

GROOVECLAM

Andrea Giacomo Baldan 579117

May 25, 2015

Contents

1	Analisi Dei Requisiti	2
2	Progettazione concettuale	2
2.1	Classi	2
2.2	Associazioni	4
2.3	Schema E/R	6
3	Progettazione Logica	7
3.1	Gerarchie	7
3.2	Chiavi Primarie	7
3.3	Associazioni	7
4	Implementazione Fisica	9
4.1	Trigger	13
4.2	Funzioni e Procedure	14
5	Query	15
6	Interfaccia Web	17
6.1	Organizzazione e Struttura Generale	17
6.1.1	Esempi	17
6.2	Pagine Principali	17
6.3	Mantenimento Stato Pagine	17
6.4	Note	18

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 ordinari 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**.

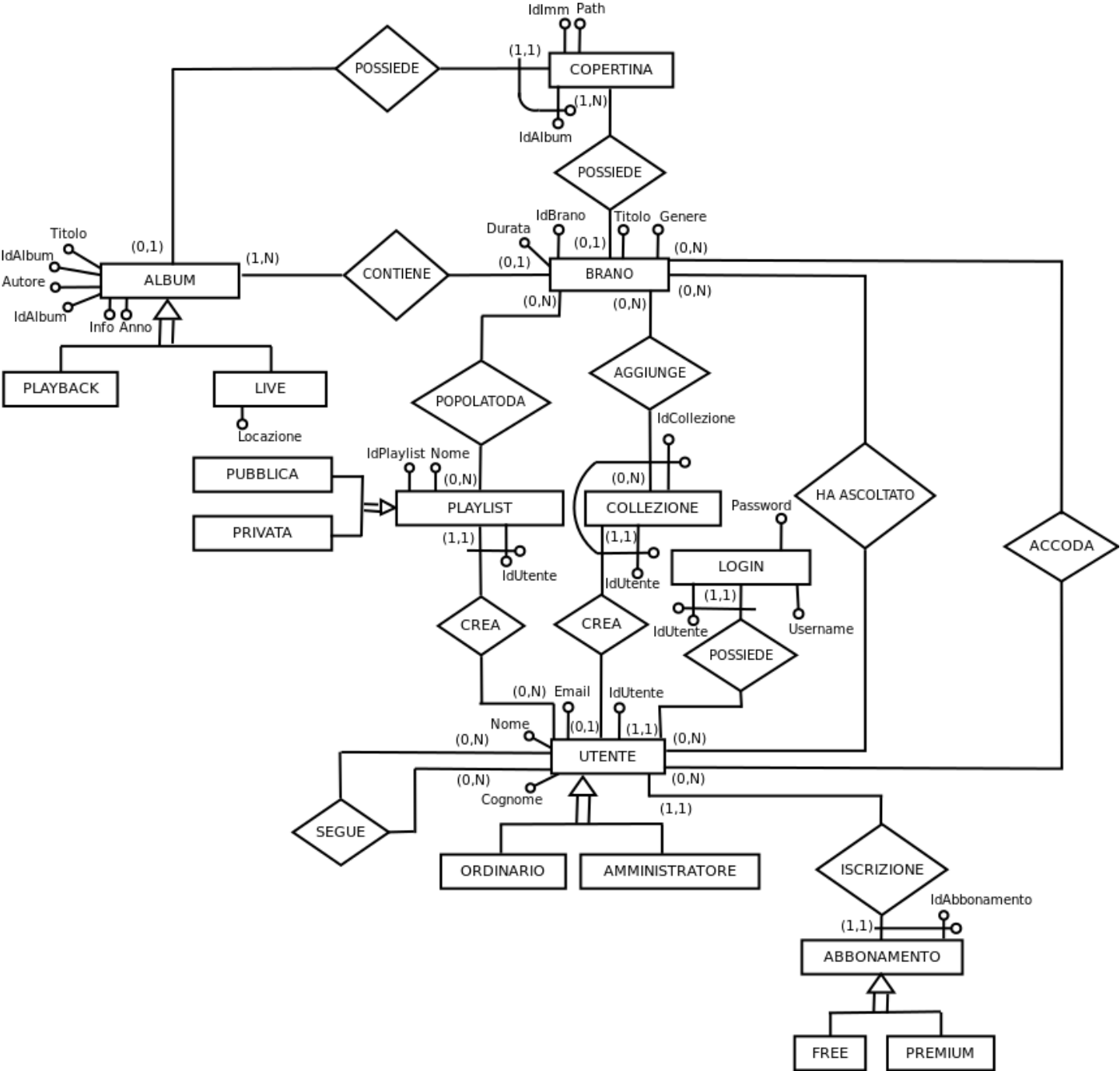
- **Utenti-Iscrizioni:** "Iscritto"

- 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



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 giustificare 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**)»

* IdBranco: *Int* «PK» «FK(Branci)»

* 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-Branci:** "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 **Branci**.
- Nuova tabella **BranciPlaylist**, attributi:
 - * IdPlaylist: *Int* «PK» «FK(Playlist)»
 - * IdBranco: *Int* «PK» «FK(Branci)»

- **Branci-Album:** "AppartieneA"

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

```

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 'Collezione';
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     'IdBranco' 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('IdBranco'),
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
94     CASCADE
95 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
96 -- Table Playlist
97 CREATE TABLE IF NOT EXISTS 'Playlist' (
98     'IdPlaylist' INT(11) NOT NULL AUTO_INCREMENT,
99     'IdUtente' INT(11) NOT NULL,
100     'Nome' VARCHAR(40) NOT NULL,
101     'Privata' BOOLEAN DEFAULT FALSE,
102     PRIMARY KEY('IdPlaylist'),
103     FOREIGN KEY('IdUtente') REFERENCES Utenti('IdUtente') ON DELETE CASCADE ON UPDATE CASCADE
104 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
105 -- Table BraniPlaylist
106 CREATE TABLE IF NOT EXISTS 'BraniPlaylist' (
107     'IdPlaylist' INT(11) NOT NULL,
108     'IdBrano' INT(11) NOT NULL,
109     'Posizione' INT(11) NOT NULL,
110     CONSTRAINT PRIMARY KEY pk('IdPlaylist', 'IdBrano'),
111     FOREIGN KEY('IdPlaylist') REFERENCES Playlist('IdPlaylist') ON DELETE CASCADE ON UPDATE
112     CASCADE,
113     FOREIGN KEY('IdBrano') REFERENCES Brani('IdBrano') ON DELETE CASCADE ON UPDATE CASCADE
114 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
115 -- Table Code
116 CREATE TABLE IF NOT EXISTS 'Code' (
117     'IdUtente' INT(11) NOT NULL,
118     'IdBrano' INT(11) NOT NULL,
119     'Posizione' INT(11) NOT NULL,
120     CONSTRAINT PRIMARY KEY pk('IdUtente', 'IdBrano', 'Posizione'),
121     FOREIGN KEY('IdUtente') REFERENCES Utenti('IdUtente') ON DELETE CASCADE ON UPDATE CASCADE,
122     FOREIGN KEY('IdBrano') REFERENCES Brani('IdBrano') ON DELETE CASCADE ON UPDATE CASCADE
123 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
124 -- Table Ascoltate
125 CREATE TABLE IF NOT EXISTS 'Ascoltate' (
126     'IdUtente' INT(11) NOT NULL,
127     'IdBrano' INT(11) NOT NULL,
128     'Timestamp' TIMESTAMP NOT NULL,
129     CONSTRAINT PRIMARY KEY pk('IdUtente', 'IdBrano', 'Timestamp'),
130     FOREIGN KEY('IdUtente') REFERENCES Utenti('IdUtente') ON DELETE CASCADE ON UPDATE CASCADE,
131     FOREIGN KEY('IdBrano') REFERENCES Brani('IdBrano') ON DELETE CASCADE ON UPDATE CASCADE
132 ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
133 -- INSERT POPULATION
134 -- Insert into Utente
135 INSERT INTO Utenti('Nome', 'Cognome', 'Email', 'Amministratore', 'Username', 'Password')
136 VALUES('Andrea', 'Baldan', 'a.g.baldan@gmail.com', 0, 'codep', MD5('ciao')),
137         ('Federico', 'Angi', 'angiracing@gmail.com', 0, 'keepcalm', MD5('calm')),
138         ('Marco', 'Rossi', 'rossi@gmail.com', 0, 'rossi', MD5('marco')),
139         ('Luca', 'Verdi', 'verdi@yahoo.it', 0, 'verdi', MD5('luca')),
140         ('Alessia', 'Neri', 'neri@gmail.com', 0, 'neri', MD5('alessia'));
141 -- Insert into Subscription
142 INSERT INTO Iscrizioni('IdUtente', 'Tipo') VALUES(1, 'Free'), (2, 'Free');

```

```

141 -- 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
144 Great Compositor Hans Zimmer', '2010', 0, NULL),
145 ('The Good, the Bad and the Ugly: Original Motion Picture Soundtrack', 'Ennio
146 Morricone', 'Homonym movie soundtrack, created by the Legendary composer The
147 Master Ennio Morricone', '1966', 0, NULL),
148 ('Hollywood in Vienna 2014', 'Randy Newman - David Newman', 'Annual cinematographic
149 review hosted in Vienna', '2014', 1, 'Vienna'),
150 ('The Fragile', 'Nine Inch Nails', 'The Fragile is the third album and a double
151 album by American industrial rock band Nine Inch Nails, released on September
152 21, 1999, by Interscope Records.', '1999', 0, NULL),
153 ('American IV: The Man Comes Around', 'Johnny Cash', 'American IV: The Man Comes
154 Around is the fourth album in the American series by Johnny Cash(and his 87th
155 overall), released in 2002. The majority of songs are covers which Cash performs
156 in his own spare style, with help from producer Rick Rubin.', '2002', 0, NULL),
157 ('Greatest Hits', 'Neil Young', 'Rock & Folk Rock greatest success songs by Neil
158 Young', '2004', 0, NULL);
159
160 -- Insert into Brani
161 INSERT INTO Brani('IdAlbum', 'Titolo', 'Genere', 'Durata', 'IdImm')
162 VALUES(1, 'Mind Heist', 'Orchestra', 203, 1),
163 (1, 'Dream is collapsing', 'Orchestra', 281, 1),
164 (1, 'Time', 'Orchestra', 215, 1),
165 (1, 'Half Remembered Dream', 'Orchestra', 71, 1),
166 (1, 'We Built Our Own World', 'Orchestra', 115, 1),
167 (1, 'Radical Notion', 'Orchestra', 222, 1),
168 (1, 'Paradox', 'Orchestra', 205, 1),
169 (2, 'Il Tramonto', 'Orchestra', 72, 2),
170 (2, 'L'estasi dell'oro', 'Orchestra', 202, 2),
171 (2, 'Morte di un soldato', 'Orchestra', 185, 2),
172 (2, 'Il Triello', 'Orchestra', 434, 2),
173 (3, 'The Simpsons', 'Orchestra', 172, 3),
174 (3, 'The war of the Roses', 'Orchestra', 272, 3),
175 (4, 'Somewhat Damaged', 'Industrial Metal', 271, 4),
176 (4, 'The Day The Whole World Went Away', 'Industrial Metal', 273, 4),
177 (4, 'We're In This Together', 'Industrial Metal', 436, 4),
178 (4, 'Just Like You Imagined', 'Industrial Metal', 229, 4),
179 (4, 'The Great Below', 'Industrial Metal', 317, 4),
180 (5, 'Hurt', 'Country', 218, 5),
181 (5, 'Danny Boy', 'Country', 199, 5),
182 (6, 'Old Man', 'Rock', 203, 6),
183 (6, 'Southern Man', 'Rock', 331, 6);
184
185 -- Insert into Copertine
186 INSERT INTO Copertine('IdImm', 'IdAlbum', 'Path')
187 VALUES(1, 1, 'img/covers/inception.png'),
188 (2, 2, 'img/covers/morricone.jpg'),
189 (3, 3, 'img/covers/hivlogo.jpg'),
190 (4, 4, 'img/covers/fragile.jpg'),
191 (5, 5, 'img/covers/nocover.jpg'),
192 (6, 6, 'img/covers/nocover.jpg');
193
194 -- Insert into Collezioni
195 INSERT INTO Collezioni('IdUtente') VALUES(1), (2);
196
197 -- Insert into BraniCollezione
198 INSERT INTO BraniCollezione('IdBrano', 'IdCollezione') VALUES(1, 1), (2, 1), (3, 1), (2, 2);
199
200 -- Insert into Playlist
201 INSERT INTO Playlist('IdUtente', 'Nome', 'Privata') VALUES(1, 'Score & Soundtracks', 0), (1, '
202 Southern Rock', 0), (2, 'Colonne sonore western', 0);
203
204 -- Insert into BraniPlaylist
205 INSERT INTO BraniPlaylist('IdPlaylist', 'IdBrano', 'Posizione') VALUES(1, 1, 1), (1, 2, 2), (1,
206 3, 3), (1, 4, 4), (1, 5, 5), (2, 21, 1), (2, 22, 2), (3, 5, 1), (3, 7, 2), (3, 4, 3);
207
208 -- Insert into Code
209 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', 'IdBranco', '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.

```

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

```

4.2 Funzioni e Procedure

Alcune funzioni e procedure implementate.

```
1 DROP FUNCTION IF EXISTS albumTotalDuration;
2 DROP FUNCTION IF EXISTS eligibleForPrize;
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 $$
17
18 CREATE FUNCTION eligibleForPrize(IdUser INT, Genre VARCHAR(50))
19 RETURNS BOOLEAN
20 BEGIN
21 DECLARE Seconds INT UNSIGNED DEFAULT 0;
22 DECLARE Eligibility BOOLEAN DEFAULT FALSE;
23 SELECT SUM(b.Durata) INTO Seconds
24 FROM Ascoltate a INNER JOIN Utenti u ON(a.IdUtente = u.IdUtente)
25             INNER JOIN Brani b ON(a.IdBranco = b.IdBranco)
26 WHERE b.Genere = 'Orchestra' AND a.IdUtente = IdUser;
27 IF(Seconds >= 1000) THEN
28     SET Eligibility = TRUE;
29 END IF;
30 RETURN Eligibility;
31 END $$
32
33 DELIMITER ;
34
35 DROP PROCEDURE IF EXISTS RAISE_ERROR;
36 DROP PROCEDURE IF EXISTS GENRE_DISTRIBUTION;
37 DROP PROCEDURE IF EXISTS SWAP_POSITION;
38
39 DELIMITER $$
40
41 CREATE PROCEDURE RAISE_ERROR (IN ERROR VARCHAR(256))
42 BEGIN
43 DECLARE V_ERROR VARCHAR(256);
44 SET V_ERROR := CONCAT('[ERROR: ', ERROR, ']');
45 INSERT INTO Errors VALUES(V_ERROR);
46 END $$
47
48 DELIMITER ;
49
50 DELIMITER $$
51
52 CREATE PROCEDURE GENRE_DISTRIBUTION()
53 BEGIN
54 DECLARE Total INT DEFAULT 0;
55 DROP TEMPORARY TABLE IF EXISTS 'Distribution';
56 CREATE TEMPORARY TABLE 'Distribution' (
57     'Genere' VARCHAR(100),
58     'Percentuale' VARCHAR(6)
```

```

59 ) ENGINE=InnoDB;
60 SELECT count(b.Genere) INTO Total FROM Brani b;
61 INSERT INTO Distribution (Genere, Percentuale)
62 SELECT Genere, CONCAT(FLOOR((count(Genere) / Total) * 100), "%")
63 FROM Brani GROUP BY Genere;
64 END $$
65
66 DELIMITER ;
67
68 DELIMITER $$
69
70 CREATE PROCEDURE SWAP_POSITION(IN a INT, IN b INT, IN id INT, IN tab INT)
71 BEGIN
72 DECLARE AUX INT DEFAULT -1;
73 CASE tab
74     WHEN 1 THEN
75         UPDATE Code SET Posizione = AUX WHERE Posizione = a AND IdUtente = id;
76         UPDATE Code SET Posizione = a WHERE Posizione = b AND IdUtente = id;
77         UPDATE Code SET Posizione = b WHERE Posizione = AUX AND IdUtente = id;
78     ELSE
79         UPDATE BraniPlaylist SET Posizione = AUX WHERE Posizione = a AND IdPlaylist = id;
80         UPDATE BraniPlaylist SET Posizione = a WHERE Posizione = b AND IdPlaylist = id;
81         UPDATE BraniPlaylist SET Posizione = b WHERE Posizione = AUX AND IdPlaylist = id;
82 END CASE;
83 END $$
84
85 DELIMITER ;

```

5 Query

Alcune query significative.

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

```

1 SELECT b.Titolo, a.Titolo as TitoloAlbum, u.Username, h.Timestamp
2 FROM Brani b INNER JOIN Album a ON(b.IdAlbum = a.IdAlbum)
3     INNER JOIN Ascoltate h ON(h.IdBrano = b.IdBrano)
4     INNER JOIN Seguaci f ON(f.IdSeguace = h.IdUtente)
5     INNER JOIN Utenti u ON(u.IdUtente = f.IdSeguace)
6 WHERE h.Timestamp BETWEEN ADDDATE(CURDATE(), -7) AND CURDATE()
7     AND u.IdUtente IN (SELECT u.IdUtente FROM Utenti u INNER JOIN Seguaci f ON(f.IdSeguace
8     = u.IdUtente) WHERE f.IdUtente = 1)
9     ORDER BY h.Timestamp DESC LIMIT 10;

```

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

```

1 SELECT COUNT(b.IdBrano) AS Conto, u.Username
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 b.Titolo = 'Paradox' AND f.IdUtente = 1 GROUP BY u.Username ORDER BY Conto DESC;

```

3. Username, titolo e conto delle canzoni più 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;

```

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);

```

5. Username e secondi 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, SUM(b.Durata) 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;

```

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 Ascolte 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 ADDDATE(CURDATE(), -30) AND NOW()
7 GROUP BY a.IdUtente;
8 SELECT ca.*
9 FROM CanzoniUtente ca
10 WHERE ca.Conto > (SELECT AVG(ce.Conto) FROM CanzoniAscoltate ce) ORDER BY ca.Conto DESC;

```

7. Trova gli utenti e il numero di brani di genere 'Country' nella sua 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, MAX(Conteggio)
8 FROM Conteggi
9 WHERE Genere = 'Country';
10 DROP VIEW IF EXISTS Conteggi;

```

Output:

Username	Genere	max(Conteggio)
codep	Country	2

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 operazioni 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, accessibili 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 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 dà 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.