
# Emotional Songs – Documento di progettazione del Database

Università degli Studi dell'Insubria – Laurea Triennale in Informatica
Progetto Laboratorio B

Sviluppato da:

**Claudio Della Motta**, matricola 750667
**Diana Cantaluppi**, matricola 744457
**Edoardo Ballabio**, matricola 745115
**Joele Vallone**, matricola 744775

## Indice

# 1.0 - Raccolta e analisi dei requisiti

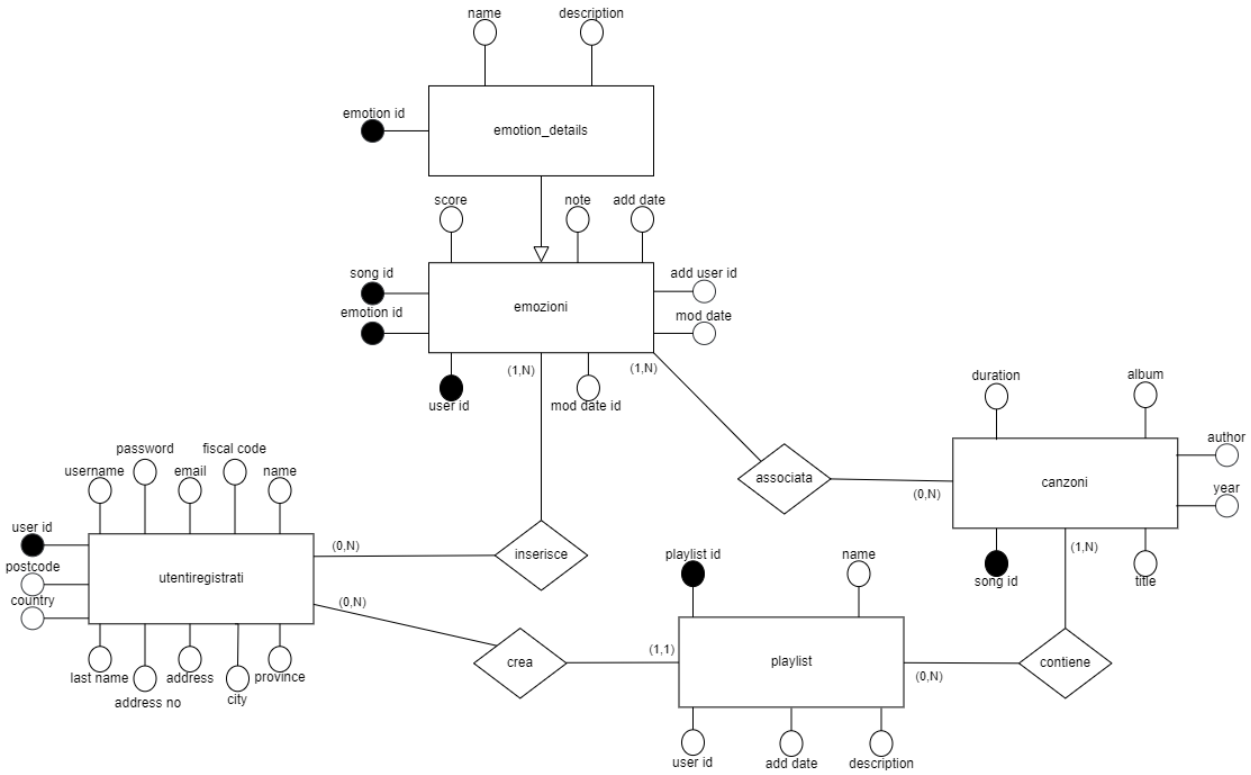
L'applicazione EmotionalSongs mette a disposizione diverse funzionalità con scopi e dati diversi, è quindi necessario andare a creare una tabella per ogni scopo e dato. Abbiamo identificato quindi e costruito le seguenti tabelle:

- utenti_registrati: contiene tutti i dati degli utenti registrati
- song: contiene tutti i dati delle canzoni
- emotion_details: contiene la descrizione delle emozioni
- playlist: contiene la testata di tutte le playlist dei vari utenti
- playlist_song: contiene tutte le canzoni contenute nelle playlist degli utenti
- emozioni: contiene tutti i feedback registrati dagli utenti

Ogni tabella ha un campo che identifica ogni riga e assume il ruolo di chiave primaria. Inoltre, in alcune tabelle ci sono dei riferimenti a dati di altre tabelle come, ad esempio, nella tabella playlist c'è il riferimento all'utente proprietario della playlist che è salvato nella tabella utenti_registrati .

L'applicazione EmotionalSongs richiede inoltre che alcuni campi, come ad esempio lo username e la password, siano valorizzati sempre, quindi all'interno del database ci saranno dei che non assumeranno mai valori nulli.

# 2.0 - Schema relazionale



# 3.0 - Scelte progettuali

Il modello mostrato è stato sviluppato tenendo in considerazione le richieste del progetto. La nostra attenzione si è focalizzata sulla correttezza, completezza, minimalità e efficienza dello schema. Infatti, abbiamo scelto di sviluppare un database con delle tabelle semplici, chiare e uniformi.

La strada percorsa ha permesso di ridurre le informazioni ridondanti e i concetti superflui.

Ogni relazione ha al suo interno gli attributi richiesti nelle specifiche del progetto e alcuni campi opzionali. La differenza tra le due "tipologie" di attributo la si può evincere dalla dicitura "NOT NULL" presente nel DDL della tabella.
Alcuni attributi hanno dei valori di default definiti a livello di DDL, questo ci permette di ometterli nelle query di inserimento se il valore da inserire è quello standard. Questo ci è tornato utile per la gestione dei valori delle chiavi primarie di alcune tabelle.

Ogni tabella ha la sua chiave primaria e questa può essere:

- semplice: la PK è un valore numerico auto incrementale gestito da una sequence
- composta: la PK è definita da un'insieme di attributi

Abbiamo scelto di usare delle sequence per andare a gestire le chiavi primarie perché questo rende più semplice identificare una tupla all'interno della relazione e rende più semplice la modifica dei valori associati alla chiave primaria. Prendiamo ad esempio la tabella playlist: se la chiave primaria della relazione fosse l'attributo nome e per questo attributo è previsto che il valore possa cambiare, bisognerebbe andare a gestire l'aggiornamento di tutte le tuple della relazione playlist_song in cui questa è riferita.

Tutte le chiavi esterne, le foreign key, hanno all'interno del DDL il rifermento alla tabella di origine, se queste possono essere omesse sono dichiarate come NULL, invece se sono obbligatorie vengono dichiarate come NOT NULL.

In alcune tabelle abbiamo utilizzato gli indici per andare ad indicizzare determinati attributi per rendere più veloci le interrogazioni che "filtrano" per un determinato attributo. Un esempio di questa implementazione lo si può vedere applicato alla ricerca di un brano e alla ricerca di un utente.

Gli attributi "add_user_id", "add_date", "mod_user_id" e "mod_date" presenti solo in alcune tabelle sono utili e servono per tenere tracci di quando una tupla è stata inserita e/o modificata e da chi. Questi campi si rivelano fondamentali se in futuro si vorrà implementare una relazione di storico, ovvero una relazione che salva al suo interno tutte le operazioni fatte dall'utente.

### 3.1 - Funzioni custum

PostgreSQL mette a disposizione diverse funzioni come, ad esempio, la funzione stringAgg che permette di aggregare in un'unica tringa il valore di un attributo di diverse tuple in un unico risultato.

Per poter popolare le tabelle in modo dinamico in fase di configurazione del database, abbiamo "sviluppato" una funzione in linguaggio PLPGSQL che dati due numeri restituisce un numero casuale generato nel range definito dai due numeri passati come parametri.

Di seguito la definizione della funzione:

```
CREATE OR REPLACE FUNCTION emotional_songs.random_between(low numeric,
high numeric)
RETURNS INT AS
$$
     BEGIN
```

```
            RETURN floor(random()* (high-low + 1) + low);
      END;
$$ language 'plpgsql' STRICT
;
```

### 3.2 - Collegamento tra relazioni

| Relazione A | Tipologia di relazione | Relazione B | Motivazione |
|---|---|---|---|
| utenti_registrati | 1 a N | playlist | Un utente può avere 1 o più playlist |
| playlist | N a N (molti a molti) | playlist_song | Una playlist può contenere più canzoni |
| song | N a N (molti a molti) | playlist_song | Una canzone può essere in più playlist |
| utenti_registrati | 1 a N (uno a molti) | emozioni | Un utente può rilasciare 1 o più feedback |
| song | 1 a N (uno a molti) | emozioni | Una canzone può avere più feedback |
| emozioni | 1 a 1 (uno a uno) | emotion_details | Un feedback può avere una sola emozione |

**Tipologia di relazione**

- 1 a 1: uno a uno
- 1 a N: uno a molti
- N a N: molti a molti

## 4.0 - Progettazione Pratica

```
DROP SCHEMA IF EXISTS emotional_songs CASCADE;
```

Questo comando verifica l'esistenza dello schema "emotional_songs". Se lo schema esiste, lo elimina insieme a tutti gli oggetti al suo interno, come tabelle, viste o altri elementi, grazie all'opzione CASCADE. Se lo schema non esiste, il comando viene eseguito senza generare errori.

```
CREATE SCHEMA emotional_songs;
```

Questo comando crea uno schema denominato "emotional_songs" nel database.

```
CREATE SEQUENCE emotional_songs.seq_utenti_registrati;
CREATE TABLE emotional_songs.utenti_registrati (
    user_id numeric(38) NOT NULL DEFAULT
nextval('emotional_songs.seq_utenti_registrati'::regclass),
    username varchar(255) NOT NULL,
    "password" varchar NOT NULL,
    fiscal_code varchar(255) NULL,
    email varchar(255) NULL,
    "name" varchar(255) NULL,
    last_name varchar(255) NULL,
    address varchar(255) NULL,
    address_no varchar(255) NULL,
    city varchar(255) NULL,
    province varchar(255) NULL,
    postcode varchar(255) NULL,
    country varchar(255) NULL,
    CONSTRAINT utenti_registrati_pk PRIMARY KEY (user_id)
);
CREATE INDEX utenti_registrati_username_idx ON
emotional_songs.utenti_registrati (username);
```

Il blocco di comandi SQL sopra elencato viene utilizzato per definire una sequenza e creare una tabella chiamata "utenti_registrati" all'interno dello schema del database "emotional_songs". Viene inoltre creato un indice sul campo username.

```
CREATE SEQUENCE emotional_songs.seq_song;
CREATE TABLE emotional_songs.song (
    song_id numeric(38) NOT NULL DEFAULT
nextval('emotional_songs.seq_song'::regclass),
    title varchar(255) NOT NULL,
    author varchar(255) NULL,
    "year" varchar(16) NULL,
    album varchar(255) NULL,
    duration numeric(24,3) NULL,
    CONSTRAINT song_pk PRIMARY KEY (song_id)
);
CREATE INDEX song_title_idx ON emotional_songs.song (title);
CREATE INDEX song_author_idx ON emotional_songs.song (author);
CREATE INDEX song_year_idx ON emotional_songs.song ("year");
CREATE INDEX song_album_idx ON emotional_songs.song (album);
```

Questo blocco di comandi SQL è utilizzato per definire una sequenza e creare una tabella chiamata "song" all'interno dello schema del database "emotional_songs". Inoltre vengono creati degli indici sui campi title, author, year e album della tabella.

```
CREATE TABLE emotional_songs.emotion_details (
    emotion_id numeric(38) NOT NULL,
    "name" varchar(255) NULL,
    description varchar(255) NULL,
    CONSTRAINT emotion_details_pk PRIMARY KEY (emotion_id)
);
```

Questo comando è utilizzato per creare una tabella denominata "emotion_details" all'interno dello schema del database "emotional_songs".

```
INSERT INTO emotional_songs.emotion_details (emotion_id, "name",
description) VALUES (1, 'stupore', 'Sensazione di meraviglia o
felicità');
INSERT INTO emotional_songs.emotion_details (emotion_id, "name",
description) VALUES(3, 'tenerezza', 'Sensualità, affetto, sentimento d
amore');
INSERT INTO emotional_songs.emotion_details (emotion_id, "name",
description) VALUES(4, 'nostalgia ', 'Sensazioni sognanti,
malinconiche, sentimentali');
INSERT INTO emotional_songs.emotion_details (emotion_id, "name",
description) VALUES(5, 'calma', 'Relax, serenità, meditazione');
INSERT INTO emotional_songs.emotion_details (emotion_id, "name",
description) VALUES(6, 'potenza', 'Sentirsi forte, eroico, trionfante,
energico');
INSERT INTO emotional_songs.emotion_details (emotion_id, "name",
description) VALUES(7, 'gioia', 'Voglia di ballare, sentirsi animato,
divertito, vivo');
INSERT INTO emotional_songs.emotion_details (emotion_id, "name",
description) VALUES(8, 'tensione', 'Sentirsi nervosi, impazienti,
irritati');
INSERT INTO emotional_songs.emotion_details (emotion_id, "name",
description) VALUES(9, 'tristezza', 'Sentirsi depresso, addolorato');
```

```
INSERT INTO emotional_songs.emotion_details (emotion_id, "name",
description) VALUES(2, 'solennità', 'Sensazione di trascendenza,
ispirazione.');
```

Questo blocco di comandi SQL è utilizzato per inserire dati nella tabella "emotion_details". Ogni comando inserisce una riga corrispondente all'emozione specificata.

```
CREATE SEQUENCE emotional_songs.seq_playlist;
CREATE TABLE emotional_songs.playlist (
    playlist_id numeric(38) NOT NULL DEFAULT
nextval('emotional_songs.seq_playlist'::regclass),
    "name" varchar(255) NOT NULL,
    user_id numeric(38) NOT NULL,
    add_date timestamp NULL,
    description varchar(255) NULL,
    CONSTRAINT playlist_pk PRIMARY KEY (playlist_id),
    CONSTRAINT playlist_user_id_fk FOREIGN KEY (user_id) REFERENCES
emotional_songs.utenti_registrati(user_id)
);
CREATE INDEX playlist_user_id_idx ON emotional_songs.playlist
(user_id);
CREATE INDEX playlist_name_idx ON emotional_songs.playlist ("name");
```

Il seguente blocco di comandi SQL è utilizzato per definire una sequenza e creare una tabella chiamata "playlist" all'interno dello schema del database "emotional_songs". Inoltre vengono creati degli indici sui campi user_id e name della tabella "playlist".

```
CREATE TABLE emotional_songs.playlist_song (
    playlist_id numeric(38) NOT NULL,
    song_id numeric(38) NOT NULL,
    add_date timestamp NULL DEFAULT current_timestamp,
    mod_date timestamp NULL DEFAULT current_timestamp,
    CONSTRAINT playlist_song_pk PRIMARY KEY (playlist_id, song_id),
    CONSTRAINT playlist_song_playlist_id_fk FOREIGN KEY (playlist_id)
REFERENCES emotional_songs.playlist(playlist_id),
    CONSTRAINT playlist_song_song_id_fk FOREIGN KEY (song_id)
REFERENCES emotional_songs.song(song_id)
);
CREATE INDEX playlist_song_playlist_id_idx ON
emotional_songs.playlist_song (playlist_id);
```

Questo blocco di comandi SQL è utilizzato per creare una tabella chiamata "playlist_song" all'interno dello schema del database "emotional_songs". Inoltre viene creato un indice sul campo playlist_id della tabella.

```
CREATE TABLE emotional_songs.emozioni (
    song_id numeric(38) NOT NULL,
    emotion_id numeric(38) NOT NULL,
    user_id numeric(38) NOT NULL,
    score numeric(1) NULL,
    note varchar(255) NULL,
    add_date timestamp NULL DEFAULT CURRENT_TIMESTAMP,
    add_user_id varchar(255) NULL,
    mod_date timestamp NULL DEFAULT CURRENT_TIMESTAMP,
    mod_user_id varchar(255) NULL,
    CONSTRAINT emozioni_pk PRIMARY KEY (song_id, emotion_id, user_id),
    CONSTRAINT emozioni_song_id_fk FOREIGN KEY (song_id) REFERENCES
emotional_songs.song(song_id),
    CONSTRAINT emozioni_emotion_id_fk FOREIGN KEY (emotion_id)
REFERENCES emotional_songs.emotion_details(emotion_id),
```

```
    CONSTRAINT emozioni_user_id_fk FOREIGN KEY (user_id) REFERENCES
emotional_songs.utenti_registrati(user_id)
);
CREATE INDEX emozioni_song_id_idx ON emotional_songs.emozioni
(song_id);
CREATE INDEX emozioni_emotion_id_idx ON emotional_songs.emozioni
(emotion_id);
CREATE INDEX emozioni_user_id_idx ON emotional_songs.emozioni
(user_id);
```

Questo blocco di comandi SQL è utilizzato per creare una tabella chiamata "emozioni" all'interno dello schema del database "emotional_songs". Inoltre vengono creati degli indici sui campi song_id, emotion_id, e user_id della tabella.

## 5.0 – Bibliografia e sitografia


- https://www.postgresql.org/docs/current/indexes.html
- https://www.postgresql.org/docs/current/sql-copy.html