GROOVECLAM

Andrea Giacomo Baldan 579117

May 27, 2015

Contents

Abstract

A seguito degli eventi riguardanti il caso 'Napster' nei primi anni 2000, l'industria musicale e la distribuzione del materiale digitale ha subito notevoli cambiamenti e negli anni successivi prese piede il fenomeno del P2P (scambio tra utenti di files musicali, e non solo, mediante la rete) avviato da 'Napster', seguito da piattaforme

e siti che offrono un servizio di streaming di file audio/video nel (quasi) totale rispetto dei diritti sugli album pubblicati. Grooveclam è una piattaforma online sulla linea del recente defunto Grooveshark, un sito di streaming audio, che si propone di offrire un servizio di condivisione musicale tra utenti, permettendo di selezionare brani MP3 per l'ascolto, organizzarli in playlist che possono essere condivise tra utenti connessi tra di loro o in semplici code di riproduzione anonime. Offre in più la possibilità di generare e popolare la propria libreria personale di brani e di contribuire al popolamento della base di dati su cui poggia la piattaforma aggiungendo le proprie canzoni, rendendole così disponibili per l'ascolto a tutti gli utenti.

1 Analisi Dei Requisiti

Si vuole realizzare una base di dati per la gestione di una libreria musicale condivisa e la relativa interfaccia web che permetta interazione tra gli utenti.
Il cuore della libreria è formato da un insieme di album, ogni album è identificato da un codice. E' inoltre formato da alcuni metadati (titolo, autore, anno di pubblicazione), è specificato se si tratta di un album registrato in studio o una versione live e, in quest'ultimo caso, è possibile specificare la città in cui si è svolta la registrazione del concerto, può possedere inoltre informazioni opzionali di carattere generale (critiche ricevute, recensioni o breve storia sulla realizzazione dell'album). Infine ogni album può avere una copertina, a cui fanno riferimento anche tutti i brani che contiene.
Un album contiene più brani musicali. Ogni brano contenuto nell'album è identificato da un codice, ed è formato da alcuni metadati quali titolo, genere, durata. Esistono due tipi di utenti che possono accedere alla libreria, ordinari e amministratori, di entrambi interessano l'indirizzo e-mail, uno username e una password, sono opzionali i dati anagrafici quali nome e cognome. Gli utenti odinari possono decidere di seguire altri utenti ordinari, eccetto se stessi. Ogni utente ordinario ha la possibilità di creare una propria collezione di brani preferiti selezionandoli dalla libreria, può creare una coda di riproduzione anonima, o creare delle playlist delle quali interessa sapere il nome. Interessa inoltre sapere se si tratta di playlist pubbliche o private.
All'interno della collezione i brani non possono ripetersi mentre nelle code di riproduzione o nelle playlist uno stesso brano può comparire più volte. All'atto di registrazione un utente può decidere se attivare un abbonamento free o utilizzare un piano premium.

2 Progettazione concettuale

2.1 Classi

- **Utenti**: Rappresenta un utente del servizio.
  - IdUtente: *Int* «PK»
  - Nome: *String*
  - Cognome: *String*
  - Email: *String*

  Sono definite le seguenti sottoclassi disgiunte:
  - **Amministratore**: Rappresenta un utente con privilegi amministrativi.
  - **Ordinario**: Rappresenta un utente ordinario.

- **Login**: Rappresenta delle credenziali d'accesso per un utente.
  - Username: *String* «PK»
  - Password: *String*
- **Iscrizioni**: Modella un piano di iscrizione.
  - Tipo: *Enum* ['Free', 'Premium']
- **Brani**: Rappresenta un brano.
  - IdBrano: *Int* «PK»
  - Titolo: *String*
  - Genere: *String*
  - Durata: *Float*
- **Album**: Modella un album di brani.
  - IdAlbum: *Int* «PK»
  - Titolo: *String*
  - Autore: *String*
  - Info: *String*
  - Anno: *Date*

  Sono definite le seguenti sottoclassi disgiunte con vincolo di partizionamento.
  - **Live**: Rappresenta un album registrato durante una performance live.
    - Locazione: *String*
  - **Studio**: Rappresenta un album registrato in studio.
- **Copertine**: Rappresenta una generica cover di album.
  - IdImm: *Int* «PK»
  - Path: *String*
- **Playlist**: Modella una playlist.
  - Nome: *String*

  Sono definite le seguenti sottoclassi disgiunte con vincolo di partizionamento.
  - **Pubblica**: Rappresenta una playlist pubblica, a cui tutti gli utenti possono accedere all'ascolto.
  - **Privata**: Rappresenta una playlist privata, solo il creatore può accedervi all'ascolto
- **Collezioni**: Rappresenta una collezione di brani preferiti dall'utente.
  - IdCollezione: *Int* «PK»

2.2 Associazioni

- **Utenti-Collezioni**: "Crea"
  - Ogni utente può creare zero o una collezione, ogni collezione può essere creata da un solo utente.
  - Molteplicità 1 : 1
  - Parziale verso **Utenti**, totale verso **Collezioni**.
- **Utenti-Brani**: "Ascolta"
  - Ogni utente può ascoltare zero o più brani, ogni brano può essere ascoltato da zero o più utenti.
  - Molteplicità N : N
  - Parziale verso **Utenti**, parziale verso **Brani**.
  - Attributi:
    - Timestamp: *Timestamp*
- **Utenti-Brani**: "Accoda"
  - Ogni utente può accodare zero o più brani, ogni brano può essere accodato da zero o più utenti.
  - Molteplicità N : N
  - Parziale verso **Utenti**, parziale verso **Brani**.
  - Attributi:
    - Timestamp: *Timestamp*
- **Utenti-Utenti**: "Segue"
  - Ogni utente può seguire zero o più utenti, ogni utente può essere seguito da zero o più utenti.
  - Molteplicità N : N
  - Parziale verso entrambi.
- **Utenti-Playlist**: "Crea"
  - Ogni utente può creare zero o più playlist, ogni playlist può essere creata da un solo utente.
  - Molteplicità N : 1
  - Parziale verso **Utenti**, totale verso **Playlist**.
- **Utenti-Iscrizioni**: "Iscritto"
  - Ogni utente può avere una sola iscrizione, ogni iscrizione può essere associata ad un solo utente.
  - Molteplicità 1 : 1
  - Totale verso **Utenti** e verso **Iscrizioni**.
- **Playlist-Brani**: "PopolataDa"

  - Ogni playlist è popolata da zero o più brani, ogni brano popola zero o più playlist.
  - Molteplicità N : N
  - Parziale verso **Playlist**, parziale verso **Brani**.

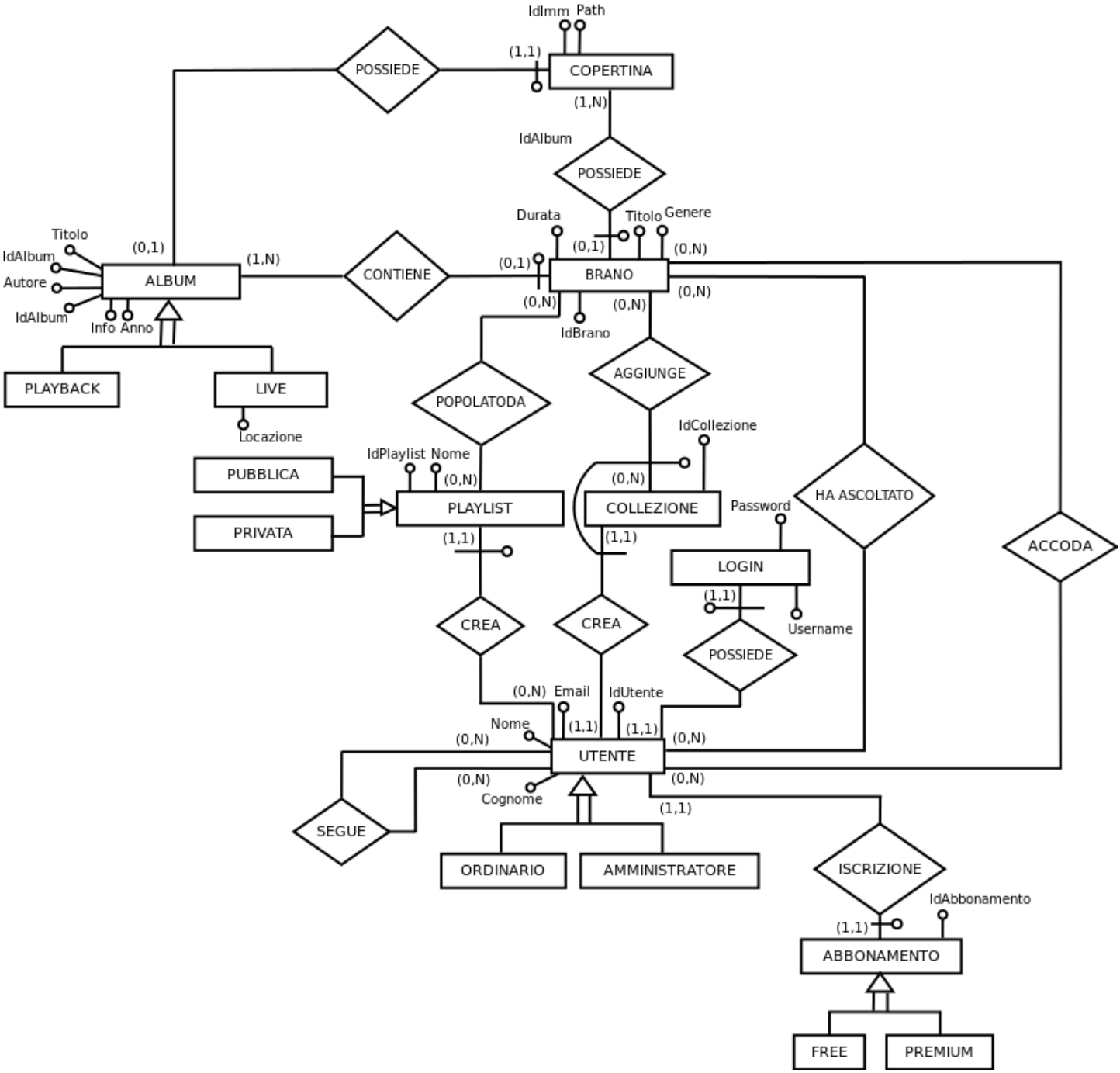
- **Brani-Album**: "AppartieneA"
  - Ogni brano appartiene a zero o un brano, ogni brano contiene uno o più brani.
  - Molteplicità 1 : N
  - Parziale verso **Brani**, totale verso **Album**.

- **Album-Copertine**: "Possiede"
  - Ogni album possiede zero o una copertina, ogni copertina è posseduta da un solo album.
  - Molteplicità 1 : 1
  - Parziale verso **Album**, totale verso **Copertine**.

- **Brani-Copertine**: "Possiede"
  - Ogni brano possiede zero o una cover, ogni cover è posseduta da una o più brani.
  - Molteplicità 1 : N
  - Totale verso **Brani**, totale verso **Copertine**.

- **Collezioni-Brani**: "PopolateDa"
  - Ogni collezione è popolata da zero o più brani, ogni brano popola zero o più collezioni.
  - Molteplicità N : N
  - Parziale verso **Collezioni**, parziale verso **Brani**.

2.3 Schema E/R


IdImm
Path
POSSIEDE
(1,1)
COPERTINA
(1,N)
IdAlbum
POSSIEDE
Durata
Titolo
Genere
(0,1)
(0,1)
(0,N)
Titolo
IdAlbum
Autore
IdAlbum
(0,1)
ALBUM
Info
Anno
(1,N)
CONTIENE
(0,1)
BRANO
(0,N)
(0,N)
(0,N)
IdBrano
PLAYBACK
LIVE
Locazione
POPOLATODA
AGGIUNGE
IdCollezione
IdPlaylist
Nome
(0,N)
(0,N)
PUBBLICA
PRIVATA
PLAYLIST
COLLEZIONE
Password
HA ASCOLTATO
ACCODA
(1,1)
(1,1)
LOGIN
(1,1)
Username
CREA
CREA
POSSIEDE
(0,N)
Email
IdUtente
Nome
(0,N)
(1,1)
(1,1)
(0,N)
UTENTE
(0,N)
Cognome
(0,N)
(1,1)
SEGUE
ORDINARIO
AMMINISTRATORE
ISCRIZIONE
IdAbbonamento
(1,1)
ABBONAMENTO
FREE
PREMIUM

3 Progettazione Logica

3.1 Gerarchie

Tutte le gerarchie presenti nella progettazione concettuale sono state risolte mediante accorpamento in tabella unica, questo perchè nessuna di esse possedeva sottoclassi con un numero significativo di attributi o associazioni entranti da giustificarne un partizionamento di qualche genere.

3.2 Chiavi Primarie

Sono state create alcune chiavi primarie per identificare le istanze di alcune tabelle, quali *IdPlaylist* a **Playlist**.

3.3 Associazioni

- **Utenti-Collezioni**: "Crea"
  - Ogni utente può creare zero o una collezione, ogni collezione può essere creata da un solo utente.
  - Molteplicità 1 : 1
  - Parziale verso **Utenti**, totale verso **Collezioni**.
  - Chiave esterna non-nulla in **Collezioni** verso **Utenti**.
- **Utenti-Brani**: "Ascolta"
  - Ogni utente può ascoltare zero o più brani, ogni brano può essere ascoltato da zero o più utenti.
  - Molteplicità N : N
  - Parziale verso **Utenti**, parziale verso **Brani**.
  - Attributi:
    - Timestamp: *Timestamp*
  - Nuova tabella **Ascoltate**, attributi:
    - IdUtente: *Int* «PK» «FK(Utenti)»
    - IdBrano: *Int* «PK» «FK(Brani)»
    - Timestamp: *Timestamp* «PK»
- **Utenti-Brani**: "Accoda"
  - Ogni utente può accodare zero o più brani, ogni brano può essere accodato da zero o più utenti.
  - Molteplicità N : N
  - Parziale verso **Utenti**, parziale verso **Brani**.
  - Attributi:
    - Timestamp: *Timestamp*
  - Nuova tabella **Code**, attributi:
    - IdUtente: *Int* «PK» «FK(Utenti)»

    - IdBrano: *Int* «PK» «FK(Brani)»
    - Timestamp: *Timestamp* «PK»

- **Utenti-Utenti**: "Segue"
  - Ogni utente può seguire zero o più utenti, ogni utente può essere seguito da zero o più utenti.
  - Molteplicità N : N
  - Parziale verso entrambi.
  - Nuova tabella **Seguaci**, attributi:
    - IdUtente: *Int* «PK» «FK(Utenti)»
    - IdSeguace: *Int* «PK» «FK(Utenti)»

- **Utenti-Playlist**: "Crea"
  - Ogni utente può creare zero o più playlist, ogni playlist può essere creata da un solo utente.
  - Molteplicità N : 1
  - Parziale verso **Utenti**, totale verso **Playlist**.
  - Chiave esterna non-nulla in **Playlist** verso **Utenti**.

- **Utenti-Iscrizioni**: "Iscritto"
  - Ogni utente può avere una sola iscrizione, ogni iscrizione può essere associata ad un solo utente.
  - Molteplicità 1 : 1
  - Totale verso **Utenti** e verso **Iscrizioni**.
  - Chiave esterna non-nulla in **Iscrizioni** verso **Utenti**.

- **Playlist-Brani**: "PopolataDa"
  - Ogni playlist è popolata da zero o più brani, ogni brano popola zero o più playlist.
  - Molteplicità N : N
  - Parziale verso **Playlist**, parziale verso **Brani**.
  - Nuova tabella **BraniPlaylist**, attributi:
    - IdPlaylist: *Int* «PK» «FK(Playlist)»
    - IdBrano: *Int* «PK» «FK(Brani)»

- **Brani-Album**: "AppartieneA"
  - Ogni brano appartiene a zero o un brano, ogni brano contiene uno o più brani.
  - Molteplicità 1 : N
  - Parziale verso **Brani**, totale verso **Album**.
  - Chiave esterna non-nulla in **Brani** verso **Album**.

- **Album-Copertine**: "Possiede"
  - Ogni album possiede zero o una cover, ogni cover è posseduta da un solo album.
  - Molteplicità 1 : 1
  - Parziale verso **Album**, totale verso **Copertine**.
  - Chiave esterna non-nulla in **Copertine** verso **Album**.
- **Brani-Copertine**: "Possiede"
  - Ogni brano possiede zero o una copertina, ogni copertina è posseduta da una o più brani.
  - Molteplicità 1 : N
  - Totale verso **Brani**, totale verso **Copertine**.
  - Chiave esterna non-nulla in **Brani** verso **Copertine**.
- **Collezioni-Brani**: "PopolateDa"
  - Ogni collezione è popolata da zero o più brani, ogni brano popola zero o più collezioni.
  - Molteplicità N : N
  - Parziale verso **Collezioni**, parziale verso **Brani**.
  - Nuova tabella **BraniCollezione**, attributi:
    - IdBrano: *int* «PK» «FK(Brani)»
    - IdCollezione: *int* «PK «FK(Collezioni)»

4 Implementazione Fisica

Query di implementazione DDL SQL della base di dati. Sorgente in `grooveclam.sql`, popolamento in `populate.sql`. E' stata implementata una tabella **Errori**, riempita mediante procedura a sua volta richiamata dai trigger che ne fanno uso, contiene i messaggi d'errore rilevati. `functions.sql` contiene invece le funzioni, i trigger e le procedure implementate.

```
1  SET FOREIGN_KEY_CHECKS = 0;
2
3  DROP TABLE IF EXISTS `Errori`;
4  DROP TABLE IF EXISTS `Album`;
5  DROP TABLE IF EXISTS `Brani`;
6  DROP TABLE IF EXISTS `Copertine`;
7  DROP TABLE IF EXISTS `Utenti`;
8  DROP TABLE IF EXISTS `Seguaci`;
9  DROP TABLE IF EXISTS `Iscrizioni`;
10 DROP TABLE IF EXISTS `Collezioni`;
11 DROP TABLE IF EXISTS `BraniCollezione`;
12 DROP TABLE IF EXISTS `Playlist`;
13 DROP TABLE IF EXISTS `BraniPlaylist`;
```

```
14 DROP TABLE IF EXISTS `Code`;
15 DROP TABLE IF EXISTS `Ascoltate`;
16
17 -- Table di supporto Errori
18 CREATE TABLE IF NOT EXISTS `Errori` (
19         `Errore` VARCHAR(256) DEFAULT NULL
20 ) ENGINE=InnoDB DEFAULT CHARSET=Latin1;
21 -- Table Album
22 CREATE TABLE IF NOT EXISTS `Album` (
23     `IdAlbum` INT(11) NOT NULL AUTO_INCREMENT,
24     `Titolo` VARCHAR(140) NOT NULL,
25     `Autore` VARCHAR(140) NOT NULL,
26     `Info` VARCHAR(300) DEFAULT NULL,
27     `Anno` YEAR DEFAULT NULL,
28     `Live` BOOLEAN DEFAULT FALSE,
29     `Locazione` VARCHAR(40) DEFAULT NULL,
30     PRIMARY KEY(`IdAlbum`)
31 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
32 -- Table Brani
33 CREATE TABLE IF NOT EXISTS `Brani` (
34     `IdBrano` INT(11) NOT NULL AUTO_INCREMENT,
35     `IdAlbum` INT(11) NOT NULL,
36     `Titolo` VARCHAR(140) NOT NULL,
37     `Genere` VARCHAR(40) NOT NULL,
38     `Durata` INT(11),
39     `IdImm` INT(11) NOT NULL,
40     PRIMARY KEY(`IdBrano`),
41     FOREIGN KEY(`IdAlbum`) REFERENCES Album(`IdAlbum`) ON DELETE CASCADE ON
           UPDATE CASCADE,
42     FOREIGN KEY(`IdImm`) REFERENCES Copertine(`IdImm`) ON DELETE CASCADE ON
           UPDATE CASCADE
43 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
44 -- Table Copertine
45 CREATE TABLE IF NOT EXISTS `Copertine` (
46     `IdImm` INT(11) NOT NULL AUTO_INCREMENT,
47     `IdAlbum` INT(11) NOT NULL,
48     `Path` VARCHAR (40) NOT NULL DEFAULT "img/covers/nocover.jpg",
49     PRIMARY KEY(`IdImm`),
50     FOREIGN KEY(`IdAlbum`) REFERENCES Album(`IdAlbum`) ON DELETE CASCADE ON
           UPDATE CASCADE
51 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
52 -- Table Utenti
53 CREATE TABLE IF NOT EXISTS `Utenti` (
54     `IdUtente` INT(11) NOT NULL AUTO_INCREMENT,
55     `Nome` VARCHAR(40) DEFAULT NULL,
56     `Cognome` VARCHAR(40) DEFAULT NULL,
57     `Email` VARCHAR(40) NOT NULL,
58     `Amministratore` BOOLEAN NOT NULL,
59     `Username` VARCHAR(40) NOT NULL,
```

```
60      `Password` VARCHAR(40) NOT NULL,
61      PRIMARY KEY(`IdUtente`),
62      UNIQUE(`Username`)
63  ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
64  -- Table Seguaci
65  CREATE TABLE IF NOT EXISTS `Seguaci` (
66      `IdUtente` INT(11) NOT NULL,
67      `IdSeguace` INT(11) NOT NULL,
68      CONSTRAINT PRIMARY KEY pk(`IdUtente`, `IdSeguace`),
69      FOREIGN KEY(`IdUtente`) REFERENCES Utenti(`IdUtente`) ON DELETE CASCADE ON
            UPDATE CASCADE,
70      FOREIGN KEY(`IdSeguace`) REFERENCES Utenti(`IdUtente`) ON DELETE CASCADE
            ON UPDATE CASCADE,
71      CHECK(`IdUtente` != `IdSeguace`)
72  ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
73  -- Table Iscrizioni
74  CREATE TABLE IF NOT EXISTS `Iscrizioni` (
75      `IdUtente` INT(10) NOT NULL,
76      `Tipo` ENUM('Free', 'Premium') NOT NULL,
77      PRIMARY KEY(`IdUtente`),
78      FOREIGN KEY(`IdUtente`) REFERENCES Utenti(`IdUtente`) ON DELETE CASCADE ON
            UPDATE CASCADE
79  ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
80  -- Table Collezioni
81  CREATE TABLE IF NOT EXISTS `Collezioni` (
82      `IdCollezione` INT(11) NOT NULL AUTO_INCREMENT,
83      `IdUtente` INT(11) NOT NULL,
84      PRIMARY KEY(`IdCollezione`),
85      FOREIGN KEY(`IdUtente`) REFERENCES Utenti(`IdUtente`) ON DELETE CASCADE ON
            UPDATE CASCADE
86  ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
87  -- Table BraniCollezione
88  CREATE TABLE IF NOT EXISTS `BraniCollezione` (
89      `IdBrano` INT(11) NOT NULL,
90      `IdCollezione` INT(11) NOT NULL,
91      CONSTRAINT PRIMARY KEY pk(`IdCollezione`, `IdBrano`),
92      FOREIGN KEY(`IdBrano`) REFERENCES Brani(`IdBrano`) ON DELETE CASCADE ON
            UPDATE CASCADE,
93      FOREIGN KEY(`IdCollezione`) REFERENCES Collezioni(`IdCollezione`) ON
            DELETE CASCADE ON UPDATE CASCADE
94  ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
95  -- Table Playlist
96  CREATE TABLE IF NOT EXISTS `Playlist` (
97      `IdPlaylist` INT(11) NOT NULL AUTO_INCREMENT,
98      `IdUtente` INT(11) NOT NULL,
99      `Nome` VARCHAR(40) NOT NULL,
100     `Privata` BOOLEAN DEFAULT FALSE,
101     PRIMARY KEY(`IdPlaylist`),
102     FOREIGN KEY(`IdUtente`) REFERENCES Utenti(`IdUtente`) ON DELETE CASCADE ON
```

```
        UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
-- Table BraniPlaylist
CREATE TABLE IF NOT EXISTS `BraniPlaylist` (
    `IdPlaylist` INT(11) NOT NULL,
    `IdBrano` INT(11) NOT NULL,
    `Posizione` INT(11) NOT NULL,
    CONSTRAINT PRIMARY KEY pk(`IdPlaylist`, `IdBrano`),
    FOREIGN KEY(`IdPlaylist`) REFERENCES Playlist(`IdPlaylist`) ON DELETE
        CASCADE ON UPDATE CASCADE,
    FOREIGN KEY(`IdBrano`) REFERENCES Brani(`IdBrano`) ON DELETE CASCADE ON
        UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
-- Table Code
CREATE TABLE IF NOT EXISTS `Code` (
    `IdUtente` INT(11) NOT NULL,
    `IdBrano` INT(11) NOT NULL,
    `Posizione` INT(11) NOT NULL,
    CONSTRAINT PRIMARY KEY pk(`IdUtente`, `IdBrano`, `Posizione`),
    FOREIGN KEY(`IdUtente`) REFERENCES Utenti(`IdUtente`) ON DELETE CASCADE ON
         UPDATE CASCADE,
    FOREIGN KEY(`IdBrano`) REFERENCES Brani(`IdBrano`) ON DELETE CASCADE ON
        UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
-- Table Ascoltate
CREATE TABLE IF NOT EXISTS `Ascoltate` (
    `IdUtente` INT(11) NOT NULL,
    `IdBrano` INT(11) NOT NULL,
    `Timestamp` TIMESTAMP NOT NULL,
    CONSTRAINT PRIMARY KEY pk(`IdUtente`, `IdBrano`, `Timestamp`),
    FOREIGN KEY(`IdUtente`) REFERENCES Utenti(`IdUtente`) ON DELETE CASCADE ON
         UPDATE CASCADE,
    FOREIGN KEY(`IdBrano`) REFERENCES Brani(`IdBrano`) ON DELETE CASCADE ON
        UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
-- INSERT POPULATION
-- Insert into Utente
INSERT INTO Utenti(`Nome`, `Cognome`, `Email`, `Amministratore`, `Username`, `
    Password`)
        VALUES('Andrea', 'Baldan', 'a.g.baldan@gmail.com', 0, 'codep', MD5('
            ciao')),
            ('Federico', 'Angi', 'angiracing@gmail.com', 0, 'keepcalm', MD5('
                calm')),
            ('Marco', 'Rossi', 'rossi@gmail.com', 0, 'rossi', MD5('marco')),
            ('Luca', 'Verdi', 'verdi@yahoo.it', 0, 'verdi', MD5('luca')),
            ('Alessia', 'Neri', 'neri@gmail.com', 0, 'neri', MD5('alessia'));
-- Insert into Subscrition
INSERT INTO Iscrizioni(`IdUtente`, `Tipo`) VALUES(1, 'Free'), (2, 'Free');
-- Insert into Album
```

```
142 INSERT INTO Album(`Titolo`, `Autore`, `Info`, `Anno`, `Live`, `Locazione`)
143         VALUES('Inception Suite', 'Hans Zimmer', 'Inception movie soundtrack,
                composed by the Great Compositor Hans Zimmer', '2010', 0, NULL),
144                ('The Good, the Bad and the Ugly: Original Motion Picture
                    Soundtrack', 'Ennio Morricone', 'Homonym movie soundtrack,
                    created by the Legendary composer The Master Ennio Morricone',
                     '1966', 0, NULL),
145                ('Hollywood in Vienna 2014', 'Randy Newman - David Newman', '
                    Annual cinematographic review hosted in Vienna', '2014', 1, '
                    Vienna'),
146                ('The Fragile', 'Nine Inch Nails', 'The Fragile is the third
                    album and a double album by American industrial rock band Nine
                     Inch Nails, released on September 21, 1999, by Interscope
                    Records.', '1999', 0, NULL),
147                ('American IV: The Man Comes Around', 'Johnny Cash', 'American IV
                    : The Man Comes Around is the fourth album in the American
                    series by Johnny Cash(and his 87th overall), released in 2002.
                     The majority of songs are covers which Cash performs in his
                    own spare style, with help from producer Rick Rubin.', '2002',
                     0, NULL),
148                ('Greatest Hits', 'Neil Young', 'Rock & Folk Rock greatest
                    success songs by Neil Young', '2004', 0, NULL);
149 -- Insert into Brani
150 INSERT INTO Brani(`IdAlbum`, `Titolo`, `Genere`, `Durata`, `IdImm`)
151         VALUES(1, 'Mind Heist', 'Orchestra', 203, 1),
152                (1, 'Dream is collapsing', 'Orchestra', 281, 1),
153                (1, 'Time', 'Orchestra', 215, 1),
154                (1, 'Half Remembered Dream', 'Orchestra', 71, 1),
155                (1, 'We Built Our Own World', 'Orchestra', 115, 1),
156                (1, 'Radical Notion', 'Orchestra', 222, 1),
157                (1, 'Paradox', 'Orchestra', 205, 1),
158                (2, 'Il Tramonto', 'Orchestra', 72, 2),
159                (2, 'L\'estasi dell\'oro', 'Orchestra', 202, 2),
160                (2, 'Morte di un soldato', 'Orchestra', 185, 2),
161                (2, 'Il Triello', 'Orchestra', 434, 2),
162                (3, 'The Simpsons', 'Orchestra', 172, 3),
163                (3, 'The war of the Roses', 'Orchestra', 272, 3),
164                (4, 'Somewhat Damaged', 'Industrial Metal', 271, 4),
165                (4, 'The Day The Whole World Went Away', 'Industrial Metal', 273,
                     4),
166                (4, 'We\'re In This Together', 'Industrial Metal', 436, 4),
167                (4, 'Just Like You Imagined', 'Industrial Metal', 229, 4),
168                (4, 'The Great Below', 'Industrial Metal', 317, 4),
169                (5, 'Hurt', 'Country', 218, 5),
170                (5, 'Danny Boy', 'Country', 199, 5),
171                (6, 'Old Man', 'Rock', 203, 6),
172                (6, 'Southern Man', 'Rock', 331, 6);
173 -- Insert into Copertine
174 INSERT INTO Copertine(`IdImm`, `IdAlbum`, `Path`)
```

```
175         VALUES(1, 1, 'img/covers/inception.png'),
176         (2, 2, 'img/covers/morricone.jpg'),
177         (3, 3, 'img/covers/hivlogo.jpg'),
178         (4, 4, 'img/covers/fragile.jpg'),
179         (5, 5, 'img/covers/nocover.jpg'),
180         (6, 6, 'img/covers/nocover.jpg');
181 -- Insert into Collezioni
182 INSERT INTO Collezioni(`IdUtente`) VALUES(1), (2);
183 -- Insert into BraniCollezione
184 INSERT INTO BraniCollezione(`IdBrano`, `IdCollezione`) VALUES(1, 1), (2, 1),
       (3, 1), (2, 2);
185 -- Insert into Playlist
186 INSERT INTO Playlist(`IdUtente`, `Nome`, `Privata`) VALUES(1, 'Score &
       Soundtracks', 0), (1, 'Southern Rock', 0), (2, 'Colonne sonore western', 0)
       ;
187 -- Insert into BraniPlaylist
188 INSERT INTO BraniPlaylist(`IdPlaylist`, `IdBrano`, `Posizione`) VALUES(1, 1,
       1), (1, 2, 2), (1, 3, 3), (1, 4, 4), (1, 5, 5), (2, 21, 1), (2, 22, 2), (3,
        5, 1), (3, 7, 2), (3, 4, 3);
189 -- Insert into Code
190 INSERT INTO Code(`IdUtente`, `IdBrano`, `Posizione`)
191         VALUES(1, 1, 1),
192         (1, 5, 2),
193         (1, 1, 3),
194         (1, 12, 4),
195         (1, 10, 5),
196         (2, 1, 1);
197 -- Insert into Ascoltate
198 INSERT INTO Ascoltate(`IdUtente`, `IdBrano`, `Timestamp`)
199         VALUES(1, 1, '2015-04-28 18:50:03'),
200         (1, 5, '2015-04-28 18:54:06'),
201         (1, 1, '2015-04-28 19:01:43'),
202         (3, 7, '2015-04-29 18:51:02'),
203         (3, 11, '2015-04-29 17:23:15'),
204         (2, 9, '2015-04-30 21:12:52'),
205         (2, 1, '2015-05-02 22:21:22');
206 -- Insert into Seguaci
207 INSERT INTO Seguaci(`IdUtente`, `IdSeguace`) VALUES(1, 2), (1, 3), (2, 1), (3,
       1);
208 SET FOREIGN_KEY_CHECKS = 1;
```

4.1 Trigger

Di seguito i trigger creati. Sono trigger tipicamente di controllo.

- `checkDuration`: Trigger di controllo sull'inserimento della durata obbligatoriamente positiva di un brano, simula il comportamento di una clausola `CHECK Durata > 0`.

- `checkFollower`: Trigger di controllo sull'inserimento di nuovi seguaci, dove un utente non puo inserire il proprio id come seguace, simula il comportamento di una clauso `CHECK IdUtente <> IdSeguace`.
- `checkCoverImage`: Trigger di controllo sull'inserimento di una nuova Copertina, se il valore del path e' vuoto, viene inserito il path standard '`img/covers/nocover.jpg`'.
- `insertAutoCollection`: Trigger di controllo sull'inserimento di un nuovo utente, si occupa di generare una collezione vuota per il nuovo utente inserito, creando un entry nella tabella `Collezioni`.
- `errorTrigger`: Trigger di supporto, utilizzato per simulare un sistema di segnalazione errori, esegue un `SET NEW = NEW.errore;` che genera un messaggio in quanto `NEW` non puo essere manipolato e visualizza il messaggio passato alla procedura `RAISE_ERROR`.

```
1  DROP TRIGGER IF EXISTS checkDuration;
2  DROP TRIGGER IF EXISTS errorTrigger;
3  DROP TRIGGER IF EXISTS checkFollower;
4  DROP TRIGGER IF EXISTS checkCoverImage;
5  DROP TRIGGER IF EXISTS insertAutoCollection;
6
7  DELIMITER $$
8
9  CREATE TRIGGER checkDuration
10 BEFORE INSERT ON `Brani`
11 FOR EACH ROW
12 BEGIN
13 IF(NEW.Durata < 0) THEN
14     CALL RAISE_ERROR('La durata di un brano non pu  essere negativa');
15 END IF;
16 END $$
17
18 CREATE TRIGGER errorTrigger
19 BEFORE INSERT ON `Errori`
20 FOR EACH ROW
21 BEGIN
22     SET NEW = NEW.errore;
23 END $$
24
25 DELIMITER ;
26
27 DELIMITER $$
28
29 CREATE TRIGGER checkFollower
30 BEFORE INSERT ON `Seguaci`
31 FOR EACH ROW
32 BEGIN
33     IF NEW.IdUtente = NEW.IdSeguace THEN
34         CALL RAISE_ERROR('Un utente non pu  seguire se stesso (IdUtente e
               IdSeguace devono essere diversi fra loro)');
35     END IF;
```

```
END $$
DELIMITER ;

DELIMITER $$

CREATE TRIGGER checkCoverImage
BEFORE INSERT ON `Copertine`
FOR EACH ROW
BEGIN
    IF NEW.Path = '' THEN
       SET NEW.Path = 'img/covers/nocover.jpg';
    END IF;
END $$

DELIMITER ;
```

4.2 Funzioni e Procedure

Alcune funzioni e procedure implementate. Si tratta di funzioni e procedure di utilita' generale.

4.2.1 Funzioni

- `albumTotalDuration`: Dato un Id intero che rappresenta la chiave primaria di un album all'interno della base di dati, calcola la durata totale dell'album sommando le singole durate di ogni brano appartenente a tale album, convertendo il risultato finale in minuti. Utilizzando la funzione `CONCAT` restituisce una stringa formattata mm:ss.

- `elegibleForPrize`: Dato un id intero `IdUser` che rappresenta la chiave primaria di un utente all'interno della base di dati e una stringa che rappresenta un genere musicale, calcola la durata totale di ascolto su quel genere musicale da parte dell'utente rappresentato da `IdUser`. Restituisce un booleano, true nel caso in cui l'ascolto totale in secondi sia >= 1000, false altrimenti.

```
DROP FUNCTION IF EXISTS albumTotalDuration;
DROP FUNCTION IF EXISTS elegibleForPrize;

DELIMITER $$

CREATE FUNCTION albumTotalDuration(IdAlbum INT)
RETURNS VARCHAR(5)
BEGIN
DECLARE Seconds INT UNSIGNED;
SELECT SUM(b.Durata) INTO Seconds FROM Brani b WHERE b.IdAlbum = IdAlbum;
RETURN CONCAT(FLOOR(Seconds / 60), ':', (Seconds % 60));
END $$

DELIMITER ;

```

```
16 DELIMITER $$
17
18 CREATE FUNCTION elegibleForPrize(IdUser INT, Genre VARCHAR(50))
19 RETURNS BOOLEAN
20 BEGIN
21 DECLARE Seconds INT UNSIGNED DEFAULT 0;
22 DECLARE Elegibility BOOLEAN DEFAULT FALSE;
23 SELECT SUM(b.Durata) INTO Seconds
24 FROM Ascoltate a INNER JOIN Utenti u ON(a.IdUtente = u.IdUtente)
25                  INNER JOIN Brani b ON(a.IdBrano = b.IdBrano)
26 WHERE b.Genere = 'Orchestra' AND a.IdUtente = IdUser;
27 IF(Seconds >= 1000) THEN
28            SET Elegibility = TRUE;
29 END IF;
30 RETURN Elegibility;
31 END $$
32
33 DELIMITER ;
```

4.2.2 Procedure

- `GENRE_DISTRIBUTION`: Calcola la distribuzione dei generi di brani presenti all'interno della base di dati restituendo le percentuali di presenza dei vari generi. Per farlo crea una `temporary table` e la popola con le percentuali calcolate contando il totale delle canzoni e i parziali riferiti ad ogni genere, e formatta il risultato in % grazie all'utilizzo della funzione `CONCAT`.
- `USER_GENRE_DURATION`: Riprende il concetto di `GENRE_DISTRIBUTION` ma lo applica ad un utente identificato dall'Id intero passato in input, utilizzando un cursore, inserisce in una `temporary table` il numero di brani raggruppati per genere e ne calcola la percentuale sul totale di brani presenti all'interno della collezione dell'utente.
- `SWAP_POSITION`: Procedura di utilita', utilizzata in alcune pagine dell'interfaccia web, permette di scambiare i valori di due colonne (anche `unique` o `primary key`) all'interno delle tabelle `Code` o `Playlist`, lo scopo e' la possibilita di modificare l'ordine dei brani all'interno delle `code` o delle `playlist`.
- `RAISE_ERROR`: Procedura di supporto utilizzata in congiunta con il trigger `errorTrigger` e la tabella `Errori` per simulare messaggi d'errore, inserisce la stringa passata come parametro in ingresso all'interno della tabella `Errori`, il trigger si occupera' di sollevare il messaggio.

```
1 DROP PROCEDURE IF EXISTS RAISE_ERROR;
2 DROP PROCEDURE IF EXISTS GENRE_DISTRIBUTION;
3 DROP PROCEDURE IF EXISTS USER_GENRE_DISTRIBUTION;
4 DROP PROCEDURE IF EXISTS SWAP_POSITION;
5
6 DELIMITER $$
7
8 CREATE PROCEDURE RAISE_ERROR (IN ERROR VARCHAR(256))
```

```
9  BEGIN
10 DECLARE V_ERROR VARCHAR(256);
11 SET V_ERROR := CONCAT('[ERROR: ', ERROR, ']');
12 INSERT INTO Errors VALUES(V_ERROR);
13 END $$
14
15 DELIMITER ;
16
17 DELIMITER $$
18
19 CREATE PROCEDURE GENRE_DISTRIBUTION()
20 BEGIN
21 DECLARE Total INT DEFAULT 0;
22 DROP TEMPORARY TABLE IF EXISTS `Distribution`;
23 CREATE TEMPORARY TABLE `Distribution` (
24        `Genere` VARCHAR(100),
25        `Percentuale` VARCHAR(6)
26 ) ENGINE=InnoDB;
27 SELECT count(b.Genere) INTO Total FROM Brani b;
28 INSERT INTO Distribution (Genere, Percentuale)
29 SELECT Genere, CONCAT(FLOOR((count(Genere) / Total) * 100), "%")
30 FROM Brani GROUP BY Genere;
31 END $$
32
33 DELIMITER ;
34
35 DELIMITER $$
36
37 CREATE PROCEDURE USER_GENRE_DISTRIBUTION(IN IdUser INT)
38 BEGIN
39 DECLARE Done INT DEFAULT 0;
40 DECLARE Total INT DEFAULT 0;
41 DECLARE Genre VARCHAR(100) DEFAULT "";
42 DECLARE Counter INT DEFAULT 0;
43 DECLARE D_CURSOR CURSOR FOR
44        SELECT b.Genere, COUNT(b.IdBrano)
45        FROM Brani b INNER JOIN BraniCollezione bc ON (b.IdBrano = bc.IdBrano)
46                     INNER JOIN Collezioni c ON(c.IdCollezione = bc.
                            IdCollezione)
47        WHERE c.IdUtente = IdUser
48        GROUP BY b.Genere, c.IdUtente;
49 DECLARE CONTINUE HANDLER
50 FOR NOT FOUND SET Done = 1;
51 SELECT COUNT(b.IdBrano) INTO Total
52 FROM Brani b INNER JOIN BraniCollezione bc ON(b.IdBrano = bc.IdBrano)
53              INNER JOIN Collezioni c ON(bc.IdCollezione = c.IdCollezione)
54 WHERE c.IdUtente = IdUser;
55 DROP TEMPORARY TABLE IF EXISTS `Distribution`;
56 CREATE TEMPORARY TABLE `Distribution` (
```

```
57          `Genere` VARCHAR(100),
58          `Percentuale` VARCHAR(6)
59  ) ENGINE=InnoDB;
60  OPEN D_CURSOR;
61  REPEAT
62          FETCH D_CURSOR INTO Genre, Counter;
63          IF NOT Done THEN
64              INSERT INTO Distribution (Genere, Percentuale)
65              VALUES(Genre, CONCAT(FLOOR((Counter / Total) * 100), "%"));
66          END IF;
67  UNTIL Done END REPEAT;
68  CLOSE D_CURSOR;
69  SELECT * FROM `Distribution` ORDER BY Percentuale DESC;
70  DROP TABLE `Distribution`;
71  END $$
72
73  DELIMITER ;
74
75  DELIMITER $$
76
77  CREATE PROCEDURE SWAP_POSITION(IN a INT, IN b INT, IN id INT, IN tab INT)
78  BEGIN
79  DECLARE AUX INT DEFAULT -1;
80  CASE tab
81       WHEN 1 THEN
82            UPDATE Code SET Posizione = AUX WHERE Posizione = a AND IdUtente =
                 id;
83            UPDATE Code SET Posizione = a WHERE Posizione = b AND IdUtente = id;
84            UPDATE Code SET Posizione = b WHERE Posizione = AUX AND IdUtente =
                 id;
85       ELSE
86            UPDATE BraniPlaylist SET Posizione = AUX WHERE Posizione = a AND
                 IdPlaylist = id;
87            UPDATE BraniPlaylist SET Posizione = a WHERE Posizione = b AND
                 IdPlaylist = id;
88            UPDATE BraniPlaylist SET Posizione = b WHERE Posizione = AUX AND
                 IdPlaylist = id;
89  END CASE;
90  END $$
91
92  DELIMITER ;
```

5 Query

Alcune query significative.

1. Titolo, album e username dell'utente, degli ultimi 10 brani ascoltati tra i followers.

```
SELECT b.Titolo, a.Titolo as TitoloAlbum, u.Username, DATE_FORMAT(h.
    Timestamp, '%d-%m-%Y %T') AS Data
FROM Brani b INNER JOIN Album a ON(b.IdAlbum = a.IdAlbum)
             INNER JOIN Ascoltate h ON(h.IdBrano = b.IdBrano)
             INNER JOIN Seguaci f ON(f.IdSeguace = h.IdUtente)
             INNER JOIN Utenti u ON(u.IdUtente = f.IdSeguace)
WHERE h.Timestamp BETWEEN DATE_SUB(CURDATE(), INTERVAL 7 DAY) AND
    CURDATE()
AND u.IdUtente IN (SELECT u.IdUtente
                   FROM Utenti u INNER JOIN Seguaci f ON(f.IdSeguace = u
                       .IdUtente)
                   WHERE f.IdUtente = 1)
ORDER BY h.Timestamp DESC LIMIT 10;
```

Output:

```
+-----------------------------------+-----------------+----------+---------------------+
| Titolo                            | TitoloAlbum     | Username | Data                |
+-----------------------------------+-----------------+----------+---------------------+
| The Day The Whole World Went Away | The Fragile     | keepcalm | 26-05-2015 15:04:37 |
| Paradox                           | Inception Suite | keepcalm | 26-05-2015 15:04:36 |
+-----------------------------------+-----------------+----------+---------------------+
2 rows in set (0.00 sec)
```

2. Username e numero di volte che è stata ascoltata la canzone Paradox dai follower dell'user id 1

```
SELECT COUNT(b.IdBrano) AS Conto, u.Username
FROM Brani b INNER JOIN Ascoltate h ON(b.IdBrano = h.IdBrano)
             INNER JOIN Seguaci f ON(h.IdUtente = f.IdSeguace)
             INNER JOIN Utenti u ON(f.IdSeguace = u.IdUtente)
WHERE b.Titolo = 'Paradox' AND f.IdUtente = 1 GROUP BY u.Username ORDER
    BY Conto DESC;
```

Output:

```
+-------+----------+
| Conto | Username |
+-------+----------+
|     1 | keepcalm |
|     1 | rossi    |
+-------+----------+
2 rows in set (0.00 sec)
```

3. Username, titolo e conto delle canzoni piu ascoltate dai follower dell'user id 1

```
1  SELECT u.Username, b.Titolo, COUNT(b.IdBrano) AS Conto
2  FROM Brani b INNER JOIN Ascoltate h ON(b.IdBrano = h.IdBrano)
3               INNER JOIN Seguaci f ON(h.IdUtente = f.IdSeguace)
4               INNER JOIN Utenti u ON(f.IdSeguace = u.IdUtente)
5  WHERE f.IdUtente = 1 GROUP BY b.Titolo ORDER BY Conto DESC;
```

Output:

```
+----------+-----------------------------------+-------+
| Username | Titolo                            | Conto |
+----------+-----------------------------------+-------+
| keepcalm | Paradox                           |     2 |
| keepcalm | We Built Our Own World            |     1 |
| keepcalm | The Day The Whole World Went Away |     1 |
| keepcalm | Mind Heist                        |     1 |
| keepcalm | The Simpsons                      |     1 |
| keepcalm | L'estasi dell'oro                 |     1 |
| rossi    | Il Triello                        |     1 |
+----------+-----------------------------------+-------+
7 rows in set (0.00 sec)
```

4. Username e numero brani nella collezione dell'utente con più canzoni di genere 'Orchestra'

```
1  DROP VIEW IF EXISTS ContoBrani;
2  CREATE VIEW ContoBrani AS
3  SELECT u.Username, COUNT(b.Genere) as Conteggio
4  FROM Brani b INNER JOIN BraniCollezione bc ON(b.IdBrano = bc.IdBrano)
5               INNER JOIN Collezioni c ON(bc.IdCollezione = c.IdCollezione
                    )
6               INNER JOIN Utenti u ON(c.IdUtente = u.IdUtente)
7  WHERE b.Genere = 'Orchestra' GROUP BY c.IdUtente;
8  SELECT * FROM ContoBrani HAVING MAX(Conteggio);
9  DROP VIEW IF EXISTS ContoBrani;
```

Output:

```
+----------+-----------+
| Username | Conteggio |
+----------+-----------+
| codep    |         6 |
+----------+-----------+
1 row in set (0.00 sec)
```

5. Username e minuti di ascolto dei 3 utenti che ascolta più musica di genere 'Orchestra'

```
1   DROP VIEW IF EXISTS UtentiGenere;
2   CREATE VIEW UtentiGenere AS
3   SELECT u.Username, b.Genere, CONCAT(FLOOR(SUM(b.Durata) / 60), ":", (SUM
        (b.Durata) % 60)) AS DurataTotale
4   FROM Ascoltate a INNER JOIN Utenti u ON(a.IdUtente = u.IdUtente)
5                    INNER JOIN Brani b ON(a.IdBrano = b.IdBrano)
6   WHERE b.Genere = 'Orchestra' GROUP BY a.IdUtente ORDER BY DurataTotale
        DESC;
7   SELECT * FROM UtentiGenere LIMIT 3;
8   DROP VIEW IF EXISTS UtentiGenere;
```

Output:

```
Query OK, 0 rows affected, 1 warning (0.00 sec)

Query OK, 0 rows affected (0.01 sec)

+----------+-----------+--------------+
| Username | Genere    | DurataTotale |
+----------+-----------+--------------+
| verdi    | Orchestra | 29:13        |
| codep    | Orchestra | 20:42        |
| keepcalm | Orchestra | 14:57        |
+----------+-----------+--------------+
3 rows in set (0.03 sec)

Query OK, 0 rows affected (0.02 sec)

Query OK, 0 rows affected (0.00 sec)
```

6. Trova gli utenti che hanno ascoltato un numero di canzoni sopra alla media nell'ultimo mese

```
1   DROP VIEW IF EXISTS CanzoniAscoltate;
2   CREATE VIEW CanzoniAscoltate AS
3   SELECT u.Username, COUNT(a.IdBrano) as Conto
4   FROM Ascoltate a INNER JOIN Brani b ON(a.IdBrano = b.IdBrano)
5                    INNER JOIN Utenti u ON(a.IdUtente = u.IdUtente)
6   WHERE a.Timestamp BETWEEN DATE_SUB(CURDATE(), INTERVAL 30 DAY) AND NOW()
7   GROUP BY a.IdUtente;
8   SELECT ca.*
9   FROM CanzoniUtente ca
10  WHERE ca.Conto > (SELECT AVG(ce.Conto)
11                    FROM CanzoniAscoltate ce)
12  ORDER BY ca.Conto DESC;
13  DROP VIEW IF EXISTS CanzoniAscoltate;
```

```
  Output:
+----------+-------+
| Username | Conto |
+----------+-------+
| verdi    |    10 |
| codep    |    10 |
+----------+-------+
2 rows in set (0.02 sec)

Query OK, 0 rows affected (0.00 sec)
```

7. Trova gli utenti e il numero di brani di genere 'Country' nella propria collezione

```
1  CREATE VIEW Conteggi AS
2  SELECT u.Username, b.Genere, COUNT(b.IdBrano) AS Conteggio
3  FROM BraniCollezione c INNER JOIN Brani b ON(c.IdBrano = b.IdBrano)
4                         INNER JOIN Collezioni cn ON(c.IdCollezione = cn.
                             IdCollezione)
5                         INNER JOIN Utenti u ON(cn.IdUtente = u.IdUtente)
6  GROUP BY b.Genere, c.IdCollezione;
7  SELECT Username, Conteggio
8  FROM Conteggi
9  WHERE Genere = 'Country' HAVING Conteggio = (SELECT MAX(Conteggio)
10                                              FROM Conteggi
11                                              WHERE Genere = 'Country');
12 DROP VIEW IF EXISTS Conteggi;
```

```
  Output:
+----------+-----------+
| Username | Conteggio |
+----------+-----------+
| keepcalm |         2 |
+----------+-----------+
1 row in set (0.00 sec)
```

8. Trova gli utenti con più di 5 brani nella propria collezione che non hanno mai ascoltato brani country nell'ultimo mese

```
1  SELECT DISTINCT u.Username
2  FROM Utenti u INNER JOIN Ascoltate a ON(u.IdUtente = a.IdUtente)
3  WHERE u.IdUtente NOT IN (
4     SELECT DISTINCT u1.IdUtente
5     FROM Ascoltate a1 INNER JOIN Utenti u1 ON(a1.IdUtente = u1.IdUtente)
6                       INNER JOIN Brani b ON(a1.IdBrano = b.IdBrano)
```

```
7       WHERE b.Genere = 'Country')
8   AND a.Timestamp BETWEEN DATE_SUB(CURDATE(), INTERVAL 30 DAY) AND NOW()
9   AND u.IdUtente IN (SELECT u2.IdUtente
10                      FROM Utenti u2 INNER JOIN Ascoltate a2 ON(u2.IdUtente
                           = a2.IdUtente)
11                      GROUP BY a2.IdUtente
12                      HAVING COUNT(a2.IdBrano) > 5);
```

```
  Output:
+----------+
| Username |
+----------+
| keepcalm |
| verdi    |
+----------+
2 rows in set (0.02 sec)
```

6 Interfaccia Web

Per l'interfaccia web è stato seguito un pattern MVC molto rudimentale, che tuttavia ha permesso di semplificarne la realizzazione modularizzando le operazione da effettuare sulla base di dati mediante le pagine.

6.1 Organizzazione e Struttura Generale

La struttura generale dell'interfaccia consiste di 3 cartelle principali e 2 pagine di servizio contenenti rispettivamente un singleton dedicato esclusivamente alla connessione alla base di dati e un singleton dedicato alla creazione e manipolazione delle sessioni. Le cartelle /models, /views, /controllers seguono le tipiche linee guida del pattern MVC, all'interno di /models troviamo infatti i modelli, oggetti atti ad interfacciarsi con la base di dati ed eseguire le query richieste dalle pagine (routes) contenute nei controllers, infine le view, pagine "di template" contenenti per lo più codice HTML e brevi tratti di PHP, vengono popolate mediante le chiamate ai controllers. La navigazione vera e propria tra le pagine avviene mediante parametri GET che si occupano di selezionare il controller richiesto e l'azione da eseguire (funzioni all'interno del controller richiesto).

6.1.1 Esempi

- Richiedere la pagina albums:

`/basidati/~abaldan/?controller=albums&action=index`

- Visualizzazione brano con id = 4:

`/basidati/~abaldan/?controller=songs&action=show&id=4`

6.2 Pagine Principali

Ci sono 6 pagine principali che consentono la navigazione all'interno dell'interfaccia, accedibili mediante un menù laterale a sinistra. **Home** contiene alcune statistiche sullo stato della BD, ad esempio i brani ascoltati recentemente dai propri followers, questo solo dopo aver effettuato l'accesso con un proprio account registrato, altrimenti in home, come pure in ogni pagina che richiede di essere loggati, viene mostrato un form di login mediantei il quale è anche possibile registrare un account. **Songs** è la pagina adibita alla visualizzazione di tutte le canzoni contenute nella BD o, nel caso di account loggato, offre la possibilità di aggiungere i propri brani alla BD, aggiungerne alla propria collezione o alla coda di riproduzione; **Albums** contiene tutti gli album presenti nella piattaforma, sempre previa autenticazione permette di inserirne di nuovi ed è possibile visualizzare i dettagli di ogni album e brano contenuto in esso. **Collection** e **Playlist** sono rispettivamente le pagine di gestione della propria collezione brani e playlist, con la possibilità di privatizzare o rendere pubbliche le proprie playlist. **Queue** infine ospita la coda di riproduzione, ordinate in base ai timestamp di aggiunta. E' possibile modificare i dati relativi al proprio account, incluso il piano di'iscrizione, utilizzando la pagina accessibile clickando sul bottone in alto a sinistra **settings**, solo dopo aver loggato.

6.3 Mantenimento Stato Pagine

L'interfaccia da la possibilità di ascoltare canzoni come utente visitatore (anonimo), ma per le operazioni più specifiche, ad esempio la creazione e gestione di una personale collezione è necessario registrarsi e loggare utilizzando le credenziali scelte, è stato pertanto creato un sistema di gestione delle sessioni mediante la classe singleton *GrooveSession*, nel file session.php.
Essa contiene i campi dati basilari quali l'id della sessione che si va a creare e l'istanza dell'oggetto che la contiene, e i metodi necessari alla gestione con la possibilità di aggiungere variabili utili.
Alcuni account di prova:

- codep : ciao
- rossi : marco
- verdi : luca

6.4 Note

Trattandosi di un interfaccia "simulativa", in quanto la principale materia d'interesse è la struttura della base di dati su cui poggia, la riproduzione effettiva dei brani non è stata implementata, e non esistono fisicamente file Mp3 caricati all'interno della base di dati, è stato tuttavia implementato un semplice e rudimentale riproduttore in poche righe di javascript atto a dare un'idea dell'effettivo utilizzo che una completa implementazione della piattaforma porterebbe ad avere. Non sono stati scritti controlli di alcun tipo sull'input da parte dell'utente.