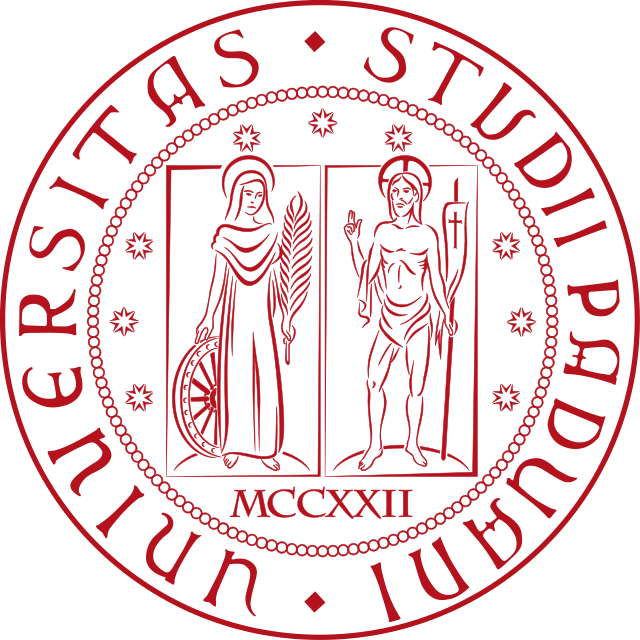
**Università degli studi di Padova**

**Anno Accademico 2023/24**

****

Progetto Basi di Dati

**Soundify DBMS**

Realizzato da:

Alberto Canavese - N. Matricola 2076423

Cheng Feng Liao - N. Matricola 2082843

# Abstract

Obiettivo del progetto: Realizzare un DBMS che gestisca le transizioni e l’iscrizione di utenti registrati a un servizio di streaming per l’ascolto di contenuti come musica e podcast.  
Ogni utente è libero di creare nuove playlist per gestire al meglio la propria musica.

Le tabelle all'interno del database sono interconnesse attraverso relazioni, permettendo di

collegare le informazioni in modo significativo: ad esempio, si possono collegare un artista

agli album che ha pubblicato, collegare ogni utente alle sue canzoni preferite e molto altro ancora.

# Analisi dei requisiti

## Descrizione testuale

Nel database sono presenti i dati degli **artisti** che hanno pubblicato almeno un album o un podcast che è presente nella base di dati.

Di ogni artista sono noti:

* Nome
* E-mail
* Data di nascita
* IBAN

Un artista può appartenere ad un’**etichetta discografica**, di queste si conoscono:

* P. IVA
* Nome dell’etichetta
* Indirizzo della sede legale
* Anno di fondazione
* % di royalties guadagnate dall’etichetta

Ogni artista che raggiunge un certo successo viene retribuito dal servizio di streaming tramite dei **pagamenti**, per ogni pagamento è necessario conoscere:

* ID transazione
* IBAN beneficiario
* Data d’esecuzione
* Importo

Un artista può essere di 2 tipologie: musicista oppure podcast.

Un **podcast** può pubblicare episodi singoli, dei quali sono noti:

* Titolo
* Genere
* Durata
* Riproduzioni
* Data di pubblicazione

Mentre un artista può pubblicare **album musicali**, dei quali si conoscono:

* Titolo
* Genere
* Data di pubblicazione
* Durata
* Riproduzioni

Un album può contenere una o più **canzoni** delle quali sono noti:

* Titolo
* Genere
* Durata
* Riproduzioni
* Data di pubblicazione
* Numero della traccia all’interno dell’album

Tutti questi beni possono essere usufruiti da **utenti** che possono esprimere preferenze di canzoni ed episodi di podcast, di ogni utente si conoscono:

* Username
* Nome
* Cognome
* E-mail

Un utente può creare delle **playlist** contenenti vari brani, di ogni playlist si conoscono:

* Nome
* Condizione di visibilità

Per accedere a brani e podcast ogni utente deve sottoscrivere un **abbonamento**, del quale è noto:

* ID abbonamento
* Data sottoscrizione
* Dada ti scadenza
* Prezzo mensile
* Prezzo annuale

Per pagare l’abbonamento è necessario utilizzare un metodo di pagamento tra **carta di credito/debito** o un **servizio offerto da terze parti**.

Di una **carta di credito** devono essere noti:

* Numero
* Intestatario
* Data di scadenza
* CVV
* Circuito

Mentre, per quanto riguarda i **servizi terzi**, ovvero Apple Pay, Google Pay e PayPal, è necessario conoscere rispettivamente:

* Apple ID
* Google ID
* PayPal ID

# Progettazione concettuale

## Lista entità

Se non specificato l’attributo è *not null*

* **Artista** (Generalizzazione di **musicista** e **podcast**):

CF: varchar (16), primary key

Nome: varchar (100)

E-mail: varchar (100)

Data di nascita: date

IBAN: varchar (27)

* **Pagamento:**

ID transazione: varchar(100), primary key

IBAN beneficiario: varchar (100)

Data esecuzione: date

Importo: real

* **Etichetta discografica:**

P.IVA: varchar(11), primary key

Nome: varchar (100)

Anno fondazione: Integer

Royalties: real

* **Album:**

Titolo: varchar(100), primary key

Genere: varchar(100)

Durata: integer

Riproduzioni: integer

Data pubblicazione: date

* **Canzone:**

Titolo: varchar(100), primary key

Genere: varchar(100)

Durata: integer

Riproduzioni: integer

Data pubblicazione: date

Numero traccia: integer

Album: varchar(100), foreign key

* **Episodio:**

Titolo: varchar(100), primary key

Genere: varchar(100)

Durata: integer

Riproduzioni: integer

Data pubblicazione: date

* **Playlist:**

Nome: varchar(100), primary key

Visibilità\_pubblica: boolean

utente\_creatore: varchar (100) foreign key

* **Utente:**

Username: varchar(100), primary key

Nome: varchar(100)

Cognome: varchar(100)

E-mail: varchar(100)

* **Abbonamento:**

ID\_abbonamento: varchar(100), primary key

Data sottoscrizione: date

Data scadenza: date

Prezzo: real

* **Metodo di pagamento** (Generalizzazione di **carta di credito** e **servizio terzo**)**:**

**Carta di credito:**

Numero: varchar (16), primary key\*

Intestatario: varchar (100)\*

Data di scadenza: date\*

CVV: varchar (3)\*

Circuito: varchar(100)\*

* **Servizio terzo** (generalizzazione di **Apple Pay, Google Pay** e **PayPal):**

Apple Pay:

Apple ID: varchar (100), primary key\*

Google Pay:

Google ID: varchar (100), primary key\*

PayPal:

PayPal ID: varchar (100), primary key\*

*\*l’attributo può essere null*

## Tabella delle relazioni

|  |  |  |
| --- | --- | --- |
| **Relazione** | **Entità coinvolte** | **Descrizione** |
| Possiede | Etichetta discografica (0,n)  Artista (0,1) | Un’etichetta discografica può gestire 0 o molti artisti.  Un artista può o essere indipendente o essere contrattualizzato da un’etichetta discografica. |
| Retribuito | Pagamento (1,1)  Artista (0,n) | Se un artista genera abbastanza ascolti, viene retribuito dalla piattaforma di streaming. |
| Pubblica | Musicista (1,n)  Album (1,n) | Un musicista, per essere registrato sulla piattaforma deve pubblicare almeno un album.  Un album può essere pubblicato da solo un artista, oppure, può essere il frutto della collaborazione tra più musicisti |
| Contiene | Album (1,n)  Canzone (1,1) | Un album più contenere una o più canzoni. Una canzone può appartenere a un solo album. |
| Registra | Podcast (1,n) Episodio (1,1) | Un podcast può pubblicare uno o più episodi.  Un episodio può essere pubblicato da un solo podcast. |
| Appartiene | Canzone (0,n)  Playlist (1,n) | Una canzone può essere presente in una o più playlist.  Una playlist può contenere una o più canzoni. |
| Crea | Utente (0,n)  Playlist (1,1) | Un utente può creare 0 o molte playlist.  Una playlist può essere creata da solo un utente |
| Preferenza brani | Canzone (0,n)  Utente (0,n) | Una canzone può essere selezionata da nessuno o molti utenti.  Un utente può selezionare nessuna o molte canzoni. |
| Preferenza episodi | Canzone (0,n)  Utente (0,n) | Un episodio può essere selezionato da nessuno o molti utenti.  Un utente può selezionare nessuno o molti episodi. |
| Sottoscrive | Utente (1,1)  Abbonamento (0,n) | Un utente può sottoscrivere un solo abbonamento alla volta.  Un abbonamento può essere sottoscritto da nessuno o più utenti. |
| Pagamento | Utente (1,1)  Metodo di pagamento (1,n) | Ogni utente può utilizzare un metodo di pagamento offerto da operatori di terze parti, o con carta o entrambi. |

## Schema concettuale ER

Immagine che contiene diagramma, schizzo, Disegno tecnico, Piano

Descrizione generata automaticamente

## Vincoli non rappresentabili nello schema ER:

Se in **Metodo di pagamento** un utente ha E-mail NOT NULL allora Numero Carta può essere NULL e viceversa.

# Progettazione logica

## Ristrutturazione

* **Eliminazione delle generalizzazioni**

|  |  |
| --- | --- |
| **Generalizzazione** | **Risoluzione** |
| Musicista, podcast → artista | Le entità Musicista e Podcast, che sono specializzazioni dell’entità “artista”, possono essere eliminate poiché aggiungono solo un'informazione supplementare alla generalizzazione, che può essere facilmente sostituita con un attributo. |
| Carta di credito, servizio terzo → metodo di pagamento | La generalizzazione Metodi di pagamento può essere rimossa poiché ciascuna delle sue specializzazioni rappresenta entità diverse, che devono essere distinte. Di conseguenza, vengono aggiunti un attributo per identificare il metodo di pagamento digitale e un altro per identificare la carta. |
| Apple Pay, Google Pay, PayPal → Servizio terzo | La generalizzazione “servizio terzo” può essere rimossa poiché tutte le sue specializzazioni condividono gli stessi attributi. Pertanto, è possibile unificarle in un’unica entità, aggiungendo un attributo che ne distingua la tipologia. |

## Analisi delle ridondanze

L’attributo **data di pubblicazione** dell’entità **Canzone** è ridondante, poiché la data di uscita di una canzone può essere dedotta dalla data di pubblicazione dell’**Album** in cui la canzone è contenuta.

**Operazioni con presenza di ridondanza:**

**Operazione 1:** Recuperare la data di pubblicazione di una canzone direttamente dall’entità **Canzone**.

* **Concetto**: Canzoni
* **Costrutto**: E
* **Accessi**: 1

**Operazione 2:** Recuperare la data di pubblicazione di un album.

* **Concetto**: Album
* **Costrutto**: E
* **Accessi**: 1

**Operazioni con assenza di ridondanza:**

**Operazione 1 senza ridondanza:** Recuperare la data di pubblicazione di una canzone attraverso l’album a cui appartiene.

* **Concetto**: Canzoni, Album, Contiene
* **Costrutto**: E, E, R
* **Accessi**: 1 (Canzoni) + 1 (Contiene) + 1 (Album) = 3

**Operazione 2 senza ridondanza:** Recuperare la data di pubblicazione di un album.

* **Concetto**: Album
* **Costrutto**: E
* **Accessi**: 1

L’eliminazione della ridondanza dell’attributo **data di pubblicazione** nella tabella **Canzone** comporterebbe un aumento del numero di accessi in lettura da 1 a 3 per ogni operazione che richiede la data di pubblicazione della canzone.

Considerando che l'attributo **data di pubblicazione** occupa uno spazio minimo, è più conveniente mantenere la ridondanza per ottimizzare il numero di accessi al database, specialmente in situazioni con elevato traffico di query legate alla data di pubblicazione delle canzoni.

Inoltre, l’incremento del volume di dati dovuto alla ridondanza è trascurabile rispetto ai benefici in termini di performance del sistema.

## Analisi dell’integrità referenziale

**Aggiunta del Codice Fiscale (CF) come chiave primaria per l’entità "Artista":**

**Motivazione**: Per migliorare l'integrità referenziale, è stato sostituito il precedente sistema di chiavi composte (Nome + Email) con una chiave primaria basata sul Codice Fiscale (CF).

Questa scelta evita problematiche legate alla possibile omonimia e al cambiamento delle email, garantendo una chiave unica e stabile per l'identificazione degli artisti.

**Vantaggi**: L'adozione del CF come chiave primaria assicura una maggiore stabilità nelle relazioni tra entità e rafforza l'integrità referenziale del database.

* **Aggiunta di chiavi esterne per le entità "Album", "Canzone" ed "Episodio":**

**Motivazione**: Per evitare ambiguità e garantire una corretta associazione dei contenuti agli artisti, sono state introdotte chiavi esterne nelle entità "Album", "Canzone" ed “Episodio".

Questo è particolarmente importante nel caso di contenuti con nomi simili o identici, ma pubblicati da artisti diversi.

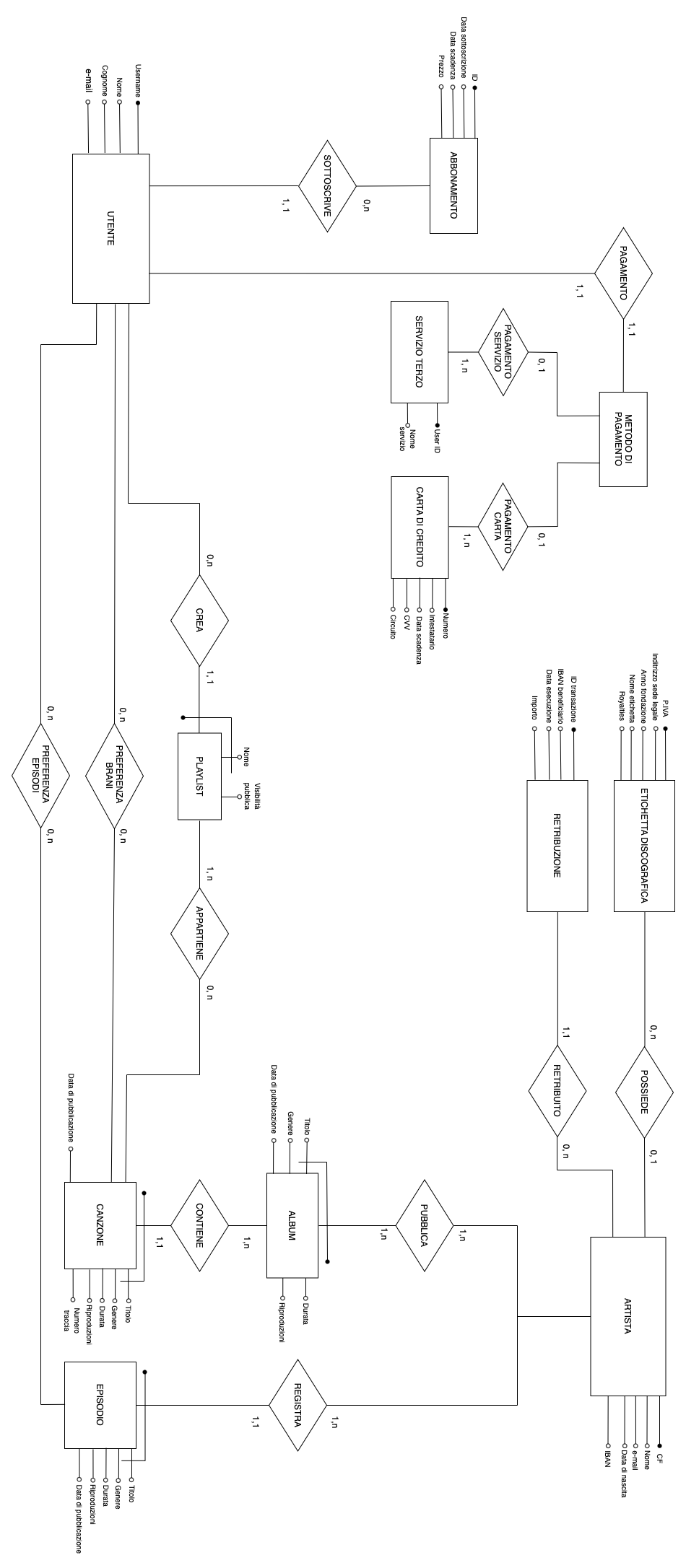
**Vantaggi**: Questa modifica rafforza l'integrità referenziale, assicurando che ogni entità sia collegata in modo univoco e corretto al proprio artista, facilitando anche la precisione nelle ricerche e nelle operazioni sul database.

* **Cambiamento del nome dell’entità "Pagamento" in "Retribuzione":**

**Motivazione**: Per migliorare la chiarezza e ridurre il rischio di ambiguità, è stato cambiato il nome dell'entità "Pagamento" in "Retribuzione". Questo cambiamento evita confusione con l'entità "Metodo di Pagamento", migliorando la comprensibilità del modello.

**Vantaggi**: La modifica contribuisce a una migliore integrità referenziale eliminando possibili ambiguità tra le entità del database

## Schema ER ristrutturato



## Creazione delle tabelle

*(A→B indica che B è chiave esterna di A)*

**Artista** (CF, Nome, e-mail, Data\_di\_nascita, IBAN, PIVA\_etichetta →Etichetta\_discografica.PIVA))

***Nota*:** Il campo *PIVA\_etichetta* può essere nullo, consentendo la presenza di artisti indipendenti non associati ad alcuna etichetta discografica.

**Etichetta discografica** (P.IVA, Indirizzo\_sede\_legale, anno\_fondazione, nome\_etichetta, percentuale\_royalties)

**Retribuzione** (ID\_transazione, data\_esecuzione, importo, IBAN\_beneficiario → artista.IBAN

**Album** (Titolo, CF\_artista → Artista.CF, Genere, Durata, data\_di\_pubblicazione, Riproduzioni)

**Canzone** (Titolo, CF\_artista → Artista.CF, Titolo\_album → Album.Titolo, Genere, Durata, Riproduzioni, numero\_traccia, data\_di\_pubblicazione)

**Episodio** (Titolo, CF\_artista → Artista.CF, Genere, Durata, riproduzioni, data\_di\_pubblicazione)

**Utente** (Username, Nome, Cognome, e-mail)

**Playlist** (Nome, Username\_utente → Utente.Username, Visibilità\_pubblica)

**Appartiene** (Nome\_playlist **→** playlist.Nome, Username\_utente **→** Utente.username, Titolo\_canzone **→** Canzone.titolo, CF\_artista **→** artista.CF, Titolo\_album**→** album.Titolo)

**Preferenza brani** (Username\_utente → Utente.username, Titolo\_canzone → Canzone.Titolo, CF\_artista → Artista.CF, Titolo\_album → Album.Titolo)

**Preferenza Episodi** (Username\_utente → Utente.username, Titolo\_episodio → Episodio.Titolo, CF\_artista → Artista.CF)

**Abbonamento** (ID, Data\_sottoscrizione, Data\_scadenza, Prezzo, utente\_abbonato → Utente.Username)

**Servizio\_terzo** (UserID, nome servizio)

**Carta\_di\_credito** (Numero, intestatario, data scadenza, CVV, circuito)

**Metodo di pagamento**(Username → Utente.Username , UserID → Servizio\_terzo.UserID, NumeroCarta → Carta\_di\_credito.Numero)

* *Se nel Metodo di Pagamento un utente ha un UserID non nullo (indicando l'utilizzo di un servizio terzo), allora il campo NumeroCarta può essere nullo. Questo permette all'utente di associare un metodo di pagamento specifico al proprio account senza utilizzare una carta di credito.  
  Viceversa, se NumeroCarta è non nullo (indicando l'utilizzo di una carta di credito), allora UserID può essere nullo. Questa configurazione permette all'utente di optare per un metodo di pagamento basato su carta di credito senza associare un servizio terzo.*

# Query e indici

## Queries



* Recupera la lista degli artisti e il numero totale di riproduzioni delle loro canzoni:

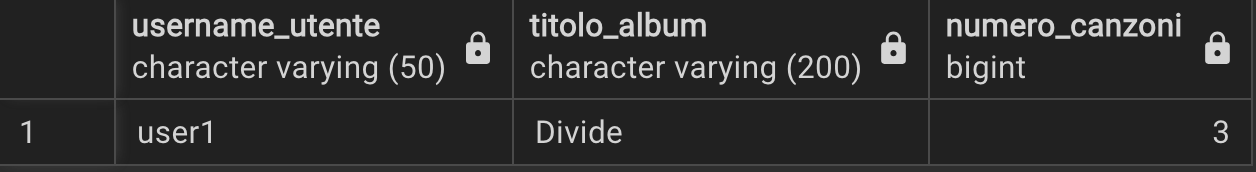
SELECT A.Nome AS Artista, SUM(C.Riproduzioni) AS Totale\_Riproduzioni

FROM Artista A

JOIN Canzone C ON A.CF = C.CF\_artista

GROUP BY A.Nome

ORDER BY Totale\_Riproduzioni DESC;

* Trova gli utenti che hanno almeno 3 canzoni preferite da un singolo album:

SELECT PB.Username\_utente, C.Titolo\_album, COUNT(\*) AS Numero\_Canzoni

FROM Preferenza\_Brani PB

JOIN Canzone C ON PB.Titolo\_canzone = C.Titolo AND PB.CF\_artista = C.CF\_artista AND PB.Titolo\_album = C.Titolo\_album

GROUP BY PB.Username\_utente, C.Titolo\_album

HAVING COUNT(\*) >= 3;

* Trova gli episodi più popolari di ciascun podcast, ovvero quelli con il maggior numero di riproduzioni:

SELECT A.Nome AS Podcast, E.Titolo AS Episodio, E.riproduzioni

FROM Episodio E

JOIN Artista A ON E.CF\_artista = A.CF

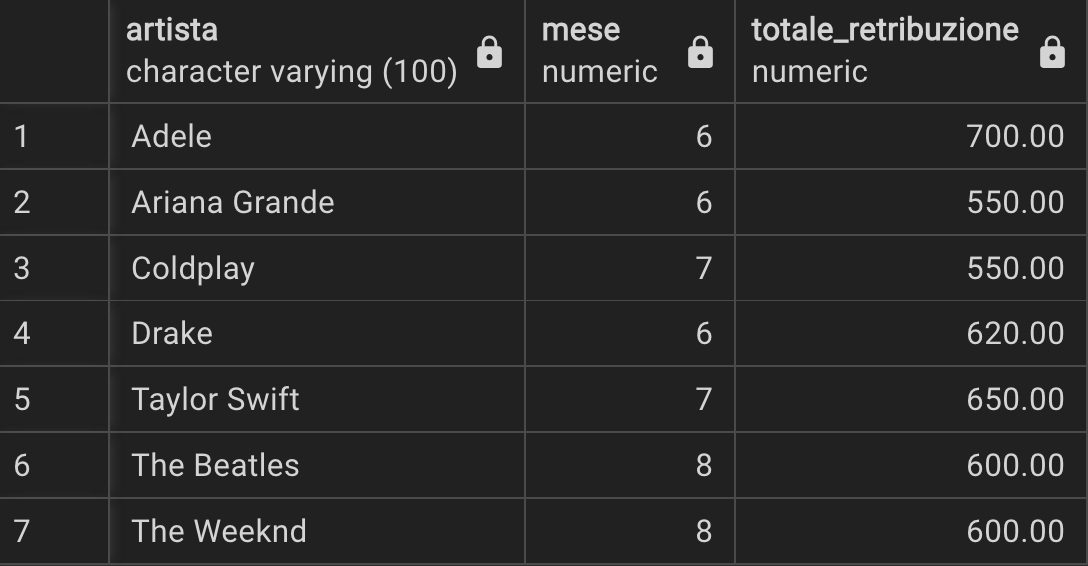
JOIN (

SELECT CF\_artista, MAX(riproduzioni) AS MaxRiproduzioni

FROM Episodio

GROUP BY CF\_artista

) AS SubQuery ON E.CF\_artista = SubQuery.CF\_artista AND E.riproduzioni = SubQuery.MaxRiproduzioni;

* Calcola la retribuzione totale ricevuta da ciascun artista, raggruppata per artista e suddivisa per mese, e mostra solo i mesi in cui un artista ha ricevuto più di 500 euro:

SELECT

A.Nome AS Artista,

EXTRACT(MONTH FROM R.data\_esecuzione) AS Mese,

SUM(R.importo) AS Totale\_Retribuzione

FROM

Artista A

JOIN

Retribuzione R ON A.IBAN = R.IBAN\_beneficiario

GROUP BY

A.Nome, EXTRACT(MONTH FROM R.data\_esecuzione)

HAVING

SUM(R.importo) > 500

ORDER BY

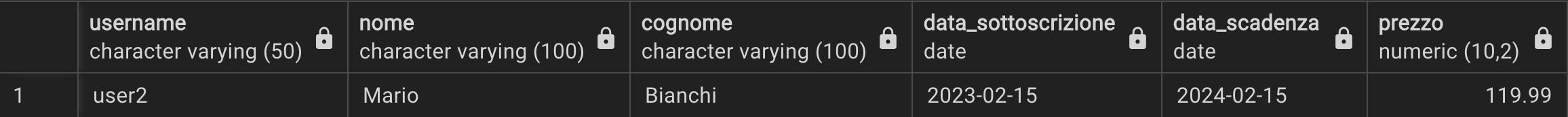
A.Nome, Mese;

* Elenca tutti gli utenti con un abbonamento annuale:

SELECT

U.Username,

U.Nome,

U.Cognome,

A.Data\_sottoscrizione,

A.Data\_scadenza,

A.Prezzo

FROM

Utente U

JOIN

Abbonamento A ON U.Username = A.utente\_abbonato

WHERE

A.Data\_scadenza = A.Data\_sottoscrizione + INTERVAL '1 year'

ORDER BY

A.Data\_sottoscrizione DESC;

## Indici

Per la definizione di un indice prendiamo in esame la **query 5**, che mostra tutti gli utenti con un abbonamento annuale, determinato dalla data di sottoscrizione e dalla data di scadenza.

**Motivazione per la scelta dell’indice:**

Questa query utilizza un JOIN tra la tabella **Utente** e la tabella **Abbonamento**, basandosi sulla colonna **Username** della tabella Utente e sulla colonna **utente\_abbonato** della tabella Abbonamento. Inoltre, la query effettua una selezione basata su un intervallo di date, confrontando le colonne Data\_sottoscrizione e Data\_scadenza nella tabella Abbonamento.

L'indicizzazione delle colonne Username e utente\_abbonato, utilizzate nel JOIN, migliorerà significativamente le prestazioni della query, riducendo i tempi necessari per l'operazione di unione tra le due tabelle. Inoltre, indicizzare la colonna Data\_sottoscrizione può velocizzare le operazioni di confronto per determinare gli abbonamenti annuali.

**Creazione degli indici:**

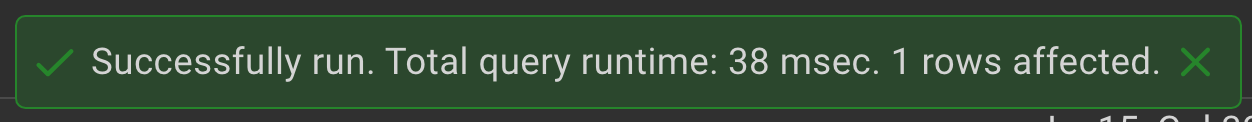
Indice sulla colonna **utente\_abbonato** nella tabella **Abbonamento**:

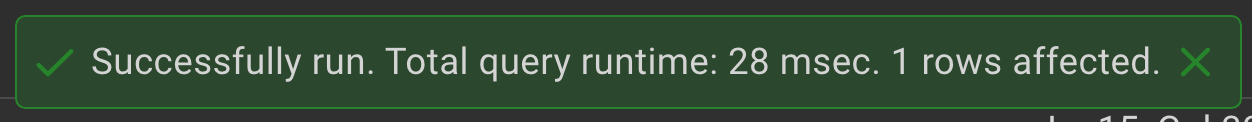
CREATE INDEX idx\_abbonamento\_utente\_abbonato ON Abbonamento (utente\_abbonato);

Indice sulla colonna **Data\_sottoscrizione** nella tabella **Abbonamento**:

CREATE INDEX idx\_abbonamento\_data\_sottoscrizione ON Abbonamento (Data\_sottoscrizione);

L'inclusione di questi indici garantirà che le operazioni di ricerca e unione tra le tabelle Utente e Abbonamento, nonché il confronto delle date, siano eseguite in modo più efficiente. Di conseguenza, i tempi di risposta della query saranno ridotti, migliorando le prestazioni complessive del database.

*Tempo impiegato per eseguire la query 5 senza indici:*

*Tempo impiegato per eseguire la query 5 con gli indici appena descritti:*

Come si può notare il tempo d’esecuzione della query è inferiore quando vengono utilizzati degli indici.

# Codice C

Il codice C per l’esecuzione delle query consiste in un unico file .c che va compilato

utilizzando da linux Ubuntu il seguente comando:

*cd <percorso all'interno della cartella contenente il file.c>*

*gcc -o my\_program queries.c -lpq*

In seguito alla compilazione l’esecuzione:

*./queries*

Per quanto riguarda la compilazione su MacOS sono stati riscontrati diversi problemi di

compatibilità e metodi, per questo nella cartella contenente il codice C si possono trovare

due versioni di “queries.c” in base al sistema operativo utilizzato.

Per comodità le librerie aggiuntive su MacOS sono state installate utilizzando il tool

“Homebrew”, di conseguenza i comandi per compilare e eseguire il codice C su Mac sono

i seguenti:

Compilazione:

*gcc -o queries <percorso del file .c> -I/opt/homebrew/opt/libpq/include -L/opt/homebrew/opt/libpq/lib -lpq*

Esecuzione:

*./queries*

Prima della compilazione è necessario sostituire nelle variabili globali **PG\_HOST**,

**PG\_USER**, **PG\_DB**, **PG\_PASS**, **PG\_PORT** rispettivamente l’indirizzo dell’host, il nome

utente di PostgreSQL, il nome del database, la password e la porta utilizzata.

Il programma utilizza la funzione PQconnectdb per connettersi al database, la funzione

PQstatus per verificare lo stato della connessione, la funzione PQfinish per terminare la

connessione.

Una volta avviato correttamente il programma sullo schermo apparirà il seguente menù con la lista delle query, identificate da 1 a 4, mentre lo 0 terminerà.

Alcune query richiedono l’inserimento di dati aggiuntivi da parte dell’utente, in particolare:

* **La query numero 2** restituisce tutti i brani di un artista specifico in ordine decrescente di ascolti. L'utente deve inserire il codice fiscale dell'artista per identificare in modo univoco l'artista desiderato.
* **La query numero 3** restituisce tutti i brani che hanno un numero di riproduzioni superiore a un valore specificato dall'utente. All'utente è richiesto di inserire il numero minimo di riproduzioni.
* **La query numero 4** restituisce tutti gli episodi pubblicati in un certo intervallo di tempo. L'utente deve inserire la data di inizio e la data di fine dell'intervallo desiderato.

Queste query forniscono informazioni rilevanti sulla base di dati, e quelle che richiedono input utente permettono di ottenere risultati personalizzati in base a esigenze specifiche.