GROOVECLAM

Andrea Giacomo Baldan 579117

May 20, 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

- **User:** Rappresenta un utente del servizio.
  - IdUser: *Int* «PK»
  - Name: *String*
  - Surname: *String*
  - Email: *String*

  Sono definite le seguenti sottoclassi disgiunte:
  - **Administrator:** 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*

- **Subscription:** Modella un piano di iscrizione.
  - Type: *Enum* ['Free', 'Premium']

- **Song:** Rappresenta un brano.
  - IdSong: *Int* «PK»
  - Title: *String*
  - Genre: *String*
  - Duration: *Float*

- **Album:** Modella un album di brani.
  - IdAlbum: *Int* «PK»
  - Title: *String*
  - Author: *String*
  - Info: *String*
  - Year: *Date*

  Sono definite le seguenti sottoclassi disgiunte con vincolo di partizionamento.
  - **Live:** Rappresenta un album registrato durante una performance live.
    - Location: *String*
  - **Studio:** Rappresenta un album registrato in studio.

- **Cover:** Rappresenta una generica cover di album.
  - IdImage: *Int* «PK»
  - Path: *String*

- **Playlist:** Modella una playlist.
  - Name: *String*

- **Collection:** Rappresenta una collezione di brani preferiti dall'utente.
  - IdCollection: *Int* «PK»

2.2 Associazioni

- **User-Collection:** "Crea"
  - Ogni utente può creare zero o una collezione, ogni collezione può essere creata da un solo utente.
  - Molteplicità 1 : 1
  - Parziale verso **User**, totale verso **Collection**.
- **User-Song:** "Ascolta"
  - Ogni utente può ascoltare zero o più brani, ogni brano può essere ascoltato da zero o più utenti.
  - Molteplicità N : N
  - Parziale verso **User**, parziale verso **Song**.
  - Attributi:
    - Timestamp: *Timestamp*
- **User-Song:** "Accoda"
  - Ogni utente può accodare zero o più brani, ogni brano può essere accodato da zero o più utenti.
  - Molteplicità N : N
  - Parziale verso **User**, parziale verso **Song**.
  - Attributi:
    - Timestamp: *Timestamp*
- **User-User:** "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.
- **User-Playlist**: "Crea"
  - Ogni utente può creare zero o più playlist, ogni playlist può essere creata da un solo utente.
  - Molteplicità N : 1
  - Parziale verso **User**, totale verso **Playlist**.
- **User-Subscription:** "Iscritto"
  - Ogni utente può avere una sola iscrizione, ogni iscrizione può essere associata ad un solo utente.
  - Molteplicità 1 : 1
  - Totale verso **User** e verso **Subscription**.
- **Playlist-Song:** "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 **Song**.

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

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

- **Song-Cover:** "Possiede"
  - Ogni brano possiede una cover, ogni cover è posseduta da una o più canzoni.
  - Molteplicità 1 : N
  - Totale verso **Song**, totale verso **Cover**.

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

- **User-Collection:** "Crea"
  - Ogni utente può creare zero o una collezione, ogni collezione può essere creata da un solo utente.
  - Molteplicità 1 : 1
  - Parziale verso **User**, totale verso **Collection**.
  - Chiave esterna non-nulla in **Collection** verso **User**.

- **User-Song:** "Ascolta"

  - Ogni utente può ascoltare zero o più brani, ogni brano può essere ascoltato da zero o più utenti.
  - Molteplicità N : N
  - Parziale verso **User**, parziale verso **Song**.
  - Attributi:
    * Timestamp: *Timestamp*
  - Nuova tabella **Heard**, attributi:
    * IdUser: *Int* «PK» «FK(User)»
    * IdSong: *Int* «PK» «FK(Song)»
    * Timestamp: *Timestamp* «PK»

- **User-Song:** "Accoda"
  - Ogni utente può accodare zero o più brani, ogni brano può essere accodato da zero o più utenti.
  - Molteplicità N : N
  - Parziale verso **User**, parziale verso **Song**.
  - Attributi:
    * Timestamp: *Timestamp*
  - Nuova tabella **Queue**, attributi:
    * IdUser: *Int* «PK» «FK(User)»
    * IdSong: *Int* «PK» «FK(Song)»
    * Timestamp: *Timestamp* «PK»

- **User-User:** "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 **Follow**, attributi:
    * IdUser: *Int* «PK» «FK(User)»
    * IdFellow: *Int* «PK» «FK(User)»

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

- **User-Subscription:** "Iscritto"

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

- **Playlist-Song:** "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 **Song**.
  - Nuova tabella **PlaylistSong**, attributi:
    * IdPlaylist: *Int* «PK» «FK(Playlist)»
    * IdSong: *Int* «PK» «FK(Song)»

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

- **Album-Cover:** "Possiede"
  - Ogni album possiede zero o una cover, ogni cover è posseduta da un solo album.
  - Molteplicità 1 : 1
  - Parziale verso **Album**, totale verso **Cover**.
  - Chiave esterna non-nulla in **Cover** verso **Album**.

- **Song-Cover:** "Possiede"
  - Ogni brano possiede una cover, ogni cover è posseduta da una o più canzoni.
  - Molteplicità 1 : N
  - Totale verso **Song**, totale verso **Cover**.
  - Chiave esterna non-nulla in **Song** verso **Cover**.

4 Implementazione Fisica

Query di implementazione DDL SQL della base di dati. Sorgente in *genera.sql*, popolamento in *popola.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. *funproc.sql* contiene invece le funzioni, i trigger e le procedure implementate.

```
SET FOREIGN_KEY_CHECKS = 0;

DROP TABLE IF EXISTS `Errors`;
DROP TABLE IF EXISTS `Album`;
DROP TABLE IF EXISTS `Song`;
DROP TABLE IF EXISTS `Cover`;
DROP TABLE IF EXISTS `User`;
DROP TABLE IF EXISTS `Follow`;
DROP TABLE IF EXISTS `Subscription`;
DROP TABLE IF EXISTS `Collection`;
DROP TABLE IF EXISTS `SongCollection`;
DROP TABLE IF EXISTS `Playlist`;
DROP TABLE IF EXISTS `PlaylistSong`;
DROP TABLE IF EXISTS `Queue`;
DROP TABLE IF EXISTS `Heard`;

-- Support Table Error
CREATE TABLE IF NOT EXISTS `Errors` (
        `Error` VARCHAR(256) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=Latin1;
-- Table Album
CREATE TABLE IF NOT EXISTS `Album` (
    `IdAlbum` INT(11) NOT NULL AUTO_INCREMENT,
    `Title` VARCHAR(140) NOT NULL,
    `Author` VARCHAR(140) NOT NULL,
    `Info` VARCHAR(300) DEFAULT NULL,
    `Year` DATE NOT NULL,
    `Live` BOOLEAN NOT NULL,
    `Location` VARCHAR(40) DEFAULT NULL,
    PRIMARY KEY(`IdAlbum`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
-- Table Song
CREATE TABLE IF NOT EXISTS `Song` (
    `IdSong` INT(11) NOT NULL AUTO_INCREMENT,
    `IdAlbum` INT(11) NOT NULL,
    `Title` VARCHAR(140) NOT NULL,
    `Genre` VARCHAR(40) NOT NULL,
    `Duration` INT(11),
    `IdImage` INT(11) NOT NULL,
    PRIMARY KEY(`IdSong`),
    FOREIGN KEY(`IdAlbum`) REFERENCES Album(`IdAlbum`) ON DELETE CASCADE ON UPDATE CASCADE,
    FOREIGN KEY(`IdImage`) REFERENCES Cover(`IdImage`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
-- Table Cover
CREATE TABLE IF NOT EXISTS `Cover` (
    `IdImage` INT(11) NOT NULL AUTO_INCREMENT,
    `IdAlbum` INT(11) NOT NULL,
    `Path` VARCHAR (40) NOT NULL DEFAULT "img/covers/nocover.jpg",
    PRIMARY KEY(`IdImage`),
    FOREIGN KEY(`IdAlbum`) REFERENCES Album(`IdAlbum`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
-- Table User
CREATE TABLE IF NOT EXISTS `User` (
    `IdUser` INT(11) NOT NULL AUTO_INCREMENT,
    `Name` VARCHAR(40) DEFAULT NULL,
    `Surname` VARCHAR(40) DEFAULT NULL,
    `Email` VARCHAR(40) NOT NULL,
    `Administrator` BOOLEAN NOT NULL,
    `Username` VARCHAR(40) NOT NULL,
    `Password` VARCHAR(40) NOT NULL,
    PRIMARY KEY(`IdUser`),
```

```
62      UNIQUE(`Username`)
63  ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
64  -- Table Follow
65  CREATE TABLE IF NOT EXISTS `Follow` (
66      `IdUser` INT(11) NOT NULL,
67      `IdFellow` INT(11) NOT NULL,
68      CONSTRAINT PRIMARY KEY pk(`IdUser`, `IdFellow`),
69      FOREIGN KEY(`IdUser`) REFERENCES User(`IdUser`) ON DELETE CASCADE ON UPDATE CASCADE,
70      FOREIGN KEY(`IdFellow`) REFERENCES User(`IdUser`) ON DELETE CASCADE ON UPDATE CASCADE,
71      CHECK(`IdUser` != `IdFellow`)
72  ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
73  -- Table Subscription
74  CREATE TABLE IF NOT EXISTS `Subscription` (
75      `IdUser` INT(10) NOT NULL,
76      `Type` ENUM('Free', 'Premium') NOT NULL,
77      PRIMARY KEY(`IdUser`),
78      FOREIGN KEY(`IdUser`) REFERENCES User(`IdUser`) ON DELETE CASCADE ON UPDATE CASCADE
79  ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
80  -- Table Collection
81  CREATE TABLE IF NOT EXISTS `Collection` (
82      `IdCollection` INT(11) NOT NULL AUTO_INCREMENT,
83      `IdUser` INT(11) NOT NULL,
84      PRIMARY KEY(`IdCollection`),
85      FOREIGN KEY(`IdUser`) REFERENCES User(`IdUser`) ON DELETE CASCADE ON UPDATE CASCADE
86  ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
87  -- Table SongCollection
88  CREATE TABLE IF NOT EXISTS `SongCollection` (
89      `IdSong` INT(11) NOT NULL,
90      `IdCollection` INT(11) NOT NULL,
91      CONSTRAINT PRIMARY KEY pk(`IdCollection`, `IdSong`),
92      FOREIGN KEY(`IdSong`) REFERENCES Song(`IdSong`) ON DELETE CASCADE ON UPDATE CASCADE,
93      FOREIGN KEY(`IdCollection`) REFERENCES Collection(`IdCollection`) 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      `IdUser` INT(11) NOT NULL,
99      `Name` VARCHAR(40) NOT NULL,
100     `Private` BOOLEAN DEFAULT FALSE,
101     PRIMARY KEY(`IdPlaylist`),
102     FOREIGN KEY(`IdUser`) REFERENCES User(`IdUser`) ON DELETE CASCADE ON UPDATE CASCADE
103 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
104 -- Table PlaylistSong
105 CREATE TABLE IF NOT EXISTS `PlaylistSong` (
106     `IdPlaylist` INT(11) NOT NULL,
107     `IdSong` INT(11) NOT NULL,
108     CONSTRAINT PRIMARY KEY pk(`IdPlaylist`, `IdSong`),
109     FOREIGN KEY(`IdPlaylist`) REFERENCES Playlist(`IdPlaylist`) ON DELETE CASCADE ON UPDATE CASCADE,
110     FOREIGN KEY(`IdSong`) REFERENCES Song(`IdSong`) ON DELETE CASCADE ON UPDATE CASCADE
111 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
112 -- Table Queue
113 CREATE TABLE IF NOT EXISTS `Queue` (
114     `IdUser` INT(11) NOT NULL,
115     `IdSong` INT(11) NOT NULL,
116     `TimeStamp` TIMESTAMP NOT NULL,
117     CONSTRAINT PRIMARY KEY pk(`IdUser`, `IdSong`, `TimeStamp`),
118     FOREIGN KEY(`IdUser`) REFERENCES User(`IdUser`) ON DELETE CASCADE ON UPDATE CASCADE,
119     FOREIGN KEY(`IdSong`) REFERENCES Song(`IdSong`) ON DELETE CASCADE ON UPDATE CASCADE
120 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
121 -- Table Heard
122 CREATE TABLE IF NOT EXISTS `Heard` (
```

```
123     `IdUser` INT(11) NOT NULL,
124     `IdSong` INT(11) NOT NULL,
125     `TimeStamp` TIMESTAMP NOT NULL,
126     CONSTRAINT PRIMARY KEY pk(`IdUser`, `IdSong`, `TimeStamp`),
127     FOREIGN KEY(`IdUser`) REFERENCES User(`IdUser`) ON DELETE CASCADE ON UPDATE CASCADE,
128     FOREIGN KEY(`IdSong`) REFERENCES Song(`IdSong`) ON DELETE CASCADE ON UPDATE CASCADE
129 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
130 -- INSERT POPULATION
131 -- Insert into User
132 INSERT INTO User(`Name`, `Surname`, `Email`, `Administrator`, `Username`, `Password`)
133        VALUES('Andrea', 'Baldan', 'a.g.baldan@gmail.com', 0, 'codep', 'ciao'),
134              ('Federico', 'Angi', 'angiracing@gmail.com', 0, 'keepcalm', 'calm'),
135              ('Marco', 'Rossi', 'rossi@gmail.com', 0, 'rossi', 'marco'),
136              ('Luca', 'Verdi', 'verdi@yahoo.it', 0, 'verdi', 'luca'),
137              ('Alessia', 'Neri', 'neri@gmail.com', 0, 'neri', 'alessia');
138 -- Insert into Subscrition
139 INSERT INTO Subscription(`IdUser`, `Type`) VALUES(1, 'Free'), (2, 'Free');
140 -- Insert into Album
141 INSERT INTO Album(`Title`, `Author`, `Info`, `Year`, `Live`, `Location`)
142        VALUES('Inception Suite', 'Hans Zimmer', 'Inception movie soundtrack, composed by the Great
            Compositor Hans Zimmer', '2010-07-13', 0, NULL),
143              ('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', '1966-12-29', 0, NULL),
144              ('Hollywood in Vienna 2014', 'Randy Newman - David Newman', 'Annual cinematographic review
                   hosted in Vienna', '2014-09-23', 1, 'Vienna'),
145              ('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.', '1999-09-21', 0, NULL),
146              ('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.', '2002-06-19', 0, NULL),
147              ('Greatest Hits', 'Neil Young', 'Rock & Folk Rock greatest success songs by Neil Young', '
                  2004-06-21', 0, NULL);
148 -- Insert into Song
149 INSERT INTO Song(`IdAlbum`, `Title`, `Genre`, `Duration`, `IdImage`)
150        VALUES(1, 'Mind Heist', 'Orchestra', 203, 1),
151              (1, 'Dream is collapsing', 'Orchestra', 281, 1),
152              (1, 'Time', 'Orchestra', 215, 1),
153              (1, 'Half Remembered Dream', 'Orchestra', 71, 1),
154              (1, 'We Built Our Own World', 'Orchestra', 115, 1),
155              (1, 'Radical Notion', 'Orchestra', 222, 1),
156              (1, 'Paradox', 'Orchestra', 205, 1),
157              (2, 'Il Tramonto', 'Orchestra', 72, 2),
158              (2, 'L\'estasi dell\'oro', 'Orchestra', 202, 2),
159              (2, 'Morte di un soldato', 'Orchestra', 185, 2),
160              (2, 'Il Triello', 'Orchestra', 434, 2),
161              (3, 'The Simpsons', 'Orchestra', 172, 3),
162              (3, 'The war of the Roses', 'Orchestra', 272, 3),
163              (4, 'Somewhat Damaged', 'Industrial Metal', 271, 4),
164              (4, 'The Day The Whole World Went Away', 'Industrial Metal', 273, 4),
165              (4, 'We\'re In This Together', 'Industrial Metal', 436, 4),
166              (4, 'Just Like You Imagined', 'Industrial Metal', 229, 4),
167              (4, 'The Great Below', 'Industrial Metal', 317, 4),
168              (5, 'Hurt', 'Country', 218, 5),
169              (5, 'Danny Boy', 'Country', 199, 5),
170              (6, 'Old Man', 'Rock', 203, 6),
171              (6, 'Southern Man', 'Rock', 331, 6);
172 -- Insert into Cover
173 INSERT INTO Cover(`IdImage`, `IdAlbum`, `Path`)
174        VALUES(1, 1, 'img/covers/inception.png'),
```

```
175         (2, 2, 'img/covers/morricone.jpg'),
176         (3, 3, 'img/covers/hivlogo.jpg'),
177         (4, 4, 'img/covers/fragile.jpg'),
178         (5, 5, 'img/covers/nocover.jpg'),
179         (6, 6, 'img/covers/nocover.jpg');
180 -- Insert into Collection
181 INSERT INTO Collection(`IdUser`) VALUES(1), (2);
182 -- Insert into SongCollection
183 INSERT INTO SongCollection(`IdSong`, `IdCollection`) VALUES(1, 1), (2, 1), (3, 1), (2, 2);
184 -- Insert into Playlist
185 INSERT INTO Playlist(`IdUser`, `Name`, `Private`) VALUES(1, 'Score & Soundtracks', 0), (1, 'Southern
        Rock', 0), (2, 'Colonne sonore western', 0);
186 -- Insert into PlaylistSong
187 INSERT INTO PlaylistSong(`IdPlaylist`, `IdSong`) VALUES(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 21),
        (2, 22), (3, 5), (3, 7), (3, 4);
188 -- Insert into Queue
189 INSERT INTO Queue(`IdUser`, `IdSong`, `TimeStamp`)
190         VALUES(1, 1, '2015-04-28 18:50:03'),
191         (1, 5, '2015-04-28 18:54:06'),
192         (1, 1, '2015-04-28 19:01:43');
193 -- Insert into Heard
194 INSERT INTO Heard(`IdUser`, `IdSong`, `TimeStamp`)
195         VALUES(1, 1, '2015-04-28 18:50:03'),
196         (1, 5, '2015-04-28 18:54:06'),
197         (1, 1, '2015-04-28 19:01:43'),
198         (3, 7, '2015-04-29 18:51:02'),
199         (3, 11, '2015-04-29 17:23:15'),
200         (2, 9, '2015-04-30 21:12:52'),
201         (2, 1, '2015-05-02 22:21:22');
202 -- Insert into Follow
203 INSERT INTO Follow(`IdUser`, `IdFellow`) VALUES(1, 2), (1, 3), (2, 1), (3, 1);
204 SET FOREIGN_KEY_CHECKS = 1;
```

4.1 Trigger

Di seguito i trigger creati.

```
1  DROP TRIGGER IF EXISTS checkDuration;
2  DROP TRIGGER IF EXISTS errorTrigger;
3
4  DELIMITER $$
5
6  CREATE TRIGGER checkDuration
7  BEFORE INSERT ON `Song`
8  FOR EACH ROW
9  BEGIN
10 IF(NEW.Duration < 0) THEN
11     CALL RAISE_ERROR('Song duration cannot be negative');
12 END IF;
13 END $$
14
15 CREATE TRIGGER errorTrigger
16 BEFORE INSERT ON `Errors`
17 FOR EACH ROW
18 BEGIN
19     SET NEW = NEW.error;
20 END $$
21
22 DELIMITER ;
```

4.2 Funzioni e Procedure

Alcune funzioni e procedure implementate.

```
DROP FUNCTION IF EXISTS AlbumTotalDuration;

DELIMITER $$

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

DELIMITER ;

DROP PROCEDURE IF EXISTS RAISE_ERROR;

DELIMITER $$

CREATE PROCEDURE RAISE_ERROR (IN ERROR VARCHAR(256))
BEGIN
DECLARE V_ERROR VARCHAR(256);
SET V_ERROR := CONCAT('[ERROR: ', ERROR, ']');
INSERT INTO Errors VALUES(V_ERROR);
END $$

DELIMITER ;
```

5 Query

Alcune query significative.

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

```
SELECT s.Title, a.Title as AlbumTitle, u.Username, h.Timestamp
FROM Song s INNER JOIN Album a ON(s.IdAlbum = a.IdAlbum)
            INNER JOIN Heard h ON(h.IdSong = s.IdSong)
            INNER JOIN Follow f ON(f.IdFellow = h.IdUser)
            INNER JOIN User u ON(u.IdUser = f.IdFellow)
WHERE h.Timestamp BETWEEN ADDDATE(CURDATE(), -7) AND CURDATE()
    AND u.IdUser IN (SELECT u.IdUser FROM User u INNER JOIN Follow f ON(f.IdFellow = u.IdUser) WHERE f.
        IdUser = 1)
    ORDER BY h.Timestamp DESC LIMIT 10;
```

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 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.