Progetto “Collectors”

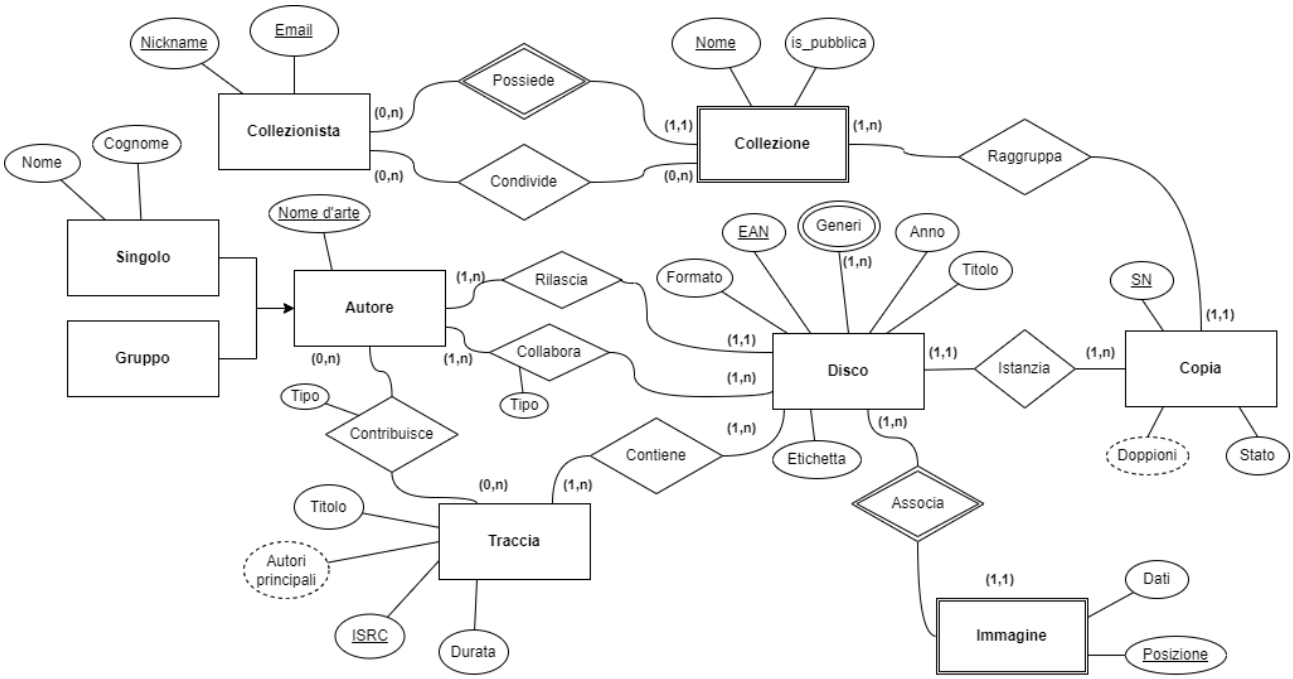
Daniele Borgna

5 luglio 2023

Contents

1 Modello entità-relazione

Di seguito il modello E-R con relative note a margine per concetti ed assunzioni non modellabili.



1.1 Assunzioni

1. Ogni singola copia di un disco può appartenere ad esattamente una collezione. Ciò perché, essendo un sistema fortemente orientato alle collezioni, ritengo non sia utile caricare copie di dischi senza inserirle in alcuna collezione.

2. Ogni disco può essere classificato in almeno un genere.

3. Un disco è rilasciato da un solo artista (principale) ma possono esserci eventuali artisti secondari.

4. Due dischi sono uguali se hanno stesso EAN (barcode). L'EAN è infatti uguale per due prodotti uguali, ma diverso se i prodotti sono diversi anche in un minimo particolare (es: versione deluxe del disco, cd oppure vinile, immagini diverse ...).

5. Per distinguere invece copie dello stesso disco (quindi stesso EAN) utilizziamo il Serial Number (SN): tali copie possono variare solo nello stato di conservazione, mentre formato, EAN, anno e tutti gli altri attributi devono essere identici.

6. Uno stesso autore può contribuire a tracce diverse in modi diversi: ad esempio per una traccia può essere il produttore, mentre per un'altra può essere il cantante.

7. Un gruppo è rappresentato solo da un nome d'arte, senza indicare nome e cognome dei singoli componenti. Assumo inoltre che non esistano autori (gruppi o singoli) con stesso nome d'arte.

8. Il codice ISRC (international standard recording code) garantisce l'univocità di supporti audio e video musicali (fonte: fimi.it)

1.2 Note a margine

1. Come detto nell'assunzione 3, il "tipo" di contributo in una traccia (o disco) da parte di un autore può variare da traccia a traccia. Di conseguenza, l'attributo "Tipo" è legato alle relazioni "Contribuisce" e "Rilascia" e non all'autore.

2. Il flag "is_pubblica" dell'entità "Collezione" indica, come facilmente intuibile, se una collezione è pubblica (ovvero visibile da chiunque) o meno. Nel caso in cui non fosse pubblica, si può distinguere ulteriormente in **condivisa**, se la lista dei collezionisti con cui è condivisa non è vuota, oppure **privata** se tale lista è vuota.

3. L'entità "Disco" contiene le informazioni generali, mentre l'entità "Copia" si riferisce alla singola copia in possesso del collezionista.

4. Nell'entità "Disco", l'attributo anno si riferisce all'anno di pubblicazione.

5. L' "immagine" è identificata dal disco di cui fa parte e dalla posizione in cui è allocata. Non possono infatti esserci più immagini nella stessa posizione sullo stesso disco (qualsiasi copia di quel disco).

6. L'attributo dati dell'entità "Immagine" consiste nel file contenente l'immagine.

7. Gli autori principali nell'entità "Traccia" sono gli autori del disco di cui la traccia fa parte. Quelli che contribuiscono sono invece eventuali featuring, che possono anche non esserci.

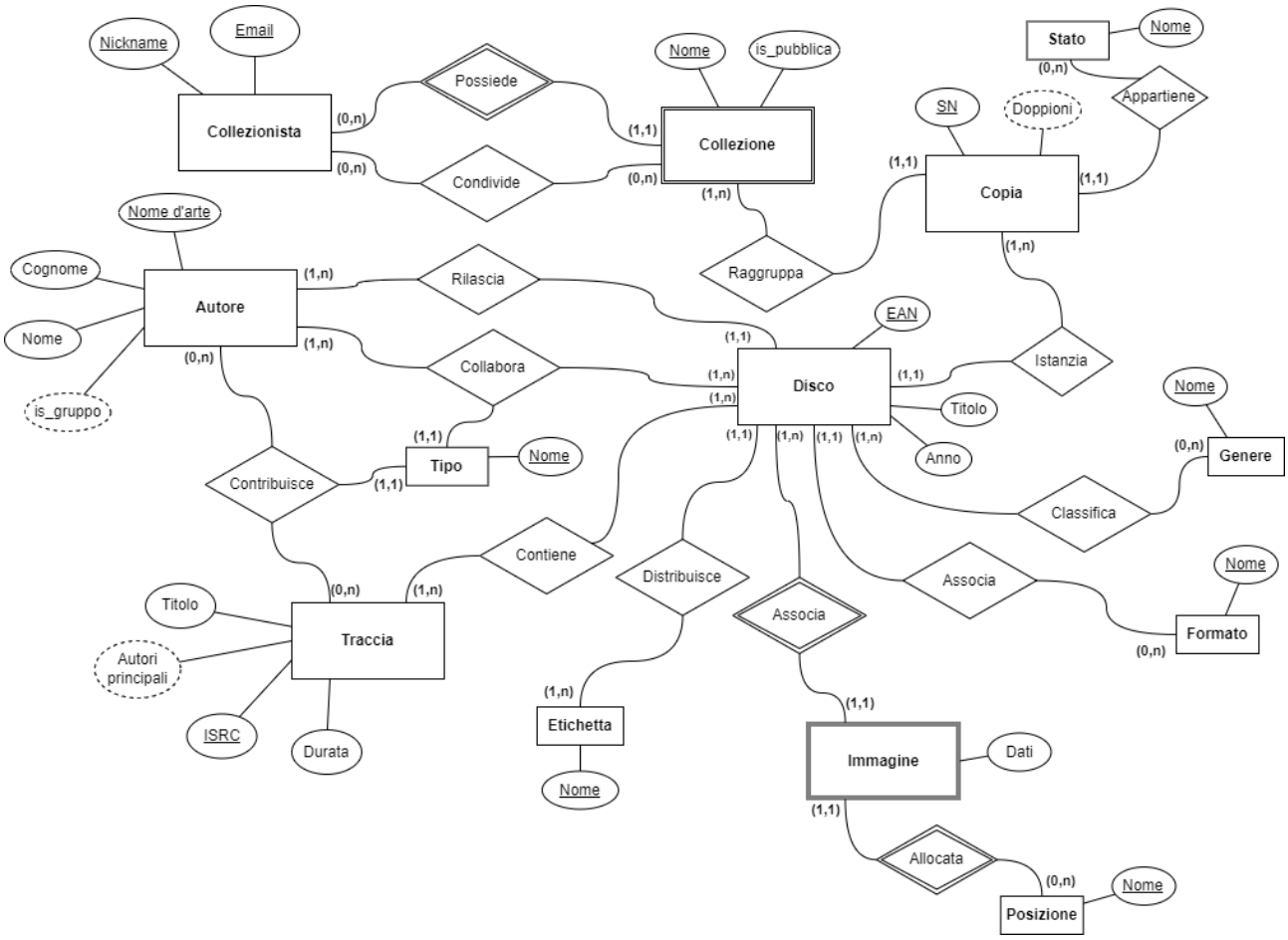
8. La "Collezione" è identificata univocamente dal suo nome e dai dati del suo collezionista. Collezioni diverse di collezionisti diversi possono quindi avere lo stesso nome.

9. L'attributo doppioni dell'entità "Copia" è derivabile contando il numero di copie che si riferiscono allo stesso disco e allo stesso collezionista. Tale attributo è quindi un intero che conta il numero delle copie con tali caratteristiche, inclusa quella "attuale".

10. Dato che l'etichetta è caratterizzata solo dal nome, posso trattarla come gli altri tipi enumerativi (genere, formato, ...).

2 Modello entità-relazione ristrutturato

Di seguito il modello E-R ristrutturato con relative note a margine per concetti non modellabili.



2.1 Trasformazioni effettuate

1. Tutti gli attributi enumerativi (genere e formato per l'entità "Disco", stato per l'entità "Copia" e tipo per le relazioni "Contribuisce" e "Rilascia") sono stati trasformati in entità.
2. In particolare, l'attributo multiplo genere è stato trasformato in una relazione (1,n).
3. La generalizzazione di "Autore" (singolo e gruppo) è stata rimossa lasciando spazio al flag is_gruppo ricavabile da nome e cognome (se sono nulli, l'autore è un gruppo).

3 Modello relazionale

- **Collezionista**(ID, nickname, email)
  [nickname UNIQUE, email UNIQUE]
- **Collezione**(ID, nome, isPubblica, id_Collezionista)
  [(nome, id_Collezionista) UNIQUE]
- **Copia**(ID, SN, doppioni, id_Disco, nome_Stato, id_Collezione)
  [SN UNIQUE]
- **Disco**(ID, EAN, titolo, anno, id_Etichetta, nome_Formato, id_Autore)
  [EAN UNIQUE, (titolo, anno, id_Etichetta, nome_Formato, id_Autore) UNIQUE]
- **Autore**(ID, nomeDarte, nome, cognome, isGruppo)
  [nomeDarte UNIQUE, nome NULL, cognome NULL]
- **Traccia**(ID, titolo, ISRC, Durata)
  [ISRC UNIQUE]
- **Immagine**(ID, dati, id_Disco, nome_Posizione)
  [(id_Disco, nome_Posizione) UNIQUE]
- **Etichetta**(nome)
- **Stato**(nome)
- **Genere**(nome)
- **Formato**(nome)
- **Posizione**(nome)
- **Tipo**(nome)
- **condivide**(id_Collezionista, id_Collezione)
- **collabora**(id_Autore, id_Disco, nome_Tipo)
- **contribuisce**(id_Autore, id_Traccia, nome_Tipo)
- **contiene**(id_Traccia, id_Disco)
- **classifica**(id_Disco, id_Genere)

4 Implementazione della struttura

```sql
DROP DATABASE IF EXISTS collectors;

CREATE DATABASE collectors;
USE collectors;

DROP USER IF EXISTS 'collectorsUser'@'localhost';
CREATE USER 'collectorsUser'@'localhost' IDENTIFIED BY 'collectorsPwd';
GRANT select,insert,update,delete,execute ON collectors.* TO 'collectorsUser'@'localhost';

CREATE TABLE Collezionista (
    ID INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    nickname VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL
);

CREATE TABLE Collezione (
    ID INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    nome VARCHAR(50) NOT NULL,
    isPubblica BOOLEAN NOT NULL,
    id_Collezionista INT UNSIGNED NOT NULL,
    CONSTRAINT collezione_distinta UNIQUE (nome, id_Collezionista),
    CONSTRAINT collezione_collezionista FOREIGN KEY (id_Collezionista) REFERENCES
        Collezionista(ID) ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE Etichetta (
    nome VARCHAR(50) PRIMARY KEY
);

CREATE TABLE Genere (
    nome VARCHAR(50) PRIMARY KEY
);

CREATE TABLE Formato (
    nome VARCHAR(50) PRIMARY KEY
);

CREATE TABLE Disco (
    ID INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    EAN CHAR(13) UNIQUE NOT NULL,
    --EAN-13 ha una lunghezza fissa di 13 caratteri
    titolo VARCHAR(50) NOT NULL,
    anno SMALLINT UNSIGNED NOT NULL,
    nome_Etichetta VARCHAR(50) NOT NULL,
    nome_Formato VARCHAR(50) NOT NULL,
    id_Autore INT UNSIGNED NOT NULL,
    CONSTRAINT disco_distinto UNIQUE (titolo, anno, nome_Etichetta, nome_Formato, id_Autore),
    --Non ha senso che esistano dischi uguali con EAN distinti
    CONSTRAINT disco_etichetta FOREIGN KEY (nome_Etichetta) REFERENCES Etichetta(nome) ON DELETE
        NO ACTION ON UPDATE CASCADE,
    CONSTRAINT disco_formato FOREIGN KEY (nome_Formato) REFERENCES Formato(nome) ON DELETE NO
        ACTION ON UPDATE CASCADE,
    CONSTRAINT controllo_anno CHECK (anno > 1930 AND anno < 2100)

    --Nel 1931, la casa discografica Rca-Victor distribuì il primo Lp della storia: la Quinta
        sinfonia di Beethoven.
);

CREATE TABLE classifica (
    id_Disco INT UNSIGNED,
    nome_Genere VARCHAR(50),
    PRIMARY KEY (id_Disco, nome_Genere),
    CONSTRAINT classifica_disco FOREIGN KEY (id_Disco) REFERENCES Disco(ID) ON DELETE CASCADE ON
        UPDATE CASCADE,
```

```
60          CONSTRAINT classifica_genere FOREIGN KEY (nome_Genere) REFERENCES Genere(nome) ON DELETE
                CASCADE ON UPDATE CASCADE
61      );
62
63      CREATE TABLE Stato (
64          nome VARCHAR(50) PRIMARY KEY
65      );
66
67
68      -- Facendo riferimento all'assunzione 1, essendo la copia strettamente collegata alla
            collezione, ritengo corretto eliminare le copie di una collezione quando essa viene
            eliminata.
69
70      CREATE TABLE Copia (
71          ID INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
72          SN CHAR(12) UNIQUE,
73          doppioni INT UNSIGNED NOT NULL DEFAULT 1,
74          --Quando si inserisce una copia, il conteggio viene chiaramente inizializzato a uno (ovvero
                la copia stessa)
75          id_Disco INT UNSIGNED NOT NULL,
76          nome_Stato VARCHAR(50) NOT NULL,
77          id_Collezione INT UNSIGNED NOT NULL,
78          CONSTRAINT copia_disco FOREIGN KEY (id_Disco) REFERENCES Disco(ID) ON DELETE NO ACTION ON
                UPDATE CASCADE,
79          CONSTRAINT copia_stato FOREIGN KEY (nome_Stato) REFERENCES Stato(nome) ON DELETE NO ACTION
                ON UPDATE CASCADE,
80          CONSTRAINT copia_collezione FOREIGN KEY (id_Collezione) REFERENCES Collezione(ID) ON DELETE
                CASCADE ON UPDATE CASCADE
81      );
82
83      CREATE TABLE Autore (
84          ID INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
85          nomeDarte VARCHAR(100) UNIQUE NOT NULL,
86          nome VARCHAR(100),
87          cognome VARCHAR(100),
88          isGruppo BOOLEAN NOT NULL
89      );
90
91
92      -- Controllo sul vincolo tra nome, cognome e isGruppo
93      DELIMITER $$
94      CREATE TRIGGER AggiuntaAutore BEFORE INSERT ON Autore FOR EACH ROW
95      BEGIN
96          IF NEW.nome IS NOT NULL AND NEW.cognome IS NOT NULL AND NEW.isGruppo = TRUE THEN
97              SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Se nome e cognome non sono nulli, autore
                    deve essere un singolo';
98          END IF;
99      END$$
100     DELIMITER ;
101
102
103     CREATE TABLE Traccia (
104         ID INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
105         titolo VARCHAR(100) NOT NULL,
106         ISRC CHAR(12) UNIQUE NOT NULL,
107         -- Codice ISRC composto da 12 cifre alfanumeriche
108         Durata TIME NOT NULL
109     );
110
111     CREATE TABLE Posizione (
112         nome VARCHAR(50) PRIMARY KEY
113     );
114
115     CREATE TABLE Immagine (
116         ID INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
117         URL VARCHAR(2083),
```

```
118         id_Disco INT UNSIGNED NOT NULL,
119         nome_Posizione VARCHAR(50) NOT NULL,
120         CONSTRAINT immagine_distinta UNIQUE (id_Disco, nome_Posizione),
121         CONSTRAINT immagine_disco FOREIGN KEY (id_Disco) REFERENCES Disco(ID) ON DELETE CASCADE ON
                UPDATE CASCADE,
122         CONSTRAINT immagine_posizione FOREIGN KEY (nome_Posizione) REFERENCES Posizione(nome) ON
                DELETE NO ACTION ON UPDATE CASCADE
123     );
124
125     CREATE TABLE Tipo (
126         nome VARCHAR(50) PRIMARY KEY
127     );
128
129     CREATE TABLE condivide (
130         id_Collezionista INT UNSIGNED,
131         id_Collezione INT UNSIGNED,
132         PRIMARY KEY (id_Collezionista, id_Collezione),
133         CONSTRAINT condivide_collezionista FOREIGN KEY (id_Collezionista) REFERENCES
                Collezionista(ID) ON DELETE CASCADE ON UPDATE CASCADE,
134         CONSTRAINT condivide_collezione FOREIGN KEY (id_Collezione) REFERENCES Collezione(ID) ON
                DELETE CASCADE ON UPDATE CASCADE
135     );
136
137     CREATE TABLE collabora (
138         id_Autore INT UNSIGNED,
139         id_Disco INT UNSIGNED,
140         nome_Tipo VARCHAR(50) NOT NULL,
141         PRIMARY KEY (id_Autore, id_Disco),
142         CONSTRAINT collabora_autore FOREIGN KEY (id_Autore) REFERENCES Autore(ID) ON DELETE NO
                ACTION ON UPDATE CASCADE,
143         CONSTRAINT collabora_disco FOREIGN KEY (id_Disco) REFERENCES Disco(ID) ON DELETE CASCADE ON
                UPDATE CASCADE,
144         CONSTRAINT collabora_tipo FOREIGN KEY (nome_Tipo) REFERENCES Tipo(nome) ON DELETE NO ACTION
                ON UPDATE CASCADE
145     );
146
147     CREATE TABLE contribuisce (
148         id_Autore INT UNSIGNED,
149         id_Traccia INT UNSIGNED,
150         nome_Tipo VARCHAR(50) NOT NULL,
151         PRIMARY KEY (id_Autore, id_Traccia),
152         CONSTRAINT contribuisce_autore FOREIGN KEY (id_Autore) REFERENCES Autore(ID) ON DELETE
                CASCADE ON UPDATE CASCADE,
153         CONSTRAINT contribuisce_traccia FOREIGN KEY (id_Traccia) REFERENCES Traccia(ID) ON DELETE
                CASCADE ON UPDATE CASCADE,
154         CONSTRAINT contribuisce_tipo FOREIGN KEY (nome_Tipo) REFERENCES Tipo(nome) ON DELETE NO
                ACTION ON UPDATE CASCADE
155         );
156
157         CREATE TABLE contiene (
158         id_Traccia INT UNSIGNED,
159         id_Disco INT UNSIGNED,
160         PRIMARY KEY (id_Traccia, id_Disco),
161         CONSTRAINT contiene_traccia FOREIGN KEY (id_Traccia) REFERENCES Traccia(ID) ON DELETE
                CASCADE ON UPDATE CASCADE,
162         CONSTRAINT contiene_disco FOREIGN KEY (id_Disco) REFERENCES Disco(ID) ON DELETE CASCADE ON
                UPDATE CASCADE
163         );
```

5 Operazioni richieste

In questa sezione troviamo le operazioni richieste al database. Ogni operazione è inclusa in una procedura e nella prima parte sono presenti delle procedure ausiliarie che ho utilizzato durante lo sviluppo. All'inizio del documento sono presenti le seguenti righe di codice:

```
USE collectors;
DELIMITER $$
```

5.1 Procedure ausiliarie

5.1.1 Stampa ogni coppia disco-traccia con i relativi attributi

```
DROP PROCEDURE IF EXISTS DischiTracce$$

CREATE PROCEDURE DischiTracce()
BEGIN
SELECT D.titolo as "Titolo Disco", D.EAN, T.titolo as "Titolo Traccia", T.ISRC, T.durata,
    A.nomeDarte as "Autore principale" FROM Disco D, Traccia T, Contiene C, Autore A WHERE D.id
    = C.id_disco AND T.id = C.id_traccia AND A.id = D.id_autore ORDER BY D.titolo;
END$$
```

5.1.2 Stampa ogni coppia collezione-collezionista delle condivisioni

```
DROP PROCEDURE IF EXISTS Condivisioni$$

CREATE PROCEDURE Condivisioni()
BEGIN
    SELECT C.nickname as "Owner", CL.nome as "Collezione", C2.nickname as "Guest" FROM
        Collezionista C, Collezione CL, Collezionista C2, Condivide CO WHERE CL.isPubblica =
        false AND CL.id_collezionista = C.id AND CL.id = CO.id_collezione AND C2.id =
        CO.id_collezionista;
END$$
```

5.1.3 Stampa ogni coppia collezione-copia

```
DROP PROCEDURE IF EXISTS CollezioniCopie$$

CREATE PROCEDURE CollezioniCopie()
BEGIN
    SELECT CL.nickname, C.nome, CO.SN, CO.nome_Stato AS "Stato", CO.doppioni, C.isPubblica AS
        Pubblica, D.titolo, D.EAN, A.nomeDarte AS "Autore principale" FROM Autore A,
        Collezionista CL, Collezione C, Copia CO, Disco D WHERE C.id = CO.id_collezione AND
        CO.id_disco = D.id AND C.id_collezionista = CL.id AND A.id = D.id_autore ORDER BY
        C.nome;
END$$
```

5.1.4 Stampa ogni coppia disco-copia

```
DROP PROCEDURE IF EXISTS DischiCopie$$

CREATE PROCEDURE DischiCopie()
BEGIN
    SELECT D.titolo, D.EAN, C.SN, C.nome_Stato, A.nomeDarte AS "Autore principale" FROM Disco D,
        Copia C, Autore A WHERE C.id_disco = D.id AND A.id = D.id_autore;
END$$
```

5.1.5 Stampa ogni coppia disco-autore secondario

```
DROP PROCEDURE IF EXISTS DischiAutoriSec$$

CREATE PROCEDURE DischiAutoriSec()
BEGIN
    SELECT D.titolo, D.EAN, A.nomeDarte AS "Autore Secondario", C.nome_tipo AS "Tipo" FROM Disco
        D, Autore A, Collabora C WHERE D.id = C.id_disco AND C.id_autore = A.id;
END$$
```

5.1.6 Stampa ogni coppia traccia-autore secondario

```
DROP PROCEDURE IF EXISTS TracceAutoriSec$$

CREATE PROCEDURE TracceAutoriSec()
BEGIN
    SELECT T.titolo, t.ISRC, A.nomeDarte AS "Autore Secondario", C.nome_tipo AS "Tipo" FROM
        Traccia T, Autore A, Contribuisce C WHERE T.id = C.id_traccia AND C.id_autore = A.id;
END$$
```

5.1.7 Dato nome della collezione e nickname del collezionista, restituisce l'id della collezione

```
DROP FUNCTION IF EXISTS GetCollezione$$

CREATE FUNCTION GetCollezione (
    p_nome_collezione VARCHAR(50),
    p_nickname_collezionista VARCHAR(50)
) RETURNS INT UNSIGNED DETERMINISTIC
BEGIN
    RETURN (SELECT id FROM Collezione WHERE nome = p_nome_collezione AND id_collezionista =
        (SELECT id FROM Collezionista WHERE nickname = p_nickname_collezionista));
END$$
```

1. Inserimento di una nuova collezione

```
DROP PROCEDURE IF EXISTS InserisciCollezione$$

CREATE PROCEDURE InserisciCollezione(
    IN p_nome VARCHAR(50),
    IN p_isPubblica BOOLEAN,
    IN p_nickname_collezionista VARCHAR(50)
)
BEGIN
    DECLARE p_id_collezionista INT UNSIGNED;
    SET p_id_collezionista = (SELECT id FROM Collezionista WHERE nickname =
        p_nickname_collezionista);

    IF p_id_collezionista IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Non esiste un collezionista con questo
            nickname';
    ELSE
    -- Inserimento della nuova riga nella tabella Collezione
        INSERT INTO Collezione (nome, isPubblica, id_Collezionista)
        VALUES (p_nome, p_isPubblica, p_id_Collezionista);
    END IF;
END$$
```

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

2a. Inserimento di un disco non presente nel sistema

```
CREATE PROCEDURE InserisciDisco(
    IN p_EAN CHAR(13),
```

```
        IN p_titolo VARCHAR(50),
        IN p_anno SMALLINT,
        IN p_nome_Etichetta VARCHAR(50),
        IN p_nome_Formato VARCHAR(50),
        IN p_nomeDarte_autore_principale VARCHAR(50),
        IN p_nome_genere_principale VARCHAR(50)
    )
    BEGIN
        DECLARE p_id_autore INT UNSIGNED;
        DECLARE p_id_nuovo_disco INT UNSIGNED;
        SET p_id_autore = (SELECT id FROM Autore WHERE nomeDarte = p_nomeDarte_autore_principale);

        IF p_id_autore IS NULL THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Non esiste un autore con questo nome d arte';
        ELSEIF (SELECT * FROM Genere WHERE nome = p_nome_genere_principale) IS NULL THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Il genere indicato non esiste';
        ELSE
            IF (SELECT * FROM Etichetta WHERE nome = p_nome_Etichetta) IS NULL THEN
                INSERT INTO Etichetta VALUE (p_nome_Etichetta);
            END IF;
            INSERT INTO Disco(EAN,titolo,anno,nome_Etichetta,nome_Formato,id_Autore) VALUE
                (p_EAN,p_titolo,p_anno,p_nome_Etichetta,p_nome_Formato,p_id_autore);
            SET p_id_nuovo_disco = last_insert_id();
            INSERT INTO Classifica VALUE (p_id_nuovo_disco, p_nome_genere_principale);
        END IF;
    END$$
```

2b. Aggiunta di un genere secondario al disco

```
    DROP PROCEDURE IF EXISTS AggiungiGenereAlDisco$$

    CREATE PROCEDURE AggiungiGenereAlDisco(
        IN p_nome_genere VARCHAR(50),
        IN p_EAN_disco CHAR(13)
    )
    BEGIN
        IF (SELECT * FROM Genere WHERE nome = p_nome_genere) IS NULL THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Non esiste un genere con questo nome';
        ELSEIF (SELECT id FROM Disco WHERE EAN = p_EAN_disco) IS NULL THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Non esiste un disco con questo EAN';
        ELSEIF EXISTS (SELECT * FROM Classifica WHERE id_disco = (SELECT id FROM Disco WHERE EAN =
            p_EAN_disco) AND nome_genere = p_nome_genere) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Genere gia associato al disco';
        ELSE
            INSERT INTO Classifica VALUE ((SELECT id FROM Disco WHERE EAN = p_EAN_disco),
                p_nome_genere);
        END IF;
    END$$
```

2c. Aggiunta autore secondario al disco

```
    DROP PROCEDURE IF EXISTS AggiungiAutoreAlDisco$$

    CREATE PROCEDURE AggiungiAutoreAlDisco(
        IN p_nomeDarte_autore VARCHAR(50),
        IN p_EAN_disco CHAR(13),
        IN p_nome_tipo VARCHAR(50)
    )
    BEGIN
        DECLARE p_id_autore INT UNSIGNED;
        DECLARE p_id_disco INT UNSIGNED;
        SET p_id_autore = (SELECT id FROM Autore WHERE nomeDarte = p_nomeDarte_autore);
        SET p_id_disco = (SELECT id FROM Disco WHERE EAN = p_EAN_disco);
        IF p_id_autore IS NULL THEN
```

```
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'L autore indicato non esiste';
        ELSEIF p_id_disco IS NULL THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Non esiste un disco con questo EAN';
        ELSEIF (SELECT * FROM Tipo WHERE nome = p_nome_tipo) IS NULL THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Non esiste un tipo con questo nome';
        ELSEIF EXISTS ((SELECT * FROM collabora WHERE id_disco = p_id_disco AND id_autore =
            p_id_autore AND nome_Tipo = p_nome_tipo)) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Autore gia associato al disco in quel ruolo';
        ELSEIF EXISTS (SELECT id FROM Disco WHERE id_autore = p_id_autore and id = p_id_disco) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Autore gia associato al disco in quel ruolo';
        ELSE
            INSERT INTO collabora VALUE (p_id_autore, p_id_disco, p_nome_tipo);
        END IF;
    END$$
```

2d. Aggiunta di nuova copia ed assegnamento ad una collezione

```
    DROP PROCEDURE IF EXISTS InserisciCopia$$

    CREATE PROCEDURE InserisciCopia(
        IN p_SN CHAR(12),
        IN p_EAN_Disco CHAR(13),
        IN p_nome_Stato VARCHAR(50),
        IN p_nome_Collezione VARCHAR(50),
        IN p_nickname_Collezionista VARCHAR(50)
    )
    BEGIN
        DECLARE num_doppioni INT;
        DECLARE p_id_Disco INT UNSIGNED;
        DECLARE p_id_collezione INT UNSIGNED;
        SET p_id_Disco = (SELECT D.id FROM Disco D WHERE D.EAN = p_EAN_Disco);
        SET p_id_collezione = GetCollezione(p_nome_collezione, p_nickname_Collezionista);
        IF EXISTS (SELECT id FROM Copia WHERE SN = p_SN) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Esiste gia una copia con questo SN';
        ELSEIF p_id_Disco IS NULL THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Non esiste un disco con questo EAN';
        ELSEIF p_id_collezione IS NULL THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Non esiste una collezione con questo nome
                associata a questo nickname';
        ELSE
            -- Inserimento della nuova riga nella tabella Copia
            INSERT INTO Copia (SN, id_Disco, nome_Stato, id_Collezione)
            VALUES (p_SN, p_id_Disco, p_nome_Stato, p_id_Collezione);

            -- Calcolo del numero di doppioni per l'id_disco appena inserito
            SET num_doppioni = (SELECT COUNT(*) FROM Copia WHERE id_Disco = p_id_Disco);

            -- Aggiornamento della colonna doppioni per tutte le copie con lo stesso id_disco
            UPDATE Copia C
            SET C.doppioni = num_doppioni
            WHERE C.id_Disco = p_id_Disco;
        END IF;
    END$$
```

2e. Spostamento copia gia esistente da una collezione ad un altra dello stesso utente

```
    DROP PROCEDURE IF EXISTS CopiaInCollezione$$

    CREATE PROCEDURE CopiaInCollezione(
        IN p_nome_collezione VARCHAR(50),
        IN p_SN_copia CHAR(12)
    )
    BEGIN
        DECLARE p_id_collezionista INT UNSIGNED;
```

```
        SET p_id_collezionista = (SELECT CZ.id_Collezionista FROM Copia C, Collezione CZ WHERE C.SN
            = p_SN_copia AND CZ.id = C.id_Collezione);

        -- Un collezionista non puo spostare una sua copia nella collezione di un altro utente
        IF EXISTS(SELECT CZ.nome FROM Copia C, Collezione CZ WHERE C.SN = p_SN_copia AND
            C.id_collezione = CZ.id AND CZ.nome = p_nome_collezione AND CZ.id_collezionista =
            p_id_collezionista) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Copia gia nella collezione';
        ELSEIF EXISTS(SELECT CZ.nome FROM Collezione CZ WHERE CZ.nome = p_nome_collezione AND
            CZ.id_collezionista = p_id_collezionista) THEN
            UPDATE Copia C
            SET C.id_Collezione = (SELECT C.id FROM Collezione C, Collezionista CL WHERE C.nome =
                p_nome_collezione AND CL.id = p_id_collezionista)
            WHERE C.SN = p_SN_copia;
        ELSE
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La collezione indicata non esiste';
        END IF;
    END$$
```

2f. inserimento di una nuova traccia ed assegnamento a un disco esistente

```
    DROP PROCEDURE IF EXISTS NuovaTracciaInDisco$$

    CREATE PROCEDURE NuovaTracciaInDisco(
        p_titolo VARCHAR(100),
        p_ISRC CHAR(12),
        p_durata TIME,
        p_EAN CHAR(13)
    )
    BEGIN
        DECLARE p_id_disco INT UNSIGNED;
        DECLARE p_id_nuova_traccia INT UNSIGNED;
        SET p_id_disco = (SELECT D.id FROM Disco D WHERE D.EAN = p_EAN);

        IF p_id_disco IS NULL THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Il disco indicato non esiste';
        ELSE
            INSERT INTO Traccia(titolo, ISRC, durata) VALUES (p_titolo, p_ISRC, p_durata);
            SET p_id_nuova_traccia = last_insert_id();
            INSERT INTO Contiene VALUES (p_id_nuova_traccia,p_id_disco);
        END IF;
    END$$
```

2g. Inserimento traccia esistente in un disco

```
    DROP PROCEDURE IF EXISTS TracciaInDisco$$

    CREATE PROCEDURE TracciaInDisco(
        IN p_ISRC_traccia CHAR(12),
        IN p_EAN_disco CHAR(13)
    )
    BEGIN
        DECLARE p_id_traccia INT UNSIGNED;
        DECLARE p_id_disco INT UNSIGNED;
        SET p_id_traccia = (SELECT T.id FROM Traccia T WHERE T.ISRC = p_ISRC_traccia);
        SET p_id_disco = (SELECT D.id FROM Disco D WHERE D.EAN = p_EAN_disco);

        IF p_id_disco IS NULL THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Il disco indicato non esiste';
        ELSEIF p_id_traccia IS NULL THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La traccia indicata non esiste';
        ELSEIF EXISTS (SELECT C.* FROM Contiene C WHERE C.id_traccia = p_id_traccia AND C.id_disco =
            p_id_disco) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Traccia gia nel disco';
```

```
        ELSE
            INSERT INTO Contiene (id_traccia, id_disco) VALUES (p_id_traccia, p_id_disco);
        END IF;
    END$$
```

3. Modifica dello stato di pubblicazione e aggiunta nuove condivisioni

3a. Modifica dello stato di condivisione

```
    DROP PROCEDURE IF EXISTS switchCondivisione$$

    CREATE PROCEDURE switchCondivisione(
        IN p_nome_Collezione VARCHAR(50),
        IN p_nickname_Collezionista VARCHAR(50)
    )
    BEGIN
        DECLARE p_stato_attuale BOOLEAN;
        DECLARE p_id_collezione INT UNSIGNED;

        SET p_id_collezione = GetCollezione(p_nome_Collezione, p_nickname_Collezionista);
        SET p_stato_attuale = (SELECT isPubblica FROM Collezione WHERE id = p_id_collezione);

        UPDATE Collezione SET isPubblica = NOT p_stato_attuale WHERE id = p_id_collezione;

        IF p_id_collezione IS NULL THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La collezione non esiste';
        ELSEIF (p_stato_attuale = TRUE) THEN
            DELETE FROM condivide WHERE id_collezione = p_id_collezione;
            # Se sto switchando da Privata a Pubblica, rimuovo i collezionisti con cui era condivisa
        END IF;
    END$$
```

3a. Aggiunta di una nuova condivisione (se privata)

```
    DROP PROCEDURE IF EXISTS AggiungiCondivisione$$

    CREATE PROCEDURE AggiungiCondivisione(
        IN p_nickname_owner VARCHAR(50),
        IN p_nickname_guest VARCHAR(50),
        IN p_nome_collezione VARCHAR(50)
        )
    BEGIN
        DECLARE p_id_collezione INT UNSIGNED;
        DECLARE p_id_owner INT UNSIGNED;
        DECLARE p_id_guest INT UNSIGNED;

        SET p_id_owner = (SELECT id FROM Collezionista WHERE nickname = p_nickname_owner);

        IF p_id_owner IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Non esiste un utente con il nickname indicato
            per l owner';
        ELSE
            SET p_id_collezione = (SELECT id FROM Collezione WHERE nome = p_nome_collezione AND
                id_collezionista = p_id_owner);

            IF p_id_collezione IS NULL THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La collezione non esiste';
                ELSE
                SET p_id_guest = (SELECT id FROM Collezionista WHERE nickname = p_nickname_guest);

                IF p_id_guest IS NULL THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Non esiste un utente con il nickname
                    indicato per il guest';
                ELSE
```

```
            IF (SELECT isPubblica FROM Collezione WHERE id = p_id_collezione) = TRUE THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'collezione pubblica, renderla privata e riprovare';
            ELSE
            IF EXISTS (SELECT * FROM Condivide WHERE id_collezionista = p_id_guest AND id_collezione = p_id_collezione) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Collezione gia condivisa con questo utente';
            ELSE
            INSERT INTO Condivide VALUES (p_id_guest, p_id_collezione);
            END IF;
                END IF;
                END IF;
        END IF;
        END IF;
END$$
```

4. Rimozione di un disco da una collezione

```
DROP PROCEDURE IF EXISTS RimuoviCopiaDaCollezione$$

-- Come detto nell'assunzione 1, ritengo poco utile mantenere nel sistema copie senza una collezione. Di conseguenza, rimuovere una copia da una collezione significa rimuoverla dal sistema.

CREATE PROCEDURE RimuoviCopiaDaCollezione(
    p_SN_copia CHAR(12),
    p_nome_collezione VARCHAR(50)
)
BEGIN
    DECLARE p_id_copia INT UNSIGNED;
    DECLARE p_id_collezione INT UNSIGNED;
    SET p_id_copia = (SELECT id FROM Copia WHERE SN = p_SN_copia);
    IF p_id_copia IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Non esiste una copia con questo SN';
    ELSE
        SET p_id_collezione = (SELECT C.id FROM Collezione C, Copia CP WHERE CP.id = p_id_copia AND C.id = CP.id_collezione AND C.nome = p_nome_collezione);
        IF p_id_collezione IS NULL THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La copia non fa parte della collezione indicata';
        ELSE
            DELETE FROM Copia WHERE id = p_id_copia;
        END IF;
    END IF;
END$$
```

5. Rimozione di una collezione

```
DROP PROCEDURE IF EXISTS RimuoviCollezione$$

CREATE PROCEDURE RimuoviCollezione(
    p_nickname_collezionista VARCHAR(50),
    p_nome_collezione VARCHAR(50)
)
BEGIN
    DECLARE p_id_collezione INT UNSIGNED;
    SET p_id_collezione = GetCollezione(p_nome_collezione, p_nickname_collezionista);

    IF p_id_collezione IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La collezione indicata non esiste';
    ELSE
        DELETE FROM Collezione WHERE id = p_id_collezione;
```

```
    END IF;
END$$
```

6. Lista di tutti i dischi in una collezione

```
DROP PROCEDURE IF EXISTS DischiInCollezione$$

CREATE PROCEDURE DischiInCollezione(
    p_nickname_collezionista VARCHAR(50),
    p_nome_collezione VARCHAR(50)
)
BEGIN
    DECLARE p_id_collezione INT UNSIGNED;
    SET p_id_collezione = GetCollezione(p_nome_collezione, p_nickname_collezionista);

    IF p_id_collezione IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La collezione indicata non esiste';
    ELSE
        SELECT DISTINCT D.* FROM Disco D, Copia C WHERE D.id = C.id_disco AND C.id_collezione =
            p_id_collezione;
    END IF;
END$$
```

7. Track list di un disco

```

DROP PROCEDURE IF EXISTS TrackList$$

CREATE PROCEDURE TrackList(
    p_EAN CHAR(13)
)
BEGIN
    DECLARE p_id_disco INT UNSIGNED;
    SET p_id_disco = (SELECT id FROM Disco WHERE EAN = p_EAN);

    IF p_id_disco IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Il disco indicato non esiste';
    ELSE
        SELECT DISTINCT T.* FROM Traccia T, Contiene C WHERE C.id_traccia = T.id AND C.id_disco
            = p_id_disco;
    END IF;
END$$
```

8. Ricerca dischi in base a diversi criteri

```
DROP PROCEDURE IF EXISTS RicercaDischi$$

-- Per non includere l autore nella ricerca, inserisco null come argomento dell'autore. Stesso
    discorso per il titolo.
CREATE PROCEDURE RicercaDischi(
    IN p_nome_autore VARCHAR(50),
    IN p_titolo VARCHAR(50),
    IN p_nome_collezionista VARCHAR(50),
    IN p_includi_private BOOLEAN,
    IN p_includi_condivise BOOLEAN,
    IN p_includi_pubbliche BOOLEAN
)
BEGIN
    DECLARE p_id_collezionista INT UNSIGNED;
    DECLARE p_id_autore INT UNSIGNED;
    SET p_id_collezionista = (SELECT id FROM Collezionista WHERE nickname =
        p_nome_collezionista);
```

```
16
17          SET p_id_autore = (SELECT id FROM Autore WHERE nomeDarte = p_nome_autore);
18          IF p_id_autore IS NULL AND p_nome_autore IS NOT NULL THEN
19              SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'L autore indicato non esiste';
20          ELSE
21              -- Ricerca in base ai criteri specificati
22              SELECT DISTINCT D.* FROM Disco D, collabora C WHERE
23              ((p_titolo IS NOT NULL AND p_id_autore IS NULL AND D.titolo LIKE CONCAT('%', p_titolo,
                    '%'))
24              OR
25              (p_id_autore IS NOT NULL AND p_titolo IS NULL AND C.id_autore = p_id_autore AND D.id =
                    C.id_disco)
26              OR
27              (p_titolo IS NOT NULL AND p_id_autore IS NOT NULL AND D.titolo LIKE CONCAT('%',
                    p_titolo, '%') AND C.id_autore = p_id_autore AND D.id = C.id_disco))
28              AND
29              ((p_includi_private = 1 AND EXISTS (SELECT * FROM Copia C WHERE C.id_disco = D.id AND
                    C.id_collezione IN (SELECT id FROM Collezione WHERE id_collezionista =
                    p_id_collezionista)))
30              OR
31              (p_includi_condivise = 1 AND EXISTS (SELECT * FROM Copia C, Condivide CC WHERE
                    C.id_disco = D.id AND C.id_collezione = CC.id_collezione AND CC.id_collezionista =
                    p_id_collezionista))
32              OR
33              (p_includi_pubbliche = 1 AND EXISTS (SELECT * FROM Copia C, Collezione CZ WHERE
                    C.id_disco = D.id AND C.id_collezione = CZ.id AND CZ.isPubblica = 1 AND
                    CZ.id_collezionista != p_id_collezionista))); -- escludo le collezioni pubbliche che
                    appartengono al collezionista che fa la query
34          END IF;
35      END$$
```

9. Verifica visibilità collezione

```
1       DROP PROCEDURE IF EXISTS CheckVisibilita$$
2
3       CREATE PROCEDURE CheckVisibilita(
4           IN p_nome_collezione VARCHAR(50),
5           IN p_nickname_owner VARCHAR(50),
6           IN p_nickname_guest VARCHAR(50)
7       )
8       BEGIN
9           DECLARE p_id_collezione INT UNSIGNED;
10          SET p_id_collezione = GetCollezione(p_nome_collezione, p_nickname_owner);
11
12          IF (SELECT id FROM Collezionista WHERE nickname = p_nickname_guest) IS NULL THEN
13              SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Il collezionista indicato non esiste';
14          ELSEIF p_id_collezione IS NULL THEN
15              SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'La collezione indicata non esiste';
16          ELSE
17              IF (SELECT ((p_nickname_owner = p_nickname_guest) OR (SELECT isPubblica FROM Collezione
                    WHERE id = p_id_collezione) = TRUE OR EXISTS (SELECT * FROM Condivide WHERE
                    id_collezione = p_id_collezione AND id_collezionista = (SELECT id FROM Collezionista
                    WHERE nickname = p_nickname_guest)))) = true THEN
18                      SELECT "Visibile" AS Risultato;
19                  ELSE
20                      SELECT "Non visibile" AS Risultato;
21                  END IF;
22          END IF;
23      END$$
```

10. Numero di brani di un autore nelle collezioni pubbliche

```
1       DROP PROCEDURE IF EXISTS ContaBraniByAutore$$
```

```
CREATE PROCEDURE ContaBraniByAutore(
    IN p_nomeDarte VARCHAR(50)
)
BEGIN
    DECLARE p_id_autore INT UNSIGNED;
    SET p_id_autore = (SELECT id FROM Autore WHERE nomeDarte = p_nomeDarte);

    IF p_id_autore IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'L autore indicato non esiste';
    ELSE
        SELECT count(DISTINCT T.id) AS "Numero tracce" FROM Traccia T, Disco D, Contiene C,
            Collezione CZ, Copia CO, Contribuisce CB, Collabora CL WHERE
            ((T.id = CB.id_traccia AND CB.id_autore = p_id_autore) OR (D.id = CL.id_disco AND
                CL.id_autore = p_id_autore AND T.id = C.id_traccia AND C.id_disco = D.id) OR
                (D.id_autore = p_id_autore AND T.id = C.id_traccia AND C.id_disco = D.id)) AND
            T.id = C.id_traccia AND C.id_disco = D.id AND CO.id_disco = D.id AND
                CO.id_collezione = CZ.id AND CZ.isPubblica = true;
    END IF;
END$$
```

11. Tempo di musica di un autore nelle collezioni pubbliche

```
DROP PROCEDURE IF EXISTS TempoBraniByAutore$$

CREATE PROCEDURE TempoBraniByAutore(
    IN p_nomeDarte VARCHAR(50)
)
BEGIN
    DECLARE p_id_autore INT UNSIGNED;
    SET p_id_autore = (SELECT id FROM Autore WHERE nomeDarte = p_nomeDarte);

    IF p_id_autore IS NULL THEN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'L autore indicato non esiste';
    ELSE
        SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(durata))) AS "Tempo di musica"
            FROM (SELECT DISTINCT T.id, T.durata AS durata FROM Traccia T, Disco D, Contiene C,
                Collezione CZ, Copia CO, Contribuisce CB, Collabora CL WHERE
            ((T.id = CB.id_traccia AND CB.id_autore = p_id_autore) OR (D.id = CL.id_disco AND
                CL.id_autore = p_id_autore AND T.id = C.id_traccia AND C.id_disco = D.id) OR
                (D.id_autore = p_id_autore AND T.id = C.id_traccia AND C.id_disco = D.id)) AND
            T.id = C.id_traccia AND C.id_disco = D.id AND CO.id_disco = D.id AND
                CO.id_collezione = CZ.id AND CZ.isPubblica = true) as SubQuery;
    END IF;
END$$
```

12a. Numero di collezioni di ciascun collezionista

```
DROP PROCEDURE IF EXISTS ContaCollezioni$$

CREATE PROCEDURE ContaCollezioni()
BEGIN
    SELECT C.nickname, count(DISTINCT CO.id) as "Numero" FROM Collezionista C, Collezione CO
        WHERE CO.id_collezionista = C.id GROUP BY C.id;
END$$
```

12b. Numero di dischi per genere

```
DROP PROCEDURE IF EXISTS ContaDischiPerGenere$$

CREATE PROCEDURE ContaDischiPerGenere()
BEGIN
```

```
    SELECT G.nome, count(DISTINCT D.id) as "Numero" FROM Genere G, Classifica C, Disco D WHERE
        C.id_disco = D.id AND C.nome_genere = G.nome GROUP BY G.nome;
END$$
```

13. Ricerca coerente con i dati in input

```
DROP PROCEDURE IF EXISTS RicercaCoerente$$

CREATE PROCEDURE RicercaCoerente(
    IN p_EAN CHAR(13),
    IN p_titolo VARCHAR(50),
    IN p_nomeDarte VARCHAR(50)
)
-- Utilizzo due volte soundex per rendere il range di compatibilita piu ampio, l'ordine delle
    query e definito dall ordine di importanza dei criteri usati
BEGIN
    IF p_titolo = "" THEN
        SET p_titolo = null; -- altrimenti cercando con il titolo vuoto mi verrebbero restituiti
            tutti i dischi del sistema
    END IF;

    IF p_nomeDarte = "" THEN
        SET p_nomeDarte = null; -- altrimenti cercando con il nome d'arte vuoto mi verrebbero
            restituiti tutti i dischi del sistema
    END IF;

    -- MATCH con i dischi con lo stesso EAN
    (SELECT D.*, A.nomeDarte AS "Autore principale" FROM Disco D, Autore A WHERE D.EAN = p_EAN
        AND A.id = D.id_autore )
    UNION
    -- MATCH con i dischi che contengono la parola cercata come titolo e come autore principale
    (SELECT D.*, A.nomeDarte AS "Autore principale" FROM Disco D, Autore A WHERE D.titolo LIKE
        CONCAT("%",p_titolo,"%")
    AND D.id_autore = A.id AND A.nomeDarte LIKE CONCAT("%",p_nomeDarte,"%"))
    UNION
    -- MATCH con i dischi che contengono la parola cercata come titolo e si avvicinano
        foneticamente alla parola cercata come autore nell'autore principale
    (SELECT D.*, A.nomeDarte AS "Autore principale" FROM Disco D, Autore A WHERE D.titolo LIKE
        CONCAT("%",p_titolo,"%") AND A.id = D.id_autore AND soundex(soundex(A.nomeDarte)) =
        soundex(soundex(p_nomeDarte)))
    UNION
    -- MATCH con i dischi che contengono la parola cercata come autore nell'autore principale e
        si avvicinano foneticamente alla parola cercata come titolo nel titolo
    (SELECT D.*, A.nomeDarte AS "Autore principale" FROM Disco D, Autore A WHERE
        soundex(soundex(D.titolo)) = soundex(soundex(p_titolo)) AND D.id_autore = A.id AND
        A.nomeDarte LIKE CONCAT("%",p_nomeDarte,"%"))
    UNION
    -- MATCH con i dischi che si avvicinano foneticamente alla parola cercata sia come titolo
        nel titolo che come autore nell'autore principale
    (SELECT D.*, A.nomeDarte AS "Autore principale" FROM Disco D, Autore A WHERE
        soundex(soundex(D.titolo)) = soundex(soundex(p_titolo)) AND D.id_autore = A.id AND
        soundex(soundex(A.nomeDarte)) = soundex(soundex(p_nomeDarte)))
    UNION
    -- MATCH con i dischi che contengono la parola cercata come titolo e come autore secondario
    (SELECT D.*, AP.nomeDarte AS "Autore principale" FROM Disco D, Collabora C, Autore A, Autore
        AP WHERE D.titolo LIKE CONCAT("%",p_titolo,"%")
    AND D.id = C.id_disco AND C.id_autore = A.id AND A.nomeDarte LIKE
        CONCAT("%",p_nomeDarte,"%") AND AP.id = D.id_autore)
    UNION
    -- MATCH con i dischi che contengono la parola cercata come autore nell'autore secondario e
        si avvicinano foneticamente alla parola cercata come titolo nel titolo
    (SELECT D.*, AP.nomeDarte AS "Autore principale" FROM Disco D, Collabora C, Autore A, Autore
        AP WHERE soundex(soundex(D.titolo)) = soundex(soundex(p_titolo))
    AND D.id = C.id_disco AND C.id_autore = A.id AND A.nomeDarte LIKE
        CONCAT("%",p_nomeDarte,"%") AND AP.id = D.id_autore)
```

```
41          UNION
42          -- MATCH con i dischi che contengono la parola cercata come titolo nel titolo e si
                avvicinano foneticamente alla parola cercata come autore nell'autore secondario
43          (SELECT D.*, AP.nomeDarte AS "Autore principale" FROM Disco D, Collabora C, Autore A, Autore
                AP WHERE D.titolo LIKE CONCAT("%",p_titolo,"%")
44          AND D.id = C.id_disco AND C.id_autore = A.id AND soundex(soundex(A.nomeDarte)) =
                soundex(soundex(p_nomeDarte)) AND AP.id = D.id_autore )
45          UNION
46          -- MATCH con i dischi che si avvicinano foneticamente alla parola cercata sia come titolo
                nel titolo che come autore nell'autore secondario
47          (SELECT D.*, AP.nomeDarte AS "Autore principale" FROM Disco D, Collabora C, Autore A, Autore
                AP WHERE soundex(soundex(D.titolo)) = soundex(soundex(p_titolo))
48          AND D.id = C.id_disco AND C.id_autore = A.id AND soundex(soundex(A.nomeDarte)) =
                soundex(soundex(p_nomeDarte)) AND AP.id = D.id_autore)
49          UNION
50          -- MATCH con i dischi che si avvicinano foneticamente alla parola cercata come autore
                nell'autore principale e con qualsiasi titolo
51          (SELECT D.*, A.nomeDarte AS "Autore principale" FROM Disco D, Autore A WHERE D.id_autore =
                A.id AND soundex(soundex(A.nomeDarte)) = soundex(soundex(p_nomeDarte)))
52          UNION
53          -- MATCH con i dischi che si avvicinano foneticamente alla parola cercata come autore negli
                autori secondari e con qualsiasi titolo
54          (SELECT D.*, AP.nomeDarte AS "Autore principale" FROM Disco D, Collabora C, Autore A, Autore
                AP WHERE D.id = C.id_disco
55          AND C.id_autore = A.id AND soundex(soundex(A.nomeDarte)) = soundex(soundex(p_nomeDarte)) AND
                AP.id = D.id_autore)
56          UNION
57          -- MATCH con i dischi che si avvicinano foneticamente alla parola cercata come titolo nel
                titolo e con qualsiasi autore
58          (SELECT D.*, A.nomeDarte AS "Autore principale" FROM Disco D, Autore A WHERE
                (soundex(soundex(D.titolo)) = soundex(soundex(p_titolo))
59          OR D.titolo LIKE CONCAT("%",p_titolo,"%")) AND A.id = D.id_autore);
60      END$$
```

Fine del file

```
1       DELIMITER ;
```

6 Inserimento dati

In questa sezione troviamo il codice di popolazione del database, con alcuni esempi il più possibile realistici.

```
USE collectors;

DELETE FROM Copia;
DELETE FROM collabora;
DELETE FROM Autore;
DELETE FROM Collezione;
DELETE FROM Collezionista;
DELETE FROM Condivide;
DELETE FROM Contiene;
DELETE FROM Contribuisce;
DELETE FROM Disco;
DELETE FROM Etichetta;
DELETE FROM Formato;
DELETE FROM Genere;
DELETE FROM Immagine;
DELETE FROM Posizione;
DELETE FROM Stato;
DELETE FROM Tipo;
DELETE FROM Traccia;

INSERT INTO Autore(nomeDarte,nome,cognome,isGruppo) VALUE ("The Rolling Stones",null,null,true);
INSERT INTO Autore(nomeDarte,nome,cognome,isGruppo) VALUE ("Vasco Rossi","Vasco","Rossi",false);
INSERT INTO Autore(nomeDarte,nome,cognome,isGruppo) VALUE
    ("Annalisa","Annalisa","Scarrone",false);
INSERT INTO Autore(nomeDarte,nome,cognome,isGruppo) VALUE ("Zef","Stefano","Tognini",false);

INSERT INTO Collezionista(nickname,email) VALUES
    ("pippo123","pippo123@ex.it"),
    ("pluto456","pluto456@ex.it"),
    ("mickey789","mickey789@ex.it");

CALL InserisciCollezione("Rock",false,"pippo123");
CALL InserisciCollezione("Rock",true,"pluto456");
CALL InserisciCollezione("Vinili Pop",false,"pluto456");
CALL InserisciCollezione("La mia preferita",true,"mickey789");

INSERT INTO Condivide VALUES
    ((SELECT C.ID FROM Collezionista C Where C.nickname = "pluto456"),(SELECT C.ID FROM
        Collezione C Where C.nome = "Rock" AND C.id_collezionista = ((SELECT C.ID FROM
        Collezionista C Where C.nickname = "pippo123")))),
    ((SELECT C.ID FROM Collezionista C Where C.nickname = "mickey789"),(SELECT C.ID FROM
        Collezione C Where C.nome = "Rock" AND C.id_collezionista = ((SELECT C.ID FROM
        Collezionista C Where C.nickname = "pippo123")))),
    ((SELECT C.ID FROM Collezionista C Where C.nickname = "pippo123"),(SELECT C.ID FROM
        Collezione C Where C.nome = "Vinili Pop" AND C.id_collezionista = ((SELECT C.ID FROM
        Collezionista C Where C.nickname = "pluto456"))));

INSERT INTO Etichetta(nome) VALUES
    ("Polydor Records"),
    ("Carosello"),
    ("Warner Music Italy");

INSERT INTO Formato(nome) VALUES
    ("CD"),
    ("Vinile"),
    ("Digitale"),
    ("Cassetta");

INSERT INTO Genere(nome) VALUES
    ("Rock"),
    ("Pop"),
    ("Blues");

```

```
INSERT INTO Traccia(titolo,ISRC,Durata) VALUES
--ISRC trovati su isrcsearch.ifpi.org
    ("Just Your Fool","GBUM71604197",000216),
    ("Commit A Crime","GBUM71604638",000338),
    ("I Gotta Go","GBUM71604634",000326),
    ("Vita Spericolata", "ITB269999067",000442),
    ("Bollicine","ITB260000024",000536),
    ("Vado Al Massimo","ITB268299062",000340),
    ("Mon Amour","ITQ002300239",000323);

INSERT INTO Tipo VALUES
    ("Cantante"),
    ("Produttore"),
    ("Testo");


    -- Alcuni EAN non sono reali in quanto non reperibili sul web
CALL InserisciDisco("0602557149463","Blue&Lonesome Deluxe Edition",2016, "Polydor Records",
    "CD", "The Rolling Stones", "Rock");
CALL InserisciDisco("0834738293948","Mon Amour",2023, "Warner Music Italy",
    "Digitale","Annalisa","Pop");
CALL InserisciDisco("8032529710025","Vado al massimo", 1982, "Carosello","CD","Vasco
    Rossi","Pop");
CALL InserisciDisco("8034125847372","Bollicine", 1983, "Carosello", "Vinile", "Vasco
    Rossi","Pop");
CALL AggiungiGenereAlDisco("Blues","0602557149463");

CALL AggiungiAutoreAlDisco("Vasco Rossi","0834738293948","Testo");


CALL TracciaInDisco("ITQ002300239","0834738293948");
-- Mon Amour in Mon Amour
CALL TracciaInDisco("GBUM71604197","0602557149463");
-- Just Your Fool in Blue&Lonesome Deluxe Edition
CALL TracciaInDisco("GBUM71604638","0602557149463");
-- Commit a Crime in Blue&Lonesome Deluxe Edition
CALL TracciaInDisco("GBUM71604634","0602557149463");
-- I Gotta Go in Blue&Lonesome Deluxe Edition
CALL TracciaInDisco("ITB269999067","8034125847372");
-- Vita spericolata in Bollicine
CALL TracciaInDisco("ITB268299062","8032529710025");
-- Vado al massimo in Vado al massimo
CALL TracciaInDisco("ITB260000024","8034125847372");
-- Bollicine in Bollicine

CALL NuovaTracciaInDisco("Bellissima","ITQ002200517",000321,"0834738293948");
-- Bellissima in Mon Amour

INSERT INTO Contribuisce VALUES
    ((SELECT A.id FROM Autore A WHERE A.nomeDarte="Zef"),(SELECT T.id FROM Traccia T WHERE
        T.ISRC="ITQ002300239"),"Testo");

INSERT INTO Stato VALUES
    ("Nuovo"),
    ("Ottimo"),
    ("Buono"),
    ("Scarso"),
    ("Pessimo");

CALL InserisciCopia("000000000001", "0834738293948", "Ottimo", "La Mia Preferita","Mickey789");
-- Mon Amour
CALL InserisciCopia("000000000002", "0602557149463", "Ottimo", "Rock","Pluto456");
-- Blue&Lonesome Deluxe Edition
CALL InserisciCopia("000000000003", "0834738293948", "Buono", "La Mia Preferita","Mickey789");
-- Mon Amour
CALL InserisciCopia("000000000004", "0602557149463", "Nuovo", "Rock" , "Pippo123");
```

```
    -- Blue&Lonesome Deluxe Edition
    CALL InserisciCopia("000000000005", "0602557149463", "Pessimo", "Rock", "Pippo123");
    -- Blue&Lonesome Deluxe Edition
    CALL InserisciCopia("000000000006", "8034125847372", "Scarso","Rock","Pippo123");
    -- Bollicine
    CALL InserisciCopia("000000000008", "8034125847372", "Scarso","Rock","Pluto456");
    -- Bollicine
    CALL InserisciCopia("000000000007", "8032529710025", "Ottimo","La Mia Preferita","Mickey789");
    -- Vado al massimo

    INSERT INTO Posizione VALUES
        ("Fronte"),
        ("Retro"),
        ("Libretto"),
        ("Digitale"),
        ("Fronte disco");

    INSERT INTO Immagine(URL,id_Disco,nome_Posizione) VALUES
        ("https://www.google.com/url?sa=i&url=https%3A%2F%2F
        music.amazon.it%2Falbums%2FB0BZ6LJRPP&psig=AOvVaw0-b4Fo8NERBp_dPyt-TX-P&us
        t=1687701656779000&source=images&cd=vfe&ved=0CBEQjRxqFwoTCOCnnd-I3P8CFQAAAAAdAAAAABAE",(SELECT
            D.id FROM Disco D WHERE D.EAN = "0834738293948"),"Digitale");
```