GROOVECLAM

Andrea Giacomo Baldan 579117

June 17, 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 collezione personale di brani.

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.
Degli utenti che possono accedere alla libreria, interessano l'indirizzo e-mail e i dati anagrafici quali nome e cognome (opzionali). Ogni utente possiede una login, formata da username e password.
Gli utenti possono decidere di seguire altri utenti, eccetto se stessi. Ogni utente ha la possibilità di creare una propria collezione di brani preferiti selezionandoli tra quelli disponibili nella 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*
- **Login**: Rappresenta delle credenziali d'accesso per un utente.
  - Username: *String* «PK»
  - Password: *String*
  - Amministratore: *Boolean*
- **Iscrizioni**: Modella un piano di iscrizione.
  - Tipo: *Enum* ['Free', 'Premium']
- **Brani**: Rappresenta un brano musicale.
  - 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.
- **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

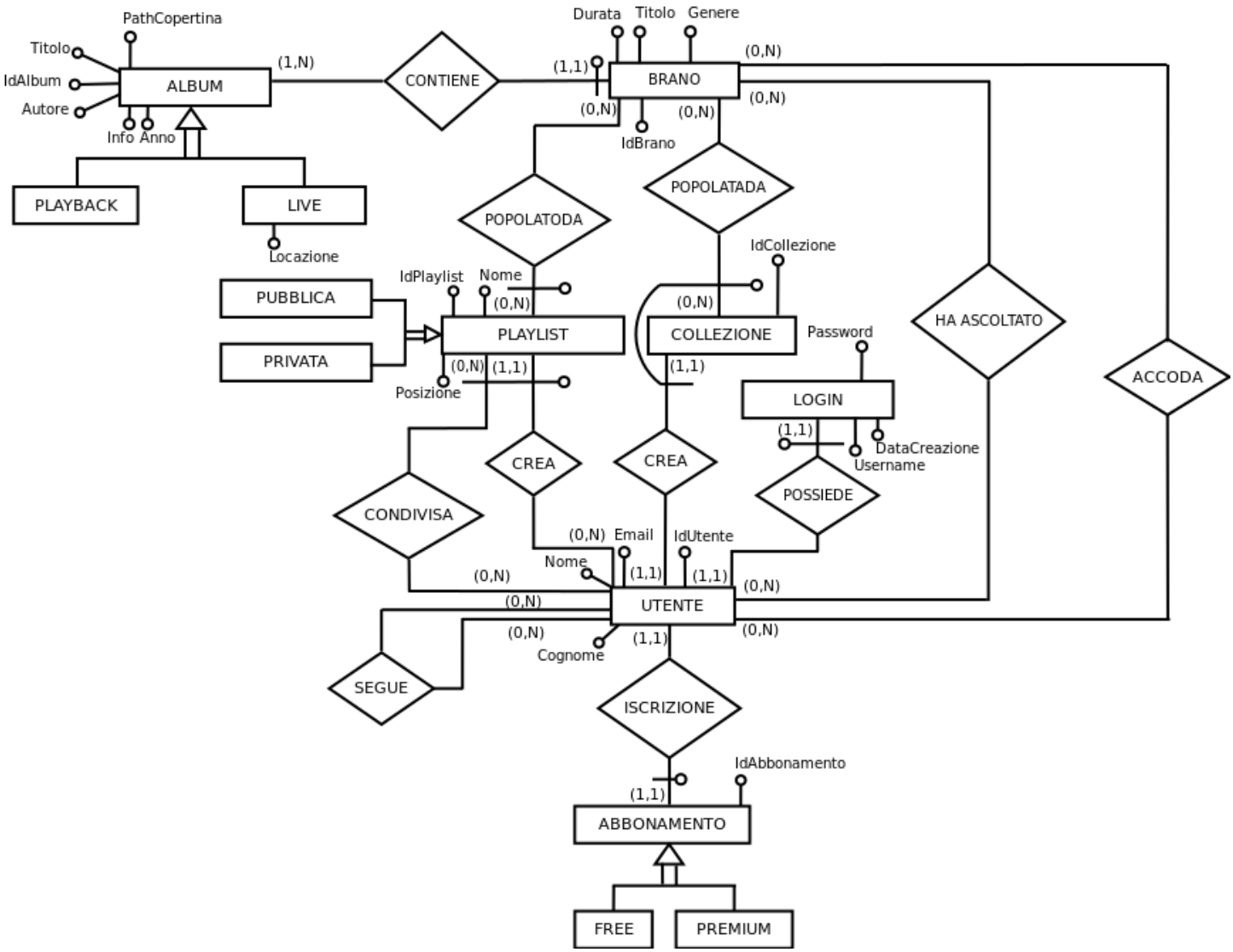
- **Collezioni**: Rappresenta una collezione di brani preferiti dall'utente.
  - IdCollezione: *Int* «PK»

2.2 Associazioni

- **Utenti-Login**: "Possiede"
  - Ogni utente possiede una login, ogni login e' posseduta da un utente.
  - Molteplicità 1 : 1
  - Totale verso **Utenti**, totale verso **Login**.
- **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**.
- **Playlist-Utente**: "Condivide"
  - Ogni playlist può essere condivisa con zero o più, ogni utente può condividere zero o più playlist.
  - Molteplicità N : N
  - Parziale verso **Playlist**, parziale 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**.
- **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**.
- **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



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 Schema Relazionale

Sono state create le tabelle **BraniCollezione** e **BraniPlaylist** per rappresentare le relazioni N:N tra **Brani-Collezioni** e tra **Brani-Playlist**. Inoltre per lo stesso motivo sono state create **Ascoltate** e **Code** tra **Utenti-Brani**.

Utenti (IdUtente, Nome*, Cognome*, EMail)

- PK(IdUtente)

Login (Username, Password, Amministratore, IdUtente)

- PK(Username)
- IdUtente FK(Utenti)

Iscrizioni (IdUtente, Tipo)

- PK(IdUtente)
- IdUtente FK(Utenti)

Brani (IdBrano, IdAlbum, Titolo, Genere, Durata)

- PK(IdBrano)
- IdAlbum FK(Album)

Album (IdAlbum, Titolo, Autore, Info*, Anno, Live, Locazione*, PathCopertina*)

- PK(IdAlbum)

Seguaci (IdUtente, IdSeguace)

- PK(IdUtente, IdSeguace)
- IdUtente FK(Utenti)
- IdSeguace FK(Utenti)

Collezioni (IdCollezione, IdUtente)

- PK(IdCollezione)
- IdUtente FK(Utenti)

BraniCollezione (IdBrano, IdCollezione)

- PK(IdBrano, IdCollezione)

- IdBrano FK(Brani)
- IdCollezione FK(Collezioni)

Playlist (IdPlaylist, IdUtente, Nome, Privata)

- PK(IdPlaylist)
- IdUtente FK(Utenti)

BraniPlaylist (IdPlaylist, IdBrano, Posizione)

- PK(IdPlaylist, IdBrano)
- IdPlaylist FK(Playlist)
- IdBrano FK(Brani)

Condivise (IdPlaylist, IdUtente)

- PK(IdPlaylist, IdUtente)
- IdPlaylist FK(Playlist)
- IdUtente FK(IdUtente)

Ascoltate (IdUtente, IdBrano, Timestamp)

- PK(IdUtente, IdBrano)
- IdUtente FK(Utenti)
- IdBrano FK(Brani)

Code (IdUtente, IdBrano, Posizione)

- PK(IdUtente, IdBrano)
- IdUtente FK(Utenti)
- IdBrano FK(Brani)

3.3.1 Note

Nel passaggio dalla progettazione concettuale alla progettazione logica, in fase di ristrutturazione ho deciso di creare una tabella **Code** per rappresentare la relazione **Accoda**, la cui implementazione fisica portera' alla creazione di una tabella molto simile alla tabella di giunzione che verra' a formarsi tra **Playlist** e **Brani**.
Si sono presentate 2 alternative:

- Inserire mediante un trigger una playlist "standard" nominata `CodaRiproduzione` ogni volta che viene inserito un nuovo utente, la quale conterra' la coda di riproduzione dell'utente in questione.
- Creare una tabella apposita per la gestione delle code di riproduzione.

Ho infine optato per la seconda soluzione, cosi' da mantenere una separazione dei concetti ed evitare eventuali ridondanze, e maggiore chiarezza .

3.4 Analisi Ridondanze

Nella tabella **Album** vi è la possibilità di inserire un attributo ridondante (nBrani) al fine di diminuire il carico di lavoro della BD. Per valutare la convenienza o meno della scelta si è deciso di utilizzare una tabella di carico riferita a dati verosimili, per un istanza della base di dati ad uno stato ancora "giovane" (cioè con ancora poche entry). Quindi circa 200 album caricati e 1600 brani.

- Operazioni:
  - Memorizzare un nuovo brano con relativo album di appartenenza
  - Stampare tutti i dati di un album(incluso il numero di brani)

Table 1: Tavola dei volumi

Concetto	Tipo	Volume
Album	E	200
Brani	E	1600
Contiene	R	1600

Table 2: Tavola delle operazioni

Operazione	Frequenza
Op 1	50
Op 2	1000

- **Caso con attributo ridondante**
  Assumendo che il numero di brani presenti in un album richieda 1 byte (3 cifre sono più che sufficienti per memorizzare il numero di brani in un album), abbiamo che il dato ridondante richiede 1 x 200 = 200 byte di memoria aggiuntiva.
  L'operazione 1 richiede un accesso in scrittura all'entità **Brani**, un accesso in scrittura all'associazione **Contiene**, un accesso in lettura e uno in scrittura all'entità **Album**, per cercare l'album interessato e per incrementare il numero di brani rispettivamente, il tutto ripetuto 50 volte al giorno, per un totale di 150 accessi in scrittura e 50 in lettura. Il costo dell'operazione 2 richiede solo un accesso in lettura all'entità **Album** ripetuto 1000 volte al giorno.
  Supponendo che gli accessi in scrittura abbiano un costo doppio rispetto agli accessi in lettura abbiamo che il costo totale è di 150 + 100 = 250 per l'operazione 1 e 1000 per l'operazione 2, totale 1250 accessi al giorno.

- **Caso senza attributo ridondante**
  Abbiamo un accesso in scrittura all'entità **Brani** ed uno in scrittura all'associazione **Contiene** per un totale di 100 accessi in scrittura al giorno per l'operazione 1.
  Per l'operazione 2 abbiamo un accesso in lettura all'entità **Album** e 5 accessi in lettura in media all'associazione **Contiene**, tutto ripetuto 1000 volte, abbiamo un totale di 1000 + 8000 + 200 = 9200 accessi al giorno senza ridondanza.
  Si può dunque concludere che ~8000 accessi in più contro un risparmio di 200 byte giustificano l'utilizzo di un attributo ridondante nella tabella **Album**.

3.4.1 Note

Tale ragionamento puo' tovare applicazione in maniera equivalente o comunque molto simile anche nelle tabelle **Playlist-Brani** e **Collezioni-Brani**, ma non ho ritenuto di procedere data la poca utilita' di tale scelta in ragione della struttura dell'interfaccia web progettata, dove utilizzo funzioni PHP come `count` per ottenere la lunghezza di un array, ad esempio l'array contenente i brani di una collezione ottenendo quindi il numero di brani della collezione effettivo.

3.5 Associazioni

- **Utenti-Login**: "Possiede"
  - Ogni utente possiede una login, ogni login e' posseduta da un utente.
  - Molteplicità $1:1$
  - Totale verso **Utenti**, totale verso **Login**.
  - Chiave esterna non-nulla in **Login** verso **Utenti**.
- **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**.

- **Playlist-Utenti**: "Condivisa"
    - Ogni playlist può essere condivisa con zero o più, ogni utente può condividere zero o più playlist.
    - Molteplicità N : N
    - Parziale verso **Playlist**, parziale verso **Utenti**.
    - Nuova tabella **Condivise**, attributi:
        * IdPlaylist: *Int* «PK» «FK(Playlist)»
        * IdUtente: *Int* «PK» «FK(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**.

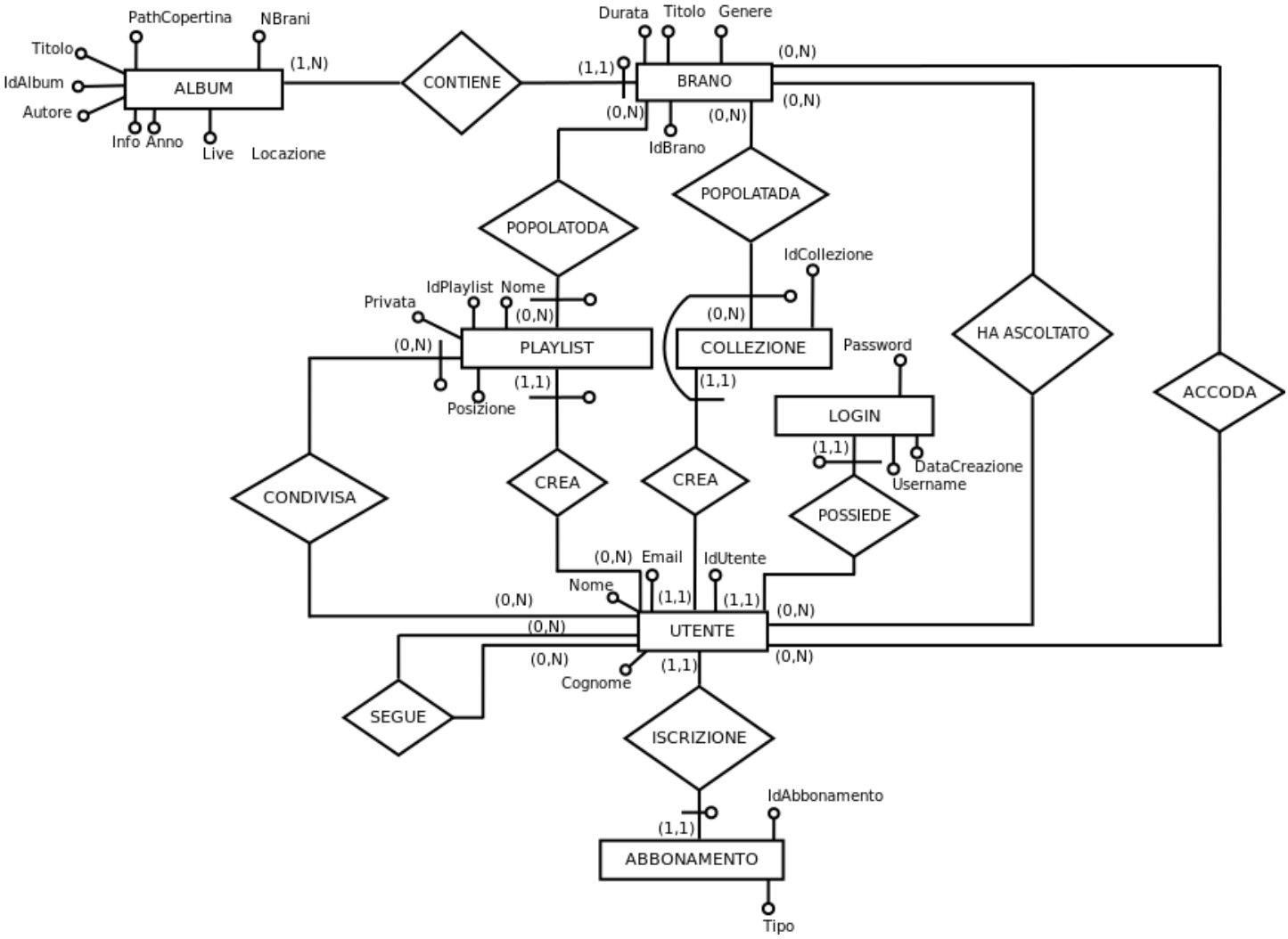
- **Playlist-Brani**: "PopolataDa"
    - Ogni playlist è popolata da zero o più brani, ogni brano popola zero o più playlist.
    - Molteplicità N : N

  - 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**.

- **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)»

3.6 Schema E/R ristrutturato



3.6.1 Note ristrutturazione

A seguito delle analisi eseguite sullo schema concettuale, ho deciso di risolvere le gerarchie accorpandole all'entita' padre, in quanto nessuna di esse ha interazioni significative con le altre entita' della base di dati. Visti i risultati dell'analisi sulle ridondanze ho deciso di inserire l'attributo ridondante `NBrani` nella tabella `Album`, in quanto il rapporto tra il costo computazionale e l'effettivo carico maggiorato favoriva di molto l'opzione valutata.

4 Implementazione Fisica

Query di implementazione DDL SQL della base di dati. Sorgente in `grooveclam.sql`, popolamento in `fill.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 sono contenuti in `triggers.sql` e le procedure in `procedures.sql`.

```sql
SET FOREIGN_KEY_CHECKS = 0;

DROP TABLE IF EXISTS `Errori`;
DROP TABLE IF EXISTS `Album`;
DROP TABLE IF EXISTS `Brani`;
DROP TABLE IF EXISTS `Utenti`;
DROP TABLE IF EXISTS `Seguaci`;
DROP TABLE IF EXISTS `Iscrizioni`;
DROP TABLE IF EXISTS `Collezioni`;
DROP TABLE IF EXISTS `BraniCollezione`;
DROP TABLE IF EXISTS `Playlist`;
DROP TABLE IF EXISTS `BraniPlaylist`;
DROP TABLE IF EXISTS `Condivise`;
DROP TABLE IF EXISTS `Code`;
DROP TABLE IF EXISTS `Ascoltate`;
DROP TABLE IF EXISTS `Login`;

-- Table di supporto Errori
CREATE TABLE IF NOT EXISTS `Errori` (
    `Errore` VARCHAR(256) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=Latin1;
-- Table Album
CREATE TABLE IF NOT EXISTS `Album` (
    `IdAlbum` INT(11) NOT NULL AUTO_INCREMENT,
    `Titolo` VARCHAR(200) NOT NULL,
    `Autore` VARCHAR(200) NOT NULL,
    `Info` VARCHAR(300) DEFAULT NULL,
    `NBrani` INT(11) NOT NULL,
    `Anno` YEAR DEFAULT NULL,
    `Live` BOOLEAN DEFAULT FALSE,
    `Locazione` VARCHAR(100) DEFAULT NULL,
    `PathCopertina` VARCHAR(100) NOT NULL DEFAULT "img/covers/nocover.jpg",
    PRIMARY KEY(`IdAlbum`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
-- Table Brani
CREATE TABLE IF NOT EXISTS `Brani` (
    `IdBrano` INT(11) NOT NULL AUTO_INCREMENT,
    `IdAlbum` INT(11) NOT NULL,
    `Titolo` VARCHAR(200) NOT NULL,
    `Genere` VARCHAR(40) NOT NULL,
    `Durata` INT(11),
    PRIMARY KEY(`IdBrano`),
    FOREIGN KEY(`IdAlbum`) REFERENCES Album(`IdAlbum`) ON DELETE CASCADE ON
        UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

```
45 -- Table Utenti
46 CREATE TABLE IF NOT EXISTS `Utenti` (
47     `IdUtente` INT(11) NOT NULL AUTO_INCREMENT,
48     `Nome` VARCHAR(40) DEFAULT NULL,
49     `Cognome` VARCHAR(40) DEFAULT NULL,
50     `Email` VARCHAR(40) NOT NULL,
51     PRIMARY KEY(`IdUtente`)
52 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
53 -- Table Login
54 CREATE TABLE IF NOT EXISTS `Login` (
55     `Username` VARCHAR(40) NOT NULL,
56     `Password` VARCHAR(40) NOT NULL,
57     `DataCreazione` TIMESTAMP NOT NULL,
58     `IdUtente` INT(11) NOT NULL,
59     PRIMARY KEY(`Username`),
60     FOREIGN KEY(`IdUtente`) REFERENCES Utenti(`IdUtente`) ON DELETE CASCADE ON
           UPDATE CASCADE
61 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
62 -- Table Seguaci
63 CREATE TABLE IF NOT EXISTS `Seguaci` (
64     `IdUtente` INT(11) NOT NULL,
65     `IdSeguace` INT(11) NOT NULL,
66     CONSTRAINT PRIMARY KEY pk(`IdUtente`, `IdSeguace`),
67     FOREIGN KEY(`IdUtente`) REFERENCES Utenti(`IdUtente`) ON DELETE CASCADE ON
           UPDATE CASCADE,
68     FOREIGN KEY(`IdSeguace`) REFERENCES Utenti(`IdUtente`) ON DELETE CASCADE
          ON UPDATE CASCADE,
69     CHECK(`IdUtente` != `IdSeguace`)
70 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
71 -- Table Iscrizioni
72 CREATE TABLE IF NOT EXISTS `Iscrizioni` (
73     `IdUtente` INT(10) NOT NULL,
74     `Tipo` ENUM('Free', 'Premium') NOT NULL,
75     PRIMARY KEY(`IdUtente`),
76     FOREIGN KEY(`IdUtente`) REFERENCES Utenti(`IdUtente`) ON DELETE CASCADE ON
           UPDATE CASCADE
77 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
78 -- Table Collezioni
79 CREATE TABLE IF NOT EXISTS `Collezioni` (
80     `IdCollezione` INT(11) NOT NULL AUTO_INCREMENT,
81     `IdUtente` INT(11) NOT NULL,
82     PRIMARY KEY(`IdCollezione`),
83     FOREIGN KEY(`IdUtente`) REFERENCES Utenti(`IdUtente`) ON DELETE CASCADE ON
           UPDATE CASCADE
84 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
85 -- Table BraniCollezione
86 CREATE TABLE IF NOT EXISTS `BraniCollezione` (
87     `IdBrano` INT(11) NOT NULL,
88     `IdCollezione` INT(11) NOT NULL,
```

```
89      CONSTRAINT PRIMARY KEY pk(`IdCollezione`, `IdBrano`),
90      FOREIGN KEY(`IdBrano`) REFERENCES Brani(`IdBrano`) ON DELETE CASCADE ON
            UPDATE CASCADE,
91      FOREIGN KEY(`IdCollezione`) REFERENCES Collezioni(`IdCollezione`) ON
            DELETE CASCADE ON UPDATE CASCADE
92  ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
93  -- Table Playlist
94  CREATE TABLE IF NOT EXISTS `Playlist` (
95      `IdPlaylist` INT(11) NOT NULL AUTO_INCREMENT,
96      `IdUtente` INT(11) NOT NULL,
97      `Nome` VARCHAR(40) NOT NULL,
98      `Tipo` ENUM('Pubblica', 'Privata') DEFAULT 'Pubblica',
99      PRIMARY KEY(`IdPlaylist`),
100     FOREIGN KEY(`IdUtente`) REFERENCES Utenti(`IdUtente`) ON DELETE CASCADE ON
            UPDATE CASCADE
101 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
102 -- Table BraniPlaylist
103 CREATE TABLE IF NOT EXISTS `BraniPlaylist` (
104     `IdPlaylist` INT(11) NOT NULL,
105     `IdBrano` INT(11) NOT NULL,
106     `Posizione` INT(11) NOT NULL,
107     CONSTRAINT PRIMARY KEY pk(`IdPlaylist`, `IdBrano`),
108     FOREIGN KEY(`IdPlaylist`) REFERENCES Playlist(`IdPlaylist`) ON DELETE
            CASCADE ON UPDATE CASCADE,
109     FOREIGN KEY(`IdBrano`) REFERENCES Brani(`IdBrano`) ON DELETE CASCADE ON
            UPDATE CASCADE
110 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
111 -- Table Condivise
112 CREATE TABLE IF NOT EXISTS `Condivise` (
113     `IdPlaylist` INT(11) NOT NULL,
114     `IdUtente` INT(11) NOT NULL,
115     CONSTRAINT PRIMARY KEY pk(`IdPlaylist`, `IdUtente`),
116     FOREIGN KEY(`IdPlaylist`) REFERENCES Playlist(`IdPlaylist`) ON DELETE
            CASCADE ON UPDATE CASCADE,
117     FOREIGN KEY(`IdUtente`) REFERENCES Utenti(`IdUtente`) ON DELETE CASCADE ON
            UPDATE CASCADE
118 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
119 -- Table Code
120 CREATE TABLE IF NOT EXISTS `Code` (
121     `IdUtente` INT(11) NOT NULL,
122     `IdBrano` INT(11) NOT NULL,
123     `Posizione` INT(11) NOT NULL,
124     CONSTRAINT PRIMARY KEY pk(`IdUtente`, `IdBrano`, `Posizione`),
125     FOREIGN KEY(`IdUtente`) REFERENCES Utenti(`IdUtente`) ON DELETE CASCADE ON
            UPDATE CASCADE,
126     FOREIGN KEY(`IdBrano`) REFERENCES Brani(`IdBrano`) ON DELETE CASCADE ON
            UPDATE CASCADE
127 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
128 -- Table Ascoltate
```

```
129 CREATE TABLE IF NOT EXISTS `Ascoltate` (
130     `IdUtente` INT(11) NOT NULL,
131     `IdBrano` INT(11) NOT NULL,
132     `Timestamp` TIMESTAMP NOT NULL,
133     CONSTRAINT PRIMARY KEY pk(`IdUtente`, `IdBrano`, `Timestamp`),
134     FOREIGN KEY(`IdUtente`) REFERENCES Utenti(`IdUtente`) ON DELETE CASCADE ON
          UPDATE CASCADE,
135     FOREIGN KEY(`IdBrano`) REFERENCES Brani(`IdBrano`) ON DELETE CASCADE ON
        UPDATE CASCADE
136 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
137 -- Insert into Utente
138 INSERT INTO Utenti(`Nome`, `Cognome`, `Email`)
139        VALUES('Andrea', 'Baldan', 'a.g.baldan@gmail.com'),
140              ('Federico', 'Angi', 'angiracing@gmail.com'),
141              ('Marco', 'Rossi', 'rossi@gmail.com'),
142              ('Luca', 'Verdi', 'verdi@yahoo.it'),
143              ('Alessia', 'Neri', 'neri@gmail.com');
144 -- Insert into Login
145 INSERT INTO Login(`Username`, `Password`, `DataCreazione`, `IdUtente`)
146        VALUES('codep', MD5('ciao'), '2015-04-29 18:51:00', 1),
147              ('keepcalm', MD5('calm'), '2015-05-24 19:50:01', 2),
148              ('rossi', MD5('marco'), '2015-05-28 19:50:04', 3),
149              ('verdi', MD5('luca'), '2015-05-29 19:50:07', 4),
150              ('neri', MD5('Alessia'), '2015-05-29 20:50:09', 5);
151 -- Insert into Iscrizioni
152 INSERT INTO Iscrizioni(`IdUtente`, `Tipo`)
153        VALUES(1, 'Free'),
154              (2, 'Free'),
155              (3, 'Premium'),
156              (4, 'Free'),
157              (5, 'Premium');
158 -- Insert into Album
159 INSERT INTO Album(`Titolo`, `Autore`, `Info`, `Anno`, `Live`, `Locazione`, `
       PathCopertina`)
160        VALUES('Inception Suite', 'Hans Zimmer', 'Inception movie soundtrack,
             composed by the Great Compositor Hans Zimmer', '2010', 0, NULL, 'img
             /covers/inception.png'),
161              ('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, 'img/covers/morricone.jpg'),
162              ('Hollywood in Vienna 2014', 'Randy Newman - David Newman', '
                 Annual cinematographic review hosted in Vienna', '2014', 1, '
                 Vienna', 'img/covers/hivlogo.jpg'),
163              ('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, 'img/covers/fragile.jpg'),
164              ('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, 'img/covers/nocover.jpg'),
165             ('Greatest Hits', 'Neil Young', 'Rock & Folk Rock greatest
                success songs by Neil Young', '2004', 0, NULL, 'img/covers/
                nocover.jpg');
166 -- Insert into Brani
167 INSERT INTO Brani(`IdAlbum`, `Titolo`, `Genere`, `Durata`)
168         VALUES(1, 'Mind Heist', 'Orchestra', 203),
169               (1, 'Dream is collapsing', 'Orchestra', 281),
170               (1, 'Time', 'Orchestra', 215),
171               (1, 'Half Remembered Dream', 'Orchestra', 71),
172               (1, 'We Built Our Own World', 'Orchestra', 115),
173               (1, 'Radical Notion', 'Orchestra', 222),
174               (1, 'Paradox', 'Orchestra', 205),
175               (2, 'Il Tramonto', 'Orchestra', 72),
176               (2, 'L\'estasi dell\'oro', 'Orchestra', 202),
177               (2, 'Morte di un soldato', 'Orchestra', 185),
178               (2, 'Il Triello', 'Orchestra', 434),
179               (3, 'The Simpsons', 'Orchestra', 172),
180               (3, 'The war of the Roses', 'Orchestra', 272),
181               (4, 'Somewhat Damaged', 'Industrial Metal', 271),
182               (4, 'The Day The Whole World Went Away', 'Industrial Metal', 273)
                  ,
183               (4, 'We\'re In This Together', 'Industrial Metal', 436),
184               (4, 'Just Like You Imagined', 'Industrial Metal', 229),
185               (4, 'The Great Below', 'Industrial Metal', 317),
186               (5, 'Hurt', 'Country', 218),
187               (5, 'Danny Boy', 'Country', 199),
188               (6, 'Old Man', 'Rock', 203),
189               (6, 'Southern Man', 'Rock', 331);
190 -- Insert into BraniCollezione
191 INSERT INTO BraniCollezione(`IdBrano`, `IdCollezione`) VALUES(1, 1), (2, 1),
      (3, 1), (7, 1), (14, 1), (12, 1), (17, 1), (18, 1), (2, 2);
192 -- Insert into Playlist
193 INSERT INTO Playlist(`IdUtente`, `Nome`, `Tipo`) VALUES(1, 'Score &
      Soundtracks', 'Pubblica'), (1, 'Southern Rock', 'Pubblica'), (2, 'Colonne
      sonore western', 'Pubblica');
194 -- Insert into BraniPlaylist
195 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);
196 -- Insert Condivise
197 -- Insert into Code
198 INSERT INTO Code(`IdUtente`, `IdBrano`, `Posizione`)
199         VALUES(1, 1, 1),
200               (1, 5, 2),
```

```
201             (1, 1, 3),
202             (1, 12, 4),
203             (1, 10, 5),
204             (2, 1, 1);
205 -- Insert into Ascoltate
206 INSERT INTO Ascoltate(`IdUtente`, `IdBrano`, `Timestamp`)
207       VALUES(1, 1, '2015-04-28 18:50:03'),
208             (1, 5, '2015-04-28 18:54:06'),
209             (1, 1, '2015-04-28 19:01:43'),
210             (2, 7, '2015-04-29 18:51:02'),
211             (3, 11, '2015-04-29 17:23:15'),
212             (3, 9, '2015-04-30 21:12:52'),
213             (2, 1, '2015-05-02 22:21:22');
214 -- Insert into Seguaci
215 INSERT INTO Seguaci(`IdUtente`, `IdSeguace`) VALUES(1, 2), (1, 3), (2, 1), (3,
        1);
216
217 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`.
- `insertAutoAdminSubs`: Trigger di controllo sull'inserimento di un nuovo utente con privilegi di amministrazione, crea un record nella tabella `Iscrizioni` associato al nuovo amministratore creato, dotandolo di privilegi `Premium`.
- `updateAutpAdminSubs`: Trigger di controllo sull'aggiornamento di un utente già presente nella base di dati, se viene aggiornato il campo `Amministratore` a `TRUE`, viene creato o, aggiornato se già esistente, il campo `Tipo` nella tabella `Iscrizioni`.
- `insertAutoSongNumber`: Trigger di aggiornamento, dopo l'inserimento nella tabella `Brani` aumenta di 1 il contatore `NBrani` all'interno della tabella `Album` sul record associato al brano inserito.
- `updateAutoSongNumber`: Trigger di controllo, come sopra ma cattura l'evento `UPDATE`, e si occupa di decrementare il contatore del vecchio album a cui il brano era associato.

- `checkCollectionSize`: Trigger di controllo, prima dell'inserimento nella tabella `Brani` controlla il tipo di iscrizione dell'utente che intende eseguire l'inserimento, e, nel caso si tratti di un utente "Free" genera un errore e impedisce l'inserimento.
- `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  DROP TRIGGER IF EXISTS insertAutoAdminSubs;
7  DROP TRIGGER IF EXISTS updateAutoAdminSubs;
8  DROP TRIGGER IF EXISTS insertAutoSongNumber;
9  DROP TRIGGER IF EXISTS updateAutoSongNumber;
10
11 DELIMITER $$
12
13 CREATE TRIGGER checkDuration
14 BEFORE INSERT ON `Brani`
15 FOR EACH ROW
16 BEGIN
17 IF(NEW.Durata < 0) THEN
18     CALL RAISE_ERROR('La durata di un brano non pu  essere negativa');
19 END IF;
20 END $$
21
22 DELIMITER ;
23
24 DELIMITER $$
25
26 CREATE TRIGGER errorTrigger
27 BEFORE INSERT ON `Errori`
28 FOR EACH ROW
29 BEGIN
30     SET NEW = NEW.errore;
31 END $$
32
33 DELIMITER ;
34
35 DELIMITER $$
36
37 CREATE TRIGGER checkFollower
38 BEFORE INSERT ON `Seguaci`
39 FOR EACH ROW
40 BEGIN
41     IF NEW.IdUtente = NEW.IdSeguace THEN
```

```
42          CALL RAISE_ERROR('Un utente non pu  seguire se stesso (IdUtente e
                IdSeguace devono essere diversi fra loro)');
43      END IF;
44  END $$
45
46  DELIMITER ;
47
48  DELIMITER $$
49
50  CREATE TRIGGER checkCoverImage
51  BEFORE INSERT ON `Copertine`
52  FOR EACH ROW
53  BEGIN
54      IF NEW.Path = '' THEN
55          SET NEW.Path = 'img/covers/nocover.jpg';
56      END IF;
57  END $$
58
59  DELIMITER ;
60
61  DELIMITER $$
62
63  CREATE TRIGGER insertAutoAdminSubs
64  BEFORE INSERT ON `Login`
65  FOR EACH ROW
66  BEGIN
67      IF(NEW.Amministratore = 1) THEN
68          INSERT INTO `Iscrizioni` (`IdUtente`, `Tipo`) VALUES(NEW.IdUtente, '
                Premium')
69          ON DUPLICATE KEY UPDATE Tipo = 'Premium';
70      ELSE
71          INSERT INTO `Iscrizioni` (`IdUtente`, `Tipo`) VALUES(NEW.IdUtente, '
                Free');
72      END IF;
73  END $$
74
75  DELIMITER ;
76
77  DELIMITER $$
78
79  CREATE TRIGGER updateAutoAdminSubs
80  BEFORE UPDATE ON `Login`
81  FOR EACH ROW
82  BEGIN
83      IF(NEW.Amministratore = 1) THEN
84          INSERT INTO `Iscrizioni` (`IdUtente`, `Tipo`) VALUES(NEW.IdUtente, '
                Premium')
85          ON DUPLICATE KEY UPDATE Tipo = 'Premium';
86      END IF;
```

```
87  END $$
88
89  DELIMITER ;
90
91  DELIMITER $$
92
93  CREATE TRIGGER insertAutoSongNumber
94  AFTER INSERT ON `Brani`
95  FOR EACH ROW
96  BEGIN
97      DECLARE ida INTEGER DEFAULT -1;
98      SELECT a.IdAlbum INTO ida
99      FROM `Album` a
100     WHERE a.IdAlbum = NEW.IdAlbum;
101     IF(ida <> -1) THEN
102         UPDATE `Album` SET NBrani = NBrani + 1 WHERE IdAlbum = ida;
103     END IF;
104 END $$
105
106 DELIMITER ;
107
108 DELIMITER $$
109
110 CREATE TRIGGER updateAutoSongNumber
111 AFTER UPDATE ON `Brani`
112 FOR EACH ROW
113 BEGIN
114     DECLARE ida INTEGER DEFAULT -1;
115     SELECT a.IdAlbum INTO ida
116     FROM `Album` a
117     WHERE a.IdAlbum = NEW.IdAlbum;
118     IF(ida <> -1) THEN
119         UPDATE `Album` SET NBrani = NBrani - 1 WHERE IdAlbum = OLD.IdAlbum;
120         UPDATE `Album` SET NBrani = NBrani + 1 WHERE IdAlbum = ida;
121     END IF;
122 END $$
123
124 DELIMITER ;
125
126 DELIMITER $$
127 CREATE TRIGGER checkCollectionSize
128 BEFORE INSERT ON `BraniCollezione`
129 FOR EACH ROW
130 BEGIN
131     DECLARE numSong INTEGER DEFAULT 0;
132     DECLARE idUser INTEGER DEFAULT 0;
133     DECLARE subType VARCHAR(7) DEFAULT '';
134     SELECT DISTINCT c.IdUtente INTO idUser
```

```
135     FROM BraniCollezione bc INNER JOIN Collezioni c ON(bc.IdCollezione = c.
            IdCollezione)
136     WHERE bc.IdCollezione = NEW.IdCollezione;
137     SELECT COUNT(IdBrano) INTO numSong
138     FROM BraniCollezione
139     WHERE IdCollezione = NEW.IdCollezione;
140     SELECT DISTINCT i.Tipo INTO subType
141     FROM Iscrizioni i INNER JOIN Login l ON(i.IdUtente = l.IdUtente)
142     WHERE l.IdUtente = idUser;
143     IF(numSong = 50 && subType = 'Free') THEN
144         CALL RAISE_ERROR('Maximum limit for collected songs reached for a free
                subscription account.');
145     END IF;
146 END $$
147
148 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.

```
1  DROP FUNCTION IF EXISTS albumTotalDuration;
2  DROP FUNCTION IF EXISTS elegibleForPrize;
3
4  DELIMITER $$
5
6  CREATE FUNCTION albumTotalDuration(IdAlbum INT)
7  RETURNS VARCHAR(5)
8  BEGIN
9  DECLARE Seconds INT UNSIGNED;
10 SELECT SUM(b.Durata) INTO Seconds FROM Brani b WHERE b.IdAlbum = IdAlbum;
11 RETURN CONCAT(FLOOR(Seconds / 60), ':', (Seconds % 60));
12 END $$
13
14 DELIMITER ;
```

```
15
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 Errori 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.

5.1 Query 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)
```

5.2 Query 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)
```

5.3 Query 3

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

```
SELECT u.Username, b.Titolo, COUNT(b.IdBrano) AS Conto
FROM Brani b INNER JOIN Ascoltate h ON(b.IdBrano = h.IdBrano)
             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)
```

5.4 Query 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)
```

5.5 Query 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)
```

5.6 Query 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)
```

5.7 Query 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)
```

5.8 Query 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)
```

5.9 Query 9

Trova gli utenti con account inattivo da almeno 60 giorni e stampa la data del loro ultimo ascolto

```
1   SELECT u.*, a.Timestamp AS UltimoAscolto
2   FROM Utenti u INNER JOIN Login l ON(u.IdUtente = l.IdUtente)
```

```
3                 INNER JOIN Ascoltate a ON(u.IdUtente = a.IdUtente)
4 WHERE l.DataCreazione < DATE_SUB(CURDATE(), INTERVAL 60 DAY)
5 AND u.IdUtente IN (SELECT a1.IdUtente
6                     FROM Ascoltate a1
7                     WHERE a1.Timestamp < DATE_SUB(CURDATE(), INTERVAL 60 DAY))
8 ORDER BY a.Timestamp DESC LIMIT 1;
```

```
+----------+--------+---------+---------------------+---------------------+
| IdUtente | Nome   | Cognome | Email               | Timestamp           |
+----------+--------+---------+---------------------+---------------------+
|        2 | Andrea | Baldan  | a.g.baldan@gmail.com | 2015-04-28 19:01:43 |
+----------+--------+---------+---------------------+---------------------+
1 row in set (0.00 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 mediante 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 propria collezione o alla coda di riproduzione; **Albums** contiene tutti gli album presenti nella piattaforma, 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, accedibili solo dopo l'autenticazione, danno la possibilità di privatizzare o rendere pubbliche le proprie playlist. **Queue** infine ospita la coda di riproduzione, ordinate in base alla posizione di accodamento, modificabile.
E' possibile modificare i dati relativi al proprio account, incluso il piano di'iscrizione, utilizzando la pagina accessibile clickando sul bottone in alto a destra **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.
Per i controlli sull'effettiva autenticazione ad esempio, viene creata una variabile di sessione `logged`, essa vale 1 se esiste un utente loggato, mentre la variabile `uid` si occupa di tenere traccia dell'id dell'utente autenticato.
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.