nuovo', 'come nuovo', 'ottimo', 'buono',
'accettabile'),
descrizione_conservazione varchar(255),
ID_etichetta integer,
foreign key (ID_etichetta) references etichetta(ID) on update cascade on
delete set null,
ID_genere integer,
foreign key (ID_genere) references genere(id) on update cascade on
delete set null
);

create table traccia(
ID integer primary key auto_increment not null,
titolo varchar(50) unique not null,
durata time not null,
ID_disco integer,
foreign key (ID_disco) references disco(ID) on update cascade on delete
```

```
cascade
);

create table autore(
ID integer primary key auto_increment not null,
nome varchar(20),
cognome varchar(20),
data_nascita date not null,
nome_autore varchar(25) not null,
info varchar(255),
ruolo enum('esecutore', 'compositore', 'poliedrico')
);

create table immagine(
ID integer primary key auto_increment not null,
file varchar(255) not null,
didascalia varchar(1000),
ID_disco integer,
foreign key (ID_disco) references disco(ID) on update cascade on delete
cascade
);



/*-------- outer tables --------*/

create table condivide(
ID_collezione integer not null,
foreign key (ID_collezione) references collezione(ID) on update cascade
on delete cascade,
ID_collezionista integer not null,
foreign key (ID_collezionista) references collezionista(ID) on update
cascade on delete cascade
);

create table colleziona_dischi(
numero_duplicati integer not null,
ID_collezionista integer not null,
foreign key (ID_collezionista) references collezionista(ID) on update
cascade on delete cascade,
ID_disco integer not null,
foreign key (ID_disco) references disco(ID) on update cascade on delete
cascade
);

create table comprende_dischi(
```

```
ID_collezione integer not null,
foreign key (ID_collezione) references collezione(ID) on update cascade
on delete cascade,
ID_disco integer not null,
foreign key (ID_disco) references disco(ID) on update cascade on delete
cascade
);

create table produce_disco(
ID_disco integer not null,
foreign key (ID_disco) references disco(ID) on update cascade on delete
cascade,
ID_autore integer not null,
foreign key (ID_autore) references autore(ID) on update cascade on
delete cascade
);

create table realizza_traccia(
ID_traccia integer not null,
foreign key (ID_traccia) references traccia(ID) on update cascade on
delete cascade,
ID_autore integer not null,
foreign key (ID_autore) references autore(ID) on update cascade on
delete cascade
);
```

6.2 - Implementazione dei vincoli

Inizialmente erano stati realizzati dei triggers per la gestione dei duplicati dei dischi, in seguito tali triggers sono stati implementati in Java per cui sono stati rimossi dal database, tuttavia sono ancora visibili all'interno dello script SQL "**triggers.sql**", assieme ad altri due triggers tutt'ora funzionanti che controllano che l'anno di uscita del disco all'inserimento e alla modifica rispettino i vincoli esistenti.

Riportiamo di seguito questi due triggers:

```
drop trigger if exists Check_Anno_Uscita_Inserimento_Disco;
drop trigger if exists Check_Anno_Uscita_Aggiornamento_Disco;

create trigger Check_Anno_Uscita_Inserimento_Disco
before insert
on disco
for each row
begin
if NEW.anno_uscita > year(current_date) or NEW.anno_uscita < 1900 then
signal sqlstate '45000' set message_text = 'La data inserita non è
valida';
end if;
end$

create trigger Check_Anno_Uscita_Aggiornamento_Disco
before update
on disco
for each row
begin
if NEW.anno_uscita > year(current_date) or NEW.anno_uscita < 1900 then
signal sqlstate '45000' set message_text = 'La data inserita non è
valida';
end if;
end$
```

6.3 - Implementazione funzionalità richieste

Di seguito riportiamo l'implementazione di tutte le funzionalità richieste dalla specifica, esse sono state implementate tutte in MySQL tranne la tredicesima funzionalità che è stata implementata in Java. Oltre a tali funzionalità ne sono state implementate delle altre che reputiamo di minore importanza in quanto specifiche per il funzionamento di una piccola e semplice app che abbiamo realizzato per mostrare il funzionamento del database.

```
use collectors;
drop function if exists inserisci_collezione;
drop procedure if exists inserisci_disco_collezione;
drop procedure if exists inserisci_tracce_disco;
drop procedure if exists modifica_flag_collezione;
drop procedure if exists rimozione_disco_collezione;
drop procedure if exists rimozione_collezione;
drop procedure if exists lista_dischi_collezione;
drop procedure if exists tracklist_disco;
drop procedure if exists ricerca_dischi_per_titolo_autore;
drop view if exists lista_dischi;
drop function if exists verifica_visibilita_collezione;
drop procedure if exists gestione_disco;
drop procedure if exists ricerca_collezione;
drop function if exists
numero_tracce_distinte_per_autore_collezioni_pubbliche;
drop function if exists minuti_totali_musica_pubblica_per_autore;
drop procedure if exists statistiche_dischi_per_genere;
drop procedure if exists statistiche_numero_collezioni;
drop procedure if exists ricerca_dischi_per_autore_titolo;
drop view if exists lista_dischi_generale;
drop procedure if exists ricerca_dischi_per_titolo_autore;

delimiter $

-- Funzionalità 1
-- inserimento di una nuova collezione.
create function inserisci_collezione(nome varchar(25), flag boolean,
ID_collezionista integer)
    returns integer
    deterministic
begin
```

```
    declare id int unsigned;

    insert into collezione (nome, flag, ID_collezionista)
    values (nome, flag, ID_collezionista);
    set id = last_insert_id();

    return id;
    -- notare che possono essere create anche collezioni con stesso
nome
end$


-- Funzionalità 2
-- Aggiunta di dischi a una collezione e di tracce a un disco.

create procedure inserisci_disco_collezione(in ID_disco integer, in
ID_collezione integer)
begin
    insert into comprende_dischi(ID_disco, ID_collezione) values
(ID_disco, ID_collezione);
end$

create procedure inserisci_tracce_disco(in ID_disco integer, in
titolo varchar(35), in durata time)
begin
    insert into traccia(titolo, durata, ID_disco) values (titolo,
durata, ID_disco);
end$


-- Funzionalità 3
-- Modifica dello stato di pubblicazione di una collezione (da
privata a pubblica e viceversa) e aggiunta di nuove condivisioni a
una collezione.
create procedure modifica_flag_collezione(in ID_collezione integer,
in flag boolean)
begin
    update collezione set collezione.flag = flag where ID_collezione
= ID; -- flag = 0 - collezione privata | flag = 1 - collezione
```

```
pubblica
end$


-- Funzionalità 4
-- Rimozione di un disco da una collezione.
create procedure rimozione_disco_collezione(in ID_disco integer, in
ID_collezione integer)
begin
    delete from comprende_dischi c where c.ID_disco = ID_disco and
c.ID_collezione = ID_collezione;
end$

-- Funzionalità 5
-- Rimozione di una collezione.
create procedure rimozione_collezione(in ID_collezione integer)
begin
    delete from collezione where ID = ID_collezione;
end$

-- Funzionalità 6
-- Lista di tutti i dischi in una collezione

-- creo la view dei dischi
create view lista_dischi as
select disco.ID,
       titolo,
       anno_uscita    as 'anno di uscita',
       genere.nome    as genere,
       formato,
       stato_conservazione,
       descrizione_conservazione,
       barcode,
       etichetta.nome as azienda,
       sede_legale,
       email
from disco
         join etichetta on disco.ID_etichetta = etichetta.ID
         join genere on disco.ID_genere = genere.ID;
```

```
create procedure lista_dischi_collezione(in ID_collezione integer)
begin
    select id,
           titolo,
           lista_dischi.`anno di uscita`,
           barcode,
           formato,
           stato_conservazione,
           descrizione_conservazione
    from comprende_dischi
             join lista_dischi on ID_disco = lista_dischi.ID
    where comprende_dischi.ID_collezione = ID_collezione;
end$

-- Funzionalità 7
-- Tracklist di un disco
create procedure tracklist_disco(in ID_disco integer)
begin
    select id, titolo, durata
    from traccia
    where traccia.ID_disco = ID_disco;
end$

-- Funzionalità 8
/* Ricerca di dischi in base a nomi di autori/compositori/interpreti
e/o titoli. Si
potrà decidere di includere nella ricerca le collezioni di un certo
collezionista
e/o quelle condivise con lo stesso collezionista e/o quelle
pubbliche. */

create view lista_dischi_generale as
select c.nome, c1.nickname, d.*, a.nome_autore, c.flag,
ID_collezionista
from disco d
         join produce_disco pd on d.ID = pd.ID_disco
         join autore a on pd.ID_autore = a.ID
```

```
        join comprende_dischi cd on d.ID = cd.ID_disco
        join collezione c on cd.ID_collezione = c.ID
        join collezionista c1 on ID_collezionista = c1.ID;

-- lasciare null qualsiasi campo eccetto ID_collezionista per non
effettuare la ricerca su quel campo
create procedure ricerca_dischi_per_autore_titolo(in nome_autore
varchar(25), in titolo_disco varchar(50),
                                                  in pubbliche
boolean, in condivise boolean, in private boolean,
                                                  in ID_collezionista
int)
begin
    select distinct -- nickname as 'proprietario collezione',
                    ID,
                    titolo,
                    anno_uscita               as 'anno di uscita',
                    barcode,
                    formato,
                    stato_conservazione       as 'stato di
conservazione',
                    descrizione_conservazione as 'descrizione
conservazione'
                    -- lUl.nome_autore as 'nome autore'
    from (select *
          from lista_dischi_generale l
          where (
                      lower(l.titolo) regexp lower(titolo_disco) or
lower(l.nome_autore) regexp lower(nome_autore)
              )
            and l.ID_collezionista = ID_collezionista
            and private = 1
            -- ricerca nelle collezioni del collezionista

          union
          -- unione per ottenere entrambe le ricerche

          select *
          from lista_dischi_generale l
```

```
          where (
                      lower(l.titolo) regexp lower(titolo_disco) or
lower(l.nome_autore) regexp lower(nome_autore)
              )
            and flag = 1
            and pubbliche = 1
            -- ricerca nelle collezioni pubbliche

          union

          select ldg.*
          from collezione c
                   join condivide cd on c.ID = cd.ID_collezione
                   join comprende_dischi d on c.ID = d.ID_collezione
                   join lista_dischi_generale ldg on d.ID_disco =
ldg.ID
          where (
                      lower(ldg.titolo) regex lower(titolo_disco) or
lower(ldg.nome_autore) regex lower(nome_autore)
              )
            and cd.ID collezionista = IL collezionista
            and condivise = 1 -- ricerca nelle condivise
        ) as `lUl`;
end$

-- Funzionalità 9
-- verifica della visibilità di una collezione da parte di un
collezionista

create function verifica_visibilita_collezione(ID_collezione integer,
ID_collezionista integer)
    returns boolean
    deterministic
begin
    if (select flag from collezione where collezione.ID =
ID_collezione) = 1 then -- se la collezione ha flag impostato a true
allora la collezione è visibile a tutti i collezionisti
        return true;
    end if;
```

```
    if (select ID_collezionista
        from collezione
        where collezione.ID = ID_collezione
          and collezione.ID_collezionista = ID_collezionista) then --
se la collezione appartiene al collezionista, allora può visionarla
        return true;
    end if;
    if (select count(*)
        from condivide c
        where c.ID_collezione = ID_collezione
          and c.ID_collezionista = ID_collezionista) = 1 then
        return true;
    end if;
    return false;
end$


-- Funzionalità 10
-- numero di tracce di dischi distinti di un certo autore presenti
nelle collezioni pubbliche

create function
numero_tracce_distinte_per_autore_collezioni_pubbliche(ID_autore
integer)
    returns integer
    deterministic
begin
    return (select count(distinct traccia.ID)
            from traccia
                     join disco on traccia.ID_disco = disco.ID
                     join produce_disco on disco.ID =
produce_disco.ID_disco
                     join autore on produce_disco.ID_autore =
autore.ID
                     join comprende_dischi on disco.ID =
comprende_dischi.ID_disco
                     join collezione on
comprende_dischi.ID_collezione = collezione.ID
            where lower(autore.nome_autore) = lower(nome_autore)
```

```
              and collezione.flag = 1);
end$


-- Funzionalità 11
-- Minuti totali di musica riferibili a un certo autore memorizzati
nelle collezioni pubbliche
create function minuti_totali_musica_pubblica_per_autore(ID_autore
integer)
    returns integer
    deterministic
begin
    return (select sum(traccia.durata) / 60
            from traccia
                     join disco on traccia.ID_disco = disco.ID
                     join produce_disco on disco.ID =
produce_disco.ID_disco
                     join autore on produce_disco.ID_autore =
autore.ID
                     join comprende_dischi on disco.ID =
comprende_dischi.ID_disco
                     join collezione on
comprende_dischi.ID_collezione = collezione.ID
            where autore.ID = ID_autore
              and collezione.flag = 1);
end$

-- Funzionalità 12

-- 12a
-- statistiche: numero collezioni di ciascun collezionista

create procedure statistiche_numero_collezioni()
begin
    select collezionista.*, count(all collezione.ID)
    from collezione
             join collezionista on collezione.ID_collezionista =
collezionista.ID
    group by nickname;
```

```
end$


-- 12b
-- statistiche: numero di dischi per genere nel sistema

create procedure statistiche_dischi_per_genere()
begin
    select genere.*, count(all disco.id)
    from disco
             join genere on disco.ID_genere = genere.ID
    group by nome;
end$

-- 13
-- la funzionalità 13 è stata implementata in Java, nella classe
Query_JDBC.
```

```
// Funzionalità 13
/* dati il barcode, il titolo del disco e il nome dell'autore, si
individuino i dischi più coerenti con questi dati, per implementare ciò
abbiamo deciso di attribuire dei pesi alla ricerca dei dischi, dando
maggior importanza al peso dell’immissione o meno del barcode*/

public HashMap<Disco, Integer> dischiSimiliA(String barcode, String
titolo, String autore) {
HashMap<Disco, Integer> dischi = new HashMap<>();
try {
PreparedStatement queryBarcode = connection.prepareStatement("select *
from disco where barcode regexp ?");
PreparedStatement queryTitolo = connection.prepareStatement("select *
from disco where titolo regexp ?");
PreparedStatement queryAutore = connection.prepareStatement(
"select d.* from " +
"disco d join produce_disco p on d.ID = p.ID_disco " +
"join autore a on p.ID_autore=a.ID " +
"where nome_autore regexp ?"
);
```

```
queryBarcode.setString(1, RicercaController.regexify(barcode));
queryTitolo.setString(1, RicercaController.regexify(titolo));
queryAutore.setString(1, RicercaController.regexify(autore));



ResultSet barcodeResult = queryBarcode.executeQuery();
ResultSet titoloResult = queryTitolo.executeQuery();
ResultSet autoreResult = queryAutore.executeQuery();

while (barcodeResult.next()) {
dischi.put(
new Disco(
barcodeResult.getInt(1), //ID
barcodeResult.getString(2), // titolo
barcodeResult.getInt(3), //anno di uscita
barcodeResult.getString(4), //barcode
barcodeResult.getString(5), //formato
barcodeResult.getString(6), // stato di conservazione
barcodeResult.getString(7) // descrizione conservazione
), 3
);
}

while (titoloResult.next()) {
Disco disco = new Disco(

titoloResult.getInt(1), //ID
titoloResult.getString(2), // titolo
titoloResult.getInt(3), //anno di uscita
titoloResult.getString(4), //barcode
titoloResult.getString(5), //formato
titoloResult.getString(6), // stato di conservazione
titoloResult.getString(7) // descrizione conservazione
);

if (dischi.containsKey(disco))
dischi.put(disco, dischi.get(disco) + 1);
else
dischi.put(disco, 1);
}

while (autoreResult.next()) {
Disco disco = new Disco(
autoreResult.getInt(1), //ID
```

```
autoreResult.getString(2), // titolo
autoreResult.getInt(3), //anno di uscita
autoreResult.getString(4), //barcode
autoreResult.getString(5), //formato
autoreResult.getString(6), // stato di conservazione
autoreResult.getString(7) // descrizione conservazione
);
if (dischi.containsKey(disco))
dischi.put(disco, dischi.get(disco) + 1);
else
dischi.put(disco, 1);
}

queryBarcode.close();
queryTitolo.close();
queryAutore.close();

} catch (SQLException sqlException) {
sqlException.printStackTrace();
}
return dischi;
}
```

7 - Interfaccia verso il database

Abbiamo deciso di realizzare una semplice interfaccia grafica in Java (la struttura del progetto è in Maven) con l'ausilio di JavaFX in modo tale da illustrare un possibile scenario di funzionamento del database.

Purtroppo non tutte le funzionalità implementate al suo interno sono state collocate in modo tale da permettere un utilizzo omogeneo dell'applicazione, ma i requisiti funzionali di maggiore impatto (quelli richiesti dalla specifica) sono nel giusto contesto all'interno dell'app.

Forniamo di seguito un nickname ed un indirizzo e-mail per potersi identificare all'interno del sistema e provare le varie funzionalità:

Nickname: **Diana** - E-mail: **rossidiana@gmail.com**
