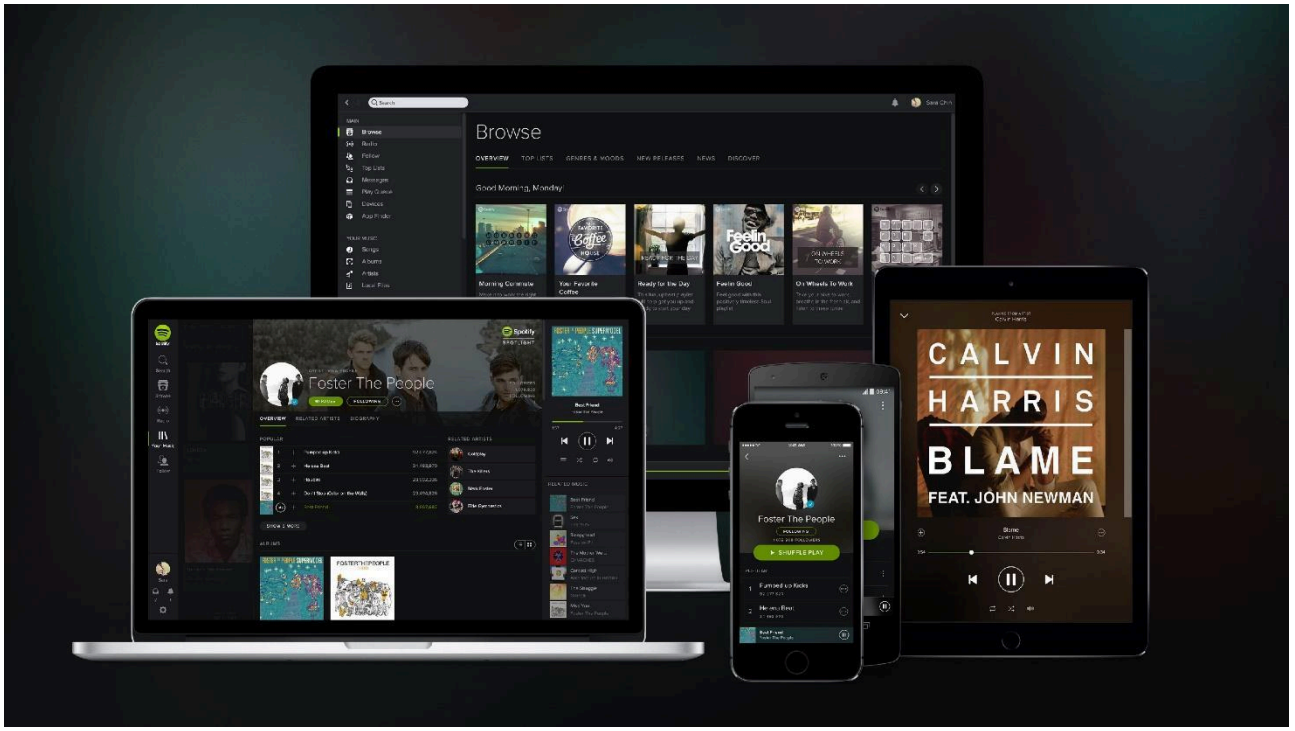
Spotify®

Un servizio musicale digitale che ti consente di accedere a milioni di brani.



A.A. 2016/2017

Fabiano Golluscio

Introduzione

Il database in questione é stato progettato e realizzato per rappresentare la mia idea del reale database utilizzato dalla startup svedese Spotify AB per il loro servizio musicale di streaming ondemand.

Lo scopo é quello di analizzare le preferenze di ciascun utente.

Si pone quindi l'attenzione in particolare sulle possibilitá che ha l'utente all'interno del software, come seguire altri utenti o artisti, comprare la loro merce o i biglietti per i loro concerti oltre alle classiche attivitá relative ai brani musicali.

Descrizione del DataBase

Il database é realizzato in modo tale da gestire:

- i follow di ciascun utente, sia verso altri utenti che verso gli artisti;
- lo storico di abbonamenti registrati;
- il catalogo musicale dei brani e le relative informazioni (es. scrittore);
- le playlist create e i brani aggiunti dagli utenti;
- gli album pubblicati dagli artisti e i concerti che terranno;
- i biglietti dei relativi concerti;
- il merchandising di un gruppo/artista;
- gli acquisti di un utente.

Descrizione delle Entitá

Abbonamento: una sottoscrizione mensile che permette di accedere al servizio Spotify Premium, il quale consente il download dei brani o di poter scegliere nello specifico quale brano ascoltare, inoltre elimina gli spot pubblicitari periodici.

Album: una raccolta di brani musicali accomunati dallo stesso artista, diversamente dalle playlist questi non sono modificabili dall'utente.

Artista: identifica la singola persona o il gruppo musicale che performa un brano, in Spotify é possibile seguire un artista per ricevere tutti i suoi aggiornamenti e riprodurre la sua discografia.

Autore: l'entitá rappresenta l'autore del brano.
Chi lo scrive, spesso, non corrisponde a chi lo performa, un brano puo' essere scritto da piú persone.

Biglietto: i biglietti per accedere ai concerti, acquistabili direttamente dall'applicazione.

Brano: l'entitá rappresenta il brano musicale che puo' essere usato per comporre una playlist e registra il numero di ascolti.
Quando un utente riproduce un brano, questo viene aggiunto ad un elenco, legato al profilo, di tutti i brani riprodotti.

Concerto: in Spotify é presente una sezione dedicata ai concerti con tutte le informazioni utili per accedervi; ad un concerto possono esibirsi piú artisti.

Playlist: questa raccolta di brani senza vincoli viene creata dall'utente, é condivisibile e personalizzabile, tra le varie impostazioni associate si ha la possibilitá di renderla collaborativa e privata; una playlist registra il numero di utenti che la seguono.

Prodotto: nel profilo di ogni artista é possibile accedere al suo merchandising, si possono quindi comprare magliette e altri tipi di prodotti a tema.

Utente: é il cliente di Spotify, che puo' essere di tipo premium o standard, le azioni che puo' compiere all'interno del client sono numerose.
Dato che Spotify ha un lato "social", l'utente ha la possibilitá di seguire altri utenti, in questo modo puo' vedere in diretta cosa stanno ascoltando e puo' accedere alle playlist pubbliche.
É anche presente una funzione di messaggistica e di condivisione privata tra utenti, inoltre é possibile collegare il profilo ai social come Facebook in modo da mantenere le informazioni sempre aggiornate. Si contano i follower e i following di ciascun utente e si registra uno storico di tutte le transazioni relative agli acquisti di biglietti e/o prodotti dei vari artisti.

Descrizione delle Relazioni

Aggiunge: quando un utente riproduce un brano, quest'ultimo viene aggiunto ad un elenco di tutti i brani riprodotti.

Composto: gli album sono composti da una serie di brani, un album appartiene ad un artista e di conseguenza pure tutti i brani in esso contenuti.

Compra: gli utenti possono acquistare dei prodotti, venduti dagli artisti, all'interno del software.

Contiene: i brani possono essere raccolti in playlist personalizzate dagli utenti, queste possono contenere qualsiasi brano.

Crea: gli utenti possono creare playlist da personalizzare a loro piacimento.

Esibisce: un artista puó esibirsi in un determinato concerto.

Scrive: un brano puo' essere scritto da uno o piú autori, i quali possono essere o meno gli stessi artisti.

Segueart: gli utenti possono seguire gli artisti e riceve cosí gli aggiornamenti.

Segueut: gli utenti possono seguire anche altri utenti.

CompraBig: gli utenti possono comprare i biglietti per partecipare ai concerti.

Distribuisce: un concerto "distribuisce" una seria di biglietti.

Performa: ogni brano viene performato da un artista.

Pubblica: gli artisti pubblicano i loro album.

Sottoscrive: Spotify è un servizio a pagamento, quindi gli utenti possono (facoltativamente) sottoscrivere un abbonamento mensile.

Vende: gli artisti vendono prodotti con il loro marchio.

Progettazione Concettuale

Analisi Formale dei Requisiti

Il modello entitá/relazione (**E/R**), è uno strumento per analizzare le caratteristiche di una realtá in modo indipendente dagli eventi che in essa accadono, cioé per costruire un modello concettuale dei dati indipendenti dalle applicazioni.

Il risultato di questo lavoro è la definizione di una rappresentazione grafica, detta **Schema E/R**, che mette in evidenza gli aspetti fondamentali del modello concettuale, con i dati caratterizzati e le associazioni tra essi.

Il modello descrive lo schema concettuale di un problema o di una gestione aziendale e non si occupa dell'efficienza delle operazioni di manipolazione e ritrovamento dei dati sugli archivi fisici.

Gli elementi di un modello Entity/Relationship sono:

Entitá: è un oggetto (concreto o astratto) che ha un significato anche quando viene considerato in modo isolato ed è di interesse per la realtà che si vuole modellare.

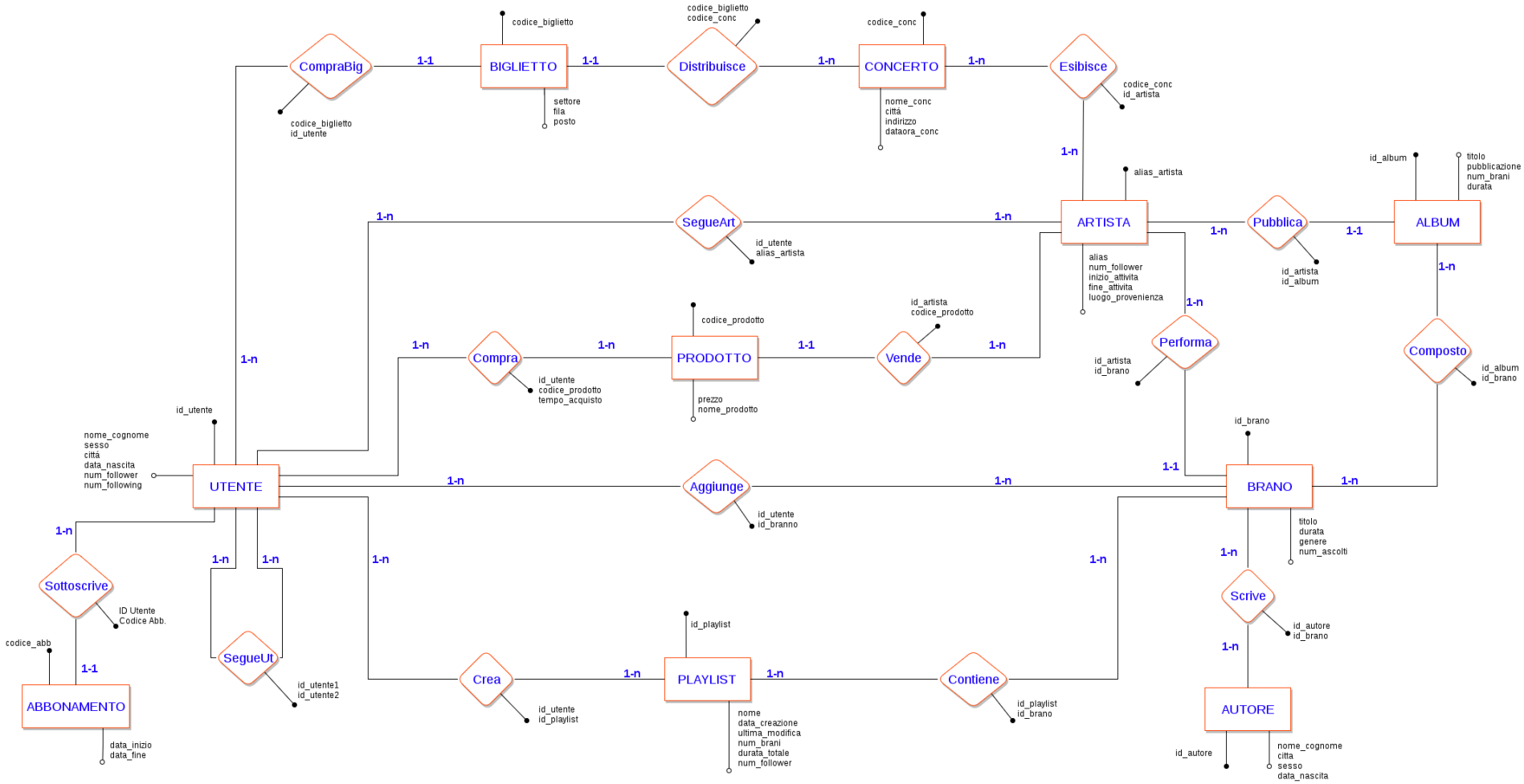
Relazione: è un legame che stabilisce un'interazione tra le entità.

Attributi: le proprietà delle entità e delle associazioni sono descritte attraverso gli attributi.

NOME	PRIMARY KEY	ATTRIBUTI	RELAZIONI
abbonamento	codice	data_inizio data_fine id_utente (FK)	utente
album	id_album	alias_artista (FK) num_brani titolo pubblicazione	artista brano
artista	alias_artista	num_follower	album brano concerto prodotto utente
biglietto	codice_biglietto	codice_conc (FK) fila id_utente (FK) posto settore	concerto utente
brano	id_brano	alias_artista (FK) durata genere num_ascolti titolo	album artista playlist scrittore utente
concerto	codice_concerto	cittá data_conc luogo nome_conc ora	artista biglietto
playlist	id_playlist	data_creazione durata_totale nome num_brani num_follower ultima_modifica	brano utente
prodotto	codice_prodotto	alias_artista (FK) id_utente (FK) nome_prodotto prezzo	artista utente
scrittore	alias_scrittore	nome cognome	brano
utente	id_utente	nome cognome num_follower num_following	abbonamento artista biglietto brano playlist prodotto utente

NOME	IDENTIFICATORI	COLLEGA
aggiunge	id_utente id_brano	utente brano
composto	id_album id_brano	album brano
compra	id_utente codice_prodotto	utente prodotto
compraBig	id_utente codice_biglietto	utente biglietto
contiene	id_playlist id_brano	playlist brano
crea	id_utente id_playlist	utente playlist
distribuisce	codice_conc codice_biglietto	concerto biglietto
esibisce	codice_conc id_artista	artista concerto
performa	id_brano id_artista	brano artista
pubblica	id_artista id_album	artista album
scrive	id_autore id_brano	autore brano
segueArt	id_utente id_artista	utente artista
segueUt	id_utente1 id_utente2	utente utente
sottoscrive	id_utente codice_abb	utente abbonamento
vende	id_artista codice_prodotto	artista prodotto

Diagramma E/R


CompraBig
codice_biglietto
id_utente
BIGLIETTO
codice_biglietto
settore
fila
posto
Distribuisce
codice_biglietto
codice_conc
CONCERTO
codice_conc
nome_conc
città
indirizzo
dataora_conc
Esibisce
codice_conc
id_artista
ARTISTA
alias_artista
alias
num_follower
inizio_attività
fine_attività
luogo_provenienza
SegueArt
id_utente
alias_artista
Pubblica
id_artista
id_album
ALBUM
id_album
titolo
pubblicazione
num_brani
durata
Performa
id_artista
id_brano
Composto
id_album
id_brano
Compra
id_utente
codice_prodotto
tempo_acquisto
PRODOTTO
codice_prodotto
prezzo
nome_prodotto
Vende
id_artista
codice_prodotto
UTENTE
id_utente
nome_cognome
sesso
città
data_nascita
num_follower
num_following
Aggiunge
id_utente
id_branno
BRANO
id_brano
titolo
durata
genere
num_ascolti
Scrive
id_autore
id_brano
AUTORE
id_autore
nome_cognome
citta
sesso
data_nascita
Sottoscrive
ID Utente
Codice Abb.
ABBONAMENTO
codice_abb
data_inizio
data_fine
SegueUt
id_utente1
id_utente2
Crea
id_utente
id_playlist
PLAYLIST
id_playlist
nome
data_creazione
ultima_modifica
num_brani
durata_totale
num_follower
Contiene
id_playlist
id_brano
1-1
1-n

Progettazione Logica

L'obiettivo della progettazione logica è quello di costruire uno schema logico in grado di descrivere, in maniera corretta ed efficiente, tutte le informazioni contenute nello schema E/R prodotto nella fase di progettazione concettuale.

La semplificazione dello schema si rende necessaria perché non tutti i costrutti del modello E/R hanno una traduzione naturale nei modelli logici, ad esempio le generalizzazioni.

Utilizzeremo anche la normalizzazione che ci permetterá la ristrutturazione dello schema in modo tale da definire lo schema fisico che sará la base per la costruzione del nostro database.

Normalizzazione

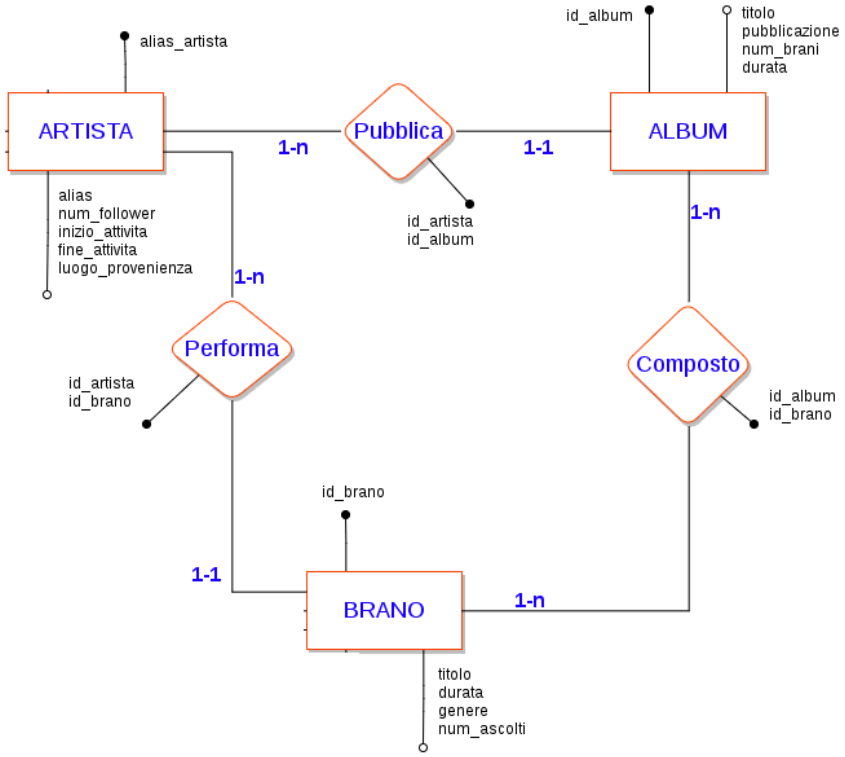
Una relazione è un legame che stabilisce un'interazione tra le entitá.

La cardinalitá di un verso della relazione è la caratteristica che indica quante istanze dell'entitá di arrivo si associano all'istanza dell'entitá di partenza.

La cardinalitá puó essere **a uno** oppure **a molti** e pertanto le relazioni tra due entitá si classificano in:

- **1 : 1**
  Ad ogni elemento del primo insieme corrisponde un unico elemento del secondo insieme e viceversa.
- **1 : N**
  Ad un elemento del primo insieme possono corrispondere piú elementi del secondo, mentre ad ogni elemento del secondo insieme deve corrispondere un unico elemento del primo.
- **N : M**
  Ad ogni elemento del primo insieme possono corrispondere piú elementi del secondo insieme e viceversa.

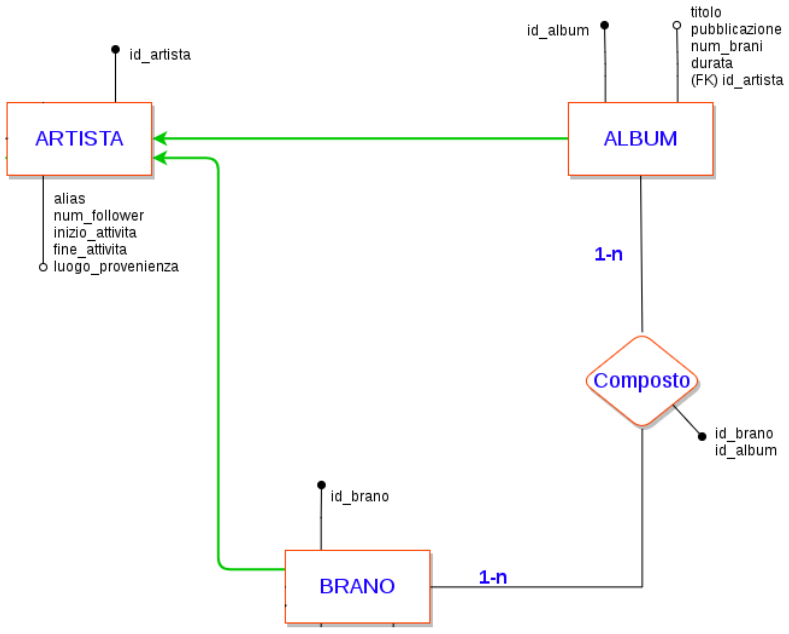
Adesso vediamo quali relazioni dovranno essere normalizzate per dedurre lo **Schema Logico**:



Prendiamo per esempio la relazione **Performa** che collega le due entitá **Brano** e **Artista**, questa ha cardinalitá (1-1) - (1-N), infatti un artista puo' performare un qualunque numero di brani, ma uno specifico brano è performato solo da quell'artista, nel caso di ripetizioni allora l'identitá del brano è diversa.

Lo schema, in questo caso, è quindi normalizzabile eliminando la relazione **Performa** e facendo diventare la chiave primaria di **Artista** un attributo di **Brano**.

Si ha lo stesso caso con la relazione **Pubblica** che collega **Album** e **Artista**.



Le Tre Forme Normali

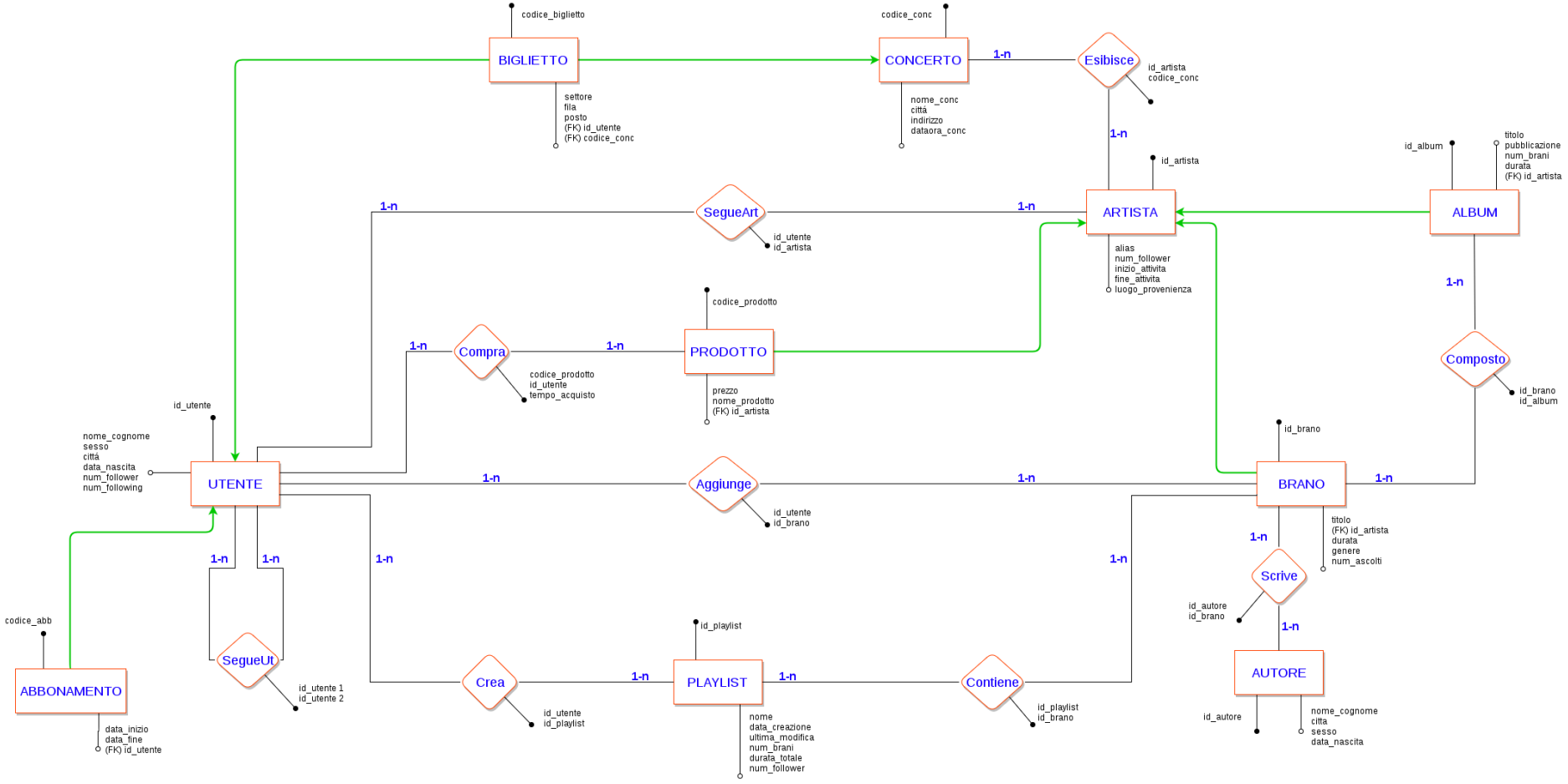
- **Prima Forma Normale:**
  Una relazione è in prima forma normale (1FN) quando rispetta i requisiti fondamentali del modello relazionale, cioé:
  - Tutte le righe della tabella contengono lo stesso numero di colonne;
  - Gli attributi rappresentano informazioni elementari;
  - I valori che compaiono in una colonna sono dello stesso tipo, cioé appartengono allo stesso dominio;
  - Ogni riga è diversa da tutte le altre, cioé non ci possono essere due righe con gli stessi valori nelle colonne;
  - L'ordine con il quale le righe compaiono nella tabella è irrilevante.
- **Seconda Forma Normale:**
  Una relazione è in seconda forma normale (2FN) quando è in prima forma normale e tutti i suoi attributi non-chiave dipendono dall'intera chiave.
  Cioé non possiede attributi che dipendono soltanto da una parte della chiave.
- **Terza Forma Normale:**
  Una relazione è in terza forma normale (3FN) quando è in seconda forma normale e tutti gli attributi non-chiave dipendono direttamente dalla chiave.
  Cioé non possiede attributi non-chiave che dipendono da altri attributi non-chiave.

Queste forme normali, insieme ai processi effettuati in precedenza rappresentano il signifcato della normalizzazione.

Si deve osservare che la normalizzazione diminuisce la ridondanza dei dati e la possibilitá di inconsistenza, ma rende piú complesse le operazioni di ritrovamento dei dati.

La normalizzazione è importante nel modello di un database perché l'integritá e la consistenza dei dati sono prioritarie rispetto alla velocitá di ritrovamento dei dati, che rimane comunque un fattore essenziale.

Schema Logico


codice_biglietto
BIGLIETTO
settore
fila
posto
(FK) id_utente
(FK) codice_conc
codice_conc
CONCERTO
nome_conc
città
indirizzo
dataora_conc
1-n
Esibisce
id_artista
codice_conc
1-n
id_artista
ARTISTA
alias
num_follower
inizio_attivita
fine_attivita
luogo_provenienza
id_album
titolo
pubblicazione
num_brani
durata
(FK) id_artista
ALBUM
1-n
Composto
id_brano
id_album
1-n
SegueArt
id_utente
id_artista
1-n
codice_prodotto
PRODOTTO
prezzo
nome_prodotto
(FK) id_artista
1-n
Compra
codice_prodotto
id_utente
tempo_acquisto
1-n
id_utente
nome_cognome
sesso
città
data_nascita
num_follower
num_following
UTENTE
1-n
Aggiunge
id_utente
id_brano
1-n
id_brano
BRANO
titolo
(FK) id_artista
durata
genere
num_ascolti
1-n
Scrive
id_autore
id_brano
1-n
AUTORE
id_autore
nome_cognome
citta
sesso
data_nascita
1-n
1-n
SegueUt
id_utente 1
id_utente 2
1-n
Crea
id_utente
id_playlist
1-n
id_playlist
PLAYLIST
nome
data_creazione
ultima_modifica
num_brani
durata_totale
num_follower
1-n
Contiene
id_playlist
id_brano
1-n
codice_abb
ABBONAMENTO
data_inizio
data_fine
(FK) id_utente

Progettazione Fisica

Livello Fisico

Il livello fisico rappresenta l'effettiva installazione degli archivi elettronici, esso indica l'ubicazione dei dati nelle memorie di massa.

Il livello fisico è quindi l'implementazione del livello logico sui supporti per la registrazione fisica dei dati.

In questo capitolo svilupperemo il database tramite MySQL usato come **Relationale DataBase Management System** (**RDMS**).

Creazione del Database

Adesso quindi creiamo la nostra base di dati con il nome che desideriamo, **spotify** quindi, ed usiamola.

Utilizziamo quindi i seguenti comandi.

```
1  create database spotify;
2
3  use spotify;
```

Adesso abbiamo il pieno controllo del database e possiamo eseguire tutte le operazioni volute, come la creazione delle tabelle.

Creazione delle Tabelle

Adesso andremo a creare la struttura del nostro database, implementando le tabelle attraverso i comandi che consentono di farlo.

Quando si crea una nuova tabella è possibile specificare quale motore di archiviazione usare aggiungendo l'opzione **ENGINE** al comando **CREATE TABLE**.

Se questa opzione viene omessa, allora viene utilizzato il metodo di archiviazione di default che per MySQL 5.7 è InnoDB.

```
mysql> select @@default_storage_engine;
+--------------------------+
| @@default_storage_engine |
+--------------------------+
| InnoDB                   |
+--------------------------+
1 row in set (0,00 sec)
```

Con questa query abbiamo verificato che il metodo di archiviazione è realmente InnoDB.

Possiamo adesso procedere con la creazione di tutte le tabelle nel database, seguendo un ordine tale che non generi conflitti con le foreign key.

Quindi utilizzeremo il comando CREATE TABLE per creare le tabelle e DESCRIBE TABLE per vedere come queste sono definite.

Tabella 1. Autore

```
 5  create table autore(
 6    id_autore int not null primary key,
 7    nome_cognome varchar(60) not null,
 8    sesso varchar(10) not null,
 9    citta varchar(60) not null,
10    data_nascita date not null
11  );
```

```
mysql> describe autore;
+--------------+-------------+------+-----+---------+-------+
| Field        | Type        | Null | Key | Default | Extra |
+--------------+-------------+------+-----+---------+-------+
| id_autore    | int(11)     | NO   | PRI | NULL    |       |
| nome_cognome | varchar(60) | NO   |     | NULL    |       |
| sesso        | varchar(10) | NO   |     | NULL    |       |
| citta        | varchar(60) | NO   |     | NULL    |       |
| data_nascita | date        | NO   |     | NULL    |       |
+--------------+-------------+------+-----+---------+-------+
5 rows in set (0,00 sec)
```

Come possiamo vedere, nella tabella Autore sono presenti 5 attributi tra cui “id_autore” che è la **PRIMARY KEY** della tabella.

Questa fa si che tutti i record che andremo a inserire non saranno uguali per tale attributo, in modo da rispettare i criteri della forma normale.

Tabella 2. Artista

```
13  create table artista(
14    id_artista int not null primary key,
15    alias varchar(100) not null,
16    num_follower int,
17    inizio_attivita date not null,
18    fine_attivita date,
19    luogo_provenienza varchar(60) not null
20  );
```

```
mysql> describe artista;
+-------------------+--------------+------+-----+---------+-------+
| Field             | Type         | Null | Key | Default | Extra |
+-------------------+--------------+------+-----+---------+-------+
| id_artista        | int(11)      | NO   | PRI | NULL    |       |
| alias             | varchar(100) | NO   |     | NULL    |       |
| num_follower      | int(11)      | YES  |     | NULL    |       |
| inizio_attivita   | date         | NO   |     | NULL    |       |
| fine_attivita     | date         | YES  |     | NULL    |       |
| luogo_provenienza | varchar(60)  | NO   |     | NULL    |       |
+-------------------+--------------+------+-----+---------+-------+
6 rows in set (0,00 sec)
```

Nella tabella Artista notiamo due attributi “num_follower” e “fine_attivita” i quali hanno la possibilitá di essere NULL.

Per quanto riguarda l’attributo “num_follower”, è stato implementato un trigger che permette un aggiornamento automatico di tale valore.

Ma questo argomento verrá trattato piú avanti.

Tabella 3. Utente

```
22  create table utente(
23    id_utente int not null primary key,
24    nome_cognome varchar(60) not null,
25    data_nascita date not null,
26    sesso varchar(10) not null,
27    citta varchar(60) not null,
28    num_follower int,
29    num_following int
30  );
```

```
mysql> describe utente;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| id_utente     | int(11)     | NO   | PRI | NULL    |       |
| nome_cognome  | varchar(60) | NO   |     | NULL    |       |
| data_nascita  | date        | NO   |     | NULL    |       |
| sesso         | varchar(10) | NO   |     | NULL    |       |
| citta         | varchar(60) | NO   |     | NULL    |       |
| num_follower  | int(11)     | YES  |     | NULL    |       |
| num_following | int(11)     | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
7 rows in set (0,00 sec)
```

Tabella 4. Abbonamento

```
32  create table abbonamento(
33    codice int not null primary key,
34    data_inizio date not null,
35    data_fine date not null,
36    id_utente int not null,
37    foreign key(id_utente) references utente(id_utente)
38  );
```

```
mysql> describe abbonamento;
+-------------+---------+------+-----+---------+-------+
| Field       | Type    | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| codice      | int(11) | NO   | PRI | NULL    |       |
| data_inizio | date    | NO   |     | NULL    |       |
| data_fine   | date    | NO   |     | NULL    |       |
| id_utente   | int(11) | NO   | MUL | NULL    |       |
+-------------+---------+------+-----+---------+-------+
4 rows in set (0,00 sec)
```

TABELLA 5. PLAYLIST

```
40  create table playlist(
41    id_playlist int not null primary key,
42    nome varchar(100) not null,
43    num_brani int,
44    durata_totale time,
45    num_follower int not null,
46    data_creazione date not null,
47    ultima_modifica date
48  );
```

```
mysql> describe playlist;
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| id_playlist     | int(11)      | NO   | PRI | NULL    |       |
| nome            | varchar(100) | NO   |     | NULL    |       |
| num_brani       | int(11)      | YES  |     | NULL    |       |
| durata_totale   | time         | YES  |     | NULL    |       |
| num_follower    | int(11)      | NO   |     | NULL    |       |
| data_creazione  | date         | NO   |     | NULL    |       |
| ultima_modifica | date         | YES  |     | NULL    |       |
+-----------------+--------------+------+-----+---------+-------+
7 rows in set (0,00 sec)
```

Tabella 6. Album

```
50  create table album(
51    id_album int not null primary key,
52    id_artista int not null,
53    titolo varchar(100) not null,
54    pubblicazione date not null,
55    num_brani int not null,
56    durata time not null,
57    foreign key(id_artista) references artista(id_artista)
58  );
```

```
mysql> describe album;
+---------------+--------------+------+-----+---------+-------+
| Field         | Type         | Null | Key | Default | Extra |
+---------------+--------------+------+-----+---------+-------+
| id_album      | int(11)      | NO   | PRI | NULL    |       |
| id_artista    | int(11)      | NO   | MUL | NULL    |       |
| titolo        | varchar(100) | NO   |     | NULL    |       |
| pubblicazione | date         | NO   |     | NULL    |       |
| num_brani     | int(11)      | NO   |     | NULL    |       |
| durata        | time         | NO   |     | NULL    |       |
+---------------+--------------+------+-----+---------+-------+
6 rows in set (0,00 sec)
```

Nella tabella Album vediamo un primo utilizzo della **FOREIGN KEY**, questa serve a collegare l'attributo "id_artista" di Album all'attributo omonimo della tabella Artista.

In questo modo si crea un collegamento tra le due tabelle, in quanto uno fa riferimento all'altro.

Tabella 7. Brano

```
60  create table brano(
61    id_brano int not null primary key,
62    titolo varchar(100) not null,
63    durata time not null,
64    genere varchar(30) not null,
65    num_ascolti int,
66    id_artista int not null,
67    foreign key(id_artista) references artista(id_artista)
68  );
```

```
mysql> describe brano;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| id_brano    | int(11)      | NO   | PRI | NULL    |       |
| titolo      | varchar(100) | NO   |     | NULL    |       |
| durata      | time         | NO   |     | NULL    |       |
| genere      | varchar(30)  | NO   |     | NULL    |       |
| num_ascolti | int(11)      | YES  |     | NULL    |       |
| id_artista  | int(11)      | NO   | MUL | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
6 rows in set (0,00 sec)
```

Tabella 8. Prodotto

```
70  create table prodotto(
71    codice_prodotto int not null primary key,
72    nome_prodotto varchar(200) not null,
73    prezzo int not null,
74    id_artista int not null,
75    foreign key(id_artista) references artista(id_artista)
76  );
```

```
mysql> describe prodotto;
+-----------------+--------------+------+-----+---------+-------+
| Field           | Type         | Null | Key | Default | Extra |
+-----------------+--------------+------+-----+---------+-------+
| codice_prodotto | int(11)      | NO   | PRI | NULL    |       |
| nome_prodotto   | varchar(200) | NO   |     | NULL    |       |
| prezzo          | int(11)      | NO   |     | NULL    |       |
| id_artista      | int(11)      | NO   | MUL | NULL    |       |
+-----------------+--------------+------+-----+---------+-------+
4 rows in set (0,00 sec)
```

TABELLA 9. CONCERTO

```
78  create table concerto(
79    codice_conc int not null primary key,
80    nome_conc varchar(100) not null,
81    citta varchar(60) not null,
82    indirizzo varchar(200) not null,
83    dataora_conc datetime not null,
84  );
```

```
mysql> describe concerto;
+--------------+--------------+------+-----+---------+-------+
| Field        | Type         | Null | Key | Default | Extra |
+--------------+--------------+------+-----+---------+-------+
| codice_conc  | int(11)      | NO   | PRI | NULL    |       |
| nome_conc    | varchar(100) | NO   |     | NULL    |       |
| citta        | varchar(60)  | NO   |     | NULL    |       |
| indirizzo    | varchar(200) | NO   |     | NULL    |       |
| dataora_conc | datetime     | NO   |     | NULL    |       |
+--------------+--------------+------+-----+---------+-------+
5 rows in set (0,00 sec)
```

Tabella 10. Biglietto

```
86  create table biglietto(
87    codice_biglietto int not null primary key,
88    settore varchar(10) not null,
89    fila varchar(5) not null,
90    posto int not null,
91    id_utente int,
92    codice_conc int not null,
93    foreign key(id_utente) references utente(id_utente),
94    foreign key(codice_conc) references concerto(codice_conc)
95  );
```

```
mysql> describe biglietto;
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| codice_biglietto | int(11)     | NO   | PRI | NULL    |       |
| settore          | varchar(10) | NO   |     | NULL    |       |
| fila             | varchar(5)  | NO   |     | NULL    |       |
| posto            | int(11)     | NO   |     | NULL    |       |
| id_utente        | int(11)     | YES  | MUL | NULL    |       |
| codice_conc      | int(11)     | NO   | MUL | NULL    |       |
+------------------+-------------+------+-----+---------+-------+
6 rows in set (0,00 sec)
```

Tabella 11. SegueUt

```
97  create table segueut(
98    id_utente1 int not null,
99    id_utente2 int not null,
100   primary key(id_utente1,id_utente2),
101   foreign key(id_utente1) references utente(id_utente),
102   foreign key(id_utente2) references utente(id_utente)
103 );
```

```
mysql> describe segueut;
+------------+---------+------+-----+---------+-------+
| Field      | Type    | Null | Key | Default | Extra |
+------------+---------+------+-----+---------+-------+
| id_utente1 | int(11) | NO   | PRI | NULL    |       |
| id_utente2 | int(11) | NO   | PRI | NULL    |       |
+------------+---------+------+-----+---------+-------+
2 rows in set (0,01 sec)
```

Tabella 12. Crea

```
105  create table crea(
106    id_utente int not null,
107    id_playlist int not null,
108    primary key(id_utente,id_playlist),
109    foreign key(id_utente) references utente(id_utente),
110    foreign key(id_playlist) references playlist(id_playlist)
111  );
```

```
mysql> describe crea;
+-------------+---------+------+-----+---------+-------+
| Field       | Type    | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| id_utente   | int(11) | NO   | PRI | NULL    |       |
| id_playlist | int(11) | NO   | PRI | NULL    |       |
+-------------+---------+------+-----+---------+-------+
2 rows in set (0,00 sec)
```

Tabella 13. Contiene

```
113  create table contiene(
114    id_playlist int not null,
115    id_brano int not null,
116    primary key(id_playlist,id_brano),
117    foreign key(id_playlist) references playlist(id_playlist),
118    foreign key(id_brano) references brano(id_brano)
119  );
```

```
mysql> describe contiene;
+-------------+---------+------+-----+---------+-------+
| Field       | Type    | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| id_playlist | int(11) | NO   | PRI | NULL    |       |
| id_brano    | int(11) | NO   | PRI | NULL    |       |
+-------------+---------+------+-----+---------+-------+
2 rows in set (0,00 sec)
```

Tabella 14. Scrive

```
121  create table scrive(
122    id_brano int not null,
123    id_autore int not null,
124    primary key(id_brano,id_autore),
125    foreign key(id_brano) references brano(id_brano),
126    foreign key(id_autore) references autore(id_autore)
127  );
```

```
mysql> describe scrive;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| id_brano  | int(11) | NO   | PRI | NULL    |       |
| id_autore | int(11) | NO   | PRI | NULL    |       |
+-----------+---------+------+-----+---------+-------+
2 rows in set (0,00 sec)
```

Tabella 15. Aggiunge

```
129  create table aggiunge(
130    id_utente int not null,
131    id_brano int not null,
132    primary key(id_utente,id_brano),
133    foreign key(id_brano) references brano(id_brano),
134    foreign key(id_utente) references utente(id_utente)
135  );
```

```
mysql> describe aggiunge;
+-----------+---------+------+-----+---------+-------+
| Field     | Type    | Null | Key | Default | Extra |
+-----------+---------+------+-----+---------+-------+
| id_utente | int(11) | NO   | PRI | NULL    |       |
| id_brano  | int(11) | NO   | PRI | NULL    |       |
+-----------+---------+------+-----+---------+-------+
2 rows in set (0,00 sec)
```

Tabella 16. SegueArt

```
137  create table segueart(
138    id_artista int not null,
139    id_utente int not null,
140    primary key(id_utente,id_artista),
141    foreign key(id_utente) references utente(id_utente),
142    foreign key(id_artista) references artista(id_artista)
143  );
```

```
mysql> describe segueart;
+------------+---------+------+-----+---------+-------+
| Field      | Type    | Null | Key | Default | Extra |
+------------+---------+------+-----+---------+-------+
| id_artista | int(11) | NO   | PRI | NULL    |       |
| id_utente  | int(11) | NO   | PRI | NULL    |       |
+------------+---------+------+-----+---------+-------+
2 rows in set (0,00 sec)
```

Tabella 17. Esibisce

```
145  create table esibisce(
146    codice_conc int not null,
147    id_artista int not null,
148    primary key(id_artista,codice_conc),
149    foreign key(codice_conc) references concerto(codice_conc),
150    foreign key(id_artista) references artista(id_artista)
151  );
```

```
mysql> describe esibisce;
+-------------+---------+------+-----+---------+-------+
| Field       | Type    | Null | Key | Default | Extra |
+-------------+---------+------+-----+---------+-------+
| codice_conc | int(11) | NO   | PRI | NULL    |       |
| id_artista  | int(11) | NO   | PRI | NULL    |       |
+-------------+---------+------+-----+---------+-------+
2 rows in set (0,00 sec)
```

Tabella 18. Composto

```
153  create table composto(
154    id_brano int not null,
155    id_album int not null,
156    primary key(id_brano,id_album),
157    foreign key(id_brano) references brano(id_brano),
158    foreign key(id_album) references album(id_album)
159  );
```

```
mysql> describe composto;
+----------+---------+------+-----+---------+-------+
| Field    | Type    | Null | Key | Default | Extra |
+----------+---------+------+-----+---------+-------+
| id_brano | int(11) | NO   | PRI | NULL    |       |
| id_album | int(11) | NO   | PRI | NULL    |       |
+----------+---------+------+-----+---------+-------+
2 rows in set (0,00 sec)
```

Tabella 19. Compra

```
create table compra(
  codice_prodotto int not null,
  id_utente int not null,
  tempo_acquisto datetime not null,
  primary key(codice_prodotto,id_utente,tempo_acquisto),
  foreign key(id_utente) references utente(id_utente),
  foreign key(codice_prodotto) references prodotto(codice_prodotto)
);
```

```
mysql> describe compra;
+-----------------+----------+------+-----+---------+-------+
| Field           | Type     | Null | Key | Default | Extra |
+-----------------+----------+------+-----+---------+-------+
| codice_prodotto | int(11)  | NO   | PRI | NULL    |       |
| id_utente       | int(11)  | NO   | PRI | NULL    |       |
| tempo_acquisto  | datetime | NO   | PRI | NULL    |       |
+-----------------+----------+------+-----+---------+-------+
3 rows in set (0,00 sec)
```

Inserimento dei Dati

L'inserimento dei dati all'interno del database avviene tramite codice da terminale.

Si utilizza il seguente comando:

```
INSERT INTO tabella (campo1,campo2) VALUES (value1,value2);
```

Delle **INSERT** verranno riportati solo pochi esempi per ciascuna tabella.

Per visualizzare gli inserimenti fatti si utilizza, invece, il seguente comando:

```
SELECT * FROM tabella;
```

Le tabelle hanno tutte centinaia di migliaia di elementi, quindi la **SELECT** di ogni tabella verrá limitata a 5 elementi, verrá poi riportata, per ciascuna di esse, la reale quantitá di record contenuti.

```sql
SELECT * FROM tabella limit 20;
SELECT count(*) from tabella;
```

Inserimenti Autore

```sql
INSERT INTO Autore VALUES (0,'Christopher Hess','femmina','Rockford','2013-04-23');
INSERT INTO Autore VALUES (1,'Iliana Allison','maschio','Jordan Valley','2013-05-11');
INSERT INTO Autore VALUES (2,'Chaim Bowman','femmina','Scottsdale','2010-11-17');
INSERT INTO Autore VALUES (3,'Ruth Parrish','maschio','Boulder City','2009-01-10');
INSERT INTO Autore VALUES (4,'Uriah Kramer','maschio','Springfield','2008-12-18');
```

```
mysql> select * from autore limit 5;
+-----------+------------------+---------+---------------+--------------+
| id_autore | nome_cognome     | sesso   | citta         | data_nascita |
+-----------+------------------+---------+---------------+--------------+
|         0 | Matthew Barry    | maschio | Torrington    | 2010-01-31   |
|         1 | Dolan Morton     | maschio | Pullman       | 2017-09-13   |
|         2 | Leo Shields      | maschio | Santa Clara   | 2010-02-16   |
|         3 | Madison Lindsey  | maschio | Hazleton      | 2013-09-24   |
|         4 | Baxter Castaneda | maschio | Lawrenceville | 2013-07-23   |
+-----------+------------------+---------+---------------+--------------+
```

```
mysql> select count(*) from autore;
+----------+
| count(*) |
+----------+
|   300000 |
+----------+
```

Tutti gli inserimenti sono stati generati utilizzando il programma Spawner che, dopo aver definito ogni attributo, genera casualmente record che rispettano le caratteristiche che sono state inserite nella definizione dell'attributo.

Inserimenti Artista

```
INSERT INTO artista VALUES (0,'Audra Ball',47560,'2019-10-06','2020-12-11','Orange');
INSERT INTO artista VALUES (1,'Anne Boyer',479809,'2019-03-28','2018-12-17','Yukon');
INSERT INTO artista VALUES (2,'Keefe Ramirez',479465,'2019-04-29','2021-04-15','Wheaton');
INSERT INTO artista VALUES (3,'Phillip Alvarado',272152,'2021-05-28','2020-04-24','Atwater');
INSERT INTO artista VALUES (4,'Dorian Wallace',235974,'2021-11-05','2021-01-20','Jeffersonville');
```

```
mysql> select * from artista limit 5;
+------------+-----------------+--------------+-----------------+---------------+-------------------+
| id_artista | alias           | num_follower | inizio_attivita | fine_attivita | luogo_provenienza |
+------------+-----------------+--------------+-----------------+---------------+-------------------+
|          0 | Joy Copeland    |       295350 | 2020-02-03      | 2020-05-23    | Carrollton        |
|          1 | Preston Carlson |       484907 | 2018-03-30      | 2019-03-25    | Vernon            |
|          2 | Chandler Morrow |       251114 | 2020-12-28      | 2020-05-21    | Burlington        |
|          3 | Chester Cortez  |        99684 | 2018-05-13      | 2021-07-31    | Spokane           |
|          4 | Signe Buckley   |       222722 | 2020-08-05      | 2020-10-14    | New London        |
+------------+-----------------+--------------+-----------------+---------------+-------------------+
```

```
mysql> mysql> count(*) from artista;
+----------+
| count(*) |
+----------+
|   300000 |
+----------+
```

Inserimenti Utente

```
INSERT INTO utente VALUES (0,'Forrest Hansen','2025-02-24','maschio','Jacksonville',73252,44071);
INSERT INTO utente VALUES (1,'Brock William','2018-11-24','femmina','Santa Monica',46748,14956);
INSERT INTO utente VALUES (2,'Kelly Albert','2023-07-30','femmina','Canton',37378,42923);
INSERT INTO utente VALUES (3,'Chelsea Lewis','2024-10-20','femmina','East Rutherford',53312,59644);
INSERT INTO utente VALUES (4,'Brynn Moses','2018-01-16','femmina','Saginaw',20117,60840);
```

```
mysql> select * from utente limit 5;
+-----------+---------------+--------------+---------+---------------+--------------+---------------+
| id_utente | nome_cognome  | data_nascita | sesso   | citta         | num_follower | num_following |
+-----------+---------------+--------------+---------+---------------+--------------+---------------+
|         0 | Porter Hunt   | 1988-12-01   | maschio | Laguna Beach  |        40964 |         65286 |
|         1 | Lucian Hammond | 1957-11-14   | femmina | Wynne         |         2620 |         23535 |
|         2 | Elaine Noel   | 2007-07-06   | maschio | Citrus Heights |        27231 |         40705 |
|         3 | Steel Fuentes | 1983-12-31   | femmina | Allentown     |        88416 |         44506 |
|         4 | Amity Everett | 1976-08-06   | femmina | Kingston      |        17082 |         92295 |
+-----------+---------------+--------------+---------+---------------+--------------+---------------+
```

```
mysql> select count(*) from utente;
+----------+
| count(*) |
+----------+
|   800000 |
+----------+
```

Inserimenti Abbonamento

```
INSERT INTO abbonamento VALUES (0,'2017-09-28','2004-10-18',752517);
INSERT INTO abbonamento VALUES (1,'2002-04-27','2017-05-06',722842);
INSERT INTO abbonamento VALUES (2,'2016-09-11','2004-08-11',422392);
INSERT INTO abbonamento VALUES (3,'2008-09-25','2003-10-06',449320);
INSERT INTO abbonamento VALUES (4,'2015-07-14','1999-06-01',562810);
```

```
mysql> select * from abbonamento limit 5;
+--------+-------------+------------+-----------+
| codice | data_inizio | data_fine  | id_utente |
+--------+-------------+------------+-----------+
|      0 | 2003-09-26  | 2002-07-18 |    533746 |
|      1 | 2009-02-11  | 2007-05-14 |    345488 |
|      2 | 1998-07-26  | 2001-12-28 |     73496 |
|      3 | 2016-10-19  | 1998-11-07 |    131828 |
|      4 | 1999-11-14  | 2016-02-20 |    387194 |
+--------+-------------+------------+-----------+
```

```
mysql> select count(*) from abbonamento;
+----------+
| count(*) |
+----------+
|   500000 |
+----------+
```

Inserimenti Playlist

```
INSERT INTO playlist VALUES (0,'luctus semper gravida congue',172,'00:01:54',67300,'2015-10-02','2006-09-16');
INSERT INTO playlist VALUES (1,'molestie netus viverra Etiam',300,'00:07:49',49202,'1999-01-09','2003-05-25');
INSERT INTO playlist VALUES (2,'nibh imperdiet vehicula nulla',315,'00:03:55',14163,'2017-02-21','1996-03-21');
INSERT INTO playlist VALUES (3,'parturient per tellus per',181,'00:02:49',53661,'2015-08-02','2005-01-07');
INSERT INTO playlist VALUES (4,'Etiam lectus rutrum bibendum',413,'00:05:08',63856,'1999-11-19','1999-12-30');
```

```
mysql> select * from playlist limit 5;
+-------------+---------------------------------+-----------+---------------+--------------+---------------+-----------------+
| id_playlist | nome                            | num_brani | durata_totale | num_follower | data_creazione | ultima_modifica |
+-------------+---------------------------------+-----------+---------------+--------------+---------------+-----------------+
|           0 | natoque bibendum Proin vestibulum |       431 | 00:04:51      |         2587 | 2007-05-05     | 1999-06-08      |
|           1 | lacinia massa Suspendisse varius  |       325 | 00:06:33      |        90989 | 1993-06-01     | 1998-10-03      |
|           2 | leo ante placerat rutrum          |       174 | 00:06:52      |        30679 | 2010-07-26     | 2016-10-03      |
|           3 | mi egestas nulla dolor            |       266 | 00:07:31      |        86948 | 1999-10-03     | 2013-12-20      |
|           4 | rutrum odio rutrum fringilla      |       429 | 00:07:04      |        38164 | 2009-08-25     | 2013-11-17      |
+-------------+---------------------------------+-----------+---------------+--------------+---------------+-----------------+
```

```
mysql> select count(*) from playlist;
+----------+
| count(*) |
+----------+
|   800000 |
+----------+
```

Inserimenti Album

```
INSERT INTO album VALUES (0,209745,'ligula gravida ornare est','2017-12-18',5,'00:02:58');
INSERT INTO album VALUES (1,119129,'magnis sociis leo sodales','2017-12-18',20,'00:03:49');
INSERT INTO album VALUES (2,131336,'senectus varius Vestibulum ultricies','2017-12-18',19,'00:02:40');
INSERT INTO album VALUES (3,210677,'est primis odio libero','2017-12-14',11,'00:06:52');
INSERT INTO album VALUES (4,152830,'velit litora Ut euismod','2017-12-20',16,'00:01:31');
```

```
mysql> select * from album limit 5;
+----------+------------+--------------------------------+---------------+-----------+----------+
| id_album | id_artista | titolo                         | pubblicazione | num_brani | durata   |
+----------+------------+--------------------------------+---------------+-----------+----------+
|        0 |      64828 | lacus consequat odio mauris    | 1937-04-09    |         3 | 00:08:07 |
|        1 |     246476 | lectus laoreet metus id        | 1950-10-12    |         4 | 00:08:03 |
|        2 |      28895 | euismod leo gravida massa      | 1973-02-28    |         7 | 00:04:16 |
|        3 |     204995 | sociis ad vehicula senectus    | 1923-12-20    |         6 | 00:04:54 |
|        4 |     139189 | penatibus Duis nonummy rhoncus | 1947-05-27    |        17 | 00:06:04 |
+----------+------------+--------------------------------+---------------+-----------+----------+
```

```
mysql> select count(*) from album;
+----------+
| count(*) |
+----------+
|   500000 |
+----------+
```

Inserimenti Brano

```
INSERT INTO brano VALUES (0,'torquent montes nulla sollicitudin','00:06:04','Musica tradizionale',67508,220458);
INSERT INTO brano VALUES (1,'mollis cubilia Donec porta','00:06:09','Folk',15801,55960);
INSERT INTO brano VALUES (2,'montes Donec urna malesuada','00:01:55','Breakbeat',16277,38807);
INSERT INTO brano VALUES (3,'vitae dis primis orci','00:02:17','Rap',77119,25144);
INSERT INTO brano VALUES (4,'ullamcorper bibendum dignissim viverra','00:01:50','Rock elettronico',66349,32163);
```

```
mysql> select * from brano limit 5;
+----------+-----------------------------------+----------+--------------------+-------------+------------+
| id_brano | titolo                            | durata   | genere             | num_ascolti | id_artista |
+----------+-----------------------------------+----------+--------------------+-------------+------------+
|        0 | facilisis ultrices Cum Ut         | 00:01:42 | Musica da film     |       86951 |      97338 |
|        1 | at tortor dapibus varius          | 00:03:41 | Musica pop         |       93852 |      12699 |
|        2 | rhoncus facilisis per pharetra    | 00:04:13 | Folk               |       91561 |     245171 |
|        3 | Vivamus condimentum Quisque Mauris | 00:05:45 | Musica elettronica |       61012 |     275362 |
|        4 | molestie mattis gravida venenatis | 00:01:12 | Funk               |       96254 |     158998 |
+----------+-----------------------------------+----------+--------------------+-------------+------------+
```

```
mysql> select count(*) from brano;
+----------+
| count(*) |
+----------+
|  1000000 |
+----------+
```

Inserimenti Prodotto

```
INSERT INTO prodotto VALUES (0,'libero et libero sagittis',14,25182);
INSERT INTO prodotto VALUES (1,'vehicula primis accumsan erat',97,240060);
INSERT INTO prodotto VALUES (2,'tortor in in Maecenas',58,287477);
INSERT INTO prodotto VALUES (3,'fames lorem natoque imperdiet',58,136660);
INSERT INTO prodotto VALUES (4,'suscipit Integer nascetur ullamcorper',41,77120);
```

```
mysql> select * from prodotto limit 5;
+-----------------+---------------------------------------+--------+------------+
| codice_prodotto | nome_prodotto                         | prezzo | id_artista |
+-----------------+---------------------------------------+--------+------------+
|               0 | euismod faucibus sem mauris           |     23 |     246621 |
|               1 | consectetuer nulla pulvinar placerat  |     75 |     196788 |
|               2 | hendrerit varius Donec dapibus        |      3 |     116191 |
|               3 | risus consequat Pellentesque arcu     |     81 |     187673 |
|               4 | vehicula lobortis ligula leo          |     36 |     169329 |
+-----------------+---------------------------------------+--------+------------+
```

```
mysql> select count(*) from prodotto;
+----------+
| count(*) |
+----------+
|   200000 |
+----------+
```

Inserimenti Concerto

```
INSERT INTO concerto VALUES (0,'ut cubilia ligula sodales','Pasco','9408 North College Station Blvd.','2018-12-29 09:46:13');
INSERT INTO concerto VALUES (1,'est orci fames interdum','Somersworth','6088 West British Indian Ocean Territory Way','2018-07-01 19:42:19');
INSERT INTO concerto VALUES (2,'placerat nisl eget lectus','Mankato','56200 West Timor-leste Ln.','2019-04-16 22:19:33');
INSERT INTO concerto VALUES (3,'et suscipit Lorem ad','Wahoo','70360 East Dominica Way','2019-05-11 12:06:17');
INSERT INTO concerto VALUES (4,'eget aptent dui scelerisque','Palos Verdes Estates','58387 South Buena Park Ln.','2019-01-24 16:19:25');
```

```
mysql> select * from concerto limit 5;
+-------------+----------------------------------+-----------+-----------------------------------------+---------------------+
| codice_conc | nome_conc                        | citta     | indirizzo                               | dataora_conc        |
+-------------+----------------------------------+-----------+-----------------------------------------+---------------------+
|           0 | montes ac pharetra litora        | Hoboken   | 73387 East Greece Ct.                   | 2019-10-17 06:00:28 |
|           1 | magnis hendrerit eleifend suscipit | Canton    | 31728 East British Indian Ocean Territory Way | 2018-05-10 12:15:14 |
|           2 | adipiscing Duis odio natoque     | Hoboken   | 41103 South Birmingham Ct.              | 2017-12-30 06:53:59 |
|           3 | convallis dis a at               | Providence | 92639 West Carson Ave.                  | 2018-02-13 01:38:09 |
|           4 | urna conubia et Fusce            | Roswell   | 48684 North Guadeloupe Way              | 2018-05-06 14:26:37 |
+-------------+----------------------------------+-----------+-----------------------------------------+---------------------+
```

```
mysql> select count(*) from concerto;
+----------+
| count(*) |
+----------+
|   200000 |
+----------+
```

Inserimenti Biglietto

```
1  INSERT INTO biglietto VALUES (0,'C4','w',48,719481,101420);
2  INSERT INTO biglietto VALUES (1,'7th','S',50,542387,152990);
3  INSERT INTO biglietto VALUES (2,'Pyd','a',2,670581,137852);
4  INSERT INTO biglietto VALUES (3,'17VR','a',28,487152,109147);
5  INSERT INTO biglietto VALUES (4,'oRSR','q',37,63701,77733);
```

```
mysql> select * from biglietto limit 5;
+-----------------+---------+------+-------+-----------+-------------+
| codice_biglietto | settore | fila | posto | id_utente | codice_conc |
+-----------------+---------+------+-------+-----------+-------------+
|               0 | f0x     | N    |     4 |    492185 |      176823 |
|               1 | mL      | n    |    31 |    725166 |      117842 |
|               2 | E       | G    |    31 |    302382 |      142913 |
|               3 | 4       | v    |    32 |    336748 |       57671 |
|               4 | 8W      | H    |     3 |    754551 |       21970 |
+-----------------+---------+------+-------+-----------+-------------+
```

```
mysql> select count(*) from biglietto;
+----------+
| count(*) |
+----------+
|   200000 |
+----------+
```

Inserimenti SegueUt

```
1  INSERT INTO segueut VALUES (67597,222700);
2  INSERT INTO segueut VALUES (285940,413550);
3  INSERT INTO segueut VALUES (509219,168385);
4  INSERT INTO segueut VALUES (778756,351396);
5  INSERT INTO segueut VALUES (629668,445097);
```

```
mysql> select * from segueut limit 5;
+-----------+-----------+
| id_utente1 | id_utente2 |
+-----------+-----------+
|    573637 |          1 |
|    515086 |          4 |
|    326212 |          7 |
|    558263 |          8 |
|    163053 |         10 |
+-----------+-----------+
```

```
mysql> select count(*) from segueut;
+----------+
| count(*) |
+----------+
|   800000 |
+----------+
```

Inserimenti Crea

```
1  INSERT INTO crea VALUES (33592,545212);
2  INSERT INTO crea VALUES (554725,355229);
3  INSERT INTO crea VALUES (154569,180229);
4  INSERT INTO crea VALUES (555470,583331);
5  INSERT INTO crea VALUES (121918,304483);
```

```
mysql> select * from crea limit 5;
+-----------+-------------+
| id_utente | id_playlist |
+-----------+-------------+
|    283588 |           0 |
|    292196 |           0 |
|     22154 |           1 |
|    159461 |           1 |
|    178671 |           1 |
+-----------+-------------+
```

```
mysql> select count(*) from crea;
+----------+
| count(*) |
+----------+
|   800000 |
+----------+
```

Inserimenti Contiene

```
1  INSERT INTO contiene VALUES (685114,187674);
2  INSERT INTO contiene VALUES (413884,387220);
3  INSERT INTO contiene VALUES (191592,665587);
4  INSERT INTO contiene VALUES (572389,58909);
5  INSERT INTO contiene VALUES (598570,638475);
```

```
mysql> select * from contiene limit 5;
+-------------+----------+
| id_playlist | id_brano |
+-------------+----------+
|      712978 |        0 |
|      752022 |        0 |
|      442480 |        1 |
|      534629 |        1 |
|       61364 |        2 |
+-------------+----------+
```

```
mysql> select count(*) from contiene;
+----------+
| count(*) |
+----------+
|  1000000 |
+----------+
```

Inserimenti Scrive

```
1  INSERT INTO scrive VALUES (387218,172030);
2  INSERT INTO scrive VALUES (99688,55255);
3  INSERT INTO scrive VALUES (248444,215970);
4  INSERT INTO scrive VALUES (638531,73883);
5  INSERT INTO scrive VALUES (588935,272368);
```

```
mysql> select * from scrive limit 5;
+----------+-----------+
| id_brano | id_autore |
+----------+-----------+
|   345366 |         0 |
|   479012 |         0 |
|   685543 |         0 |
|   900466 |         0 |
|   992349 |         0 |
+----------+-----------+
```

```
mysql> select count(*) from scrive;
+----------+
| count(*) |
+----------+
|   900000 |
+----------+
```

Inserimenti Aggiunge

```
1  INSERT INTO aggiunge VALUES (731543,980898);
2  INSERT INTO aggiunge VALUES (70004,675008);
3  INSERT INTO aggiunge VALUES (530357,619236);
4  INSERT INTO aggiunge VALUES (646821,78793);
5  INSERT INTO aggiunge VALUES (269595,549356);
```

```
mysql> select * from aggiunge limit 5;
+-----------+----------+
| id_utente | id_brano |
+-----------+----------+
|    109135 |        0 |
|    461528 |        1 |
|    540908 |        2 |
|      2863 |        4 |
|    416576 |        4 |
+-----------+----------+
```

```
mysql> select count(*) from aggiunge;
+----------+
| count(*) |
+----------+
|   900000 |
+----------+
```

Inserimenti SegueArt

```
1  INSERT INTO segueart VALUES (80011,709386);
2  INSERT INTO segueart VALUES (83822,569094);
3  INSERT INTO segueart VALUES (265056,347380);
4  INSERT INTO segueart VALUES (135762,191613);
5  INSERT INTO segueart VALUES (52676,351581);
```

```
mysql> select * from segueart limit 5;
+------------+-----------+
| id_artista | id_utente |
+------------+-----------+
|          0 |    730673 |
|          2 |    227955 |
|          3 |    349017 |
|          3 |    592818 |
|          3 |    799168 |
+------------+-----------+
```

```
mysql> select count(*) from segueart;
+----------+
| count(*) |
+----------+
|   500000 |
+----------+
```

Inserimenti Esibisce

```
1  INSERT INTO esibisce VALUES (4979,262265);
2  INSERT INTO esibisce VALUES (56746,24475);
3  INSERT INTO esibisce VALUES (29086,252849);
4  INSERT INTO esibisce VALUES (82171,224168);
5  INSERT INTO esibisce VALUES (185222,246842);
```

```
mysql> select * from esibisce limit 5;
+-------------+------------+
| codice_conc | id_artista |
+-------------+------------+
|           1 |      88463 |
|           1 |      90112 |
|           1 |     136028 |
|           3 |      64080 |
|           8 |     114243 |
+-------------+------------+
```

```
mysql> select count(*) from esibisce;
+----------+
| count(*) |
+----------+
|   200000 |
+----------+
```

Inserimenti Composto

```
1  INSERT INTO composto VALUES (90851,181735);
2  INSERT INTO composto VALUES (728645,67082);
3  INSERT INTO composto VALUES (578839,137319);
4  INSERT INTO composto VALUES (253130,182390);
5  INSERT INTO composto VALUES (164213,220904);
```

```
mysql> select * from composto limit 5;
+----------+----------+
| id_brano | id_album |
+----------+----------+
|   212001 |        0 |
|   273462 |        0 |
|   288512 |        0 |
|   336099 |        0 |
|   530002 |        0 |
+----------+----------+
```

```
mysql> select count(*) from composto;
+----------+
| count(*) |
+----------+
|   900000 |
+----------+
```

Inserimenti Compra

```
1  INSERT INTO compra VALUES (173995,562500);
2  INSERT INTO compra VALUES (9756,665925);
3  INSERT INTO compra VALUES (106520,173814);
4  INSERT INTO compra VALUES (130637,443841);
5  INSERT INTO compra VALUES (155972,706253);
```

```
mysql> select * from compra limit 5;
+----------------+----------+---------------------+
| codice_prodotto | id_utente | tempo_acquisto      |
+----------------+----------+---------------------+
|          16417 |        0 | 2017-12-21 17:08:12 |
|          34146 |        0 | 2017-12-21 17:08:10 |
|          41079 |        0 | 2017-12-21 17:08:12 |
|         166298 |        0 | 2017-12-21 17:08:10 |
|         104096 |        1 | 2017-12-21 17:08:11 |
+----------------+----------+---------------------+
```

```
mysql> select count(*) from compra;
+----------+
| count(*) |
+----------+
|   500000 |
+----------+
```

Trigger

A partire dalla versione 5.0.2, MySQL, ha introdotto i Trigger, ovvero un meccanismo attraverso il quale è possibile automatizzare tali operazioni al verificarsi di eventi riguardanti i dati, quali **INSERT**, **UPDATE** o **DELETE**.

Quando definiamo un Trigger in MySQL dobbiamo stabilire se questo debba innescarsi prima o dopo un certo evento.

Potremo quindi definire dei Trigger per queste diverse combinazioni di tempo/evento:

- BEFORE INSERT
- BEFORE UPDATE
- BEFORE DELETE
- AFTER INSERT
- AFTER UPDATE
- AFTER DELETE

Ciascun Trigger deve necessariamente essere associato ad una tabella.

La sintassi base di un Trigger è la seguente.

```
delimiter//
CREATE TRIGGER nome_trigger
  tempistica evento ON nome_tabella
  FOR EACH ROW
BEGIN

  ...
  ...
  ...

END//
```

Trigger 1

```
delimiter //
create trigger regola_abbonamento
  before insert on abbonamento for each row
  begin
  if exists (
    select abbonamento.id_utente, abbonamento.data_fine, abbonamento.data_inizio
    from abbonamento
    where new.id_utente = abbonamento.id_utente
    and (
      (
        new.data_inizio > abbonamento.data_inizio
        and new.data_inizio < abbonamento.data_fine
      ) or (
        new.data_fine > abbonamento.data_inizio
        and new.data_fine < abbonamento.data_fine
      )
    )
  )
  then
  signal sqlstate'45000'
  set message_text ='per uno stesso utente, non ci possono essere due abbonamenti sovrapposti';
  end if;
end //
```

Questo trigger impedisce ai record della tabella **Abbonamento** di avere gli intervalli di tempo sovrapposti per uno stesso utente. In pratica, per l'utente x, non ci puo' essere un abbonamento che inizia prima che finisca quello precedente e viceversa.

Il trigger viene applicato per ogni record prima di un inserimento; nel caso in cui dovesse avvenire un inserimento simile allora quello verrebbe rigettato con quello stato di errore.

Trigger 2

```
delimiter //
create trigger update_data_playlist
  before insert on playlist for each row
  begin
  if (
    new.ultima_modifica < new.data_creazione
  )
  then
  signal sqlstate'45000'
  set message_text ='una playlist non puó essere modificata prima di essere creata';
  end if;
end //
```

Trigger 3

```
delimiter //
create trigger insert_regola_playlist
  before insert on contiene for each row
  begin
  declare a time;
  declare b time;
  declare c time;
  declare d int;
  set a = (
    select brano.durata
    from brano
    where brano.id_brano = new.id_brano
  );
  set b = (
    select playlist.durata_totale
    from playlist
    where playlist.id_playlist = new.id_playlist
  );
  set c = a + b;
  update playlist
    set playlist.durata_totale = c
    where new.id_playlist = playlist.id_playlist;
  update playlist
    set playlist.ultima_modifica = date(now())
    where new.id_playlist = playlist.id_playlist;
  update playlist
    set playlist.num_brani = playlist.num_brani + 1
    where new.id_playlist = playlist.id_playlist;
end //
```

Questo trigger ha un utilizzo differente dai due precedenti.

Si applica sulla tabella **Contiene**, la relazione che lega i brani e le playlist.

Il trigger si occupa di 3 operazioni:

1. quando un brano viene aggiunto a una playlist, quest'ultima aumenta la sua durata totale della durata del brano aggiunto;
2. alla playlist viene aggiornata la data "ultima_modifica" con quella corrente;
3. viene incrementato il campo "num_brani" di 1.

Trigger 4

```
delimiter //
create trigger delete_regola_playlist
  before delete on contiene for each row
  begin
  declare a time;
  declare b time;
  declare c time;
  set a = (
    select brano.durata
    from brano
    where brano.id_brano = old.id_brano
  );
  set b = (
    select playlist.durata_totale
    from playlist
    where playlist.id_playlist = old.id_playlist
  );
  set c = b - a;
  update playlist
    set playlist.durata_totale = c
    where old.id_playlist = playlist.id_playlist;
  update playlist
    set playlist.ultima_modifica = date(now())
    where old.id_playlist = playlist.id_playlist;
  update playlist
    set playlist.num_brani = playlist.num_brani - 1
    where old.id_playlist = playlist.id_playlist;
end //
```

Il funzionamento è inverso al trigger precedente, in quanto questo viene applicato dopo l'eliminazione di un brano dalla playlist.

Trigger 5

```
266  delimiter //
267  create trigger add_count_follow_artista
268    before insert on segueart for each row
269    begin
270    update artista
271      set artista.num_follower = artista.num_follower + 1
272      where new.id_artista = artista.id_artista;
273      end //
```

Trigger 6

```
275  delimiter //
276  create trigger del_count_follow_artista
277    before delete on segueart for each row
278    begin
279    update artista
280      set artista.num_follower = artista.num_follower - 1
281      where old.id_artista = artista.id_artista;
282      end //
```

I trigger 5 e 6 servono a tenere il conto del numero di follower di un artista, tramite la relazione **segueArt**.

Lo stesso funzionamento avranno i trigger successivi ma per tabelle differenti.

Trigger 7

```
delimiter //
create trigger add_count_follow_utente
  before insert on segueut for each row
  begin
  if new.id_utente1 = new.id_utente2
  then
  signal sqlstate'45000'
  set message_text ='Un utente non puó seguire se stesso.';
  end if;
  update utente
    set utente.num_following = utente.num_following + 1
    where new.id_utente1 = utente.id_utente;
  update utente
    set utente.num_follower = utente.num_follower + 1
    where new.id_utente2 = utente.id_utente;
    end //
```

Trigger 8

```
delimiter //
create trigger del_count_follow_utente
  before delete on segueut for each row
  begin
  update utente
    set utente.num_following = utente.num_following - 1
    where old.id_utente1 = utente.id_utente;
  update utente
    set utente.num_follower = utente.num_follower - 1
    where old.id_utente2 = utente.id_utente;
    end //
```

Trigger 9

```
313  delimiter //
314  create trigger add_num_ascolti
315    before insert on aggiunge for each row
316    begin
317    update brano
318      set brano.num_ascolti = brano.num_ascolti + 1
319      where new.id_brano = brano.id_brano;
320      end //
```

Trigger 10

```
322  delimiter //
323  create trigger regola_artista
324    before insert on artista for each row
325    begin
326    if new.inizio_attivita > new.fine_attivita
327    then
328    signal sqlstate'45000'
329    set message_text ='la carriera di un artista non puó finire prima di cominciare';
330    end if;
331  end //
```

Trigger 11

```
delimiter //
create trigger composto_album_add
  before insert on composto for each row
  begin
  declare a time;
  declare b time;
  declare c time;
  set a = (
    select brano.durata
    from brano
    where brano.id_brano = new.id_brano
  );
  set b = (
    select album.durata
    from album
    where album.id_album = new.id_album
  );
  set c = a + b;
  update album
    set album.num_brani = album.num_brani + 1
    where new.id_album = album.id_album;
  update album
    set album.durata = c
    where new.id_album = album.id_album;
    end //
```

Trigger 12

```
delimiter //
create trigger composto_album_del
  before delete on composto for each row
  begin
  declare a time;
  declare b time;
  declare c time;
  set a = (
    select brano.durata
    from brano
    where brano.id_brano = old.id_brano
  );
  set b = (
    select album.durata
    from album
    where album.id_album = old.id_album
  );
  set c = b - a;
  update album
    set album.num_brani = album.num_brani - 1
    where old.id_album = album.id_album;
  update album
    set album.durata = c
    where old.id_album = album.id_album;
    end //
```

Utilizzo del Database (Query)

Tramite le **Query** andremo ad interrogare il nostro Database. Sfruttando il comando **SELECT** che abbiamo già visto precedentemente, arricchendolo però di vincoli e unioni che andremo a vedere nelle diverse Query che porremo.

Query 1

Selezionare NOME e COGNOME degli UTENTI che seguono un ARTISTA che si è ESIBITO nella loro stessa CITTÁ.

```
1  select utente.nome_cognome, utente.citta, artista.alias, concerto.nome_conc
2  from utente, artista, concerto, segueart, esibisce
3  where utente.id_utente = segueart.id_utente
4  and segueart.id_artista = artista.id_artista
5  and artista.id_artista = esibisce.id_artista
6  and esibisce.codice_conc = concerto.codice_conc
7  and concerto.citta = utente.citta;
```

```
+--------------------+------------------+---------------------+-------------------------------------------+
| nome_cognome       | citta            | alias               | nome_conc                                 |
+--------------------+------------------+---------------------+-------------------------------------------+
| Ariel Chambers     | Columbia         | Gregory Dunn        | tempor cursus bibendum Nunc               |
| Garrett Villarreal | Cape May         | Zahir Moses         | taciti enim orci scelerisque              |
| Stephen Terry      | Coral Springs    | Jack Glover         | magna dignissim euismod Nunc              |
| Sonia Franks       | Montpelier       | Nadine Gutierrez    | vulputate facilisi feugiat Nulla          |
| Plato Horne        | Rochester        | Cheyenne Campbell   | Vivamus nibh Suspendisse Cras             |
| Palmer Lane        | Statesboro       | Abbot Richard       | eleifend sociosqu ut dignissim            |
| Cherokee Slater    | Hawaiian Gardens | Nicholas Potter     | venenatis porta Etiam sollicitudin        |
| Zena Cunningham    | New Brunswick    | Ori Clemons         | libero iaculis senectus semper            |
| Rhonda Dorsey      | Seaford          | Leroy Garcia        | ultricies sagittis Nullam sem             |
| Barclay Erickson   | Agawam           | Fritz Lancaster     | placerat netus ut montes                  |
| Allen Paul         | Jordan Valley    | Dalton Case         | conubia malesuada lobortis Fusce          |
| Perry Bowers       | Merced           | Vaughan Banks       | faucibus lacinia dignissim dolor          |
| Mufutau Patton     | Aspen            | Magee Landry        | aliquet suscipit justo mattis             |
+--------------------+------------------+---------------------+-------------------------------------------+
353 rows in set (5,51 sec)
```

Questo è il risultato della Query e in basso tra parentesi tonde () troviamo il tempo in cui il Database è riuscito a rispondere alla nostra richiesta.

Poiché questa query restituisce molti record (353) mostriamo al lettore soltanto l'inizio e la fine del risultato ottenuto.

Query 2

Selezionare NOME e COGNOME degli UTENTI che hanno AGGIUNTO un brano scritto dall'AUTORE Timon Kent ed è contenuto in una PLAYLIST con minimo 50 brani.

```
25  select utente.nome_cognome, utente.id_utente
26  from (
27    select playlist.id_playlist
28    from playlist
29    where playlist.num_brani >= 50
30  )as play, (
31    select brano.id_brano
32    from brano,scrive,autore
33    where brano.id_brano = scrive.id_brano
34    and scrive.id_autore = autore.id_autore
35    and autore.nome_cognome = 'Timon Kent'
36  ) as aut, utente, contiene, aggiunge
37  where utente.id_utente = aggiunge.id_utente
38  and aggiunge.id_brano = aut.id_brano
39  and aut.id_brano = contiene.id_brano
40  and contiene.id_playlist = play.id_playlist
41  group by utente.id_utente;
```

```
+-----------------+------------+
| nome_cognome    | id_utente  |
+-----------------+------------+
| Abel Albert     |      32177 |
| Acton Galloway  |     218533 |
| Griffin Noble   |     244230 |
+-----------------+------------+
3 rows in set (0,17 sec)
```

Query 3

Selezionare NOME e COGNOME degli UTENTI e quanto hanno speso in totale in PRODOTTI.

```
19  select utente.nome_cognome,sum(prodotto.prezzo)
20  from utente, compra, prodotto
21  where utente.id_utente = compra.id_utente
22  and compra.codice_prodotto = prodotto.codice_prodotto
23  group by utente.id_utente;
```

```
+------------------+-----------------------+
| nome_cognome     | sum(prodotto.prezzo)  |
+------------------+-----------------------+
| Elaine Noel      |                    22 |
| Steel Fuentes    |                    60 |
| Amity Everett    |                    56 |
| Phoebe Hayden    |                    94 |
| Keefe Everett    |                    74 |
| Deanna Gross     |                    91 |
```

```
| Clayton Padilla          |                    18 |
| Perry Scott              |                     3 |
| Brianna Harper           |                    91 |
| Marny Buchanan           |                    89 |
| Shad Olson               |                    63 |
| Rebekah Clements         |                   188 |
| Rahim Barrera            |                    62 |
| Noble Mills              |                   100 |
+--------------------------+-----------------------+
486711 rows in set (8,35 sec)
```

Query 4

Selezionare NOME e COGNOME degli UTENTI che hanno acquistato un ABBONAMENTO ed un BIGLIETTO per un CONCERTO nello stesso periodo.

```
43  select utente.nome_cognome, concerto.nome_conc, concerto.dataora_conc,
44  abbonamento.data_inizio, abbonamento.data_fine
45  from utente,abbonamento,biglietto,concerto
46  where abbonamento.id_utente = utente.id_utente
47  and biglietto.id_utente = utente.id_utente
48  and biglietto.codice_conc = concerto.codice_conc
49  and DATE(concerto.dataora_conc) <= abbonamento.data_fine
50  and DATE(concerto.dataora_conc) >= abbonamento.data_inizio;
```

```
+--------------+-------------------------+---------------------+-------------+------------+
| nome_cognome | nome_conc               | dataora_conc        | data_inizio | data_fine  |
+--------------+-------------------------+---------------------+-------------+------------+
| April Cantu  | nulla arcu luctus varius | 2017-12-16 11:56:49 | 1999-12-11  | 2017-12-17 |
+--------------+-------------------------+---------------------+-------------+------------+
1 row in set (7,40 sec)
```

Query 5

Selezionare NOME e COGNOME degli UTENTI abbonati nel 2016 e tutti gli ABBONAMENTI rispettivi.

```
52  select utente.nome_cognome, abbonamento.data_inizio, abbonamento.data_fine
53  from utente, abbonamento
54  where utente.id_utente = abbonamento.id_utente
55  and (
56    YEAR(abbonamento.data_inizio) = 2016
57    or YEAR(abbonamento.data_fine) = 2016
58  );
```

```
+-----------------+-------------+------------+
| nome_cognome    | data_inizio | data_fine  |
+-----------------+-------------+------------+
| Berk Dixon      | 2016-10-19  | 1998-11-07 |
| Laura Wallace   | 1999-11-14  | 2016-02-20 |
| Juliet Wilcox   | 2016-10-20  | 2003-11-07 |
| Kirsten Barrett | 2016-08-11  | 2011-02-23 |
```

```
| Erich Lane            | 2016-01-15  | 2015-02-18 |
| Mariko Guerra         | 2016-04-15  | 1998-04-11 |
| Virginia Harper       | 2012-07-29  | 2016-07-24 |
| Graiden Blair         | 2002-11-03  | 2016-01-04 |
+-----------------------+-------------+------------+
48774 rows in set (1,00 sec)
```

Query 6

Selezionare gli UTENTI che SEGUONO altri UTENTI i quali hanno creato una PLAYLIST dalla DURATA di almeno 7 minuti.

```
60  select uno.nome_cognome, due.nome_cognome, playlist.nome, playlist.durata_totale
61  from utente uno, utente due, playlist, crea, segueut
62  where uno.id_utente = segueut.id_utente1
63  and segueut.id_utente2 = due.id_utente
64  and due.id_utente = crea.id_utente
65  and crea.id_playlist = playlist.id_playlist
66  and playlist.durata_totale >= '00:07:00';
```

```
+------------------+------------------+------------------------------+----------------+
| nome_cognome     | nome_cognome     | nome                         | durata_totale  |
+------------------+------------------+------------------------------+----------------+
| Giacomo Parsons  | Dalton Mcmillan | montes fames pulvinar Class | 00:07:06       |
| Amena Brown      | Rose Clemons     | montes fames pulvinar Class | 00:07:06       |
| Amos Sims        | Xavier Carney    | nonummy Maecenas Ut magna   | 00:07:03       |
| Halla Golden     | Keefe Whitney    | posuere sit ullamcorper Cum | 00:07:37       |
| Helen Miranda    | Keefe Whitney    | posuere sit ullamcorper Cum | 00:07:37       |
| Skyler Castro    | Xandra Drake     | lectus conubia Nunc ullamcorper | 00:07:50   |
| Lucian Faulkner  | Yeo Clements     | ullamcorper elit nostra dictum | 00:07:45    |
| Oren Caldwell    | Yeo Clements     | ullamcorper elit nostra dictum | 00:07:45    |
| Magee Park       | Ira Wagner       | justo bibendum augue vulputate | 00:07:32    |
| Paula Rivera     | Lionel Odonnell  | natoque et penatibus porttitor | 00:07:31    |
+------------------+------------------+------------------------------+----------------+
128339 rows in set (4,87 sec)
```

Query 7

Selezionare le PLAYLIST CREATE da UTENTI che SEGUONO altri UTENTI i quali hanno sottoscritto un ABBONAMENTO valido nel 2015.

```
68  select playlist.nome
69  from utente uno, utente due, playlist, crea, abbonamento, segueut
70  where playlist.id_playlist = crea.id_playlist
71  and crea.id_utente = uno.id_utente
72  and uno.id_utente = segueut.id_utente1
73  and segueut.id_utente2 = due.id_utente
74  and due.id_utente = abbonamento.id_utente
75  and (
76    YEAR(abbonamento.data_inizio) = 2015
77    or YEAR(abbonamento.data_fine) = 2015
78  );
```

```
+-----------------------------------+
| nome                              |
+-----------------------------------+
| Aenean risus molestie fermentum   |
| augue et dignissim faucibus       |
| cursus cursus pretium ornare      |
| facilisis Lorem netus sit         |
| Pellentesque sodales libero morbi |
| Maecenas vestibulum commodo Curabitur |
+-----------------------------------+
49153 rows in set (4,90 sec)
```

Query 8

Selezionare le PLAYLIST che CONTENGONO BRANI di un ARTISTA che si è ESIBITO nel Canton.

```
80  select y.nome, x.alias, x.nome_conc
81  from (
82    select concerto.nome_conc, artista.alias, artista.id_artista
83    from concerto, esibisce, artista
84    where artista.id_artista = esibisce.id_artista
85    and esibisce.codice_conc = concerto.codice_conc
86    and concerto.citta = 'Canton'
87  ) as x, (
88    select playlist.nome, artista.id_artista
89    from playlist, contiene, brano, artista
90    where playlist.id_playlist = contiene.id_playlist
91    and contiene.id_brano = brano.id_brano
92    and brano.id_artista = artista.id_artista
93  ) as y
94  where y.id_artista = x.id_artista;
```

```
+----------------------------------------+---------------+-----------------------------------+
| nome                                   | alias         | nome_conc                         |
+----------------------------------------+---------------+-----------------------------------+
| Mauris fermentum porttitor dapibus     | Harrison Rush | magnis hendrerit eleifend suscipit |
| non vehicula aliquet pretium           | Harrison Rush | magnis hendrerit eleifend suscipit |
| justo litora fringilla condimentum     | Harrison Rush | magnis hendrerit eleifend suscipit |
| porttitor fames tellus Pellentesque    | Harrison Rush | magnis hendrerit eleifend suscipit |
| leo consequat congue purus             | Harrison Rush | magnis hendrerit eleifend suscipit |
| fringilla sed sit molestie             | Joan Pugh     | nascetur tortor adipiscing Sed    |
| gravida ante tortor Phasellus          | Bert Fuller   | nascetur tortor adipiscing Sed    |
| facilisis iaculis nec nonummy          | Bert Fuller   | nascetur tortor adipiscing Sed    |
| Quisque Proin ullamcorper Cum          | Bert Fuller   | nascetur tortor adipiscing Sed    |
| Phasellus purus laoreet pede           | Bert Fuller   | nascetur tortor adipiscing Sed    |
| scelerisque sed conubia sit            | Risa Stevens  | scelerisque id malesuada dictum   |
| ipsum magnis varius Praesent           | Risa Stevens  | scelerisque id malesuada dictum   |
| luctus penatibus sagittis aliquam      | Risa Stevens  | scelerisque id malesuada dictum   |
+----------------------------------------+---------------+-----------------------------------+
689 rows in set (0,24 sec)
```

Query 9

Selezionare le PLAYLIST CREATE da UTENTI maschi con esattamente 100000 follower.

```
 96  select playlist.nome, utente.nome_cognome, utente.num_follower
 97  from playlist, utente, crea
 98  where playlist.id_playlist = crea.id_playlist
 99  and crea.id_utente = utente.id_utente
100  and utente.sesso = 'maschio'
101  and utente.num_follower = 100000;
```

```
+-------------------------------+-------------------+--------------+
| nome                          | nome_cognome      | num_follower |
+-------------------------------+-------------------+--------------+
| consequat In Donec condimentum | Pandora Nash      |       100000 |
| pharetra velit quis senectus  | Pandora Nash      |       100000 |
| nulla lorem cursus ipsum      | Chester Blanchard |       100000 |
| cursus cubilia congue per     | Forrest Dillard   |       100000 |
| purus quam lacinia Cum        | Forrest Dillard   |       100000 |
| velit ridiculus sed porta     | Lavinia Lynn      |       100000 |
+-------------------------------+-------------------+--------------+
6 rows in set (0,71 sec)
```

Query 10

Selezionare le PLAYLIST CREATE da UTENTI che seguono un artista con la stessa cittá di provenienza.

```
103  select playlist.nome, utente.nome_cognome, artista.alias, utente.citta
104  from playlist, crea, utente, segueart, artista
105  where playlist.id_playlist = crea.id_playlist
106  and crea.id_utente = utente.id_utente
107  and utente.id_utente = segueart.id_utente
108  and segueart.id_artista = artista.id_artista
109  and artista.luogo_provenienza = utente.citta;
```

```
+-----------------------------------------+---------------------+---------------------+-------------------------+
| nome                                    | nome_cognome        | alias               | citta                   |
+-----------------------------------------+---------------------+---------------------+-------------------------+
| ridiculus enim risus nonummy            | Alfonso Gibbs       | Flynn Velasquez     | Hartford                |
| mattis Sed diam semper                  | Aretha Mclaughlin   | Lewis Blanchard     | Yuma                    |
| cubilia Proin ac non                    | Derek Woods         | Brynne Hull         | Torrance                |
| vel cursus cursus malesuada             | Tanner Macdonald    | TaShya Chapman      | Chester                 |
| Quisque varius venenatis est            | Adrian Wise         | Brittany Roberts    | Mount Vernon            |
| leo semper lorem scelerisque            | Ivan Kirk           | Colleen Blair       | Juneau                  |
| est nunc nibh sem                       | Winter Moore        | Hedley Munoz        | Lander                  |
| libero hendrerit Donec euismod          | Nathaniel Gallegos  | Dane Stout          | Westminster             |
| in lobortis condimentum cubilia         | Nathaniel Gallegos  | Dane Stout          | Westminster             |
| magnis orci convallis tempor            | Keiko Brennan       | Basil Sanford       | The Dalles              |
+-----------------------------------------+---------------------+---------------------+-------------------------+
478 rows in set (8,20 sec)
```

Query 11

Selezionare gli autori che hanno scritto brani rock con meno di 100 ascolti.

```
101  select autore.nome_cognome, brano.titolo, brano.num_ascolti
102  from autore, scrive, brano
103  where autore.id_autore = scrive.id_autore
104  and scrive.id_brano = brano.id_brano
105  and brano.genere = 'rock'
106  and brano.num_ascolti <= 100;
```

```
+--------------------+----------------------------------------+-------------+--------+
| nome_cognome       | titolo                                 | num_ascolti | genere |
+--------------------+----------------------------------------+-------------+--------+
| Armando Mcleod     | quam at ante sociosqu                  |          46 | Rock   |
| Constance Jacobson | quam at ante sociosqu                  |          46 | Rock   |
| Veronica Merritt   | Nullam sollicitudin tempus ridiculus   |          69 | Rock   |
| Alvin Bowen        | massa dapibus bibendum Maecenas        |          95 | Rock   |
| Gray Strong        | hymenaeos sapien arcu Aenean           |          11 | Rock   |
| Erasmus Simmons    | sociosqu lorem lobortis at             |          27 | Rock   |
| Zorita Bennett     | rutrum posuere sollicitudin bibendum   |          76 | Rock   |
| Tiger Ellison      | laoreet eu volutpat In                 |          42 | Rock   |
| Macy Dodson        | vitae ultricies pharetra felis         |          98 | Rock   |
| Kaitlin Church     | Mauris et a id                         |          31 | Rock   |
| Victoria Bowman    | Mauris et a id                         |          31 | Rock   |
| Florence Workman   | vel Etiam lectus sem                   |          44 | Rock   |
| Tana Sharp         | vel Etiam lectus sem                   |          44 | Rock   |
| Ifeoma Coffey      | nisi netus primis nascetur             |          80 | Rock   |
| Oren Andrews       | iaculis convallis fermentum at         |          93 | Rock   |
| Savannah Douglas   | molestie scelerisque faucibus pulvinar |           9 | Rock   |
| Ainsley Lynn       | fermentum quis Quisque dolor           |          74 | Rock   |
+--------------------+----------------------------------------+-------------+--------+
17 rows in set (1,02 sec)
```

Query 12

Selezionare i concerti tenuti a Farmington nel 2017 che hanno venduto biglietti agli utenti abbonati nello stesso periodo

```
select concerto.nome_conc
from concerto,(
  select biglietto.codice_conc
  from biglietto,utente,abbonamento
  where biglietto.id_utente = utente.id_utente
  and utente.id_utente = abbonamento.id_utente
  and (
    YEAR(abbonamento.data_inizio) = 2017
    or YEAR(abbonamento.data_fine) = 2017
  )
) as x
where year(concerto.dataora_conc) = 2017
and x.codice_conc = concerto.codice_conc
and concerto.citta = 'Farmington';
```

```
+----------------------------------+
| nome_conc                        |
+----------------------------------+
| at sociis a Lorem                |
| feugiat pellentesque taciti quam |
+----------------------------------+
2 rows in set (0,08 sec)
```

Query 13

Selezionare i prodotti venduti ad utenti di etá superiore ai 18 anni.

```
123  select prodotto.nome_prodotto
124  from prodotto,compra,utente
125  where prodotto.codice_prodotto = compra.codice_prodotto
126  and compra.id_utente = utente.id_utente
127  and year(now()) - year(utente.data_nascita) >= 18
128  group by prodotto.nome_prodotto;
```

```
+---------------------+
| nome_prodotto       |
+---------------------+
| a a adipiscing Donec |
| a a ante Sed        |
| a a blandit purus   |
| a a cursus primis   |
```

```
| vulputate vulputate inceptos Aenean                |
| vulputate vulputate libero pulvinar                |
| vulputate vulputate taciti tortor                  |
| vulputate vulputate vulputate eros                 |
+----------------------------------------------------+
190449 rows in set (4,07 sec)
```

Query 14

Selezionare gli artisti provenienti da New Madrid che sono seguiti da utenti maschi che non hanno MAI sottoscritto un abbonamento.

```
130  select artista.alias
131  from artista,segueart,utente,abbonamento
132  where artista.luogo_provenienza = 'New Madrid'
133  and artista.id_artista = segueart.id_artista
134  and segueart.id_utente = utente.id_utente
135  and utente.sesso = 'maschio'
136  and utente.id_utente != abbonamento.id_utente
137  group by artista.alias;
```

```
+---------------------+
| alias               |
+---------------------+
| Acton Norton        |
| Addison Ramos       |
| Adrian Figueroa     |
| Aimee Leach         |
```

```
| Zachary Jenkins     |
| Zachery Todd        |
| Zelenia Chavez      |
| Zelenia Nguyen      |
| Zia Watson          |
+---------------------+
179 rows in set (1 min 30,57 sec)
```

Query 15

Selezionare gli artisti che si sono esibiti nel loro stesso luogo di provenienza nel 2017.

```
139  select artista.alias
140  from artista, esibisce, concerto
141  where artista.id_artista = esibisce.id_artista
142  and esibisce.codice_conc = concerto.codice_conc
143  and concerto.citta = artista.luogo_provenienza
144  and year(concerto.dataora_conc) = 2017;
```

```
+--------------+
| alias        |
+--------------+
| Hashim Hyde |
+--------------+
1 row in set (0,88 sec)
```

Query 16

Selezionare i brani aggiunti dagli utenti che hanno acquistato il prodotto ‘Curabitur neque ipsum Praesent’ nel 2014.

```
146  select brano.titolo
147  from brano, aggiunge, utente, compra, prodotto
148  where brano.id_brano = aggiunge.id_brano
149  and aggiunge.id_utente = utente.id_utente
150  and utente.id_utente = compra.id_utente
151  and compra.codice_prodotto = prodotto.codice_prodotto
152  and prodotto.nome_prodotto = 'Curabitur neque ipsum Praesent'
153  and year(compra.tempo_acquisto) = 2014;
```

```
+----------------------------+
| titolo                     |
+----------------------------+
| dapibus eleifend vitae Ut |
| Ut eget Curae Vestibulum  |
+----------------------------+
2 rows in set (0,10 sec)
```

Query 17

Contare i brani aggiunti da utenti che sono andati ai concerti dei rispettivi artisti.

```
155  select count(Y.id_brano)
156  from (
157    select utente.id_utente as ut1, artista.id_artista as art1
158    from utente, biglietto, concerto, esibisce, artista
159    where utente.id_utente = biglietto.id_utente
160    and biglietto.codice_conc = concerto.codice_conc
161    and concerto.codice_conc = esibisce.codice_conc
162    and esibisce.id_artista = artista.id_artista
163  ) as X, (
164    select utente.id_utente as ut2, artista.id_artista as art2, brano.id_brano
165    from utente, aggiunge, brano, artista
166    where utente.id_utente = aggiunge.id_utente
167    and aggiunge.id_brano = brano.id_brano
168    and brano.id_artista = artista.id_artista
169  ) as Y
170  where X.ut1 = Y.ut2
171  and X.art1 = Y.art2
```

```
+--------------------+
| count(Y.id_brano) |
+--------------------+
|                  0 |
+--------------------+
1 row in set (16,16 sec)
```

```
155  select count(Y.id_brano)
156  from (
157    select brano.id_brano
158    from utente, biglietto, concerto, esibisce, artista, aggiunge, brano
159    where utente.id_utente = biglietto.id_utente
160    and biglietto.codice_conc = concerto.codice_conc
161    and concerto.codice_conc = esibisce.codice_conc
162    and esibisce.id_artista = artista.id_artista
163    and utente.id_utente = aggiunge.id_utente
164    and aggiunge.id_brano = brano.id_brano
165    and brano.id_artista = artista.id_artista
166  ) as Y;
```

Stessa query scritta differentemente.

Query 18

Selezionare i brani aggiunti da utenti che hanno acquistato almeno un prodotto dello stesso artista.

```
168  select brano.titolo, artista.alias, utente.nome_cognome
169  from (
170    select artista.id_artista as art1, utente.id_utente as ut1
171    from utente, compra, prodotto, artista
172    where utente.id_utente = compra.id_utente
173    and compra.codice_prodotto = prodotto.codice_prodotto
174    and prodotto.id_artista = artista.id_artista
175  ) as X, (
176    select brano.id_brano as id_b,utente.id_utente as ut2,artista.id_artista as art2
177    from utente, aggiunge, brano, artista
178    where utente.id_utente = aggiunge.id_utente
179    and aggiunge.id_brano = brano.id_brano
180    and brano.id_artista = artista.id_artista
181  ) as Y, brano, artista, utente
182  where Y.id_b = brano.id_brano
183  and X.art1 = Y.art2
184  and X.ut1 = Y.ut2
185  and X.art1 = artista.id_artista
186  and X.ut1 = utente.id_utente;
```

```
+-------------------------------------+-------------------+-------------------+
| titolo                              | alias             | nome_cognome      |
+-------------------------------------+-------------------+-------------------+
| pellentesque porttitor luctus dolor | Hammett Maldonado | Roary Strong      |
| purus Class sed litora              | Karyn Moss        | Gage Mccall       |
| lorem massa scelerisque blandit     | Zeus Bolton       | Marsden Donaldson |
| lacus Etiam non purus               | Julian Buckley    | Shana Jarvis      |
+-------------------------------------+-------------------+-------------------+
4 rows in set (1 min 6,96 sec)
```

Query 19

Selezionare gli album che durano piú di 5 minuti e con piú di 10 brani pubblicati nel 2017, i cui artisti hanno tenuto un concerto nello stesso anno.

```
188  select album.titolo, artista.alias, concerto.nome_conc
189  from concerto, esibisce, artista, album
190  where album.durata >= '00:05:00'
191  and album.num_brani >= 5
192  and year(album.pubblicazione)=2017
193  and album.id_artista = artista.id_artista
194  and artista.id_artista = esibisce.id_artista
195  and esibisce.codice_conc = concerto.codice_conc
196  and year(concerto.dataora_conc) = 2017;
```

```
+----------------------------------------+---------------------+----------------------------------------+
| titolo                                 | alias               | nome_conc                              |
+----------------------------------------+---------------------+----------------------------------------+
| turpis Cum habitant Sed                | Hope Cannon         | semper feugiat ullamcorper facilisis   |
| malesuada pede pharetra inceptos       | Desiree Sweeney     | gravida Aenean Class habitant          |
| Suspendisse ullamcorper Maecenas augue | Rhona Beck          | pede malesuada pharetra felis          |
| viverra Aenean Class lacus             | Steven Key          | risus cursus nec lobortis              |
| lectus fames nec leo                   | Cruz Hardy          | Suspendisse eleifend diam sem          |
| in posuere Phasellus Nunc              | Cain Beasley        | rhoncus sollicitudin primis Lorem      |
| Etiam libero litora in                 | Gil Dickson         | libero facilisis ridiculus auctor      |
+----------------------------------------+---------------------+----------------------------------------+
34 rows in set (0,73 sec)
```

Query 20

Selezionare i brani aggiunti da utenti maschi che seguono utenti femmine.

```
198  select brano.titolo, ut1.nome_cognome, ut2.nome_cognome
199  from utente ut1, utente ut2, aggiunge, brano, segueut
200  where brano.id_brano = aggiunge.id_brano
201  and aggiunge.id_utente = ut1.id_utente
202  and ut1.sesso = 'maschio'
203  and ut1.id_utente = segueut.id_utente1
204  and segueut.id_utente2 = ut2.id_utente
205  and ut2.sesso = 'femmina';
```

```
+--------------------------------+---------------+---------------+
| titolo                         | nome_cognome  | nome_cognome  |
+--------------------------------+---------------+---------------+
| Curae dapibus primis Nulla     | Martena Hood  | Gillian Orr   |
| mollis enim penatibus ad       | Rafael Sloan  | Audrey Dean   |
| per mus Proin neque            | Zachery Estes | Karina Kramer |
| purus pede ultricies hymenaeos | Clarke Wells  | Brianna Harper |
| Duis molestie venenatis Curabitur | Clarke Wells | Brianna Harper |
| volutpat orci tortor nibh      | Silas Moore   | Brianna Harper |
+--------------------------------+---------------+---------------+
225384 rows in set (22,07 sec)
```

Query 21

Selezionare gli album che contengono brani di artisti con oltre 200.000 follower scritti da autori donne.

```
207  select album.titolo, artista.alias, autore.nome_cognome
208  from album, brano, artista, scrive, autore, composto
209  where album.id_album = composto.id_album
210  and composto.id_brano = brano.id_brano
211  and brano.id_artista = artista.id_artista
212  and artista.num_follower >= 200000
213  and brano.id_brano = scrive.id_brano
214  and scrive.id_autore = autore.id_autore
215  and autore.sesso = 'femmina';
```

```
+-----------------------------------+-----------------+---------------+
| titolo                            | alias           | nome_cognome  |
+-----------------------------------+-----------------+---------------+
| scelerisque velit Duis tempus     | Janna Reese     | Shelby Haley  |
| elit lobortis hymenaeos pretium   | Janna Reese     | Shelby Haley  |
| condimentum fringilla odio ac     | Janna Reese     | Shelby Haley  |
| varius Proin primis nostra        | Janna Reese     | Shelby Haley  |
| quis a Mauris metus               | Elmo Navarro    | Shelby Haley  |
| tortor varius sociis euismod      | Elmo Navarro    | Shelby Haley  |
| magna conubia justo sit           | Cyrus Freeman   | Shelby Haley  |
| sociis id mattis justo            | Molly Hewitt    | Mari Workman  |
| penatibus venenatis Etiam mi      | Dacey Humphrey  | Mallory Chapman |
| consequat elementum taciti ac     | Dacey Humphrey  | Mallory Chapman |
| facilisis massa cursus Vestibulum | Dacey Humphrey  | Mallory Chapman |
| Cum tempor Curae iaculis          | Alden Flores    | April Kidd    |
| vitae erat orci fermentum         | Alden Flores    | April Kidd    |
+-----------------------------------+-----------------+---------------+
244250 rows in set (9,32 sec)
```

Query 22

Selezionare le playlist con piú di 85000 follower create da utenti con piú di 85000 follower e following, i quali seguono almeno un utente con altrettanti follower e following.

```
217  select playlist.nome
218  from playlist, crea, utente ut1, utente ut2, segueut
219  where playlist.num_follower >= 85000
220  and playlist.id_playlist = crea.id_playlist
221  and crea.id_utente = ut1.id_utente
222  and ut1.num_follower >= 85000
223  and ut1.num_following >= 85000
224  and ut1.id_utente = segueut.id_utente1
225  and segueut.id_utente2 = ut2.id_utente
226  and ut2.num_follower >= 85000
227  and ut2.num_following >= 85000;
```

```
+-----------------------------------------+
| nome                                    |
+-----------------------------------------+
| feugiat condimentum nulla imperdiet     |
| tellus Proin aliquet lobortis           |
| ad Class felis habitant                 |
| litora Etiam Sed Quisque                |
| Phasellus tempor volutpat purus         |
| magnis sollicitudin Integer fermentum   |
```

```
| posuere erat tellus venenatis           |
| tempor dui enim habitant                |
| risus diam vehicula tempor              |
| rutrum sociis eu Praesent               |
| Vestibulum nec egestas adipiscing       |
+-----------------------------------------+
70 rows in set (0,51 sec)
```

Ottimizzazione Query

In questa parte del progetto faremo uso del comando **EXPLAIN**, che ci fornirà informazioni sulle Query, in particolare su come MySQL intende interpretarle ed eseguirle.

Ottimizzazione Query 14

```
130  select artista.alias
131  from artista,segueart,utente,abbonamento
132  where artista.luogo_provenienza = 'New Madrid'
133  and artista.id_artista = segueart.id_artista
134  and segueart.id_utente = utente.id_utente
135  and utente.sesso = 'maschio'
136  and utente.id_utente != abbonamento.id_utente
137  group by artista.alias;
```

```
+----+-------------+------------+------------+--------+---------------------------+------------------+---------+---------------------------+--------+----------+-------
-------------------------------------------------------+
| id | select_type | table      | partitions | type   | possible_keys             | key              | key_len | ref                       | rows   | filtered | Extra
                                                       |
+----+-------------+------------+------------+--------+---------------------------+------------------+---------+---------------------------+--------+----------+-------
-------------------------------------------------------+
|  1 | SIMPLE      | artista    | NULL       | ref    | PRIMARY,luogo_provenienza | luogo_provenienza | 62      | const                     |    312 |   100.00 | Using
temporary; Using filesort                              |
|  1 | SIMPLE      | segueart   | NULL       | ref    | PRIMARY,id_utente,id_artista | id_artista     | 4       | spotify.artista.id_artista |      2 |   100.00 | Using
index                                                  |
|  1 | SIMPLE      | utente     | NULL       | eq_ref | PRIMARY,sesso             | PRIMARY          | 4       | spotify.segueart.id_utente |      1 |    50.00 | Using
where                                                  |
|  1 | SIMPLE      | abbonamento | NULL      | index  | NULL                      | id_utente        | 4       | NULL                      | 499223 |    90.00 | Using
where; Using index; Using join buffer (Block Nested Loop) |
+----+-------------+------------+------------+--------+---------------------------+------------------+---------+---------------------------+--------+----------+-------
-------------------------------------------------------+
4 rows in set, 1 warning (0,00 sec)
```

ALTER TABLE artista ADD INDEX(luogo_provenienza)

```
| Yuri Beach           |
| Zachary Jenkins      |
| Zachery Todd         |
| Zelenia Chavez       |
| Zelenia Nguyen       |
| Zia Watson           |
+----------------------+
179 rows in set (54,11 sec)
```

La query richiede circa 40 secondi in meno.

Ottimizzazione Query 18

```
168  select brano.titolo, artista.alias, utente.nome_cognome
169  from (
170    select artista.id_artista as art1, utente.id_utente as ut1
171    from utente, compra, prodotto, artista
172    where utente.id_utente = compra.id_utente
173    and compra.codice_prodotto = prodotto.codice_prodotto
174    and prodotto.id_artista = artista.id_artista
175  ) as X, (
176    select brano.id_brano as id_b,utente.id_utente as ut2,artista.id_artista as art2
177    from utente, aggiunge, brano, artista
178    where utente.id_utente = aggiunge.id_utente
179    and aggiunge.id_brano = brano.id_brano
180    and brano.id_artista = artista.id_artista
181  ) as Y, brano, artista, utente
182  where Y.id_b = brano.id_brano
183  and X.art1 = Y.art2
184  and X.ut1 = Y.ut2
185  and X.art1 = artista.id_artista
186  and X.ut1 = utente.id_utente;
```

```
+----+-------------+---------+------------+--------+-------------------+-----------+---------+-------------------------------+--------+----------+-------------+
| id | select_type | table   | partitions | type   | possible_keys     | key       | key_len | ref                           | rows   | filtered | Extra       |
+----+-------------+---------+------------+--------+-------------------+-----------+---------+-------------------------------+--------+----------+-------------+
|  1 | SIMPLE      | prodotto | NULL       | index  | PRIMARY,id_artista | id_artista | 4       | NULL                          | 199540 |   100.00 | Using index |
|  1 | SIMPLE      | artista  | NULL       | eq_ref | PRIMARY           | PRIMARY   | 4       | spotify.prodotto.id_artista   |      1 |   100.00 | Using index |
|  1 | SIMPLE      | artista  | NULL       | eq_ref | PRIMARY           | PRIMARY   | 4       | spotify.prodotto.id_artista   |      1 |   100.00 | Using index |
|  1 | SIMPLE      | artista  | NULL       | eq_ref | PRIMARY           | PRIMARY   | 4       | spotify.prodotto.id_artista   |      1 |   100.00 | NULL        |
|  1 | SIMPLE      | compra   | NULL       | ref    | PRIMARY,id_utente | PRIMARY   | 4       | spotify.prodotto.codice_prodotto |      3 |   100.00 | Using index |
|  1 | SIMPLE      | utente   | NULL       | eq_ref | PRIMARY           | PRIMARY   | 4       | spotify.compra.id_utente      |      1 |   100.00 | Using index |
|  1 | SIMPLE      | utente   | NULL       | eq_ref | PRIMARY           | PRIMARY   | 4       | spotify.compra.id_utente      |      1 |   100.00 | Using index |
|  1 | SIMPLE      | utente   | NULL       | eq_ref | PRIMARY           | PRIMARY   | 4       | spotify.compra.id_utente      |      1 |   100.00 | NULL        |
|  1 | SIMPLE      | aggiunge | NULL       | ref    | PRIMARY,id_brano  | PRIMARY   | 4       | spotify.compra.id_utente      |      1 |   100.00 | Using index |
|  1 | SIMPLE      | brano    | NULL       | eq_ref | PRIMARY,id_artista | PRIMARY   | 4       | spotify.aggiunge.id_brano     |      1 |     5.00 | Using where |
|  1 | SIMPLE      | brano    | NULL       | eq_ref | PRIMARY           | PRIMARY   | 4       | spotify.aggiunge.id_brano     |      1 |   100.00 | NULL        |
+----+-------------+---------+------------+--------+-------------------+-----------+---------+-------------------------------+--------+----------+-------------+
11 rows in set, 1 warning (0,01 sec)
```

ALTER TABLE prodotto ADD INDEX(id_artista)

```
+-------------------------------------+--------------------+--------------------+
| titolo                              | alias              | nome_cognome       |
+-------------------------------------+--------------------+--------------------+
| pellentesque porttitor luctus dolor | Hammett Maldonado | Roary Strong       |
| purus Class sed litora              | Karyn Moss         | Gage Mccall        |
| lorem massa scelerisque blandit     | Zeus Bolton        | Marsden Donaldson  |
| lacus Etiam non purus               | Julian Buckley     | Shana Jarvis       |
+-------------------------------------+--------------------+--------------------+
4 rows in set (30,64 sec)
```

Adesso la query richiede circa 30 secondi in meno.

Interrogazione Algebra Relazionale

Algebra Relazionale

L' algebra relazionale è un linguaggio di interrogazione di basi di dati.

E' un linguaggio procedurale in cui le operazioni complesse vengono specificate descrivendo il procedimento da seguire per ottenere la soluzione.

L'algebra relazionale è basata su concetti di tipo algebrico.

Sostanzialmente questo linguaggio è costituito da un insieme di operatori, definiti su relazioni, che producono ancora relazioni come risultati.

Interrogazione su Query 3

Selezionare NOME e COGNOME degli UTENTI e quanto hanno speso in totale in PRODOTTI.

$$\pi \text{ utente.nome\_cognome, sum(prodotto.prezzo)} \Big(\sigma \Big(\text{UTENTE} \bowtie \text{utente.id\_utente = compra.id\_utente COMPRA} \bowtie \text{compra.codice\_prodotto = prodotto.codice\_prodotto PRODOTTO} \Big) \Big)$$

Interrogazione su Query 5

Selezionare NOME e COGNOME degli UTENTI abbonati nel 2016 e tutti gli ABBONAMENTI rispettivi.

$$\pi_{\text{utente.nome\_cognome, abbonamento.data\_inizio, abbonamento.data\_fine}} \Big(\sigma_{\text{YEAR(abbonamento.data\_inizio) = 2016 or YEAR(abbonamento.data\_fine) = 2016}} \big(\text{UTENTE} \bowtie_{\text{utente.id\_utente = abbonamento.id\_utente}} \text{ABBONAMENTO} \big) \Big)$$

MongoDB

MongoDB è un DBMS non relazionale, orientato ai documenti.

E' classificato come un database di tipo NoSQL poiché si allontana dalla struttura tradizionale basata sulle tabelle dei database relazionali in favore di documenti in stile JSON con schema dinamico, rendendo l'integrazione di alcuni tipi di applicazioni più facili e veloci.

Tra i punti chiave di MongoDB troviamo:

- Query ad hoc
- Indicizzazione
- Alta affidabilità
- Sharding e bilanciamento dei dati
- File storage
- Aggregazione
- Capped collection

Creazione DataBase e Collection

Utilizzando un CMD entriamo in Mongo e tramite il comando USE creiamo il nostro DataBase per MongoDB e allo stesso tempo Mongo effettua il cambio di database su questo appena creato.

```
> use db_spotify
switched to db db_spotify
```

MongoDB immagazzina tutti i documenti in delle Collection. Le Collezioni sono l'analogo delle Tabelle nei DataBase Relazionali.

Per creare la nostra Collection scegliamo il Ramo del nostro DataBase di MySQL dove sono presenti le Tabelle :

- Brano
- Artista
- Album
- Utente
- Autore

Creiamo su MySQL un tabella fittizia, tramite il comando VIEW, che comprende tutti i campi necessari a noi in MongoDB.

```
238  create view baaau as
239    select brano.id_brano, artista.id_artista, album.id_album, autore.id_autore, utente.id_utente,
240    brano.titolo as titolo_brano, brano.genere, brano.durata as durata_brano, brano.num_ascolti,
241    album.titolo as titolo_album, album.num_brani, album.durata as durata_album, album.pubblicazione,
242    autore.nome_cognome as alias_autore, autore.citta as citta_autore, autore.sesso as sesso_autore,
243    autore.data_nascita as nascita_autore, artista.alias as alias_artista, artista.luogo_provenienza,
244    utente.nome_cognome as nome_utente, utente.sesso as sesso_utente, utente.citta as citta_utente,
245    utente.data_nascita as nascita_utente
246    from brano, artista, album, autore, utente, scrive, aggiunge, composto
247    where utente.id_utente = aggiunge.id_utente
248    and aggiunge.id_brano = brano.id_brano
249    and brano.id_brano = scrive.id_brano
250    and scrive.id_autore = autore.id_autore
251    and brano.id_brano = composto.id_brano
252    and composto.id_album = album.id_album
253    and album.id_artista = artista.id_artista;
```

Il risultato di questa VIEW è il seguente (limit 10):

id_brano	id_artista	id_album	id_autore	id_utente	titolo_brano	genere	durata_brano	num_ascolti
0	0	0	0	0	facilisis ultrices Cum Ut	Musica da film	00:01:42	86951
0	0	0	1	0	facilisis ultrices Cum Ut	Musica da film	00:01:42	86951
0	0	0	2	0	facilisis ultrices Cum Ut	Musica da film	00:01:42	86951
0	0	0	3	0	facilisis ultrices Cum Ut	Musica da film	00:01:42	86951
0	0	0	4	0	facilisis ultrices Cum Ut	Musica da film	00:01:42	86951
0	0	0	5	0	facilisis ultrices Cum Ut	Musica da film	00:01:42	86951
0	0	0	6	0	facilisis ultrices Cum Ut	Musica da film	00:01:42	86951
0	0	0	7	0	facilisis ultrices Cum Ut	Musica da film	00:01:42	86951
0	0	0	8	0	facilisis ultrices Cum Ut	Musica da film	00:01:42	86951
0	0	0	9	0	facilisis ultrices Cum Ut	Musica da film	00:01:42	86951

titolo_album	num_brani	durata_album	pubblicazione	alias_autore	citta_autore	sesso_autore
lacus consequat odio mauris	3	00:08:07	1937-04-09	Matthew Barry	Torrington	maschio
lacus consequat odio mauris	3	00:08:07	1937-04-09	Dolan Morton	Pullman	maschio
lacus consequat odio mauris	3	00:08:07	1937-04-09	Leo Shields	Santa Clara	maschio
lacus consequat odio mauris	3	00:08:07	1937-04-09	Madison Lindsey	Hazleton	maschio
lacus consequat odio mauris	3	00:08:07	1937-04-09	Baxter Castaneda	Lawrenceville	maschio
lacus consequat odio mauris	3	00:08:07	1937-04-09	Shelby Haley	Des Moines	femmina
lacus consequat odio mauris	3	00:08:07	1937-04-09	Ingrid Jones	San Marino	femmina
lacus consequat odio mauris	3	00:08:07	1937-04-09	Lev Rutledge	Half Moon Bay	femmina
lacus consequat odio mauris	3	00:08:07	1937-04-09	Walker Sanchez	Beacon	maschio
lacus consequat odio mauris	3	00:08:07	1937-04-09	Idona Le	DuBois	maschio

nascita_autore	alias_artista	luogo_provenienza	nome_utente	sesso_utente	citta_utente	nascita_utente
2010-01-31	Joy Copeland	Carrollton	Porter Hunt	maschio	Laguna Beach	1988-12-01
2017-09-13	Joy Copeland	Carrollton	Porter Hunt	maschio	Laguna Beach	1988-12-01
2010-02-16	Joy Copeland	Carrollton	Porter Hunt	maschio	Laguna Beach	1988-12-01
2013-09-24	Joy Copeland	Carrollton	Porter Hunt	maschio	Laguna Beach	1988-12-01
2013-07-23	Joy Copeland	Carrollton	Porter Hunt	maschio	Laguna Beach	1988-12-01
2011-01-25	Joy Copeland	Carrollton	Porter Hunt	maschio	Laguna Beach	1988-12-01
2015-01-06	Joy Copeland	Carrollton	Porter Hunt	maschio	Laguna Beach	1988-12-01
2015-05-11	Joy Copeland	Carrollton	Porter Hunt	maschio	Laguna Beach	1988-12-01
2016-04-01	Joy Copeland	Carrollton	Porter Hunt	maschio	Laguna Beach	1988-12-01
2012-03-07	Joy Copeland	Carrollton	Porter Hunt	maschio	Laguna Beach	1988-12-01

Una volta Creata la Tabella fittizia, dobbiamo esportarla in formato CSV per poi tramite Software esterni convertirla in JSON.

Per esportare la la tabella **baaau** eseguiamo una select con il seguente comando aggiuntivo:

```
252  select *
253  from baaau
254  into outfile '/home/fabiano/Scrivania/baaau.csv'
255  fields enclosed by '"'
256  terminated by ','
257  lines terminated by '\n';
```

Una volta terminata l'esportazione e la conversione, utilizziamo un CMD e con il comando MONGOIMPORT importiamo e creiamo la Collection per contenere tutti i nostri dati.

```
fabiano@debian:~/Scrivania$ mongoimport -d db_spotify4 -c baaau --type csv --file baaau.csv --headerline
2018-02-02T15:49:42.786+0100    connected to: localhost
2018-02-02T15:49:45.779+0100    [##......................] db_spotify4.baaau    20.8MB/205MB (10.1%)
2018-02-02T15:49:48.779+0100    [#####...................] db_spotify4.baaau    43.1MB/205MB (21.0%)
2018-02-02T15:49:51.780+0100    [#######.................] db_spotify4.baaau    68.1MB/205MB (33.1%)
2018-02-02T15:49:54.780+0100    [###########.............] db_spotify4.baaau    95.9MB/205MB (46.7%)
2018-02-02T15:49:57.780+0100    [##############..........] db_spotify4.baaau    123MB/205MB (60.1%)
2018-02-02T15:50:00.779+0100    [#################.......] db_spotify4.baaau    151MB/205MB (73.5%)
2018-02-02T15:50:03.779+0100    [####################....] db_spotify4.baaau    180MB/205MB (87.6%)
2018-02-02T15:50:06.577+0100    [########################] db_spotify4.baaau    205MB/205MB (100.0%)
2018-02-02T15:50:06.579+0100    imported 728636 documents
```

Ottenendo infine un Collection chiamata **baaau**.

Effettuando una FIND possiamo ottenere un estratto del contenuto presente nella collection.

```
> db.baaau.find().pretty()
{
        "_id" : ObjectId("5a747a86e12d9e2981ab1ce0"),
        "id_brano" : 79538,
        "id_artista" : 28895,
        "id_album" : 2,
        "id_autore" : 123072,
        "id_utente" : 503268,
        "titolo_brano" : "eget nibh sociis euismod",
        "genere" : "Trance",
        "durata_brano" : "00:06:23",
        "num_ascolti" : 65314,
        "titolo_album" : "euismod leo gravida massa",
        "num_brani" : 7,
        "durata_album" : "00:04:16",
        "pubblicazione" : "1973-02-28",
        "alias_autore" : "Orson Bryant",
        "citta_autore" : "Bellflower",
        "sesso_autore" : "femmina",
        "nascita_autore" : "2016-09-15",
        "alias_artista" : "Hope Hickman",
        "luogo_provenienza" : "Pittsfield",
        "nome_utente" : "Tasha Berry",
        "sesso_utente" : "femmina",
        "citta_utente" : "West Haven",
        "nascita_utente" : "1936-04-15"
}
```

La find restituendo tutti i dati della Collection (quindi una quantità di dati abbastanza grande), non ci stampa tutto il risultato, ma si ferma, mostrandoci il comando IT che se digitato, può mostrarci altri dati.

Query MongoDB

Query 1 MongoDB

Selezionare i BRANI di genere ‘rock’ di ARTISTI provenienti da ‘Allentown’

```
> db.baaau.find({
... genere : "Rock",
... luogo_provenienza : "Allentown"},
... { _id : 0,
... titolo_brano : 1,
... alias_artista : 1}).pretty()
{
        "titolo_brano" : "ullamcorper nostra risus sociis",
        "alias_artista" : "Fiona Downs"
}
{
        "titolo_brano" : "vulputate dictum nulla est",
        "alias_artista" : "Octavia Fuentes"
}
{
        "titolo_brano" : "Aenean laoreet justo Etiam",
        "alias_artista" : "Lance Soto"
}
{
        "titolo_brano" : "ut Proin fermentum Aenean",
        "alias_artista" : "Josephine Benson"
}
```

Query 2 MongoDB

Selezionare UTENTI e BRANI di ARTISTI provenienti dalla stessa cittá

```
var a = [];
var a = db.baaau.find({});
var array = [];
for ( var i=0; i<a.length(); i++ ){
  if( a[i].citta_utente == a[i].luogo_provenienza ){
    array.push(a[i].titolo_brano);
    array.push(a[i].alias_artista);
    array.push(a[i].nome_utente);
  }
}
var t = array[0];
var array2 = [];
var i = 0;
array2[i] = t;
for(var j=1; j < 15; j++){
  if(array[j] == t){}
  else {
    i++;
    array2[i] = array[j];
    t = array[j];
  }
}
```

```
> array2
[
        "convallis odio porta Nulla",
        "Hiram Parrish",
        "Aphrodite Sloan",
        "sapien justo est condimentum",
        "Lucy Hampton",
        "Cairo Berry",
        "sapien justo est condimentum",
        "Lucy Hampton",
        "Cairo Berry",
        "pharetra accumsan consectetuer Ut",
        "Bert Franks",
        "Bradley Huff",
        "pharetra accumsan consectetuer Ut",
        "Bert Franks",
        "Bradley Huff"
]
```