Traccia

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.

Laboratorio di Basi di Dati: *Progetto Collectors*

Gruppo di lavoro:

Matricola	Nome	Cognome	Contributo al progetto
279061	Enrico	Menichelli	Si

Data di consegna del progetto: 26/07/2023

Analisi della traccia e dominio

Di seguito sono riportate le entità scoperte a seguito dell'analisi della traccia.
Il compito del progetto è quello di permettere a **Collezionisti** di tenere traccia delle proprie collezioni di dischi, che possono essere anche condivise con altri. Ogni collezione contiene una lista di dischi, dove ogni disco posseduto da un collezionista specifica il proprio formato, lo stato di conservazione e il numero di copie che possiede. Ogni disco contiene varie informazioni, immagini e tracce.

- **Collezionista**: Un utente identificato tramite il proprio nickname e una mail, possiede varie collezioni e fa parte di collezioni di altri collezionisti
- **Collezione**: Una lista di dischi appartenenti a questa collezione, incluso il nome della collezione e se è pubblica o meno. La collezione può essere condivisa con altri collezionisti.
- **Disco**: Contiene i dettagli del disco come il nome, l'autore, il titolo, l'anno di uscita, l'etichetta, il genere, lo stato di conservazione, il formato, il barcode se disponibile, una lista di tracce, e delle immagini
- **Immagine**: L'immagine di un disco e la sua tipologia
- **Traccia**: Una canzone, contiene un titolo, durata, e opzionalmente informazioni sul contributo da parte di artisti.
- **Artista**: L'artista che produce un disco o collabora in una canzone, con il proprio nome d'arte

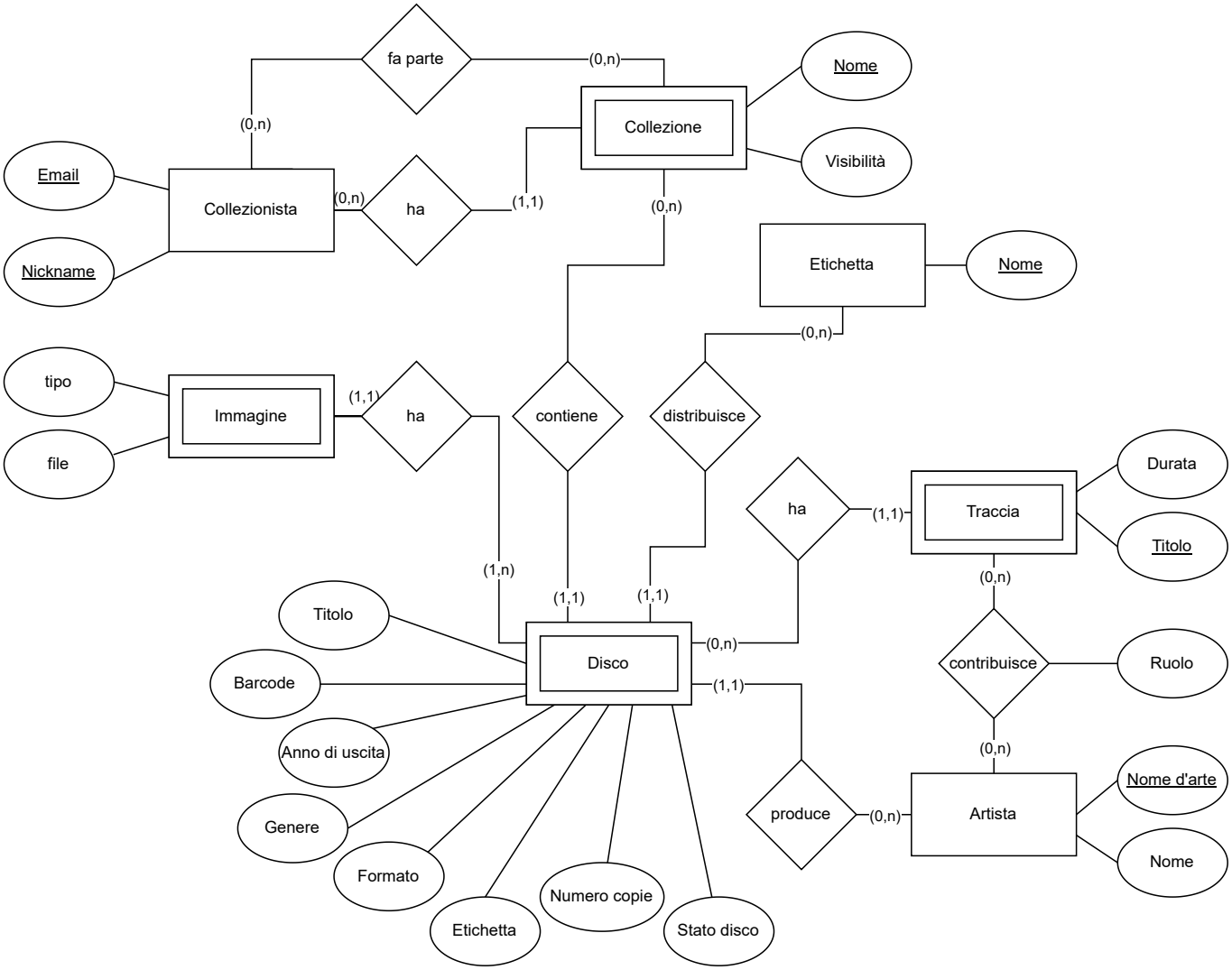
Implementazioni analizzate

Sono state analizzate due implementazioni diverse, una più restrittiva e una più permissiva:

L'implementazione più **restrittiva** permette ai collezionisti di creare un database univoco e condiviso da tutti, un disco (e tutti i suoi attributi strutturati) specifico è quindi rappresentato in maniera univoca, e ogni possedimento da parte di un collezionista ne rappresenta solo il formato e lo stato del disco.
Ha il pro di rimuovere eventuale ridondanza nel database, ma ha il contro di non poter essere modificabile da parte dei collezionisti, infatti saranno obbligati ad usare le informazioni sul disco già presente, o crearne uno nuovo con simili informazioni, quindi annullando i vantaggi precedenti.

L'implementazione più **permissiva** (quella scelta) permette la ridondanza dei dati dei dischi, tracce, immagini, etc..., a favore di un possesso di dati da parte dei collezionisti, infatti, essendo tutti i dischi potenzialmente diversi da quelli già presenti, permette ad ogni collezionista di modificare a piacimento il proprio disco, e con funzionalità da parte dell'interfaccia grafica, si facilita l'inserimento di dischi già esistenti, "clonandoli".

Progettazione concettuale

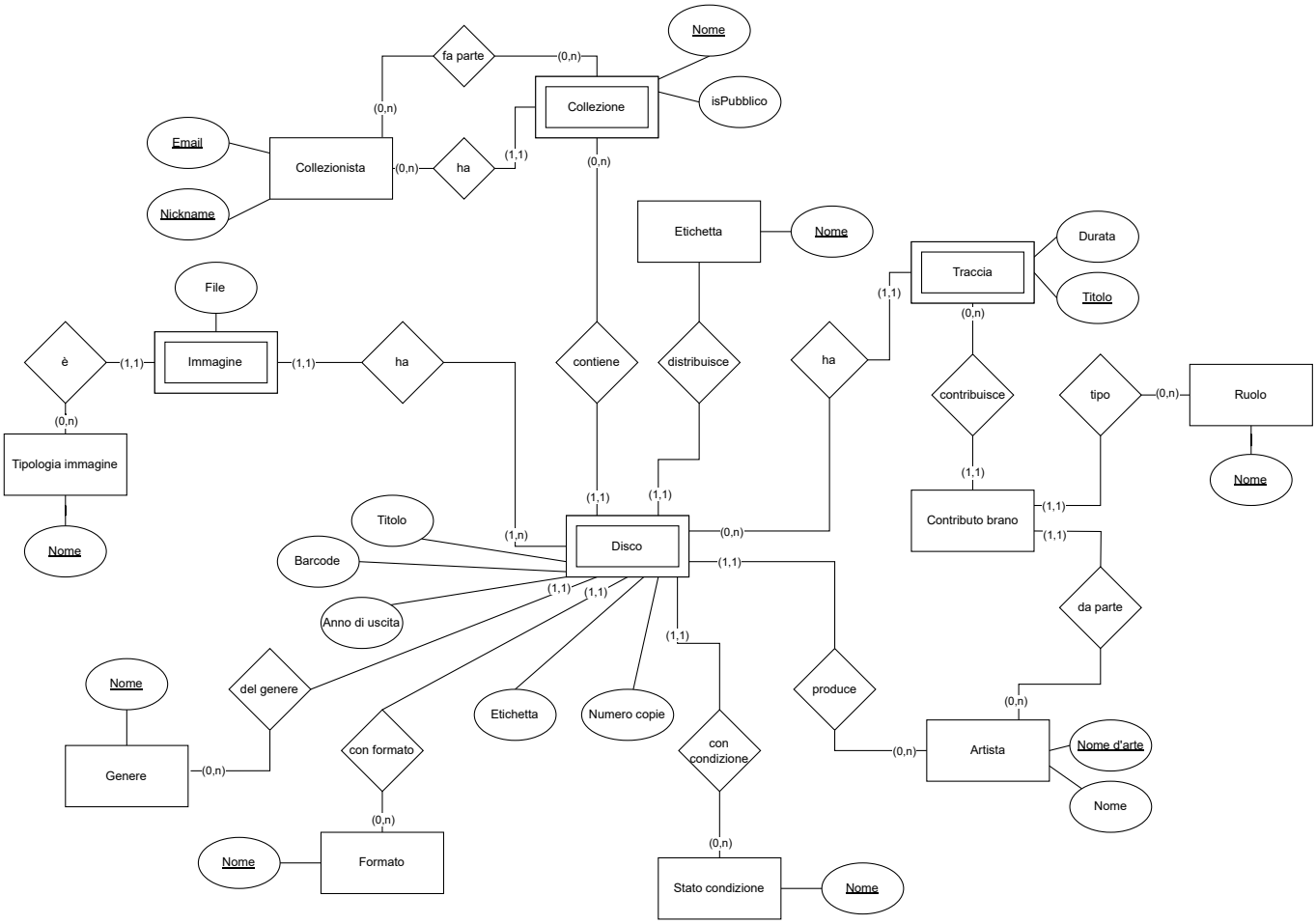


Formalizzazione dei vincoli non esprimibili nel modello ER

Tutti gli attributi sono NOT NULL ad eccezione di "barcode" che può esserlo in caso che non sia conosciuto. In oltre nelle entità:

- **Collezionista**: Sia email che nickname sono unici
- **Collezione**: Il nome di una collezione è unico sul singolo collezionista e una collezione non può essere condivisa con se stessi

Progettazione logica



- Sono state create delle nuove entità a partire dagli attributi **genere**, **stato**, **ruolo**, **tipologia**, **formato**, per rendere più semplice l'aggiunta di nuovi, la eventuale modifica del nome e l'aggiunta di più informazioni se necessario.
- Il contributo da parte di un artista è stato generalizzato in una nuova entità "Contributo brano" che specifica il contributo di un artista tramite un certo ruolo (tipo cantante, scrittore, chitarrista etc...)
- La visibilità di una collezione è stata modificata per un valore booleano "isPubblico"
- Anche se abbastanza grande, non si è scelto la suddivisione della entità "Disco" dato che tutte le informazioni del disco verranno sempre usate nelle query.

Traduzione del modello ER nel modello relazionale

- **Collezionista**(ID, username, email)
- **Collezione**(ID,nome, *IDCollezionista*, isPubblico)
- **CollezioneCondivisa**(*IDCollezione*, *IDCollezionista*)
- **Etichetta**(ID, nome)
- **Traccia**(ID, *IDDisco*, durata, titolo)
- **Artista**(ID, nomeArte, nome)
- **Immagine**(ID, *IDBrano*, file, *tipoImmagine*)

- **ContributoBrano**(*IDBrano*, *tipoContributo*, *artista*)
- **Disco**(ID, *IDCollezione*, titolo, barcode, annoDiUscita, numeroCopie, *genere*, *formato*, *IDEtichetta*, *statoDisco*, *IDArtista*)
- **Formato**(nome)
- **Genere**(nome)
- **TipologiaImmagine**(nome)
- **RuoloArtista**(nome)
- **StatoCondizione**(nome)

Progettazione fisica

Implementazione del modello relazionale

```
CREATE TABLE IF NOT EXISTS condition_status(
    condition_name VARCHAR(40) PRIMARY KEY
);
CREATE TABLE IF NOT EXISTS artist_role(
    role_name VARCHAR(40) PRIMARY KEY
);
CREATE TABLE IF NOT EXISTS image_type(
    type_name VARCHAR(40) PRIMARY KEY
);
CREATE TABLE IF NOT EXISTS disc_genre(
    genre_name VARCHAR(40) PRIMARY KEY
);
CREATE TABLE IF NOT EXISTS disc_format(
    format_name VARCHAR(40) PRIMARY KEY
);
CREATE TABLE IF NOT EXISTS artist(
    id INT AUTO_INCREMENT PRIMARY KEY,
    stage_name VARCHAR(100) UNIQUE NOT NULL,
    artist_name VARCHAR(100) NOT NULL
);
CREATE TABLE IF NOT EXISTS label(
    id INT AUTO_INCREMENT PRIMARY KEY,
    label_name VARCHAR(100) UNIQUE NOT NULL
);
CREATE TABLE IF NOT EXISTS collector(
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(100) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL
);
CREATE TABLE IF NOT EXISTS collection(
    id INT AUTO_INCREMENT PRIMARY KEY,
```

```
    collection_name VARCHAR(100) NOT NULL, #collectin name is unique for the owner
    collector_id INT NOT NULL,
    is_public BOOLEAN NOT NULL,

    FOREIGN KEY (collector_id) REFERENCES collector(id)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
CREATE TABLE IF NOT EXISTS disc(
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    barcode VARCHAR(50),
    release_year INT NOT NULL,
    number_of_copies INT NOT NULL,
    genre VARCHAR(40) NOT NULL,
    disc_format VARCHAR(40) NOT NULL,
    label_id INT NOT NULL,
    collection_id INT NOT NULL,
    disc_status varchar(40) NOT NULL,
    artist_id INT NOT NULL,

    FOREIGN KEY (artist_id) REFERENCES artist(id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    FOREIGN KEY (disc_status) REFERENCES condition_status(condition_name)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    FOREIGN KEY (collection_id) REFERENCES collection(id)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (label_id) REFERENCES label(id)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    FOREIGN KEY (disc_format) REFERENCES disc_format(format_name)
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    FOREIGN KEY (genre) REFERENCES disc_genre(genre_name)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
);
CREATE TABLE IF NOT EXISTS image(
    id INT AUTO_INCREMENT PRIMARY KEY,
    image_path VARCHAR(200) NOT NULL,
    image_format VARCHAR(40),
    disc_id INT NOT NULL,
```

```
    FOREIGN KEY (disc_id) REFERENCES disc(id)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    FOREIGN KEY (image_format) REFERENCES image_type(type_name)
        ON UPDATE CASCADE
        ON DELETE RESTRICT
);
CREATE TABLE IF NOT EXISTS shared_collection(
    collection_id INT NOT NULL,
    collector_id INT NOT NULL,

    FOREIGN KEY (collection_id) REFERENCES collection(id)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    FOREIGN KEY (collector_id) REFERENCES collector(id)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    PRIMARY KEY (collection_id, collector_id)
);
CREATE TABLE IF NOT EXISTS track(
    id INT AUTO_INCREMENT PRIMARY KEY,
    track_length INT NOT NULL,
    title VARCHAR(100) NOT NULL,
    disc_id INT NOT NULL,

    FOREIGN KEY (disc_id) REFERENCES disc(id)
        ON UPDATE CASCADE
        ON DELETE CASCADE
);
CREATE TABLE IF NOT EXISTS track_contribution(
    track_id INT NOT NULL,
    artist_id INT NOT NULL,
    contribution_type VARCHAR(40) NOT NULL,

    FOREIGN KEY (track_id) REFERENCES track(id)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    FOREIGN KEY (artist_id) REFERENCES artist(id)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    FOREIGN KEY (contribution_type) REFERENCES artist_role(role_name)
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    PRIMARY KEY (track_id, artist_id, contribution_type)
);
```

Script per aggiunta di dati al database:

```
INSERT INTO condition_status (condition_name) VALUES
    ('New'),
    ('Good'),
    ('Scratched'),
    ('Damaged');

INSERT INTO artist_role (role_name) VALUES
    ('Lead Singer'),
    ('Guitarist'),
    ('Bassist'),
    ('Drummer'),
    ('Keyboardist'),
    ('Producer'),
    ('Writer');

INSERT INTO image_type (type_name) VALUES
    ('Front'),
    ('Back'),
    ('Disc'),
    ('Booklet'),
    ('Other');

INSERT INTO disc_genre (genre_name) VALUES
    ('Rock'),
    ('Pop'),
    ('Metal'),
    ('Jazz'),
    ('Hip Hop'),
    ('Electronic'),
    ('Classical'),
    ('Country'),
    ('Indie'),
    ('Folk'),
    ('Rap');

INSERT INTO disc_format (format_name) VALUES
    ('Vinyl'),
    ('CD'),
    ('Cassette'),
    ('Digital');

INSERT INTO artist (stage_name, artist_name) VALUES
    ('Freddie Mercury', 'Freddie Mercury'),
    ('John Lennon', 'John Lennon'),
```

```
    ('Michael Jackson', 'Michael Jackson'),
    ('Elvis', 'Elvis Presley'),
    ('David Bowie', 'David Bowie'),
    ('Madonna', 'Madonna Louise Veronica');

INSERT INTO label (label_name) VALUES
    ('EMI'),
    ('Sony Music'),
    ('Warner Bros'),
    ('Universal Music'),
    ('Atlantic Records'),
    ('Columbia Records');

INSERT INTO collector (username, email) VALUES
    ('test-user', 'test-user@example.com'),
    ('vinyljunkie', 'vinyljunkie@example.com'),
    ('cdcollector', 'cdcollector@example.com'),
    ('metalhead88', 'metalhead88@example.com');

INSERT INTO collection (collection_name, collector_id, is_public) VALUES
    ('My Vinyl Collection', 1, 0),
    ('Favorite CDs', 3, 0),
    ('Metal Albums', 4, 1),
    ('Classic Rock', 2, 1),
    ('Madonna', 2, 0);

INSERT INTO shared_collection (collection_id, collector_id) VALUES
    (1, 2),
    (2, 1),
    (3, 3),
    (4, 1);

INSERT INTO disc (title, barcode, release_year, number_of_copies, genre,
disc_format, label_id, collection_id, disc_status, artist_id) VALUES
    ('Bohemian Rhapsody', '1234567890', 1975, 1, 'Rock', 'Vinyl', 1, 1, 'New', 1),
    ('Thriller', '9876543210', 1982, 15, 'Pop', 'Vinyl', 2, 2, 'Good', 3),
    ('Black Album', '4567890123', 1991, 3, 'Metal', 'CD', 3, 3, 'Scratched', 4),
    ('Sgt. Pepper', '1111111111', 1967, 14, 'Rock', 'Vinyl', 4, 4, 'Good', 2),
    ('Like a Virgin', '2222222222', 1984, 8, 'Pop', 'CD', 5, 2, 'New', 6),
    ('Space Oddity', '3333333333', 1969, 4, 'Rock', 'Vinyl', 6, 4, 'Damaged', 5);

INSERT INTO image (image_path, image_format, disc_id) VALUES
    ('https://picsum.photos/500/500', 'Front', 1),
    ('https://picsum.photos/500/500', 'Back', 1),
    ('https://picsum.photos/500/500', 'Front', 2),
    ('https://picsum.photos/500/500', 'Back', 2),
```

```
    ('https://picsum.photos/500/500', 'Front', 3),
    ('https://picsum.photos/500/500', 'Back', 4),
    ('https://picsum.photos/500/500', 'Front', 4),
    ('https://picsum.photos/500/500', 'Back', 5);

INSERT INTO track (track_length, title, disc_id) VALUES
    (355, 'Bohemian Rhapsody', 1),
    (398, 'Another One Bites the Dust', 1),
    (402, 'Thriller', 2),
    (320, 'Beat It', 2),
    (500, 'Enter Sandman', 3),
    (312, 'Sad but True', 3),
    (177, 'Sgt. Pepper', 4),
    (52, 'With a Little Help from My Friends', 4),
    (389, 'Like a Virgin', 5),
    (322, 'Material Girl', 5),
    (312, 'Space Oddity', 6),
    (237, 'Starman', 6);

INSERT INTO track_contribution (track_id, artist_id, contribution_type) VALUES
    (1, 1, 'Lead Singer'),
    (1, 2, 'Guitarist'),
    (2, 1, 'Lead Singer'),
    (2, 2, 'Guitarist'),
    (3, 3, 'Lead Singer'),
    (4, 3, 'Lead Singer'),
    (5, 4, 'Lead Singer'),
    (5, 4, 'Guitarist'),
    (6, 4, 'Lead Singer'),
    (6, 4, 'Guitarist'),
    (7, 2, 'Lead Singer'),
    (7, 2, 'Guitarist'),
    (8, 2, 'Lead Singer'),
    (8, 2, 'Guitarist'),
    (9, 5, 'Lead Singer'),
    (10, 5, 'Lead Singer'),
    (11, 6, 'Lead Singer'),
    (12, 6, 'Lead Singer');
```

Implementazione dei vincoli

```
CREATE TRIGGER check_collection_uniqueness BEFORE INSERT ON collection FOR EACH ROW
BEGIN
    DECLARE number_of_collections INT;
    DECLARE collector_name varchar(100);
    SET exists_collection = (
```

```
        SELECT COUNT(*)
        FROM collection c
        WHERE c.collector_id = NEW.collector_id AND c.collection_name =
NEW.collection_name
    );
    IF (number_of_collections > 0) THEN
        SET collector_name = (
            SELECT c.username
            FROM collector c
            WHERE c.id = NEW.collector_id
        );
        SET error_message = CONCAT("Collection: ", NEW.collection_name, " Already
exists for collector: ", collector_name);
        SIGNAL SQLSTATE "45000" SET MESSAGE_TEXT = error_message;
    END IF;
END$

CREATE TRIGGER check_collection_share BEFORE INSERT ON shared_collection FOR EACH
ROW
BEGIN
    DECLARE owner_id INT;
    SET owner_id = (
        SELECT c.collector_id
        FROM collection c
        WHERE c.id = NEW.collection_id
    );
    IF (owner_id = NEW.collector_id) THEN
        SET error_message = CONCAT("Collector: ", NEW.collector_id, " is already
the owner of collection: ", NEW.collection_id);
        SIGNAL SQLSTATE "45000" SET MESSAGE_TEXT = error_message;
    END IF;
END$
```

Implementazione funzionalità richieste

Funzionalità 1

> Inserimento di una nuova collezione.

```
CREATE PROCEDURE create_collection(
    IN collection_name VARCHAR(100),
    IN collector_id INT,
    IN is_public BOOLEAN
)
BEGIN
    INSERT INTO collection(collection_name, collector_id, is_public)
```

```
    VALUES (collection_name, collector_id, is_public);
    SELECT LAST_INSERT_ID() AS collection_id;
END$
```

Funzionalità 2

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

```
CREATE PROCEDURE create_disc(
    IN title VARCHAR(100),
    IN barcode VARCHAR(50),
    IN release_year INT,
    IN number_of_copies INT,
    IN genre VARCHAR(40),
    IN disc_format VARCHAR(40),
    IN label_id INT,
    IN collection_id INT,
    IN disc_status VARCHAR(40),
    IN artist_id INT
)
BEGIN
    INSERT INTO disc(title, barcode, release_year, number_of_copies, genre,
disc_format, label_id, collection_id, disc_status, artist_id)
    VALUES (title, barcode, release_year, number_of_copies, genre, disc_format,
label_id, collection_id, disc_status, artist_id);
    SELECT LAST_INSERT_ID() AS disc_id;
END$
-- ------------------------------------------------
CREATE PROCEDURE create_track(
    IN track_length INT,
    IN title VARCHAR(100),
    IN disc_id INT
)
BEGIN
    INSERT INTO track(track_length, title, disc_id)
    VALUES (track_length, title, disc_id);
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 set_collection_visibility(
    IN collection_id INT,
```

```
    IN is_public BOOLEAN
)
BEGIN
    UPDATE collection c
    SET c.is_public = is_public
    WHERE c.id = collection_id;
END$
-- ------------------------------------------------
CREATE PROCEDURE add_contributor_to_collection(
    IN collection_id INT,
    IN collector_username VARCHAR(100)
)
BEGIN
    DECLARE collector_id INT;
    DECLARE error_message VARCHAR(200);
    SET collector_id = (
        SELECT c.id
        FROM collector c
        WHERE c.username = collector_username
    );

    IF (collector_id IS NULL) THEN
        SET error_message = CONCAT("Collector: ", collector_username, " does not
exist");
        SIGNAL SQLSTATE "45000" SET MESSAGE_TEXT = error_message;
    END IF;

    INSERT INTO shared_collection(collection_id, collector_id)
    VALUES (collection_id, collector_id);

END$
```

Funzionalità 4

> Rimozione di un disco da una collezione.

```
CREATE PROCEDURE remove_disc_from_collection(
    IN disc_id INT
)
BEGIN
    DELETE FROM disc d
    WHERE d.id = disc_id;
END$
```

Funzionalità 5

> Rimozione di una collezione

```
CREATE PROCEDURE remove_collection(
    IN collection_id INT
)
BEGIN
    DELETE FROM collection c
    WHERE c.id = collection_id;
END$
```

Funzionalità 6

> Lista di tutti i dischi in una collezione

```
CREATE PROCEDURE get_discs_of_collection(
    IN collection_id INT
)
BEGIN
    SELECT
        d.id AS disc_id,
        d.title AS disc_title,
        d.barcode AS disc_barcode,
        d.release_year AS disc_release_year,
        d.number_of_copies AS disc_number_of_copies,
        d.genre AS disc_genre,
        d.disc_format AS disc_format,
        d.disc_status AS disc_status,
        a.stage_name AS artist_stage_name,
        l.label_name AS label_name,
        d.collection_id AS collection_id,
        l.id AS label_id,
        a.id AS artist_id
    FROM disc d
    JOIN artist a ON d.artist_id = a.id
    JOIN label l ON d.label_id = l.id
    WHERE d.collection_id = collection_id;
END$
```

Funzionalità 7

> Track list di un disco

```
CREATE PROCEDURE get_disc_tracks(
    IN disc_id INT
)
```

```
BEGIN
    SELECT
        t.id AS track_id,
        t.track_length AS track_length,
        t.title AS track_title,
        t.disc_id AS disc_id
    FROM track t
    WHERE t.disc_id = disc_id;
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. *(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)*

```
CREATE PROCEDURE search_discs(
    IN search_disc_title VARCHAR(100),
    IN search_artist_stage_name VARCHAR(100),
    IN collector_id INT,
    IN search_in_owned_collections BOOLEAN,
    IN search_in_shared_collections BOOLEAN,
    IN search_in_public_collections BOOLEAN
)
BEGIN
    DECLARE artist_id INT;
    SET artist_id = (
        SELECT a.id
        FROM artist a
        WHERE a.stage_name LIKE search_artist_stage_name
    );

    SELECT DISTINCT
        d.id AS disc_id,
        d.title AS disc_title,
        d.barcode AS disc_barcode,
        d.release_year AS disc_release_year,
        d.number_of_copies AS disc_number_of_copies,
        d.genre AS disc_genre,
        d.disc_format AS disc_format,
        d.disc_status AS disc_status,
        a.stage_name AS artist_stage_name,
        l.label_name AS label_name,
```

```
        d.collection_id AS collection_id,
        l.id AS label_id,
        a.id AS artist_id
    FROM disc d
    JOIN artist a ON d.artist_id = a.id
    JOIN label l ON d.label_id = l.id
    JOIN collection c ON d.collection_id = c.id
    LEFT JOIN shared_collection sc ON d.collection_id = sc.collection_id AND
sc.collector_id = collector_id
    WHERE
        (d.title = COALESCE(search_disc_title, d.title)) AND
        (d.artist_id = COALESCE(artist_id, d.artist_id)) AND
        (
            (search_in_owned_collections AND c.collector_id = collector_id) OR
            (search_in_shared_collections AND sc.collector_id = collector_id) OR
            (search_in_public_collections AND c.is_public = TRUE)
        );
END$
```

Funzionalità 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)*

```
CREATE FUNCTION is_collection_visible_by_collector(
    collection_id INT,
    collector_id INT
)
RETURNS BOOLEAN DETERMINISTIC
BEGIN
    DECLARE is_visible BOOLEAN;
    SET is_visible = (SELECT
        CASE
            WHEN
                c.is_public OR
                c.collector_id = collector_id OR
                EXISTS (
                    SELECT *
                    FROM shared_collection sc
                    WHERE sc.collection_id = c.id AND sc.collector_id =
collector_id
                )
            THEN TRUE
            ELSE FALSE
        END
    FROM collection c
```

```
    WHERE c.id = collection_id);
    IF (is_visible IS NULL) THEN
        RETURN FALSE;
    ELSE
        RETURN is_visible;
    END IF;
END$
```

Funzionalità 10

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

```
CREATE FUNCTION get_artist_id(
    artist_stage_name VARCHAR(100)
)
RETURNS INT DETERMINISTIC
BEGIN
    DECLARE artist_id INT;
    SET artist_id = (
        SELECT a.id
        FROM artist a
        WHERE a.stage_name LIKE artist_stage_name
    );
    RETURN artist_id;
END$
-- ------------------------------------------------
CREATE FUNCTION count_tracks_of_author_in_public_collections(
    artist_stage_name VARCHAR(100)
)
RETURNS INT DETERMINISTIC
BEGIN
    DECLARE artist_id INT;
    DECLARE number_of_tracks INT;
    DECLARE error_message VARCHAR(200);
    SET artist_id = get_artist_id(artist_stage_name);
    IF (artist_id IS NULL) THEN
        SET error_message = CONCAT("Artist: ", artist_stage_name, " does not
exist");
        SIGNAL SQLSTATE "45000" SET MESSAGE_TEXT = error_message;
    END IF;

    SET number_of_tracks = (
        SELECT COUNT(DISTINCT t.id)
        FROM track t
        JOIN track_contribution tc ON t.id = tc.track_id
```

```
        JOIN disc d ON t.disc_id = d.id
        JOIN collection c ON d.collection_id = c.id
        WHERE c.is_public = TRUE AND tc.artist_id = artist_id
    );
    IF (number_of_tracks IS NULL) THEN
        RETURN 0;
    ELSE
        RETURN number_of_tracks;
    END IF;
END$
```

Funzionalità 11

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

```
CREATE FUNCTION count_total_track_time_of_artist_in_public_collections(
    artist_stage_name VARCHAR(100)
)
RETURNS INT DETERMINISTIC
BEGIN
    DECLARE artist_id INT;
    DECLARE total_track_time INT;
    DECLARE error_message VARCHAR(200);
    SET artist_id = get_artist_id(artist_stage_name);
    IF (artist_id IS NULL) THEN
        SET error_message = CONCAT("Artist: ", artist_stage_name, " does not
exist");
        SIGNAL SQLSTATE "45000" SET MESSAGE_TEXT = error_message;
    END IF;

    SET total_track_time = (
        SELECT SUM(track_length)
        FROM (
            SELECT DISTINCT t.id, t.track_length
            FROM track t
            JOIN track_contribution tc ON t.id = tc.track_id
            JOIN disc d ON t.disc_id = d.id
            JOIN collection c ON d.collection_id = c.id
            WHERE c.is_public = TRUE AND tc.artist_id = artist_id
        ) AS unique_tracks
    );
    IF (total_track_time IS NULL) THEN
        RETURN 0;
    ELSE
        RETURN total_track_time;
```

```
    END IF;
END$
```

Funzionalità 12

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

```
CREATE PROCEDURE aggregate_number_of_collections_per_collector()
BEGIN
    SELECT
        collector.username AS collector_username,
        COUNT(c.id) AS number_of_collections
    FROM collection c
    RIGHT JOIN collector ON collector.id = c.collector_id
    GROUP BY collector.username;
END$
-- ------------------------------------------------
CREATE PROCEDURE aggregate_number_of_discs_per_genre()
BEGIN
    SELECT
        g.genre_name AS genre,
        COUNT(d.id) AS number_of_discs
    FROM disc d
    RIGHT JOIN disc_genre g ON g.genre_name = d.genre
    GROUP BY g.genre_name;
END$
```

Funzionalità 13

> 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.

```
CREATE PROCEDURE find_best_match_of_disc_from(
    IN barcode VARCHAR(50),
    IN title VARCHAR(100),
    IN artist_stage_name VARCHAR(100)
)
BEGIN
    SELECT
```

```
        d.id AS disc_id,
        d.title AS disc_title,
        d.barcode AS disc_barcode,
        d.release_year AS disc_release_year,
        d.number_of_copies AS disc_number_of_copies,
        d.genre AS disc_genre,
        d.disc_format AS disc_format,
        d.disc_status AS disc_status,
        a.stage_name AS artist_stage_name,
        l.label_name AS label_name,
        d.collection_id AS collection_id,
        l.id AS label_id,
        a.id AS artist_id
    FROM disc d
    JOIN artist a ON d.artist_id = a.id
    JOIN label l ON d.label_id = l.id
    WHERE d.barcode = barcode OR d.title LIKE CONCAT('%', title, '%')
    ORDER BY CASE
        WHEN d.barcode = barcode THEN 1
        WHEN d.title LIKE CONCAT('%', title, '%') AND a.stage_name NOT LIKE
CONCAT('%', artist_stage_name, '%') THEN 2
        WHEN d.title LIKE CONCAT('%', title, '%') AND a.stage_name LIKE CONCAT('%',
artist_stage_name, '%') THEN 3
        ELSE 4
    END
    LIMIT 25;
END$
```

Interfaccia verso il database

è stata creata un interfaccia grafica che implementa le query e le procedure descritte in precedenza tramite l'uso del linguaggio Typescript, Svelte (un framework/linguaggio per la creazione di applicazioni web) e electron con node.js per la creazione di un'applicazione desktop.

Esecuzione da sorgente

Il codice sorgente può essere scaricato su questa repository github incluso anche il sorgente markdown di questo documento, i diagrammi e gli script sql
Si assume una connessione mysql con le seguenti credenziali:

```
export const DEFAULT_CONNECTION = {
    host: "localhost",
    port: 3306,
```

```
    user: "root",
    password: "root",
} satisfies ConnectionOptions
```

può essere in caso modificata nel file `GUI/electron/src/db/db.ts`.
L'applicazione effettuerà da sola la creazione del database `collectors` con tutte le sue tabelle, procedure e funzioni, per poi popolarlo con i dati di esempio.
Tutti gli script sql che verranno eseguiti possono essere trovati in `GUI/electron/src/db/sql-scripts/` e sono eseguiti all'interno del file `GUI/electron/src/db/collectors-db.ts` dove si possono trovare anche tutte le funzioni di interfaccia con il database e conversione dei risultati delle query in oggetti javascript.

Per eseguire l'applicazione è necessario aver installato node.js e npm (preinstallato con node.js) e poi eseguire il seguente comando nella cartella `GUI`:

```
npm run install-and-run
```

che installerà tutte le dipendenze necessarie, compilerà il codice sorgente (in preview) e avvierà l'applicazione.