Università di Catania

Università degli Studi di Catania
Dipartimento di Matematica e Informatica

Progetto di Basi di Dati:
Catalogo Musicale Online

Riccardo Rubino
1000014805

Indice:

1. Descrizione della base di dati

Si vuole progettare una base di dati per la gestione di un catalogo musicale online.
Il catalogo offre agli utenti lo streaming di canzoni e di album realizzate da artisti (solisti o gruppi). Le canzoni hanno un titolo, una durata in secondi, una categorizzazione per genere e un numero di ascolti.
I pezzi possono essere singoli oppure appartenere ad un album.
Più artisti possono collaborare all'interno di un pezzo.
Ogni canzone è realizzata da un produttore musicale (inteso come un musicista o gruppo capace di creare una strumentale), da un cantante (può essere assente) e da uno scrittore (può essere assente) che possono anche coincidere. Gli album sono composti da un numero variabile di canzoni, hanno un titolo e un artista che li realizza. Inoltre gli album sono distribuiti da un'azienda di distribuzione (major). Gli artisti sono identificati da un nome/pseudonimo e possiedono una biografia. Le major sono descritte da un identificativo e dalla propria sede.
Gli utenti registrati devono scegliere un username oltre ad essere identificati dal loro nome e cognome e possono creare playlist e recensire le canzoni dando una valutazione che va da 1 a 10.
Le playlist hanno inoltre un indice di gradimento che indica la popolarità della stessa.
Si preveda che dalla piattaforma si possano cancellare solo gli utenti, le playlist, le recensioni e le canzoni dalle playlist.
Si vogliano anche implementare le operazioni quali l'aggiornamento degli ascolti di una canzone, la pubblicazione di un brano e/o album, la creazione di una nuova playlist, l'inserimento di una canzone in una playlist, l'inserimento di un nuovo artista, l'inserimento di una nuova recensione, il calcolo del numero totale di ascolti di un artista o major. Implementare le operazioni per stilare una classifica dei 100 brani più ascoltati della settimana, dei 50 album più ascoltati del mese e dei 10 artisti più ascoltati dell'anno oltre alle alla visualizzazione dei 10 brani preferiti e dei 5 artisti preferiti di un utente.

2.a Glossario dei termini

TERMINE	DESCRIZIONE	SINONIMI	LEGAME
Canzone	prodotto musicale realizzato da uno o più artisti	traccia, pezzo, brano	Album, Artista, Playlist
Album	raccolta di canzoni realizzata da un artista e distribuito da una major		Canzone, Artista, Major
Artista	individuo creativo che partecipa alla realizzazione di prodotti come canzoni o album	solista, gruppo, cantante, produttore, scrittore	Canzone, Album
Major	azienda che si occupa del distribuire sulla piattaforma gli album degli artisti e del creare nuove playlist	distribuzione	Album, Playlist
Playlist	raccolta di canzoni creata da un utente con indice di gradimento		Canzone, Utente
Utente	individuo registrato alla piattaforma che può creare playlist e recensire canzoni		Canzone, Playlist

2.b Specifiche dei Dati

Dati Generali
Si vuole progettare una base di dati per la gestione di un catalogo musicale online. Il catalogo offre agli utenti lo streaming di canzoni e di album realizzate da artisti.

Dati sulle Canzoni
Le canzoni hanno un titolo, una durata in secondi, una categorizzazione per genere e un numero di ascolti. Inoltre possono essere singoli oppure appartenere ad un album. Ogni canzone è realizzata da un produttore musicale, da un cantante (può essere assente) e da uno scrittore (può essere assente) che possono anche coincidere, oltre al fatto che più artisti possono collaborare all'interno di una canzone.

Dati sugli Album
Gli album sono composti da un numero variabile di canzoni, hanno un titolo e un artista che li realizza. Inoltre gli album sono distribuiti da una major.

Dati sugli Artisti
Gli artisti possono essere solisti o gruppi, sono identificati da un nome/pseudonimo e possiedono una biografia. Inoltre possono collaborare all'interno di canzoni di altri artisti.

Dati sulle Major
Le major distribuiscono gli album, sono descritte da un identificativo e dalla propria sede.

Dati sulle Playlist
Le playlist sono create dagli utenti in base ai propri gusti. Ogni playlist si compone di pezzi e da un valore che indica l'indice di gradimento.

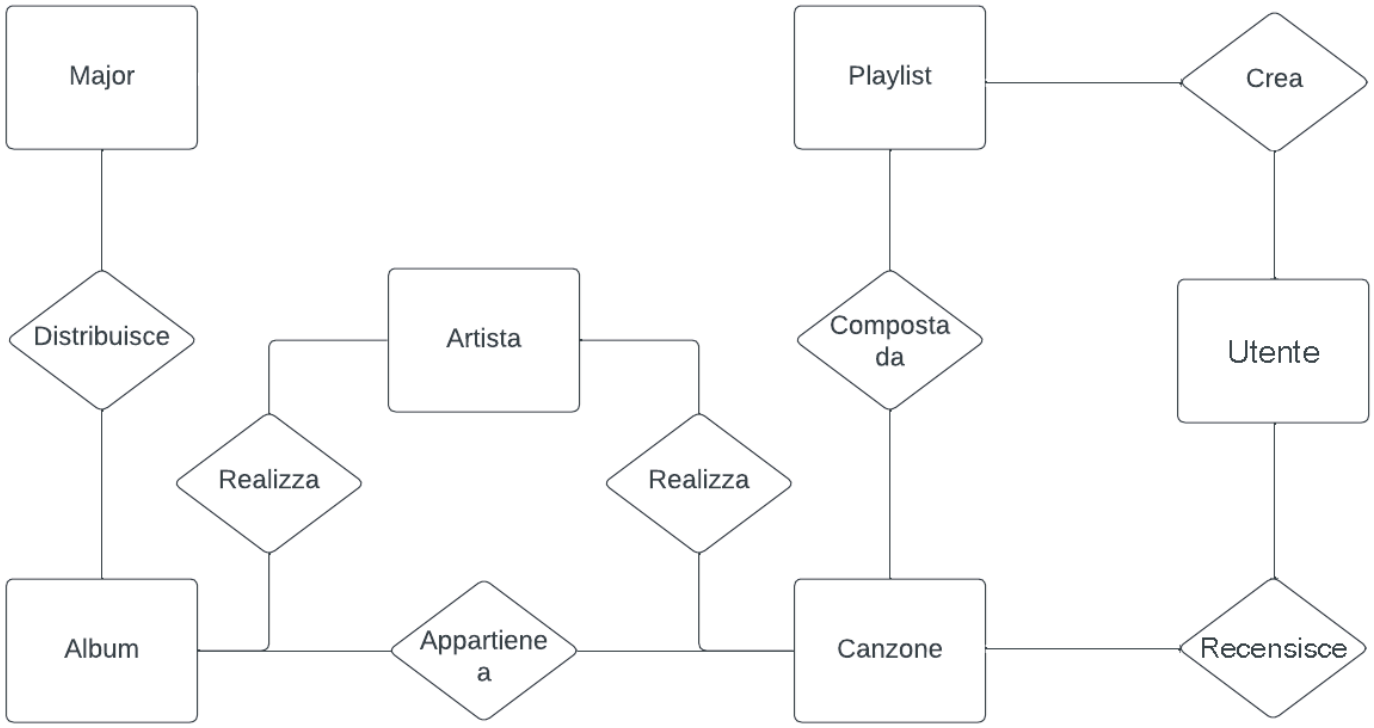
Dati sugli Utenti
Gli utenti alla registrazione devono scegliere un username oltre ad essere identificati dal loro nome e cognome.

2.c Specifiche delle Operazioni

O1. Aggiornamento degli ascolti di una canzone (50 volte al giorno)
O2. Aggiornamento del gradimento di una playlist (1 volta al giorno)
O3. Inserimento di una nuova canzone (100 volte al giorno)
O4. Inserimento di un nuovo album (2 volte a settimana)
O5. Inserimento di una nuova playlist (5 volte a settimana)
O6. Inserimento di una canzone in una playlist (10 volte al giorno)
O7. Inserimento di un nuovo artista (50 volte al giorno)
O8. Inserimento di una nuova recensione (10 volte al giorno)
O9. Calcolo ascolti totali di un artista (10 volte a settimana)
O10. Calcolo ascolti totali di una major (2 volte al mese)
O11. Creazione classifica 100 brani più ascoltati (1 volta a settimana)
O12. Creazione classifica 50 album più ascoltati (1 volta al mese)
O13. Creazione classifica 10 artisti più ascoltati (1 volta all'anno)
O14. Visualizzazione 10 brani preferiti da un utente (10 volte al mese)
O15. Visualizzazione 5 artisti preferiti da un utente (10 volte all'anno)

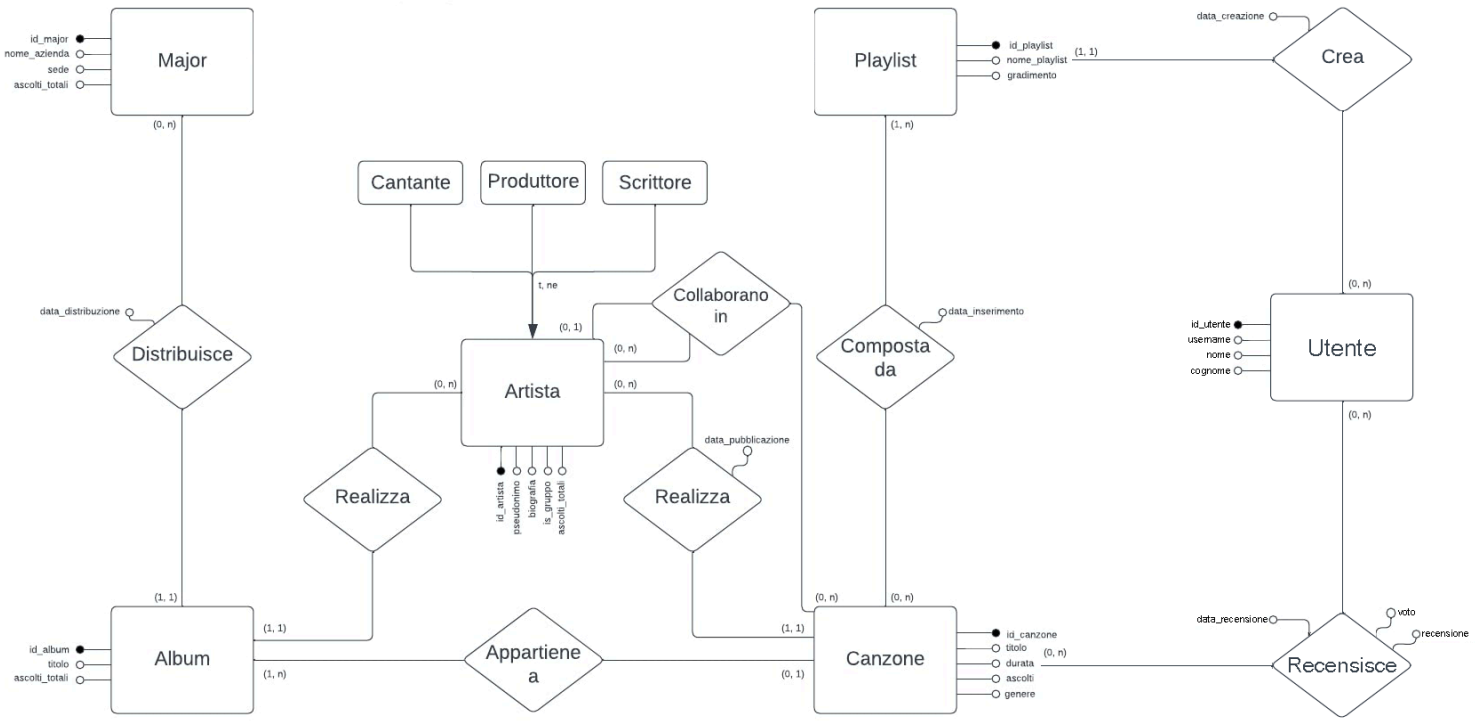
3.a Schema Scheletro

Basandosi sulle informazioni specificate nella descrizione per la realizzazione della base di dati e rielaborate nell'analisi dei requisiti è stato prodotto il seguente schema scheletro:



3.b Schema Concettuale

Lo schema scheletro è stato esteso con gli attributi delle entità e delle relazioni, oltre ad aver introdotto gli attributi ridondanti "*ascolti_totali*" per le entità **Artista**, **Album** e **Major** che verranno ulteriormente approfonditi nella progettazione logica. È stata inoltre sviluppata una gerarchia sull'entità **Artista** oltre al nuovo attributo *is_gruppo* per distinguere le istanze di tipo gruppo da quelle di tipo solista.
Infine è stata introdotta la nuova relazione **Collaborano In** per permettere di memorizzare le collaborazioni tra più artisti all'interno delle canzoni.



3.c Dizionario dati delle Entità

ENTITÀ	DESCRIZIONE	ATTRIBUTI	IDENTIFICATORE
Canzone	prodotto musicale realizzato da uno o più artisti	id_canzone, titolo, durata, ascolti, genere	id_canzone
Album	raccolta di canzoni realizzata da un artista e distribuito da una major	id_album, titolo, ascolti_totali	id_album
Artista	individuo creativo che partecipa alla realizzazione di prodotti come canzoni o album	id_artista, pseudonimo, biografia, is_gruppo, ascolti_totali	id_artista
Major	azienda che si occupa del distribuire sulla piattaforma gli album degli artisti	id_major, nome_azienda, sede, ascolti_totali	id_major
Playlist	raccolta di canzoni scelte da un utente	id_playlist, nome_playlist, gradimento	id_playlist
Utente	individio registrato alla piattaforma che crea playlist e recensisce canzoni	id_utente, username, nome, cognome	id_utente

L'attributo ridondante *ascolti_totali* verrà analizzato per ogni entità e ogni operazione in cui è coinvolto nella sezione inerente alla progettazione logica.

3.d Dizionario dati delle Relazioni

RELAZIONE	ENTITÀ PARTECIPANTI	DESCRIZIONE	ATTRIBUTI
Realizza (Canzone)	Artista, Canzone	Un artista realizza una canzone	data_pubblicazione
Collaborano in	Artista, Artista, Canzone	Un artista collabora nella canzone di un altro artista	
Realizza (Album)	Artista, Album	Un artista realizza un album	
Appartiene a	Canzone, Album	Una canzone appartiene ad un album	
Distribuisce	Major, Album	Una major distribuisce un album	data_distribuzione
Crea	Utente, Playlist	Un utente crea una playlist	data_creazione
Composta da	Playlist, Canzone	Una playlist è composta da una o più canzoni	data_inserimento
Recensisce	Utente, Canzone	Un utente recensisce una canzone	data_recensione, recensione, voto

3.e Vincoli d'Integrità

- Ogni canzone ha un produttore musicale.
- Una canzone pubblicata all'interno di un album deve avere la stessa data di distribuzione dell'album.
- Un artista deve essere un cantante, un produttore o uno scrittore.
- Un artista non può rilasciare due canzoni con lo stesso titolo.
- Una canzone inserita all'interno di un album deve avere lo stesso artista dell'album.
- Il voto di una recensione va da 1 a 10.

4.a Tabella dei Volumi

CONCETTO	TIPO	VOLUME
Canzone	*E*	150000
Album	*E*	10000
Artista	*E*	800
Major	*E*	200
Playlist	*E*	20000
Utente	*E*	100000
Realizza (Canzone)	*R*	150000
Realizza (Album)	*R*	10000
Distribuisce	*R*	10000
Crea	*R*	200000
Collaborano in	*R*	300000
Appartiene a	*R*	110000
Composta da	*R*	2000000
Recensisce	*R*	75000

4.b Tabella delle frequenze

OPERAZIONE	DESCRIZIONE	FREQUENZA	TIPO
O1	Aggiornamento degli ascolti di una canzone	50 volte al giorno	*B*
O2	Aggiornamento del gradimento di una playlist	1 volta al giorno	*B*
O3	Inserimento di una nuova canzone	100 volte al giorno	*I*
O4	Inserimento di un nuovo album	2 volte a settimana	*I*
O5	Inserimento di una nuova playlist	5 volte a settimana	*I*
O6	Inserimento di una nuova canzone in una playlist	10 volte al giorno	*I*
O7	Inserimento di un nuovo artista	50 volte al giorno	*I*
O8	Inserimento di una nuova recensione	10 volte al giorno	*I*
O9	Calcolo ascolti totali di un artista	10 volte a settimana	*B*
O10	Calcolo ascolti totali di una major	2 volte a settimana	*B*
O11	Creazione classifica 100 brani più ascoltati	1 volta a settimana	*B*
O12	Creazione classifica 50 album più ascoltati	1 volta al mese	*B*
O13	Creazione classifica 10 artisti più ascoltati	1 volta all’anno	*B*
O14	Visualizzazione 10 brani preferiti da un utente	10 volte al mese	*B*
O15	Visualizzazione 5 artisti preferiti da un utente	10 volte all’anno	*B*

4.c Analisi delle ridondanze

L'attributo *ascolti_totali* nelle entità **Album**, **Artista** e **Major** è ridondante è ridondante in quanto può essere calcolato sommando gli ascolti delle istanze dell'entità **Canzone** in relazione con l'entità per il quale si vuole ottenere il numero totale di ascolti.

Nell'entità **Artista**, l'attributo è coinvolto nelle operazioni **O1**, **O9** e **O13**.
O13 in assenza di ridondanza verrà eseguita sommando gli ascolti di ogni canzone di ogni artista e selezionando poi i primi 10 artisti per ascolti.
Tenendo a mente che un accesso alla memoria in scrittura vale quanto due accessi in memoria in lettura (**1S = 2L**), la tabella degli accessi su base annuale è la seguente:

CON RIDONDANZA	**SENZA RIDONDANZA**
O1: 1S in Canzone 1L in Canzone 1S in Artista 5 accessi * 50 volte al giorno = 250 accessi al giorno	**O1:** 1S in Canzone 2 accessi * 50 volte al giorno = 100 accessi al giorno
O9: 1L in Artista 10 accessi a settimana	**O9:** 150000 / 800 ≃ 188L in Artista 188 accessi * 10 volte a settimana = 1880 accessi a settimana
O13: 10L in Artista 10 accessi all'anno	**O13:** 150000L in Canzone 150000 accessi all'anno
TOT: 91250 + 520 + 10 = **91780 accessi all'anno**	**TOT:** 36500 + 97760 + 150400 = **284660 accessi all'anno**

Ne segue che conviene mantenere l'attributo ridondante *ascolti_totali* nell'entità **Artista**.

Nell'entità **Album** l'attributo *ascolti_totali* è coinvolto nelle operazioni **O1**, **O4** (quando un singolo già pubblicato viene inserito in un album), **O10** (supponendo di non mantenere l'attributo *ascolti_totali* nell'entità **Major**) e **O12**. Quest'ultima, in assenza di ridondanza verrà eseguita sommando gli ascolti di ogni canzone di ogni album e selezionando poi i primi 50 album per ascolti. Supponendo che in media ogni album al rilascio contiene 2 singoli già pubblicati, la tabella degli accessi su base mensile è la seguente:

CON RIDONDANZA	**SENZA RIDONDANZA**
O1: 1S in Canzone 1L in Canzone 1S in Album 5 accessi * 50 volte al giorno = 250 accessi al giorno	**O1:** 1S in Canzone 2 accessi * 50 volte al giorno = 100 accessi al giorno
O4: 1S in Album 2L in Canzone 1S in Album 6 accessi * 2 volte a settimana = 12 accessi a settimana	**O4:** 1S in Album 4 accessi a settimana
O10: 10000 / 200 = 50L in Album 50 accessi * 2 volte a settimana = 100 accessi a settimana	**O10:** 10000 / 200 = 50L in Album *ogni album ha in media 15 canzoni* 50 * 15 = 750L in Canzone 800 accessi * 2 volte a settimana = 1600 accessi a settimana
O12: 50L in Album 50 accessi al mese	**O12:** 110000L in Canzone *(110000 è il volume dell'entità* ***Appartiene a****)* 110000 accessi al mese
TOT: 7500 + 48 + 400 + 50 = **7998 accessi al mese**	**TOT:** 3000 + 16 + 6400 + 110000 = **119416 accessi al mese**

Ne segue che conviene mantenere l'attributo ridondante *ascolti_totali* nell'entità **Album**.

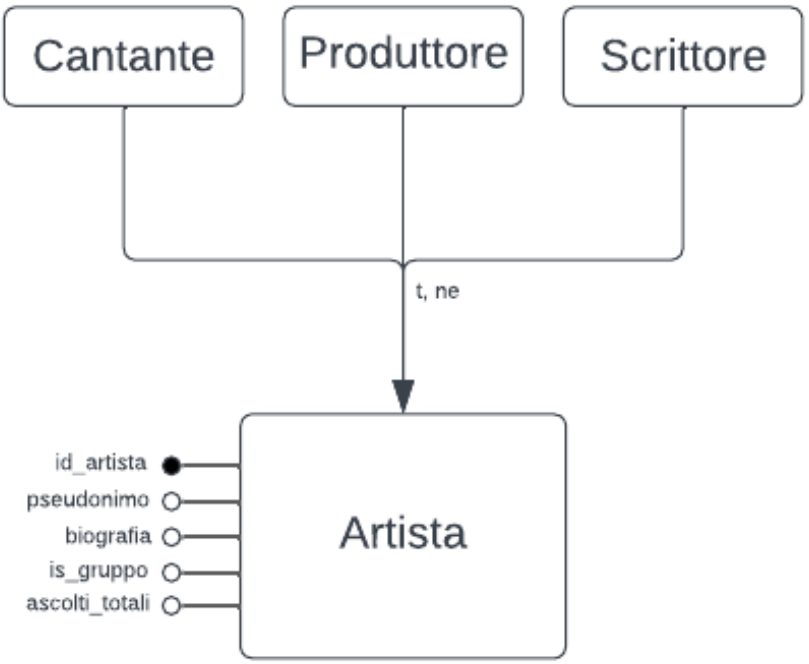
Nell'entità **Major** l'attributo *ascolti_totali* è coinvolto nelle operazioni **O1**, **O4** (quando un singolo già pubblicato viene inserito in un album) e **O10** (tenendo in considerazione di aver già mantenuto l'attributo *ascolti_totali* nell'entità **Album**). Ricordando che ogni album al rilascio contiene in media 2 singoli già pubblicati, la tabella degli accessi su base settimanale è la seguente:

CON RIDONDANZA	**SENZA RIDONDANZA**
O1: 1S in Canzone 1L in Canzone 1S in Major 5 accessi * 50 volte al giorno = 250 accessi al giorno	**O1:** 1S in Canzone 2 accessi * 50 volte al giorno = 100 accessi al giorno
O4: 1S in Album 2L in Canzone 1S in Album 1S in Major 8 accessi * 2 volte a settimana = 16 accessi a settimana	**O4:** 1S in Album 2L in Canzone 1S in Album 6 accessi * 2 volte a settimana = 12 accessi a settimana
O10: 1L in Major 2 accessi a settimana	**O10:** 10000 / 200 = 50L in Album 50 accessi * 2 volte a settimana = 100 accessi a settimana
TOT: 1750 + 16 + 2 = **1768 accessi a settimana**	**TOT:** 700 + 12 + 100 = **812 accessi a settimana**

Ne segue che conviene rimuovere l'attributo ridondante *ascolti_totali* dall'entità **Major**.

4.d Eliminazione delle Gerarchie

L’unica gerarchia presente nello schema ER è la seguente:



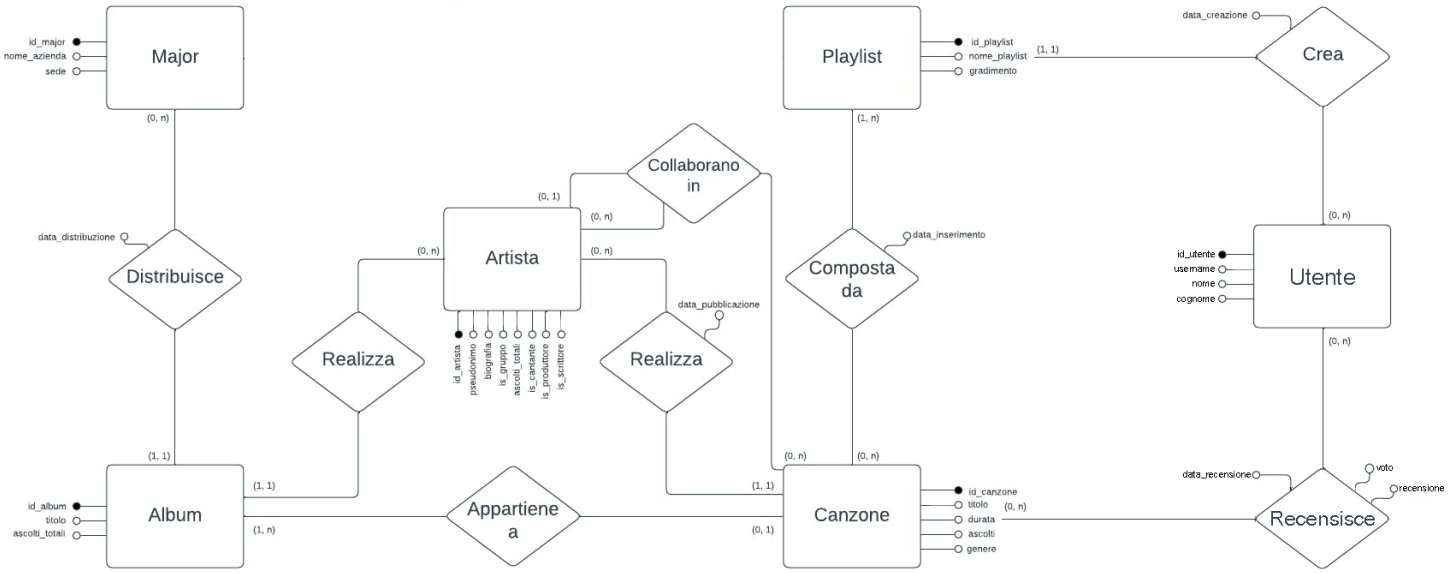
Essendo la gerarchia di tipo **totale** e **non esclusiva** si preferisce adottare la strategia del collasso verso l’alto e, dato che ogni entità figlia di Artista non ha attributi, si decide di introdurre tre nuovi attributi nell’entità **Artista**:

- *is_cantante*
- *is_produttore*
- *is_scrittore*

Non verrà introdotto nessun attributo *selettore* dato che la gerarchia è di tipo **non esclusiva**.

4.e Schema ristrutturato

A seguito dell'analisi delle ridondanze e dell'eliminazione delle gerarchie, la ristrutturazione dello schema concettuale ha prodotto lo schema ER:



4.f Traduzione verso lo Schema Relazionale

Le associazioni **Appartiene a**, **Crea**, **Distribuisce** e **Realizza (Album)** essendo del tipo 1 a molti verranno eliminate incorporando gli attributi nelle rispettive entità che partecipano con cardinalità (1, 1), essi ovviamente potranno essere NULL se le entità partecipano con cardinalità (0, 1).
L'associazione **Realizza (canzone)**, a causa del vincolo d'integrità che impone la presenza di un produttore musicale nella realizzazione di una canzone, nasconde le associazioni *Canta*, *Produce* e *Scrive* che per semplicità verranno eliminate introducendo gli attributi *cantante*, *produttore* e *scrittore*, dei quali solo *produttore* **NON** può essere NULL, oltre all'attributo *artista* dato che possono esistere canzoni appartenenti ad album dove l'artista principale è il cantante oppure canzoni dove non vi sono cantanti o scrittori.
Le associazioni **Collaborano in**, **Composta da**, e **Recensisce** essendo del tipo molti a molti verranno tradotte in relazioni e ridenominate rispettivamente in **Featuring**, **CanzoneInPlaylist** e **Recensione**.

4.g Schema Relazionale

Canzone (**id_canzone**, titolo, durata, ascolti, genere, artista, cantante, produttore, scrittore, album, data_pubblicazione)

Album (**id_album**, titolo, ascolti_totali, artista, major_distribuzione, data_distribuzione)

Artista (**id_artista**, pseudonimo, biografia, ascolti_totali, is_gruppo, is_cantante, is_produttore, is_scrittore)

Major (**id_major**, nome_azienda, sede)

Playlist (**id_playlist**, nome_playlist, gradimento, creatore, data_creazione)

Utente (**id_utente**, username, nome, cognome)

Featuring (**artista_featuring, canzone**)

CanzoneInPlaylist (**canzone, playlist**, data_inserimento)

Recensione (**utente, canzone**, recensione, voto, data_recensione)

legenda: ***Relazione***, **chiave primaria**, chiave esterna

4.h Normalizzazione

Di seguito gli insieme di dipendenze funzionali dei vari schemi:

$F_{Canzone}$ (id_canzone → titolo, durata, ascolti, genere, artista, cantante, produttore, scrittore, album, data_pubblicazione)

essendo *id_canzone* chiave dello schema **Canzone** esso è in BCNF;

F_{Album} (id_album → titolo, ascolti_totali, artista, major_distribuzione, data_distribuzione)

essendo *id_album* chiave dello schema **Album** esso sarebbe in BCNF in assenza dell'attributo *ascolti_totali* dato che esso può essere calcolato dalla relazione **Canzone** ma come confutato nella sezione dell'analisi delle ridondanze si preferisce mantenere l'attributo ridondante;

$F_{Artista}$ (id_artista → pseudonimo, biografia, ascolti_totali, is_gruppo, is_cantante, is_produttore, is_scrittore)

essendo *id_artista* chiave dello schema **Artista** esso sarebbe in BCNF in assenza dell'attributo *ascolti_totali* dato che esso può essere calcolato dalla relazione **Canzone** ma come confutato nella sezione dell'analisi delle ridondanze si preferisce mantenere l'attributo ridondante;

F_{Major} (id_major → nome_azienda, sede)

essendo *id_major* chiave dello schema **Major** esso è in BCNF;

$F_{Playlist}$ (id_playlist → nome_playlist, gradimento, creatore, data_creazione)

essendo *id_playlist* chiave dello schema **Playlist** esso è in BCNF;

F_{Utente} (id_utente → username, nome, cognome)

essendo *id_utente* chiave dello schema **Utente** esso è in BCNF;

$F_{Featuring}$ (Ø)

lo schema non ha dipendenze funzionali per cui esso è già in BCNF;

$F_{CanzoneInPlaylist}$ (canzone, playlist → data_inserimento)

essendo *canzone* e *playlist* chiavi dello schema **CanzoneInPlaylist** esso è in BCNF;

$F_{Recensione}$ (utente, canzone → recensione, voto, data_recensione)

essendo *utente* e *canzone* chiavi dello schema **Recensione** esso è in BCNF;

In conclusione lo schema relazionale della base di dati potrebbe essere scomposto in BCNF rimuovendo gli attributi ridondanti dalle relazioni **Album** e **Artista**, ma si preferisce mantenerlo in forma non-normale per quanto visto nella sezione riguardante l'analisi delle ridondanze.

5.a Implementazione tabelle in SQL

Per l'implementazione fisica verrà utilizzato il DBMS **MySQL**.
Ogni tabella verrà implementata utilizzando lo storage engine **InnoDB**.
Per comodità verranno implementate prima le relazioni senza chiavi esterne per evitare di dover eseguire comandi del tipo **ALTER TABLE**.

Artista (**id_artista**, pseudonimo, biografia, ascolti_totali, is_gruppo, is_cantante, is_produttore, is_scrittore):

```
CREATE TABLE Artista (id_artista INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
                      pseudonimo VARCHAR(128) NOT NULL,
                      biografia VARCHAR(1024) NOT NULL,
                      ascolti_totali INT UNSIGNED NOT NULL DEFAULT 0,
                      is_gruppo BOOL NOT NULL,
                      is_cantante BOOL NOT NULL,
                      is_produttore BOOL NOT NULL,
                      is_scrittore BOOL NOT NULL,
                      UNIQUE (pseudonimo),
                      CONSTRAINT chk_artista_valido CHECK ((is_cantante = FALSE
                      AND is_produttore = FALSE AND is_scrittore = FALSE) = FALSE));
```

Utente (**id_utente**, username, nome, cognome)

```
CREATE TABLE Utente (id_utente INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
                     username VARCHAR(64) NOT NULL,
                     nome VARCHAR(32) NOT NULL,
                     cognome VARCHAR(32) NOT NULL);
```

Major (**id_major**, nome_azienda, sede):

```
CREATE TABLE Major (id_major INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
                    nome_azienda VARCHAR(128) NOT NULL,
                    sede VARCHAR(128) NOT NULL);
```

Playlist (**id_playlist**, nome_playlist, gradimento, creatore, data_creazione):

```
CREATE TABLE Playlist (id_playlist INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
                       nome_playlist VARCHAR(128) NOT NULL,
                       gradimento FLOAT NOT NULL DEFAULT 0,
                       creatore INT NOT NULL,
                       data_creazione TIMESTAMP NOT NULL DEFAULT
                       CURRENT_TIMESTAMP,
                       FOREIGN KEY (creatore) REFERENCES Utente(id_utente));
```

Album (**id_album**, titolo, ascolti_totali, artista, major_distribuzione, data_distribuzione):

```
CREATE TABLE Album (id_album INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
                    titolo VARCHAR(128) NOT NULL,
                    ascolti_totali INT UNSIGNED NOT NULL DEFAULT 0,
                    artista INT NOT NULL,
                    major_distribuzione INT NOT NULL,
                    data_distribuzione TIMESTAMP NOT NULL DEFAULT
                    CURRENT_TIMESTAMP,
                    FOREIGN KEY (artista) REFERENCES Artista(id_artista),
                    FOREIGN KEY (major_distribuzione) REFERENCES Major(id_major));
```

Canzone (**id_canzone**, titolo, durata, ascolti, genere, artista, cantante, produttore, scrittore, album, data_pubblicazione):

```
CREATE TABLE Canzone (id_canzone INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
                      titolo VARCHAR(128) NOT NULL,
                      durata INT UNSIGNED NOT NULL,
                      ascolti INT UNSIGNED NOT NULL DEFAULT 0,
                      genere VARCHAR(64) NOT NULL,
                      artista INT NOT NULL,
                      cantante INT,
                      produttore INT NOT NULL,
                      scrittore INT,
                      album INT,
                      data_pubblicazione TIMESTAMP NOT NULL DEFAULT
                      CURRENT_TIMESTAMP,
                      UNIQUE (titolo, artista),
                      FOREIGN KEY (artista) REFERENCES Artista(id_artista),
                      FOREIGN KEY (cantante) REFERENCES Artista(id_artista),
                      FOREIGN KEY (produttore) REFERENCES Artista(id_artista),
                      FOREIGN KEY (scrittore) REFERENCES Artista(id_artista),
                      FOREIGN KEY (album) REFERENCES Album(id_album));
```

Featuring (**artista_featuring, canzone**):

```
CREATE TABLE Featuring (artista_featuring INT NOT NULL,
                        canzone INT NOT NULL,
                        PRIMARY KEY (artista_featuring, canzone),
                        FOREIGN KEY (artista_featuring) REFERENCES Artista(id_artista),
                        FOREIGN KEY (canzone) REFERENCES Canzone(id_canzone));
```

CanzoneInPlaylist (**canzone, playlist**, data_inserimento):

```
CREATE TABLE CanzoneInPlaylist (canzone INT NOT NULL,
                                playlist INT NOT NULL,
                                data_inserimento TIMESTAMP NOT NULL DEFAULT
                                CURRENT_TIMESTAMP,
                                PRIMARY KEY (canzone, playlist),
                                FOREIGN KEY (canzone) REFERENCES Canzone(id_canzone),
                                FOREIGN KEY (playlist) REFERENCES Playlist(id_playlist)
                                ON DELETE CASCADE);
```

Recensione (**utente, canzone**, recensione, voto, data_recensione)

```
CREATE TABLE Recensione (utente INT NOT NULL,
                         canzone INT NOT NULL,
                         recensione VARCHAR(1024) NOT NULL,
                         voto INT UNSIGNED NOT NULL,
                         data_recensione TIMESTAMP NOT NULL DEFAULT
                         CURRENT_TIMESTAMP,
                         PRIMARY KEY (utente, canzone),
                         FOREIGN KEY (utente) REFERENCES Utente(id_utente)
                         ON DELETE CASCADE,
                         FOREIGN KEY (canzone) REFERENCES Canzone(id_canzone),
                         CONSTRAINT chk_voto_valido CHECK (voto >= 1 AND voto <= 10));
```

5.b Implementazione operazioni in SQL

O1: *Aggiornamento degli ascolti di una canzone*:
Per l'implementazione di **O1** basta effettuare query del tipo:

```
UPDATE Canzone SET ascolti = nuovi_ascolti WHERE id_canzone = canzone_da_aggiornare;
```

in combinazione ai trigger chk_nuovi_ascolti e upd_ascolti_totali:

```
CREATE TRIGGER chk_nuovi_ascolti
BEFORE UPDATE ON Canzone
FOR EACH ROW
BEGIN
        IF NEW.ascolti < OLD.ascolti THEN
                SIGNAL SQLSTATE "45000"
                SET MESSAGE_TEXT = "Impossibile aggiornare il numero
                di ascolti con un valore minore.";
        END IF;
END;

CREATE TRIGGER upd_ascolti_totali
AFTER UPDATE ON Canzone
FOR EACH ROW
BEGIN
        IF NEW.ascolti <> OLD.ascolti THEN
                IF NEW.album IS NOT NULL THEN
                        UPDATE Album
                        SET ascolti_totali = ascolti_totali + (NEW.ascolti - OLD.ascolti)
                        WHERE id_album = NEW.album;
                END IF;
                UPDATE Artista
                SET ascolti_totali = ascolti_totali + (NEW.ascolti - OLD.ascolti)
                WHERE id_artista = NEW.artista;
        END IF;
END;
```

O2: *Aggiornamento del gradimento di una playlist*:
Per l'implementazione di **O2** basta effettuare query del tipo:

```
UPDATE Playlist SET gradimento = nuovo_gradimento WHERE id_playlist = playlist_da_aggiornare;
```

in questo caso è necessario implementare alcun trigger, poiché l'attributo gradimento può essere decrementato.

Nell'implementazione delle operazioni del tipo *Inserimento nuovo elemento in tabella* verranno omessi gli attributi per i quali sono stati assegnati dei valori **DEFAULT** o **AUTO_INCREMENT** nella definizione della tabella, questi potranno comunque essere specificati modificando le query.

O3: *Inserimento di una nuova canzone*:
Per l'implementazione di **O3** bisogna effettuare query del tipo:

```
INSERT INTO Canzone (titolo, durata, genere, artista, cantante, produttore, scrittore, album)
                    VALUES (dati nuova canzone…);
```

insieme alla seguente query per eventuali featuring:

```
INSERT INTO Featuring VALUES (artista_featuring_nuova_canzone, id_nuova_canzone);
```

in combinazione ai trigger chk_artisti_canzone, upd_ascolti_totali_on_ins, chk_featuring, chk_artista_album_on_ins e upd_data_canzoni_album:

```
CREATE TRIGGER chk_artisti_canzone
BEFORE INSERT ON Canzone
FOR EACH ROW
BEGIN
        IF (NEW.cantante IS NOT NULL AND
            NOT EXISTS (SELECT * FROM Artista WHERE id_artista = NEW.cantante
                                AND is_cantante = TRUE)) THEN
            SIGNAL SQLSTATE "45000" SET MESSAGE_TEXT = "Cantante inesistente";
        END IF;
        IF (NOT EXISTS (SELECT * FROM Artista WHERE id_artista = NEW.produttore
                                AND is_produttore = TRUE)) THEN
            SIGNAL SQLSTATE "45000" SET MESSAGE_TEXT = "Produttore inesistente";
        END IF;
        IF (NEW.scrittore IS NOT NULL AND
            NOT EXISTS (SELECT * FROM Artista WHERE id_artista = NEW.scrittore
                                AND is_scrittore = TRUE)) THEN
            SIGNAL SQLSTATE "45000" SET MESSAGE_TEXT = "Scrittore inesistente";
        END IF;
END;
```

il trigger upd_ascolti_totali_on_ins è analogo ad upd_ascolti_totali implementato in **O1** ma **AFTER INSERT** (utilizzato nel caso in cui venga inserita una canzone con numero di ascolti diverso da 0).

```
CREATE TRIGGER chk_featuring
BEFORE INSERT ON Featuring
FOR EACH ROW
BEGIN
        DECLARE a, c, p, s INT;
        SELECT artista, cantante, produttore, scrittore INTO a, c, p, s FROM Canzone
        WHERE id_canzone = NEW.canzone;
        IF (NEW.artista_featuring IN (a, c, p, s)) THEN
           SIGNAL SQLSTATE "45000"
           SET MESSAGE_TEXT = "Artista featuring già presente nella canzone";
        END IF;
        IF (NOT EXISTS (SELECT * FROM Artista WHERE id_artista = NEW.artista_featuring
                        AND (is_cantante = FALSE AND is_produttore = FALSE) = FALSE)) THEN
           SIGNAL SQLSTATE "45000"
           SET MESSAGE_TEXT = "Artista featuring non valido";
        END IF;
END;

CREATE TRIGGER chk_artista_album_on_ins
BEFORE INSERT ON Canzone
FOR EACH ROW
BEGIN
        DECLARE artista_album INT;
        IF NEW.album IS NOT NULL THEN
                SELECT artista INTO artista_album
                FROM Album WHERE id_album = NEW.album;
                IF NEW.artista <> artista_album THEN
                        SIGNAL SQLSTATE "45000"
                        SET MESSAGE_TEXT = "Artista dell'album diverso dall'artista della canzone.";
                END IF;
        END IF;
END;
```

per mantenere il vincolo d'integrità verrà implementato il trigger chk_artista_album_on_upd analogo a chk_artista_album_on_ins ma BEFORE UPDATE.

```
CREATE TRIGGER upd_data_canzoni_album
BEFORE INSERT ON Canzone
FOR EACH ROW
BEGIN
        DECLARE data_album TIMESTAMP;
        IF NEW.album IS NOT NULL THEN
                SELECT data_distribuzione INTO data_album
      FROM Album WHERE id_album = NEW.album;
      SET NEW.data_pubblicazione = data_album;
   END IF;
END;
```

Per l'inserimento di nuovi utenti bisogna effettuare query del tipo:

INSERT INTO **Utente** (username, nome, cognome) **VALUES** (*dati nuovo utente…*)**;**

mentre per la cancellazione bisogna effettuare query del tipo:

DELETE FROM **Utente** **WHERE** id_utente = utente_da_cancellare**;**

O4: *Inserimento di un nuovo album*:
Per l'implementazione di **O4** basta effettuare query del tipo:

INSERT INTO **Album** (titolo, artista, major_distribuzione) **VALUES** (*dati nuovo album…*)**;**

insieme ad **O3** per l'inserimento delle nuove canzoni e query del tipo:

UPDATE **Canzone** **SET** album = *id_nuovo_album* **WHERE** id_canzone = *canzone_da_aggiungere***;**

per aggiornare singoli già pubblicati e aggiungerli all'album.

O5: *Inserimento di una nuova playlist*:
Per l'implementazione di **O5** basta effettuare query del tipo:

INSERT INTO **Playlist** (nome_playlist, creatore) **VALUES** (*dati nuova playlist…*)**;**

mentre per la cancellazione bisogna effettuare query del tipo:

DELETE FROM **Playlist** **WHERE** id_playlist = *playlist_da_cancellare***;**

O6: *Inserimento di una nuova canzone in una playlist*:
Per l'implementazione di **O6** basta effettuare query del tipo:

INSERT INTO **CanzoneInPlaylist** (canzone, playlist) **VALUES** (*dati nuova canzone in playlist…*)**;**

mentre per la cancellazione bisogna effettuare query del tipo:

DELETE FROM **CanzoneInPlaylist** **WHERE** canzone = *canzone_da_rimuovere*
AND playlist = *playlist_dalla_quale_rimuovere_la_canzone***;**

O7: *Inserimento di un nuovo artista*:
Per l'implementazione di **O7** basta effettuare query del tipo:

INSERT INTO **Artista** (pseudonimo, biografia, is_gruppo, is_cantante, is_produttore, is_scrittore)
VALUES (*dati nuovo artista…*)**;**

in questo caso non è necessario implementare alcun trigger, poiché i vincoli d'integrità sono già stati specificati nella creazione della tabella **Artista**.

O8: *Inserimento di una nuova recensione*:
Per l'implementazione di **O8** basta effettuare query del tipo:

INSERT INTO Recensione (utente, canzone, recensione, voto) **VALUES** (*dati nuova recensione…*)**;**

mentre per la cancellazione basta effettuare query del tipo:

DELETE FROM Recensione WHERE utente = *autore_della_recensione*
AND canzone = *canzone_recensita;*

Per aggiungere una nuova major bisogna effettuare query del tipo:

INSERT INTO Major (nome_azienda, sede) **VALUES** (*dati nuova major…*)**;**

O9: *Calcolo ascolti totali di un artista*:
Per l'implementazione di **O9** basta effettuare la query:

CREATE VIEW AscoltiTotaliArtista AS
SELECT ascolti_totali **FROM Artista WHERE** id_artista = *artista_interessato*;

O10: *Calcolo ascolti totali di una major*:
Per l'implementazione di **O10** basta effettuare la query:

CREATE VIEW AscoltiTotaliMajor AS
SELECT SUM(ascolti_totali) **AS** ascolti_totali **FROM Album**
WHERE major_distribuzione = *major_interessata*;

Nell'implementazione delle operazioni del tipo *creazione classifica **n** elementi più ascoltati* verrà utilizzata l'istruzione **LIMIT** che permette di ottenere **n** record dall'operazione **SELECT.**

O11: *Creazione classifica 100 brani più ascoltati*:
Per l'implementazione di **O11** bisogna effettuare la query:

CREATE VIEW ClassificaCanzoni AS
SELECT id_canzone, titolo, ascolti **FROM Canzone**
ORDER BY ascolti **DESC LIMIT 100**;

O12: *Creazione classifica 50 album più ascoltati*:
Per l'implementazione di **O12** bisogna effettuare la query:

CREATE VIEW ClassificaAlbum AS
SELECT id_album, titolo, ascolti_totali **FROM Album**
ORDER BY ascolti_totali **DESC LIMIT 50**;

O13: *Creazione classifica 10 artisti più ascoltati*:
Per l'implementazione di **O13** bisogna effettuare la query:

```
CREATE VIEW ClassificaArtisti AS
SELECT id_artista, pseudonimo, ascolti_totali FROM Artista
ORDER BY ascolti_totali DESC LIMIT 10;
```

O14: *Visualizzazione 10 brani preferiti da un utente*:
Per l'implementazione di **O14** bisogna effettuare la query:

```
CREATE VIEW BraniPreferitiUtente AS
SELECT id_canzone, titolo, voto FROM Recensione, Canzone
WHERE utente = utente_interessato AND id_canzone = canzone
ORDER BY voto DESC LIMIT 10;
```

O15: *Visualizzazione 5 artisti preferiti da un utente*:
Per l'implementazione di **O15** bisogna effettuare la query:

```
CREATE VIEW ArtistiPreferitiUtente AS
SELECT id_artista, pseudonimo, AVG(voto) AS voto_medio FROM Recensione, Canzone, Artista
WHERE utente = utente_interessato AND id_canzone = canzone AND id_artista = artista
GROUP BY id_artista ORDER BY voto_medio DESC LIMIT 5;
```

6. Esempio di Dati

In seguito all'implementazione di tutte le tabelle e tutti i trigger in SQL, dopo essere stata popolata con dati di prova, si avrà un risultato del genere:

Esempio di Canzoni:

id_canzone	titolo	durata	ascolti	genere	artista	cantante	produttore	scrittore	album	data_pubblicazione
65	Prison Song	204	4830382	Metal	12	12	12	12	5	2001-09-04 00:00:00
66	Needles	193	16417895	Metal	12	12	12	12	5	2001-09-04 00:00:00
67	Deer Dance	176	2678360	Metal	12	12	12	12	5	2001-09-04 00:00:00
68	Jet Pilot	126	5194849	Metal	12	12	12	12	5	2001-09-04 00:00:00
69	X	118	4476613	Metal	12	12	12	12	5	2001-09-04 00:00:00
70	Chop Suey!	208	1344944631	Metal	12	12	12	12	5	2001-09-04 00:00:00
71	Bounce	115	8038881	Metal	12	12	12	12	5	2001-09-04 00:00:00
72	Forest	240	20567848	Metal	12	12	12	12	5	2001-09-04 00:00:00
73	ATWA	177	4732849	Metal	12	12	12	12	5	2001-09-04 00:00:00
74	Science	162	8508305	Metal	12	12	12	12	5	2001-09-04 00:00:00
75	Shimmy	110	6490375	Metal	12	12	12	12	5	2001-09-04 00:00:00
76	Toxicity	224	868781010	Metal	12	12	12	12	5	2001-09-04 00:00:00
77	Psycho	227	1828483	Metal	12	12	12	12	5	2001-09-04 00:00:00
78	Aerials	243	446858657	Metal	12	12	12	12	5	2001-09-04 00:00:00
79	Arto	133	1924227	Metal	12	12	12	12	5	2001-09-04 00:00:00
105	L'Amour Toujours	242	467005787	Italod…	16	NULL	16	NULL	NULL	2024-02-22 13:47:03
106	Bla Bla Bla	191	72383380	Italod…	16	NULL	16	NULL	NULL	2024-02-22 13:47:51
107	The Riddle	206	93821033	Italod…	16	NULL	16	NULL	NULL	2024-02-22 13:48:39
83	Harder, Better, Faster, Stronger	222	133853269	Elettr…	13	NULL	13	13	6	2001-02-26 00:00:00
80	One More Time	321	490402644	Elettr…	13	NULL	13	13	6	2001-02-26 00:00:00

Esempio di Artisti:

id_artista	pseudonimo	biografia	ascolti_totali	is_gruppo	is_cantante	is_produttore	is_scrittore
11	Marracash	Marracash, pseudonimo di Fabio Bartolo Rizzo (Nicosia, 22 magg…	9030475	0	1	0	1
12	System Of A Down	I System of a Down (conosciuti anche con l'acronimo SOAD) son…	2746273365	1	1	1	1
13	Daft Punk	I Daft Punk sono stati un gruppo musicale francese di musica ele…	942495368	1	0	1	1
14	Rkomi	Rkomi, nome d'arte di Mirko Manuele Martorana (Milano, 19 april…	51651585	0	1	0	1
15	Ghali	Ghali, nome d'arte di Ghali Amdouni (Milano, 21 maggio 1993), p…	137985268	0	1	0	1
16	Gigi d'Agostino	Gigi D'Agostino, pseudonimo di Luigino Celestino Di Agostino (To…	633210200	0	0	1	0

Esempio di Album:

id_album	titolo	ascolti_totali	artista	major_distribuzione	data_distribuzione
4	XDVR Reloaded	87885138	4	1	2015-11-23 00:00:00
5	Toxicity	2746273365	12	2	2001-09-04 00:00:00
6	Discovery	942495368	13	3	2001-02-26 00:00:00

Per un esempio pratico è possibile caricare in MySQL i file allegati:

- **tabelle_e_triggers.sql** per l'implementazione di tabelle con i relativi triggers per le varie operazioni.
- **dati_esempio.sql** per riempire la base di dati con dati di esempio basati sui gusti personali del sottoscritto insieme ad utenti, recensioni e playlist fittizie.
- **viste.sql** per la creazione di viste basate sui dati caricati in precedenza