LABORATORIO DI BASE DI DATI: ***PROGETTO “COLLECTORS”***

**Gruppo di lavoro:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Matricola** | **Nome** | **Cognome** | **Contributo al progetto** |
| 288804 | Francesca | Ciccarelli | Progettazione ER + relazionale, Implementazione MySQL, Interfaccia Java |
| 246996 | Federico | Falcone | Progettazione ER + relazionale, Implementazione MySQL |
| 278439 | Gea | Viozzi | Progettazione ER + relazionale, Implementazione MySQL, Interfaccia Java |

**Data di consegna del progetto:** 28/06/2023

* **Analisi dei requisiti**

ll database "Collectors" memorizza informazioni relative a collezioni di dischi. Nel database andranno registrati i dati relativi ai collezionisti (nickname, email, anagrafica) e i dati relativi alle loro collezioni (ogni collezionista può creare più collezioni, ognuna con un nome distinto). Ogni collezione è formata da dischi, per i quali abbiamo differenziato la tipologia come CD, vinile, musicassetta, EP e digitale. In aggiunta abbiamo indicato titolo, anno di uscita, etichetta, stato di conservazione e barcode (se presente). Ogni disco è inoltre caratterizzato da una o più immagini che possono trovarsi su fronte, retro, facciate interne, libretti ecc. ed è composto da brani (tracce), dove andranno specificati titolo, durata, esecutore/i, compositore/i e genere. Il collezionista può inoltre possedere una o più copie di uno stesso disco, magari a seguito di scambi o perchè ne prevede la rivendita. I collezionisti possono condividere la propria collezione con specifici utenti o in maniera pubblica; ogni collezione avrà quindi un flag privato/pubblico e la lista di collezionisti con cui è condivisa.

In seguito riportiamo un glossario dei termini, relativo al primo approccio che abbiamo avuto con il progetto:

|  |  |  |
| --- | --- | --- |
| **Termine** | **Descrizione** | **Collegamenti** |
| Collezionista | Utente che può creare collezioni e che può decidere se condividerle con altri utenti. Include anche i collezionisti con cui vengono condivise le collezioni. | Collezione |
| Collezione | Insieme di dischi posseduto dal collezionista. | Collezionista, Disco |
| Disco | Oggetto contenuto in una collezione, che può essere gestito in diverse copie. | Collezione, Etichetta, Brano, Artista |
| Brano | Unità che compone un disco. | Disco, Artista |

* **Progettazione concettuale**

Nel file "Allegato1" riportiamo il diagramma ER realizzato. Di seguito riportiamo il dizionario relativo al diagramma ER:

**Entità**

|  |  |  |  |
| --- | --- | --- | --- |
| **Entità** | **Descrizione** | **Attributi** | **Identificatore** |
| Collezionista | Utente che crea collezioni | email, nickname, anagrafica¹ | nickname |
| Collezione | Insieme di dischi posseduti dal collezionista | nome, flag² | nome, Collezionista |
| Artista | Colui che compone e/o esegue il brano | anagrafica³ |  |
| Disco | Oggetto contenuto in una collezione | titolo, anno\_uscita, tipo⁴,barcode, stato\_di\_conservazione | titolo, anno\_uscita, Artista |
| Brano | Traccia che compone un disco | ISRC, durata, titolo, genere⁵ | ISRC |
| Solista | Singola persona che forma l’artista | nome\_dArte | nome\_dArte |
| Gruppo | Insieme di persone che formano l’artista | nome\_dArte | nome\_dArte |
| Etichetta | Azienda che produce un disco | nome, p\_iva | nome |
| Immagine | Figura presente sul/nel disco | sorgente, collocazione, par\_tecnici⁶ | sorgente |
| ¹ nome, cognome | | ⁴ CD, vinile, EP, musicassetta, digitale | |
| ² privato, pubblico | | ⁵ rock, pop, metal, blues, musica classica, altro | |
| ³ nome, cognome | | ⁶ nome\_file, formato\_file, dimensione\_file | |

Per l’anagrafica abbiamo preferito non appesantire la modelizzazione con dati non rilevanti per il nostro progetto come data di nascita o indirizzo; lo stesso ragionamento è stato utilizzato per l’attributo genere dove abbiamo specificato una piccola parte dei possibili generi musicali, aggiungendo poi l’attributo “altro”.

**Relazioni**

|  |  |  |  |
| --- | --- | --- | --- |
| **Relazione** | **Descrizione** | **Entità coinvolte** | **Attributi** |
| crea | Associa un collezionista in quanto possessore e creatore ad una collezione | Collezionista(1,N), Collezione(1,1) |  |
| condivisa | Associa una collezione ad un collezionista con il quale è condivisa | Collezionista(0,N), Collezione(1,1) |  |
| contiene | Associa una collezione con un disco | Collezione(0,N), Disco(1,1) | quantita |
| pubblica | Associa un disco con un artista | Disco(1,1), Artista(1,N) |  |
| compone | Associa un artista ad un brano in quanto compositore | Artista(1,N), Brano(1,N) |  |
| esegue | Associa un artista ad un brano in quanto esecutore | Artista(1,N), Brano(1,N) |  |
| forma | Associa un brano ad un disco | Brano(1,1), Disco(1,N) |  |
| produce | Associa un disco ad un’etichetta | Disco(1,1), Etichetta(1,N) |  |
| raffigura | Associa un disco ad un’immagine | Disco(0,N), Immagine(1,1) |  |

**Generalizzazioni**

|  |  |  |
| --- | --- | --- |
| **Classe Padre** | **Classi Figlie** | **Tipologia della generalizzazione** |
| Artista | Solista, Gruppo | Totale |

**Formalizzazione dei vincoli non esprimibili nel modello ER**

* Il nickname associato ad ogni *Collezionista* deve essere unico e non nullo;
* Il nome della *Collezione* che viene creata dal *Collezionista* deve essere unico e non nullo;
* Il nome dell’*Etichetta* deve essere unico e non nullo;
* Il numero di *Collezioni* condivise non può superare il numero di *Collezioni* totali possedute dal *Collezionista*;
* Un *Collezionista* non può condividere la stessa *Collezione* con uno stesso utente con il quale è stata già precedentemente condivisa;
* Una *Collezione* non può essere formata da un numero di dischi superiore a quelli posseduti;
* Un *Disco* deve necessariamente far parte di una *Collezione* che è stata creata dal *Collezionista*.
* **Progettazione logica**

**Ristrutturazione ed ottimizzazione del modello ER**

Nel file "Allegato2" riportiamo il diagramma ER ristrutturato. Le ristrutturazioni che sono state effettuate sono le seguenti:

**Eliminazione degli attributi multivalore**

* L'attributo *anagrafica* dell'entità *Collezionista* è stato esploso in due nuovi attributi: *nome, cognome;*
* L'attributo *anagrafica* dell'entità *Artista* è stato esploso in due nuovi attributi: *nome*, *cognome*;
* L'attributo *par\_tecnici* dell'entità *Immagine* è stato esploso nei seguenti nuovi attributi: *nome\_file, dimensione\_file, formato\_file.*

**Eliminazione delle gerarchie**

* La generalizzazione totale con classe padre *Artista* e classi figlie *Solista* e *Gruppo* è stata eliminata; le classi figlie sono state fuse con la classe padre ed è stato aggiunto all’entità *Artista*  un attributo booleano "gruppo", che sarà True se l'artista considerato è un gruppo, False se è un solista; l’entità *Artista* acquisisce inoltre l’attributo *nome\_dArte* che diventa chiave.

**Fusione/Decomposizione di entità e relazioni**

* L'attributo *tipo* legato all'entità *Disco* è specifico per ogni disco che possiede il collezionista, poichè è possibile possedere lo stesso disco in formati/tipi diversi (ad esempio un CD ed un vinile); per questo motivo è stata creata una nuova entità *Tipo* con l'attributo *nome*, legata a *Disco* tramite la relazione *descrive*;
* Le relazioni *esegue* e *compone* che legano gli attributi *Artista* e *Brano* rappresentano le stesse informazioni, distinguendo solo se l'artista considerato è il compositore e/o esecutore del brano. Il modello ER è stato ristrutturato considerando una singola relazione *appartiene* che ha come attributo un flag enum(esecutore/compositore/entrambi);
* L'attributo *genere* relativo all'entità *Brano* può contenere un'ampia lista di generi musicali in continuo aggiornamento. Per gestire la nascita di nuovi generi musicali, è stata creata un'entità *Genere* che ha come attributo *nome*, che abbiamo preferito legare all’entità *Disco* tramite la relazione *caratterizza*.

**Altre ristrutturazioni**

* È possibile possedere più copie di uno stesso disco, ed ogni copia può avere uno stato di conservazione diverso; poiché creare un’entità *Copia* con attributi *barcode* e *stato\_di\_conservazione* avrebbe introdotto overhead sulla base di dati senza effettivamente aumentarne l’efficienza, si è deciso di denormalizzare i dati dell’entità *Disco,* aggiungendo ridondanza alla stessa tale che, ogni copia, con diverso stato di conservazione, venga considerata come una singola entrata della tabella;
* Per facilitare la modifica dell'entità *Disco* da parte del *Collezionista* abbiamo creato la relazione *possiede* che associa ad una singola copia del disco uno ed un solo collezionista.

**Traduzione del modello ER nel modello relazionale**

Riportiamo qui di seguito il modello relazionale relativo allo schema ER ristrutturato.

In **grassetto** sono evidenziate le chiavi primarie mentre le chiavi esterne sono riportate sottolineate.

* Collezionista(**ID**, nickname, email, nome, cognome)
* condivisa(**ID\_collezionista**, **ID\_collezione**)
* Collezione(**ID**, ID\_collezionista, nome, stato)
* Artista(**ID**, nome\_dArte, nome, cognome, gruppo)
* Disco(**ID**, barcode, stato\_di\_conservazione, titolo, ID\_artista, ID\_etichetta, ID\_collezionista, ID\_collezione, ID\_genere, ID\_tipo, anno\_uscita)
* Tipo(**ID**, nome)
* Genere(**ID**, nome)
* Brano(**ID**, ISRC, durata, titolo, ID\_disco)
* appartiene(**ID\_artista**, **ID\_brano**, flag)
* Etichetta(**ID**, p\_iva, nome)
* Immagine(**ID**, nome, dimensione, formato, url\_sorgente, collocazione, ID\_disco)
* **Progettazione fisica**

**Implementazione del modello relazionale**

***Struttura del DB***

DROP DATABASE IF EXISTS Collectors;

CREATE DATABASE IF NOT EXISTS Collectors;

USE Collectors;

DROP TABLE IF EXISTS collezionista;

DROP TABLE IF EXISTS collezione;

DROP TABLE IF EXISTS artista;

DROP TABLE IF EXISTS disco;

DROP TABLE IF EXISTS brano;

DROP TABLE IF EXISTS etichetta;

DROP TABLE IF EXISTS immagine;

DROP TABLE IF EXISTS condivisa;

DROP TABLE IF EXISTS quantizza;

DROP TABLE IF EXISTS appartiene;

DROP TABLE IF EXISTS genere;

DROP TABLE IF EXISTS tipo;

-- Creiamo l'utente che accederà ai dati

DROP USER IF EXISTS 'collectorsUser'@'localhost';

CREATE USER 'collectorsUser'@'localhost' IDENTIFIED BY 'collectorsPwd€123';

GRANT select,insert,update,delete,execute ON Collectors.\* TO 'collectorsUser'@'localhost';

/\* Abbiamo deciso di impostare tutti i VARCHAR ad una lunghezza di 100 caratteri per dare

più libertà agli utenti che utilizzeranno l'applicazione. \*/

CREATE TABLE collezionista(

 ID INTEGER UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

 nickname VARCHAR(100) NOT NULL UNIQUE,

 email VARCHAR(100) NOT NULL UNIQUE,

 nome VARCHAR(100) NOT NULL,

 cognome VARCHAR(100) NOT NULL

    );

/\*In riferimento alla chiave esterna del collezionista, vogliamo che in caso di cancellazione

di un collezionista la collezione venga eliminata. \*/

CREATE TABLE collezione(

ID INTEGER UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

nome VARCHAR(100) NOT NULL UNIQUE,

stato enum('pubblico','privato') DEFAULT 'privato',

ID\_collezionista INTEGER UNSIGNED NOT NULL,

CONSTRAINT collezione\_collezionista FOREIGN KEY (ID\_collezionista)

        REFERENCES collezionista (ID)

        ON DELETE RESTRICT ON UPDATE CASCADE

    );

CREATE TABLE artista(

 ID INTEGER UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

 nome\_dArte VARCHAR(100) NOT NULL,

 nome VARCHAR(100),

 cognome VARCHAR(100),

 gruppo BOOLEAN

    );

CREATE TABLE etichetta(

 ID INTEGER UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

 p\_iva VARCHAR(100) UNIQUE,

 nome VARCHAR(100) UNIQUE NOT NULL

    );

CREATE TABLE genere(

    ID INTEGER UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

    nome VARCHAR(100)

    );

CREATE TABLE tipo(

    ID INTEGER UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

    nome VARCHAR(100)

    );

/\* In riferimento alle chiavi esterne di artista, etichetta, genere e tipo, vogliamo che non sia possibile cancellare dal db

un artista, un'etichetta, un genere o un tipo a cui sia referenziato un disco; per le chiavi esterne di collezionista e collezione

invece vogliamo che, in caso di cancellazione di una delle due tabelle, venga eliminato anche il disco.  \*/

CREATE TABLE disco(

 ID INTEGER UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

 barcode VARCHAR(100),

 stato\_di\_conservazione ENUM ('OTTIMO','BUONO','USURATO'),

 titolo VARCHAR(100) NOT NULL,

 ID\_artista INTEGER UNSIGNED NOT NULL,

 ID\_etichetta INTEGER UNSIGNED NOT NULL,

 ID\_collezionista INTEGER UNSIGNED NOT NULL,

 ID\_collezione INTEGER UNSIGNED NOT NULL,

 ID\_genere INTEGER UNSIGNED NOT NULL,

 ID\_tipo INTEGER UNSIGNED NOT NULL,

 anno\_uscita INTEGER UNSIGNED,

 CONSTRAINT disco\_artista FOREIGN KEY (ID\_artista)

        REFERENCES artista (ID)

        ON DELETE RESTRICT ON UPDATE CASCADE,

  CONSTRAINT disco\_etichetta FOREIGN KEY (ID\_etichetta)

        REFERENCES etichetta (ID)

        ON DELETE RESTRICT ON UPDATE CASCADE,

CONSTRAINT disco\_collezionista FOREIGN KEY (ID\_collezionista)

        REFERENCES collezionista (ID)

        ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT disco\_collezione FOREIGN KEY (ID\_collezione)

        REFERENCES collezione (ID)

        ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT disco\_genere FOREIGN KEY (ID\_genere)

        REFERENCES genere (ID)

        ON DELETE RESTRICT ON UPDATE CASCADE,

CONSTRAINT disco\_tipo FOREIGN KEY (ID\_tipo)

        REFERENCES tipo (ID)

        ON DELETE RESTRICT ON UPDATE CASCADE

    );

/\* In riferimento alla chiave esterna disco, vogliamo che in caso venga eliminato un disco sia eliminato anche il brano. \*/

/\* L'ISRC non è unico per via del  fatto che abbiamo aggiunto la possibilità di avere più copie dello stesso

   disco contenenti gli stessi brani. Per di più abbiamo preso in considerazione la possibilità che uno stesso brano possa

   essere in dischi diversi. \*/

CREATE TABLE brano(

 ID INTEGER UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

 ISRC VARCHAR(100) NOT NULL,

 durata TIME,

 titolo VARCHAR(100),

 ID\_disco INTEGER UNSIGNED NOT NULL,

CONSTRAINT brano\_disco FOREIGN KEY (ID\_disco)

        REFERENCES disco (ID)

        ON DELETE CASCADE ON UPDATE CASCADE

    );

/\* In riferimento alla chiave esterna disco, vogliamo che in caso venga eliminato un disco sia eliminata anche l'immagine

ad esso riferita. \*/

CREATE TABLE immagine(

 ID INTEGER UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

 nome VARCHAR(100),

 dimensione VARCHAR(100),

 formato VARCHAR(100),

 url\_sorgente VARCHAR(255),

 collocazione VARCHAR(100),

 ID\_disco INTEGER UNSIGNED NOT NULL,

 CONSTRAINT immagine\_disco FOREIGN KEY (ID\_disco)

        REFERENCES disco (ID)

        ON DELETE CASCADE ON UPDATE CASCADE

    );

/\* In caso di cancellazione di una collezione e/o di un collezionista, viene eliminata anche

la condivisione che vi è tra quella collezione e quel collezionista. \*/

CREATE TABLE condivisa(

    ID\_collezionista INTEGER UNSIGNED NOT NULL,

    ID\_collezione INTEGER UNSIGNED NOT NULL,

    PRIMARY KEY (ID\_collezionista , ID\_collezione),

    CONSTRAINT condivisa\_collezione FOREIGN KEY (ID\_collezione)

        REFERENCES collezione (ID)

        ON DELETE CASCADE ON UPDATE CASCADE,

    CONSTRAINT condivisa\_collezionista FOREIGN KEY (ID\_collezionista)

        REFERENCES collezionista (ID)

        ON DELETE CASCADE ON UPDATE CASCADE

    );

 /\* Se l'artista è referenziato da un brano non possiamo eliminarlo mentre, se viene eliminato il brano,

 vogliamo che venga eliminata la relazione tra il brano e l'artista. \*/

 CREATE TABLE appartiene(

  ID\_artista INTEGER UNSIGNED NOT NULL,

  ID\_brano INTEGER UNSIGNED NOT NULL,

  flag ENUM("ESECUTORE","COMPOSITORE","ENTRAMBI") DEFAULT "ESECUTORE",

  PRIMARY KEY (ID\_artista , ID\_brano),

    CONSTRAINT appartiene\_artista FOREIGN KEY (ID\_artista)

        REFERENCES artista (ID)

        ON DELETE RESTRICT ON UPDATE CASCADE,

    CONSTRAINT appartiene\_brano FOREIGN KEY (ID\_brano)

        REFERENCES brano (ID)

        ON DELETE CASCADE ON UPDATE CASCADE

    );

***Popolamento del DB***

-- collezionista(ID, nickname, email, nome, cognome)

INSERT INTO `collezionista`

VALUES(1, "jess\_vzz", "geaviozzi@icloud.com", "Gea", "Viozzi");

INSERT INTO `collezionista`

VALUES(2, "user", "user@icloud.com", "User", "User");

INSERT INTO `collezionista`

VALUES(3, "fedefalco", "federico@icloud.com", "Federico", "Falcone");

INSERT INTO `collezionista`

VALUES(4, "leomasci", "leonardo@icloud.com", "Leonardo", "Masci");

INSERT INTO `collezionista`

VALUES(5, "fabdam", "fabio@icloud.com", "Fabio", "D'Andreamatteo");

INSERT INTO `collezionista`

VALUES(6, "cicca\_cicca", "francesca@icloud.com", "Francesca", "Ciccarelli");

-- collezione(ID, nome, flag, ID\_collezionista)

INSERT INTO `collezione`

VALUES(1, "cantautori italiani", "privato", 1);

INSERT INTO `collezione`

VALUES(2, "dischi rock", "pubblico", 2);

INSERT INTO `collezione`

VALUES(3, "in my feels","privato", 3);

INSERT INTO `collezione`

VALUES(4, "sunday", "privato", 5);

INSERT INTO `collezione`

VALUES(5, "work", "pubblico", 4);

INSERT INTO `collezione`

VALUES(6, "workout","privato", 3);

INSERT INTO `collezione`

VALUES(7, "study time","privato", 1);

INSERT INTO `collezione`

VALUES(8, "relax", "pubblico", 6);

-- artista(ID, nomeDarte, nome, cognome, gruppo)

INSERT INTO `artista`

VALUES(1, "Francesco Guccini", "Francesco", "Guccini", false);

INSERT INTO `artista`

VALUES(2, "Red Hot Chilli Pepper",  null, null, true);

INSERT INTO `artista`

VALUES(3, "System Of A Down",  null, null, true);

INSERT INTO `artista`

VALUES(4, "Taylor Swift",  "Taylor", "Swift", false);

INSERT INTO `artista`

VALUES(5, "Jon Bellion",  "Jon", "Bellion", false);

INSERT INTO `artista`

VALUES(6, "Eminem",  "Eminem", null , false);

INSERT INTO `artista`

VALUES(7, "Imagine Dragons", null, null , true);

INSERT INTO `artista`

VALUES(8, "Jovanotti",  "Lorenzo", "Cherubini" , false);

-- etichetta(ID, p\_iva, nome)

INSERT INTO `etichetta`

VALUES(1, "10199720151", "EMI Italia");

INSERT INTO `etichetta`

VALUES(2, "00896521002", "Warner Bros");

INSERT INTO `etichetta`

VALUES(3, "04913851004", "Sony");

-- genere(ID, nome)

INSERT INTO `genere`

VALUES(1, "pop");

INSERT INTO `genere`

VALUES(2, "rap");

INSERT INTO `genere`

VALUES(3, "metal");

INSERT INTO `genere`

VALUES(4, "rock");

INSERT INTO `genere`

VALUES(5, "altro");

-- tipo(ID, nome)

INSERT INTO `tipo`

VALUES (1, "CD");

INSERT INTO `tipo`

VALUES (2, "ALTRO");

INSERT INTO `tipo`

VALUES (3, "MUSICASSETTA");

INSERT INTO `tipo`

VALUES (4, "VINILE");

INSERT INTO `tipo`

VALUES (5, "DIGITALE");

-- disco(ID, barcode, stato\_di\_conservazione, titolo, ID\_artista, ID\_etichetta,

-- ID\_collezionista, ID\_collezione, ID\_genere, ID\_tipo, anno\_uscita)

INSERT INTO `disco`

VALUES(1, "123456789", "OTTIMO", "D'amore di morte e di altre sciocchezze", 1, 1, 1, 1, 1, 1, 1996);

INSERT INTO `disco`

VALUES(2, "123256789", "BUONO", "Disco1", 5, 3, 4, 4, 2, 3, 1995);

INSERT INTO `disco`

VALUES(3, "423456789", "OTTIMO", "Disco2", 1, 2, 3, 3, 3, 1, 2000);

INSERT INTO `disco`

VALUES(4, "145456789", "USURATO", "Disco3", 6, 1, 2, 2, 4, 5, 1970);

INSERT INTO `disco`

VALUES(5, "023456789", "BUONO", "Disco4", 2, 2, 1, 1, 1, 4, 1990);

INSERT INTO `disco`

VALUES(6, "123446789", "BUONO", "Disco5", 1, 1, 1, 1, 2, 2, 1976);

INSERT INTO `disco`

VALUES(7, "123456389", "OTTIMO", "Disco6", 1, 3, 4, 5, 3, 1, 1995);

INSERT INTO `disco`

VALUES(8, "123406789", "OTTIMO", "Disco7", 1, 1, 5, 4, 1, 3, 2000);

INSERT INTO `disco`

VALUES(9, "103406789", "USURATO", "Disco7", 1, 1, 5, 4, 1, 2, 2000);

INSERT INTO `disco`

VALUES(10, "103406489", "BUONO", "Disco7", 1, 1, 6, 8, 1, 2, 2000);

INSERT INTO `disco`

VALUES(11, "103406419", "BUONO", "Disco11", 8, 2, 2, 2, 1, 2, 1975);

INSERT INTO `disco`

VALUES(12, "100406489", "OTTIMO", "Disco12", 8, 3, 5, 7, 3, 1, 2017);

-- brano(ID, ISRC, durata, titolo, ID\_disco)

INSERT INTO `brano`

VALUES(1, "IT-D00-07-00111", "00:05:21", "Vorrei", 1);

INSERT INTO `brano`

VALUES(2, "IT-D00-07-00112", "00:04:30", "Brano1", 2);

INSERT INTO `brano`

VALUES(3, "IT-D00-07-00113", "00:03:43", "Brano2", 1);

INSERT INTO `brano`

VALUES(4, "IT-D00-07-00114", "00:02:23", "Brano3", 6);

INSERT INTO `brano`

VALUES(5, "IT-D00-07-00115", "00:05:03", "Brano4", 6);

INSERT INTO `brano`

VALUES(6, "IT-D00-07-00116", "00:04:31", "Brano5", 6);

INSERT INTO `brano`

VALUES(7, "IT-D00-07-00117", "00:02:31", "Brano6", 7);

INSERT INTO `brano`

VALUES(8, "IT-D00-07-00118", "00:03:31", "Brano7", 11);

INSERT INTO `brano`

VALUES(9, "IT-D00-07-00119", "00:04:01", "Brano8", 12);

INSERT INTO `brano`

VALUES(10, "IT-D00-07-00120", "00:02:51", "Brano9", 11);

INSERT INTO `brano`

VALUES(11, "IT-D00-07-00121", "00:03:07", "Brano10", 11);

-- condivisa(ID\_collezionista, ID\_collezione)

INSERT INTO `condivisa`

VALUES(2, 4);

INSERT INTO `condivisa`

VALUES(2, 1);

INSERT INTO `condivisa`

VALUES(2, 6);

-- appartiene(ID\_artista, ID\_brano, flag)

INSERT INTO `appartiene`

VALUES(1, 1, "ESECUTORE");

INSERT INTO `appartiene`

VALUES(1, 2, "ENTRAMBI");

INSERT INTO `appartiene`

VALUES(1, 3, "ESECUTORE");

INSERT INTO `appartiene`

VALUES(2, 4, "COMPOSITORE");

INSERT INTO `appartiene`

VALUES(3, 5, "ESECUTORE");

INSERT INTO `appartiene`

VALUES(4, 6, "ESECUTORE");

INSERT INTO `appartiene`

VALUES(4, 7, "COMPOSITORE");

***Trigger***

-- Trigger 1 --

-- una condivisione non puo' essere ricondivisa con lo stesso collezionista, con sè stessi oppure se è pubblica

DROP TRIGGER IF EXISTS check\_condivisione;

DELIMITER $$

CREATE TRIGGER check\_condivisione BEFORE INSERT ON condivisa

FOR EACH ROW

BEGIN

    DECLARE count SMALLINT UNSIGNED;

    DECLARE pubblica\_count SMALLINT UNSIGNED;

    DECLARE stesso\_collezionista\_count SMALLINT UNSIGNED;

    -- conto quante collezioni con il nuovo ID sono condivise con il nuovo collezionista

    SELECT COUNT(\*) INTO count

    FROM condivisa

    WHERE ID\_collezionista = NEW.ID\_collezionista AND ID\_collezione = NEW.ID\_collezione;

   -- conto le collezioni con il nuovo ID che sono pubbliche

    SELECT COUNT(\*) INTO pubblica\_count

    FROM collezione

    WHERE ID = NEW.ID\_collezione AND stato = 'pubblico';

  -- conto le collezioni con il nuovo ID che appartengono al collezionista

    SELECT COUNT(\*) INTO stesso\_collezionista\_count

    FROM collezione

    WHERE ID = NEW.ID\_collezione AND ID\_collezionista = NEW.ID\_collezionista;

-- se ho già quella collezione condivisa con me, o pubblica o è mia, ritorna errore

    IF ((count > 0) OR (pubblica\_count > 0) OR (stesso\_collezionista\_count > 0)) THEN

        SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Errore! Impossibile condividere questa collezione con il collezionista indicato.';

    END IF;

END$$

DELIMITER ;

-- Trigger 2 --

-- un collezionista non puo' avere più di una collezione con lo stesso nome.

DROP TRIGGER IF EXISTS check\_collezioni\_nomi;

DELIMITER $$

CREATE TRIGGER check\_collezioni\_nomi BEFORE INSERT ON collezione

FOR EACH ROW

BEGIN

    DECLARE count SMALLINT UNSIGNED;

    SELECT COUNT(\*) INTO count

    FROM collezione

    WHERE ID\_collezionista = NEW.ID\_collezionista AND nome = NEW.nome;

    IF (count > 0) THEN

        SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Errore! Impossibile utilizzare questo nome poichè è già stato utilizzato.';

    END IF;

END $$

DELIMITER ;

-- Trigger 3 --

DROP TRIGGER IF EXISTS check\_anno\_uscita;

DELIMITER $$

CREATE TRIGGER check\_anno\_uscita BEFORE INSERT ON disco

FOR EACH ROW

BEGIN

  IF(NEW.disco.anno\_uscita > YEAR(NOW())) THEN

        SIGNAL SQLSTATE "45000" SET MESSAGE\_TEXT="Errore nella data di uscita del disco.";

  END IF;

END $$

DELIMITER ;

***Query***

/\* QUERY 1: Inserimento di una nuova collezione \*/

DROP PROCEDURE IF EXISTS inserimento\_collezione;

DELIMITER $$

CREATE PROCEDURE inserimento\_collezione(ID\_collezionista INTEGER UNSIGNED, nome VARCHAR(100), stato ENUM("PUBBLICO","PRIVATO"))

    BEGIN

    INSERT INTO collezione(ID, ID\_collezionista, nome, stato)

    VALUES(ID, ID\_collezionista, nome, stato);

    END $$

DELIMITER ;

/\* QUERY 2a: Aggiunta di dischi ad una collezione \*/

DROP PROCEDURE IF EXISTS inserimento\_disco;

DELIMITER $$

    CREATE PROCEDURE inserimento\_disco(barcode VARCHAR(100), stato\_di\_conservazione ENUM("OTTIMO","BUONO","USURATO"), titolo VARCHAR(100), ID\_artista INTEGER UNSIGNED, ID\_etichetta INTEGER UNSIGNED, ID\_collezionista INTEGER UNSIGNED, ID\_collezione INTEGER UNSIGNED, ID\_genere INTEGER UNSIGNED, ID\_tipo INTEGER UNSIGNED, anno\_uscita INTEGER UNSIGNED)

    BEGIN

    INSERT INTO disco

    VALUES(ID, barcode, stato\_di\_conservazione, titolo, ID\_artista, ID\_etichetta, ID\_collezionista, ID\_collezione, ID\_genere, ID\_tipo, anno\_uscita);

    END $$

DELIMITER ;

/\* QUERY 2b: Aggiunta di tracce ad un disco \*/

DROP PROCEDURE IF EXISTS inserimento\_traccia;

DELIMITER $$

CREATE PROCEDURE inserimento\_traccia(ISRC VARCHAR(100), durata VARCHAR(100), titolo VARCHAR(100), ID\_disco INTEGER UNSIGNED, flag ENUM('ESECUTORE', 'COMPOSITORE', 'ENTRAMBI'), ID\_artista INTEGER UNSIGNED)

BEGIN

  DECLARE ID\_brano INTEGER UNSIGNED;

  -- Inserimento del brano nella tabella "brano"

  INSERT INTO brano(ISRC, durata, titolo, ID\_disco)

  VALUES (ISRC, durata, titolo, ID\_disco);

  -- Ottiene l'ID del brano appena inserito

  SET ID\_brano = LAST\_INSERT\_ID();

  -- Inserimento nella tabella "appartiene"

  INSERT INTO appartiene(ID\_artista, ID\_brano, flag)

  VALUES (ID\_artista, ID\_brano, flag);

END $$

DELIMITER ;

/\* QUERY 3a: Modifica dello stato di pubblicazione di una collezione (da privata a pubblica e viceversa)\*/

DROP PROCEDURE IF EXISTS modifica\_pubblicazione;

DELIMITER $$

    CREATE PROCEDURE modifica\_pubblicazione(ID1 INTEGER) -- ID1 = ID\_collezione

    BEGIN

    DECLARE stato\_corrente VARCHAR(10);

    SELECT stato INTO stato\_corrente FROM collezione WHERE ID = ID1;

    IF stato\_corrente = 'pubblico' THEN

        UPDATE collezione SET stato = 'privato' WHERE ID = ID1;

    ELSEIF stato\_corrente = 'privato' THEN

        UPDATE collezione SET stato = 'pubblico' WHERE ID = ID1;

        -- tolgo tutti i record di condivisa che contengono quella collezione.

        DELETE FROM condivisa WHERE ID\_collezione = ID1;

    END IF;

    SELECT ID, stato FROM collezione WHERE ID = ID1;

END $$

DELIMITER ;

/\* QUERY 3b: Aggiunta di nuove condivisioni ad una collezione \*/

DROP PROCEDURE IF EXISTS inserimento\_condivisioni;

DELIMITER $$

    CREATE PROCEDURE inserimento\_condivisioni(ID\_collezione INTEGER UNSIGNED, ID\_collezionista INTEGER UNSIGNED)

    BEGIN

    INSERT INTO condivisa(ID\_collezione, ID\_collezionista)

    VALUES(ID\_collezione, ID\_collezionista);

    END $$

DELIMITER ;

/\* QUERY 4: Rimozione di un disco da una collezione. \*/

DROP PROCEDURE IF EXISTS rimozione\_disco;

DELIMITER $$

    CREATE PROCEDURE rimozione\_disco(ID1 INTEGER UNSIGNED) -- ID1 = ID\_disco

    BEGIN

    DELETE FROM disco WHERE ID = ID1;

    END $$

DELIMITER ;

/\* QUERY 5: Rimozione di una collezione. \*/

DROP PROCEDURE IF EXISTS rimozione\_collezione;

DELIMITER $$

    CREATE PROCEDURE rimozione\_collezione(ID1 INTEGER UNSIGNED) -- ID1 = ID\_collezione

    BEGIN

    DELETE FROM collezione WHERE collezione.ID = ID1;

    END $$

DELIMITER ;

/\* QUERY 6: Lista di tutti i dischi in una collezione. \*/

DROP PROCEDURE IF EXISTS lista\_dischi;

DELIMITER $$

    CREATE PROCEDURE lista\_dischi(ID1 INTEGER UNSIGNED) -- ID1 = ID\_collezione

    BEGIN

    SELECT d.ID, d.titolo, d.ID\_artista, d.ID\_etichetta, d.ID\_collezionista, d.ID\_genere,

           d.anno\_uscita, d.stato\_di\_conservazione, d.ID\_tipo, d.barcode FROM disco d

    WHERE d.ID\_collezione = ID1;

    END $$

DELIMITER ;

/\* QUERY 7: Track list di un disco. \*/

DROP PROCEDURE IF EXISTS tracklist;

DELIMITER $$

    CREATE PROCEDURE tracklist(ID1 INTEGER UNSIGNED) -- ID1 = ID\_disco

    BEGIN

    SELECT b.ID, b.titolo, b.ISRC, b.durata FROM brano b

        JOIN disco d ON (b.ID\_disco = d.ID)

    WHERE d.ID = ID1;

    END $$

DELIMITER ;

/\* QUERY 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. \*/

/\* Per la query in oggetto si è deciso di procedere nel seguente modo: passiamo in input una stringa che può essere

sia il nome dell’artista sia il titolo del disco, l’ID del collezionista e tre parametri booleani che ci serviranno per

effettuare la ricerca in base alle collezioni possedute dal collezionista, a quelle condivise con lui e/o alle collezioni pubbliche.

Abbiamo creato tre procedure differenti:

-ricerca1 inserisce in una tabella temporanea (ric1) tutte le collezioni che sono possedute dal collezionista;

-ricerca2 inserisce in una tabella temporanea (ric2) tutte le collezioni condivise con il collezionista;

-ricerca3 inserisce in una tabella temporanea (ric3) tutte le collezioni che sono pubbliche e quindi visibili dal collezionista.

A questo punto, grazie alla procedura ricerca\_dischi, sarà possibile fare diversi tipi di ricerca in base al valore dei

parametri booleani in input (tramite la UNION DISTINCT delle temporary table); in particolare possiamo:

-ricercare le collezioni che sono possedute da un collezionista, condivise con lui e pubbliche;

-ricercare le collezioni solo possedute dal collezionista;

-ricercare solo le collezioni condivise con il collezionista;

-ricercare tutte le collezioni pubbliche;

-ricercare le collezioni possedute dal collezionista e condivise con lui ma non pubbliche;

-ricercare le collezioni possedute dal collezionista e pubbliche;

-ricercare le collezioni che sono state condivise con il collezionista e che sono pubbliche. \*/

DROP PROCEDURE IF EXISTS ricerca\_dischi;

DELIMITER $$

CREATE PROCEDURE ricerca\_dischi(stringa VARCHAR(100), ID\_coll INTEGER UNSIGNED, posseduta BOOLEAN, condivisa BOOLEAN, pubblica BOOLEAN)

BEGIN

    IF (posseduta AND condivisa AND pubblica) THEN

    CALL ricerca1(stringa, ID\_coll) ;

    CALL ricerca2(stringa, ID\_coll) ;

    CALL ricerca3(stringa, ID\_coll);

    (SELECT \* FROM ric1) UNION DISTINCT (SELECT \* FROM ric2) UNION DISTINCT (SELECT \* FROM ric3);

    ELSEIF (posseduta AND NOT pubblica AND NOT condivisa) THEN

    CALL ricerca1(stringa, ID\_coll);

    SELECT \* FROM ric1;

    ELSEIF (condivisa AND NOT pubblica AND NOT posseduta) THEN

    CALL ricerca2(stringa, ID\_coll);

    SELECT \* FROM ric2;

    ELSEIF (pubblica AND NOT posseduta AND NOT condivisa) THEN

    CALL ricerca3(stringa, ID\_coll);

    SELECT \* FROM ric3;

    ELSEIF (posseduta AND condivisa AND NOT pubblica) THEN

    CALL ricerca1(stringa, ID\_coll);

    CALL ricerca2(stringa, ID\_coll);

    (SELECT \* FROM ric1) UNION DISTINCT (SELECT \* FROM ric2);

    ELSEIF (posseduta AND pubblica AND NOT condivisa) THEN

    CALL ricerca1(stringa, ID\_coll);

    CALL ricerca3(stringa, ID\_coll);

    (SELECT \* FROM ric1) UNION DISTINCT (SELECT \* FROM ric3);

    ELSEIF (condivisa AND pubblica AND NOT posseduta ) THEN

    CALL ricerca2(stringa, ID\_coll);

    CALL ricerca3(stringa, ID\_coll);

    SELECT \* FROM ric2 UNION DISTINCT SELECT \* FROM ric3;

    END IF;

    END $$

DELIMITER ;

DROP PROCEDURE IF EXISTS ricerca1;

DELIMITER $$

CREATE PROCEDURE ricerca1(stringa VARCHAR(100), ID\_coll INTEGER UNSIGNED)

BEGIN

DROP TEMPORARY TABLE IF EXISTS ric1;

CREATE TEMPORARY TABLE ric1 AS

    SELECT DISTINCT d.ID, d.titolo, d.ID\_etichetta, d.ID\_collezionista, d.ID\_genere,

           d.anno\_uscita, d.stato\_di\_conservazione, d.ID\_tipo, d.barcode, d.ID\_artista, a.nome\_dArte AS artista\_nome

        FROM disco d

        JOIN artista a ON d.ID\_artista = a.ID

        WHERE (d.titolo = stringa OR a.nome\_dArte = stringa)

        AND d.ID\_collezionista = ID\_coll;

END $$

DELIMITER ;

DROP PROCEDURE IF EXISTS ricerca2;

DELIMITER $$

CREATE PROCEDURE ricerca2(stringa VARCHAR(100), ID\_coll INTEGER UNSIGNED)

BEGIN

DROP TEMPORARY TABLE IF EXISTS ric2;

CREATE TEMPORARY TABLE ric2 AS

        SELECT DISTINCT d.ID, d.titolo, d.ID\_etichetta, d.ID\_collezionista, d.ID\_genere,

           d.anno\_uscita, d.stato\_di\_conservazione, d.ID\_tipo, d.barcode,  d.ID\_artista, a.nome\_dArte AS artista\_nome

        FROM disco d

        JOIN artista a ON d.ID\_artista = a.ID

        WHERE (d.titolo = stringa OR a.nome\_dArte = stringa) AND (d.ID\_collezione IN(

            SELECT ID\_collezione FROM condivisa con WHERE con.ID\_collezionista = ID\_coll));

END $$

DELIMITER ;

DROP PROCEDURE IF EXISTS ricerca3;

DELIMITER $$

CREATE PROCEDURE ricerca3(stringa VARCHAR(100), ID\_coll INTEGER UNSIGNED)

BEGIN

DROP TEMPORARY TABLE IF EXISTS ric3;

CREATE TEMPORARY TABLE ric3 AS

        SELECT DISTINCT d.ID, d.titolo, d.ID\_etichetta, d.ID\_collezionista, d.ID\_genere,

           d.anno\_uscita, d.stato\_di\_conservazione, d.ID\_tipo, d.barcode, d.ID\_artista, a.nome\_dArte AS artista\_nome

            FROM disco d

            JOIN artista a ON d.ID\_artista = a.ID

            WHERE (d.titolo = stringa OR a.nome\_dArte = stringa) AND (d.ID\_collezione IN(

                    SELECT ID

                    FROM collezione

                    WHERE stato = 'pubblico'

                )

            );

END $$

DELIMITER ;

/\* QUERY 9: Verifica della visibilità di una collezione da parte di un collezionista. \*/

DROP PROCEDURE IF EXISTS check\_visibilita;

DELIMITER $$

    CREATE PROCEDURE check\_visibilita(ID1 INTEGER UNSIGNED, ID2 INTEGER UNSIGNED) -- ID1 = ID\_collezione; ID2 = ID\_collezionista

    BEGIN

    DECLARE p BOOL;

    SET p = false;

    SELECT 1 INTO p

    FROM collezione c

    WHERE c.ID = ID1 AND (c.stato = 'pubblico' OR c.ID\_collezionista = ID2);

    IF p = false then

    SELECT 1 INTO p

    FROM condivisa c

    WHERE c.ID\_collezione = ID1 AND c.ID\_collezionista = ID2;

    END IF;

    SELECT p;

    END $$

DELIMITER ;

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

DROP PROCEDURE IF EXISTS conta\_brani\_autore;

DELIMITER $$

CREATE PROCEDURE conta\_brani\_autore(autore\_id INTEGER UNSIGNED)

BEGIN

    DECLARE brani\_count INTEGER UNSIGNED;

    SELECT COUNT(b.ID) INTO brani\_count FROM brano b

        JOIN appartiene a ON (a.ID\_brano = b.ID)

        JOIN disco d ON (d.ID = b.ID\_disco)

        JOIN collezione c ON (c.ID = d.ID\_collezione)

    WHERE a.ID\_artista = autore\_id AND c.stato = "pubblico";

    SELECT brani\_count;

END $$

DELIMITER ;

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

DROP PROCEDURE IF EXISTS conta\_minuti;

DELIMITER $$

CREATE PROCEDURE conta\_minuti(autore\_id INTEGER UNSIGNED)

BEGIN

    DECLARE minuti\_count INT;

    SELECT SUM(TIME\_TO\_SEC(b.durata)) INTO minuti\_count FROM brano b

        JOIN appartiene a ON (a.ID\_brano = b.ID)

        JOIN disco d ON (d.ID = b.ID\_disco)

        JOIN collezione c ON (c.ID = d.ID\_collezione)

    WHERE a.ID\_artista = autore\_id AND c.stato = 'pubblico';

    SELECT SEC\_TO\_TIME(minuti\_count) AS minuti\_totali;

END $$

DELIMITER ;

/\* QUERY 12a: Statistica per numero di collezioni di ciascun collezionista. \*/

DROP PROCEDURE IF EXISTS conta\_collezioni\_per\_collezionista;

DELIMITER $$

CREATE PROCEDURE conta\_collezioni\_per\_collezionista()

BEGIN

    SELECT c.ID\_collezionista, COUNT(\*) AS numero\_collezioni

    FROM collezione c

    GROUP BY ID\_collezionista;

END $$

DELIMITER ;

/\* QUERY 12b: Statistica per numero di dischi per genere nel sistema. \*/

DROP PROCEDURE IF EXISTS conta\_dischi\_per\_genere;

DELIMITER $$

CREATE PROCEDURE conta\_dischi\_per\_genere()

BEGIN

    SELECT g.nome, COUNT(\*) AS numero\_dischi

    FROM disco d

        JOIN genere g ON (g.ID = d.ID\_genere)

    GROUP BY ID\_genere;

END $$

DELIMITER ;

/\* QUERY 12\_Alternativa: Statistica per numero di collezioni di uno specifico collezionista. \*/

DROP PROCEDURE IF EXISTS conta\_collezioniXcollezionista;

DELIMITER $$

CREATE PROCEDURE conta\_collezioniXcollezionista(ID\_coll INTEGER UNSIGNED)

BEGIN

    DECLARE collezioni\_count INT;

    SELECT COUNT(c.ID) INTO collezioni\_count FROM collezione c

    WHERE c.ID\_collezionista = ID\_coll;

    SELECT collezioni\_count;

END $$

DELIMITER ;

-- Query 13

/\* Abbiamo differenziato due casi: uno in cui il barcode è nullo e uno in cui il barcode non è nullo. \*/

-- Questa query è stata implementata in java quando si cerca di aggiungere un disco già esistente.

DROP PROCEDURE IF EXISTS trova\_dischi\_simili\_barcode\_nullo;

DELIMITER $$

    CREATE PROCEDURE trova\_dischi\_simili\_barcode\_nullo(titolo VARCHAR(100), autore VARCHAR(100), coll INTEGER UNSIGNED)

    BEGIN

        SELECT d.ID, d.barcode, d.stato\_di\_conservazione, d.titolo, a.nome\_dArte, d.ID\_artista , d.ID\_collezionista FROM disco d

            JOIN artista a ON (a.ID = d.ID\_artista)

        WHERE (a.nome\_dArte LIKE CONCAT('%', autore, '%') OR d.titolo LIKE CONCAT('%', titolo, '%'))

               AND d.ID\_collezionista = coll;

    END $$

    DELIMITER ;

DROP PROCEDURE IF EXISTS trova\_dischi\_simili\_barcode;

DELIMITER $$

    CREATE PROCEDURE trova\_dischi\_simili\_barcode(barcode VARCHAR(100), coll INTEGER UNSIGNED)

    BEGIN

        SELECT d.ID, d.barcode, d.stato\_di\_conservazione, d.titolo, d.ID\_artista, d.ID\_collezionista FROM disco d

        WHERE d.barcode = barcode AND d.ID\_collezionista = coll;

    END $$

    DELIMITER ;

-- PROCEDURES AGGIUNTIVE CHE CI SONO TORNATE UTILI

/\* Funzione per contare le copie di un disco di un collezionista \*/

DROP FUNCTION IF EXISTS conta\_copie;

DELIMITER $$

CREATE FUNCTION conta\_copie(ID\_disco INTEGER)

RETURNS INTEGER UNSIGNED DETERMINISTIC

BEGIN

    DECLARE numero\_copie INTEGER UNSIGNED;

    DECLARE disco\_titolo VARCHAR(255);

    DECLARE disco\_artista INTEGER;

    DECLARE disco\_collezionista INTEGER;

    SELECT titolo, ID\_artista, ID\_collezionista INTO disco\_titolo, disco\_artista, disco\_collezionista

    FROM disco WHERE ID = ID\_disco;

    SET numero\_copie = (

        SELECT COUNT(DISTINCT disc.ID) FROM disco disc

        WHERE disc.ID\_artista = disco\_artista AND disc.titolo = disco\_titolo AND disc.ID\_collezionista = disco\_collezionista

    );

    RETURN numero\_copie;

END $$

DELIMITER ;

-- procedura A: aggiunge un disco già esistente in una collezione

DROP PROCEDURE IF EXISTS aggiungi\_disco\_esistente;

DELIMITER $$

CREATE PROCEDURE aggiungi\_disco\_esistente(id\_disco INTEGER UNSIGNED, id\_coll INTEGER UNSIGNED)

BEGIN

    INSERT INTO disco (barcode, stato\_di\_conservazione, titolo, ID\_artista, ID\_etichetta, ID\_collezionista, ID\_collezione, ID\_genere, ID\_tipo, anno\_uscita)

    SELECT barcode, stato\_di\_conservazione, titolo, ID\_artista, ID\_etichetta, ID\_collezionista, id\_coll, ID\_genere, ID\_tipo, anno\_uscita

    FROM disco

    WHERE ID = id\_disco;

    SET @new\_disco\_id = LAST\_INSERT\_ID();

   -- inserisco i brani

    INSERT INTO brano (ISRC, titolo, durata, ID\_disco)

    SELECT b.ISRC, b.titolo, b.durata, @new\_disco\_id

    FROM brano b

    INNER JOIN disco d ON b.ID\_disco = d.ID

    WHERE d.ID = id\_disco;

END $$

DELIMITER ;

* **Interfacciamento Java/DBMS**

È stata creata un’interfaccia scritta in Java sul modello di programmazione ad oggetti utilizzando le librerie di JavaFX e progetto Maven.

Inizialmente bisogna dichiarare come VM Arguments la stringa:

*--module-path "C:\java\javafx-sdk-18.0.1\lib" --add-modules javafx.controls,javafx.fxml*

Nell’interfaccia è possibile accedere inserendo nickname ed email (ad esempio “user” e “user@icloud.com”):

Nel caso in cui ci si voglia registrare basterà mettere nickname, email e cliccare Sign up; abbiamo deciso di omettere una schermata di registrazione più complessa, poiché non pertinente al progetto.

Immagine che contiene testo, schermata, software, computer

Descrizione generata automaticamente

Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamenteNella schermata Home sarà possibile:

1. Vedere le mie collezioni;
2. Vedere le collezioni condivise con me;
3. Aggiungere Artisti o Etichette discografiche;
4. Fare delle ricerche combinate;
5. Vedere delle statistiche.
6. Andando nella sezione “My Collections” sarà possibile vedere tutte le collezioni da me possedute, modificarne lo stato (query 3a), condividerle (query 3b-9), ed è possibile, cliccando sul tasto View, visualizzare tutti i dischi contenuti in quella collezione (query 6), le caratteristiche del disco e i brani che compongono quel disco (query 7). È possibile inoltre eliminare una collezione (query 5) e/o un disco (query 4).

A screenshot of a computer

Description automatically generated with medium confidence A screenshot of a computer

Description automatically generated

Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamente

Sarà inoltre possibile aggiungere tracce al disco tramite il bottone “Add tracks” specificandone il titolo, il codice ISRC, la durata del brano, l’artista e specificando se l’artista è il compositore e/o l’esecutore del brano (query 2b).

Immagine che contiene testo, software, Icona del computer, Pagina Web

Descrizione generata automaticamente

È possibile anche aggiungere un disco ad una collezione tramite il bottone “Add disk” (query 2a) o aggiungere una copia di un disco già esistente tramite il pulsante “Add Existing Disk” (query 13). È stata inoltre implementata una procedure aggiuntiva (Procedure A) che ti permette di aggiungere un determinato disco cercato alla collezione tramite il bottone “Choose”.

Immagine che contiene testo, software, Icona del computer, Software multimediale

Descrizione generata automaticamente Immagine che contiene testo, software, Icona del computer, Software multimediale

Descrizione generata automaticamente

Si può anche aggiungere una nuova collezione nella lista delle collezioni di un collezionista tramite il pulsante “Add Collection” (query 1).

Immagine che contiene testo, software, Icona del computer, Sistema operativo

Descrizione generata automaticamente

1. Cliccando invece sul pulsante “Friends’ Collection” è possibile vedere tutte le collezioni che sono state condivise con me.

Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamente

Anche qui è possibile vedere nel dettaglio da quali dischi è formata la collezione e quali sono i brani dei dischi; in questo caso è possibile solo la lettura delle informazioni da parte del collezionista ma non la modifica.

Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamente Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamente

1. Attraverso il pulsante “Add Artist/Label” è possibile aggiungere un nuovo Artista o una nuova Etichetta, specificando in entrambi i casi i relativi attributi.

Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamente Immagine che contiene testo, software, computer, Icona del computer

Descrizione generata automaticamente

Immagine che contiene testo, software, Icona del computer, Software multimediale

Descrizione generata automaticamente

Seguendo lo stesso modello, è possibile aggiungere nuovi generi musicali e nuovi tipi di dischi; abbiamo preferito non implementarli per non appesantire l’interfaccia Java.

1. Cliccando sul pulsante “Search” è possibile fare diversi tipi di ricerca: le mie collezioni e/o le collezioni condivise con me e/o le collezioni pubbliche(query 8).

Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamente

1. È possibile, tramite il tasto “Stats” della schermata Home, andare ad effettuare delle statistiche varie come il numero di copie di un certo disco, il numero di tracce presenti di un certo autore e/o il numero di minuti totali di musica per autore (query 10-11 + function per contare le copie).

Immagine che contiene testo, schermata, software, Icona del computer

Descrizione generata automaticamente

L’unica query che non è stata implementata è la query 12; abbiamo fatto questa scelta perché non abbiamo trovato uno scopo per quella query nei limiti della nostra applicazione. Tuttavia nell’eventualità di futuri sviluppi potrebbe essere una query interessante da implementare in un profilo dedicato a degli amministratori.